Advanced Descriptive Analysis of Tabular Data

Methods and Tools for Exploratory Analysis

Antoine Soetewey & Cédric Heuchenne

2026-03-31

Table of contents

Preface

Descriptive statistics are often treated as a preliminary step (a routine box to check before moving on to inference, prediction, or causal analysis). Yet in practice, understanding the structure, associations, and patterns within complex tabular data is neither trivial nor purely mechanical. It requires sophisticated methods, thoughtful visualization, and clear communication.

This book synthesizes advanced techniques for descriptive analysis of tabular data, drawing on recent developments in machine learning, network analysis, and interactive visualization. We aim to equip researchers, analysts, policymakers, and data journalists with tools that go beyond means and standard deviations, enabling them to extract actionable insights from multivariate datasets.

The methods and tools we present here are not intended as a repackaging of standard exploratory data analysis. They reflect methodological syntheses, implementation choices, and applied workflows developed through research and field practice. In that sense, this book also serves as a portfolio: a concrete, citable body of work that documents contributions to descriptive analytics and foregrounds substantive methodological originality.

The material emerged from postdoctoral research at the intersection of applied statistics, machine learning, and data visualization. It reflects a pragmatic philosophy: methods can be interpretable, visual, and suitable for communicating findings to statistically literate but non-technical audiences.

Who This Book Is For

This book is intended for readers who already possess a solid foundation in statistics (including regression analysis, hypothesis testing, and basic multivariate methods). We assume familiarity with concepts like correlation, variance decomposition, and model evaluation.

Our intended audience includes:

- **Researchers and applied scientists** seeking exploratory tools for complex datasets
- **Policymakers and analysts** in government and public institutions
- **Data journalists** investigating patterns in social, economic, or health data
- **Consultants and analytical teams** in private firms
- **Graduate students** in statistics, data science, public policy, or related fields

The material is suitable for a Master-level university course and may serve as a foundation for doctoral-level methodological training.

Philosophy and Approach

The unifying thread throughout this book is: **How do we move beyond standard descriptive statistics to extract, visualize, and communicate structure in complex tabular data?**

We emphasize:

- **Interpretability**: Methods that produce understandable results
- **Visual analytics**: Graphs and interactive tools as primary analytical instruments
- **Methodological transparency**: Explicit discussion of assumptions, trade-offs, and limitations
- **Communication**: Presenting results to diverse audiences, from technical peers to policy stakeholders

Rather than offering purely theoretical exposition, we ground each method in real applied use cases, showing how techniques perform on actual data challenges.

Structure of the Book

The book is organized into seven parts:

Part I establishes the conceptual framework, including an advanced preparation step based on semantic variable selection from metadata, and then introducing mixed-type variables and multivariate descriptive structure.

Part II focuses on association analysis: measuring relationships between pairs of variables of different types and representing these associations as networks.

Part III introduces interactive visual analytics, including the AssociationExplorer Shiny application, which operationalizes association-focused methods in a unified exploratory interface.

Parts IV-VI present three families of advanced methods for higher-dimensional structure: tree-based models for segmentation and description, interpretable machine learning techniques for understanding complex patterns, and AutoML approaches that automate exploration.

Part VII presents extended applied case studies from policy analysis, public health, and business analytics, demonstrating how these methods solve real-world problems.

Acknowledgments

This work has been shaped by collaborations with colleagues, conversations with practitioners, and feedback from students and the open-source community. We are particularly grateful to researchers from UCLouvain Saint-Louis Brussels involved in the Beamm research project for their insights and support.

We also acknowledge the open-source software communities whose tools make this work possible, including R, Python, Shiny, Quarto, and countless contributed packages.

How to Use This Book

You can read chapters sequentially or selectively, depending on your background and interests. Readers already familiar with data preprocessing and basic descriptive statistics might skip Part I, while those primarily interested in tree-based methods could jump directly to Part IV for example.

Throughout the book, we provide code examples primarily in R, with selective Python examples when methodologically appropriate (notably for text-embedding retrieval in preparation workflows), and references to accompanying interactive tools. All datasets, code and source files are available on GitHub.

We encourage readers to experiment with the methods on their own data. Descriptive analysis is learned through practice, and one of the best ways to internalize these techniques is to apply them to problems you care about.

About the authors

Antoine Soetewey is a postdoctoral researcher in data science and statistics at HEC Liège and UCLouvain Saint-Louis Brussels. He works on statistical methods and data analysis, with a focus on clear communication and practical applications. More information is available at antoinesoetewey.com.

Cédric Heuchenne is a professor at UCLouvain and HEC Liège. He is affiliated with research and teaching units in data analysis and modeling, and contributes expertise in statistics and applied methods.

Introduction

The Challenge of Describing Complex Data

When we confront datasets with dozens or hundreds of variables (mixing continuous measurements, categorical indicators, ordinal scales, and binary flags), it can be difficult to make sense of the whole. How might we surface the most informative patterns, notice unexpected associations, and communicate what we find in a way that is useful to others?

Traditional descriptive statistics (means, medians, frequency tables, correlation matrices) remain essential, but they can become insufficient as data complexity grows. A correlation matrix for 50 variables contains 1,225 pairwise relationships, many of which may be noise. Standard summary statistics offer limited guidance on which variables matter most, how they interact, or which segments or subpopulations may exist in the data.

This book aims to address the challenge of **advanced descriptive analysis**: moving beyond elementary summaries to extract meaningful structure from complex tabular data. We position it as a methodological portfolio that brings together syntheses, practical tooling, and reproducible workflows, and we hope readers find the combination helpful for their own work.

What Is Descriptive Analysis?

Descriptive analysis characterizes the properties of a dataset, including its distributions, central tendencies, variability, associations, and patterns, without making claims about causation or population-level inference. While often contrasted with inferential or predictive analysis, descriptive work is neither simpler nor less valuable.

Thoughtful descriptive analysis can:

- **Reveal structure**: identifying clusters, segments, or natural groupings
- **Quantify associations**: measuring relationships between variables of mixed types
- **Guide further analysis**: highlighting variables and relationships worthy of deeper investigation
- **Communicate findings**: translating complex patterns into actionable insights

In many applied contexts (policy evaluation, exploratory journalism, early-stage research), descriptive analysis is often the primary goal rather than a prelude to inference.

Why Advanced Methods?

Standard descriptive tools have limitations that become more apparent as complexity increases:

- **Univariate summaries** ignore multivariate structure and conditional relationships
- **Correlation coefficients** only capture linear associations between continuous variables
- **Cross-tabulations** become unwieldy with many categories or variables
- **Scatterplot matrices** fail to scale beyond a handful of variables

Advanced methods can help address these limitations by:

1. **Handling mixed-type variables**: combining continuous, categorical, ordinal, and binary data in a unified framework
2. **Capturing nonlinear relationships**: detecting patterns that correlation coefficients can miss
3. **Automating discovery**: using algorithmic approaches to identify important features and interactions
4. **Visualizing high-dimensional structure**: representing complex associations through networks, trees, and interactive graphics
5. **Enabling exploration**: providing interactive tools that allow analysts to interrogate data from multiple angles

Methods Covered in This Book

This book brings together several methodological traditions, progressing from simpler pairwise relationships to increasingly complex multivariate methods:

Association Measures

A central challenge in descriptive analysis is measuring association when variables are not all continuous. Real-world datasets routinely mix quantitative, qualitative, ordinal, and binary variables, making classical measures like Pearson correlation inadequate or misleading. This book presents a **type-aware framework for association** that selects and scales association measures according to the specific combination of variable types being related. For continuous-continuous pairs, we discuss Pearson, Spearman, and distance-based correlations (Pearson 1895; Anderson 1985; Spearman 1961; Székely, Rizzo, and Bakirov 2007). For categorical-categorical associations, we cover Cramér's V and related measures. For mixed pairs, we employ model-based measures and mutual information approaches. We interpret these measures descriptively rather than inferentially: the goal is not null hypothesis testing, but **comparability and ranking** (identifying which variable pairs exhibit relatively strong relationships meriting closer

inspection). By placing heterogeneous associations on a common scale (often $[0, 1]$), analysts can scan large multivariate datasets and focus attention on the relationships that appear most informative, regardless of variable type. This pragmatic, unified treatment aims to turn a fragmented set of statistical tools into a coherent exploratory framework.

Network Representations

When a dataset contains dozens or hundreds of variables, even a well-chosen association measure can produce an overwhelming matrix of pairwise relationships. We try to address this scalability challenge by encoding associations as **variable networks** where nodes represent variables and edges represent relationships exceeding a chosen threshold. Edge weights or colors encode association magnitudes, while network layouts position variables spatially so that strongly related variables cluster near each other. This spatial organization can make high-dimensional association structure more visible and interpretable at a glance. Beyond individual associations, network analysis can reveal **global structure**: communities of tightly interconnected variables that may represent distinct domains or latent constructs, hub variables that bridge multiple domains, and peripheral variables carrying unique information. Centrality measures (degree, betweenness, eigenvector) help identify influential variables. Community detection algorithms partition variables into meaningful groups. These higher-order network properties are difficult to discern from association matrices or pairwise plots alone, yet they often provide useful insights into data structure. Network representations can therefore serve as a cognitive map of the dataset, guiding exploratory analysis through high-dimensional association space.

Interactive Visual Analytics

Static visualizations answer pre-determined questions; interactive tools can enable more dynamic exploration. Interactive graphics (particularly Shiny applications) allow analysts to filter data by conditions, aggregate across subgroups, adjust plot parameters, and link multiple views in real time. This interactivity supports hypothesis generation and refinement: analysts can test "what if" scenarios, drill down into subpopulations, and detect patterns that might not appear in static plots. Dashboards combine multiple interactive visualizations into coordinated workflows, allowing stakeholders to explore data according to their own questions rather than passively receiving predetermined findings. For descriptive analysis in particular, interactivity can be essential for handling high-dimensional data. An interactive tool can present association networks, tree structures, and distributions while allowing users to focus on subsets, time periods, or demographic groups of interest. This chapter introduces Shiny-based applications and demonstrates how to build interactive descriptive tools that can scale to real-world data complexity.

Tree-Based Methods

Regression and classification trees can offer a powerful yet interpretable approach to segmenting populations and understanding conditional structure in data (Breiman et al. 2017). Trees recursively partition data based on variable thresholds, producing interpretable decision rules that separate observations into relatively homogeneous groups. Unlike black-box predictive models, tree structures are transparent: practitioners can often explain why a particular observation falls into a specific segment. Trees naturally reveal which variables are most discriminative and at what thresholds decisions change. This makes them useful for exploratory work, program targeting, and communicating findings to stakeholders who value transparency. Moreover, ensemble extensions (combining multiple trees through random forests or boosting) can improve robustness while preserving the ability to extract interpretable variable importance measures and identify complex interactions (Breiman 2001; J. H. Friedman 2001).

Interpretable Machine Learning

Modern machine learning models often achieve strong predictive accuracy compared to classical statistical methods, but at the cost of interpretability. Interpretable machine learning bridges this gap by providing techniques to understand what complex models have learned from data. Methods like permutation-based feature importance identify which variables the model relies on most heavily (A. Fisher, Rudin, and Dominici 2019). Individual conditional expectation curves visualize how predictions change as individual features vary, revealing nonlinear relationships and thresholds (Goldstein et al. 2015). Shapley values (grounded in cooperative game theory) decompose each prediction into additive contributions from each feature, providing both global importance rankings and local explanations for individual observations (Shapley 1953; Lundberg and Lee 2017). These post-hoc interpretation tools can help transform predictive models into descriptive instruments, enabling analysts to extract actionable insights about variable relationships while leveraging the flexibility of modern ML algorithms.

AutoML for Exploration

Automated machine learning platforms systematically search across hundreds or thousands of model configurations, feature transformations, and hyperparameters to identify strong-performing models for a given task. Rather than viewing AutoML purely as a prediction tool, we use it here as an exploratory instrument. AutoML workflows can help surface which features matter, which transformations improve predictive signals, and which variable interactions appear important. By screening a vast model space, AutoML can identify complex patterns that might be missed through manual feature engineering or simpler methods. When interpreted descriptively (i.e., focusing on which transformations boost performance rather than out-of-sample accuracy itself), AutoML can become a rapid hypothesis-generation engine, especially valuable for preliminary analysis of new datasets or when domain expertise is limited. The

rankings and performances of different models can also reveal which features and interactions the data best supports.

Real-World Applications

Each method is illustrated with applied examples drawn from:

- **Public policy**: understanding determinants of program participation, analyzing survey data on citizen attitudes
- **Public health**: exploring risk factors in epidemiological data, characterizing patient populations
- **Business analytics**: segmenting customers, identifying drivers of satisfaction or churn
- **Social science research**: analyzing survey responses, detecting patterns in observational data
- **Data journalism**: investigating patterns in government data, economic indicators, or social trends

Throughout the book, we implement these methods in R and demonstrate them on real datasets. A recurring example is the **AssociationExplorer** Shiny application, developed as part of the research underlying this work, which integrates multiple descriptive techniques into a unified interactive interface (Soetewey et al. 2026). While all methods can be implemented using standard statistical software, AssociationExplorer provides a practical tool for immediate exploratory use. These examples are intended to show that advanced descriptive methods need not be academic exercises alone; they can help address genuine problems faced by analysts across diverse fields and highlight contributions that may be useful in practice.

Relationship to Other Analytical Goals

Descriptive analysis intersects with but differs from:

- **Exploratory Data Analysis (EDA)**: Descriptive analysis is a form of EDA, but emphasizes quantitative measures and formal methods alongside graphical exploration
- **Predictive modeling**: We use predictive models descriptively, focusing on interpretation rather than out-of-sample performance
- **Causal inference**: Descriptive analysis identifies associations but does not claim causation; it can, however, inform causal hypotheses
- **Dimension reduction**: Methods like PCA and MCA reduce dimensionality (Pearson 1901; Jolliffe and Greenacre 1986); we emphasize interpretable summaries that preserve variable identities

Structure and Learning Path

We proceed from foundations to applications:

- **Chapters 1-3** establish conceptual groundwork, including semantic variable selection from metadata, mixed-type data challenges, and advanced descriptive summaries
- **Chapters 4-6** focus on association measures and network representations
- **Chapters 7-9** introduce interactive visual analytics
- **Chapters 10-18** present three families of advanced methods (trees, interpretable ML, AutoML)
- **Chapters 19-21** present extended applied case studies
- **Chapter 22** concludes with reflections and future directions

Readers can follow a linear path or jump to chapters matching their immediate needs. Code examples and exercises throughout encourage hands-on practice.

Computational Tools

We use R for most examples, chosen for its rich ecosystem of statistical graphics and modeling packages, and we occasionally use Python where modern NLP tooling is especially relevant (for example, sentence-embedding retrieval from metadata). Key R packages include:

- `{tidyverse}` for data manipulation and workflow
- `{ggplot2}` and `{ggraph}` for visualization and network graphs
- `{rpart}` and `{rpart.plot}` for tree-based methods and visualization
- `{igraph}` for network analysis and representation
- `{shiny}` for interactive applications

We provide reproducible code examples throughout the book, with data sourced from public datasets, package-included examples, or linked repositories where applicable.

Looking Ahead

The chapters that follow aim to offer a coherent toolkit for advanced descriptive analysis. While methods vary, the underlying goal remains constant: to help you see more deeply into your data, communicate findings clearly, and make better-informed decisions.

Descriptive analysis is both art and science, requiring statistical rigor, visual judgment, and domain knowledge. We hope this book equips you with methods and perspectives that enhance all three.

Part I

Foundations

1 Data Preparation for Descriptive Analysis

1.1 The Necessary Groundwork

Data preparation remains foundational in any empirical workflow. Before moving toward advanced descriptive methods, we still need to inspect variable classes, detect implausible values, document coding conventions, handle missingness, and keep a reproducible audit trail of preprocessing decisions. These steps are not optional in advanced work, they are what makes advanced work interpretable.

Many readers will already be familiar with this groundwork. We therefore keep it concise in this chapter. For missing-data handling, for example, simple baselines such as median or mean imputation can be useful as first passes, while multiple imputation approaches are often more appropriate when uncertainty propagation is important (Rubin 1987; van Buuren and Groothuis-Oudshoorn 2011). Similarly, metadata dictionaries, type checks, and transformation logs remain good practice regardless of the downstream method.

The focus here is different: we concentrate on an advanced preparation step that becomes critical when variable spaces are large, namely semantic variable selection from metadata descriptions.

1.2 The Variable Selection Problem in Large Surveys

In modern applied settings, analysts frequently work with hundreds or thousands of documented variables. For example, the ESS Round 11 integrated file includes more than 600 variables (European Social Survey European Research Infrastructure (ESS ERIC) 2025, 2024). In parallel survey infrastructures, such as Eurostat documentation workflows, combined dictionaries can exceed 1,700 variables across several surveys.

At this scale, manual browsing of variable lists is slow and error-prone. The challenge is not only volume, it is semantic mismatch: research questions are expressed in natural language, while variable names are often terse technical codes (e.g., `HY040G`, `PL031`, `DA1000`). A researcher interested in "rental income" or "health attitudes" may not know where to start from code names alone.

This motivates a preparation technique that maps natural-language queries to variable descriptions automatically, before any statistical modeling begins.

1.3 From Keywords to Meanings: Sentence Embeddings

Sentence embeddings represent text as dense vectors in a high-dimensional semantic space. Texts with similar meaning tend to be close in that space, even when they do not share the exact same keywords. Transformer-based models such as BERT (Devlin et al. 2019) and Sentence-BERT (Reimers and Gurevych 2019) make this practical for retrieval tasks.

In our setting, each variable description is encoded as a vector. A user query is encoded in the same space. We then rank variables using cosine similarity, following the vector-space retrieval logic developed in information retrieval (Salton, Wong, and Yang 1975). The highest-ranked variables form a short list of candidates for subsequent analysis.

For didactic purposes, we use the lightweight model `all-MiniLM-L6-v2` from the `sentence-transformers` ecosystem. This is smaller than multilingual production models, but well-suited for a chapter example.

1.4 A Semantic Retrieval Pipeline in Python

The following implementation illustrates a compact retriever class for metadata CSV files with two columns, `Variable` and `Description`.

```
from pathlib import Path
import csv
import math
import re
import sys
import pandas as pd

from IPython.display import display, Markdown

# Show full text in table cells for chapter outputs.
pd.set_option("display.max_colwidth", None)

try:
    from sentence_transformers import SentenceTransformer
    HAS_SENTENCE_TRANSFORMERS = True
except Exception:
    HAS_SENTENCE_TRANSFORMERS = False


class CSVSemanticRetriever:
    """Semantic variable retriever from a metadata CSV."""
```

```
    def __init__(
        self,
        csv_path,
        variable_col="Variable",
        description_col="Description",
        model_name="all-MiniLM-L6-v2",
    ):
        self.csv_path = Path(csv_path)
        self.variable_col = variable_col
        self.description_col = description_col
        self.model_name = model_name

        self.records = self._load_metadata()

        self.model = None
        if HAS_SENTENCE_TRANSFORMERS:
            self.model = SentenceTransformer(self.model_name)

        self.backend = "sentence-transformers" if self.model is not None else "lexical-fallba

        self.embeddings = None
        if self.model is not None:
            self.embeddings = self._build_embeddings([r["Description"] for r in self.records]

        # Lexical fallback index for environments where sentence-transformers is unavailable
        self.doc_tokens = [set(self._tokenize(r["Description"])) for r in self.records]
        token_df = {}
        for tokens in self.doc_tokens:
            for tok in tokens:
                token_df[tok] = token_df.get(tok, 0) + 1

        n_docs = max(len(self.doc_tokens), 1)
        self.idf = {tok: math.log((1 + n_docs) / (1 + doc_freq)) + 1.0 for tok, doc_freq in t

    def _load_metadata(self):
        if not self.csv_path.exists():
            raise FileNotFoundError(f"Metadata file not found: {self.csv_path}")

        with self.csv_path.open("r", encoding="utf-8-sig", newline="") as f:
            reader = csv.DictReader(f)
            if reader.fieldnames is None:
                raise ValueError(f"Could not read header from {self.csv_path}.")
```

```
            normalized = {name.strip(): name for name in reader.fieldnames}
            var_key = normalized.get(self.variable_col)
            desc_key = normalized.get(self.description_col)

            if var_key is None or desc_key is None:
                raise ValueError(
                    f"Expected columns '{self.variable_col}' and '{self.description_col}' in
                )

            records = []
            for row in reader:
                var = str(row.get(var_key, "")).strip()
                desc = str(row.get(desc_key, "")).strip()
                if var and desc:
                    records.append({"Variable": var, "Description": desc})

        if not records:
            raise ValueError(
                f"No valid rows found in {self.csv_path}. Expected columns '{self.variable_co
            )

        return records

    @staticmethod
    def _tokenize(text):
        tokens = re.findall(r"[a-z0-9]+", text.lower())
        stopwords = {
            "a", "an", "and", "are", "as", "at", "be", "by", "for", "from", "how", "in", "is
            "it", "of", "on", "or", "that", "the", "this", "to", "was", "were", "what", "whe
            "where", "which", "who", "with", "you", "your",
        }
        return [tok for tok in tokens if len(tok) > 2 and tok not in stopwords]

    def _build_embeddings(self, texts, is_query=False):
        # E5 models are trained with explicit query/passage prefixes.
        if "e5" in self.model_name.lower():
            prefix = "query: " if is_query else "passage: "
            texts = [prefix + t for t in texts]

        return self.model.encode(texts, normalize_embeddings=True)

    @staticmethod
```

```
    def _cosine(a, b):
        if hasattr(a, "tolist"):
            a = a.tolist()
        if hasattr(b, "tolist"):
            b = b.tolist()
        return sum(x * y for x, y in zip(a, b))

    def retrieve(self, query, top_k=5):
        use_semantic = self.model is not None and self.embeddings is not None
        q_emb = self._build_embeddings([query], is_query=True)[0] if use_semantic else None
        q_tokens = None if use_semantic else set(self._tokenize(query))

        scored = []
        for i, rec in enumerate(self.records):
            if use_semantic:
                score = self._cosine(q_emb, self.embeddings[i])
            else:
                overlap = q_tokens.intersection(self.doc_tokens[i])
                if not overlap:
                    score = 0.0
                else:
                    q_union_d = q_tokens.union(self.doc_tokens[i])
                    inter_weight = sum(self.idf.get(tok, 1.0) for tok in overlap)
                    union_weight = sum(self.idf.get(tok, 1.0) for tok in q_union_d)
                    score = inter_weight / max(union_weight, 1e-12)

            scored.append({
                "Variable": rec["Variable"],
                "Description": rec["Description"],
                "Similarity": float(score),
            })

        scored.sort(key=lambda x: x["Similarity"], reverse=True)
        return scored[:top_k]


def results_to_dataframe(rows):
    df = pd.DataFrame(rows)
    if not df.empty:
        df["Similarity"] = df["Similarity"].round(3)
        df = df[["Variable", "Similarity", "Description"]]
    return df
```

This class follows the same conceptual structure used in larger retrieval pipelines: load metadata, embed descriptions, embed query, rank by cosine similarity, return top results.

1.4.1 Note for R Users (Conceptual Workflow)

The same logic can be implemented in R without changing the analytical objective. Conceptually, the workflow remains identical: (1) read the metadata dictionary from CSV, (2) compute text embeddings for variable descriptions with an available embedding interface, (3) encode user queries in the same vector space, (4) compute cosine similarity between query and description vectors, and (5) return the top-k ranked variables for review. In practice, this can be done with native R text tooling or by calling Python embedding libraries through an R bridge when model coverage is needed.

1.5 Applying the Pipeline to ESS Round 11

We now apply the retriever to the ESS metadata file (available in the data folder of the GitHub repository). It acts as a variable dictionary with one row per variable and two key fields, `Variable` and `Description`, which are exactly the inputs required for semantic retrieval.

Before running retrieval, it is helpful to inspect the first lines of the metadata file to verify its expected structure.

```
metadata_preview = pd.read_csv("data/ESS11_variable_description.csv", encoding="utf-8-sig").h
display(metadata_preview)
```

	Variable	Description
0	nwspol	News about politics and current affairs: watching: reading or listening: in minutes
1	netusoft	Internet use: how often
2	netustm	Internet use: how much time on typical day: in minutes
3	ppltrst	Most people can be trusted or you can't be too careful
4	pplfair	Most people try to take advantage of you: or try to be fair

```
retriever = CSVSemanticRetriever("data/ESS11_variable_description.csv")

print(f"Number of variable descriptions loaded: {len(retriever.records)}")
print(f"Sentence-transformers available: {HAS_SENTENCE_TRANSFORMERS}")
print(f"Retrieval backend: {retriever.backend}")
print(f"Python executable: {sys.executable}")
```

```python
queries = [
    "attitudes towards health and well-being",
    "household income and financial situation",
]

for q in queries:
    display(Markdown(f"**Query:** {q}"))
    top = retriever.retrieve(q, top_k=5)
    display(results_to_dataframe(top))
```

```
Warning: You are sending unauthenticated requests to the HF Hub. Please set a HF_TOKEN to ena

Loading weights:   0%|          | 0/103 [00:00<?, ?it/s]

BertModel LOAD REPORT from: sentence-transformers/all-MiniLM-L6-v2
Key                     | Status     |  |
------------------------+------------+--+-
embeddings.position_ids | UNEXPECTED |  |

Notes:
- UNEXPECTED:   can be ignored when loading from different task/architecture; not ok if you c

Number of variable descriptions loaded: 468
Sentence-transformers available: True
Retrieval backend: sentence-transformers
Python executable: /Users/antoinesoetewey/Documents/GitHub/Advanced-descriptive-
analysis-of-tabular-data/.venv/bin/python
```

Query: attitudes towards health and well-being

	Variable	Similarity	Description
0	health	0.565	Subjective general health
1	iphlppla	0.538	Important to help people and care for others well-being
2	stflife	0.506	How satisfied with life as a whole
3	ipeqopta	0.386	Important that people are treated equally and have equal opportunities
4	hlthhmp	0.382	Hampered in daily activities by illness/disability/infirmity/mental problem

Query: household income and financial situation

	Variable	Similarity	Description
0	hincfel	0.746	Feeling about household's income nowadays
1	hincsrca	0.741	Main source of household income
2	hinctnta	0.681	Household's total net income: all sources
3	fnsdfml	0.645	Severe financial difficulties in family when growing up: how often
4	hhmmb	0.566	Number of people living regularly as member of household

For the first query, we typically expect variables related to perceived health, life satisfaction, and subjective well-being. For the second query, we expect variables concerning household income levels, financial strain, and material conditions. In practice, this first-pass shortlist already reduces the manual search space substantially.

1.6 Interpreting Similarity Scores

Cosine scores are relative rather than absolute quality guarantees. A score of 0.72, for instance, can be highly relevant in one dictionary and less informative in another. It is often more useful to inspect rank order, score gaps, and thematic coherence of the top results than to rely on one universal threshold.

Three practical cautions are worth noting:

- Polysemy can create ambiguous matches when the same word has several meanings.
- Domain shift can reduce quality when metadata vocabulary differs from the model's training distribution.
- Very short descriptions can limit semantic context and produce unstable ranking.

Even with these limits, embedding-based retrieval is generally more robust than exact keyword search when wording varies across documentation files.

1.7 Placing This Step in the Broader Pipeline

This chapter covers step 1 of a broader descriptive workflow: semantic variable selection from metadata. Its output is a focused set of candidate variables aligned with the research question.

Subsequent steps belong to later chapters: richer exploratory summaries in Chapter 3, type-aware association measurement in Chapter 4, and network-based structure mapping in Chapter 6. Interactive communication layers are then developed further in Chapter 8.

The key point is conceptual: semantic retrieval is a preparation operation on metadata, not a statistical analysis of the observed sample itself.

1.8 Summary and Key Takeaways

- **Groundwork still matters**: Missing-data handling, plausibility checks, encoding decisions, and reproducible pipelines remain essential before advanced analysis.
- **Scale changes the preparation problem**: With hundreds or thousands of variables, manual selection from codebooks becomes inefficient and inconsistent.
- **Embeddings bridge language and variable codes**: Sentence embeddings map free-text research questions and variable descriptions into a common semantic space.
- **Cosine ranking yields practical shortlists**: Top-k retrieval provides an operational way to identify candidate variables before statistical analysis.
- **This is preparation, not modeling**: The output is a curated variable subset that feeds downstream descriptive and association methods.
- **Interpretation remains necessary**: Similarity scores support ranking, but domain review is still needed to validate final variable choices.

1.9 Looking Ahead

With a semantically curated variable shortlist in hand, we can now examine variable types more formally. The next chapter focuses on data types in tabular data, which is a necessary step before selecting appropriate descriptive and association measures.

2 Data Types in Tabular Data

2.1 Why Variable Type Matters

Descriptive analysis depends on **measurement scale**. A statistic that is meaningful for continuous variables can be misleading for categorical or ordinal data. Before we compute any association, it helps to classify variables carefully.

This chapter provides a compact guide to data types and the practical decisions they imply. The goal is to build a shared vocabulary for the rest of the book.

2.2 Measurement Scales

Most tabular variables fall into one of four measurement scales:

- **Nominal**: unordered labels (e.g., region, industry, color)
- **Ordinal**: ordered categories (e.g., education level, Likert scales)
- **Interval**: numeric scales with equal spacing but no true zero (e.g., temperature in °C)
- **Ratio**: numeric scales with a meaningful zero (e.g., income, height)

In descriptive analysis, the key distinction is **order vs. no order** and **numeric vs. non-numeric**. These distinctions determine whether ranks, distances, or variance-based measures are appropriate.

2.3 Common Data Types in Tabular Data

In practice, we usually work with these operational types:

- **Continuous**: real-valued measurements (income, height, temperature)
- **Discrete**: non-negative integers (counts of events, number of visits)
- **Categorical (nominal)**: unordered labels (industry, region)
- **Ordinal**: ordered categories (education level, satisfaction)
- **Binary**: two-level indicators (yes/no, 0/1)

Binary variables can be treated as a special case of categorical or ordinal data, though they often benefit from their own association measures for clarity and comparability.

2.4 Practical Heuristics for Classification

When codebooks are incomplete, a few simple heuristics can help classify variables:

1. **Check data types**: numeric vs. character/factor.
2. **Inspect unique values**: few unique values often signal categorical or ordinal variables.
3. **Look for ordering**: ordered factors or natural rank in labels.
4. **Validate units**: ratios and intervals differ in interpretability (e.g., 20°C is not "twice" 10°C).

A misclassified variable can distort downstream association measures, so it is often helpful to include these checks in any descriptive pipeline.

2.5 Common Pitfalls

- **Numeric codes for categories**: e.g., 1 = "North", 2 = "South". It is usually better to treat these as categorical, not continuous.
- **Ordinal scales stored as text**: e.g., "low", "medium", "high" without explicit ordering.
- **Sparse categories**: rare levels can inflate or destabilize association measures.
- **Mixed scales**: variables constructed from heterogeneous sources (e.g., indices) may not behave like true ratio scales.

2.6 Checklist Before Computing Associations

- Confirm variable types with documentation or exploratory checks.
- Convert numeric codes to factors when they represent labels.
- Encode ordinal variables with explicit ordering.
- Decide how to handle rare categories (merge, drop, or keep with caution).

2.7 Summary and Key Takeaways

- **Variable type determines appropriate methods**: The measurement scale (nominal, ordinal, interval, ratio) shapes which descriptive statistics and association measures are valid.
- **Five operational types cover most cases**: Continuous, discrete, categorical, ordinal, and binary variables each call for different analytical approaches.
- **Misclassification can lead to misleading results**: Treating numeric codes as continuous or ignoring ordinal structure can distort associations and summaries.

- **Heuristics help when codebooks are incomplete**: Checking data types, unique values, natural ordering, and units provides practical guidance for classification.
- **Common pitfalls are avoidable**: Being attentive to numeric codes for categories, unordered ordinal text, sparse categories, and mixed-scale indices helps prevent errors.
- **Explicit classification supports comparability**: Type-aware analysis helps ensure that different variable pairs are measured with appropriate, comparable methods.

2.8 Looking Ahead

The next chapter expands the descriptive toolkit beyond basic summaries. With data types clarified, we can interpret richer multivariate structure and prepare for systematic association measurement in later parts of the book.

3 Beyond Basic Descriptive Statistics

3.1 Why "Basic" Summaries Are Not Enough

A mean, a median, and a standard deviation can be computed in seconds, but they rarely tell the whole story. Two datasets can share identical means and variances while having radically different shapes, outlier structures, or subgroup patterns. In high-dimensional datasets, univariate summaries also conceal interaction and conditional relationships that often matter more than any marginal distribution.

Basic descriptive statistics are often necessary but not always sufficient. This chapter expands the descriptive toolbox with methods that remain **non-inferential** while offering richer insight into data structure. We aim to explore more nuanced questions:

- Where are the *mass* and *tails* of the distribution?
- Are there **subpopulations** with distinct profiles?
- Are relationships **nonlinear**, **heterogeneous**, or **conditional**?
- Which variables are **stable** versus **volatile** across subgroups?

3.2 Distributional Shape: Beyond Mean and Variance

3.2.1 Quantiles and Tail Behavior

Quantiles describe where the data live, not just how they average out. In applied settings, percentiles often carry more operational meaning than averages. The 90th percentile of response time, income, or waiting time is usually more informative than a mean.

Common descriptive quantiles:

- **Median** ($Q_{0.5}$): robust center
- **Interquartile range (IQR)**: spread of the middle 50%
- **Tail quantiles**: $Q_{0.9}$, $Q_{0.95}$, $Q_{0.99}$ for risk or extreme behavior

These can be especially important in skewed or heavy-tailed distributions where the mean can be misleading.

3.2.2 Skewness, Kurtosis, and Robust Alternatives

Skewness and kurtosis summarize asymmetry and tail heaviness, but they are sensitive to outliers. In descriptive work, **robust measures** often provide more stable diagnostics:

- **Median absolute deviation (MAD)** as a scale measure
- **Robust z-scores** using median and MAD instead of mean and standard deviation (SD)
- **Quantile ratios** (e.g., $Q_{0.9}/Q_{0.5}$) for skewness proxies

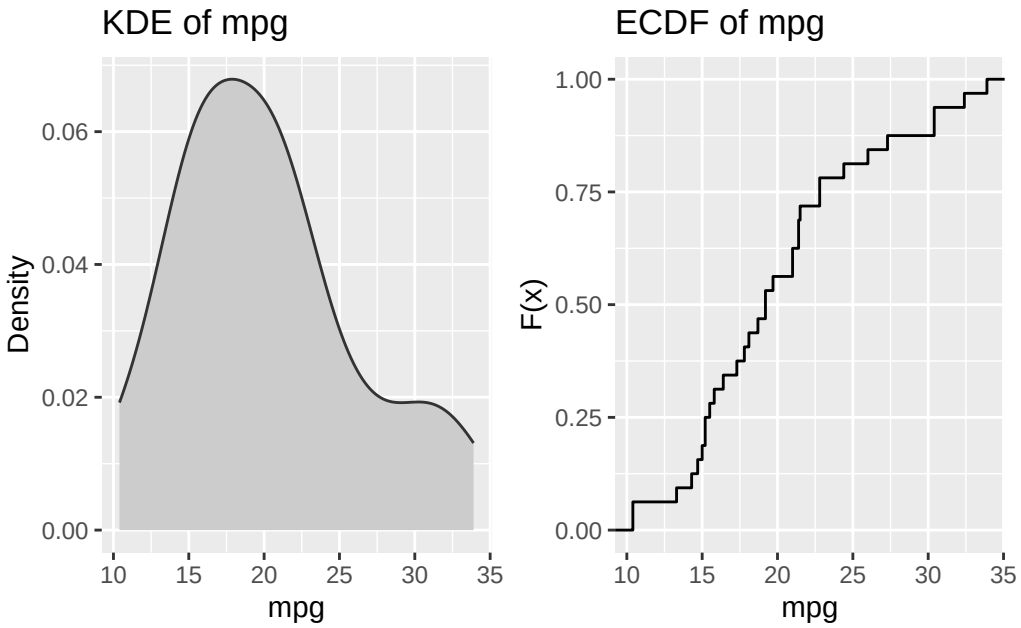
These measures preserve descriptive intent while reducing sensitivity to extreme observations.

3.2.3 Density and Empirical Distribution Functions

Histograms can be misleading due to binning choices. Kernel density estimates (KDEs) and empirical CDFs show shape more faithfully (Parzen 1962). ECDFs are particularly useful for comparing distributions because they show the full cumulative structure without smoothing.

For this first example we use `mtcars`, a built-in R dataset of 32 automobile models (from the 1973-74 *Motor Trend* era) with fuel consumption and design/performance variables such as miles per gallon (`mpg`), weight (`wt`), horsepower (`hp`), and transmission type (`am`).

```
# Density and ECDF side by side
p1 <- mtcars |>
  ggplot(aes(x = mpg)) +
  geom_density(fill = "grey80", color = "grey20") +
  labs(title = "KDE of mpg", x = "mpg", y = "Density")

p2 <- mtcars |>
  ggplot(aes(x = mpg)) +
  stat_ecdf(geom = "step") +
  labs(title = "ECDF of mpg", x = "mpg", y = "F(x)")

p1 + p2
```



3.3 Multimodality and Mixtures

A single distribution can conceal multiple regimes. For example, household income often reflects a mixture of wage earners, retirees, and business owners. Multimodality is a descriptive signal of underlying subpopulations. Techniques to detect it include:

- **Kernel density plots** with multiple peaks
- **Bimodality coefficients** or dip tests (used descriptively)
- **Mixture summaries** (e.g., fitting a Gaussian mixture model purely for segmentation)

Even without formal modeling, visual inspection and stratified summaries can help reveal important mixtures.

3.4 Bivariate and Conditional Descriptives

3.4.1 Conditional Means and Quantiles

Univariate summaries hide conditional variation. A variable may have a stable mean overall but vary dramatically across categories. Conditional statistics are simple to compute and often reveal key structure:

- $E(Y \mid X)$: mean outcomes by group

wt_bin	mpg_median	mpg_iqr
1	30.40	4.75
2	21.00	1.10
3	18.95	2.82
4	15.35	1.80
5	14.00	4.85

- $Q_{0.5}(Y \mid X)$: median outcomes by group
- $\text{IQR}(Y \mid X)$: spread by group

These summaries can be visualized using grouped boxplots, violin plots, or ridgeline plots (David and Tukey 1977; Hintze and Nelson 1998).

3.4.2 Nonlinear Relationships

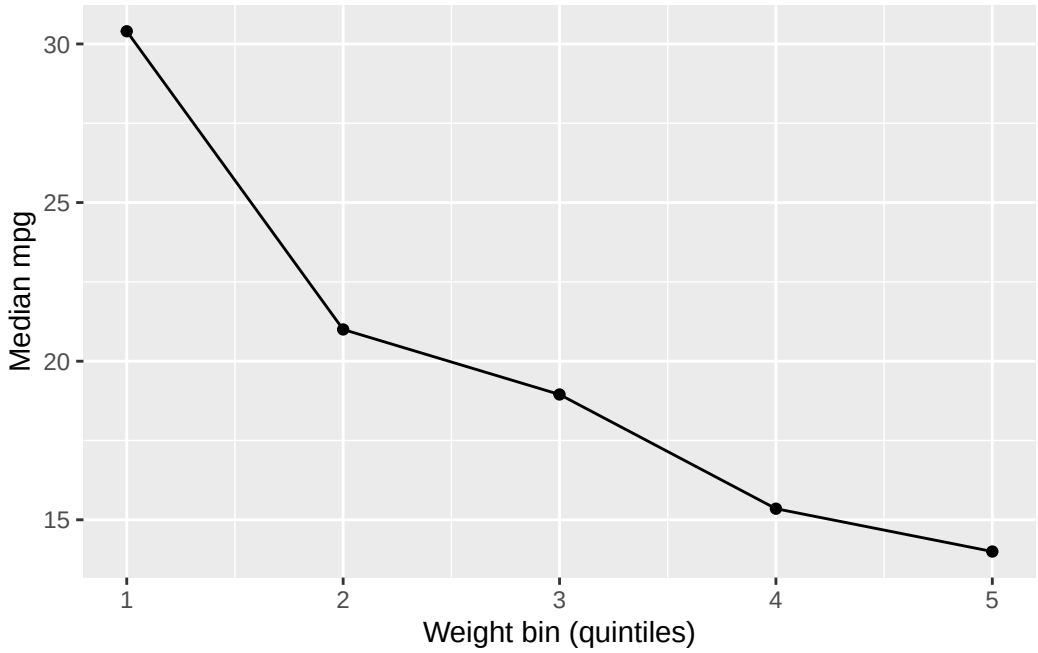
Scatterplots with smoothers (e.g., LOESS) often reveal nonlinear trends that correlations miss (Cleveland 1979). A zero correlation does not imply "no relationship"; it may reflect a U-shape, threshold effect, or segmented pattern.

A useful descriptive strategy is to compute **binned summaries**: divide a continuous predictor into quantile bins and summarize the response within each bin. This provides a simple approximation of conditional structure without invoking a full model.

```
# Binned summaries to reveal nonlinear structure
mtcars |>
  mutate(wt_bin = ntile(wt, 5)) |>
  group_by(wt_bin) |>
  summarise(
    mpg_median = median(mpg, na.rm = TRUE),
    mpg_iqr = IQR(mpg, na.rm = TRUE),
    .groups = "drop"
  ) |>
  gt() |>
  fmt_number(columns = c(mpg_median, mpg_iqr), decimals = 2)
```

```
mtcars |>
  mutate(wt_bin = ntile(wt, 5)) |>
  group_by(wt_bin) |>
  summarise(mpg_median = median(mpg), .groups = "drop") |>
  ggplot(aes(x = wt_bin, y = mpg_median)) +
```

```
  geom_line() +
  geom_point() +
  labs(x = "Weight bin (quintiles)", y = "Median mpg")
```



3.4.3 Association Heterogeneity

Associations can differ across subgroups. An overall correlation might hide a strong relationship within a subgroup or even mask a reversal (Simpson's paradox). Descriptive analysis can therefore benefit from reporting **stratified associations** when meaningful groupings exist.

3.5 Outliers, Extremes, and Influence

Outliers are not always errors. They often carry substantive meaning: high-risk patients, exceptional transactions, or rare events. It can be helpful to treat outliers as *signals* first, and errors second.

Key descriptive checks include:

- **Tail inspection**: list the largest/smallest observations
- **Influence screening**: compare summaries with and without extreme values
- **Robust summaries**: medians, trimmed means, and MAD

variable	missing_rate
Ozone	24.2%
Solar.R	4.6%
Wind	0.0%
Temp	0.0%
Month	0.0%
Day	0.0%

A practical workflow is to compute both standard and robust summaries side by side. Large divergence is a flag that distributional extremes matter.

3.6 Missing Data as Descriptive Information

Missingness is itself informative. The *pattern* of missing data can reveal survey fatigue, data collection issues, or systematic exclusion of certain groups.

Descriptive checks include:

- **Missingness rates by variable**
- **Missingness by subgroup** (e.g., higher nonresponse among certain demographics)
- **Co-missingness patterns** (variables missing together)

Understanding missingness patterns is often a prerequisite for credible descriptive analysis because it reveals which parts of the data are under-observed or biased.

To illustrate missing-data diagnostics, we use `airquality`, another built-in R dataset containing daily New York air quality measurements (May to September 1973), including ozone, solar radiation, wind, and temperature, with known missing values in several variables.

```
# Simple missingness profile
airquality |>
  summarise(across(everything(), ~ mean(is.na(.x)))) |>
  pivot_longer(everything(), names_to = "variable", values_to = "missing_rate") |>
  arrange(desc(missing_rate)) |>
  gt() |>
  fmt_percent(columns = missing_rate, decimals = 1)
```

```
# Co-missingness count matrix
miss_mat <- airquality |> mutate(across(everything(), is.na))
co_miss <- t(as.matrix(miss_mat)) %*% as.matrix(miss_mat)
```

variable	Ozone	Solar.R	Wind	Temp	Month	Day
Ozone	37	2	0	0	0	0
Solar.R	2	7	0	0	0	0
Wind	0	0	0	0	0	0
Temp	0	0	0	0	0	0
Month	0	0	0	0	0	0
Day	0	0	0	0	0	0

```
co_miss |>
  as.data.frame() |>
  tibble::rownames_to_column(var = "variable") |>
  gt()
```

3.7 Scaling, Standardization, and Comparability

When comparing variables on different scales, raw summaries can mislead. Standardization puts variables on a common metric:

$$Z = \frac{X - \mu}{\sigma}.$$

However, standardization is not always desirable. For skewed or heavy-tailed distributions, **robust scaling** using medians and MAD can be more appropriate (Huber 1981):

$$Z_{\text{robust}} = \frac{X - \text{median}(X)}{\text{MAD}(X)}.$$

Standardization is especially useful when building **profiles** of observations across many variables, a topic revisited in later chapters on clustering and tree-based methods.

3.8 Multivariate Profiles and "Descriptive Models"

As dimensionality increases, summaries often need to become multivariate. Two simple but powerful tools are:

1. **Profile tables**: compare multiple variables across key subgroups (e.g., demographic segments)

2. **Composite indices**: average or weighted sums of standardized variables to create a high-level descriptive score

Composite indices are not causal models; they are descriptive constructs that summarize a multidimensional concept (e.g., socioeconomic status, health risk, engagement intensity). It helps to maintain transparency in construction for interpretability.

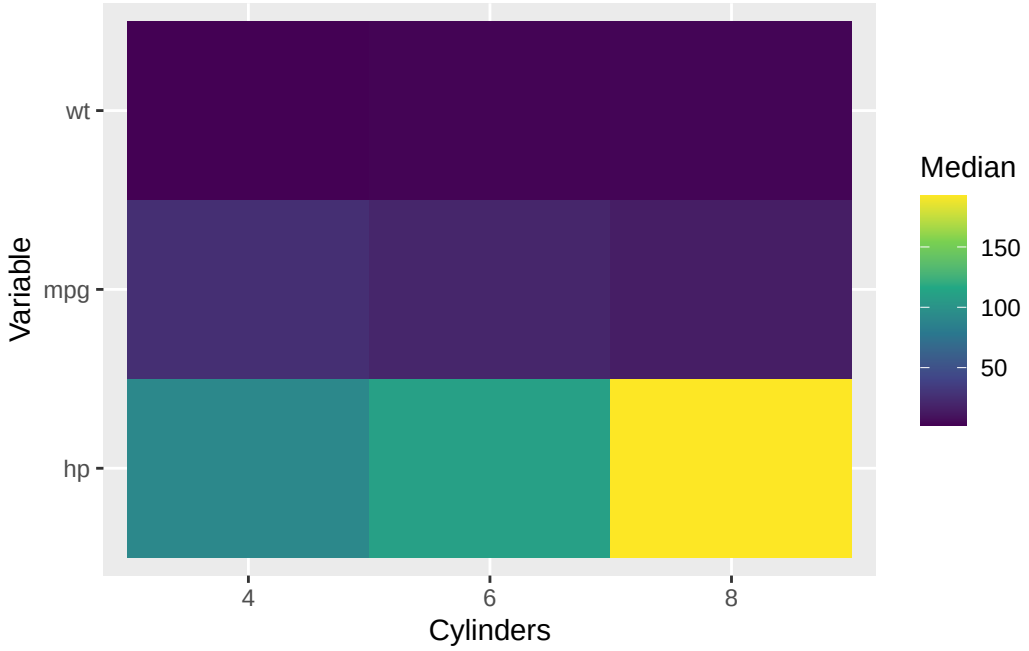
3.9 Visual Descriptives That Scale

Certain visual tools provide richer descriptive information than conventional charts:

- **Violin plots**: show full distribution and density
- **Boxen plots**: emphasize tails across many observations
- **Ridge plots**: compare distributions across many groups
- **Heatmaps**: visualize large tables of summary statistics
- **ECDFs**: compare distributions without binning

These graphics remain descriptive but give a more faithful sense of distributional complexity and subgroup structure.

```
# Heatmap of median summary statistics across groups
mtcars |>
  select(mpg, hp, wt, cyl) |>
  group_by(cyl) |>
  summarise(across(c(mpg, hp, wt), median), .groups = "drop") |>
  pivot_longer(-cyl, names_to = "variable", values_to = "median") |>
  ggplot(aes(x = factor(cyl), y = variable, fill = median)) +
  geom_tile() +
  scale_fill_viridis_c() +
  labs(x = "Cylinders", y = "Variable", fill = "Median")
```



3.10 A Practical Workflow

A robust descriptive workflow often follows this sequence:

1. **Univariate inspection**: quantiles, density plots, robust summaries
2. **Missingness mapping**: rates, patterns, co-missingness
3. **Bivariate exploration**: conditional summaries, scatterplots with smoothers
4. **Stratified checks**: subgroup comparisons, heterogeneity in associations
5. **Multivariate summaries**: profiles and composite indices

This workflow remains entirely descriptive while systematically uncovering structure that basic summary tables would miss.

3.11 Example: A Small R Template

The following minimal template illustrates how to go beyond means and SDs with a few descriptive enhancements:

```
# Robust summary for a numeric variable (tidy output)
summary_stats <- function(x) {
  tibble(
```

cyl	mean	median	sd	mad	q10	q90
4	26.66	26.00	4.51	6.52	21.50	32.40
6	19.74	19.70	1.45	1.93	17.98	21.16
8	15.10	15.20	2.56	1.56	11.27	18.28

```
    mean = mean(x, na.rm = TRUE),
    median = median(x, na.rm = TRUE),
    sd = sd(x, na.rm = TRUE),
    mad = mad(x, na.rm = TRUE),
    q10 = quantile(x, 0.10, na.rm = TRUE),
    q90 = quantile(x, 0.90, na.rm = TRUE)
  )
}

# Conditional summaries by group
mtcars |>
  group_by(cyl) |>
  summarise(summary_stats(mpg), .groups = "drop") |>
  gt() |>
  fmt_number(columns = c(mean, median, sd, mad, q10, q90), decimals = 2)
```

The aim is not sophistication but *discipline*: it helps to inspect robust, conditional, and distributional features before moving to advanced methods.

3.12 Summary and Key Takeaways

- **Basic summaries are insufficient**: Means and standard deviations conceal distributional shape, subgroup patterns, outliers, and nonlinear relationships.
- **Quantiles reveal more than averages**: Percentiles, IQR, and tail quantiles provide robust descriptions of where data actually live, especially in skewed distributions.
- **Robust measures reduce outlier sensitivity**: MAD, median-based scaling, and trimmed means provide stable descriptions when extremes are present.
- **Multimodality signals subpopulations**: Multiple peaks in density plots often reveal distinct regimes or groups worth examining separately.
- **Conditional statistics uncover heterogeneity**: Group-wise summaries and stratified associations reveal structure that overall statistics miss.
- **Nonlinear relationships require visual exploration**: LOESS smoothers, binned summaries, and ECDFs detect patterns that correlation coefficients cannot capture.

- **Missingness patterns are informative**: Understanding where and why data are missing is essential for credible descriptive analysis.
- **Standardization enables comparability**: When variables differ in scale, robust standardization or min-max scaling allows meaningful cross-variable comparisons.
- **Visual tools scale descriptive insight**: Violin plots, ridge plots, heatmaps, and ECDFs convey distributional complexity that tables cannot.

3.13 Looking Ahead

This chapter expands the descriptive mindset beyond simple averages. The next chapters formalize these ideas for mixed data types and systematic association measures. Once we correctly handle variable types, we can build **type-aware association matrices**, visualize them as networks, and scale descriptive analysis to hundreds of variables.

One key lesson is simple: **descriptive analysis improves when we stop summarizing variables in isolation and start describing structure**.

Part II

Association Analysis

4 Unified Association Measures for Mixed-Type Variables

4.1 Introduction: The Challenge of Mixed-Type Data

Real-world datasets rarely consist of purely continuous or purely categorical variables. A typical health survey contains continuous biomarkers, ordinal symptom scales, categorical diagnoses, and binary treatment indicators. A business database combines numeric sales figures, categorical product types, ordinal customer satisfaction ratings, and binary contract status.

A helpful framing question is: **How do we measure association across such mixed data in a unified, comparable way?**

Chapter 2 established that different variable types require different association measures. But practitioners often face a practical problem: given a dataset with mixed types, how do we compute a single association matrix that allows us to rank relationships, identify anomalies, and prepare for network analysis or further modeling?

This chapter presents a practical framework for unified association measurement. We aim to:

1. Survey classical type-specific measures and their limitations
2. Introduce modern alternatives that are more flexible and often more robust
3. Build a type-aware framework for selecting appropriate measures
4. Demonstrate implementation on real mixed-type data

4.2 Variable Type Classification

Association is not a single concept. The way we measure relationships depends on the **measurement scale** of the variables involved. A correlation coefficient makes sense for two continuous variables, but it becomes misleading when applied to categorical or ordinal data. A good descriptive workflow usually begins by classifying variable types and selecting association measures accordingly.

We classify variables into five types:

Type	Examples	Characteristics
Continuous	income, height, temperature	Interval/ratio scale, numeric measurements
Discrete	count data, number of events	Non-negative integers, often right-skewed
Ordinal	education level, Likert scales	Ordered categories, rank structure matters
Categorical	region, industry, color	Unordered labels, no inherent order
Binary	yes/no, 0/1, presence/absence	Two categories, often treatment or outcome indicator

The core goal is **comparability**: use measures that help us rank associations across mixed types without forcing invalid assumptions.

4.3 Classical Measures by Variable Type

Classical association measures are well-understood, computationally efficient, and interpretable. However, each is designed for specific type combinations. We review the most important measures organized by the types of variables they handle.

4.3.1 Continuous-Continuous: Pearson and Spearman

Pearson's r measures **linear association** between two continuous variables (Pearson 1895; Anderson 1985). It is scale-invariant, interpretable, and widely understood. However, it has significant limitations:

- Sensitive to outliers
- Assumes bivariate normality for inference (though we focus on description)
- Misses nonlinear relationships (two variables can be perfectly dependent with $r = 0$)

```
# Pearson correlation on mtcars
mtcars |>
    select(mpg, wt, hp, disp) |>
    cor(method = "pearson") |>
    as.data.frame() |>
    tibble::rownames_to_column(var = "Variable") |>
    gt() |>
    fmt_number(columns = -Variable, decimals = 3)
```

Variable	mpg	wt	hp	disp
mpg	1.000	-0.868	-0.776	-0.848
wt	-0.868	1.000	0.659	0.888
hp	-0.776	0.659	1.000	0.791
disp	-0.848	0.888	0.791	1.000

Measure	r
Pearson	0.880
Spearman	0.877

Spearman's ρ ranks data first, then computes Pearson's r on ranks (Spearman 1961). This makes it:

- Robust to outliers
- Sensitive to monotonic (not just linear) relationships
- Suitable for ordinal data

```
# Compare Pearson and Spearman on data with nonlinear association
set.seed(42)
data_nonlin <- tibble(
  x = seq(0, 4, length.out = 100),
  y = x^2 + rnorm(100, 0, 2)
)

tibble(
  Measure = c("Pearson", "Spearman"),
  r = c(
    cor(data_nonlin$x, data_nonlin$y, method = "pearson"),
    cor(data_nonlin$x, data_nonlin$y, method = "spearman")
  )
) |>
  gt() |>
  fmt_number(columns = r, decimals = 3)
```

Recommendation: Spearman is often a sensible default for continuous–continuous pairs due to its robustness; Pearson can be helpful when linearity is established and outliers are minimal.

V
0.523

4.3.2 Categorical-Categorical: Cramér's V

For two categorical variables, **Cramér's V** normalizes the chi-square statistic to $[0, 1]$ (David and Cramer 1947):

$$V = \sqrt{\frac{\chi^2}{n(k-1)}}$$

where $k = \min(\text{rows}, \text{columns})$. This is intuitive and widely used, though it tends to underestimate association in sparse tables.

```
# Cramér's V for categorical variables
cramers_v <- function(x, y) {
  tab <- table(x, y)
  chi2 <- suppressWarnings(chisq.test(tab, correct = FALSE)$statistic)
  n <- sum(tab)
  k <- min(nrow(tab), ncol(tab))
  sqrt(as.numeric(chi2) / (n * (k - 1)))
}

# Example: cylinder type vs transmission type
mtcars |>
  summarise(V = cramers_v(factor(cyl), factor(am))) |>
  gt() |>
  fmt_number(columns = V, decimals = 3)
```

4.3.3 Continuous-Categorical: Eta-Squared

When a continuous variable varies across groups defined by a categorical variable, **eta-squared** (η^2) captures the proportion of variance explained (S. R. A. Fisher 1990; J. Cohen 2013):

$$\eta^2 = \frac{\text{SS}_{\text{between}}}{\text{SS}_{\text{total}}}.$$

This is directly interpretable as effect size and ranges from 0 to 1.

eta2
0.732

```
# Eta-squared: mpg explained by number of cylinders
eta_squared <- function(y, group) {
  df <- tibble(y = y, group = group)
  grand_mean <- mean(df$y, na.rm = TRUE)
  ss_total <- sum((df$y - grand_mean)^2, na.rm = TRUE)
  ss_between <- df |>
    group_by(group) |>
    summarise(n = n(), mean_y = mean(y, na.rm = TRUE), .groups = "drop") |>
    mutate(ss = n * (mean_y - grand_mean)^2) |>
    summarise(ss = sum(ss)) |>
    pull(ss)
  ss_between / ss_total
}

mtcars |>
  summarise(eta2 = eta_squared(mpg, factor(cyl))) |>
  gt() |>
  fmt_number(columns = eta2, decimals = 3)
```

4.3.4 Ordinal Variables: Kendall's Tau

Ordinal variables carry rank information that is often best retained. **Kendall's** τ is a natural choice for ordinal–ordinal associations and is robust to ties (Kendall 1938). **Spearman's** ρ can also be used for ordinal data.

```
# Ordinal example using a cut of mpg
mtcars |>
  mutate(mpg_ordinal = cut(mpg, breaks = 4, ordered_result = TRUE)) |>
  summarise(
    spearman = cor(as.numeric(mpg_ordinal), wt, method = "spearman"),
    kendall = cor(as.numeric(mpg_ordinal), wt, method = "kendall")
  ) |>
  gt() |>
  fmt_number(columns = c(spearman, kendall), decimals = 3)
```

spearman	kendall
-0.777	-0.656

phi
0.168

4.3.5 Binary-Binary: Phi Coefficient

Binary variables can be treated as a special case of categorical data. For two binary indicators, the **phi coefficient** is equivalent to Pearson correlation on 0/1 coding.

```
# Phi coefficient for two binary variables
phi <- function(x, y) {
  cor(as.integer(x), as.integer(y))
}

mtcars |>
  mutate(am_bin = am == 1, vs_bin = vs == 1) |>
  summarise(phi = phi(am_bin, vs_bin)) |>
  gt() |>
  fmt_number(columns = phi, decimals = 3)
```

4.4 Modern Alternatives: Beyond Classical Measures

Classical measures work well within their designed domains but lack flexibility. We present modern association measures that offer broader applicability and often improved robustness.

4.4.1 Distance Correlation

Distance correlation (Székely, Rizzo, and Bakirov 2007) is a measure of dependence that:

- Works for any dimension (univariate, multivariate)
- Detects both linear and nonlinear associations
- Equals zero if and only if the variables are independent
- Is not sensitive to outliers in the same way as Pearson's r

The distance correlation is computed from **distance matrices** of the two variables by centering these matrices and computing their correlation.

dcor	pearson
0.871	-0.868

```
# Distance correlation via the energy package
library(energy)

  # Example
  mtcars |>
    summarise(
      dcor = dcor(mpg, wt),
      pearson = cor(mpg, wt, method = "pearson")
    ) |>
    gt() |>
    fmt_number(columns = c(dcor, pearson), decimals = 3)
```

Practical advantages:

- Captures nonlinear relationships
- Robust to outliers
- Theoretically appealing (zero iff independent)

Disadvantages:

- Computationally expensive for large samples
- Less interpretable than Pearson's r or Spearman's ρ

4.4.2 Maximal Information Coefficient (MIC)

The **Maximal Information Coefficient** (Reshef et al. 2011) measures association by finding the grid partition of the data that maximizes mutual information. It detects diverse functional relationships (linear, nonlinear, exponential, etc.) and is normalized to $[0, 1]$.

Strengths:

- Detects diverse functional relationships
- Symmetric and normalized
- No assumption about relationship shape

Weaknesses:

- Computationally demanding
- Can be noisy with small sample sizes

Measure	Value
MIC	0.729

- Requires specialized software

```
# MIC example (requires the minerva package)
  library(minerva)

  # Compute MIC for a pair
  set.seed(42)
  x <- rnorm(200)
  y <- x^2 + rnorm(200, 0, 0.5)

  mic_result <- mine(x, y)
  tibble(
    Measure = "MIC",
    Value = mic_result$MIC
  ) |>
    gt() |>
    fmt_number(columns = Value, decimals = 3)
```

4.4.3 Mutual Information

Mutual information (MI) quantifies the amount of information one variable contains about another (Shannon 1948; Cover and Thomas 2001):

$$I(X;Y) = \sum_x \sum_y p(x,y) \log \frac{p(x,y)}{p(x)p(y)}.$$

Mutual information is:

- Symmetric and non-negative
- Works for any combination of variable types
- Invariant to monotonic transformations
- Zero if and only if variables are independent

Interpretation challenge: MI is measured in "nats" or "bits" and lacks an intuitive 0-1 scale, making comparisons across variable pairs difficult without normalization.

nmi
0.576

```r
# Normalized mutual information
mutual_info_norm <- function(x, y) {
  # Discretize continuous variables if needed
  x_cat <- if (is.numeric(x)) cut(x, breaks = 5) else factor(x)
  y_cat <- if (is.numeric(y)) cut(y, breaks = 5) else factor(y)

  tab <- table(x_cat, y_cat)
  px <- rowSums(tab) / sum(tab)
  py <- colSums(tab) / sum(tab)
  pxy <- tab / sum(tab)

  # Mutual information
  mi <- 0
  for (i in 1:nrow(pxy)) {
    for (j in 1:ncol(pxy)) {
      if (pxy[i, j] > 0) {
        mi <- mi + pxy[i, j] * log(pxy[i, j] / (px[i] * py[j]))
      }
    }
  }

  # Normalize by entropy
  hx <- -sum(px * log(px + 1e-10))
  hy <- -sum(py * log(py + 1e-10))
  2 * mi / (hx + hy)
}

# Example
mtcars |>
  summarise(
    nmi = mutual_info_norm(mpg, wt)
  ) |>
  gt() |>
  fmt_number(columns = nmi, decimals = 3)
```

kendall_tau	spearman_rho	pearson_r
-0.728	-0.886	-0.868

4.4.4 Copula-Based Measures

Copulas separate marginal distributions from the dependence structure (Sklar 1959). Common copula-derived measures include **Kendall's tau** and **Spearman's rho**, which are rank-based and capture dependence structure independent of marginal scales.

```
# Kendall's tau and Spearman's rho
mtcars |>
  summarise(
    kendall_tau = cor(mpg, wt, method = "kendall"),
    spearman_rho = cor(mpg, wt, method = "spearman"),
    pearson_r = cor(mpg, wt, method = "pearson")
  ) |>
  gt() |>
  fmt_number(columns = everything(), decimals = 3)
```

4.5 A Type-Aware Framework for Mixed Data

Given the diversity of measures, we need a practical approach that applies the most appropriate measure to each variable pair based on their types.

4.5.1 Association Measure Selection Matrix

The table below summarizes recommended measures for each type combination. This consolidated matrix serves as the primary reference for selecting appropriate measures:

X → Y	Continuous	Discrete	Ordinal	Categorical	Binary
Continuous	Spearman (default), Pearson, distance correlation	Spearman, distance correlation	Spearman, polyserial	η^2, distance correlation	Point-biserial, η^2
Discrete	Spearman, distance correlation	Spearman	Spearman	η^2, distance correlation	η^2

X → Y	Continuous	Discrete	Ordinal	Categorical	Binary
Ordinal	Spearman, polyserial	Spearman	Kendall's τ, Spearman	Polychoric, Cramér's V	Rank-biserial
Categorical	η^2, distance correlation	η^2, distance correlation	Polychoric, Cramér's V	Cramér's V, mutual information	ϕ, Cramér's V
Binary	Point-biserial, η^2	η^2	Rank-biserial	ϕ, Cramér's V	ϕ

Rationale for selections:

- **Continuous-Continuous**: Use Spearman as default (robust to outliers); add distance correlation to detect nonlinearity
- **Continuous-Categorical**: Use η^2 to measure variance explained by group membership
- **Categorical-Categorical**: Use Cramér's V (normalized chi-square)
- **Binary pairs**: Use phi coefficient (equivalent to Pearson on 0/1 encoding)
- **Mixed discrete/ordinal**: Prefer rank-based measures

4.5.2 Implementation: Type-Aware Association Matrix

We now implement a complete workflow that classifies variables, selects appropriate measures, and computes a unified association matrix.

```
# Define automatic variable classification
classify_variable <- function(x, n_unique_threshold = 10) {
  # Remove NA for classification
  x_clean <- x[!is.na(x)]
  n_unique <- n_distinct(x_clean)

  if (is.numeric(x)) {
    if (all(x_clean == as.integer(x_clean)) && min(x_clean, na.rm = TRUE) >= 0) {
      if (n_unique <= n_unique_threshold) "ordinal" else "discrete"
    } else {
      "continuous"
    }
  } else if (is.factor(x) || is.character(x)) {
    if (n_unique == 2) "binary" else "categorical"
  } else {
    "unknown"
  }
}
```

```r
# Define type-aware association computation
compute_association <- function(x, y, type_x, type_y) {
  # Remove pairs with missing data
  valid_idx <- !is.na(x) & !is.na(y)
  x <- x[valid_idx]
  y <- y[valid_idx]

  if (length(x) == 0) return(NA_real_)

  # Handle numeric conversion for type checking
  if (is.logical(x)) x <- as.integer(x)
  if (is.logical(y)) y <- as.integer(y)

  # Continuous-Continuous, Continuous-Discrete, or Continuous-Ordinal
  if ((type_x %in% c("continuous", "discrete")) &&
      (type_y %in% c("continuous", "discrete", "ordinal"))) {
    return(cor(x, as.numeric(y), method = "spearman", use = "complete.obs"))
  }

  if ((type_y %in% c("continuous", "discrete")) &&
      (type_x %in% c("continuous", "discrete", "ordinal"))) {
    return(cor(as.numeric(x), y, method = "spearman", use = "complete.obs"))
  }

  # Continuous-Categorical or Continuous-Binary (Eta-squared)
  if ((type_x == "continuous" && type_y %in% c("categorical", "binary")) ||
      (type_y == "continuous" && type_x %in% c("categorical", "binary"))) {
    if (type_x %in% c("categorical", "binary")) {
      return(eta_squared(y, x))
    } else {
      return(eta_squared(x, y))
    }
  }

  # Categorical-Categorical, Categorical-Binary, or Ordinal-Categorical
  if ((type_x %in% c("categorical", "binary", "ordinal")) &&
      (type_y %in% c("categorical", "binary", "ordinal"))) {
    return(cramers_v(factor(x), factor(y)))
  }

  return(NA_real_)
}
```

var_name	var_type
mpg	continuous
wt	continuous
hp	discrete
cyl_cat	categorical
am_bin	binary
gear_cat	categorical

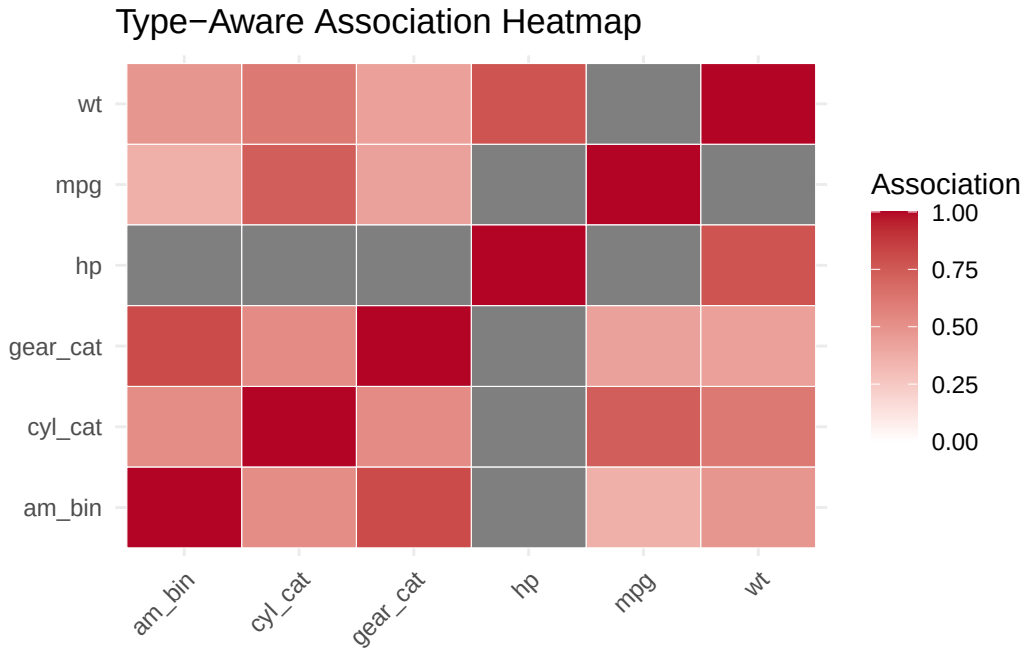
```
# Prepare mtcars with mixed types
mtcars_mixed <- mtcars |>
  mutate(
    cyl_cat = factor(cyl),
    am_bin = factor(am),
    vs_bin = factor(vs),
    gear_cat = factor(gear)
  ) |>
  select(mpg, wt, hp, cyl_cat, am_bin, gear_cat)

# Classify variables
var_types <- tibble(
  var_name = names(mtcars_mixed),
  var_type = map_chr(mtcars_mixed, classify_variable)
)

var_types |>
  gt()
```

```
# Compute association matrix
vars <- names(mtcars_mixed)
n_vars <- length(vars)
assoc_matrix <- matrix(1.0, nrow = n_vars, ncol = n_vars)
rownames(assoc_matrix) <- vars
colnames(assoc_matrix) <- vars

for (i in 1:n_vars) {
  for (j in i:n_vars) {
    if (i == j) {
      assoc_matrix[i, j] <- 1.0
    } else {
      assoc <- compute_association(
```

Variable	mpg	wt	hp	cyl_cat	am_bin	gear_cat
mpg	1.000	-0.886	-0.895	0.732	0.360	0.429
wt	-0.886	1.000	0.775	0.612	0.480	0.434
hp	-0.895	0.775	1.000	NA	NA	NA
cyl_cat	0.732	0.612	NA	1.000	0.523	0.531
am_bin	0.360	0.480	NA	0.523	1.000	0.809
gear_cat	0.429	0.434	NA	0.531	0.809	1.000

```
        mtcars_mixed[[i]], mtcars_mixed[[j]],
        var_types$var_type[i], var_types$var_type[j]
      )
      assoc_matrix[i, j] <- assoc
      assoc_matrix[j, i] <- assoc
    }
  }
}

# Display association matrix as table
assoc_matrix |>
  as.data.frame() |>
  tibble::rownames_to_column(var = "Variable") |>
  gt() |>
  fmt_number(columns = -Variable, decimals = 3)
```

```
# Display as heatmap
assoc_matrix |>
  as.data.frame() |>
  tibble::rownames_to_column(var = "x") |>
  pivot_longer(-x, names_to = "y", values_to = "assoc") |>
  ggplot(aes(x = x, y = y, fill = assoc)) +
  geom_tile(color = "white") +
  scale_fill_gradient2(low = "#3b4cc0", mid = "white", high = "#b40426",
                       limits = c(0, 1), breaks = seq(0, 1, 0.25)) +
  labs(
    title = "Type-Aware Association Heatmap",
    x = NULL, y = NULL, fill = "Association"
  ) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



4.6 Normalization and Comparability

Different measures operate on different scales. Some are naturally bounded to $[0, 1]$; others are unbounded. To compare associations across variable pairs, we often need to normalize.

4.6.1 Standard Normalization Approaches

1. **Min-max scaling**: Scale a measure to $[0, 1]$ using its theoretical bounds

$$\text{Normalized} = \frac{\text{Measure} - \min}{\max - \min}$$

2. **Absolute value**: For measures that can be negative (e.g., correlation), use absolute value if directionality is not of interest

$$\text{Normalized} = |\text{Measure}|$$

3. **Fisher z-transformation**: For Pearson's r specifically, use $z = \frac{1}{2} \ln\left(\frac{1+r}{1-r}\right)$ to stabilize variance
4. **Mutual information normalization**: Divide by maximum possible MI or entropy

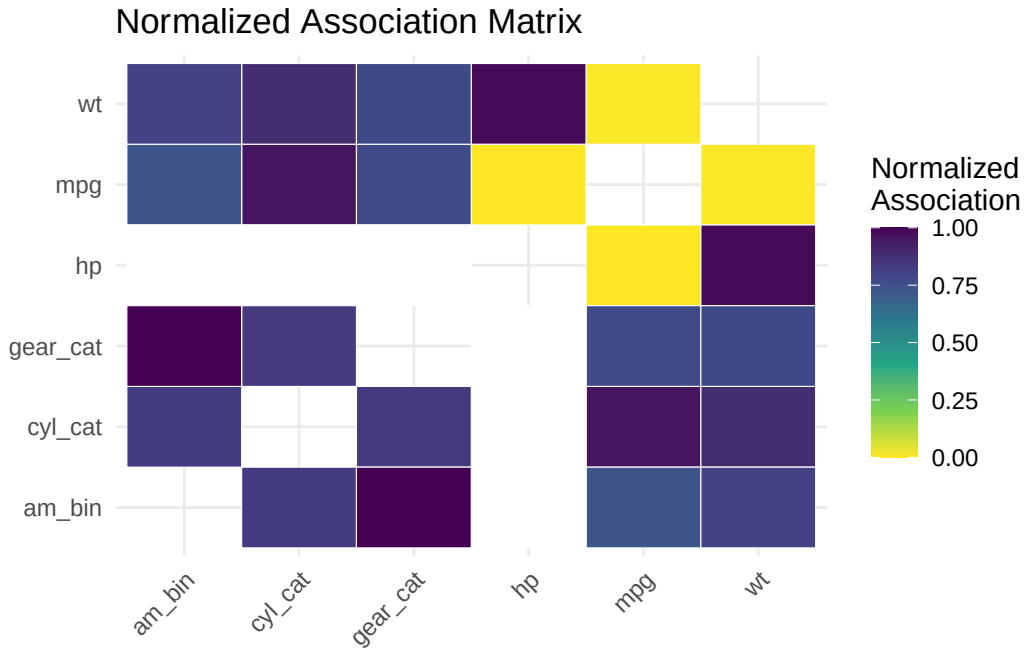
$$\text{Normalized MI} = \frac{I(X;Y)}{\min(H(X), H(Y))}$$

4.6.2 Important Caveats

Normalization enables **ranking** but **not direct comparison across measure types**. A normalized value of 0.6 for Cramér's V is not equivalent to 0.6 for Spearman's ρ. It helps to:

- Document your normalization choice
- Interpret results within measure families first
- Use normalization to identify anomalies and ranking, not to claim equivalence

```r
# Example: normalize association matrix to [0, 1]
assoc_matrix_norm <- assoc_matrix
diag(assoc_matrix_norm) <- NA  # Exclude diagonal (all 1s)

# Normalize each off-diagonal value
min_assoc <- min(assoc_matrix_norm, na.rm = TRUE)
max_assoc <- max(assoc_matrix_norm, na.rm = TRUE)

assoc_matrix_norm <- (assoc_matrix_norm - min_assoc) / (max_assoc - min_assoc)
diag(assoc_matrix_norm) <- 1  # Restore diagonal

# Heatmap of normalized values
assoc_matrix_norm |>
  as.data.frame() |>
  tibble::rownames_to_column(var = "x") |>
  pivot_longer(-x, names_to = "y", values_to = "assoc") |>
  filter(x != y) |>  # Exclude diagonal for clarity
  ggplot(aes(x = x, y = y, fill = assoc)) +
  geom_tile(color = "white") +
  scale_fill_viridis_c(direction = -1, na.value = "white") +
  labs(
    title = "Normalized Association Matrix",
    x = NULL, y = NULL, fill = "Normalized\nAssociation"
  ) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



4.7 Handling Special Cases

4.7.1 Missing Data

Associations with missing data require explicit handling:

1. **Listwise deletion**: Remove all rows with any missing values (simple but wasteful)
2. **Pairwise deletion**: Compute each association only from complete cases for that pair (more data, but samples vary by pair)
3. **Imputation**: Replace missing values with estimates before computing associations

```
# Example: comparing pairwise deletion methods
mtcars_with_na <- mtcars |>
  mutate(
    mpg = ifelse(row_number() %in% c(5, 10), NA, mpg),
    wt = ifelse(row_number() %in% c(3, 8), NA, wt)
  )

tibble(
  Method = c("Listwise", "Pairwise (mpg-wt)"),
  N = c(
    nrow(na.omit(mtcars_with_na[, c("mpg", "wt")])),
```

Method	N	Correlation
Listwise	28	-0.875
Pairwise (mpg-wt)	28	-0.875

```
    sum(!is.na(mtcars_with_na$mpg) & !is.na(mtcars_with_na$wt))
  ),
  Correlation = c(
    cor(na.omit(mtcars_with_na[, c("mpg", "wt")])[, 1],
        na.omit(mtcars_with_na[, c("mpg", "wt")])[, 2]),
    cor(mtcars_with_na$mpg, mtcars_with_na$wt, use = "complete.obs")
  )
) |>
  gt() |>
  fmt_number(columns = c(Correlation), decimals = 3)
```

4.7.2 Outliers and Robustness

Outliers can distort Pearson correlations but have less impact on rank-based measures. When outliers are present:

- Use Spearman or Kendall instead of Pearson
- Inspect scatterplots for unusual patterns
- Consider robust alternatives (winsorization, trimmed correlation)

```
# Impact of outliers
set.seed(42)
x <- rnorm(50)
y <- 0.7 * x + rnorm(50, 0, 0.5)

# Add outliers
x_outlier <- c(x, 10, 10)
y_outlier <- c(y, -8, 12)

tibble(
  Scenario = c("No outliers", "With outliers"),
  Pearson = c(
    cor(x, y, method = "pearson"),
    cor(x_outlier, y_outlier, method = "pearson")
  ),
```

Scenario	Pearson	Spearman
No outliers	0.840	0.845
With outliers	0.310	0.751

```
  Spearman = c(
    cor(x, y, method = "spearman"),
    cor(x_outlier, y_outlier, method = "spearman")
  )
) |>
  gt() |>
  fmt_number(columns = c(Pearson, Spearman), decimals = 3)
```

4.8 Practical Workflow Summary

Here is a step-by-step workflow for building a unified association matrix:

1. **Classify variables** by type (continuous, discrete, ordinal, categorical, binary)
2. **Handle missing data** explicitly (document your choice)
3. **Inspect distributions** for outliers, skewness, and multimodality
4. **Select measures** based on the type-aware selection matrix above
5. **Compute pairwise associations** with appropriate measures
6. **Normalize** if comparability across pairs is desired
7. **Visualize** as a heatmap or network (see later chapters)
8. **Interpret in context** (avoid overstating quantitative results)

4.9 Summary and Key Takeaways

- **Variable types matter**: Different measurement scales require different association measures to avoid misleading results.
- **Classical measures** (Pearson, Spearman, Cramér's V, η^2) are efficient, interpretable, and widely understood, but each is specialized for specific type combinations.
- **Modern alternatives** (distance correlation, MIC, mutual information) offer greater flexibility and can detect nonlinear relationships, but are often more computationally expensive and less interpretable.
- **Type-aware framework**: A systematic approach to selecting appropriate measures based on variable type combinations ensures valid and comparable association estimates.
- **Normalization enables ranking** but does not erase conceptual differences between measures. It helps to document our choices and interpret results carefully.

- **Special cases** (missing data, outliers) require explicit handling. Rank-based measures are more robust than Pearson correlation to outliers.

4.10 Looking Ahead

With a unified framework for association measurement now in hand, the next chapter extends this work to **nonlinear and conditional associations**. We plan to explore copulas, partial correlations, and graphical models to capture more complex dependence structures. Then, we move to **network representations** that visualize multivariate association patterns in a format that stakeholders can interpret and act upon.

5 Extensions of Correlation: Nonlinear and Conditional

5.1 Introduction: Why Standard Correlation Falls Short

Pearson and Spearman correlations are powerful descriptive tools, but they capture only *marginal* associations (Pearson 1895; Anderson 1985; Spearman 1961). In real data, relationships often depend on context: a strong positive association for one subgroup may disappear or reverse in another. Additionally, standard correlations assume that relationships are *linear* (or at least monotonic), which can miss complex functional forms that drive patterns in nonlinear systems.

This chapter extends the correlation toolkit in two directions:

1. **Nonlinear associations**: Methods that detect and describe relationships that curve, oscillate, or follow other complex patterns
2. **Conditional associations**: Techniques that reveal how associations *change* across levels of other variables or contextual conditions

5.2 Part I: Nonlinear Association Methods

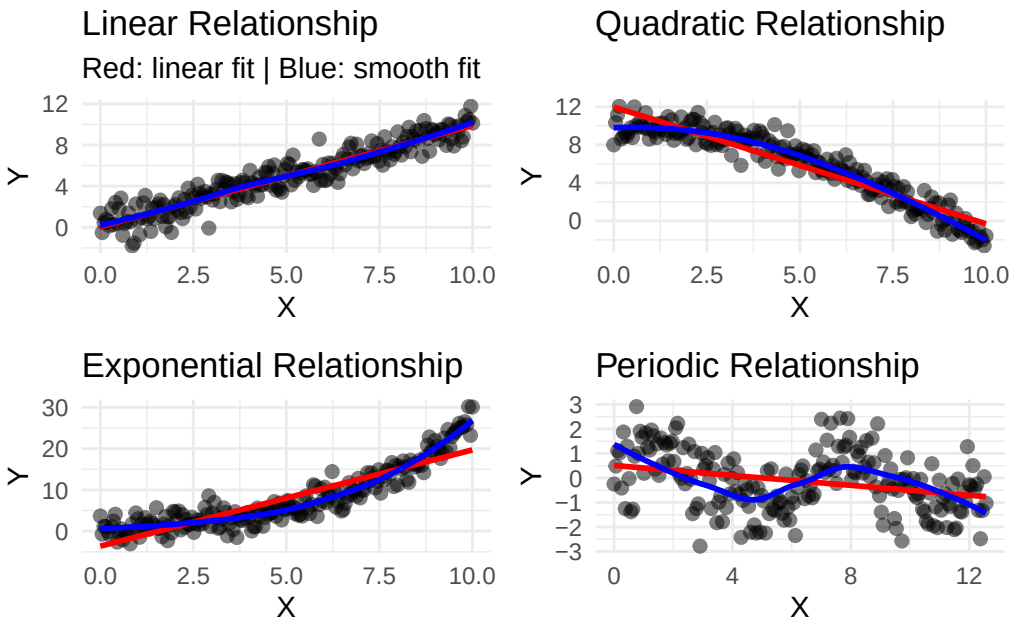
5.2.1 Visual Detection of Nonlinearity

Before applying formal measures, it helps to start with visual inspection. A scatterplot with a smooth trend line can reveal whether linearity is a reasonable assumption.

```
# Create dataset with various nonlinear relationships
set.seed(42)
n <- 200

nonlin_data <- tibble(
  Linear = seq(0, 10, length.out = n),
  Quadratic = seq(0, 10, length.out = n),
  Exponential = seq(0, 10, length.out = n),
  Sine = seq(0, 4 * pi, length.out = n),
```

```
  x_lin = rnorm(n, 0, 1),
  x_quad = rnorm(n, 0, 1),
  x_exp = rnorm(n, 0, 1),
  x_sin = rnorm(n, 0, 1)
) |>
  mutate(
    y_lin = Linear + x_lin,
    y_quad = 10 - Quadratic^2 / 8 + x_quad,
    y_exp = exp(Exponential / 3) + x_exp * 2,
    y_sin = sin(Sine) + x_sin
  )

# Visualize multiple relationship types
p1 <- nonlin_data |>
  ggplot(aes(x = Linear, y = y_lin)) +
  geom_point(alpha = 0.5, size = 2) +
  geom_smooth(method = "lm", se = FALSE, color = "red", linewidth = 1) +
  geom_smooth(method = "loess", se = FALSE, color = "blue", linewidth = 1) +
  labs(
    title = "Linear Relationship",
    x = "X", y = "Y",
    subtitle = "Red: linear fit | Blue: smooth fit"
  ) +
  theme_minimal()

p2 <- nonlin_data |>
  ggplot(aes(x = Quadratic, y = y_quad)) +
  geom_point(alpha = 0.5, size = 2) +
  geom_smooth(method = "lm", se = FALSE, color = "red", linewidth = 1) +
  geom_smooth(method = "loess", se = FALSE, color = "blue", linewidth = 1) +
  labs(
    title = "Quadratic Relationship",
    x = "X", y = "Y"
  ) +
  theme_minimal()

p3 <- nonlin_data |>
  ggplot(aes(x = Exponential, y = y_exp)) +
  geom_point(alpha = 0.5, size = 2) +
  geom_smooth(method = "lm", se = FALSE, color = "red", linewidth = 1) +
  geom_smooth(method = "loess", se = FALSE, color = "blue", linewidth = 1) +
  labs(
```

```
    title = "Exponential Relationship",
    x = "X", y = "Y"
  ) +
  theme_minimal()

p4 <- nonlin_data |>
  ggplot(aes(x = Sine, y = y_sin)) +
  geom_point(alpha = 0.5, size = 2) +
  geom_smooth(method = "lm", se = FALSE, color = "red", linewidth = 1) +
  geom_smooth(method = "loess", se = FALSE, color = "blue", linewidth = 1) +
  labs(
    title = "Periodic Relationship",
    x = "X", y = "Y"
  ) +
  theme_minimal()

(p1 + p2) / (p3 + p4)
```



Interpretation: When the blue smooth curve (locally weighted scatterplot smoothing, LOESS) deviates substantially from the red linear fit, nonlinearity is present (Cleveland 1979). This visual diagnostic often provides helpful context before quantitative analysis.

5.2.2 Quantifying Nonlinearity: The R^2 Contrast

A simple descriptive measure of nonlinearity is to compare the R^2 from a **linear model** to the R^2 from a **smooth nonparametric model** (such as LOESS or a spline):

$$\text{Nonlinearity Index} = R^2_{\text{smooth}} - R^2_{\text{linear}}.$$

The difference indicates how much variance can be explained by nonlinear structure that the linear model misses.

```
# Function to compute nonlinearity index
nonlinearity_index <- function(x, y) {
  df <- tibble(x = x, y = y) |>
    na.omit()

  # Linear model
  lm_fit <- lm(y ~ x, data = df)
  r2_linear <- summary(lm_fit)$r.squared

  # LOESS smooth fit
  loess_fit <- loess(y ~ x, data = df)
  r2_smooth <- 1 - (sum(loess_fit$residuals^2) / sum((df$y - mean(df$y))^2))

  tibble(
    R2_linear = r2_linear,
    R2_smooth = r2_smooth,
    Nonlinearity_Index = r2_smooth - r2_linear
  )
}

# Apply to our synthetic data
comparison <- bind_rows(
  nonlinearity_index(nonlin_data$Linear, nonlin_data$y_lin) |>
    mutate(Relationship = "Linear"),
  nonlinearity_index(nonlin_data$Quadratic, nonlin_data$y_quad) |>
    mutate(Relationship = "Quadratic"),
  nonlinearity_index(nonlin_data$Exponential, nonlin_data$y_exp) |>
    mutate(Relationship = "Exponential"),
  nonlinearity_index(nonlin_data$Sine, nonlin_data$y_sin) |>
    mutate(Relationship = "Periodic")
)
```

R2_linear	R2_smooth	Nonlinearity_Index	Relationship
0.897	0.899	0.002	Linear
0.883	0.939	0.057	Quadratic
0.801	0.924	0.122	Exponential
0.095	0.284	0.189	Periodic

```
comparison |>
  gt() |>
  fmt_number(columns = c(R2_linear, R2_smooth, Nonlinearity_Index), decimals = 3)
```

Interpretation: A high nonlinearity index can suggest that a linear model substantially underestimates the association strength. However, this measure is best **paired with visual inspection**, as sometimes the smooth fit overfits noise.

5.2.3 Distance Correlation: A Revisit with Nonlinearity in Focus

Recall from Chapter 4 that **distance correlation** detects both linear and nonlinear associations. Let's examine how it performs on nonlinear data:

```
# Distance correlation comparison
library(energy)

comparison_dcor <- bind_rows(
  tibble(
    Relationship = "Linear",
    Pearson = cor(nonlin_data$Linear, nonlin_data$y_lin, method = "pearson"),
    Spearman = cor(nonlin_data$Linear, nonlin_data$y_lin, method = "spearman"),
    DistanceCorr = dcor(nonlin_data$Linear, nonlin_data$y_lin)
  ),
  tibble(
    Relationship = "Quadratic",
    Pearson = cor(nonlin_data$Quadratic, nonlin_data$y_quad, method = "pearson"),
    Spearman = cor(nonlin_data$Quadratic, nonlin_data$y_quad, method = "spearman"),
    DistanceCorr = dcor(nonlin_data$Quadratic, nonlin_data$y_quad)
  ),
  tibble(
    Relationship = "Exponential",
    Pearson = cor(nonlin_data$Exponential, nonlin_data$y_exp, method = "pearson"),
    Spearman = cor(nonlin_data$Exponential, nonlin_data$y_exp, method = "spearman"),
    DistanceCorr = dcor(nonlin_data$Exponential, nonlin_data$y_exp)
```

Relationship	Pearson	Spearman	DistanceCorr
Linear	0.947	0.955	0.946
Quadratic	-0.939	-0.942	0.947
Exponential	0.895	0.915	0.907
Periodic	-0.307	-0.315	0.332

```
  ),
  tibble(
    Relationship = "Periodic",
    Pearson = cor(nonlin_data$Sine, nonlin_data$y_sin, method = "pearson"),
    Spearman = cor(nonlin_data$Sine, nonlin_data$y_sin, method = "spearman"),
    DistanceCorr = dcor(nonlin_data$Sine, nonlin_data$y_sin)
  )
)

comparison_dcor |>
  gt() |>
  fmt_number(columns = c(Pearson, Spearman, DistanceCorr), decimals = 3)
```

Observation: Distance correlation captures all relationship types more uniformly than Pearson or Spearman, which may completely miss periodic patterns.

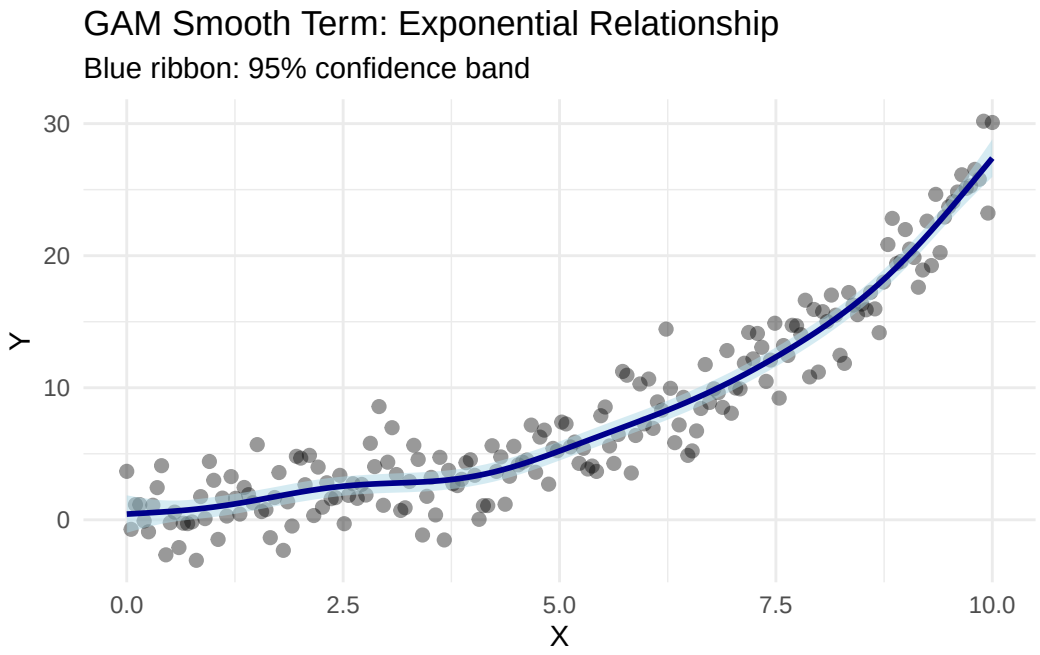
5.2.4 Generalized Additive Models (GAMs) for Smooth Associations

Generalized Additive Models extend linear regression by allowing smooth functions instead of linear terms (Hastie and Tibshirani 2017). For description, GAMs serve two purposes:

1. Visualize the functional form of a relationship
2. Quantify association strength via model diagnostics

```
# Fit GAM to exponential data
gam_fit <- gam(y_exp ~ s(Exponential), data = nonlin_data)

# Extract and visualize smooth term
plot_data <- tibble(
  Exponential = seq(min(nonlin_data$Exponential),
                    max(nonlin_data$Exponential),
                    length.out = 100)
)
```

```r
# Predict from GAM
preds <- predict(gam_fit, newdata = plot_data, se.fit = TRUE)

plot_data <- plot_data |>
  mutate(
    fit = preds$fit,
    se = preds$se.fit,
    upper = fit + 1.96 * se,
    lower = fit - 1.96 * se
  )

# Visualize with original data
ggplot(nonlin_data, aes(x = Exponential, y = y_exp)) +
  geom_point(alpha = 0.4, size = 2) +
  geom_ribbon(data = plot_data, aes(y = fit, ymin = lower, ymax = upper),
              fill = "lightblue", color = NA, alpha = 0.5) +
  geom_line(data = plot_data, aes(y = fit), color = "darkblue", linewidth = 1) +
  labs(
    title = "GAM Smooth Term: Exponential Relationship",
    x = "X", y = "Y",
    subtitle = "Blue ribbon: 95% confidence band"
  ) +
  theme_minimal()
```



Metric	Value
Deviance Explained	92.7%
GCV Score	4.509

```
# Model summary
tibble(
  Metric = c("Deviance Explained", "GCV Score"),
  Value = c(
    paste0(round(summary(gam_fit)$dev.expl * 100, 1), "%"),
    round(gam_fit$gcv.ubre, 3)
  )
) |>
  gt()
```

Interpretation:

- **Deviance Explained** shows how much variance the GAM accounts for (analogous to R^2)
- **GCV Score** (Generalized Cross-Validation) balances fit quality and complexity (lower is better)
- The smooth visualization reveals the precise functional form of the relationship

5.2.5 Additive Models and Interaction Effects

Beyond univariate smooths, GAMs allow **tensor product interactions** $s(x, z)$, capturing how the relationship between two variables depends on a third:

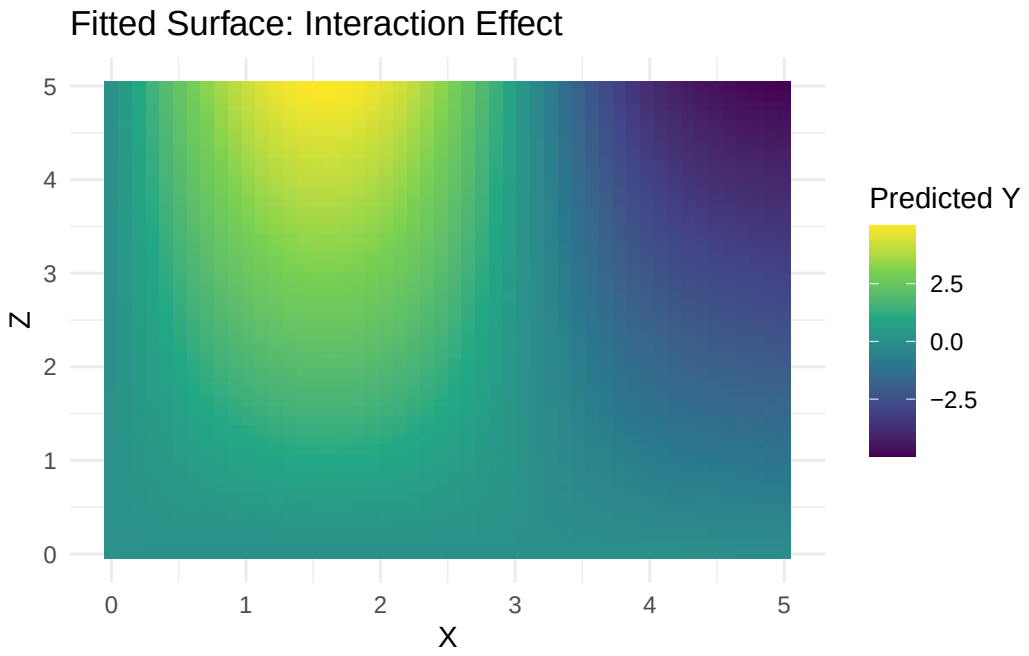
```
# Create data with interaction: y depends on x and z together
set.seed(42)
interaction_data <- expand_grid(
  x = seq(0, 5, length.out = 30),
  z = seq(0, 5, length.out = 30)
) |>
  mutate(
    y = sin(x) * z + rnorm(n(), 0, 0.3)
  )

# Fit GAM with tensor product interaction
gam_interact <- gam(y ~ ti(x) + ti(z) + ti(x, z),
                    data = interaction_data)
```

```r
# Visualize the interaction surface
summary(gam_interact)
```

```
Family: gaussian
Link function: identity

Formula:
y ~ ti(x) + ti(z) + ti(x, z)

Parametric coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.29400    0.01015   28.97   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:
          edf Ref.df      F p-value
ti(x)   3.995      4 7765.4  <2e-16 ***
ti(z)   1.000      1  303.7  <2e-16 ***
ti(x,z) 3.986      4 2697.3  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =  0.979   Deviance explained = 97.9%
GCV = 0.093733  Scale est. = 0.092693  n = 900
```

```r
# Contour plot of fitted surface
pred_grid <- expand_grid(
  x = seq(0, 5, length.out = 50),
  z = seq(0, 5, length.out = 50)
)
pred_grid$y_hat <- predict(gam_interact, newdata = pred_grid)

ggplot(pred_grid, aes(x = x, y = z, fill = y_hat)) +
  geom_tile() +
  scale_fill_viridis_c() +
  labs(
    title = "Fitted Surface: Interaction Effect",
    x = "X", y = "Z", fill = "Predicted Y"
```

```
  ) +
  theme_minimal()
```



5.3 Part II: Conditional Associations

5.3.1 The Concept of Conditional Correlation

A **conditional correlation** estimates the association between two variables while "holding constant" (controlling for) a third variable

$$r_{XY|Z} = \frac{\text{Cov}(X - \hat{X}_Z, Y - \hat{Y}_Z)}{\sqrt{\text{Var}(X - \hat{X}_Z) \cdot \text{Var}(Y - \hat{Y}_Z)}}.$$

where $\hat{X}_Z$ and $\hat{Y}_Z$ are predictions from regressions of X and Y on Z. Mathematically, it measures the correlation between the **residuals** of X and Y after removing the linear effects of Z (Pearson 1895; Anderson 1985; Goodall 1991; Baba, Shibata, and Sibuya 2004).

Pair	Correlation
mpg-wt	-0.868
mpg-hp	-0.776
wt-hp	0.659

5.3.2 Partial Correlation

Partial correlation is the most common way to estimate conditional association. It measures the linear association between two variables after regressing out the linear effects of control variables.

```
# Example: relationship between body mass (wt) and fuel economy (mpg),
# controlling for engine power (hp)

mtcars_subset <- mtcars |>
  select(mpg, wt, hp, qsec) |>
  na.omit()

# Bivariate correlations
bivariat_cors <- tibble(
  Pair = c("mpg-wt", "mpg-hp", "wt-hp"),
  Correlation = c(
    cor(mtcars_subset$mpg, mtcars_subset$wt),
    cor(mtcars_subset$mpg, mtcars_subset$hp),
    cor(mtcars_subset$wt, mtcars_subset$hp)
  )
)

bivariat_cors |>
  gt() |>
  fmt_number(columns = Correlation, decimals = 3)
```

```
# Partial correlation: mpg ~ wt | hp
# Method: residuals approach
fit_mpg_hp <- lm(mpg ~ hp, data = mtcars_subset)
fit_wt_hp <- lm(wt ~ hp, data = mtcars_subset)

partial_cor_mpg_wt_given_hp <- cor(
  residuals(fit_mpg_hp),
  residuals(fit_wt_hp)
```

Association	Correlation
mpg-wt (bivariate)	-0.868
mpg-wt (controlling for hp)	-0.751
mpg-hp (bivariate)	-0.776
mpg-hp (controlling for wt)	-0.547

```
)

# Partial correlation: mpg ~ hp | wt
fit_mpg_wt <- lm(mpg ~ wt, data = mtcars_subset)
fit_hp_wt <- lm(hp ~ wt, data = mtcars_subset)

partial_cor_mpg_hp_given_wt <- cor(
  residuals(fit_mpg_wt),
  residuals(fit_hp_wt)
)

partial_summary <- tibble(
  Association = c(
    "mpg-wt (bivariate)",
    "mpg-wt (controlling for hp)",
    "mpg-hp (bivariate)",
    "mpg-hp (controlling for wt)"
  ),
  Correlation = c(
    cor(mtcars_subset$mpg, mtcars_subset$wt),
    partial_cor_mpg_wt_given_hp,
    cor(mtcars_subset$mpg, mtcars_subset$hp),
    partial_cor_mpg_hp_given_wt
  )
)

partial_summary |>
  gt() |>
  fmt_number(columns = Correlation, decimals = 3)
```

Key Insight: Notice how partial correlations can differ markedly from bivariate correlations. The relationship between `wt` and `mpg` remains strong even after controlling for `hp`, suggesting that weight has an independent effect on fuel economy.

Type	mpg_wt
Bivariate	-0.868
Partial (both adjusted)	-0.751
Semi-partial (X adjusted)	-0.474

5.3.3 Semi-Partial (Part) Correlation

Semi-partial correlation is related but slightly different. It measures the association between X and Y after removing the effect of Z from *only one* variable (usually the predictor) (P. Cohen et al. 2014):

$$r_{Y(X.Z)} = \frac{\text{Cov}(X - \hat{X}_Z, Y)}{\sqrt{\text{Var}(X - \hat{X}_Z) \cdot \text{Var}(Y)}}.$$

```
# Semi-partial: association between wt and mpg, removing hp effects from wt only
residuals_wt_hp <- residuals(lm(wt ~ hp, data = mtcars_subset))

semi_partial_mpg_wt_given_hp <- cor(residuals_wt_hp, mtcars_subset$mpg)

comparison_correlations <- tibble(
  Type = c("Bivariate", "Partial (both adjusted)", "Semi-partial (X adjusted)"),
  mpg_wt = c(
    cor(mtcars_subset$mpg, mtcars_subset$wt),
    partial_cor_mpg_wt_given_hp,
    semi_partial_mpg_wt_given_hp
  )
)

comparison_correlations |>
  gt() |>
  fmt_number(columns = mpg_wt, decimals = 3)
```

5.3.4 Stratified Analysis: Examining Associations Within Groups

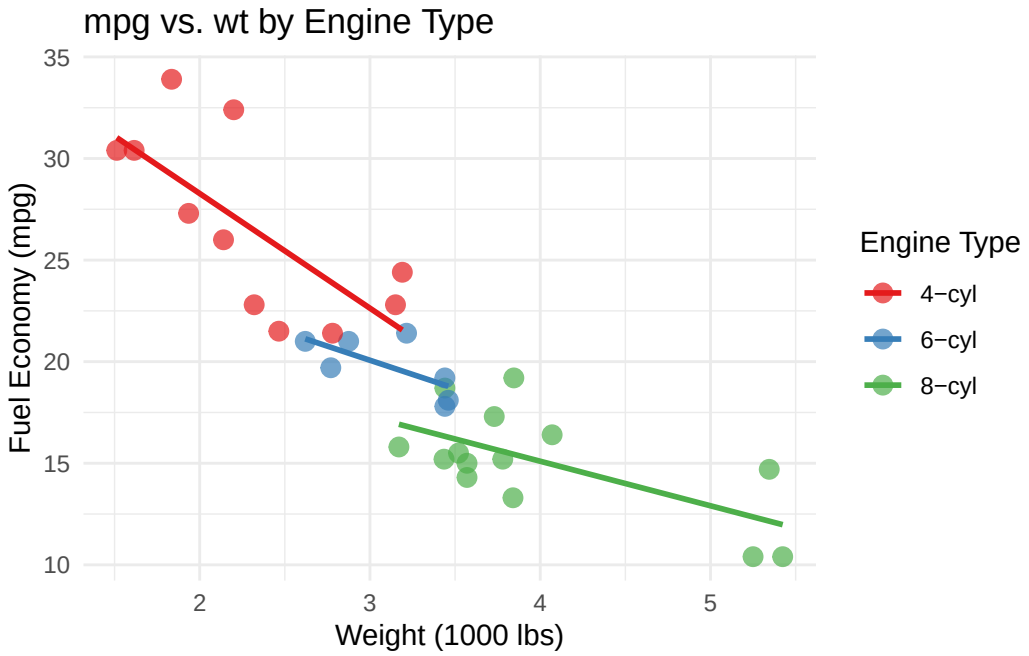
Sometimes the appropriate "control" is not a continuous covariate but a categorical stratification. **Stratified analysis** computes associations separately within each group, revealing how patterns differ.

cyl	N	Correlation
4-cyl	11	-0.713
6-cyl	7	-0.682
8-cyl	14	-0.650

```
# Stratify mtcars by number of cylinders
mtcars_strat <- mtcars |>
  mutate(cyl = factor(cyl, labels = c("4-cyl", "6-cyl", "8-cyl")))

# Correlation of mpg and wt within each stratum
strat_cors <- mtcars_strat |>
  group_by(cyl) |>
  summarise(
    N = n(),
    Correlation = cor(mpg, wt),
    .groups = "drop"
  )

strat_cors |>
  gt() |>
  fmt_number(columns = Correlation, decimals = 3)
```

```
# Visualize stratified associations
mtcars_strat |>
  ggplot(aes(x = wt, y = mpg, color = cyl)) +
  geom_point(size = 3, alpha = 0.7) +
  geom_smooth(method = "lm", se = FALSE, linewidth = 1) +
  scale_color_brewer(palette = "Set1") +
  labs(
    title = "mpg vs. wt by Engine Type",
    x = "Weight (1000 lbs)", y = "Fuel Economy (mpg)",
    color = "Engine Type"
  ) +
  theme_minimal()
```

```
`geom_smooth()` using formula = 'y ~ x'
```



Observation: The relationship between weight and fuel economy is relatively consistent across engine types, though the 4-cylinder group shows slightly higher correlations.

5.3.5 Correlation Networks: Detecting Conditional Independence

With many variables, **graphical models** can identify which associations are direct versus mediated through other variables. The **graphical lasso** (or other sparse methods) estimates a sparse precision matrix (edges that remain after removing the effects of all other variables) (J. Friedman, Hastie, and Tibshirani 2007).

While full graphical model estimation is beyond this descriptive chapter, we can identify **strong conditional associations** by computing partial correlations across all pairs, visualizing only the strongest connections, and comparing to bivariate correlations to identify mediated effects.

```
# Simple conditional independence example using partial correlations
library(corpcor)  # For partial correlations

# Create a subset of continuous variables
mtcars_cont <- mtcars |>
  select(mpg, wt, hp, disp, qsec) |>
  na.omit()

# Compute correlations
df_corr <- cor(mtcars_cont)
```

Pair	Bivariate	Partial
mpg-wt	-0.868	-0.574
mpg-hp	-0.776	-0.224
wt-hp	0.659	0.096

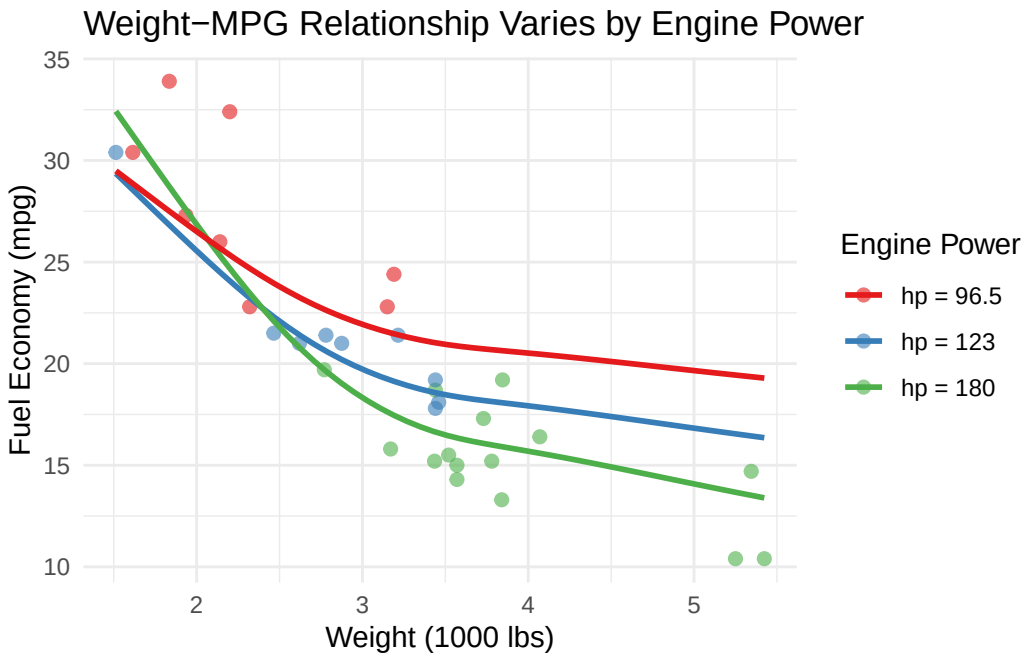
```
# Compute partial correlations (adjusting for all other variables)
# Approximate using precision matrix inverse
df_precision <- solve(df_corr)
df_partial <- -cov2cor(df_precision)
diag(df_partial) <- 1

# Compare formats
comparison_corr_types <- tibble(
  Pair = c("mpg-wt", "mpg-hp", "wt-hp"),
  Bivariate = c(df_corr["mpg", "wt"], df_corr["mpg", "hp"], df_corr["wt", "hp"]),
  Partial = c(df_partial["mpg", "wt"], df_partial["mpg", "hp"], df_partial["wt", "hp"])
)

comparison_corr_types |>
  gt() |>
  fmt_number(columns = c(Bivariate, Partial), decimals = 3)
```

5.4 Part III: Combining Nonlinear and Conditional Analysis

Real-world analysis often requires both nonlinear and conditional thinking. We can combine these ideas in a **semiparametric model** that allows smooth associations while controlling for other variables.

```
# Example: how does the effect of weight on mpg vary by engine size?
# Allow smooth mpg ~ wt relationship, varying smoothly with hp

gam_semi <- gam(mpg ~ s(wt, by = hp) + s(hp),
                data = mtcars)

# Visualize the varying slope effect
hp_values <- quantile(mtcars$hp, probs = c(0.25, 0.5, 0.75))

hp_levels <- paste0("hp = ", hp_values)
```

```r
mtcars_col <- mtcars |>
  mutate(
    hp_label = case_when(
      hp <= hp_values[1] ~ hp_levels[1],
      hp <= hp_values[2] ~ hp_levels[2],
      TRUE               ~ hp_levels[3]
    ),
    hp_label = factor(hp_label, levels = hp_levels)
  )

pred_smooth <- map_df(hp_values, function(hp_val) {
  pred_grid <- tibble(
    wt = seq(min(mtcars$wt), max(mtcars$wt), length.out = 50),
    hp = hp_val
  )
  pred_grid$mpg_pred <- predict(gam_semi, newdata = pred_grid)
  pred_grid$hp_label <- paste0("hp = ", hp_val)
  pred_grid
})

  ggplot(mtcars_col, aes(x = wt, y = mpg, color = hp_label)) +
  geom_point(alpha = 0.6, size = 2) +
  geom_line(
    data = pred_smooth,
    aes(y = mpg_pred),
    linewidth = 1
  ) +
  scale_color_brewer(palette = "Set1") +
  labs(
    title = "Weight-MPG Relationship Varies by Engine Power",
    x = "Weight (1000 lbs)",
    y = "Fuel Economy (mpg)",
    color = "Engine Power"
  ) +
  theme_minimal()
```



Interpretation: The smooth trends indicate a consistently negative relationship between vehicle weight and fuel economy across all engine power levels. However, this relationship becomes steeper as engine power increases: for a given increase in weight, high-horsepower vehicles experience a larger reduction in mpg than low-horsepower vehicles. This suggests that heavier, more powerful cars are disproportionately penalized in terms of fuel efficiency.

5.5 Summary: When to Use Each Method

Situation	Recommended Method
Exploring whether association is linear	Nonlinearity index, visual inspection with LOESS
Describing nonlinear relationship shape	GAM smooth terms, scatterplot with lowess
Testing for independence (any form)	Distance correlation, mutual information
Adjusting for one continuous confounder	Partial correlation, residual method
Examining association in subgroups	Stratified analysis, conditional plots
Multiple confounders simultaneously	GAM with adjustment terms, graphical models

Situation	Recommended Method
Context-dependent nonlinear effects	Semiparametric models (GAM with interactions)

5.6 Summary and Key Takeaways

- **Linear correlations can miss real patterns**: It helps to visualize associations with smooth trend lines to detect nonlinearity
- **Nonlinearity indices** compare simple vs. complex model fits to quantify how much structure is missed by linearity
- **Distance correlation** and **GAMs** are flexible tools that capture diverse functional forms
- **Partial correlations** separate direct from mediated effects, revealing which associations are conditionally independent
- **Stratified analysis** reveals how associations vary across subpopulations
- **Combining approaches** (nonlinear + conditional) yields richer understanding than either alone
- **Visualization precedes quantification**: Scatterplots with smooth fits can usefully accompany numeric results

5.7 Looking Ahead

With nonlinear and conditional association methods now in hand, the next chapter moves to **network representations** of association patterns. Networks elegantly visualize how multiple associations connect and interact, transforming correlation matrices into interpretable visual structures. This bridges descriptive statistics and exploratory analytics, setting the stage for interactive visualization and machine learning approaches in later chapters.

6 Network Representations of Multivariate Associations

6.1 Introduction: From Matrices to Networks

In previous chapters, we computed correlation matrices and visualized associations as rectangular arrays or heatmaps. Yet when many variables are involved, these representations can become difficult to parse. A correlation matrix with 50 variables contains 1,225 pairwise associations, which is more than most of us can comfortably scan at once.

Network representations offer an alternative visual and analytical framework. Instead of viewing all associations at once, networks can highlight meaningful connections while suppressing noise, allowing us to:

1. **Identify clusters** of strongly related variables
2. **Detect hub variables** that associate with many others
3. **Reveal hierarchical structures** in complex data
4. **Quantify network properties** that describe overall structure

A network (or *graph*) consists of:

- **Nodes**: Representing entities (in our case, variables)
- **Edges**: Representing relationships (correlations, associations, dependencies)
- **Weights** (optional): Representing edge strength

This chapter walks through how to transform association matrices into networks, visualize them effectively, and compute network metrics to understand multivariate structure.

6.2 Part I: From Associations to Networks

6.2.1 Thresholding: Creating Sparsity

A correlation matrix for p variables is typically *dense*, as nearly every cell contains a value. When we plot this as a network, it creates a hairball of connections, revealing little structure.

Variable	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Sepal.Length	1.000	-0.118	0.872	0.818
Sepal.Width	-0.118	1.000	-0.428	-0.366
Petal.Length	0.872	-0.428	1.000	0.963
Petal.Width	0.818	-0.366	0.963	1.000

One common solution is **thresholding**: we retain only associations that exceed some cutoff in absolute value, discarding weaker associations to create sparse networks.

We begin with `iris`, a classic built-in R dataset with 150 flower observations from three species (`setosa`, `versicolor`, `virginica`) and four numeric measurements (sepal length/width, petal length/width), which makes it ideal for compact association networks.

```
# Load example data: Iris dataset
data(iris)

# Select numeric variables
iris_numeric <- iris |>
  select(-Species) |>
  as.data.frame()

# Compute correlation matrix
cor_matrix <- cor(iris_numeric)

# Display raw correlations
cor_matrix |>
  as.data.frame() |>
  tibble::rownames_to_column(var = "Variable") |>
  gt() |>
  fmt_number(columns = -Variable, decimals = 3)
```

6.2.2 Creating a Network from Correlation Thresholds

```
# Step 1: Define threshold
threshold <- 0.5

# Step 2: Create adjacency matrix (1 if |correlation| > threshold, 0 otherwise)
adjacency_matrix <- (abs(cor_matrix) > threshold) * 1
```

```r
# Remove diagonal (self-loops)
diag(adjacency_matrix) <- 0

# Step 3: Convert to igraph object with weighted values
iris_network <- graph_from_adjacency_matrix(
  adjacency_matrix,
  mode = "undirected",
  weighted = cor_matrix,
  diag = FALSE
)

# Display basic network properties
cat("Network Summary:\n")
```

```
Network Summary:
```

```r
cat("Number of nodes:", vcount(iris_network), "\n")
```

```
Number of nodes: 4
```

```r
cat("Number of edges:", ecount(iris_network), "\n")
```

```
Number of edges: 3
```

```r
cat("Network density:", edge_density(iris_network), "\n")
```

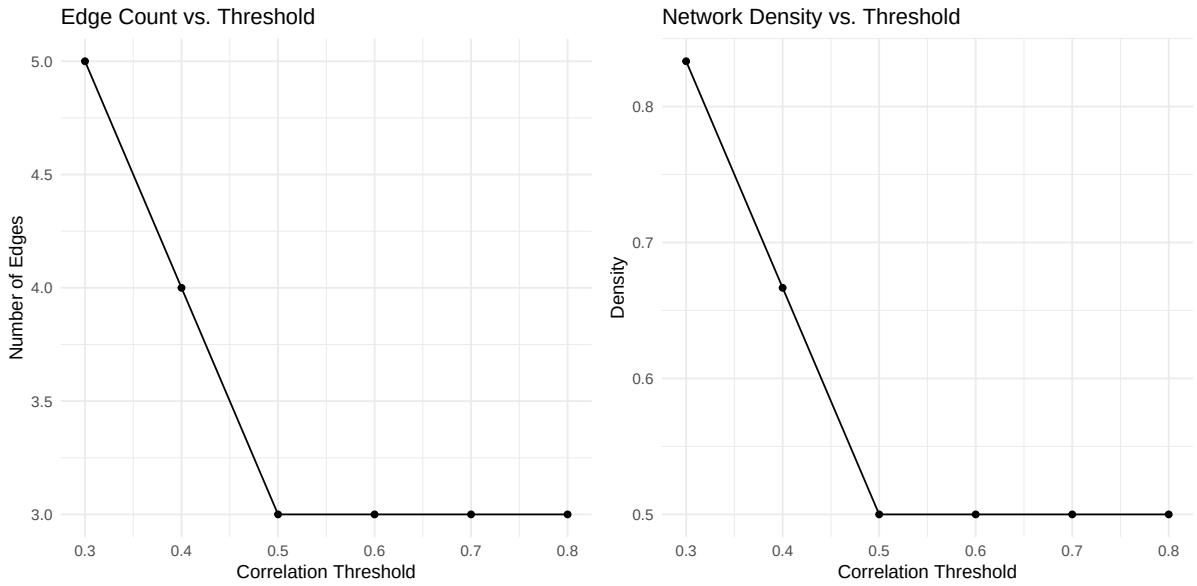
```
Network density: 0.5
```

6.2.3 Choosing the Right Threshold

The threshold is important: too high and we lose structure; too low and we recreate the hairball.

```r
# Test multiple thresholds
thresholds <- seq(0.3, 0.8, by = 0.1)

threshold_analysis <- tibble(
  Threshold = thresholds,
```

```
  Edges = map_dbl(thresholds, ~{
    adj <- (abs(cor_matrix) > .x) * 1
    diag(adj) <- 0
    sum(adj) / 2  # Divide by 2 for undirected graph
  }),
  Density = map_dbl(thresholds, ~{
    adj <- (abs(cor_matrix) > .x) * 1
    diag(adj) <- 0
    n <- nrow(adj)
    sum(adj) / (n * (n - 1))
  })
)

# Visualize threshold effects
p1 <- ggplot(threshold_analysis, aes(x = Threshold, y = Edges)) +
  geom_line() +
  geom_point() +
  labs(title = "Edge Count vs. Threshold",
       x = "Correlation Threshold",
       y = "Number of Edges") +
  theme_minimal()

p2 <- ggplot(threshold_analysis, aes(x = Threshold, y = Density)) +
  geom_line() +
  geom_point() +
  labs(title = "Network Density vs. Threshold",
       x = "Correlation Threshold",
       y = "Density") +
  theme_minimal()

p1 + p2
```



6.3 Part II: Network Visualization

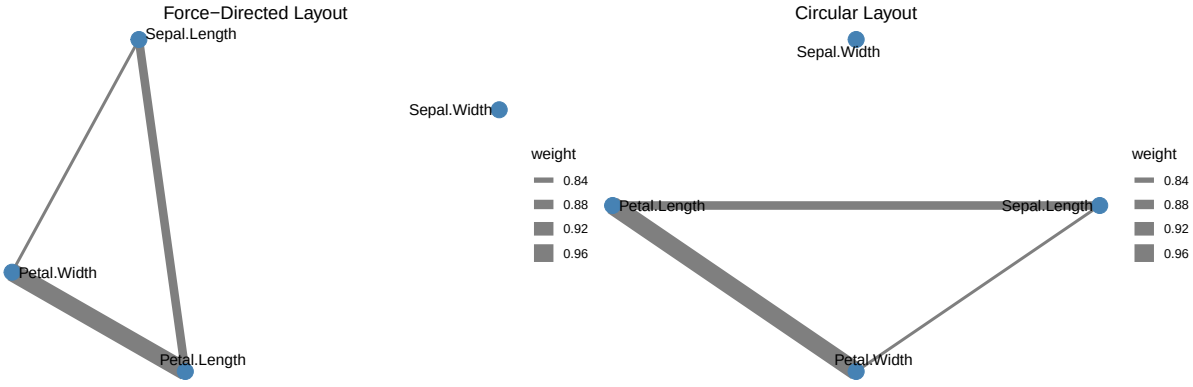
6.3.1 Graph Layouts

How nodes are positioned crucially affects interpretability. Different layout algorithms emphasize different structural properties:

- **Force-directed** (Fruchterman-Reingold): Nodes repel each other, and edges act as springs. Reveals clusters.
- **Circular**: Arranges nodes in a circle. Useful for showing all labels clearly.
- **Hierarchical**: Arranges nodes by levels. Useful for tree-like structures.

```
# Create networks with different thresholds for visualization
threshold_tight <- 0.6
adj_tight <- (abs(cor_matrix) > threshold_tight) * 1
diag(adj_tight) <- 0
net_tight <- graph_from_adjacency_matrix(adj_tight, mode = "undirected", diag = FALSE)

# Assign correlation values as edge attributes
edge_list_tight <- as_edgelist(net_tight)
edge_weights_tight <- numeric(nrow(edge_list_tight))
for (i in 1:nrow(edge_list_tight)) {
  edge_weights_tight[i] <- cor_matrix[edge_list_tight[i, 1], edge_list_tight[i, 2]]
}
```

```r
E(net_tight)$weight <- abs(edge_weights_tight)

# Convert to tidygraph for ggraph
net_tidygraph <- as_tbl_graph(net_tight)

# Visualize with different layouts
p1 <- ggraph(net_tidygraph, layout = 'fr') +
  geom_edge_link(aes(width = weight), edge_colour = "gray50") +
  geom_node_point(size = 5, colour = "steelblue") +
  geom_node_text(aes(label = name), repel = TRUE) +
  labs(title = "Force-Directed Layout") +
  theme_void() +
  theme(plot.title = element_text(hjust = 0.5))

p2 <- ggraph(net_tidygraph, layout = 'circle') +
  geom_edge_link(aes(width = weight), edge_colour = "gray50") +
  geom_node_point(size = 5, colour = "steelblue") +
  geom_node_text(aes(label = name), repel = TRUE) +
  labs(title = "Circular Layout") +
  theme_void() +
  theme(plot.title = element_text(hjust = 0.5))

  p1 + p2
```



6.3.2 Enhancing Network Visualizations

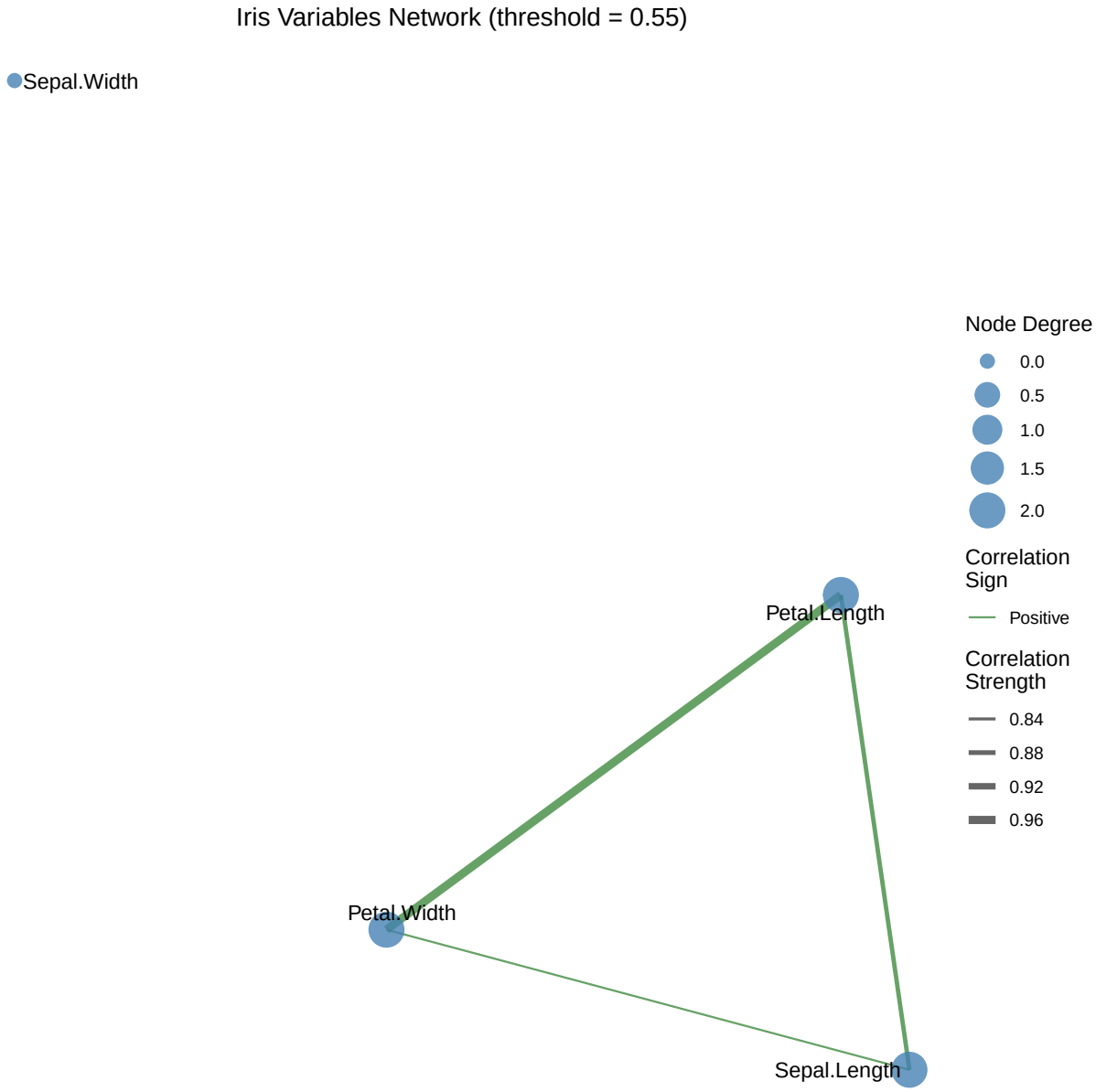
We can encode additional information through visual channels:

- **Node color**: Variable type, importance, or cluster membership
- **Node size**: Degree (number of connections) or other centrality measure

- **Edge width**: Correlation strength
- **Edge color**: Correlation sign (positive vs. negative)

```
# Compute node degrees and color by degree
threshold_vis <- 0.55
adj_vis <- (abs(cor_matrix) > threshold_vis) * 1
diag(adj_vis) <- 0

# Create network from adjacency matrix
net_vis <- graph_from_adjacency_matrix(
  adj_vis,
  mode = "undirected",
  diag = FALSE
)

# Extract and assign correlation values as edge weights
edge_list <- as_edgelist(net_vis)
edge_weights <- numeric(nrow(edge_list))
for (i in 1:nrow(edge_list)) {
  edge_weights[i] <- cor_matrix[edge_list[i, 1], edge_list[i, 2]]
}
E(net_vis)$weight <- edge_weights
E(net_vis)$edge_sign <- ifelse(edge_weights > 0, "Positive", "Negative")

# Create a more sophisticated visualization
net_vis_tidy <- as_tbl_graph(net_vis) |>
  mutate(
    degree = centrality_degree(),
    betweenness = centrality_betweenness()
  )

ggraph(net_vis_tidy, layout = 'fr') +
  geom_edge_link(
    aes(
      width = abs(weight),
      colour = edge_sign
    ),
    alpha = 0.6
  ) +
  geom_node_point(aes(size = degree), colour = "steelblue", alpha = 0.8) +
  geom_node_text(aes(label = name), repel = TRUE, size = 4) +
  scale_edge_width(range = c(0.5, 2)) +
  scale_edge_colour_manual(values = c("Positive" = "darkgreen", "Negative" = "darkred")) +
```

```
  scale_size_continuous(range = c(3, 8)) +
  labs(
    title = "Iris Variables Network (threshold = 0.55)",
    edge_width = "Correlation\nStrength",
    edge_colour = "Correlation\nSign",
    size = "Node Degree"
  ) +
  theme_void() +
  theme(legend.position = "right", plot.title = element_text(hjust = 0.5))
```



6.4 Part III: Network Metrics

Network analysis provides quantitative measures of structure. These help us describe and compare networks objectively.

Variable	Degree	Betweenness	Closeness	Eigenvector
Sepal.Length	2.000	0.000	0.592	0.946
Petal.Length	2.000	0.000	0.545	1.000
Petal.Width	2.000	0.000	0.562	0.981
Sepal.Width	0.000	0.000	NaN	0.000

6.4.1 Centrality Measures

Centrality measures identify "important" nodes (those with prominent roles in the network).

```
# Calculate centrality measures
centrality_measures <- tibble(
  Variable = names(iris_numeric),
  Degree = degree(net_vis),
  Betweenness = betweenness(net_vis),
  Closeness = closeness(net_vis),
  Eigenvector = eigen_centrality(net_vis)$vector
) |>
  arrange(desc(Degree))

gt(centrality_measures) |>
  fmt_number(columns = -Variable, decimals = 3)
```

6.4.2 Clustering and Community Detection

Community detection algorithms partition nodes into groups that are more densely connected within than between groups.

```
# Detect communities using Louvain algorithm
communities <- cluster_louvain(net_vis)

# Add community membership to network
V(net_vis)$community <- membership(communities)

cat("Number of communities detected:", length(communities), "\n")
```

```
Number of communities detected: 2
```

```r
cat("Community sizes:\n")
```

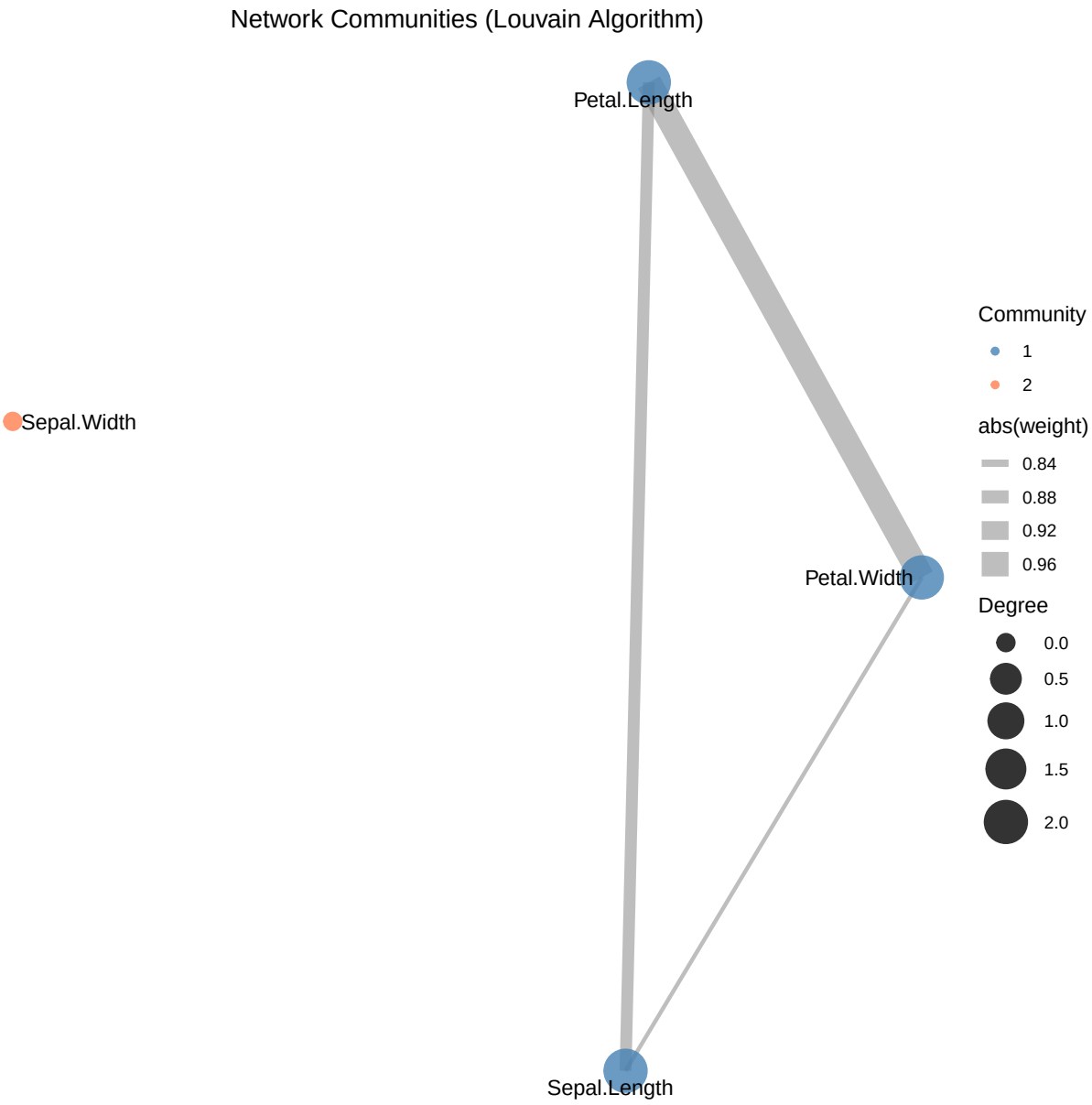
```
Community sizes:
```

```r
print(sizes(communities))
```

```
Community sizes
1 2
3 1
```

Visualizing communities:

```r
# Create a color palette for communities
comm_colors <- c("steelblue", "coral", "lightgreen", "gold")

net_vis_comm <- as_tbl_graph(net_vis) |>
  mutate(community = V(net_vis)$community)

ggraph(net_vis_comm, layout = 'fr') +
  geom_edge_link(aes(width = abs(weight)), edge_colour = "gray50", alpha = 0.5) +
  geom_node_point(aes(colour = as.factor(community), size = degree(net_vis)), alpha = 0.8) +
  geom_node_text(aes(label = name), repel = TRUE, size = 4) +
  scale_colour_manual(values = comm_colors) +
  scale_size_continuous(range = c(4, 10)) +
  labs(
    title = "Network Communities (Louvain Algorithm)",
    colour = "Community",
    size = "Degree"
  ) +
  theme_void() +
  theme(legend.position = "right", plot.title = element_text(hjust = 0.5))
```



6.4.3 Global Network Properties

```
# Compute global network properties
global_properties <- tibble(
  Property = c(
    "Number of Nodes",
    "Number of Edges",
```

Property	Value
Number of Nodes	4.000
Number of Edges	3.000
Network Density	0.500
Average Degree	1.500
Average Path Length	0.884
Diameter	0.963
Transitivity (Clustering Coefficient)	1.000
Number of Communities	2.000

```
    "Network Density",
    "Average Degree",
    "Average Path Length",
    "Diameter",
    "Transitivity (Clustering Coefficient)",
    "Number of Communities"
  ),
  Value = c(
    vcount(net_vis),
    ecount(net_vis),
    edge_density(net_vis),
    mean(degree(net_vis)),
    mean_distance(net_vis),
    diameter(net_vis),
    transitivity(net_vis),
    length(communities)
  )
)

gt(global_properties) |>
  fmt_number(columns = Value, decimals = 3)
```

6.5 Part IV: Practical Applications

6.5.1 Application 1: Multivariate Data Exploration

Networks help us understand which variables tend to move together, revealing natural groupings:

```r
# Load a dataset with more variables
# Using mtcars for richer examples
data(mtcars)

# Compute correlation matrix
mtcars_cor <- cor(mtcars)

# Create network with moderate threshold
threshold_mtcars <- 0.5
adj_mtcars <- (abs(mtcars_cor) > threshold_mtcars) * 1
diag(adj_mtcars) <- 0

net_mtcars <- graph_from_adjacency_matrix(
  adj_mtcars,
  mode = "undirected",
  weighted = TRUE
)

# Detect communities
comm_mtcars <- cluster_louvain(net_mtcars)

cat("mtcars Variable Groups:\n")
```

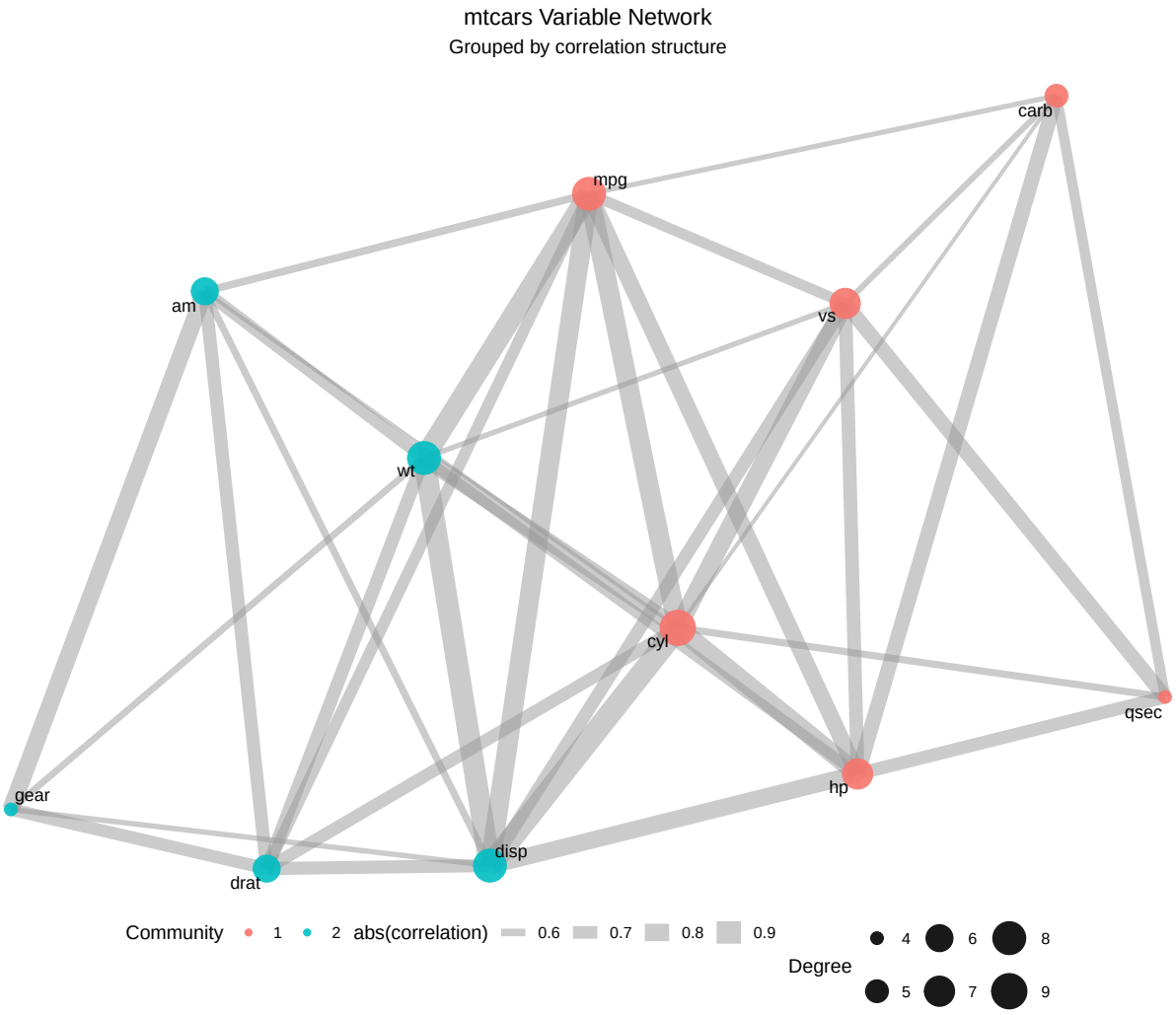
```
mtcars Variable Groups:
```

```r
for (i in seq_along(comm_mtcars)) {
  cat("Group", i, ":", paste(names(mtcars)[membership(comm_mtcars) == i], collapse = ", "), "
}
```

```
Group 1 : mpg, cyl, hp, qsec, vs, carb
Group 2 : disp, drat, wt, am, gear
```

```r
# Visualize mtcars network
V(net_mtcars)$community <- membership(comm_mtcars)

# Extract and assign edge weights from correlation matrix
edge_list_mtcars <- as_edgelist(net_mtcars)
edge_weights_mtcars <- numeric(nrow(edge_list_mtcars))
for (i in 1:nrow(edge_list_mtcars)) {
  edge_weights_mtcars[i] <- mtcars_cor[edge_list_mtcars[i, 1], edge_list_mtcars[i, 2]]
}
```

```r
# Use absolute values for layout algorithm (Fruchterman-Reingold requires positive weights)
E(net_mtcars)$weight <- abs(edge_weights_mtcars)
E(net_mtcars)$correlation <- edge_weights_mtcars

net_mtcars_tidy <- as_tbl_graph(net_mtcars)

ggraph(net_mtcars_tidy, layout = 'fr') +
  geom_edge_link(
    aes(width = abs(correlation)),
    edge_colour = "gray60",
    alpha = 0.5
  ) +
  geom_node_point(
    aes(
      colour = as.factor(V(net_mtcars)$community),
      size = degree(net_mtcars)
    ),
    alpha = 0.9
  ) +
  geom_node_text(aes(label = name), repel = TRUE, size = 3.5) +
  scale_size_continuous(range = c(3, 9)) +
  labs(
    title = "mtcars Variable Network",
    subtitle = "Grouped by correlation structure",
    colour = "Community",
    size = "Degree"
  ) +
  theme_void() +
  theme(legend.position = "bottom", plot.title = element_text(hjust = 0.5), plot.subtitle = e
```



6.5.2 Application 2: Comparison of Networks

Networks can be compared across subgroups or time periods to identify changing associations:

```
# Split iris by species and compute networks
iris_setosa <- iris |> filter(Species == "setosa") |> select(-Species)
iris_versicolor <- iris |> filter(Species == "versicolor") |> select(-Species)
iris_virginica <- iris |> filter(Species == "virginica") |> select(-Species)

cor_setosa <- cor(iris_setosa)
cor_versicolor <- cor(iris_versicolor)
cor_virginica <- cor(iris_virginica)
```

```r
# Create networks for each
threshold_species <- 0.5

create_species_network <- function(cor_mat) {
  adj <- (abs(cor_mat) > threshold_species) * 1
  diag(adj) <- 0
  graph_from_adjacency_matrix(adj, mode = "undirected", weighted = TRUE)
}

net_setosa <- create_species_network(cor_setosa)
net_versicolor <- create_species_network(cor_versicolor)
net_virginica <- create_species_network(cor_virginica)

# Compare network properties
network_comparison <- tibble(
  Species = c("Setosa", "Versicolor", "Virginica"),
  Edges = c(ecount(net_setosa), ecount(net_versicolor), ecount(net_virginica)),
  Density = c(
    edge_density(net_setosa),
    edge_density(net_versicolor),
    edge_density(net_virginica)
  ),
  Avg_Degree = c(
    mean(degree(net_setosa)),
    mean(degree(net_versicolor)),
    mean(degree(net_virginica))
  ),
  Clustering_Coeff = c(
    transitivity(net_setosa),
    transitivity(net_versicolor),
    transitivity(net_virginica)
  )
)

gt(network_comparison) |>
  fmt_number(columns = -Species, decimals = 3)
```

Species	Edges	Density	Avg_Degree	Clustering_Coeff
Setosa	1.000	0.167	0.500	NaN
Versicolor	6.000	1.000	3.000	1.000
Virginica	2.000	0.333	1.000	NaN

6.5.3 Application 3: Partial Correlation Networks

With many variables, simple correlations may be misleading due to confounding. **Partial correlations** (associations after removing effects of other variables) often reveal cleaner networks.

```
# Note: Partial correlation computation often uses inverse covariance
# For this example, we'll demonstrate the concept with a simplified approach
# using residuals from regression

compute_partial_corr_approx <- function(data) {
  n_vars <- ncol(data)
  partial_corr <- matrix(1, n_vars, n_vars)
  colnames(partial_corr) <- colnames(data)
  rownames(partial_corr) <- colnames(data)

  if (n_vars < 2) {
    return(partial_corr)
  }

  for (i in 1:(n_vars - 1)) {
    for (j in (i + 1):n_vars) {
      # Residuals of X_i after removing effects of other variables
      x_other <- data[, -c(i, j), drop = FALSE]

      # Create a data frame with response and predictors for regression
      df_i <- cbind(data[, i, drop = FALSE], x_other)
      colnames(df_i)[1] <- "y"
      fit_i <- lm(y ~ ., data = df_i)
      resid_i <- residuals(fit_i)

      # Residuals of X_j after removing effects of other variables
      df_j <- cbind(data[, j, drop = FALSE], x_other)
      colnames(df_j)[1] <- "y"
      fit_j <- lm(y ~ ., data = df_j)
      resid_j <- residuals(fit_j)
```

```
      # Correlation of residuals
      partial_corr[i, j] <- cor(resid_i, resid_j)
      partial_corr[j, i] <- cor(resid_i, resid_j)
    }
  }

  partial_corr
}

# Compute partial correlations
partial_cor_iris <- compute_partial_corr_approx(iris_numeric)

# Create network from partial correlations
threshold_partial <- 0.3
adj_partial <- (abs(partial_cor_iris) > threshold_partial) * 1
diag(adj_partial) <- 0

net_partial <- graph_from_adjacency_matrix(
  adj_partial,
  mode = "undirected",
  diag = FALSE
)

# Extract and assign edge weights
edge_list_partial <- as_edgelist(net_partial)
edge_weights_partial <- numeric(nrow(edge_list_partial))
for (i in 1:nrow(edge_list_partial)) {
  edge_weights_partial[i] <- partial_cor_iris[edge_list_partial[i, 1], edge_list_partial[i, 2
}
E(net_partial)$weight <- abs(edge_weights_partial)

cat("\nPartial Correlation Network Properties:\n")
```

```
Partial Correlation Network Properties:
```

```
cat("Edges:", ecount(net_partial), "\n")
```

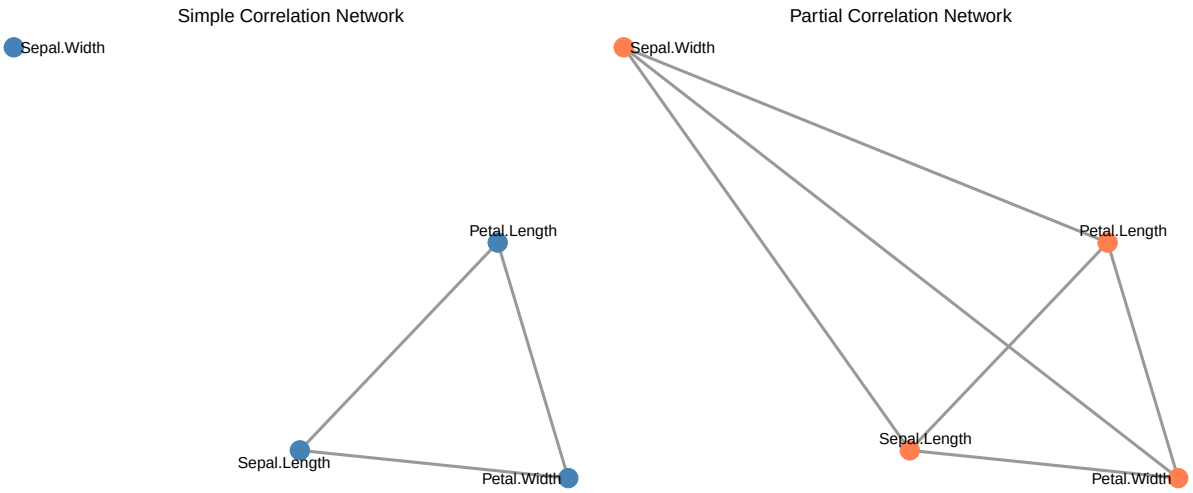
```
Edges: 6
```

```r
cat("Density:", edge_density(net_partial), "\n")
```

```
Density: 1
```

Comparing simple vs. partial correlation networks:

```r
# Simple correlation network
threshold_simple <- 0.55
adj_simple <- (abs(cor_matrix) > threshold_simple) * 1
diag(adj_simple) <- 0
net_simple <- graph_from_adjacency_matrix(adj_simple, mode = "undirected", weighted = TRUE)

# Prepare both networks for plotting
layout_coords <- layout_with_fr(net_simple)

p1 <- ggraph(as_tbl_graph(net_simple), layout = layout_coords) +
  geom_edge_link(edge_colour = "gray60", width = 1) +
  geom_node_point(size = 6, colour = "steelblue") +
  geom_node_text(aes(label = name), repel = TRUE) +
  labs(title = "Simple Correlation Network") +
  theme_void() +
  theme(plot.title = element_text(hjust = 0.5))

p2 <- ggraph(as_tbl_graph(net_partial), layout = layout_coords) +
  geom_edge_link(edge_colour = "gray60", width = 1) +
  geom_node_point(size = 6, colour = "coral") +
  geom_node_text(aes(label = name), repel = TRUE) +
  labs(title = "Partial Correlation Network") +
  theme_void() +
  theme(plot.title = element_text(hjust = 0.5))

p1 + p2
```

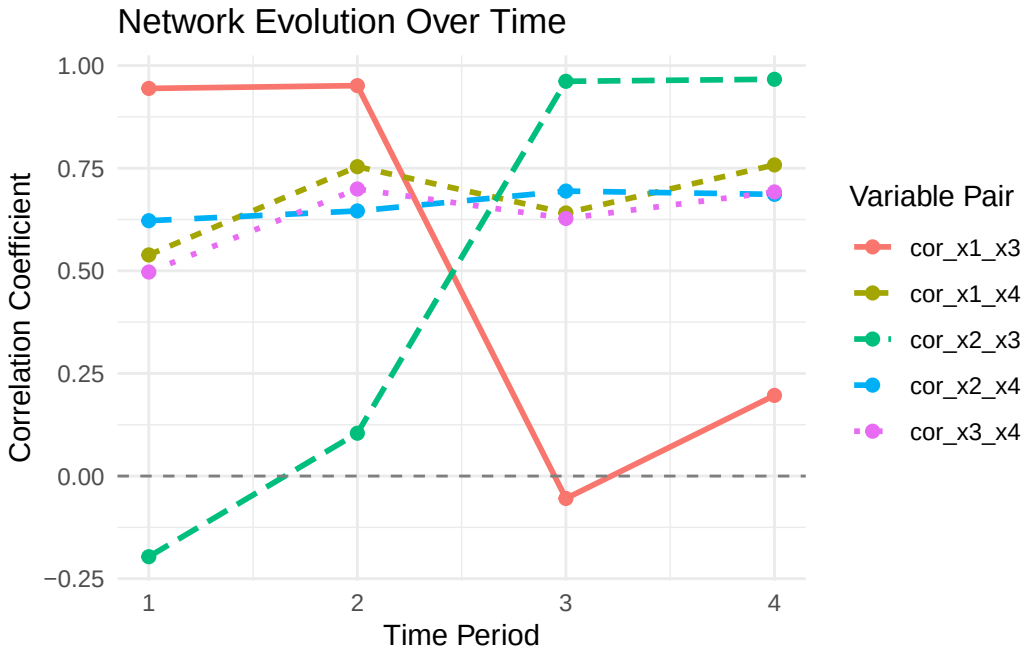


6.6 Part V: Dynamic and Temporal Networks

Networks also reveal patterns across time. When observations represent time periods, we can track how associations evolve.

```
# Simulate time series data: changing correlations
set.seed(123)
n_obs <- 100
n_periods <- 4

# Create data where correlations change over time
ts_data <- expand_grid(
  period = 1:n_periods,
  obs = 1:n_obs
) |>
  mutate(
    # Variables with period-dependent relationships
    x1 = rnorm(n()),
    x2 = rnorm(n()),
    x3 = ifelse(period <= 2, x1 + rnorm(n(), 0, 0.3), x2 + rnorm(n(), 0, 0.3)),
    x4 = x1 + x2 + rnorm(n(), 0, 0.5)
  )

# Compute correlations per period
cors_by_period <- ts_data |>
  group_by(period) |>
  summarise(
```

```
    cor_x1_x3 = cor(x1, x3),
    cor_x1_x4 = cor(x1, x4),
    cor_x2_x3 = cor(x2, x3),
    cor_x2_x4 = cor(x2, x4),
    cor_x3_x4 = cor(x3, x4),
    .groups = "drop"
  )

# Visualize changing correlations
cors_by_period |>
  pivot_longer(cols = -period, names_to = "Edge", values_to = "Correlation") |>
  ggplot(aes(x = period, y = Correlation, colour = Edge, linetype = Edge)) +
  geom_line(linewidth = 1) +
  geom_point(size = 2) +
  geom_hline(yintercept = 0, linetype = "dashed", colour = "gray50") +
  scale_x_continuous(breaks = 1:n_periods) +
  labs(
    title = "Network Evolution Over Time",
    x = "Time Period",
    y = "Correlation Coefficient",
    colour = "Variable Pair",
    linetype = "Variable Pair"
  ) +
  theme_minimal() +
  theme(legend.position = "right")
```



6.7 Summary and Key Takeaways

- **Networks transform complexity**: Large correlation matrices become interpretable visualizations and metrics.
- **Thresholding is essential**: Choose thresholds that balance information retention with interpretability.
- **Layout matters**: Different graph layouts reveal different structural properties. Force-directed layouts typically highlight clustering.
- **Centrality measures identify hubs**: Variables with high degree or betweenness centrality are important connectors.
- **Communities reveal natural groupings**: Automated algorithms partition variables into clusters with strong internal associations.
- **Partial correlations remove confounding**: Networks based on partial correlations often reveal cleaner, more interpretable structure.
- **Networks change over time**: Tracking network evolution reveals how associations shift across contexts or periods.

6.8 Looking Ahead

With network representations now providing a topological view of associations, we shift from static visualization to **interactive exploration**. The next chapter introduces Shiny applications

that enable dynamic network exploration: hovering over nodes to highlight connections, filtering by community membership, adjusting thresholds in real-time, and comparing networks across subgroups without recomputing layouts. This interactivity transforms networks from publication-ready static images into powerful exploratory tools for discovering patterns that static visualizations cannot convey. Interactive tools amplify our ability to interrogate high-dimensional data structures and communicate findings effectively to diverse audiences.

Part III

Interactive Visual Analytics

7 Interactive Exploration with Shiny

7.1 Introduction: The Limitations of Static Visualization

In previous chapters, we computed association matrices, created correlation heatmaps, and drew network graphs. These static visualizations are publication-ready and reproducible, but they can be inherently limited: each figure represents one choice of parameters, one threshold, one layout, one subset of data.

Consider a researcher working with network data:

- A network visualization is created with a threshold of 0.5. But what happens at 0.6? Or 0.4?
- One layout algorithm reveals a community structure. Does the Louvain algorithm find the same communities as Leiden?
- A variable appears central in the full dataset. But how important is it in a particular subgroup?

Static analysis answers one question at a time. Each new question typically requires code modification, recomputation, and a new figure.

Interactive analysis can empower exploration. Users can adjust parameters, filter data, and compare perspectives without waiting for recomputation, enabling discovery of patterns that static workflows often miss.

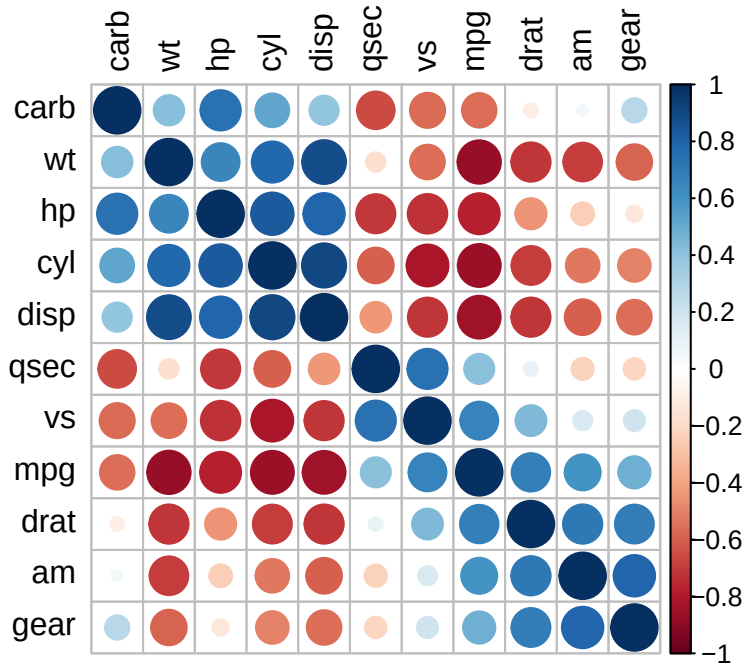
This chapter introduces **Shiny**, an R web framework that can transform exploratory analysis from a batch workflow into an interactive conversation with data. Rather than asking "what does this plot show?", interactive tools let practitioners ask "what if I change this parameter?" or "how does this look in this subgroup?" and receive near-instant visual feedback.

7.2 Part I: Principles of Interactive Data Exploration

7.2.1 The Problem with Single Summaries

A single figure is frozen in time and parameter space. Consider a correlation matrix visualization:

```r
# Static correlation heatmap - one view only
data(mtcars)
cor_mtcars <- cor(mtcars)

# Choose one color scheme, one ordering, one threshold
corrplot(cor_mtcars, method = "circle", order = "hclust",
         tl.col = "black")
```



This figure effectively communicates the correlation structure, but:

1. **No threshold filtering**: The plot shows all correlations, including weak ones
2. **No subset comparison**: You cannot quickly examine correlations for sedans vs. sports cars
3. **No parameter exploration**: What if the clustering order were different?
4. **No interaction**: Hovering over cells, highlighting related variables, or zooming is impossible

Practitioners often resort to creating multiple figures to show different perspectives, one for each threshold, subgroup, or parameter choice. This multiplies the number of static outputs and makes **comparison harder**, not easier.

7.2.2 Benefits of Interactivity

Interactive tools address these limitations by enabling:

1. **Real-time parameter adjustment**: Change a threshold and watch the network update instantly
2. **Dynamic subsetting**: Filter data by group or criteria without restarting the analysis
3. **Hover interactions**: Explore details on demand (e.g., exact correlation values, variable names)
4. **Cross-linked views**: Highlight a node in a network and see its correlations in a table
5. **No computation overhead**: After the initial render, interactions are nearly instantaneous

These capabilities can transform exploration from passive viewing to active interrogation.

7.2.3 When Interactive Tools Matter Most

Interactive exploration is most valuable when:

- **Parameter sensitivity is high**: Small changes in thresholds or algorithms yield very different results
- **Subgroup heterogeneity exists**: Patterns differ across demographic or categorical groups
- **Stakeholders are non-technical**: Interactive tools make findings accessible without requiring coding
- **The data are high-dimensional**: Static visualizations of 50+ variables become unreadable; interactivity allows focus on relevant subsets
- **Iterative discovery is the goal**: Rather than hypothesis testing, you are searching for patterns

Conversely, we still need static visualizations for **publication** and **presentation**, where reproducibility and editorial control matter more than exploration.

7.3 Part II: Introduction to Shiny

7.3.1 Shiny Basics: UI and Server

Shiny separates an application into two components:

1. **User Interface (UI)**: Layout, controls (sliders, dropdowns, buttons), and output placeholders
2. **Server logic**: Reactive computations that respond to user inputs

A minimal Shiny app has this structure:

```
library(shiny)

# Define UI
ui <- fluidPage(
  titlePanel("My First Shiny App"),
  sidebarLayout(
    sidebarPanel(
      sliderInput("threshold", "Correlation Threshold:",
                  min = 0, max = 1, value = 0.5, step = 0.05)
    ),
    mainPanel(
      plotOutput("correlation_plot")
    )
  )
)

# Define server logic
server <- function(input, output) {
  output$correlation_plot <- renderPlot({
    # This code runs whenever input$threshold changes
    cor_mtcars <- cor(mtcars)
    # Create network based on current threshold
    adj <- (abs(cor_mtcars) > input$threshold) * 1
    diag(adj) <- 0
    # Create visualization...
  })
}

# Run the app
shinyApp(ui, server)
```

The key concept is that when a user adjusts the slider, the value in `input$threshold` updates, triggering a recomputation of `output$correlation_plot`.

7.3.2 Reactivity: The Heart of Shiny

Reactivity is Shiny's mechanism for connecting inputs to outputs. A reactive expression recomputes only when its dependencies change:

```
server <- function(input, output) {
  # Reactive expression: computes only when threshold changes
  net_data <- reactive({
    cor_mtcars <- cor(mtcars)
    adj <- (abs(cor_mtcars) > input$threshold) * 1
    diag(adj) <- 0
    graph_from_adjacency_matrix(adj, mode = "undirected", weighted = TRUE)
  })

  # Outputs can depend on reactive expressions
  output$net_viz <- renderPlot({
    net <- net_data()
    plot(net)
  })

  output$summary_stats <- renderTable({
    net <- net_data()
    tibble(
      Metric = c("Nodes", "Edges", "Density"),
      Value = c(vcount(net), ecount(net), edge_density(net))
    )
  })
}
```

Notice: `net_data()` is defined once but used in multiple outputs. Both outputs automatically update together when `input$threshold` changes. This dependency tracking is automatic in Shiny (you need not explicitly code update sequences).

7.3.3 Common UI Input Controls

Shiny provides many input types:

```
# Slider for continuous values
sliderInput("threshold", "Threshold:", min = 0, max = 1, value = 0.5)

# Select dropdown for choices
selectInput("variable", "Variable:", choices = c("x1", "x2", "x3"))

# Radio buttons for mutually exclusive options
radioButtons("method", "Method:",
             choices = c("Pearson", "Spearman", "Kendall"))
```

```r
# Checkbox for binary options
checkboxInput("show_labels", "Show node labels", value = TRUE)

# Multiple selection
selectInput("subgroups", "Filter by group:",
            choices = unique(data$group),
            multiple = TRUE)

# Date range picker
dateRangeInput("date_range", "Select date range:",
               start = "2020-01-01", end = "2023-12-31")
```

7.3.4 Common Output Types

Similarly, outputs can render various visualizations and tables:

```r
# Plot output
plotOutput("myplot")

# Table output
tableOutput("mytable")
# or for prettier tables:
dataTableOutput("interactive_table")

# Text output
textOutput("summary_text")

# HTML output (for custom formatting)
htmlOutput("custom_html")

# UI output (dynamically generated controls)
uiOutput("dynamic_controls")
```

7.4 Part III: Interactive Association Exploration

7.4.1 Example: Threshold Explorer

Let us build a practical interactive tool that explores how network structure changes as the correlation threshold varies:

```r
library(shiny)
library(igraph)
library(tidygraph)
library(ggraph)
library(ggplot2)
library(dplyr)

ui <- fluidPage(
  titlePanel("Correlation Network Threshold Explorer"),
  sidebarLayout(
    sidebarPanel(
      sliderInput("threshold",
                  "Correlation Threshold:",
                  min = 0, max = 1, value = 0.5, step = 0.05),
      helpText("Adjust to see how network structure changes."),
      br(),
      downloadButton("downloadPlot", "Download Plot")
    ),
    mainPanel(
      tabsetPanel(
        tabPanel("Network Visualization",
                 plotOutput("network_plot", height = "600px")),
        tabPanel("Network Statistics",
                 tableOutput("network_stats")),
        tabPanel("Correlation Distribution",
                 plotOutput("cor_distribution"))
      )
    )
  )
)

server <- function(input, output, session) {
  # Reactive network data
  net_reactive <- reactive({
    cor_mtcars <- cor(mtcars)
    adj <- (abs(cor_mtcars) > input$threshold) * 1
    diag(adj) <- 0
    graph_from_adjacency_matrix(adj, mode = "undirected", weighted = TRUE)
  })

  # Network plot
  output$network_plot <- renderPlot({
```

```
  net <- net_reactive()
  net_tidy <- as_tbl_graph(net)

  ggraph(net_tidy, layout = 'fr') +
    geom_edge_link(edge_colour = "gray60", alpha = 0.5) +
    geom_node_point(size = 5, colour = "steelblue") +
    geom_node_text(aes(label = name), repel = TRUE, size = 3) +
    labs(title = paste("Network at Threshold =", input$threshold)) +
    theme_void()
})

# Network statistics table
output$network_stats <- renderTable({
  net <- net_reactive()
  tibble(
    Metric = c("Nodes", "Edges", "Density", "Average Degree", "Diameter"),
    Value = c(
      vcount(net),
      ecount(net),
      round(edge_density(net), 3),
      round(mean(degree(net)), 3),
      ifelse(ecount(net) > 0, diameter(net), NA)
    )
  )
}, striped = TRUE, bordered = TRUE)

# Distribution of all correlations
output$cor_distribution <- renderPlot({
  cor_mtcars <- cor(mtcars)
  cor_values <- cor_mtcars[upper.tri(cor_mtcars)]

  ggplot(tibble(correlation = cor_values), aes(x = correlation)) +
    geom_histogram(bins = 20, fill = "steelblue", alpha = 0.7) +
    geom_vline(xintercept = input$threshold, colour = "red", linetype = "dashed", size = 1)
    geom_vline(xintercept = -input$threshold, colour = "red", linetype = "dashed", size =
    labs(title = "Distribution of All Correlations",
         subtitle = "Red lines show current threshold",
         x = "Correlation", y = "Frequency") +
    theme_minimal()
})

# Download handler
```

```
  output$downloadPlot <- downloadHandler(
    filename = function() {
      paste0("network_threshold_", input$threshold, ".png")
    },
    content = function(file) {
      png(file, width = 800, height = 600)
      net <- net_reactive()
      net_tidy <- as_tbl_graph(net)
      plot(ggraph(net_tidy, layout = 'fr') +
             geom_edge_link(edge_colour = "gray60", alpha = 0.5) +
             geom_node_point(size = 5, colour = "steelblue") +
             geom_node_text(aes(label = name), repel = TRUE) +
             theme_void())
      dev.off()
    }
  )
}

shinyApp(ui, server)
```

7.4.2 Example: Subgroup Comparison

Interactive tools excel at comparing patterns across subgroups. Here is an application that compares correlation networks across iris species:

```
ui <- fluidPage(
  titlePanel("Iris Network by Species"),
  sidebarLayout(
    sidebarPanel(
      selectInput("species", "Select Species:",
                  choices = unique(iris$Species)),
      sliderInput("threshold",
                  "Correlation Threshold:",
                  min = 0, max = 1, value = 0.5, step = 0.05),
      radioButtons("layout", "Graph Layout:",
                   choices = c("Force-Directed" = "fr",
                               "Circle" = "circle"))
    ),
    mainPanel(
      plotOutput("species_network", height = "600px"),
      br(),
```

```
      tableOutput("centrality_table")
    )
  )
)

server <- function(input, output, session) {
  # Filter data by species
  species_data <- reactive({
    iris |>
      filter(Species == input$species) |>
      select(-Species)
  })

  # Compute network for selected species
  species_net <- reactive({
    data <- species_data()
    cor_mat <- cor(data)
    adj <- (abs(cor_mat) > input$threshold) * 1
    diag(adj) <- 0
    graph_from_adjacency_matrix(adj, mode = "undirected", weighted = TRUE)
  })

  # Plot network with user-selected layout
  output$species_network <- renderPlot({
    net <- species_net()
    net_tidy <- as_tbl_graph(net)

    layout_func <- if (input$layout == "fr") {
      "fr"
    } else {
      "circle"
    }

    ggraph(net_tidy, layout = layout_func) +
      geom_edge_link(edge_colour = "gray60", alpha = 0.6) +
      geom_node_point(size = 6, colour = "coral") +
      geom_node_text(aes(label = name), repel = TRUE, size = 4) +
      labs(title = paste("Network for", input$species)) +
      theme_void()
  })

  # Node centrality measures
```

```
  output$centrality_table <- renderTable({
    net <- species_net()
    tibble(
      Variable = names(species_data()),
      Degree = degree(net),
      Betweenness = round(betweenness(net), 2),
      Closeness = round(closeness(net), 3)
    ) |>
      arrange(desc(Degree))
  }, striped = TRUE)
}

shinyApp(ui, server)
```

7.5 Part IV: Performance and Reactive Programming Patterns

7.5.1 Avoiding Redundant Computation

As apps grow complex, they can become slow if computations repeat unnecessarily. The solution is strategic use of **reactive expressions**:

```
# Bad: Correlations computed twice
server <- function(input, output) {
  output$heatmap <- renderPlot({
    cor_data <- cor(mtcars)  # Computed here
    # plot heatmap
  })

  output$network <- renderPlot({
    cor_data <- cor(mtcars)  # Computed again!
    # plot network
  })
}

# Good: Correlations computed once, reused
server <- function(input, output) {
  cor_reactive <- reactive({
    cor(mtcars)
  })
```

```
  output$heatmap <- renderPlot({
    cor_data <- cor_reactive()  # Uses cached result
    # plot heatmap
  })

  output$network <- renderPlot({
    cor_data <- cor_reactive()  # Reuses same result
    # plot network
  })
}
```

7.5.2 Debouncing and Lazy Evaluation

Some inputs change very rapidly (e.g., a slider dragged quickly). Without debouncing, Shiny recomputes on every drag. Solutions include:

```
# Debounce: only recompute 0.5 seconds after slider stops moving
threshold_debounced <- debounce(reactive(input$threshold), 500)

server <- function(input, output) {
  output$plot <- renderPlot({
    # Uses debounced threshold; only recomputes after user stops dragging
    threshold <- threshold_debounced()
    # ... plot code ...
  })
}

# Alternatively, use an "Apply" button to batch updates
server <- function(input, output) {
  apply_button <- eventReactive(input$apply_button, {
    list(threshold = input$threshold,
         species = input$species)
  })

  output$plot <- renderPlot({
    params <- apply_button()  # Only recomputes when button is clicked
    # ... plot code ...
  })
}
```

7.5.3 Conditional Panels

Apps often need dynamic UI: show certain controls only when relevant. Shiny provides `conditionalPanel()`:

```
ui <- fluidPage(
  sidebarPanel(
    selectInput("plot_type", "Plot Type:",
                choices = c("Network", "Heatmap", "Scatter")),

    # Show network options only when "Network" is selected
    conditionalPanel(
      condition = "input.plot_type == 'Network'",
      sliderInput("threshold", "Threshold:", min = 0, max = 1, value = 0.5),
      radioButtons("layout", "Layout:",
                   choices = c("fr", "circle", "spring"))
    ),

    # Show heatmap options only when "Heatmap" is selected
    conditionalPanel(
      condition = "input.plot_type == 'Heatmap'",
      radioButtons("heatmap_method", "Distance Method:",
                   choices = c("euclidean", "correlation"))
    )
  ),
  mainPanel(
    plotOutput("main_plot")
  )
)
```

7.6 Part V: Design Principles for Interactive Exploration Tools

7.6.1 Principle 1: Progressive Disclosure

Users can be overwhelmed by too many options at once. It helps to group controls logically and use tabs or collapsible panels:

```
ui <- fluidPage(
  tabsetPanel(
    tabPanel("Data Overview",
             # Simple view: just load and show data
```

```
            fileInput("data_file", "Choose CSV"),
            tableOutput("data_preview")),

    tabPanel("Exploration",
            # Intermediate view: threshold, subset options
            sliderInput("threshold", "Threshold:", 0, 1, 0.5)),

    tabPanel("Advanced",
            # Advanced view: algorithms, layouts, statistics
            radioButtons("algorithm", "Algorithm:", ...),
            radioButtons("layout", "Layout:", ...))
  )
)
```

7.6.2 Principle 2: Instant Feedback

Users often benefit from immediate visual feedback. Try to avoid long computations. Instead:

- Precompute expensive steps (e.g., large network layouts) at app startup
- Use `withProgress()` to show a progress bar for longer operations
- Debounce rapid input changes

7.6.3 Principle 3: Clarity Over Features

Too many controls can confuse users. Consider prioritizing:

1. **Essential controls**: What do users want to change first?
2. **Sensible defaults**: Pre-populate with reasonable starting values
3. **Documentation**: Include help text explaining each control

7.6.4 Principle 4: Comparison Facilitates Discovery

When comparing across subgroups or time periods, arrange views side-by-side or with linked highlighting:

```
ui <- fluidPage(
  fluidRow(
    column(6, h3("Species A"), plotOutput("net_a")),
    column(6, h3("Species B"), plotOutput("net_b"))
  )
)
```

7.7 Part VI: Limitations of Interactive Tools

While powerful, interactive exploration has drawbacks:

1. **Reproducibility**: Interactive explorations are harder to document than scripts. Always maintain an underlying analysis script.
2. **Scalability**: Real-time computation works only for moderately large datasets. Networks with 10,000+ nodes often require pre-computation or sampling.
3. **Learning curve**: Non-technical users need training to use interactive apps effectively.
4. **Maintenance burden**: Apps require hosting and updates; static reports are simpler to archive.

Best practice: Use interactive tools for *exploration*, then document findings in static reports.

7.8 Part VII: From Principles to Practice

The chapters so far have built the foundation for association analysis:

- **Chapters 1-3**: Data types and univariate/bivariate summaries
- **Chapter 4**: Association measures for mixed types
- **Chapter 5**: Nonlinear and conditional associations
- **Chapter 6**: Network representations
- **Chapter 7**: Interactive exploration (this chapter)

Now we integrate all these components into a cohesive, interactive tool. The next chapter introduces **AssociationExplorer**, a Shiny application designed specifically for exploratory association analysis. AssociationExplorer combines:

- **Flexible association measures**: Automatically select appropriate measures for mixed-type data
- **Interactive parameter adjustment**: Threshold, community detection algorithms, layouts
- **Multiple linked views**: Network, correlation matrix, summary statistics, and variable details
- **Data filtering and subgrouping**: Explore associations for specific observations or categories
- **Publication-ready export**: Save visualizations and data summaries

AssociationExplorer demonstrates how the principles and techniques from this chapter, combined with sound statistical practice and user-centered design, enable practitioners to explore high-dimensional association structures interactively, discover patterns that static analysis would miss, and communicate findings effectively to diverse audiences.

7.9 Summary and Key Takeaways

- **Static analysis freezes findings in parameter space**: Each figure answers one specific question; new questions require recoding.
- **Interactive tools enable iterative discovery**: Parameters adjust in real-time, subsets filter instantly, and multiple views stay synchronized.
- **Shiny separates UI and server logic**: Inputs drive outputs through reactive expressions that automatically track dependencies.
- **Reactivity is central to Shiny's power**: Declaring dependencies once means outputs update together without explicit orchestration.
- **Performance requires strategic caching**: Use reactive expressions to compute expensive operations once and reuse results.
- **Good interactive tools follow design principles**: Progressive disclosure, instant feedback, clarity, and comparison-oriented layouts make exploration intuitive.
- **Interactive and static methods are complementary**: Use interactivity for exploration, static reports for communication.

7.10 Looking Ahead

With Shiny principles established, we now turn to a complete, production-ready application: **AssociationExplorer**. The following chapter showcases how these principles integrate into a comprehensive tool for exploring, understanding, and communicating association structures in complex tabular data.

8 The AssociationExplorer Application

8.1 Introduction: From Principles to Practice

The previous chapter suggested that **interactive tools can transform association analysis from static snapshots into dynamic exploration**. We learned:

- How reactive expressions automate dependency tracking
- How UI controls enable parameter adjustment without code modification
- How multiple linked views facilitate pattern discovery
- How performance patterns keep interactions instantaneous

We now demonstrate these principles in practice through **AssociationExplorer**, a production-ready Shiny application designed for exploratory association analysis (Soetewey et al. 2026). Rather than building toy examples, we examine a complete, deployed application that serves real users: journalists analyzing survey data, educators explaining correlation structures to students, and researchers discovering patterns in complex multivariate datasets.

8.1.1 How to Open the App

You can launch **AssociationExplorer** directly from GitHub:

```
library(shiny)
runGitHub("AssociationExplorer", "AntoineSoetewey")
```

Or via the CRAN package:

```
library(AssociationExplorer2)
run_associationexplorer()
```

8.2 Design Philosophy

8.2.1 Who Is AssociationExplorer For?

AssociationExplorer targets **non-technical practitioners** who need to explore associations but lack advanced statistical or programming expertise. This design choice shapes many architectural decisions:

1. **Automatic variable type detection**: Users need not specify whether variables are quantitative or qualitative; the app infers types and applies appropriate association measures.
2. **Sensible defaults**: The first time a user loads the app, they encounter preselected thresholds and clear navigation without overwhelming options.
3. **Minimal statistical jargon**: Labels like "Correlation Network" replace "Directed Acyclic Graphs." Help text defines technical terms inline.
4. **One-click visualization**: Rather than forcing users to write code, a single button produces a publication-ready network plot.

8.2.2 What Problems Does It Solve?

Before interactive tools, analysts asking "What associations exist in this dataset?" faced three sub-questions:

1. **Which associations matter?** Compute pairwise measures for all variables, then manually filter by strength.
2. **How do they interconnect?** Draw a network by hand or spend hours in visualization software.
3. **Do I trust the pattern?** Validate by hand-checking individual variable pairs.

AssociationExplorer streamlines this workflow into: upload data → adjust sliders → explore. The app automatically handles:

- Heterogeneous variable types (mixing quantitative and categorical variables)
- Missing data (computing pairwise associations only on complete cases)
- Computational efficiency (precomputing expensive operations)
- Visual clarity (automatically arranging network layout)

8.3 Part I: Application Architecture

8.3.1 The Workflow Tabs

AssociationExplorer organizes the analysis pipeline into five sequential **tabs**, each corresponding to a distinct analytic stage. This structure implements the **progressive disclosure** principle from Chapter 7: novice users follow tabs in order, while experienced users jump between tabs as needed.

8.3.1.1 Tab 1: Data Upload

Users begin in the **Data** tab, which handles three core tasks:

1. **Upload the dataset** in CSV or Excel format
2. **(Optionally) upload variable descriptions** for interpretability
3. **Automatically clean and validate** the data

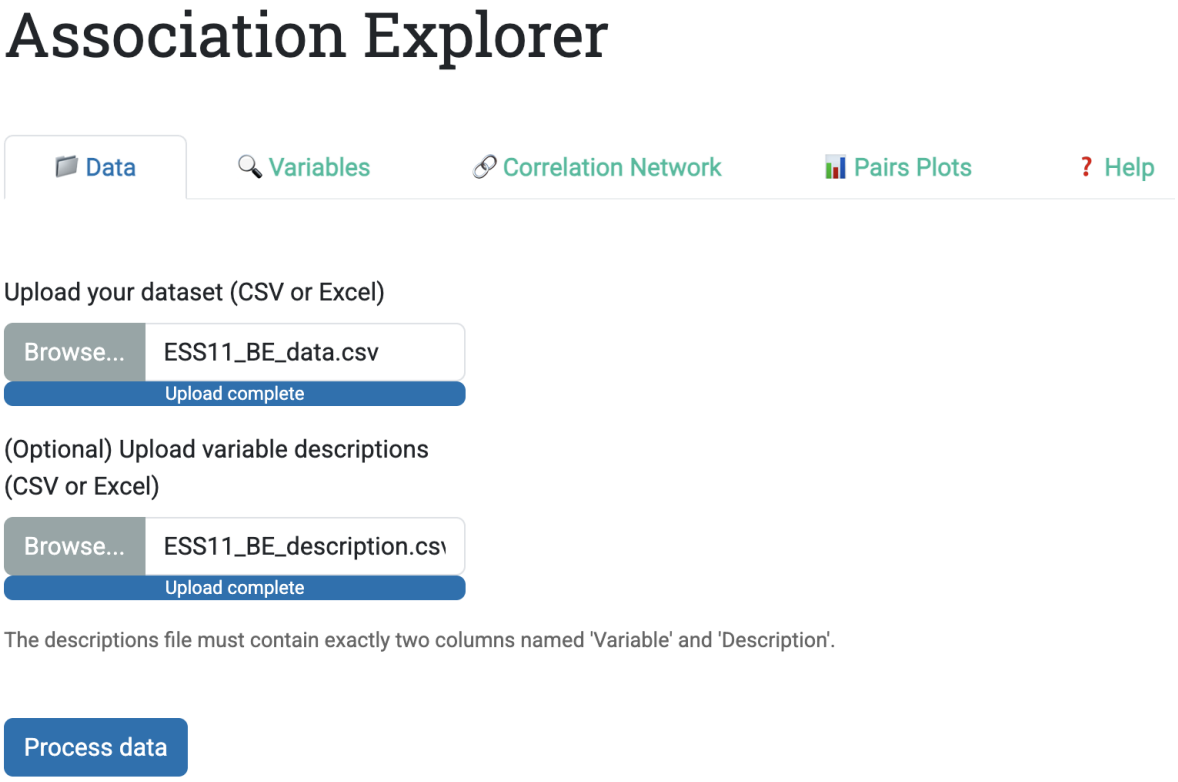


Figure 8.1: The Data tab showing the upload interface for datasets and variable descriptions

The upload interface is deliberately simple: two file inputs (data + descriptions) and one button ("Process data"). Behind this simplicity sits a set of practical checks:

- CSV files require comma separation and dot decimals (European format support exists but requires explicit note in UI)
- Excel files (.xlsx) are read directly without format specification
- Variables with zero variance (only one unique value) are automatically removed with a warning
- Missing data is preserved, and the app computes pairwise associations only on complete cases

Why this matters: Many data exploration tools struggle with real data because they ignore missing values or crash on degenerate variables. AssociationExplorer anticipates these problems and handles them quietly.

8.3.1.2 Tab 2: Variable Selection

Once data is loaded, the **Variables** tab allows users to select which variables to analyze. Implementation details matter:

- A **multi-select dropdown** (not checkboxes) scales to 50+ variables without overwhelming the UI
- Selected variables are retained as defaults, reducing clicks if users make small adjustments
- If a descriptions file was uploaded, a **reactive table** displays selected variables alongside their descriptions

This design supports two common workflows:

- **Exploratory**: Start with all variables, then iteratively exclude irrelevant ones
- **Confirmatory**: Select a pre-defined subset of theoretically important variables

8.3.1.3 Tab 3: Correlation Network

The **Correlation Network** tab is where association patterns become visible. The sidebar contains two key controls:

1. **Threshold for quantitative-quantitative and quantitative-categorical associations**: Filtered by R^2 (the coefficient of determination), ranging 0-1. Default: 0.5.
2. **Threshold for categorical-categorical associations**: Filtered by Cramér's V, ranging 0-1. Default: 0.5.

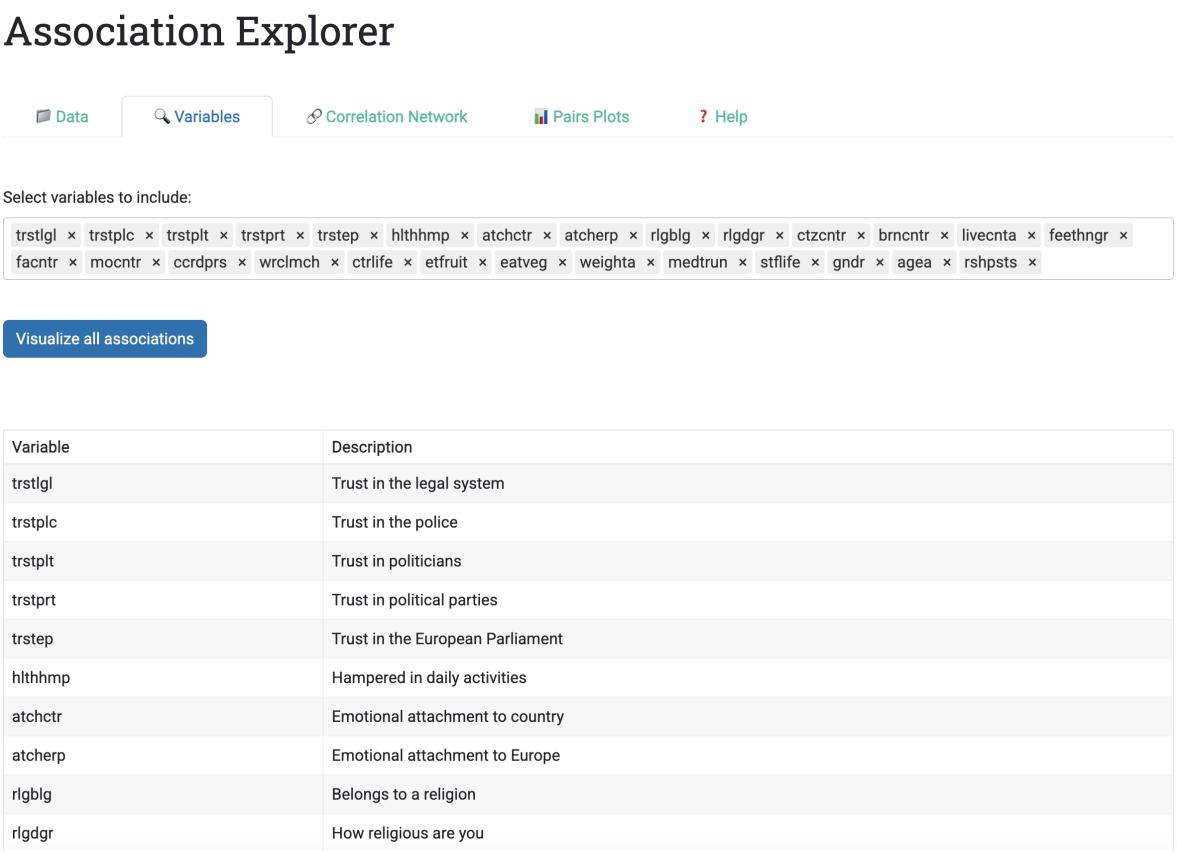


Figure 8.2: The Variables tab showing selected variables and their descriptions

Why two thresholds? Because association measures are **scale-dependent**: Pearson's r ranges $[-1, 1]$, Eta ranges $[0, 1]$, and Cramér's V ranges $[0, 1]$. Setting a single threshold can systematically bias results toward one variable type. AssociationExplorer lets users adjust each threshold independently.

The main panel displays an **interactive network visualization** built with the `visNetwork` R package. The visualization encodes association strength in two ways:

- **Edge thickness**: Stronger associations lead to thicker edges (visual pop-out effect)
- **Edge length**: Stronger associations lead to shorter edges (physical clustering)

For numeric pairs, edge color conveys direction:

- **Blue edges**: Positive associations
- **Red edges**: Negative associations

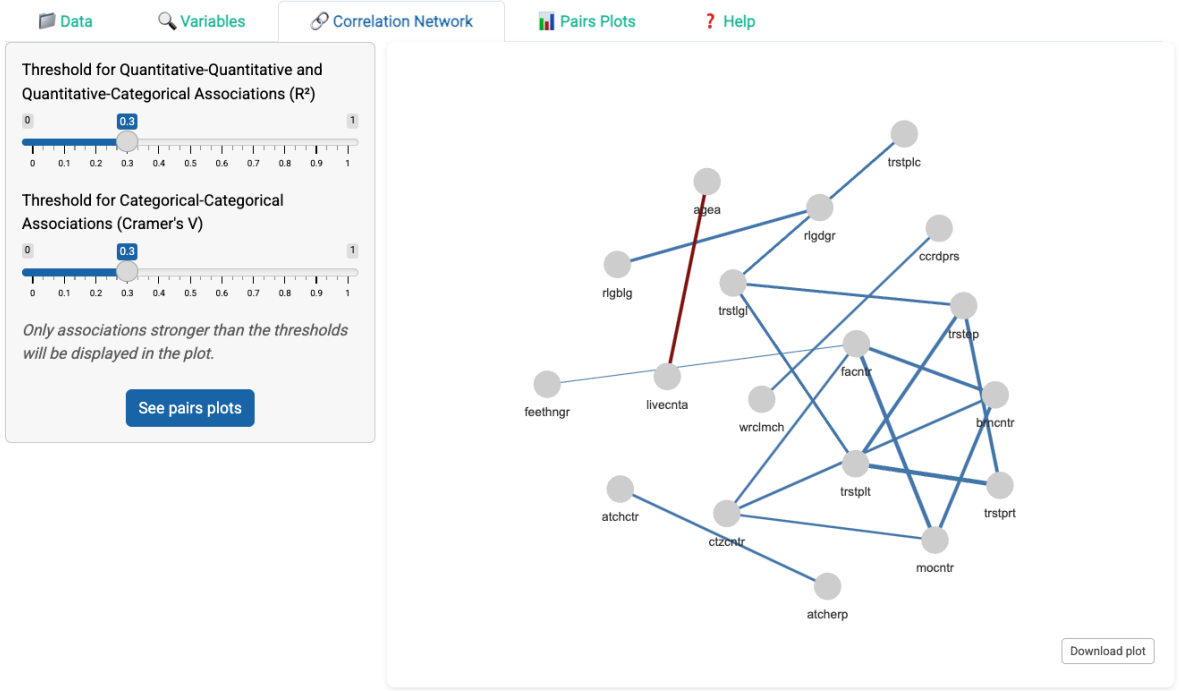


Figure 8.3: The Correlation Network tab showing the interactive network visualization with adjustable thresholds

Hovering over a node reveals its full description (if provided in the descriptions file), transforming cryptic variable codes into readable labels.

8.3.1.4 Tab 4: Pairs Plots

Once users identify associations of interest in the network, the **Pairs Plots** tab displays bivariate visualizations for each association above the thresholds. The app generates three plot types automatically based on variable types:

1. **Numeric vs. Numeric**: Scatter plot with overlaid linear regression line (with jitter to reveal density)
2. **Numeric vs. Categorical**: Horizontal bar chart of category means, sorted to reveal patterns
3. **Categorical vs. Categorical**: Color-coded contingency table with marginal totals

Each plot includes a **download button** enabling one-click export as PNG. This removes the usual export friction: no copy-paste, no screenshot tools, just one click and the image is ready for presentations or reports.

8.3.1.5 Tab 5: Help

The **Help** tab provides a concise, numbered workflow guide. Unlike static documentation, it lives in the app itself, always available and always relevant.

8.3.2 The Computation Engine

Behind the UI lies the **correlation computation engine**, which automatically:

1. **Detects variable types** for each pairwise combination
2. **Applies the appropriate association measure**:
    - Pearson's r for numeric-numeric pairs
    - Eta for numeric-categorical pairs
    - Cramér's V for categorical-categorical pairs
3. **Applies thresholds** (dropping weak associations)
4. **Constructs an association matrix** used by all downstream outputs

This engine is reactive: when users adjust a threshold slider, the matrix recomputes quickly (thanks to strategic caching with `reactive()` expressions). All downstream outputs (network visualization, pairs plots, summary tables) update together.

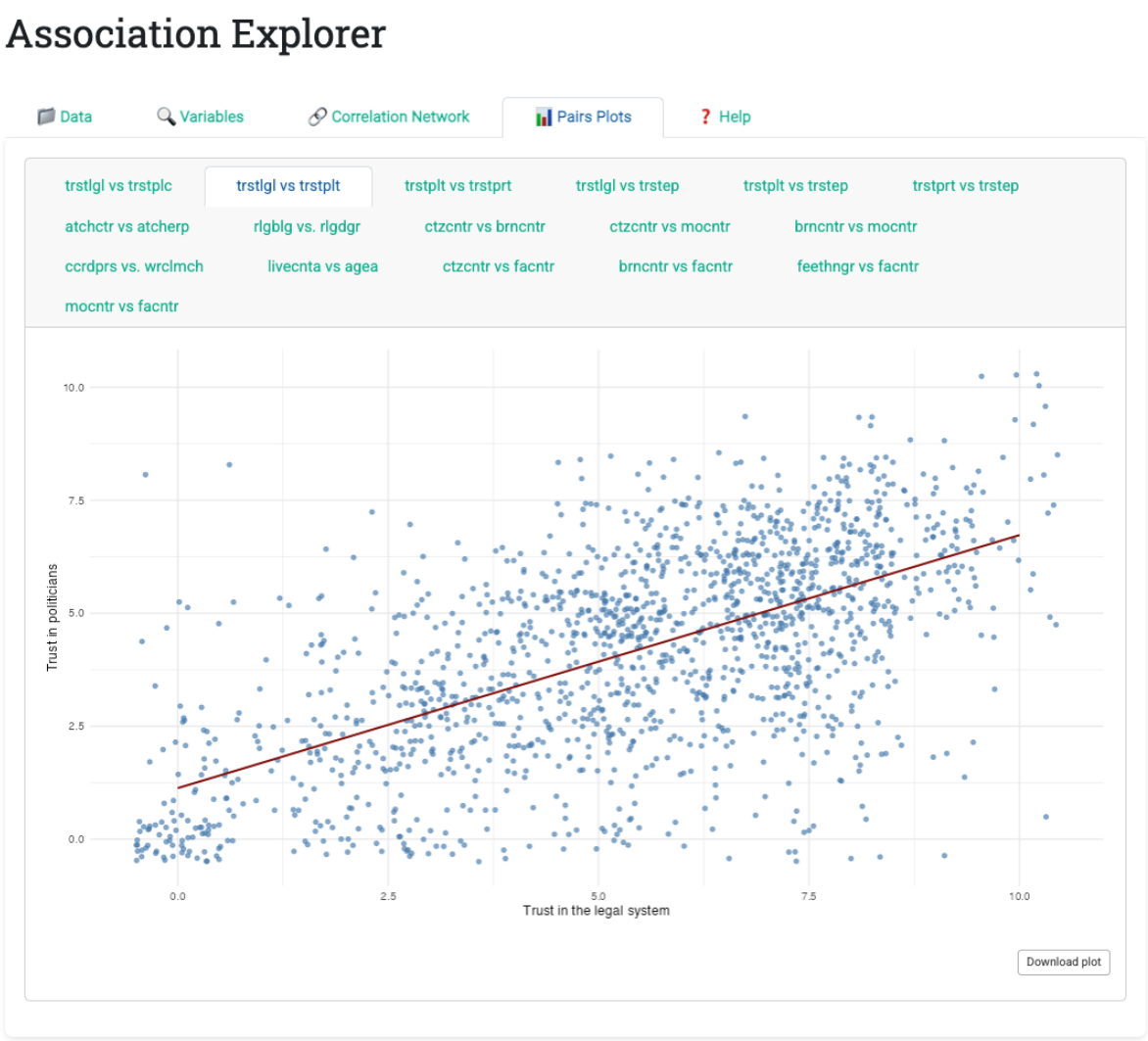


Figure 8.4: Example of a scatter plot from the Pairs Plots tab showing a numeric-numeric association

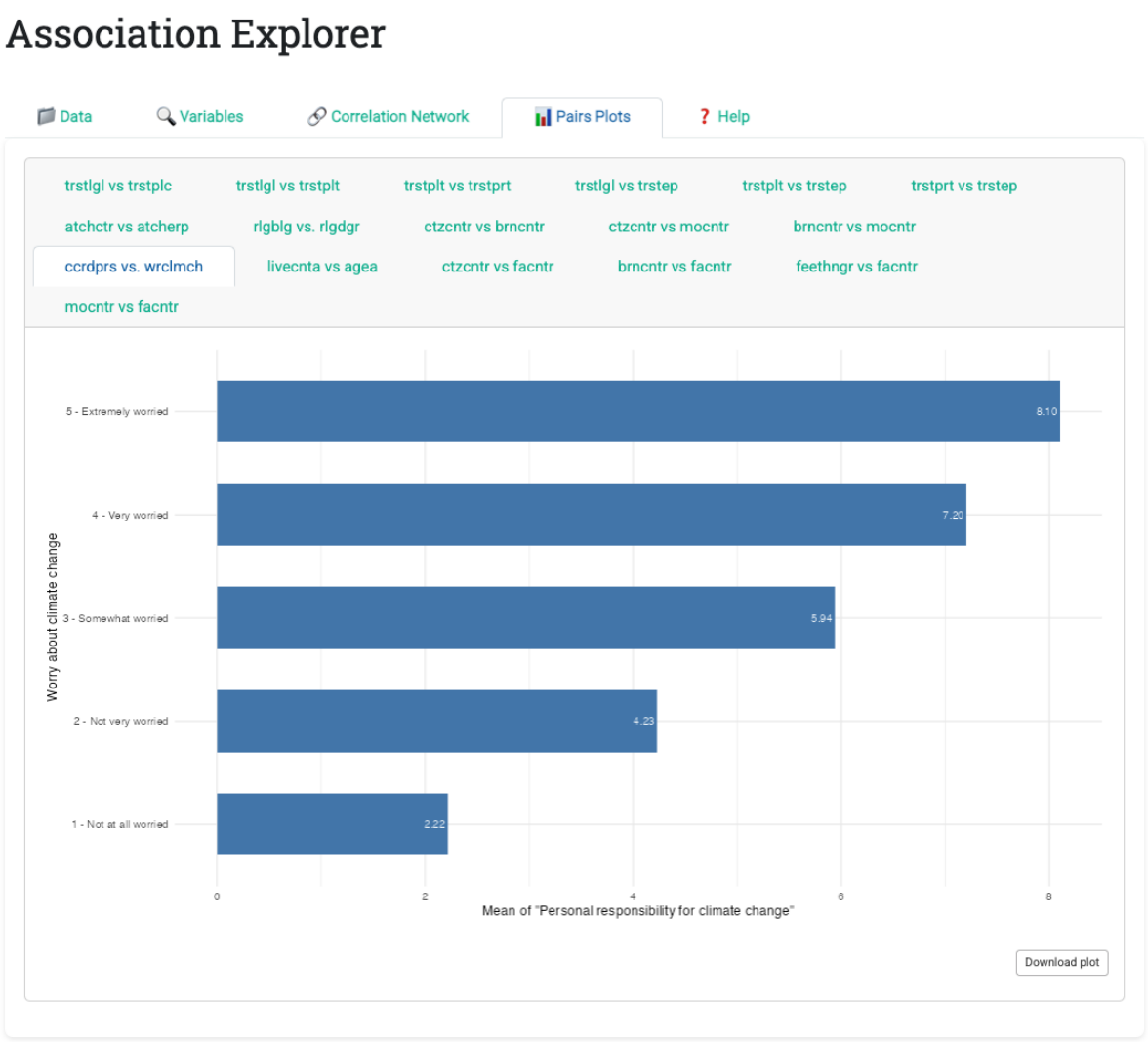


Figure 8.5: Example of a bar chart from the Pairs Plots tab showing a numeric-categorical association

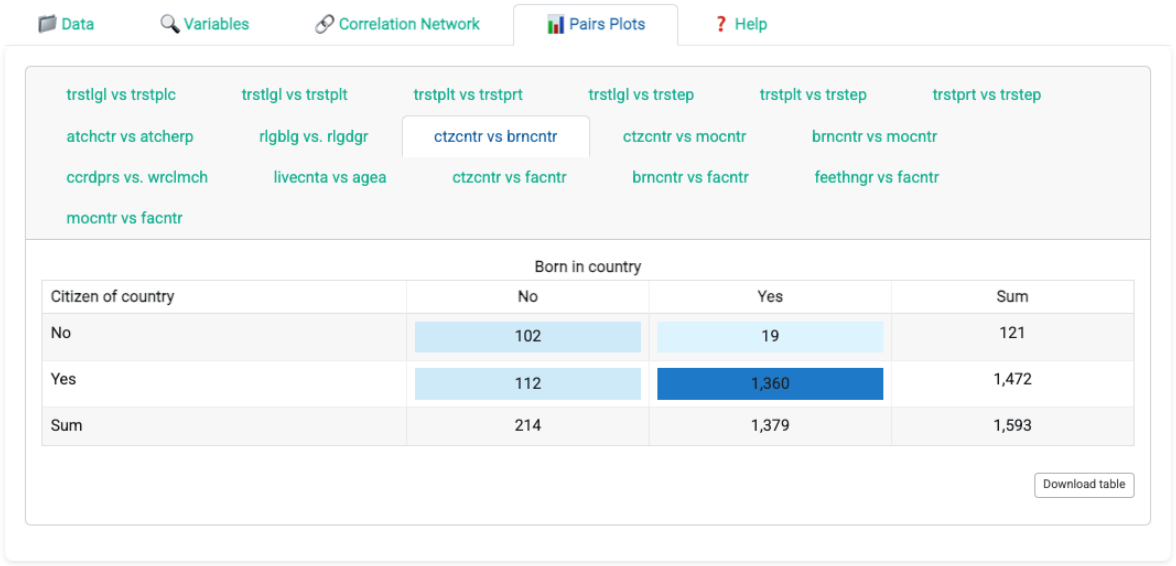


Figure 8.6: Example of a contingency table from the Pairs Plots tab showing a categorical-categorical association

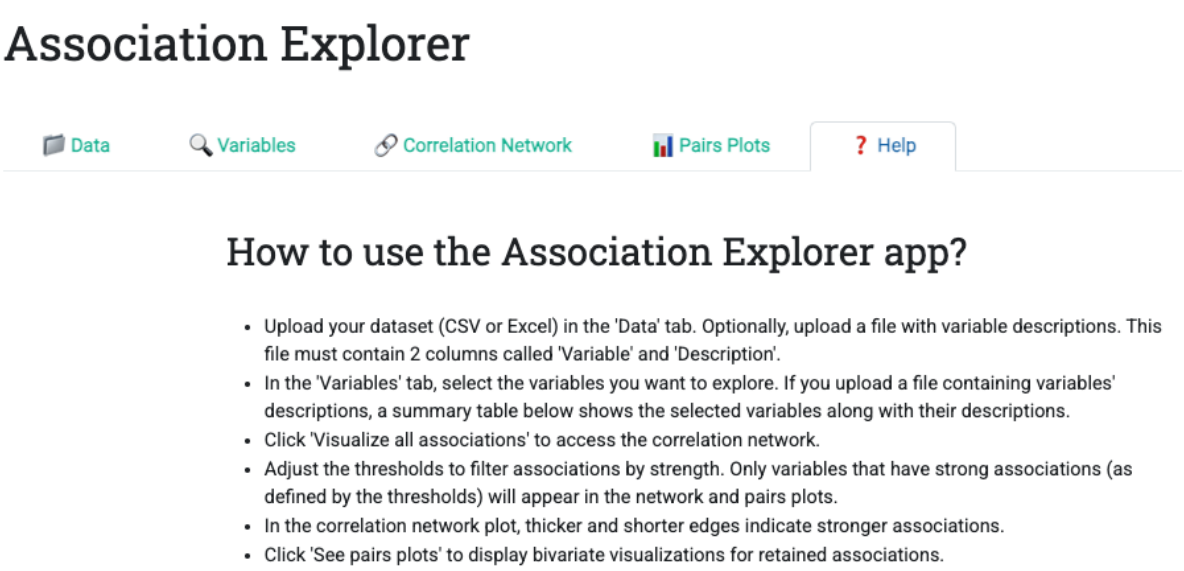


Figure 8.7: The Help tab providing guidance on how to use the application

8.3.3 Reactivity in Practice

AssociationExplorer demonstrates several reactive programming patterns worth studying:

Pattern 1: Cached Computation

```
cor_matrix_reactive <- reactive({
  req(data())
  selected_vars <- valid_selected_vars()
  selected_data <- data()[, selected_vars, drop = FALSE]
  calculate_correlations(selected_data,
                         input$threshold_num,
                         input$threshold_cat)
})
```

This reactive expression:

- Waits until `data()` is available (`req()` blocks evaluation if NULL)
- Recomputes only when selected variables or thresholds change
- Returns both the correlation matrix and the type of each correlation (Pearson, Eta, Cramér)

All downstream outputs (network, pairs plots, statistics) depend on this single expression, ensuring consistency.

Pattern 2: Filtered Subsets

```
filtered_data_for_pairs <- reactive({
  mat <- cor_matrix_vals()$cor_matrix
  nodes_to_keep <- rowSums(abs(mat) > 0) > 1  # Keep variables with >1 edge
  filtered_matrix <- mat[nodes_to_keep, nodes_to_keep]
  data()[, colnames(filtered_matrix), drop = FALSE]
})
```

This expression identifies which variables appear in the final network (i.e., have at least one association above the threshold), then filters the raw data to include only those variables. Subsequent bivariate plots are computed only for this reduced dataset, improving performance.

Pattern 3: Conditional Rendering

```
output$pairs_plot <- renderUI({
  req(input$main_tabs == "pairs_tab")  # Only compute if tab is active
  pairs <- significant_pairs()
  if (is.null(pairs) || nrow(pairs) == 0) {
    return(tags$p("No associations exceed the threshold..."))
  }
  # ... generate plots for all pairs
})
```

Plots are generated only when the **Pairs Plots** tab is active (`req(input$main_tabs == "pairs_tab")`), avoiding wasted computation if users never view that tab.

8.4 Part II: Handling Real Data Complexity

8.4.1 Mixed Variable Types

A defining feature of AssociationExplorer is its **automatic handling of mixed variable types**. Traditional statistical software forces users to specify analysis type: "correlation for numeric variables" or "chi-square for categorical variables" (Pearson 1992). AssociationExplorer unifies these under a single framework.

The underlying principle is that for each pair of variables, the appropriate association measure is chosen based on their types.

Variable 1 Type	Variable 2 Type	Association Measure	Interpretation
Numeric	Numeric	Pearson's r	Linear correlation; range [-1, 1]
Numeric	Categorical	Eta	Association strength; range [0, 1]
Categorical	Categorical	Cramér's V	Association strength; range [0, 1]

Implementation challenge: Ensuring comparability across measures. The app thresholds using R^2 (coefficient of determination) for numeric measures:

- Pearson: threshold uses r^2 (squared correlation)
- Eta: threshold uses η^2 (squared effect size)

For Cramér's V, the threshold applies directly (since V already ranges [0, 1]).

This design choice has implications:

- A threshold of 0.5 on R^2 can be read as "variables explain 50% of each other's variance"
- The same threshold on Cramér's V can be read as "variables have a specific association strength of at least 0.71" (since $0.71^2 \approx 0.5$)

The **Help** tab clarifies these distinctions; users need not understand the mathematics, but they benefit from consistent, interpretable results.

8.4.2 Missing Data Handling

Real datasets contain missing values. AssociationExplorer handles them pragmatically:

1. **Pairwise deletion**: For each variable pair, compute the association using only observations with no missing values in either variable
2. **Separate thresholds**: Apply thresholds independently to each measure type
3. **Transparent reporting**: Display the number of valid cases used in each bivariate plot

Example: If a dataset has 1000 rows, and a particular numeric pair has 950 complete cases, the correlation is computed on those 950. The scatter plot displays these 950 points, with jitter applied to reveal overlapping observations.

This approach balances **statistical rigor** (use all available data) with **practical clarity** (show users exactly what is being plotted).

8.4.3 Performance Optimization

AssociationExplorer processes datasets up to 10,000+ rows interactively. Performance is managed through:

1. **Precomputation**: Upon data load, the app computes all pairwise associations once and caches them
2. **Lazy rendering**: Bivariate plots are generated only when their tab is viewed
3. **Subset filtering**: The pairs plot engine generates only plots for variable pairs above the threshold
4. **Efficient network visualization**: The `visNetwork` package uses GPU acceleration for layout algorithms

8.5 Part III: User Experience Patterns

8.5.1 Progressive Disclosure in Action

The tab structure exemplifies the **progressive disclosure** principle from Chapter 7. Rather than overwhelming users with all options simultaneously, AssociationExplorer reveals functionality in stages:

Implementation: Tab-based navigation (Data → Variables → Network → Pairs → Help) guides users through a linear workflow with optional shortcuts for advanced users. Each tab presents one analytic decision without exposing irrelevant options.

1. **First-time users** follow tabs sequentially. Each step introduces one analytic stage, building understanding gradually.
2. **Experienced users** jump strategically: Load data, skip to Network tab, adjust thresholds, then examine pairs plots of interest.
3. **Iterative explorers** cycle between Network and Pairs Plots, discovering patterns through threshold adjustment and focused bivariate inspection.

Benefit: Novices are never confronted with all options; the UI supports all three workflows without requiring configuration.

8.5.2 Visual Feedback and Instant Response

AssociationExplorer implements **instant feedback** (Chapter 7 design principle) to help users understand how their actions affect results:

Implementation: Reactive expressions ensure threshold adjustments update the network and pairs plots. Visual cues appear at every interaction:

- **Threshold adjustment**: Network updates in real-time, with edges appearing/disappearing as users drag sliders
- **Hover interactions**: Hovering over network nodes reveals full descriptions; hovering over table cells highlights the cell
- **Loading indicators**: Spinning loaders appear during computation, preventing confusion about whether the app is responsive
- **Empty-state messaging**: If no associations exceed the threshold, a clear message explains why and suggests adjusting thresholds

Benefit: This feedback loop is essential for non-technical users to build **mental models** of how thresholds affect results. Users see consequences of their actions immediately, enabling rapid exploration and hypothesis testing.

8.5.3 Comparison-Oriented Layouts

AssociationExplorer implements **comparison-oriented layouts** (Chapter 7 design principle) recognizing that patterns emerge through juxtaposition:

Implementation: Multiple linked views (network + pairs plots) enable side-by-side comparison. The pairs plots tab uses a **tabbed interface** allowing users to scroll through bivariate patterns across multiple associations without leaving the tab.

Users can ask questions like:

- "Do all numeric pairs show linear relationships, or are some nonlinear?"
- "How is the relationship between age and satisfaction?"
- "Which categorical pairs have more balanced contingency tables?"

Benefit: The **tabbed interface** enables rapid comparison; the **color-coded contingency tables** make patterns jump out. Users discover patterns by comparing perspectives. The network reveals which variables associate; the pairs plots reveal how.

8.6 Part IV: Real Example: European Social Survey

AssociationExplorer ships with a curated dataset: Belgian respondents from the **European Social Survey** (ESS11). This example demonstrates the app in a realistic setting. The full ESS dataset, codebook, and documentation are freely available at ess.sikt.no (European Social Survey European Research Infrastructure (ESS ERIC) 2025, 2024).

8.6.1 Dataset Description

The ESS dataset includes approximately 1,600 observations across 60 variables:

- **Demographic**: Age, gender, education level
- **Socioeconomic**: Occupational status, household income, employment situation
- **Attitudinal**: Trust in government, satisfaction with life, immigration attitudes

Variables mix numeric (continuous and discrete) and categorical (ordered and unordered) types, precisely the heterogeneity that makes traditional analysis workflows cumbersome.

8.6.2 Workflow Example

Step 1: Load the data
User uploads the ESS CSV file and (optionally) a descriptions file with human-readable labels.

Step 2: Select variables
All variables are preselected. User can keep all of them for a comprehensive analysis, or deselect some to focus on subsets.

Step 3: Set thresholds
User adjusts the R^2 and Cramér's V thresholds, starting with defaults (0.5) and then experimenting with lower (higher) thresholds to reveal more (fewer) associations.

Step 4: Inspect the network
The network visualization reveals clusters. For example, demographic variables cluster together, while attitudinal variables form another cluster. Some variables (e.g., education level) act as bridges between clusters, suggesting they associate with both demographics and attitudes.

Step 5: Dive into bivariate plots
User examines the education-immigration attitudes scatter plot, revealing a clear negative association: higher education correlates with more favorable immigration attitudes. The regression line visually communicates the pattern.

Next, the user compares life satisfaction across employment categories (full-time, part-time, unemployed) using the mean plots, discovering that life satisfaction is strongest for full-time workers.

8.6.3 Key Insights

This workflow demonstrates how AssociationExplorer **scales across complexity**:

- Simple datasets: Users see the network instantly, with clear clusters and bridging variables
- Complex datasets: Users adjust thresholds to focus on strong associations, avoiding overwhelm
- Mixed types: The app seamlessly compares numeric and categorical associations on equal footing

8.6.4 Additional Design Principle: Clarity Over Features

One more principle exemplified throughout: **clarity over features**. Only essential controls are visible by default. Advanced features (e.g., custom color schemes, algorithm selection) are omitted in favor of sensible defaults. The app remains approachable for non-technical users while remaining powerful enough for professional analysts.

8.7 Part V: Limitations and Future Work

8.7.1 Current Limitations

1. **Scalability**: Real-time computation becomes slow for datasets with 100+ variables. Future versions may implement sampling or pre-computed dissimilarity matrices.
2. **Advanced Measures**: The app supports only Pearson's r, Eta, and Cramér's V. Extensions could include polychoric correlations (for ordered categorical data) or mutual information (for nonlinear associations).
3. **Temporal Data**: No built-in support for time series. Extending the app to handle longitudinal data would require additional reactive architecture.
4. **Reproducibility**: Interactive explorations are hard to document and reproduce. Future versions may offer the possibility to export a summary report (PDF or HTML) capturing the final configuration and findings.

8.7.2 Strengths vs. Static Analysis

Aspect	Interactive (AssociationExplorer)	Static (Script-Based)
Exploration	Iterative, parameter-driven	Question-driven, manual
Accessibility	Non-technical users	Requires programming knowledge
Reproducibility	Challenging (requires capturing parameter settings)	Natural (code is the documentation)
Publication	Requires export step	Direct from script output
Parameter Sensitivity	Easy to visualize	Requires manual recomputation

Recommendation: Use AssociationExplorer for discovery and exploration; use static scripts for validation and publication.

8.8 Part VIII: Bridging to Publication

A critical challenge in exploratory analysis is **transitioning from discovery to communication**. AssociationExplorer addresses this through:

1. **One-click visualization export**: Save high-resolution PNG images ready for presentations

2. **Publication-ready defaults**: Network colors, edge widths, and font sizes are pre-tuned for readability
3. **Reproducibility metadata**: The app could (in future versions) export the full configuration used in exploration, enabling others to reproduce findings

Recommended workflow:

1. **Exploration phase**: Use AssociationExplorer to discover patterns
2. **Validation phase**: Confirm findings with formal statistical tests (hypothesis tests, confidence intervals)
3. **Communication phase**: Create static figures and tables for reports or presentations

8.9 Summary and Key Takeaways

- **Automatic type detection** enables analysis of heterogeneous datasets without manual preprocessing
- **Progressive disclosure** through tabs guides novices while empowering experienced users
- **Reactive computation** ensures that exploration feels instantaneous; users adjust parameters and see results immediately
- **Mixed association measures** unify numeric and categorical analysis under one framework
- **Multiple linked views** facilitate pattern discovery through comparison
- **User-centered design** prioritizes accessibility for non-technical practitioners over advanced statistical features
- **Iterative exploration** uncovers patterns that single-hypothesis testing often misses
- **Interactive and static workflows are complementary**: Use interactivity for discovery, static output for communication

8.10 Looking Ahead

With both Shiny principles (Chapter 7) and their practical implementation in AssociationExplorer (this chapter) established, we now turn to the next frontier: **interpreting and communicating association patterns effectively**.

The chapters that follow move beyond pair-wise associations to higher-level patterns:

- **Community detection** in networks: Which groups of variables cluster together?
- **Causal reasoning**: How do we move from observed associations to causal claims?
- **Visualization best practices**: How do we communicate complex patterns to non-technical audiences?

AssociationExplorer provides the **interactive tool** for exploration; subsequent chapters provide the **conceptual frameworks** for interpretation and communication.

9 Communicating Findings Through Visualization

9.1 Introduction: From Discovery to Communication

The previous chapters focused on discovery: we computed association measures, visualized networks, and built interactive tools for exploration. But discovery is only part of the journey. **Findings are most useful when they can be communicated clearly.**

Consider the workflow of a data analyst:

1. **Exploration phase** (Chapters 3–8): Compute summaries, test thresholds, examine associations interactively
2. **Validation phase**: Confirm patterns with formal tests, assess robustness to data perturbations
3. **Communication phase** (this chapter): Translate findings into visualizations and narratives for non-technical audiences

Many analyses struggle at step 3. A researcher may discover a striking pattern in an interactive tool, then find it difficult to convey in a static presentation. The network can look clear on screen but become hard to read in a printed report. The statistical p-value may be decisive for the analyst but less meaningful to a journalist or policymaker.

This chapter addresses the communication gap. We focus on principles and practices for translating association patterns into visualizations that inform rather than confuse, and that persuade through clarity rather than complexity.

9.2 Part I: Visualization Principles for Association Patterns

9.2.1 Encode the Right Information

Not all information deserves visual encoding. In association analysis, the key insights are typically:

- **Which variables associate?** (Identity)
- **How strong is the association?** (Magnitude)

- **What is the direction?** (Sign: positive or negative)
- **Are there outliers or exceptions?** (Variance)

Secondary information (exact correlation coefficients, sample sizes, confidence intervals) can appear in tables or captions without cluttering the visualization.

Principle: Encode primary insights visually; reserve tables for secondary details.

9.2.1.1 Visual Variables and Their Strengths

Different visual channels carry different cognitive weight:

Visual Channel	Strength	Best For	Pitfalls
Position	Strongest	Numeric magnitudes (scatterplot axes)	Requires careful scale choice
Length/Size	Very strong	Comparing quantities (bar height, node size)	Easy to misinterpret without baseline
Color (hue)	Moderate	Categorical distinctions	Not perceptually uniform; avoid rainbow palettes
Color (saturation/value)	Strong	Ordered categories or numeric ranges	Requires careful selection for accessibility
Texture/Pattern	Weak	Last resort for categorical data	Hard to distinguish; rarely necessary

Implication for association networks:

- Use **position** (network layout) to reveal clustering
- Use **edge thickness/length** (size channel) to encode strength
- Use **edge color** (hue) for direction or variable type
- Use **node size** only if a node attribute (e.g., degree) deserves emphasis

9.2.2 Design for Your Audience

Visualizations are most effective when tailored to audience expertise and context:

Academic audience (researchers, statisticians):

- Precision and rigor are valued
- Legends with exact scale ranges are expected

- Multiple views may be appropriate (show robustness)
- Methodological details belong in captions

Professional audience (managers, policymakers):

- Action and insight are valued
- Simple narratives beat nuanced caveats
- One clean visualization often beats three complex ones
- Consider trading precision for clarity when needed

Public audience (journalists, general readers):

- Story and emotion matter as much as data
- Metaphors and analogies help
- Avoid jargon where possible
- One key message per visualization

A single dataset may benefit from different visualizations for these audiences.

9.2.3 Reduce Cognitive Load

Our visual working memory is limited. Too many elements, colors, or overlapping lines can create **cognitive overload**, causing viewers to disengage rather than engage.

Strategies to reduce load:

- **Declutter**: Remove gridlines, axis spines, and decorative elements
- **Group and order**: Sort categories by value, arrange elements meaningfully (not alphabetically)
- **Highlight selectively**: Use color sparingly; make important elements pop by muting others
- **Segment**: Split complex patterns into small multiples (faceted plots) rather than one dense visualization
- **Annotate directly**: Label key points on the plot rather than forcing readers to match legend to data

9.2.4 Use Color Deliberately

Color is powerful but easily misused. Poor color choices can mislead, exclude readers with colorblindness, or fail in grayscale printing.

For categorical data: Use distinct, qualitatively different colors. Tools like ColorBrewer (colorbrewer2.org) provide tested palettes. Avoid rainbow palettes; they are perceptually non-uniform and difficult to distinguish.

For numeric data: Use a sequential palette (light to dark) for increasing values, or a diverging palette (dark-light-dark) for data with a meaningful midpoint (e.g., negative to positive correlation).

For accessibility: Include a colorblind-friendly option when possible. Test visualizations with tools like Coblis or Color Oracle. When in doubt, supplementary patterns or text labels can ensure clarity without relying on color alone.

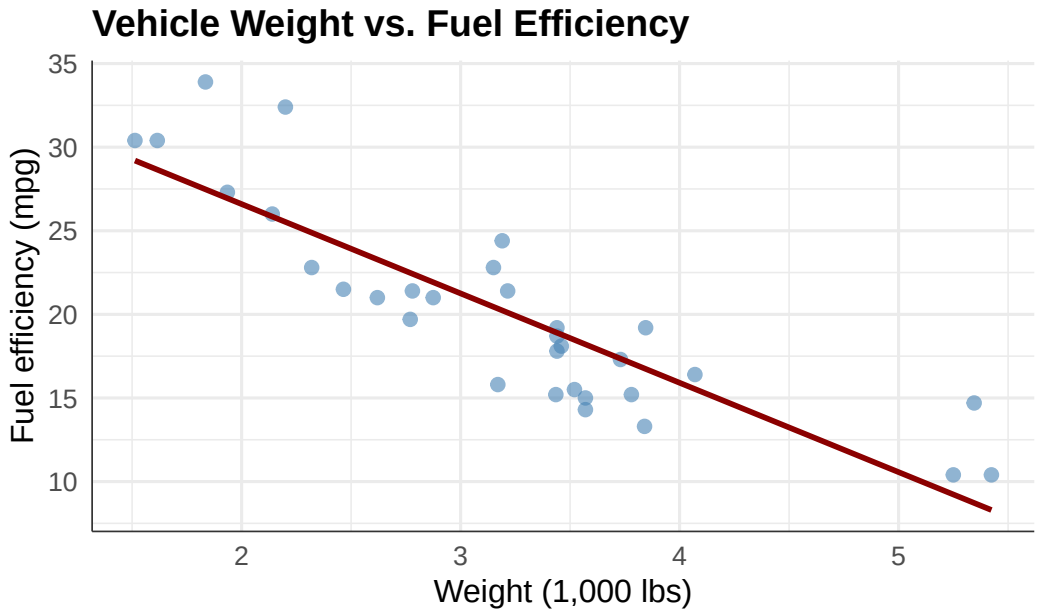
9.3 Part II: Visualizing Bivariate Associations

9.3.1 Numeric × Numeric: Scatterplots and Alternatives

The scatterplot is the default, but several variants serve different purposes:

9.3.1.1 Basic Scatterplot with Trend

```
# Clean, minimal scatterplot
mtcars |>
  ggplot(aes(x = wt, y = mpg)) +
  geom_point(alpha = 0.6, size = 2, color = "steelblue") +
  geom_smooth(method = "lm", se = FALSE, color = "darkred", linewidth = 1) +
  labs(
    title = "Vehicle Weight vs. Fuel Efficiency",
    x = "Weight (1,000 lbs)",
    y = "Fuel efficiency (mpg)",
    caption = "Linear regression line shown in red."
  ) +
  theme_minimal(base_size = 12) +
  theme(
    plot.title = element_text(face = "bold", size = 14),
    axis.line = element_line(color = "gray20", linewidth = 0.3)
  )
```



Linear regression line shown in red.

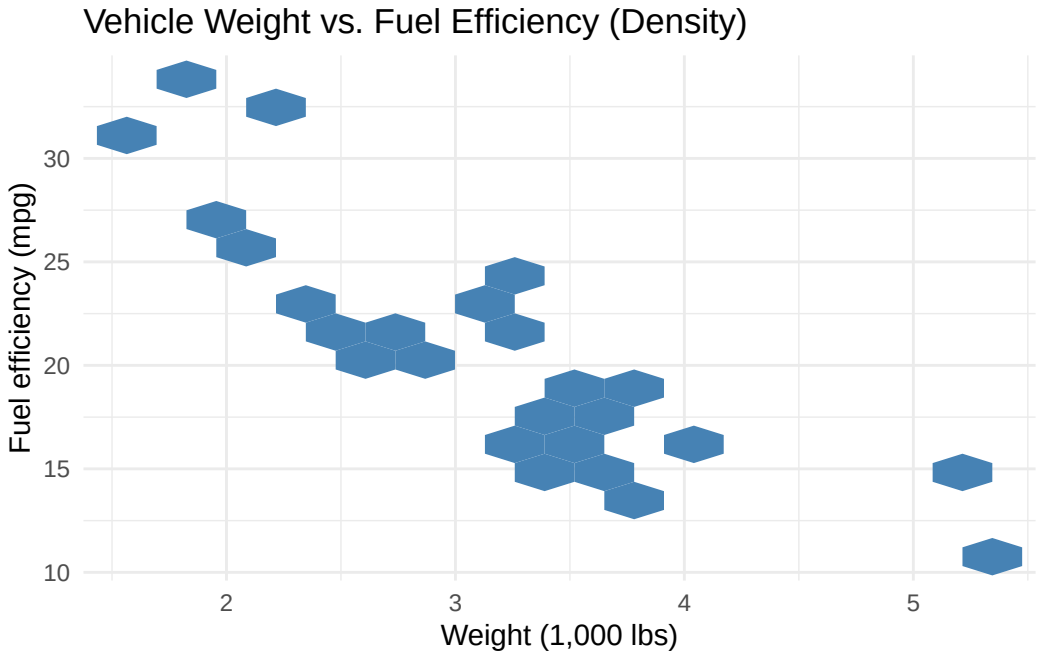
Strengths: Shows individual data points, reveals outliers, allows visual assessment of linearity and scatter

Weaknesses: Overplotting if many points overlap; hard to see underlying density

When to use: Small to moderate datasets (< 1,000 points), when individual points matter or outliers are interesting

9.3.1.2 Hexbin or 2D Density for Overplotting

```
# Heatmap version for many overlapping points
mtcars |>
  ggplot(aes(x = wt, y = mpg)) +
  geom_hex(bins = 15, fill = "steelblue") +
  scale_fill_gradient(low = "white", high = "darkblue", name = "Count") +
  labs(
    title = "Vehicle Weight vs. Fuel Efficiency (Density)",
    x = "Weight (1,000 lbs)",
    y = "Fuel efficiency (mpg)"
  ) +
  theme_minimal()
```



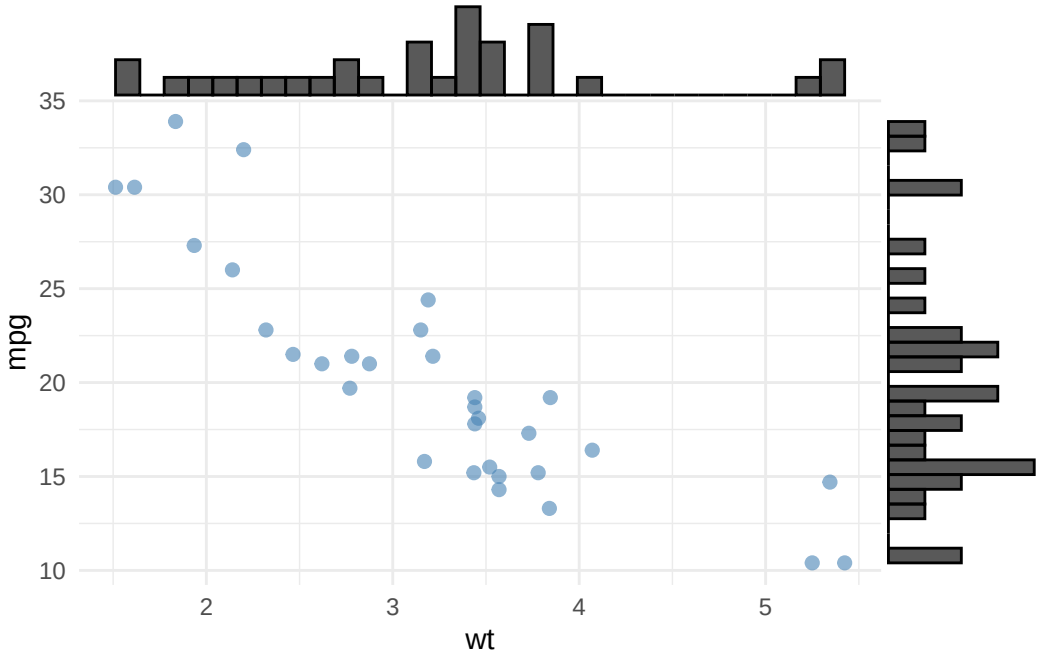
Strengths: Reveals density where points overlap; works for large datasets

Weaknesses: Obscures individual data; harder to spot outliers

When to use: Large datasets ($>$ 5,000 points), when patterns at scale matter more than individuals

9.3.1.3 Marginal Distributions

```
# Scatterplot with marginal histograms (using ggExtra or similar)
base_plot <- mtcars |>
  ggplot(aes(x = wt, y = mpg)) +
  geom_point(alpha = 0.6, size = 2, color = "steelblue") +
  theme_minimal()

# Combine with marginal histograms (pseudo-code)
ggExtra::ggMarginal(base_plot, type = "histogram", margins = "both")
```



Strengths: Shows bivariate and univariate patterns simultaneously; reveals skewness or modality in each variable

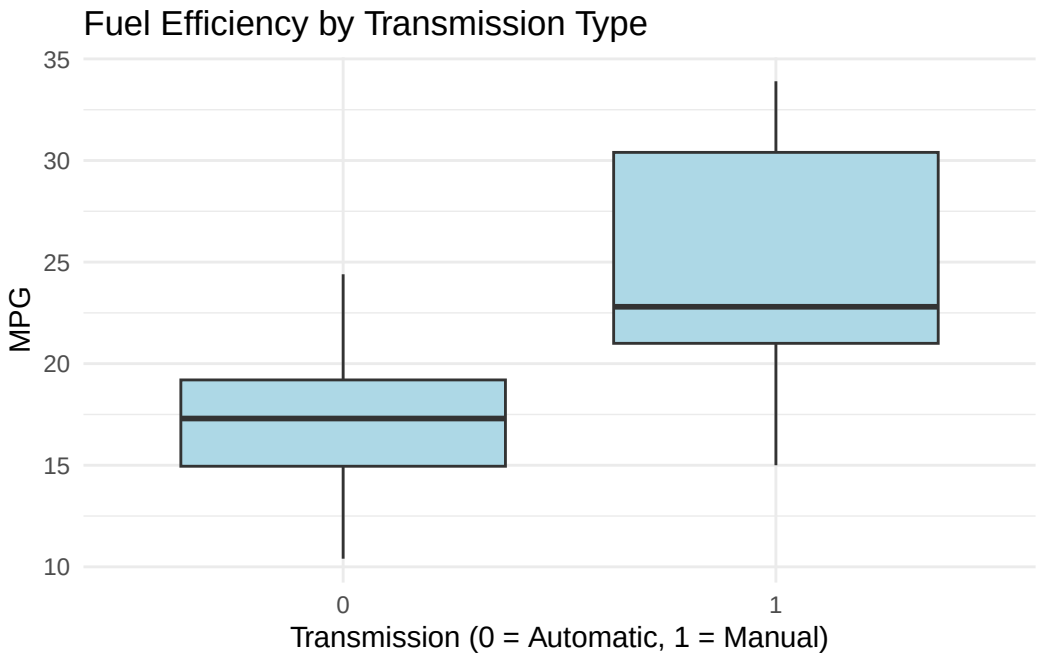
Weaknesses: More complex; requires additional packages

When to use: Academic or detailed exploratory contexts where understanding each variable's shape matters

9.3.2 Numeric × Categorical: Comparing Distributions

9.3.2.1 Boxplots

```
# Boxplot: median, quartiles, outliers
mtcars |>
  ggplot(aes(x = factor(am), y = mpg)) +
  geom_boxplot(fill = "lightblue", color = "gray20") +
  labs(
    title = "Fuel Efficiency by Transmission Type",
    x = "Transmission (0 = Automatic, 1 = Manual)",
    y = "MPG"
  ) +
  theme_minimal()
```

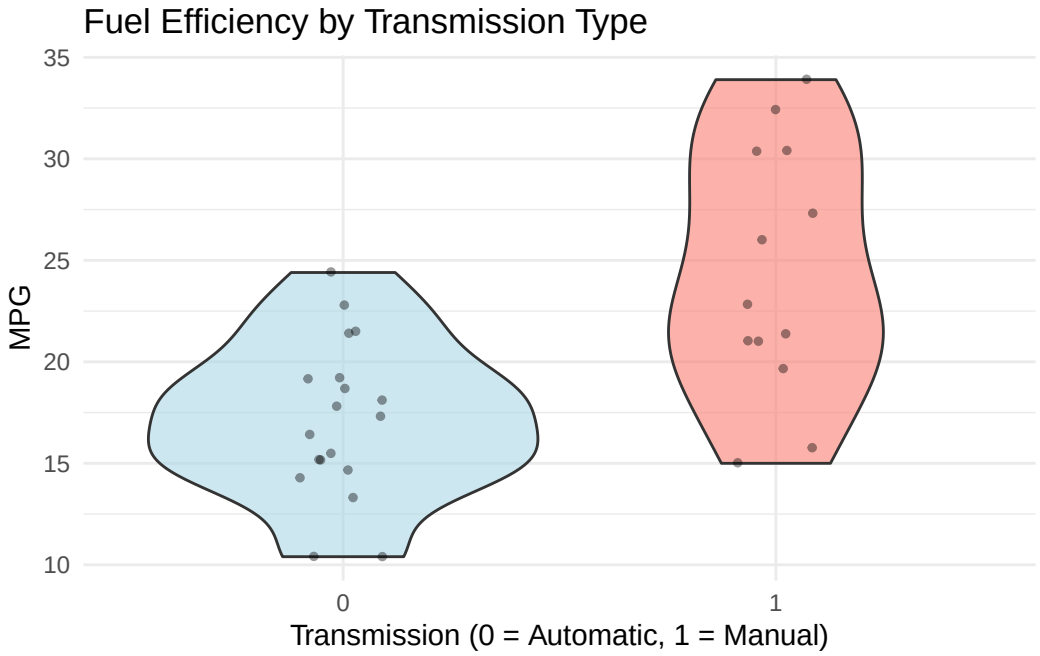


Strengths: Compact, shows quartiles and outliers, familiar to technical audiences

Weaknesses: Hides multimodality; loses individual data points

9.3.2.2 Violin Plots

```
# Violin plot: full density by category
mtcars |>
  ggplot(aes(x = factor(am), y = mpg, fill = factor(am))) +
  geom_violin(alpha = 0.6) +
  geom_jitter(width = 0.1, alpha = 0.4, size = 1) +
  scale_fill_manual(values = c("lightblue", "salmon"), guide = "none") +
  labs(
    title = "Fuel Efficiency by Transmission Type",
    x = "Transmission (0 = Automatic, 1 = Manual)",
    y = "MPG"
  ) +
  theme_minimal()
```



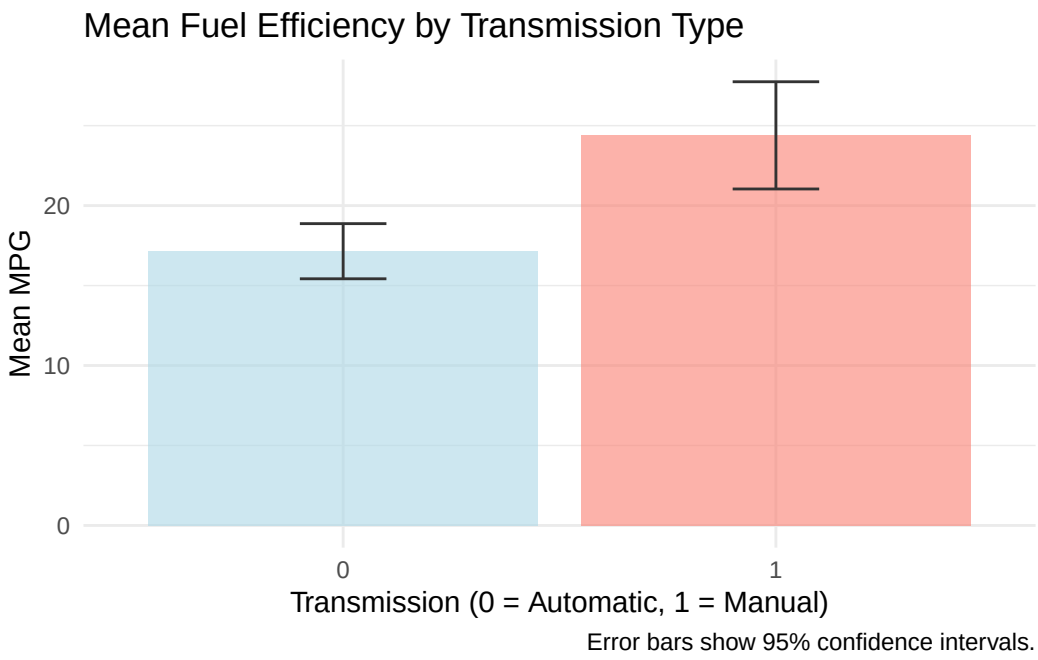
Strengths: Shows full distribution, reveals bimodality, includes jittered points

Weaknesses: Larger visual footprint; less familiar to non-technical audiences

9.3.2.3 Mean Plots with Error Bars

```
# Mean with error bars (SE or CI)
mtcars |>
  group_by(am) |>
  summarise(
    mean_mpg = mean(mpg),
    se = sd(mpg) / sqrt(n()),
    .groups = "drop"
  ) |>
  ggplot(aes(x = factor(am), y = mean_mpg, fill = factor(am))) +
  geom_col(alpha = 0.6) +
  geom_errorbar(aes(ymin = mean_mpg - 1.96 * se, ymax = mean_mpg + 1.96 * se),
                width = 0.2, color = "gray20") +
  scale_fill_manual(values = c("lightblue", "salmon"), guide = "none") +
  labs(
    title = "Mean Fuel Efficiency by Transmission Type",
    x = "Transmission (0 = Automatic, 1 = Manual)",
    y = "Mean MPG",
```

```
    caption = "Error bars show 95% confidence intervals."
  ) +
  theme_minimal()
```



Strengths: Clear, actionable, easy to compare magnitudes, includes uncertainty quantification

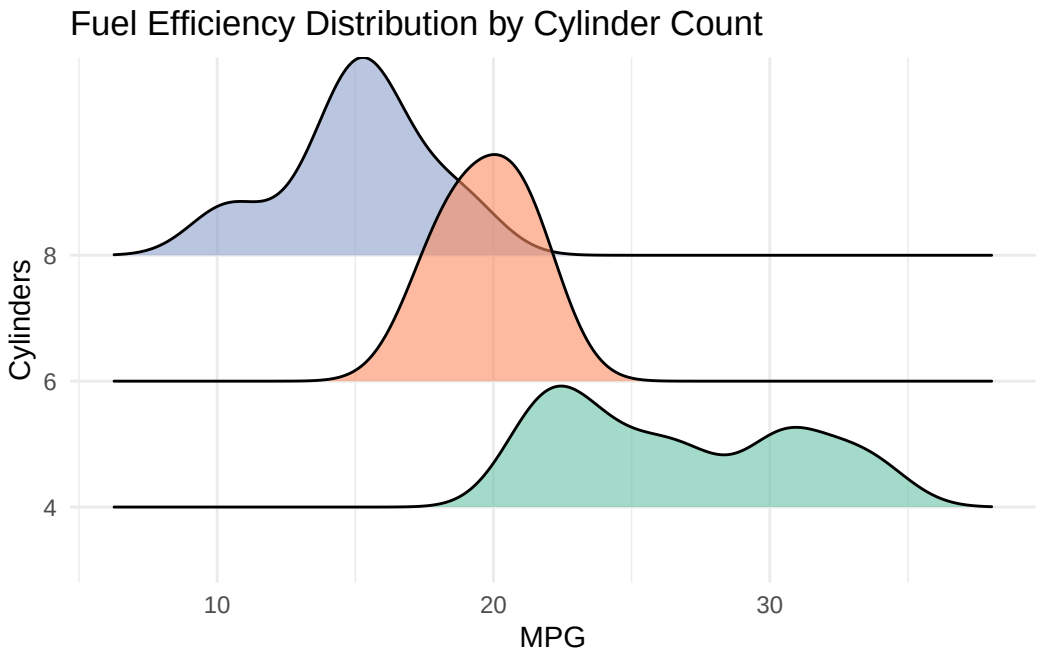
Weaknesses: Oversimplifies distribution; hides individual variation and outliers

When to use: When comparing group averages is the primary goal; good for presentations

9.3.2.4 Ridgeline Plots

```
# Ridgeline: density for each category
mtcars |>
  ggplot(aes(x = mpg, y = factor(cyl), fill = factor(cyl))) +
  geom_density_ridges(alpha = 0.6) +
  scale_fill_brewer(palette = "Set2", guide = "none") +
  labs(
    title = "Fuel Efficiency Distribution by Cylinder Count",
    x = "MPG",
    y = "Cylinders"
```

```
  ) +
  theme_minimal()
```



Strengths: Elegant, shows multiple distributions, reveals shape differences

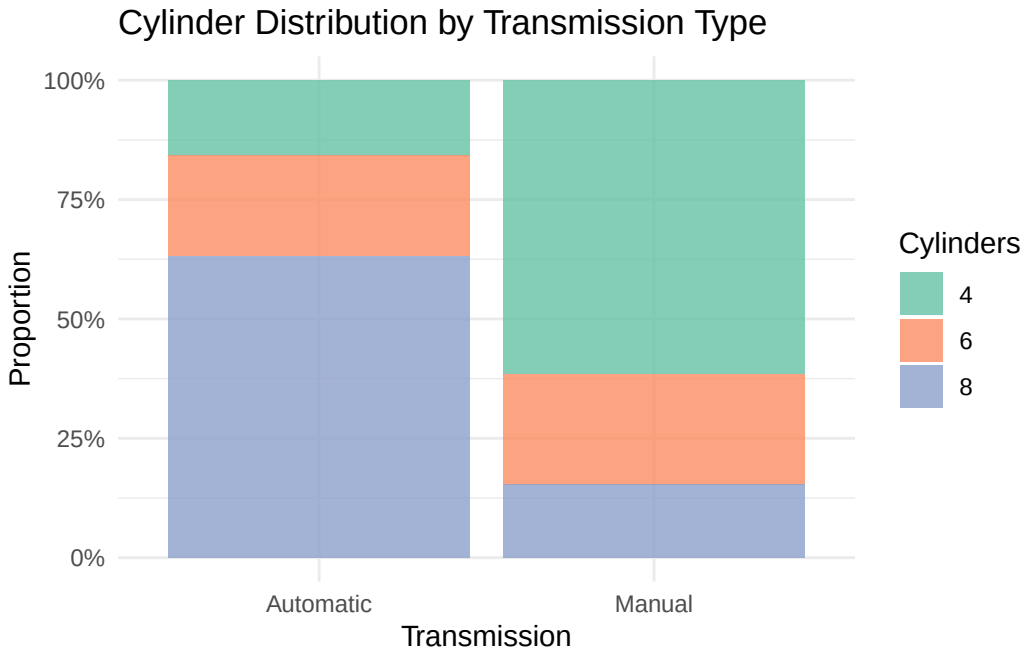
Weaknesses: Requires familiarity with density plots; may seem decorative

9.3.3 Categorical × Categorical: Contingency Visualization

9.3.3.1 Stacked/Grouped Bar Charts

```
# Stacked bar chart: proportions by category
mtcars |>
  mutate(
    am = factor(am, labels = c("Automatic", "Manual")),
    cyl = factor(cyl)
  ) |>
  group_by(am, cyl) |>
  count() |>
  group_by(am) |>
  mutate(prop = n / sum(n)) |>
  ggplot(aes(x = am, y = prop, fill = cyl)) +
```

```
  geom_col(alpha = 0.8) +
  scale_y_continuous(labels = scales::percent) +
  scale_fill_brewer(palette = "Set2", name = "Cylinders") +
  labs(
    title = "Cylinder Distribution by Transmission Type",
    x = "Transmission",
    y = "Proportion"
  ) +
  theme_minimal()
```

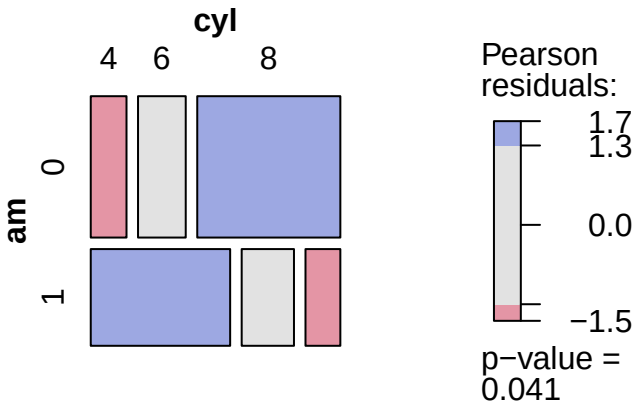


Strengths: Shows proportions, easy to compare total height across groups

Weaknesses: Hard to compare non-baseline categories (only first category starts at zero)

9.3.3.2 Mosaic Plots (Hartigan and Kleiner 1981)

```
# Mosaic plot: cell area = frequency
vcd::mosaic(~ am + cyl, data = mtcars, shade = TRUE, gp = vcd::shading_max)
```

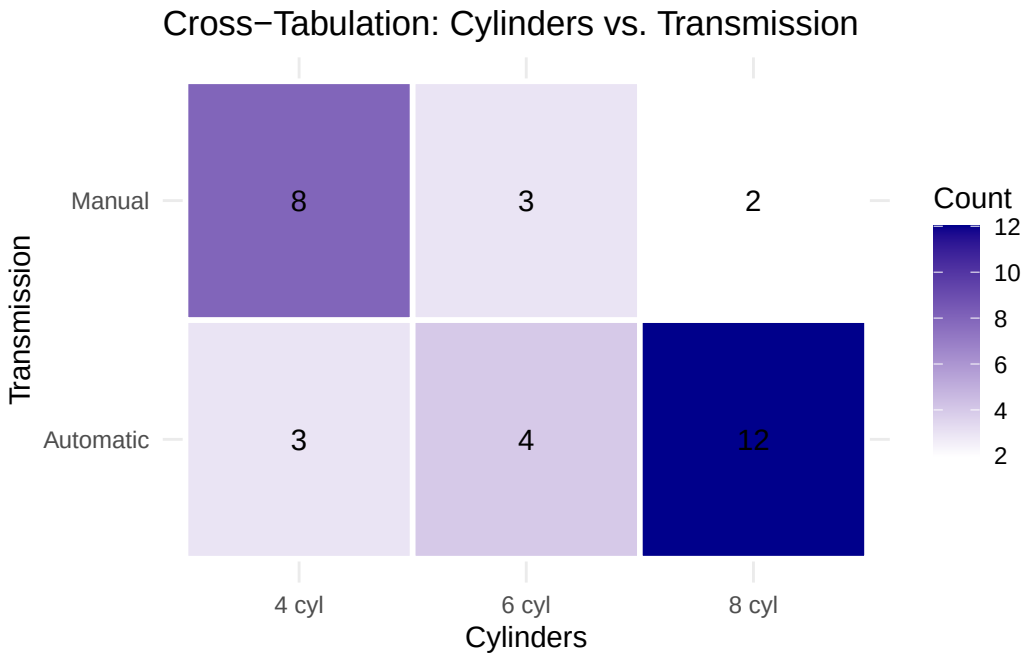


Strengths: Cell area directly represents frequency; includes statistical shading for association

Weaknesses: Requires explanation; less intuitive for general audiences

9.3.3.3 Heatmap with Color Intensity

```
# Heatmap: cell color represents count or association strength
mtcars |>
  mutate(
    am = factor(am, labels = c("Automatic", "Manual")),
    cyl = factor(cyl, labels = c("4 cyl", "6 cyl", "8 cyl"))
  ) |>
  group_by(am, cyl) |>
  count() |>
  ggplot(aes(x = cyl, y = am, fill = n)) +
  geom_tile(color = "white", linewidth = 1) +
  geom_text(aes(label = n), color = "black", size = 4) +
  scale_fill_gradient(low = "white", high = "darkblue", name = "Count") +
  labs(
    title = "Cross-Tabulation: Cylinders vs. Transmission",
    x = "Cylinders",
    y = "Transmission"
  ) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 0))
```



Strengths: Clean, numbers visible, color intensity and numbers reinforce each other

Weaknesses: Not suitable for very large tables; limited to 2D tables

9.4 Part III: Visualizing Multivariate Patterns

9.4.1 Association Networks Revisited

In Chapter 6, we introduced network visualizations for associations. Here, we focus on **communication** rather than exploration.

9.4.1.1 Network Refinement for Audiences

For academic audiences, networks may include:

- Weighted edges (thickness = association strength)
- Color-coded edge types (Pearson, Eta, Cramér's V)
- Node labels with abbreviations defined in caption
- Layout algorithm noted in caption

For professional audiences:

- Simplified networks (only strong associations, threshold > 0.7)

- Clear node grouping (e.g., demographics vs. attitudes)
- Minimal text labels; detailed legend in appendix
- Color-coded clusters showing interpretable groupings

For public audiences:

- Extremely sparse networks (top 10–15 associations only)
- Nodes labeled in plain English (not variable codes)
- Edge direction shown intuitively (positive = solid, negative = dashed)
- Accompanying narrative: "Education and life satisfaction are strongly linked…"

9.4.1.2 Layout and Readability

Network layouts (force-directed, hierarchical, circular) have dramatic effects on readability:

- **Force-directed layouts** (Fruchterman-Reingold, ForceAtlas2): Reveal natural clusters; good for exploration and exploratory communication
- **Hierarchical layouts**: Good for directed networks showing cause → effect chains
- **Circular layouts**: Good for symmetric networks with balanced structure

For communication, consider:

1. **Manual tuning**: Adjust node positions to minimize edge crossings
2. **Consistent positioning**: If comparing two networks, use identical layouts for easy comparison
3. **Clear focal points:** Highlight key nodes (e.g., surprising connectors, key predictors)

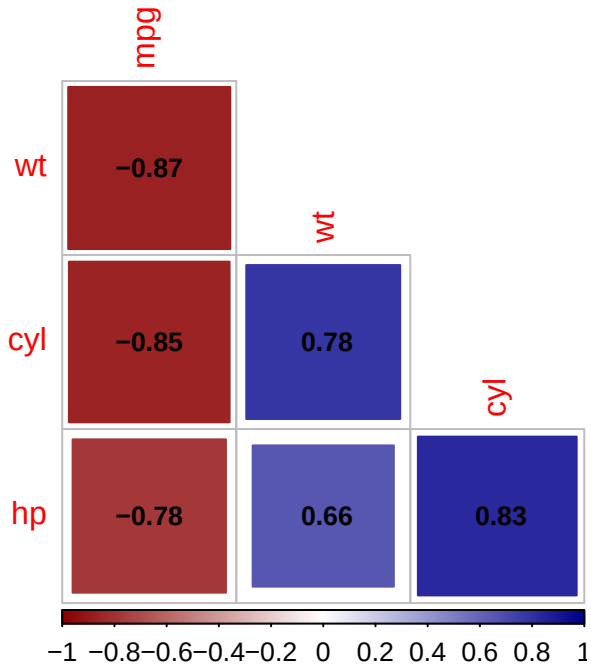
9.4.2 Correlation Heatmaps

Heatmaps remain popular for showing all pairwise associations simultaneously. But they can mislead if poorly designed.

9.4.2.1 Well-Designed Heatmap

```
# Heatmap with careful ordering and color scale
cor_matrix <- cor(mtcars[, c("mpg", "cyl", "hp", "wt")])

# Reorder by hierarchical clustering for interpretability
library(corrplot)
corrplot(cor_matrix,
         method = "square",
```

```
        type = "lower",
        order = "hclust",
        col = colorRampPalette(c("darkred", "white", "darkblue"))(200),
        diag = FALSE,
        addCoef.col = "black",
        number.cex = 0.8)
```



Design choices explained:

- **Order = "hclust"**: Reorders rows/columns by similarity, revealing clusters
- **Type = "lower"**: Shows only lower triangle to reduce redundancy
- **Diverging color**: Dark red (negative), white (zero), dark blue (positive)
- **Coefficients shown**: Exact values added to cells (optional but helpful)

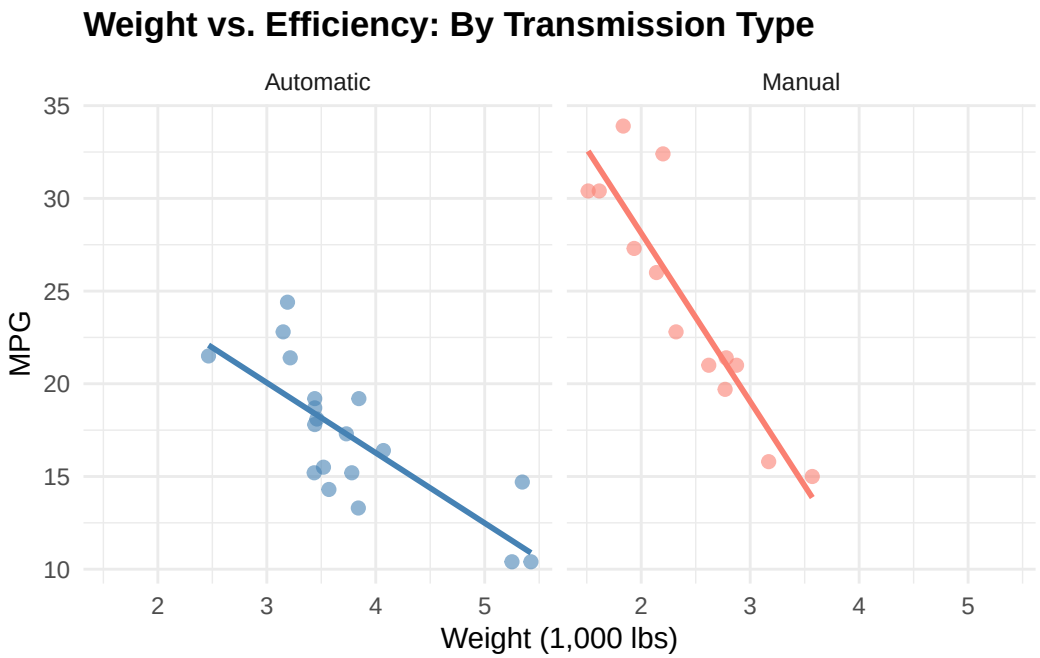
Common mistakes to avoid:

- Using rainbow palettes (perceptually non-uniform)
- Showing the full symmetric matrix (redundant)
- Alphabetical ordering (hides structure)
- Very large matrices without annotation (unreadable)

9.4.3 Faceted Plots

When comparing the same relationship across groups, **faceted plots** are superior to overlaid versions:

```
# Instead of overlaying, facet by group
mtcars |>
  ggplot(aes(x = wt, y = mpg, color = factor(am))) +
  geom_point(alpha = 0.6, size = 2) +
  geom_smooth(method = "lm", se = FALSE, linewidth = 1) +
  facet_wrap(~am, labeller = labeller(am = c("0" = "Automatic", "1" = "Manual"))) +
  scale_color_manual(values = c("steelblue", "salmon"), guide = "none") +
  labs(
    title = "Weight vs. Efficiency: By Transmission Type",
    x = "Weight (1,000 lbs)",
    y = "MPG"
  ) +
  theme_minimal() +
  theme(plot.title = element_text(face = "bold"))
```



Advantages:

- Each group gets its own visual space (no overlap)
- Differences in trend, scatter, or outliers become obvious
- Easier to perceive separate patterns than overlaid colors

9.5 Part IV: Narrative and Context

Visualization alone rarely tells a complete story. The best communicators pair images with narrative.

9.5.1 Titles, Captions, and Annotations

Title: Can answer "What?" or "So what?"

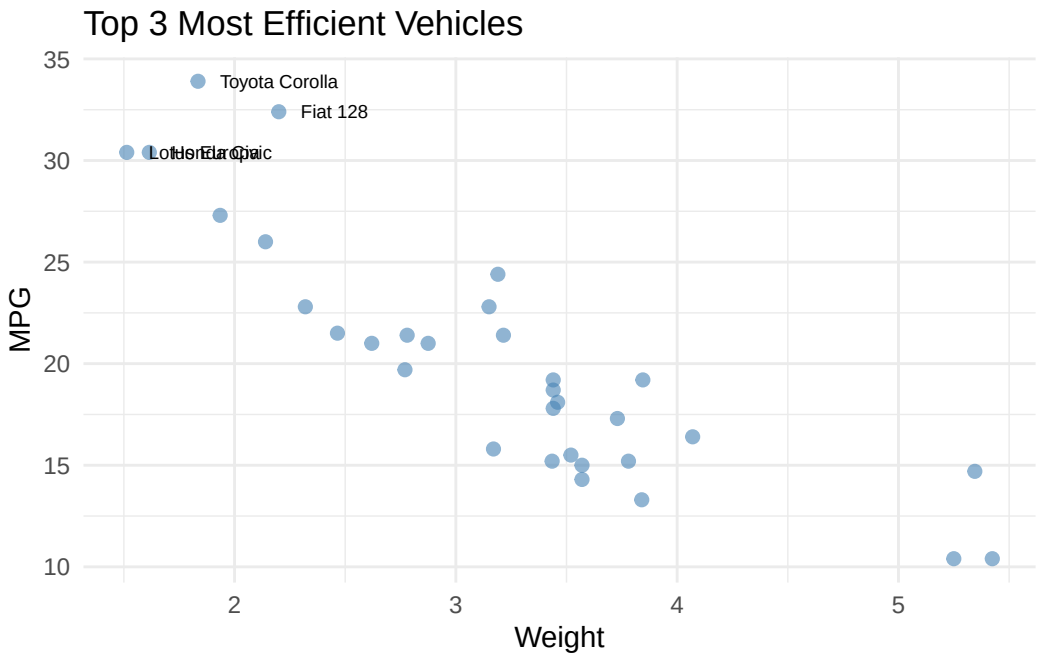
- **Weak**: "Vehicle Characteristics"
- **Strong**: "Heavier vehicles consume more fuel"
- **Strategic**: "The Weight-Efficiency Trade-Off: Why Lighter Cars Win"

Caption/Annotation: Can explain methodology and caveats

- "Data: 32 vehicles from 1974. Linear regression line shown in red. Two outliers visible above the trend."

Direct labeling: Rather than force readers to match colors to legends, label key points directly on the plot

```
# Direct labeling example
mtcars_labeled <- mtcars |>
  rownames_to_column(var = "car")

mtcars_labeled |>
  ggplot(aes(x = wt, y = mpg)) +
  geom_point(color = "steelblue", alpha = 0.6, size = 2) +
  geom_text(aes(label = car), size = 2.5, hjust = 0, nudge_x = 0.1,
            data = mtcars_labeled |> slice_max(n = 3, order_by = mpg)) +
  labs(
    title = "Top 3 Most Efficient Vehicles",
    x = "Weight", y = "MPG"
  ) +
  theme_minimal()
```



9.5.2 Context and Audience Adaptation

The same finding can be presented three ways:

Academic framing: "Among 32 vehicles, a strong negative correlation ($r = -0.87$, 95% CI $[-0.93, -0.74]$) exists between weight and fuel efficiency, suggesting that vehicle mass is a significant determinant of energy consumption."

Professional framing: "Lighter vehicles consistently achieve better fuel economy. For every 1,000-pound increase in weight, we observe approximately 5.4 fewer miles per gallon."

Public framing: "Heavy SUVs and trucks guzzle gas. Compact cars and hybrids go further on less fuel. It's physics."

Each version is truthful but optimized for its audience.

9.6 Part V: Common Pitfalls and How to Avoid Them

9.6.1 Overplotting and Obscuration

Problem: Too many overlapping points; underlying pattern invisible

Solution: Use transparency (alpha), jitter slightly, or switch to density plot or hexbin

9.6.2 Dual-Axis Plots

Problem: Two different scales on same plot can manipulate perception (axis scaling changes appear misleading)

Solution: Facet instead; if both axes are necessary, use same scale or clearly label both

9.6.3 3D Visualizations

Problem: 3D plots distort distances and make comparison harder, not easier

Solution: Use faceted 2D plots or interactive 3D plots (but not static 3D printed plots)

9.6.4 Truncated Axes

Problem: Starting y-axis at non-zero value exaggerates differences

Solution: Always start at zero for bar charts; for scatterplots, include enough padding to show context

9.6.5 Chart Junk and Decoration

Problem: Decorative elements (3D effects, gradient fills, gridlines, background images) distract from data

Solution: Minimize. Use white or light gray backgrounds, remove unnecessary gridlines, use simple shapes

9.6.6 Inconsistent Scales Across Panels

Problem: Different scales in faceted plots make comparison impossible

Solution: Always use `scales = "fixed"` in faceted plots unless scale variation is the story

9.7 Part VI: Accessibility and Inclusive Design

Thoughtful visualization serves diverse audiences and abilities.

9.7.1 Colorblind-Friendly Design

Approximately 8% of men and 0.5% of women have color blindness. Designs are best when they avoid relying solely on color distinctions:

- Use colorblind-friendly palettes (Viridis, ColorBrewer's "colorblind-safe" option)
- Supplement color with pattern or text (e.g., labels on nodes)
- Test visualizations with colorblind simulation tools

9.7.2 Visual Clarity for Low-Vision Readers

- Large fonts (minimum 12pt for body text, 14pt+ for titles)
- High contrast (dark text on light background or vice versa)
- Non-default line widths (thicker lines for visibility)
- Avoid light gray text

9.7.3 Interpretability for Non-Technical Readers

- Avoid jargon; define technical terms in caption
- Include units (not just "mpg" but "miles per gallon")
- Provide a "take-home" sentence above or below the plot

9.8 Summary and Key Takeaways

- **Design for discovery**: Visualizations can aim to reveal patterns, not obscure them through decoration or complexity
- **Match form to audience**: Academic precision, professional clarity, public simplicity—three audiences, three visualizations
- **Encode wisely**: Use visual channels (position, size, color) strategically; encode most important information first
- **Reduce cognitive load**: Declutter, group, highlight, annotate directly
- **Pair visualization with narrative**: A chart without explanation leaves viewers confused; text without visualization misses emotional impact
- **Test accessibility**: Color, contrast, and clarity matter for inclusivity
- **Export thoughtfully**: Move from interactive exploration to static communication; recreate visualizations with intentional design
- **Context is king**: The same data can support multiple true stories, depending on framing and audience

9.9 Looking Ahead

With the foundations of association analysis (Chapters 3–6), interactive tools (Chapters 7–8), and communication strategies (this chapter) established, we now move beyond pairwise relationships to **higher-order patterns and predictive modeling**.

The chapters that follow introduce methods for:

- **Regression trees** (Chapter 10): Non-parametric discovery of feature interactions and segmentation
- **Classification trees** (Chapter 11): Predicting discrete outcomes while maintaining interpretability
- **Ensemble methods** (Chapter 12): Combining multiple models for improved predictions
- **Interpretable machine learning** (Chapters 13–16): Feature importance, partial dependence, Shapley values—techniques for understanding complex models

These methods shift perspective: instead of asking "What associations exist?", we ask "Can we predict or segment based on these associations?" But interpretability remains paramount. We aim to avoid black-box models, instead focusing on methods that stay true to the descriptive, exploratory spirit of this book.

Part IV

Tree-Based Methods for Description

10 Regression Trees for Exploratory Segmentation

10.1 Introduction: From Associations to Segments

The previous chapters emphasized associations and their communication. A natural next step is to ask a different kind of question. Instead of summarizing global relationships, we may want to **partition a dataset into interpretable subgroups** and describe how outcomes differ across those groups. Regression trees provide a principled and transparent way to do so.

Regression trees are not only predictive tools. In descriptive analysis, their value comes from **segmentation**. They reveal how combinations of predictors define subpopulations with different outcome levels. This chapter focuses on that descriptive role, with careful attention to interpretation, stability, and communication.

10.2 The Regression Tree Idea

Regression trees approximate a complex relationship by **recursively splitting** the predictor space. Each split divides observations into two groups, and each terminal node, also called a leaf, predicts the outcome by the average within that leaf. The goal is to find splits that reduce within group variation.

At each step, the algorithm searches for a split that minimizes the **residual sum of squares** in the resulting partitions. For a single split, the criterion is (Breiman et al. 2017):

$$RSS = \sum_{i \in L} (y_i - \bar{y}_L)^2 + \sum_{i \in R} (y_i - \bar{y}_R)^2,$$

where L and R are the left and right child nodes. This local optimization is repeated until a stopping rule is met.

Regression trees therefore trade smoothness for interpretability. They yield a piecewise constant approximation with clear decision rules, which is especially useful when we want to communicate **who belongs to which segment**.

10.3 A Reproducible Example with Housing Prices

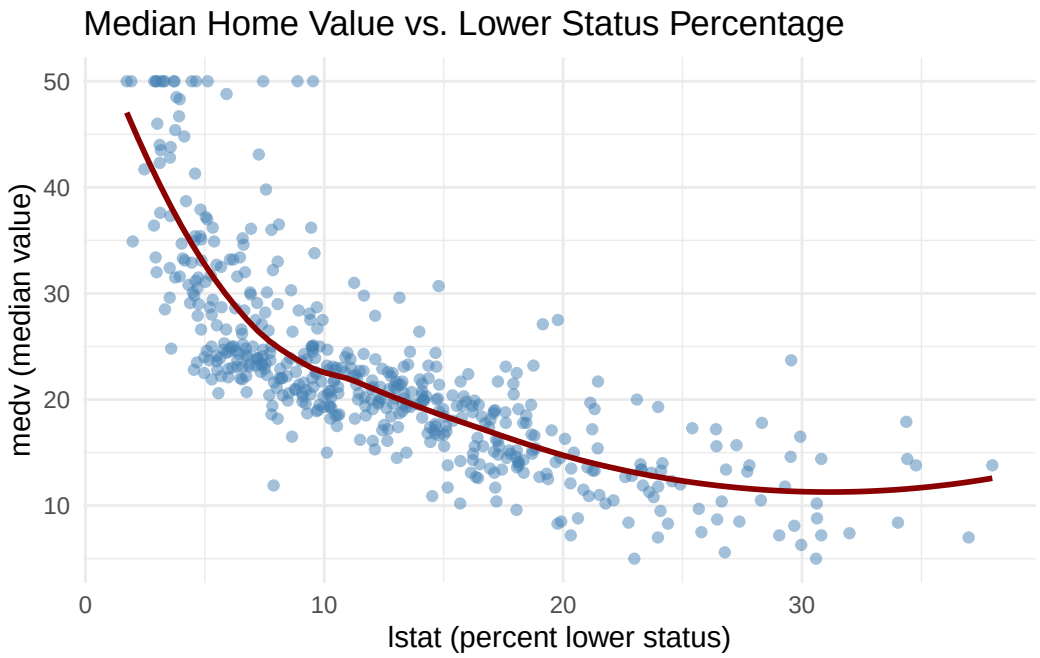
We use the Boston housing dataset from the MASS package. The outcome is the median home value, `medv`, and we focus on a subset of predictors to keep the narrative compact.

```
data(Boston, package = "MASS")
glimpse(Boston)
```

```
Rows: 506
Columns: 14
$ crim    <dbl> 0.00632, 0.02731, 0.02729, 0.03237, 0.06905, 0.02985, 0.08829,~
$ zn      <dbl> 18.0, 0.0, 0.0, 0.0, 0.0, 0.0, 12.5, 12.5, 12.5, 12.5, 12.5, 1~
$ indus   <dbl> 2.31, 7.07, 7.07, 2.18, 2.18, 2.18, 7.87, 7.87, 7.87, 7.87, 7.~
$ chas    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
$ nox     <dbl> 0.538, 0.469, 0.469, 0.458, 0.458, 0.458, 0.524, 0.524, 0.524,~
$ rm      <dbl> 6.575, 6.421, 7.185, 6.998, 7.147, 6.430, 6.012, 6.172, 5.631,~
$ age     <dbl> 65.2, 78.9, 61.1, 45.8, 54.2, 58.7, 66.6, 96.1, 100.0, 85.9, 9~
$ dis     <dbl> 4.0900, 4.9671, 4.9671, 6.0622, 6.0622, 6.0622, 5.5605, 5.9505~
$ rad     <int> 1, 2, 2, 3, 3, 3, 5, 5, 5, 5, 5, 5, 5, 4, 4, 4, 4, 4, 4, 4, 4,~
$ tax     <dbl> 296, 242, 242, 222, 222, 222, 311, 311, 311, 311, 311, 311, 31~
$ ptratio <dbl> 15.3, 17.8, 17.8, 18.7, 18.7, 18.7, 15.2, 15.2, 15.2, 15.2, 15~
$ black   <dbl> 396.90, 396.90, 392.83, 394.63, 396.90, 394.12, 395.60, 396.90~
$ lstat   <dbl> 4.98, 9.14, 4.03, 2.94, 5.33, 5.21, 12.43, 19.15, 29.93, 17.10~
$ medv    <dbl> 24.0, 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 15~
```

Before fitting a tree, we inspect a few relationships. This step is not strictly required, but it helps us connect the tree splits to familiar bivariate patterns.

```
Boston |>
    ggplot(aes(x = lstat, y = medv)) +
    geom_point(alpha = 0.5, color = "steelblue") +
    geom_smooth(method = "loess", se = FALSE, color = "darkred", linewidth = 1) +
    labs(
        title = "Median Home Value vs. Lower Status Percentage",
        x = "lstat (percent lower status)",
        y = "medv (median value)"
    ) +
    theme_minimal()
```



The nonlinearity in this relationship suggests that a tree, which allows different slopes in different regions, may provide an interpretable segmentation.

10.4 Fitting a Regression Tree

We fit a regression tree using `rpart`, which is designed for recursive partitioning. We set a modest minimum node size to avoid very small leaves and a small complexity parameter to grow a reasonably large initial tree.

```
set.seed(123)
tree_fit <- rpart(
    medv ~ lstat + rm + ptratio + indus + nox + crim,
    data = Boston,
    method = "anova",
    control = rpart.control(minsplit = 30, cp = 0.001)
)
```

The console printout is informative but not always easy to read in a narrative document. Instead, we focus on clearer outputs in the next sections, including pruning diagnostics, a modern tree visualization, and a compact segment summary.

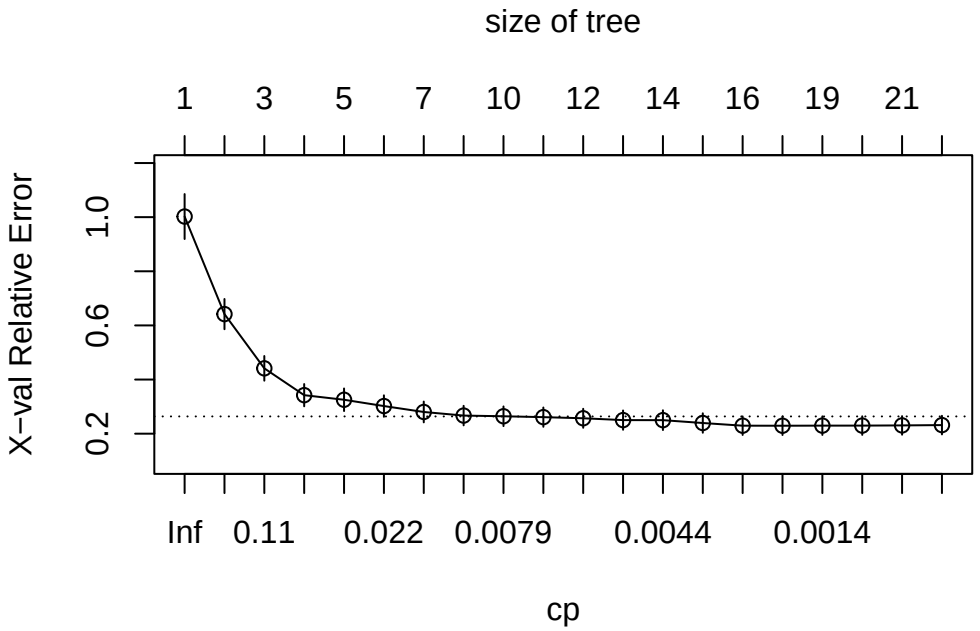
10.5 Controlling Complexity and Pruning

Large trees can be unstable and difficult to interpret. The `rpart` object includes cross-validated error estimates for a sequence of subtrees. We can inspect and select a complexity parameter that balances fit and interpretability.

```
printcp(tree_fit)
```

```
Regression tree:
rpart(formula = medv ~ lstat + rm + ptratio + indus + nox + crim,
    data = Boston, method = "anova", control = rpart.control(minsplit = 30,
        cp = 0.001))

Variables actually used in tree construction:
[1] crim    indus   lstat   nox     ptratio rm

Root node error: 42716/506 = 84.42

n= 506

          CP nsplit rel error  xerror     xstd
1  0.4527442      0   1.00000 1.00232 0.083091
2  0.1711724      1   0.54726 0.64177 0.055837
3  0.0716578      2   0.37608 0.44136 0.045649
4  0.0342882      3   0.30443 0.34229 0.041186
5  0.0266130      4   0.27014 0.32531 0.041755
6  0.0180237      5   0.24352 0.30207 0.039685
7  0.0105823      6   0.22550 0.28023 0.038437
8  0.0086921      7   0.21492 0.26684 0.036096
9  0.0072654      9   0.19753 0.26433 0.036373
10 0.0066497     10   0.19027 0.26103 0.035667
11 0.0061263     11   0.18362 0.25688 0.035097
12 0.0048053     12   0.17749 0.25013 0.035717
13 0.0039410     13   0.17269 0.24994 0.036574
14 0.0035366     14   0.16875 0.23937 0.035966
15 0.0022354     15   0.16521 0.22957 0.034303
16 0.0014183     16   0.16297 0.22935 0.034341
17 0.0013243     18   0.16014 0.22971 0.034359
18 0.0013053     19   0.15881 0.22986 0.034360
19 0.0011373     20   0.15751 0.23048 0.034359
20 0.0010000     21   0.15637 0.23170 0.034370
```

```r
plotcp(tree_fit)
```



We select the complexity parameter that minimizes cross-validated error and prune the tree accordingly.

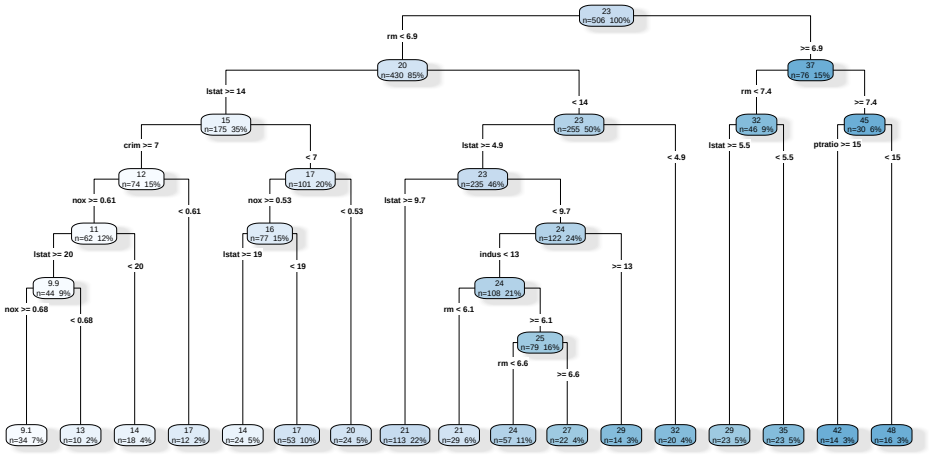
```r
best_cp <- tree_fit$cptable[which.min(tree_fit$cptable[, "xerror"]), "CP"]
pruned_tree <- prune(tree_fit, cp = best_cp)

best_cp
```

```
[1] 0.001418281
```

A compact visualization helps us communicate the segmentation. We use a clean tree plot that balances readability and detail without overwhelming the page.

```r
rpart.plot(
    pruned_tree,
    type = 4,
    extra = 101,
    fallen.leaves = TRUE,
    box.palette = "Blues",
    branch.lty = 1,
    shadow.col = "gray90",
```

```
    main = "Regression Tree for Housing Prices"
)
```



The resulting tree provides a small number of segments with clear rules. It is often helpful to summarize the leaf characteristics numerically, which turns the tree into a compact descriptive table.

```
boston_nodes <- Boston |>
    mutate(node = factor(pruned_tree$where))

segment_summary <- boston_nodes |>
    group_by(node) |>
    summarise(
        n = n(),
        medv_mean = mean(medv),
        medv_sd = sd(medv),
        lstat_mean = mean(lstat),
        rm_mean = mean(rm),
        .groups = "drop"
    ) |>
    arrange(desc(medv_mean))

segment_summary |>
    gt() |>
    cols_label(
        n = "N",
```

node	N	Mean medv	SD medv	Mean lstat	Mean rm
33	16	47.98	2.88	3.89	7.92
32	14	41.81	7.29	4.33	8.07
30	23	35.25	4.25	4.33	7.11
26	20	31.57	7.05	4.15	6.60
25	14	29.07	9.07	7.62	6.16
29	23	28.98	6.91	9.01	7.17
24	22	27.17	2.61	7.17	6.74
23	57	24.09	2.90	7.28	6.39
18	113	20.77	2.56	12.03	6.00
21	29	20.62	2.24	8.23	5.97
15	24	20.02	3.07	19.21	5.82
14	53	17.23	2.46	16.61	6.02
10	12	16.63	4.51	20.25	5.88
13	24	14.04	2.86	24.68	5.65
9	18	13.92	2.10	17.05	6.27
8	10	12.63	1.31	25.53	5.72
7	34	9.11	2.21	25.69	5.73

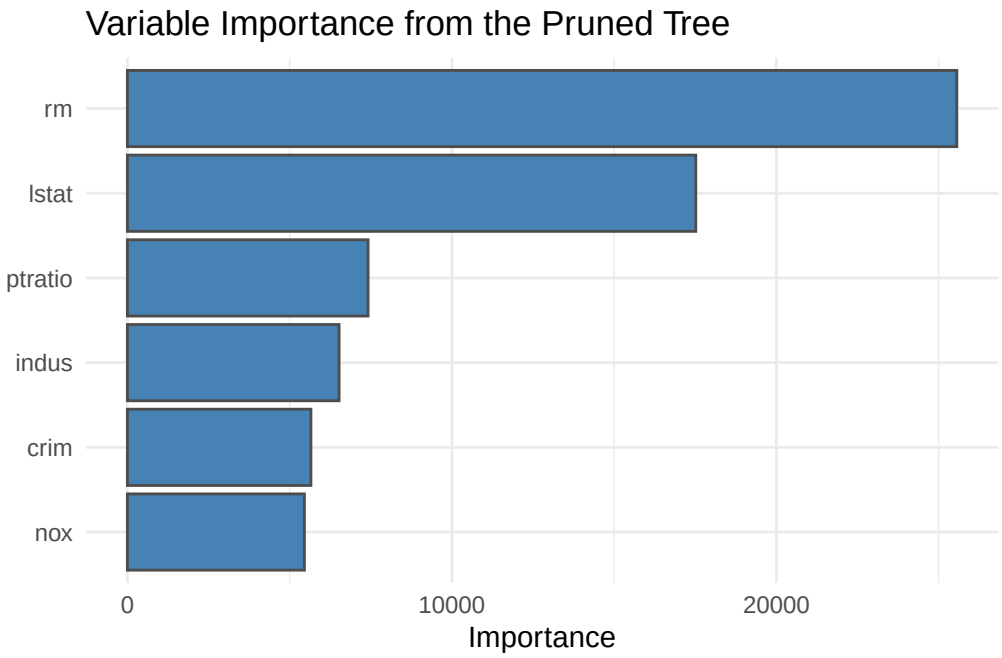
```
        medv_mean = "Mean medv",
        medv_sd = "SD medv",
        lstat_mean = "Mean lstat",
        rm_mean = "Mean rm"
    ) |>
    fmt_number(columns = c(medv_mean, medv_sd, lstat_mean, rm_mean), decimals = 2)
```

This table highlights how the segmentation aligns with substantive variables. For example, leaves with low `lstat` and higher `rm` tend to show higher average prices. The standard deviations remind us that even within a leaf, variation remains, so a tree is a simplification rather than a perfect partition.

10.6 Variable Importance as a Descriptive Lens

A tree already encodes which variables matter, but a variable importance plot can provide a concise summary of the relative contribution of predictors. This view complements the segmentation table by highlighting which variables the tree relied on most.

```r
vip(pruned_tree, num_features = 6) +
    ggtitle("Variable Importance from the Pruned Tree") +
    geom_bar(stat = "identity", color = "gray30", fill = "steelblue") +
    theme_minimal()
```



In descriptive work, variable importance is best treated as a ranking rather than a precise scale. It helps us focus attention, while the split rules and segment summaries provide the substantive interpretation.

10.7 Interactive Tuning in Practice

Small changes in pruning parameters, such as `cp`, `maxdepth`, or the minimum number of observations per leaf, can noticeably change the tree. Interactive tools can make this trade-off tangible by allowing rapid adjustments and immediate feedback on the resulting segmentation. Even without a full application, we can adopt the same mindset in static analysis, choosing parameter values that balance interpretability and stability rather than maximizing fit alone.

10.8 Interpretation as Descriptive Modeling

Regression trees are often treated as predictive models, but in descriptive analysis we can read them as **structured summaries**:

- Each split highlights a predictor that separates higher and lower outcome regions.
- The order of splits suggests the relative importance of predictors in the segmentation.
- Each leaf corresponds to a segment with a clear rule and a mean outcome.

This perspective is valuable when we need to explain findings to collaborators or stakeholders. Instead of presenting a regression table with coefficients, we can present a short set of decision rules and segment means. The interpretation is more discrete, but also more transparent.

10.9 Strengths and Limitations

Strengths

- Interpretability, since each segment is defined by explicit rules.
- Flexibility for nonlinear relationships and interactions without manual specification.
- Natural alignment with exploratory goals, especially for segmentation and profiling.

Limitations

- Instability, small changes in data can yield different trees.
- Loss of smoothness, the model is piecewise constant.
- Potential bias toward variables with many split points, which can influence variable importance.

In descriptive workflows, these limitations suggest a cautious stance. We can treat the tree as one lens among several, corroborating its segments with other summaries or sensitivity checks.

10.10 Summary and Key Takeaways

- Regression trees provide interpretable segmentation for continuous outcomes, with each leaf defining a clear subgroup.
- Pruning balances fit and readability, and cross validation offers a principled guide for tree size.
- Tree plots and segment summaries communicate structure more effectively than raw console output.
- Variable importance is a helpful ranking, but it gains meaning only when paired with split rules and segment profiles.

10.11 Looking Ahead

Regression trees focus on continuous outcomes. The next chapter extends the same logic to **classification trees**, where the outcome is categorical and performance is often summarized with a confusion matrix. The transition is conceptually smooth, with the same recursive partitioning framework and many of the same interpretive benefits.

11 Classification Trees and Confusion Matrix Insights

11.1 Introduction: From Segments to Classes

Regression trees helped us segment continuous outcomes into interpretable subgroups. Classification trees extend the same recursive partitioning logic to **categorical outcomes**, where each leaf predicts a class label rather than a mean value. The descriptive appeal is similar, a small set of decision rules can summarize how predictors separate categories.

In this chapter we focus on how classification trees communicate structure and performance. We emphasize confusion matrices (including their binary 2×2 form), class probabilities and the Brier score, and the interpretive meaning of splits, rather than only predictive accuracy.

11.2 The Classification Tree Idea

Classification trees divide the predictor space into regions that are as **class homogeneous** as possible. Each split is chosen to reduce impurity in the child nodes. Common impurity measures include the Gini index and cross-entropy (deviance). For the Gini index (Breiman et al. 2017),

$$G = 1 - \sum_{k=1}^{K} p_k^2,$$

where p_k is the proportion of class k in a node. A good split reduces impurity by creating child nodes with more concentrated class distributions.

As with regression trees, classification trees are piecewise constant models. They are not smooth, but they are interpretable, a valuable feature when the goal is explanation and communication.

11.3 A Reproducible Example with the Iris Data

We use the classic `iris` dataset. The response is species, and the predictors are the four flower measurements. The dataset is small enough to examine directly, and it produces a compact tree.

```
data(iris)
glimpse(iris)
```

```
Rows: 150
Columns: 5
$ Sepal.Length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9, 5.4, 4.~
$ Sepal.Width  <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.~
$ Petal.Length <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5, 1.5, 1.~
$ Petal.Width  <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.2, 0.~
$ Species      <fct> setosa, setosa, setosa, setosa, setosa, setosa, setosa, s~
```

Before fitting a tree, we can visualize how two features separate the classes. This is not required, but it provides intuition for the splits that follow.

```
iris |>
    ggplot(aes(x = Petal.Length, y = Petal.Width, color = Species)) +
    geom_point(alpha = 0.7, size = 2) +
    stat_ellipse(type = "norm", linewidth = 0.7) +
    labs(
        title = "Iris Species by Petal Measurements",
        x = "Petal length",
        y = "Petal width"
    ) +
    theme_minimal()
```

11.4 Fitting a Classification Tree

We fit a tree using `rpart` with `method = "class"`. We set a small complexity parameter to grow an initial tree and then prune it based on cross-validated error.

```
set.seed(123)
class_tree <- rpart(
    Species ~ .,
    data = iris,
    method = "class",
    control = rpart.control(minsplit = 10, cp = 0.001)
)
```

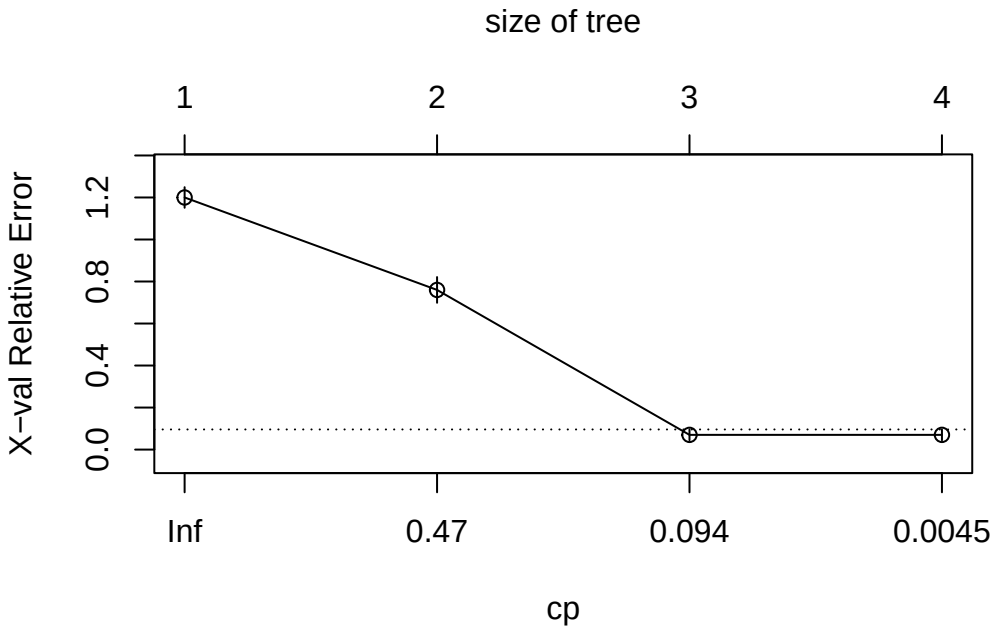
As in the regression case, the console output is not the most readable summary. We move quickly to pruning diagnostics and visual representations.

11.5 Pruning and Visualizing the Tree

```
printcp(class_tree)
```

```
Classification tree:
rpart(formula = Species ~ ., data = iris, method = "class", control = rpart.control(minsplit
    cp = 0.001))

Variables actually used in tree construction:
[1] Petal.Length Petal.Width

Root node error: 100/150 = 0.66667

n= 150

     CP nsplit rel error xerror     xstd
1 0.500      0      1.00   1.20 0.048990
2 0.440      1      0.50   0.76 0.061232
3 0.020      2      0.06   0.07 0.025833
4 0.001      3      0.04   0.07 0.025833
```

```r
plotcp(class_tree)
```



We select the complexity parameter that minimizes cross-validated error and prune the tree.

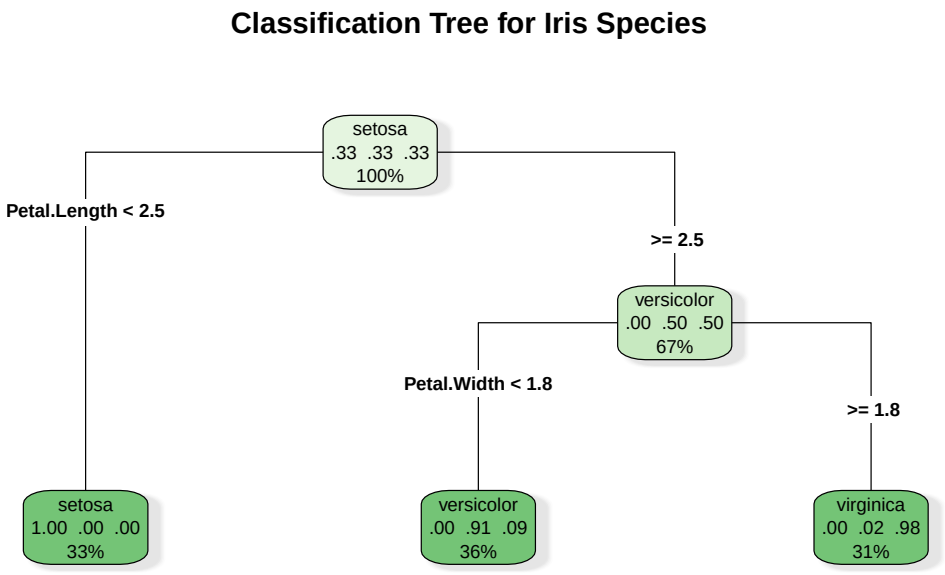
```r
best_cp <- class_tree$cptable[which.min(class_tree$cptable[, "xerror"]), "CP"]
pruned_class_tree <- prune(class_tree, cp = best_cp)
```

```
best_cp
```

```
[1] 0.02
```

The plot below emphasizes readability, with class labels and node sizes that make the main splits easy to follow.

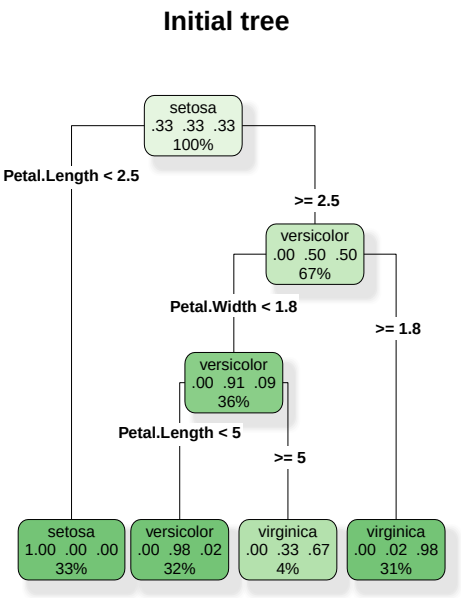
```
rpart.plot(
    pruned_class_tree,
    type = 4,
    extra = 104,
    fallen.leaves = TRUE,
    box.palette = "Greens",
    branch.lty = 1,
    shadow.col = "gray90",
    main = "Classification Tree for Iris Species"
)
```

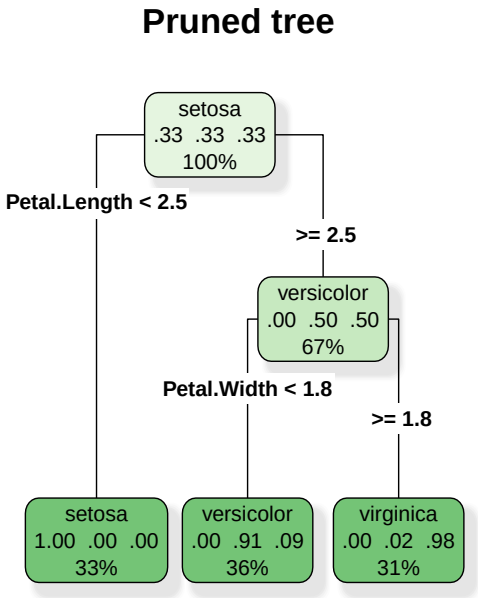


To see the effect of pruning more directly, we can compare the initial tree to the pruned tree side by side. The pruned version is smaller, with fewer terminal nodes and clearer rules.

```
op <- par(mfrow = c(1, 2), mar = c(1, 1, 3, 1)) # for side-by-side plots

rpart.plot(
```

```
    class_tree,
    type = 4,
    extra = 104,
    fallen.leaves = TRUE,
    box.palette = "Greens",
    branch.lty = 1,
    shadow.col = "gray90",
    main = "Initial tree"
)
rpart.plot(
    pruned_class_tree,
    type = 4,
    extra = 104,
    fallen.leaves = TRUE,
    box.palette = "Greens",
    branch.lty = 1,
    shadow.col = "gray90",
    main = "Pruned tree"
)
```





	setosa	versicolor	virginica
setosa	50	0	0
versicolor	0	49	1
virginica	0	5	45

11.6 Confusion Matrix and Summary Metrics

Classification trees are often evaluated with a confusion matrix. Even in descriptive work, the matrix reveals where the tree distinguishes classes well and where it confuses them.

```
pred_class <- predict(pruned_class_tree, iris, type = "class")
conf_mat <- table(Actual = iris$Species, Predicted = pred_class)

conf_mat_df <- as.data.frame.matrix(conf_mat)
conf_mat_df <- tibble::rownames_to_column(conf_mat_df, var = "Actual")

conf_mat_df |>
    gt(rowname_col = "Actual")
```

We can also compute summary metrics. For a multiclass setting, it is common to report overall accuracy and macro-averaged precision, recall, and F1 scores.

```
accuracy <- sum(diag(conf_mat)) / sum(conf_mat)

classes <- rownames(conf_mat)
precision <- recall <- f1 <- numeric(length(classes))

for (i in seq_along(classes)) {
    precision[i] <- conf_mat[i, i] / sum(conf_mat[, i])
    recall[i] <- conf_mat[i, i] / sum(conf_mat[i, ])
    f1[i] <- 2 * (precision[i] * recall[i]) / (precision[i] + recall[i])
}

metrics_df <- data.frame(
    Class = classes,
    Precision = precision,
    Recall = recall,
    F1 = f1
)
```

Class	Precision	Recall	F1
setosa	1.000	1.000	1.000
versicolor	0.907	0.980	0.942
virginica	0.978	0.900	0.938

```
metrics_df |>
    gt() |>
    cols_label(
        Class = "Class",
        Precision = "Precision",
        Recall = "Recall",
        F1 = "F1"
    ) |>
    fmt_number(columns = c(Precision, Recall, F1), decimals = 3)
```

```
accuracy
```

```
[1] 0.96
```

```
macro_f1 <- mean(f1)
macro_f1
```

```
[1] 0.9599359
```

The confusion matrix gives a granular view of errors, while the macro averages summarize performance across classes. In descriptive contexts, these summaries indicate whether the tree is separating groups in a meaningful way, rather than merely fitting noise.

11.6.1 The 2×2 Case: Binary Classification

The iris example uses three classes, but in practice the most common setting is a **binary outcome**: two classes such as Yes/No, positive/negative, or event/no-event. The confusion matrix then takes a specific 2×2 form that has its own conventional vocabulary worth knowing, because it appears throughout the later chapters.

The four cells of a binary confusion matrix are:

	Predicted Negative	Predicted Positive
Actual Negative	True Negative (TN)	False Positive (FP)
Actual Positive	False Negative (FN)	True Positive (TP)

From these four counts, the summary metrics we computed above reduce to familiar binary formulas:

- **Accuracy**: $(TP + TN)/(TP + TN + FP + FN)$ — overall fraction correctly classified
- **Precision**: $TP/(TP + FP)$ — of those predicted positive, how many actually are
- **Recall** (also called sensitivity): $TP/(TP + FN)$ — of those actually positive, how many were caught
- **Specificity**: $TN/(TN + FP)$ — of those actually negative, how many were correctly identified as such

In descriptive work, the confusion matrix is most useful not as a performance scorecard but as a diagnostic: it shows whether a tree is systematically confusing one class for another, which can reveal structure in the data. A tree that consistently misclassifies one group is telling you something about how those cases are distributed in the predictor space, not just about model quality.

The layout of the matrix (which class is "positive") matters for interpretation. When one class carries more practical weight (for instance, disease cases in a health setting, or high-value customers in a business setting), it is conventional to treat it as the positive class so that false negatives and false positives have a clear asymmetric meaning.

11.6.2 The Brier Score: Evaluating Predicted Probabilities

The metrics above, that is, accuracy, precision, recall, and F1, all evaluate hard class predictions: the tree assigns each observation to a single class and we check whether it was right. But classification trees also produce **class probabilities** at each leaf (the proportion of training observations of each class in that node), and those probabilities carry information that hard labels discard.

The **Brier score** measures how well-calibrated predicted probabilities are, by computing the average squared deviation between the predicted probability and the actual binary outcome (BRIER 1950):

$$BS = \frac{1}{n}\sum_{i=1}^{n}(p_i - y_i)^2$$

where p_i is the predicted probability of the positive class for observation i, and y_i is 1 if the event occurred and 0 otherwise. A Brier score of 0 means perfect probability predictions; a

score of 1 is the worst possible. A naive model that always predicts the base rate has a Brier score equal to $p(1-p)$, which provides a useful reference point.

In a binary setting, the computation is straightforward:

```
# Brier score for a binary outcome (illustration with a two-class version of iris)
iris_binary <- iris |>
  filter(Species != "virginica") |>
  mutate(Species = droplevels(Species))

tree_binary <- rpart(
  Species ~ .,
  data = iris_binary,
  method = "class",
  control = rpart.control(cp = 0.001)
)

pred_prob_binary <- predict(tree_binary, iris_binary, type = "prob")[, "versicolor"]
actual_binary <- as.integer(iris_binary$Species == "versicolor")

brier_score <- mean((pred_prob_binary - actual_binary)^2)
brier_score
```

```
[1] 0
```

The Brier score is most informative when compared to this baseline:

```
base_rate <- mean(actual_binary)
brier_baseline <- base_rate * (1 - base_rate)
brier_baseline
```

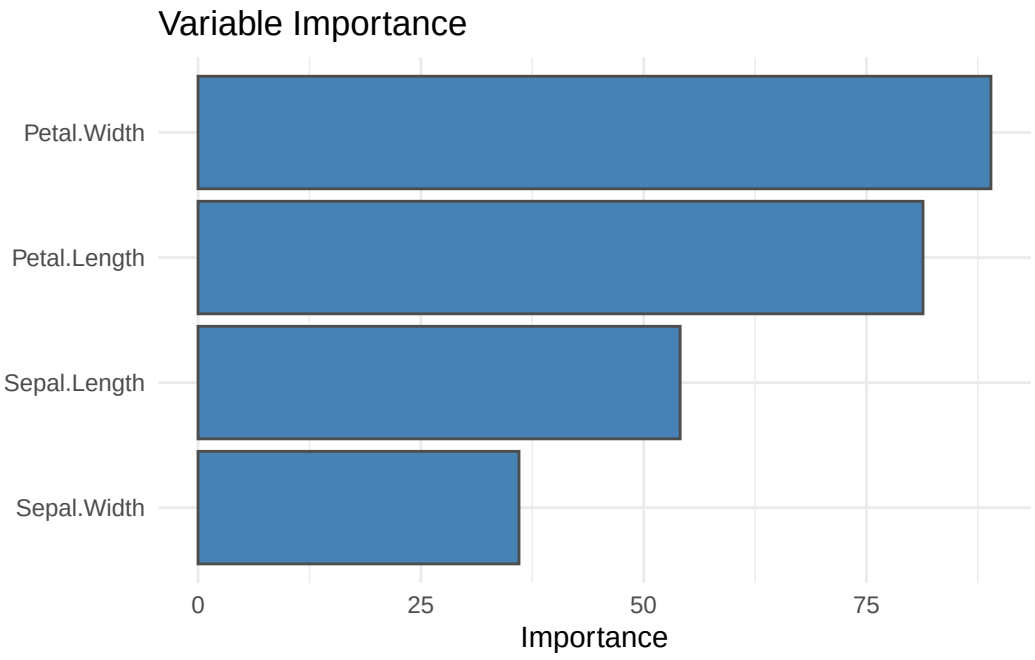
```
[1] 0.25
```

A Brier score well below the baseline suggests the model's probability estimates are adding genuine information beyond the marginal class distribution. In descriptive work, this matters because it indicates whether the leaf-level probability estimates are trustworthy enough to support interpretation, for instance, when communicating risk gradients or ranking observations by predicted likelihood.

11.7 Variable Importance for Classification Trees

Trees provide a built-in measure of variable importance, based on how much each variable reduces impurity across splits. This is a useful complement to the tree structure itself.

```
imp <- sort(pruned_class_tree$variable.importance, decreasing = TRUE)
imp_df <- data.frame(
    variable = names(imp),
    importance = as.numeric(imp)
)

imp_df |>
    ggplot(aes(x = reorder(variable, importance), y = importance)) +
    geom_col(fill = "steelblue", color = "gray30") +
    coord_flip() +
    labs(
        title = "Variable Importance",
        x = NULL,
        y = "Importance"
    ) +
    theme_minimal()
```



Importance is best read as a ranking. It suggests which predictors shape the tree most strongly, while the split rules clarify how those predictors partition the classes.

11.8 Interpretation and Practical Considerations

Classification trees are attractive because they express decision boundaries as explicit rules. That said, the following considerations are often important in practice:

- **Class probabilities** can be more informative than hard labels, especially when classes overlap.
- **Imbalanced classes** may require cost-sensitive splits or reweighting to avoid dominance by the majority class.
- **Stability** remains a concern, small changes in data can alter splits, so we treat a single tree as a descriptive summary rather than a definitive model.

These points reinforce the value of using trees as one component in a broader descriptive toolkit.

11.9 Summary and Key Takeaways

- Classification trees extend recursive partitioning to categorical outcomes, yielding interpretable decision rules.
- Pruning is essential for readability and stability, and cross-validation offers a reasonable default for tree size.
- Confusion matrices reveal where the tree succeeds or confuses classes, while macro metrics summarize overall balance.
- In binary classification, the 2×2 confusion matrix introduces a standard vocabulary (true positives, false negatives, specificity) that supports asymmetric interpretation when one class carries more practical weight.
- The Brier score complements hard-label metrics by evaluating the quality of predicted probabilities, using the base rate as a natural reference point.
- Variable importance provides a ranking of predictors, best interpreted alongside the actual split structure.

11.10 Looking Ahead

Classification trees provide a transparent baseline, but they can be unstable and sometimes less accurate than ensemble methods. The next chapter introduces tree ensembles, where many trees are combined to improve stability and performance while retaining interpretive cues.

12 Ensemble Methods as Descriptive Instruments

12.1 Introduction: From Single Trees to Ensembles

The previous two chapters introduced regression and classification trees as transparent tools for segmentation. They are appealing precisely because they are easy to read, but in some cases they may also be **unstable**. For instance, small changes in the data can lead to different splits and different segment definitions. Ensemble methods respond to that instability by combining many trees into a single model. The result is often more accurate and more stable, at the cost of some immediacy in interpretation.

In a descriptive workflow, ensembles can still be valuable. They provide **robust summaries of variable influence** and a smoother picture of how predictors shape outcomes. We can treat them as instruments for stabilizing patterns that appear in single trees, rather than as black boxes that replace interpretive judgment.

This chapter introduces three ensemble ideas that are central in practice: bagging, random forests, and gradient boosting. We focus on their descriptive roles, how they average or accumulate information, and how we can extract interpretable summaries from them.

12.2 Why Ensembling Helps

Ensemble methods rest on a simple principle. If we fit many models that are each somewhat noisy, then **averaging** them can reduce variance. For tree based models, this is especially important because recursive partitioning is high variance by construction. Even a single split can change if one observation is perturbed.

From a descriptive standpoint, the key idea is that repeated tree fitting can help identify **consistent signals**. Rather than trusting one tree, we can look at patterns that persist across many resamples. Variable importance and partial dependence are two common summaries, and we will return to them in later chapters.

12.3 Bagging: Bootstrap Aggregation

Bagging, short for bootstrap aggregation, is the simplest ensemble approach (Breiman 1996). We take many bootstrap samples of the data, fit a tree on each sample, and average their predictions. For regression, we average predicted values. For classification, we average class probabilities and then choose a class if needed.

Bagging reduces variance because each tree is fitted on a slightly different dataset. A useful side benefit is the **out of bag** (OOB) observations, the roughly one third of data points not included in each bootstrap sample. By predicting those points, we obtain a built in estimate of predictive error without a separate validation set.

12.4 Random Forests: Decorrelated Trees

Random forests add one more layer of randomness (Breiman 2001). At each split, the algorithm considers only a random subset of predictors, rather than all of them. This tends to **decorrelate** the trees, which makes averaging more effective. The result is typically lower variance and improved accuracy compared to bagging.

Random forests also provide **variable importance** measures that summarize how much each predictor contributes to reducing impurity across the forest. These scores are not causal or definitive, but they provide a stable ranking that is often useful for descriptive purposes.

12.5 Gradient Boosting: Sequential Refinement

Boosting combines trees in a different way (J. H. Friedman 2001). Instead of building trees independently and averaging them, boosting builds trees **sequentially**. Each new tree is trained to predict the residuals of the current model, so the ensemble gradually improves its fit.

Gradient boosting can be seen as an additive model of many small trees. For descriptive analysis, this can be attractive because it yields smooth, flexible fits while still allowing us to inspect variable importance and functional relationships. However, the model is more complex, so interpretability depends heavily on the summaries we choose to extract.

12.6 A Reproducible Example with Housing Prices

We return to the Boston housing data. The outcome is `medv`, and we use a modest set of predictors. We will compare a single regression tree, a bagged tree ensemble, a random forest, and a gradient boosted model using a simple train test split.

```r
data(Boston, package = "MASS")

set.seed(123)
idx <- sample(seq_len(nrow(Boston)), size = 0.7 * nrow(Boston))
train <- Boston[idx, ]
test <- Boston[-idx, ]

predictors <- c("lstat", "rm", "ptratio", "indus", "nox", "crim")
formula <- as.formula(paste("medv ~", paste(predictors, collapse = " + ")))
mtry_reg <- length(attr(terms(formula), "term.labels"))
```

12.6.1 A Single Regression Tree

We begin with a tree as a baseline. The goal is not to optimize hyperparameters, but to provide a comparable reference for the ensemble models.

```r
set.seed(123)
tree_fit <- rpart(
    formula,
    data = train,
    method = "anova",
    control = rpart.control(minsplit = 30, cp = 0.001)
)

tree_pred <- predict(tree_fit, newdata = test)
tree_rmse <- sqrt(mean((test$medv - tree_pred)^2))

tree_rmse
```

```
[1] 4.677431
```

RMSE, the root mean squared error, summarizes the typical prediction error by taking the square root of the average squared difference between observed and predicted values. It is expressed in the same units as `medv`, so smaller values indicate closer fit to the observed housing prices.

12.6.2 Bagging and Random Forests

Bagging is equivalent to a random forest where all predictors are considered at each split. We fit both bagging and a standard random forest to compare their behavior.

```
set.seed(123)
bag_fit <- randomForest(
    formula,
    data = train,
    mtry = mtry_reg,
    ntree = 500,
    importance = TRUE
)

rf_fit <- randomForest(
    formula,
    data = train,
    mtry = max(1, floor(mtry_reg / 3)),
    ntree = 500,
    importance = TRUE
)

bag_pred <- predict(bag_fit, newdata = test)
rf_pred <- predict(rf_fit, newdata = test)

bag_rmse <- sqrt(mean((test$medv - bag_pred)^2))
rf_rmse <- sqrt(mean((test$medv - rf_pred)^2))

bag_rmse
```

```
[1] 3.93907
```

```
rf_rmse
```

```
[1] 3.554599
```

12.6.3 Gradient Boosting

For boosting we use a standard configuration with shallow trees and a small learning rate. The number of trees is selected by cross validation within the `gbm` routine.

```
set.seed(123)
gbm_fit <- gbm(
    formula,
    data = train,
```

```
    distribution = "gaussian",
    n.trees = 2000,
    interaction.depth = 3,
    shrinkage = 0.01,
    bag.fraction = 0.7,
    cv.folds = 5,
    verbose = FALSE
)

best_iter <- gbm.perf(gbm_fit, method = "cv", plot.it = FALSE)
gbm_pred <- predict(gbm_fit, newdata = test, n.trees = best_iter)
gbm_rmse <- sqrt(mean((test$medv - gbm_pred)^2))

best_iter
```

```
[1] 1990
```

```
gbm_rmse
```

```
[1] 3.88826
```

12.6.4 Comparing Errors

We summarize the test set RMSE values. The exact numbers depend on the split, but the pattern is often consistent, ensembles tend to outperform a single tree in predictive accuracy.

```
rmse_tbl <- tibble::tibble(
    Model = c("Single tree", "Bagging", "Random forest", "Gradient boosting"),
    RMSE = c(tree_rmse, bag_rmse, rf_rmse, gbm_rmse)
)

rmse_tbl |>
    gt() |>
    fmt_number(columns = RMSE, decimals = 3)
```

When we compare the four RMSE values, we read them as competing summaries of average prediction error on the same test data. A lower RMSE indicates that the model, on average, is closer to observed values. If the ensemble models show smaller RMSE than the single tree, this supports the idea that averaging or sequential refinement reduces variance and improves accuracy. Differences that are small relative to the outcome scale are still worth noting, but they

Model	RMSE
Single tree	4.677
Bagging	3.939
Random forest	3.555
Gradient boosting	3.888

may not imply large substantive gains, so it is helpful to interpret them alongside descriptive goals rather than only as a ranking.

With that regression baseline in place, we now shift to a binary outcome to see how the same ensemble ideas translate to classification.

12.7 A Classification Example: Pima Diabetes

Ensembles are equally useful when the outcome is categorical. We use the Pima diabetes data from MASS. The outcome indicates diabetes status, and the predictors are clinical measurements. We use the provided training and test splits and repeat the same ensemble comparisons. A common summary is accuracy, the proportion of cases assigned to the correct class, but we also inspect a confusion matrix and macro averaged precision and recall to account for class imbalance.

```
data(Pima.tr, package = "MASS")
data(Pima.te, package = "MASS")

train_pima <- Pima.tr
test_pima <- Pima.te

pima_formula <- type ~ npreg + glu + bp + skin + bmi + ped + age
mtry_pima <- length(attr(terms(pima_formula), "term.labels"))
```

12.7.1 A Single Classification Tree

We begin with a single classification tree as a baseline, comparable to the regression tree in the Boston example.

```
set.seed(123)
pima_tree_fit <- rpart(
    pima_formula,
    data = train_pima,
```

```
    method = "class",
    control = rpart.control(minsplit = 20, cp = 0.01)
)

pima_tree_pred <- predict(pima_tree_fit, newdata = test_pima, type = "class")
pima_tree_acc <- mean(pima_tree_pred == test_pima$type)

pima_tree_acc
```

```
[1] 0.7319277
```

12.7.2 Ensemble Methods for Classification

Now we fit the three ensemble methods for comparison.

```
set.seed(123)
pima_bag <- randomForest(
    pima_formula,
    data = train_pima,
    mtry = mtry_pima,
    ntree = 500
)

pima_rf <- randomForest(
    pima_formula,
    data = train_pima,
    mtry = max(1, floor(mtry_pima / 2)),
    ntree = 500,
    importance = TRUE
)

pima_bag_pred <- predict(pima_bag, newdata = test_pima, type = "class")
pima_rf_pred <- predict(pima_rf, newdata = test_pima, type = "class")

pima_bag_acc <- mean(pima_bag_pred == test_pima$type)
pima_rf_acc <- mean(pima_rf_pred == test_pima$type)

set.seed(123)
pima_gbm <- gbm(
    type_num ~ npreg + glu + bp + skin + bmi + ped + age,
    data = train_pima |> mutate(type_num = ifelse(type == "Yes", 1, 0)),
```

```
    distribution = "bernoulli",
    n.trees = 1500,
    interaction.depth = 2,
    shrinkage = 0.05,
    bag.fraction = 0.7,
    cv.folds = 5,
    verbose = FALSE
)

pima_best <- gbm.perf(pima_gbm, method = "cv", plot.it = FALSE)
pima_prob <- predict(pima_gbm, newdata = test_pima, n.trees = pima_best, type = "response")
pima_gbm_pred <- ifelse(pima_prob > 0.5, "Yes", "No")
pima_gbm_pred <- factor(pima_gbm_pred, levels = levels(test_pima$type))

pima_gbm_acc <- mean(pima_gbm_pred == test_pima$type)

pima_bag_acc
```

```
[1] 0.75
```

```
pima_rf_acc
```

```
[1] 0.7560241
```

```
pima_gbm_acc
```

```
[1] 0.7951807
```

```
acc_tbl <- tibble::tibble(
    Model = c("Single tree", "Bagging", "Random forest", "Gradient boosting"),
    Accuracy = c(pima_tree_acc, pima_bag_acc, pima_rf_acc, pima_gbm_acc)
)

acc_tbl |>
    gt() |>
    fmt_number(columns = Accuracy, decimals = 3)
```

The ensemble methods show reasonably comparable performance on this test set, and they generally improve upon the single tree baseline. Exact accuracy values depend on the train test

Model	Accuracy
Single tree	0.732
Bagging	0.750
Random forest	0.756
Gradient boosting	0.795

Actual	No	Yes
No	191	32
Yes	49	60

split and hyperparameter choices, but all three ensemble approaches demonstrate the value of combining multiple trees to obtain stable predictions. Given the small test set, we should not over-interpret small differences in accuracy. Instead, we examine class specific performance to understand how well the models distinguish between the two diabetes classes.

12.7.3 Detailed Performance Metrics for Random Forest

We now inspect the confusion matrix and class specific metrics for the random forest model, which we can use as a representative ensemble to understand detailed classification behavior.

```
conf_mat_pima <- table(Actual = test_pima$type, Predicted = pima_rf_pred)
conf_mat_pima_df <- as.data.frame.matrix(conf_mat_pima)
conf_mat_pima_df <- tibble::rownames_to_column(conf_mat_pima_df, var = "Actual")

conf_mat_pima_df |>
    gt()
```

In settings with class imbalance, accuracy can be misleading because a model can perform well by favoring the majority class. Macro averaged precision and recall give each class equal weight, which can be more informative when we care about minority classes.

```
classes_pima <- rownames(conf_mat_pima)
precision_pima <- recall_pima <- f1_pima <- numeric(length(classes_pima))

for (i in seq_along(classes_pima)) {
    precision_pima[i] <- conf_mat_pima[i, i] / sum(conf_mat_pima[, i])
    recall_pima[i] <- conf_mat_pima[i, i] / sum(conf_mat_pima[i, ])
    f1_pima[i] <- 2 * (precision_pima[i] * recall_pima[i]) / (precision_pima[i] + recall_pima
}
```

Class	Precision	Recall	F1
No	0.796	0.857	0.825
Yes	0.652	0.550	0.597

```
metrics_pima <- data.frame(
    Class = classes_pima,
    Precision = precision_pima,
    Recall = recall_pima,
    F1 = f1_pima
)

metrics_pima |>
    gt() |>
    cols_label(
        Class = "Class",
        Precision = "Precision",
        Recall = "Recall",
        F1 = "F1"
    ) |>
    fmt_number(columns = c(Precision, Recall, F1), decimals = 3)
```

```
macro_precision <- mean(precision_pima)
macro_recall <- mean(recall_pima)
macro_f1 <- mean(f1_pima)

macro_precision
```

```
[1] 0.7240036
```

```
macro_recall
```

```
[1] 0.7034805
```

```
macro_f1
```

```
[1] 0.7110345
```

Reading these macro averages alongside accuracy helps us judge whether high overall performance is shared across classes or driven mainly by the easiest class to predict. Macro averaged metrics provide a balanced view when classes are imbalanced, ensuring that we do not overlook performance on the minority class.

12.8 Variable Importance as a Descriptive Summary

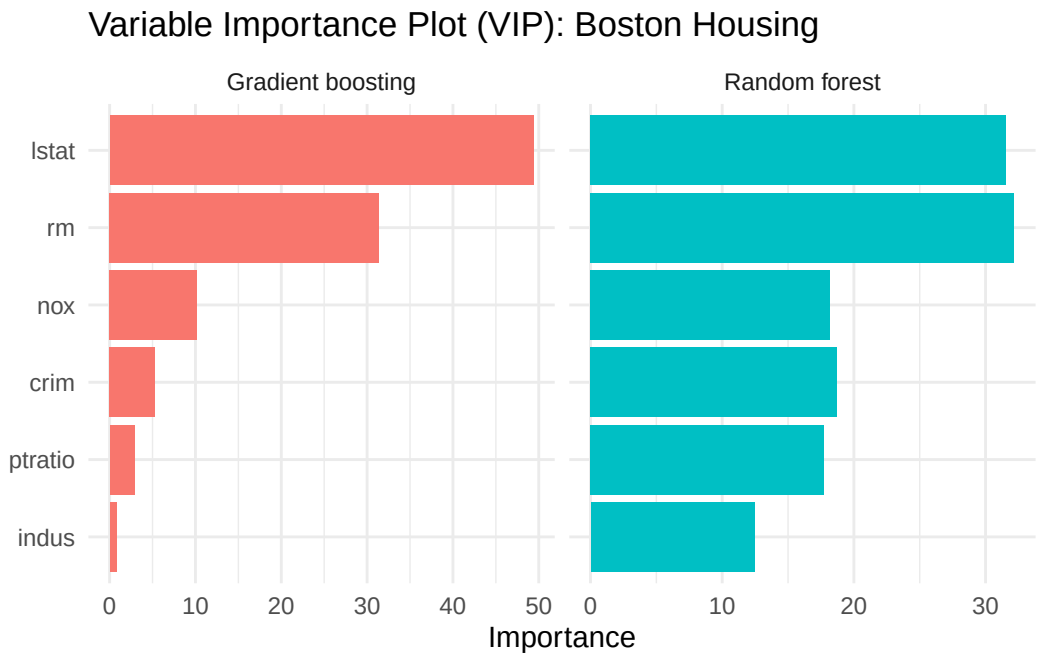
Ensemble models are less transparent than a single tree, but they can still provide interpretable summaries. A common choice is variable importance. We can extract importance rankings for both the random forest and the boosted model and plot them in a comparable format. We illustrate this for both the Boston housing regression and the Pima diabetes classification examples.

We focus on random forest and gradient boosting because they represent two fundamentally different ensemble strategies, aggregated independent trees versus sequential refinement, while bagging would yield variable importance very similar to random forest since it differs only in using all predictors at each split.

12.8.1 Variable Importance for Boston Housing

```
rf_imp <- importance(rf_fit, type = 1)
rf_imp_df <- data.frame(
    variable = rownames(rf_imp),
    importance = rf_imp[, 1],
    model = "Random forest"
)

gbm_imp <- summary(gbm_fit, n.trees = best_iter, plotit = FALSE)
gbm_imp_df <- data.frame(
    variable = gbm_imp$var,
    importance = gbm_imp$rel.inf,
    model = "Gradient boosting"
)

imp_df <- bind_rows(rf_imp_df, gbm_imp_df)

imp_df |>
    ggplot(aes(x = reorder(variable, importance), y = importance, fill = model)) +
    geom_col(show.legend = FALSE) +
    coord_flip() +
```

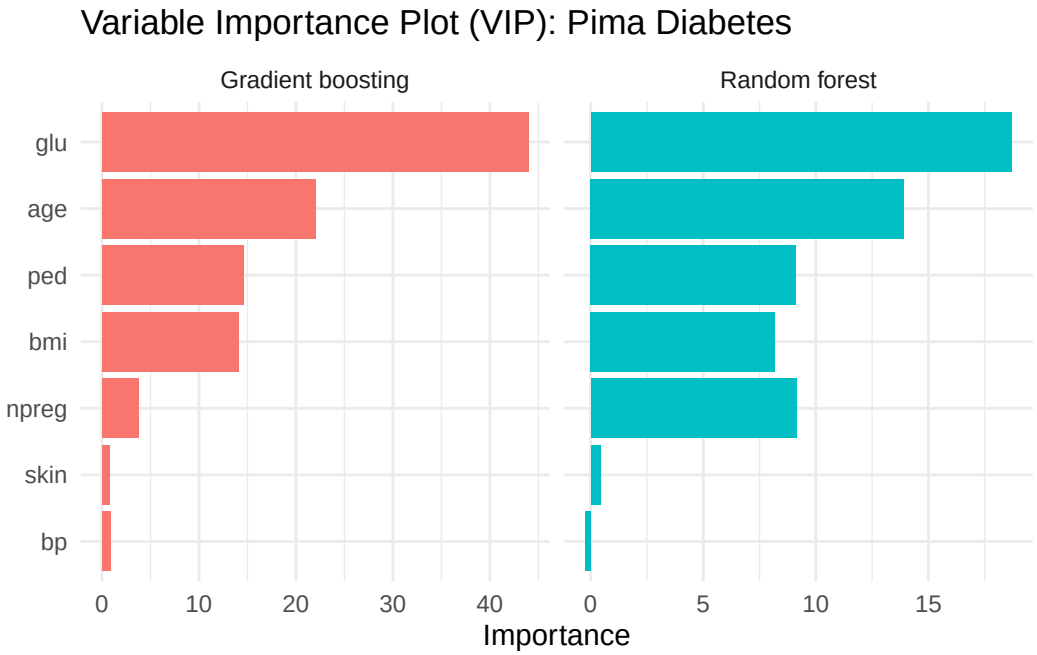
```
    facet_wrap(~ model, scales = "free_x") +
    labs(
        title = "Variable Importance Plot (VIP): Boston Housing",
        x = NULL,
        y = "Importance"
    ) +
    theme_minimal()
```



12.8.2 Variable Importance for Pima Diabetes

```
pima_rf_imp <- importance(pima_rf, type = 1)
pima_rf_imp_df <- data.frame(
    variable = rownames(pima_rf_imp),
    importance = pima_rf_imp[, 1],
    model = "Random forest"
)

pima_gbm_imp <- summary(pima_gbm, n.trees = pima_best, plotit = FALSE)
pima_gbm_imp_df <- data.frame(
    variable = pima_gbm_imp$var,
    importance = pima_gbm_imp$rel.inf,
```

```
    model = "Gradient boosting"
)

pima_imp_df <- bind_rows(pima_rf_imp_df, pima_gbm_imp_df)

pima_imp_df |>
    ggplot(aes(x = reorder(variable, importance), y = importance, fill = model)) +
    geom_col(show.legend = FALSE) +
    coord_flip() +
    facet_wrap(~ model, scales = "free_x") +
    labs(
        title = "Variable Importance Plot (VIP): Pima Diabetes",
        x = NULL,
        y = "Importance"
    ) +
    theme_minimal()
```



The importance rankings are often more stable than the splits of a single tree. They do not tell us the direction of an effect or the shape of a relationship, but they highlight the predictors that the ensemble relies on most heavily.

12.9 Interpretation and Practical Considerations

Ensembles provide strong performance and stable summaries, but they also come with interpretive costs. In descriptive settings, a few considerations are especially relevant:

- **Transparency** is reduced compared to a single tree, so summaries like variable importance and partial dependence become essential.
- **Hyperparameters** influence results, including tree depth, learning rate, and the number of trees. We gain stability but also introduce modeling choices that should be documented.
- **Communication** can focus on patterns that persist across methods, for example predictors that are consistently ranked high in both random forests and boosting.

These points encourage a balanced stance. Ensembles are not replacements for careful descriptive reasoning, but they can help confirm and stabilize the patterns that single trees suggest.

12.10 Summary and Key Takeaways

- Bagging reduces variance by averaging trees fit to bootstrap samples, with OOB error as a natural diagnostic.
- Random forests add predictor subsampling, which decorrelates trees and often improves stability and accuracy.
- Gradient boosting builds trees sequentially and yields flexible fits, with interpretability depending on extracted summaries.
- Variable importance provides a robust descriptive ranking, but it should be paired with substantive interpretation and complementary plots.

12.11 Looking Ahead

Ensembles shift attention from individual decision rules to stable summaries of model behavior. The next chapter expands this idea by surveying interpretable machine learning approaches that help us understand complex models, including ensembles, through structured diagnostics and explanations.

Part V

Interpretable Machine Learning

13 Interpretable ML: An Overview

13.1 Introduction: From Predictive Strength to Explanatory Clarity

In the previous chapters we used tree based models, including ensembles, as descriptive instruments. We saw that ensembling can improve stability and predictive performance, but we also saw a recurrent tension, as model complexity grows, direct inspection becomes harder.

Interpretable machine learning addresses this tension. The goal is not to choose between prediction and interpretation in absolute terms. Instead, we try to recover useful explanations about model behavior, and to do so in a way that is technically sound and substantively meaningful.

This chapter provides a conceptual map of interpretable ML. We distinguish global and local explanations, model specific and model agnostic methods, and intrinsic and post hoc interpretability. We then work through a reproducible example that illustrates how these ideas can be used in practice.

13.2 What We Mean by "Interpretability"

Interpretability is often used loosely, so it helps to be explicit. In this book, we treat an explanation as useful when it helps us answer at least one of the following questions:

- Which predictors are most associated with the model output?
- How does the predicted response change when a predictor changes, while others are held fixed or averaged over?
- Why did the model produce a particular prediction for one observation?

These questions are related, but not identical. Variable ranking gives a broad sense of relevance, functional summaries describe shape, and case level explanations address specific predictions. A single summary rarely answers all three questions.

Interpretability is also contextual. An explanation that is useful for exploratory analysis may not be sufficient for auditing, policy communication, or scientific reporting. For that reason, we usually combine several complementary summaries.

Interpretive question	Scope	Typical tools
Which variables matter most overall?	Global	Permutation importance, impurity based i
How does the response vary with one predictor?	Global	Partial dependence, accumulated local effe
Why this specific prediction?	Local	Local perturbation profiles, Shapley style
Can we summarize a black box with simple rules?	Global	Surrogate trees, sparse rule lists

13.3 A Practical Taxonomy of Interpretability Methods

Two dimensions are especially useful in practice:

- **Scope:** global methods summarize model behavior over many observations, whereas local methods focus on one prediction or a small neighborhood.
- **Model dependence:** model specific methods use internal model structure, whereas model agnostic methods only require predictions.

The distinction between intrinsic and post hoc interpretability is also useful. Intrinsic interpretability is built into the model class (for example, a small decision tree). Post hoc interpretability is added after fitting a flexible model.

This taxonomy is not rigid. Many methods can be adapted to different settings. It still provides a useful framework for deciding what to compute and how to communicate it.

13.4 Running Example: Random Forest on Boston Housing Data

To make the ideas concrete, we use the Boston housing dataset and fit a random forest model for `medv` (median home value). The model is intentionally moderate in complexity, which keeps interpretation manageable while preserving nonlinear structure.

```
data(Boston, package = "MASS")

set.seed(123)
idx <- sample(seq_len(nrow(Boston)), size = 0.7 * nrow(Boston))
train <- Boston[idx, ]
test <- Boston[-idx, ]

predictors <- c("lstat", "rm", "ptratio", "indus", "nox", "crim")
rf_formula <- as.formula(paste("medv ~", paste(predictors, collapse = " + ")))

set.seed(123)
rf_fit <- randomForest(
```

```
    rf_formula,
    data = train,
    ntree = 500,
    mtry = 2,
    importance = TRUE
)

rf_pred <- predict(rf_fit, newdata = test)
rf_rmse <- sqrt(mean((test$medv - rf_pred)^2))

rf_rmse
```

```
[1] 3.5575
```

The test RMSE gives a baseline for predictive quality. Interpretability summaries should be read alongside this baseline, because explanations of a weak model are often unstable or misleading.

13.5 Global Explanation 1: Permutation Importance

Permutation importance measures how much model error increases when a predictor is randomly shuffled. Shuffling breaks the predictor response relationship while preserving the predictor's marginal distribution.

If shuffling variable X_j increases error substantially, the model appears to rely on X_j for prediction. Formally, for an error function L (Breiman 2001; A. Fisher, Rudin, and Dominici 2019),

$$I_j = \mathbb{E}\left[L\left(Y, \hat{f}(X_{-j}, \pi(X_j))\right)\right] - \mathbb{E}\left[L\left(Y, \hat{f}(X)\right)\right],$$

where $\pi(X_j)$ denotes a random permutation of feature j.

In the next chapter, we return to permutation importance with an explicit finite sample estimator, so the conceptual definition introduced here is connected directly to implementation and reporting practice.

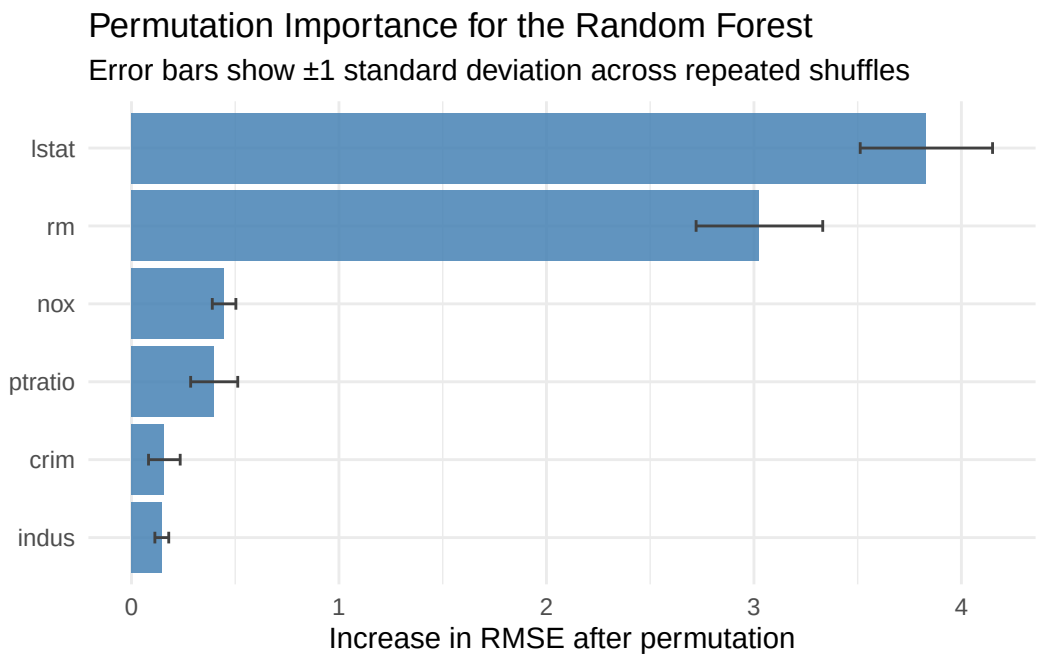
We now implement this procedure for our random forest model. We define a function that computes permutation importance for each predictor by repeatedly shuffling it and recording the change in test RMSE. We then apply this function to all six predictors and visualize the results to identify which features the model relies on most heavily.

```r
perm_importance <- function(model, data, outcome, vars, n_repeats = 8, seed = 123) {
    set.seed(seed)
    base_pred <- predict(model, newdata = data)
    base_rmse <- sqrt(mean((data[[outcome]] - base_pred)^2))

    results <- lapply(vars, function(v) {
        deltas <- numeric(n_repeats)
        for (b in seq_len(n_repeats)) {
            perm_data <- data
            perm_data[[v]] <- sample(perm_data[[v]])
            perm_pred <- predict(model, newdata = perm_data)
            perm_rmse <- sqrt(mean((data[[outcome]] - perm_pred)^2))
            deltas[b] <- perm_rmse - base_rmse
        }
        tibble::tibble(
            variable = v,
            mean_delta_rmse = mean(deltas),
            sd_delta_rmse = sd(deltas)
        )
    })

    dplyr::bind_rows(results) |>
        arrange(desc(mean_delta_rmse))
}

perm_imp <- perm_importance(
    model = rf_fit,
    data = test,
    outcome = "medv",
    vars = predictors,
    n_repeats = 10
)

perm_imp |>
    gt() |>
    fmt_number(columns = where(is.numeric), decimals = 3)
```

```r
perm_imp |>
    ggplot(aes(x = reorder(variable, mean_delta_rmse), y = mean_delta_rmse)) +
    geom_col(fill = "steelblue", alpha = 0.85) +
    geom_errorbar(
```

variable	mean_delta_rmse	sd_delta_rmse
lstat	3.831	0.319
rm	3.026	0.305
nox	0.446	0.057
ptratio	0.399	0.113
crim	0.158	0.076
indus	0.146	0.033

```
        aes(ymin = mean_delta_rmse - sd_delta_rmse, ymax = mean_delta_rmse + sd_delta_rmse),
        width = 0.15,
        color = "gray25"
    ) +
    coord_flip() +
    labs(
        title = "Permutation Importance for the Random Forest",
        subtitle = "Error bars show ±1 standard deviation across repeated shuffles",
        x = NULL,
        y = "Increase in RMSE after permutation"
    ) +
    theme_minimal()
```



Permutation importance is often more comparable across model classes than impurity based

scores. At the same time, it can be sensitive to correlated predictors, because one predictor can partially substitute for another.

13.6 Global Explanation 2: Partial Dependence

A partial dependence profile averages model predictions over the observed distribution of other features, while varying one focal predictor. For predictor X_s (J. H. Friedman 2001),

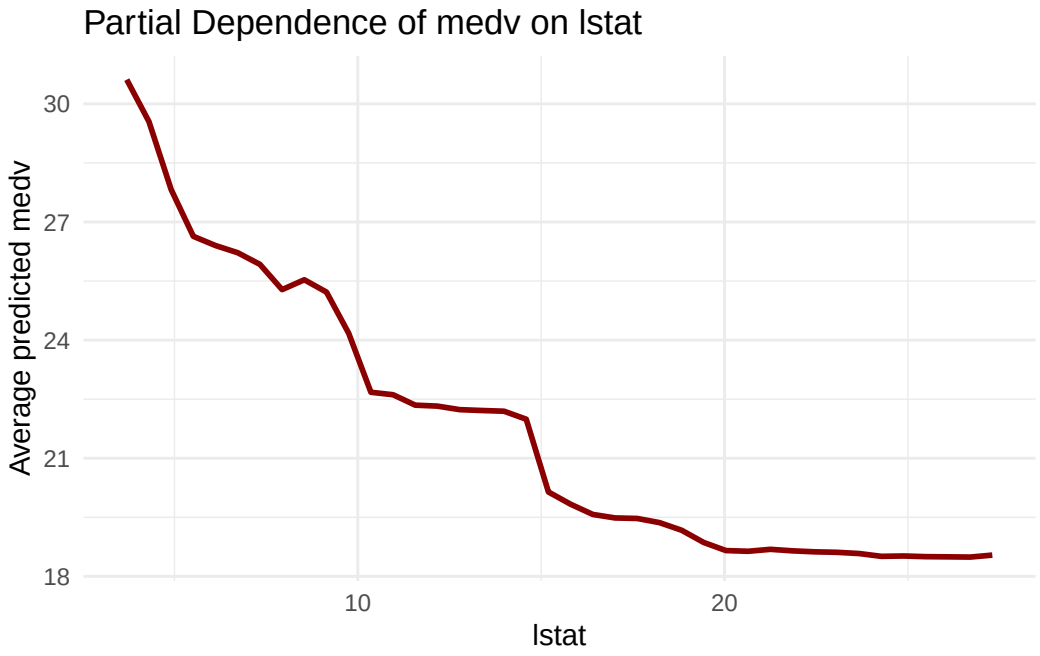
$$\hat{f}_{PD}(x_s) = \frac{1}{n}\sum_{i=1}^{n} \hat{f}(x_s, x_{i,C}),$$

where C indexes all non focal predictors.

To illustrate this, we compute the partial dependence profile for `lstat` (percentage of lower status population). We create a grid of `lstat` values spanning the 5th to 95th percentile, then for each grid point we set all observations to that `lstat` value and average the model predictions. The resulting curve shows how the expected prediction changes as `lstat` varies.

```
partial_dependence <- function(model, data, variable, grid_values) {
    pd_values <- sapply(grid_values, function(g) {
        new_data <- data
        new_data[[variable]] <- g
        mean(predict(model, newdata = new_data))
    })

    tibble::tibble(
        value = grid_values,
        pd = pd_values,
        variable = variable
    )
}

lstat_grid <- seq(
    from = quantile(train$lstat, 0.05),
    to = quantile(train$lstat, 0.95),
    length.out = 40
)

pd_lstat <- partial_dependence(
    model = rf_fit,
    data = train,
```

```
    variable = "lstat",
    grid_values = lstat_grid
)

pd_lstat |>
    ggplot(aes(x = value, y = pd)) +
    geom_line(color = "darkred", linewidth = 1) +
    labs(
        title = "Partial Dependence of medv on lstat",
        x = "lstat",
        y = "Average predicted medv"
    ) +
    theme_minimal()
```



This curve describes model behavior, not a causal response curve. It is best interpreted as a summary of how predictions tend to vary with `lstat` within the observed data region.

13.7 Local Explanation: A Case Based Perturbation Profile

Global summaries can hide heterogeneity. Local explanations focus on one observation and ask how its prediction changes when selected features are perturbed.

For this demonstration, we select a single observation from the test set and examine how its prediction would change if we replaced each of its feature values with low (10th percentile) or high (90th percentile) values from the training distribution. This counterfactual approach reveals which features have the largest leverage on this particular prediction.

```
case_id <- 10
case_obs <- test[case_id, , drop = FALSE]
case_pred <- as.numeric(predict(rf_fit, newdata = case_obs))

local_profile <- lapply(predictors, function(v) {
    q_low <- as.numeric(quantile(train[[v]], 0.10))
    q_high <- as.numeric(quantile(train[[v]], 0.90))

    case_low <- case_obs
    case_high <- case_obs
    case_low[[v]] <- q_low
    case_high[[v]] <- q_high

    pred_low <- as.numeric(predict(rf_fit, newdata = case_low))
    pred_high <- as.numeric(predict(rf_fit, newdata = case_high))

    tibble::tibble(
        variable = v,
        prediction_low = pred_low,
        prediction_high = pred_high,
        delta_low = pred_low - case_pred,
        delta_high = pred_high - case_pred
    )
}) |>
    dplyr::bind_rows()

local_profile |>
    transmute(
        Variable = variable,
        `Prediction (10th pct)` = prediction_low,
        `Prediction (90th pct)` = prediction_high,
        `Change from observed prediction, 10th pct` = delta_low,
        `Change from observed prediction, 90th pct` = delta_high
    ) |>
    gt() |>
    fmt_number(columns = where(is.numeric), decimals = 2)
```

This local profile is a sensitivity summary, not an additive attribution. It is still useful in

Variable	Prediction (10th pct)	Prediction (90th pct)	Change from observed prediction, 10th pct
lstat	22.54	13.98	4.29
rm	18.37	24.54	0.12
ptratio	20.13	18.66	1.88
indus	19.74	18.93	1.49
nox	18.81	18.86	0.57
crim	18.39	19.52	0.15

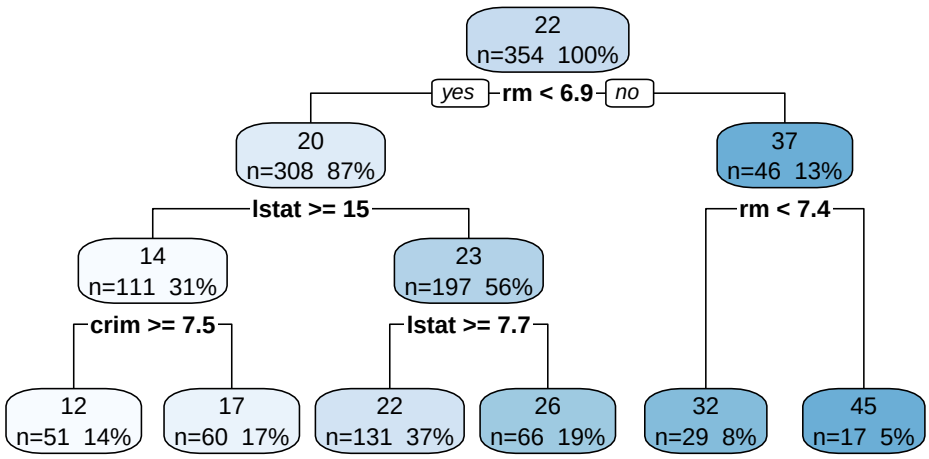
descriptive work because it shows which feature perturbations move this specific prediction the most.

13.8 Interpretable Surrogates: Approximating a Complex Model with a Tree

Another common strategy is to fit an interpretable surrogate model to predictions from a complex model. Here we fit a shallow regression tree to random forest predictions.

```
rf_train_pred <- predict(rf_fit, newdata = train)

surrogate_tree <- rpart(
    rf_train_pred ~ lstat + rm + ptratio + indus + nox + crim,
    data = train,
    method = "anova",
    control = rpart.control(maxdepth = 3, minsplit = 40, cp = 0.001)
)

rpart.plot(
    surrogate_tree,
    type = 2,
    extra = 101,
    fallen.leaves = TRUE,
    box.palette = "Blues",
    main = "Surrogate Tree for Random Forest Predictions"
)
```

Metric	Value
Correlation with RF predictions	0.939
RMSE against RF predictions	3.107



To assess how well this surrogate captures the original model's behavior, we compute fidelity diagnostics on the test set. We generate predictions from both the random forest and the surrogate tree, then calculate the correlation between them and the RMSE of their differences. High correlation and low RMSE indicate that the simpler tree approximates the complex model reasonably well within the observed data region.

```
rf_test_pred <- predict(rf_fit, newdata = test)
sur_test_pred <- predict(surrogate_tree, newdata = test)

surrogate_fidelity_cor <- cor(rf_test_pred, sur_test_pred)
surrogate_fidelity_rmse <- sqrt(mean((rf_test_pred - sur_test_pred)^2))

tibble::tibble(
    Metric = c("Correlation with RF predictions", "RMSE against RF predictions"),
    Value = c(surrogate_fidelity_cor, surrogate_fidelity_rmse)
) |>
    gt() |>
    fmt_number(columns = Value, decimals = 3)
```

Surrogate models can make complex behavior easier to communicate, but they remain approximations. Fidelity diagnostics help us avoid treating the surrogate as if it were the original

model.

13.9 Interpretation Pitfalls and Good Practice

Interpretable ML methods are informative, but they are not neutral or assumption free. Several points are worth keeping in view:

- **Association is not causation**. Explanations describe model behavior under the observed data generating process.
- **Correlated predictors complicate attribution**. Importance and effect summaries can shift when predictors carry overlapping information.
- **Extrapolation risk** matters for profile based methods. Curves are most reliable where data support is adequate.
- **Stability should be checked**. Explanations may vary across resamples, especially in small datasets.

In practice, we often obtain stronger descriptive insight by triangulating, for example, using importance rankings, functional profiles, and local perturbation summaries together.

13.10 Summary and Key Takeaways

This chapter introduced the conceptual foundations of interpretable machine learning and demonstrated a practical workflow for explaining model behavior. Several themes are worth emphasizing:

- **Interpretability is multifaceted**. Global explanations summarize overall model behavior, while local explanations address individual predictions. Model agnostic methods work across different model classes, while model specific methods exploit internal structure. Both perspectives are valuable.
- **Different questions require different tools**. Variable importance rankings answer "which features matter?", partial dependence profiles show "how do predictions vary with a feature?", and case based perturbations explain "why this specific prediction?". A single summary rarely answers all relevant questions.
- **Explanations are descriptive, not causal**. Permutation importance, partial dependence, and other methods describe model behavior under the observed data distribution. They do not identify causal effects or guarantee generalization beyond the training context.
- **Context shapes interpretation**. An explanation sufficient for exploratory analysis may not meet requirements for auditing, regulatory compliance, or scientific reporting. Always consider the intended use when selecting and presenting interpretability summaries.

- **Triangulation strengthens insight**. Combining importance rankings, functional profiles, and local explanations helps us build a more complete picture of how a model organizes information. Correlated predictors, nonlinear interactions, and data sparsity all complicate interpretation, so multiple perspectives are often more reliable than a single metric.

The practical examples in this chapter used a random forest on the Boston housing data, but the principles extend naturally to other model classes and application domains.

13.11 Looking Ahead

The following chapters develop each interpretability component in greater technical depth. We begin with feature importance measures, exploring both model specific and model agnostic approaches. We then examine partial dependence and related profile methods that characterize functional relationships. Finally, we introduce Shapley based decompositions, which provide a principled framework for attributing predictions to individual features. Together, these tools offer a more complete descriptive account of how complex models organize information in tabular data.

14 Feature Importance and Variable Selection

14.1 Introduction: Ranking Predictors in Complex Models

In the previous chapter, we introduced a general map of interpretable machine learning and distinguished global from local explanations. Feature importance is often the first global summary we compute, because it gives a compact answer to a central descriptive question, which predictors carry the strongest signal for model predictions.

At the same time, feature importance is not a single quantity with a universal interpretation. Different methods encode different perturbations, assumptions, and notions of contribution. For descriptive analysis, this is not a limitation as much as a reminder that interpretation is comparative and contextual.

This chapter develops feature importance from that perspective. We review common importance definitions, compute them on reproducible examples, and connect ranking to variable selection. The goal is to establish a careful workflow that remains useful for communication while respecting technical nuance.

14.2 Why Feature Importance Matters in Descriptive Work

When models include nonlinearities and interactions, coefficient style interpretation is often unavailable or incomplete. Importance summaries help us recover structure by organizing predictors into an approximate relevance ordering.

A useful ranking can support several descriptive tasks:

- prioritizing variables for deeper inspection,
- comparing model behavior across datasets or time periods,
- identifying redundant predictors before simplification,
- communicating model focus to collaborators.

These uses are practical, but they are not causal claims. If an importance score is high, we learn that the model relied on that variable for prediction under the observed data distribution. We do not learn that intervening on the variable would necessarily change the outcome.

14.3 Two Broad Families of Importance Measures

In applied tabular analysis, two families appear frequently.

14.3.1 Impurity Based Importance (Model Specific)

Tree based models evaluate splits by impurity reduction. In regression trees, impurity is commonly measured by residual sum of squares. In classification trees, impurity is often measured by Gini or entropy style criteria. Aggregating split improvements across trees gives an internal importance score.

For a feature X_j, a generic form is (Breiman 2001):

$$I_j^{imp} = \sum_{t=1}^{T} \sum_{s \in S_{j,t}} \Delta i_{s,t},$$

where T is the number of trees, $S_{j,t}$ is the set of splits using feature j in tree t, and $\Delta i_{s,t}$ is impurity decrease at split s.

These scores are efficient to compute, but they can favor variables with many potential split points, and they can distribute credit unevenly when predictors are correlated.

14.3.2 Permutation Importance (Mostly Model Agnostic)

Permutation importance asks a different question. If we randomly shuffle a feature and break its link to the outcome while preserving its marginal distribution, how much does prediction error increase?

As discussed in the previous chapter, the population level target can be written as a difference in expected loss before and after permutation. Here we emphasize the empirical version that we estimate in practice on a dataset of size n (A. Fisher, Rudin, and Dominici 2019):

$$\hat{I}_j^{perm} = \frac{1}{n} \sum_{i=1}^{n} L\left(y_i, \hat{f}(x_{i,-j}, \pi(x_{i,j}))\right) - \frac{1}{n} \sum_{i=1}^{n} L\left(y_i, \hat{f}(x_i)\right).$$

If shuffling X_j leads to a substantial error increase, the model appears to rely on that feature. This definition is often more comparable across model classes, though correlation can still dilute or redistribute signal.

14.4 Running Example: Random Forest on Boston Housing

We continue with a familiar setup from earlier chapters and fit a random forest for `medv` on a moderate set of predictors.

```
data(Boston, package = "MASS")

set.seed(123)
idx <- sample(seq_len(nrow(Boston)), size = 0.7 * nrow(Boston))
train <- Boston[idx, ]
test <- Boston[-idx, ]

predictors <- c("lstat", "rm", "ptratio", "indus", "nox", "crim")
rf_formula <- as.formula(paste("medv ~", paste(predictors, collapse = " + ")))

set.seed(123)
rf_fit <- randomForest(
    rf_formula,
    data = train,
    ntree = 500,
    mtry = 2,
    importance = TRUE
)

rf_pred <- predict(rf_fit, newdata = test)
rf_rmse <- sqrt(mean((test$medv - rf_pred)^2))

rf_rmse
```

```
[1] 3.5575
```

This test RMSE provides context for interpretation. Importance summaries from models with weak predictive signal are often unstable, so predictive adequacy and interpretability should be read together.

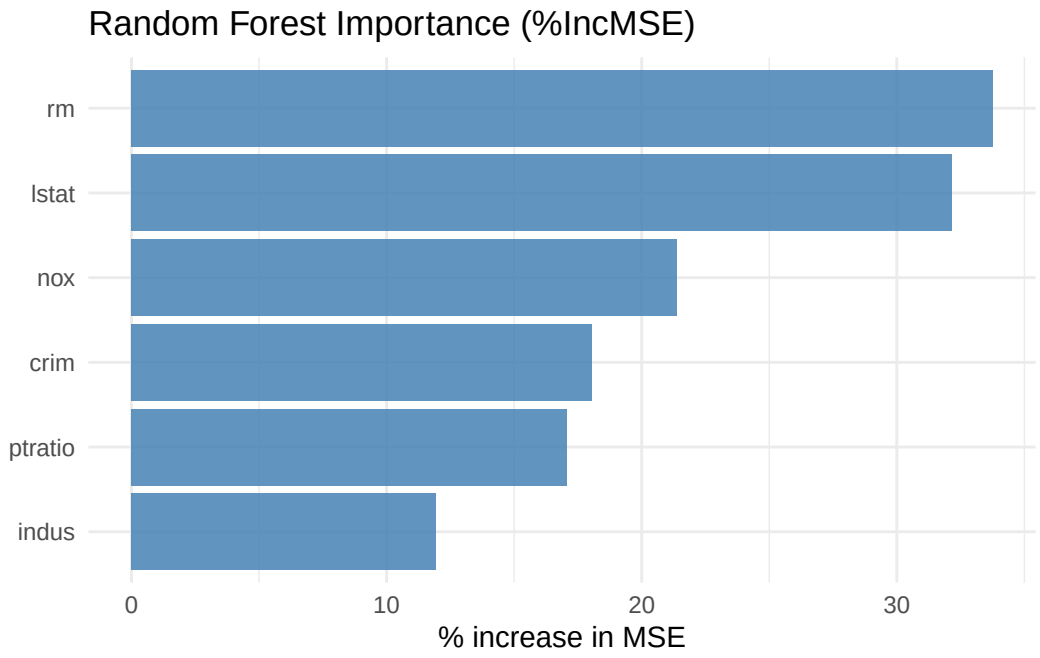
14.5 Model Specific Importance from the Forest Object

The `randomForest` object provides two common regression importance summaries, `%IncMSE` and `IncNodePurity`. We collect both to compare rankings.

Variable	% increase in MSE	Increase in node purity
rm	33.752	8,924.274
lstat	32.154	9,536.612
nox	21.368	2,615.381
crim	18.043	2,590.015
ptratio	17.088	2,735.910
indus	11.933	1,822.111

```
rf_importance <- importance(rf_fit)

imp_tbl <- tibble::tibble(
    variable = rownames(rf_importance),
    inc_mse = rf_importance[, "%IncMSE"],
    inc_node_purity = rf_importance[, "IncNodePurity"]
) |>
    arrange(desc(inc_mse))

imp_tbl |>
    gt() |>
    cols_label(
        variable = "Variable",
        inc_mse = "% increase in MSE",
        inc_node_purity = "Increase in node purity"
    ) |>
    fmt_number(columns = c(inc_mse, inc_node_purity), decimals = 3)
```

```
imp_tbl |>
    ggplot(aes(x = reorder(variable, inc_mse), y = inc_mse)) +
    geom_col(fill = "steelblue", alpha = 0.85) +
    coord_flip() +
    labs(
        title = "Random Forest Importance (%IncMSE)",
        x = NULL,
        y = "% increase in MSE"
    ) +
    theme_minimal()
```



This ranking is often informative, but it should not be treated as a fixed truth. Small rank changes can occur with different seeds, data splits, or hyperparameters.

14.6 Permutation Importance on Held Out Data

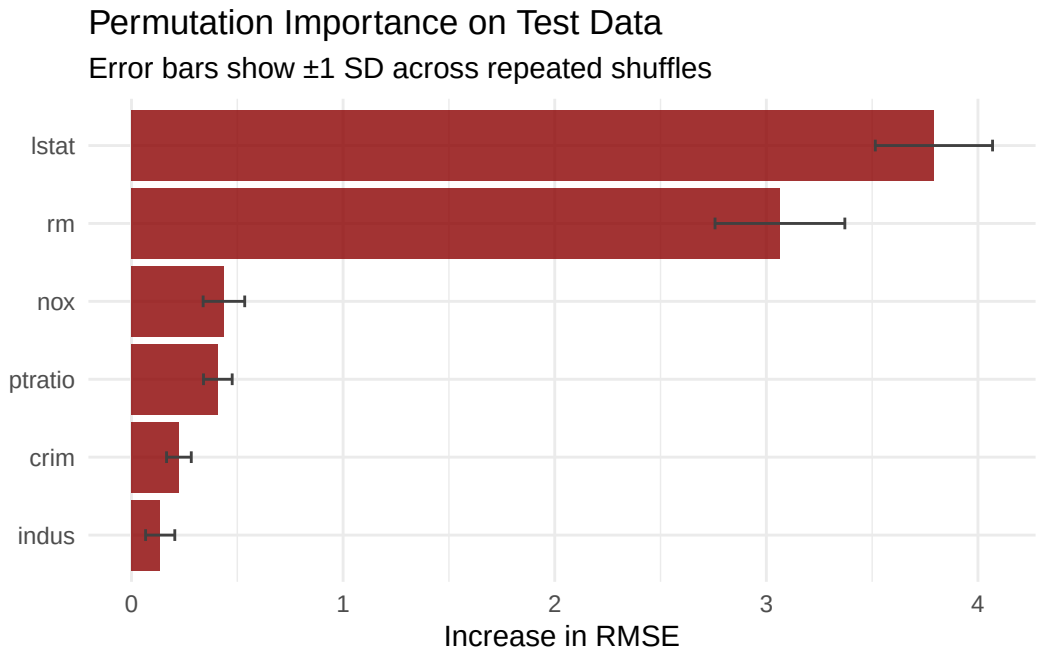
To complement model specific scores, we compute permutation importance directly on the test set with repeated shuffles.

```
perm_importance <- function(model, data, outcome, vars, n_repeats = 12, seed = 123) {
    set.seed(seed)

    base_pred <- predict(model, newdata = data)
    base_rmse <- sqrt(mean((data[[outcome]] - base_pred)^2))

    results <- lapply(vars, function(v) {
        deltas <- numeric(n_repeats)

        for (b in seq_len(n_repeats)) {
            perm_data <- data
            perm_data[[v]] <- sample(perm_data[[v]])
            perm_pred <- predict(model, newdata = perm_data)
            perm_rmse <- sqrt(mean((data[[outcome]] - perm_pred)^2))
            deltas[b] <- perm_rmse - base_rmse
```

Variable	Mean increase in RMSE	SD across permutations
lstat	3.792	0.277
rm	3.064	0.307
nox	0.437	0.098
ptratio	0.408	0.068
crim	0.224	0.058
indus	0.136	0.069

```
        }

        tibble::tibble(
            variable = v,
            mean_delta_rmse = mean(deltas),
            sd_delta_rmse = sd(deltas)
        )
    })

    dplyr::bind_rows(results) |>
        arrange(desc(mean_delta_rmse))
}

perm_imp <- perm_importance(
    model = rf_fit,
    data = test,
    outcome = "medv",
    vars = predictors,
    n_repeats = 15,
    seed = 123
)

perm_imp |>
    gt() |>
    cols_label(
        variable = "Variable",
        mean_delta_rmse = "Mean increase in RMSE",
        sd_delta_rmse = "SD across permutations"
    ) |>
    fmt_number(columns = c(mean_delta_rmse, sd_delta_rmse), decimals = 3)
```

```r
perm_imp |>
    ggplot(aes(x = reorder(variable, mean_delta_rmse), y = mean_delta_rmse)) +
    geom_col(fill = "darkred", alpha = 0.80) +
    geom_errorbar(
        aes(
            ymin = mean_delta_rmse - sd_delta_rmse,
            ymax = mean_delta_rmse + sd_delta_rmse
        ),
        width = 0.15,
        color = "gray25"
    ) +
    coord_flip() +
    labs(
        title = "Permutation Importance on Test Data",
        subtitle = "Error bars show ±1 SD across repeated shuffles",
        x = NULL,
        y = "Increase in RMSE"
    ) +
    theme_minimal()
```



The dispersion bars help us avoid over interpreting very close ranks. If two variables have overlapping uncertainty ranges, it can be preferable to report them as similarly important rather than forcing a strict order.

Variable	% increase in MSE	Permutation ΔRMSE	Rank (%IncMSE)	Rank (Permutation)
lstat	32.154	3.792	2	1
rm	33.752	3.064	1	2
nox	21.368	0.437	3	3
ptratio	17.088	0.408	5	4
crim	18.043	0.224	4	5
indus	11.933	0.136	6	6

14.7 Comparing Importance Definitions

A practical check is to compare rank agreement between methods.

```
rank_compare <- imp_tbl |>
    dplyr::select(variable, inc_mse) |>
    left_join(perm_imp |> dplyr::select(variable, mean_delta_rmse), by = "variable") |>
    mutate(
        rank_inc_mse = rank(-inc_mse, ties.method = "average"),
        rank_perm = rank(-mean_delta_rmse, ties.method = "average")
    )

rank_cor <- cor(rank_compare$rank_inc_mse, rank_compare$rank_perm,
                method = "spearman")

rank_compare |>
    arrange(rank_perm) |>
    gt() |>
    cols_label(
        variable = "Variable",
        inc_mse = "% increase in MSE",
        mean_delta_rmse = "Permutation ΔRMSE",
        rank_inc_mse = "Rank (%IncMSE)",
        rank_perm = "Rank (Permutation)"
    ) |>
    fmt_number(columns = c(inc_mse, mean_delta_rmse), decimals = 3)
```

```
rank_cor
```

```
[1] 0.8857143
```

lstat	rm	ptratio	indus	nox	crim
1.00	-0.60	0.40	0.60	0.60	0.46
-0.60	1.00	-0.39	-0.41	-0.31	-0.22
0.40	-0.39	1.00	0.36	0.20	0.29
0.60	-0.41	0.36	1.00	0.78	0.41
0.60	-0.31	0.20	0.78	1.00	0.42
0.46	-0.22	0.29	0.41	0.42	1.00

High rank agreement can increase confidence that a variable is consistently influential under more than one definition. Disagreement is also informative, because it often signals correlation structure or interaction effects that deserve deeper inspection.

14.8 Correlated Predictors and Shared Signal

When predictors overlap strongly, attribution becomes ambiguous. We can diagnose this by inspecting the predictor correlation matrix.

```
cor_mat <- cor(train[, predictors])

cor_mat |>
    as.data.frame() |>
    round(2) |>
    gt()
```

If two variables are strongly correlated, shuffling one may have a limited effect because the other can partially recover the same information. In that case, a low marginal importance does not necessarily mean substantive irrelevance.

For communication, it is often helpful to pair feature importance with grouped interpretation, for example, discussing signal carried by related predictor families rather than only isolated variables.

14.9 From Ranking to Variable Selection

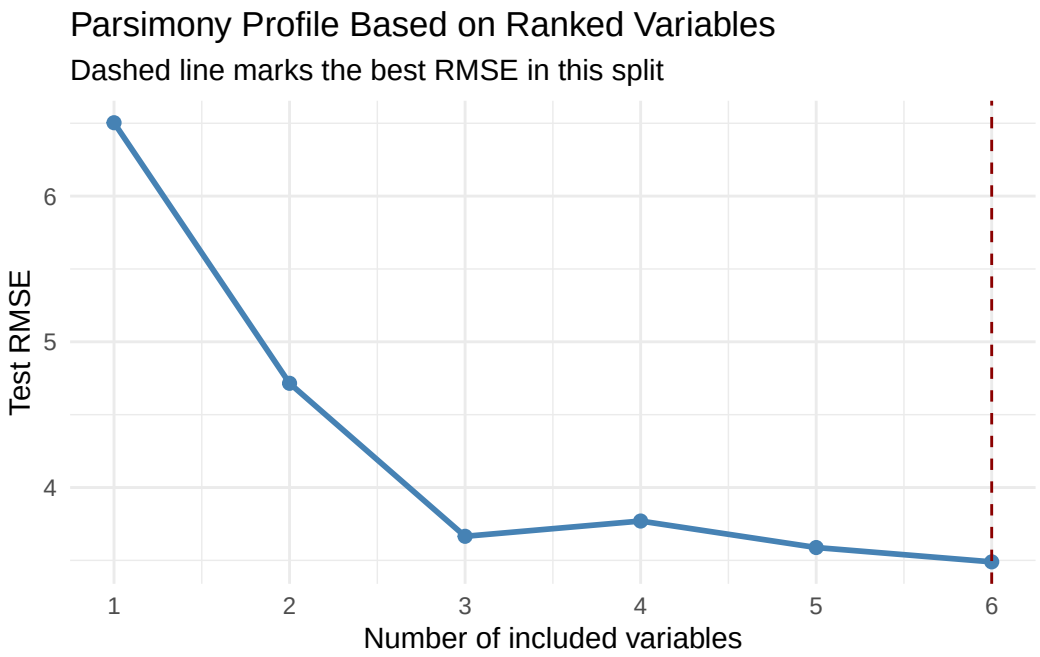
Importance ranking often motivates a second question, how many predictors do we need for a stable descriptive model? A simple approach is to order variables by importance, fit models with the top k variables, and inspect out of sample error as k grows.

```
ordered_vars <- perm_imp$variable

subset_perf <- lapply(seq_along(ordered_vars), function(k) {
    vars_k <- ordered_vars[1:k]
    formula_k <- as.formula(paste("medv ~", paste(vars_k, collapse = " + ")))

    set.seed(200 + k)
    fit_k <- randomForest(
        formula_k,
        data = train,
        ntree = 400,
        mtry = max(1, floor(sqrt(k)))
    )

    pred_k <- predict(fit_k, newdata = test)
    rmse_k <- sqrt(mean((test$medv - pred_k)^2))

    tibble::tibble(
        k = k,
        rmse = rmse_k,
        variables = paste(vars_k, collapse = ", ")
    )
}) |>
    bind_rows()

subset_perf |>
    gt() |>
    cols_label(
        k = "Number of top variables",
        rmse = "Test RMSE",
        variables = "Variables included"
    ) |>
    fmt_number(columns = rmse, decimals = 3)
```

```
best_k <- subset_perf$k[which.min(subset_perf$rmse)]

subset_perf |>
    ggplot(aes(x = k, y = rmse)) +
    geom_line(color = "steelblue", linewidth = 1) +
    geom_point(color = "steelblue", size = 2) +
    geom_vline(xintercept = best_k, linetype = "dashed", color = "darkred") +
```

Number of top variables	Test RMSE	Variables included
1	6.504	lstat
2	4.715	lstat, rm
3	3.665	lstat, rm, nox
4	3.770	lstat, rm, nox, ptratio
5	3.588	lstat, rm, nox, ptratio, crim
6	3.490	lstat, rm, nox, ptratio, crim, indus

```
    scale_x_continuous(breaks = seq_len(length(ordered_vars))) +
    labs(
        title = "Parsimony Profile Based on Ranked Variables",
        subtitle = "Dashed line marks the best RMSE in this split",
        x = "Number of included variables",
        y = "Test RMSE"
    ) +
    theme_minimal()
```



This profile is intentionally pragmatic. It does not claim to identify a uniquely correct subset, but it helps us evaluate the trade off between parsimony and predictive adequacy in a transparent way.

14.10 A Brief Classification Illustration

Feature importance behaves similarly for classification, although impurity and loss definitions differ. We use the Pima diabetes data for a compact illustration.

```
data(Pima.tr, package = "MASS")
data(Pima.te, package = "MASS")

set.seed(321)
rf_cls <- randomForest(
    type ~ npreg + glu + bp + skin + bmi + ped + age,
    data = Pima.tr,
    ntree = 500,
    mtry = 3,
    importance = TRUE
)

cls_imp <- importance(rf_cls)

cls_tbl <- tibble::tibble(
    variable = rownames(cls_imp),
    mean_decrease_accuracy = cls_imp[, "MeanDecreaseAccuracy"],
    mean_decrease_gini = cls_imp[, "MeanDecreaseGini"]
) |>
    arrange(desc(mean_decrease_accuracy))

cls_tbl |>
    gt() |>
    cols_label(
        variable = "Variable",
        mean_decrease_accuracy = "Mean decrease accuracy",
        mean_decrease_gini = "Mean decrease Gini"
    ) |>
    fmt_number(columns = c(mean_decrease_accuracy, mean_decrease_gini), decimals = 3)
```

Even in this short example, ranking should be interpreted jointly with domain context and class balance considerations. Importance identifies model reliance, not clinical causality.

14.11 Good Practice for Reporting Feature Importance

A careful descriptive report often benefits from a short checklist:

Variable	Mean decrease accuracy	Mean decrease Gini
glu	20.921	24.081
age	14.010	14.625
bmi	10.670	12.735
ped	9.824	13.883
npreg	8.470	8.622
skin	0.812	8.212
bp	-0.798	7.298

- report which importance definition was used,
- indicate whether evaluation is in sample, OOB, or held out,
- include some notion of variability (for example repeated permutations),
- note potential correlation induced dilution,
- avoid causal language unless causal identification is justified separately.

These practices do not eliminate ambiguity, but they make interpretation more transparent and reproducible.

14.12 Summary and Key Takeaways

- Feature importance is a global explanatory tool that ranks model reliance on predictors.
- Impurity based and permutation based scores answer related but distinct questions, and both can be informative.
- Correlated predictors can redistribute importance, so low marginal importance is not always low substantive relevance.
- Ranking can guide variable selection through parsimony profiles that compare error across top k subsets.
- Importance is descriptive unless paired with a separate causal design.

14.13 Looking Ahead

Feature importance tells us which predictors matter most for model performance, but it does not describe the functional shape of those effects. The next chapter therefore focuses on partial dependence and related profile methods, where we examine how predictions vary as individual predictors change across their observed ranges.

15 Partial Dependence and Individual Conditional Expectation

15.1 Introduction: Moving from Importance to Functional Interpretation

In the previous chapter, we focused on feature importance as a global ranking device. That perspective helps us identify which predictors the model appears to rely on most. It does not yet tell us how predictions change as a predictor varies.

Partial dependence and individual conditional expectation (ICE) plots address this next layer of interpretation. Rather than asking only which variables matter, we ask how model predictions respond across the range of one or more predictors. This shift is central in descriptive work with nonlinear models, because it reveals shape, heterogeneity, and potential interaction structure.

This chapter develops both methods from first principles, then implements them in reproducible R code on a tree based model. We emphasize practical interpretation, known limitations, and communication choices that keep conclusions technically accurate.

15.2 Why Profile Based Explanations Matter

Feature rankings are compact and useful, but they collapse functional detail. Two predictors may receive similar importance scores while having very different predictive patterns, for example, one roughly monotonic and another strongly nonlinear.

Profile based summaries help with questions such as:

- how predictions evolve over the observed range of a predictor,
- whether effects appear approximately linear, threshold-like, or saturating,
- whether different observations exhibit similar or divergent response profiles.

These are descriptive statements about fitted model behavior under the empirical data distribution. As discussed earlier in the book, they should not be interpreted as causal response functions.

15.3 Formal Definitions

Let a fitted predictor be denoted by $\hat{f}(x)$, and partition the feature vector as (x_s, x_C), where x_s is the focal feature (or feature subset) and x_C collects the remaining features.

15.3.1 Partial Dependence Function

Building on the introductory formulation presented earlier in the book, we now write partial dependence for a subset of focal features S (with complement C):

$$f_S^{PD}(x_S) = \mathbb{E}_{X_C}\left[\hat{f}(x_S, X_C)\right].$$

In practice, this expectation is estimated by Monte Carlo averaging over observed rows (J. H. Friedman 2001):

$$\hat{f}_S^{PD}(x_S) = \frac{1}{n}\sum_{i=1}^{n} \hat{f}(x_S, x_C^{(i)}).$$

This notation accommodates one dimensional and multi dimensional profiles in a unified way. Operationally, we evaluate the estimator on a grid of x_S values and average predictions at each grid point.

15.3.2 Individual Conditional Expectation Curves

For observation i, the corresponding ICE curve is (Goldstein et al. 2015):

$$\hat{f}_{ICE}^{(i)}(x_s) = \hat{f}(x_s, x_{i,C}).$$

PD is the average of ICE curves over observations. This relationship is important, because it shows what PD can hide. If ICE curves differ substantially in shape, the average can mask heterogeneity.

15.3.3 Centered ICE Curves

A common variant is centered ICE (c-ICE), where each ICE curve is shifted to a common reference value x_s^{ref} (Goldstein et al. 2015):

$$\hat{f}_{cICE}^{(i)}(x_s) = \hat{f}_{ICE}^{(i)}(x_s) - \hat{f}_{ICE}^{(i)}(x_s^{ref}).$$

Centering removes vertical offsets and makes slope differences easier to inspect.

15.4 Running Example: Random Forest on Boston Housing

To maintain continuity with the interpretability sequence, we continue with a random forest model for `medv` using a compact predictor set.

```
data(Boston, package = "MASS")

set.seed(123)
idx <- sample(seq_len(nrow(Boston)), size = 0.7 * nrow(Boston))
train <- Boston[idx, ]
test <- Boston[-idx, ]

predictors <- c("lstat", "rm", "ptratio", "indus", "nox", "crim")
rf_formula <- as.formula(paste("medv ~", paste(predictors, collapse = " + ")))

set.seed(123)
rf_fit <- randomForest(
    rf_formula,
    data = train,
    ntree = 500,
    mtry = 2,
    importance = TRUE
)

rf_pred <- predict(rf_fit, newdata = test)
rf_rmse <- sqrt(mean((test$medv - rf_pred)^2))

rf_rmse
```

```
[1] 3.5575
```

The predictive error gives context for interpretation. Profile based explanations are most informative when the model captures a meaningful share of signal.

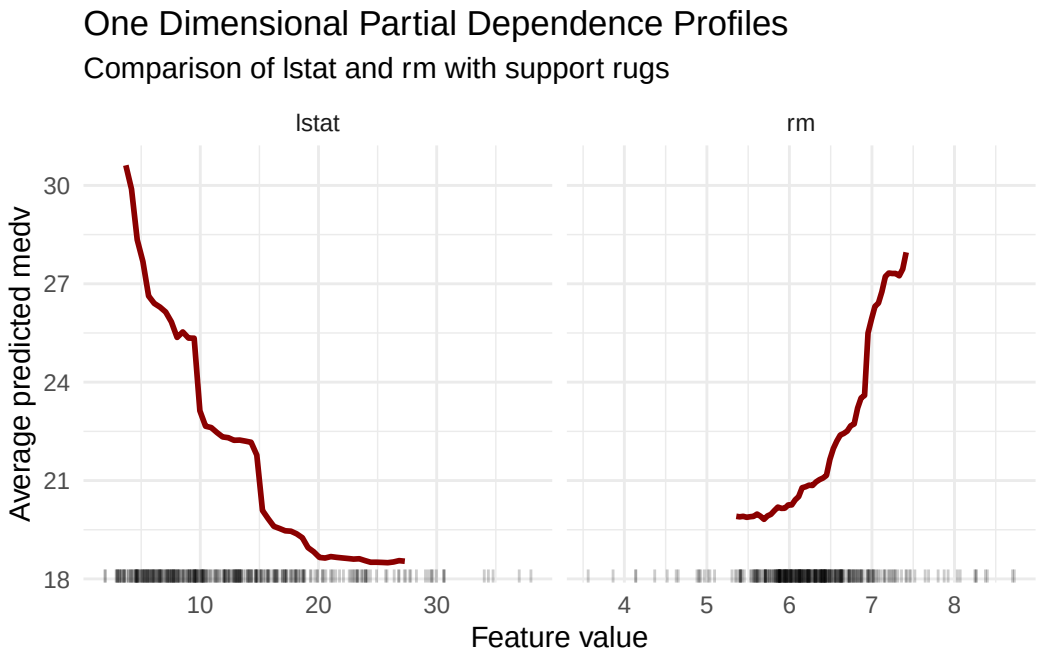
15.5 Computing One Dimensional Partial Dependence

As introduced earlier in the book (see Chapter 13), one dimensional PD is obtained by averaging predictions while varying one focal feature over a grid. Here we move one step further by comparing PD profiles for two predictors, `lstat` and `rm`, to emphasize that profile interpretation is inherently comparative.

```r
partial_dependence <- function(model, data, variable, grid_values) {
    pd_values <- sapply(grid_values, function(g) {
        new_data <- data
        new_data[[variable]] <- g
        mean(predict(model, newdata = new_data))
    })

    tibble::tibble(
        value = grid_values,
        pd = pd_values,
        variable = variable
    )
}

lstat_grid <- seq(
    from = as.numeric(quantile(train$lstat, 0.05)),
    to = as.numeric(quantile(train$lstat, 0.95)),
    length.out = 50
)

rm_grid <- seq(
    from = as.numeric(quantile(train$rm, 0.05)),
    to = as.numeric(quantile(train$rm, 0.95)),
    length.out = 50
)

pd_lstat <- partial_dependence(
    model = rf_fit,
    data = train,
    variable = "lstat",
    grid_values = lstat_grid
)

pd_rm <- partial_dependence(
    model = rf_fit,
    data = train,
    variable = "rm",
    grid_values = rm_grid
)

pd_profiles <- bind_rows(
    pd_lstat,
```

```
    pd_rm
)

rug_profiles <- bind_rows(
    tibble(value = train$lstat, variable = "lstat"),
    tibble(value = train$rm, variable = "rm")
)
```

```
pd_profiles |>
    ggplot(aes(x = value, y = pd)) +
    geom_line(color = "darkred", linewidth = 1) +
    geom_rug(
        data = rug_profiles,
        aes(x = value, y = NULL),
        inherit.aes = FALSE,
        sides = "b",
        alpha = 0.20
    ) +
    facet_wrap(~ variable, scales = "free_x") +
    labs(
        title = "One Dimensional Partial Dependence Profiles",
        subtitle = "Comparison of lstat and rm with support rugs",
        x = "Feature value",
        y = "Average predicted medv"
    ) +
    theme_minimal()
```



The two panels highlight complementary directions of association. As `lstat` increases, the average prediction decreases, while larger `rm` values are associated with higher predicted `medv`. In both cases, the curvature suggests that effects are not well summarized by a single linear slope.

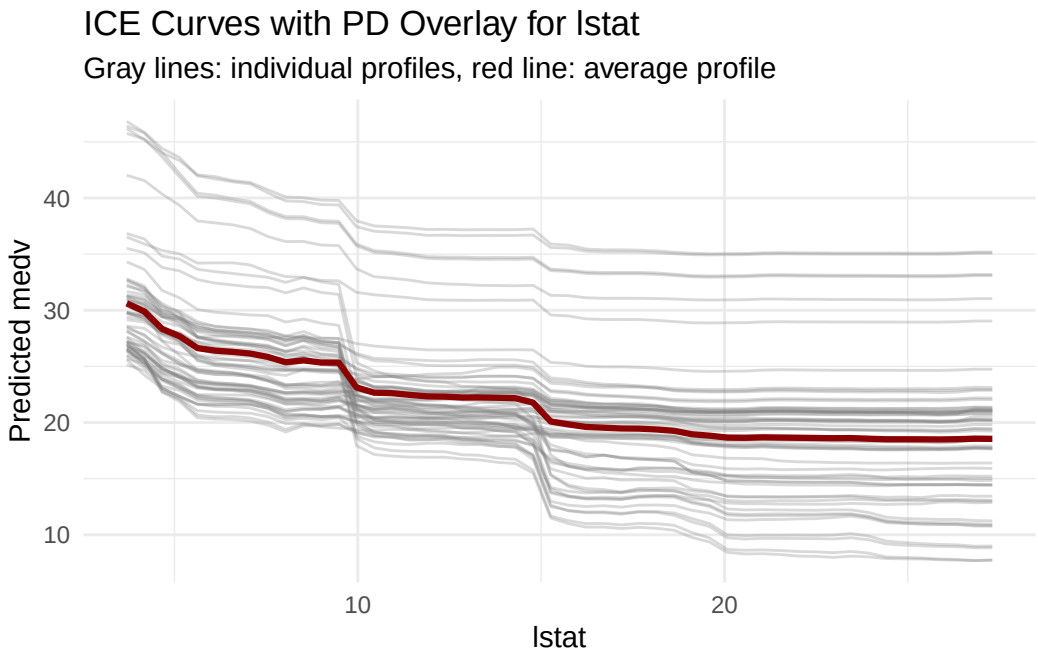
15.6 Individual Conditional Expectation and Heterogeneity

To inspect whether this average pattern is broadly shared, we compute ICE curves for a subsample of observations.

```
ice_curves <- function(model, data, variable, grid_values, ids = NULL) {
    if (is.null(ids)) {
        ids <- seq_len(nrow(data))
    }

    ice_list <- lapply(ids, function(i) {
        obs <- data[i, , drop = FALSE]
        preds <- sapply(grid_values, function(g) {
            new_obs <- obs
            new_obs[[variable]] <- g
            as.numeric(predict(model, newdata = new_obs))
        })
```

```
        tibble::tibble(
            obs_id = i,
            value = grid_values,
            ice = preds
        )
    })

    dplyr::bind_rows(ice_list)
}

set.seed(321)
ice_ids <- sample(seq_len(nrow(train)), size = 60)

ice_lstat <- ice_curves(
    model = rf_fit,
    data = train,
    variable = "lstat",
    grid_values = lstat_grid,
    ids = ice_ids
)
```

```
ice_lstat |>
    ggplot(aes(x = value, y = ice, group = obs_id)) +
    geom_line(color = "gray50", alpha = 0.30) +
    geom_line(
        data = pd_lstat,
        aes(x = value, y = pd),
        inherit.aes = FALSE,
        color = "darkred",
        linewidth = 1.1
    ) +
    labs(
        title = "ICE Curves with PD Overlay for lstat",
        subtitle = "Gray lines: individual profiles, red line: average profile",
        x = "lstat",
        y = "Predicted medv"
    ) +
    theme_minimal()
```



The spread of gray lines shows that observations do not all respond identically. PD remains useful as a summary, but ICE reveals that a single average curve can hide meaningful variation.

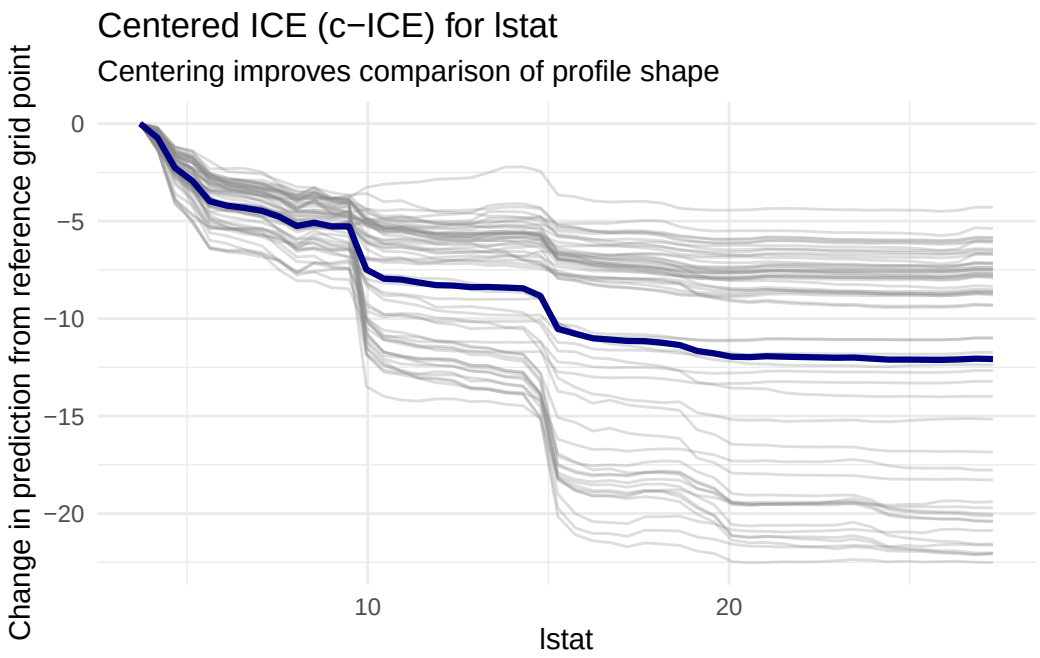
15.7 Centered ICE for Shape Comparison

Vertical differences between ICE curves often reflect baseline prediction level rather than slope behavior. Centering helps isolate shape differences.

```
ice_centered <- ice_lstat |>
    group_by(obs_id) |>
    mutate(ice_c = ice - first(ice)) |>
    ungroup()

pd_centered <- pd_lstat |>
    mutate(pd_c = pd - first(pd))
```

```
ice_centered |>
    ggplot(aes(x = value, y = ice_c, group = obs_id)) +
    geom_line(color = "gray55", alpha = 0.30) +
    geom_line(
        data = pd_centered,
        aes(x = value, y = pd_c),
```

```
        inherit.aes = FALSE,
        color = "navy",
        linewidth = 1.1
    ) +
    labs(
        title = "Centered ICE (c-ICE) for lstat",
        subtitle = "Centering improves comparison of profile shape",
        x = "lstat",
        y = "Change in prediction from reference grid point"
    ) +
    theme_minimal()
```



In this representation, we can compare profile slopes more directly. Divergence in c-ICE trajectories suggests interaction effects or observation specific nonlinearities.

15.8 Quantifying ICE Dispersion Along the Grid

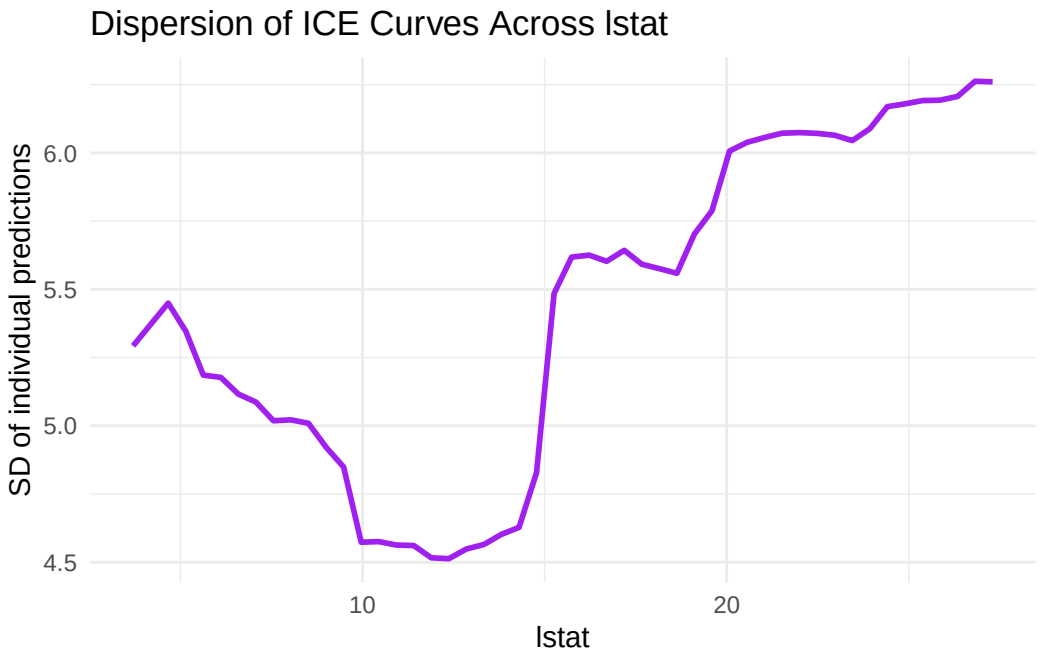
Visual inspection is important, and a small numeric summary can also help. We compute the standard deviation of ICE predictions at each grid point.

```
ice_dispersion <- ice_lstat |>
    group_by(value) |>
```

Minimum SD	Median SD	Maximum SD
4.513	5.522	6.263

```
    summarise(sd_ice = sd(ice), .groups = "drop")

dispersion_summary <- ice_dispersion |>
    summarise(
        min_sd = min(sd_ice),
        median_sd = median(sd_ice),
        max_sd = max(sd_ice)
    )

dispersion_summary |>
    gt() |>
    cols_label(
        min_sd = "Minimum SD",
        median_sd = "Median SD",
        max_sd = "Maximum SD"
    ) |>
    fmt_number(columns = everything(), decimals = 3)
```

```
ice_dispersion |>
    ggplot(aes(x = value, y = sd_ice)) +
    geom_line(color = "purple", linewidth = 1) +
    labs(
        title = "Dispersion of ICE Curves Across lstat",
        x = "lstat",
        y = "SD of individual predictions"
    ) +
    theme_minimal()
```



Higher dispersion regions indicate where individual profiles differ more strongly. These zones are often where interaction structure is more pronounced.

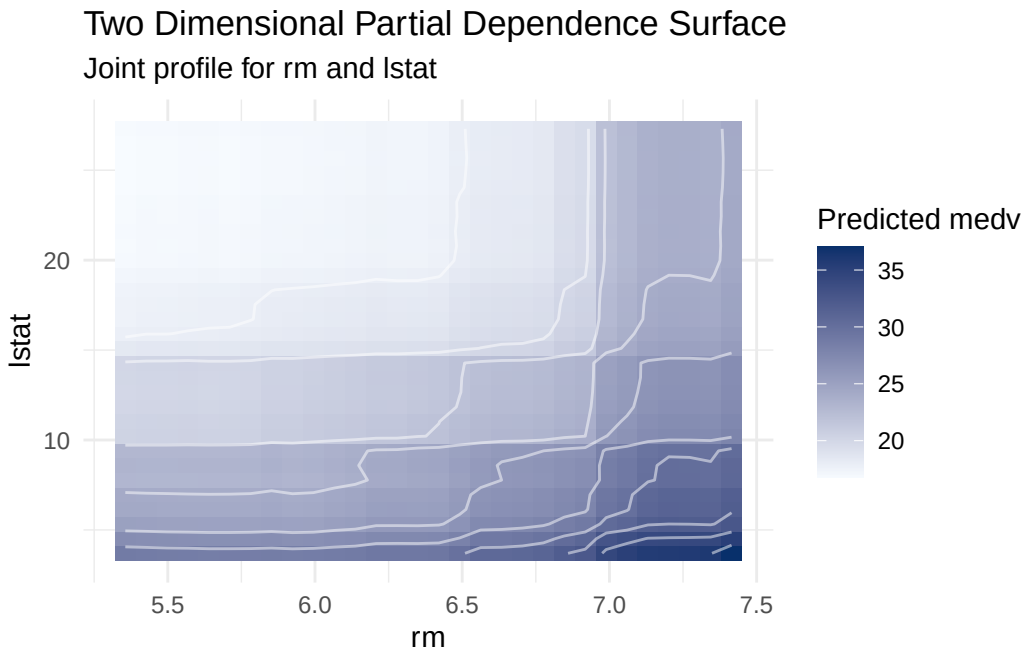
15.9 Two Dimensional Partial Dependence

When interactions are suspected between two predictors, a two dimensional PD surface can provide a useful global view.

```
partial_dependence_2d <- function(model, data, var1, var2, grid1, grid2) {
    grid <- expand.grid(
        value1 = grid1,
        value2 = grid2,
        KEEP.OUT.ATTRS = FALSE,
        stringsAsFactors = FALSE
    )

    pd_vals <- mapply(function(g1, g2) {
        new_data <- data
        new_data[[var1]] <- g1
        new_data[[var2]] <- g2
        mean(predict(model, newdata = new_data))
    }, grid$value1, grid$value2)
```

```
    tibble::tibble(
        value1 = grid$value1,
        value2 = grid$value2,
        pd = pd_vals
    )
}

rm_grid <- seq(
    from = as.numeric(quantile(train$rm, 0.05)),
    to = as.numeric(quantile(train$rm, 0.95)),
    length.out = 30
)

lstat_grid_2d <- seq(
    from = as.numeric(quantile(train$lstat, 0.05)),
    to = as.numeric(quantile(train$lstat, 0.95)),
    length.out = 30
)

pd_rm_lstat <- partial_dependence_2d(
    model = rf_fit,
    data = train,
    var1 = "rm",
    var2 = "lstat",
    grid1 = rm_grid,
    grid2 = lstat_grid_2d
)
```

```
pd_rm_lstat |>
    ggplot(aes(x = value1, y = value2, z = pd)) +
    geom_tile(aes(fill = pd)) +
    geom_contour(color = "white", alpha = 0.5) +
    scale_fill_gradient(low = "#f7fbff", high = "#08306b") +
    labs(
        title = "Two Dimensional Partial Dependence Surface",
        subtitle = "Joint profile for rm and lstat",
        x = "rm",
        y = "lstat",
        fill = "Predicted medv"
    ) +
    theme_minimal()
```



This surface supports a joint reading. Predictions tend to increase with larger `rm` and decrease with larger `lstat`, and contour geometry helps us assess whether the combined pattern is approximately additive or more interaction-like.

15.10 Practical Limitations and Interpretation Cautions

PD and ICE are powerful, but they rely on assumptions that deserve explicit attention.

- **Correlation among predictors** can create unrealistic combinations when we vary one feature while keeping others fixed from observed rows. This can affect profile shape.
- **Support awareness** matters. Curves are most reliable in regions with adequate data density, which is why rugs or density overlays are useful.
- **Averaging can obscure heterogeneity**. PD should be interpreted jointly with ICE whenever interaction effects are plausible.
- **Descriptive, not interventional**. These curves describe model predictions under observed covariate structure, not causal effects of manipulating predictors.

In practice, we often combine profile methods with other interpretability summaries, such as permutation importance and local attributions, to obtain a more balanced account.

15.11 Summary and Key Takeaways

Partial dependence and ICE add functional detail to model interpretation. Several points from this chapter are central:

- **Partial dependence is a global average profile** over non focal predictors.
- **ICE exposes observation level variation** that PD can hide.
- **Centered ICE improves shape comparison** by removing baseline offsets.
- **Two dimensional PD helps inspect joint structure** and potential interactions between predictors.
- **Interpretation quality depends on data support and dependence structure**, so profile reading should be paired with caution and complementary diagnostics.

Together, these tools provide a richer descriptive view of model behavior than variable ranking alone.

15.12 Looking Ahead

The next chapter turns to Shapley value based explanations. Whereas partial dependence and ICE focus on functional profiles across feature values, Shapley methods decompose individual predictions into additive feature contributions. This gives us a complementary local perspective, and helps connect global patterns to case specific explanations.

16 Shapley Values and Additive Explanations

16.1 Introduction: From Functional Profiles to Local Additive Attribution

In the previous chapter, we examined partial dependence and ICE curves as tools for reading model behavior across feature values. That perspective is global and profile based, which makes it useful for understanding shape and heterogeneity.

Shapley value explanations address a complementary question. For a specific observation, how can we decompose the model prediction into additive feature contributions in a principled way? This local decomposition is often valuable when we want to connect broad model behavior to case specific interpretation.

In this chapter, we develop Shapley values from their game theoretic definition, discuss practical estimation in tabular settings, and implement a reproducible workflow in R. We also connect local attributions to global summaries, because descriptive practice often moves back and forth between these two levels.

16.2 Why Shapley Values Matter in Descriptive Analysis

In complex predictive models, local explanations help us understand why two observations with different predictions are separated by the model. Additive decompositions are particularly appealing because they map each prediction into interpretable components relative to a reference level.

For descriptive work, this supports several goals:

- explaining individual predictions in a structured and comparable format,
- auditing whether model behavior aligns with domain expectations,
- aggregating local contributions into global relevance summaries,
- communicating model behavior with transparent bookkeeping.

As in earlier interpretability chapters, these are predictive attributions, not causal effects. A positive contribution indicates that the model prediction moves upward relative to a baseline when the feature is incorporated under the chosen attribution definition.

16.3 Shapley Values as a Cooperative Game

Let $M = \{1, \dots, p\}$ denote feature indices, and let $v_x(S)$ be a value function that measures the expected prediction when only features in subset S are known for observation x.

For feature j, the Shapley value is (Shapley 1953; Lundberg and Lee 2017):

$$\phi_j(x) = \sum_{S \subseteq M \setminus \{j\}} \frac{|S|!(p - |S| - 1)!}{p!} \left[v_x(S \cup \{j\}) - v_x(S) \right].$$

The weighting term averages marginal contributions of feature j over all possible feature ordering contexts. This symmetry is a central reason why Shapley values are often used for additive explanation.

Under a given value function definition, Shapley values satisfy familiar axiomatic properties:

- **Efficiency (local accuracy)**, contributions sum to prediction minus baseline,
- **Symmetry**, features with identical marginal roles receive identical credit,
- **Dummy (missingness)**, features with no marginal effect receive zero contribution,
- **Additivity**, attribution is linear across games.

These properties are mathematical guarantees for the specified game. In applications, the practical interpretation still depends on how $v_x(S)$ is defined and estimated.

16.4 Choosing the Value Function in Tabular Data

Two common choices appear in practice.

1. **Marginal (interventional style) value functions**, where unknown features are integrated over a background distribution without conditioning on selected features.
2. **Conditional value functions**, where unknown features are integrated conditionally on selected features.

The marginal approach is often easier to compute and aligns with many software defaults, but it can evaluate feature combinations that are less realistic under strong dependence. The conditional approach can better respect feature dependence, but requires additional modeling assumptions.

To keep the workflow transparent and reproducible, we use a marginal Monte Carlo approximation in this chapter, with explicit discussion of its limits.

16.5 Running Example: Random Forest on Boston Housing

We continue the interpretability sequence with a random forest predicting `medv` from a compact set of predictors.

```
data(Boston, package = "MASS")

set.seed(123)
idx <- sample(seq_len(nrow(Boston)), size = 0.7 * nrow(Boston))
train <- Boston[idx, ]
test <- Boston[-idx, ]

predictors <- c("lstat", "rm", "ptratio", "indus", "nox", "crim")
rf_formula <- as.formula(paste("medv ~", paste(predictors, collapse = " + ")))

set.seed(123)
rf_fit <- randomForest(
    rf_formula,
    data = train,
    ntree = 500,
    mtry = 2,
    importance = TRUE
)

rf_pred <- predict(rf_fit, newdata = test)
rf_rmse <- sqrt(mean((test$medv - rf_pred)^2))

rf_rmse
```

```
[1] 3.5575
```

The RMSE gives context for subsequent interpretation. As discussed earlier in the book, explanatory summaries are typically more informative when the predictive model has useful signal.

16.6 Monte Carlo Approximation of Shapley Values

Exact Shapley values require averaging over all subsets, which scales exponentially with the number of predictors. For tabular models, permutation based Monte Carlo estimators are often used.

The function below estimates Shapley values for one observation by averaging marginal contributions across random feature orderings and random background rows.

```
predict_rf <- function(model, newdata) {
    as.numeric(predict(model, newdata = newdata))
}

shapley_single_mc <- function(model,
                              x,
                              background,
                              features,
                              predict_fn,
                              n_permutations = 300,
                              seed = 123) {
    set.seed(seed)

    contrib <- setNames(rep(0, length(features)), features)

    for (b in seq_len(n_permutations)) {
        perm <- sample(features)
        z <- background[sample(seq_len(nrow(background)), 1), features, drop = FALSE]

        current <- z
        pred_prev <- predict_fn(model, current)

        for (f in perm) {
            current[[f]] <- x[[f]]
            pred_new <- predict_fn(model, current)
            contrib[f] <- contrib[f] + (pred_new - pred_prev)
            pred_prev <- pred_new
        }
    }

    contrib / n_permutations
}
```

Next, we compute local attributions for one test observation and verify approximate local accuracy.

```
background <- train[, predictors, drop = FALSE]
baseline <- mean(predict_rf(rf_fit, background))
```

Quantity	Value
Baseline	22.2935
Prediction	18.2458
Baseline + Sum(phi)	18.6192
Difference	-0.3734

```
obs_id <- 10
x_obs <- test[obs_id, predictors, drop = FALSE]
pred_obs <- predict_rf(rf_fit, x_obs)

phi_obs <- shapley_single_mc(
    model = rf_fit,
    x = x_obs,
    background = background,
    features = predictors,
    predict_fn = predict_rf,
    n_permutations = 400,
    seed = 123
)

check_tbl <- tibble::tibble(
    quantity = c("Baseline", "Prediction", "Baseline + Sum(phi)", "Difference"),
    value = c(
        baseline,
        pred_obs,
        baseline + sum(phi_obs),
        pred_obs - (baseline + sum(phi_obs))
    )
)

check_tbl |>
    gt() |>
    cols_label(
        quantity = "Quantity",
        value = "Value"
    ) |>
    fmt_number(columns = value, decimals = 4)
```

The final row should be near zero, up to Monte Carlo approximation error.

Feature	Shapley contribution
rm	-2.230
ptratio	-1.252
indus	-0.431
crim	0.217
lstat	0.090
nox	-0.069

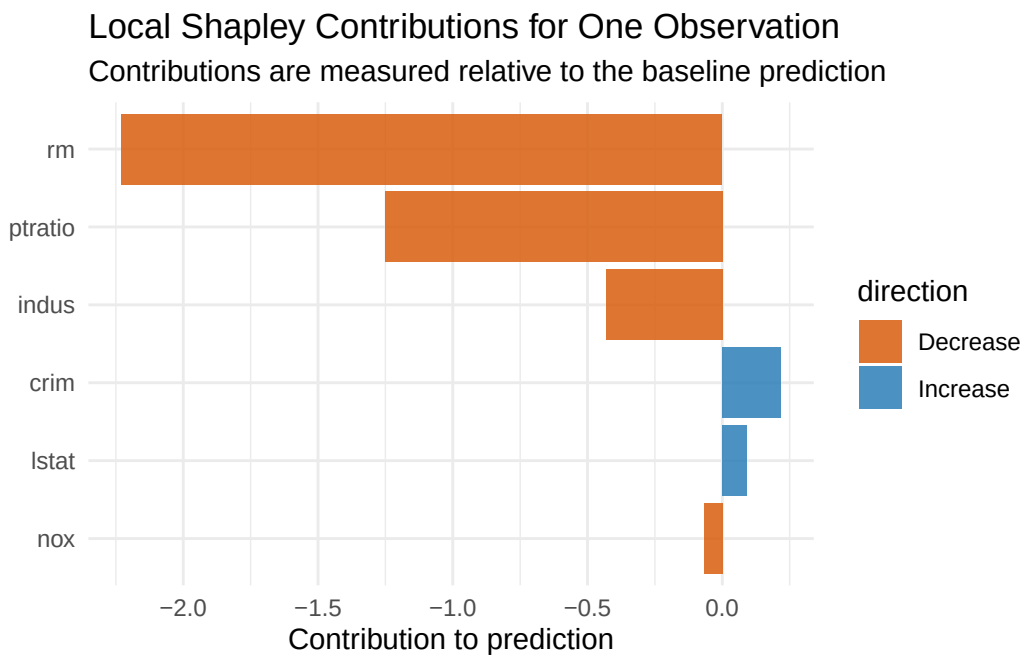
16.7 Visualizing Local Contributions

A local decomposition is often easiest to read when contributions are ordered by absolute magnitude.

```
local_tbl <- tibble::tibble(
    feature = names(phi_obs),
    phi = as.numeric(phi_obs)
) |>
    arrange(desc(abs(phi)))

local_tbl |>
    gt() |>
    cols_label(
        feature = "Feature",
        phi = "Shapley contribution"
    ) |>
    fmt_number(columns = phi, decimals = 3)
```

```
plot_tbl <- local_tbl |>
    mutate(
        direction = ifelse(phi >= 0, "Increase", "Decrease"),
        feature = factor(feature, levels = rev(feature))
    )

plot_tbl |>
    ggplot(aes(x = feature, y = phi, fill = direction)) +
    geom_col(alpha = 0.85) +
    coord_flip() +
    scale_fill_manual(values = c("Increase" = "#2c7fb8", "Decrease" = "#d95f0e")) +
    labs(
        title = "Local Shapley Contributions for One Observation",
```

```
        subtitle = "Contributions are measured relative to the baseline prediction",
        x = NULL,
        y = "Contribution to prediction"
    ) +
    theme_minimal()
```



Positive bars move the prediction above baseline, while negative bars move it below baseline. Reading contributions together with the original feature values helps preserve context.

```
value_context_tbl <- tibble::tibble(
    feature = predictors,
    feature_value = as.numeric(x_obs[1, predictors]),
    shapley = as.numeric(phi_obs[predictors])
)

value_context_tbl |>
    gt() |>
    cols_label(
        feature = "Feature",
        feature_value = "Observed value",
        shapley = "Shapley contribution"
    ) |>
    fmt_number(columns = c(feature_value, shapley), decimals = 3)
```

Feature	Observed value	Shapley contribution
lstat	11.690	0.090
rm	5.456	-2.230
ptratio	21.000	-1.252
indus	8.140	-0.431
nox	0.538	-0.069
crim	0.803	0.217

16.8 From Local to Global: Mean Absolute Shapley

A common global summary is the mean absolute contribution for each feature across many observations (Lundberg and Lee 2017):

$$I_j^{SHAP} = \frac{1}{n} \sum_{i=1}^{n} |\phi_j(x_i)|.$$

This quantity is not a replacement for local analysis, but it provides a useful bridge between per case decomposition and global feature relevance.

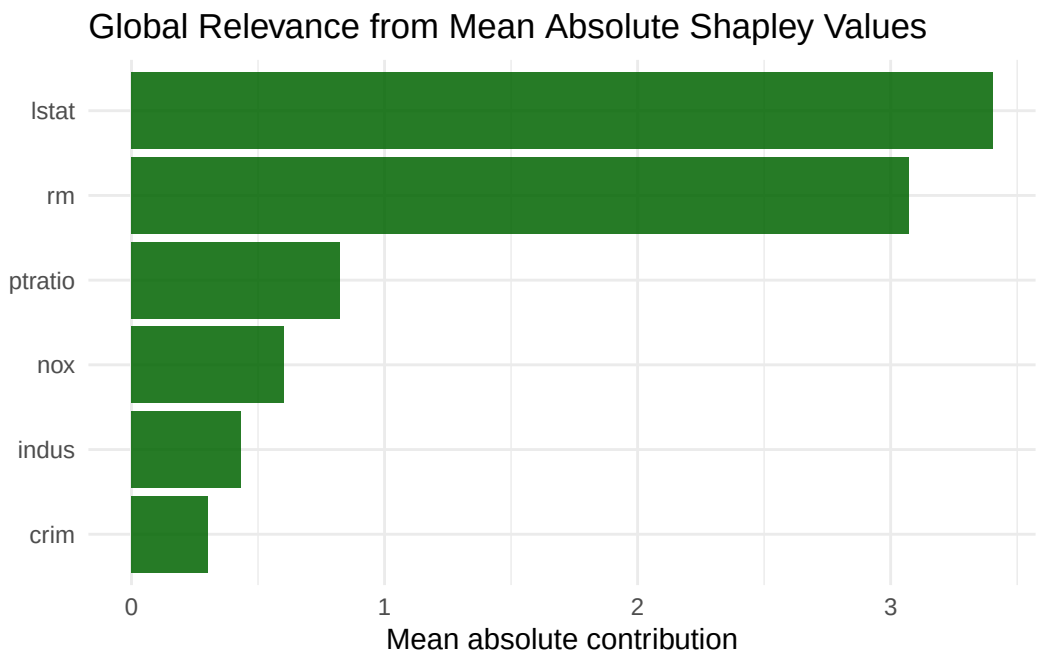
```
shapley_many_mc <- function(model,
                            data,
                            background,
                            features,
                            predict_fn,
                            ids,
                            n_permutations = 250,
                            seed = 123) {
    res <- lapply(seq_along(ids), function(k) {
        i <- ids[k]
        phi_i <- shapley_single_mc(
            model = model,
            x = data[i, features, drop = FALSE],
            background = background,
            features = features,
            predict_fn = predict_fn,
            n_permutations = n_permutations,
            seed = seed + k
        )

        as.data.frame(as.list(phi_i)) |>
```

```
            mutate(obs_id = i)
    })

    bind_rows(res)
}

set.seed(999)
ids_explain <- sample(seq_len(nrow(test)), size = 40)

shap_values <- shapley_many_mc(
    model = rf_fit,
    data = test,
    background = background,
    features = predictors,
    predict_fn = predict_rf,
    ids = ids_explain,
    n_permutations = 300,
    seed = 300
)

global_shap <- tibble::tibble(
    feature = predictors,
    mean_abs_phi = sapply(predictors, function(v) mean(abs(shap_values[[v]])))
) |>
    arrange(desc(mean_abs_phi))

global_shap |>
    gt() |>
    cols_label(
        feature = "Feature",
        mean_abs_phi = "Mean |Shapley|"
    ) |>
    fmt_number(columns = mean_abs_phi, decimals = 3)
```

```
global_shap |>
    ggplot(aes(x = reorder(feature, mean_abs_phi), y = mean_abs_phi)) +
    geom_col(fill = "darkgreen", alpha = 0.85) +
    coord_flip() +
    labs(
        title = "Global Relevance from Mean Absolute Shapley Values",
        x = NULL,
```

Feature	Mean |Shapley|
lstat	3.403
rm	3.074
ptratio	0.822
nox	0.603
indus	0.434
crim	0.302

```
        y = "Mean absolute contribution"
    ) +
    theme_minimal()
```

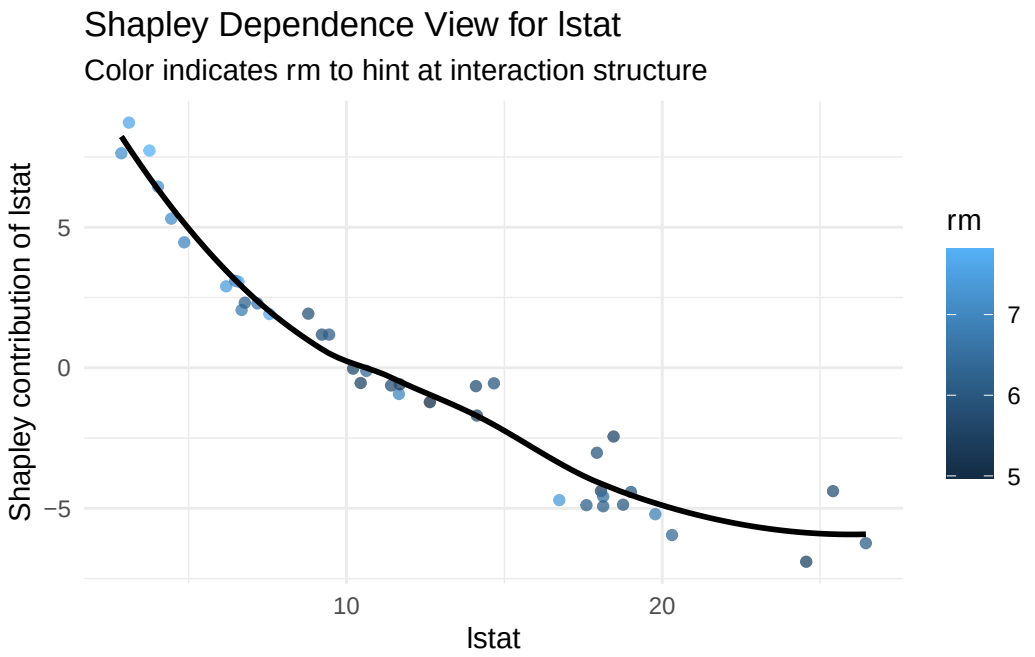


This summary can be compared with feature importance and partial dependence patterns from previous chapters. Agreement across methods often strengthens interpretation, while disagreement can signal dependence or interaction structure that deserves closer inspection.

16.9 Dependence Style View of a Shapley Feature

A practical diagnostic is to plot one feature against its Shapley contribution across observations. This helps us connect local attributions to profile style interpretation.

```r
dep_tbl <- shap_values |>
    mutate(
        lstat_value = test$lstat[obs_id],
        rm_value = test$rm[obs_id]
    )

dep_tbl |>
    ggplot(aes(x = lstat_value, y = lstat, color = rm_value)) +
    geom_point(alpha = 0.75) +
    geom_smooth(method = "loess", se = FALSE, color = "black") +
    labs(
        title = "Shapley Dependence View for lstat",
        subtitle = "Color indicates rm to hint at interaction structure",
        x = "lstat",
        y = "Shapley contribution of lstat",
        color = "rm"
    ) +
    theme_minimal()
```



This display is not identical to PD or ICE, but it often reveals similar shape information and can highlight interaction patterns through the color gradient.

16.10 Practical Limitations and Interpretation Cautions

Shapley values provide a coherent additive decomposition, but interpretation remains conditional on modeling and estimation choices.

- **Dependence sensitivity**. Under correlated predictors, attributions can shift when we change value function assumptions or background data.
- **Computational cost**. Monte Carlo estimation can be expensive for large samples or many predictors.
- **Monte Carlo variability**. Small contributions may fluctuate across runs when permutation counts are low.
- **Baseline dependence**. Contributions are defined relative to a reference expectation, so absolute values depend on baseline choice.
- **Predictive attribution, not intervention**. Contributions summarize model behavior, not causal response to manipulations.

In descriptive workflows, these limitations are manageable when we report assumptions explicitly and compare conclusions with complementary tools such as partial dependence and permutation importance.

16.11 Suggested Workflow for Robust Use

Building on the sequence of interpretability chapters so far, a practical workflow can be organized as follows:

1. establish predictive adequacy for the fitted model,
2. inspect global relevance summaries,
3. study profile based behavior with PD and ICE,
4. use Shapley values for case specific decomposition,
5. aggregate local contributions back to global summaries,
6. communicate assumptions and uncertainty about attribution.

This layered process balances readability with technical rigor, and helps avoid over interpreting any single explanation technique in isolation.

16.12 Summary and Key Takeaways

Shapley values extend the interpretability toolkit by providing additive local attributions with a clear mathematical foundation. The main points from this chapter are:

- **Shapley values decompose individual predictions** relative to a baseline expectation.

- **Axiomatic properties support fairness style allocation of contribution credit** for a specified value function.
- **Monte Carlo estimators make Shapley analysis practical** in tabular models, with approximation error that can be monitored.
- **Mean absolute Shapley values provide a bridge to global relevance summaries** across observations.
- **Interpretation quality depends on dependence assumptions, baseline choice, and computational settings**.

Used jointly with profile and importance methods, Shapley explanations provide a detailed and communicable account of model behavior at both local and global scales.

16.13 Looking Ahead

The next chapter moves toward automated machine learning as a descriptive tool. With the interpretability components established in this part of the book, we can study how automated model search can be integrated with transparent diagnostics rather than treated as a purely black box procedure.

Part VI

AutoML for Exploration

17 AutoML as a Descriptive Tool

17.1 Introduction: From Local Explanations to Automated Model Search

In the previous chapters, we moved from global summaries to local additive explanations. We now take a complementary step and consider how model construction itself can be partially automated, while still serving descriptive goals.

AutoML (automated machine learning) refers to the use of algorithms to automate parts of the model building process, such as trying multiple model types, tuning hyperparameters, and ranking candidates with a common evaluation metric. It is often presented as a performance oriented engineering workflow. That view is useful, but in descriptive analysis we can also treat AutoML as a structured instrument for learning about data. By exploring multiple model families and hyperparameters under a common evaluation protocol, we obtain a comparative map of predictive structure.

This chapter develops that perspective. We formalize the main ingredients of AutoML, implement a reproducible search workflow in R, and discuss how to interpret AutoML outputs in a careful descriptive way. We also revisit the same logic through two production platforms, Vertex AI AutoML (Google Cloud Console) and SageMaker Canvas (Amazon Web Services), to illustrate how these ideas are operationalized in common no-code and low-code environments. The objective is not to replace domain reasoning with automation, but to use automation to support transparent and technically grounded model comparison.

17.2 Why AutoML Can Be Useful for Descriptive Analysis

When we fit one model at a time, we often learn deeply about that model, but we can miss broader patterns. A modest AutoML workflow helps us answer additional questions:

- does a flexible nonlinear model materially outperform a simpler baseline,
- how sensitive performance is to hyperparameter choices,
- whether several candidate models are practically tied,
- how robust model rankings are across resamples.

These are descriptive questions about predictive structure and model behavior under the observed data distribution. As in earlier chapters, they should not be interpreted as causal claims.

17.3 AutoML as an Optimization Problem

A generic AutoML procedure searches over a space of pipelines and minimizes an estimated generalization loss.

Let $\mathcal{A}$ denote a set of algorithms, and let Λ_a be the hyperparameter space for algorithm $a \in \mathcal{A}$. A candidate is a pair (a, λ) with $\lambda \in \Lambda_a$. For loss function L, the target can be written as (Feurer et al. 2019; *Automated Machine Learning: Methods, Systems, Challenges* 2019):

$$(a^\star, \lambda^\star) = \arg \min_{a \in \mathcal{A},\, \lambda \in \Lambda_a} \widehat{R}(a, \lambda),$$

where $\widehat{R}(a, \lambda)$ is an estimate of predictive risk (for example, cross validated RMSE in regression).

In practice, an AutoML system also includes design choices that matter substantively:

- search strategy (grid, random, Bayesian, evolutionary),
- resampling protocol (holdout, cross validation, repeated cross validation),
- budget constraints (time, number of evaluations, early stopping),
- objective definition (single metric or multi objective with complexity penalties).

For descriptive work, these choices should be reported explicitly because they shape conclusions.

17.4 Running Example: A Lightweight AutoML Workflow on Boston Housing

To keep continuity with recent chapters, we use the Boston housing data and predict `medv`. We compare three model families:

- linear regression,
- regression trees (`rpart`),
- random forests.

The workflow is intentionally transparent and uses only packages already present in this book sequence.

```
data(Boston, package = "MASS")

set.seed(123)
idx_test <- sample(seq_len(nrow(Boston)), size = 0.2 * nrow(Boston))

analysis_data <- Boston[-idx_test, ]
test_data <- Boston[idx_test, ]

predictors <- c("lstat", "rm", "ptratio", "indus", "nox", "crim")
target <- "medv"

model_formula <- as.formula(paste(target, "~", paste(predictors, collapse = " + ")))

nrow(analysis_data)
```

```
[1] 405
```

```
nrow(test_data)
```

```
[1] 101
```

We reserve a final test set and perform model search only on `analysis_data`. This separation helps us avoid optimistic reporting when comparing many candidates.

17.5 A Reproducible Search and Evaluation Engine

We now define helper functions for fold creation, RMSE computation, and candidate evaluation.

```
rmse <- function(actual, predicted) {
    sqrt(mean((actual - predicted)^2))
}

make_folds <- function(n, k = 5, seed = 123) {
    set.seed(seed)
    fold_id <- sample(rep(seq_len(k), length.out = n))
```

```
    fold_id
}

evaluate_candidate <- function(data,
                               outcome,
                               fold_id,
                               fit_fun,
                               pred_fun,
                               params = list()) {
    k <- length(unique(fold_id))
    fold_rmse <- numeric(k)

    for (fold in seq_len(k)) {
        train_fold <- data[fold_id != fold, , drop = FALSE]
        valid_fold <- data[fold_id == fold, , drop = FALSE]

        fit_obj <- do.call(fit_fun, c(list(train_data = train_fold), params))
        pred <- pred_fun(fit_obj, valid_fold)
        fold_rmse[fold] <- rmse(valid_fold[[outcome]], pred)
    }

    tibble::tibble(
        mean_rmse = mean(fold_rmse),
        sd_rmse = sd(fold_rmse)
    )
}
```

Next, we define model specific fit and predict wrappers.

```
fit_lm_model <- function(train_data, formula_obj) {
    lm(formula_obj, data = train_data)
}

pred_lm_model <- function(model_obj, new_data) {
    as.numeric(predict(model_obj, newdata = new_data))
}

fit_tree_model <- function(train_data, formula_obj, cp, maxdepth, minsplit) {
    rpart(
        formula_obj,
        data = train_data,
        method = "anova",
```

```
        control = rpart.control(cp = cp, maxdepth = maxdepth, minsplit = minsplit)
    )
}

pred_tree_model <- function(model_obj, new_data) {
    as.numeric(predict(model_obj, newdata = new_data))
}

fit_rf_model <- function(train_data, formula_obj, mtry, ntree, seed = 123) {
    set.seed(seed)
    randomForest(
        formula_obj,
        data = train_data,
        mtry = mtry,
        ntree = ntree
    )
}

pred_rf_model <- function(model_obj, new_data) {
    as.numeric(predict(model_obj, newdata = new_data))
}
```

17.6 Candidate Space and Leaderboard Construction

We specify a compact candidate space, evaluate each candidate with five fold cross validation, and collect results in one table.

```
fold_id <- make_folds(n = nrow(analysis_data), k = 5, seed = 456)

results_list <- list()
row_counter <- 1

# 1) Linear regression baseline
lm_eval <- evaluate_candidate(
    data = analysis_data,
    outcome = target,
    fold_id = fold_id,
    fit_fun = fit_lm_model,
    pred_fun = pred_lm_model,
    params = list(formula_obj = model_formula)
```

```
)

results_list[[row_counter]] <- tibble::tibble(
    family = "Linear regression",
    config = "standard OLS",
    mean_rmse = lm_eval$mean_rmse,
    sd_rmse = lm_eval$sd_rmse
)
row_counter <- row_counter + 1

# 2) Regression tree grid
tree_grid <- expand.grid(
    cp = c(0.001, 0.01),
    maxdepth = c(3, 6),
    minsplit = c(10, 20)
)

for (i in seq_len(nrow(tree_grid))) {
    this_eval <- evaluate_candidate(
        data = analysis_data,
        outcome = target,
        fold_id = fold_id,
        fit_fun = fit_tree_model,
        pred_fun = pred_tree_model,
        params = list(
            formula_obj = model_formula,
            cp = tree_grid$cp[i],
            maxdepth = tree_grid$maxdepth[i],
            minsplit = tree_grid$minsplit[i]
        )
    )

    results_list[[row_counter]] <- tibble::tibble(
        family = "Regression tree",
        config = paste0(
            "cp=", tree_grid$cp[i],
            ", maxdepth=", tree_grid$maxdepth[i],
            ", minsplit=", tree_grid$minsplit[i]
        ),
        mean_rmse = this_eval$mean_rmse,
        sd_rmse = this_eval$sd_rmse
    )
```

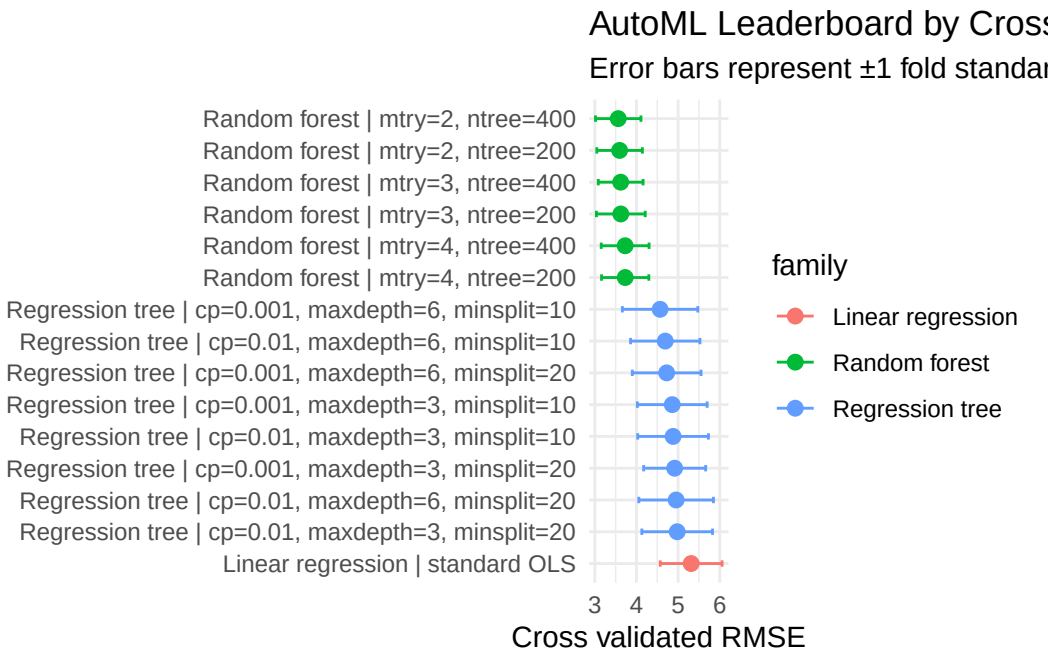
```
    row_counter <- row_counter + 1
}

# 3) Random forest grid
rf_grid <- expand.grid(
    mtry = c(2, 3, 4),
    ntree = c(200, 400)
)

for (i in seq_len(nrow(rf_grid))) {
    this_eval <- evaluate_candidate(
        data = analysis_data,
        outcome = target,
        fold_id = fold_id,
        fit_fun = fit_rf_model,
        pred_fun = pred_rf_model,
        params = list(
            formula_obj = model_formula,
            mtry = rf_grid$mtry[i],
            ntree = rf_grid$ntree[i],
            seed = 100 + i
        )
    )

    results_list[[row_counter]] <- tibble::tibble(
        family = "Random forest",
        config = paste0("mtry=", rf_grid$mtry[i], ", ntree=", rf_grid$ntree[i]),
        mean_rmse = this_eval$mean_rmse,
        sd_rmse = this_eval$sd_rmse
    )
    row_counter <- row_counter + 1
}

leaderboard <- bind_rows(results_list) |>
    arrange(mean_rmse)

leaderboard |>
    mutate(rank = row_number()) |>
    dplyr::select(rank, family, config, mean_rmse, sd_rmse) |>
    gt() |>
    cols_label(
        rank = "Rank",
```

Rank	Model family	Configuration	CV mean RMSE	CV SD
1	Random forest	mtry=2, ntree=400	3.562	0.546
2	Random forest	mtry=2, ntree=200	3.595	0.547
3	Random forest	mtry=3, ntree=400	3.621	0.538
4	Random forest	mtry=3, ntree=200	3.624	0.586
5	Random forest	mtry=4, ntree=400	3.728	0.574
6	Random forest	mtry=4, ntree=200	3.729	0.568
7	Regression tree	cp=0.001, maxdepth=6, minsplit=10	4.567	0.906
8	Regression tree	cp=0.01, maxdepth=6, minsplit=10	4.691	0.833
9	Regression tree	cp=0.001, maxdepth=6, minsplit=20	4.725	0.825
10	Regression tree	cp=0.001, maxdepth=3, minsplit=10	4.859	0.838
11	Regression tree	cp=0.01, maxdepth=3, minsplit=10	4.879	0.850
12	Regression tree	cp=0.001, maxdepth=3, minsplit=20	4.917	0.745
13	Regression tree	cp=0.01, maxdepth=6, minsplit=20	4.953	0.896
14	Regression tree	cp=0.01, maxdepth=3, minsplit=20	4.978	0.849
15	Linear regression	standard OLS	5.314	0.742

```
        family = "Model family",
        config = "Configuration",
        mean_rmse = "CV mean RMSE",
        sd_rmse = "CV SD"
    ) |>
    fmt_number(columns = c(mean_rmse, sd_rmse), decimals = 3)
```

```
top_plot <- leaderboard |>
    mutate(
        model_id = paste(family, config, sep = " | "),
        model_id = factor(model_id, levels = rev(model_id))
    )

top_plot |>
    ggplot(aes(x = model_id, y = mean_rmse, color = family)) +
    geom_point(size = 2.3) +
    geom_errorbar(aes(ymin = mean_rmse - sd_rmse, ymax = mean_rmse + sd_rmse), width = 0.2) +
    coord_flip() +
    labs(
        title = "AutoML Leaderboard by Cross Validated RMSE",
        subtitle = "Error bars represent ±1 fold standard deviation",
        x = NULL,
        y = "Cross validated RMSE"
```

```
    ) +
    theme_minimal()
```



The leaderboard provides more than a winner. It shows whether performance differences are large, modest, or practically negligible relative to resampling variability.

17.7 Selecting a Final Candidate and Evaluating on Holdout Data

We now fit the best ranked candidate on all `analysis_data` and evaluate once on the untouched test set.

```
best_row <- leaderboard[1, ]

best_row |>
    gt() |>
    cols_label(
        family = "Selected family",
        config = "Selected configuration",
        mean_rmse = "CV mean RMSE",
        sd_rmse = "CV SD"
    ) |>
    fmt_number(columns = c(mean_rmse, sd_rmse), decimals = 3)
```

Selected family	Selected configuration	CV mean RMSE	CV SD
Random forest	mtry=2, ntree=400	3.562	0.546

```
fit_best_model <- function(best_row, train_data, formula_obj) {
    if (best_row$family == "Linear regression") {
        return(fit_lm_model(train_data = train_data, formula_obj = formula_obj))
    }

    if (best_row$family == "Regression tree") {
        parts <- strsplit(best_row$config, ", ")[[1]]
        cp_val <- as.numeric(sub("cp=", "", parts[1]))
        maxdepth_val <- as.numeric(sub("maxdepth=", "", parts[2]))
        minsplit_val <- as.numeric(sub("minsplit=", "", parts[3]))

        return(
            fit_tree_model(
                train_data = train_data,
                formula_obj = formula_obj,
                cp = cp_val,
                maxdepth = maxdepth_val,
                minsplit = minsplit_val
            )
        )
    }

    parts <- strsplit(best_row$config, ", ")[[1]]
    mtry_val <- as.numeric(sub("mtry=", "", parts[1]))
    ntree_val <- as.numeric(sub("ntree=", "", parts[2]))

    fit_rf_model(
        train_data = train_data,
        formula_obj = formula_obj,
        mtry = mtry_val,
        ntree = ntree_val,
        seed = 999
    )
}

predict_best_model <- function(best_row, model_obj, new_data) {
    if (best_row$family == "Linear regression") {
        return(pred_lm_model(model_obj, new_data))
```

Selected family	Selected configuration	CV RMSE	Holdout RMSE
Random forest	mtry=2, ntree=400	3.562	3.750

```
    }
    if (best_row$family == "Regression tree") {
        return(pred_tree_model(model_obj, new_data))
    }
    pred_rf_model(model_obj, new_data)
}

best_model <- fit_best_model(
    best_row = best_row,
    train_data = analysis_data,
    formula_obj = model_formula
)

test_pred <- predict_best_model(best_row, best_model, test_data)
test_rmse <- rmse(test_data[[target]], test_pred)

tibble::tibble(
    best_family = best_row$family,
    best_config = best_row$config,
    cv_rmse = best_row$mean_rmse,
    test_rmse = test_rmse
) |>
    gt() |>
    cols_label(
        best_family = "Selected family",
        best_config = "Selected configuration",
        cv_rmse = "CV RMSE",
        test_rmse = "Holdout RMSE"
    ) |>
    fmt_number(columns = c(cv_rmse, test_rmse), decimals = 3)
```

The gap between cross validated and holdout RMSE offers a quick diagnostic of selection stability. A small gap suggests that the search process did not overfit strongly to resampling noise.

Rank among near-optimal	Model family	Configuration	CV mean RMSE	CV SD
1	Random forest	mtry=2, ntree=400	3.562	0.546
2	Random forest	mtry=2, ntree=200	3.595	0.547
3	Random forest	mtry=3, ntree=400	3.621	0.538
4	Random forest	mtry=3, ntree=200	3.624	0.586
5	Random forest	mtry=4, ntree=400	3.728	0.574
6	Random forest	mtry=4, ntree=200	3.729	0.568

17.8 Near Optimal Models and Practical Equivalence

A useful descriptive habit is to avoid over interpreting tiny leaderboard differences. One pragmatic criterion is to treat candidates as near optimal when their mean RMSE is within one standard deviation of the best candidate.

```
threshold <- leaderboard$mean_rmse[1] + leaderboard$sd_rmse[1]

near_optimal <- leaderboard |>
    filter(mean_rmse <= threshold) |>
    mutate(rank = row_number())

near_optimal |>
    dplyr::select(rank, family, config, mean_rmse, sd_rmse) |>
    gt() |>
    cols_label(
        rank = "Rank among near-optimal",
        family = "Model family",
        config = "Configuration",
        mean_rmse = "CV mean RMSE",
        sd_rmse = "CV SD"
    ) |>
    fmt_number(columns = c(mean_rmse, sd_rmse), decimals = 3)
```

When several models are near tied, we can use secondary criteria, for example, computational cost, interpretability, or robustness under distribution shift assumptions.

17.9 Descriptive Interpretation of AutoML Results

In a descriptive workflow, the AutoML output is not only a model selector. It is also a structured summary of what the data seem to support under competing functional assumptions.

For instance:

- if linear regression is competitive, the dominant structure may be close to additive linear signal,
- if trees and forests improve substantially, nonlinearities or interactions likely matter,
- if many configurations tie, model uncertainty should be communicated explicitly.

This interpretation remains predictive and data dependent. As discussed earlier in the book, moving from predictive regularities to causal statements requires additional design assumptions and identification strategies.

17.10 Good Practice for Reporting AutoML in Descriptive Work

To keep reporting transparent and reproducible, it is often useful to document:

- search space (model families and hyperparameter ranges),
- resampling protocol and random seeds,
- optimization metric and any tie breaking rule,
- final holdout performance,
- whether conclusions rely on one model or a near optimal set.

These elements help readers understand how much of the final conclusion is driven by data signal versus search design.

17.11 Limits and Cautions

AutoML can improve coverage of model space, but it does not eliminate substantive judgment.

Key limits include:

- performance metrics can hide subgroup specific errors,
- search spaces can encode strong implicit priors,
- correlated predictors can produce unstable model rankings,
- repeated experimentation can still induce selection bias if holdout data are reused.

A careful descriptive workflow therefore combines automation with explicit diagnostics, domain context, and communication of uncertainty.

17.12 AutoML in Practice: Vertex AI and SageMaker Canvas

The R workflow above makes each modeling choice explicit. In practice, many teams implement similar logic through managed platforms. To connect this chapter to operational settings, we briefly consider two widely used platforms, Vertex AI AutoML and SageMaker Canvas, using the same evaluative lens developed earlier in the chapter.

17.12.1 Overview of Vertex AI AutoML

Vertex AI AutoML is Google's managed environment for automated model training and selection. In tabular settings, it supports supervised tasks such as classification and regression by combining data preparation steps, candidate model search, hyperparameter tuning, and evaluation reporting within one workflow.

The platform workflow typically follows these steps:

1. users upload tabular data and specify the prediction target;
2. the system performs automatic data exploration and feature engineering;
3. candidates are generated by trying multiple model architectures and hyperparameter combinations;
4. each candidate is evaluated via cross-validation on a held-out portion of training data;
5. a leaderboard ranks models by performance metrics such as RMSE (regression) or AUC-ROC (classification);
6. the top-ranked model or an ensemble is deployed for inference.

Vertex AI emphasizes broad model exploration and provides strong predictive results, especially when sufficient computational budget is available. The platform generates detailed evaluation reports including feature importance scores and model comparison tables that align with the leaderboard concept discussed in this chapter.

17.12.2 Overview of SageMaker Canvas

SageMaker Canvas is AWS's visual interface for building machine learning models with limited coding. For tabular data, it offers guided workflows for data ingestion, model building, evaluation, and prediction generation, with optional quick build versus standard optimization modes.

The Canvas workflow is similarly structured:

1. users import data via the web interface or cloud storage;
2. the system performs exploratory data analysis and data profiling;
3. the user specifies the target variable and builds a model with minimal configuration;
4. Canvas automatically tries different model families and hyperparameters;

5. results including predictions, model metrics, and feature importance are presented in a dashboard;
6. predictions can be generated on new data directly within the interface.

SageMaker Canvas prioritizes usability and rapid iteration. In applied contexts, this design can lower entry barriers for analysts who need interpretable summaries and quick feedback before investing in heavier engineering pipelines. The platform includes model explanation features that communicate variable importance to non-technical stakeholders.

17.12.3 Practical Comparison

Comparing Vertex AI AutoML and SageMaker Canvas from a descriptive analysis perspective highlights several structural patterns.

Similarities include:

- both platforms automate model selection and hyperparameter tuning from tabular data,
- both report standard evaluation metrics and feature importance summaries,
- both allow prediction generation without manual pipeline coding,
- both maintain a candidate leaderboard or ranking of explored models.

Structural differences documented in official platform descriptions include:

- Vertex AI's architecture is designed to explore a broader search space and can generate ensemble models for higher predictive accuracy, at the cost of longer training time,
- SageMaker Canvas emphasizes quick iteration with a "quick build" mode for rapid baseline models and a "standard build" mode for more thorough search, trading speed for computational cost control,
- Canvas aims for a beginner-friendly visual interface with drag-and-drop data preparation,
- Vertex AI and Canvas differ in their approach to uncertainty quantification; Vertex AI provides detailed training metrics and hyperparameter sensitivity reports, while Canvas focuses on prediction confidence intervals,
- pricing models differ, with Vertex AI billed by training time and SageMaker Canvas based on model training duration and complexity.

These contrasts should be interpreted as reflecting different design priorities rather than universal rankings. Organization policies, existing cloud investments, team technical expertise, and project requirements all influence which platform is more suitable in practice.

17.12.4 Strengths, Limitations, and Use Cases

For advanced descriptive analysis of tabular data, a platform choice can be framed in terms of analytic priorities.

- Vertex AI AutoML can be attractive when broader model search and comprehensive feature importance analysis are central goals, and when organization infrastructure already uses Google Cloud Platform.
- SageMaker Canvas can be attractive when rapid iteration, visual collaboration with non-technical stakeholders, and AWS ecosystem integration are priorities.
- In both platforms, automated outputs still require statistical interpretation, especially when predictors are correlated, classes are imbalanced, or subgroup performance differs.

An analytically balanced approach is therefore to treat these platforms as structured experimentation environments documented by official resources. They can accelerate candidate discovery and provide transparent leaderboards for model comparison, while inferential framing, diagnostic scrutiny, and domain interpretation remain human responsibilities.

17.12.5 Implications for Descriptive and Exploratory Analysis

From the perspective of this book, the main value of these platforms is not only faster model fitting. Their broader contribution is methodological: they make it easier to compare many predictive hypotheses under a consistent evaluation framework, capture that comparison in transparent leaderboards, and generate feature importance summaries at scale.

That capacity aligns directly with advanced descriptive analysis, where we use predictive tools to characterize pattern strength, nonlinear structure, interaction potential, and uncertainty in model choice. Vertex AI and SageMaker Canvas operationalize the leaderboard and ranking logic developed in the R workflow of this chapter. When used with careful interpretation of results and explicit documentation of search design, platform-based AutoML can complement the transparent R-first workflow developed earlier in this chapter.

17.13 Summary and Key Takeaways

- AutoML can be used as a descriptive instrument, not only as a predictive optimizer.
- A transparent search setup provides comparative evidence about functional form and model sensitivity.
- Leaderboards are most informative when interpreted with variability and near tie structure.
- Platform implementations such as Vertex AI AutoML and SageMaker Canvas extend this logic to operational settings, with trade-offs in accuracy, speed, usability, and cost that should be evaluated in context.

- Final model selection should be paired with untouched holdout evaluation and explicit reporting choices.
- Automation supports analysis, but it does not replace substantive interpretation.

17.14 Looking Ahead

This chapter used automation to search model and hyperparameter spaces. The next chapter extends that idea to the predictor space itself, where automated feature engineering helps us discover transformed variables and interaction candidates that can improve both predictive performance and descriptive resolution.

18 Automated Feature Engineering and Interaction Discovery

18.1 Introduction: From Model Search to Feature Space Search

In the previous chapter, we framed AutoML as a descriptive instrument for comparing model families under a common validation protocol. That perspective can be extended in a natural way. Beyond searching over algorithms and hyperparameters, we can also search over representations of the predictors themselves.

Automated feature engineering refers to procedures that generate, screen, and select transformed or combined predictors, often at scale. In tabular settings, this frequently includes nonlinear transformations, pairwise interactions, and simple basis expansions. The goal is not to replace substantive understanding of variables, but to broaden the set of candidate structures that can be evaluated transparently.

This chapter develops that workflow in a reproducible R implementation. We focus on interaction discovery and transformed predictors in a regression setting, then discuss interpretation limits and reporting choices that keep conclusions technically grounded.

18.2 Why Feature Engineering Can Be Automated in Descriptive Work

Earlier in the book, we repeatedly saw that model conclusions depend on how signal is represented. If an important interaction is omitted, an additive model may appear systematically biased in parts of the covariate space. If a relation is nonlinear, raw linear terms can underfit even when model diagnostics look acceptable in aggregate.

A modest automated feature engineering workflow can support descriptive analysis by helping us:

- detect candidate nonlinear structure that is not visible in additive baselines,
- rank interaction candidates under a common cross validation design,
- compare several near-optimal representations instead of one handcrafted specification,
- document feature space uncertainty, not only parameter uncertainty.

As in the previous chapter, these are predictive and descriptive statements about fitted behavior under observed data conditions. They do not, by themselves, identify causal effects.

18.3 A Formal View: Search Over Feature Mappings

Let $X \in \mathbb{R}^{n \times p}$ denote the original predictor matrix and y the outcome. A feature engineering procedure defines a mapping family

$$\Phi = \{\phi_1, \dots, \phi_m\}, \quad \phi_j : \mathbb{R}^p \to \mathbb{R}^{q_j},$$

where each mapping can represent, for example, polynomial terms, logarithmic transforms, or interactions. For a chosen modeling algorithm $\mathcal{A}$ with parameters θ, we evaluate candidates of the form

$$\hat{f}_{j,\theta}(x) = \mathcal{A}(\phi_j(x); \theta).$$

Automated feature engineering then becomes a structured optimization problem:

$$(j^\star, \theta^\star) = \arg\min_{j,\theta} \widehat{R}(\hat{f}_{j,\theta}),$$

with $\widehat{R}$ estimated by cross validation or a related resampling protocol.

In practice, this search can be large. A key methodological point is that search design (candidate library, screening rule, evaluation budget) is part of the inferential context and should be reported explicitly.

18.4 Running Example: Boston Housing With a Transparent Candidate Library

To remain consistent with the previous chapters, we use the Boston housing data and predict `medv`. We first fit a baseline additive linear model, then evaluate engineered candidates one at a time on top of that baseline.

```
data(Boston, package = "MASS")

set.seed(123)
idx_test <- sample(seq_len(nrow(Boston)), size = 0.2 * nrow(Boston))

analysis_data <- Boston[-idx_test, ]
test_data <- Boston[idx_test, ]

predictors <- c("lstat", "rm", "ptratio", "indus", "nox", "crim")
target <- "medv"

baseline_formula <- as.formula(
    paste(target, "~", paste(predictors, collapse = " + "))
)

nrow(analysis_data)
```

```
[1] 405
```

```
nrow(test_data)
```

```
[1] 101
```

We keep a final holdout test set untouched during candidate screening. This separation helps reduce optimism after searching across many engineered terms.

18.5 A Reproducible Evaluation Engine

We implement helper functions for fold assignment and cross validated RMSE. The code below is intentionally compact and explicit, so that each design choice is auditable.

```
rmse <- function(actual, predicted) {
    sqrt(mean((actual - predicted)^2))
}

make_folds <- function(n, k = 5, seed = 123) {
    set.seed(seed)
    sample(rep(seq_len(k), length.out = n))
}
```

```
cv_rmse_lm <- function(data, formula_obj, fold_id) {
    k <- length(unique(fold_id))
    fold_errors <- numeric(k)

    for (fold in seq_len(k)) {
        train_fold <- data[fold_id != fold, , drop = FALSE]
        valid_fold <- data[fold_id == fold, , drop = FALSE]

        fit <- lm(formula_obj, data = train_fold)
        pred <- as.numeric(predict(fit, newdata = valid_fold))
        fold_errors[fold] <- rmse(valid_fold[[target]], pred)
    }

    tibble::tibble(
        mean_rmse = mean(fold_errors),
        sd_rmse = sd(fold_errors)
    )
}
```

18.6 Constructing Candidate Features

For each predictor, we generate two transformation candidates, a quadratic term and a log transformation. We also generate all pairwise interactions among the selected predictors.

```
transform_terms <- unlist(lapply(predictors, function(v) {
    c(
        paste0("I(", v, "^2)"),
        paste0("log1p(", v, ")")
    )
}))

interaction_terms <- combn(predictors, 2, simplify = FALSE) |>
    lapply(function(pair) paste(pair, collapse = ":")) |>
    unlist()

candidate_terms <- c(transform_terms, interaction_terms)

length(candidate_terms)
```

```
[1] 27
```

```
head(candidate_terms)
```

```
[1] "I(lstat^2)"     "log1p(lstat)"   "I(rm^2)"        "log1p(rm)"
[5] "I(ptratio^2)"   "log1p(ptratio)"
```

This library is not exhaustive, but it is broad enough to illustrate automated screening. In applied projects, candidate generation is often guided by domain constraints, monotonicity expectations, and operational interpretability requirements.

18.7 Screening Candidates by Cross Validated Improvement

We now evaluate each candidate term by augmenting the baseline formula with one additional engineered feature and measuring out of fold RMSE.

```
fold_id <- make_folds(n = nrow(analysis_data), k = 5, seed = 456)

baseline_cv <- cv_rmse_lm(
    data = analysis_data,
    formula_obj = baseline_formula,
    fold_id = fold_id
)

screen_results <- lapply(candidate_terms, function(term) {
    formula_term <- as.formula(
        paste(target, "~", paste(c(predictors, term), collapse = " + "))
    )

    res <- cv_rmse_lm(
        data = analysis_data,
        formula_obj = formula_term,
        fold_id = fold_id
    )

    tibble::tibble(
        term = term,
        term_type = ifelse(grepl(":", term), "Interaction", "Transformation"),
        mean_rmse = res$mean_rmse,
        sd_rmse = res$sd_rmse,
        delta_rmse = baseline_cv$mean_rmse - res$mean_rmse
    )
```

CV mean RMSE	CV SD	Model
5.314	0.742	Baseline additive model

```
}) |>
    bind_rows() |>
    arrange(desc(delta_rmse), mean_rmse)

baseline_cv |>
    mutate(model = "Baseline additive model") |>
    gt() |>
    cols_label(
        model = "Model",
        mean_rmse = "CV mean RMSE",
        sd_rmse = "CV SD"
    ) |>
    fmt_number(columns = c(mean_rmse, sd_rmse), decimals = 3)
```

```
screen_results |>
    slice_head(n = 12) |>
    mutate(rank = row_number()) |>
    dplyr::select(rank, term, term_type, mean_rmse, sd_rmse, delta_rmse) |>
    gt() |>
    cols_label(
        rank = "Rank",
        term = "Candidate term",
        term_type = "Type",
        mean_rmse = "CV mean RMSE",
        sd_rmse = "CV SD",
        delta_rmse = "Improvement vs baseline"
    ) |>
    fmt_number(columns = c(mean_rmse, sd_rmse, delta_rmse), decimals = 3)
```

The ranking helps us identify promising terms, while the SD column provides context about stability across folds. Small differences should be interpreted cautiously when they are comparable to fold level variability.

Rank	Candidate term	Type	CV mean RMSE	CV SD	Improvement vs baseline
1	lstat:rm	Interaction	4.570	0.825	0.744
2	log1p(lstat)	Transformation	4.629	0.471	0.685
3	I(rm^2)	Transformation	4.664	0.998	0.650
4	log1p(rm)	Transformation	4.703	1.036	0.611
5	I(lstat^2)	Transformation	4.791	0.477	0.523
6	rm:ptratio	Interaction	4.792	0.761	0.522
7	rm:indus	Interaction	4.943	0.882	0.371
8	rm:nox	Interaction	4.992	0.920	0.322
9	nox:crim	Interaction	5.202	0.755	0.112
10	log1p(crim)	Transformation	5.263	0.743	0.051
11	log1p(nox)	Transformation	5.276	0.655	0.038
12	I(crim^2)	Transformation	5.278	0.750	0.036

18.8 Building an Engineered Model From Top Candidates

A practical compromise between flexibility and readability is to keep only the strongest positive candidates. Here we retain up to five terms with positive estimated gain.

```
selected_terms <- screen_results |>
    filter(delta_rmse > 0) |>
    slice_head(n = 5) |>
    pull(term)

if (length(selected_terms) == 0) {
    selected_terms <- screen_results |> slice_head(n = 1) |> pull(term)
}

engineered_formula <- as.formula(
    paste(target, "~", paste(c(predictors, selected_terms), collapse = " + "))
)

engineered_formula
```

```
medv ~ lstat + rm + ptratio + indus + nox + crim + lstat:rm +
    log1p(lstat) + I(rm^2) + log1p(rm) + I(lstat^2)
```

```
baseline_fit <- lm(baseline_formula, data = analysis_data)
engineered_fit <- lm(engineered_formula, data = analysis_data)
```

Model	Holdout RMSE
Baseline additive	5.177
Engineered linear	4.540

```
pred_baseline <- as.numeric(predict(baseline_fit, newdata = test_data))
pred_engineered <- as.numeric(predict(engineered_fit, newdata = test_data))

comparison_tbl <- tibble::tibble(
    model = c("Baseline additive", "Engineered linear"),
    test_rmse = c(
        rmse(test_data[[target]], pred_baseline),
        rmse(test_data[[target]], pred_engineered)
    )
)

comparison_tbl |>
    gt() |>
    cols_label(
        model = "Model",
        test_rmse = "Holdout RMSE"
    ) |>
    fmt_number(columns = test_rmse, decimals = 3)
```

This test set comparison is only one realization, but it indicates whether screened candidates transfer beyond the resampling environment used during selection.

18.9 Inspecting the Interaction Signal

Screening tables are useful, but interaction discovery is often easier to communicate w ith a response surface. We extract the top ranked interaction term, if available, and visualize fitted values while keeping other predictors at their analysis set medians.

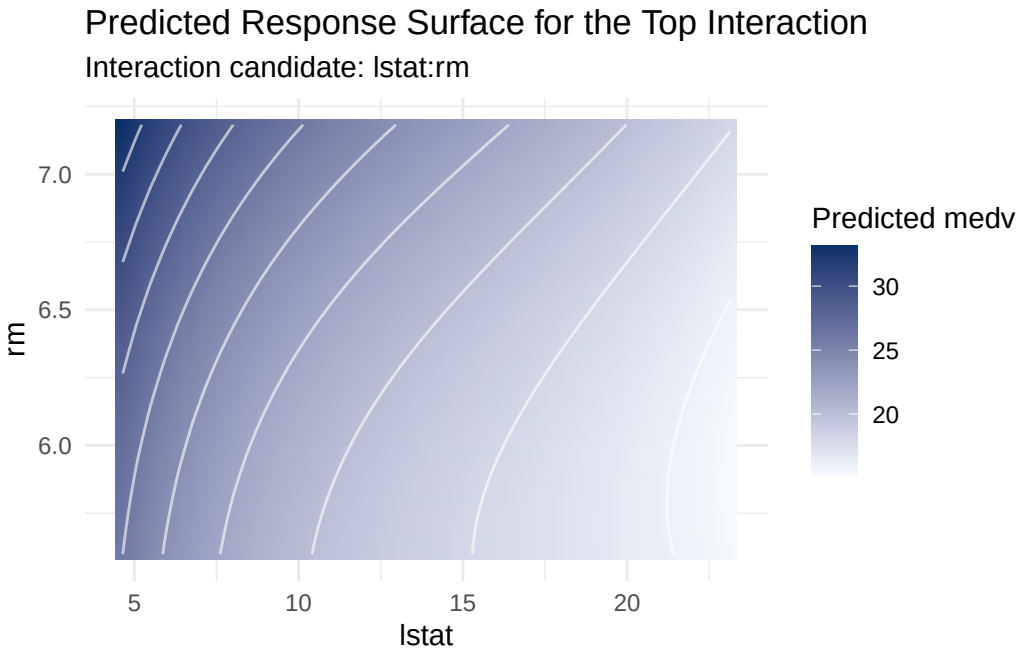
```
top_interaction <- screen_results |>
    filter(term_type == "Interaction") |>
    slice_head(n = 1)

top_interaction |>
    gt() |>
    cols_label(
```

Top Interaction Term	Type	CV mean RMSE	CV SD	Improvement vs baseline
lstat:rm	Interaction	4.570	0.825	0.744

```
        term = "Top Interaction Term",
        term_type = "Type",
        mean_rmse = "CV mean RMSE",
        sd_rmse = "CV SD",
        delta_rmse = "Improvement vs baseline"
    ) |>
    fmt_number(columns = c(mean_rmse, sd_rmse, delta_rmse), decimals = 3)
```

```
if (nrow(top_interaction) == 1) {
    vars <- strsplit(top_interaction$term[1], ":", fixed = TRUE)[[1]]
    v1 <- vars[1]
    v2 <- vars[2]

    grid_v1 <- seq(
        as.numeric(quantile(analysis_data[[v1]], 0.1)),
        as.numeric(quantile(analysis_data[[v1]], 0.9)),
        length.out = 45
    )

    grid_v2 <- seq(
        as.numeric(quantile(analysis_data[[v2]], 0.1)),
        as.numeric(quantile(analysis_data[[v2]], 0.9)),
        length.out = 45
    )

    surface_data <- expand.grid(
        x1 = grid_v1,
        x2 = grid_v2
    )
    names(surface_data) <- c(v1, v2)

    for (p in predictors) {
        if (!(p %in% c(v1, v2))) {
            surface_data[[p]] <- median(analysis_data[[p]])
        }
    }
```

```
    surface_data$pred <- as.numeric(predict(engineered_fit, newdata = surface_data))

    surface_data |>
        ggplot(aes(x = .data[[v1]], y = .data[[v2]], z = pred, fill = pred)) +
        geom_raster(interpolate = TRUE) +
        geom_contour(color = "white", alpha = 0.6) +
        scale_fill_gradient(low = "#f7fbff", high = "#08306b") +
        labs(
            title = "Predicted Response Surface for the Top Interaction",
            subtitle = paste("Interaction candidate:", top_interaction$term[1]),
            x = v1,
            y = v2,
            fill = "Predicted medv"
        ) +
        theme_minimal()
}
```



The surface should be read as model behavior conditional on fixed values of remaining predictors. It is a useful descriptive map, but not direct evidence of causal interaction.

18.10 Practical Considerations and Failure Modes

Automated feature engineering can increase descriptive resolution, but several risks are worth monitoring.

- **Search induced optimism** can appear when large candidate libraries are screened with weak validation protocols.
- **Collinearity inflation** can destabilize coefficient level interpretation in engineered linear models.
- **Data leakage** can occur if transformations use information outside the analysis fold.
- **Interpretability drift** can arise when many engineered terms are retained without pruning.

These issues do not invalidate automated workflows. They highlight why fold aware preprocessing, explicit holdout testing, and conservative reporting are central parts of the methodology.

18.11 Reporting Guidelines for Automated Feature Engineering

For transparent descriptive reporting, it is often helpful to document:

- candidate generation rules (which transforms and interactions were considered),
- validation design and random seeds,
- selection criterion (for example, improvement in cross validated RMSE),
- final retained terms and their practical contribution,
- holdout performance relative to a simpler baseline.

When several engineered specifications are near tied, presenting a small candidate set can be more informative than emphasizing a single winner.

18.12 Summary and Key Takeaways

- Automated feature engineering extends AutoML logic from model space to feature space.
- Interaction and transformation screening can reveal predictive structure not captured by additive baselines.
- Cross validated ranking is most useful when read together with variability measures and holdout confirmation.
- Interaction surfaces support interpretation, while remaining conditional predictive summaries.
- Transparent documentation of the candidate library and validation design is essential for credible descriptive conclusions.

18.13 Looking Ahead

This chapter focused on representation search in a controlled tabular workflow. In the next chapters, we turn to integrated case studies where model search, feature engineering, and interpretation are combined in full descriptive analyses for policy, health, and business contexts.

Part VII

Applied Case Studies

19 Case Study: Public Policy and Program Evaluation

19.1 Introduction: Bringing the Toolkit Together in a Policy Setting

Across the previous chapters, we developed a broad descriptive toolkit for tabular data, from type-aware association measures and network representations to tree-based segmentation, ensemble summaries, and communication principles. This case study is an opportunity to use those components together in an end-to-end policy workflow.

Our aim is modest and practical. We study county-level socioeconomic indicators to understand how structural characteristics co-occur with poverty rates, and how descriptive patterns can support policy interpretation and program design. We focus on transparent summaries and segmentation rules, while keeping a clear distinction between predictive description and causal evaluation.

19.2 Policy Context and Analytical Objective

We use the `midwest` dataset from `ggplot2`, which contains county-level demographic and socioeconomic variables for U.S. Midwest states. Although this is not a program trial dataset, it reflects the kinds of administrative and survey indicators that often appear in policy analysis.

For this chapter, we treat `percbelowpoverty` (percent of individuals below the poverty threshold) as a policy-relevant outcome proxy. We ask three connected descriptive questions:

1. Which variables show the strongest association with county poverty rates, in a mixed-type setting?
2. How those associations organize into a multivariate network structure?
3. Which interpretable county segments emerge from recursive partitioning?

These questions align with exploratory policy work where the immediate goal is targeting, profiling, and communication, not effect identification.

19.3 Data Preparation and Initial Profiling

We create a compact analysis table with both continuous and categorical variables. The mix of variable types lets us reuse the unified association logic introduced earlier in the book.

```
policy_raw <- ggplot2::midwest

policy_data <- policy_raw |>
    transmute(
        county,
        state,
        inmetro = factor(ifelse(inmetro == 1, "Metro", "Non-metro")),
        category = factor(category),
        popdensity = as.numeric(popdensity),
        log_popdensity = log1p(popdensity),
        percbelowpoverty = as.numeric(percbelowpoverty),
        perchsd = as.numeric(perchsd),
        percollege = as.numeric(percollege),
        percprof = as.numeric(percprof),
        percwhite = as.numeric(percwhite),
        percblack = as.numeric(percblack)
    ) |>
    drop_na()

nrow(policy_data)
```

```
[1] 437
```

Before moving to multivariate structure, we inspect a small baseline profile.

```
baseline_profile <- policy_data |>
    summarise(
        counties = n(),
        mean_poverty = mean(percbelowpoverty),
        sd_poverty = sd(percbelowpoverty),
        mean_college = mean(percollege),
        mean_professional = mean(percprof),
        mean_hsd = mean(perchsd)
    )

baseline_profile |>
```

Counties	Mean poverty rate	SD poverty rate	Mean college share	Mean professional share	Mean
437	12.51	5.15	18.27	4.45	

County type	N	Mean poverty rate	Mean college share	Mean population density
Metro	150	10.29	22.46	7,205.46
Non-metro	287	13.67	16.08	950.85

```
    gt() |>
    cols_label(
        counties = "Counties",
        mean_poverty = "Mean poverty rate",
        sd_poverty = "SD poverty rate",
        mean_college = "Mean college share",
        mean_professional = "Mean professional share",
        mean_hsd = "Mean high school share"
    ) |>
    fmt_number(columns = -counties, decimals = 2)
```

```
policy_data |>
    group_by(inmetro) |>
    summarise(
        n = n(),
        mean_poverty = mean(percbelowpoverty),
        mean_college = mean(percollege),
        mean_density = mean(popdensity),
        .groups = "drop"
    ) |>
    gt() |>
    cols_label(
        inmetro = "County type",
        n = "N",
        mean_poverty = "Mean poverty rate",
        mean_college = "Mean college share",
        mean_density = "Mean population density"
    ) |>
    fmt_number(columns = c(mean_poverty, mean_college, mean_density), decimals = 2)
```

These summaries establish scale and heterogeneity, then the association analysis refines the structure in a pairwise and type-aware way.

19.4 Association Analysis for Mixed-Type Policy Variables

We assemble a mixed set of continuous and categorical variables and compute a unified association matrix with the following rules:

- absolute Spearman correlation for continuous-continuous pairs (Spearman 1961),
- eta-squared for continuous-categorical pairs (S. R. A. Fisher 1990; J. Cohen 2013),
- Cramer's V for categorical-categorical pairs (David and Cramer 1947).

This keeps association values on a common nonnegative scale and supports network construction in the next step.

```
analysis_vars <- policy_data |>
    select(
        percbelowpoverty,
        perchsd,
        percollege,
        percprof,
        percwhite,
        percblack,
        log_popdensity,
        inmetro,
        category
    )

cramers_v <- function(x, y) {
    tab <- table(x, y)
    if (min(dim(tab)) < 2) {
        return(NA_real_)
    }

    chi2 <- suppressWarnings(chisq.test(tab, correct = FALSE)$statistic)
    n <- sum(tab)
    k <- min(nrow(tab), ncol(tab))
    as.numeric(sqrt(chi2 / (n * (k - 1))))
}

eta_squared <- function(y, group) {
    df <- tibble(y = y, group = as.factor(group))
    grand_mean <- mean(df$y, na.rm = TRUE)
    ss_total <- sum((df$y - grand_mean)^2, na.rm = TRUE)

    if (ss_total == 0) {
```

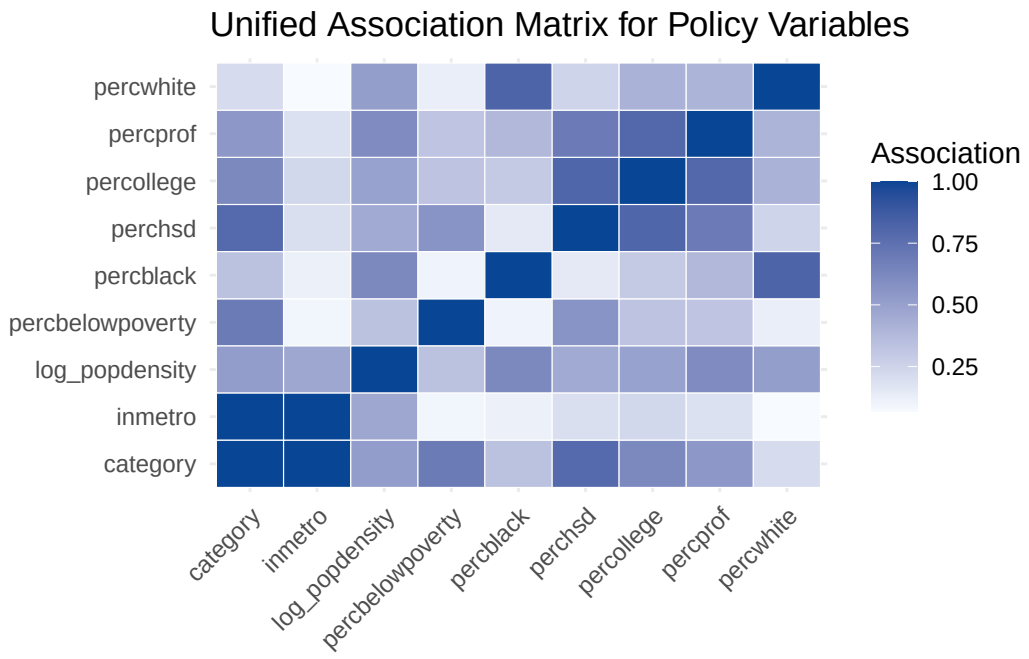
```
        return(NA_real_)
    }

    ss_between <- df |>
        group_by(group) |>
        summarise(n = n(), mean_y = mean(y, na.rm = TRUE), .groups = "drop") |>
        mutate(ss = n * (mean_y - grand_mean)^2) |>
        summarise(ss = sum(ss), .groups = "drop") |>
        pull(ss)

    as.numeric(ss_between / ss_total)
}

association_measure <- function(x, y) {
    if (is.numeric(x) && is.numeric(y)) {
        return(abs(cor(x, y, method = "spearman", use = "pairwise.complete.obs")))
    }

    if (is.factor(x) && is.factor(y)) {
        return(cramers_v(x, y))
    }

    if (is.numeric(x) && is.factor(y)) {
        return(eta_squared(x, y))
    }

    if (is.factor(x) && is.numeric(y)) {
        return(eta_squared(y, x))
    }

    NA_real_
}

var_names <- names(analysis_vars)
p <- length(var_names)

assoc_mat <- matrix(NA_real_, nrow = p, ncol = p, dimnames = list(var_names, var_names))

for (i in seq_len(p)) {
    for (j in seq_len(p)) {
        if (i == j) {
            assoc_mat[i, j] <- 1
```

Variable 1	Variable 2	Association strength
category	inmetro	1.000
percblack	percwhite	0.821
perchsd	percollege	0.810
percollege	percprof	0.797
category	perchsd	0.787
category	percbelowpoverty	0.699
perchsd	percprof	0.695
log_popdensity	percblack	0.626
category	percollege	0.622
log_popdensity	percprof	0.608
percbelowpoverty	perchsd	0.566
category	percprof	0.545

```
        } else {
            assoc_mat[i, j] <- association_measure(analysis_vars[[i]], analysis_vars[[j]])
        }
    }
}

assoc_pairs <- as.data.frame(as.table(assoc_mat), stringsAsFactors = FALSE) |>
    rename(var1 = Var1, var2 = Var2, association = Freq) |>
    filter(var1 < var2) |>
    arrange(desc(association))

assoc_pairs |>
    slice_head(n = 12) |>
    gt() |>
    cols_label(
        var1 = "Variable 1",
        var2 = "Variable 2",
        association = "Association strength"
    ) |>
    fmt_number(columns = association, decimals = 3)
```

```
assoc_plot_data <- as.data.frame(as.table(assoc_mat), stringsAsFactors = FALSE) |>
    rename(var1 = Var1, var2 = Var2, association = Freq)

assoc_plot_data |>
    ggplot(aes(x = var1, y = var2, fill = association)) +
```

```
    geom_tile(color = "white") +
    scale_fill_gradient(low = "#f7fbff", high = "#084594") +
    labs(
        title = "Unified Association Matrix for Policy Variables",
        x = NULL,
        y = NULL,
        fill = "Association"
    ) +
    theme_minimal() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Even in this compact analysis set, we can identify association blocks related to education, occupational structure, and racial composition. This pairwise view is informative, and the network representation helps us see the same structure at a systems level.

19.5 Network Representation of the Association Structure

To avoid a fully dense graph, we retain only associations above the 75th percentile of off-diagonal association strengths. The resulting graph highlights the strongest links while preserving weighted edge information.

Network statistic	Value
Nodes	9.000
Edges	9.000
Density	0.250
Threshold used	0.611

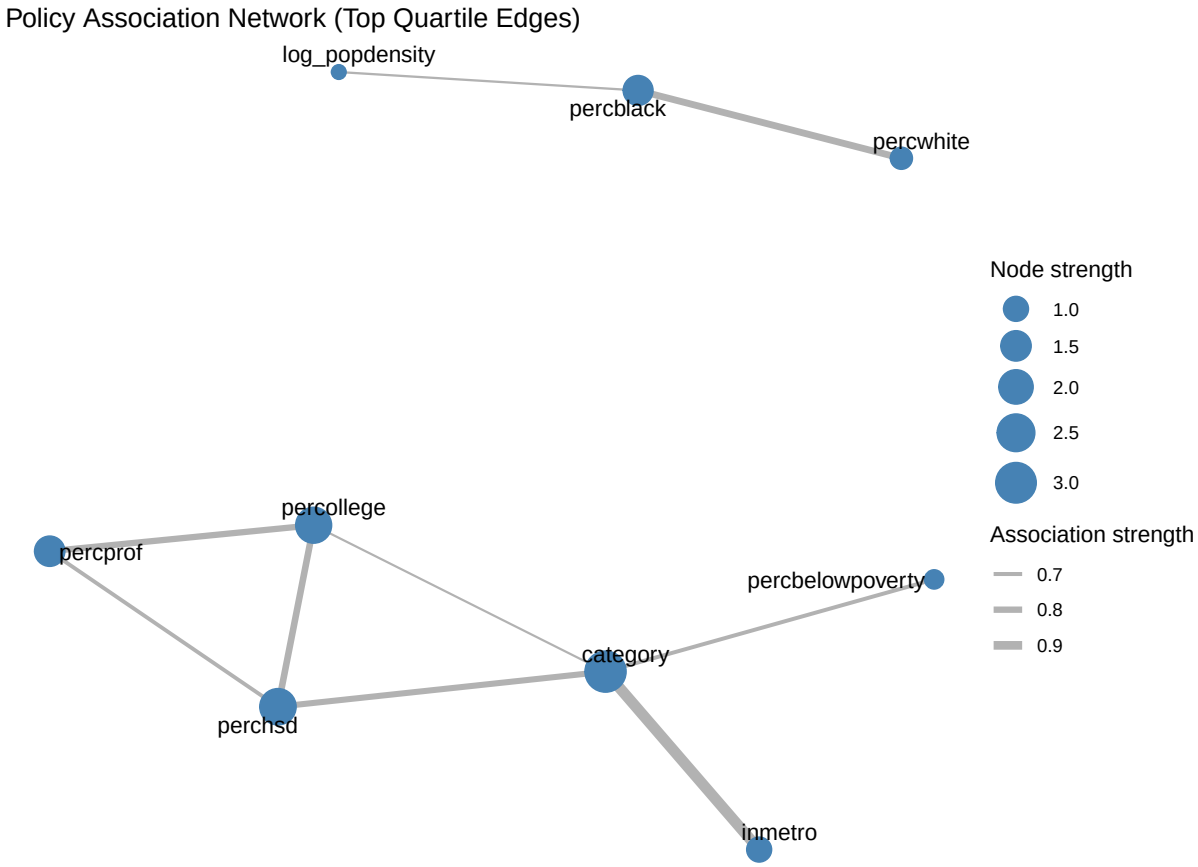
```
threshold <- quantile(assoc_pairs$association, probs = 0.75, na.rm = TRUE)

adjacency_weighted <- assoc_mat
adjacency_weighted[adjacency_weighted < threshold] <- 0
diag(adjacency_weighted) <- 0

policy_net <- graph_from_adjacency_matrix(
    adjacency_weighted,
    mode = "undirected",
    weighted = TRUE,
    diag = FALSE
)

tibble::tibble(
    statistic = c("Nodes", "Edges", "Density", "Threshold used"),
    value = c(
        vcount(policy_net),
        ecount(policy_net),
        edge_density(policy_net),
        as.numeric(threshold)
    )
) |>
    gt() |>
    cols_label(
        statistic = "Network statistic",
        value = "Value"
    ) |>
    fmt_number(columns = value, decimals = 3)
```

```
centrality_tbl <- tibble::tibble(
    variable = names(degree(policy_net)),
    degree = as.numeric(degree(policy_net)),
    strength = as.numeric(strength(policy_net, weights = E(policy_net)$weight)),
    betweenness = as.numeric(betweenness(policy_net, directed = FALSE, normalized = TRUE))
```

Variable	Degree	Weighted degree	Normalized betweenness
category	4	3.108	0.250
perchsd	3	2.293	0.000
percollege	3	2.229	0.107
percprof	2	1.492	0.000
percblack	2	1.447	0.036
inmetro	1	1.000	0.000
percwhite	1	0.821	0.000
percbelowpoverty	1	0.699	0.000
log_popdensity	1	0.626	0.000

```
) |>
    arrange(desc(strength))

centrality_tbl |>
    gt() |>
    cols_label(
        variable = "Variable",
        degree = "Degree",
        strength = "Weighted degree",
        betweenness = "Normalized betweenness"
    ) |>
    fmt_number(columns = c(strength, betweenness), decimals = 3)
```

```
policy_tbl_graph <- as_tbl_graph(policy_net) |>
    activate(nodes) |>
    left_join(centrality_tbl, by = c("name" = "variable"))

ggraph(policy_tbl_graph, layout = "fr") +
    geom_edge_link(aes(width = weight), edge_alpha = 0.6, edge_colour = "gray50") +
    geom_node_point(aes(size = strength), color = "steelblue") +
    geom_node_text(aes(label = name), repel = TRUE, size = 3.8) +
    scale_edge_width(range = c(0.5, 2.5)) +
    scale_size_continuous(range = c(3, 9)) +
    labs(
        title = "Policy Association Network (Top Quartile Edges)",
        edge_width = "Association strength",
        size = "Node strength"
    ) +
    theme_void()
```



The network view supports a policy-oriented reading: variables that are central in this graph are those that connect multiple structural dimensions simultaneously. This can help prioritize which indicators are monitored together in reporting dashboards.

19.6 Tree-Based Segmentation of County Contexts

Pairwise and network summaries describe structure globally. We now turn to segmentation, using a regression tree to partition counties into interpretable groups with different average poverty rates.

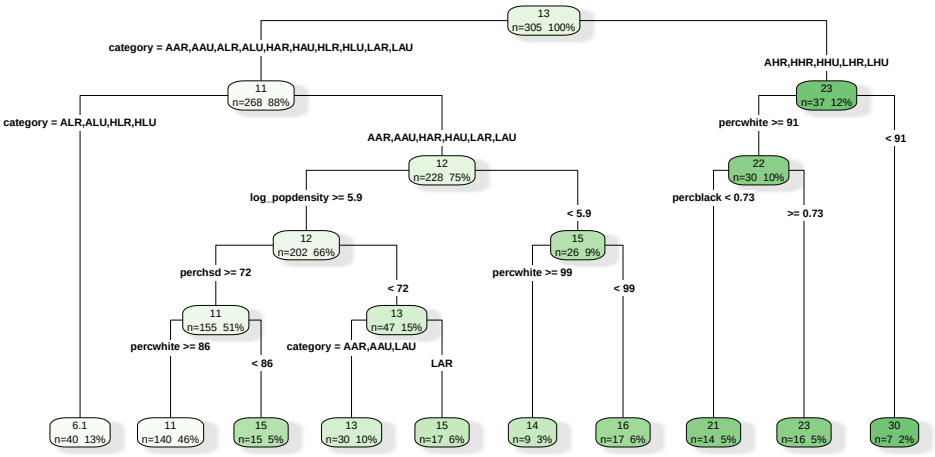
```
set.seed(123)
idx_train <- sample(seq_len(nrow(policy_data)), size = 0.7 * nrow(policy_data))

train_policy <- policy_data[idx_train, ]
test_policy <- policy_data[-idx_train, ]

policy_formula <- percbelowpoverty ~
```

```
    perchsd + percollege + percprof + percwhite + percblack +
    log_popdensity + inmetro + category

tree_fit <- rpart(
    formula = policy_formula,
    data = train_policy,
    method = "anova",
    control = rpart.control(minsplit = 20, cp = 0.001, maxdepth = 5)
)

best_cp <- tree_fit$cptable[which.min(tree_fit$cptable[, "xerror"]), "CP"]
pruned_tree <- prune(tree_fit, cp = best_cp)

best_cp
```

```
[1] 0.001170203
```

```
rpart.plot(
    pruned_tree,
    type = 4,
    extra = 101,
    fallen.leaves = TRUE,
    box.palette = "Greens",
    branch.lty = 1,
    shadow.col = "gray90",
    main = "Policy Segmentation Tree for County Poverty Rates"
)
```

Policy Segmentation Tree for County Poverty Rates



To keep the segmentation grounded, we compare out-of-sample RMSE for the pruned tree and a random forest fit on the same training split. The goal is not model competition, but a calibration check for segmentation stability.

```
rmse <- function(actual, predicted) {
    sqrt(mean((actual - predicted)^2))
}

tree_pred_test <- as.numeric(predict(pruned_tree, newdata = test_policy))
tree_rmse <- rmse(test_policy$percbelowpoverty, tree_pred_test)

set.seed(123)
rf_fit <- randomForest(
    formula = policy_formula,
    data = train_policy,
    ntree = 500,
    mtry = 3,
    importance = TRUE
)

rf_pred_test <- as.numeric(predict(rf_fit, newdata = test_policy))
rf_rmse <- rmse(test_policy$percbelowpoverty, rf_pred_test)

tibble::tibble(
    model = c("Pruned regression tree", "Random forest"),
    holdout_rmse = c(tree_rmse, rf_rmse)
```

Model	Holdout RMSE
Pruned regression tree	2.440
Random forest	2.197

```
) |>
    gt() |>
    cols_label(
        model = "Model",
        holdout_rmse = "Holdout RMSE"
    ) |>
    fmt_number(columns = holdout_rmse, decimals = 3)
```

The tree remains our primary segmentation tool because it yields explicit rules. The ensemble result helps us gauge whether the tree captures a substantial part of the descriptive signal.

19.7 Segment Profiles and Policy Interpretation

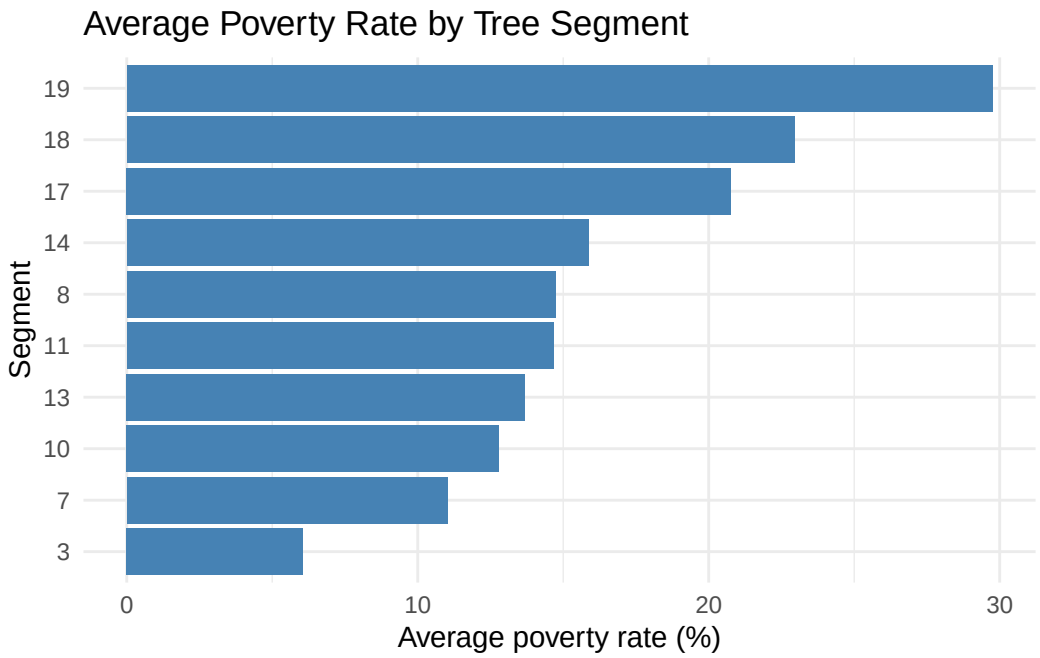
We summarize the leaves of the pruned tree on the training sample. The resulting profiles can support differentiated policy interpretation, for example by identifying segments where poverty is jointly associated with education and labor-market composition indicators.

```
train_with_leaf <- train_policy |>
    mutate(leaf = factor(pruned_tree$where))

segment_tbl <- train_with_leaf |>
    group_by(leaf) |>
    summarise(
        n = n(),
        mean_poverty = mean(percbelowpoverty),
        mean_college = mean(percollege),
        mean_prof = mean(percprof),
        mean_hsd = mean(perchsd),
        mean_density = mean(popdensity),
        metro_share = mean(inmetro == "Metro"),
        .groups = "drop"
    ) |>
    arrange(desc(mean_poverty))

segment_tbl |>
```

Segment	N	Mean poverty rate	Mean college share	Mean professional share	Mean high school
19	7	29.75	16.06	4.00	
18	16	22.94	17.61	5.24	
17	14	20.76	11.82	2.76	
14	17	15.88	16.44	3.47	
8	15	14.73	25.60	7.17	
11	17	14.66	12.04	2.92	
13	9	13.69	15.18	2.86	
10	30	12.80	14.38	3.32	
7	140	11.03	18.88	4.50	
3	40	6.05	21.62	5.26	

```
    gt() |>
    cols_label(
        leaf = "Segment",
        n = "N",
        mean_poverty = "Mean poverty rate",
        mean_college = "Mean college share",
        mean_prof = "Mean professional share",
        mean_hsd = "Mean high school share",
        mean_density = "Mean population density",
        metro_share = "Metro share"
    ) |>
    fmt_number(
        columns = c(mean_poverty, mean_college, mean_prof, mean_hsd, mean_density, metro_shar
        decimals = 2
    )
```

```
segment_tbl |>
    ggplot(aes(x = reorder(leaf, mean_poverty), y = mean_poverty)) +
    geom_col(fill = "steelblue") +
    coord_flip() +
    labs(
        title = "Average Poverty Rate by Tree Segment",
        x = "Segment",
        y = "Average poverty rate (%)"
    ) +
    theme_minimal()
```



In policy settings, this kind of segmentation can help organize discussion around differentiated contexts rather than one average county profile. It can also support resource allocation conversations when combined with program costs and feasibility constraints.

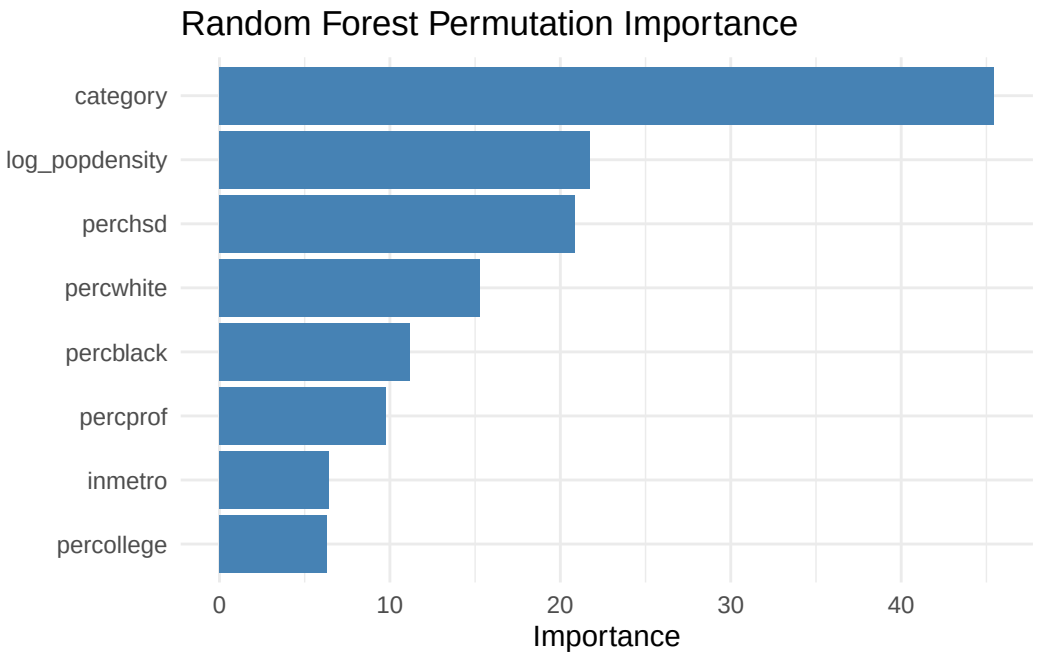
19.8 Ensemble-Based Variable Stability Check

As discussed in the ensemble chapter (Chapter 12), variable importance from random forests can complement tree rules by indicating which predictors consistently contribute across many bootstrap samples.

```
importance_tbl <- tibble::tibble(
    variable = rownames(importance(rf_fit, type = 1)),
    rf_perm_importance = as.numeric(importance(rf_fit, type = 1)[, 1])
) |>
    arrange(desc(rf_perm_importance))

importance_tbl |>
    gt() |>
    cols_label(
        variable = "Variable",
        rf_perm_importance = "Permutation importance"
    ) |>
    fmt_number(columns = rf_perm_importance, decimals = 3)
```

Variable	Permutation importance
category	45.456
log_popdensity	21.757
perchsd	20.846
percwhite	15.286
percblack	11.185
percprof	9.756
inmetro	6.430
percollege	6.327

```
importance_tbl |>
    ggplot(aes(x = reorder(variable, rf_perm_importance), y = rf_perm_importance)) +
    geom_col(fill = "steelblue") +
    coord_flip() +
    labs(
        title = "Random Forest Permutation Importance",
        x = NULL,
        y = "Importance"
    ) +
    theme_minimal()
```



When tree split variables and ensemble importance rankings point in similar directions, policy

interpretation can be communicated with greater confidence. When they differ, that difference itself is informative and may suggest interaction effects or near-substitute predictors.

19.9 Communicating Descriptive Findings in Policy Work

This workflow supports several practical communication outputs:

- a concise association ranking that clarifies which indicators move together,
- a network map that highlights structurally central indicators,
- a segment table that turns model structure into county profiles,
- an ensemble stability check to contextualize split-based conclusions.

For non-technical stakeholders, these outputs are often easier to discuss than raw model objects. At the same time, careful wording remains important. We describe patterns in observed data and model behavior, while avoiding causal language unless a separate identification strategy is in place.

19.10 Limitations and Reproducibility Notes

Three limits are worth keeping in view.

1. This is an observational dataset, so the analysis supports descriptive interpretation, not causal attribution.
2. County-level indicators aggregate heterogeneous populations, which can mask within-county variation.
3. Segmentation rules are sample-dependent, so stability checks and sensitivity analyses are useful in applied work.

For reproducibility, all random components in this chapter use explicit seeds. In practice, it is often helpful to report sensitivity across several seeds and threshold choices.

19.11 Summary and Key Takeaways

- Mixed-type association measures provide a coherent starting point for policy tabular data.
- Network representations help prioritize variables that connect multiple structural domains.
- Regression trees translate multivariate structure into interpretable county segments.
- Ensemble importance can be used as a stability lens for segmentation narratives.
- Descriptive findings are most useful when presented as context for policy deliberation, not as definitive causal evidence.

19.12 Looking Ahead

This policy case study emphasized association structure, networks, and segmentation in administrative-style tabular data. The next case study shifts to public health and epidemiological analysis, where interactive exploration and interpretable machine learning will play a more central role in communicating results to broad stakeholder groups.

20 Case Study: Public Health and Epidemiological Data

20.1 Introduction: Integrating Exploration and Explanation in Public Health

Across the book, we developed a descriptive workflow that moves from type-aware summaries and associations to interactive exploration, interpretable machine learning, and communication-oriented visualization. This case study brings these pieces together in a public health setting.

The focus is intentionally practical. We work with a diabetes screening dataset and examine how interactive exploration and interpretable model summaries can support epidemiological understanding. We emphasize descriptive interpretation, transparent assumptions, and communication choices for mixed technical and non-technical audiences.

20.2 Public Health Context and Analytical Questions

We use the Pima diabetes data available in `MASS` (`Pima.tr` and `Pima.te`). The dataset includes physiological measurements and a binary diabetes outcome. While the sample is modest, it illustrates a common public health workflow where we combine exploratory visualization with interpretable probabilistic modeling.

We organize the analysis around four questions:

1. How does diabetes prevalence vary across key demographic and clinical strata?
2. How can an interactive Shiny interface support rapid epidemiological exploration?
3. What do partial dependence and Shapley-based summaries reveal about model behavior?
4. How can we communicate these findings clearly to non-technical stakeholders?

20.3 Data Assembly and Baseline Epidemiological Profile

We merge the training and test partitions into one analysis table, then create a binary indicator used in prevalence calculations.

Sample size	Diabetes prevalence	Mean glucose	Mean BMI	Mean age
532	33.3%	121.03	32.89	31.61

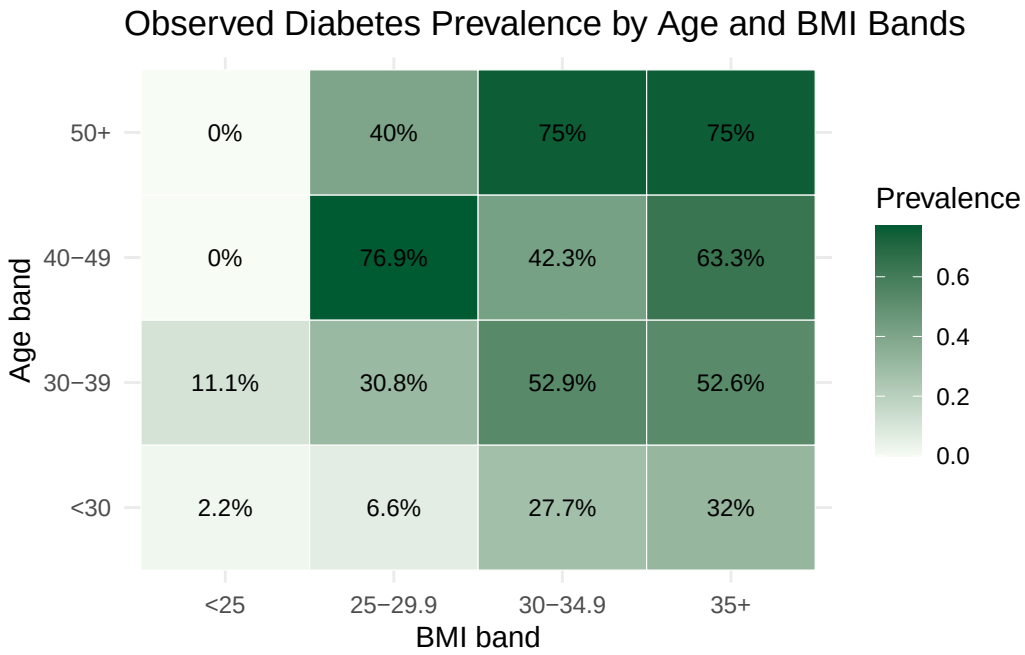
```
data(Pima.tr, package = "MASS")
data(Pima.te, package = "MASS")

health_data <- bind_rows(Pima.tr, Pima.te) |>
    mutate(
        type = factor(type, levels = c("No", "Yes")),
        diabetes = ifelse(type == "Yes", 1, 0)
    )

nrow(health_data)
```

```
[1] 532
```

```
health_overview <- health_data |>
    summarise(
        n = n(),
        diabetes_prevalence = mean(diabetes),
        mean_glu = mean(glu),
        mean_bmi = mean(bmi),
        mean_age = mean(age)
    )

health_overview |>
    gt() |>
    cols_label(
        n = "Sample size",
        diabetes_prevalence = "Diabetes prevalence",
        mean_glu = "Mean glucose",
        mean_bmi = "Mean BMI",
        mean_age = "Mean age"
    ) |>
    fmt_percent(columns = diabetes_prevalence, decimals = 1) |>
    fmt_number(columns = c(mean_glu, mean_bmi, mean_age), decimals = 2)
```

To establish epidemiological context, we examine prevalence jointly by age and BMI bands.

```r
epi_grid <- health_data |>
    mutate(
        age_band = cut(
            age,
            breaks = c(-Inf, 30, 40, 50, Inf),
            labels = c("<30", "30-39", "40-49", "50+"),
            right = FALSE
        ),
        bmi_band = cut(
            bmi,
            breaks = c(-Inf, 25, 30, 35, Inf),
            labels = c("<25", "25-29.9", "30-34.9", "35+"),
            right = FALSE
        )
    ) |>
    group_by(age_band, bmi_band) |>
    summarise(
        n = n(),
        prevalence = mean(diabetes),
        .groups = "drop"
    )

epi_grid |>
    ggplot(aes(x = bmi_band, y = age_band, fill = prevalence)) +
    geom_tile(color = "white") +
    geom_text(aes(label = paste0(round(100 * prevalence, 1), "%")), size = 3.3) +
    scale_fill_gradient(low = "#f7fcf5", high = "#005a32") +
    labs(
        title = "Observed Diabetes Prevalence by Age and BMI Bands",
        x = "BMI band",
        y = "Age band",
        fill = "Prevalence"
    ) +
    theme_minimal()
```



This baseline view highlights heterogeneity that motivates the interactive and model-based steps that follow.

20.4 Interactive Exploration with a Shiny Workflow

As established in Chapter 7 and Chapter 8, interactive tools transform association analysis from a static snapshot into a dynamic conversation with data. Rather than each new question requiring code modification and recomputation, a Shiny interface allows filtering by subgroup, switching indicators, and adjusting parameters in real time. This iterative workflow is particularly valuable in public health settings, where the relevant question often shifts depending on which stratum or indicator the analyst focuses on.

The core Shiny architecture for a health indicator explorer follows the same reactive pattern developed in Chapter 7: a UI with controls for indicator selection and optional subgroup filtering, and a server that recomputes and re-renders plots reactively whenever an input changes.

```
library(shiny)

# Minimal structure of the health indicator explorer
ui <- fluidPage(
    titlePanel("Public Health Indicator Explorer"),
    sidebarLayout(
```

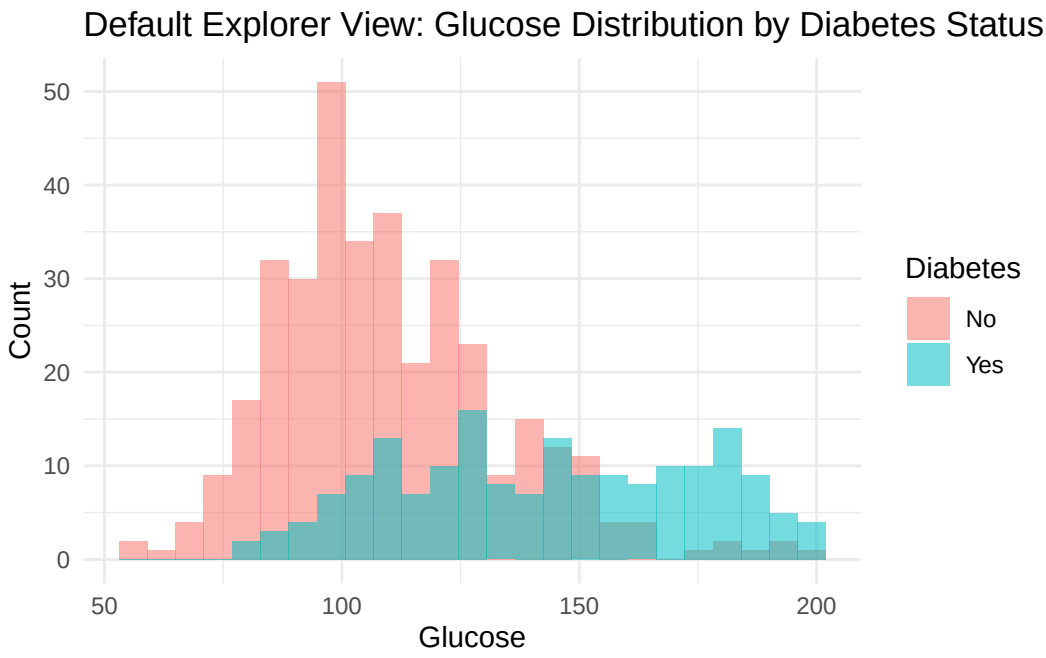
```
        sidebarPanel(
            selectInput(
                inputId  = "indicator",
                label    = "Select indicator",
                choices  = c("glu", "bp", "skin", "bmi", "ped", "age", "npreg"),
                selected = "glu"
            ),
            checkboxGroupInput(
                inputId  = "diabetes_filter",
                label    = "Diabetes status",
                choices  = c("No", "Yes"),
                selected = c("No", "Yes")
            )
        ),
        mainPanel(
            plotOutput("distribution_plot"),
            plotOutput("comparison_plot")
        )
    )
)

server <- function(input, output) {
    # Reactive subset: updates whenever indicator or filter changes
    filtered_data <- reactive({
        health_data |>
            filter(as.character(type) %in% input$diabetes_filter)
    })

    output$distribution_plot <- renderPlot({
        filtered_data() |>
            ggplot(aes(x = .data[[input$indicator]], fill = type)) +
            geom_histogram(alpha = 0.55, position = "identity", bins = 25) +
            labs(x = input$indicator, y = "Count", fill = "Diabetes") +
            theme_minimal()
    })

    output$comparison_plot <- renderPlot({
        filtered_data() |>
            ggplot(aes(x = type, y = .data[[input$indicator]], fill = type)) +
            geom_boxplot(alpha = 0.7, width = 0.55) +
            labs(x = "Diabetes status", y = input$indicator) +
            theme_minimal() +
```

```
            theme(legend.position = "none")
    })
}
```

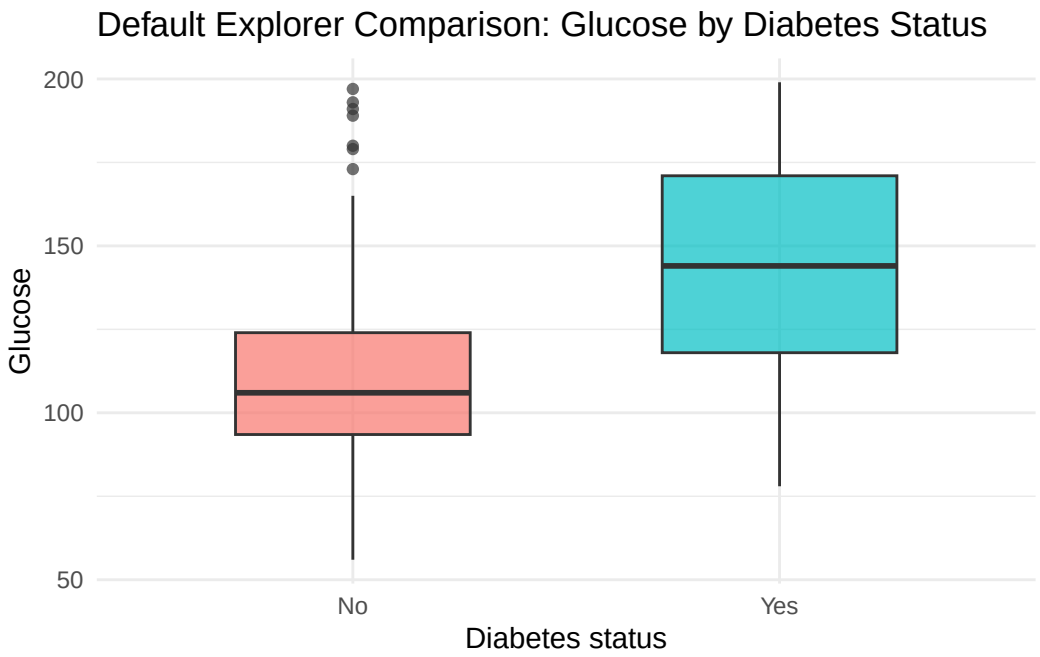
The reactive expression `filtered_data()` is the organizing principle: both plots depend on it, so changing either the indicator or the diabetes filter triggers a single re-evaluation of the subset and automatic re-rendering of all downstream outputs. This is the same reactive dependency graph pattern introduced in Chapter 7.

We now show a static default view equivalent to one state of the live app, then embed the deployed application.

```
default_indicator <- "glu"

health_data |>
    ggplot(aes(x = .data[[default_indicator]], fill = type)) +
    geom_histogram(alpha = 0.55, position = "identity", bins = 25) +
    labs(
        title = "Default Explorer View: Glucose Distribution by Diabetes Status",
        x = "Glucose",
        y = "Count",
        fill = "Diabetes"
    ) +
    theme_minimal()
```



```
health_data |>
    ggplot(aes(x = type, y = glu, fill = type)) +
    geom_boxplot(alpha = 0.7, width = 0.55) +
    labs(
        title = "Default Explorer Comparison: Glucose by Diabetes Status",
        x = "Diabetes status",
        y = "Glucose",
        fill = "Diabetes"
    ) +
    theme_minimal() +
    theme(legend.position = "none")
```



In non-HTML output formats, the live application cannot be embedded. Access it at https://antoinesoetewey.shinyapps.io/public-health-indicator-explorer/.

The full source code of the application is available on GitHub.

This app-level workflow is useful when we want rapid subgroup checks before committing to a fixed modeling specification. The interactive layer also supports communication with non-technical stakeholders, who can explore the data themselves without reading code, aligning with the communication principles developed in Chapter 9.

20.5 Interpretable Model for Descriptive Risk Mapping

We now fit a random forest classifier to obtain smooth probability estimates and then interpret those estimates with partial dependence and Shapley-based summaries. The objective is descriptive mapping of model behavior, not predictive competition.

```
set.seed(123)
idx_train <- sample(seq_len(nrow(health_data)), size = 0.7 * nrow(health_data))

train_health <- health_data[idx_train, ]
test_health <- health_data[-idx_train, ]

health_formula <- type ~ npreg + glu + bp + skin + bmi + ped + age

rf_health <- randomForest(
    formula = health_formula,
    data = train_health,
    ntree = 600,
    mtry = 3,
    importance = TRUE
)

predict_prob <- function(model, newdata) {
    as.numeric(predict(model, newdata = newdata, type = "prob")[, "Yes"])
}

test_pred_prob <- predict_prob(rf_health, test_health)
test_pred_class <- factor(
    ifelse(test_pred_prob >= 0.5, "Yes", "No"),
    levels = levels(test_health$type)
)

accuracy <- mean(test_pred_class == test_health$type)
brier <- mean((test_health$diabetes - test_pred_prob)^2)

tibble::tibble(
    metric = c("Holdout accuracy", "Holdout Brier score", "Observed prevalence", "Mean predi
    value = c(
        accuracy,
        brier,
        mean(test_health$diabetes),
        mean(test_pred_prob)
```

Metric	Value
Holdout accuracy	0.800
Holdout Brier score	0.141
Observed prevalence	0.356
Mean predicted probability	0.335

Observed	No	Yes
No	91	12
Yes	20	37

```
    )
) |>
    gt() |>
    cols_label(metric = "Metric", value = "Value") |>
    fmt_number(columns = value, decimals = 3)
```

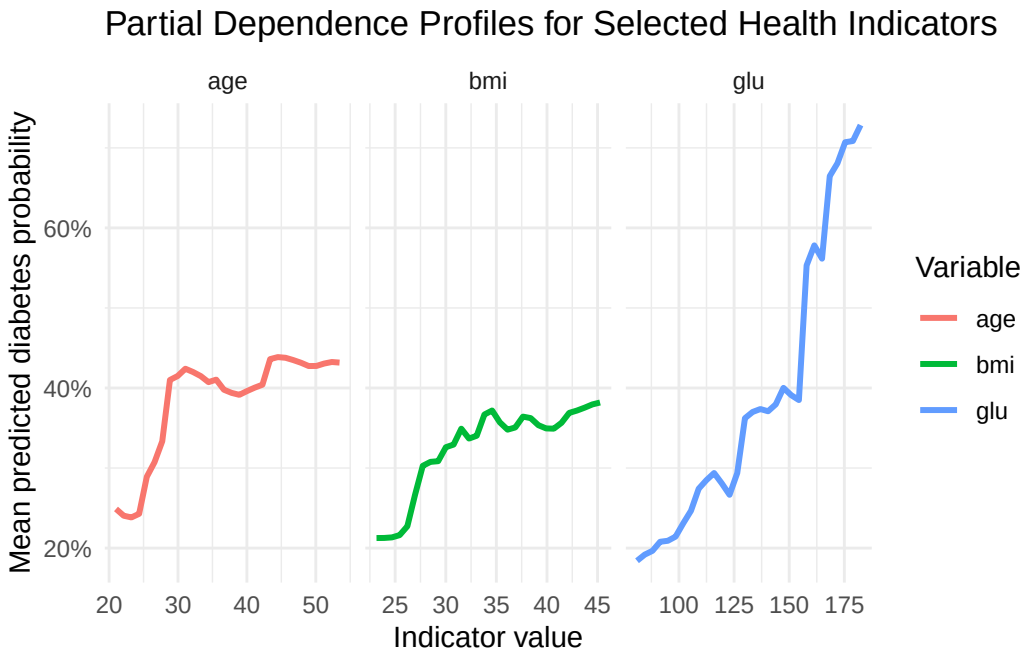
```
conf_mat <- table(Observed = test_health$type, Predicted = test_pred_class)

conf_mat |>
    as.data.frame.matrix() |>
    tibble::rownames_to_column(var = "Observed") |>
    gt()
```

20.6 Partial Dependence Profiles

To study global model behavior, we compute partial dependence curves for glucose, BMI, and age. These are average model responses over a grid of values while integrating over the empirical distribution of other features.

```
partial_dependence_prob <- function(model, data, variable, grid_size = 30) {
    grid <- seq(
        as.numeric(quantile(data[[variable]], 0.05)),
        as.numeric(quantile(data[[variable]], 0.95)),
        length.out = grid_size
    )

    pd_rows <- lapply(grid, function(v) {
```

```
        data_mod <- data
        data_mod[[variable]] <- v
        tibble::tibble(
            value = v,
            mean_pred_prob = mean(predict_prob(model, data_mod))
        )
    })

    bind_rows(pd_rows) |>
        mutate(variable = variable)
}

pd_tbl <- bind_rows(
    partial_dependence_prob(rf_health, train_health, "glu"),
    partial_dependence_prob(rf_health, train_health, "bmi"),
    partial_dependence_prob(rf_health, train_health, "age")
)

pd_tbl |>
    ggplot(aes(x = value, y = mean_pred_prob, color = variable)) +
    geom_line(linewidth = 1.05) +
    facet_wrap(~ variable, scales = "free_x") +
    scale_y_continuous(labels = function(x) paste0(round(100 * x), "%")) +
    labs(
        title = "Partial Dependence Profiles for Selected Health Indicators",
        x = "Indicator value",
        y = "Mean predicted diabetes probability",
        color = "Variable"
    ) +
    theme_minimal()
```




```
pd_summary_tbl <- pd_tbl |>
    group_by(variable) |>
    summarise(
        pd_low = first(mean_pred_prob),
        pd_high = last(mean_pred_prob),
        pd_change = pd_high - pd_low,
        .groups = "drop"
    ) |>
    arrange(desc(pd_change))

pd_summary_tbl |>
    gt() |>
    cols_label(
        variable = "Variable",
        pd_low = "Predicted probability at low grid value",
        pd_high = "Predicted probability at high grid value",
        pd_change = "Change"
    ) |>
    fmt_percent(columns = c(pd_low, pd_high, pd_change), decimals = 1)
```

In a public health framing, these profiles are useful for communicating directional gradients without reducing interpretation to a single coefficient.

Variable	Predicted probability at low grid value	Predicted probability at high grid value	Change
glu	18.4%	72.8%	54.4%
age	24.9%	43.2%	18.3%
bmi	21.3%	38.2%	16.9%

20.7 Local and Global Shapley-Based Explanations

Partial dependence provides global structure. We complement it with local additive decomposition, following the logic introduced in the chapter on Shapley values (Chapter 16).

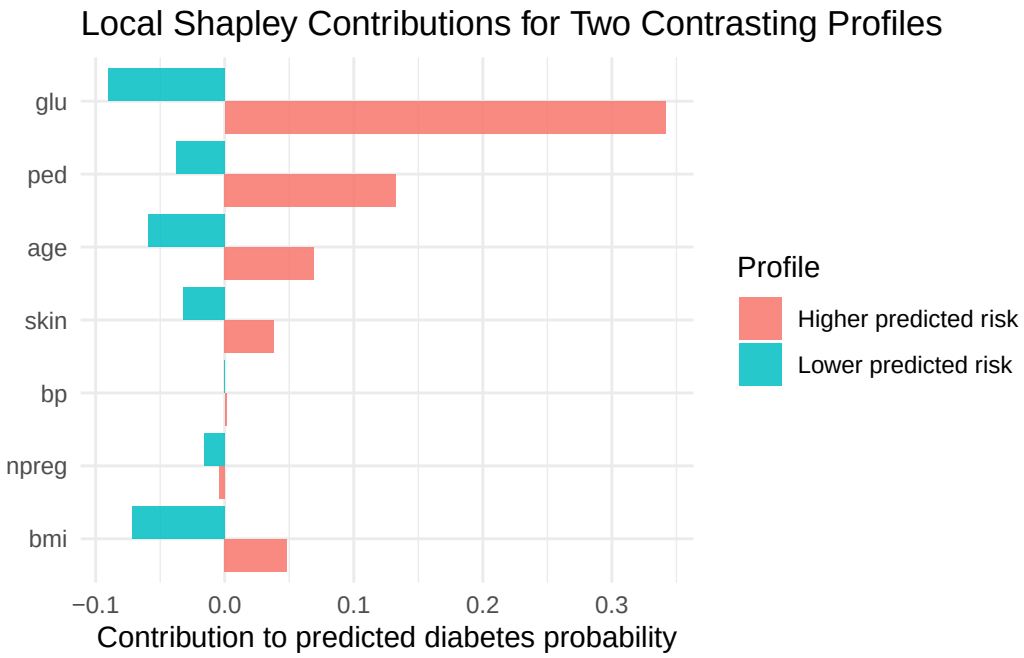
```
shapley_single_mc_prob <- function(model,
                                   x_row,
                                   background,
                                   features,
                                   n_permutations = 300,
                                   seed = 123) {
    set.seed(seed)
    contributions <- setNames(rep(0, length(features)), features)

    for (b in seq_len(n_permutations)) {
        perm <- sample(features)
        z <- background[sample(seq_len(nrow(background)), 1), features, drop = FALSE]

        current <- z
        pred_prev <- predict_prob(model, current)

        for (f in perm) {
            current[[f]] <- x_row[[f]]
            pred_new <- predict_prob(model, current)
            contributions[f] <- contributions[f] + (pred_new - pred_prev)
            pred_prev <- pred_new
        }
    }

    contributions / n_permutations
}

feature_set <- c("npreg", "glu", "bp", "skin", "bmi", "ped", "age")
background <- train_health[, feature_set, drop = FALSE]

test_scored <- test_health |>
```

```
    mutate(pred_prob = test_pred_prob)

idx_higher <- which.max(test_scored$pred_prob)
idx_lower <- which.min(test_scored$pred_prob)

phi_higher <- shapley_single_mc_prob(
    model = rf_health,
    x_row = test_scored[idx_higher, feature_set, drop = FALSE],
    background = background,
    features = feature_set,
    n_permutations = 350,
    seed = 1001
)

phi_lower <- shapley_single_mc_prob(
    model = rf_health,
    x_row = test_scored[idx_lower, feature_set, drop = FALSE],
    background = background,
    features = feature_set,
    n_permutations = 350,
    seed = 1002
)

baseline_prob <- mean(predict_prob(rf_health, background))

local_accuracy_tbl <- tibble::tibble(
    profile = c("Higher predicted risk profile", "Lower predicted risk profile"),
    prediction = c(test_scored$pred_prob[idx_higher], test_scored$pred_prob[idx_lower]),
    baseline_plus_shap = c(
        baseline_prob + sum(phi_higher),
        baseline_prob + sum(phi_lower)
    ),
    difference = prediction - baseline_plus_shap
)

local_accuracy_tbl |>
    gt() |>
    cols_label(
        profile = "Profile",
        prediction = "Predicted probability",
        baseline_plus_shap = "Baseline + sum(Shapley)",
        difference = "Difference"
```

Profile	Predicted probability	Baseline + sum(Shapley)	Difference
Higher predicted risk profile	94.83%	95.06%	-0.23%
Lower predicted risk profile	0.00%	1.71%	-1.71%

```
    ) |>
    fmt_percent(columns = c(prediction, baseline_plus_shap, difference), decimals = 2)
```

```
local_shap_tbl <- tibble::tibble(
    feature = feature_set,
    higher_profile = as.numeric(phi_higher[feature_set]),
    lower_profile = as.numeric(phi_lower[feature_set])
) |>
    pivot_longer(
        cols = c(higher_profile, lower_profile),
        names_to = "profile",
        values_to = "contribution"
    ) |>
    mutate(
        profile = recode(
            profile,
            higher_profile = "Higher predicted risk",
            lower_profile = "Lower predicted risk"
        )
    )

local_shap_tbl |>
    ggplot(aes(x = reorder(feature, contribution), y = contribution, fill = profile)) +
    geom_col(position = "dodge", alpha = 0.85) +
    coord_flip() +
    labs(
        title = "Local Shapley Contributions for Two Contrasting Profiles",
        x = NULL,
        y = "Contribution to predicted diabetes probability",
        fill = "Profile"
    ) +
    theme_minimal()
```



For a global relevance view, we estimate mean absolute Shapley contributions on a subset of holdout observations.

```
set.seed(456)
ids_explain <- sample(seq_len(nrow(test_health)), size = min(40, nrow(test_health)))

shap_many <- lapply(seq_along(ids_explain), function(k) {
    i <- ids_explain[k]
    phi_i <- shapley_single_mc_prob(
        model = rf_health,
        x_row = test_health[i, feature_set, drop = FALSE],
        background = background,
        features = feature_set,
        n_permutations = 150,
        seed = 500 + k
    )
    as.data.frame(as.list(phi_i))
})

shap_many_df <- bind_rows(shap_many)

global_shap_tbl <- tibble::tibble(
    feature = feature_set,
    mean_abs_shap = sapply(feature_set, function(v) mean(abs(shap_many_df[[v]])))
```

Feature	Mean absolute Shapley contribution
glu	10.68%
age	6.97%
bmi	4.80%
ped	4.41%
npreg	3.14%
skin	3.04%
bp	1.38%

```
) |>
    arrange(desc(mean_abs_shap))

global_shap_tbl |>
    gt() |>
    cols_label(
        feature = "Feature",
        mean_abs_shap = "Mean absolute Shapley contribution"
    ) |>
    fmt_percent(columns = mean_abs_shap, decimals = 2)
```

20.8 Communicating Results to Non-Technical Stakeholders

In line with the communication principles developed earlier in the book, we translate model outputs into message-oriented summaries.

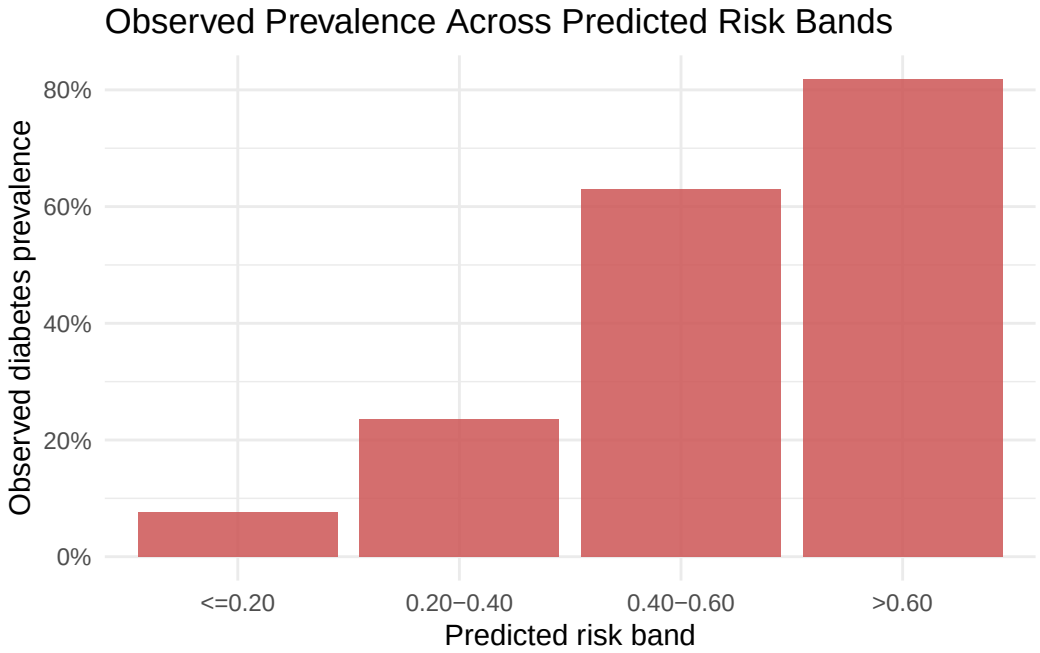
First, we create risk bands from holdout predicted probabilities and compare them with observed outcome frequencies.

```
risk_band_tbl <- test_scored |>
    mutate(
        risk_band = cut(
            pred_prob,
            breaks = c(-Inf, 0.20, 0.40, 0.60, Inf),
            labels = c("<=0.20", "0.20-0.40", "0.40-0.60", ">0.60")
        )
    ) |>
    group_by(risk_band) |>
    summarise(
        n = n(),
```

Predicted risk band	N	Observed diabetes prevalence	Mean glucose	Mean BMI
<=0.20	66	7.6%	98.80	29.03
0.20-0.40	34	23.5%	113.47	32.26
0.40-0.60	27	63.0%	134.44	36.19
>0.60	33	81.8%	161.18	37.25

```
        observed_prevalence = mean(diabetes),
        mean_glucose = mean(glu),
        mean_bmi = mean(bmi),
        .groups = "drop"
    )

risk_band_tbl |>
    gt() |>
    cols_label(
        risk_band = "Predicted risk band",
        n = "N",
        observed_prevalence = "Observed diabetes prevalence",
        mean_glucose = "Mean glucose",
        mean_bmi = "Mean BMI"
    ) |>
    fmt_percent(columns = observed_prevalence, decimals = 1) |>
    fmt_number(columns = c(mean_glucose, mean_bmi), decimals = 2)
```

```
risk_band_tbl |>
    ggplot(aes(x = risk_band, y = observed_prevalence)) +
    geom_col(fill = "indianred3", alpha = 0.85) +
    scale_y_continuous(labels = function(x) paste0(round(100 * x), "%")) +
    labs(
        title = "Observed Prevalence Across Predicted Risk Bands",
        x = "Predicted risk band",
        y = "Observed diabetes prevalence"
    ) +
    theme_minimal()
```



Second, we summarize the analytical narrative in a message-evidence format that can be reused in technical briefs or public health dashboards.

Message 1. Higher glucose values are associated with higher model-estimated diabetes probability. The supporting evidence is that the partial dependence profile for glucose shows the largest increase from low to high values.

Message 2. BMI and age contribute to substantial between-profile risk heterogeneity. The supporting evidence is that Shapley summaries indicate meaningful local and global contributions from BMI and age.

Message 3. Predicted risk strata correspond to increasing observed prevalence in holdout data. The supportifng evidence is that observed prevalence increases across ordered predicted-probability bands.

20.9 Limitations and Reproducibility Notes

Three considerations remain central.

1. The analysis is observational and descriptive, so interpretation should avoid causal claims.
2. The sample is specific and relatively compact, which limits transportability.
3. Shapley approximations are Monte Carlo estimates, so small contributions can vary with sampling settings.

All random components use explicit seeds. In practice, it is often useful to check sensitivity across additional resamples and alternative model classes.

20.10 Summary and Key Takeaways

- Interactive Shiny exploration helps identify stable subgroup patterns before model fitting.
- Partial dependence provides a clear global view of predictor-response gradients.
- Shapley-based decomposition complements global profiles with case-level interpretation.
- Risk-band summaries and message-evidence tables support communication with non-technical audiences.
- The full workflow remains descriptive and should be communicated with corresponding caution.

20.11 Looking Ahead

This chapter centered on interactive exploration and interpretable machine learning in a public health context. The next case study shifts to business analytics, where AutoML and automated feature engineering will be used as descriptive instruments for customer insight generation.

21 Case Study: Business Analytics and Customer Insights

21.1 Introduction: AutoML and Automated Feature Engineering for Customer Insight

Across the previous chapters, we developed a descriptive analytical toolkit that moves from exploratory association analysis through segmentation to interpretable model summaries and communication. This case study applies the final two layers of that toolkit to a business setting: automated feature engineering, which structures the search for informative derived predictors, and AutoML-style model comparison, which surveys multiple model families under a common evaluation protocol.

The analytical emphasis is intentionally non-predictive. In business analytics, the goal is often to understand *which* customer behaviors drive value heterogeneity and *how* they interact, not to optimize a prediction system. AutoML and automated feature engineering support this by making the search process explicit and transparent rather than driven by ad hoc choices. They allow us to document what the data supports across a range of model families and feature representations, rather than committing to a single handcrafted specification.

21.2 Business Context and Customer Analytics Questions

We analyze a subscription business with recurring transaction data. The analytical questions we address are:

1. How do customer transaction patterns vary across segments?
2. Which transformation and interaction terms most improve our understanding of customer value, according to a systematic candidate screening procedure (following Chapter 18)?
3. How does a multi-model AutoML comparison reveal which model family best captures customer value structure (following Chapter 17)?
4. What interpretable customer profiles emerge from variable importance rankings?
5. How can we communicate customer insights to product and marketing teams?

21.3 Customer Data and Baseline Profiling

We construct a customer dataset from transaction-level data aggregated to customer level. For reproducibility, we simulate transaction data that reflects realistic e-commerce patterns.

```
set.seed(789)

n_customers <- 500

customer_transactions <- expand_grid(
    customer_id = 1:n_customers,
    month = 1:12
) |>
    mutate(
        n_trans = rpois(n(), lambda = 2 + rnorm(n(), mean = 0, sd = 0.5)) + 1,
        avg_transaction_value = runif(n(), min = 20, max = 200),
        trans_amount = n_trans * avg_transaction_value
    ) |>
    slice_sample(prop = 0.7)

nrow(customer_transactions)
```

```
[1] 4200
```

From transaction data, we aggregate to customer-level metrics that serve as both features and communication targets.

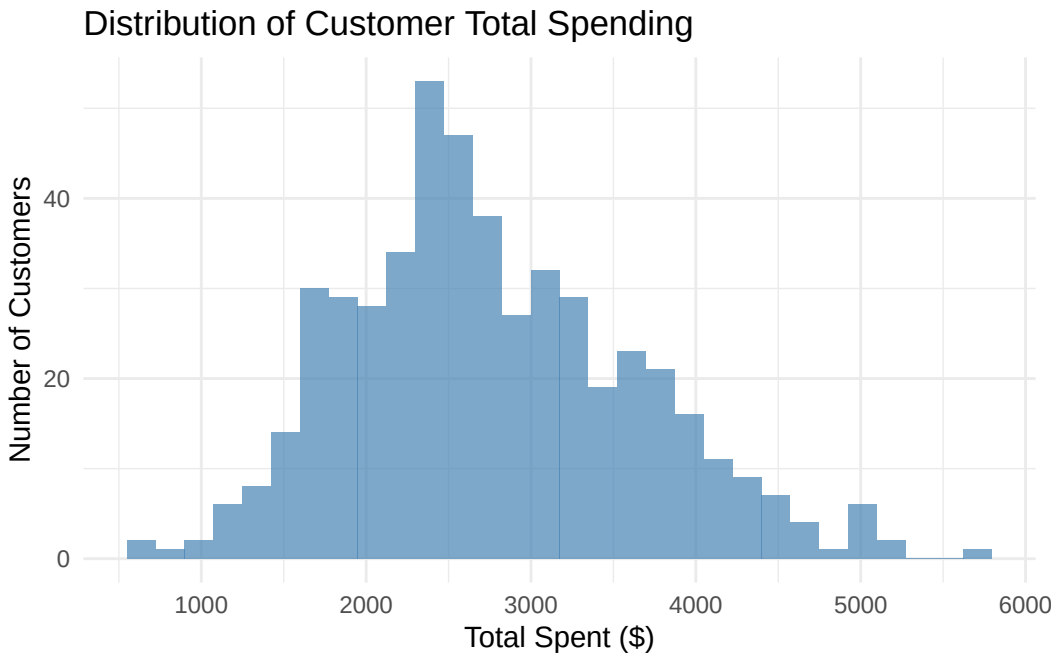
```
customer_base <- customer_transactions |>
    group_by(customer_id) |>
    summarise(
        n_transactions = sum(n_trans),
        total_spent = sum(trans_amount),
        avg_order_value = mean(avg_transaction_value),
        months_active = n_distinct(month),
        std_spending = sd(trans_amount),
        .groups = "drop"
    ) |>
    mutate(std_spending = ifelse(is.na(std_spending), 0, std_spending)) |>
    drop_na()

nrow(customer_base)
```

Number of Customers	Mean Total Spent	Median Total Spent	Mean Transactions	Mean AOV	M
500	2,764.49	2,629.08	24.98	110.24	

```
[1] 500
```

```
customer_overview <- customer_base |>
    summarise(
        n_customers = n(),
        mean_total_spent = mean(total_spent),
        median_total_spent = median(total_spent),
        mean_transactions = mean(n_transactions),
        mean_aov = mean(avg_order_value),
        mean_months = mean(months_active)
    )

customer_overview |>
    gt() |>
    cols_label(
        n_customers = "Number of Customers",
        mean_total_spent = "Mean Total Spent",
        median_total_spent = "Median Total Spent",
        mean_transactions = "Mean Transactions",
        mean_aov = "Mean AOV",
        mean_months = "Mean Months Active"
    ) |>
    fmt_number(columns = -n_customers, decimals = 2)
```

```
customer_base |>
    ggplot(aes(x = total_spent)) +
    geom_histogram(fill = "steelblue", alpha = 0.7) +
    labs(
        title = "Distribution of Customer Total Spending",
        x = "Total Spent ($)",
        y = "Number of Customers"
    ) +
    theme_minimal()
```



21.4 RFM-Style Baseline Segmentation

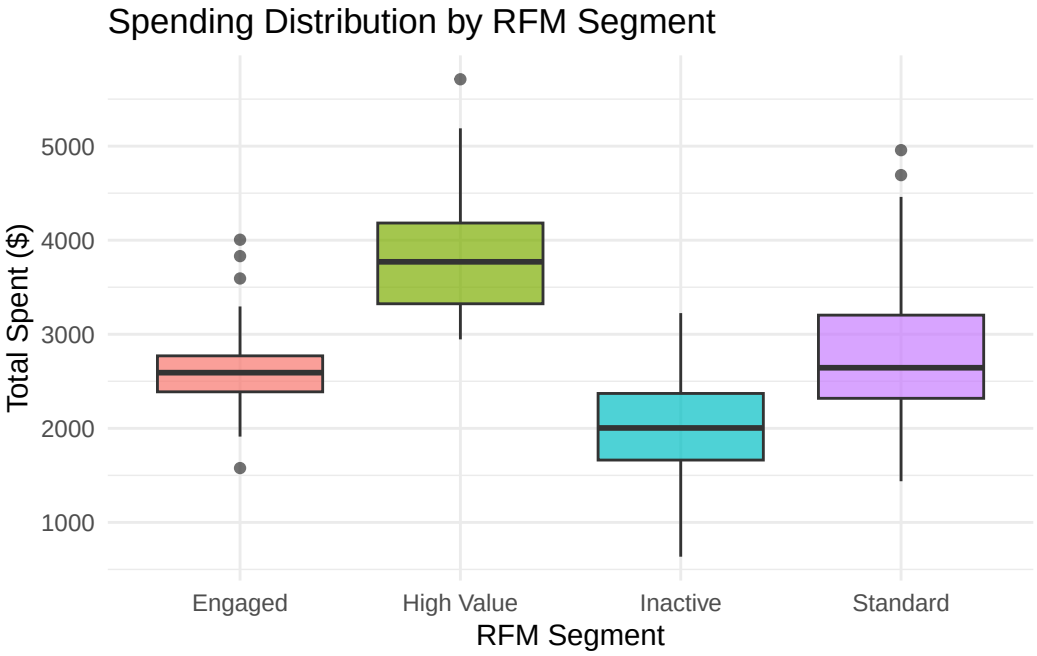
Before automated methods, we create a simple baseline segmentation based on RFM, a widely used customer analytics framework: Recency (how recently customers bought), Frequency (how often they buy), and Monetary value (how much they spend). We use `months_active` as a proxy for recency, `n_transactions` for frequency, and `total_spent` for monetary value. We create quintile-based scores for each dimension and define 4 segments based on combinations of these scores.

```
rfm_segments <- customer_base |>
    mutate(
        r_score = ntile(months_active, 5),
        f_score = ntile(n_transactions, 5),
        m_score = ntile(total_spent, 5),
        rfm_segment = case_when(
            r_score >= 4 & f_score >= 4 & m_score >= 4 ~ "High Value",
            r_score <= 2 & f_score <= 2 ~ "Inactive",
            r_score >= 4 & f_score >= 3 ~ "Engaged",
            TRUE ~ "Standard"
        )
    )
```

Segment	N	Avg. Spending	Avg. Transactions	Avg. Months Active
Engaged	67	2,602.33	26.61	9.76
High Value	109	3,823.70	32.61	10.14
Inactive	138	2,020.00	18.14	6.59
Standard	186	2,754.55	24.98	8.24

```
segment_summary <- rfm_segments |>
    group_by(rfm_segment) |>
    summarise(
        n_customers = n(),
        mean_spent = mean(total_spent),
        mean_trans = mean(n_transactions),
        mean_months = mean(months_active),
        .groups = "drop"
    )

segment_summary |>
    gt() |>
    cols_label(
        rfm_segment = "Segment",
        n_customers = "N",
        mean_spent = "Avg. Spending",
        mean_trans = "Avg. Transactions",
        mean_months = "Avg. Months Active"
    ) |>
    fmt_number(columns = -c(rfm_segment, n_customers), decimals = 2)
```

```
rfm_segments |>
    ggplot(aes(x = rfm_segment, y = total_spent, fill = rfm_segment)) +
    geom_boxplot(alpha = 0.7) +
    labs(
        title = "Spending Distribution by RFM Segment",
        x = "RFM Segment",
        y = "Total Spent ($)",
        fill = "Segment"
    ) +
    theme_minimal() +
    theme(legend.position = "none")
```



This baseline segmentation provides a business-interpretable starting point. The automated feature engineering and model search steps that follow will reveal whether richer representations of customer behavior capture meaningful additional structure.

21.5 Automated Feature Engineering with Candidate Screening

RFM is an effective summary, but it discards information about the *shape* of customer behavior. Automated feature engineering, following the workflow developed in Chapter 18, provides a principled alternative: rather than manually selecting derived features, we generate a library of candidate transformations and interaction terms, then screen them by cross-validated predictive improvement.

We reserve a held-out test set before any screening, following the same separation used in Chapter 18.

```
set.seed(999)
idx_test <- sample(seq_len(nrow(customer_base)), size = 0.2 * nrow(customer_base))

analysis_data <- customer_base[-idx_test, ]
test_data     <- customer_base[idx_test, ]

cat("Analysis set:", nrow(analysis_data), "customers\n")
```

```
Analysis set: 400 customers
```

```
cat("Test set:    ", nrow(test_data), "customers\n")
```

```
Test set:     100 customers
```

21.5.1 Constructing the Candidate Library

We define four raw features and generate two classes of candidate terms: element-wise transformations (log and square root, which are useful for right-skewed customer metrics) and pairwise interactions (which capture how feature combinations relate to spending).

```
raw_features <- c("n_transactions", "avg_order_value", "months_active", "std_spending")

# Transformation candidates
transform_candidates <- unlist(lapply(raw_features, function(v) {
    c(paste0("log1p(", v, ")"), paste0("sqrt(", v, ")"))
}))

# Pairwise interaction candidates
interaction_candidates <- combn(raw_features, 2, simplify = FALSE) |>
    lapply(function(pair) paste(pair, collapse = ":")) |>
    unlist()

all_candidates <- c(transform_candidates, interaction_candidates)
cat("Transformation candidates:", length(transform_candidates), "\n")
```

```
Transformation candidates: 8
```

```
cat("Interaction candidates:   ", length(interaction_candidates), "\n")
```

```
Interaction candidates:    6
```

```
cat("Total candidates:         ", length(all_candidates), "\n")
```

```
Total candidates:          14
```

21.5.2 Screening by Cross-Validated Improvement

We define a baseline additive linear model and evaluate each candidate term by augmenting this baseline with one additional feature at a time. The evaluation uses five-fold cross-validation on the analysis set, with holdout RMSE as the criterion.

```
rmse <- function(actual, predicted) {
    sqrt(mean((actual - predicted)^2))
}

make_folds <- function(n, k = 5, seed = 123) {
    set.seed(seed)
    sample(rep(seq_len(k), length.out = n))
}

cv_rmse_lm <- function(data, formula_obj, fold_id) {
    k <- length(unique(fold_id))
    fold_errors <- numeric(k)

    for (fold in seq_len(k)) {
        train_fold <- data[fold_id != fold, , drop = FALSE]
        valid_fold <- data[fold_id == fold, , drop = FALSE]

        fit <- lm(formula_obj, data = train_fold)
        pred <- as.numeric(predict(fit, newdata = valid_fold))
        fold_errors[fold] <- rmse(valid_fold[["total_spent"]], pred)
    }

    mean(fold_errors)
}

fold_id <- make_folds(n = nrow(analysis_data), k = 5, seed = 456)

baseline_formula <- total_spent ~ n_transactions + avg_order_value + months_active + std_spe
baseline_cv <- cv_rmse_lm(analysis_data, baseline_formula, fold_id)

cat("Baseline CV RMSE:", round(baseline_cv, 2), "\n")
```

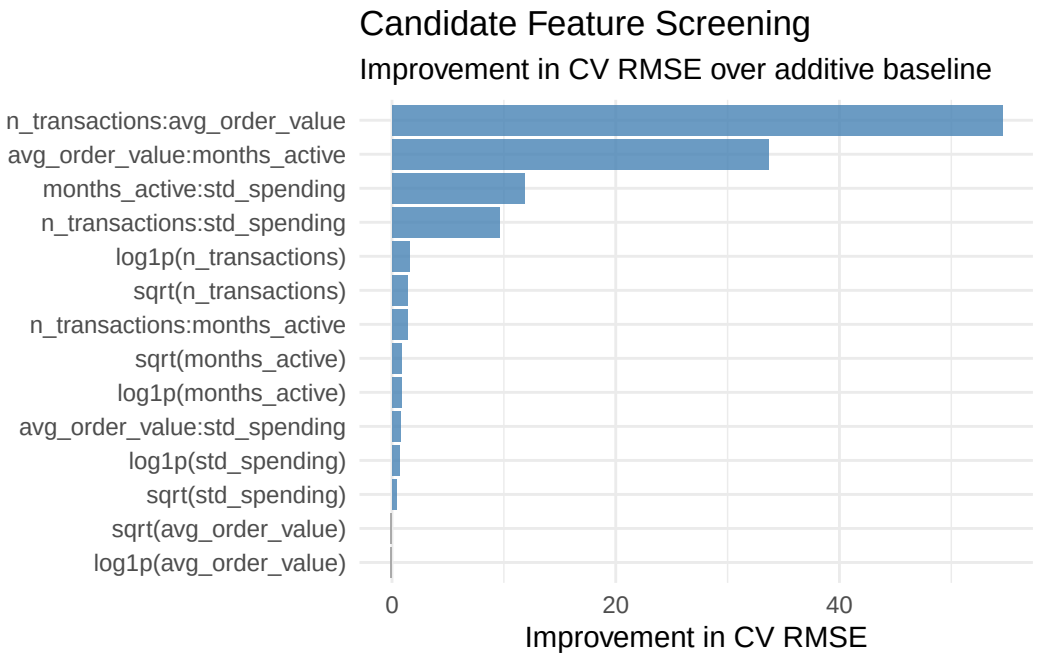
```
Baseline CV RMSE: 190.44
```

```
screening_results <- lapply(all_candidates, function(term) {
    aug_formula <- as.formula(
        paste("total_spent ~ n_transactions + avg_order_value + months_active + std_spending
    )
    tryCatch({
        cv_val <- cv_rmse_lm(analysis_data, aug_formula, fold_id)
        tibble::tibble(
            candidate = term,
            cv_rmse   = cv_val,
            improvement = baseline_cv - cv_val
        )
    }, error = function(e) {
        tibble::tibble(candidate = term, cv_rmse = NA_real_, improvement = NA_real_)
    })
})

feature_leaderboard <- bind_rows(screening_results) |>
    drop_na() |>
    arrange(desc(improvement))

feature_leaderboard |>
    gt() |>
    cols_label(
        candidate   = "Candidate feature",
        cv_rmse     = "CV RMSE",
        improvement = "Improvement vs. baseline"
    ) |>
    fmt_number(columns = c(cv_rmse, improvement), decimals = 2)
```

```
feature_leaderboard |>
    ggplot(aes(
        x    = reorder(candidate, improvement),
        y    = improvement,
        fill = improvement > 0
    )) +
    geom_col(alpha = 0.8) +
    coord_flip() +
    scale_fill_manual(values = c("TRUE" = "steelblue", "FALSE" = "gray60"),
                      guide = "none") +
    labs(
        title = "Candidate Feature Screening",
```

Candidate feature	CV RMSE	Improvement vs. baseline
n_transactions:avg_order_value	135.88	54.56
avg_order_value:months_active	156.77	33.67
months_active:std_spending	178.55	11.88
n_transactions:std_spending	180.86	9.58
log1p(n_transactions)	188.86	1.57
sqrt(n_transactions)	189.02	1.42
n_transactions:months_active	189.07	1.37
sqrt(months_active)	189.57	0.86
log1p(months_active)	189.58	0.86
avg_order_value:std_spending	189.67	0.77
log1p(std_spending)	189.74	0.70
sqrt(std_spending)	190.03	0.40
sqrt(avg_order_value)	190.58	-0.14
log1p(avg_order_value)	190.61	-0.17

```
        subtitle = "Improvement in CV RMSE over additive baseline",
        x     = NULL,
        y     = "Improvement in CV RMSE"
    ) +
    theme_minimal()
```



The screening leaderboard shows which feature representations add descriptive value beyond the additive baseline. Interaction terms that appear at the top reflect genuine behavioral coupling between customer metrics: for example, a high-frequency customer with a high average order value represents a qualitatively different profile from one where only one dimension is elevated. This kind of multiplicative structure is not captured by the additive baseline and is exactly the type of pattern that automated feature engineering helps surface.

It is worth noting a methodological point: some interaction candidates may show strong improvement partly because of the mathematical construction of the data. In practice, a business analyst should inspect top-ranked features carefully to distinguish between genuine behavioral insights and artefacts of variable construction.

21.6 AutoML-Style Model Search and Leaderboard

Automated feature engineering answers the question of *what* to include as inputs. An AutoML model search, following Chapter 17, answers the complementary question: *which model family* best captures the structure of customer value heterogeneity across those inputs?

We compare three model families under a common five-fold cross-validation protocol: a linear regression baseline, regression trees with several hyperparameter configurations, and random forests with several `mtry` values. This mirrors the search structure developed in Chapter 17, applied to the customer data.

21.6.1 Evaluation Engine

```
evaluate_candidate <- function(data,
                               outcome,
                               fold_id,
                               fit_fun,
                               pred_fun,
                               params = list()) {
    k <- length(unique(fold_id))
    fold_rmse <- numeric(k)

    for (fold in seq_len(k)) {
        train_fold <- data[fold_id != fold, , drop = FALSE]
        valid_fold <- data[fold_id == fold, , drop = FALSE]

        fit_obj <- do.call(fit_fun, c(list(train_data = train_fold), params))
        pred    <- pred_fun(fit_obj, valid_fold)
```

```
        fold_rmse[fold] <- rmse(valid_fold[[outcome]], pred)
    }

    tibble::tibble(mean_rmse = mean(fold_rmse), sd_rmse = sd(fold_rmse))
}
```

```
automl_formula <- total_spent ~ n_transactions + avg_order_value + months_active + std_spend

fit_lm_model <- function(train_data, formula_obj) {
    lm(formula_obj, data = train_data)
}

pred_lm_model <- function(model_obj, new_data) {
    as.numeric(predict(model_obj, newdata = new_data))
}

fit_tree_model <- function(train_data, formula_obj, cp, maxdepth, minsplit) {
    rpart(
        formula_obj,
        data    = train_data,
        method  = "anova",
        control = rpart.control(cp = cp, maxdepth = maxdepth, minsplit = minsplit)
    )
}

pred_tree_model <- function(model_obj, new_data) {
    as.numeric(predict(model_obj, newdata = new_data))
}

fit_rf_model <- function(train_data, formula_obj, mtry, ntree, seed = 123) {
    set.seed(seed)
    randomForest(formula_obj, data = train_data, mtry = mtry, ntree = ntree)
}

pred_rf_model <- function(model_obj, new_data) {
    as.numeric(predict(model_obj, newdata = new_data))
}
```

21.6.2 Candidate Space and Leaderboard

```
results_list    <- list()
row_counter     <- 1
formula_registry <- list()   # stores formulas keyed by "family||config"

# 1a. Linear regression baseline
lm_eval <- evaluate_candidate(
    data     = analysis_data,
    outcome  = "total_spent",
    fold_id  = fold_id,
    fit_fun  = fit_lm_model,
    pred_fun = pred_lm_model,
    params   = list(formula_obj = automl_formula)
)

results_list[[row_counter]] <- tibble::tibble(
    family    = "Linear regression",
    config    = "standard OLS",
    mean_rmse = lm_eval$mean_rmse,
    sd_rmse   = lm_eval$sd_rmse
)
formula_registry[["Linear regression||standard OLS"]] <- automl_formula
row_counter <- row_counter + 1

# 1b. Linear regression augmented with top screened features
top_engineered <- feature_leaderboard |>
    filter(improvement > 0) |>
    slice_head(n = 3) |>
    pull(candidate)

for (k in seq_along(top_engineered)) {
    aug_terms    <- paste(top_engineered[1:k], collapse = " + ")
    aug_formula  <- as.formula(
        paste("total_spent ~ n_transactions + avg_order_value + months_active + std_spending
              aug_terms)
    )
    config_label <- paste0("+ top ", k, ifelse(k == 1, " feature", " features"))
    formula_registry[[paste0("Linear regression (engineered)||", config_label)]] <- aug_formu

    aug_eval <- evaluate_candidate(
        data     = analysis_data,
```

```
        outcome  = "total_spent",
        fold_id  = fold_id,
        fit_fun  = fit_lm_model,
        pred_fun = pred_lm_model,
        params   = list(formula_obj = aug_formula)
    )
    results_list[[row_counter]] <- tibble::tibble(
        family    = "Linear regression (engineered)",
        config    = config_label,
        mean_rmse = aug_eval$mean_rmse,
        sd_rmse   = aug_eval$sd_rmse
    )
    row_counter <- row_counter + 1
}

# 2. Regression tree grid
tree_grid <- expand.grid(
    cp       = c(0.001, 0.01),
    maxdepth = c(3, 6),
    minsplit = c(10, 20)
)

for (i in seq_len(nrow(tree_grid))) {
    this_eval <- evaluate_candidate(
        data     = analysis_data,
        outcome  = "total_spent",
        fold_id  = fold_id,
        fit_fun  = fit_tree_model,
        pred_fun = pred_tree_model,
        params   = list(
            formula_obj = automl_formula,
            cp          = tree_grid$cp[i],
            maxdepth    = tree_grid$maxdepth[i],
            minsplit    = tree_grid$minsplit[i]
        )
    )

    results_list[[row_counter]] <- tibble::tibble(
        family   = "Regression tree",
        config   = paste0("cp=", tree_grid$cp[i], ", depth=", tree_grid$maxdepth[i],
                          ", minsplit=", tree_grid$minsplit[i]),
        mean_rmse = this_eval$mean_rmse,
```

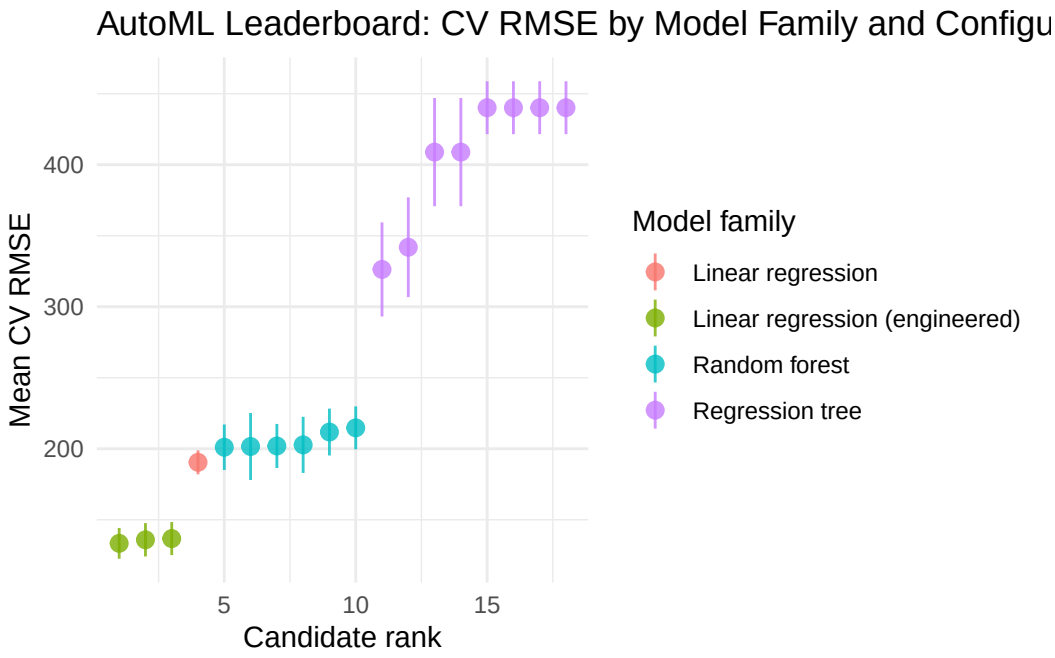
```
        sd_rmse   = this_eval$sd_rmse
    )
    row_counter <- row_counter + 1
}

# 3. Random forest grid
rf_grid <- expand.grid(
    mtry  = c(2, 3, 4),
    ntree = c(200, 400)
)

for (i in seq_len(nrow(rf_grid))) {
    this_eval <- evaluate_candidate(
        data     = analysis_data,
        outcome  = "total_spent",
        fold_id  = fold_id,
        fit_fun  = fit_rf_model,
        pred_fun = pred_rf_model,
        params   = list(
            formula_obj = automl_formula,
            mtry        = rf_grid$mtry[i],
            ntree       = rf_grid$ntree[i],
            seed        = 100 + i
        )
    )

    results_list[[row_counter]] <- tibble::tibble(
        family   = "Random forest",
        config   = paste0("mtry=", rf_grid$mtry[i], ", ntree=", rf_grid$ntree[i]),
        mean_rmse = this_eval$mean_rmse,
        sd_rmse   = this_eval$sd_rmse
    )
    row_counter <- row_counter + 1
}

automl_leaderboard <- bind_rows(results_list) |>
    arrange(mean_rmse)
```

```
automl_leaderboard |>
    gt() |>
    cols_label(
        family    = "Model family",
```

Model family	Configuration	Mean CV RMSE	SD CV RMSE
Linear regression (engineered)	+ top 3 features	133.39	10.81
Linear regression (engineered)	+ top 1 feature	135.88	11.73
Linear regression (engineered)	+ top 2 features	136.73	11.65
Linear regression	standard OLS	190.44	8.45
Random forest	mtry=3, ntree=200	201.05	16.03
Random forest	mtry=4, ntree=200	201.58	23.62
Random forest	mtry=3, ntree=400	201.94	15.53
Random forest	mtry=4, ntree=400	202.72	19.78
Random forest	mtry=2, ntree=400	211.76	16.51
Random forest	mtry=2, ntree=200	214.71	15.13
Regression tree	cp=0.001, depth=6, minsplit=10	326.26	33.14
Regression tree	cp=0.001, depth=6, minsplit=20	341.90	35.14
Regression tree	cp=0.01, depth=6, minsplit=10	408.88	38.09
Regression tree	cp=0.01, depth=6, minsplit=20	408.88	38.09
Regression tree	cp=0.001, depth=3, minsplit=10	440.14	18.63
Regression tree	cp=0.01, depth=3, minsplit=10	440.14	18.63
Regression tree	cp=0.001, depth=3, minsplit=20	440.14	18.63
Regression tree	cp=0.01, depth=3, minsplit=20	440.14	18.63

```
        config    = "Configuration",
        mean_rmse = "Mean CV RMSE",
        sd_rmse   = "SD CV RMSE"
    ) |>
    fmt_number(columns = c(mean_rmse, sd_rmse), decimals = 2)
```

```
automl_leaderboard |>
    mutate(rank = seq_len(n())) |>
    ggplot(aes(x = rank, y = mean_rmse, color = family,
               ymin = mean_rmse - sd_rmse, ymax = mean_rmse + sd_rmse)) +
    geom_pointrange(alpha = 0.8) +
    labs(
        title  = "AutoML Leaderboard: CV RMSE by Model Family and Configuration",
        x      = "Candidate rank",
        y      = "Mean CV RMSE",
        color  = "Model family"
    ) +
    theme_minimal()
```



The leaderboard now spans four model families: the plain linear baseline, linear regression augmented with top screened features from the previous section, regression trees, and random forests. This is the natural combination of the feature engineering workflow (Chapter 18) and the model search workflow (Chapter 17): screening candidates first and then searching over model families that include those candidates.

The leaderboard identifies the winning configuration, both in terms of model family and feature representation, under a common evaluation protocol. If a plain linear model with raw features wins, the relationship is additive and no engineered features are needed. If an augmented linear model wins, the interaction or transformation terms improve things but complex nonlinear models do not. If ensemble models win, there is genuine nonlinear or interaction structure that linear representations cannot capture regardless of feature engineering.

Comparing the winning CV RMSE back to the baseline (plain linear regression, evaluated in the same folds) provides a calibrated sense of how much the combined feature engineering and model search workflow gained over the starting point.

The winning model, linear model augmented with the top 3 features, is used for all downstream analysis.

21.7 Feature Importance and Variable Selection

We fit the winning model on the full analysis set and rank the original raw predictors by model-agnostic permutation importance, following Chapter 14. Permutation importance is

computed by measuring how much holdout RMSE increases when each feature is randomly shuffled, breaking its relationship with the outcome while leaving the model unchanged. A large increase signals a feature the model relies on heavily; a near-zero increase signals that the model can predict almost as well without it.

Permuting a raw feature (for example `n_transactions`) affects all formula terms that depend on it, including any engineered terms such as `log1p(n_transactions)` or interaction terms. This means the importance scores correctly attribute signal to the original customer behavior variables rather than to their derived representations, regardless of which formula won the leaderboard.

```
# Recover the formula of the best linear model from the registry
winner_key     <- paste(automl_leaderboard$family[1],
                        automl_leaderboard$config[1], sep = "||")
winning_formula <- if (!is.null(formula_registry[[winner_key]])) {
    formula_registry[[winner_key]]
} else {
    automl_formula   # fallback: plain baseline
}

lm_final <- lm(winning_formula, data = analysis_data)
cat("Winning formula:\n")
```

```
Winning formula:
```

```
print(winning_formula)
```

```
total_spent ~ n_transactions + avg_order_value + months_active +
    std_spending + n_transactions:avg_order_value + avg_order_value:months_active +
    months_active:std_spending
```

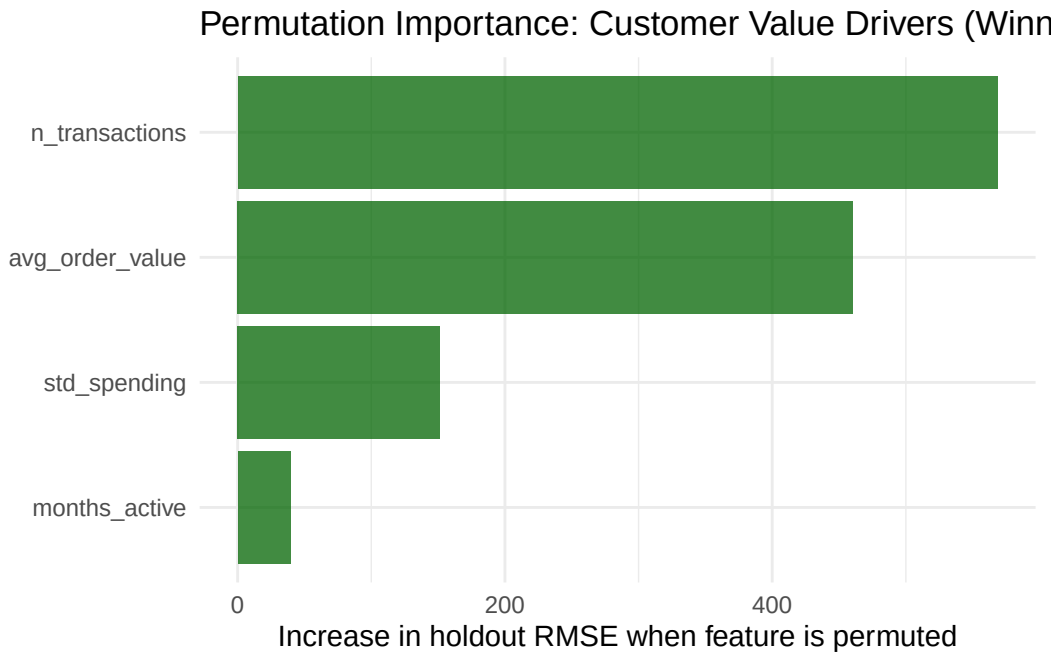
```
permute_importance <- function(model, data, outcome, features, n_perm = 100, seed = 456) {
    set.seed(seed)
    baseline_pred <- as.numeric(predict(model, newdata = data))
    baseline_rmse <- rmse(data[[outcome]], baseline_pred)

    imp <- sapply(features, function(v) {
        perm_rmse <- replicate(n_perm, {
            data_perm       <- data
            data_perm[[v]]  <- sample(data_perm[[v]])
            rmse(data[[outcome]], as.numeric(predict(model, newdata = data_perm)))
```

Feature	Permutation Importance (increase in holdout RMSE)
n_transactions	568.64
avg_order_value	460.57
std_spending	151.53
months_active	39.60

```
        })
        mean(perm_rmse) - baseline_rmse
    })

    tibble::tibble(
        feature    = names(imp),
        importance = as.numeric(imp)
    ) |>
        arrange(desc(importance))
}

features_to_rank <- c("n_transactions", "avg_order_value", "months_active", "std_spending")

importance_tbl <- permute_importance(
    model    = lm_final,
    data     = test_data,
    outcome  = "total_spent",
    features = features_to_rank
)

importance_tbl |>
    gt() |>
    cols_label(
        feature    = "Feature",
        importance = "Permutation Importance (increase in holdout RMSE)"
    ) |>
    fmt_number(columns = importance, decimals = 2)
```

```
importance_tbl |>
    ggplot(aes(x = reorder(feature, importance), y = importance)) +
    geom_col(fill = "darkgreen", alpha = 0.75) +
    coord_flip() +
    labs(
        title = "Permutation Importance: Customer Value Drivers (Winning Linear Model)",
```

```
        x     = NULL,
        y     = "Increase in holdout RMSE when feature is permuted"
    ) +
    theme_minimal()
```



Cross-referencing importance rankings with the feature screening leaderboard provides a consistency check. When the same predictors appear both as top candidates in the screening step and as high-importance variables in the winning model, there is convergent evidence that these features carry genuine descriptive signal. When rankings differ, the discrepancy may reflect collinearity between predictors or sensitivity to how the candidate term interacts with the baseline model form.

21.8 Customer Targeting and Segmentation

Using the trained model, we score all customers and partition them by predicted value into four business-relevant tiers.

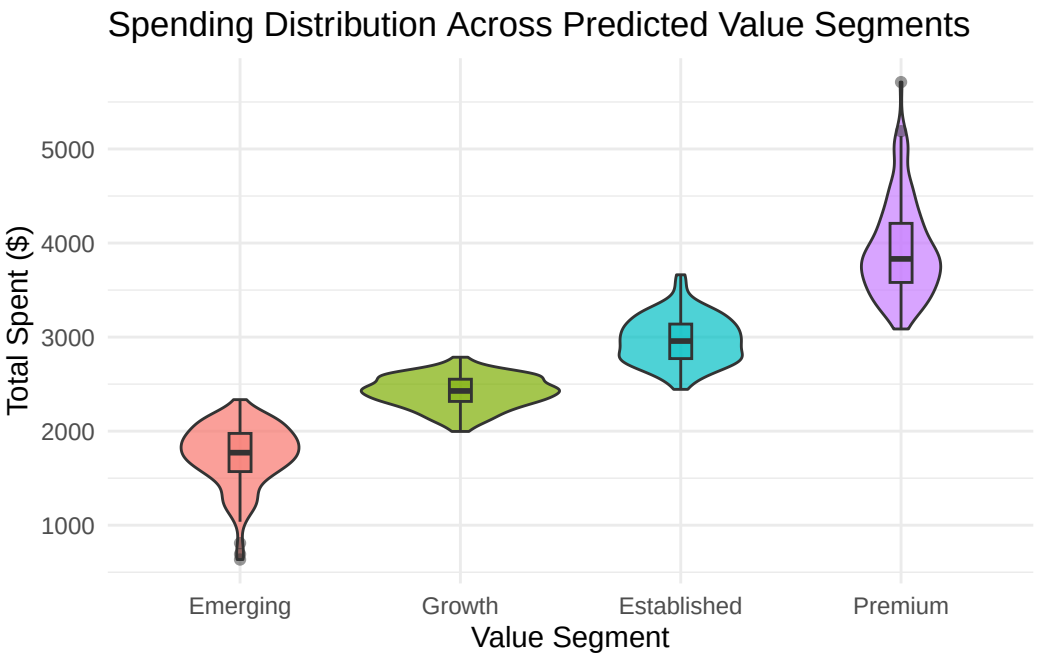
```
customer_scored <- customer_base |>
    mutate(
        predicted_value = as.numeric(predict(lm_final, newdata = customer_base)),
        value_segment   = cut(
            predicted_value,
```

Segment	N	Actual Avg. Spending	Predicted Avg. Spending	Mean Transactions	Mean Mon
Emerging	125	1,741.30	1,745.07	18.65	
Growth	125	2,414.82	2,409.82	23.10	
Established	125	2,969.78	2,974.38	26.18	
Premium	125	3,932.06	3,932.73	31.97	

```
            breaks        = quantile(predicted_value, probs = c(0, 0.25, 0.50, 0.75, 1.0)),
            labels        = c("Emerging", "Growth", "Established", "Premium"),
            include.lowest = TRUE
        )
    )

segment_profile <- customer_scored |>
    group_by(value_segment) |>
    summarise(
        n_customers          = n(),
        actual_avg_spend     = mean(total_spent),
        predicted_avg_spend  = mean(predicted_value),
        mean_transactions    = mean(n_transactions),
        mean_months_active   = mean(months_active),
        .groups = "drop"
    )

segment_profile |>
    gt() |>
    cols_label(
        value_segment        = "Segment",
        n_customers          = "N",
        actual_avg_spend     = "Actual Avg. Spending",
        predicted_avg_spend  = "Predicted Avg. Spending",
        mean_transactions    = "Mean Transactions",
        mean_months_active   = "Mean Months Active"
    ) |>
    fmt_number(columns = -c(value_segment, n_customers), decimals = 2)
```

```
customer_scored |>
    ggplot(aes(x = value_segment, y = total_spent, fill = value_segment)) +
    geom_violin(alpha = 0.7) +
    geom_boxplot(width = 0.1, alpha = 0.5) +
    labs(
```

```
        title = "Spending Distribution Across Predicted Value Segments",
        x     = "Value Segment",
        y     = "Total Spent ($)",
        fill  = "Segment"
    ) +
    theme_minimal() +
    theme(legend.position = "none")
```



These data-driven segments can inform differentiated marketing, pricing, or retention strategies. The model-based partitioning complements the rule-based RFM segmentation from earlier in the chapter: the two approaches can be compared directly to examine whether the automated method adds resolution beyond the manual baseline.

21.9 Communicating Customer Insights to Business Stakeholders

Data-driven customer segmentation is most valuable when translated into actionable business narratives. The following insights bridge the feature screening and AutoML results with business language.

Insight 1: Transaction frequency and average order value are the primary value drivers. Permutation importance indicates that `n_transactions` and `avg_order_value` are the two strongest contributors to predicted customer spending. The feature screening step confirms that their interaction term provides additional explanatory power beyond either

predictor in isolation. This suggests that growth actions should target basket expansion (higher order value) and repeat purchase behavior (higher transaction volume) simultaneously, rather than treating them as independent levers.

Insight 2: Customer value is well described by an additive linear model, complexity is not required. The AutoML leaderboard shows that linear regression outperforms all tree-based and ensemble alternatives. This is a substantive finding: it means that the drivers of total spending combine approximately additively, and a linear combination of transaction frequency, order value, tenure, and spending variability captures the main patterns. For business planning, this simplicity is an asset: additive contributions are directly interpretable as marginal value drivers without requiring complex model explanations.

Insight 3: A four-tier segmentation is operationally useful. The model separates customers into four value tiers with rising average spend from one tier to the next. Each tier can receive a different plan, for instance: low-cost activation for Emerging, frequency-building offers for Growth, loyalty reinforcement for Established, and high-touch retention for Premium. The feature importance rankings also give each tier a behavioral profile that supports message targeting within that tier.

These insights translate feature rankings and model structure into business language that can guide strategy without requiring stakeholders to engage with the underlying algorithmic details.

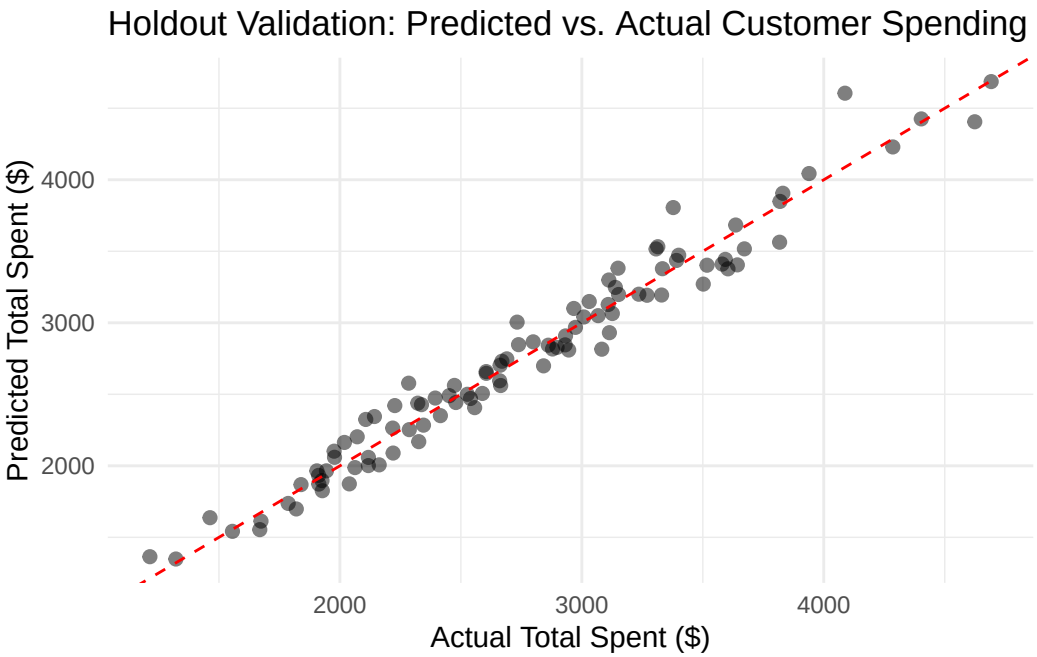
21.10 Holdout Validation and Model Performance

To assess whether the segmentation generalizes beyond the analysis sample, we evaluate model performance on the held-out test set reserved at the start of the analysis.

```
test_pred <- as.numeric(predict(lm_final, newdata = test_data))

holdout_rmse <- rmse(test_data$total_spent, test_pred)
holdout_mae  <- mean(abs(test_data$total_spent - test_pred))
r_squared    <- cor(test_data$total_spent, test_pred)^2

tibble::tibble(
    metric = c("Root Mean Squared Error", "Mean Absolute Error", "R-squared"),
    value  = c(holdout_rmse, holdout_mae, r_squared)
) |>
    gt() |>
    cols_label(metric = "Metric", value = "Value") |>
    fmt_number(columns = value, decimals = 2)
```

Metric	Value
Root Mean Squared Error	140.05
Mean Absolute Error	109.51
R-squared	0.96

```
tibble::tibble(
    actual    = test_data$total_spent,
    predicted = test_pred
) |>
    ggplot(aes(x = actual, y = predicted)) +
    geom_point(alpha = 0.5, size = 2) +
    geom_abline(intercept = 0, slope = 1, color = "red", linetype = "dashed") +
    labs(
        title = "Holdout Validation: Predicted vs. Actual Customer Spending",
        x     = "Actual Total Spent ($)",
        y     = "Predicted Total Spent ($)"
    ) +
    theme_minimal()
```



Holdout performance confirms that the model's segmentation and feature ranking reflect stable patterns in customer behavior. Comparing CV RMSE from the leaderboard to the holdout RMSE provides a consistency check: a large discrepancy would suggest over-optimistic

leaderboard estimates, possibly due to overfitting during the feature screening or model search phase.

21.11 Limitations and Reproducibility Notes

Several caveats warrant attention.

1. This case study uses simulated data. Real customer data often contains temporal autocorrelation, seasonality, sparse records, and missing values that require additional preprocessing. The feature-outcome relationships illustrated here are deliberately simple to make the workflow visible.
2. The strong contribution of the interaction between `n_transactions` and `avg_order_value` in the feature screening partly reflects the mathematical construction of `total_spent` in this dataset. In applied settings, careful examination of whether top-ranked features encode genuine behavioral insight or are artefacts of variable construction is essential.
3. The AutoML search is intentionally lightweight. Production systems would explore larger candidate spaces, use repeated cross-validation, and apply model-specific regularization. The principles remain the same, but the engineering budget is different.
4. Feature importance can be sensitive to feature correlation and scale. When predictors are correlated, importance scores should be interpreted as group-level signals rather than individual contributions.

All random components use set seeds for reproducibility. In applied settings, stability checks across alternative model specifications and multiple random seeds are recommended before using results to inform strategic decisions.

21.12 Summary and Key Takeaways

- Automated feature engineering systematizes the search for informative derived predictors by generating a candidate library and screening it by cross-validated improvement, following the workflow from Chapter 18.
- AutoML-style model comparison surveys multiple model families under a common evaluation protocol, producing a leaderboard that describes how much structure different algorithms can extract from the data, following Chapter 17.
- The combination of feature screening and model leaderboard provides two complementary views of the same question: what drives customer value, and how complex is that structure?
- Model-agnostic permutation importance applied to the winning model translates AutoML and feature engineering results into a ranked list of customer behavior drivers, regardless of which model family prevailed.

- Data-driven customer segmentation supports differentiated targeting and resource allocation.
- The emphasis throughout is descriptive: AutoML and automated feature engineering are instruments for learning about data, not for deploying prediction systems.

21.13 Looking Ahead

This case study focused on automated feature engineering and AutoML-style model comparison as descriptive instruments for customer insight. The next chapter consolidates the core lessons from all three case studies (policy, health, and business), highlights the main practical principles of descriptive tabular analysis, and closes with recommendations for applying this workflow responsibly in real-world projects.

22 Conclusion

This book has explored a particular way of doing data analysis, one that takes description seriously as an intellectual and practical task in its own right. We began with foundational work on preparation and structure, including semantic variable selection from metadata, moved through association and dependence, expanded into interactive visual analytics, and then engaged with tree-based, ensemble, and interpretable machine learning tools. We then brought these ideas into applied settings in public policy, public health, and business analytics, ending with a case study focused on customer insights.

Across this progression, the aim has not been to position advanced methods as a replacement for careful descriptive reasoning, but to show how they can support it. In many real settings, the central analytical challenge is not only to estimate or predict, but to understand, compare, communicate, and deliberate. Descriptive analysis, especially when carried out with methodological transparency and visual clarity, helps make those tasks possible.

22.1 Description as a Serious Analytical Goal

One recurring theme throughout the book is that description is not a preliminary step to "real" modeling. It is itself a mode of inquiry, with its own standards of rigor. To describe well, we need to move beyond surface summaries and ask structured questions about variation, association, heterogeneity, and context. We need tools that can reveal patterns without obscuring uncertainty, and complexity without abandoning interpretability.

This orientation has shaped the methodological choices we have discussed. The chapters on association measures and correlation extensions emphasized that relationships in data are often richer than linear summaries suggest. The chapters on interactive visual analytics and the AssociationExplorer highlighted how exploratory work can be both flexible and disciplined when interfaces make assumptions visible and comparisons tractable. The chapters on tree-based and ensemble approaches showed that methods often associated with prediction can also be used to characterize structure in data, provided we remain explicit about purpose and limits.

In this sense, advanced descriptive analysis is not defined by algorithmic sophistication alone. It is defined by the alignment between method and question. When the question is descriptive, our criteria shift. We care about whether a method helps us see meaningful structure, whether it supports transparent interpretation, and whether its outputs can be communicated responsibly to audiences with different forms of expertise.

22.2 Interpretability, Transparency, and Communication

A second unifying theme has been interpretability, not as a single technical property, but as a relational one. An analysis is interpretable to someone, in a context, for a purpose. The chapters on feature importance, partial dependence, and Shapley-style decompositions illustrated how interpretability tools can make model behavior more legible. At the same time, they also showed that interpretability is never automatic. Different tools answer different questions, and each comes with assumptions that shape what can be seen.

For this reason, methodological transparency matters as much as methodological choice. Transparency includes documenting preprocessing decisions, clarifying what a metric does and does not capture, and distinguishing descriptive signal from causal claims. It also includes making visual and narrative choices that reduce ambiguity rather than amplifying it.

Communication is therefore not an "after" stage of analysis. It is part of analysis itself. A descriptive result that cannot be meaningfully explained to collaborators, decision makers, or domain stakeholders is limited in practice, even if it is technically sophisticated. Conversely, a well-communicated result can support better collective reasoning, including disagreement, revision, and follow-up inquiry. The case studies in this book have tried to model this orientation by treating context and audience as analytical constraints rather than external concerns.

22.3 Advanced Methods in Service of Description

The chapters on automated workflows and AutoML framed another important point. Automation can help us explore model spaces, benchmark alternative representations, and accelerate iterative work. Yet automation does not remove the need for judgment. In descriptive settings, where the goal is understanding rather than predictive dominance, a "best" model is rarely best in an absolute sense. It is better to ask whether an approach improves our reading of data while preserving interpretive coherence.

This distinction has practical consequences. It encourages us to compare methods not only by aggregate performance metrics, but also by stability, intelligibility, and communicability. It encourages us to inspect local and subgroup behavior, rather than relying only on global summaries. It reminds us that methodological pluralism is often productive, especially when different techniques converge on compatible descriptive insights, or when they surface tensions that require deeper domain interpretation.

Seen this way, advanced descriptive analysis is less about selecting a final winner among tools, and more about building a defensible evidentiary narrative from multiple complementary views of the same data.

22.4 Lessons from Applied Contexts

The three applied case studies provided a final opportunity to test these ideas against real constraints. Public policy analysis highlighted the importance of fairness, institutional context, and interpretability for public accountability. Public health analysis emphasized heterogeneity, measurement quality, and the need for communication that is both precise and cautious. Business analytics, including the customer insights case study in the previous chapter, showed how descriptive modeling can inform segmentation, prioritization, and strategic interpretation without reducing analysis to short-horizon prediction.

These contexts differ in objectives, governance, and risk, but they share a common challenge: turning complex data into structured, communicable understanding. In each setting, descriptive analysis supports decisions indirectly, by improving the quality of the conversation that precedes action. This contribution can appear modest compared with claims of optimization or full automation, but in practice it is often where robust decisions begin.

22.5 Limits and Open Challenges

The approaches in this book have clear limitations, and those limitations should remain visible. First, descriptive findings are sensitive to data quality and representativeness. No amount of methodological sophistication can fully compensate for systematic gaps, biased measurement, or unstable data-generating processes.

Second, interpretability tools can be overread. Feature importance and effect summaries can create a sense of explanatory closure that the data do not warrant. Without careful framing, there is a risk of presenting model artifacts as substantive mechanisms.

Third, interactive and automated workflows introduce governance questions of their own. Which analytical choices are made explicit to users, and which remain hidden? Who is included in the interpretive loop, and who is excluded? How are uncertainty and ambiguity represented when results are communicated to non-technical audiences?

Finally, descriptive analysis remains vulnerable to context loss. Patterns that are statistically stable can still be substantively misleading if domain histories, institutional processes, or practical constraints are ignored. This is not a technical failure alone, it is a reminder that data analysis is an interdisciplinary practice.

22.6 Looking Ahead

Several methodological directions appear especially promising for the future of advanced descriptive analysis. One is the continued integration of interactive visualization with interpretable modeling, so that analysts can move more fluidly between global structure and local detail.

Another is the development of richer uncertainty communication for descriptive outputs, particularly in settings where stakeholders need to compare competing narratives rather than consume a single headline result.

A related direction concerns robustness. As datasets become larger and more heterogeneous, we need descriptive workflows that make sensitivity analyses more routine, not exceptional. This includes greater attention to stability across preprocessing choices, subgroup definitions, and model classes.

There is also room for deeper collaboration between methodological and domain communities. Many open questions are not purely statistical or computational. They concern what counts as a meaningful pattern, what level of abstraction supports action, and how to communicate limitations without disabling use. Progress on these questions depends on shared standards across disciplines, not only on new algorithms.

More broadly, the future of this field may depend on whether we can sustain an ethos of analytical humility alongside technical ambition. Advanced tools are valuable, but they are most valuable when they help us ask better questions, articulate uncertainty more clearly, and communicate findings in ways that support collective learning.

22.7 Final Reflections

If there is a single thread connecting the chapters of this book, it is that descriptive analysis can be both technically advanced and intellectually careful. We do not have to choose between complexity and clarity, or between computational power and interpretive responsibility. The challenge is to combine them in ways that remain transparent, communicable, and context-aware.

This conclusion is not an endpoint so much as an invitation to continue that work. The methods discussed here are tools for inquiry, not final answers. Their value depends on how we use them, how openly we report their limits, and how thoughtfully we connect quantitative patterns to substantive understanding. In that ongoing process, description remains central, not as a preliminary step before explanation or prediction, but as a durable practice of making data interpretable for real people and real decisions.

References

Anderson, T. W. 1985. "An Introduction to Multivariate Statistical Analysis, 2nd Edition." *Biometrics* 41 (3): 815. https://doi.org/10.2307/2531310.

Automated Machine Learning: Methods, Systems, Challenges. 2019. *The Springer Series on Challenges in Machine Learning*. Springer International Publishing. https://doi.org/10.1007/978-3-030-05318-5.

Baba, Kunihiro, Ritei Shibata, and Masaaki Sibuya. 2004. "PARTIAL CORRELATION AND CONDITIONAL CORRELATION AS MEASURES OF CONDITIONAL INDEPENDENCE." *Australian & New Zealand Journal of Statistics* 46 (4): 657–64. https://doi.org/10.1111/j.1467-842x.2004.00360.x.

Breiman, Leo. 1996. "Bagging Predictors." *Machine Learning* 24 (2): 123–40. https://doi.org/10.1007/bf00058655.

———. 2001. "Random Forests." *Machine Learning* 45 (1): 5–32. https://doi.org/10.1023/a:1010933404324.

Breiman, Leo, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. 2017. *Classification and Regression Trees*. Routledge. https://doi.org/10.1201/9781315139470.

BRIER, GLENN W. 1950. "VERIFICATION OF FORECASTS EXPRESSED IN TERMS OF PROBABILITY." *Monthly Weather Review* 78 (1): 1–3. https://doi.org/10.1175/1520-0493(1950)078%3C0001:vofeit%3E2.0.co;2.

Cleveland, William S. 1979. "Robust Locally Weighted Regression and Smoothing Scatterplots." *Journal of the American Statistical Association* 74 (368): 829–36. https://doi.org/10.1080/01621459.1979.10481038.

Cohen, Jacob. 2013. *Statistical Power Analysis for the Behavioral Sciences*. Routledge. https://doi.org/10.4324/9780203771587.

Cohen, Patricia, Patricia Cohen, Stephen G. West, and Leona S. Aiken. 2014. *Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences*. Psychology Press. https://doi.org/10.4324/9781410606266.

Cover, Thomas M., and Joy A. Thomas. 2001. *Elements of Information Theory*. Wiley. https://doi.org/10.1002/0471200611.

David, F. N., and H. Cramer. 1947. "Mathematical Methods of Statistics." *Biometrika* 34 (3/4): 374. https://doi.org/10.2307/2332454.

David, F. N., and John W. Tukey. 1977. "Exploratory Data Analysis." *Biometrics* 33 (4): 768. https://doi.org/10.2307/2529486.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. In *Proceedings of the 2019 Conference of the North*, 4171–86. Association for Computational Linguistics. https://doi.org/10.18653/v1/n19-1423.

European Social Survey European Research Infrastructure (ESS ERIC). 2024. "European Social Survey (ESS), Round 11 - 2023." Sikt – Norwegian Agency for Shared Services in Education; Research. https://doi.org/10.21338/ESS11-2023.

———. 2025. "ESS11 - Integrated File, Edition 3.0." Sikt – Norwegian Agency for Shared Services in Education; Research. https://doi.org/10.21338/ESS11E03_0.

Feurer, Matthias, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. 2019. "Auto-Sklearn: Efficient and Robust Automated Machine Learning." In *Automated Machine Learning*, 113–34. Springer International Publishing. https://doi.org/10.1007/978-3-030-05318-5_6.

Fisher, Aaron, Cynthia Rudin, and Francesca Dominici. 2019. "All Models Are Wrong, but Many Are Useful: Learning a Variable's Importance by Studying an Entire Class of Prediction Models Simultaneously." *Journal of Machine Learning Research* 20 (177): 1–81. https://jmlr.org/papers/v20/18-760.html.

Fisher, Sir Ronald A. 1990. "Statistical Methods for Research Workers." *Statistical Methods, Experimental Design, and Scientific Inference*. Oxford University PressOxford. https://doi.org/10.1093/oso/9780198522294.002.0003.

Friedman, Jerome H. 2001. "Greedy Function Approximation: A Gradient Boosting Machine." *The Annals of Statistics* 29 (5). https://doi.org/10.1214/aos/1013203451.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. 2007. "Sparse Inverse Covariance Estimation with the Graphical Lasso." *Biostatistics* 9 (3): 432–41. https://doi.org/10.1093/biostatistics/kxm045.

Goldstein, Alex, Adam Kapelner, Justin Bleich, and Emil Pitkin. 2015. "Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation." *Journal of Computational and Graphical Statistics* 24 (1): 44–65. https://doi.org/10.1080/10618600.2014.907095.

Goodall, Colin R. 1991. *Graphical Models in Applied Multivariate Statistics. Technometrics*. Vol. 33. 4. Informa UK Limited. https://doi.org/10.1080/00401706.1991.10484880.

Hartigan, J. A., and B. Kleiner. 1981. "Mosaics for Contingency Tables." In *Computer Science and Statistics: Proceedings of the 13th Symposium on the Interface*, 268–73. Springer US. https://doi.org/10.1007/978-1-4613-9464-8_37.

Hastie, T. J., and R. J. Tibshirani. 2017. "Generalized Additive Models." In *Generalized Additive Models*, 136–73. Routledge. https://doi.org/10.1201/9780203753781-6.

Hintze, Jerry L., and Ray D. Nelson. 1998. "Violin Plots: A Box Plot-Density Trace Synergism." *The American Statistician* 52 (2): 181–84. https://doi.org/10.1080/00031305.1998.10480559.

Huber, Peter J. 1981. *Robust Statistics. Wiley Series in Probability and Statistics*. Wiley. https://doi.org/10.1002/0471725250.

Jolliffe, I. T., and M. J. Greenacre. 1986. "Theory and Applications of Correspondence Analysis." *Biometrics* 42 (1): 223. https://doi.org/10.2307/2531266.

Kendall, M. G. 1938. "A New Measure of Rank Correlation." *Biometrika* 30 (1/2): 81. https://doi.org/10.2307/2332226.

Lundberg, Scott M, and Su-In Lee. 2017. "A Unified Approach to Interpreting Model Predictions." In *Advances in Neural Information Processing Systems*, edited by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol.

30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf.

Parzen, Emanuel. 1962. "On Estimation of a Probability Density Function and Mode." *The Annals of Mathematical Statistics* 33 (3): 1065–76. https://doi.org/10.1214/aoms/1177704472.

Pearson, Karl. 1895. "VII. Note on Regression and Inheritance in the Case of Two Parents." *Proceedings of the Royal Society of London* 58 (347–352): 240–42. https://doi.org/10.1098/rspl.1895.0041.

———. 1901. "LIII. On Lines and Planes of Closest Fit to Systems of Points in Space." *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2 (11): 559–72. https://doi.org/10.1080/14786440109462720.

———. 1992. "On the Criterion That a Given System of Deviations from the Probable in the Case of a Correlated System of Variables Is Such That It Can Be Reasonably Supposed to Have Arisen from Random Sampling." In *Breakthroughs in Statistics*, 11–28. Springer New York. https://doi.org/10.1007/978-1-4612-4380-9_2.

Reimers, Nils, and Iryna Gurevych. 2019. "Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks." In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 3980–90. Association for Computational Linguistics. https://doi.org/10.18653/v1/d19-1410.

Reshef, David N., Yakir A. Reshef, Hilary K. Finucane, Sharon R. Grossman, Gilean McVean, Peter J. Turnbaugh, Eric S. Lander, Michael Mitzenmacher, and Pardis C. Sabeti. 2011. "Detecting Novel Associations in Large Data Sets." *Science* 334 (6062): 1518–24. https://doi.org/10.1126/science.1205438.

Rubin, Donald B. 1987. *Multiple Imputation for Nonresponse in Surveys. Wiley Series in Probability and Statistics*. Wiley. https://doi.org/10.1002/9780470316696.

Salton, G., A. Wong, and C. S. Yang. 1975. "A Vector Space Model for Automatic Indexing." *Communications of the ACM* 18 (11): 613–20. https://doi.org/10.1145/361219.361220.

Shannon, C. E. 1948. "A Mathematical Theory of Communication." *Bell System Technical Journal* 27 (3): 379–423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x.

Shapley, L. S. 1953. "17. A Value for n-Person Games." In *Contributions to the Theory of Games (AM-28), Volume II*, 307–18. Princeton University Press. https://doi.org/10.1515/9781400881970-018.

Sklar, M. 1959. "Fonctions de répartition à n Dimensions Et Leurs Marges." In *Annales de l'ISUP*, 8:229–31. 3.

Soetewey, Antoine, Cédric Heuchenne, Arnaud Claes, and Antonin Descampe. 2026. "AssociationExplorer: A User-Friendly Shiny Application for Exploring Associations and Visual Patterns." *SoftwareX* 33 (February): 102483. https://doi.org/10.1016/j.softx.2025.102483.

Spearman, C. 1961. "The Proof and Measurement of Association Between Two Things." In *Studies in Individual Differences: The Search for Intelligence.*, 45–58. Appleton-Century-Crofts. https://doi.org/10.1037/11491-005.

Székely, Gábor J., Maria L. Rizzo, and Nail K. Bakirov. 2007. "Measuring and Testing Dependence by Correlation of Distances." *The Annals of Statistics* 35 (6). https://doi.org/

10.1214/009053607000000505.

van Buuren, Stef, and Karin Groothuis-Oudshoorn. 2011. "mice: Multivariate Imputation by Chained Equations in r." *Journal of Statistical Software* 45 (3): 1–67. https://doi.org/10.18637/jss.v045.i03.