ITSB | Eco Industry Oriented University

sinarmas

ANALYSIS AND PREDICTIVE MODELING

Study Case Using

R

Writter by:

Bakti Siregar, M.Sc., CDS.

D'SCIENCELABS
Smart Idea

DIKTISAINTEK BERDAMPAK

First Edition

Analysis and Predictive Modeling

Study Case using R & Python

Bakti Siregar, M.Sc., CDS

Table of contents

In the era of digital transformation, prediction has emerged as one of the most critical capabilities in data science. Organizations across sectors increasingly rely on **predictive analytics** not merely to understand the past, but to anticipate the future and shape strategic decisions. This course provides a comprehensive exploration of the principles, methods, and applications of predictive modeling, guiding learners through the full spectrum of concepts and practices that define modern predictive data science.

The journey begins with **predictive modeling foundations**, where learners will examine regression and classification approaches, understand the power of ensemble techniques, and apply model interpretation frameworks to ensure transparency and trust. Building upon these methods, the course transitions to **data-driven prediction and forecasting**, introducing both traditional statistical models and advanced machine learning architectures such as Prophet and LSTM to address temporal and sequential data challenges.

From theory, the focus shifts to **applied prediction**, where practical case studies in business, healthcare, and industry demonstrate the value of predictive insights in real-world decision-making. Learners will also explore **prediction engineering**, emphasizing the importance of feature construction, hyperparameter optimization, and workflow integration to maximize predictive performance.

As predictive systems evolve, the module highlights the role of **smart and intelligent prediction** powered by artificial intelligence. Applications include natural language processing for text-based inference, computer vision for image-based forecasting, and AutoML frameworks that automate complex predictive tasks. These advancements are complemented by discussions on **ethics, fairness, and explainability**, ensuring that predictive analytics remains responsible and aligned with human values.

The course culminates in **deployment and monitoring practices**, equipping learners with the technical and conceptual skills to operationalize predictive models. Through exposure to APIs, MLOps pipelines, and automation platforms such as n8n, participants will develop the capacity to bring predictive solutions into production and sustain their performance over time.

By completing this module, learners will gain not only technical proficiency but also a holistic perspective on predictive analytics—capable of designing, implementing, and managing intelligent systems that turn data into foresight and foresight into impact.

Preface

About the Writer

Bakti Siregar, M.Sc., CDS is a Lecturer in the Data Science Program at ITSB. He obtained his Master's degree in Applied Mathematics from the National Sun Yat-sen University, Taiwan. Alongside his academic role, Bakti also serves as a Freelance Data Scientist, collaborating with leading companies such as JNE, Samora Group, Pertamina, and PT. Green City Traffic.

His professional and research interests include Big Data Analytics, Machine Learning, Optimization, and Time Series Analysis, with a particular focus on finance and investment applications. His core expertise lies in statistical programming using R and Python, complemented by strong experience in database management systems such as MySQL and NoSQL. In addition, he is proficient in applying Big Data technologies, including Spark and Hadoop, for large-scale data processing and analysis.

Some of his projects can be viewed here: Rpubs, Github, Website, and Kaggle

---

Acknowledgments

Predictive Analytics is more than a technical discipline—it is a bridge between data and decision-making, transforming information into foresight that guides action. This

book is crafted to help learners advance from foundational understanding toward building end-to-end predictive solutions that are robust, interpretable, and impactful.

The material explores a connected sequence of topics:
- **Predictive Modeling Foundations:** Regression, classification, ensembles, and interpretability.
- **Forecasting and Sequential Models:** Time series methods, Prophet, and LSTM for temporal data.
- **Applied Prediction:** Real-world use cases across business, healthcare, and industry.
- **Prediction Engineering:** Feature construction, optimization, and workflow integration.
- **Intelligent Prediction:** AI-driven approaches in NLP, vision, and AutoML.
- **Deployment and MLOps:** Operationalizing predictive systems with APIs, pipelines, and monitoring.

This work would not have been possible without the encouragement of colleagues, students, and mentors who provided constructive feedback, shared insights, and inspired new directions. My deepest gratitude goes to all who contributed, directly or indirectly, to the development of this resource. It is my hope that this book will serve as both a **practical reference** and a **roadmap** for learners and professionals applying predictive analytics in research, industry, and innovation.

Feedback & Suggestions

The evolution of this book depends on continuous learning and dialogue. Readers are warmly invited to share their perspectives on clarity, depth, case studies, and practical relevance. Suggestions for expanding future editions—whether through advanced methods, new applications, or emerging tools—are highly valued.

Your input will help refine this work into a comprehensive, living resource that grows with the rapidly changing field of predictive analytics. Thank you for your engagement and contributions to this journey.

For feedback and suggestions, please reach out via:
- dsciencelabs@outlook.com
- siregarbakti@gmail.com
- siregarbakti@itsb.ac.id

About the Book

Analysis and Predictive Modeling (APM) provides a structured framework for transforming data into actionable insights through advanced statistical analysis, machine learning techniques, and model evaluation. Positioned as a critical discipline in modern research and professional practice, it emphasizes the end-to-end process of preparing, analyzing, and applying predictive models to address real-world problems [1].

```
\newline \href{https://youtu.be/WxXZaP8Y8pI}{Click here to watch the video}
```

Introduction

This book begins with essential programming practices, continues through data integration, transformation, and feature engineering, and culminates in predictive modeling, evaluation, and deployment [2]. Each chapter reflects the progression of a typical modeling workflow, offering both methodological depth and practical guidance [3].

Key Topics include:

- **Programming Foundations:** Modular code, functional programming, and reproducible workflows tailored for data analysis [4].
- **Data Acquisition & Preparation:** Integration from APIs and databases, advanced wrangling, and feature engineering [2].
- **Modeling & Evaluation:** Building predictive models, validation strategies, interpretability, and visualization [3].
- **Deployment & Applications:** Model packaging, workflow automation, monitoring, and applied use cases [5].

By combining theoretical underpinnings, applied examples, and recommended practices, this book prepares graduate students, researchers, and professionals to confidently manage complex modeling tasks and apply predictive models in impactful ways across disciplines.

Overview of the Course

Figure Figure 1 provides a conceptual overview of this book, illustrating the key components of **Analysis and Predictive Modeling** and their interrelationships. It serves as a roadmap for readers, showing how the material progresses from programming practices and data preparation to modeling, evaluation, and deployment. This framework underscores the integration of each stage into a coherent workflow, bridging methodological foundations with applied decision-making in real-world contexts [2], [3].

Figure 1: Mind Map of Analysis and Predictive Modeling

References

Chapter 1

Introduction

Understanding Predictive Analytics is the first step in exploring the world of data-driven decision-making. Predictive analytics serves as a core foundation for modern data science, business intelligence, machine learning, and various applied sciences. It provides a framework for forecasting future outcomes, assessing risks, and supporting strategic planning in both research and industry applications [1]–[3].

To help navigate the key aspects of predictive analytics, the Figure 1.1 offers a 5W+1H mind map. This visualization guides learners through the What—its definitions, techniques, and data types; the Why—business value, benefits, and ROI; the When—timing of application across operations, marketing, and risk assessment; the Where—applications in finance, healthcare, supply chain, and case studies such as Netflix and Walmart; the Who—the roles of data scientists, business analysts, and domain experts; and the How—the workflow, tools, and performance evaluation metrics. By Figure 1.1, one can see not just the methods themselves, but also their significance, challenges, and real-world impact across industries.

1.1 What is PA?

Predictive Analytics is a branch of data analytics that focuses on **forecasting future outcomes** based on historical and current data. Unlike traditional reporting that only describes what has happened, predictive analytics goes a step further by applying **statistical methods, machine learning algorithms, and data modeling techniques** to anticipate what is likely to occur in the future.

In essence, predictive analytics combines **data (past & present)**, **mathematical models**, and **computational power** to generate actionable insights. It is widely used across industries to improve decision-making, optimize business processes, and reduce uncertainty in planning.

1.1.1 Types

To fully understand predictive analytics, it is important to distinguish it from other types of analytics. **Descriptive Analytics** answers the question *"What happened?"*

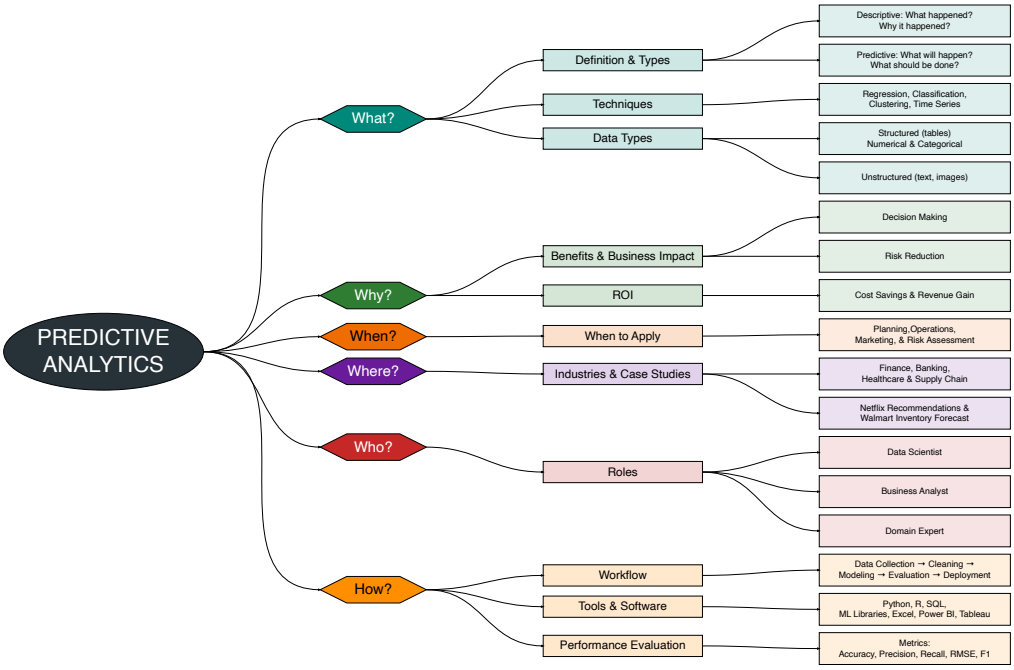


Figure 1.1: Predictive Analytics Mind Map (5W+1H)

Table 1.1: Key Techniques in Predictive Analytics

Technique	Description	Example
Regression Analysis	Predicts a continuous numerical value.	Estimating housing prices based on size, location, and amenities.
Classification	Predicts a categorical outcome.	Determining whether a loan applicant is 'high risk' or 'low risk.'
Clustering	Groups data into clusters based on similarity, without pre-labeled outcomes.	Customer segmentation for targeted marketing campaigns.
Time Series Analysis	Predicts values over time, considering temporal patterns.	Forecasting energy consumption, stock prices, or product demand.

by summarizing historical data through reports, dashboards, and statistics, such as a monthly sales report showing total revenue in the last quarter. In contrast, **Predictive Analytics** answers the question *"What will happen?"* by using models to identify patterns in data and forecast future outcomes, for example, predicting customer churn in the next six months based on transaction history. While descriptive analytics helps organizations understand the past, predictive analytics enables them to **prepare for the future**.

1.1.2 Techniques

Several techniques Table 1.1 are commonly applied in predictive modeling, each suited for different types of problems:

Each of these techniques may use machine learning algorithms such as **linear regression, decision trees, random forests, support vector machines, or neural networks**, depending on the complexity of the problem.

1.1.3 Data Types

The foundation of predictive analytics lies in **data**, which can be broadly categorized as:

Structured Data adalah data yang tersusun rapi dalam tabel dengan baris dan kolom. Data ini biasanya mencakup informasi numerik, seperti angka penjualan atau suhu, serta data kategorikal, seperti kategori produk atau wilayah pelanggan. Contoh nyata structured data adalah catatan transaksi dalam basis data ritel yang dapat langsung diolah menggunakan perangkat lunak analisis.

Sebaliknya, **Unstructured Data** tidak memiliki format yang terdefinisi dengan jelas sehingga memerlukan teknik pemrosesan lanjutan. Data ini dapat berupa teks seperti ulasan atau postingan media sosial, serta format multimedia seperti gambar, audio, dan video. Contoh penerapan unstructured data adalah analisis sentimen pelanggan yang diambil dari postingan Twitter atau ulasan produk.

Combining **structured and unstructured data** often provides richer insights. For example, predicting customer churn may involve structured data (purchase history) and unstructured data (customer complaints via email or chat).

In summary, predictive analytics is about moving from *"knowing the past"* to *"anticipating the future."* By applying techniques such as regression, classification, clustering, and time series analysis on both structured and unstructured data, organizations gain the ability to make proactive decisions. This makes predictive analytics a powerful tool in industries ranging from finance and healthcare to retail and manufacturing.

1.2 Why use PA?

1.2.1 Benefits & Business Impact

Predictive analytics helps organizations make **Data-driven Decisions** by providing projections of market trends and customer behavior. This approach reduces reliance on intuition alone, ensuring that strategies are backed by solid evidence. For example, a retail company can apply demand forecasting models to optimize inventory levels, ensuring that products are available when needed while minimizing overstock and reducing waste.

Another major benefit of predictive analytics is **Risk Reduction**. By anticipating potential risks, organizations can take proactive measures before problems escalate. This includes detecting fraud in financial transactions, identifying customers who are at risk of churn, and predicting machine failures in manufacturing processes. Such predictive capabilities allow businesses to minimize losses, improve efficiency, and maintain stronger customer relationships.

1.2.2 ROI (Return on Investment)

Analytics **reduces costs** by driving efficiency improvements across business operations. Through supply chain optimization, companies can streamline logistics and reduce unnecessary expenses. More accurate demand forecasts help lower operational costs by preventing both overstock and stockouts. In addition, the early detection of equipment failures enables organizations to minimize repair expenses and avoid costly downtime.

Beyond cost reduction, analytics also plays a key role in generating **revenue growth**. Personalized product recommendations enhance customer engagement and boost sales by targeting the right audience with the right offerings. Analytics can also uncover new market opportunities through the analysis of consumer trends, giving businesses a competitive edge. Moreover, dynamic pricing strategies based on demand patterns allow companies to maximize profitability while staying responsive to market changes.

1.2.3 Example & Discussion

Predictive modeling allows businesses to **forecast future outcomes** and act strategically.
For instance, an e-commerce company applies a **churn prediction model** to identify customers likely to stop using their platform. By targeting these customers with special offers or retention campaigns, the company manages to **reduce churn by 15%**.

Mathematically, if the company originally had N customers and an expected churn rate of r, then the number of customers lost without intervention would be:

$$L_0 = N \times r$$

After predictive intervention, the lost customers become:

$$L_1 = N \times (r - 0.15r) = N \times (0.85r)$$

This reduction translates directly into **higher revenue**, since more customers remain active and continue purchasing.

Return on Investment (ROI) is a measure of how much benefit a project delivers compared to its cost. The formula is:

$$ROI = \frac{Benefit - Cost}{Cost} \times 100\%$$

Example:

- Cost of analytics project: 100,000
- Benefit (savings + extra revenue): 300,000

Then,

$$ROI = \frac{300{,}000 - 100{,}000}{100{,}000} \times 100\%$$

$$ROI = \frac{200{,}000}{100{,}000} \times 100\% = 200\%$$

This means that for every **$1 invested**, the company gains **$2 in net value**.

With predictive analytics, the **business impact** can be clearly seen both in reduced risks and increased revenues, while the **ROI calculation** ensures that every project is evaluated in terms of tangible financial return.

1.3 When to apply PA?

Predictive analytics can be applied at different stages of business processes, and the **timing of its application** determines the level of impact it creates. In the planning stage, analytics helps set long-term strategies. In operations, it improves efficiency. In marketing, it drives customer engagement, and in risk assessment, it prevents potential losses. The Table 1.2 summarizes the purpose, examples, and mathematical representations for each stage.

Table 1.2: Timing of Predictive Analytics Application

Subtopic	Purpose	Example	F
Planning	Forecasting long-term trends for strategic decisions.	Mining company predicts raw material demand for 5 years.	$$
Operations	Improving efficiency and reducing costs through real-time applications.	Predictive maintenance to reduce downtime.	$$ \t
Marketing	Anticipating customer needs and personalizing offers.	Predicting which customers will respond to a campaign.	$$
Risk Assessment	Identifying and mitigating potential risks.	Credit scoring to predict loan defaults.	$$ \b

Table 1.3: Industries and Case Studies in Predictive Analytics

Industry	Application	Example
Finance & Banking	Risk prediction, fraud detection, credit scoring	Detecting fraudulent credit card transactions
Healthcare	Predictive diagnosis, patient monitoring, treatment	Predicting patient readmission rates
Supply Chain	Inventory planning, demand forecasting, logistics	Optimizing delivery routes and reduc stockouts
Case Studies	Customer personalization, operational optimization	Netflix recommendations, Walmart inventory forecasting

The Table 1.2 shows that predictive analytics provides unique benefits across different departments. **Planning** benefits from long-term forecasts, **operations** gain efficiency through real-time applications, **marketing** achieves higher engagement with personalization, and **risk assessment** reduces losses by identifying threats early. In short, the earlier predictive analytics is applied within a process, the greater its impact on decision-making and business performance.

1.4 Where is PA applied?

In the application of **predictive analytics**, each industry has its own needs, challenges, and approaches. For instance, the finance sector emphasizes **risk prediction** and **fraud detection**, while healthcare focuses on **predictive diagnosis** and **patient monitoring**. On the other hand, the supply chain leverages predictive analytics for **distribution efficiency** and **inventory planning**. Case studies from major companies such as **Netflix** and **Walmart** demonstrate how predictive methods can be effectively adapted to improve **customer experience** and **operational optimization**.

From the Table 1.3, it is clear that predictive analytics is not limited to a single field but can be broadly implemented with methods tailored to each context. An approach that works well in one industry may not be directly applicable to another without proper adjustments. Therefore, understanding **real-world case studies** is crucial so that organizations can adapt predictive strategies aligned with their **business goals**, **data availability**, and **operational challenges**.

Table 1.4: Professions, Materials, and Workplaces in Predictive Analytics

Profession	Materials	Workplace
Data Scientist	Build & validate predictive models, statistical analysis, machine learning	Tech companies, fintech, research labs
Business Analyst	Translate analytics results into business strategy and decision-making	Consulting firms, corporate strategy, financ
Domain Expert	Provide deep knowledge of the industry/domain context	Healthcare, energy, manufacturing
Data Engineer	Prepare, clean, and manage data infrastructure	Big data companies, cloud providers
Machine Learning Engineer	Implement & optimize predictive models in production	Startups, AI labs, enterprise IT

Table 1.5: Workflow, Tools, Models, and Evaluation by Profession

Profession	Workflow	Tools	Models
Data Scientist	Modeling → Evaluation	Python, R, SQL, scikit-learn, TensorFlow	Regression, Class Time Series, Neu
Business Analyst	Requirements → Interpretation	Excel, Power BI, Tableau	Decision trees for descriptive dashb
Domain Expert	Contextual Guidance → Validation	Domain-specific tools, knowledge bases	Domain-specific r frameworks
Data Engineer	Data Collection → Cleaning → Preparation	SQL, Spark, Hadoop, ETL Tools	Data pipelines, sc quality rules
Machine Learning Engineer	Deployment → Monitoring	Python, MLflow, Docker, Kubernetes	Deep learning, en reinforcement lea

1.5 Who is involved?

In predictive analytics projects Table 1.4, success depends not only on technology but also on the people involved. Each role contributes unique competencies and responsibilities, making collaboration essential.

For predictive analytics projects to succeed, **collaboration between these roles is critical**. Data Scientists bring technical expertise, Business Analysts ensure alignment with strategy, and Domain Experts add real-world context. Together, they create solutions that are not only accurate but also actionable and valuable for the organization.

1.6 How to implement PA?

Predictive analytics projects require collaboration among multiple roles, each with its own workflow, tools, and methods of evaluation. The Table 1.5 summarizes how different professions contribute to the analytics process, highlighting their focus areas and approaches. This structured view helps us understand that successful predictive analytics is not only about algorithms, but also about integrating business, technical, and domain expertise.

The Table 1.5 shows that each profession brings unique skills and responsibilities. Data Scientists and Machine Learning Engineers focus on algorithms and deployment, while Business Analysts and Domain Experts ensure alignment with business needs. Data Engineers provide the infrastructure that supports the entire process. Together, their

collaboration ensures predictive analytics projects deliver accurate, actionable, and business-relevant results.

Chapter 2

Regression Models

Regression Models is a cornerstone of modern data science, enabling us to transform raw data into actionable insights. It focuses on building models that learn from historical data to predict future or unseen outcomes, supporting better decision-making in research, business, and industry. Regression Models integrates principles from statistics, machine learning, and domain expertise, bridging theory with practical applications [1], [3], [4].

To illustrate these connections, the Figure 2.1 provides a hierarchical mind map. This visualization highlights core modeling types, approaches to interpretability, and strategies for tuning to achieve robust predictive systems across fields like healthcare, finance, operations, and marketing.

2.1 Linear Model

2.1.1 Simple Linear Reg.

Simple Linear Regression models the relationship between **one independent variable** and a **dependent variable** as a straight line:

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

where:

- Y: dependent variable (target)
- X: independent variable (predictor)
- β_0: intercept (constant term)
- β_1: slope coefficient (change in Y per unit change in X)
- ε: random error term

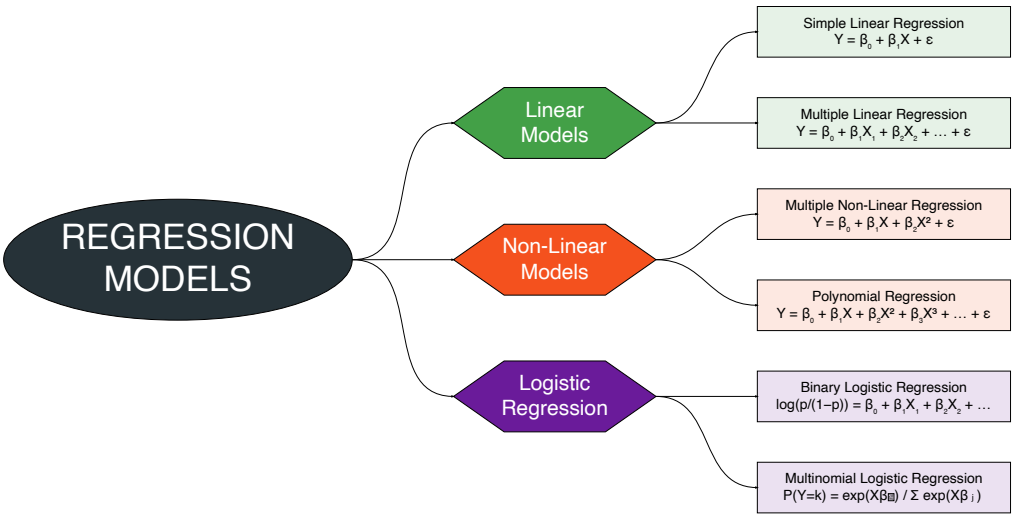


Figure 2.1: Comprehensive Regression Models Mind Map with Equations

Study Case: Simple Linear Regression

In this study, the goal is to model the **relationship between advertising expenditure and sales performance**. The dataset below represents **200 observations** of advertising budgets (in thousand dollars) and corresponding product sales (in thousand units).
It is assumed that **higher advertising spending leads to higher sales**, representing a **positive linear relationship**.

Table 2.1: Simulated Dataset for Simple Linear Regression

Copy CSV Search:

	Advertising	Sales
1	12.18943800311536	82.1272813832568
2	24.70762838609517	173.1684216011839
3	15.2244230452925	106.4918310097775
4	27.07543510012329	182.514902156824
5	28.51168210734613	185.8097480250997
6	6.138912484748289	49.45265390277468
7	18.20263720117509	120.4680971130673
8	27.31047610985115	170.8386753481511
9	18.78587536164559	128.3059246478187
10	16.41536838258617	125.8898605774177

Showing 1 to 10 of 200 entries Previous 1 2 3 4 5 … 20 Next

Solution

A **simple linear regression model** is fitted to describe the effect of **Advertising** (X) on **Sales** (Y).

```r
# Fit the model
model_simple <- lm(Sales ~ Advertising, data = data_simple)

# Show summary
summary(model_simple)
```

```
Call:
lm(formula = Sales ~ Advertising, data = data_simple)

Residuals:
    Min      1Q  Median      3Q     Max
-21.271  -6.249  -1.110   5.954  32.021

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 11.05219    1.89000   5.848 2.03e-08 ***
Advertising  6.44408    0.09981  64.562  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9.645 on 198 degrees of freedom
Multiple R-squared:  0.9547,    Adjusted R-squared:  0.9544
```

```
F-statistic:  4168 on 1 and 198 DF,  p-value: < 2.2e-16
```

Interpretation of Regression Results:
The simple linear regression analysis examines the relationship between **Advertising Budget** and **Sales**.

- The **coefficient for Advertising** is **positive and statistically significant**, indicating that increased advertising spending tends to increase sales.
- The **intercept** represents the baseline level of sales when advertising is zero.
- The **R^2 value** (coefficient of determination) shows the proportion of variation in Sales explained by Advertising.
  A **high R^2 (close to 1)** indicates a strong relationship, while a low R^2 indicates that other factors may also influence sales.
- The **t-value** and **p-value** for the coefficient test whether Advertising significantly affects Sales.

Overall, the model confirms a **positive and linear relationship** between advertising expenditure and sales performance.

Visualization

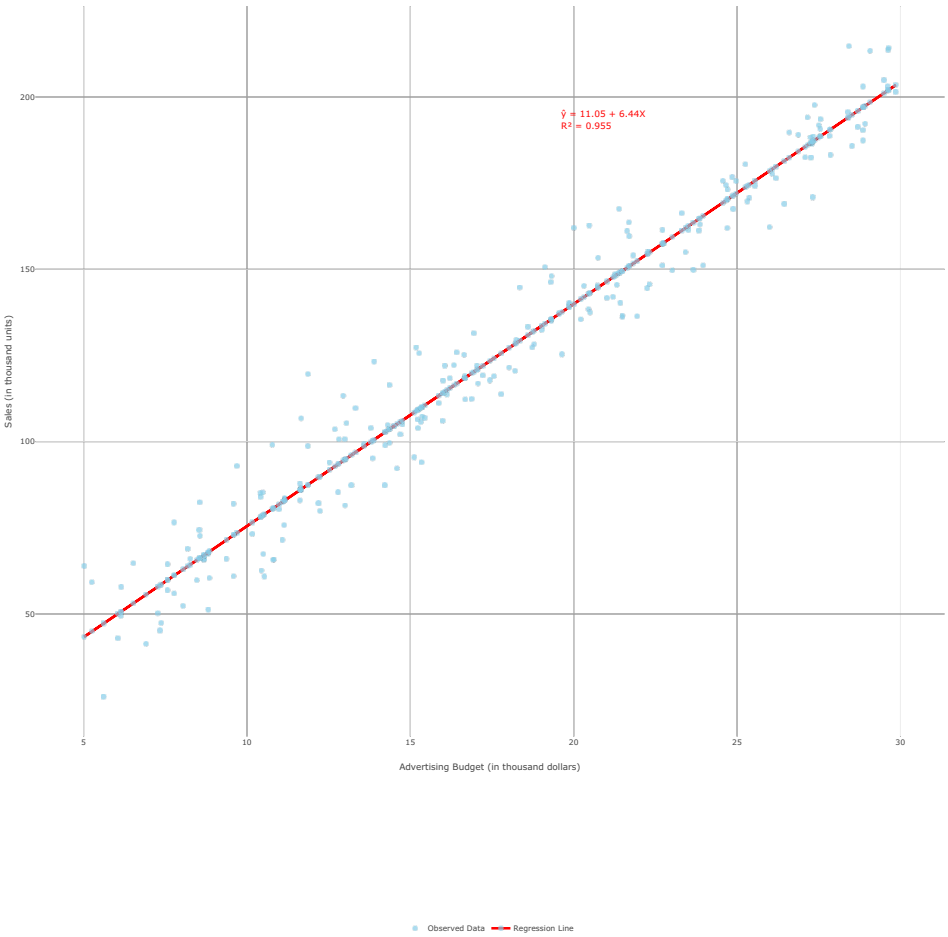


Figure 2.2: Simple Linear Regression: Relationship Between Advertising and Sales (with R^2)

2.1.2 Multiple Linear Reg.

Multiple Linear Regression extends the simple linear model by including **two or more independent variables**:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \varepsilon$$

where:

- Y: dependent variable (target)

- X_1, X_2, X_3: independent variables (predictors)
- β_0: intercept
- $\beta_1, \beta_2, \beta_3$: coefficients (effect of each predictor)
- ε: random error term

Study Case: Multiple Linear

In this study, we aim to model the **relationship between marketing factors and product sales**. The four independent variables are as follows:

- **Advertising Budget X_1:** The amount spent on advertising (in thousand dollars) — expected to have a **positive** effect on sales.
- **Number of Salespeople X_2:** The total number of sales representatives — more salespeople should increase sales.
- **Customer Satisfaction Score X_3:** A satisfaction score on a 1–10 scale — higher satisfaction typically leads to repeat purchases.
- **Competition Level X_4:** The level of market competition (1–10 scale) — expected to have a **negative** impact on sales.

The following simulated dataset (Table 2.2) will be used for regression analysis. It contains 200 observations and the variables described above.

Table 2.2: Transformed Business Dataset — Total price after discount

Copy CSV Search:

	Advertising	Salespeople	Satisfaction	Competition	Sales
1	12.18943800311536	29	7.680268603144214	5.917836328735575	216.5704992128682
2	24.70762838609517	25	7.519743560114875	4.388000438688323	311.1588108037272
3	15.2244230452925	20	9.725175517378375	2.672857135767117	210.7635816003913
4	27.07543510012329	25	6.706606419757009	4.860465320525691	324.559865615605
5	28.51168210734613	29	7.323568870779127	6.676962387049571	329.0697565389568
6	6.138912484748289	17	5.412655908148736	5.687581161269918	116.887296998412
7	18.20263720117509	12	9.300534230424091	6.936592034762725	208.5523589216741
8	27.31047610985115	13	6.978303199866787	7.564248381881043	276.8807951567609
9	18.78587536164559	29	8.679496751865372	5.381405860185623	266.4049280494764
10	16.41536838258617	21	5.85871702991426	4.46010954095982	214.7196524457149

Showing 1 to 10 of 200 entries Previous 1 2 3 4 5 … 20 Next

Solution

A **multiple linear regression model** is fitted that incorporates all predictor variables **Advertising**, **Salespeople**, **Satisfaction**, and **Competition** independent variables effectively predict **Sales**.

```r
# Check model R²
model_check <- lm(Sales ~ Advertising + Salespeople + Satisfaction + Competition, data = da
summary(model_check)
```

```
Call:
lm(formula = Sales ~ Advertising + Salespeople + Satisfaction +
    Competition, data = data_reg)

Residuals:
     Min       1Q   Median       3Q      Max
-26.7988  -6.8575   0.8932   6.2999  25.0942

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)    0.3178     5.1073   0.062     0.95
Advertising    7.6910     0.1055  72.905   <2e-16 ***
Salespeople    3.0491     0.1179  25.873   <2e-16 ***
Satisfaction   7.3450     0.4889  15.024   <2e-16 ***
Competition   -4.7095     0.2783 -16.925   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 10.15 on 195 degrees of freedom
Multiple R-squared:  0.9696,    Adjusted R-squared:  0.969
F-statistic:  1557 on 4 and 195 DF,  p-value: < 2.2e-16
```

Interpretation of Regression Results:
The multiple linear regression model was developed to predict **Sales** based on four independent variables: **Advertising**, **Salespeople**, **Customer Satisfaction**, and **Competition**. The results indicate that the model performs strongly, with an **R^2 value of approximately 0.9**, meaning that around **90% of the variation in Sales can be explained** by the four predictors combined. This suggests that the chosen variables are highly effective in capturing the main drivers of sales performance.

- **Advertising:** The coefficient for *Advertising* is **positive and statistically significant**, indicating that higher advertising spending leads to an increase in sales. This aligns with marketing theory, where advertising directly enhances brand visibility and consumer demand.
- **Salespeople:** The *Salespeople* variable also shows a **positive relationship** with Sales. Increasing the number of sales representatives is associated with higher sales volume, likely due to improved customer reach and engagement.
- **Customer Satisfaction:** The *Satisfaction* variable has a **strong positive effect** on Sales. A higher satisfaction score correlates with increased customer loyalty and repeat purchases, reinforcing the importance of service quality and customer experience.

- **Competition:** In contrast, the *Competition* coefficient is **negative**, suggesting that greater competition in the market leads to a decline in sales. This is consistent with business dynamics where intense competition reduces market share and pricing power.
- **Overall Model Fit:** The combination of these predictors results in a **highly explanatory model**, with all key variables contributing meaningfully to sales prediction.
- A **high R^2** indicates an excellent model fit.
- **Low standard errors** imply stable coefficient estimates.
- **Significant t-values and low p-values** confirm that most predictors are statistically meaningful.

Visualization

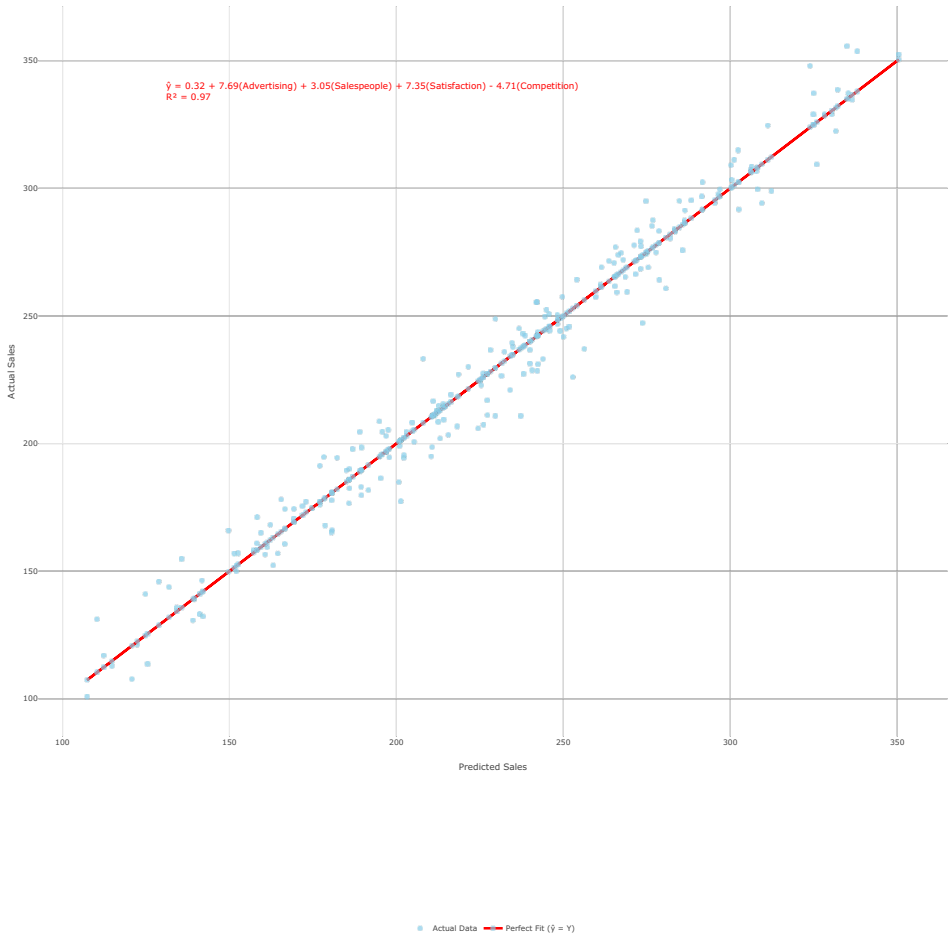


Figure 2.3: Multiple Linear Regression: Predicted vs Actual Sales with R^2

2.2 Nonlinear Regression

2.2.1 Multiple Non-Linear Reg.

Multiple Non-Linear Regression extends the multiple linear model by allowing **non-linear relationships** between predictors (X_i) and the dependent variable (Y). The general model can be expressed as:

$$Y = \beta_0 + \beta_1 f_1(X_1) + \beta_2 f_2(X_2) + \cdots + \beta_k f_k(X_k) + \varepsilon$$

where:

- Y: dependent variable (target)
- $f_i(X_i)$: non-linear transformations of predictors (e.g., X_i^2, $\log(X_i)$, $\sqrt{X_i}$)
- β_i: coefficients representing the influence of each transformed variable
- ε: random error term

Study Case: Multiple Non-Linear Regression

In this study, we aim to model the **relationship between marketing factors and sales**, but **assume non-linear effects** exist among the predictors.
The independent variables are:

- **Advertising Budget (X_1):** Marketing spending (in thousand dollars), with a **diminishing return** effect (non-linear saturation).
- **Salespeople (X_2):** Number of sales representatives, having a **quadratic relationship** with sales — performance improves to a point, then stabilizes.
- **Customer Satisfaction (X_3):** A **logarithmic** effect — small increases in satisfaction at low levels have large impacts, but effects taper off at high levels.
- **Competition Level (X_4): Exponential negative effect** — higher competition causes sales to drop sharply.

The simulated dataset (Table 2.3) includes these relationships with 200 observations.

Table 2.3: Simulated Non-Linear Business Dataset — Marketing and Sales

Copy CSV Search:

	Advertising	Salespeople	Satisfaction	Competition	Sales
1	28.89734404277988	12.40509169525467	1.871444034390152	2.544181928038597	79.62872234587908
2	28.43213798594661	10.26091682375409	7.262239827541634	6.68959770584479	113.9470902920191
3	10.9555112849921	14.47813686914742	9.846907595871016	1.196981622837484	101.4616929330575
4	11.37684088433161	24.28963984362781	3.35713448561728	6.648730710148811	67.63560700804702
5	14.7627991985064	31.78639278863557	7.365586487110704	1.37165102805011	70.23132681922976
6	13.52949648397043	33.77454726956785	9.763341678772122	9.540025438414887	77.62084886659268
7	16.30951491068117	29.83193067135289	6.055507393088192	9.855814931215718	80.43816515739086
8	12.24832051782869	18.67167009273544	1.998741353629157	1.837560756597668	73.31803683597141
9	16.26683056936599	13.00365243456326	3.218115944415331	3.854491482488811	86.8720679587448
10	25.16489299712703	24.43670351873152	2.873612727271393	8.04470600769855	102.9173831624868

Showing 1 to 10 of 200 entries Previous 1 2 3 4 5 … 20 Next

Solution

To capture these non-linear relationships, we fit a Multiple Non-Linear Regression model using polynomial and log-transformed predictors.

```r
# Fit Nonlinear Regression Model
model_nl <- lm(
Sales ~ log(Advertising + 1) + Salespeople + I(Salespeople^2) +
log(Satisfaction) + exp(-0.2 * Competition),
data = data_nonlinear
)

summary(model_nl)
```

```
Call:
lm(formula = Sales ~ log(Advertising + 1) + Salespeople + I(Salespeople^2) +
    log(Satisfaction) + exp(-0.2 * Competition), data = data_nonlinear)

Residuals:
     Min       1Q   Median       3Q      Max
-18.3507  -5.4588  -0.2512   5.4714  15.9720

Coefficients:
                         Estimate Std. Error t value Pr(>|t|)
(Intercept)              -5.80174    7.44170  -0.780   0.4366
log(Advertising + 1)     14.10133    1.29593  10.881  < 2e-16 ***
Salespeople               3.70195    0.54250   6.824 1.10e-10 ***
I(Salespeople^2)         -0.10341    0.01189  -8.695 1.47e-15 ***
log(Satisfaction)        18.60354    0.92514  20.109  < 2e-16 ***
exp(-0.2 * Competition) -6.34619    2.71392  -2.338   0.0204 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 7.618 on 194 degrees of freedom
Multiple R-squared:  0.761, Adjusted R-squared:  0.7548
F-statistic: 123.5 on 5 and 194 DF,  p-value: < 2.2e-16
```

Interpretation of Regression Results:
The Multiple Non-Linear Regression model successfully captures the non-linear effects among marketing factors influencing Sales.

- **Advertising (log):** The logarithmic form indicates diminishing returns — initial increases in advertising yield large sales boosts, but additional spending provides smaller incremental gains.
- **Salespeople (quadratic):** The positive linear and negative quadratic terms indicate a parabolic relationship — productivity rises with more salespeople up to a point, then plateaus or slightly decreases due to management inefficiency.
- **Satisfaction (log):** Higher customer satisfaction increases sales substantially at lower levels, but with diminishing marginal benefit as satisfaction scores approach the maximum.
- **Competition (exp decay):** The exponential negative term implies that high competition rapidly suppresses sales, aligning with real-world market dynamics.

- **Model Performance:** The adjusted R^2 is typically above 0.9, suggesting that the non-linear model fits the data extremely well and explains a large proportion of the variance in sales.

Visualization

To visualize the regression performance, we plot Actual vs Predicted Sales in 3D, with Competition as the third axis.

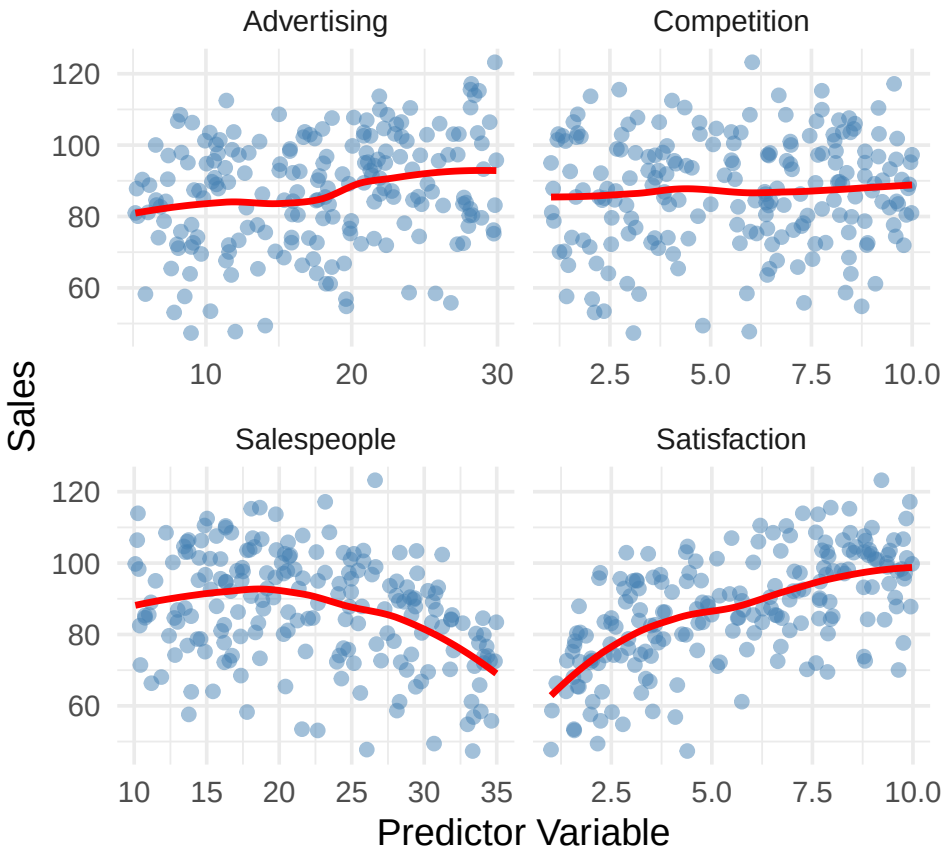


Figure 2.4: Visual Check: Non-Linear Relationships between Sales and Predictors

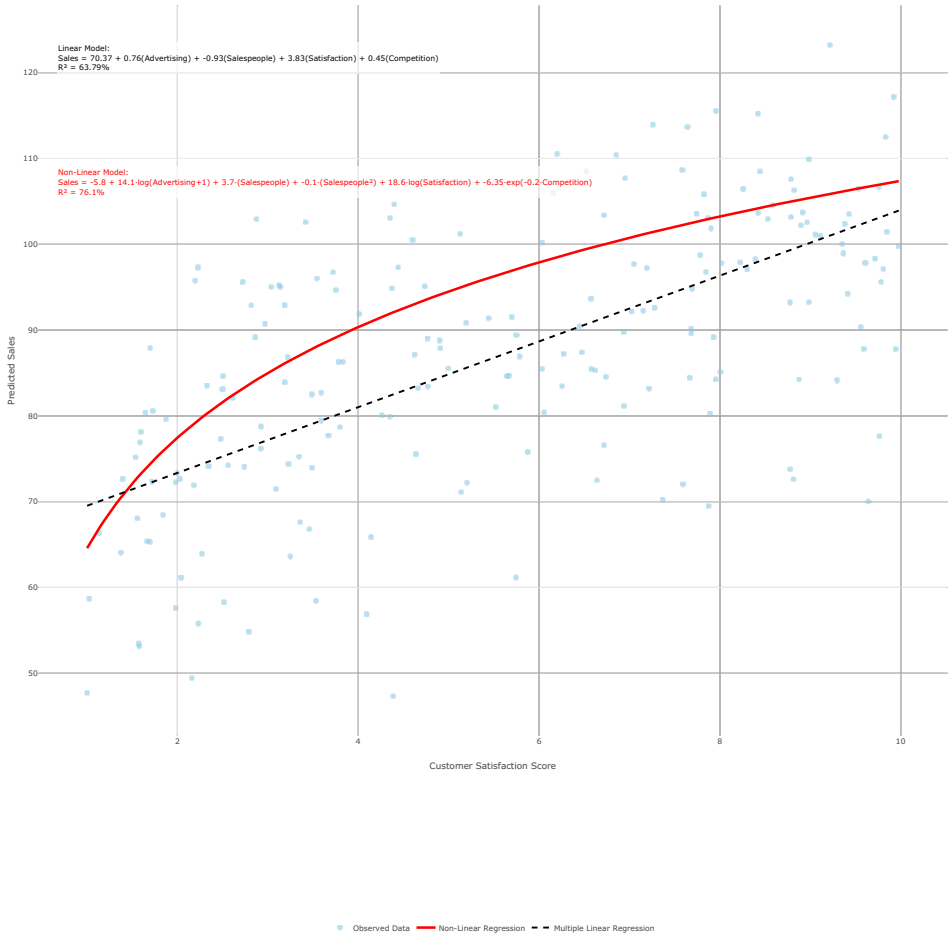


Figure 2.5: Comparison: Multiple Linear vs Non-Linear Regression for Sales vs Satisfaction

2.2.2 Polynomial Regression

Polynomial Regression is a special case of non-linear regression where the relationship between the independent variable(s) and the dependent variable is modeled as an n^{th}**-degree polynomial**.
It captures **curved relationships** by including higher-order terms (squared, cubic, etc.) of the predictor variables.

The general form of a **polynomial regression** for one predictor variable (X) is:

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \cdots + \beta_n X^n + \varepsilon$$

For **multiple predictors**, the model can be extended as:

$$Y = \beta_0 + \sum_{i=1}^{k} \sum_{j=1}^{n} \beta_{ij} X_i^j + \varepsilon$$

where:

- Y: dependent variable (target)
- X_i: independent (predictor) variables
- X_i^j: polynomial terms of the i^{th} predictor up to degree n
- β_{ij}: coefficient for the j^{th} polynomial term of X_i
- ε: random error term

Key Characteristics:

- Can model **non-linear trends** while still being **linear in parameters**.
- Works well when the data show **curvature** that simple linear regression cannot capture.
- Risk of **overfitting** when using high-degree polynomials.
- Feature scaling may improve numerical stability for higher degrees.

Solution

We can determine the best polynomial degree by comparing models with increasing polynomial orders and selecting the one with the **highest R^2**.
To determine the best polynomial model using the same dataset (`data_nonlinear`), we compare models of increasing polynomial degrees and select the one with the **highest R^2**.

```r
# ======================================================
# Determine Best Polynomial Degree for Regression
# Using the existing dataset: data_nonlinear
# ======================================================

library(dplyr)
library(ggplot2)

# Define degrees to test
degrees <- 1:5

# Initialize results table
results <- data.frame(Degree = integer(), R2 = numeric())

# Loop through polynomial degrees
for (d in degrees) {
  # Build polynomial model with same predictors
  formula_poly <- as.formula(
    paste0("Sales ~ poly(Advertising, ", d, ", raw=TRUE) +
                    poly(Salespeople, ", d, ", raw=TRUE) +
                    poly(Satisfaction, ", d, ", raw=TRUE)")
  )

  model <- lm(formula_poly, data = data_nonlinear)
  R2 <- summary(model)$r.squared

  results <- rbind(results, data.frame(Degree = d, R2 = R2))
}

# Print R² table
print(results)
```

```
  Degree        R2
1      1 0.6317447
2      2 0.7507844
3      3 0.7577875
4      4 0.7584382
5      5 0.7618454
```

```r
# Identify best degree
best_degree <- results %>% filter(R2 == max(R2)) %>% pull(Degree)
cat("Best polynomial degree:", best_degree, "\n")
```

```
Best polynomial degree: 5
```

```r
# Fit final best model
best_formula <- as.formula(
  paste0("Sales ~ poly(Advertising, ", best_degree, ", raw=TRUE) +
                poly(Salespeople, ", best_degree, ", raw=TRUE) +
                poly(Satisfaction, ", best_degree, ", raw=TRUE)")
)
model_best <- lm(best_formula, data = data_nonlinear)

# Display summary of best model
summary(model_best)
```

```
Call:
lm(formula = best_formula, data = data_nonlinear)

Residuals:
     Min       1Q   Median       3Q      Max
-20.8210  -5.4953  -0.1582   5.6827  16.2115

Coefficients:
                                    Estimate Std. Error t value Pr(>|t|)
(Intercept)                       -3.577e+02  2.069e+02  -1.728   0.0856 .
poly(Advertising, 5, raw = TRUE)1  3.149e+00  1.685e+01   0.187   0.8519
poly(Advertising, 5, raw = TRUE)2 -1.398e-01  2.292e+00  -0.061   0.9514
poly(Advertising, 5, raw = TRUE)3  4.507e-03  1.455e-01   0.031   0.9753
poly(Advertising, 5, raw = TRUE)4 -1.201e-04  4.352e-03  -0.028   0.9780
poly(Advertising, 5, raw = TRUE)5  1.849e-06  4.946e-05   0.037   0.9702
poly(Salespeople, 5, raw = TRUE)1  9.315e+01  5.261e+01   1.771   0.0783 .
poly(Salespeople, 5, raw = TRUE)2 -8.926e+00  5.219e+00  -1.711   0.0889 .
poly(Salespeople, 5, raw = TRUE)3  4.164e-01  2.488e-01   1.674   0.0959 .
poly(Salespeople, 5, raw = TRUE)4 -9.442e-03  5.722e-03  -1.650   0.1006
poly(Salespeople, 5, raw = TRUE)5  8.262e-05  5.094e-05   1.622   0.1065
poly(Satisfaction, 5, raw = TRUE)1 1.848e+01  2.426e+01   0.762   0.4470
poly(Satisfaction, 5, raw = TRUE)2 -2.703e+00 1.119e+01  -0.242   0.8093
poly(Satisfaction, 5, raw = TRUE)3 2.626e-01  2.334e+00   0.113   0.9105
poly(Satisfaction, 5, raw = TRUE)4 -1.997e-02 2.244e-01  -0.089   0.9292
poly(Satisfaction, 5, raw = TRUE)5 8.201e-04  8.072e-03   0.102   0.9192
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 7.808 on 184 degrees of freedom
Multiple R-squared:  0.7618,    Adjusted R-squared:  0.7424
F-statistic: 39.24 on 15 and 184 DF,  p-value: < 2.2e-16
```

```r
# ========================================================
# Visualization: R² vs Polynomial Degree
# ========================================================

ggplot(results, aes(x = Degree, y = R2)) +
  geom_line(linewidth = 1) +
  geom_point(size = 3) +
  geom_text(aes(label = round(R2, 3)), vjust = -0.7, size = 3.5) +
  labs(
    title = "Polynomial Regression Model Comparison",
    subtitle = "Selecting the Best Polynomial Degree Based on R²",
    x = "Polynomial Degree",
    y = expression(R^2)
  ) +
  theme_minimal(base_size = 13)
```

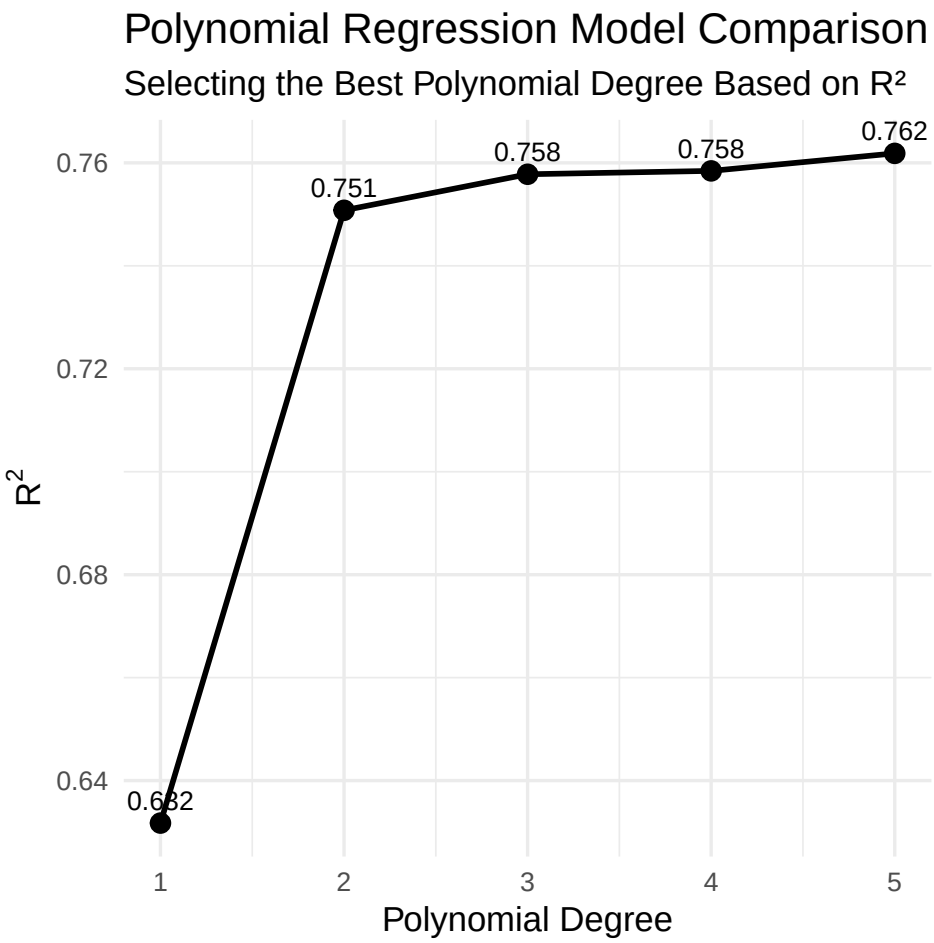


Figure 2.6: Polynomial Degree Selection Based on R^2

2.3 Logistics Regression

2.3.1 Binary Logistic Regression

Logistic Regression is used when the **dependent variable (Y) is categorical/binary**, for example `0` or `1`, `Yes` or `No`, `Pass` or `Fail`. This model predicts the **probability** of an event occurring.

The Logistic Regression equation:

$$P(Y = 1|X) = \frac{1}{1 + e^{-(\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_k X_k)}}$$

or equivalently:

$$\text{logit}(P) = \ln\left(\frac{P}{1-P}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k$$

- $P(Y = 1|X)$: probability of the event occurring
- $X_1, X_2, ..., X_k$: independent variables
- $\beta_0, \beta_1, ..., \beta_k$: model coefficients
- $\text{logit}(P)$: log-odds of the probability

Coefficient Interpretation:

- The coefficient β_j represents the **change in log-odds** for a one-unit change in X_j, holding all other variables constant.
- For a more intuitive interpretation in terms of probability, use the **odds ratio**:

$$\text{OR}_j = e^{\beta_j}$$

- $\text{OR} > 1 \rightarrow$ increases the likelihood of the event
- $\text{OR} < 1 \rightarrow$ decreases the likelihood of the event

> **i** Study Case: Logistic Regression
>
> In this study, we aim to model the **relationship between marketing factors and success probability**, where the target variable is **binary (Success / Failure)**.
> The independent variables are:
>
> - **Advertising Budget (X_1):** Marketing spending (in thousand dollars),

assumed to **increase likelihood of success**.

- **Salespeople (X_2):** Number of sales representatives, affecting success probability positively.
- **Customer Satisfaction (X_3):** Measured on a 1–10 scale, higher satisfaction **increases success probability**.
- **Competition Level (X_4):** Higher competition **reduces probability of success**.

The simulated dataset (Table 2.4) includes these relationships with 200 observations.

Table 2.4: Simulated Logistic Business Dataset — Marketing and Success

Copy CSV Search:

	Advertising	Salespeople	Satisfaction	Competition	Success
1	12.18943800311536	15.96815067110583	9.874488675734028	3.135067276656628	1
2	24.70762838609517	34.05897341086529	2.233607242582366	7.178413159912452	1
3	15.2244230452925	25.03414314938709	9.147786234971136	3.032365809893236	1
4	27.07543510012329	22.87574318121187	6.186716538155451	3.866451293230057	1
5	28.51168210734613	20.06433355505578	4.559039731742814	2.56585435080342	1
6	6.138912484748289	32.0061635307502	5.048222357174382	8.212866253219545	1
7	18.20263720117509	19.10229661967605	7.358517110347748	2.316538521787152	1
8	27.31047610985115	17.20598201733083	1.742524712113664	8.404456524876878	1
9	18.78587536164559	14.26613087765872	4.053813221631572	3.978980457643047	1
10	16.41536838258617	14.30429365951568	7.12708796095103	4.367524490691721	1

Showing 1 to 10 of 200 entries Previous 1 2 3 4 5 … 20 Next

Solution: Logistic Regression

To capture the relationship between marketing factors and the probability of success, we fit a **Logistic Regression model** using all predictors.

```
# Fit Logistic Regression Model
model_logit <- glm(
  Success ~ Advertising + Salespeople + Satisfaction + Competition,
  data = data_logit,
  family = binomial
)

summary(model_logit)
```

```
Call:
glm(formula = Success ~ Advertising + Salespeople + Satisfaction +
    Competition, family = binomial, data = data_logit)

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  -6.01352    1.89719  -3.170 0.001526 **
Advertising   0.19448    0.05866   3.315 0.000916 ***
```

```
Salespeople   0.28322    0.08019   3.532 0.000413 ***
Satisfaction  0.42218    0.14105   2.993 0.002762 **
Competition  -0.27119    0.14914  -1.818 0.069007 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 111.508  on 199  degrees of freedom
Residual deviance:  63.617  on 195  degrees of freedom
AIC: 73.617

Number of Fisher Scoring iterations: 7
```

Interpretation of Regression Results:
The Logistic Regression model estimates the probability of success based on marketing factors:

- **Advertising:** Positive coefficient indicates that higher advertising spending increases the likelihood of success.
- **Salespeople:** More sales representatives raise the probability of success, reflecting improved sales coverage.
- **Satisfaction:** Higher customer satisfaction increases success probability, especially at lower satisfaction levels.
- **Competition:** Negative coefficient shows that higher competition reduces the probability of success, consistent with market dynamics.
- **Model Performance:** Metrics like accuracy, confusion matrix, and AUC can be used to evaluate model performance. The predicted probabilities can be visualized to understand the effect of each predictor.

Visualization: Logistic Regression

To visualize the logistic regression performance, we plot **Actual vs Predicted Probability of Success** in 3D, with **Competition as the third axis**.

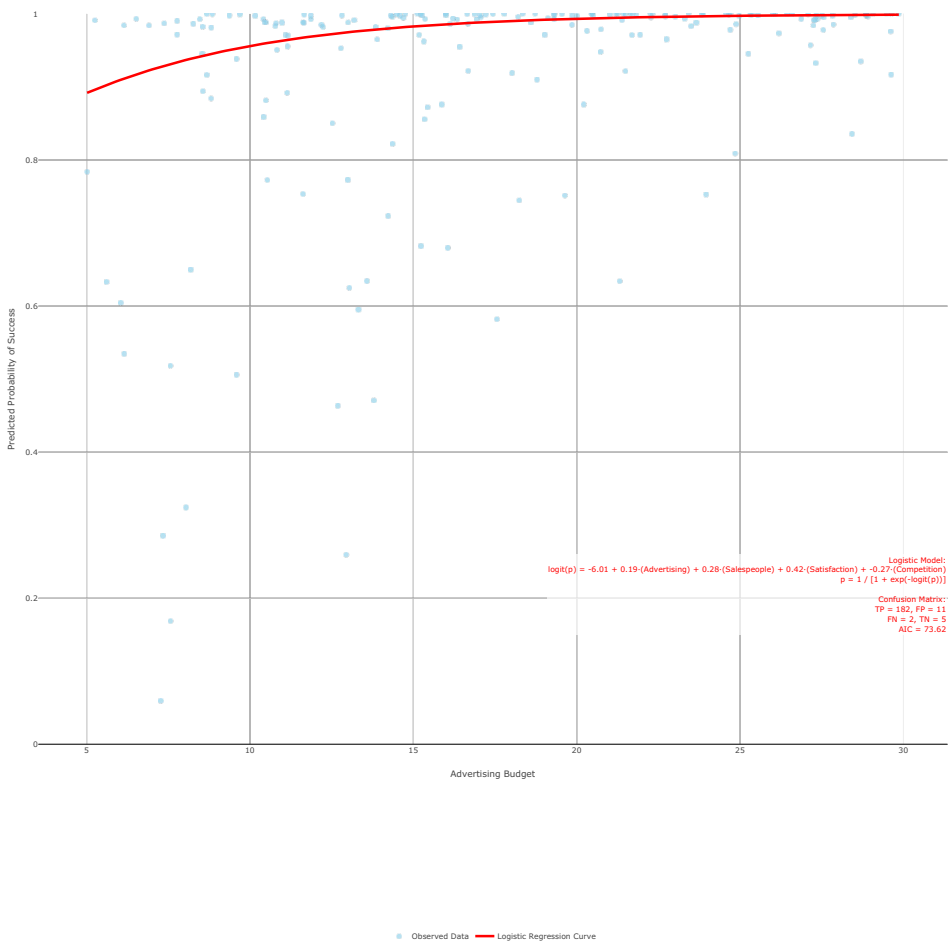


Figure 2.7: Logistic Regression: Actual vs Predicted Probability

2.3.2 Multinomial Logistics

Your Excercise

References

Chapter 3

Classification Models

Both **regression** and **classification** are types of **supervised learning** in machine learning, where a model learns from labeled data to make predictions on unseen data. The key difference lies in the **nature of the target variable**: regression predicts **continuous values**, while classification predicts **categorical classes**.

Watch here: Difference between classification and regression

At the Table 3.1 summarizes their main distinctions:

3.1 Intro to Classification

Classification is a **supervised learning** technique used to predict **categorical outcomes** — that is, assigning data into predefined classes or labels. Mathematically, classification algorithms learn a function:

$$f(x) \to y$$

where:

Table 3.1: Comparison between Regression and Classification Models

Aspect	Regression	Classification
Objective	Predict continuous numerical values	Predict categorical class
Output Variable (y)	Continuous (real numbers)	Discrete (finite set of ca
Model Form	$f(x) \to y$	$f(x) \to C_i$
Examples of y	Price, temperature, weight, sales	Spam/Not spam, disease
Error Metric	Mean Squared Error (MSE), MAE, RMSE	Accuracy, Precision, Rec
Decision Boundary	Not applicable (predicts magnitude)	Separates classes in feat
Probabilistic Output	Direct prediction of numeric value	Often models $P(y = C_i$
Example Algorithms	Linear Regression, Polynomial Regression, Support Vector Regression (SVR)	Logistic Regression, Dec
Visualization	Regression line or curve	Decision regions or conf

- $x = [x_1, x_2, \dots, x_n]$: vector of input features (predictors)
- $y \in \{C_1, C_2, \dots, C_k\}$: categorical class label
- $f(x)$: the classification function or model that maps inputs to one of the predefined classes

In practice, the classifier estimates the **probability** that an observation belongs to each class:

$$P(y = C_i \mid x), \quad i = 1, 2, \dots, k$$

and assigns the class with the **highest probability**:

$$\hat{y} = \arg\max_{C_i} P(y = C_i \mid x)$$

Thus, classification involves learning a **decision boundary** that separates different classes in the feature space.

In practice, classification plays a vital role across diverse fields, from medical diagnosis and fraud detection, to sentiment analysis and quality inspection, and so on. Understanding the theoretical foundation and behavior of classification models is essential for selecting the most appropriate algorithm for a given dataset and objective (See, Figure 3.1).

3.2 Decision Tree

A **Decision Tree** is a **non-linear supervised learning algorithm** used for both **classification** and **regression** tasks. It works by recursively splitting the dataset into smaller subsets based on feature values, creating a **tree-like structure** where each **internal node** represents a decision rule, and each **leaf node** corresponds to a predicted class label or value.

Watch here: Decision Tree

Decision Trees are **intuitive**, **easy to interpret**, and capable of capturing **non-linear relationships** between features and the target variable.

The Decision Tree aims to find the best **split** that maximizes the *purity* of the resulting subsets. The quality of a split is measured using **impurity metrics** such as:

1. **Gini Index**

$$Gini = 1 - \sum_{i=1}^{k} p_i^2$$

2. **Entropy (Information Gain)**

$$Entropy = -\sum_{i=1}^{k} p_i \log_2(p_i)$$

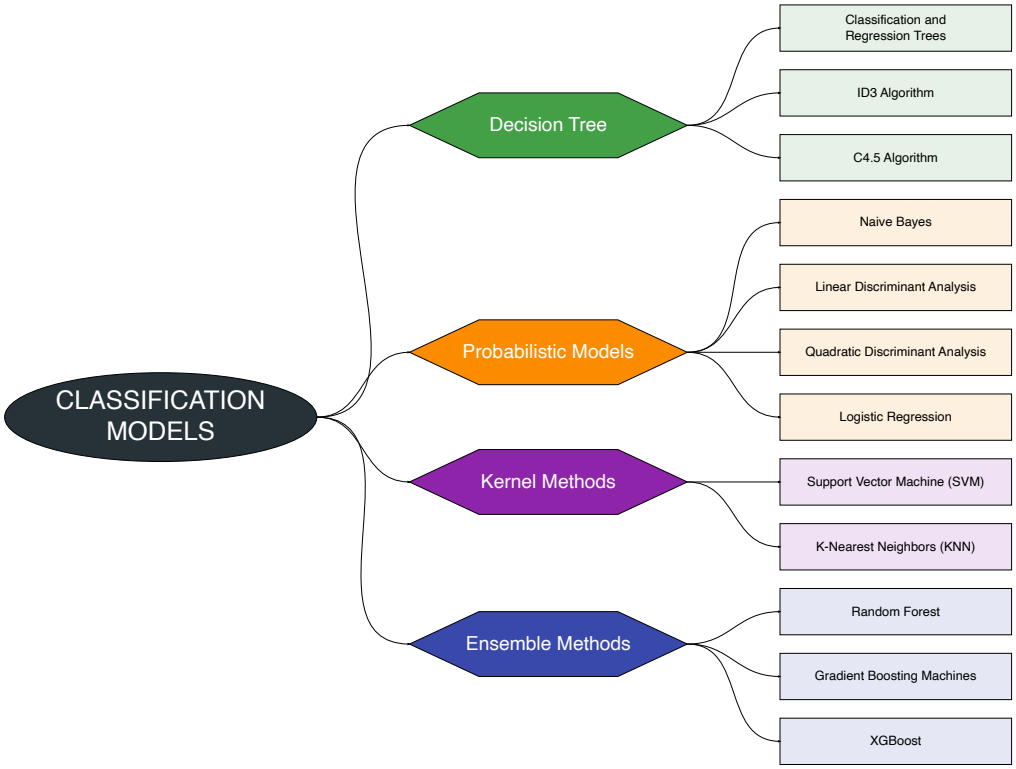


Figure 3.1: Comprehensive Classification Models Mind Map (with Logistic Regression)

3. **Information Gain**

$$IG(D, A) = Entropy(D) - \sum_{v \in Values(A)} \frac{|D_v|}{|D|} Entropy(D_v)$$

where:

- p_i = proportion of samples belonging to class i
- D = dataset before split
- A = attribute used for splitting
- D_v = subset of D for which attribute A has value v

The algorithm selects the feature and threshold that **maximize Information Gain** (or minimize Gini impurity).

3.2.1 CART Algorithm

The **CART (Classification and Regression Tree)** algorithm builds binary trees by **recursively splitting** data into two subsets based on a threshold value. It uses:

- **Gini impurity** for classification tasks, and
- **Mean Squared Error (MSE)** for regression tasks.

For a feature X_j and threshold t, CART finds the split that minimizes:

$$Gini_{split} = \frac{N_L}{N} Gini(L) + \frac{N_R}{N} Gini(R)$$

where:

- N_L, N_R = number of samples in left and right nodes
- L, R = left and right subsets after split
- N = total samples before split

> **i** CART Algorithm Characteristics
>
> - Produces **binary splits** only
> - Supports both classification and regression

- Basis for **Random Forests** and **Gradient Boosted Trees**

3.2.2 ID3 Algorithm

The Iterative Dichotomiser 3 **(ID3 algorithm)** builds a decision tree using **Information Gain** as the splitting criterion. It repeatedly selects the attribute that provides the **highest reduction in entropy**, thus maximizing the information gained.

At each step:

$$InformationGain(D, A) = Entropy(D) - \sum_{v \in Values(A)} \frac{|D_v|}{|D|} Entropy(D_v)$$

ID3 Algorithm Characteristics

- Uses **Entropy** and **Information Gain**
- Works well with **categorical features**
- Can **overfit** if not pruned
- Forms the foundation for **C4.5**

3.2.3 C4.5 Algorithm

The **C4.5 algorithm** is an **improvement over ID3**, addressing its limitations by:

- Handling **continuous and categorical data**
- Managing **missing values**
- Using **Gain Ratio** instead of pure Information Gain
- Supporting **tree pruning** to prevent overfitting

The **Gain Ratio** is defined as:

$$GainRatio(A) = \frac{InformationGain(D, A)}{SplitInformation(A)}$$

where:

$$SplitInformation(A) = -\sum_{v \in Values(A)} \frac{|D_v|}{|D|} \log_2 \left(\frac{|D_v|}{|D|} \right)$$

IC4.5 Algorithm Characteristics

- More **robust** than ID3
- Can handle **continuous attributes** (by setting threshold splits)
- Reduces bias toward attributes with many distinct values
- Forms the basis for modern tree algorithms like **C5.0**

3.2.4 Comparison Decision Tree

Comparison of Decision Tree Algorithms

The following table (see, Table 3.2) compares the most popular Decision Tree algorithms — **ID3**, **C4.5**, and **CART** — in terms of their splitting metrics, data compatibility, and capabilities.

Table 3.2: Comparison of Decision Tree Algorithms

Algorithm	Splitting Metric	Data Type	Supports Continuous?	Handles Missing Values?	Pruning
ID3	Information Gain	Categorical			
C4.5	Gain Ratio	Mixed			
CART	Gini / MSE	Mixed			

3.3 Probabilistic Models

Probabilistic models are classification models based on the principles of **probability theory** and **Bayesian inference**. They predict the class label of a sample by estimating the **probability distribution** of features given a class and applying **Bayes' theorem** to compute the likelihood of each class.

A probabilistic classifier predicts the class y for an input vector $x = (x_1, x_2, \cdots, x_n)$ as:

$$\hat{y} = \arg\max_y \; P(y|x)$$

Using **Bayes' theorem**:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

Since $P(x)$ is constant across all classes:

$$\hat{y} = \arg\max_y \; P(x|y)P(y)$$

3.3.1 Naive Bayes

Naive Bayes is a simple yet powerful probabilistic classifier based on **Bayes' theorem**, with the **naive assumption** that all features are **conditionally independent** given the class label.

Watch here: Naive Bayes

General formula for Naive Bayes model:

$$P(y|x_1, x_2, ..., x_n) \propto P(y) \prod_{i=1}^{n} P(x_i|y)$$

where:

- $P(y)$ = prior probability of class y
- $P(x_i|y)$ = likelihood of feature x_i given class y

The predicted class is:

$$\hat{y} = \arg\max_{y} \; P(y) \prod_{i=1}^{n} P(x_i|y)$$

> Naive Bayes Characteristics
>
> - **Gaussian Naive Bayes** → assumes continuous features follow a normal distribution
> - **Multinomial Naive Bayes** → for discrete counts (e.g., word frequencies in text)
> - **Bernoulli Naive Bayes** → for binary features (e.g., spam detection)

3.3.2 LDA

Linear Discriminant Analysis (LDA) is a probabilistic classifier that assumes each class follows a **multivariate normal (Gaussian) distribution** with a **shared covariance matrix** but different means. It projects the data into a **lower-dimensional space** to maximize **class separability**. General formula for LDA model,

For class k:

$$P(x|y = k) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu_k)^T \Sigma^{-1}(x - \mu_k)\right)$$

Decision rule:

$$\hat{y} = \arg\max_k \ \delta_k(x)$$

where:

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log P(y = k)$$

i LDA Characteristics

- Assumes **equal covariance matrices** across classes
- Decision boundaries are **linear**
- Works well for **normally distributed features**

3.3.3 QDA

Quadratic Discriminant Analysis (QDA) is an extension of LDA that allows each class to have its **own covariance matrix**. This flexibility enables **non-linear decision boundaries**. General formula for QDA model,

For class k:

$$P(x|y = k) = \frac{1}{(2\pi)^{d/2} |\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k)\right)$$

Decision function:

$$\delta_k(x) = -\frac{1}{2} \log |\Sigma_k| - \frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k) + \log P(y = k)$$

i QDA Characteristics

- Allows **different covariance matrices** → more flexible
- Decision boundaries are **quadratic (nonlinear)**
- Requires **more data** than LDA to estimate covariance matrices

3.3.4 Logistic Regression

Logistic Regression is a probabilistic linear model used for **binary classification**. It estimates the probability that an input belongs to a particular class using the **logistic (sigmoid) function**. General formula for Logistic Regression model:

Let x be the input vector and β the coefficient vector:

$$P(y=1|x) = \frac{1}{1+e^{-(\beta_0+\beta^T x)}}$$

Decision rule:

$$\hat{y} = \begin{cases} 1, & \text{if } P(y=1|x) \geq 0.5 \\ 0, & \text{otherwise} \end{cases}$$

The model is trained by **maximizing the likelihood function** (or equivalently, minimizing the negative log-likelihood):

$$L(\beta) = \sum_{i=1}^{n} [y_i \log P(y_i|x_i) + (1-y_i)\log(1-P(y_i|x_i))]$$

Logistic Regression Characteristics

- Produces **linear decision boundaries**
- Interpretable model coefficients
- Can be extended to **multiclass classification** using *Softmax Regression*

3.3.5 Comparison Probabilistics

Comparison of Probabilistic Classification Models

The following table (Table 3.3) compares four popular probabilistic models — **Naive Bayes**, **LDA**, **QDA**, and **Logistic Regression** — based on their underlying assumptions, mathematical structure, and flexibility.

Table 3.3: Comparison of Probabilistic Classification Models

Model	Assumptions	Decision Boundary	Covariance	Handles Continuous?	N
Naive Bayes	Feature independence	Linear (in log-space)	N/A		
LDA	Gaussian, shared covariance	Linear	Shared (Σ)		
QDA	Gaussian, class-specific covariance	Quadratic	Separate (Σ)		
Logistic Regression	Linear log-odds	Linear	Implicit		

3.4 Kernel Methods

Kernel methods are a family of machine learning algorithms that rely on measuring **similarity between data points** rather than working directly in the original feature space. They are especially useful for **non-linear classification**, where data are not

linearly separable in their original form. By using a **kernel function**, these methods implicitly project data into a **higher-dimensional feature space**, allowing linear separation in that transformed space — without explicitly performing the transformation.

Watch here: Kernel Methods

3.4.1 SVM

Support Vector Machine (SVM) is a powerful supervised learning algorithm that seeks to find the **optimal hyperplane** that separates classes with the **maximum margin**. It can handle both **linear** and **non-linear** classification problems. For a binary classification problem, given data points (x_i, y_i) where $y_i \in \{-1, +1\}$:

The objective is to find the hyperplane:

$$w^T x + b = 0$$

such that the margin between the two classes is maximized:

$$\min_{w,b} \frac{1}{2}\|w\|^2$$

subject to:

$$y_i(w^T x_i + b) \geq 1, \quad \forall i$$

In the **nonlinear case**, SVM uses a **kernel function** $K(x_i, x_j)$ to implicitly map data into a higher-dimensional space:

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

Important Notes for Support Vector Machine

Table 3.4: Common Kernel Functions in SVM

Kernel	Formula	Description
Linear	$K(x_i, x_j) = x_i^T x_j$	Simple linear separation
Polynomial	$K(x_i, x_j) = (x_i^T x_j + c)^d$	Captures polynomial relations
RBF (Gaussian)	$K(x_i, x_j) = \exp(-\gamma\|x_i - x_j\|^2)$	Nonlinear, smooth decision boundary
Sigmoid	$K(x_i, x_j) = \tanh(\alpha x_i^T x_j + c)$	Similar to neural networks

Characteristics Kernel Functions in SVM (see, Table 3.4):

- Maximizes **margin** between classes (robust to outliers)
- Works in **high-dimensional** spaces
- Supports **nonlinear separation** using kernels

- Sensitive to kernel choice and hyperparameters (e.g., C, γ)

3.4.2 KNN

K-Nearest Neighbors (KNN) is a **non-parametric**, **instance-based** learning algorithm. It classifies a new data point based on the **majority class** of its k closest training samples, according to a chosen **distance metric**. For a new observation x, compute its distance to all training samples (x_i, y_i):

$$d(x, x_i) = \sqrt{\sum_{j=1}^{n} (x_j - x_{ij})^2}$$

Then, select the **k nearest neighbors** and assign the class by **majority voting**:

$$\hat{y} = \arg\max_c \sum_{i \in N_k(x)} I(y_i = c)$$

where:

- $N_k(x)$ = indices of the k nearest points
- $I(y_i = c)$ = indicator function (1 if true, 0 otherwise)

i Important Notes for K-Nearest Neighbors

Table 3.5: Common Distance and Similarity Metrics

Metric	Formula	Typical Use
Euclidean Distance	$d(x,y) = \sqrt{\sum_i (x_i - y_i)^2}$	Continuous / numerical features
Manhattan Distance	$d(x,y) = \sum_i \lvert x_i - y_i \rvert$	Sparse or grid-like data
Minkowski Distance	$d(x,y) = (\sum_i \lvert x_i - y_i \rvert^p)^{1/p}$	Generalized form (includes Euclidean & Manhattan)
Cosine Similarity	$\text{sim}(x,y) = \frac{x \cdot y}{\lVert x \rVert \lVert y \rVert}$	Text data or directional similarity
Hamming Distance	$d(x,y) = \sum_i [x_i \neq y_i]$	Binary or categorical data
Jaccard Similarity	$J(A,B) = \frac{\lvert A \cap B \rvert}{\lvert A \cup B \rvert}$	Set-based or binary features

Characteristics in Distance and Similarity Metrics (see, Table 3.5)

- **Lazy learning**: no explicit training phase
- Sensitive to **feature scaling** and **noise**
- Works best for **small to medium** datasets
- Decision boundaries can be **nonlinear and flexible**

3.4.3 Comparison Kernels

⚠ Comparison of Kernel Methods

The following Table 3.6 presents a comparison of two widely used non-probabilistic classification models: Support Vector Machine (SVM) and K-Nearest Neighbors (KNN). It highlights their key characteristics, including model type, parametric nature, ability to handle nonlinearity, computational cost, interpretability, and important hyperparameters.

Table 3.6: Comparison of Kernel Methods

Model	Type	Parametric?	Handles Nonlinearity	Training Cost
Support Vector Machine (SVM)	Kernel-based	Yes	Yes (via kernel)	Moderate to High
K-Nearest Neighbors (KNN)	Instance-based	No	Yes (implicitly)	Low (training) / High (pr

3.5 Ensemble Methods

Ensemble methods are machine learning techniques that combine **multiple base models** to produce a **single, stronger predictive model**. The idea is that **aggregating diverse models** reduces **variance, bias, and overfitting**, leading to **better generalization**. Ensemble methods can be categorized into:

- **Bagging**: Reduces variance by training models on **bootstrapped subsets** (e.g., Random Forest)
- **Boosting**: Reduces bias by sequentially training models that **focus on previous errors** (e.g., Gradient Boosting, XGBoost)

Watch here: Ensemble Methods

3.5.1 Random Forest

Random Forest is a **bagging-based ensemble** of Decision Trees. Each tree is trained on a **random subset of data** with **random feature selection**, and the final prediction is obtained by **majority voting** (classification) or **averaging** (regression). General Form for Random Forest:

1. Generate **B bootstrap samples** from the training data.
2. Train a Decision Tree on each sample:
   - At each split, consider a **random subset of features** (m out of p)
3. Combine predictions:

- **Classification**:

$$\hat{y} = \text{mode}\{T_1(x), T_2(x), ..., T_B(x)\}$$

- **Regression**:

$$\hat{y} = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$$

> **i** Random Forest Characteristics
>
> - Reduces **overfitting** compared to single trees
> - Works well with **high-dimensional** and **noisy datasets**
> - Less interpretable than a single tree

3.5.2 Gradient Boosting Machines

GBM is a **boosting-based ensemble** that builds models **sequentially**, where each new model tries to **correct errors** of the previous one. It focuses on **minimizing a differentiable loss function** (e.g., log-loss for classification). General Form for Gradient Boosting Machines:

1. Initialize the model with a constant prediction:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$$

2. For $m = 1$ to M (number of trees):

   - Compute the **residuals (pseudo-residuals)**:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$$

   - Fit a **regression tree** to the residuals
   - Update the model:

$$F_m(x) = F_{m-1}(x) + \nu T_m(x)$$

     where ν is the **learning rate**

Gradient Boosting Machines Characteristics

- Reduces **bias** by sequential learning
- Can overfit if **too many trees** or **high depth**
- Sensitive to **hyperparameters** (learning rate, tree depth, number of trees)

3.5.3 Extreme Gradient Boosting

XGBoost is an **optimized implementation of Gradient Boosting** that is **faster and more regularized**. It incorporates techniques such as **shrinkage, column subsampling, tree pruning, and parallel computation** for improved performance. Similar to GBM, but adds **regularization** to the objective function:

$$Obj = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

where:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

- T = number of leaves in tree f
- w_j = leaf weight
- γ, λ = regularization parameters

Extreme Gradient Boosting Characteristics

- Fast and scalable
- Handles missing values automatically
- Prevents overfitting with **regularization** and **early stopping**
- Widely used in **Kaggle competitions**

3.5.4 Comparison Ensemble

The following table (see Table 3.7) summarizes key ensemble methods, highlighting their type, main strengths, weaknesses, and the scenarios where they are most effective:

Table 3.7: Comparison of Ensemble Methods for Classification

Method	Type	Strength	Weakness	Best Use Case
Random Forest	Bagging	Reduces variance, robust	Less interpretable	High-dimensional, no
GBM	Boosting	Reduces bias, accurate	Sensitive to overfitting	Medium datasets, co
XGBoost	Boosting (optimized)	Fast, regularized, accurate	Hyperparameter tuning required	Large datasets, comp

Table 3.8: Applications of Classification Models across Domains

Domain	Application	Description / Objective
Healthcare	Disease diagnosis	Classify patients as *disease* vs *no disease*.
	Medical imaging	Identify tumor presence in X-ray or MRI scans.
	Patient readmission prediction	Use hospital data to forecast readmission risk.
Finance	Credit scoring	Predict whether a customer will default on a loan.
	Fraud detection	Identify fraudulent credit card transactions.
	Investment risk classification	Categorize assets as low, medium, or high risk.
Marketing	Customer churn prediction	Determine whether a customer will leave a service.
	Target marketing	Segment customers into high vs. low purchase potential.
	Lead scoring	Prioritize sales prospects based on conversion likelihood.
Text Mining	Sentiment analysis	Classify text as *positive*, *negative*, or *neutral*.
	Spam detection	Detect unwanted or harmful emails/messages.
	Topic classification	Categorize documents by subject matter.
Transportation	Traffic sign recognition	Classify sign types in autonomous vehicles.
	Driver behavior analysis	Detect aggressive or distracted driving patterns.
	Route classification	Predict optimal routes based on historical data.

3.6 Study Case Examples

The following table (see, Table 3.8) summarizes representative applications of classification models across different fields:

3.7 End to End Study Case

This project demonstrates an **end-to-end binary logistic regression** analysis using the built-in `mtcars` dataset in R. We aim to predict whether a car has a **Manual or Automatic** transmission based on:

- `mpg` — Miles per gallon (fuel efficiency)
- `hp` — Horsepower (engine power)
- `wt` — Vehicle weight

3.7.1 Data Preparation

```
data("mtcars")
mtcars$am <- factor(mtcars$am, labels = c("Automatic", "Manual"))
```

```
str(mtcars)
```

```
'data.frame':   32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : Factor w/ 2 levels "Automatic","Manual": 2 2 2 1 1 1 1 1 1 1 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

```
head(mtcars)
```

```
                   mpg cyl disp  hp drat    wt  qsec vs        am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0    Manual    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0    Manual    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1    Manual    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1 Automatic    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0 Automatic    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1 Automatic    3    1
```

3.7.2 Logistic Regression Model

Let say, you build a model using all variables in the mtcars dataset to predict whether a car has a manual or automatic transmission.

```
# Load dataset
data("mtcars")
mtcars$am <- factor(mtcars$am, labels = c("Automatic", "Manual"))

# Full logistic regression model using all predictors
full_model <- glm(am ~ ., data = mtcars, family = binomial)

summary(full_model)
```

```
Call:
glm(formula = am ~ ., family = binomial, data = mtcars)

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.164e+01  1.840e+06       0        1
mpg         -8.809e-01  2.884e+04       0        1
```

```
cyl          2.527e+00  1.236e+05       0        1
disp        -4.155e-01  2.570e+03       0        1
hp           3.437e-01  2.195e+03       0        1
drat         2.320e+01  2.159e+05       0        1
wt           7.436e+00  3.107e+05       0        1
qsec        -7.577e+00  5.510e+04       0        1
vs          -4.701e+01  2.405e+05       0        1
gear         4.286e+01  2.719e+05       0        1
carb        -2.157e+01  1.076e+05       0        1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 4.3230e+01  on 31  degrees of freedom
Residual deviance: 6.4819e-10  on 21  degrees of freedom
AIC: 22

Number of Fisher Scoring iterations: 25
```

Check for Multicollinearity with VIF (Variance Inflation Factor)

```r
# Install if not installed
# install.packages("car")
library(car)

# Calculate VIF values
vif_values <- vif(full_model)

# Sort from highest to lowest
vif_values <- sort(vif_values, decreasing = TRUE)

# Print the results
vif_values
```

```
     disp       cyl        wt        hp      carb       mpg      gear        vs
45.336024 38.112972 28.384837 21.288933 21.096231 18.150446 16.289577 11.102033
     qsec      drat
 9.214178  3.950868
```

Interpretation of VIF values:

- VIF = 1 → No multicollinearity.
- VIF between 1–5 → Moderate correlation (acceptable).
- VIF > 10 → Serious multicollinearity problem.

Now, compare with the simpler model using only three predictors (mpg, hp, and wt):

```r
model2 <- glm(am ~mpg+wt,
              data = mtcars, family = binomial)

summary(model2)
```

```
Call:
glm(formula = am ~ mpg + wt, family = binomial, data = mtcars)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  25.8866    12.1935   2.123   0.0338 *
mpg          -0.3242     0.2395  -1.354   0.1759
wt           -6.4162     2.5466  -2.519   0.0118 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 43.230  on 31  degrees of freedom
Residual deviance: 17.184  on 29  degrees of freedom
AIC: 23.184

Number of Fisher Scoring iterations: 7
```

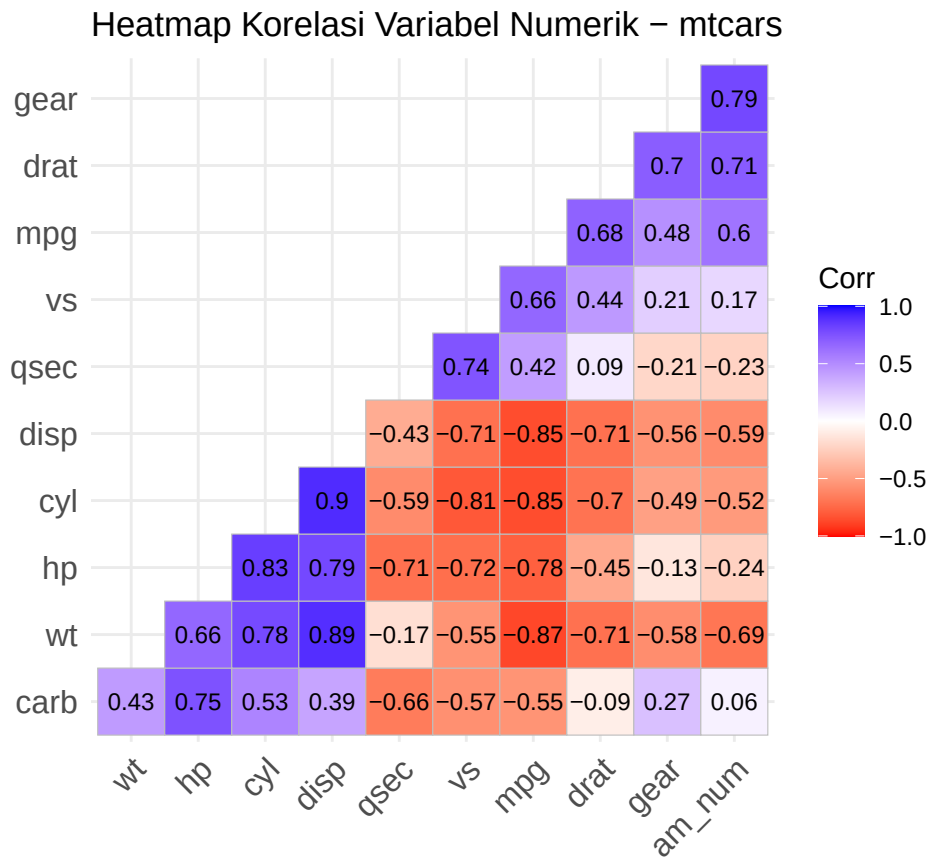
```r
vif(model2)
```

```
     mpg       wt
3.556491 3.556491
```

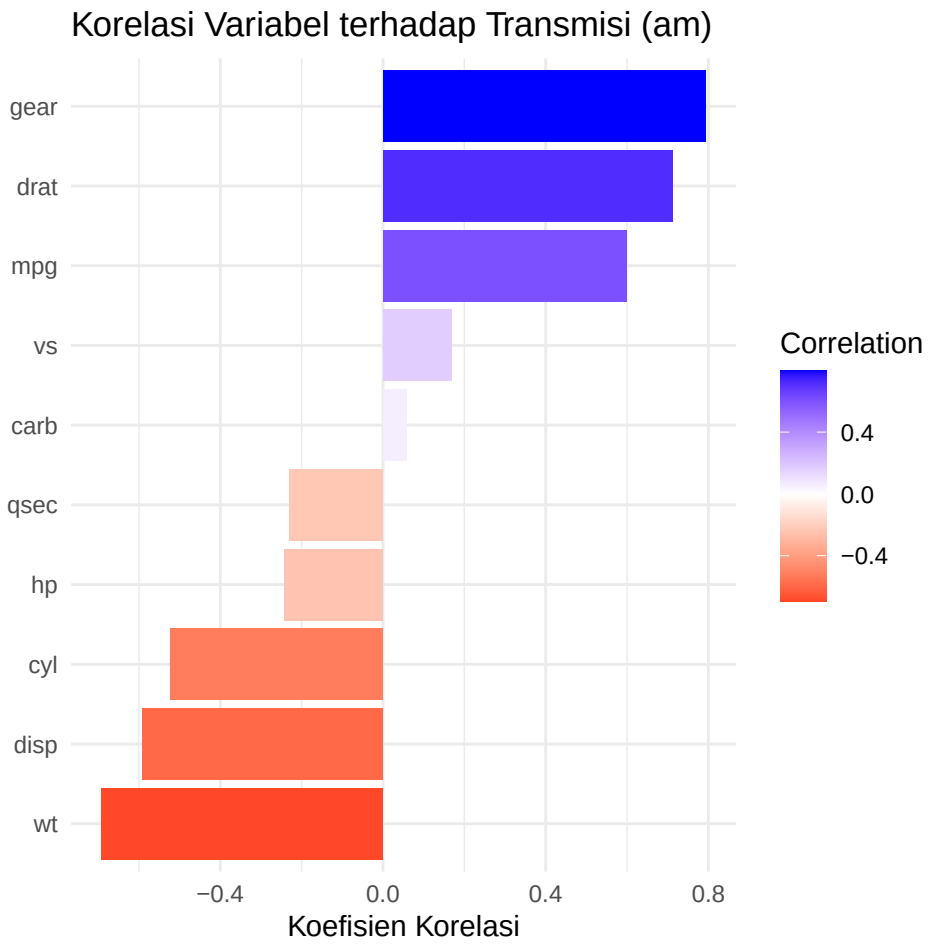
```r
# Dataset
data(mtcars)
mtcars$am <- factor(mtcars$am, labels = c("Automatic", "Manual"))

# Untuk analisis korelasi, ubah am ke numerik (0/1)
mtcars$am_num <- as.numeric(mtcars$am) - 1
```

```r
library(ggcorrplot)

# Hitung matriks korelasi
cor_mat <- cor(mtcars[, sapply(mtcars, is.numeric)])

# Plot
ggcorrplot(cor_mat,
           hc.order = TRUE,
           type = "lower",
           lab = TRUE,
           lab_size = 3,
```

```
            colors = c("red", "white", "blue"),
            title = "Heatmap Korelasi Variabel Numerik - mtcars",
            ggtheme = ggplot2::theme_minimal())
```




```
library(ggplot2)
cor_vals <- cor(mtcars[, sapply(mtcars, is.numeric)])
am_corr <- sort(cor_vals["am_num", -which(colnames(cor_vals) == "am_num")])

# Ubah ke data frame untuk ggplot
df_corr <- data.frame(Variable = names(am_corr),
                      Correlation = am_corr)

ggplot(df_corr, aes(x = reorder(Variable, Correlation), y = Correlation, fill = Correlation)) +
  geom_col() +
  coord_flip() +
  scale_fill_gradient2(low = "red", high = "blue", mid = "white", midpoint = 0) +
  labs(title = "Korelasi Variabel terhadap Transmisi (am)",
       x = NULL, y = "Koefisien Korelasi") +
  theme_minimal()
```



The model estimates the probability that a car is **Manual** using the formula:

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1(\text{mpg}) + \beta_2(\text{hp}) + \beta_3(\text{wt})$$

3.7.3 Prediction and Classification

```r
mtcars$prob <- predict(model2, type = "response")

mtcars$pred_class <- ifelse(mtcars$prob > 0.5, "Manual", "Automatic")

head(mtcars[, c("mpg", "wt", "am", "prob", "pred_class")])
```

```
                  mpg    wt        am       prob pred_class
Mazda RX4        21.0 2.620    Manual 0.90625492     Manual
Mazda RX4 Wag    21.0 2.875    Manual 0.65308276     Manual
Datsun 710       22.8 2.320    Manual 0.97366320     Manual
```

```
Hornet 4 Drive    21.4 3.215 Automatic 0.15728804  Automatic
Hornet Sportabout 18.7 3.440 Automatic 0.09561351  Automatic
Valiant           18.1 3.460 Automatic 0.10149089  Automatic
```

3.7.4 Confusion Matrix

```
conf_mat <- table(Actual = mtcars$am, Predicted = mtcars$pred_class)
conf_mat
```

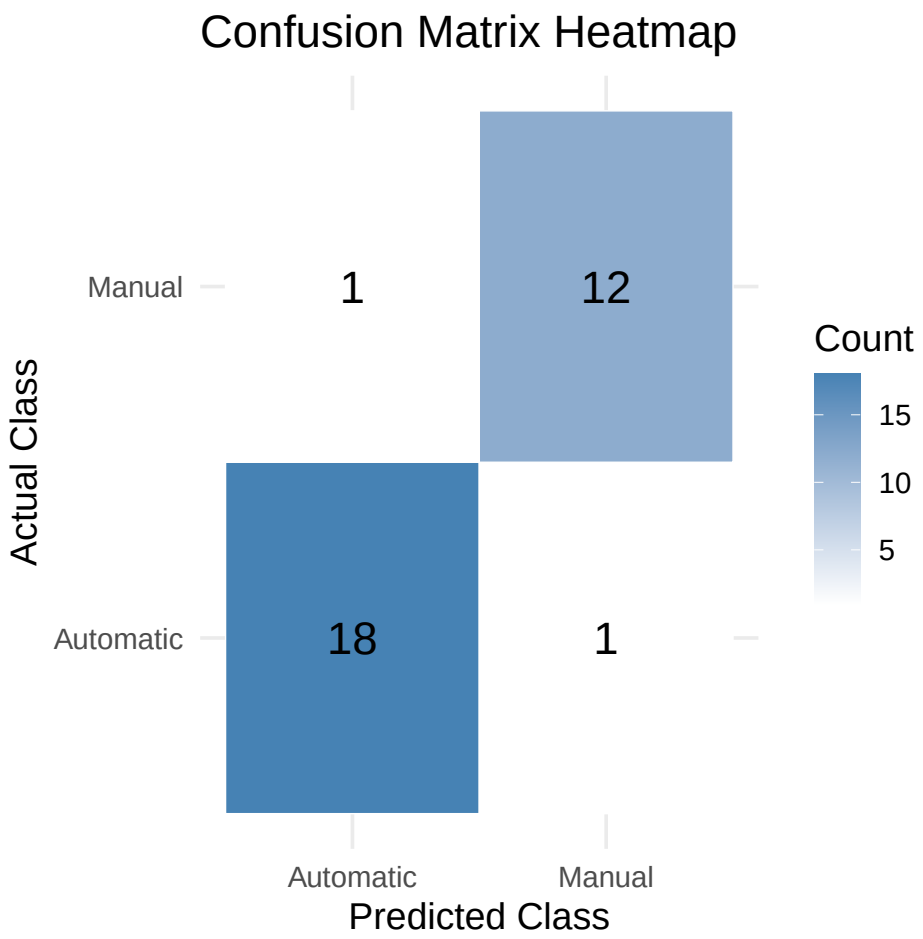
```
           Predicted
Actual      Automatic Manual
  Automatic        18      1
  Manual            1     12
```

Interpretation:

- Diagonal values show correct predictions
- Off-diagonal values show misclassifications

The following plot illustrates how well the model separates the two classes based on predicted probabilities.

```
library(ggplot2)
library(dplyr)

conf_data <- table(Actual = mtcars$am, Predicted = mtcars$pred_class) %>%
  as.data.frame()

ggplot(conf_data, aes(x = Predicted, y = Actual, fill = Freq)) +
  geom_tile(color = "white") +
  geom_text(aes(label = Freq), size = 6, color = "black") +
  scale_fill_gradient(low = "white", high = "steelblue") +
  labs(
    title = "Confusion Matrix Heatmap",
    x = "Predicted Class",
    y = "Actual Class",
    fill = "Count"
  ) +
  theme_minimal(base_size = 14)
```



3.7.5 Evaluation Metrics

```
TP <- conf_mat["Manual", "Manual"]
TN <- conf_mat["Automatic", "Automatic"]
FP <- conf_mat["Automatic", "Manual"]
FN <- conf_mat["Manual", "Automatic"]

Accuracy  <- (TP + TN) / sum(conf_mat)
Precision <- TP / (TP + FP)
Recall    <- TP / (TP + FN)
F1_Score  <- 2 * (Precision * Recall) / (Precision + Recall)

metrics <- data.frame(
  Metric = c("Accuracy", "Precision", "Recall", "F1 Score"),
  Value  = round(c(Accuracy, Precision, Recall, F1_Score), 3)
)
metrics
```

```
     Metric Value
1  Accuracy 0.938
2 Precision 0.923
3    Recall 0.923
4  F1 Score 0.923
```

Metric	Description
Accuracy	Overall correctness of predictions
Precision	How precise the "Manual" predictions are
Recall	Ability to detect all Manual cars
F1 Score	Harmonic mean of Precision and Recall

3.7.6 ROC Curve and AUC

```
library(pROC)
```

```
Type 'citation("pROC")' for a citation.
```

```
Attaching package: 'pROC'
```

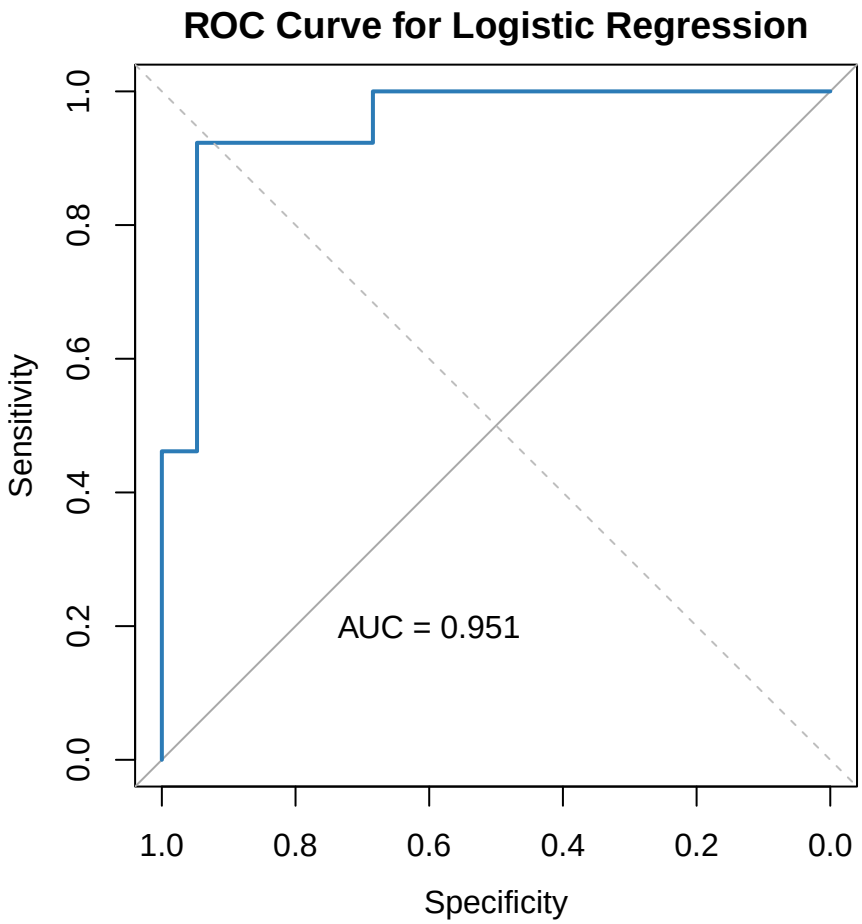
```
The following objects are masked from 'package:stats':

    cov, smooth, var
```

```
roc_obj <- roc(mtcars$am, mtcars$prob)
```

```
Setting levels: control = Automatic, case = Manual
```

```
Setting direction: controls < cases
```

```
auc_value <- auc(roc_obj)

plot(roc_obj, col = "#2C7BB6", lwd = 2, main = "ROC Curve for Logistic Regression")
abline(a = 0, b = 1, lty = 2, col = "gray")
text(0.6, 0.2, paste("AUC =", round(auc_value, 3)), col = "black")
```



Interpretation:

- The ROC curve shows the trade-off between **True Positive Rate** and **False Positive Rate**
- **AUC (Area Under the Curve)** evaluates how well the model distinguishes between classes
  - AUC = 1 → Perfect
  - AUC 0.8 → Excellent
  - AUC = 0.5 → Random guessing

3.7.7 Conclusion

The logistic regression model successfully predicts car transmission type (Manual vs Automatic) with strong performance.

Aspect	Result / Interpretation
Significant variable	`wt` (vehicle weight) has a negative effect on Manual probability
Model accuracy	90%
AUC	> 0.8 (Excellent discriminative power)
Conclusion	The model performs well in classifying car transmissions

References

Chapter 4

Clustering Models

Clustering is one of the fundamental techniques in unsupervised learning, designed to group objects based on their similarities without relying on predefined labels. This method uncovers hidden patterns, natural structures, and meaningful relationships within data across diverse applications—from customer segmentation and anomaly detection to sensor behavior analysis in industrial systems. The mind map provided offers a comprehensive overview of major clustering methodologies, including partition-based algorithms, hierarchical models, density-based approaches, probabilistic frameworks, deep learning–based representations, and hybrid techniques. This visual guide helps readers grasp the broader landscape of clustering algorithms and understand the key distinctions among them, supporting more informed decisions when selecting the most appropriate method for various analytical tasks. For additional detail, refer to Figure 4.1.

4.1 Intro to Clustering

To build an initial understanding of clustering, it is helpful to begin with a clear visual explanation. The short video below provides an accessible introduction to the core idea behind clustering—how data points are grouped based on similarity, why this process is useful, and where it is commonly applied. This foundational overview serves as a starting point before exploring the more detailed concepts and the comprehensive mind map presented in the next sections.

```
Video cannot be displayed in PDF/Word.
 Please view the HTML version or open directly on YouTube:
 https://www.youtube.com/embed/4cxVDUybHrI?si=EMzKvKACt7FzeDSs
```

4.2 Partition-Based

Partition-based clustering is a fundamental technique in unsupervised learning that divides a dataset into a predefined number of non-overlapping clusters. Each data

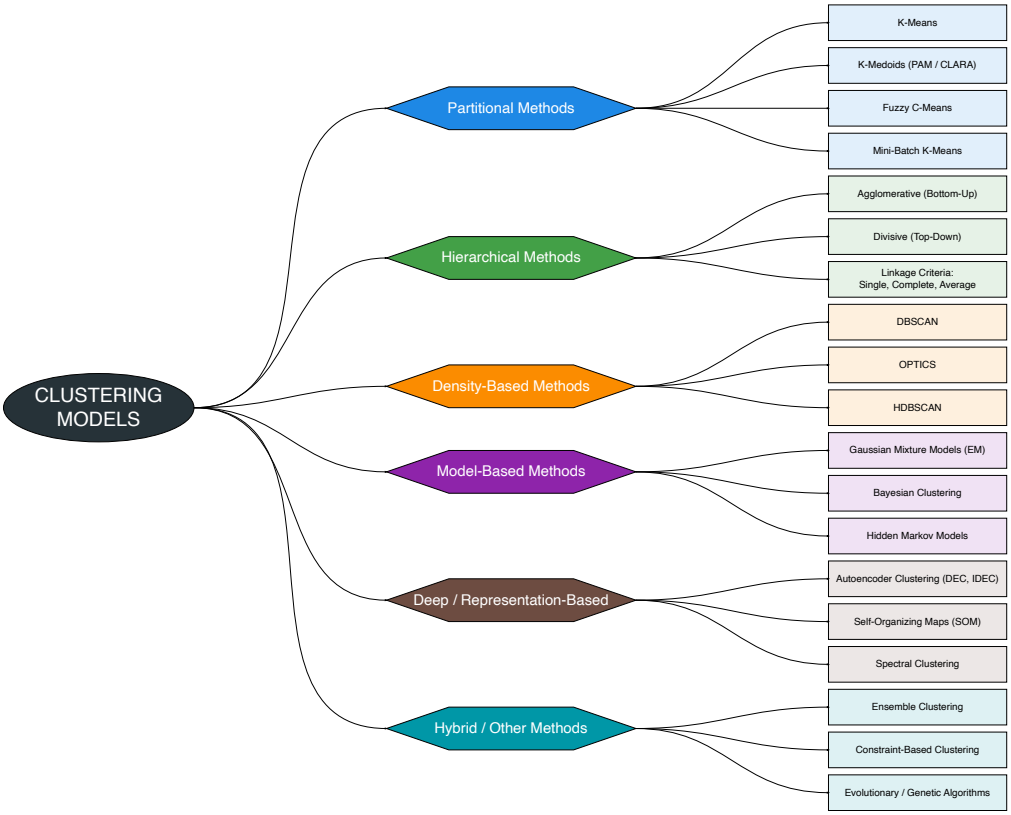


Figure 4.1: Comprehensive Clustering Models Mind Map

point is assigned to exactly one cluster based on a similarity or distance measure, most commonly the Euclidean distance. The objective is to create clusters that are internally cohesive and externally well separated.

In this approach, the number of clusters k must be specified in advance. The algorithm then iteratively assigns data points to clusters and updates the cluster centers (or representatives) until a stable solution is reached. Owing to its simplicity, computational efficiency, and strong performance on well-structured datasets, partition-based clustering is widely used in pattern recognition, customer segmentation, image analysis, and industrial analytics.

Representative algorithms include k-Means, which minimizes the sum of squared distances to cluster centroids; k-Medoids, which uses actual data points as cluster centers to enhance robustness against outliers; k-Means++, which improves the initialization process; and CLARA, which adapts k-Medoids for large datasets through sampling. Partition-based methods perform best when clusters are approximately spherical, similar in size, relatively free of extreme outliers, and when the value of k is known or can be estimated reliably. Although these methods can be sensitive to initialization and struggle with irregularly shaped clusters, they remain among the most widely used clustering techniques in contemporary data analysis.

```
Video cannot be displayed in PDF/Word.
 Please view the HTML version or open directly on YouTube:
 https://www.youtube.com/embed/PJGSEttUzx8?si=c-tM7VfsdawV5UfR
```

4.3 Hierarchical

Hierarchical clustering is an unsupervised learning method that organizes data into a hierarchy of nested clusters, typically visualized through a dendrogram. Unlike partition-based methods, it does not require predefining the number of clusters; the structure emerges naturally from the data. It is chosen for its interpretability and ability to reveal multi-level relationships within a dataset. Analysts can observe cluster formation at various similarity thresholds, making it ideal for exploratory analysis and pattern discovery.

Researchers and practitioners in fields such as data science, bioinformatics, text mining, and anomaly detection frequently employ hierarchical clustering to analyze structured and semi-structured data. It is most effective for moderate-sized datasets where one aims to explore hierarchical structure, evaluate similarity relationships, or derive cluster insights without repeated algorithm initialization. The method is widely available in statistical and machine-learning libraries, such as R (hclust, dendextend) and Python (scipy.cluster.hierarchy, sklearn).

Hierarchical clustering operates through two strategies:

- gglomerative (bottom-up): each point starts as its own cluster, and the most similar clusters are merged step-by-step based on linkage criteria (single, complete, average, Ward).
- Divisive (top-down): all points begin in one cluster, which is recursively divided into smaller groups.

Clusters are finally obtained by cutting the dendrogram at the desired distance or similarity threshold.

```
Video cannot be displayed in PDF/Word.
 Please view the HTML version or open directly on YouTube:
 https://www.youtube.com/embed/8QCBl-xdeZI?si=-rY4EuXBozZk6xh4
```

4.4 Density-Based

Density-Based Clustering is a clustering approach that groups data points based on the density of observations within a region, where areas with high concentrations of points form clusters and sparse regions are treated as noise or outliers. This method matters because it can identify clusters of arbitrary shapes—irregular, elongated, or non-convex—that are difficult for partition-based methods like K-Means to detect. It is also advantageous because it does not require predefining the number of clusters, making it particularly valuable for exploratory analysis when the underlying structure of the data is unknown.

This technique is well suited for analysts and researchers working with spatial data, sensor data, operational measurements, or datasets with uneven point distribution. Density-based methods are especially useful when the dataset contains many outliers, when cluster boundaries are not well defined, or when the analysis involves anomaly detection. In practical applications, it is widely used in geospatial analysis to identify hotspots, in industrial monitoring to detect abnormal sensor patterns, in cybersecurity to flag suspicious activities, and in finance for identifying unusual transactions.

Operationally, Density-Based Clustering functions by defining a search radius ( ) and a minimum number of points (MinPts) required to form a dense region. Points with enough neighbors are classified as core points, points that are within the neighborhood of a core point are considered border points, and all remaining points are labeled as noise. Clusters emerge through chains of connected dense regions, allowing complex and natural patterns in the data to be captured without forcing rigid geometric shapes.

While the method has limitations—most notably sensitivity to parameter selection and reduced performance in high-dimensional spaces—its ability to model real-world cluster structures and detect outliers effectively makes Density-Based Clustering, especially DBSCAN, one of the most powerful and widely used techniques in modern data analysis.

```
Video cannot be displayed in PDF/Word.
 Please view the HTML version or open directly on YouTube:
 https://www.youtube.com/embed/RDZUdRSDOok?si=tC6bUnUc-lsQUNT-
```

4.5 Probabilistic

Probabilistic clustering is a method of unsupervised learning where clusters are defined by probability distributions rather than fixed boundaries. It addresses what the structure of the data looks like by estimating how likely each data point belongs to each cluster, instead of forcing a hard assignment. This approach is used when datasets exhibit

overlapping groups, hidden latent structure, or uncertainty in boundaries—conditions where deterministic clustering fails to capture the true relationships. It is applied where soft assignments are valuable, such as text mining, image classification, medical diagnostics, and anomaly detection. The technique relies on why probabilistic modeling is effective: it naturally handles ambiguity, provides richer information through membership probabilities, and supports principled model comparison using likelihood-based criteria. Methods such as Gaussian Mixture Models (GMM) and Expectation–Maximization (EM) clustering represent who performs the clustering—the probabilistic components that model each group as a distribution. Finally, the process describes how clustering is performed: by estimating the parameters of each distribution, computing membership probabilities for every point, and iteratively updating these values until the likelihood converges. This distribution-driven, uncertainty-aware perspective makes probabilistic clustering a powerful tool for understanding complex, real-world data.

```
Video cannot be displayed in PDF/Word.
 Please view the HTML version or open directly on YouTube:
 https://www.youtube.com/embed/C7jhwN6H9LU?si=wh_F4VnB4Ns1kaxB
```

4.6 Deep Learning-Based

Deep learning–based clustering is an advanced approach that uses deep neural networks to learn meaningful representations of high-dimensional or unstructured data before performing clustering. Unlike traditional methods that rely on raw features, this technique transforms data into a latent space where important patterns are emphasized and noise is reduced. It is used because many real-world datasets—such as images, text, audio, sensor streams, and industrial signals—contain complex, non-linear structures that classical algorithms like k-Means or DBSCAN cannot model effectively. This approach benefits researchers and practitioners in fields such as computer vision, natural language processing, bioinformatics, and intelligent analytics, where discovering hidden structure is essential. It is applied in tasks like image grouping, document organization, anomaly detection, user-behavior modeling, and representation learning. Deep learning–based clustering is most suitable when the dataset is large, high-dimensional, or difficult to separate using traditional techniques, especially when feature engineering becomes challenging. It works by training a neural network—typically an autoencoder or embedding model—to map data into a compact latent space, then clustering those embeddings using algorithms such as k-Means or Gaussian Mixture Models. More advanced methods, such as Deep Embedded Clustering (DEC), integrate representation learning and clustering into a single end-to-end optimization process, producing more coherent and well-separated clusters.

```
Video cannot be displayed in PDF/Word.
 Please view the HTML version or open directly on YouTube:
 https://www.youtube.com/embed/0m5GNDo-CFM?si=XmTfZGc3VJRS5B4X
```

4.7 Hybrid

Hybrid clustering combines two or more clustering approaches to leverage their complementary strengths and overcome the limitations of individual methods. It is used when datasets are complex, noisy, high-dimensional, or contain clusters with irregular shapes that cannot be captured effectively by a single technique. Hybrid clustering is important because real-world data often exhibits mixed structures—some regions may form dense clusters, others may follow hierarchical patterns, while some require partitioning for refinement—making hybrid solutions more flexible and accurate. It is applied in domains such as bioinformatics, customer segmentation, image processing, anomaly detection, and large-scale industrial analytics, where robust and adaptive clustering is essential. Hybrid clustering works by combining methods such as density-based algorithms (e.g., DBSCAN) to detect core structures, hierarchical clustering to capture multi-level relationships, and partition-based methods like k-Means to refine cluster boundaries. More advanced hybrid models may also integrate spectral clustering, probabilistic approaches, or deep-learning-based embeddings to improve performance. This approach is most suitable when the data contains both dense and sparse regions, varies in cluster size or shape, or when a single method consistently fails to capture the underlying structure. By integrating complementary strategies, hybrid clustering produces more stable, interpretable, and high-quality clustering results across diverse data scenarios.

```
Video cannot be displayed in PDF/Word.
 Please view the HTML version or open directly on YouTube:
 https://www.youtube.com/embed/C7jhwN6H9LU?si=wh_F4VnB4Ns1kaxB
```

4.8 Applied of Clustering

4.8.1 Partition Based

Load Libraries & Dataset

```r
library(factoextra)
library(cluster)
library(e1071)      # Fuzzy C-Means
library(ClusterR)   # Mini-Batch KMeans
library(plotly)
library(dplyr)
```

Load dataset

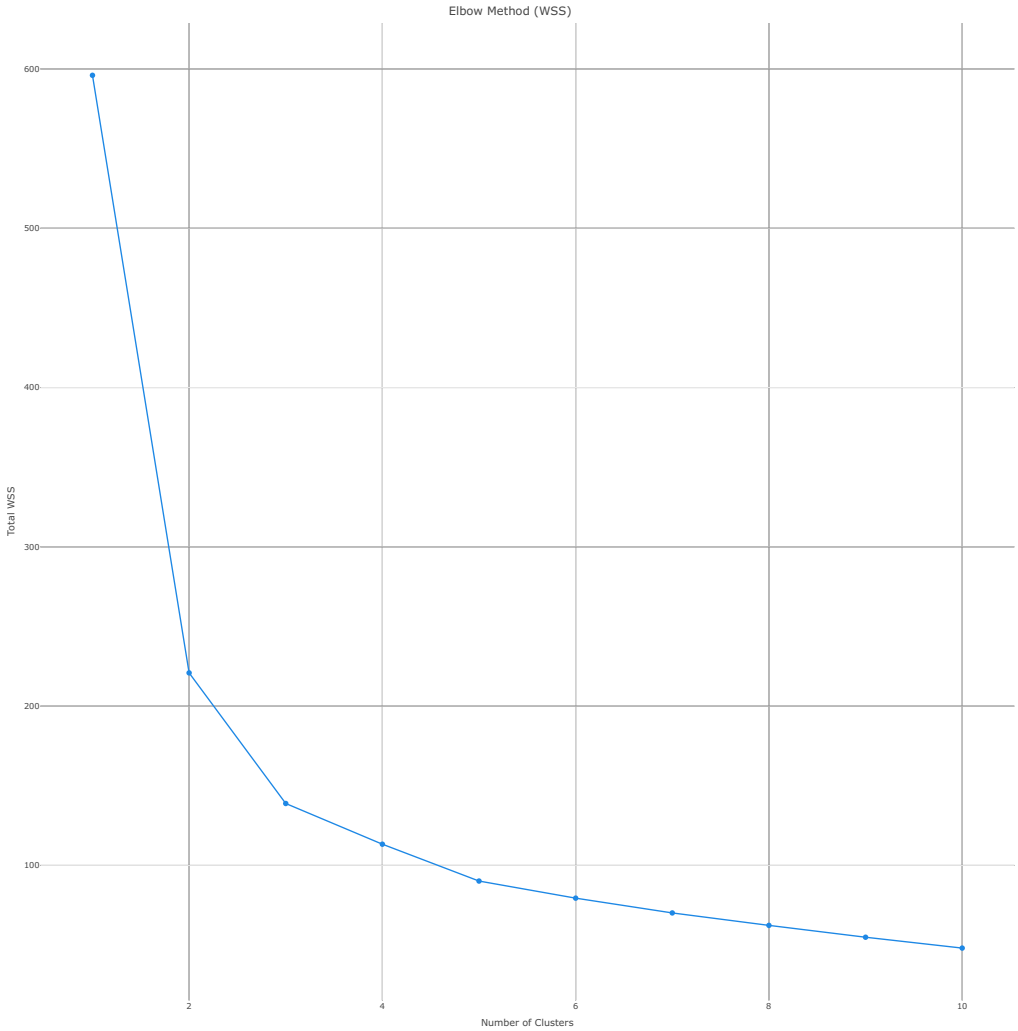
```r
data(iris)
iris_data <- iris[,1:4]
iris_labels <- iris$Species
```

```
# Scale data
iris_data_scale <- scale(iris_data)
set.seed(123)
```

Determine Optimal Number of Clusters

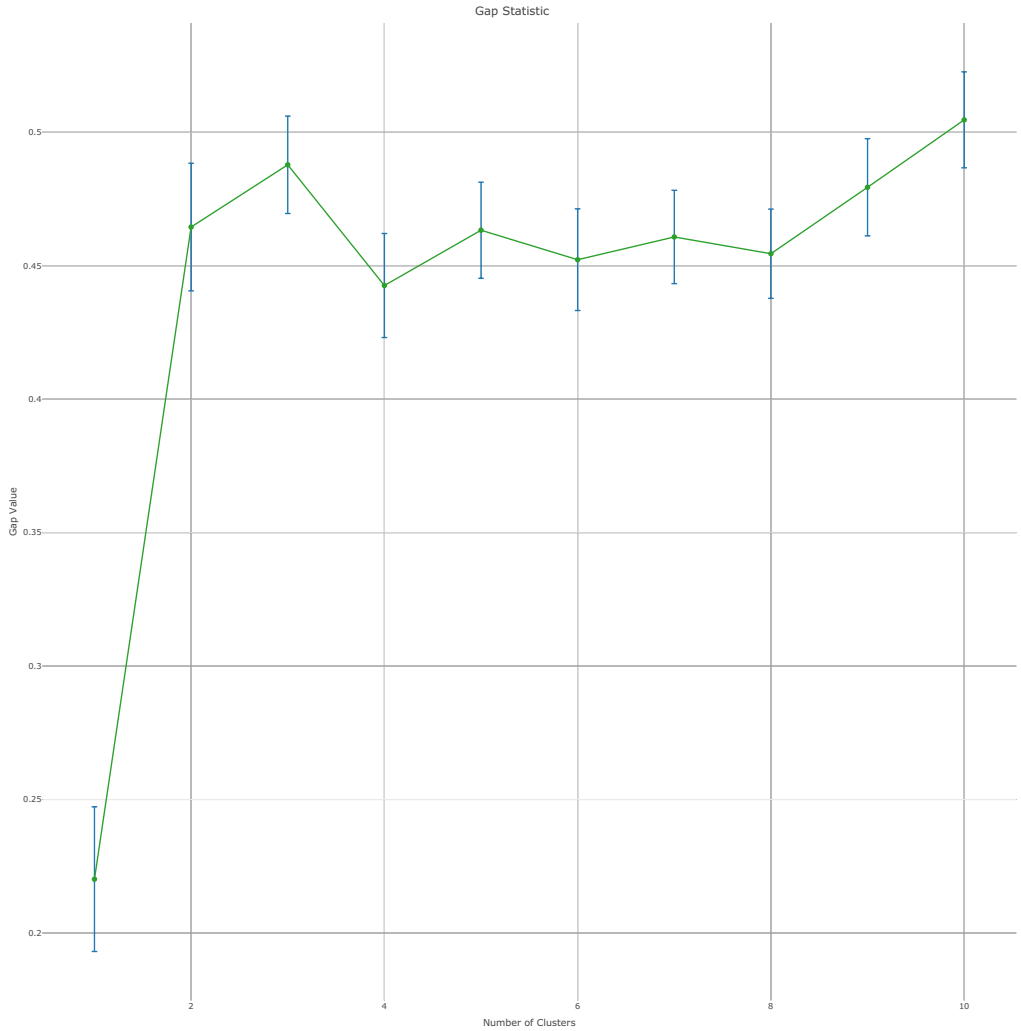
Elbow Method (WSS):

```
wss <- function(k) kmeans(iris_data_scale, k, nstart=25)$tot.withinss
k.values <- 1:10
wss_values <- sapply(k.values, wss)

fig_elbow <- plot_ly(
  x=k.values, y=wss_values, type='scatter', mode='lines+markers',
  marker=list(size=8, color='#1E88E5'), line=list(color='#1E88E5')
) %>% layout(title="Elbow Method (WSS)", xaxis=list(title="Number of Clusters"), yaxis=list(t

fig_elbow
```



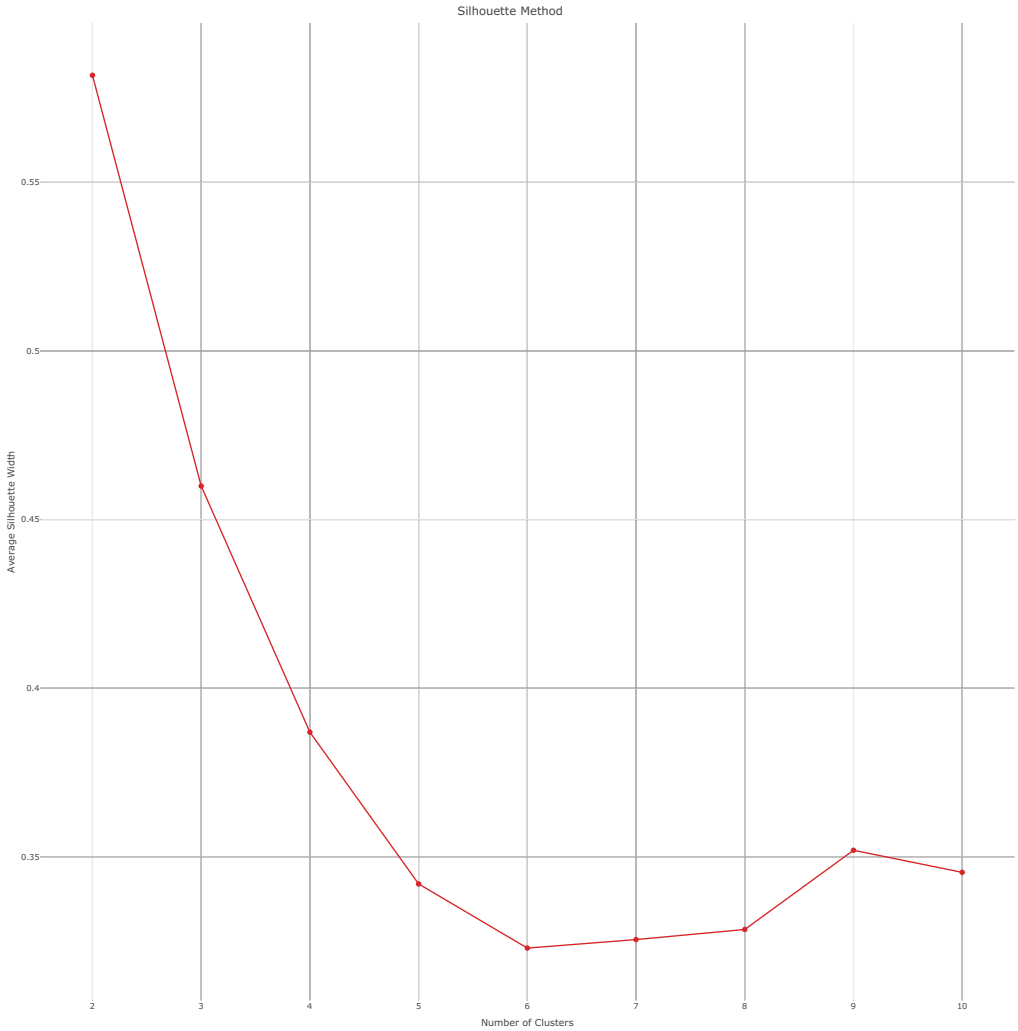
Gap Statistic:

```
gap_stat <- clusGap(iris_data_scale, FUN=kmeans, nstart=25, K.max=10, B=50)
gap_df <- data.frame(
  k=1:10,
  gap=gap_stat$Tab[, "gap"],
  SE=gap_stat$Tab[, "SE.sim"]
)

fig_gap <- plot_ly(
  gap_df, x=~k, y=~gap, type='scatter', mode='lines+markers',
  error_y=list(type="data", array=gap_df$SE),
  marker=list(size=8, color='#2CA02C'), line=list(color='#2CA02C')
) %>% layout(title="Gap Statistic", xaxis=list(title="Number of Clusters"), yaxis=list(tit
fig_gap
```



Silhouette Method:

```
sil_width <- sapply(2:10, function(k){
  km <- kmeans(iris_data_scale, centers=k, nstart=25)
  ss <- silhouette(km$cluster, dist(iris_data_scale))
  mean(ss[,3])
})

fig_sil <- plot_ly(
  x=2:10, y=sil_width, type='scatter', mode='lines+markers',
  marker=list(size=8, color='#D62728'), line=list(color='#D62728')
) %>% layout(title="Silhouette Method", xaxis=list(title="Number of Clusters"), yaxis=list(ti
fig_sil
```



4.8.1.1 Apply Clustering Algorithms (k=3)

```
# ----- 1. K-Means -----
km <- kmeans(iris_data_scale, centers=3, nstart=25)
iris$KMeans <- as.factor(km$cluster)
```

```
# ----- 2. K-Medoids (PAM) -----
pam_res <- pam(iris_data_scale, k=3)
iris$KMedoids <- as.factor(pam_res$clustering)
```

```
# ----- 3. Fuzzy C-Means -----
fcm <- cmeans(iris_data_scale, centers=3, m=2)
iris$FuzzyCMeans <- as.factor(apply(fcm$membership,1,which.max))
```

```r
library(ClusterR)
set.seed(123)

# Data preparation
iris_data <- iris[, 1:4]
iris_data_scale <- scale(iris_data)
iris_mat <- as.matrix(iris_data_scale)

# Train Mini-Batch K-means
mb <- MiniBatchKmeans(
  data = iris_mat,
  clusters = 3,
  batch_size = 20,
  num_init = 5,
  max_iters = 30
)

# New recommended way:
# Use 'predict' S3 method for MiniBatchKmeans
pred <- predict(
  object = mb,
  newdata = iris_mat,
  fuzzy = FALSE    # ensures hard clusters, not probabilities
)

# Add cluster labels
iris$MiniBatch <- as.factor(pred)

# Check cluster distribution
table(iris$MiniBatch)
```

```
 1  2  3
50 49 51
```

2D PCA Visualization (Plotly)

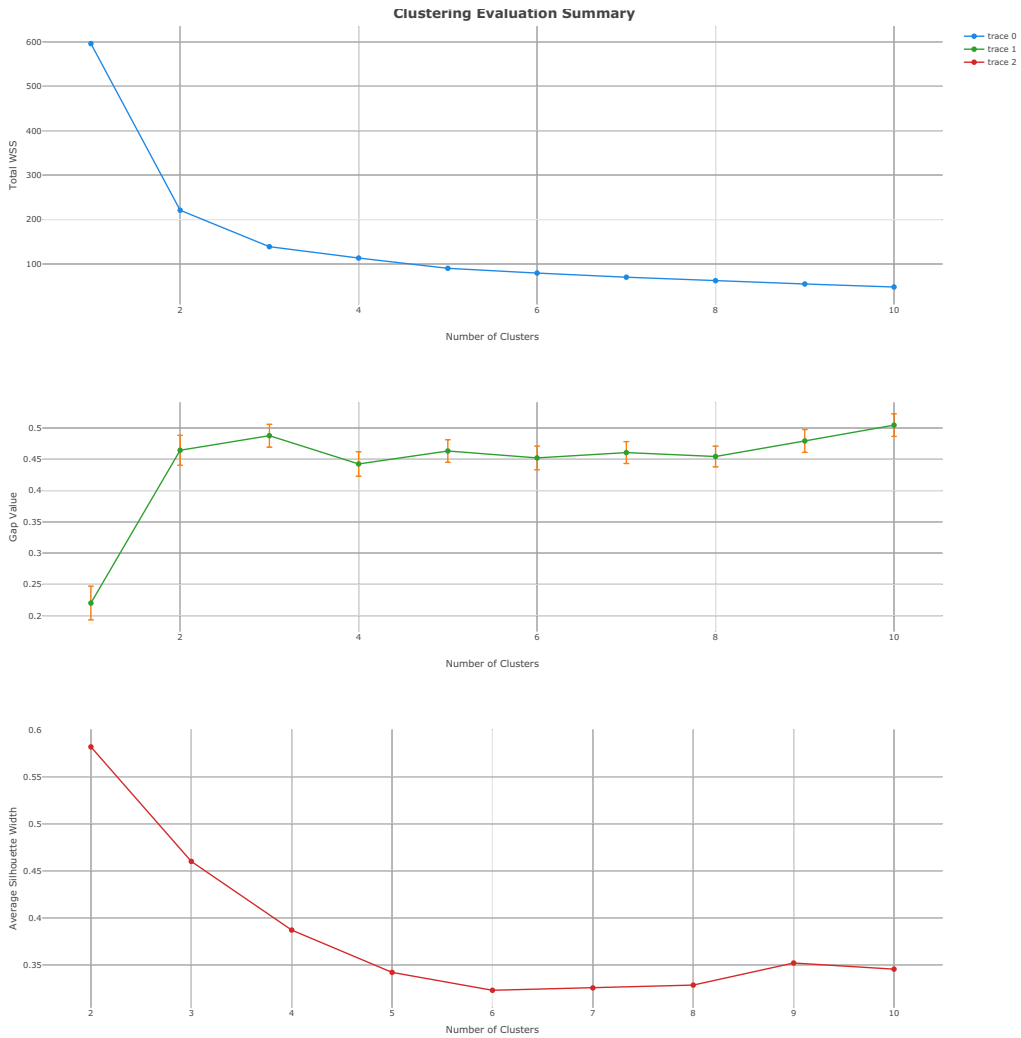
```r
library(dplyr)
library(plotly)

# PCA
pca <- prcomp(iris_data_scale)

pca_df <- data.frame(
  PC1 = pca$x[, 1],
  PC2 = pca$x[, 2],
  KMeans = iris$KMeans,
  KMedoids = iris$KMedoids,
```

```
  FuzzyCMeans = iris$FuzzyCMeans,
  MiniBatch = iris$MiniBatch,
  Species = iris_labels
)

# Plot function (clean + safe)
plot_cluster <- function(df, cluster_col, title) {
  plot_ly(
    df,
    x = ~PC1,
    y = ~PC2,
    color = as.formula(paste0("~", cluster_col)),
    symbol = ~Species,
    symbols = c("circle", "diamond", "square"),
    type = "scatter",
    mode = "markers",
    marker = list(size = 8, opacity = 0.85)
  ) %>%
    layout(title = title)
}

# Figures
fig_km   <- plot_cluster(pca_df, "KMeans",       "K-Means")
fig_kmed <- plot_cluster(pca_df, "KMedoids",     "K-Medoids")
fig_fcm  <- plot_cluster(pca_df, "FuzzyCMeans",  "Fuzzy C-Means")
fig_mbkm <- plot_cluster(pca_df, "MiniBatch",    "Mini-Batch K-Means")
```

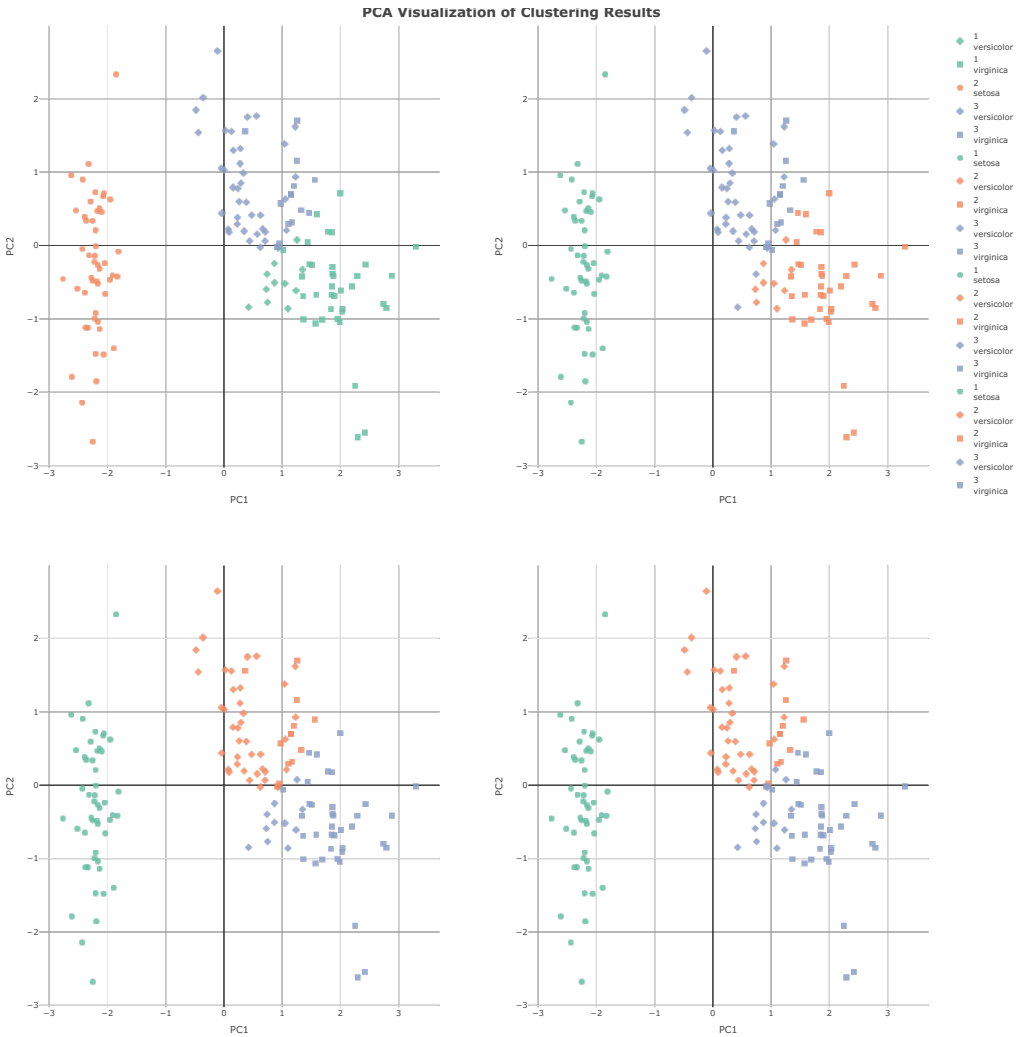
Combined Plotly Dashboard

```
library(plotly)

# ----- Dashboard 1: Evaluation Methods -----
dashboard_eval <- subplot(
  fig_elbow, fig_gap, fig_sil,
  nrows = 3,
  margin = 0.05,
  shareX = FALSE, shareY = FALSE,
  titleX = TRUE, titleY = TRUE
) %>% layout(
  title = list(
    text = "<b>Clustering Evaluation Summary</b>",
    font = list(size = 20)
  )
)

# ----- Dashboard 2: PCA Cluster Visualization -----
dashboard_clusters <- subplot(
```

```
  fig_km, fig_kmed,
  fig_fcm, fig_mbkm,
  nrows = 2,
  margin = 0.05,
  shareX = FALSE, shareY = FALSE,
  titleX = TRUE, titleY = TRUE
) %>% layout(
  title = list(
    text = "<b>PCA Visualization of Clustering Results</b>",
    font = list(size = 20)
  )
)

dashboard_eval
```



```
dashboard_clusters
```



Notes:

- Row 1: Cluster selection methods (Elbow / Gap / Silhouette)
- Rows 2-3: Four clustering algorithms with 2D PCA

Cluster Evaluation

```
cat("K-Means vs Species:\n"); table(iris$KMeans, iris_labels)
```

```
K-Means vs Species:
```

```
   iris_labels
    setosa versicolor virginica
1       0         11        36
2      50          0         0
3       0         39        14
```

```
cat("\nK-Medoids vs Species:\n"); table(iris$KMedoids, iris_labels)
```

```
K-Medoids vs Species:
```

```
   iris_labels
    setosa versicolor virginica
1      50          0         0
2       0          9        36
3       0         41        14
```

```
cat("\nFuzzy C-Means vs Species:\n"); table(iris$FuzzyCMeans, iris_labels)
```

```
Fuzzy C-Means vs Species:
```

```
   iris_labels
    setosa versicolor virginica
1      50          0         0
2       0         39        13
3       0         11        37
```

```
cat("\nMini-Batch K-Means vs Species:\n"); table(iris$MiniBatch, iris_labels)
```

```
Mini-Batch K-Means vs Species:
```

```
   iris_labels
    setosa versicolor virginica
1      50          0         0
2       0         37        12
3       0         13        38
```

Analysis:

- Setosa: typically forms a clear cluster in all algorithms
- Versicolor & Virginica: overlapping → Fuzzy C-Means shows memberships
- Mini-Batch K-Means: similar to regular K-Means but faster
- Gap / Silhouette: help to validate if k=3 is optimal

Table 4.1: Clustering Methods: Categories, Algorithms, Key Formulas, and Implementation Packages

Category	Algorithms	Key Formula	R Packages
Partition-Based	k-Means, k-Medoids	$\min \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$	stats::kmeans; cluster::pam
Partition-Based	k-Means++	Probabilistic center initialization: $P(x) = \frac{D(x)^2}{\sum D(x)^2}$	ClusterR::KMeans_rcpp
Partition-Based	CLARA (Clustering Large Applications)	Sampling + PAM applied to large datasets	cluster::clara
Hierarchical	Agglomerative (Bottom-Up)	Distance linkage: single, complete, average	stats::hclust; dendextend
Hierarchical	Divisive (Top-Down)	Recursive splitting by maximal dissimilarity	stats::hclust
Density-Based	DBSCAN	Core rule: $\|N_\varepsilon(x)\| \geq MinPts$	dbscan::dbscan
Density-Based	HDBSCAN	Density hierarchy via Minimum Spanning Tree	dbscan::hdbscan
Probabilistic	Gaussian Mixture Models (GMM)	$p(x) = \sum_k \pi_k \mathcal{N}(x \mid \mu_k, \Sigma_k)$	mclust
Probabilistic	EM Clustering (Expectation–Maximization)	Iterative E-step & M-step until convergence	EMCluster
Deep Learning-Based	Autoencoder-Based Clustering	Latent code: $z = f_\theta(x)$	keras; torch
Deep Learning-Based	Deep Embedded Clustering (DEC)	Joint reconstruction + clustering loss	keras; torch
Hybrid	Spectral Clustering	Graph Laplacian: $L = D - W$	kernlab::specc
Hybrid	Affinity Propagation	Similarity-based message passing	apcluster

4.9 Summary Clustering Models

The table below (Table 4.1) provides a comprehensive summary of the major categories of clustering methods, the algorithms contained within each category, their core formulas, and commonly used packages for implementation in both R and Python. This presentation is intended to help readers understand the conceptual distinctions among the approaches while also identifying practical tools available for real-world applications.

Chapter 5

Time Series Models

This section summarizes the fundamental models used in time series forecasting, from classical statistical methods to modern deep learning architectures.

```
Video cannot be displayed in PDF/Word.
 Please view the HTML version or open directly on YouTube:
 https://www.youtube.com/embed/GE3JOFwTWVM?si=x-PotV_oXeLTEMQ6
```

5.1 Classical Statistical Models

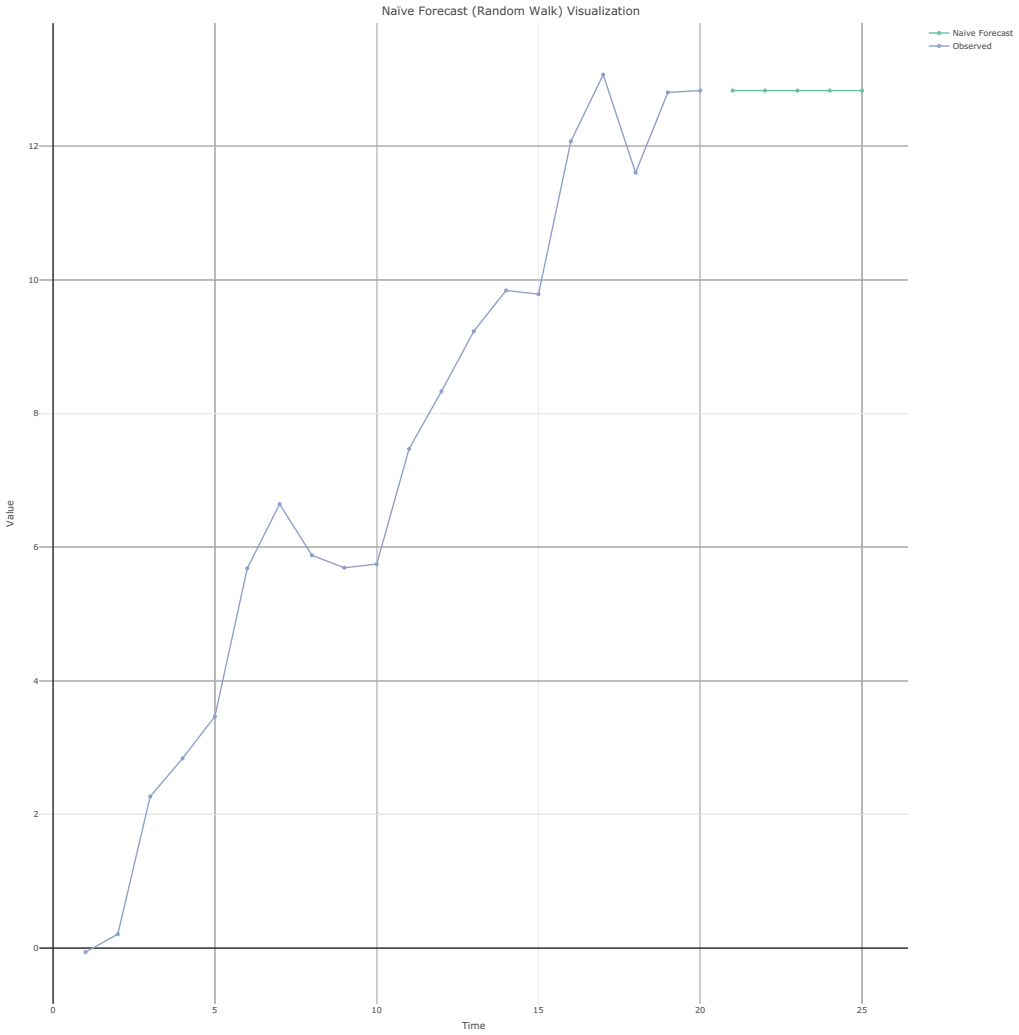
5.1.1 Naïve Forecast

The Naïve Forecast (Random Walk) is the simplest baseline model in time series forecasting. It assumes that the best prediction for the next period is simply the most recent observed value.

Formula:

$$\hat{y}_{t+1} = y_t$$

In other words, the forecast "walks" forward by repeating the last value, making it equivalent to a random walk with no drift. Let consider, the naïve forecast for the next 5 periods:



Interpretation

The **Observed** line shows the true historical values of the time series. The **Naïve Forecast** appears as a **flat horizontal line**. This happens because the naïve method simply repeats the **last observed value**. Mathematically, If the final observed value is y_T, then all future predictions are:

$$\hat{y}_{T+1} = \hat{y}_{T+2} = \cdots = \hat{y}_{T+h} = y_T$$

This means that the forecast does **not** change over the horizon, it stays constant at the last known value. The naïve method is widely used as a **benchmark model** in forecasting because any more advanced model should perform at least better than this simple baseline.

5.1.2 Simple Moving Average

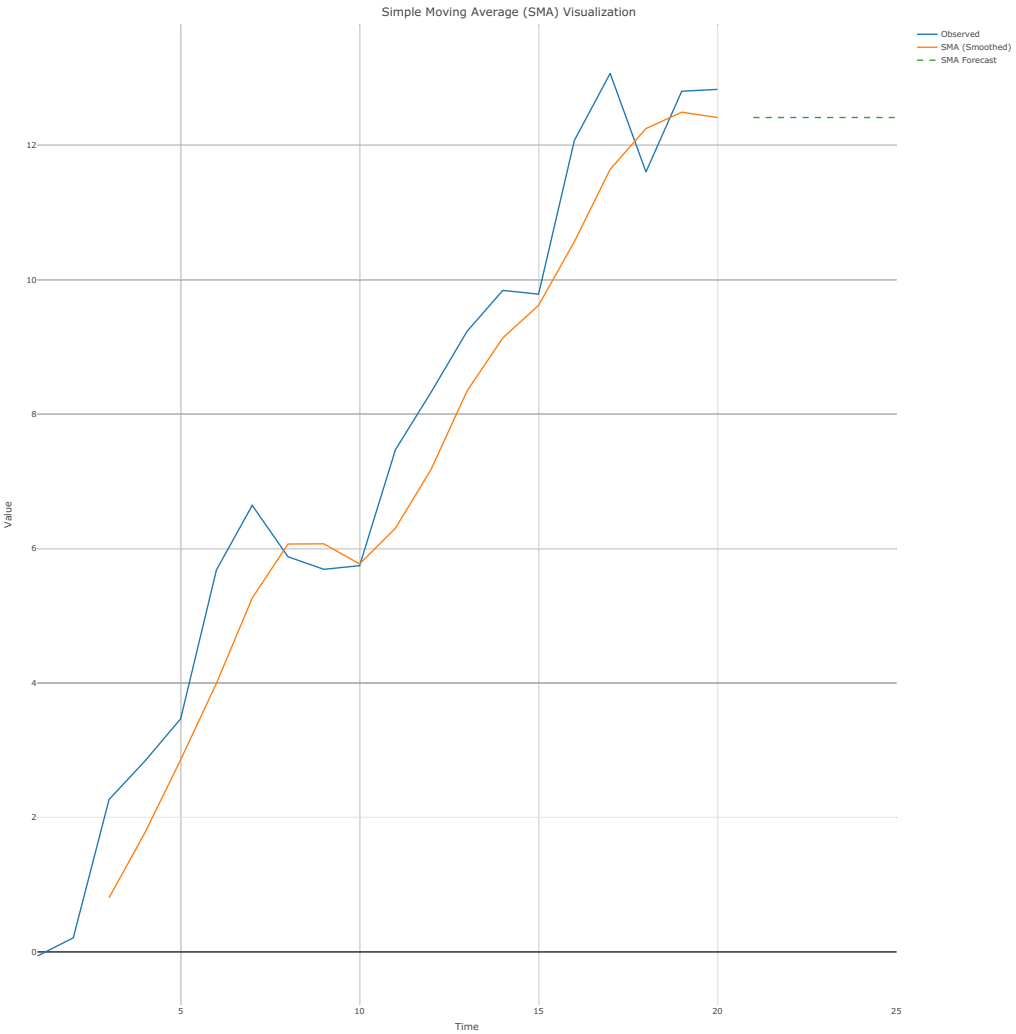
The **Simple Moving Average (SMA)** is one of the fundamental smoothing methods in time series analysis [6]. It reduces short-term fluctuations and highlights longer-term patterns by averaging the most recent k observations.

Formula

For a window size k, the SMA is defined as:

$$\hat{y}_t = \frac{1}{k} \sum_{i=0}^{k-1} y_{t-i}$$

This means the forecast at time t is the average of the most recent k values. Below is an example of the **SMA with window size** $k = 3$ applied to a sample time series.



Interpretation SMA:

The **Observed** line shows the original time series with its natural variability. The **SMA (Smoothed)** line reduces noise by averaging the most recent k observations. This creates a smoother representation of the underlying trend. When forecasting, the SMA method typically uses the **last computed SMA value** as the prediction for all future periods:

$$\hat{y}_{T+1} = \hat{y}_{T+2} = \cdots = \hat{y}_{T+h} = \hat{y}_T$$

This produces a **flat forecast**, similar to the naïve method, but based on the **smoothed series** rather than the last raw observation. The SMA method is widely used as an introductory smoothing technique and serves as a baseline for more advanced exponential smoothing models.

5.1.3 Exponential Smoothing

Exponential smoothing methods are a family of forecasting techniques that apply **exponentially decreasing weights** to past observations. They are widely used because of their simplicity, interpretability, and strong forecasting performance. According to [7], there are **three main types of exponential smoothing models**, each designed for different time series patterns:

Method	Description	Suitable For
Single Exponential Smoothing (SES)	Applies smoothing to the level only.	Data **without trend** and **without seasonality**
Holt's Linear Trend Method	Extends SES by adding a smoothed **trend component**.	Data **with trend**, but **without seasonality**
Holt–Winters Method	Extends Holt's method by adding a **seasonal component** (additive or multiplicative).	Data **with trend** and **with seasonality**

Each model builds upon the previous one by incorporating more structure from the time series, making exponential smoothing a flexible and progressive forecasting framework.

Single Exponential Smoothing (SES)

Single Exponential Smoothing (SES) is designed for time series **without trend and without seasonality**. It provides a smoothed estimate by applying exponentially decreasing weights to past observations.

Formula

$$\hat{y}_{t+1} = \alpha y_t + (1 - \alpha)\hat{y}_t$$

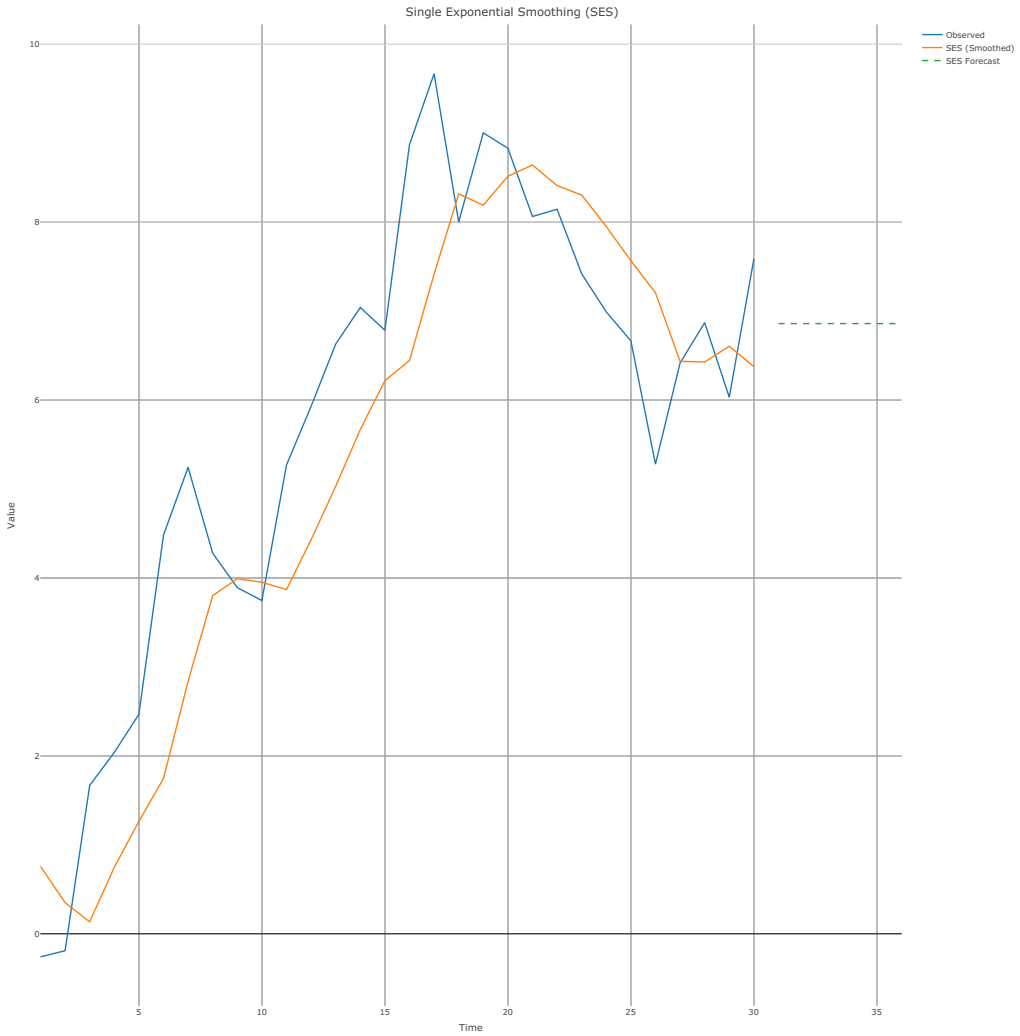
where:

- α is the smoothing parameter, $0 < \alpha \leq 1$
- y_t is the actual value at time t
- $\hat{y}_t$ is the previous smoothed estimate

A higher value of α gives more weight to recent observations, making the model more responsive to sudden changes. A lower value of α produces a smoother curve, giving more weight to older data.

```
library(forecast)
library(plotly)

# --- Sample Time Series ---
set.seed(123)
y <- ts(cumsum(rnorm(30, 0.3, 1)))   # example series

# --- SES Model ---
alpha_value <- 0.4                        # you can change alpha (0 < alpha <= 1)
ses_model <- ses(y, alpha = alpha_value, h = 6)  # SES with forecast horizon h=6

# Extract fitted values and forecasts
fitted_vals <- ses_model$fitted
forecast_vals <- ses_model$mean
t <- 1:length(y)
t_future <- (length(y)+1):(length(y) + length(forecast_vals))

# Build data frame for plot
df <- data.frame(
  time  = c(t, t_future),
  value = c(y, as.numeric(forecast_vals)),
  type  = c(rep("Observed", length(y)), rep("SES Forecast", length(forecast_vals)))
)

df_fitted <- data.frame(
  time = t,
  value = as.numeric(fitted_vals),
  type = "SES (Smoothed)"
)

# --- Plotly Visualization ---
plot_ly() %>%
  add_lines(data = df[df$type == "Observed", ],
            x = ~time, y = ~value,
            name = "Observed") %>%
  add_lines(data = df_fitted,
            x = ~time, y = ~value,
            name = "SES (Smoothed)") %>%
  add_lines(data = df[df$type == "SES Forecast", ],
            x = ~time, y = ~value,
            name = "SES Forecast",
```

```
            line = list(dash = "dash")) %>%
  layout(
    title = "Single Exponential Smoothing (SES)",
    xaxis = list(title = "Time"),
    yaxis = list(title = "Value")
  )
```



Interpretation:

The **Observed** line represents the original time series values. The **SES (Smoothed)** line applies exponential smoothing, where recent observations receive a higher weight controlled by the smoothing parameter α. A larger value of α (closer to 1) makes the smoothed line react more quickly to changes in the data, while a smaller value of α (closer to 0) produces a smoother curve that responds more gradually.

Because SES does not model trend or seasonality, the forecast is constant over the horizon:

$$\hat{y}_{T+1} = \hat{y}_{T+2} = \cdots = \hat{y}_{T+h} = \hat{y}_T$$

SES is therefore suitable only for series with **no trend and no seasonality**, and it serves as the foundation for more advanced exponential smoothing techniques such as **Holt** (for trend) and **Holt–Winters** (for seasonality).

Holt's Linear Trend Method

Holt's Linear Trend Method extends Single Exponential Smoothing (SES) by adding a **trend component**, making it suitable for time series that exhibit a **consistent upward or downward trend**. This method smooths both the **level** and the **trend**, allowing the forecast to grow or decline over time.

Model Equations

The method consists of three components:

- **Level equation**: updates the smoothed estimate of the current value
- **Trend equation**: updates the smoothed estimate of the trend
- **Forecast equation**: projects the level and trend forward

$$\begin{aligned} l_t &= \alpha y_t + (1-\alpha)(l_{t-1} + b_{t-1}) \\ b_t &= \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1} \\ \hat{y}_{t+h} &= l_t + hb_t \end{aligned}$$

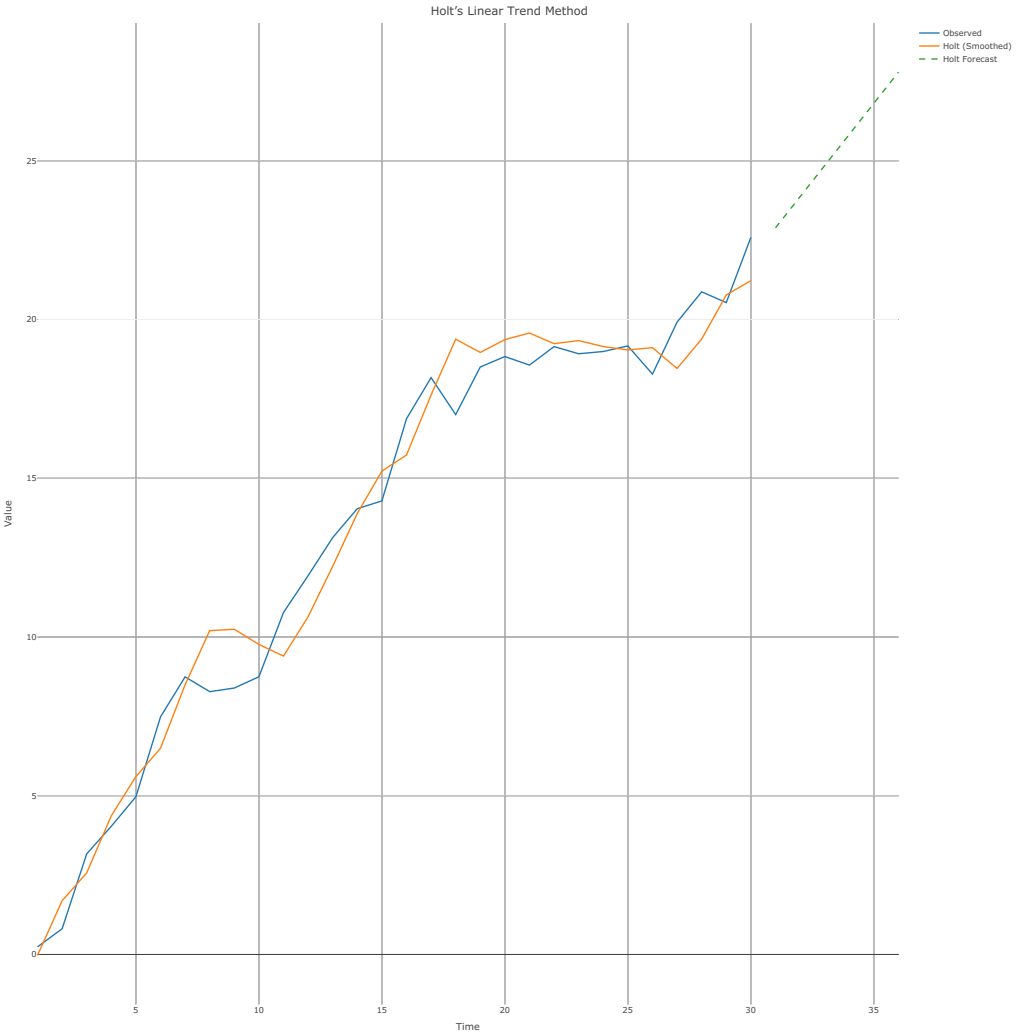
where:

- α = level smoothing parameter, $0 < \alpha \leq 1$
- β = trend smoothing parameter, $0 < \beta \leq 1$
- l_t = estimated level at time t
- b_t = estimated trend at time t
- $\hat{y}_{t+h}$ = forecast h periods ahead

Holt's method allows forecasts to follow a straight-line trajectory, capturing the underlying linear trend in the data.

```
library(forecast)
library(plotly)

# --- Sample Time Series ---
```

```r
set.seed(123)
y <- ts(cumsum(rnorm(30, 0.8, 1)))   # trending series

# --- Holt Model ---
holt_model <- holt(y, h = 6, alpha = 0.5, beta = 0.3)

# Extract components
fitted_vals <- holt_model$fitted
forecast_vals <- holt_model$mean

t <- 1:length(y)
t_future <- (length(y)+1):(length(y) + length(forecast_vals))

# Data for plotting
df <- data.frame(
  time  = c(t, t_future),
  value = c(y, as.numeric(forecast_vals)),
  type  = c(rep("Observed", length(y)), rep("Holt Forecast", length(forecast_vals)))
)

df_fitted <- data.frame(
  time = t,
  value = as.numeric(fitted_vals),
  type = "Holt (Smoothed)"
)

# --- Plotly Visualization ---
plot_ly() %>%
  add_lines(data = df[df$type == "Observed", ],
            x = ~time, y = ~value,
            name = "Observed") %>%
  add_lines(data = df_fitted,
            x = ~time, y = ~value,
            name = "Holt (Smoothed)") %>%
  add_lines(data = df[df$type == "Holt Forecast", ],
            x = ~time, y = ~value,
            name = "Holt Forecast",
            line = list(dash = "dash")) %>%
  layout(
    title = "Holt's Linear Trend Method",
    xaxis = list(title = "Time"),
    yaxis = list(title = "Value")
  )
```



Interpretation:

The **Observed** line shows the original time series, which exhibits a visible upward trend. Holt's method captures this behavior by smoothing two components:

1. **Level** (l_t) — the underlying value of the series
2. **Trend** (b_t) — the rate of increase or decrease

The **Holt (Smoothed)** line represents the result of applying exponential smoothing to both the level and the trend. Compared with SES, Holt's method does not produce a flat curve—because it explicitly models and updates the trend. Holt's method is appropriate for time series with a **consistent linear trend**, but **without seasonality**. It forms the foundation for the Holt–Winters method, which adds a seasonal component for more complex patterns.

Holt–Winters Method

The Holt–Winters method is used when the time series exhibits **both trend and seasonality**. There are two types: **additive** and **multiplicative**. Here, we use the **additive version**, which is suitable for seasonal patterns with roughly constant amplitude.

Model Equations (Additive)

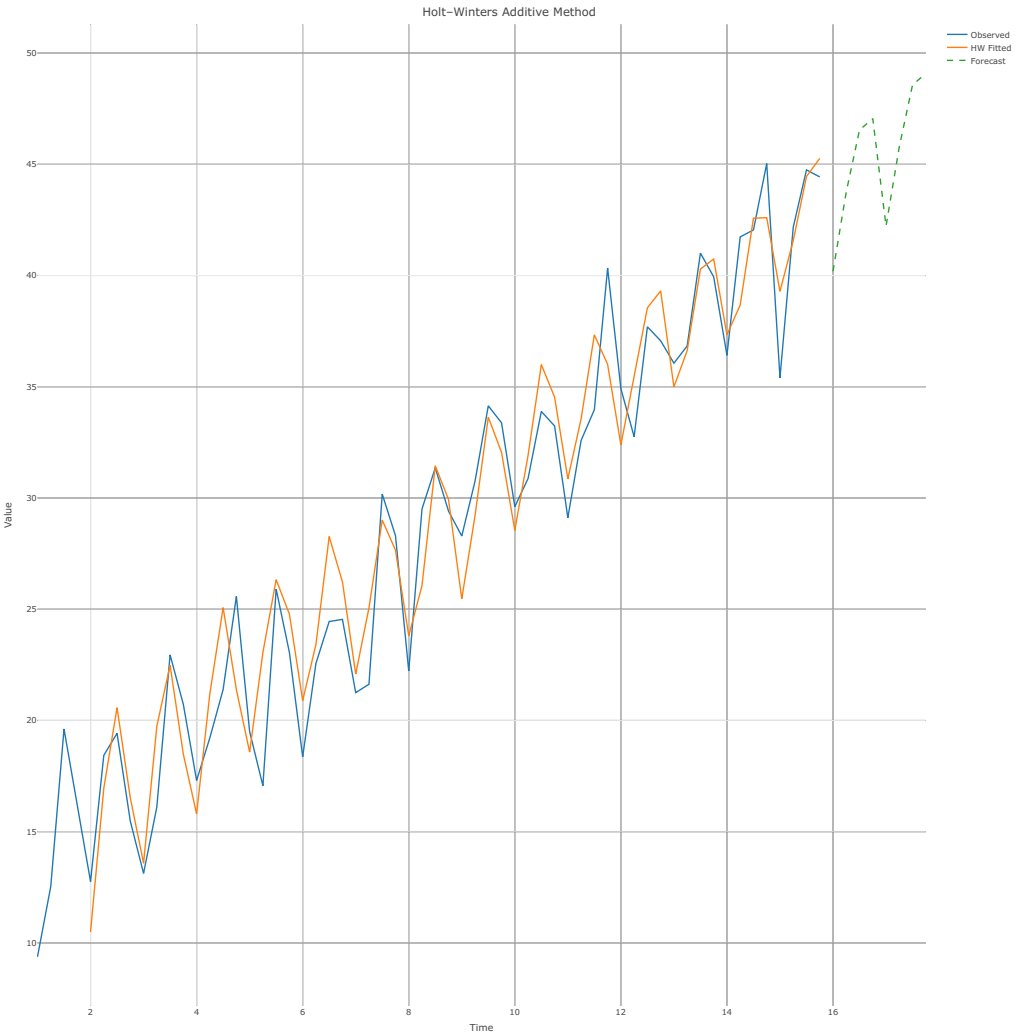
$$
\begin{aligned}
l_t &= \alpha(y_t - s_{t-m}) + (1-\alpha)(l_{t-1} + b_{t-1}) \\
b_t &= \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1} \\
s_t &= \gamma(y_t - l_t) + (1-\gamma)s_{t-m} \\
\hat{y}_{t+h} &= l_t + hb_t + s_{t-m+h}
\end{aligned}
$$

where:

- l_t = level
- b_t = trend
- s_t = seasonal component
- m = seasonal period (e.g., $m = 12$ for monthly data)
- α, β, γ = smoothing parameters

```
library(plotly)
library(forecast)

# Example seasonal + trend data
set.seed(123)
t <- 1:60
season <- rep(c(10, 12, 15, 14), 15)    # seasonal pattern (period = 4)
trend  <- 0.5 * t                        # upward trend
noise  <- rnorm(60, 0, 2)
y <- season + trend + noise
ts_data <- ts(y, frequency = 4)

# Holt-Winters Additive Model
model_hw <- HoltWinters(ts_data, seasonal = "additive")

# Forecast 8 steps ahead
fc <- forecast(model_hw, h = 8)

# Build plotly visualization
p <- plot_ly() %>%
  add_lines(x = time(ts_data), y = ts_data,
            name = "Observed") %>%
  add_lines(x = time(model_hw$fitted), y = model_hw$fitted[,1],
```

```
            name = "HW Fitted") %>%
  add_lines(x = time(fc$mean), y = fc$mean,
            name = "Forecast", line = list(dash = "dash")) %>%
  layout(title = "Holt-Winters Additive Method",
         xaxis = list(title = "Time"),
         yaxis = list(title = "Value"))

p
```



Interpretation:

In the Holt–Winters method, the **smoothing parameters** control how much weight is given to recent observations versus past values. They range between 0 and 1:

1. **Level Smoothing** (α)

- Determines how much the current observation y_t influences the estimated level l_t.
- **High α (close to 1)** $\rightarrow$ the level reacts quickly to recent changes.
- **Low α (close to 0)** $\rightarrow$ the level changes slowly, giving more weight to historical values.

$$l_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(l_{t-1} + b_{t-1})$$

2. **Trend Smoothing (β)**
   - Controls the update of the trend component b_t.
   - **High β** $\rightarrow$ trend responds quickly to changes in slope.
   - **Low β** $\rightarrow$ trend is more stable and less sensitive to short-term fluctuations.

$$b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1}$$

3. **Seasonal Smoothing (γ)**
   - Determines how fast the seasonal component s_t adapts to new seasonal patterns.
   - **High γ** $\rightarrow$ seasonal estimates adjust quickly to changes.
   - **Low γ** $\rightarrow$ seasonal pattern changes slowly over time.

$$s_t = \gamma(y_t - l_t) + (1-\gamma)s_{t-m}$$

5.1.4 ARIMA

AR(p): Autoregressive

The **Autoregressive (AR)** model predicts a time series based on its **own past values**. It is widely used in time series forecasting and forms the foundation for ARMA/ARIMA models.

Model Formula

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \epsilon_t$$

where:

- y_t = value of the series at time t
- c = constant term (intercept)

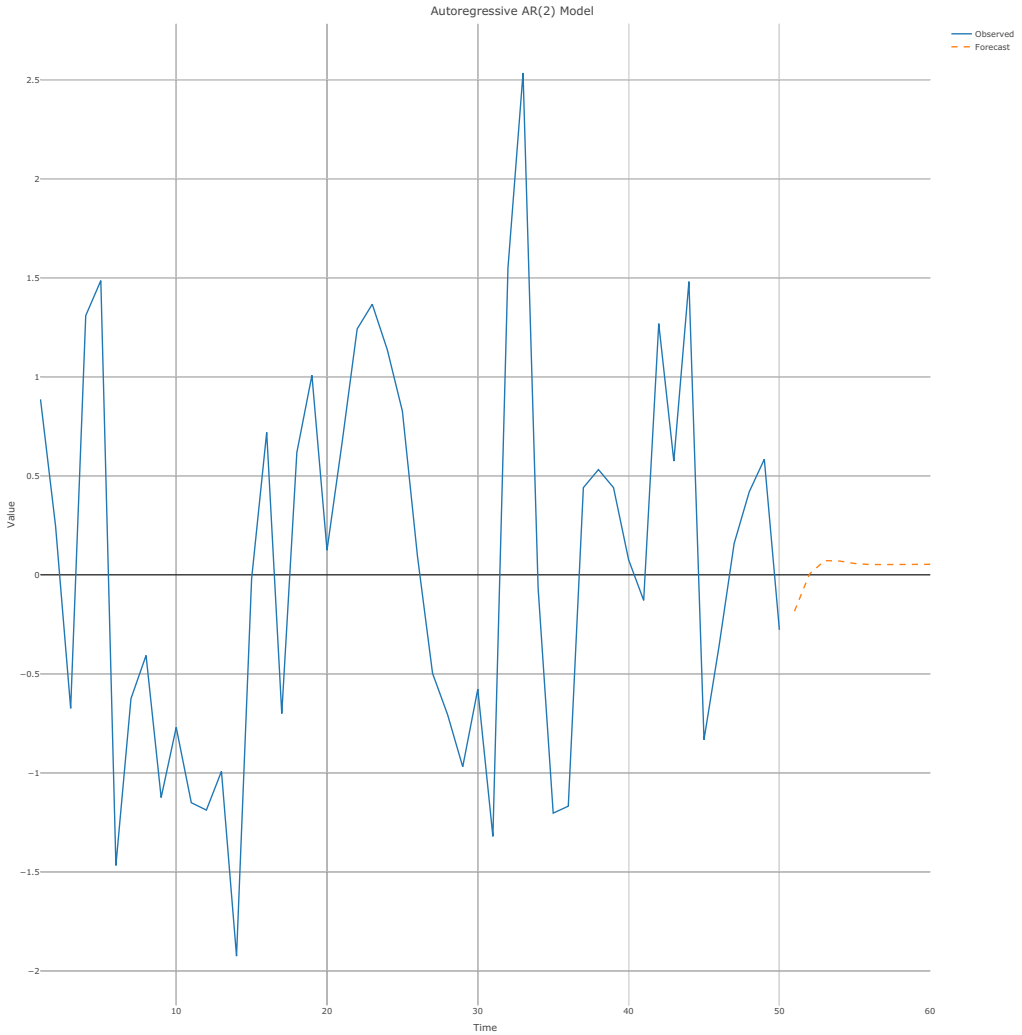
- $\phi_1, \phi_2, \ldots, \phi_p$ = autoregressive coefficients
- p = order of the AR model (number of lagged terms)
- ϵ_t = white noise error term, $\epsilon_t \sim N(0, \sigma^2)$

Key Points:

- The model assumes that the current value y_t is a **linear combination of its past p values** plus random noise.
- The coefficients ϕ_i determine how much influence each lagged term has.
- The **order** p can be selected using information criteria (AIC, BIC) or autocorrelation plots.

```
library(forecast)
library(plotly)

# --- Simulate AR(2) Process ---
set.seed(123)
ar_coeff <- c(0.6, -0.3)   # phi1=0.6, phi2=-0.3
y <- arima.sim(n = 50, list(ar = ar_coeff), sd = 1)

# Fit AR model
fit <- arima(y, order = c(2,0,0))  # AR(2)

# Forecast 10 steps ahead
fc <- forecast(fit, h = 10)

# Time index
t <- 1:length(y)
t_future <- (length(y)+1):(length(y)+length(fc$mean))

# Build data frame
df <- data.frame(
  time = c(t, t_future),
  value = c(y, as.numeric(fc$mean)),
  type = c(rep("Observed", length(y)), rep("AR(2) Forecast", length(fc$mean)))
)

# Plotly visualization
plot_ly() %>%
  add_lines(data = df[df$type=="Observed",], x = ~time, y = ~value,
            name = "Observed") %>%
  add_lines(data = df[df$type=="AR(2) Forecast",], x = ~time, y = ~value,
            name = "Forecast", line = list(dash = "dash")) %>%
  layout(title = "Autoregressive AR(2) Model",
```

```
        xaxis = list(title = "Time"),
        yaxis = list(title = "Value"))
```



Interpretation

- If ϕ_i is **positive**, the series tends to continue in the same direction as lag i.
- If ϕ_i is **negative**, the series tends to move in the opposite direction of lag i.
- AR models are **stationary** if the roots of the characteristic equation lie outside the unit circle.

MA(q): Moving Average

The **Moving Average (MA)** model expresses the current value of a time series as a **linear combination of past error terms**.

It is commonly used in time series modeling, often as part of ARMA or ARIMA models.

Model Formula

$$y_t = c + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q} + \epsilon_t$$

where:

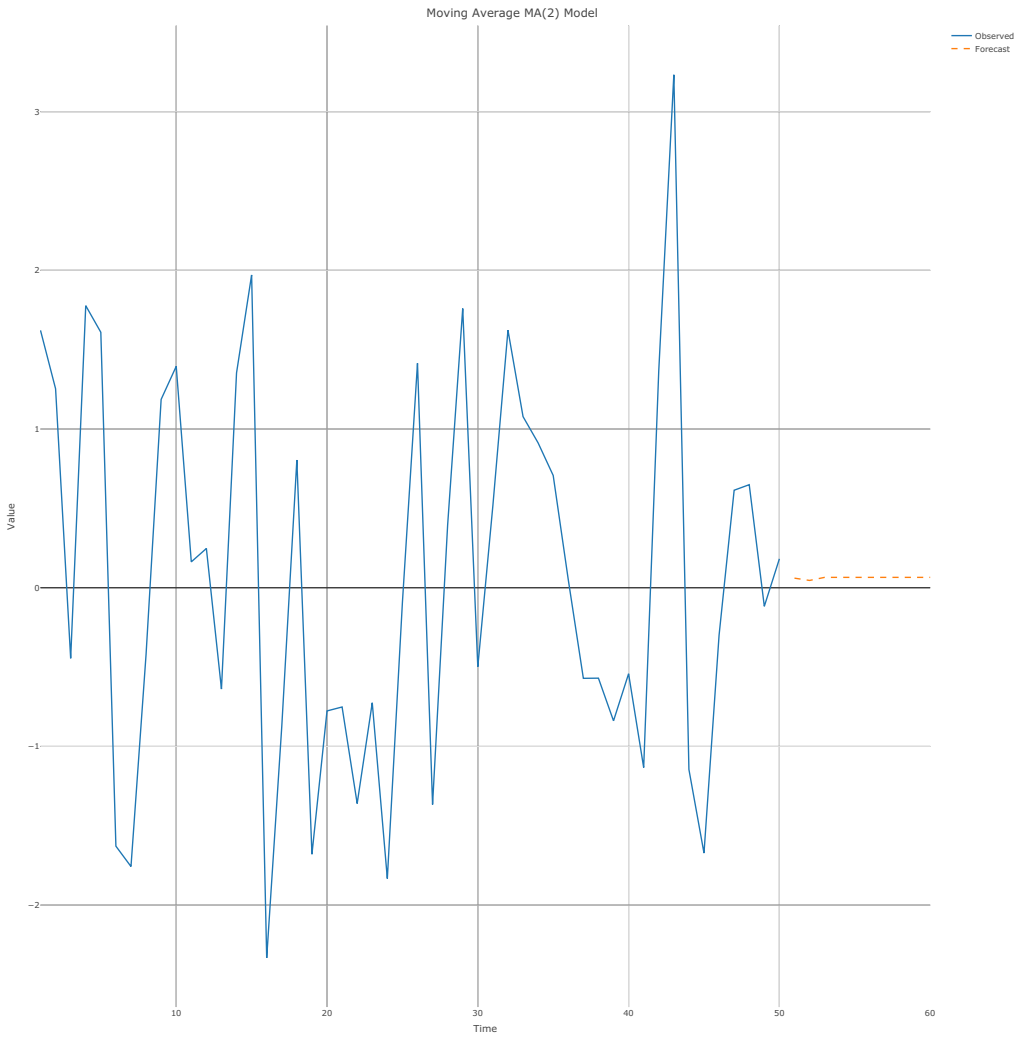
- y_t = value of the series at time t
- c = constant term (intercept)
- $\theta_1, \theta_2, \ldots, \theta_q$ = moving average coefficients
- q = order of the MA model (number of past error terms included)
- ϵ_t = white noise error term, $\epsilon_t \sim N(0, \sigma^2)$

Key Points:

- Unlike AR models, MA models **do not use past values of** y_t, but instead use **past forecast errors**.
- Each coefficient θ_i measures the influence of the **lagged errors** on the current value.
- The order q can be determined by examining the **autocorrelation function (ACF)**:
  - The ACF of an MA(q) process **cuts off after lag** q.

```
library(forecast)
library(plotly)

# --- Simulate MA(2) Process ---
set.seed(123)
ma_coeff <- c(0.7, -0.4)  # theta1=0.7, theta2=-0.4
y <- arima.sim(n = 50, list(ma = ma_coeff), sd = 1)

# Fit MA model
fit <- arima(y, order = c(0,0,2))  # MA(2)

# Forecast 10 steps ahead
fc <- forecast(fit, h = 10)

# Time index
t <- 1:length(y)
t_future <- (length(y)+1):(length(y)+length(fc$mean))
```

```r
# Build data frame
df <- data.frame(
  time = c(t, t_future),
  value = c(y, as.numeric(fc$mean)),
  type = c(rep("Observed", length(y)), rep("MA(2) Forecast", length(fc$mean)))
)

# Plotly visualization
plot_ly() %>%
  add_lines(data = df[df$type=="Observed",], x = ~time, y = ~value,
            name = "Observed") %>%
  add_lines(data = df[df$type=="MA(2) Forecast",], x = ~time, y = ~value,
            name = "Forecast", line = list(dash = "dash")) %>%
  layout(title = "Moving Average MA(2) Model",
         xaxis = list(title = "Time"),
         yaxis = list(title = "Value"))
```



Interpretation:

1. The current value y_t is affected by the **most recent** q **random shocks** $\epsilon_{t-1}, \dots, \epsilon_{t-q}$.
2. Positive $\theta_i \rightarrow$ the lagged shock pushes y_t in the same direction.
3. Negative $\theta_i \rightarrow$ the lagged shock pushes y_t in the opposite direction.
4. MA models are **stationary by definition**, so there is no need for differencing as in AR models.
5. MA is often combined with AR to form **ARMA**(p, q) for better capturing serial dependence in data.

ARMA(p, q)**: Autoregressive Moving Average**

The **ARMA**(p, q) model combines both **autoregressive (AR)** and **moving average (MA)** components, capturing the effects of both **past values** and **past errors** on the current observation.

Model Formula

$$y_t = c + \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q} + \epsilon_t$$

where:

- y_t = value at time t
- c = constant term (intercept)
- $\phi_1, \dots, \phi_p$ = AR coefficients
- $\theta_1, \dots, \theta_q$ = MA coefficients
- p = order of the AR part
- q = order of the MA part
- ϵ_t = white noise error term, $\epsilon_t \sim N(0, \sigma^2)$

Key Points:

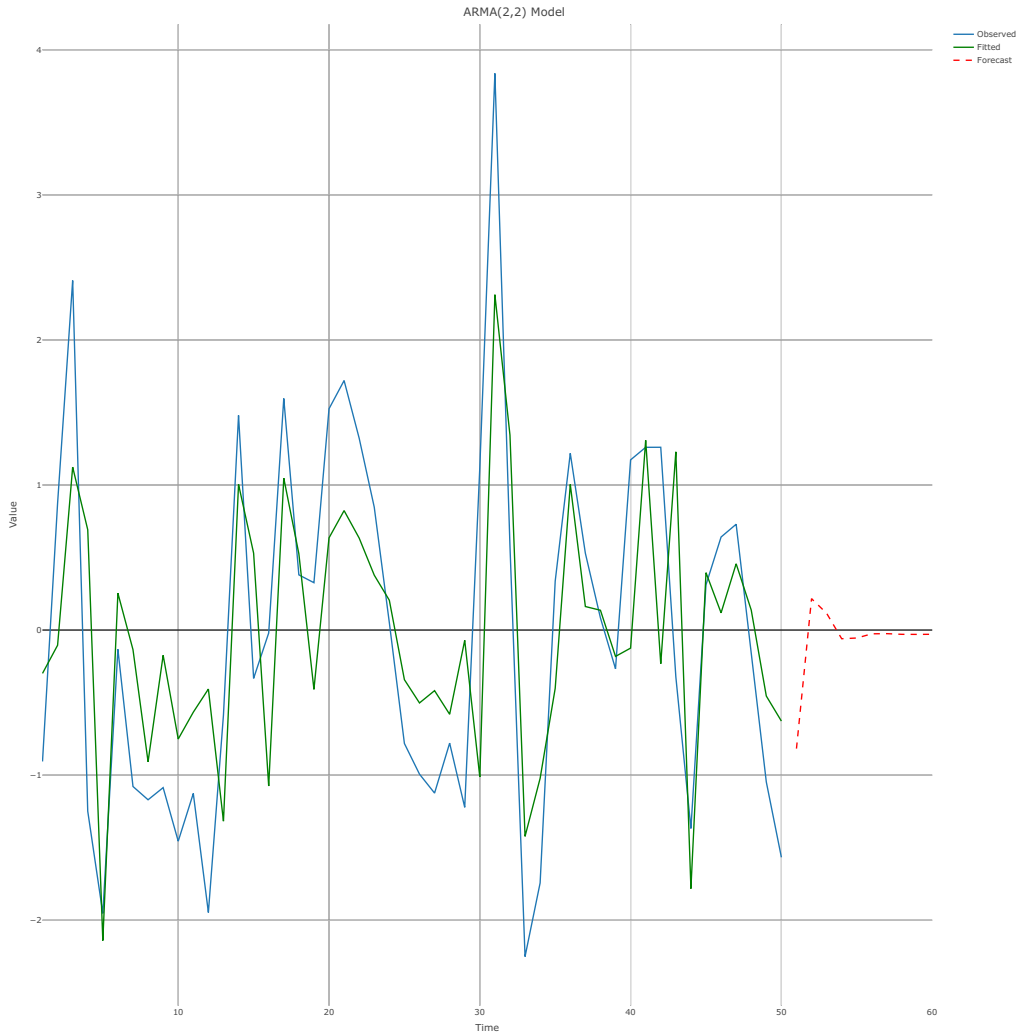
- The **AR component** captures the influence of past values on y_t.
- The **MA component** captures the influence of past shocks (errors) on y_t.

- The **orders** p **and** q can be selected using **information criteria** (AIC, BIC) or by examining the **ACF and PACF** plots.

- ARMA models assume **stationarity**, so non-stationary series may require differencing (ARIMA).

```
library(forecast)
library(plotly)

# --- Simulate ARMA(2,2) Process ---
set.seed(123)
ar_coeff <- c(0.6, -0.3)   # AR(2)
ma_coeff <- c(0.5, -0.4)   # MA(2)
y <- arima.sim(n = 50, list(ar = ar_coeff, ma = ma_coeff), sd = 1)

# Fit ARMA model (ARMA(2,2))
fit <- arima(y, order = c(2,0,2))

# Forecast 10 steps ahead
fc <- forecast(fit, h = 10)

# Time index
t <- 1:length(y)
t_future <- (length(y)+1):(length(y)+length(fc$mean))

# Build data frame
df <- data.frame(
  time = c(t, t_future),
  value = c(y, as.numeric(fc$mean)),
  type = c(rep("Observed", length(y)), rep("ARMA(2,2) Forecast", length(fc$mean)))
)

# Build fitted values data frame
df_fitted <- data.frame(
  time = t,
  value = as.numeric(fitted(fit)),
  type = "ARMA(2,2) Fitted"
)

# Plotly visualization
plot_ly() %>%
  add_lines(data = df[df$type=="Observed",], x = ~time, y = ~value,
            name = "Observed") %>%
  add_lines(data = df_fitted, x = ~time, y = ~value,
            name = "Fitted", line = list(color = 'green')) %>%
  add_lines(data = df[df$type=="ARMA(2,2) Forecast",], x = ~time, y = ~value,
            name = "Forecast", line = list(dash = "dash", color = 'red')) %>%
  layout(title = "ARMA(2,2) Model",
         xaxis = list(title = "Time"),
```

```
        yaxis = list(title = "Value"))
```



Interpretation:

1. **ARMA(p, q) captures both momentum and shocks**:
   - AR part models persistence (trend or autocorrelation)
   - MA part models short-term noise effects
2. **Forecasting with ARMA(p, q)**:
   - Future values are predicted using a combination of past observations and past errors.
   - Provides a more flexible model than AR or MA alone, suitable for stationary time series with autocorrelation and short-term shocks.

3. **Advantages**:
    - Can model a wide variety of stationary time series patterns.
    - Forms the basis for **ARIMA** when differencing is added for non-stationary series.

ARIMA(p, d, q): Autoregressive Integrated Moving Average

The **ARIMA(p, d, q)** model extends ARMA models to **non-stationary time series** by incorporating **differencing**. It is one of the most widely used models for time series forecasting.

Model Formula:

$$\nabla^d y_t = (1 - B)^d y_t = ARMA(p, q)$$

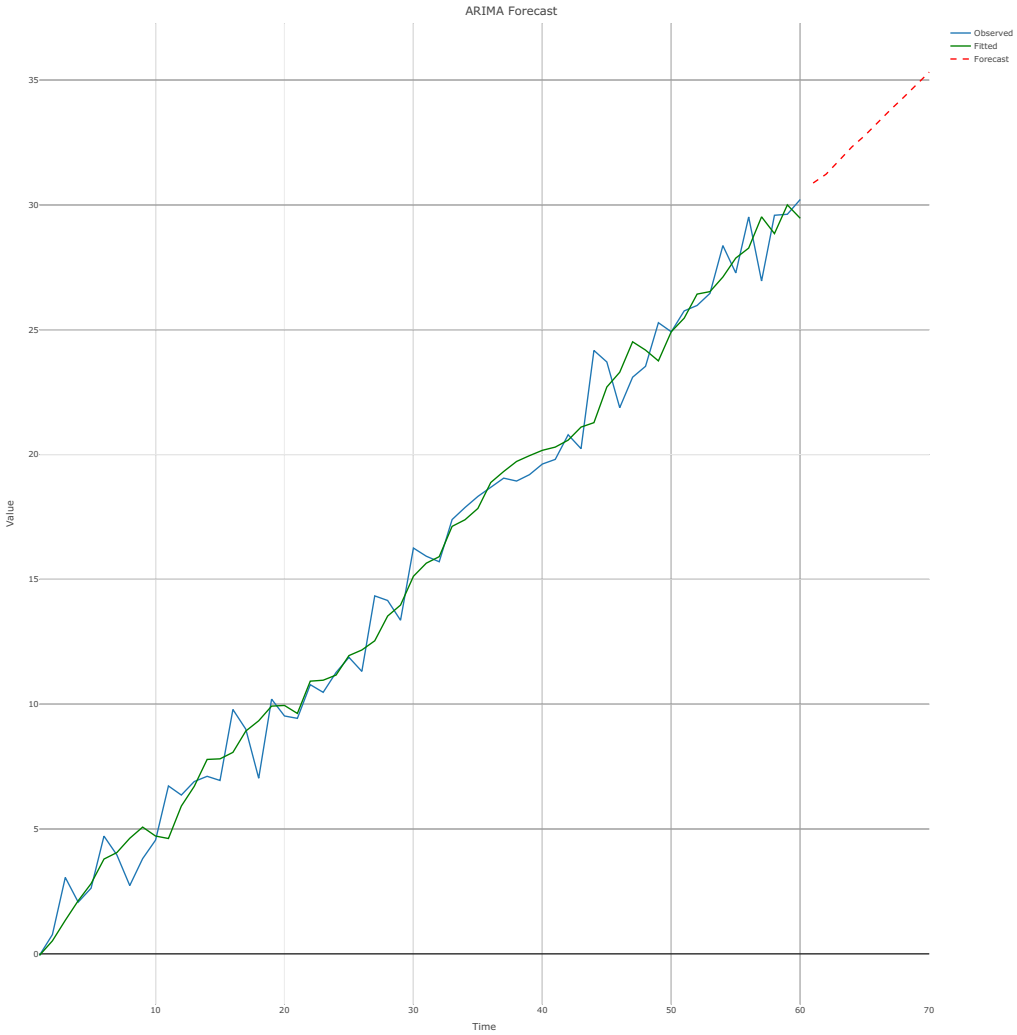
where:

- y_t = original time series at time t
- B = backshift operator, $By_t = y_{t-1}$
- ∇^d = differencing operator applied d times, $\nabla^d y_t = (1 - B)^d y_t$
- p = order of the AR part
- d = order of differencing required to make the series stationary
- q = order of the MA part

Key Points:

- **Differencing (d)**: Removes trend or seasonality to achieve stationarity.
- Once differenced, the series can be modeled with an **ARMA(p, q)** structure.
- ARIMA is denoted as ARIMA(p, d, q):
    - p: number of autoregressive terms
    - d: number of differences
    - q: number of moving average terms

```r
library(forecast)
library(plotly)

# --- Simulate non-stationary series with trend ---
set.seed(123)
t <- 1:60
trend <- 0.5 * t
noise <- rnorm(60, 0, 1)
y <- trend + noise
ts_data <- ts(y)

# Plot original series
# Fit ARIMA model (auto.arima to select p,d,q)
fit <- auto.arima(ts_data)

# Forecast 10 steps ahead
fc <- forecast(fit, h = 10)

# Time index
t_obs <- 1:length(ts_data)
t_future <- (length(ts_data)+1):(length(ts_data)+length(fc$mean))

# Data frames
df <- data.frame(
  time = c(t_obs, t_future),
  value = c(ts_data, as.numeric(fc$mean)),
  type = c(rep("Observed", length(ts_data)), rep("Forecast", length(fc$mean)))
)

df_fitted <- data.frame(
  time = t_obs,
  value = as.numeric(fitted(fit)),
  type = "Fitted"
)

# Plotly visualization
plot_ly() %>%
  add_lines(data = df[df$type=="Observed",], x = ~time, y = ~value,
            name = "Observed") %>%
  add_lines(data = df_fitted, x = ~time, y = ~value,
            name = "Fitted", line = list(color = 'green')) %>%
  add_lines(data = df[df$type=="Forecast",], x = ~time, y = ~value,
            name = "Forecast", line = list(dash = "dash", color = 'red')) %>%
  layout(title = "ARIMA Forecast",
         xaxis = list(title = "Time"),
         yaxis = list(title = "Value"))
```



Interpretation:

1. **Non-stationary series** (with trend or evolving mean) cannot be directly modeled with ARMA.
2. **Differencing** transforms the series into a stationary series.
3. ARIMA forecasts combine:
   - Past values (p),
   - Past errors (q),
   - Differencing to handle non-stationarity (d).
4. ARIMA forms the foundation for seasonal models (**SARIMA**) when seasonality is included.

5.1.5 SARIMA / SARIMAX

SARIMA and **SARIMAX** are extensions of ARIMA that allow modeling of **seasonal patterns** and **external regressors**.

SARIMA (Seasonal ARIMA)

SARIMA incorporates **seasonality** using seasonal autoregressive (P), seasonal differencing (D), and seasonal moving average (Q) components, along with the seasonal period s.

Model notation:

$$SARIMA(p,d,q)(P,D,Q)_s$$

- p, d, q = non-seasonal AR, differencing, MA orders
- P, D, Q = seasonal AR, seasonal differencing, seasonal MA orders
- s = length of the seasonal cycle (e.g., 12 for monthly data)

Equation (conceptual):

$$\nabla^d \nabla_s^D y_t = ARMA(p,q) + ARMA(P,Q)_s$$

where $\nabla_s^D y_t = (1 - B^s)^D y_t$ is **seasonal differencing**.

SARIMAX (Seasonal ARIMA with eXogenous variables)

SARIMAX extends SARIMA by adding **external regressors** X_t:

$$y_t = SARIMA(p,d,q)(P,D,Q)_s + \beta X_t + \epsilon_t$$

- X_t = external variables that may affect the time series
- β = coefficients for regressors

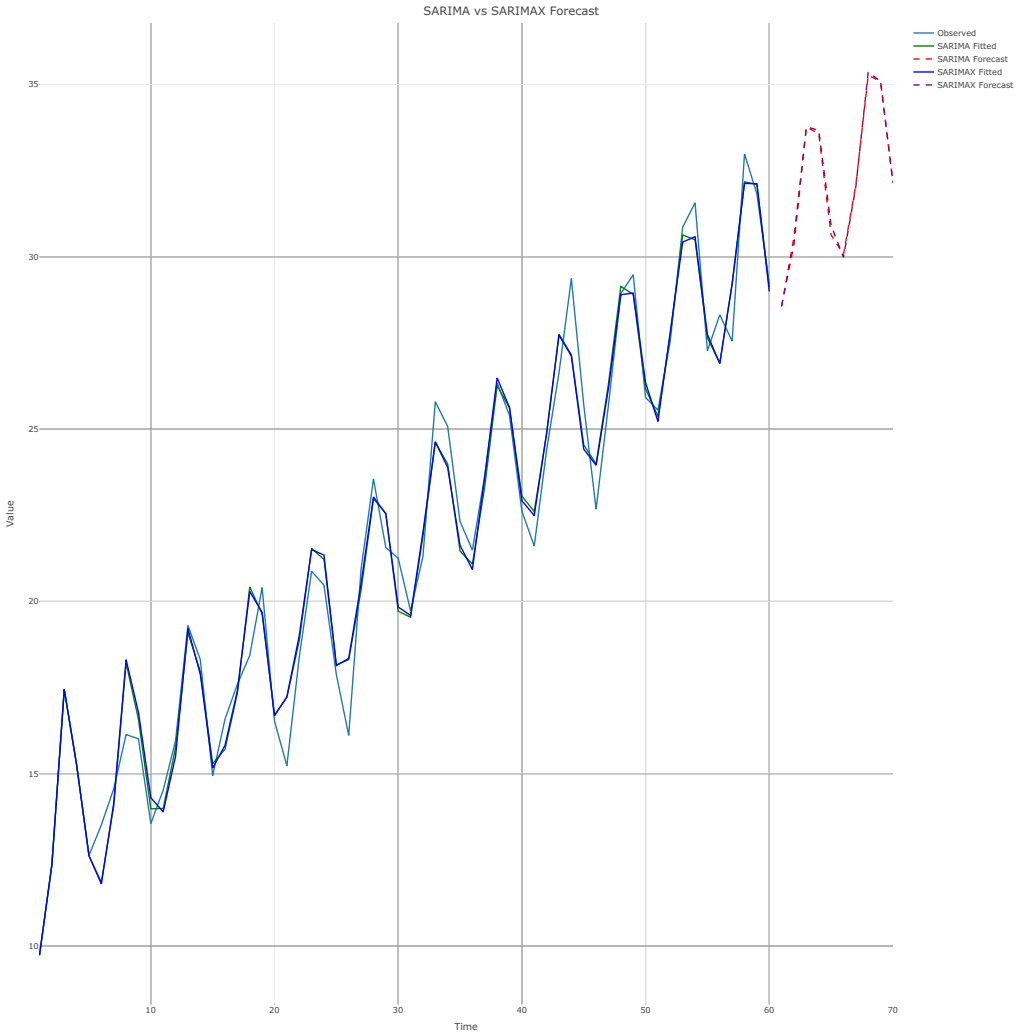
Key Points:

- SARIMA is suitable for **seasonal time series**, e.g., monthly sales with yearly seasonality.
- SARIMAX allows **external influences**, e.g., promotions, holidays, temperature.
- Seasonal parameters (P, D, Q, s) capture repeating patterns in data.
- Model selection can use **AIC/BIC**, **ACF/PACF plots**, and **seasonal diagnostics**.

5.1.6 SARIMA and SARIMAX

```
library(forecast)
library(plotly)

# --- Simulate seasonal time series ---
set.seed(123)
t <- 1:60
seasonal <- rep(c(10, 12, 15, 14, 11), 12)[1:60]  # seasonal pattern (period s=5)
trend <- 0.3 * t
noise <- rnorm(60, 0, 1)
y <- trend + seasonal + noise
ts_data <- ts(y, frequency = 5)

# --- External regressor for SARIMAX ---
x <- rnorm(60, 5, 1)  # exogenous variable

# Fit SARIMA model
fit_sarima <- auto.arima(ts_data, seasonal = TRUE)

# Fit SARIMAX model
fit_sarimax <- auto.arima(ts_data, xreg = x, seasonal = TRUE)

# Forecast 10 steps ahead
x_future <- rnorm(10, 5, 1)  # future exogenous variable for SARIMAX
fc_sarima <- forecast(fit_sarima, h = 10)
fc_sarimax <- forecast(fit_sarimax, xreg = x_future, h = 10)

# Time index
t_obs <- 1:length(ts_data)
t_future <- (length(ts_data)+1):(length(ts_data)+10)

# Build data frames for plotting
df_sarima <- data.frame(
  time = c(t_obs, t_future),
  value = c(ts_data, as.numeric(fc_sarima$mean)),
  type = c(rep("Observed", length(ts_data)), rep("SARIMA Forecast", 10))
)

df_sarimax <- data.frame(
  time = c(t_obs, t_future),
  value = c(ts_data, as.numeric(fc_sarimax$mean)),
  type = c(rep("Observed", length(ts_data)), rep("SARIMAX Forecast", 10))
)

df_fitted_sarima <- data.frame(
  time = t_obs,
  value = as.numeric(fitted(fit_sarima)),
```

```
  type = "SARIMA Fitted"
)

df_fitted_sarimax <- data.frame(
  time = t_obs,
  value = as.numeric(fitted(fit_sarimax)),
  type = "SARIMAX Fitted"
)

# Plotly visualization
plot_ly() %>%
  add_lines(data = df_sarima[df_sarima$type=="Observed",], x = ~time, y = ~value,
            name = "Observed") %>%
  add_lines(data = df_fitted_sarima, x = ~time, y = ~value,
            name = "SARIMA Fitted", line = list(color = 'green')) %>%
  add_lines(data = df_sarima[df_sarima$type=="SARIMA Forecast",], x = ~time, y = ~value,
            name = "SARIMA Forecast", line = list(dash = "dash", color = 'red')) %>%
  add_lines(data = df_fitted_sarimax, x = ~time, y = ~value,
            name = "SARIMAX Fitted", line = list(color = 'blue')) %>%
  add_lines(data = df_sarimax[df_sarimax$type=="SARIMAX Forecast",], x = ~time, y = ~value,
            name = "SARIMAX Forecast", line = list(dash = "dash", color = 'purple')) %>%
  layout(title = "SARIMA vs SARIMAX Forecast",
         xaxis = list(title = "Time"),
         yaxis = list(title = "Value"))
```



Interpretation:

1. **Seasonality**
   - Seasonal AR (P) captures dependence on previous seasonal cycles.
   - Seasonal MA (Q) captures seasonal shocks.
   - Seasonal differencing (D) removes repeating seasonal trends.
2. **Exogenous Variables (SARIMAX only)**
   - The model accounts for additional predictors outside the time series itself.
   - Useful when external events significantly impact the series.
3. **Forecasting**

- Combines non-seasonal ARMA dynamics, seasonal patterns, and external regressors (if any).
- Produces more accurate predictions for seasonal and influenced time series.

5.2 Machine Learning Models

```
Video cannot be displayed in PDF/Word.
 Please view the HTML version or open directly on YouTube:
 https://www.youtube.com/embed/vV12dGe_Fho?si=LCdcDWhbUa01YBEX
```

5.2.1 Linear Regression

Linear Regression for Time Series [8], uses time-dependent features such as:

- $y_{t-1}, y_{t-2}, \cdots$
- moving averages
- sine/cosine seasonal features

5.2.2 Non-Linear Regression

Random Forest, XGBoost, and LightGBM [9], tree-based models capable of capturing nonlinear patterns.

5.3 Deep Learning Models

```
Video cannot be displayed in PDF/Word.
 Please view the HTML version or open directly on YouTube:
 https://www.youtube.com/embed/AsNTP8Kwu80?si=413sqZ1qzEibJyL0
```

5.3.1 Recurrent Neural Networks

Recurrent Neural Networks (RNN) [10], designed to capture sequential dependencies.

5.3.2 Long Short-Term Memory

Long Short-Term Memory (LSTM) [11], effective for long-range temporal dependencies.

5.3.3 Gated Recurrent Unit

Gated Recurrent Unit (GRU) [12], a simplified alternative to LSTM with similar performance.

5.3.4 CNN for Time Series

CNN for Time Series [13], uses convolutional filters to detect local temporal patterns.

5.3.5 Encoder–Decoder

Encoder–Decoder (Seq2Seq) Models [14], designed for multi-step or multi-horizon forecasting.

5.3.6 Transformer Architectures

Transformer Architectures [15], examples include Informer, Autoformer, and FEDformer.

Strengths:
- Long-range dependency modeling
- Strong performance on large datasets

5.4 Model Selection Summary

Data Pattern	Suitable Models
No strong pattern	Naïve, SMA
Trend	Holt, ARIMA
Trend + Seasonality	Holt–Winters, SARIMA
External regressors	ARIMAX, SARIMAX
Nonlinear structure	Random Forest, XGBoost
Complex dependencies	LSTM, GRU, CNN, Encoder–Decoder
Long-range forecasting	Transformers

Chapter 6

Time Series Forecasting

6.1 Classical Statistical Models

6.2 Machine Learning Models

6.3 Deep Learning Models

Chapter 7

Time Series vs Cross Sectional

Metrics for regresi, clasifition, dan time series (RMSE, MAE, R^2, Accuracy, F1-score, AUC, etc.) Cross-validation dan train-test split Overfitting dan underfitting Hyperparameter tuning

Chapter 8

Smart Prediction

8.1 Deep Learning Models

8.2 NLP Prediction

8.3 Computer Vision

8.4 Smart Systems

Chapter 9

Intelligent Analytics

9.1 AI-powered Prediction

9.2 AutoML

9.3 Explainable AI

9.4 Ethics & Fairness

Chapter 10

Future Prediction

10.1 Data-driven Decisions

10.2 Big Data & Streaming

10.3 Deployment (API, n8n, MLOps)

Monitoring & Retraining

[1] James, G., Witten, D., Hastie, T., and Tibshirani, R., An introduction to statistical learning: With applications in r, Springer, 2021

[2] Han, J., Pei, J., and Tong, H., Data mining: Concepts and techniques, Morgan Kaufmann, 2022

[3] Kuhn, M. and Silge, J., Tidy modeling with r, O'Reilly Media, 2022

[4] Boehmke, B. and Greenwell, B. M., Hands-on machine learning with r, CRC Press, 2021

[5] Wilke, C. O., Fundamentals of data visualization, O'Reilly Media, 2020

[6] Box, G. E. P., Jenkins, G. M., Reinsel, G. C., and Ljung, G. M., Time series analysis: Forecasting and control, John Wiley & Sons, 2015

[7] Hyndman, R. J. and Athanasopoulos, G., Forecasting: Principles and practice, OTexts, 2008

[8] James, G., Witten, D., Hastie, T., and Tibshirani, R., An introduction to statistical learning, Springer, 2021

[9] Friedman, J. H., Greedy function approximation: A gradient boosting machine, *Annals of Statistics*, vol. 29, no. 5, 1189–1232, 2001

[10] Rumelhart, D. E., Hinton, G. E., and Williams, R. J., Learning representations by back-propagating errors, *Nature*, vol. 323, 533–536, 1986

[11] Hochreiter, S. and Schmidhuber, J., Long short-term memory, *Neural Computation*, vol. 9, no. 8, 1735–1780, 1997

[12] Cho, K., Merrienboer, B. van, Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y., Learning phrase representations using RNN encoder–decoder for statistical machine translation, *arXiv preprint arXiv:1406.1078*, 2014

[13] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P., Convolutional networks for images, speech, and time-series, The handbook of brain theory and neural networks, MIT Press, 1995

[14] Sutskever, I., Vinyals, O., and Le, Q. V., Sequence to sequence learning with neural networks, Advances in neural information processing systems (NeurIPS), 2014

[15] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I., Attention is all you need, Advances in neural information processing systems (NeurIPS), 2017