ITSB | Eco Industry Oriented University

sinarmas

DATA SCIENCE PROGRAMMING

Study Case Using R and Python

Writer:

Bakti Siregar, M.Sc., CDS.

D'SCIENCELABS
Smart Idea

Kampus Merdeka
INDONESIA JAYA

First Edition

Data Science Programming

Study Case Using R and Python

Bakti Siregar, M.Sc.,CDS

Table of contents

In today's digital age, data plays a crucial role in decision-making across various industries. Data Science Programming is essential for extracting meaningful insights from large datasets, automating analytical processes, and developing predictive models. Proficiency in data science programming enables professionals to analyze data effectively, optimize workflows, and create intelligent systems that drive innovation and technological advancement.

This module provides a comprehensive introduction to the fundamental concepts and practical applications of programming in data science. It covers key topics such as data manipulation, statistical analysis, machine learning, and automation using widely recognized programming languages like Python and R. Readers will have the opportunity to gain hands-on experience in handling real-world data, creating insightful visualizations, and applying statistical models to uncover patterns and trends.

Furthermore, the module explores data preprocessing, transformation, and integration, which are essential steps in preparing raw data for analysis. Readers will also develop practical skills in debugging, testing, and optimizing code to enhance efficiency and accuracy in data-driven projects.

Preface

About the Writer

Bakti Siregar, M.Sc., CDS works as a Lecturer at the ITSB Data Science Program. He earned his Master's degree from the Department of Applied Mathematics at National Sun Yat Sen University, Taiwan. In addition to teaching, Bakti also works as a Freelance Data Scientist for leading companies such as JNE, Samora Group, Pertamina, and PT. Green City Traffic.

He has a strong enthusiasm for projects (and teaching) in the fields of Big Data Analytics, Machine Learning, Optimization, and Time Series Analysis, particularly in finance and investment. His core expertise lies in statistical programming languages such as R Studio and Python. He is also experienced in implementing database systems like MySQL/NoSQL for data management and is proficient in using Big Data tools such as Spark and Hadoop.

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

---

Acknowledgments

Data Science Programming is a crucial skill in analyzing and processing large-scale data. This module covers essential programming techniques for data science, including:

- A solid foundation in **Python and R programming**
- The ability to **manipulate, analyze, and visualize data** effectively
- A deep understanding of **how domain knowledge enhances data analysis**
- Practical skills to apply **Data Science techniques in real-world scenarios**

This book are designed for beginners who aim to build a strong foundation in **Data Science Programming** while appreciating the critical role of **domain expertise**.

I appreciate the active participation of learners, whose questions and discussions enriched the training experience. I hope this material serves as a practical guide for applying programming in data science projects.

Feedback & Suggestions

Your feedback is essential in improving this module. We invite all participants to share their thoughts on the content, structure, and clarity of the materials. Suggestions for additional topics or areas requiring further explanation are highly appreciated.

With your support and contributions, we aim to refine this E-book, making it a more comprehensive resource for **Data Science Programming**. Thank you for your participation!

For feedback and suggestions, feel free to contact:

- dsciencelabs@outlook.com
- siregarbakti@gmail.com
- siregarbakti@itsb.ac.id

Introduction

In today's digital age, **data** is one of the most valuable assets for businesses, governments, and researchers. **Data Science** is the key to unlocking insights from vast amounts of information, driving innovation, and enhancing decision-making.

What is Data Science?

Data Science is an interdisciplinary field that combines statistics, mathematics, computer science, and domain knowledge to extract insights and meaningful patterns from structured and unstructured data. It involves techniques such as data analysis, machine learning, artificial intelligence (AI), and big data processing to support decision-making and automation.

Data Science is an interdisciplinary field that integrates:

- **Statistics and Mathematics** for data analysis
- **Programming (Python/R)** for data manipulation and automation
- **Machine Learning and AI** for predictive analytics and automation
- **Domain Knowledge** to provide meaningful context and interpretation

The Role of Domain Knowledge

While technical skills such as coding and statistical modeling are essential, **domain knowledge** is equally crucial. It helps in:

- Identifying relevant data sources and features for analysis
- Understanding the real-world implications of data-driven insights
- Making informed business or research decisions based on data

Key Components of Data Science

Data Science is a multi-step process that transforms raw data into valuable insights and actionable solutions. To achieve this, several key components work together, ensuring that data is collected, processed, analyzed, and utilized effectively. Below are the core components of Data Science:

- **Data Collection :** Gathering data from various sources (databases, APIs, sensors, web scraping).
- **Data Cleaning & Preprocessing :** Removing noise, handling missing values, and transforming data for analysis.
- **Exploratory Data Analysis (EDA) :** Identifying trends, patterns, and relationships in the data.
- **Machine Learning & AI :** Building predictive models and automating decision-making.
- **Data Visualization :** Communicating insights using charts, graphs, and dashboards.
- **Big Data Processing :** Managing large datasets using distributed computing (Hadoop, Spark).
- **Deployment & Decision-Making :** Implementing models into real-world applications.

Data Science Project Ideas

Untuk memahami dan menguasai konsep-konsep dalam Data Science, mengerjakan proyek praktis adalah cara terbaik. Dalam dokumen ini, kita akan membahas beberapa ide proyek Data Science yang dikategorikan berdasarkan tingkat kesulitan, mulai dari pemula hingga tingkat lanjut.

Here are some project list topics for "Applied of Data Science Across Industries":

#	Project Name	Description	Tools/Techniques
Beginner-Level Projects			
1	Exploratory Data Analysis (EDA) on Titanic Dataset	Analyze survival rates based on passenger data.	`pandas`, `ggplot2`, `dplyr`
2	Movie Recommendation System	Build a recommendation system using collaborative filtering.	MovieLens, `recommenderlab`
3	Stock Market Analysis	Perform trend analysis on historical stock data.	`ggplot2`, `matplotlib`, `pandas`
4	Sentiment Analysis on Twitter Data	Classify tweets as positive, negative, or neutral using NLP.	`rtweet`, `tidytext`
5	Fake News Detection	Develop a Machine Learning model to differentiate fake news.	`TF-IDF`, `Naïve Bayes`
Intermediate-Level Projects			

#	Project Name	Description	Tools/Techniques
6	Customer Segmentation using Clustering	Group customers based on shopping behavior using K-Means or DBSCAN.	`cluster`, `sklearn`
7	Churn Prediction Model	Predict whether a customer will stop using a service.	`Random Forest`, `XGBoost`
8	Time Series Forecasting on Sales Data	Predict future sales using time-series models.	`ARIMA`, `Prophet`, `LSTM`
9	Credit Card Fraud Detection	Build a classification model to detect fraudulent transactions.	`scikit-learn`, `XGBoost`
10	Chatbot using NLP	Create an AI chatbot for conversations.	`spaCy`, `NLTK`, `BERT`
Advanced-Level Projects			
11	Autonomous Vehicle Lane Detection	Use Computer Vision and OpenCV to detect road lanes.	`OpenCV`, `Deep Learning`
12	AI-Powered Resume Parser	Build an NLP model to extract key information from resumes.	`spaCy`, `TensorFlow`
13	Image Caption Generator	Generate automatic image descriptions using CNN + LSTM.	`TensorFlow`, `PyTorch`
14	Speech Emotion Recognition	Analyze voice data and classify emotions.	`Librosa`, `Deep Learning`
15	Medical Diagnosis with Deep Learning	Detect diseases in medical images (e.g., X-rays).	`CNN`, `Keras`, `PyTorch`
16	Real-Time Object Detection using YOLO	Develop an object detection system for real-time applications.	`YOLO`, `OpenCV`, `Deep Learning`

Why Learn Data Science Programming?

Programming is the foundation of Data Science, enabling professionals to:

- Process and clean raw data efficiently

- Perform **exploratory data analysis (EDA)** to uncover trends and patterns
- Build **machine learning models** to make accurate predictions
- Create compelling **data visualizations** for effective storytelling

The two most widely used programming languages in Data Science are **Python and R**, each with distinct advantages.

Python vs R for Data Science

Python and R are the two most popular programming languages for **Data Science and Analytics**. Both have **strong libraries, statistical tools, and machine learning capabilities**, but they excel in different areas. The following video is a detailed comparison to help determine the best choice based on use cases.

Aspect	Python	R
Ease of Learning	Intuitive, beginner-friendly	More specialized, statistical focus
Primary Use Cases	AI, Machine Learning, Web Development	Statistical computing, data visualization
Key Libraries	`pandas`, `numpy`, `matplotlib`, `seaborn`, `plotly`,`scikit-learn`	`tidyverse`, `dplyr`, `ggplot2`, `plotly`, `caret`
Performance	Optimized for large-scale computation	Designed for in-depth statistical analysis
Community Support	Industry-wide adoption	Preferred in academia & research

Key Takeaways:

- **Python** is ideal for AI, Machine Learning, and large-scale data analysis.
- **R** excels in statistical analysis and data visualization.
- **Both rogramming languages** have their unique strengths, and mastering both can be highly beneficial.

Part I

Programming

Chapter 1

Basic Programming

Basic programming involves learning how to instruct computers using a language they can interpret. It focuses on understanding how data is represented, how logic is constructed, and how interaction occurs through input and output.

1.1 Variables and Data Types

A **variable** is a symbolic name that refers to stored information. Variables can hold various types of data, and choosing the right **data type** is crucial for efficient and correct operations.

Common Data Types:

Type	Description	Example
Integer	Whole numbers	`1`, `25`, `-100`
Float	Decimal numbers	`3.14`, `-0.001`
String	Text	`"Hello"`, `'World'`
Boolean	Logical values (True or False)	`True`, `False`

1.1.1 Python Code

```
# Variable assignment
name = "Alice"           # string
age = 28                 # integer
height = 1.68            # float
is_active = True         # boolean

# Print variable types
print(type(name))        # <class 'str'>
```

```
<class 'str'>
```

```
print(type(age))          # <class 'int'>
```

```
<class 'int'>
```

```
print(type(height))       # <class 'float'>
```

```
<class 'float'>
```

```
print(type(is_active))    # <class 'bool'>
```

```
<class 'bool'>
```

1.1.2 R Code

```
# Variable assignment
name <- "Alice"           # character
age <- 28                 # integer (numeric)
height <- 1.68            # float (numeric)
is_active <- TRUE         # boolean (logical)

# Check variable types
typeof(name)              # "character"
```

```
[1] "character"
```

```
typeof(age)               # "double"
```

```
[1] "double"
```

```
typeof(height)            # "double"
```

```
[1] "double"
```

```
typeof(is_active)         # "logical"
```

```
[1] "logical"
```

1.2 Operators

Operators are symbols used to perform operations on variables and values. They are essential for calculations, comparisons, and logical decisions in programming.

Common Operator Types:

Operator Type	Python Example	R Example	Use Case
Arithmetic	`+`, `-`, `*`, `/`	`+`, `-`, `*`, `/`	Mathematical operations
Comparison	`==`, `!=`, `>`, `<`	`==`, `!=`, `>`, `<`	Comparing values
Logical	`and`, `or`, `not`	`&`, `|`, `!`	Combining logical expressions
Assignment	`=`, `+=`, `-=`	`<-`, `->`, `=`	Assigning values

1.2.1 Python Code

```
a = 10
b = 3

# Arithmetic
print(a + b)   # 13
```

```
13
```

```
print(a / b)   # 3.333...
```

```
3.3333333333333335
```

```
# Comparison
print(a > b)   # True
```

```
True
```

```
print(a == b)  # False
```

```
False
```

```
# Logical
x = True
y = False
print(x and y)  # False
```

```
False
```

```
print(x or y)   # True
```

```
True
```

1.2.2 R Code

```
a <- 10
b <- 3

# Arithmetic
a + b    # 13
```

```
[1] 13
```

```
a / b    # 3.333...
```

```
[1] 3.333333
```

```
# Comparison
a > b    # TRUE
```

```
[1] TRUE
```

```
a == b   # FALSE
```

```
[1] FALSE
```

```r
# Logical
x <- TRUE
y <- FALSE
x & y    # FALSE
```

```
[1] FALSE
```

```r
x | y    # TRUE
```

```
[1] TRUE
```

1.3 Input and Output

- **Input**: Receives data from users or external sources.
- **Output**: Displays information to the screen or interface.

1.3.1 Python Code

`input()` may cause an error when rendering this document. Use placeholder values if rendering non-interactively.

```python
try:
    name = input("Enter your name: ")
    age = int(input("Enter your age: "))
except EOFError:
    name = "TestUser"
    age = 99

print(f"Hello {name}, you are {age} years old.")
```

1.3.2 R Code

```r
# Getting user input and displaying output
name <- readline(prompt = "Enter your name: ")
age <- as.integer(readline(prompt = "Enter your age: "))
cat("Hello", name, ", you are", age, "years old.\n")
```

Chapter 2

Syntax and Control Flow

Control flow statements determine the order in which code is executed, allowing programs to make logical decisions, perform repetitive tasks, and handle errors efficiently. To learn more, consider watching the following video:

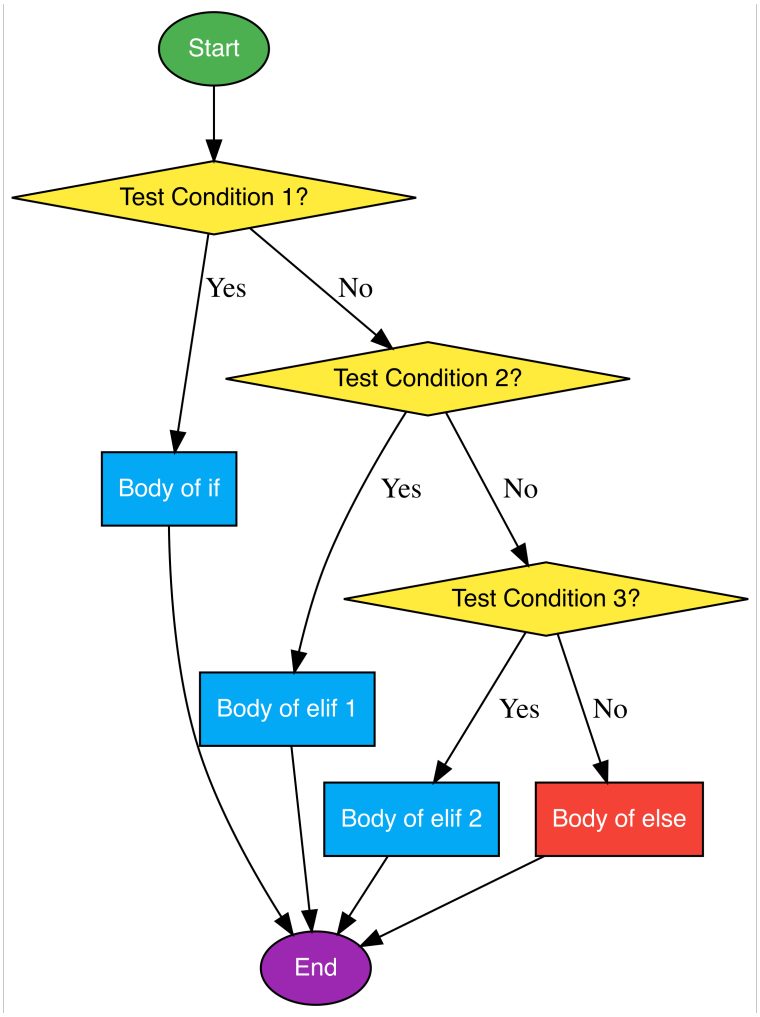
There are three main types of control flow statements:

- Conditional Statements → Enable decision-making in a program (e.g., if, if-else, if-elif-else).
- Loops → Facilitate repetition of actions (e.g., for, while, with control keywords like break and continue).
- Error Handling → Ensure smooth program execution by managing unexpected errors (e.g., try-except in Python, tryCatch in R).

2.1 Conditional Statements

Conditional statements let a program **execute different code** depending on whether a condition is **true or false**.

- **`if` statement** → Executes code only if a condition is **true**.
- **`if-else` statement** → Executes one block if **true**, another if **false**.
- **`if-elif-else` statement** → Checks multiple conditions.



2.1.1 Simple Example

Check if a number is positive, negative, or zero.

Python Code

```
x = 10

if x > 0:
    print("Positive number")
elif x == 0:
    print("Zero")
else:
    print("Negative number")
```

```
Positive number
```

R Code

```r
x <- 10

if (x > 0) {
  print("Positive number")
} else if (x == 0) {
  print("Zero")
} else {
  print("Negative number")
}
```

```
[1] "Positive number"
```

2.1.2 Medium Example

Check if a number is even or odd, with an additional condition.

Python Code

```python
try:
    x = int(input("Enter a number: "))

    if x % 2 == 0:
        print(f"{x} is even.")
        if x % 4 == 0:
            print(f"{x} is also a multiple of 4.")
    else:
        print(f"{x} is odd.")

except ValueError:
    print("Invalid input! Please enter a valid integer.")
```

R Code

```r
x <- as.integer(readline("Enter a number: "))

if (x %% 2 == 0) {
  print(paste(x, "is even."))
  if (x %% 4 == 0) {
    print(paste(x, "is also a multiple of 4."))
  }
} else {
  print(paste(x, "is odd."))
}
```

2.1.3 Complex Example

Categorizing a person's age group.

Python Code

```python
age = int(input("Enter your age: "))

if age < 0:
    print("Invalid age")
elif age <= 12:
    print("You are a child.")
elif age <= 19:
    print("You are a teenager.")
elif age <= 59:
    print("You are an adult.")
else:
    print("You are a senior.")
```

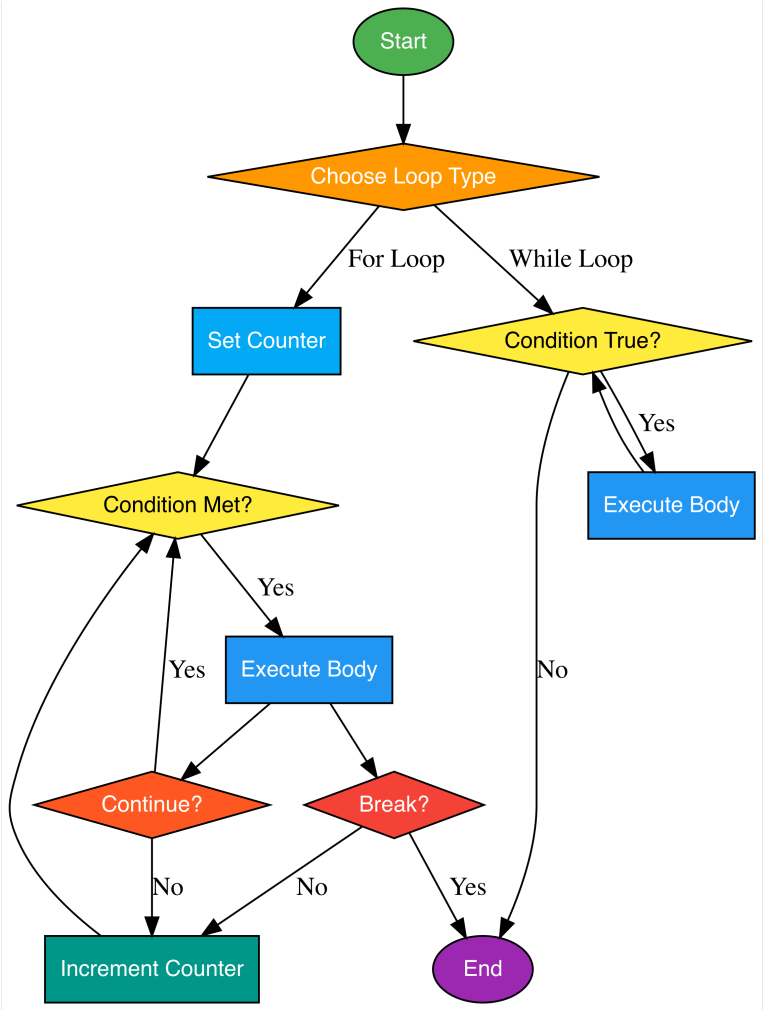
R Code

```r
age <- as.integer(readline("Enter your age: "))

if (age < 0) {
  print("Invalid age")
} else if (age <= 12) {
  print("You are a child.")
} else if (age <= 19) {
  print("You are a teenager.")
} else if (age <= 59) {
  print("You are an adult.")
} else {
  print("You are a senior.")
}
```

2.2 Loops

Loops allow programs to repeat actions multiple times.

- For Loop → Used when the number of iterations is known. The counter is set, the condition is checked, the code executes, then the counter increments until the condition is no longer met.
- While Loop → Used when looping should continue as long as the condition is true. If the condition remains valid, the code executes and is checked again until the condition becomes false.
- Break → Stops the loop early, immediately exiting the loop without completing all iterations.
- Continue → Skips the current iteration without stopping the loop, returning directly to the condition check for the next iteration. The loop stops when the condition is no longer met or a break statement is used. The loop continues if the condition remains true unless a break occurs.



2.2.1 Simple Example

Print numbers from 1 to 5 using a for loop.

Python Code

```
for i in range(1, 6):
    print(i)
```

```
1
2
3
4
5
```

R Code

```r
for (i in 1:5) {
  print(i)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

2.2.2 Medium Example

Using a while loop to print numbers up to a limit.

Python Code

```python
x = 1
while x <= 5:
    print(x)
    x += 1
```

```
1
2
3
4
5
```

R Code

```r
x <- 1
while (x <= 5) {
  print(x)
  x <- x + 1
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

2.2.3 Complex Example

Using break and continue to modify loop behavior.

Python Code

```
for i in range(1, 11):
    if i == 5:
        continue  # Skip number 5
    if i == 8:
        break     # Stop when i = 8
    print(i)
```

```
1
2
3
4
6
7
```

R Code

```
for (i in 1:10) {
  if (i == 5) {
    next          # Skip number 5
  }
  if (i == 8) {
    break         # Stop when i = 8
  }
  print(i)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 6
[1] 7
```

2.3 Error Handling

Error handling allows a program to recover from unexpected issues rather than terminating abruptly.

- try-except (Python) → Captures errors and ensures the program continues running.
- tryCatch (R) → Provides similar functionality in R, allowing controlled error management.

2.3.1 Simple Example

Handling incorrect date formats during user input

Python Code

```python
from datetime import datetime

try:
    date_str = input("Enter date (YYYY-MM-DD): ")  # Input may have wrong format
    date_obj = datetime.strptime(date_str, "%Y-%m-%d")
    print(f"Entered date: {date_obj}")
except ValueError:
    print("Error: Date format must be YYYY-MM-DD!")
```

R Code

```r
safe_date <- function() {
  date_str <- readline("Enter date (YYYY-MM-DD): ")
  tryCatch({
    date_obj <- as.Date(date_str, format="%Y-%m-%d")
    if (is.na(date_obj)) stop("Incorrect date format!")
    print(paste("Entered date:", date_obj))
  }, error = function(e) {
    print(paste("Error:", e$message))
  })
}

safe_date()
```

```
Enter date (YYYY-MM-DD):
[1] "Error: Incorrect date format!"
```

2.3.2 Medium Example

Handling missing values and invalid dates in datasets.

Python Code

```python
# inport library
import pandas as pd

# dataset example (this is dictionary type)
data = {"event": ["A", "B", "C"], "date": ["2024-01-15", "2024-15-10", None]}
df = pd.DataFrame(data)

# error handaling program
try:
    df["date"] = pd.to_datetime(df["date"], errors="coerce")  # dates to NaT
    df.dropna(subset=["date"], inplace=True)  # Remove rows with invalid dates
    print(df)
except Exception as e:
    print(f"Error processing dates: {e}")
```

R Code

```r
library(dplyr)
library(lubridate)

# dataset example (this is dictionary type)
data <- data.frame(event = c("A", "B", "C"),
                   date = c("2024-01-15", "2024-15-10", NA))

# error handaling program
safe_process_dates <- function(df) {
  tryCatch({
    df <- df %>%
      mutate(date = ymd(date)) %>%  # Convert dates
      filter(!is.na(date))          # Remove invalid dates

    print(df)
  }, error = function(e) {
    print(paste("Error processing dates:", e$message))
  })
}

safe_process_dates(data)
```

2.3.3 Complex Example

This Python code replicates the R functionality while improving date handling.

Python Code

```python
import pandas as pd
from dateutil import parser
import warnings

# Example dataset with various incorrect date formats
data = pd.DataFrame({
    "event": ["A", "B", "C", "D", "E"],
    "date": ["2024-01-15", "2024-15-10", None, "15-03-2024", "2024/04/05"]
})

def safe_process_dates(df):
    def parse_date(date_str):
        try:
            return parser.parse(date_str, dayfirst=True).strftime("%d-%m-%Y")
        except (ValueError, TypeError):
            return None

    # Apply date parsing
    df["parsed_date"] = df["date"].apply(parse_date)
```

```
    # Find invalid dates
    invalid_dates = df[df["parsed_date"].isna()]["date"].dropna().tolist()

    if invalid_dates:
        warnings.warn(f"Some dates are invalid: {', '.join(invalid_dates)}")

    # Filter out invalid dates
    df = df.dropna(subset=["parsed_date"])

    print(df)

safe_process_dates(data)
```

```
  event        date parsed_date
0     A  2024-01-15  15-01-2024
1     B  2024-15-10  15-10-2024
3     D  15-03-2024  15-03-2024
4     E  2024/04/05  04-05-2024
```

R Code

```
library(dplyr)
library(lubridate)

# Example dataset with various incorrect date formats
data <- data.frame(event = c("A", "B", "C", "D", "E"),
                   date = c("2024-01-15", "2024-15-10", NA,
                            "15-03-2024", "2024/04/05"))

# Function to handle errors in date conversion
safe_process_dates <- function(df) {
  tryCatch({
    df <- df %>%
      mutate(
        parsed_date = parse_date_time(date,
                                      orders = c("ymd", "dmy", "mdy", "Y/m/d"),
                                      quiet = TRUE)
      ) %>%
      filter(!is.na(parsed_date)) %>%   # Remove dates that failed to convert
      mutate(formatted_date = format(parsed_date, "%d-%m-%Y")) # Reformat dates

    # Check if there are any invalid dates
    invalid_dates <- df$date[is.na(df$parsed_date)]
    if (length(invalid_dates) > 0) {
      warning("Some dates are invalid: ", paste(invalid_dates, collapse = ", "))
    }

    print(df)
```

```
  }, error = function(e) {
    print(paste("Error processing dates:", e$message))
  })
}

safe_process_dates(data)
```

```
  event       date parsed_date formatted_date
1     A 2024-01-15  2024-01-15     15-01-2024
2     D 15-03-2024  2024-03-15     15-03-2024
3     E 2024/04/05  2024-04-05     05-04-2024
```

Explanation:

- `dateutil.parser.parse()` is used for flexible date recognition.
- Handles multiple date formats including `YYYY-MM-DD`, `DD-MM-YYYY`, `YYYY/MM/DD`, etc.
- Removes invalid dates and displays warnings for them.
- Formats valid dates to `YYYY-MM-DD` for consistency.

2.4 Best Practices

In this section you will learn the **Best Practices** about Syntax & Control Flow in Python and R:

2.4.1 Readability & Formatting

- **Follow PEP 8** for clean and readable code.
- **Use consistent indentation (4 spaces per level).**
- **Keep line length 79 characters.**
- **Use meaningful variable and function names.**

Python Code

```
#  Good: Readable and follows PEP 8
def calculate_total(price, tax_rate):
    "Calculate the total price after tax."
    total = price + (price * tax_rate)
    return total
```

```
#  Bad: Poor formatting and unclear naming
def calc(p, t): return p+(p*t) # One-liner (not recommended)
```

R Code

```
#  Good: Readable and follows conventions
total_price <- function(price, tax_rate) {
```

```r
  "Calculate the total price after tax."
  total <- price + (price * tax_rate)
  return(total)
}
```

```r
#  Bad: Poor formatting and unclear naming
total <- function(p, t) { p + (p * t) } # One-liner (not recommended)
```

2.4.2 Efficient Control Flow

- **Use `if-elif-else` correctly** to avoid redundant checks.
- **Avoid deep nesting**; use guard clauses to improve readability.

Python Code

```python
#  Good: Uses guard clause to return early
def check_access(user):
    if not user.is_authenticated:
        return "Access Denied"

    if user.is_admin:
        return "Access Granted: Admin"

    return "Access Granted: User"
```

```python
#  Bad: Deeply nested structure
def check_access(user):
    if user.is_authenticated:
        if user.is_admin:
            return "Access Granted: Admin"
        else:
            return "Access Granted: User"
    else:
        return "Access Denied"
```

R Code

```r
#  Good: Uses early return
check_access <- function(user) {
  if (!user$authenticated) {
    return("Access Denied")
  }

  if (user$admin) {
    return("Access Granted: Admin")
  }

  return("Access Granted: User")
```

```
}
```

2.4.3 Looping Best Practices

- **Use list comprehensions for simple loops.**
- **Use `enumerate()` instead of manually managing indexes.**
- **Use `zip()` for iterating over multiple lists simultaneously.**

Python Code

```
#  Good: List comprehension for efficiency
squares = [x**2 for x in range(10) if x % 2 == 0]

#  Good: Using enumerate
names = ["Alice", "Bob", "Charlie"]
for index, name in enumerate(names, start=1):
    print(f"{index}: {name}")

#  Good: Using zip()
keys = ["name", "age", "city"]
values = ["Alice", 25, "New York"]
person = dict(zip(keys, values))
```

```
#  Good: Vectorized operation
squares <- (0:9)^2

#  Good: Using sapply
names <- c("Alice", "Bob", "Charlie")
print(sapply(seq_along(names), function(i) paste(i, names[i])))
```

```
[1] "1 Alice"   "2 Bob"     "3 Charlie"
```

2.4.4 Common Mistakes in Loops

- **Don't modify lists while iterating** (use a copy instead).
- **Use `break` and `continue` sparingly** to maintain readability.

Python Code

```
#  Good: Iterating over a copy when modifying a list
numbers = [1, 2, 3, 4, 5]
for num in numbers[:]:  # Copy of the list
    if num % 2 == 0:
        numbers.remove(num)
```

```
#  Bad: Modifying a list while iterating (can cause unexpected behavior)
for num in numbers:
```

```
    if num % 2 == 0:
        numbers.remove(num)
```

R Code

```
#  Good: Using which to filter elements
numbers <- c(1, 2, 3, 4, 5)
numbers <- numbers[numbers %% 2 != 0]
```

2.4.5 Cleaner Conditionals

Use `match-case` instead of long `if-elif` chains when checking specific values **Python 3.10+**.

Python

```
#  Good: Using match-case
def get_status(code):
    match code:
        case 200:
            return "OK"
        case 404:
            return "Not Found"
        case 500:
            return "Internal Server Error"
        case _:
            return "Unknown Status"
```

```
#  Bad: Using multiple if-elif
def get_status(code):
    if code == 200:
        return "OK"
    elif code == 404:
        return "Not Found"
    elif code == 500:
        return "Internal Server Error"
    else:
        return "Unknown Status"
```

R Code

```
#  Good: Using switch
get_status <- function(code) {
  switch(as.character(code),
         "200" = "OK",
         "404" = "Not Found",
         "500" = "Internal Server Error",
         "Unknown Status")
}
```

2.4.6 Error Handling

- **Catch only specific exceptions** instead of using a general `except` clause.
- **Use `finally` for cleanup operations** (e.g., closing files, releasing resources).

Python Code

```python
#  Good: Handling specific exceptions
try:
    number = int(input("Enter a number: "))
    result = 10 / number
except ValueError:
    print("Invalid input! Please enter a number.")
except ZeroDivisionError:
    print("Cannot divide by zero.")
finally:
    print("Execution complete.")
```

R Code

```r
#  Good: Handling specific exceptions
safe_divide <- function(a, b) {
  tryCatch({
    result <- a / b
    return(result)
  }, warning = function(w) {
    message("Warning: ", w$message)
  }, error = function(e) {
    message("Error: Cannot divide by zero.")
  })
}
```

2.4.7 Resource Management

Use `with` statements when working with files to ensure proper cleanup.

Python Code

```python
#  Good: Using with statement (auto-closes file)
with open("data.txt", "r") as file:
    content = file.read()
```

```python
#  Bad: Forgetting to close the file
file = open("data.txt", "r")
content = file.read()
file.close()
```

R Code

```r
#  Good: Using on.exit() to clean up
read_data <- function(file) {
  con <- file(file, "r")
  on.exit(close(con))
  data <- readLines(con)
  return(data)
}
```

2.4.8 Mutable Default Arguments

Use immutable types (e.g., `None`) as default arguments instead of mutable ones.

Python Code

```python
#  Good: Using None to avoid unintended behavior
def add_item(item, items=None):
    if items is None:
        items = []
    items.append(item)
    return items
```

R Code

```r
#  Good: Using NULL to avoid unintended behavior
add_item <- function(item, items = NULL) {
  if (is.null(items)) {
    items <- list()
  }
  items <- append(items, item)
  return(items)
}
```

2.4.9 Using Generators for

Use `yield` for efficient memory usage when handling large datasets.

Python Code

```python
#  Good: Using generator function
def count_up_to(n):
    count = 1
    while count <= n:
        yield count
        count += 1

for num in count_up_to(5):
    print(num)
```

R Code

```r
#  Good: Using closures to generate a sequence
count_up_to <- function(n) {
  count <- 1
  function() {
    if (count <= n) {
      result <- count
      count <<- count + 1
      return(result)
    } else {
      return(NULL)
    }
  }
}

counter <- count_up_to(5)
while (!is.null(value <- counter())) {
  print(value)
}
```

By following these best practices, you can write **clean, efficient, and maintainable** Python and R code.

2.5 Practicum

Independent Practice: Conditional Statements and Loops in Python & R

2.5.1 Objective

1. Understand and implement **conditional statements** (`if`, `if-else`, `if-elif-else`).
2. Apply **loops** (`for loop`, `while loop`, `break`, `continue`) to analyze a dataset.

Use the following **dummy dataset**:

ID	Name	Age	Salary	Position	Performance
1	Bagas	25	5000	Staff	Good
2	Joan	30	7000	Supervisor	Very Good
3	Alya	27	6500	Staff	Average
4	Dwi	35	10000	Manager	Good
5	Nabil	40	12000	Director	Very Good

2.5.2 Conditional Statements

Determine **bonus levels** based on employee **performance**:

- **Very Good** → 20% of salary
- **Good** → 10% of salary

- **Average** → 5% of salary

Your Task:

- Write a program in **Python and R** to calculate each employee's bonus.
- Display the output in this format:
  `"Name: Bagas, Bonus: 500"`

2.5.3 Loops (For & While)

1. Use a **for loop** to list employees with a salary greater than **6000**.

Expected Output:

```
Name: Joan, Salary: 7000
Name: Alya, Salary: 6500
Name: Dwi, Salary: 10000
Name: Nabil, Salary: 12000
```

2. Use a **while loop** to display employees until a **"Manager"** is found.

Expected Output:

```
Name: Bagas, Position: Staff
Name: Joan, Position: Supervisor
Name: Alya, Position: Staff
Name: Dwi, Position: Manager (Stop here)
```

3. Use **break** to stop the loop when an employee with a salary above **10,000** is found.

Expected Output:

```
Name: Bagas, Salary: 5000
Name: Joan, Salary: 7000
Name: Alya, Salary: 6500
Name: Dwi, Salary: 10000
(Stopped because Nabil has a salary above 10,000)
```

4. Use **continue** to **skip** employees with **"Average"** performance.

Expected Output:

```
Name: Bagas, Performance: Good
Name: Joan, Performance: Very Good
Name: Dwi, Performance: Good
Name: Nabil, Performance: Very Good
(Alya is skipped because the performance is "Average")
```

Submission Guidelines:

1. Submit your **Python and R** code using your `Google Colab` and `Rpubs`.
2. Ensure the output is displayed correctly.
3. Add comments in the code to explain your logic.

Chapter 3

Functions and Loops

3.1 Introduction

In programming, we often perform the same tasks repeatedly. **Functions** and **Loops** help us write cleaner, shorter, and more efficient code.

- **Function** is a block of code that can be called anytime to perform a specific task.
- **Loop** is used to run the same code repeatedly without rewriting it.

3.2 What Is a Function?

A function is a block of code designed to perform a specific task. Using functions helps us avoid redundant code.

This visual representation helps illustrate how functions work systematically. The label "Function Machine" on the machine reinforces that it applies a specific rule to transform the input into an output. The function in the image is:

$$f(x) = x + 3$$

This means that any number inputted into the machine will have **3 added to it** before being output.

3.2.1 Function in $ax + b$

This function takes three numbers as inputs and returns their calculation.

Python Code

```
# Function to multiply 'a' with 'x' and add 'b'
def function1(a, x, b):
    return a * x + b
```

```
# Example usage
print(function1(2, 3, 4))  # Output: (2 * 3) + 4 = 10
```

```
10
```

R Code

```
# Function to multiply 'a' with 'x' and add 'b'
function1 <- function(a, x, b) {
  return(a * x + b)
}

# Example usage
print(function1(2, 3, 4))  # Output: (2 * 3) + 4 = 10
```

```
[1] 10
```

3.2.2 Value Comparator

This function analyzes two datasets by calculating their mean, median, and standard deviation, useful in data analysis.

Python Code

```
import statistics
from tabulate import tabulate

# Function to compare two datasets
def compare_data(group1, group2):
    return {
        "group1": {
            "mean": statistics.mean(group1),
            "median": statistics.median(group1),
            "std_dev": statistics.stdev(group1)
        },
        "group2": {
            "mean": statistics.mean(group2),
            "median": statistics.median(group2),
            "std_dev": statistics.stdev(group2)
        }
    }

# Sample datasets
data1 = [10, 20, 30, 40, 50]
data2 = [15, 25, 35, 45, 55]

# Get results
results = compare_data(data1, data2)
```

```
# Convert results to a table format
table = [
    ["Metric", "Group 1", "Group 2"],
    ["Mean", results["group1"]["mean"], results["group2"]["mean"]],
    ["Median", results["group1"]["median"], results["group2"]["median"]],
    ["Standard Deviation", results["group1"]["std_dev"], results["group2"]["std_dev"]]
]

# Print table
print(tabulate(table, headers="firstrow", tablefmt="grid"))
```

```
+--------------------+-----------+-----------+
| Metric             |   Group 1 |   Group 2 |
+====================+===========+===========+
| Mean               |   30      |   35      |
+--------------------+-----------+-----------+
| Median             |   30      |   35      |
+--------------------+-----------+-----------+
| Standard Deviation |   15.8114 |   15.8114 |
+--------------------+-----------+-----------+
```

R Code

```
# Load library
library(knitr)

# Function to compare two datasets
compare_data <- function(group1, group2) {
  data.frame(
    Statistic = c("Mean", "Median", "Std Dev"),
    Group1 = round(c(mean(group1), median(group1), sd(group1)), 2),
    Group2 = round(c(mean(group2), median(group2), sd(group2)), 2)
  )
}

# Sample data
data1 <- c(10, 20, 30, 40, 50)
data2 <- c(15, 25, 35, 45, 55)

# Print as formatted table
kable(compare_data(data1, data2))
```

Statistic	Group1	Group2
Mean	30.00	35.00
Median	30.00	35.00
Std Dev	15.81	15.81

Functions save time by allowing code reuse, improve program organization and read-

ability, and make debugging and future development easier.

3.2.3 Geometric Properties

In the field of computational geometry, functions are essential for converting mathematical expressions into executable code. For example, the formulas for calculating the area and perimeter of various two-dimensional shapes can be implemented as separate functions. This approach makes the development process more efficient and easier to manage. The following sections explain in detail how these geometric formulas are coded, using Python and R as examples.

Shape	Area Formula (A)	Perimeter Formula (P)	Variables Description
Triangle	$A = \frac{1}{2}(b \times h)$	$P = a + b + c$	b = base, h = height, a, b, c = sides
Rectangle	$A = l \times b$	$P = 2(l + b)$	l = length, b = breadth
Square	$A = s \times s$	$P = 4 \times s$	s = side
Circle	$A = \pi r^2$	$P = 2\pi r$	r = radius, $\pi = 3.14$ or $\frac{22}{7}$
Ellipse	$A = \pi \times a \times b$	$P = \pi(a + b)$	a = semi-major axis, b = semi-minor axis
Parallelogram	$A = b \times h$	$P = 2(a + b)$	b = base, h = height, a, b = lengths of opposite sides
Rhombus	$A = \frac{1}{2}(d_1 \times d_2)$	$P = 4 \times a$	d_1, d_2 = diagonals, a = side
Trapezium	$A = \frac{1}{2}(a + b) \times h$	Sum of all sides	a, b = lengths of parallel sides, h = height

With the formulas provided above, you can create functions that calculate the area and perimeter for different shapes. This not only makes your code modular and easier to maintain but also enables you to test individual pieces of logic in isolation. This example below in Python and R that demonstrate how to implement functions for these calculations.

Python Code

```python
import math

# Function to calculate area and perimeter for multiple shapes
def calculate_area_perimeter(shape, **kwargs):
    if shape == "triangle":
        base = kwargs.get("base")
        height = kwargs.get("height")
        side_a = kwargs.get("side_a")
        side_b = kwargs.get("side_b")
        side_c = kwargs.get("side_c")
```

```
        area = 0.5 * base * height
        perimeter = side_a + side_b + side_c
    elif shape == "rectangle":
        length = kwargs.get("length")
        breadth = kwargs.get("breadth")
        area = length * breadth
        perimeter = 2 * (length + breadth)
    elif shape == "square":
        side = kwargs.get("side")
        area = side ** 2
        perimeter = 4 * side
    elif shape == "circle":
        radius = kwargs.get("radius")
        area = math.pi * radius ** 2
        perimeter = 2 * math.pi * radius
    elif shape == "ellipse":
        a = kwargs.get("a")
        b = kwargs.get("b")
        area = math.pi * a * b
        perimeter = math.pi * (a + b)
    elif shape == "parallelogram":
        base = kwargs.get("base")
        height = kwargs.get("height")
        side_a = kwargs.get("side_a")
        side_b = kwargs.get("side_b")
        area = base * height
        perimeter = 2 * (side_a + side_b)
    elif shape == "rhombus":
        d1 = kwargs.get("d1")
        d2 = kwargs.get("d2")
        side = kwargs.get("side")
        area = 0.5 * d1 * d2
        perimeter = 4 * side
    elif shape == "trapezium":
        a = kwargs.get("a")
        b = kwargs.get("b")
        height = kwargs.get("height")
        side_a = kwargs.get("side_a")
        side_b = kwargs.get("side_b")
        area = 0.5 * (a + b) * height
        perimeter = a + b + side_a + side_b
    else:
        return "Invalid shape. Choose a valid 2D shape."

    return {"area": area, "perimeter": perimeter}
```

```
# Example usage
result = calculate_area_perimeter("triangle",
                                  base=6,
```

```
                                    height=4,
                                    side_a=5,
                                    side_b=6,
                                    side_c=7)
print("Triangle-Area & Perimeter:", result["area"], "and", result["perimeter"])
```

```
Triangle-Area & Perimeter: 12.0 and 18
```

R Code

```
# Function to calculate area and perimeter for multiple shapes
calculate_area_perimeter <- function(shape, ...) {
  args <- list(...)

  if (shape == "triangle") {
    base <- args$base
    height <- args$height
    side_a <- args$side_a
    side_b <- args$side_b
    side_c <- args$side_c
    area <- 0.5 * base * height
    perimeter <- side_a + side_b + side_c
  } else if (shape == "rectangle") {
    length <- args$length
    breadth <- args$breadth
    area <- length * breadth
    perimeter <- 2 * (length + breadth)
  } else if (shape == "square") {
    side <- args$side
    area <- side^2
    perimeter <- 4 * side
  } else if (shape == "circle") {
    radius <- args$radius
    area <- pi * radius^2
    perimeter <- 2 * pi * radius
  } else if (shape == "ellipse") {
    a <- args$a
    b <- args$b
    area <- pi * a * b
    perimeter <- pi * (a + b)
  } else if (shape == "parallelogram") {
    base <- args$base
    height <- args$height
    side_a <- args$side_a
    side_b <- args$side_b
    area <- base * height
    perimeter <- 2 * (side_a + side_b)
  } else if (shape == "rhombus") {
    d1 <- args$d1
```

```
    d2 <- args$d2
    side <- args$side
    area <- 0.5 * d1 * d2
    perimeter <- 4 * side
  } else if (shape == "trapezium") {
    a <- args$a
    b <- args$b
    height <- args$height
    side_a <- args$side_a
    side_b <- args$side_b
    area <- 0.5 * (a + b) * height
    perimeter <- a + b + side_a + side_b
  } else {
    stop("Invalid shape. Choose a valid 2D shape.")
  }

  return(list(area = area, perimeter = perimeter))
}
```

```
# Example usage
result <- calculate_area_perimeter("triangle",
                                   base = 6,
                                   height = 4,
                                   side_a = 5,
                                   side_b = 6,
                                   side_c = 7)
cat("Triangle - Area & Perimeter:", result$area, "and", result$perimeter, "\n")
```

```
Triangle - Area & Perimeter: 12 and 18
```

3.3 What Is a Loop?

Loops allow us to execute the same code multiple times without rewriting it. Loops allow us to perform repetitive calculations for mathematical analysis and data processing. Types of Loops:

- **For Loop** – Used when the number of repetitions is known.
- **While Loop** – Used when repetitions depend on a condition.

3.3.1 Fibonacci Sequence

The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding ones:

$$F(n) = F(n-1) + F(n-2)$$

Example: `$0,1,1,2,3,5,8,13,21,\dots$`

Python Code

```python
def fibonacci(n):
    fib_series = [0, 1]
    for i in range(2, n):
        fib_series.append(fib_series[-1] + fib_series[-2])
    return fib_series

print(fibonacci(10))  # Output: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

3.3.1.1 R Code

```r
fibonacci <- function(n) {
  fib_series <- c(0, 1)
  for (i in 3:n) {
    fib_series <- c(fib_series, fib_series[i-1] + fib_series[i-2])
  }
  return(fib_series)
}

print(fibonacci(10))  # Output: 0 1 1 2 3 5 8 13 21 34
```

```
 [1]  0  1  1  2  3  5  8 13 21 34
```

3.3.2 Arithmetic & Geometric Sequences

This function generates a sequence based on the type specified: either an arithmetic sequence or a geometric sequence. For an arithmetic sequence, each term is obtained by adding a constant difference to the previous term. For a geometric sequence, each term is obtained by multiplying the previous term by a constant ratio.

Python Code

```python
def generate_sequence(seq_type, n, a, d=None, r=None):
    """
    Generate an arithmetic or geometric sequence.

    Parameters:
        seq_type (str): Type of sequence - "arithmetic" or "geometric".
        n (int): The number of terms in the sequence.
        a (numeric): The first term of the sequence.
        d (numeric, optional): The common difference (required for arithmetic).
        r (numeric, optional): The common ratio (required for geometric).

    Returns:
        list: A list containing the generated sequence.
    """
    sequence = []
```

```
    if seq_type.lower() == "arithmetic":
        if d is None:
            raise ValueError("'d' must be provided for an arithmetic sequence")
        for i in range(n):
            sequence.append(a + i * d)
    elif seq_type.lower() == "geometric":
        if r is None:
            raise ValueError("'r' must be provided for a geometric sequence")
        for i in range(n):
            sequence.append(a * (r ** i))
    else:
        raise ValueError("seq_type must be either 'arithmetic' or 'geometric'")
    return sequence

# Example usage:
print(generate_sequence("arithmetic", 10, 1, d=2))
```

```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

```
print(generate_sequence("geometric", 10, 1, r=3))
```

```
[1, 3, 9, 27, 81, 243, 729, 2187, 6561, 19683]
```

R Code

```
generate_sequence <- function(seq_type, n, a, d = NULL, r = NULL) {
  #' Generate an arithmetic or geometric sequence.
  #'
  #' @param seq_type specifying the type of sequence:"arithmetic"/"geometric".
  #' @param n The number of terms in the sequence.
  #' @param a The first term of the sequence.
  #' @param d The common difference (required for arithmetic sequences).
  #' @param r The common ratio (required for geometric sequences).
  #'
  #' @return A numeric vector containing the generated sequence.

  sequence <- numeric(n)
  if (tolower(seq_type) == "arithmetic") {
    if (is.null(d)) stop("'d' must be provided for an arithmetic sequence.")
    for (i in 1:n) {
      sequence[i] <- a + (i - 1) * d
    }
  } else if (tolower(seq_type) == "geometric") {
    if (is.null(r)) stop("'r' must be provided for a geometric sequence.")
    for (i in 1:n) {
      sequence[i] <- a * (r^(i - 1))
    }
  } else {
    stop("seq_type must be either 'arithmetic' or 'geometric'")
  }
```

```
  return(sequence)
}

# Example usage:
print(generate_sequence("arithmetic", 10, 1, d = 2))
```

```
 [1]  1  3  5  7  9 11 13 15 17 19
```

```
print(generate_sequence("geometric", 10, 1, r = 3))
```

```
 [1]     1     3     9    27    81   243   729  2187  6561 19683
```

3.3.3 Simple Linear Regression

Linear regression is used to find the relationship between an independent variable X and a dependent variable Y:

$$Y = aX + b$$

where:

- a is the slope
- b is the intercept

Python Code

```
import numpy as np

# Data (X: study hours, Y: exam scores)
X = np.array([1, 2, 3, 4, 5])
Y = np.array([2, 4, 5, 4, 5])

# Calculate slope (a) and intercept (b)
n = len(X)
sum_x, sum_y = sum(X), sum(Y)
sum_xy = sum(X * Y)
sum_x2 = sum(X ** 2)

a = (n * sum_xy - sum_x * sum_y) / (n * sum_x2 - sum_x ** 2)
b = (sum_y - a * sum_x) / n

print(f"Linear Regression: Y = {a:.2f}X + {b:.2f}")
```

```
Linear Regression: Y = 0.60X + 2.20
```

R Code

```
# Data
X <- c(1, 2, 3, 4, 5)
Y <- c(2, 4, 5, 4, 5)
```

```r
# Calculate slope (a) and intercept (b)
n <- length(X)
sum_x <- sum(X)
sum_y <- sum(Y)
sum_xy <- sum(X * Y)
sum_x2 <- sum(X^2)

a <- (n * sum_xy - sum_x * sum_y) / (n * sum_x2 - sum_x^2)
b <- (sum_y - a * sum_x) / n

print(paste("Linear Regression: Y =", round(a, 2), "X +", round(b, 2)))
```

```
[1] "Linear Regression: Y = 0.6 X + 2.2"
```

Functions and loops help us create simpler and more efficient code. By understanding these two concepts, we can write better and more readable programs.

3.4 Applied of Functions and Loops

Let's apply these **Functions and Loops** to real-world data science tasks:

3.4.1 Creating a Dataset

Python Code

```python
import pandas as pd
import random

def create_employee_dataset(num_employees):
    positions = {
        "Staff": (3000, 5000, 1, 5),
        "Supervisor": (5000, 8000, 5, 10),
        "Manager": (8000, 12000, 10, 15),
        "Director": (12000, 15000, 15, 25)
    }

    departments = ["Finance", "HR", "IT", "Marketing", "Operations", "Sales"]
    locations = ["New York", "Los Angeles", "Chicago", "Houston", "Phoenix"]

    data = {
        "ID_Number": [],
        "Position": [],
        "Salary": [],
        "Age": [],
        "Experience": [],
        "Department": [],
        "Location": []
    }
```

```
    for _ in range(num_employees):
        id_number = random.randint(10000, 99999)
        position = random.choice(list(positions.keys()))
        salary = random.randint(positions[position][0],
                 positions[position][1])
        experience = random.randint(positions[position][2],
                     positions[position][3])
        age = experience + random.randint(22, 35)  # aligns with experience
        department = random.choice(departments)
        location = random.choice(locations)

        data["ID_Number"].append(id_number)
        data["Position"].append(position)
        data["Salary"].append(salary)
        data["Age"].append(age)
        data["Experience"].append(experience)
        data["Department"].append(department)
        data["Location"].append(location)

    return pd.DataFrame(data)

# Create the employee dataset
df = create_employee_dataset(20)
print(df)
```

	ID_Number	Position	Salary	Age	Experience	Department	Location
0	83290	Manager	8751	37	10	Finance	Phoenix
1	92436	Supervisor	5143	39	6	Marketing	Houston
2	94826	Staff	3510	34	2	IT	New York
3	80489	Staff	3024	29	5	HR	New York
4	90603	Manager	10444	41	11	Sales	New York
5	94417	Supervisor	7624	31	5	HR	Phoenix
6	46028	Supervisor	7742	40	9	HR	New York
7	36272	Director	14878	51	18	Operations	Phoenix
8	72004	Manager	11127	38	10	Sales	Houston
9	51466	Director	13325	54	19	Operations	New York
10	33478	Supervisor	6398	34	9	Marketing	New York
11	76018	Manager	9109	41	15	Sales	Los Angeles
12	89128	Staff	4382	31	1	IT	Los Angeles
13	87224	Director	12072	52	17	IT	New York
14	37036	Staff	4959	37	2	Marketing	New York
15	60544	Staff	3602	33	1	HR	Chicago
16	97563	Staff	4432	33	5	Finance	Phoenix
17	51450	Staff	3446	29	4	Operations	Phoenix
18	66719	Director	13178	49	24	HR	Houston
19	46435	Director	12944	50	21	Finance	Los Angeles

R Code

```r
create_employee_dataset <- function(num_employees) {
  # Define positions with corresponding salary and experience ranges
  positions <- list(
    "Staff" = c(3000, 5000, 1, 5),
    "Supervisor" = c(5000, 8000, 5, 10),
    "Manager" = c(8000, 12000, 10, 15),
    "Director" = c(12000, 15000, 15, 25)
  )

  # Define additional categorical data: departments and locations
  departments <- c("Finance", "HR", "IT", "Marketing", "Operations", "Sales")
  locations <- c("New York", "Los Angeles", "Chicago", "Houston", "Phoenix")

  # Initialize empty vectors for each column
  ID_Number <- integer(num_employees)
  Position <- character(num_employees)
  Salary <- integer(num_employees)
  Age <- integer(num_employees)
  Experience <- integer(num_employees)
  Department <- character(num_employees)
  Location <- character(num_employees)

  # Generate data for each employee
  for (i in 1:num_employees) {
    ID_Number[i] <- sample(10000:99999, 1)
    pos <- sample(names(positions), 1)
    Position[i] <- pos

    salary_range <- positions[[pos]][1:2]
    Salary[i] <- sample(salary_range[1]:salary_range[2], 1)

    exp_range <- positions[[pos]][3:4]
    Experience[i] <- sample(exp_range[1]:exp_range[2], 1)

    Age[i] <- Experience[i] + sample(22:35, 1)
    Department[i] <- sample(departments, 1)
    Location[i] <- sample(locations, 1)
  }

  # Combine the vectors into a data frame
  df <- data.frame(
    ID_Number = ID_Number,
    Position = Position,
    Salary = Salary,
    Age = Age,
    Experience = Experience,
    Department = Department,
```

```
    Location = Location,
    stringsAsFactors = FALSE
  )

  return(df)
}

# Example usage:
df <- create_employee_dataset(20)
print(df)
```

```
   ID_Number   Position Salary Age Experience Department    Location
1      74861 Supervisor   6810  31          8    Finance     Houston
2      55095    Manager  11347  42         14         IT    New York
3      83500    Manager  10823  42         11      Sales    New York
4      79374    Manager   8174  38         10      Sales     Phoenix
5      50985      Staff   4112  39          4  Marketing     Chicago
6      11962    Manager  10748  35         13         HR     Houston
7      20757      Staff   4505  28          2  Marketing    New York
8      62164      Staff   4516  29          5    Finance     Houston
9      48310      Staff   3111  27          4         HR     Houston
10     23971    Manager  11392  42         11 Operations     Chicago
11     79121      Staff   4099  37          2 Operations Los Angeles
12     29862      Staff   3107  37          2 Operations     Phoenix
13     66092      Staff   3041  30          1         HR     Phoenix
14     16468 Supervisor   6947  42         10  Marketing     Houston
15     28587 Supervisor   7343  30          6    Finance     Chicago
16     94056   Director  12336  52         25      Sales Los Angeles
17     33406 Supervisor   7292  43          8 Operations     Chicago
18     57893   Director  14101  51         18    Finance Los Angeles
19     77537    Manager   8900  37         15      Sales     Houston
20     17715 Supervisor   6720  39          8         IT     Chicago
```

3.4.2 Basic Statistics

Python Code

```
import pandas as pd
import numpy as np

def manual_statistics(df, column=None):
    def stats_for_column(values):
        # Remove missing values for accurate computations
        values = values.dropna()
        if pd.api.types.is_numeric_dtype(values):
            count = len(values)
            mean_value = np.mean(values)
            median_value = np.median(values)
            variance_value = np.var(values, ddof=1) if count > 1 else 0
```

```
            std_dev_value = np.sqrt(variance_value)
            min_value = np.min(values)
            max_value = np.max(values)
            q1 = np.percentile(values, 25)
            q3 = np.percentile(values, 75)
            return {
                "count": count,
                "mean": mean_value,
                "median": median_value,
                "variance": variance_value,
                "std_dev": std_dev_value,
                "min": min_value,
                "q1": q1,
                "q3": q3,
                "max": max_value
            }
        else:
            count = len(values)
            unique_count = values.nunique()
            mode_series = values.mode()
            mode_value = mode_series.iloc[0] if not mode_series.empty else None
            frequency = values.value_counts().to_dict()
            return {
                "count": count,
                "unique": unique_count,
                "mode": mode_value,
                "frequency": frequency
            }

    if column is not None:
        return stats_for_column(df[column])
    else:
        summary = {}
        for col in df.columns:
            summary[col] = stats_for_column(df[col])
        return summary
```

```
# Get summary statistics for all columns
stats_all = manual_statistics(df)

# Display the results in attractive tables using pandas' to_markdown()
for col, stats in stats_all.items():
    print(f"\n### Summary Statistics for '{col}'\n")
    if pd.api.types.is_numeric_dtype(df[col]):
        # Create a DataFrame for numeric statistics with Statistic and Value
        stats_df = pd.DataFrame({
            "Statistic": list(stats.keys()),
            "Value": list(stats.values())
        })
```

```
        print(stats_df.to_markdown(index=False))
    else:
        # For categorical data, create summary table and frequency distribution
        summary_df = pd.DataFrame({
            "Statistic": ["count", "unique", "mode"],
            "Value": [stats["count"], stats["unique"], stats["mode"]]
        })
        freq_dict = stats["frequency"]
        freq_df = pd.DataFrame({
            "Category": list(freq_dict.keys()),
            "Frequency": list(freq_dict.values())
        })
        print(summary_df.to_markdown(index=False))
        print("\n")
        print(freq_df.to_markdown(index=False))
```

```
### Summary Statistics for 'ID_Number'

| Statistic   |           Value |
|:------------|----------------:|
| count       |    20           |
| mean        | 69371.3         |
| median      | 74011           |
| variance    |     4.92496e+08 |
| std_dev     | 22192.3         |
| min         | 33478           |
| q1          | 50196.2         |
| q3          | 89496.8         |
| max         | 97563           |

### Summary Statistics for 'Position'

| Statistic   | Value   |
|:------------|:--------|
| count       | 20      |
| unique      | 4       |
| mode        | Staff   |


| Category   |   Frequency |
|:-----------|------------:|
| Staff      |           7 |
| Director   |           5 |
| Manager    |           4 |
| Supervisor |           4 |

### Summary Statistics for 'Salary'
```

Statistic	Value
count	20
mean	8004.5
median	7683
variance	1.53711e+07
std_dev	3920.6
min	3024
q1	4419.5
q3	11363.2
max	14878

Summary Statistics for 'Age'

Statistic	Value
count	20
mean	39.15
median	37.5
variance	64.5553
std_dev	8.03463
min	29
q1	33
q3	43
max	54

Summary Statistics for 'Experience'

Statistic	Value
count	20
mean	9.7
median	9
variance	50.2211
std_dev	7.08668
min	1
q1	4.75
q3	15.5
max	24

Summary Statistics for 'Department'

Statistic	Value
count	20
unique	6
mode	HR

```
| Category    |   Frequency |
|:------------|------------:|
| HR          |           5 |
| Finance     |           3 |
| Marketing   |           3 |
| IT          |           3 |
| Sales       |           3 |
| Operations  |           3 |

### Summary Statistics for 'Location'

| Statistic   | Value    |
|:------------|:---------|
| count       | 20       |
| unique      | 5        |
| mode        | New York |


| Category    |   Frequency |
|:------------|------------:|
| New York    |           8 |
| Phoenix     |           5 |
| Houston     |           3 |
| Los Angeles |           3 |
| Chicago     |           1 |
```

R code

```r
library(knitr)
library(kableExtra)

manual_statistics <- function(df, column = NULL) {
  # Helper function to compute statistics for a single column
  stats_for_column <- function(values) {
    # Remove NA values for accurate computations
    values <- values[!is.na(values)]

    if (is.numeric(values)) {
      count <- length(values)
      mean_value <- mean(values)
      median_value <- median(values)
      variance_value <- if (count > 1) var(values) else 0
      std_dev_value <- sqrt(variance_value)
      min_value <- min(values)
      max_value <- max(values)
      q1 <- as.numeric(quantile(values, 0.25))
      q3 <- as.numeric(quantile(values, 0.75))

      return(list(
```

```
        count    = count,
        mean     = mean_value,
        median   = median_value,
        variance = variance_value,
        std_dev  = std_dev_value,
        min      = min_value,
        q1       = q1,
        q3       = q3,
        max      = max_value
      ))
    } else {
      count <- length(values)
      unique_count <- length(unique(values))
      tab <- table(values)
      mode_value <- names(tab)[which.max(tab)]
      frequency <- as.list(tab)

      return(list(
        count     = count,
        unique    = unique_count,
        mode      = mode_value,
        frequency = frequency
      ))
    }
  }

  # If a specific column is provided, compute statistics only for that column.
  if (!is.null(column)) {
    return(stats_for_column(df[[column]]))
  } else {
    # Otherwise, compute statistics for each column in the DataFrame.
    summary <- list()
    for (col in names(df)) {
      summary[[col]] <- stats_for_column(df[[col]])
    }
    return(summary)
  }
}
```

```
# Hitung summary statistics untuk semua kolom
stats_all <- manual_statistics(df)

# Loop untuk menampilkan hasil setiap kolom dengan DT::datatable
for (col in names(stats_all)) {
  cat(paste0("<h3>Summary Statistics for '", col, "'</h3>"))

  col_stats <- stats_all[[col]]

  if (is.numeric(df[[col]])) {
```

```
    stats_df <- data.frame(
      Statistic = names(col_stats),
      Value = as.numeric(unlist(col_stats)),
      stringsAsFactors = FALSE
    )
    print(DT::datatable(stats_df,
                        caption = paste("Summary for", col),
                        options = list(pageLength = 5, autoWidth = TRUE)))
  } else {
    summary_df <- data.frame(
      Statistic = c("count", "unique", "mode"),
      Value = c(col_stats$count, col_stats$unique, col_stats$mode),
      stringsAsFactors = FALSE
    )
    freq_df <- as.data.frame(do.call(rbind, col_stats$frequency))
    freq_df <- cbind(Category = rownames(freq_df), freq_df)
    rownames(freq_df) <- NULL
    names(freq_df)[2] <- "Frequency"

    print(DT::datatable(summary_df,
                        caption = paste("Summary for", col),
                        options = list(pageLength = 5, autoWidth = TRUE)))
    cat("<br>")
    print(DT::datatable(freq_df,
                        caption = paste("Frequency Distribution for", col),
                        options = list(pageLength = 5, autoWidth = TRUE)))
  }

  cat("<br><br>")
}
```

```
FALSE <h3>Summary Statistics for 'ID_Number'</h3><br><br><h3>Summary Statistics for 'Posit
```

Part II

Data Wrangling

Chapter 4

Data Collection

Data collection (**Data gathering**) is the process of **systematically gathering, measuring, and analyzing data** to extract insights, train machine learning models, or support data-driven decision-making. It is a fundamental step in **Data Science Programming**, ensuring the availability of high-quality, relevant data for analysis. Data collection methods are broadly categorized into **Primary Data Collection**, where data is obtained firsthand, and **Secondary Data Collection**, where existing datasets are used for analysis.

4.1 Primary Data Collection

Primary data collection involves gathering **firsthand data** directly from sources for analysis in data science projects. It ensures high relevance and accuracy but may require significant time and resources.

Type	Description	Examples
Qualitative	Non-numerical data used to understand behaviors, opinions, and experiences.	User Interviews, Social Media Sentiment Analysis, Observations, Case Studies
Quantitative	Numerical data used to measure patterns, trends, and relationships.	Surveys, A/B Testing, IoT Sensor Data, Web Scraping, API Data Extraction

Qualitative methods are valuable in data science for analyzing unstructured data like text, images, and human interactions. **Quantitative methods** provide structured, numerical data, essential for building predictive models, statistical analysis, and AI-driven insights.

Choosing the right method depends on the **data science goal**—**qualitative** for understanding context and **quantitative** for model-driven decision-making.

4.1.1 Web Scraping Data with Py

Web scraping is the process of extracting data from websites using automated scripts. Python is one of the most popular languages for web scraping due to its powerful libraries and ease of use.

Why Use Python for Web Scraping?

- User-friendly: Python has simple syntax, making scraping easier.
- Powerful libraries: BeautifulSoup, Requests, and Selenium allow efficient data extraction.
- Automation capabilities: Scraping can be scheduled to collect data regularly.

Common Web Scraping Techniques in Python

- Using `BeautifulSoup` – Best for extracting data from static web pages.
- Using `Requests` – Ideal for sending HTTP requests to fetch web content.
- Using `Selenium` – Useful for scraping JavaScript-rendered websites.

Before scraping, always check the website's robots.txt file to ensure compliance with its policies.

4.1.2 Web Scraping Data with R

Web scraping is the process of extracting data from websites. In R, web scraping can be done using popular packages such as rvest, httr, and RSelenium. These tools allow users to collect, process, and analyze web data efficiently.

Why Use R for Web Scraping?

- Easy to use: Packages like rvest simplify HTML parsing.
- Powerful data manipulation: R has excellent support for data cleaning and analysis.
- Automating tasks: Scraping can be automated for large-scale data collection.

Common Web Scraping Methods in R

- Using `rvest` – Best for simple static web pages.
- Using `httr` – Useful for sending HTTP requests.
- Using `RSelenium` – Required for scraping JavaScript-heavy websites.

Notes: Before scraping, always check a website's robots.txt file to ensure compliance with its policies.

4.2 Secondary Data Collection

Secondary data collection involves using **existing datasets** instead of gathering new data. It is widely used in **data science** for historical analysis, benchmarking, and training machine learning models. These data sources come from public records, business reports, and online databases.

Type	Description	Examples
Public Data	Government and institutional datasets used for large-scale analysis.	Census Data, Open Government Data, Research Reports
Business Data	Company-generated reports and analytics used for industry insights.	Financial Reports, Market Research, Customer Data
Online Data	Digital datasets available for academic and practical applications.	Scientific Databases (Google Scholar, ResearchGate), Open Data Portals (Kaggle, UCI ML Repository), Web Scraped Data

Secondary data is **cost-effective and time-efficient** but may require preprocessing to ensure quality and relevance for data science applications. Choosing **reliable** and **well-structured** data sources is essential for accurate analysis.

4.2.1 Use Online Data

Read all data first before processing it. This makes the script more efficient because you avoid reading the Excel file multiple times in different loops.

4.2.2 Read All Sheets First

Instead of reading the Excel file multiple times inside loops, read all sheets first and store them in a list.

Python Code

```python
import pandas as pd

# File path
path = "data/bab4/Rekap_Kuesioner.xlsx"

# Read all sheets at once
sheets = pd.read_excel(path, sheet_name=None)  # Dictionary of DataFrames
```

`pd.read_excel(path, sheet_name=None)` reads all sheets into a dictionary where keys are sheet names and values are DataFrames. This avoids multiple file reads, making the process more efficient.

R Code

```r
# Import libraries
library(readxl)  # To read Excel files

# File path
path <- "data/bab4/Rekap_Kuesioner.xlsx"
```

```
# Read all sheets at once
all_sheets <- lapply(1:21, function(i) read_excel(path, sheet = i))
all_sheets
```

```
[[1]]
# A tibble: 33 x 13
   `Hasil Kuisioner` ...2  ...3  ...4  ...5  ...6  ...7  ...8  ...9  ...10 ...11
   <chr>             <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
 1 Semester : Gasal~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 2 Sesi : Semua      <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 3 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 4 Kelas             DP11~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 5 Mata Kuliah       Nirm~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 6 Program Studi     DESA~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 7 Dosen             I KE~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 8 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 9 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
10 No. Responden     Kode~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
# i 23 more rows
# i 2 more variables: ...12 <chr>, ...13 <chr>

[[2]]
# A tibble: 34 x 13
   `Hasil Kuisioner` ...2  ...3  ...4  ...5  ...6  ...7  ...8  ...9  ...10 ...11
   <chr>             <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
 1 Semester : Gasal~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 2 Sesi : Semua      <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 3 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 4 Kelas             DP11~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 5 Mata Kuliah       Gamb~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 6 Program Studi     DESA~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 7 Dosen             WILD~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 8 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 9 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
10 No. Responden     Kode~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
# i 24 more rows
# i 2 more variables: ...12 <chr>, ...13 <chr>

[[3]]
# A tibble: 33 x 13
   `Hasil Kuisioner` ...2  ...3  ...4  ...5  ...6  ...7  ...8  ...9  ...10 ...11
   <chr>             <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
 1 Semester : Gasal~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 2 Sesi : Semua      <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 3 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 4 Kelas             DP11~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 5 Mata Kuliah       Mate~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 6 Program Studi     DESA~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 7 Dosen             OEMA~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
```

```
 8 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 9 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
10 No. Responden     Kode~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
# i 23 more rows
# i 2 more variables: ...12 <chr>, ...13 <chr>

[[4]]
# A tibble: 33 x 13
   `Hasil Kuisioner` ...2  ...3  ...4  ...5  ...6  ...7  ...8  ...9  ...10 ...11
   <chr>             <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
 1 Semester : Gasal~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 2 Sesi : Semua      <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 3 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 4 Kelas             DP11~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 5 Mata Kuliah       Peng~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 6 Program Studi     DESA~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 7 Dosen             DODY~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 8 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 9 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
10 No. Responden     Kode~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
# i 23 more rows
# i 2 more variables: ...12 <chr>, ...13 <chr>

[[5]]
# A tibble: 37 x 13
   `Hasil Kuisioner` ...2  ...3  ...4  ...5  ...6  ...7  ...8  ...9  ...10 ...11
   <chr>             <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
 1 Semester : Gasal~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 2 Sesi : Semua      <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 3 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 4 Kelas             DP21~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 5 Mata Kuliah       Desa~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 6 Program Studi     DESA~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 7 Dosen             HARR~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 8 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 9 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
10 No. Responden     Kode~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
# i 27 more rows
# i 2 more variables: ...12 <chr>, ...13 <chr>

[[6]]
# A tibble: 36 x 13
   `Hasil Kuisioner` ...2  ...3  ...4  ...5  ...6  ...7  ...8  ...9  ...10 ...11
   <chr>             <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
 1 Semester : Gasal~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 2 Sesi : Semua      <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 3 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 4 Kelas             DP21~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 5 Mata Kuliah       Tekn~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
```

```
 6 Program Studi     DESA~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 7 Dosen             IYUS~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 8 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 9 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
10 No. Responden     Kode~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
# i 26 more rows
# i 2 more variables: ...12 <chr>, ...13 <chr>

[[7]]
# A tibble: 36 x 13
   `Hasil Kuisioner` ...2  ...3  ...4  ...5  ...6  ...7  ...8  ...9  ...10 ...11
   <chr>             <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
 1 Semester : Gasal~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 2 Sesi : Semua      <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 3 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 4 Kelas             DP21~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 5 Mata Kuliah       Baha~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 6 Program Studi     DESA~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 7 Dosen             I KE~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 8 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 9 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
10 No. Responden     Kode~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
# i 26 more rows
# i 2 more variables: ...12 <chr>, ...13 <chr>

[[8]]
# A tibble: 36 x 13
   `Hasil Kuisioner` ...2  ...3  ...4  ...5  ...6  ...7  ...8  ...9  ...10 ...11
   <chr>             <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
 1 Semester : Gasal~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 2 Sesi : Semua      <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 3 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 4 Kelas             DP21~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 5 Mata Kuliah       Peng~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 6 Program Studi     DESA~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 7 Dosen             HARR~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 8 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 9 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
10 No. Responden     Kode~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
# i 26 more rows
# i 2 more variables: ...12 <chr>, ...13 <chr>

[[9]]
# A tibble: 37 x 13
   `Hasil Kuisioner` ...2  ...3  ...4  ...5  ...6  ...7  ...8  ...9  ...10 ...11
   <chr>             <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
 1 Semester : Gasal~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 2 Sesi : Semua      <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 3 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
```

```
 4 Kelas             DP21~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 5 Mata Kuliah       Sema~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 6 Program Studi     DESA~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 7 Dosen             ADEL~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 8 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 9 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
10 No. Responden     Kode~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
# i 27 more rows
# i 2 more variables: ...12 <chr>, ...13 <chr>

[[10]]
# A tibble: 38 x 13
   `Hasil Kuisioner` ...2  ...3  ...4  ...5  ...6  ...7  ...8  ...9  ...10 ...11
   <chr>             <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
 1 Semester : Gasal~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 2 Sesi : Semua      <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 3 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 4 Kelas             DP31~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 5 Mata Kuliah       Desa~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 6 Program Studi     DESA~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 7 Dosen             DODY~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 8 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 9 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
10 No. Responden     Kode~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
# i 28 more rows
# i 2 more variables: ...12 <chr>, ...13 <chr>

[[11]]
# A tibble: 34 x 13
   `Hasil Kuisioner` ...2  ...3  ...4  ...5  ...6  ...7  ...8  ...9  ...10 ...11
   <chr>             <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
 1 Semester : Gasal~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 2 Sesi : Semua      <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 3 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 4 Kelas             DP31~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 5 Mata Kuliah       Ergo~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 6 Program Studi     DESA~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 7 Dosen             DODY~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 8 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 9 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
10 No. Responden     Kode~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
# i 24 more rows
# i 2 more variables: ...12 <chr>, ...13 <chr>

[[12]]
# A tibble: 38 x 13
   `Hasil Kuisioner` ...2  ...3  ...4  ...5  ...6  ...7  ...8  ...9  ...10 ...11
   <chr>             <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
 1 Semester : Gasal~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
```

```
 2 Sesi : Semua      <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 3 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 4 Kelas             DP31~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 5 Mata Kuliah       Tinj~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 6 Program Studi     DESA~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 7 Dosen             WILD~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 8 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 9 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
10 No. Responden     Kode~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
# i 28 more rows
# i 2 more variables: ...12 <chr>, ...13 <chr>

[[13]]
# A tibble: 38 x 13
   `Hasil Kuisioner` ...2  ...3  ...4  ...5  ...6  ...7  ...8  ...9  ...10 ...11
   <chr>             <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
 1 Semester : Gasal~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 2 Sesi : Semua      <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 3 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 4 Kelas             DP31~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 5 Mata Kuliah       Meto~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 6 Program Studi     DESA~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 7 Dosen             ADEL~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 8 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 9 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
10 No. Responden     Kode~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
# i 28 more rows
# i 2 more variables: ...12 <chr>, ...13 <chr>

[[14]]
# A tibble: 38 x 13
   `Hasil Kuisioner` ...2  ...3  ...4  ...5  ...6  ...7  ...8  ...9  ...10 ...11
   <chr>             <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
 1 Semester : Gasal~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 2 Sesi : Semua      <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 3 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 4 Kelas             DP31~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 5 Mata Kuliah       Work~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 6 Program Studi     DESA~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 7 Dosen             IYUS~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 8 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 9 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
10 No. Responden     Kode~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
# i 28 more rows
# i 2 more variables: ...12 <chr>, ...13 <chr>

[[15]]
# A tibble: 38 x 13
   `Hasil Kuisioner` ...2  ...3  ...4  ...5  ...6  ...7  ...8  ...9  ...10 ...11
```

```
   <chr>             <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
 1 Semester : Gasal~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 2 Sesi : Semua      <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 3 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 4 Kelas             DP31~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 5 Mata Kuliah       Pemo~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 6 Program Studi     DESA~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 7 Dosen             IYUS~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 8 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 9 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
10 No. Responden     Kode~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
# i 28 more rows
# i 2 more variables: ...12 <chr>, ...13 <chr>

[[16]]
# A tibble: 36 x 13
   `Hasil Kuisioner` ...2  ...3  ...4  ...5  ...6  ...7  ...8  ...9  ...10 ...11
   <chr>             <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
 1 Semester : Gasal~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 2 Sesi : Semua      <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 3 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 4 Kelas             DP41~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 5 Mata Kuliah       Desa~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 6 Program Studi     DESA~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 7 Dosen             ADEL~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 8 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 9 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
10 No. Responden     Kode~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
# i 26 more rows
# i 2 more variables: ...12 <chr>, ...13 <chr>

[[17]]
# A tibble: 37 x 13
   `Hasil Kuisioner` ...2  ...3  ...4  ...5  ...6  ...7  ...8  ...9  ...10 ...11
   <chr>             <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
 1 Semester : Gasal~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 2 Sesi : Semua      <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 3 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 4 Kelas             DP41~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 5 Mata Kuliah       Kerj~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 6 Program Studi     DESA~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 7 Dosen             OEMA~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 8 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 9 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
10 No. Responden     Kode~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
# i 27 more rows
# i 2 more variables: ...12 <chr>, ...13 <chr>

[[18]]
```

```
# A tibble: 37 x 13
   `Hasil Kuisioner` ...2  ...3  ...4  ...5  ...6  ...7  ...8  ...9  ...10 ...11
   <chr>             <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
 1 Semester : Gasal~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 2 Sesi : Semua      <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 3 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 4 Kelas             DP41~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 5 Mata Kuliah       Kolo~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 6 Program Studi     DESA~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 7 Dosen             DODY~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 8 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 9 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
10 No. Responden     Kode~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
# i 27 more rows
# i 2 more variables: ...12 <chr>, ...13 <chr>

[[19]]
# A tibble: 37 x 13
   `Hasil Kuisioner` ...2  ...3  ...4  ...5  ...6  ...7  ...8  ...9  ...10 ...11
   <chr>             <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
 1 Semester : Gasal~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 2 Sesi : Semua      <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 3 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 4 Kelas             DP41~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 5 Mata Kuliah       Mana~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 6 Program Studi     DESA~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 7 Dosen             I KE~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 8 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 9 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
10 No. Responden     Kode~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
# i 27 more rows
# i 2 more variables: ...12 <chr>, ...13 <chr>

[[20]]
# A tibble: 37 x 13
   `Hasil Kuisioner` ...2  ...3  ...4  ...5  ...6  ...7  ...8  ...9  ...10 ...11
   <chr>             <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
 1 Semester : Gasal~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 2 Sesi : Semua      <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 3 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 4 Kelas             DP41~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 5 Mata Kuliah       Peng~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 6 Program Studi     DESA~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 7 Dosen             HARR~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 8 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 9 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
10 No. Responden     Kode~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
# i 27 more rows
# i 2 more variables: ...12 <chr>, ...13 <chr>
```

```
[[21]]
# A tibble: 34 x 13
   `Hasil Kuisioner` ...2  ...3  ...4  ...5  ...6  ...7  ...8  ...9  ...10 ...11
   <chr>             <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
 1 Semester : Gasal~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 2 Sesi : Semua      <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 3 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 4 Kelas             DP42~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 5 Mata Kuliah       Tuga~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 6 Program Studi     DESA~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 7 Dosen             OEMA~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 8 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
 9 <NA>              <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
10 No. Responden     Kode~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
# i 24 more rows
# i 2 more variables: ...12 <chr>, ...13 <chr>
```

`lapply(1:21, function(i) read_excel(path, sheet = i))` reads all 21 sheets into a list called all_sheets. Now, `all_sheets[[i]]` represents the data from the i-th sheet.

4.2.3 Extract Course Names

Now, we extract course names from the already loaded data.

Python Code

```python
# Initialize course name list
MataKuliah = []

# Loop through sheets
for i, (sheet_name, df) in enumerate(sheets.items(), start=1):
    # Find row containing "Mata Kuliah"
    matakuliah_row = df[df.iloc[:, 0] == "Mata Kuliah"]

    if not matakuliah_row.empty:
        MataKuliah.append(matakuliah_row.iloc[0, 1])  # Get column 2 value
    else:
        MataKuliah.append(f"Sheet{i}")  # Default if not found

MataKuliah
```

```
['Nirmana I', 'Gambar I', 'Matematika Geometri', 'Pengantar Bidang Studi Desain Produk Indust
```

Searches for the row where the first column is “Mata Kuliah”. Extracts the corresponding course name from the second column. If not found, assigns a default name like “Sheet1”, “Sheet2”, etc.

R Code

```r
MataKuliah <- sapply(all_sheets, function(sheet) {
  matakuliah_row <- which(sheet[[1]] == "Mata Kuliah")
  if (length(matakuliah_row) > 0) {
    return(as.character(sheet[matakuliah_row, 2][[1]]))  # Extract course name
  } else {
    return(NA)  # If not found, return NA
  }
})

# Replace missing course names with default values
MataKuliah[is.na(MataKuliah)] <- paste0("Sheet", which(is.na(MataKuliah)))
MataKuliah
```

```
 [1] "Nirmana I"
 [2] "Gambar I"
 [3] "Matematika Geometri"
 [4] "Pengantar Bidang Studi Desain Produk Industri"
 [5] "Desain Produk I"
 [6] "Teknik Presentasi I"
 [7] "Bahan dan Proses Produksi"
 [8] "Pengantar HAKI"
 [9] "Semantika Produk"
[10] "Desain Produk III"
[11] "Ergonomi Desain I"
[12] "Tinjauan Desain"
[13] "Metodologi Desain"
[14] "Workshop Material"
[15] "Pemodelan Digital I"
[16] "Desain Produk V"
[17] "Kerja Profesi"
[18] "Kolokium Pra Tugas Akhir"
[19] "Manajemen Pemasaran Produk"
[20] "Pengantar Desain Sarana Lingkungan"
[21] "Tugas Akhir"
```

This avoids multiple read_excel() calls inside the loop. If `Mata Kuliah` is found, it extracts the course name. Otherwise, it assigns `SheetX` as a placeholder.

4.2.4 Extract `Average` Data

Now, process the average scores from each sheet.

Python Code

```python
# Define questions (Pertanyaan)
Pertanyaan = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L"]

# Define categories (Kategori)
```

```
Kategori = ["Reliability"] * 6 + ["Responsiveness"] + ["Assurance"] * 2 + ["Empathy"] + ["Tan

# Create DataFrame with questions & categories
Ganjil_23_24 = pd.DataFrame({"Pertanyaan": Pertanyaan, "Kategori": Kategori})

# Extract "Rata-rata" values
for i, (sheet_name, df) in enumerate(sheets.items()):
    rata_rata_row = df[df.iloc[:, 0] == "Rata-rata"]

    if not rata_rata_row.empty:
        selected_data = rata_rata_row.iloc[0, 1:13].astype(float).round()
        Ganjil_23_24[MataKuliah[i]] = selected_data.values
    else:
        Ganjil_23_24[MataKuliah[i]] = [None] * len(Pertanyaan)

# Display the Final Table
print(Ganjil_23_24)
```

```
   Pertanyaan        Kategori  Nirmana I  ...  Tugas Akhir  Sheet22  Sheet23
0           A     Reliability        4.0  ...          4.0     None     None
1           B     Reliability        4.0  ...          3.0     None     None
2           C     Reliability        3.0  ...          3.0     None     None
3           D     Reliability        4.0  ...          3.0     None     None
4           E     Reliability        4.0  ...          3.0     None     None
5           F     Reliability        4.0  ...          3.0     None     None
6           G  Responsiveness        4.0  ...          3.0     None     None
7           H       Assurance        4.0  ...          4.0     None     None
8           I       Assurance        4.0  ...          4.0     None     None
9           J         Empathy        4.0  ...          3.0     None     None
10          K        Tangible        2.0  ...          3.0     None     None
11          L        Tangible        3.0  ...          3.0     None     None

[12 rows x 25 columns]
```

R Code

```
# First column (questions)
Pertanyaan <- c("A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L")

# Category assignments
Kategori <- c(rep("Reliability", 6),
              "Responsiveness",
              rep("Assurance", 2),
              "Empathy",
              rep("Tangible", 2))

# Create a data frame with questions & categories
Ganjil_23_24 <- data.frame(Pertanyaan = Pertanyaan, Kategori = Kategori)
```

```
# Loop through sheets to extract "Rata-rata" values
for (i in 1:21) {
  rata_rata_row <- which(all_sheets[[i]][[1]] == "Rata-rata")

  if (length(rata_rata_row) > 0) {
    selected_data <- all_sheets[[i]][rata_rata_row, 2:13]  # Extract 12 columns
    Ganjil_23_24[[MataKuliah[i]]] <- round(as.numeric(selected_data)) # Convert
  } else {
    Ganjil_23_24[[MataKuliah[i]]] <- NA  # Assign NA if "Rata-rata" not found
  }
}

# Display the Final Table

library(knitr)        # To create formatted tables
kable(Ganjil_23_24)
```

Pertanyaan	Kategori	Nirgaran	Matematika	Pengantar Bidang Studi Desain Produk Industri	Desain Produk I	Teknologi Bahan dan Proses Produksi I	Pengantar Desain Produk II	Semantika Produk	Desain Produk III	Ergonomi Dalam Desain I	Tinjauan Desain	Metodologi Desain I	Workshop Material	Pemodelan Digital I	Desain Produk V	Pemodelan Desain Prototipe	Kerja Praktek	Tugas Akhir	Pengantar Manajemen Pembangunan Proyek	Manajemen Sarana Lingkungan	Tugas Akhir
A	Reliability	4	4	4	4	3	3	3	4	4	4	4	4	4	3	3	3	3	3	3	4
B	Reliability	4	4	4	4	3	3	3	4	4	4	4	4	4	3	3	3	3	3	3	3
C	Reliability	4	4	4	3	4	3	3	3	4	4	4	4	4	3	3	3	3	3	3	3
D	Reliability	4	4	4	4	4	3	3	3	4	4	4	4	4	3	3	3	3	3	3	3
E	Reliability	4	4	4	3	4	3	3	3	4	4	4	4	4	3	3	3	3	3	3	3
F	Reliability	4	4	4	3	3	3	3	3	4	4	4	4	4	3	3	3	3	3	3	3
G	Responsiveness	4	4	4	3	3	3	3	3	4	4	4	3	4	4	3	3	3	3	3	3
H	Assurance	4	3	4	4	3	3	3	3	4	4	4	3	4	4	3	4	3	3	3	4
I	Assurance	4	3	4	4	3	3	3	3	4	4	4	4	4	4	3	3	3	3	3	4
J	Empathy	4	3	4	4	3	3	3	3	4	4	4	3	4	4	3	3	3	3	3	3
K	Tangible	3	3	3	3	4	3	3	3	3	4	4	3	4	4	3	3	3	3	3	3
L	Tangible	3	3	4	3	3	3	3	3	4	4	4	3	4	4	3	3	3	3	3	3

Instead of reading the Excel file multiple times, we use `all_sheets[[i]]` to get the necessary data. This makes the code more efficient because the Excel file is only read once.

Chapter 5

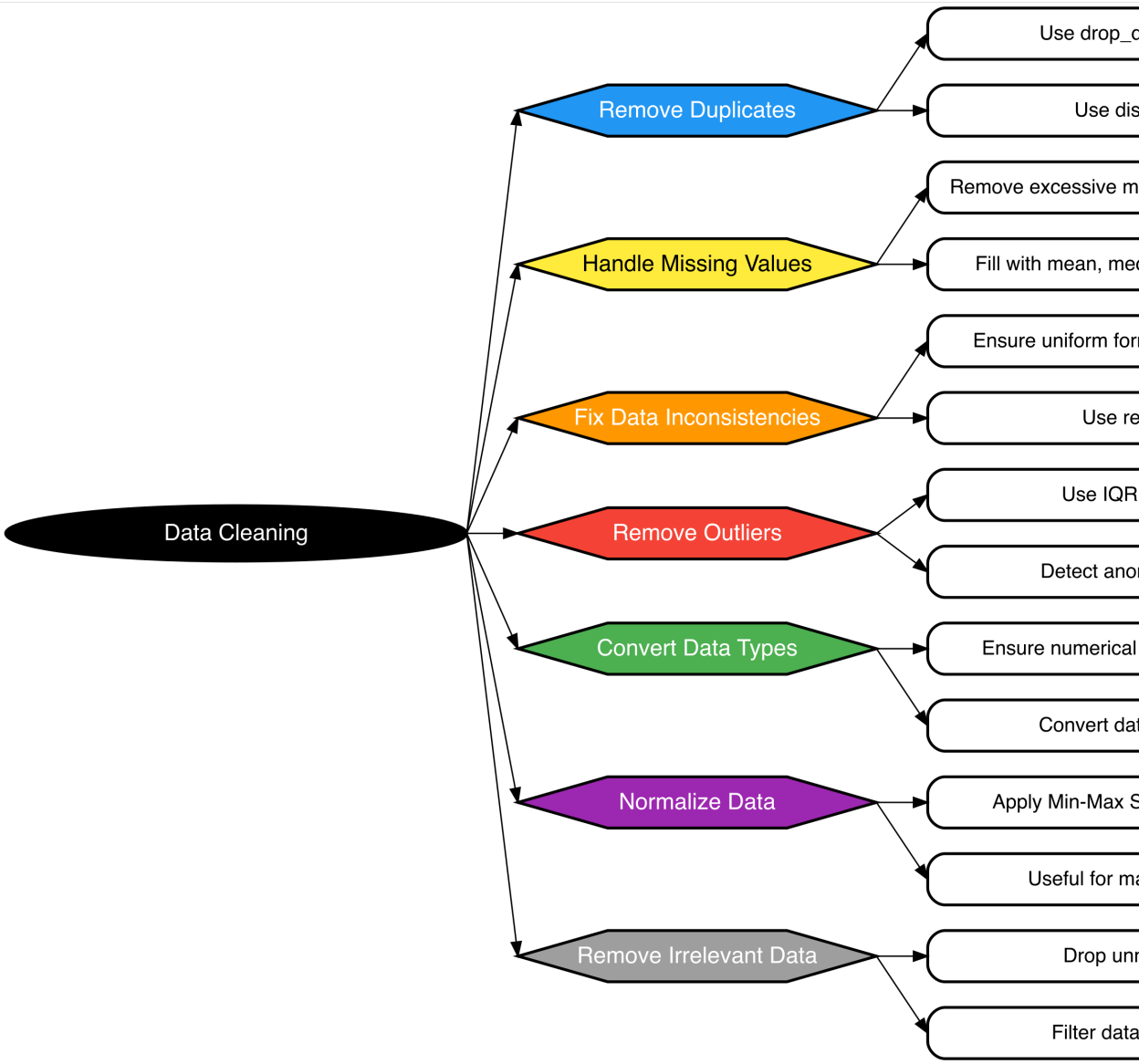
Data Cleaning

Data cleaning is the process of identifying and correcting or removing inaccurate, incomplete, irrelevant, or erroneous data in a dataset. It is a crucial step in data analysis and machine learning to ensure accurate and reliable results.

5.1 Data Cleaning Process

In the data analysis process, the first and most critical step is ensuring that the data is clean and ready for further processing. Poor-quality data can lead to errors in analysis and generate misleading insights. Therefore, Data Cleaning is an essential component of the data preprocessing pipeline.

The following Mind Map illustrates the key stages of the Data Cleaning process, including duplicate removal, handling of missing values, normalization, and the treatment of anomalies such as outliers and inconsistencies. By following these steps, data quality can be significantly improved, leading to more accurate and reliable analytical outcomes.


Data Cleaning
Remove Duplicates
Use drop_d
Use dis
Handle Missing Values
Remove excessive m
Fill with mean, med
Fix Data Inconsistencies
Ensure uniform for
Use re
Remove Outliers
Use IQR
Detect anor
Convert Data Types
Ensure numerical
Convert dat
Normalize Data
Apply Min-Max S
Useful for ma
Remove Irrelevant Data
Drop unr
Filter data

5.2 Study Case Example

5.2.1 About Dataset

This is created dummy dataset, covers the past **three years** with **12 variables**, including missing values, duplicates, and outliers.

Feature	Description
`Date`	Observation date (with missing & duplicate values)
`CPO_Price`	CPO price per ton (includes outliers)
`Production`	Production volume (contains missing values)
`Exports`	CPO export volume
`Imports`	CPO import volume
`USD_IDR`	Exchange rate (USD to IDR)
`Market_Sentiment`	Market sentiment (text inconsistencies)
`Demand_Index`	Synthetic demand index (100-200 range)
`Supply_Index`	Synthetic supply index (90-190 range)
`Rainfall`	Rainfall in mm
`Temperature`	Temperature in Celsius
`Political_Stability`	Political stability score (1-10 scale)

5.2.2 Generate Raw Dataset

Python Code

```python
import pandas as pd
import numpy as np
from datetime import datetime, timedelta

# Generate 1000 dates
start_date = datetime.today() - timedelta(days=3*365)
dates = [start_date + timedelta(days=i) for i in range(1000)]

# Random Data Generation
np.random.seed(42)
cpo_prices = np.random.normal(800, 50, 1000)
cpo_prices[np.random.randint(0, 1000, 5)] = np.random.choice([2000, 2500, 50, 100], 5)

production = np.random.randint(90, 130, 1000).astype(float)
production[np.random.randint(0, 1000, 20)] = np.nan

exports = np.random.randint(200, 500, 1000)
imports = np.random.randint(50, 150, 1000)

usd_idr = np.random.normal(14500, 300, 1000)
demand_index = np.random.randint(100, 200, 1000)
supply_index = np.random.randint(90, 190, 1000)

rainfall = np.random.randint(50, 200, 1000)
```

```
temperature = np.random.uniform(25, 35, 1000)

political_stability = np.random.randint(1, 10, 1000).astype(float)
political_stability[np.random.randint(0, 1000, 15)] = np.nan

sentiments = np.random.choice(["Positive", "neutral", "NEGATIVE"], 1000)

dates[100] = dates[99]
dates[500] = None

# Create DataFrame
df = pd.DataFrame({
    'Date': dates, 'CPO_Price': cpo_prices, 'Production': production,
    'Exports': exports, 'Imports': imports, 'USD_IDR': usd_idr,
    'Market_Sentiment': sentiments, 'Demand_Index': demand_index,
    'Supply_Index': supply_index, 'Rainfall': rainfall,
    'Temperature': temperature, 'Political_Stability': political_stability
})

# Ensure dataset is sorted by Date for Time Series modeling
df = df.sort_values(by='Date')

print(df.head())
```

```
                        Date   CPO_Price  ...  Temperature  Political_Stability
0 2022-07-04 13:54:00.561316  824.835708  ...    31.604697                  3.0
1 2022-07-05 13:54:00.561316  793.086785  ...    29.708164                  5.0
2 2022-07-06 13:54:00.561316  832.384427  ...    26.989298                  3.0
3 2022-07-07 13:54:00.561316  876.151493  ...    31.016866                  5.0
4 2022-07-08 13:54:00.561316  788.292331  ...    25.606831                  3.0

[5 rows x 12 columns]
```

R Code

```
library(dplyr)
library(lubridate)
library(tidyr)

# Generate 1000 dates
set.seed(42)
start_date <- Sys.Date() - years(3)
dates <- seq.Date(from = start_date, by = "day", length.out = 1000)

# Random Data Generation
cpo_prices <- rnorm(1000, mean = 800, sd = 50)
cpo_prices[sample(1:1000, 5)] <- sample(c(2000, 2500, 50, 100), 5, replace = TRUE)

production <- as.numeric(sample(90:130, 1000, replace = TRUE))
```

```r
production[sample(1:1000, 20)] <- NA

exports <- sample(200:500, 1000, replace = TRUE)
imports <- sample(50:150, 1000, replace = TRUE)

usd_idr <- rnorm(1000, mean = 14500, sd = 300)
demand_index <- sample(100:200, 1000, replace = TRUE)
supply_index <- sample(90:190, 1000, replace = TRUE)

rainfall <- sample(50:200, 1000, replace = TRUE)
temperature <- runif(1000, min = 25, max = 35)

political_stability <- as.numeric(sample(1:10, 1000, replace = TRUE))
political_stability[sample(1:1000, 15)] <- NA

sentiments <- sample(c("Positive", "neutral", "NEGATIVE"), 1000, replace = TRUE)

# Introduce duplicate and missing values in Date
dates[100] <- dates[99]
dates[500] <- NA

# Create DataFrame
df <- data.frame(
  Date = dates, CPO_Price = cpo_prices, Production = production,
  Exports = exports, Imports = imports, USD_IDR = usd_idr,
  Market_Sentiment = sentiments, Demand_Index = demand_index,
  Supply_Index = supply_index, Rainfall = rainfall,
  Temperature = temperature, Political_Stability = political_stability
)

# Ensure dataset is sorted by Date for Time Series modeling
df <- df %>% arrange(Date)

print(head(df))
```

```
        Date CPO_Price Production Exports Imports  USD_IDR Market_Sentiment
1 2022-07-03  868.5479        102     244      60 14442.79         Positive
2 2022-07-04  771.7651        115     287      50 14765.80          neutral
3 2022-07-05  818.1564         99     208      77 14457.90         Positive
4 2022-07-06   50.0000         90     467     107 14033.63         NEGATIVE
5 2022-07-07  820.2134        130     305     148 14881.60         NEGATIVE
6 2022-07-08  794.6938         92     490     111 14127.73         Positive
  Demand_Index Supply_Index Rainfall Temperature Political_Stability
1          159          188      124    26.44452                   9
2          131          182       74    32.59184                   3
3          198          143      118    26.40042                   2
4          144          185      158    25.83467                   4
5          198           96       97    27.25106                   3
6          132          177      192    27.59168                   7
```

5.3 Data Cleaning Process

5.3.1 Handling Missing Values

- Remove rows/columns with excessive missing data (`dropna()` in Pandas, `na.omit()` in R)
- Fill missing values using mean, median, mode, or interpolation (`fillna()` in Pandas, `mutate()` in R)

Python Code

```python
# Convert 'Date' to datetime and fix missing values
df['Date'] = pd.to_datetime(df['Date']).interpolate()

# Fill missing numerical values
df['Production'] = df['Production'].ffill()
df['Political_Stability'] = df['Political_Stability'].fillna(df['Political_Stability'].mea

# Fill categorical missing values
df['Market_Sentiment'] = df['Market_Sentiment'].fillna(df['Market_Sentiment'].mode()[0])

# Drop remaining missing values
df_dropna = df.dropna()
```

R Code

```r
library(zoo)
# Convert 'Date' to datetime and fix missing values
df$Date <- as.Date(df$Date)
df$Date <- zoo::na.approx(df$Date, na.rm = FALSE)

# Fill missing numerical values
df$Production <- tidyr::fill(df, Production, .direction = "down")$Production
df$Political_Stability[is.na(df$Political_Stability)] <- mean(df$Political_Stability, na.r

# Fill categorical missing values
df$Market_Sentiment[is.na(df$Market_Sentiment)] <- names(sort(table(df$Market_Sentiment),

# Drop remaining missing values
df_dropna <- na.omit(df)

print(head(df_dropna))
```

```
   Date CPO_Price Production Exports Imports  USD_IDR Market_Sentiment
1 19176  868.5479        102     244      60 14442.79         Positive
2 19177  771.7651        115     287      50 14765.80          neutral
3 19178  818.1564         99     208      77 14457.90         Positive
4 19179   50.0000         90     467     107 14033.63         NEGATIVE
5 19180  820.2134        130     305     148 14881.60         NEGATIVE
6 19181  794.6938         92     490     111 14127.73         Positive
```

```
  Demand_Index Supply_Index Rainfall Temperature Political_Stability
1          159          188      124    26.44452                   9
2          131          182       74    32.59184                   3
3          198          143      118    26.40042                   2
4          144          185      158    25.83467                   4
5          198           96       97    27.25106                   3
6          132          177      192    27.59168                   7
```

5.3.2 Removing Duplicates

- Use `drop_duplicates()` in Pandas (Python)
- Use `distinct()` in R (dplyr)

Python Code

```
df_duplicates= df_dropna.drop_duplicates()
```

R Code

```
# Remove duplicate rows
df_duplicates <- df_dropna %>% distinct()

print(head(df_duplicates))
```

```
   Date CPO_Price Production Exports Imports  USD_IDR Market_Sentiment
1 19176  868.5479        102     244      60 14442.79         Positive
2 19177  771.7651        115     287      50 14765.80          neutral
3 19178  818.1564         99     208      77 14457.90         Positive
4 19179   50.0000         90     467     107 14033.63         NEGATIVE
5 19180  820.2134        130     305     148 14881.60         NEGATIVE
6 19181  794.6938         92     490     111 14127.73         Positive
  Demand_Index Supply_Index Rainfall Temperature Political_Stability
1          159          188      124    26.44452                   9
2          131          182       74    32.59184                   3
3          198          143      118    26.40042                   2
4          144          185      158    25.83467                   4
5          198           96       97    27.25106                   3
6          132          177      192    27.59168                   7
```

5.3.3 Fixing Text Formatting Issues

- Ensure uniform formats for dates, numbers, and text
- Use regex to clean text data

Python Code

```
df_duplicates['Market_Sentiment'] = df_duplicates['Market_Sentiment'].str.capitalize().str.st
```

R Code

```
# Remove duplicate rows
df_duplicates <- df_dropna %>% distinct()

print(head(df_duplicates))
```

```
   Date CPO_Price Production Exports Imports  USD_IDR Market_Sentiment
1 19176  868.5479        102     244      60 14442.79         Positive
2 19177  771.7651        115     287      50 14765.80          neutral
3 19178  818.1564         99     208      77 14457.90         Positive
4 19179   50.0000         90     467     107 14033.63         NEGATIVE
5 19180  820.2134        130     305     148 14881.60         NEGATIVE
6 19181  794.6938         92     490     111 14127.73         Positive
  Demand_Index Supply_Index Rainfall Temperature Political_Stability
1          159          188      124    26.44452                    9
2          131          182       74    32.59184                    3
3          198          143      118    26.40042                    2
4          144          185      158    25.83467                    4
5          198           96       97    27.25106                    3
6          132          177      192    27.59168                    7
```

5.3.4 Handling Outliers

- Visualize with boxplots to detect anomalies
- Use IQR (Interquartile Range) or Z-score method

Python Code

```
# Remove outliers using IQR
Q1 = df_duplicates['CPO_Price'].quantile(0.25)
Q3 = df_duplicates['CPO_Price'].quantile(0.75)
IQR = Q3 - Q1
df_oulier1 = df_duplicates[(df_duplicates['CPO_Price'] >= (Q1 - 1.5 * IQR)) &
     (df_duplicates['CPO_Price'] <= (Q3 + 1.5 * IQR))]
```

```
# Remove outliers using Z-Score for 'USD_IDR'
from scipy import stats
df_outlier2 = df_duplicates[(np.abs(stats.zscore(df_duplicates['USD_IDR'])) < 3)]
```

R Code

```
# Remove outliers using IQR
Q1 <- quantile(df_duplicates$CPO_Price, 0.25)
Q3 <- quantile(df_duplicates$CPO_Price, 0.75)
IQR <- Q3 - Q1
df_outlier1 <- df_duplicates[df_duplicates$CPO_Price >= (Q1 - 1.5 * IQR) &
                             df_duplicates$CPO_Price <= (Q3 + 1.5 * IQR), ]
```

```r
# Remove outliers using Z-Score for 'USD_IDR'
z_scores <- scale(df_duplicates$USD_IDR)
df_outlier2 <- df_duplicates[abs(z_scores) < 3, ]
```

5.3.5 Standardizing & Normalizing

Apply Min-Max Scaling or Standard Scaling for statistical models and ML

Python Code

```python
# Normalize 'Temperature' & 'Rainfall'
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df[['Temperature', 'Rainfall']] = scaler.fit_transform(df[['Temperature', 'Rainfall']])
```

```python
# Standardize 'Demand_Index' & 'Supply_Index'
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df[['Demand_Index', 'Supply_Index']] = scaler.fit_transform(df[['Demand_Index', 'Supply_Index
```

R Code

```r
# Normalize 'Temperature' & 'Rainfall' using Min-Max Scaling
normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}

df$Temperature <- normalize(df$Temperature)
df$Rainfall <- normalize(df$Rainfall)
```

```r
# Standardize 'Demand_Index' & 'Supply_Index' using Z-score Standardization
df$Demand_Index <- scale(df$Demand_Index)
df$Supply_Index <- scale(df$Supply_Index)
```

5.3.6 Ensure Dataset

- Drop unnecessary columns
- Filter data that does not align with the analysis context

Python Code

```python
# Ensure there are no missing values
df.dropna(inplace=True)

# Convert categorical variables into numerical format (if needed for regression)
df = pd.get_dummies(df, columns=['Market_Sentiment'], drop_first=True)
# Final Cleaned Dataset
print(df.head())
```

```
                       Date  ...  Market_Sentiment_neutral
0 2022-07-04 13:54:00.561316  ...                      True
1 2022-07-05 13:54:00.561316  ...                     False
2 2022-07-06 13:54:00.561316  ...                     False
3 2022-07-07 13:54:00.561316  ...                      True
4 2022-07-08 13:54:00.561316  ...                      True

[5 rows x 13 columns]
```

R Code

```r
# Ensure there are no missing values
df <- na.omit(df)

# Convert categorical variables into numerical format (if needed for regression)
df <- model.matrix(~ Market_Sentiment - 1, data = df) %>% as.data.frame()

# Bind back to original dataframe (excluding the original categorical column)
df <- cbind(df, df[, !colnames(df) %in% "Market_Sentiment"])

# Print first few rows of the cleaned dataset
head(df)
```

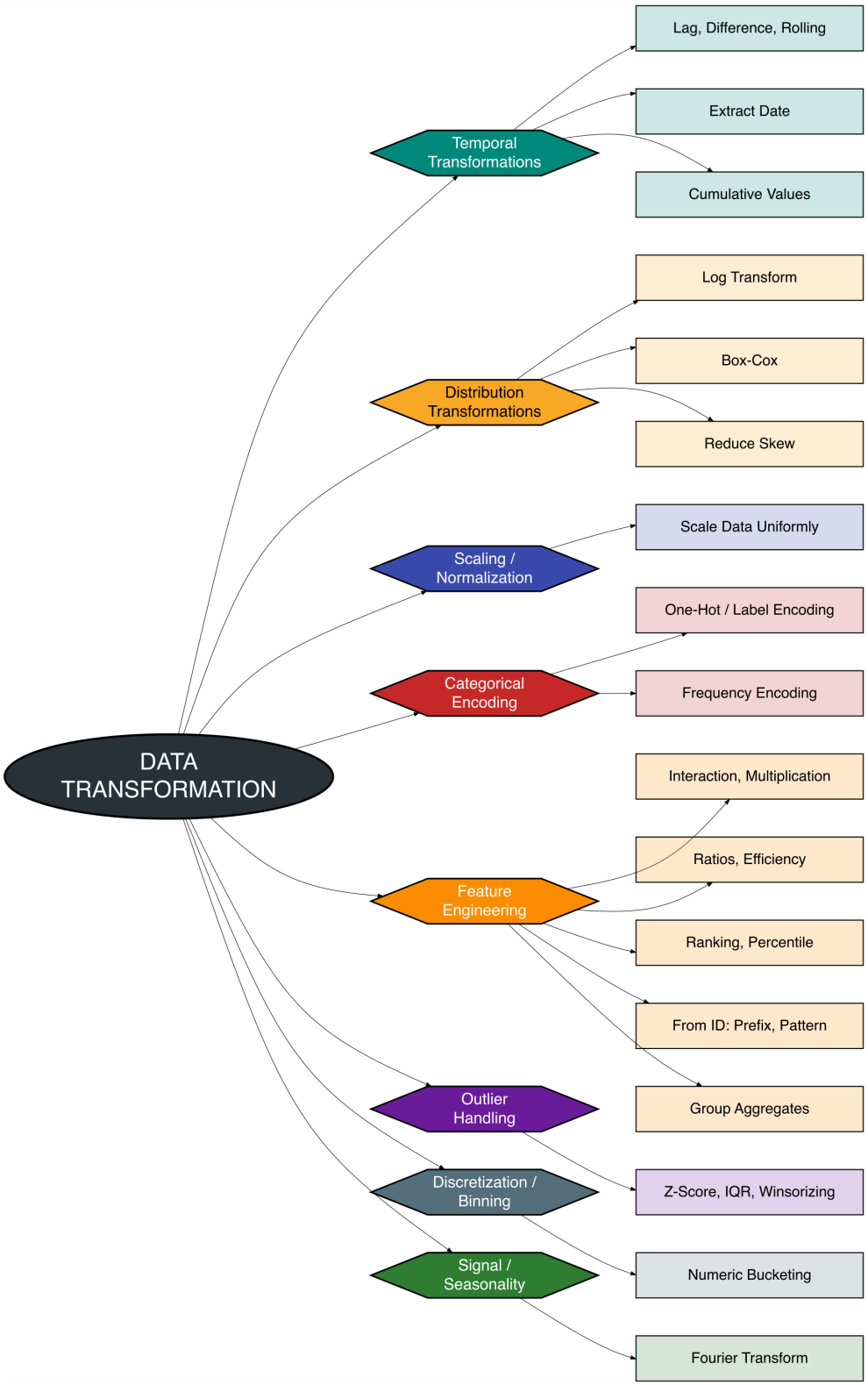
```
  Market_SentimentNEGATIVE Market_Sentimentneutral Market_SentimentPositive
1                        0                       0                        1
2                        0                       1                        0
3                        0                       0                        1
4                        1                       0                        0
5                        1                       0                        0
6                        0                       0                        1
  Market_SentimentNEGATIVE Market_Sentimentneutral Market_SentimentPositive
1                        0                       0                        1
2                        0                       1                        0
3                        0                       0                        1
4                        1                       0                        0
5                        1                       0                        0
6                        0                       0                        1
```

Chapter 6

Data-Transformation

Data transformation is a crucial step in the data analysis process, aiming to convert raw data into more useful and meaningful information. This process involves various techniques to adjust data formats, improve data quality, and convert it into a form that is easier to analyze and interpret. Data transformation also plays an important role in enhancing the accuracy and effectiveness of analytical models, especially when dealing with large and complex datasets.

In the following *mind map*, we will explore various data transformation techniques that can be used to improve data quality and generate better insights. This mind map includes major categories such as:


DATA TRANSFORMATION
Temporal Transformations
Lag, Difference, Rolling
Extract Date
Cumulative Values
Distribution Transformations
Log Transform
Box-Cox
Reduce Skew
Scaling / Normalization
Scale Data Uniformly
Categorical Encoding
One-Hot / Label Encoding
Frequency Encoding
Feature Engineering
Interaction, Multiplication
Ratios, Efficiency
Ranking, Percentile
From ID: Prefix, Pattern
Group Aggregates
Outlier Handling
Z-Score, IQR, Winsorizing
Discretization / Binning
Numeric Bucketing
Signal / Seasonality
Fourier Transform

Dummy Dataset

Below is a dummy business-related dataset created using R (demonstrated via Python). You can also generate it yourself in R using packages such as `dplyr`, `lubridate`, and `stringi`. This dataset simulates transaction data from a retail business with the following key elements:

Column	Description
`Transaction_ID`	Unique ID for each transaction
`Transaction_Date`	Transaction date
`Customer_ID`	Unique customer ID
`Product_Category`	Product category (Electronics, Clothing, etc.)
`Product_ID`	Product ID
`Quantity`	Number of items purchased
`Unit_Price`	Price per product unit
`Discount`	Transaction discount (0–0.3)
`Region`	Region (North, South, East, West)
`Sales_Channel`	Sales channel (Online / Offline)
`Total_Price`	Total price after discount

Copy CSV PDF Print Search:

Transformed Business Dataset

	Transaction_ID	Transaction_Date	Customer_ID	Product_Category	Product_ID	Quantity	Unit_Price	Discount	Region	Sales_Channel	Delivery_Time
1	7zmPHxF7XfN9	2021-07-14	BAl3Y7yxev	Clothing	P0370	2	15.18	0	North	Online	
2	y4bCY9pKTBWU	2020-11-16	TYY0h5C190	Electronics	P0185	5	10.22	0.15	West	Offline	
3	8k0B7XX19Ykf	2023-03-22	nUX640AaXg	Home	P0443	3	17.74	0.05	West	Online	
4	l8ahQz5YNOKz	2023-01-02	sBZyUSJLEP	Home	P0035	6	28.3	0.22	North	Offline	
5	kmufgw8wx5qk	2023-06-05	GMfVH2ZWNX	Groceries	P0375	3	11.91	0.13	North	Offline	
6	aI0KADT0mn7C	2023-03-15	YxqAmfTU9M	Clothing	P0447	1	5.43	0.07	North	Offline	
7	zu0iP1OBFuNl	2021-09-25	ifYw95qmoL	Groceries	P0345	2	13.37	0.3	West	Online	
8	l0Bz1iXOUzUy	2020-02-18	F5zL0GjDJh	Clothing	P0193	5	6.56	0.23	South	Online	
9	kgU3mL1c8BdA	2023-02-25	Zm9ROmGygf	Clothing	P0114	4	18.23	0.27	South	Offline	
10	MDp09wloF78Q	2023-08-19	fgldeSSYSL	Home	P0095	1	11.94	0.09	South	Online	
11	M8Wk6MDt6qq1	2020-01-24	y5n7T2vCfd	Electronics	P0309	2	21.06	0.2	South	Offline	
12	lH96iSOc84fn	2020-12-21	SS9WdTh6Gp	Books	P0402	4	9.38	0.01	North	Online	
13	NnorQ5sHloaf	2024-06-12	9l5PBRrmgl	Clothing	P0135	3	15.61	0.17	South	Online	
14	dEbUZoU5Sbu2	2020-06-13	A03t0cAGgC	Clothing	P0312	1	9.17	0.15	South	Offline	
15	o5Gqu5Y2GBZ4	2021-09-13	7hcyxq0KSE	Groceries	P0498	1	16.31	0.24	West	Online	
16	RYXFAjCauCpJ	2024-04-04	A9pnVHddZb	Electronics	P0098	2	18.49	0.09	East	Offline	

Showing 1 to 500 of 500 entries

6.1 Temporal Transformations

Temporal transformations refer to techniques used to manipulate and extract meaningful patterns from time-based data. These transformations are essential when dealing with time series or datasets containing date and time information. The goal is to uncover trends, seasonal patterns, or lagged relationships that improve the predictive power of models.

Common temporal transformation techniques include:

- **Lag, Difference, and Rolling:** Capture temporal dependencies and smooth fluctuations.
- **Extract Hour, Day, Weeek, Month, Year:** Derive time-based features that often influence behavior or outcomes.
- **Cumulative Sum / Count / Mean:** Track running totals or averages over time for trend analysis.

These methods help structure time-based data in a way that enables deeper insights and improved decision-making.

6.1.1 Lag, Diff, Rolling

```
library(knitr)

# (Contoh untuk Quantity, berdasarkan tanggal transaksi)
Tempral <- data_bisnis %>%
  arrange(Customer_ID, Transaction_Date) %>%
  group_by(Customer_ID) %>%
  mutate(
    Lag_Quantity = lag(Quantity),
    Diff_Quantity = Quantity - lag(Quantity),
    RollingMean_3 = zoo::rollapply(Quantity, width = 3, FUN = mean, fill = NA, align = 'ri
  ) %>%
  ungroup()

# Tampilkan 5 baris pertama
kable(head(Tempral,5))
```

Transaction_ID	Transaction_Date	Customer_ID	Product_Category	Product_ID	Quantity	Unit_Price	Discount	Region	Sales_Channel	Delivery_Time	Total_Price	Lag_Quantity	Diff_Quantity	RollingMean_3
[illegible]	2023-02-09	[illegible]	Groceries	P0253	3	13.11	0.07	East	Offline	10	36.58	NA	NA	NA
[illegible]	2020-11-30	[illegible]	Books	P0113	3	15.90	0.24	North	Offline	3	36.25	NA	NA	NA
[illegible]	2020-03-18	[illegible]	Clothing	P0445	5	2.65	0.20	North	Offline	8	10.60	NA	NA	NA
[illegible]	2023-05-19	[illegible]	Electronics	P0298	3	9.12	0.20	East	Online	8	21.89	NA	NA	NA
[illegible]	2020-07-07	[illegible]	Clothing	P0353	3	1.31	0.12	West	Online	2	3.46	NA	NA	NA

6.1.2 Extract Date

```
Extract <- data_bisnis %>%
  mutate(
    Day_of_Week = weekdays(Transaction_Date),
    Month = month(Transaction_Date, label = TRUE),
    Year = year(Transaction_Date),
    Is_Weekend = ifelse(Day_of_Week %in% c("Saturday", "Sunday"), 1, 0),
    Region = as.factor(Region),
```

```
    Product_Category = as.factor(Product_Category)
  ) %>%
  bind_cols(
    as.data.frame(model.matrix(~ Region - 1, data = .)),
    as.data.frame(model.matrix(~ Product_Category - 1, data = .))
  )

# Tampilkan 5 baris pertama
kable(head(Extract,5))
```

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
7zm[illegible]	2021-07-14	[illegible]	[illegible]	North	Online	[illegible]	30.36	[illegible]	[illegible]	[illegible]	0	1	0	0	0	1	0	0	0
y4b[illegible]	2020-11-16	[illegible]	[illegible]	West	Offline	[illegible]	43.44	[illegible]	[illegible]	[illegible]	0	0	0	1	0	0	1	0	0
8k0[illegible]	2023-03-22	[illegible]	[illegible]	West	Online	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	0	0	1	0	0	0	0	1
l8ah[illegible]	2023-01-02	[illegible]	[illegible]	North	Offline	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	1	0	0	0	0	0	0	1
kmu[illegible]	2023-06-05	[illegible]	[illegible]	North	Offline	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	1	0	0	0	0	0	1	0

6.1.3 Cumulative Values

```
Cumulative <- data_bisnis %>%
  group_by(Customer_ID) %>%
  mutate(
    Cumulative_Quantity = cumsum(Quantity),
    Cumulative_Spend = cumsum(Total_Price),
    Cumulative_AvgPrice = cummean(Unit_Price)
  ) %>%
  ungroup()

# Tampilkan 5 baris pertama
kable(head(Cumulative,5))
```

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	Region	Sales_Channel	[illegible]	Total_Price	[illegible]	[illegible]	[illegible]_AvgPrice
7zmP[illegible]	2021-07-14	[illegible]	[illegible]	Clothing	2	15.18	0.00	North	Online	5	30.36	2	30.36	15.18

Transaction_ID	Transaction_Date	Customer_ID	Product_ID	Product_Category	Quantity	Unit_Price	Discount	Region	Sales_Channel	Delivery_Time	Total_Price	Cumulative_Quantity	Cumulative_Spent	[illegible]_AvgPrice
[illegible]	[illegible]-11-16	[illegible]	[illegible]	Electronics	5	10.22	0.15	West	Offline	2	43.44	5	43.44	10.22
[illegible]	[illegible]-03-22	[illegible]	[illegible]	Home	3	17.74	0.05	West	Online	8	50.56	3	50.56	17.74
[illegible]	2023-01-02	[illegible]	[illegible]	Home	6	28.30	0.22	North	Offline	8	132.44	6	132.44	28.30
[illegible]	2023-06-05	[illegible]	[illegible]	Groceries	3	11.91	0.13	North	Offline	7	31.09	3	31.09	11.91

6.2 Distribution Tranformations

Distribution transformations are techniques used to modify the shape of a dataset's distribution, making it more suitable for analysis or modeling. Many statistical models assume that data follows a normal distribution; therefore, transforming skewed or irregularly distributed data can lead to better model performance and more reliable insights.

Key distribution transformation techniques include:

- **Log Transform:** Compresses large values and reduces right-skewness.
- **Box-Cox and Yeo-Johnson:** Flexible transformations that stabilize variance and normalize data.
- **Variance Stabilization / Skew Reduction:** Improves symmetry and homoscedasticity in datasets.

These transformations are particularly useful when dealing with highly skewed data, outliers, or heteroscedastic variance, allowing for more robust and interpretable analyses.

6.2.1 Log Transform

```
min_positive <- min(data_bisnis$Total_Price[data_bisnis$Total_Price > 0])

Log <- data_bisnis %>%
  mutate(
    Safe_Total_Price = ifelse(Total_Price <= 0, min_positive, Total_Price),
    Log_Total_Price = log1p(Safe_Total_Price)
  )

# Tampilkan 5 baris pertama
kable(head(Log,5))
```

Transaction_ID	Transaction_Date	Customer_ID	Product_Category	Product_ID	Quantity	Unit_Price	Discount	Region	Sales_Channel	Delivery_Time	Total_Price	Safe_Total_Price	Log_Total_Price
[illegible]	2021-07-14	[illegible]	Clothing	P0370	2	15.18	0.00	North	Online	5	30.36	30.36	3.445533
[illegible]	2020-11-16	[illegible]	Electronics	P0185	5	10.22	0.15	West	Offline	2	43.44	43.44	3.794140
[illegible]	2023-03-22	[illegible]	Home	P0443	3	17.74	0.05	West	Online	8	50.56	50.56	3.942746
[illegible]	2023-01-02	[illegible]	Home	P0035	6	28.30	0.22	North	Offline	8	132.44	132.44	4.893652
[illegible]	2023-06-05	[illegible]	Groceries	P0375	3	11.91	0.13	North	Offline	7	31.09	31.09	3.468545

6.2.2 Box-Cox

```r
library(bestNormalize)
Box_Cox <- data_bisnis %>%
  mutate(
    YeoJ_Quantity = bestNormalize(Quantity)$x.t,
    YeoJ_TotalPrice = bestNormalize(Total_Price)$x.t
  )

# Tampilkan 5 baris pertama
kable(head(Box_Cox,5))
```

Transaction_ID	Transaction_Date	Customer_ID	Product_Category	Product_ID	Quantity	Unit_Price	Discount	Region	Sales_Channel	Delivery_Time	Total_Price	YeoJ_Quantity	YeoJ_TotalPrice
[illegible]	2021-07-14	[illegible]	Clothing	P0370	2	15.18	0.00	North	Online	5	30.36	-0.5858148	-0.0570735
[illegible]	2020-11-16	[illegible]	Electronics	P0185	5	10.22	0.15	West	Offline	2	43.44	0.9985763	0.3180569
[illegible]	2023-03-22	[illegible]	Home	P0443	3	17.74	0.05	West	Online	8	50.56	0.0025066	0.6480661
[illegible]	2023-01-02	[illegible]	Home	P0035	6	28.30	0.22	North	Offline	8	132.44	1.5141020	0.0149565
[illegible]	2023-06-05	[illegible]	Groceries	P0375	3	11.91	0.13	North	Offline	7	31.09	0.0025066	-0.0282888

6.2.3 Stabilize Variance

6.3 Scaling & Normalization

Scaling and normalization are essential preprocessing steps in data transformation that ensure all numerical features contribute equally to model performance. These techniques adjust the range or distribution of data values, especially when features have different units or magnitudes.

Common techniques include:

- **Min-Max Scaling:** Rescales values to a fixed range (typically 0 to 1).
- **Standardization (Z-score):**Centers data around zero with unit variance.
- **Robust Scaling:** Uses median and IQR, making it more resistant to outliers.

Applying these transformations improves the convergence and accuracy of algorithms such as gradient descent and distance-based models (e.g., k-NN, SVM), which are sensitive to feature magnitude.

```r
# Z-Score Standardization
data_std <- data_bisnis %>%
  mutate(
    Quantity_Std = scale(Quantity),
    Unit_Price_Std = scale(Unit_Price)
  )

# Tampilkan 5 baris pertama
kable(head(data_std,5))
```

Transaction_ID	Transaction_Date	Customer_ID	Product_Category	Product_ID	Quantity	Unit_Price	Discount	Region	Sales_Channel	Delivery_Time	Total_Price	Quantity_Std	Unit_Price_Std
7zmPH[illegible]	2021-07-14	[illegible]XfN9Al3Y	Clothing	P0370	2	15.18	0.00	North	Online	5	30.36	-0.6497275	0.2623825
y4bCY[illegible]	[illegible]-11-16	[illegible]TBWYUY0E[illegible]	Electronics	P0185	5	10.22	0.15	West	Offline	2	43.44	1.0600817	-0.6056456
8k0B7X[illegible]	2023-03-22	[illegible]YkfiUX64[illegible]	Home	P0443	3	17.74	0.05	West	Online	8	50.56	-0.0797911	0.7103970
l8ahQz[illegible]	2023-01-02	[illegible]OKszBZyU[illegible]	Home	P0035	6	28.30	0.22	North	Offline	8	132.44	1.6300181	[illegible]584570
kmufgw[illegible]	2023-06-05	[illegible]qkGMfV[illegible]	Groceries	P0375	3	11.91	0.13	North	Offline	7	31.09	-0.0797911	-[illegible]098860

```r
# Min-Max Normalization
data_norm <- data_bisnis %>%
  mutate(
    Quantity_Norm = (Quantity - min(Quantity)) / (max(Quantity) - min(Quantity)),
```

```r
    Total_Price_Norm = (Total_Price - min(Total_Price)) / (max(Total_Price) - min(Total_Price
  )

# Tampilkan 5 baris pertama
kable(head(data_norm,5))
```

Transaction_ID	Transaction_Date	Customer_ID	Product_Category	Product_ID	Quantity	Unit_Price	Discount	Region	Sales_Channel	Delivery_Time	Total_Price	Quantity_Norm	Total_Price_Norm
7zmPH[illegible]	2027-07-14	N9Al3Y[illegible]	Clothing	P0370	2	15.18	0.00	North	Online	5	30.36	0.2222222	[illegible]2848526
y4bCY[illegible]	2020-11-16	WYY0E5C[illegible]	Electronics	P0185	5	10.22	0.15	West	Offline	2	43.44	0.5555556	[illegible]587259
8k0B7X[illegible]	2023-03-22	mUX640AaXg	Home	P0443	3	17.74	0.05	West	Online	8	50.56	0.3333333	[illegible]989382
l8ahQz[illegible]	2023-01-02	BZyUSJLE	Home	P0035	6	28.30	0.22	North	Offline	8	132.44	0.6666667	[illegible]613803
kmufgw[illegible]	2023-06-05	GMfVH2ZWNX	Groceries	P0375	3	11.91	0.13	North	Offline	7	31.09	0.3333333	[illegible]889755

6.4 Categorical Encoding

Categorical encoding transforms non-numeric (categorical) variables into a numerical format that can be used by machine learning algorithms. Since most models cannot handle text or label data directly, encoding is crucial for integrating categorical features into predictive modeling.

Popular encoding techniques include:

- **One-Hot Encoding:** Creates binary columns for each category, useful for nominal data.
- **Label Encoding:** Assigns each category a unique integer, best for ordinal variables.
- **Frequency Encoding:** Replaces categories with their frequency or count in the dataset.

Choosing the right encoding method helps preserve the meaning of categorical data while maintaining model performance and interpretability.

6.4.1 One-hot Encoding

```r
library(fastDummies)
one_hot <- dummy_cols(data_bisnis,
                      select_columns = c("Region",
                                         "Sales_Channel",
```

```
                                      "Product_Category"),
                         remove_first_dummy = TRUE,
                        remove_selected_columns = TRUE)

# Tampilkan 5 baris pertama
kable(head(one_hot,5))
```

Transaction_ID	Transaction_Date	Customer_ID	Product_ID	Quantity	Unit_Price	Discount	Delivery_Time	Total_Price	Region_North	Region_South	Region_West	Sales_Channel_Online	Product_Category_Clothing	Product_Category_Electronics	Product_Category_Groceries	Product_Category_Hom
[illegible]	2021-07-14	[illegible]	[illegible]	[illegible]	15.18	0.00	5	30.36	1	0	0	1	1	0	0	0
[illegible]	2020-11-16	[illegible]	[illegible]	[illegible]	10.22	0.15	2	43.44	0	0	1	0	0	1	0	0
[illegible]	2023-03-22	[illegible]	[illegible]	[illegible]	17.74	0.05	8	50.56	0	0	1	1	0	0	0	1
[illegible]	2023-01-02	[illegible]	[illegible]	[illegible]	28.30	0.22	8	132.44	1	0	0	0	0	0	0	1
[illegible]	2023-06-05	[illegible]	[illegible]	[illegible]	[illegible]	0.13	7	31.09	1	0	0	0	0	0	1	0

6.4.2 Frequency Encoding

```
freq_enc <- function(col) {
  tab <- table(col)
  return(as.numeric(tab[col]) / length(col))
}

Frequency <- data_bisnis %>%
  mutate(
    Region_freq = freq_enc(Region),
    Product_Category_freq = freq_enc(Product_Category)
  )

# Tampilkan 5 baris pertama
kable(head(Frequency,5))
```

Transaction_ID	Transaction_Date	Customer_ID	Product_Category	Product_ID	Quantity	Unit_Price	Discount	Region	Sales_Channel	Delivery_Time	Total_Price	Region_freq	Product_Category_freq
[illegible]	2021-07-14	[illegible]	Clothing	P0370	2	15.18	0.00	North	Online	5	30.36	0.282	0.222

Transaction_ID	Transaction_Date	Customer_ID	Product_ID	Product_Category	Quantity	Unit_Price	Discount	Region	Sales_Channel	Delivery_Time	Total_Price	Region_freq	Product_Category_freq
[illegible]	2020-11-16	[illegible]	P0185	Electronics	5	10.22	0.15	West	Offline	2	43.44	0.240	0.176
[illegible]	2023-03-22	[illegible]	P0443	Home	3	17.74	0.05	West	Online	8	50.56	0.240	0.220
[illegible]	2023-01-02	[illegible]	P0035	Home	6	28.30	0.22	North	Offline	8	132.44	0.282	0.220
[illegible]	2023-06-05	[illegible]	P0375	Groceries	3	11.91	0.13	North	Offline	7	31.09	0.282	0.192

6.5 Feature Engineering

Feature engineering is the process of creating new input features from existing data to improve model performance. It involves extracting, transforming, or combining variables in ways that make patterns more accessible to machine learning algorithms.

Effective feature engineering often leads to significant boosts in model accuracy and interpretability, making it one of the most impactful steps in a data pipeline. Let's consider the following feature engineering examples:

1. Product of Features, Crossed Terms
2. Price per Unit, Efficiency
3. Ranking, Percentile
4. From IDs: Prefix, Length, Pattern
5. Avg, Sum, Count by Group

```r
Feature_Eng <- data_bisnis %>%
  # 1, 2, 3, 4, 5: Row-wise transformations
  mutate(
    Price_per_Unit = Total_Price / Quantity,           # 2. Price per Unit
    Efficiency = Quantity / Delivery_Time,             # 2. Efficiency
    Feature_Interaction = Quantity * Discount,         # 1. Product of Features
    Cross_Term = paste0(Product_Category, "_", Region),# 1. Crossed Term
    ID_Prefix = substr(Product_ID, 1, 2),              # 4. ID Prefix
    ID_Length = nchar(Product_ID),                     # 5. ID Length
    ID_HasPattern = grepl("^PR", Product_ID),          # 5. Pattern Detection
    Discount_Level = ntile(Discount, 4),               # 3. Discount Percentile
    Sales_Rank = rank(-Total_Price, ties.method = "min") # 3. Sales Ranking
  ) %>%
  # 5: Group-wise aggregations
  group_by(Region) %>%
  mutate(
    Avg_Quantity_Region = mean(Quantity, na.rm = TRUE),
    Sum_Sales_Region = sum(Total_Price, na.rm = TRUE),
    Count_Product_Region = n()
```

```
  ) %>%
  ungroup()

# Tampilkan 5 baris pertama
kable(head(Feature_Eng,5))
```

[illegible]												
7zm[illegible]	2021-07-14	[illegible]	15.1[illegible]	[illegible]	North / Online	30.3[illegible]	[illegible]	Clothing	[illegible]	North	FALSE	2532.99[illegible]
y4b[illegible]	2020-11-16	[illegible]	10.2[illegible]	[illegible]	West / Offline	43.4[illegible]	[illegible]	Electronics	[illegible]	West	FALSE	1573.11[illegible]
8k0[illegible]	2023-03-22	[illegible]	17.7[illegible]	[illegible]	West / Online	50.5[illegible]	[illegible]	Home	P0_5	West	FALSE	1213.11[illegible]
l8ah[illegible]	2023-01-02	[illegible]	28.3[illegible]	[illegible]	North / Offline	132[illegible]	[illegible]	Home	P0_5	North	FALSE	4 2.99[illegible]
kmu[illegible]	2023-06-05	[illegible]	[illegible]	[illegible]	North / Offline	31.0[illegible]	[illegible]	Groceries	[illegible]	North	FALSE	2462.99[illegible]

6.6 Outlier Handling

Outlier detection is an essential part of data preprocessing, especially when working with statistical modeling or machine learning. **Outliers** are data points that differ significantly from the rest of the dataset and can adversely affect model performance or assumptions. Identifying and handling outliers ensures more accurate predictions and insights. There are two common outlier detection methods:

1. **Z-score Method**

2. **Interquartile Range (IQR) Method**

Each method has its own strengths and is suited for different types of data. Let's explore both.

6.6.1 Z-score Method

The **Z-score method** measures how many standard deviations a data point is from the mean. It is commonly used for identifying **outliers** in normally distributed data. If the Z-score of a data point exceeds a threshold (commonly 3 or -3), it can be considered an outlier.

Formula:

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

Where:

- X is the data point.
- μ is the mean of the data.
- σ is the standard deviation.

```r
# Z-score method for detecting outliers
z_scores <- scale(data_bisnis$Quantity)
z_scores_outliers <- data_bisnis %>%
  mutate(Outlier_Flag = ifelse(abs(z_scores) > 3, "Outlier", "Normal"))

# Tampilkan 5 baris pertama
kable(head(z_scores_outliers,5))
```

Transaction_ID	Transaction_Date	Customer_ID	Product_Category	Product_ID	Quantity	Unit_Price	Discount	Region	Sales_Channel	Delivery_Time	Total_Price	Outlier_Flag
7zmPHx[illegible]	[illegible]-07-14	[illegible]fN9BAl3Y[illegible]	Clothing	P0370	2	15.18	0.00	North	Online	5	30.36	Normal
y4bCY9[illegible]	[illegible]-11-16	[illegible]BWTUYY0[illegible]	Electronics	P0185	5	10.22	0.15	West	Offline	2	43.44	Normal
8k0B7X[illegible]	2023-03-22	[illegible]kf nUX64[illegible]	Home	P0443	3	17.74	0.05	West	Online	8	50.56	Normal
l8ahQz5[illegible]	[illegible]-01-02	[illegible]KzsBZyU[illegible]	Home	P0035	6	28.30	0.22	North	Offline	8	132.44	Normal
kmufgw[illegible]	[illegible]-06-05	[illegible]qk GMfV[illegible]	Groceries	P0375	3	11.91	0.13	North	Offline	7	31.09	Normal

6.6.2 IQR method

The **IQR method** is a non-parametric approach to detecting outliers. It uses the quartiles (25th percentile, $Q1$ and 75th percentile, $Q3$) to determine the spread of the middle 50% of the data. Any data point outside the range defined by:

Lower Bound:

$$\text{Lower Bound} = Q1 - 1.5 \times \text{IQR}$$

Upper Bound:

$$\text{Upper Bound} = Q3 + 1.5 \times \text{IQR}$$

is considered an outlier.

IQR Calculation:

$$\text{IQR} = Q3 - Q1$$

Where:

- $Q1$: 25th percentile.

- $Q3$: 75th percentile.

```
# IQR method for detecting and removing outliers
Q1 <- quantile(data_bisnis$Total_Price, 0.25)  # Calculate the 25th percentile
Q3 <- quantile(data_bisnis$Total_Price, 0.75)  # Calculate the 75th percentile
IQR_val <- Q3 - Q1                             # Compute the IQR

# Remove outliers outside the IQR range
IQR_outliers <- data_bisnis %>%
filter(Total_Price > (Q1 - 1.5 * IQR_val) & Total_Price < (Q3 + 1.5 * IQR_val))

# Tampilkan 5 baris pertama
kable(head(IQR_outliers,5))
```

Transaction_ID	Transaction_Date	Customer_ID	Product_Category	Product_ID	Quantity	Unit_Price	Discount	Region	Sales_Channel	Delivery_Time	Total_Price
7zmPHxUTXfN9	2022-07-14	BAl3Y7ykeh	Clothing	P0370	2	15.18	0.00	North	Online	5	30.36
y4bCY9pk0BWU	2021-11-16	TYY0h5C190	Electronics	P0185	5	10.22	0.15	West	Offline	2	43.44
8k0B7XXY-kf	2023-03-22	nUX640HaXg	Home	P0443	3	17.74	0.05	West	Online	8	50.56
kmufgw8wx5qk	2023-06-05	GMfVH2ZWNX	Groceries	P0375	3	11.91	0.13	North	Offline	7	31.09
aI0KADTomn7C	2023-03-15	YxqAmCTU9M	Clothing	P0447	1	5.43	0.07	North	Offline	9	5.05

Outliers Removed: This code removes any data points where the Total_Price is outside the range defined by $Q1 - 1.5 \times \text{IQR}$ and $Q3 + 1.5 \times \text{IQR}$.

6.7 Discretization

Discretization is the process of converting continuous numerical variables into discrete categories. This technique is useful for: - Simplifying models. - Identifying patterns. - Making data more interpretable for statistical analysis or visualization.

6.7.1 Fixed-Width Binning

Divides data into intervals with fixed width.

```
Quantity_Bin <- data_bisnis %>%
  mutate(
    Quantity_Bin = cut(Quantity, breaks = c(0, 2, 4, 6, Inf), labels = c("Low", "Medium",
  )

# Tampilkan 5 baris pertama
kable(head(Quantity_Bin,5))
```

Transaction_ID	Transaction_Date	Customer_ID	Product_Category	Product_ID	Quantity	Unit_Price	Discount	Region	Sales_Channel	Delivery_Time	Total_Price	Quantity_Bin
7zmPHxF7X	2023-07-14	fN9BAl3Y	Clothing	P0370	2	15.18	0.00	North	Online	5	30.36	Low
y4bCY9pK0	2020-11-16	TBWTUYY0h5C190	Electronics	P0185	5	10.22	0.15	West	Offline	2	43.44	High
8k0B7X2	2023-03-22	Ykf nUX64HAmXg	Home	P0443	3	17.74	0.05	West	Online	8	50.56	Medium
l8ahQz5	2023-01-02	OKzsBZyUSJLEP	Home	P0035	6	28.30	0.22	North	Offline	8	132.44	High
kmufgw8	2023-06-05	5qk GMfVIE2ZWNX	Groceries	P0375	3	11.91	0.13	North	Offline	7	31.09	Medium

6.7.2 Quantile Binning

Divides data into bins with equal number of observations using quantiles (e.g., quartiles, deciles).

```
Quantile_Bin <- data_bisnis %>%
  mutate(
    Price_Bin_Quantile = ntile(Total_Price, 4)  # Quartile binning
  )

# Tampilkan 5 baris pertama
kable(head(Quantile_Bin,5))
```

Transaction_ID	Transaction_Date	Customer_ID	Product_Category	Product_ID	Quantity	Unit_Price	Discount	Region	Sales_Channel	Delivery_Time	Total_Price	Price_Bin_Quantile
7zmPHxF7X	2023-07-14	fN9BAl3Y	Clothing	P0370	2	15.18	0.00	North	Online	5	30.36	2
y4bCY9pK0	2020-11-16	TBWTUYY0h5C190	Electronics	P0185	5	10.22	0.15	West	Offline	2	43.44	3
8k0B7X2	2023-03-22	YkfnUX64HAmXg	Home	P0443	3	17.74	0.05	West	Online	8	50.56	4
l8ahQz5	2023-01-02	OKzsBZyUSJLEP	Home	P0035	6	28.30	0.22	North	Offline	8	132.44	4
kmufgw8	2023-06-05	5qk GMfVIE2ZWNX	Groceries	P0375	3	11.91	0.13	North	Offline	7	31.09	3

6.7.3 Custom Binning

Uses business rules or custom logic to define bins.

```
Custom_Bin <- data_bisnis %>%
  mutate(
    Discount_Level = case_when(
      Discount == 0 ~ "No Discount",
      Discount <= 0.1 ~ "Low",
      Discount <= 0.2 ~ "Medium",
      TRUE ~ "High"
```

```
    )
  )

# Tampilkan 5 baris pertama
kable(head(Custom_Bin,5))
```

Transaction_ID	Transaction_Date	Customer_ID	Product_Category	Product_ID	Quantity	Unit_Price	Discount	Region	Sales_Channel	Delivery_Time	Total_Price	Discount_Level
7zmPHxF7XfN9	2023-07-14	BAl3YC	Clothing	P0370	2	15.18	0.00	North	Online	5	30.36	No Discount
y4bCY9pKI	2020-11-16	BWTUYY0h5C190	Electronics	P0185	5	10.22	0.15	West	Offline	2	43.44	Medium
8k0B7X2Ykf	2023-03-22	nUX64HAmXg	Home	P0443	3	17.74	0.05	West	Online	8	50.56	Low
l8ahQz5	2023-01-02	OKzsBZyUHJLEP	Home	P0035	6	28.30	0.22	North	Offline	8	132.44	High
kmufgw8wz5qk	2023-06-05	GMfVIG2ZVWNX	Groceries	P0375	3	11.91	0.13	North	Offline	7	31.09	Medium

6.8 Seasonality

Seasonality refers to periodic fluctuations in data that occur at regular time intervals, such as daily, monthly, or yearly patterns. In business and economic data, seasonal trends often reflect consumer behavior, holidays, weather patterns, and other recurring events.

6.8.1 Annual Seasonality

To capture **yearly seasonality**, we can use sine and cosine transformations of the day of the year. This approach embeds cyclical patterns into the data, helping models learn time-dependent behaviors.

```
library(lubridate)
library(dplyr)

Seasonality <- data_bisnis %>%
  mutate(
    Year = year(Transaction_Date),
    DayOfYear = yday(Transaction_Date),
    DaysInYear = if_else(leap_year(Transaction_Date), 366, 365),
    sin_year = sin(2 * pi * DayOfYear / DaysInYear),
    cos_year = cos(2 * pi * DayOfYear / DaysInYear)
  )

# Tampilkan 5 baris pertama
kable(head(Seasonality,5))
```

Transaction_ID	Transaction_Date	Customer_ID	Product_ID	Product_Category	Quantity	Unit_Price	Discount	Region	Sales_Channel	Delivery_Time	Total_Price	Year	Day_Of_Year	Days_In_Year	sin_year	cos_year
[illegible]	2021-07-14	[illegible]	P0370	Clothing	2	15.18	0.00	North	Online	5	30.36	2021	195	365	-0.2135209	-0.9769385
[illegible]	2020-11-16	[illegible]	P0185	Electronics	5	10.22	0.15	West	Offline	2	43.44	2020	321	366	-0.6979442	0.7161522
[illegible]	2023-03-22	[illegible]	P0443	Home	3	17.74	0.05	West	Online	8	50.56	2023	81	365	0.9844738	0.1755315
[illegible]	2023-01-02	[illegible]	P0035	Home	6	28.30	0.22	North	Offline	8	132.44	2023	2	365	0.0344216	0.9994074
[illegible]	2023-06-05	[illegible]	P0375	Garden	3	11.91	0.13	North	Offline	7	31.09	2023	156	365	0.4405188	-0.8977434

6.8.2 Linearity Over Time

Sometimes, patterns evolve in a linear fashion over time. Adding a simple time index or extracting the year from dates can help capture long-term trends:

```r
Linearity <- Seasonality %>%
  mutate(
    Linearity = as.numeric(difftime(Transaction_Date, min(Transaction_Date), units = "days"))
  )

# Tampilkan 5 baris pertama
kable(head(Linearity,5))
```

Transaction_ID	Transaction_Date	Customer_ID	Product_ID	Product_Category	Quantity	Unit_Price	Discount	Region	Sales_Channel	Delivery_Time	Total_Price	Year	Day_Of_Year	Days_In_Year	sin_year	cos_year	Linearity
[illegible]	2021-07-14	[illegible]	P0370	Clothing	2	15.18	0.00	North	Online	5	30.36	2021	195	365	-0.2135209	-0.9769385	559
[illegible]	2020-11-16	[illegible]	P0185	Electronics	5	10.22	0.15	West	Offline	2	43.44	2020	321	366	-0.6979442	0.7161522	319
[illegible]	2023-03-22	[illegible]	P0443	Home	3	17.74	0.05	West	Online	8	50.56	2023	81	365	0.9844738	0.1755315	1175
[illegible]	2023-01-02	[illegible]	P0035	Home	6	28.30	0.22	North	Offline	8	132.44	2023	2	365	0.0344216	0.9994074	1096
[illegible]	2023-06-05	[illegible]	P0375	Garden	3	11.91	0.13	North	Offline	7	31.09	2023	156	365	0.4405188	-0.8977434	1250

6.8.3 Exponential Trends

Some trends grow or decay exponentially. For instance, rapid adoption or decay can be modeled with log or exponential transformations:

```
Exponential <- Linearity %>%
  mutate(
    Exp_Growth = exp(Linearity / 1000),
    Log_Growth = log(Linearity + 1)
  )

# Tampilkan 5 baris pertama
kable(head(Exponential,5))
```

TransactionID	Transaction_Date	CustomerID	ProductID	Product_Category	Quantity	Unit_Price	Discount	Region	Sales_Channel	[illegible]	Total_Amount	Year	Day_Of_Year	Days_In_Year	sin_year	cos_year	Linearity	Exp_Growth	Log_Growth
[illegible]	2021-07-14	[illegible]	[illegible]	[illegible]	[illegible]	15.18	0.00	North	Online	5	30.36	2021	195	365	[illegible]	[illegible]	559	[illegible]	[illegible]
[illegible]	2020-11-16	[illegible]	[illegible]	[illegible]	[illegible]	10.22	0.15	West	Offline	2	43.44	2020	321	366	-0.6979442	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	2023-03-22	[illegible]	[illegible]	[illegible]	[illegible]	17.74	0.05	West	Online	8	50.56	2023	81	365	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	2023-01-02	[illegible]	[illegible]	[illegible]	[illegible]	28.30	0.22	North	Offline	8	132.44	2023	2	365	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	2023-06-05	[illegible]	[illegible]	[illegible]	[illegible]	11.91	0.13	North	Offline	7	31.09	2023	156	365	[illegible]	-0.8977434	[illegible]	[illegible]	[illegible]

By combining cyclic seasonal patterns, linear trends, and exponential growth or decay, we can build rich feature sets that improve model performance in time-dependent data analysis.

- **Sine/Cosine:** capture cyclic behavior (e.g., yearly seasonality)
- **Time Index:** reflects linear trends
- **Exponential/Log:** model non-linear, accelerating, or decaying trends

6.9 Practicum

Your task is to demonstrate the various data transformation techniques that you have learned, including Temporal Transformations, Distribution Transformations, Scaling/Normalization, Categorical Encoding, Feature Engineering, Outlier Handling, Binning, Signal/Seasonality (Fourier), and other relevant transformations.

You will be divided into several groups, and each group will receive a dataset. Your objective is to apply the appropriate transformation techniques to the dataset, analyze the outcomes, and present your findings to the class.

6.9.1 Weather

Weather plays a critical role in shaping human activities, influencing agriculture, transportation, energy consumption, and even public health. Its variability across time and space makes it both fascinating and complex to study. As global awareness grows about climate change and extreme weather events, the need to understand and analyze weather data becomes increasingly important.

Copy CSV PDF Print Search:

	Observation_ID	Date	Location	Season	Temperature	Humidity	Rainfall	Wind_Speed
1	JpeDLWuzIJdl	2021-07-14	Jakarta	Dry Season	30.7	89.5	7.9	9.5
2	EF8r6hXBCqfr	2020-11-16	Bandung	Rainy Season	26.2	70.1	4.6	5.6
3	cov0TYDwyQoF	2023-03-22	Makassar	Transitional Season	27.5	80.7	11	13.1
4	aRB0N7xSEnfa	2023-01-02	Surabaya	Rainy Season	26.8	90.4	8.7	4.4
5	fjf9bvNshjHQ	2023-06-05	Medan	Dry Season	24.9	85.4	7.5	5.2
6	jnQGmBA0NZn0	2023-03-15	Jakarta	Transitional Season	31.1	68.2	3.2	4.5
7	sUbfPmjqfjtC	2021-09-25	Medan	Dry Season	23.7	99.7	3.8	11
8	yoaWv1iSk2g9	2020-02-18	Bandung	Rainy Season	26.3	60	21.1	1.1
9	FGaHFpkT8AuG	2023-02-25	Makassar	Rainy Season	26	68.3	5.2	7
10	sHtZghGKMMLh	2023-08-19	Surabaya	Dry Season	28.6	85.4	3.4	7.2
11	6B5RvRGDnF3N	2020-01-24	Surabaya	Rainy Season	31.8	60.7	7.4	10.1
12	uBLSx62e4JGm	2020-12-21	Bandung	Rainy Season	28.5	61.1	10.3	12.6
13	8fuliMQjGlVg	2024-06-12	Jakarta	Dry Season	31.1	84.3	7.5	9.2
14	Rce2mXR4pWwF	2020-06-13	Jakarta	Dry Season	29.3	82.8	11.8	13.3
15	lSt7kQpwGxf0	2021-09-13	Medan	Dry Season	25.1	69.8	10.4	5
16	h344y6Py1qdS	2024-04-04	Surabaya	Transitional Season	27.2	96.8	11.3	14.7
17	qpTzXjrxU7F8	2021-02-13	Makassar	Rainy Season	31.1	86.9	10.8	12.1

Showing 1 to 500 of 500 entries

In this section, we introduce weather as a dataset-rich domain, where data science and statistical techniques are essential for extracting insights. Before we can build predictive models or generate visualizations, we must prepare and transform raw weather data into meaningful features.

To effectively analyze weather data, follow these key steps:

6.9.1.1 Import and Inspect the Data

- Load weather data from CSV, API, or database.
- Examine structure, date formats, and missing values.

6.9.1.2 Clean the Data

- Handle missing values (e.g., fill, remove, or impute).
- Standardize units (e.g., convert temperatures to Celsius or Fahrenheit).
- Parse datetime fields into usable components (e.g., year, month, day, hour).

6.9.1.3 Feature Engineering

- Derive new features such as:
  - Daily, weekly, and monthly averages.
  - Temperature change (lag or difference).
  - Humidity Index or Heat Index.
  - Wind direction in degrees or categories.
- Create cyclical features for seasonality using sine and cosine transformations:

$$\text{sin\_year} = \sin\left(\frac{2\pi \cdot \text{DayOfYear}}{\text{DaysInYear}}\right)$$

$$\text{cos\_year} = \cos\left(\frac{2\pi \cdot \text{DayOfYear}}{\text{DaysInYear}}\right)$$

6.9.1.4 Categorization and Binning

- Group continuous variables (e.g., rainfall intensity) into bins or labels.
- Define weather categories (e.g., “sunny”, “cloudy”, “stormy”) based on thresholds.

6.9.1.5 Detect and Handle Outliers

- Use statistical methods to flag unusual values:
  - **Z-score method**: Mark values where $|z| > 3$
  - **IQR method**:
    - Calculate:

$$\text{IQR} = Q3 - Q1$$

$$\text{Lower Bound} = Q1 - 1.5 \times \text{IQR}$$

$$\text{Upper Bound} = Q3 + 1.5 \times \text{IQR}$$

- Decide whether to keep, transform, or remove outliers depending on context.

6.9.1.6 Temporal and Rolling Features

- Calculate rolling means or moving averages (e.g., 7-day temperature trends).
- Create lag variables to capture delayed effects (e.g., yesterday’s temperature).

6.9.1.7 Normalize or Scale Data

Apply standardization (Z-score) or Min-Max normalization to continuous variables for use in machine learning models.

6.9.2 General Health

General health encompasses the overall physical, mental, and social well-being of individuals and populations. With the increasing availability of health data—from electronic medical records and health surveys to wearable devices—data science plays a pivotal role in analyzing, predicting, and improving health outcomes.

Copy CSV PDF Print Search:

	Patient_ID	Date	Age	BMI	Blood_Pressure	Cholesterol	Glucose	Heart_Rate	Location	Health_Condition	Season
1	SLy5n7T2vCfd	2021-07-14	27	31.5	122.8	131.5	77.7	70	Makassar	Healthy	Dry Season
2	SS9WdTh6Gp9l	2020-11-16	63	26.9	119.9	212.8	128.1	82.2	Jakarta	Diabetes	Rainy Season
3	5PBRrmgIA03t	2023-03-22	72	18.2	146	158.5	100.3	73.9	Surabaya	Healthy	Transitional Season
4	0cAGgC7hcyxq	2023-01-02	60	19.9	121.2	220.9	103.4	79.1	Bandung	Diabetes	Rainy Season
5	0KSEA9pnVHdd	2023-06-05	40	32.5	109.4	229.8	91.9	67	Makassar	Healthy	Dry Season
6	Zba4dbAEtGwn	2023-03-15	71	27.4	126.2	209.3	102	56.3	Bandung	Obesity	Transitional Season
7	R8Qx2GZT0XQT	2021-09-25	74	17.1	111.3	117.8	103.5	78.5	Makassar	Healthy	Dry Season
8	4CDiQyhVv9KV	2020-02-18	44	25.3	108.7	140.5	78.8	72.1	Jakarta	Hypertension	Rainy Season
9	PPPsJBNOlqxa	2023-02-25	51	18.2	91.4	285	73.6	93.7	Jakarta	Healthy	Rainy Season
10	wsS2iEHE4Sh6	2023-08-19	33	17.6	124.6	265	77.5	72.7	Bandung	Healthy	Dry Season
11	dHw1Qj2Kba4S	2020-01-24	63	25.3	113.9	199.2	70.5	85.2	Bandung	Diabetes	Rainy Season
12	8iJeK256qAc2	2020-12-21	42	25	130.8	237.5	90.6	68.6	Bandung	Hypertension	Rainy Season
13	B1Cp6SdRoNK8	2024-06-12	45	19.6	108.7	152.1	121.4	72.5	Makassar	Obesity	Dry Season
14	qhNcbTreDrX4	2020-06-13	32	25.6	119.9	225.9	91.9	88.6	Makassar	Healthy	Dry Season
15	hkKpYRePsho5	2021-09-13	66	26.8	117.5	168.2	80.5	68.8	Makassar	Hypertension	Dry Season
16	LbjXwfGVvV5N	2024-04-04	55	19.2	124.7	165.1	98.3	77.6	Surabaya	Obesity	Transitional Season
17	BiYC2oc91bEX	2021-02-13	40	26.5	112.1	202.7	82.2	81.3	Bandung	Diabetes	Rainy Season

Showing 1 to 500 of 500 entries

Before we can apply machine learning models or conduct meaningful analyses, it's essential to clean and transform raw health data into structured, analyzable features. Instructions for General Health Data Transformation

6.9.2.1 Import and Inspect the Dataset

- Load data from sources like public health records, surveys (e.g., BRFSS, NHANES), or hospital databases.
- Inspect structure, data types, and missing values.

6.9.2.2 Clean the Data

- Convert categorical variables (e.g., gender, smoking status) into factors.
- Handle missing or inconsistent entries (e.g., impute BMI or age if missing).
- Standardize units (e.g., weight in kg, height in cm).

6.9.2.3 Feature Engineering

- Create health risk indicators from raw inputs:
  - Body Mass Index (BMI):

$$BMI = \frac{\text{Weight (kg)}}{\text{Height (m)}^2}$$

  - Age group classification: young, adult, elderly.
  - Chronic condition flag (e.g., has_diabetes = TRUE if glucose > threshold).

6.9.2.4 Categorization and Binning

- Convert numerical scores into health levels:
  - Blood pressure into "normal", "elevated", "high".
  - Physical activity minutes into "low", "moderate", "high".
- Create ordinal variables for satisfaction or self-rated health.

6.9.2.5 Detect and Handle Outliers

- Use Z-score or IQR to detect abnormal values in clinical metrics (e.g., blood pressure, cholesterol):

  IQR Calculation:

$$IQR = Q3 - Q1$$

$$\text{Outlier Thresholds} = [Q1 - 1.5 \times IQR,\ Q3 + 1.5 \times IQR]$$

- Consider domain knowledge before removing outliers (e.g., extremely high glucose may be valid for diabetic patients).

6.9.2.6 Temporal and Rolling Features

If longitudinal, compute:

- Changes in weight or BMI over time.
- Rolling averages of physical activity or vitals.

6.9.2.7 Encode Categorical Variables

Convert categories to numerical codes:

- One-hot encode lifestyle habits (smoking, drinking).
- Label encode ordered health levels.

6.9.2.8 Normalize or Scale Features

- Apply Z-score normalization to clinical measurements.
- Min-max scale features for use in neural networks.

6.9.3 Financial Market

Financial markets are dynamic systems where assets such as stocks, bonds, and derivatives are traded. Analyzing financial market data requires meticulous data preparation to extract meaningful insights. This involves cleaning, transforming, and engineering features from raw data to facilitate accurate analysis and modeling.

Copy CSV PDF Print Search:

	Stock_ID	Date	Stock_Price	Volume_Traded	Market_Cap	PE_Ratio	Dividend_Yield	Return_on_Equity	Sector	Performance
1	QNaySPxCMRBC	2021-07-14	1132.43	934512	1058269424.16	17.16	0.46	5.09	Consumer Goods	Positive
2	7mHFA7pLbHSW	2020-11-16	452.33	500394	226343218.02	17.79	2.85	11.55	Energy	Stable
3	VjBZzknDmBZO	2023-03-22	824.41	134785	111118101.85	17.48	1.7	14.85	Energy	Negative
4	YXkJfMmkKltl	2023-01-02	1163.22	587495	683385933.9	23.31	2.75	15.47	Healthcare	Stable
5	4dxsSjaNTtCL	2023-06-05	990.52	438658	434499522.16	9.72	6.94	18.22	Technology	Positive
6	JbvRy59cd3ls	2023-03-15	385.52	397203	153129700.56	22.54	8.48	12.8	Consumer Goods	Stable
7	XgdclQcFxt8V	2021-09-25	1490.26	794008	1183278362.08	13.33	5.08	8.84	Finance	Stable
8	LxcceyWZdAQu	2020-02-18	100.57	966011	97151726.27	14.68	9.54	13.51	Consumer Goods	Negative
9	ZuTHoWlAUQu2	2023-02-25	389.2	133698	52035261.6	15.41	0.78	15.3	Energy	Negative
10	Zqipa4GQlX1l	2023-08-19	987.64	822394	812229210.16	12.76	5.23	17.66	Technology	Stable
11	ASjStb49sL5G	2020-01-24	124.21	69501	8632719.21	2.23	7.25	14.43	Finance	Positive
12	HM5EQaW7edKj	2020-12-21	137.43	265296	36459629.28	13.44	1	16.25	Energy	Stable
13	U2cvApEvua0z	2024-06-12	950.98	821913	781622824.74	15.21	7.84	13.32	Technology	Stable
14	pnrauUL8niaW	2020-06-13	898.76	34091	30639627.16	6.31	4.5	8.17	Finance	Stable
15	GwijdDLBKsOf	2021-09-13	444.19	72229	32083399.51	17.75	4.46	9.42	Technology	Negative
16	gJTG52cfMdmw	2024-04-04	1387.34	300106	416349058.04	12.01	7.3	17.66	Energy	Stable
17	7x17JOGBkY1u	2021-02-13	1041.1	634911	661005842.1	22.24	4.93	14.11	Healthcare	Positive

Showing 1 to 500 of 500 entries

Before conducting financial modeling or applying machine learning to stock prices or indices, it's critical to preprocess and transform raw financial data into meaningful, structured features. Below are the steps tailored for Financial Market Data Transformation.

6.9.3.1 Import and Inspect the Dataset

- Load data from CSV, APIs (e.g., Yahoo Finance), or financial terminals (e.g., Bloomberg, Refinitiv).
- Ensure **date columns** are in proper `Date` format.
- Check for missing values, incorrect symbols, or duplicate rows.

6.9.3.2 Clean the Data

- Remove non-trading days or fill missing prices using forward-fill.

- Ensure consistency of numeric types (`Open`, `High`, `Low`, `Close`, `Volume`).
- Filter for relevant date ranges or securities if needed.

6.9.3.3 Feature Engineering

- **Daily Return:**

$$\text{Return}_t = \frac{P_t - P_{t-1}}{P_{t-1}}$$

- **Log Return:**

$$\text{LogReturn}_t = \log\left(\frac{P_t}{P_{t-1}}\right)$$

- Add **lag features** for past prices and returns.

6.9.3.4 Temporal and Rolling Features

- Rolling mean (moving average) and volatility (rolling standard deviation):

$$\text{RollingMean}_t = \frac{1}{n}\sum_{i=0}^{n-1} P_{t-i}$$

$$\text{Volatility}_t = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1}(R_{t-i} - \bar{R})^2}$$

- Common rolling windows: 5, 20, 50, or 200 days.

6.9.3.5 Technical Indicators

- **RSI (Relative Strength Index)** – momentum indicator.
- **MACD (Moving Average Convergence Divergence)** – trend-following indicator.
- **Bollinger Bands** – measure price deviation from a moving average.

6.9.3.6 Categorization and Binning

- Classify returns into categories (e.g., “gain”, “loss”, “neutral”).
- Bin volatility into “low”, “medium”, “high” risk.
- Create flags for price crossing key moving averages.

6.9.3.7 Detect and Handle Outliers

Use statistical or financial logic:

- **Z-Score Method:**

$$Z = \frac{x - \mu}{\sigma}$$

- **IQR Method:**

$$IQR = Q3 - Q1$$

$$\text{Bounds} = [Q1 - 1.5 \times IQR,\ Q3 + 1.5 \times IQR]$$

- Consider keeping extreme values if they represent market crashes or bubbles.

6.9.3.8 Encode Categorical Variables

Convert sector, ticker, or market to one-hot or label encoded values.

6.9.3.9 Normalize or Scale Features

- **Z-score normalization** for returns, volume, volatility.
- **Min-max scaling** for deep learning or neural models.

Part III

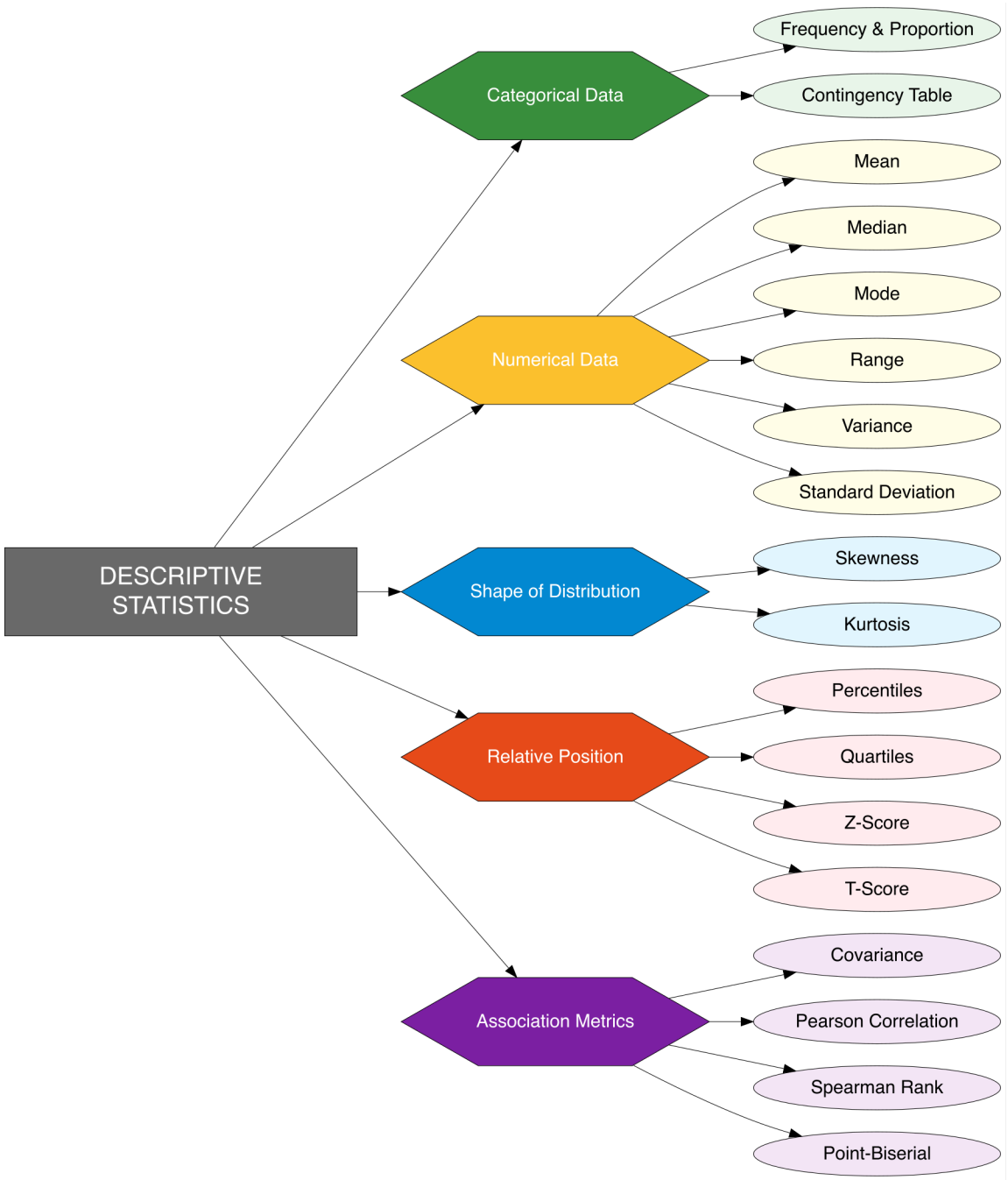
EDA

Chapter 7

Descriptive Statistics

Descriptive Statistics is an essential part of data analysis that focuses on summarizing and presenting data in a way that is easy to understand. The primary goal is to provide an overview of the available data using various techniques that describe patterns, distributions, and relationships between variables.

In the following mind map presents a comprehensive visual overview of Descriptive Statistics, covering key aspects of categorical data, numerical data, distribution shape, relative position, and association metrics between variables.



In data analysis, there is often a distinction between categorical and numerical data. Descriptive statistics for categorical data involves calculating frequency and proportion, using contingency tables, and frequency distributions. For numerical data, it focuses on measures like mean, median, mode, range, variance, and standard deviation.

It is crucial to understand the shape of the distribution of data through metrics such as skewness and kurtosis. Additionally, relative position measures like percentiles, quar-

tiles, and z-scores help in understanding how data points are positioned relative to others in the dataset.

Let say you have dataset like this:

Copy CSV PDF Print Search:

Transformed Business Dataset

	Transaction_ID	Transaction_Date	Customer_ID	Product_Category	Product_ID	Quantity	Unit_Price	Discount	Region	Sales_Channel	Delivery_Time
1	7zmPHxF7XfN9	2021-07-14	BAl3Y7yxev	Clothing	P0370	2	15.18	0	North	Online	
2	y4bCY9pKTBWU	2020-11-16	TYY0h5C190	Electronics	P0185	5	10.22	0.15	West	Offline	
3	8k0B7XX19Ykf	2023-03-22	nUX640AaXg	Home	P0443	3	17.74	0.05	West	Online	
4	l8ahQz5YNOKz	2023-01-02	sBZyUSJLEP	Home	P0035	6	28.3	0.22	North	Offline	
5	kmufgw8wx5qk	2023-06-05	GMfVH2ZWNX	Groceries	P0375	3	11.91	0.13	North	Offline	
6	aI0KADT0mn7C	2023-03-15	YxqAmfTU9M	Clothing	P0447	1	5.43	0.07	North	Offline	
7	zu0iP1OBFuNl	2021-09-25	ifYw95qmoL	Groceries	P0345	2	13.37	0.3	West	Online	
8	l0Bz1iXOUzUy	2020-02-18	F5zL0GjDJh	Clothing	P0193	5	6.56	0.23	South	Online	
9	kgU3mL1c8BdA	2023-02-25	Zm9ROmGygf	Clothing	P0114	4	18.23	0.27	South	Offline	
10	MDp09wloF78Q	2023-08-19	fgldeSSYSL	Home	P0095	1	11.94	0.09	South	Online	
11	M8Wk6MDt6qq1	2020-01-24	y5n7T2vCfd	Electronics	P0309	2	21.06	0.2	South	Offline	
12	lH96iSOc84fn	2020-12-21	SS9WdTh6Gp	Books	P0402	4	9.38	0.01	North	Online	
13	NnorQ5sHloaf	2024-06-12	9l5PBRrmgl	Clothing	P0135	3	15.61	0.17	South	Online	
14	dEbUZoU5Sbu2	2020-06-13	A03t0cAGgC	Clothing	P0312	1	9.17	0.15	South	Offline	
15	o5Gqu5Y2GBZ4	2021-09-13	7hcyxq0KSE	Groceries	P0498	1	16.31	0.24	West	Online	
16	RYXFAjCauCpJ	2024-04-04	A9pnVHddZb	Electronics	P0098	2	18.49	0.09	East	Offline	

Showing 1 to 500 of 500 entries

7.1 Categorical Data

7.1.1 Frequency and Proportion

- **Frequency**: The number of times a particular category appears in a categorical variable.
- **Proportion**: The frequency of a category divided by the total number of observations.

Given a sample of values from the `Product_Category` variable:

```
# Identify categorical variables (character or factor)
categorical_vars <- sapply(data_bisnis,
                           function(x) is.character(x) || is.factor(x))
names(data_bisnis)[categorical_vars]
```

```
[1] "Transaction_ID"   "Customer_ID"      "Product_Category" "Product_ID"
[5] "Region"           "Sales_Channel"
```

Frequency & Proportion Table

```
library(dplyr)
library(knitr)

data_bisnis %>%
  count(Product_Category) %>%
  mutate(Proportion = n / sum(n)) %>%
  kable()
```

Product_Category	n	Proportion
Books	95	0.190
Clothing	111	0.222
Electronics	88	0.176
Groceries	96	0.192

Product_Category	n	Proportion
Home	110	0.220

Proportion is calculated as:
`Proportion = Frequency / Total Observations`
Example: Clothing → 3 / 10 = 0.30

7.1.2 Contingency Table

A **contingency table** (or cross-tabulation) is a table that displays the frequency distribution of two categorical variables simultaneously.
It is useful to identify **relationships or dependencies** between the variables.

From a sample of 9 rows combining `Region` and `Sales_Channel`:

Base R

```
# Create the contingency table
contingency<- table(data_bisnis$Region, data_bisnis$Sales_Channel)

# Add the "Total" column and "Total" row
contingency_totals <- addmargins(contingency)

# Print the table
kable(contingency_totals)
```

	Offline	Online	Sum
East	63	63	126
North	76	65	141
South	64	49	113
West	59	61	120
Sum	262	238	500

Tidyverse R

```
library(tidyr)
library(dplyr)

# Create the contingency table with counts
Contingency <- data_bisnis %>%
  count(Region, Sales_Channel) %>%
  pivot_wider(
    names_from = Sales_Channel,
    values_from = n,
    values_fill = list(n = 0)
  ) %>%
  # Add Total column and Grand Total row using summarise
```

```
  mutate(Total = `Online` + `Offline`) %>%
  bind_rows(
    summarise(., Region = "Total",
              Online = sum(.$Online),
              Offline = sum(.$Offline),
              Total = sum(.$Total))
  ) %>%
  add_row(Region = "Grand Total",
          Online = sum(.$Online),
          Offline = sum(.$Offline),
          Total = sum(.$Total))

# Print the contingency table with totals
kable(Contingency)
```

Region	Offline	Online	Total
East	63	63	126
North	76	65	141
South	64	49	113
West	59	61	120
Total	262	238	500
Grand Total	524	476	1000

Interpretation:

- 2 transactions from **North** used the **Online** channel.
- 2 transactions from **North** used the **Offline** channel.
- **Total:** 4 transactions from **North**.

7.2 Numerical Data

In statistics, numerical data refers to data that consists of numbers, which can be used for various types of calculations and analysis. Numerical data is divided into two types: **discrete** (where the data can only take certain specific values) and **continuous** (where the data can take any value within a given range). Here, we will discuss key concepts related to numerical data, including the mean, median, mode, range, variance, and standard deviation.

7.2.1 Mean

The **mean** is the average of a set of numbers. It is calculated by adding all the numbers in the dataset and dividing by the total number of values.

Formula:

$$\text{Mean} = \frac{\sum X_i}{n}$$

Where:

- $\sum X_i$ is the sum of all the data points.
- n is the number of data points.

Example:

Given the dataset: 5, 7, 9, 10, 15

$$\text{Mean} = \frac{5 + 7 + 9 + 10 + 15}{5} = \frac{46}{5} = 9.2$$

7.2.2 Median

The **median** is the middle value in a dataset when the values are arranged in ascending or descending order. If the dataset has an even number of values, the median is the average of the two middle values.

How to Calculate:

1. Sort the data in ascending order.
2. If the dataset has an odd number of values, the median is the middle value.
3. If the dataset has an even number of values, the median is the average of the two middle values.

Example:

Given the dataset: 5, 7, 9, 10, 15 (sorted)

Since there are 5 values (odd number), the median is the middle value:

$$\text{Median} = 9$$

For an even number of values, e.g., 5, 7, 9, 10, 15, 20: The middle values are 9 and 10, so:

$$\text{Median} = \frac{9 + 10}{2} = 9.5$$

7.2.3 Mode

The **mode** is the value that occurs most frequently in a dataset. A dataset may have one mode (unimodal), more than one mode (multimodal), or no mode at all.

How to Calculate:

1. Count the frequency of each data point.
2. The value with the highest frequency is the mode.

Example:

Given the dataset: 5, 7, 9, 9, 10, 15, 15, 15

The mode is 15, as it appears 3 times, more than any other value.

7.2.4 Range

The **range** is the difference between the largest and smallest values in a dataset. It provides a measure of how spread out the values are.

How to Calculate:

$$\text{Range} = \text{Maximum Value} - \text{Minimum Value}$$

Example:

Given the dataset: 5, 7, 9, 10, 15

$$\text{Range} = 15 - 5 = 10$$

7.2.5 Variance

Variance is a measure of how much the values in a dataset differ from the mean. It calculates the average squared deviation of each data point from the mean. Variance is important for understanding the spread of data.

Formula:

$$\text{Variance} = \frac{\sum(X_i - \mu)^2}{n}$$

Where:

- X_i represents each data point.
- μ is the mean of the dataset.
- n is the number of data points.

7.2.6 Standard Deviation

Standard deviation is the square root of the variance. It measures the amount of variation or dispersion in a dataset.

Formula:

$$\text{Standard Deviation} = \sqrt{\text{Variance}} = \sqrt{\frac{\sum(X_i - \mu)^2}{n}}$$

Interpretation of Standard Deviation Values:

- **Low standard deviation**: Data points are close to the mean, indicating less variability.
- **High standard deviation**: Data points are spread out over a wider range, indicating more variability.

Example:

Given the dataset: 5, 7, 9, 10, 15

Step 1: Calculate the Mean

$$\text{Mean} = \frac{5 + 7 + 9 + 10 + 15}{5} = \frac{46}{5} = 9.2$$

Step 2: Variance Formula

The formula for population variance is:

$$\text{Variance} = \frac{(X_1 - \mu)^2 + (X_2 - \mu)^2 + \cdots + (X_n - \mu)^2}{n}$$

For our dataset:

$$\text{Variance} = \frac{(5 - 9.2)^2 + (7 - 9.2)^2 + (9 - 9.2)^2 + (10 - 9.2)^2 + (15 - 9.2)^2}{5}$$

Step 3: Compute Each Term

- $(5 - 9.2)^2 = (-4.2)^2 = 17.64$
- $(7 - 9.2)^2 = (-2.2)^2 = 4.84$
- $(9 - 9.2)^2 = (-0.2)^2 = 0.04$
- $(10 - 9.2)^2 = (0.8)^2 = 0.64$
- $(15 - 9.2)^2 = (5.8)^2 = 33.64$

Step 4: Variance

$$\text{Variance} = \frac{17.64 + 4.84 + 0.04 + 0.64 + 33.64}{5} = \frac{56.8}{5} = 11.36$$

Step 5: Standard Deviation

$$\text{Standard Deviation} = \sqrt{11.36} \approx 3.37$$

The **variance** is **11.36**, indicating the average squared deviation from the mean. **Standard deviation** is more interpretable than variance because it is in the same unit as the data, while variance is in squared units. It helps understand the extent to which individual data points deviate from the mean.

7.2.7 Summary

We will compute the following statistical measures:

- **Mean** is sensitive to outliers but easy to calculate.
- **Median** is robust to skewed distributions and outliers.
- **Mode** represents the most frequent value and may not always exist or be unique.
- **Range** is the simplest measure of spread but does not provide information about the distribution.
- **Variance** measures the spread of data, while **standard deviation** provides a more interpretable measure of data variability.

Single Numerical Variable

In data analysis, a **single numerical variable** (also known as univariate numerical data) provides essential information about the central tendency and dispersion of values. This section covers the analysis of one such variable from our dataset: `Total_Price`.

```
# Load library
library(dplyr)
library(knitr)

# Data numerik: Total_Price dari data_bisnis
data_numeric <- data_bisnis$Total_Price

# Hitung statistik
mean_val <- mean(data_numeric)
median_val <- median(data_numeric)
mode_val <- as.numeric(names(sort(table(data_numeric), decreasing = TRUE)[1]))
range_val <- max(data_numeric) - min(data_numeric)
variance_val <- var(data_numeric)*(length(data_numeric)-1)/length(data_numeric)
sd_val <- sqrt(variance_val)

# Buat tabel
summary_stats <- data.frame(
  Measure = c("Mean", "Median", "Mode", "Range",
              "Variance", "Standard Deviation"),
  Value = round(c(mean_val, median_val, mode_val,
                  range_val, variance_val, sd_val), 2)
)

# Tampilkan tabel
kable(summary_stats, caption="Summary Statistics for Total_Price (Population)")
```

Table 7.4: Summary Statistics for Total_Price (Population)

Measure	Value
Mean	36.66
Median	30.55
Mode	0.00
Range	177.06
Variance	764.34
Standard Deviation	27.65

7.2.7.1 Multi Numerical Variables

Each of these measures plays an important role in summarizing and analyzing numerical data, and they should be chosen based on the nature of the data and the type of analysis being conducted.

```
library(dplyr)
library(knitr)
```

```r
# Compute summary statistics in long format
summary_table_df <- data_bisnis %>%
  summarise(across(where(is.numeric),
                   list(mean = mean,
                        median = median,
                        min = min,
                        max = max,
                        variance = var),
                   .names = "{.fn}_{.col}")) %>%
  pivot_longer(cols = everything(),
               names_to = c("stat", "variable"),
               names_sep = "_",
               values_to = "value") %>%
  pivot_wider(names_from = variable, values_from = value)

# Display the table
kable(summary_table_df, caption = "Summary Statistics of Numeric Variables")
```

Table 7.5: Summary Statistics of Numeric Variables

stat	Quantity	Unit	Discount	Delivery	Total
mean	3.140000	13.68072	0.1428800	5.330000	36.66416
median	3.000000	13.30000	0.1500000	5.000000	30.54500
min	0.000000	-1.27000	0.0000000	1.000000	-2.37000
max	9.000000	30.83000	0.3000000	10.000000	174.69000
variance	3.078557	32.65094	0.0074578	8.870842	765.87086

7.3 Shape of Distribution

Skewness and kurtosis are statistical measures that help us understand the **shape of a distribution**. Both are essential for **risk analysis**, **outlier detection**, and **deeper interpretation** of business data distributions.

Example Data (10 Prices from `data_bisnis$price`)

```r
price <- c(80, 90, 100, 110, 120, 130, 140, 150, 160, 200)
```

7.3.1 Mean and Standard Deviation

- **Mean** $\bar{x}$:

$$\bar{x} = \frac{80 + 90 + \cdots + 200}{10} = 128$$

- **Standard Deviation** s:

$$s = \sqrt{\frac{1}{n-1}\sum(x_i - \bar{x})^2} = \sqrt{\frac{11756}{9}} \approx 36.1$$

7.3.2 Skewness

$$\text{Skewness} = \frac{1}{n}\sum\left(\frac{x_i - \bar{x}}{s}\right)^3$$

Price	Standardized (z)	z^3
80	-1.33	-2.35
90	-1.05	-1.15
100	-0.78	-0.47
110	-0.50	-0.13
120	-0.22	-0.01
130	0.06	0.00
140	0.33	0.04
150	0.61	0.23
160	0.89	0.71
200	1.99	7.88
Total	—	**4.75**

$$\text{Skewness} = \frac{4.75}{10} = 0.475$$

Interpretation: Slight **positive skew** → right tail longer.

7.3.3 Kurtosis

$$\text{Kurtosis} = \frac{1}{n}\sum\left(\frac{x_i - \bar{x}}{s}\right)^4$$

Price	Standardized (z)	z^4
80	-1.33	3.13
90	-1.05	1.22
100	-0.78	0.37
110	-0.50	0.06
120	-0.22	0.002
130	0.06	0.0001
140	0.33	0.012
150	0.61	0.14
160	0.89	0.63
200	1.99	15.80
Total	—	**21.37**

$$\text{Kurtosis} = \frac{21.37}{10} = 2.14$$

Interpretation: **Platykurtic** (flatter than normal distribution, fewer extreme values).

Table 7.8: Skewness and Kurtosis in Business Data (Price)

Concept	Definition	Usage…Application	Example
Skewness	Measure of asymmetry in the distribution. Formula: Skewness = E[(X -)^3] / ^3	Understand data symmetry and detect bias in distribution.	Skewness of price data = NaN
Positive Skew	Tail on the right is longer; most data on the left. Mean > Median.	Common in income, sales data. Indicates presence of high values.	Example: Sales with a few very high prices.
Negative Skew	Tail on the left is longer; most data on the right. Mean < Median.	May indicate minimum/low value dominance. Useful for returns loss modeling.	Example: Return loss data with rare but large negative values.
Zero Skew	Symmetric distribution. Mean Median.	Ideal for standard modeling assumptions (e.g., regression).	Example: Normally distributed cost data.
Kurtosis	Measure of the 'tailedness' of the distribution. Formula: Kurtosis = E[(X -)^4] / ^4	Detect outliers and model tail risk in financial/business data.	Kurtosis of price data = NaN
Leptokurtic	Heavy tails and sharp peak. High outlier risk.	High risk of extreme values. Needs robust modeling.	Example: Stock prices with frequent extreme changes.
Platykurtic	Light tails and flat peak. Less extreme values.	Low risk. Less concern about outliers.	Example: Customer purchase frequency data (low variation).
Mesokurtic	Normal tails and peak. Follows normal distribution (kurtosis 3).	Expected in many natural distributions (e.g., height, IQ).	Example: Simulated price data with normal distribution.

7.4 Relative Position

Relative position refers to how a specific data point compares to other data points in a dataset. It is often used to understand the distribution and spread of the data. By calculating relative positions, we can gain insights into how an individual observation fits within the entire data set. This is particularly useful when measuring performance, identifying outliers, and comparing data points across different distributions. Key metrics like percentiles, quartiles, Z-scores, and T-scores are commonly used to analyze and interpret relative position.

Below is an overview of these concepts:

Table 7.9: Relative Position in Business Data (Price)

Concept	Definition	Usage…Application	Example
Percentiles	Indicates the value below which a given percentage of observations fall. Percentile = (Rank / Total Observations) × 100	Used to evaluate performance levels (e.g., top 10%), set cut-off thresholds, and benchmarking.	In a test with 100 products, the 90th percentile corresponds to the score above which only 10% of the students scored higher.
Quartiles	Q1 = 25th percentile, Q2 = 50th (Median), Q3 = 75th percentile.	Used in boxplots, IQR (Q3 - Q1) calculation, and outlier detection.	In a dataset of the prices: Q1 = 50, Q2 (median) = 70, Q3 = 85, IQR = 35 (Q3-Q1).
Z-Score	Standardized score: Z = (X -) /	Measures how many standard deviations a value is from the mean. Enables comparison across distributions.	For price = 100 , Z-score = NA (Standard deviations from the mean).
T-Score	Similar to Z-score but adjusted for small samples: T = (X - $\bar{x}$) / (s / $\sqrt{n}$)	Used in hypothesis testing and small-sample inference. Preferred when population is unknown.	For price = 100 , T-score = NA (Adjusted for small sample size).

7.5 Association Metrics

In data analysis, understanding the relationships between variables is essential for drawing insights from data. Association metrics, such as covariance and correlation coefficients, are used to measure the strength and direction of relationships between variables. Below is a table summarizing the key association metrics:

Table 7.10: Association Metrics

Concept	Definition	Usage...Application	Example
Covariance	Measures the direction of the linear relationship between two variables. Positive covariance indicates both increase together, negative indicates inverse.	Used to determine the direction of the relationship between two variables but limited in scale comparison. **Formula:** Cov(X, Y) = (1 / (n - 1)) * Σ[(X - X)(Y - Ŷ)]	**cov(data_bisnis$price, data_bisnis$quantity)** to determine how price and quantity vary together.

Concept	Definition	Usage...Application	Example
Pearson Correlation Coefficient	Measures the strength and direction of the linear relationship between two variables.	Used when data is continuous and normally distributed. **Formula:** r = Cov(X, Y) / (*)	**cor(data_bisnis$price, data_bisnis$quantity, method = 'pearson')** to find the linear relationship between price and quantity.
Spearman's Rank Correlation	Non-parametric measure of the strength and direction of a monotonic relationship between two variables.	Used for ordinal data or when variables do not have a linear relationship. **Formula:** = 1 - (6 * Σ(d ²)) / (n(n² - 1))	**cor(data_bisnis$price, data_bisnis$quantity, method = 'spearman')** to find the monotonic relationship between price and quantity.
Point-Biserial Correlation	Measures the relationship between one continuous variable and one binary variable.	Commonly used in psychological and social sciences. **Formula:** r = (M - M) / * √(n * n / (n(n - 1)))	**cor(data_bisnis$price, is_discount, method = 'pearson')** to find the correlation between price and discount status.

Chapter 8

Descriptive Visualizations

Descriptive visualizations are charts and plots employed in Exploratory Data Analysis to succinctly summarize and uncover essential patterns, relationships, and anomalies within data. They transform raw data into clear, interpretable visuals that enhance comprehension and support informed decision-making. These tools are crucial for examining distributions, correlations, comparisons, trends, and outliers, providing a fundamental basis for more advanced modeling and analysis.

Descriptive visualizations differ according to data type—bar and pie charts are ideal for categorical data, whereas histograms and boxplots are more appropriate for numerical data. Selecting the appropriate visualization depends on the analytical objective: grouped bar charts facilitate comparisons across groups, line charts effectively reveal trends over time, scatter plots illustrate relationships between continuous variables, and boxplots highlight skewness, variability, and outliers. These visual tools are indispensable for distilling complex datasets into comprehensible insights, empowering analysts

and decision-makers to better understand customer behavior, monitor sales performance, evaluate operational efficiency, and identify strategic opportunities or risks. The following section will showcase these visualization techniques applied to a business dataset, demonstrating how they can uncover hidden insights, support hypotheses, and lay the groundwork for advanced analytical modeling.

Copy CSV PDF Print Search:

Transformed Business Dataset

	Transaction_ID	Transaction_Date	Customer_ID	Product_Category	Product_ID	Quantity	Unit_Price	Discount	Region	Sales_Channel	Delivery_Time
1	7zmPHxF7XfN9	2021-07-14	BAl3Y7yxev	Clothing	P0370	2	15.18	0	North	Online	
2	y4bCY9pKTBWU	2020-11-16	TYY0h5C190	Electronics	P0185	5	10.22	0.15	West	Offline	
3	8k0B7XX19Ykf	2023-03-22	nUX640AaXg	Home	P0443	3	17.74	0.05	West	Online	
4	l8ahQz5YNOKz	2023-01-02	sBZyUSJLEP	Home	P0035	6	28.3	0.22	North	Offline	
5	kmufgw8wx5qk	2023-06-05	GMfVH2ZWNX	Groceries	P0375	3	11.91	0.13	North	Offline	
6	aI0KADT0mn7C	2023-03-15	YxqAmfTU9M	Clothing	P0447	1	5.43	0.07	North	Offline	
7	zu0iP1OBFuNl	2021-09-25	ifYw95qmoL	Groceries	P0345	2	13.37	0.3	West	Online	
8	l0Bz1iXOUzUy	2020-02-18	F5zL0GjDJh	Clothing	P0193	5	6.56	0.23	South	Online	
9	kgU3mL1c8BdA	2023-02-25	Zm9ROmGygf	Clothing	P0114	4	18.23	0.27	South	Offline	
10	MDp09wloF78Q	2023-08-19	fgldeSSYSL	Home	P0095	1	11.94	0.09	South	Online	
11	M8Wk6MDt6qq1	2020-01-24	y5n7T2vCfd	Electronics	P0309	2	21.06	0.2	South	Offline	
12	lH96iSOc84fn	2020-12-21	SS9WdTh6Gp	Books	P0402	4	9.38	0.01	North	Online	
13	NnorQ5sHloaf	2024-06-12	9l5PBRrmgl	Clothing	P0135	3	15.61	0.17	South	Online	
14	dEbUZoU5Sbu2	2020-06-13	A03t0cAGgC	Clothing	P0312	1	9.17	0.15	South	Offline	
15	o5Gqu5Y2GBZ4	2021-09-13	7hcyxq0KSE	Groceries	P0498	1	16.31	0.24	West	Online	
16	RYXFAjCauCpJ	2024-04-04	A9pnVHddZb	Electronics	P0098	2	18.49	0.09	East	Offline	

Showing 1 to 500 of 500 entries

By visualizing this data, we seek to uncover hidden patterns, identify anomalies, and reveal insights that raw tables alone cannot convey. These insights will ultimately enable more informed business decisions and strategic planning.

In the forthcoming sections, we will apply a variety of visual techniques to transform this structured data into clear, interpretable graphics—laying the foundation for compelling and effective data storytelling.

8.1 Categorical Data

Visualizations for categorical data help display the distribution and frequency of categories in a dataset. Here are the most common and effective types of visualizations used for categorical variables:

8.1.1 Bar Chart

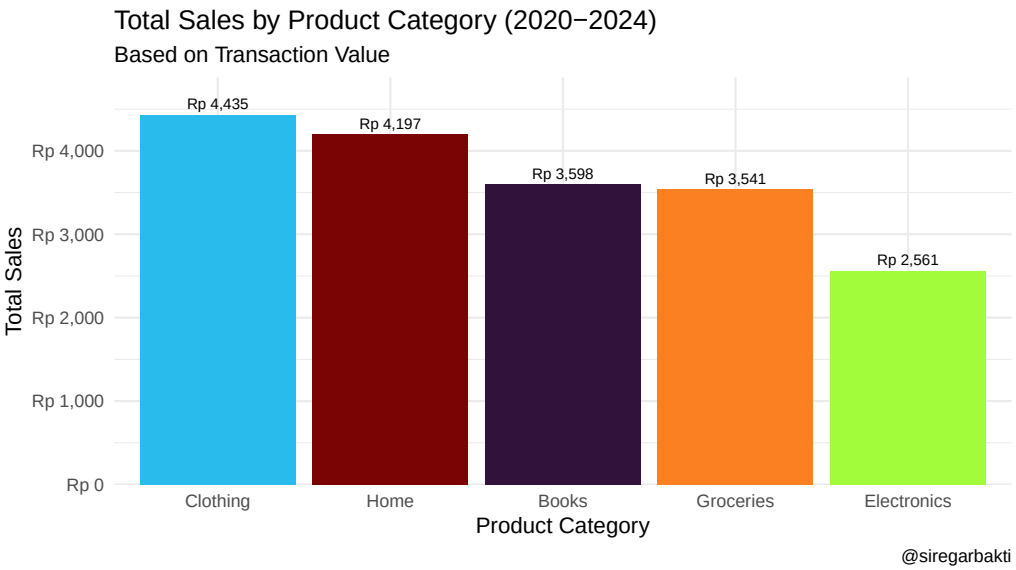
A Bar Chart is a graphical representation of categorical data in which each category is represented by a bar, with the height (or length) of the bar corresponding to the frequency or count of data in that category.

Key Characteristics of a Bar Chart:

- Represents categorical data using rectangular bars, where the height or length of each bar reflects the value or frequency of the category.
- One axis (usually the x-axis) shows the categories, while the other axis (usually the y-axis) displays the numerical values.
- Bars are separated from each other, emphasizing that the data is discrete, unlike histograms.
- Can be displayed as vertical (column chart) or horizontal bars, depending on the layout or clarity needs.
- Useful for comparing values across categories clearly and intuitively.
- Can include additional variations like grouped or stacked bar charts for comparing subcategories.

- Easily labeled and color-coded for enhanced readability and interpretation.

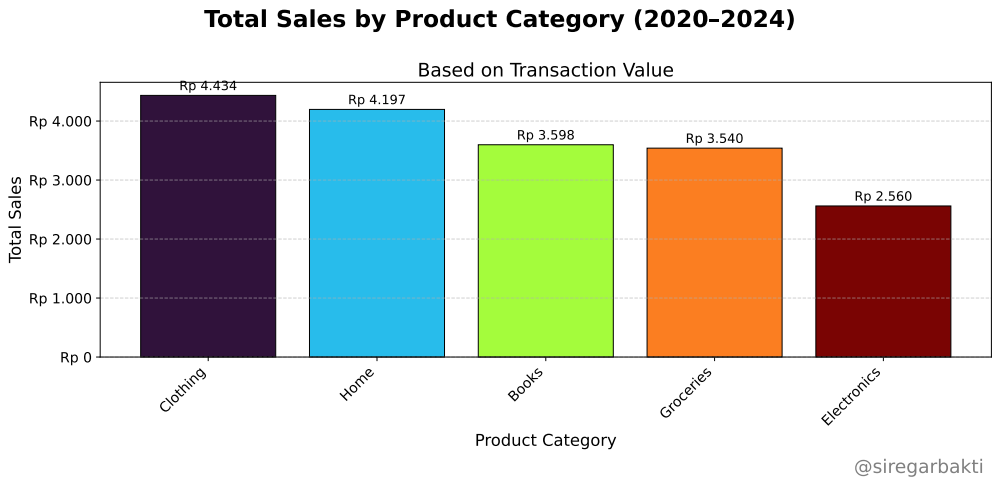
```r
# Load required libraries
library(dplyr)        # For data manipulation
library(ggplot2)      # For creating the bar chart
library(viridis)      # For color palette
library(scales)       # For formatting currency labels

# Step 1: Prepare the data
data_bisnis <- read.csv("data/bab8/data_bisnis.csv")
sales_summary <- data_bisnis %>%
  group_by(Product_Category) %>%
  summarise(Total_Sales = sum(Total_Price, na.rm = TRUE)) %>%
  arrange(desc(Total_Sales))

# Step 2: Generate a color palette
custom_colors <- viridis::turbo(n = nrow(sales_summary))

# Step 3: Create bar chart with value labels
ggplot(sales_summary, aes(x = reorder(Product_Category, -Total_Sales),
                          y = Total_Sales,
                          fill = Product_Category)) +
  geom_col(show.legend = FALSE) +
  geom_text(aes(label = scales::label_comma(prefix = "Rp ")(Total_Sales)),
            vjust = -0.5, size = 6) +
  scale_fill_manual(values = custom_colors) +
  scale_y_continuous(labels = scales::label_comma(prefix = "Rp "),
                     expand = expansion(mult = c(0, 0.1))) +
  labs(
    title = "Total Sales by Product Category (2020-2024)",
    subtitle = "Based on Transaction Value",
    x = "Product Category",
    y = "Total Sales",
    caption = "@siregarbakti") +
    theme_minimal(base_size = 25)
```



Python Code (Barchart)

```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter
from matplotlib import cm
import numpy as np

data_bisnis = pd.read_csv("data/bab8/data_bisnis.csv")

# Step 1: Prepare the data
sales_summary = (
    data_bisnis
    .groupby('Product_Category', as_index=False)
    .agg(Total_Sales=('Total_Price', 'sum'))
    .sort_values('Total_Sales', ascending=False)
)

# Step 2: Generate color palette
num_categories = sales_summary.shape[0]
colors = cm.turbo(np.linspace(0, 1, num_categories))

# Step 3: Create figure and axis
fig, ax = plt.subplots(figsize=(12, 6))

# Step 4: Plot bar chart
bars = ax.bar(
    sales_summary['Product_Category'],
    sales_summary['Total_Sales'],
    color=colors,
    edgecolor='black',
```

```
    linewidth=0.8
)

# Step 5: Format y-axis as currency
formatter = FuncFormatter(lambda x, _: f'Rp {int(x):,}'.replace(',', '.'))
ax.yaxis.set_major_formatter(formatter)
ax.grid(axis='y', linestyle='--', alpha=0.6)

# Step 6: Axis labels and ticks
ax.set_xlabel('Product Category', fontsize=14)
ax.set_ylabel('Total Sales', fontsize=14)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right', fontsize=12);
ax.tick_params(axis='y', labelsize=12)


# Step 7: Add value labels on bars
for bar in bars:
    height = bar.get_height()
    ax.text(
        bar.get_x() + bar.get_width() / 2,
        height + max(sales_summary['Total_Sales']) * 0.01,
        f'Rp {int(height):,}'.replace(',', '.'),
        ha='center',
        va='bottom',
        fontsize=11
    )

# Step 8: Titles
fig.suptitle('Total Sales by Product Category (2020-2024)',
             fontsize=20, weight='bold', y=0.93)
ax.set_title('Based on Transaction Value', fontsize=16, pad=5, loc='center')

# Step 9: Credit
fig.text(0.98, 0.01, '@siregarbakti', ha='right', fontsize=16, color='gray')

# Step 10: Layout
plt.tight_layout(rect=[0, 0.03, 1, 0.92])

# Show plot once
plt.show();
```



8.1.2 Pie Chart

A Pie Chart is a circular statistical graphic that displays categorical data as slices of a circle, where each slice represents a proportion or percentage of the whole.

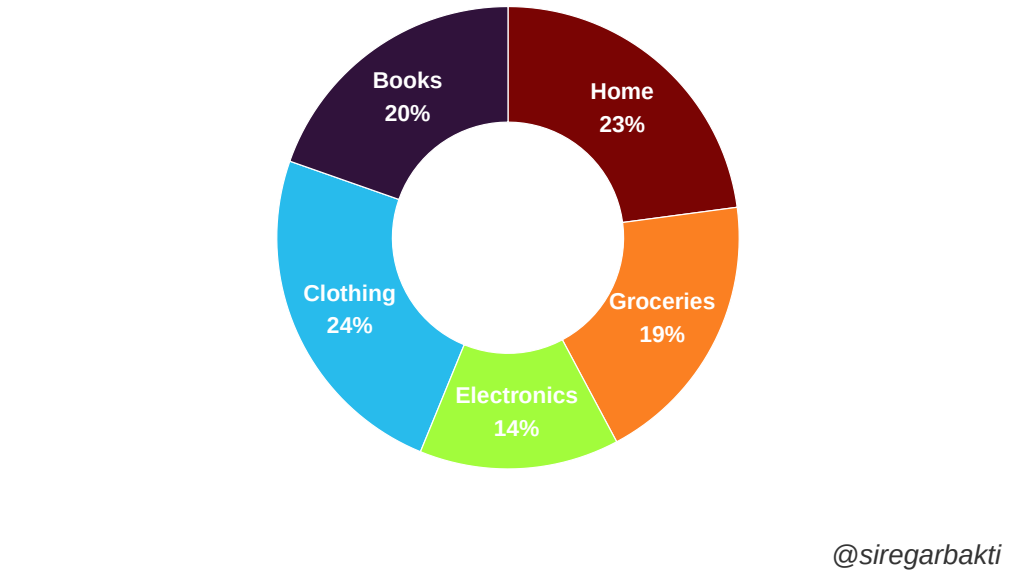
Key Characteristics of a Pie Chart:

- Represents parts of a whole — the full circle equals 100%.
- Each slice corresponds to a category, sized by its proportion or percentage of the total.
- Labels or legends are used to indicate the name and value (or percentage) of each slice.
- Best suited for categorical data with limited categories (ideally fewer than 6) to maintain clarity.
- Colors are commonly used to visually differentiate between categories.
- Slices are not ordered—they follow the input sequence unless sorted manually.
- Commonly used to highlight the dominance or imbalance of categories within a dataset.
- Often used in business reports or dashboards for quick, high-level summaries.

R Code (Pie-Chart)

```r
# Load necessary libraries
library(dplyr)      # For data manipulation
library(ggplot2)    # For data visualization
library(viridis)    # For color palettes
library(scales)     # For formatting percentages

# Step 1: Summarize total sales by product category
data_bisnis <- read.csv("data/bab8/data_bisnis.csv")
sales_summary <- data_bisnis %>%
  group_by(Product_Category) %>%
  summarise(Total_Sales = sum(Total_Price, na.rm = TRUE)) %>%
```

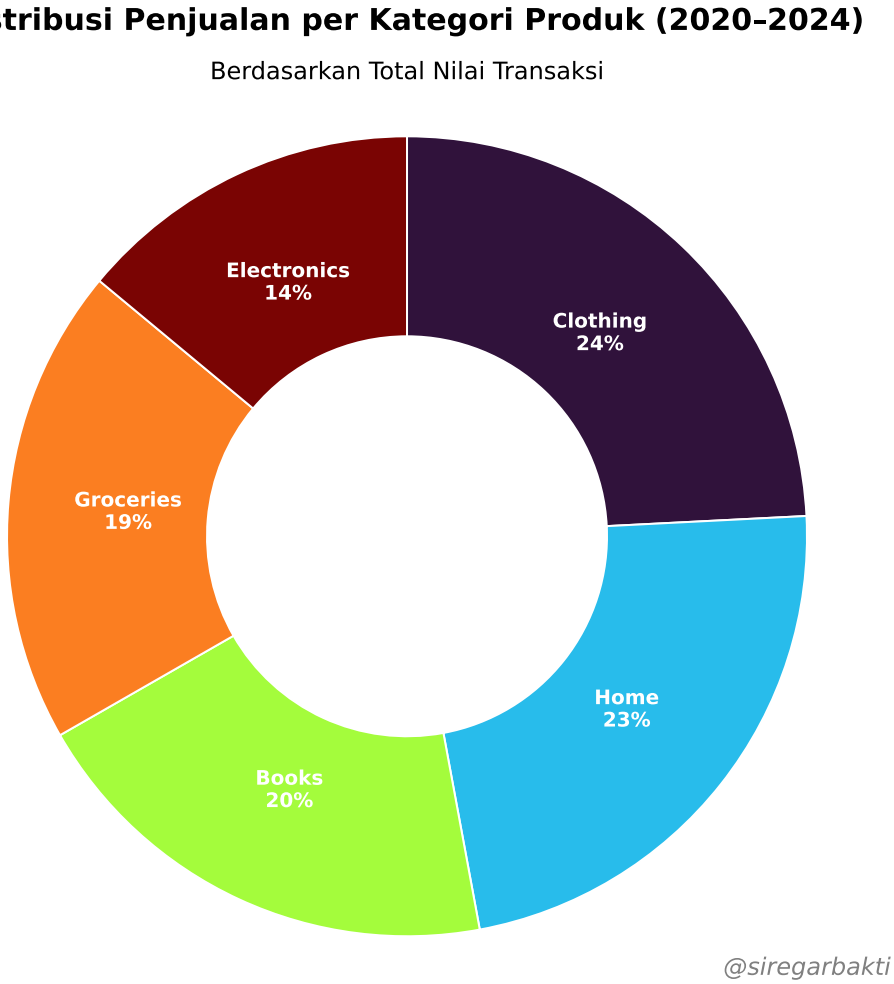
```
  arrange(desc(Total_Sales)) %>%
  mutate(
    Percentage = Total_Sales / sum(Total_Sales),# Calculate share
    Label = paste0(Product_Category, "\n",      # Create label with line break
                   scales::percent(Percentage, accuracy = 1)))

# Step 2: Create custom color palette
custom_colors <- viridis::turbo(n = nrow(sales_summary))

# Step 3: Plot donut chart
ggplot(sales_summary, aes(x =2, y = Percentage, fill = Product_Category)) +
  geom_col(width = 1, color = "white", show.legend = FALSE) + # donut slices
  coord_polar(theta = "y") +                    # Convert to circular layout
  geom_text(aes(label = Label),                 # Add labels inside slices
            position = position_stack(vjust = 0.5),
            size = 7, color = "white", fontface = "bold") +
  scale_fill_manual(values = custom_colors) +
  xlim(0.5, 2.5) +                              # Expand size of donut
  labs(
    title = "Sales Distribution by Product Category (2020-2024)",
    subtitle = "Based on Total Transaction Value",
    caption = "@siregarbakti"
  ) +
  theme_void(base_size = 30) +                  # Clean theme
  theme(
    plot.title = element_text(face = "bold", hjust = 0.5), # Centered title
    plot.subtitle = element_text(margin = margin(t = 8, b = 20), hjust = 0.5),
    plot.caption = element_text(margin = margin(t = 15), hjust = 1.5,
                                color = "gray20", face = "italic")
  )
```



Python Code (Pie-Chart)

```
import matplotlib.pyplot as plt
from matplotlib import cm
import numpy as np

data_bisnis = pd.read_csv("data/bab8/data_bisnis.csv")

# Ringkasan Total Sales per Product_Category
sales_summary = (
    data_bisnis
    .groupby('Product_Category', as_index=False)
    .agg(Total_Sales=('Total_Price', 'sum'))
    .sort_values('Total_Sales', ascending=False)
)

# Persentase dan label
sales_summary['Percentage'] = sales_summary['Total_Sales']/sales_summary['Total_Sales'].sum()
sales_summary['Label'] = sales_summary.apply(
    lambda row: f"{row['Product_Category']}\n{row['Percentage']:.0%}", axis=1
)

# Warna turbo
num_categories = sales_summary.shape[0]
colors = cm.get_cmap('turbo')(np.linspace(0, 1, num_categories))
```

```
# Plot donut chart
fig, ax = plt.subplots(figsize=(7, 7))
wedges, _ = ax.pie(
    sales_summary['Percentage'],
    labels=None,
    startangle=90,
    counterclock=False,
    colors=colors,
    wedgeprops=dict(width=0.5, edgecolor='white')
)

# Tambahkan label ke setiap sektor
for i, (wedge, label) in enumerate(zip(wedges, sales_summary['Label'])):
    angle = (wedge.theta2 + wedge.theta1) / 2
    x = np.cos(np.radians(angle)) * 0.7
    y = np.sin(np.radians(angle)) * 0.7
    ax.text(x, y, label, ha='center', va='center', fontsize=10,
            color='white', weight='bold')

# Judul dan estetika
fig.suptitle('Distribusi Penjualan per Kategori Produk (2020-2024)',
            fontsize=15, weight='bold')
ax.set_title('Berdasarkan Total Nilai Transaksi', fontsize=12, pad=10)
fig.text(0.98, 0.02, '@siregarbakti', ha='right',
        fontsize=12, color='gray', style='italic')

# Tampilan
ax.axis('equal')  # Buat pie jadi lingkaran sempurna
plt.tight_layout()
plt.show();
```



8.1.3 Word Cloud

A Word Cloud is a visual representation of textual data where the size of each word indicates its frequency or importance within a dataset. It is especially useful for quickly identifying the most prominent terms in unstructured text.

Key Characteristics of a Word Cloud:

- Displays words in varying sizes, where size reflects frequency or importance in the text data.
- Often used to summarize large volumes of unstructured text at a glance.
- Words are typically shown in a free-form, non-linear layout, often randomized for visual variety.
- Works best for exploratory text analysis, not precise quantitative interpretation.
- Commonly excludes stop words (e.g., “and”, “the”, “of”) to emphasize meaningful terms.
- Can be customized in terms of font, color, shape, orientation, and layout.
- Especially effective in identifying dominant themes or keywords from sources like

customer feedback, social media posts, and survey responses.

R Code (Word Cloud)

```
# ==============================
# 1. Install & Load Required Packages
# ==============================
packages <- c("dplyr", "tm", "wordcloud", "RColorBrewer")
new_packages <- packages[!(packages %in% installed.packages()[, "Package"])]
if(length(new_packages)) install.packages(new_packages)

library(dplyr)
library(tm)
library(wordcloud)
library(RColorBrewer)

# ==============================
# 2. Read and Combine Text Columns
# ==============================
data_bisnis <- read.csv("data/bab8/data_bisnis.csv")

# Combine text columns into one
text_data <- paste(data_bisnis$Product_Category,
                   data_bisnis$Region,
                   data_bisnis$Sales_Channel,
                   sep = " ")

# ==============================
# 3. Clean and Prepare Text
# ==============================
corpus <- VCorpus(VectorSource(text_data))

corpus_clean <- corpus %>%
  tm_map(content_transformer(tolower)) %>%  # convert to lowercase
  tm_map(removePunctuation) %>%             # remove punctuation
  tm_map(removeNumbers) %>%                 # remove numbers
  tm_map(removeWords, stopwords("english")) %>%  # remove English stopwords
  tm_map(stripWhitespace)                   # remove extra whitespace

# Remove empty documents (if any)
non_empty_idx <- sapply(corpus_clean, function(doc) {
  nchar(content(doc)) > 0
})
corpus_clean <- corpus_clean[non_empty_idx]

# ==============================
# 4. Create Term-Document Matrix & Word Frequencies
# ==============================
tdm <- TermDocumentMatrix(corpus_clean)
```

```r
m <- as.matrix(tdm)
word_freqs <- sort(rowSums(m), decreasing = TRUE)
df_words <- data.frame(word = names(word_freqs), freq = word_freqs)

# ==============================
# 5. Generate Word Cloud (Full Screen)
# ==============================
set.seed(123)
wordcloud(words = df_words$word,
          freq = df_words$freq,
          scale = c(8, 8),       # adjust for large size
          min.freq = 1,
          max.words = 300,
          random.order = FALSE,
          rot.per = 0.3,
          colors = brewer.pal(8, "Dark2"))
```

Python Code (Word Cloud)

```
import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# 1. Install and load required packages
# (Make sure nltk stopwords are downloaded)
nltk.download('stopwords')
```

```
# 2. Read and Combine Text Columns
data_bisnis = pd.read_csv("data/bab8/data_bisnis.csv")

# Combine text columns into a single string per row
text_data = data_bisnis['Product_Category'].fillna('') + " " + \
            data_bisnis['Region'].fillna('') + " " + \
            data_bisnis['Sales_Channel'].fillna('')

# 3. Clean and Prepare Text - similar to tm_map pipeline in R
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = text.lower()                          # tolower()
    text = re.sub(r'[^\w\s]', ' ', text)       # removePunctuation()
    text = re.sub(r'\d+', '', text)             # removeNumbers()
    text = re.sub(r'\s+', ' ', text)            # stripWhitespace()
    words = text.strip().split()
    words = [w for w in words if w not in stop_words]  # removeWords(stopwords)
    return " ".join(words)

cleaned_docs = text_data.apply(clean_text)

# Remove empty documents (like non_empty_idx in R)
cleaned_docs = cleaned_docs[cleaned_docs.str.strip() != ""]

# 4. Create Term-Document Matrix & Word Frequencies
vectorizer = CountVectorizer()
tdm = vectorizer.fit_transform(cleaned_docs)

# Sum the counts of each word over all documents
word_freqs = tdm.sum(axis=0).A1  # convert to 1D array
words = vectorizer.get_feature_names_out()

# Create dataframe like df_words in R
df_words = pd.DataFrame({'word': words, 'freq': word_freqs})
df_words = df_words.sort_values(by='freq', ascending=False)

# 5. Generate Word Cloud (Full Screen)
plt.figure(figsize=(16, 9))  # Full screen size similar to options(repr.plot.width=16, repr.p

wc = WordCloud(width=1200, height=900,
               background_color='white',
               max_words=300,
               min_font_size=8,
               random_state=123,
               prefer_horizontal=0.7,
               colormap='Dark2')
```

```
wc.generate_from_frequencies(dict(zip(df_words['word'], df_words['freq'])))

plt.imshow(wc, interpolation='bilinear')
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```

8.1.4 Treemap

A Tree Map is a visual representation of hierarchical data using nested rectangles, where the size and color of each rectangle reflect quantitative values. It is especially useful for displaying proportions within categories and subcategories in a compact and intuitive way.

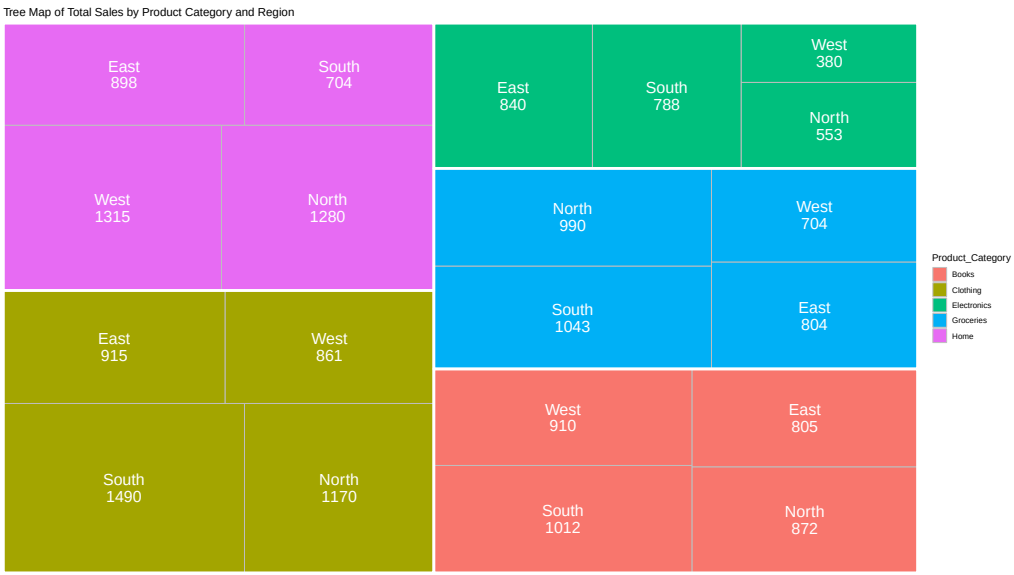
Key Characteristics of a Tree Map:

- Represents hierarchical or categorical data as nested rectangles.
- The size of each rectangle corresponds to a quantitative variable (e.g., sales volume, population, revenue).
- Can include color gradients to reflect an additional dimension, such as growth rate or change over time.
- Allows for quick comparison of relative sizes across categories and subcategories.
- Efficient use of space—can display a large amount of data in a compact form.
- Works best when comparing proportions and identifying dominant segments in a dataset.
- Commonly used in fields such as business intelligence, finance, and data dashboards.
- Ideal for visualizing data like market share, product performance, regional sales distribution, or inventory breakdown.

R Code (Treemap)

```
# ================================
# 1. Install & Load Required Packages
# ================================
packages <- c("treemapify", "dplyr", "ggplot2")
new_packages <- packages[!(packages %in% installed.packages()[, "Package"])]
if(length(new_packages)) install.packages(new_packages)

# Load libraries
library(treemapify)
library(ggplot2)
library(dplyr)

# ================================
# 2. Prepare Aggregated Treemap Data
# ================================
data_bisnis <- read.csv("data/bab8/data_bisnis.csv")
tree_data <- data_bisnis %>%
  group_by(Product_Category, Region) %>%
  summarise(
    Total_Sales = sum(Total_Price, na.rm = TRUE),
    .groups = "drop"
  ) %>%
  mutate(
    label_combined = paste0(Region, "\n", round(Total_Sales, 0))
  )

# ================================
# 3. Create Static Tree Map with Combined Labels
# ================================

ggplot(tree_data, aes(
  area = Total_Sales,
  fill = Product_Category,
  subgroup = Product_Category
)) +
  geom_treemap() +
  geom_treemap_subgroup_border(color = "white") +

  geom_treemap_text(
    aes(label = label_combined),
    colour = "white",
    place = "centre",
    grow = FALSE,
    reflow = TRUE,
    size = 50 / .pt,       # Adjust overall font size
    min.size = 3
  ) +
```

```
  labs(
    title = "Tree Map of Total Sales by Product Category and Region"
  ) +
  theme_minimal()
```

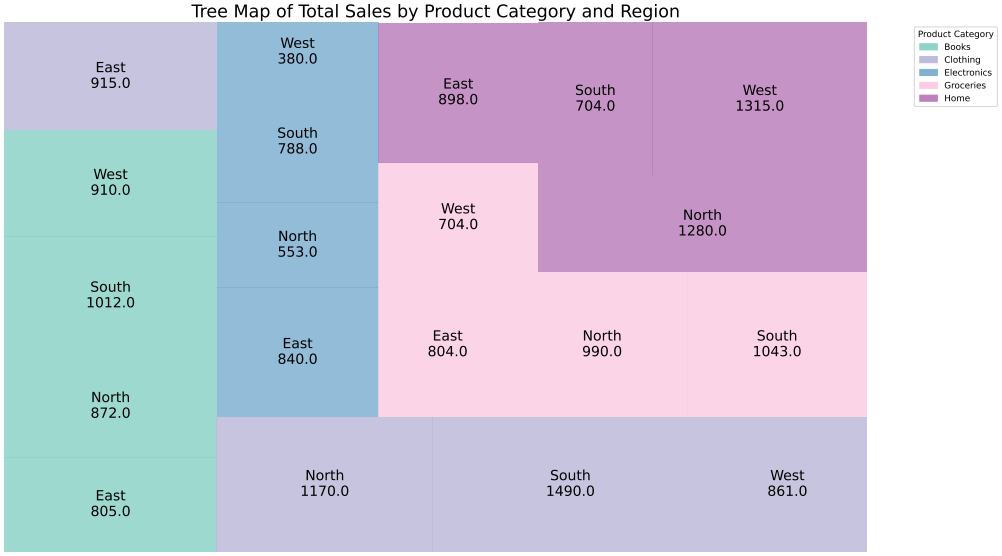


Python Code (Treemap)

```
import pandas as pd
import matplotlib.pyplot as plt
import squarify
import matplotlib.patches as mpatches

# ================================
# 1. Prepare Data
# ================================

# Load data
data_bisnis = pd.read_csv("data/bab8/data_bisnis.csv", dtype=str)

# Convert 'Total_Price' to numeric
data_bisnis['Total_Price'] = pd.to_numeric(data_bisnis['Total_Price'], errors='coerce')

# ================================
# 2. Prepare Aggregated Treemap Data
# ================================

# Aggregate sales by Product_Category and Region
tree_data = (
    data_bisnis
    .groupby(['Product_Category', 'Region'], as_index=False)
```

```
    .agg(Total_Sales=('Total_Price', 'sum'))
)

# Create combined label
tree_data['label_combined'] = tree_data.apply(
    lambda row: f"{row['Region']}\n{round(row['Total_Sales'], 0)}", axis=1
)

# ==============================
# 3. Create Static Treemap with Legend
# ==============================

# Treemap values
sizes = tree_data['Total_Sales'].values
labels = tree_data['label_combined'].values
categories = tree_data['Product_Category'].values

# Color palette like R (Set3 from ggplot2)
unique_categories = tree_data['Product_Category'].unique()
palette = plt.get_cmap('Set3')
color_dict = {
    cat: palette(i / len(unique_categories)) for i, cat in enumerate(unique_categories)
}
colors = [color_dict[cat] for cat in categories]

# Create plot
fig, ax = plt.subplots(figsize=(16, 9))
squarify.plot(
    sizes=sizes,
    label=labels,
    color=colors,
    alpha=0.85,
    ax=ax,
    text_kwargs={'fontsize': 16, 'color': 'black'}
)

# Set title and remove axis
ax.set_title("Tree Map of Total Sales by Product Category and Region", fontsize=20)
ax.axis('off')

# ==============================
# 4. Add Legend
# ==============================

# Create legend handles
legend_handles = [
    mpatches.Patch(color=color_dict[cat], label=cat) for cat in unique_categories
]
```

```
# Place legend outside the plot (right side)
plt.legend(
    handles=legend_handles,
    title='Product Category',
    bbox_to_anchor=(1.05, 1),
    loc='upper left'
)

# ==============================
# 5. Show Plot
# ==============================

plt.tight_layout()
plt.show();
```



8.2 Numerical Data

Numerical Data refers to data that consists of numbers and represents measurable quantities. It is fundamental in statistics, scientific research, economics, and data analysis due to its ability to be directly manipulated using mathematical operations.

8.2.1 Histogram

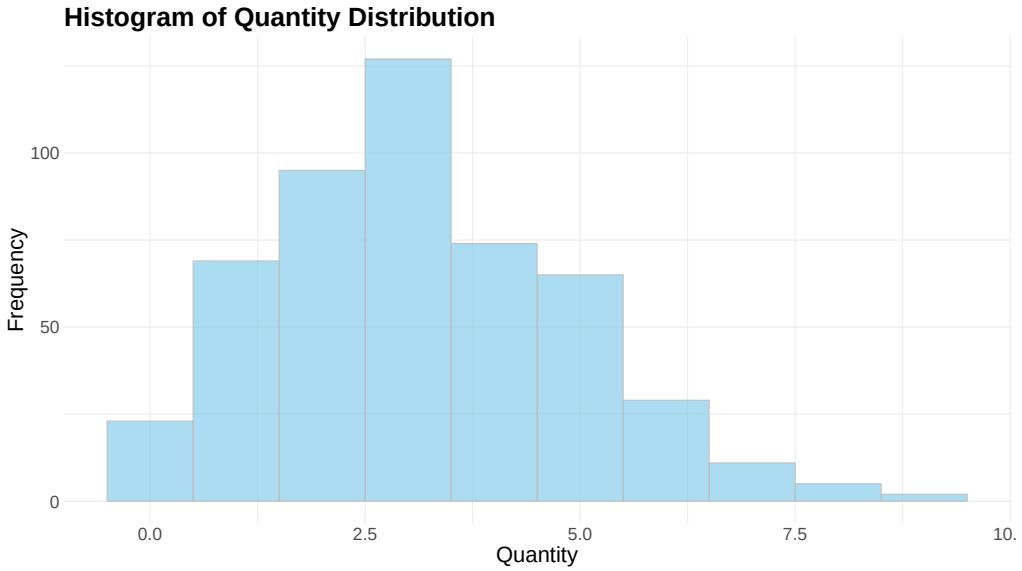
A Histogram is a graphical representation that organizes a group of data points into user-specified ranges (bins). It displays the frequency distribution of numerical data by showing how many data points fall within each bin as adjacent bars.

Key Characteristics of a Histogram:

- Visualizes the distribution of numerical data.
- Divides data into continuous intervals called bins or classes.

- The height of each bar represents the frequency (count) of data points in that bin.
- Bars are adjacent without gaps, reflecting continuous data intervals.
- Helps identify patterns such as skewness, modality (uni-, bi-, multi-modal), and spread of the data.
- Useful for detecting outliers and data concentration.
- Commonly used in statistics, quality control, finance, and many scientific fields.
- Ideal for showing data like exam scores, customer ages, sensor measurements, or production times.

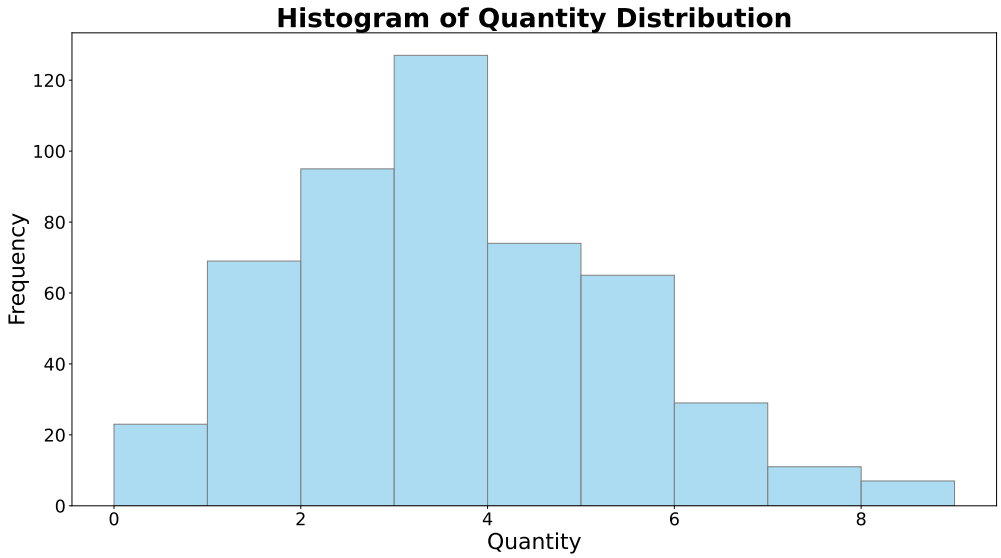
```r
# ==============================
# 1. Load Required Libraries
# ==============================
library(ggplot2)
library(dplyr)

# ==============================
# 2. Prepare Data
# ==============================
data_bisnis <- read.csv("data/bab8/data_bisnis.csv")
data_bisnis <- data_bisnis %>%
  mutate(Quantity = as.numeric(Quantity))

# ==============================
# 3. Create Histogram of Quantity with Custom Font Sizes
# ==============================
ggplot(data_bisnis, aes(x = Quantity)) +
  geom_histogram(binwidth = 1,
                 fill = "skyblue",
                 color = "gray",
                 alpha = 0.7) +
  labs(
    title = "Histogram of Quantity Distribution",
    x = "Quantity",
    y = "Frequency"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(size = 30, face = "bold"),  # Title size and bold
    axis.title.x = element_text(size = 25),               # X label size
    axis.title.y = element_text(size = 25),               # Y label size
    axis.text.x = element_text(size = 20),                # X axis numbers size
    axis.text.y = element_text(size = 20)                 # Y axis numbers size
  )
```



Python Code (Histogram)

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# ==============================
# 1. Prepare Data
# ==============================
# Assuming data_bisnis is a pandas DataFrame loaded already

# Load data
data_bisnis = pd.read_csv("data/bab8/data_bisnis.csv", dtype=str)
data_bisnis['Quantity'] = pd.to_numeric(data_bisnis['Quantity'], errors='coerce')

# Drop missing values in Quantity
data_clean = data_bisnis.dropna(subset=['Quantity'])

# ==============================
# 2. Plot Histogram with Custom Font Sizes
# ==============================
plt.figure(figsize=(16, 9))

sns.histplot(data_clean['Quantity'],
             binwidth=1,
             color='skyblue',
             alpha=0.7,
             edgecolor='gray')

plt.title('Histogram of Quantity Distribution', fontsize=30, fontweight='bold')
```

```
plt.xlabel('Quantity', fontsize=25)
plt.ylabel('Frequency', fontsize=25)

plt.xticks(fontsize=20)
plt.yticks(fontsize=20)

plt.tight_layout()
plt.show();
```



8.2.2 Density Plot

A Density Plot is a smooth, continuous curve that estimates the probability distribution of a numerical variable. Unlike histograms that group data into bins, density plots use kernel smoothing techniques to show the shape of the data distribution, providing a clearer view of underlying patterns.

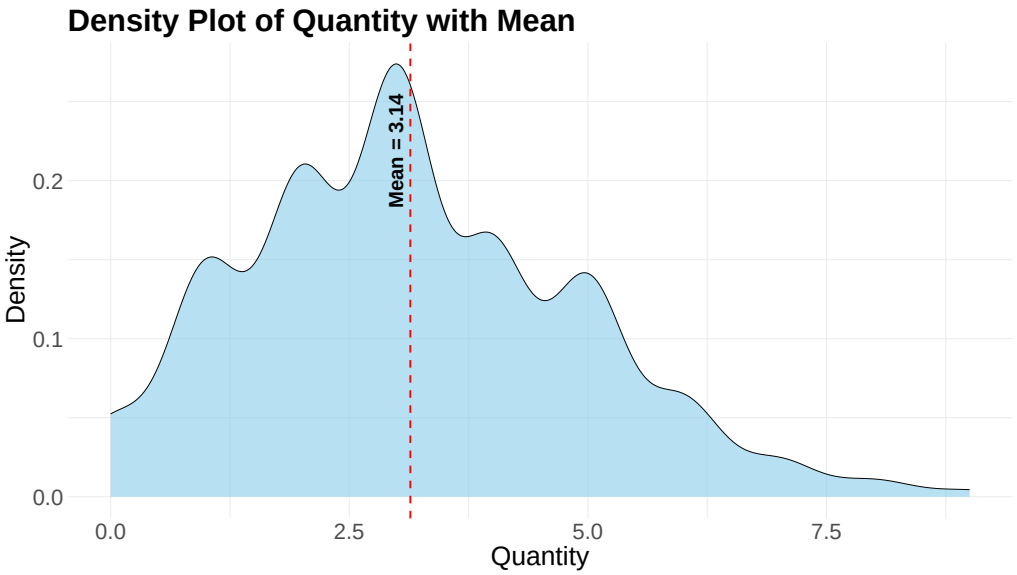
Key Characteristics of a Density Plot:

- Visualizes the distribution of numerical data as a continuous curve.
- Uses kernel density estimation (KDE) to smooth data points over the range.
- The area under the curve sums to 1, representing a probability density.
- Helps identify the shape, modality (uni-, bi-, multi-modal), and spread of the data more smoothly than histograms.
- Less sensitive to bin width or class intervals compared to histograms.
- Useful for comparing multiple distributions on the same plot.
- Commonly used in statistics, data analysis, machine learning, and scientific research.
- Ideal for showing patterns in data such as height distributions, income levels, or sensor readings.

R Code (Density-plot)

```
# ================================
# 1. Load Required Libraries
# ================================
library(ggplot2)
library(dplyr)

# ================================
# 2. Prepare Data
# ================================
data_bisnis <- read.csv("data/bab8/data_bisnis.csv")

# Ensure Quantity is numeric and remove NAs
data_bisnis <- data_bisnis %>%
  mutate(Quantity = as.numeric(Quantity)) %>%
  filter(!is.na(Quantity))

# Calculate mean of Quantity
mean_quantity <- mean(data_bisnis$Quantity, na.rm = TRUE)

# Estimate density to get y-position for label
density_data <- density(data_bisnis$Quantity)
max_y <- max(density_data$y)

# ================================
# 3. Create Density Plot with Mean Line and Label
# ================================
ggplot(data_bisnis, aes(x = Quantity)) +
  geom_density(fill = "skyblue", alpha = 0.6) +
  geom_vline(xintercept = mean_quantity, color = "red",
             linetype = "dashed", linewidth = 1) +
  geom_text(
    data = data.frame(x = mean_quantity, y = max_y * 0.8),
    aes(x = x, y = y),
    label = paste("Mean =", round(mean_quantity, 2)),
    color = "black",
    angle = 90,
    vjust = -0.5,
    size = 8,
    fontface = "bold",
    inherit.aes = FALSE
  ) +
  labs(
    title = "Density Plot of Quantity with Mean",
    x = "Quantity",
    y = "Density"
  ) +
  theme_minimal() +
```

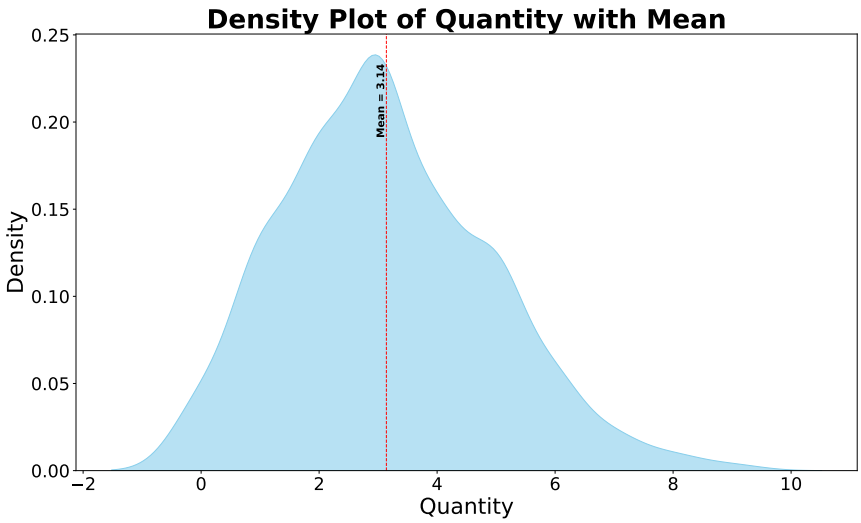
```
  theme(
    plot.title = element_text(size = 35, face = "bold"),
    axis.title = element_text(size = 30),
    axis.text = element_text(size = 25)
  )
```



Python Code (Density-plot)

```
# ================================
# 1. Load Required Libraries
# ================================
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import gaussian_kde

# ================================
# 2. Prepare Data
# ================================
# Load data
data_bisnis = pd.read_csv("data/bab8/data_bisnis.csv", dtype=str)
data_bisnis['Quantity'] = pd.to_numeric(data_bisnis['Quantity'], errors='coerce')
data_bisnis = data_bisnis.dropna(subset=['Quantity'])

mean_quantity = data_bisnis['Quantity'].mean()

# ================================
# 3. Calculate density manually (to get y max for label)
# ================================
```

```
values = data_bisnis['Quantity'].values
density = gaussian_kde(values)
x_vals = np.linspace(values.min(), values.max(), 1000)
y_vals = density(x_vals)
max_density_y = y_vals.max()

# ==============================
# 4. Plot density, mean line, and text label
# ==============================
plt.figure(figsize=(16, 9))

# Plot density with seaborn for nice fill
sns.kdeplot(data=data_bisnis, x='Quantity', fill=True, color='skyblue', alpha=0.6)

# Add vertical dashed mean line
plt.axvline(mean_quantity, color='red', linestyle='--', linewidth=1)

# Add text label near the mean line
plt.text(
    mean_quantity,
    max_density_y * 0.8,
    f'Mean = {mean_quantity:.2f}',
    rotation=90,
    verticalalignment='bottom',
    horizontalalignment='right',
    color='black',
    fontsize=12,
    fontweight='bold'
)

# Labels and title
plt.title("Density Plot of Quantity with Mean", fontsize=30, fontweight='bold')
plt.xlabel("Quantity", fontsize=25)
plt.ylabel("Density", fontsize=25)

plt.xticks(fontsize=20)
plt.yticks(fontsize=20)

plt.show();
```



8.2.3 Boxplot

A Boxplot (or box-and-whisker plot) is a graphical summary of a numerical dataset that displays its central tendency, dispersion, and skewness through five key statistics. It provides a quick visual way to identify data spread, central value, and potential outliers.

Key Characteristics of a Boxplot:

- Summarizes numerical data using five-number summary: minimum, first quartile (Q1), median (Q2), third quartile (Q3), and maximum.
- The box spans from Q1 to Q3, representing the interquartile range (IQR) which contains the middle 50% of data.
- The line inside the box marks the median (Q2), showing the central tendency.
- Whiskers extend from the box to the smallest and largest values within 1.5×IQR from the quartiles.
- Points outside the whiskers are considered outliers and plotted individually.
- Useful for comparing distributions between groups or categories.
- Efficiently highlights skewness, spread, and symmetry of the data.
- Widely used in exploratory data analysis, quality control, and statistical reporting.
- Ideal for examining test scores, financial data, experimental measurements, and any data where variability and outliers matter.

R Code (Boxplot)

```
# ==============================
# 1. Load Libraries
# ==============================
library(ggplot2)
library(dplyr)
```

```r
# ================================
# 2. Load and Prepare Data
# ================================
data_bisnis <- read.csv("data/bab8/data_bisnis.csv", stringsAsFactors = FALSE)

# Convert Quantity to numeric and filter missing
data_bisnis <- data_bisnis %>%
  mutate(Quantity = as.numeric(Quantity)) %>%
  filter(!is.na(Quantity))

# Compute IQR-based outlier bounds
Q1 <- quantile(data_bisnis$Quantity, 0.25)
Q3 <- quantile(data_bisnis$Quantity, 0.75)
IQR_value <- IQR(data_bisnis$Quantity)
lower_whisker <- Q1 - 1.5 * IQR_value
upper_whisker <- Q3 + 1.5 * IQR_value

# ================================
# 3. Summarize Statistics
# ================================
stats <- data_bisnis %>%
  summarise(
    Mean = mean(Quantity),
    Q1 = Q1,
    Median = median(Quantity),
    Q3 = Q3,
    Min = min(Quantity),
    Max = max(Quantity),
    Outliers = sum(Quantity < lower_whisker | Quantity > upper_whisker)
  )

# ================================
# 4. Basic Boxplot with Jitter and Annotations
# ================================
ggplot(data_bisnis, aes(x = factor(1), y = Quantity)) +
  # Basic boxplot
  geom_boxplot(fill = "skyblue", outlier.shape = NA) +

  # Add jittered points, highlight outliers in red
  geom_jitter(aes(color = Quantity < lower_whisker | Quantity > upper_whisker),
              width = 0.1, size = 2, alpha = 0.5) +
  scale_color_manual(values = c("FALSE" = "black", "TRUE" = "red"), guide = "none") +

  # Highlight max point if not an outlier
  geom_point(data = data_bisnis %>% filter(Quantity == stats$Max[[1]] & Quantity <= upper_
             aes(x = factor(1), y = Quantity),
             color = "red", size = 20) +

  # Annotations
```

```
  ggplot2::annotate("text", x = 1.2, y = stats$Mean[[1]],
           label = paste("Mean:", round(stats$Mean[[1]], 2)),
           hjust = 0, fontface = "bold", color = "blue") +
  ggplot2::annotate("text", x = 1.2, y = stats$Q1[[1]],
           label = paste("Q1:", round(stats$Q1[[1]], 2)),
           hjust = 0, color = "darkgreen") +
  ggplot2::annotate("text", x = 1.2, y = stats$Median[[1]],
           label = paste("Median:", round(stats$Median[[1]], 2)),
           hjust = 0, color = "purple") +
  ggplot2::annotate("text", x = 1.2, y = stats$Q3[[1]],
           label = paste("Q3:", round(stats$Q3[[1]], 2)),
           hjust = 0, color = "darkgreen") +
  ggplot2::annotate("text", x = 1.2, y = stats$Min[[1]],
           label = paste("Min:", round(stats$Min[[1]], 2)),
           hjust = 0, color = "orange") +
  ggplot2::annotate("text", x = 1.2, y = stats$Max[[1]],
           label = paste("Max:", round(stats$Max[[1]], 2)),
           hjust = 0, color = "orange") +
  ggplot2::annotate("text", x = 1, y = stats$Max[[1]] + 0.05 * stats$Max[[1]],
           label = paste("Outliers:", stats$Outliers[[1]]),
           color = "red", fontface = "italic", hjust = 0.5) +

  # Plot formatting
  labs(
    title = "Boxplot of Quantity with Jitter",
    x = NULL,
    y = "Quantity"
  ) +
  theme_minimal() +
  theme(
    axis.text.x = element_blank(),
    axis.ticks.x = element_blank(),
    plot.title = element_text(size = 50, face = "bold"),
    axis.title = element_text(size = 40),
    axis.text = element_text(size = 30)
  )
```

Python Code (Boxplot)

```
# Your task
```

8.2.4 Violin Plot

A Violin Plot is a powerful data visualization that combines the features of a boxplot with a kernel density plot, providing a richer view of the distribution of a numerical variable. It is especially useful when you want to assess both summary statistics and the distribution shape of the data.

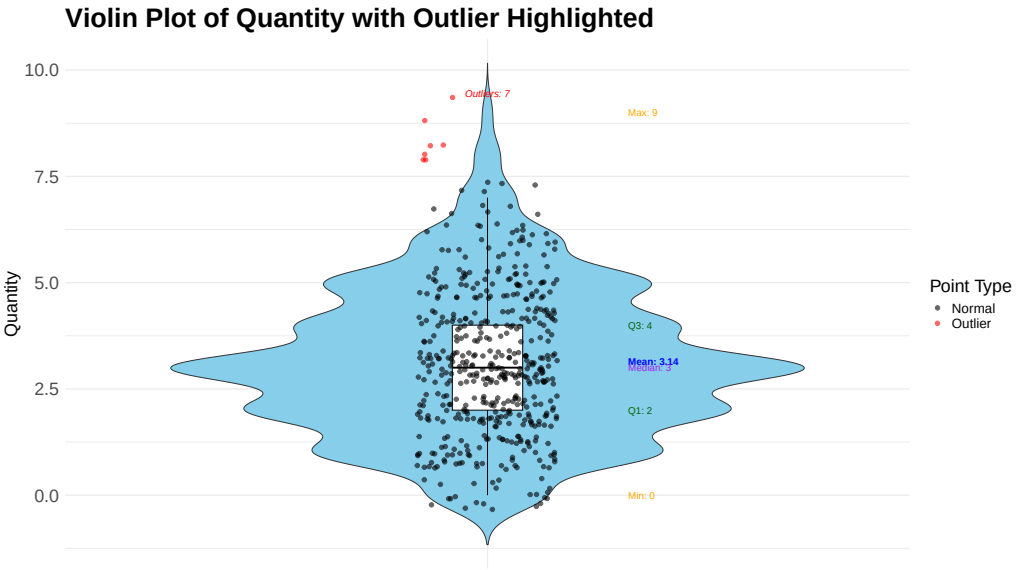
Key Characteristics of a Violin Plot:

- Combines boxplot and KDE (Kernel Density Estimate): Displays the five-number summary (like a boxplot) and a mirrored density plot showing data distribution.
- Symmetrical density shape: The wider the "violin" at a given value, the more data points exist around that value.
- Shows median and interquartile range (IQR): The central box inside the violin marks the median (Q2), first quartile (Q1), and third quartile (Q3).
- Visualizes multimodal distributions: Unlike boxplots, violin plots can reveal if the data has more than one peak (mode).
- Outliers may be plotted: Depending on the implementation (e.g., in Seaborn), individual points or extreme values can still be shown.
- Great for comparing multiple groups: Just like boxplots, violin plots are often used to compare distributions across different categories.
- Useful in identifying skewness and data spread: The shape of the violin intuitively shows if the distribution is skewed or symmetric.
- Ideal for exploratory data analysis: Especially when understanding the underlying distribution matters, such as in behavioral data, survey responses, or experimental outcomes.

R Code (Violin-plot)

```
# ==============================
# 1. Load Libraries
# ==============================
library(ggplot2)
library(dplyr)

# ==============================
# 2. Load and Prepare Data
# ==============================
data_bisnis <- read.csv("data/bab8/data_bisnis.csv", stringsAsFactors = FALSE)

# Clean and convert Quantity to numeric
data_bisnis <- data_bisnis %>%
  mutate(Quantity = as.numeric(Quantity)) %>%
  filter(!is.na(Quantity))

# Calculate quartiles and IQR for outlier detection
Q1 <- quantile(data_bisnis$Quantity, 0.25)
Q3 <- quantile(data_bisnis$Quantity, 0.75)
IQR_value <- IQR(data_bisnis$Quantity)
upper_whisker <- Q3 + 1.5 * IQR_value
lower_whisker <- Q1 - 1.5 * IQR_value

# Mark outliers
data_bisnis <- data_bisnis %>%
  mutate(
    is_outlier = ifelse(Quantity < lower_whisker | Quantity > upper_whisker, "Outlier", "Norm
  )

# ==============================
# 3. Summarize Statistics
# ==============================
stats <- data_bisnis %>%
  summarise(
    Mean = mean(Quantity),
    Q1 = Q1,
    Median = median(Quantity),
    Q3 = Q3,
    Min = min(Quantity),
    Max = max(Quantity),
    Outliers = sum(is_outlier == "Outlier")
  )

# ==============================
# 4. Create Violin Plot with Colored Jitter and Annotations
# ==============================
ggplot(data_bisnis, aes(x = factor(1), y = Quantity)) +
```

```
  geom_violin(fill = "skyblue", trim = FALSE) +
  geom_boxplot(width = 0.1, outlier.shape = NA, color = "black") +
  geom_jitter(aes(color = is_outlier), width = 0.1, alpha = 0.6, size = 2) +
  geom_point(data = data_bisnis %>%
               filter(Quantity == stats$Max[[1]] & Quantity <= upper_whisker),
             aes(x = factor(1), y = Quantity),
             color = "red", size = 8) +

  # Annotations via geom_text
  geom_text(data = stats, aes(x = 1.2, y = Mean, label = paste("Mean:", round(Mean, 2))),
            hjust = 0, color = "blue", fontface = "bold") +
  geom_text(data = stats, aes(x = 1.2, y = Q1, label = paste("Q1:", round(Q1, 2))),
            hjust = 0, color = "darkgreen") +
  geom_text(data = stats, aes(x = 1.2, y = Median, label = paste("Median:", round(Median,
            hjust = 0, color = "purple") +
  geom_text(data = stats, aes(x = 1.2, y = Q3, label = paste("Q3:", round(Q3, 2))),
            hjust = 0, color = "darkgreen") +
  geom_text(data = stats, aes(x = 1.2, y = Min, label = paste("Min:", round(Min, 2))),
            hjust = 0, color = "orange") +
  geom_text(data = stats, aes(x = 1.2, y = Max, label = paste("Max:", round(Max, 2))),
            hjust = 0, color = "orange") +
  geom_text(data = stats, aes(x = 1, y = Max + 0.05 * Max,
                              label = paste("Outliers:", Outliers)),
            color = "red", fontface = "italic", hjust = 0.5) +

  scale_color_manual(values = c("Normal" = "black", "Outlier" = "red")) +

  labs(
    title = "Violin Plot of Quantity with Outlier Highlighted",
    x = NULL,
    y = "Quantity",
    color = "Point Type"
  ) +
  theme_minimal() +
  theme_minimal(base_size = 15) +
  theme(
    axis.text.x = element_blank(),
    axis.ticks.x = element_blank(),
    plot.title = element_text(size = 30, face = "bold"),
    axis.title = element_text(size = 20),
    axis.text = element_text(size = 20),
    legend.position = "right",
    legend.title = element_text(size = 20),
    legend.text = element_text(size = 15)
  )
```



Pyton Code (Violin-plot)

```
# Your task
```

8.3 Combo

These combine categorical and numerical data to show how a numerical variable behaves across different categories. Examples are grouped bar charts, boxplots by category, ridgeline plots, lollipop charts, dot plots, and heatmaps.

8.3.1 Grouped Bar Chart

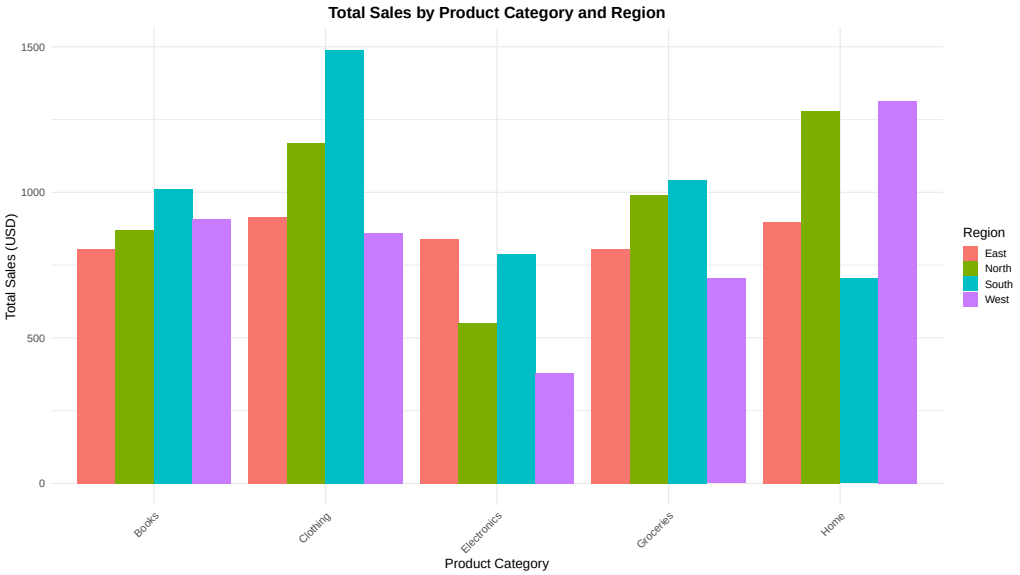
A Grouped Bar Chart (sometimes called a clustered bar chart) is a type of bar chart that displays multiple bars grouped by categories, allowing comparison of subgroups within each main category. It is an effective way to visualize and compare values across two categorical variables simultaneously.

Key Characteristics of Grouped Bar Charts:

- Multiple bars per category: Each group (main category) contains two or more bars representing subcategories or levels of another categorical variable.
- Side-by-side comparison: Bars within a group are placed next to each other (not stacked), facilitating direct visual comparison of subgroups.
- Categorical vs categorical + numerical: Typically used when one variable is categorical (groups) and the other variable provides numerical values to be compared.
- Easy to interpret: Heights or lengths of bars represent quantitative values, making differences among subgroups easy to detect.
- Color or pattern coding: Different colors or patterns are used to distinguish the subgroups within each category.
- Useful for trend and proportion analysis: Helps to identify trends, differences, or similarities among subgroups across categories.

- Common applications: Market research (comparing sales by product type across regions), survey results (response rates by age groups and gender), and many others.

```r
# ================================
# 1. Load Libraries
# ================================
library(ggplot2)
library(dplyr)

# ================================
# 2. Load Data
# ================================
data_bisnis <- read.csv("data/bab8/data_bisnis.csv", stringsAsFactors = FALSE)

# ================================
# 3. Data Summarization
# ================================
sales_summary <- data_bisnis %>%
  group_by(Product_Category, Region) %>%
  summarise(Total_Sales = sum(Total_Price, na.rm = TRUE), .groups = "drop")

# ================================
# 4. Plot Grouped Bar Chart
# ================================
ggplot(sales_summary, aes(x = Product_Category, y = Total_Sales, fill = Region)) +
  geom_bar(stat = "identity", position = position_dodge()) +
  labs(
    title = "Total Sales by Product Category and Region",
    x = "Product Category",
    y = "Total Sales (USD)",
    fill = "Region"
  ) +
  theme_minimal(base_size = 15) +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1),
    plot.title = element_text(face = "bold", hjust = 0.5)
  )
```



Pyton Code (Grouped Bar-chart)

```
# Your task
```

8.3.2 Ridgeline Plot

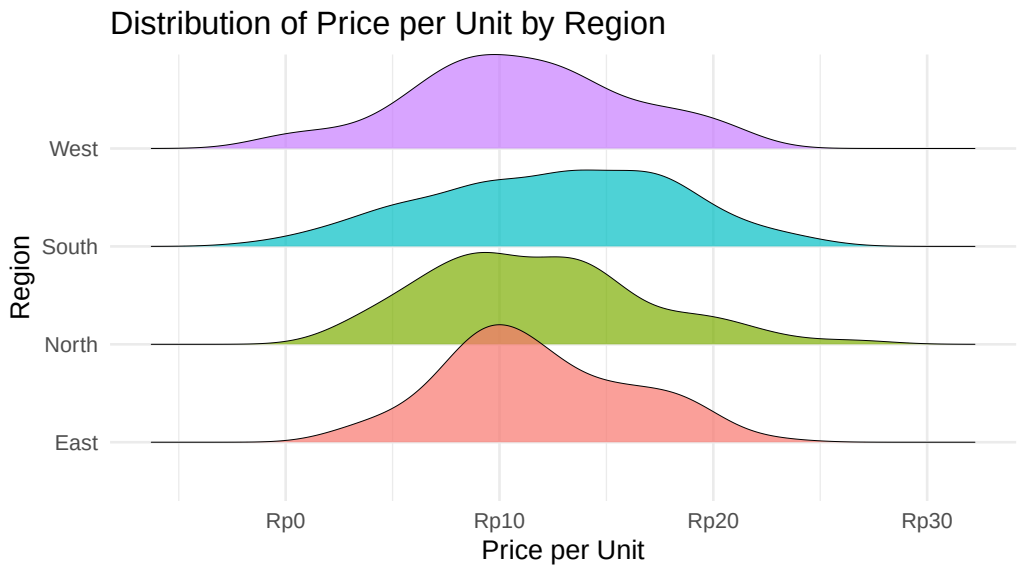
A Ridgeline Plot is a compelling data visualization technique used to display the distribution of a numerical variable across multiple groups. It is particularly useful when comparing the density distributions of different categories in a visually efficient and aesthetically pleasing manner.

Key Characteristics of a Ridgeline Plot:

- Overlapping KDEs across categories: It shows multiple kernel density estimates 8 (KDEs), each representing a subgroup, stacked vertically with partial overlap for easy comparison.
- Efficient space usage: By partially overlapping each plot, ridgeline plots allow for the comparison of many distributions without cluttering the view.
- Reveals distribution shapes: Similar to violin plots, ridgeline plots expose skewness, modality (number of peaks), and spread of data in each group.
- Customizable aesthetics: The amount of overlap, color gradient, and smoothing bandwidth can be adjusted to enhance readability or highlight differences.
- Ideal for temporal or categorical trends: Often used in time series analysis or grouped data to examine how a distribution changes across levels of a factor (e.g., years, regions, categories).
- Not focused on exact values: Unlike boxplots, it doesn't emphasize specific statistics (like median or quartiles) but rather the overall shape and relative density of the data.
- Visually engaging: Its layered, flowing style makes it popular in publications and dashboards for its ability to convey complex patterns in an intuitive and attractive format.

- Good for medium to large datasets: The KDE estimates are smoother with more data and can become cluttered or misleading with very small sample sizes.

```r
# ================================
# 1. Load Libraries
# ================================
library(ggridges)
library(ggplot2)
library(dplyr)
library(scales)

# ================================
# 2. Filter Valid Data
# ================================
# Filter out rows where Price_per_Unit is NA, Inf, or NaN
data_bisnis <- read.csv("data/bab8/data_bisnis.csv", stringsAsFactors = FALSE)

data_bisnis_filtered <- data_bisnis %>%
  filter(is.finite(Price_per_Unit))

# ================================
# 3. Create Ridgeline Plot
# ================================
ggplot(data_bisnis_filtered, aes(x = Price_per_Unit, y = Region, fill = Region)) +
  geom_density_ridges(alpha = 0.7, scale = 1.2) +
  scale_x_continuous(labels = dollar_format(prefix = "Rp", big.mark = ".", decimal.mark =
  labs(
    title = "Distribution of Price per Unit by Region",
    x = "Price per Unit",
    y = "Region"
  ) +
  theme_minimal() +
  theme_minimal(base_size = 30) +
  theme(legend.position = "none")
```



Python Code (Ridgeline-plot)

```
# Your task
```

8.3.3 Boxplot by Category

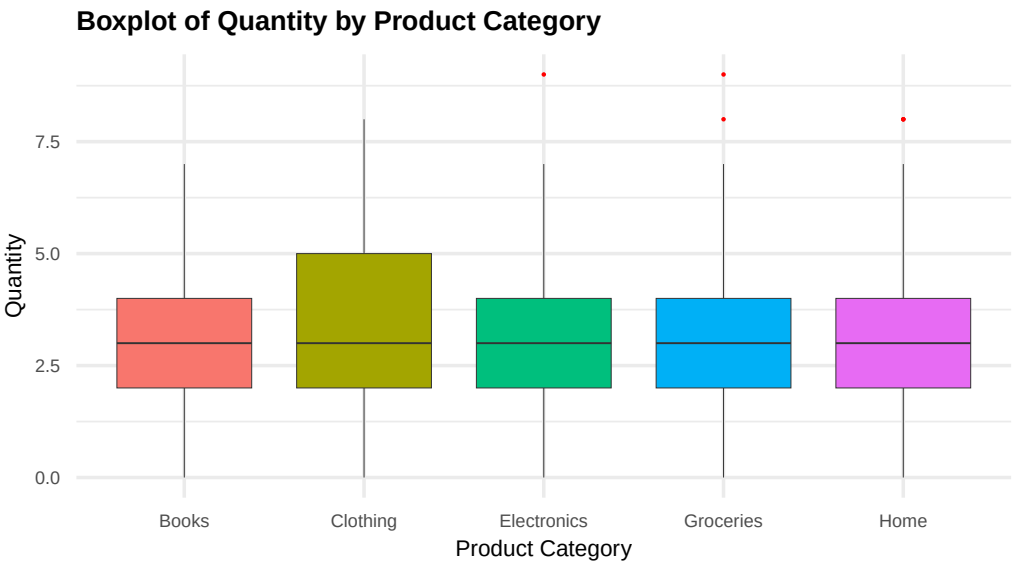
A Boxplot by Category is a fundamental data visualization technique used to summarize the distribution of a numerical variable across different categorical groups. It is especially useful for comparing statistical properties like median, spread, and outliers between multiple categories in a clear and concise way.

Key Characteristics of a Boxplot by Category:

- Summarizes key statistics: Boxplots display the median, quartiles (25th and 75th percentiles), interquartile range (IQR), and potential outliers for each category, giving a quick summary of the distribution.
- Visualizes spread and skewness: The length of the box and whiskers reveals variability and potential skewness within each category.
- Detects outliers: Points outside 1.5 times the IQR are flagged as outliers, helping to identify unusual or extreme values in the data.
- Categorical comparison: By plotting multiple boxplots side-by-side, it allows direct visual comparison of distributions across categories.
- Robust to sample size: Boxplots can represent data distributions reliably even with moderately sized samples.
- Non-parametric and distribution-agnostic: Boxplots do not assume any specific data distribution, making them versatile for various types of numeric data.
- Widely used in exploratory data analysis: They provide immediate insight into group differences, data symmetry, and range.
- Customizable: Boxplots can be colored or grouped further by additional categorical variables to reveal interaction effects.
- Ideal for detecting data quality issues: Outliers and inconsistencies can be easily spotted, prompting further investigation.

R Code (Boxplot by Category)

```r
# ================================
# 1. Load Required Libraries
# ================================
library(ggplot2)
library(dplyr)

# ================================
# 2. Prepare Data
# ================================
# Convert Quantity to numeric and remove NA
data_bisnis <- read.csv("data/bab8/data_bisnis.csv", stringsAsFactors = FALSE)
data_bisnis <- data_bisnis %>%
  mutate(Quantity = as.numeric(Quantity)) %>%
  filter(!is.na(Quantity))

# ================================
# 3. Create Boxplot
# ================================
ggplot(data_bisnis, aes(x = Product_Category, y = Quantity, fill = Product_Category)) +
  geom_boxplot(outlier.colour = "red", outlier.shape = 16, outlier.size = 2) +  # Boxplot
  labs(
    title = "Boxplot of Quantity by Product Category",
    x = "Product Category",
    y = "Quantity"
  ) +
  theme_minimal() +
  theme_minimal(base_size = 40) +
  theme(
    plot.title = element_text(size = 30, face = "bold"),
    axis.title = element_text(size = 25),
    axis.text = element_text(size = 20),
    legend.position = "none"
  )
```



Python Code (Boxplot by Category)

```
# Your task
```

8.3.4 Lollipop Chart

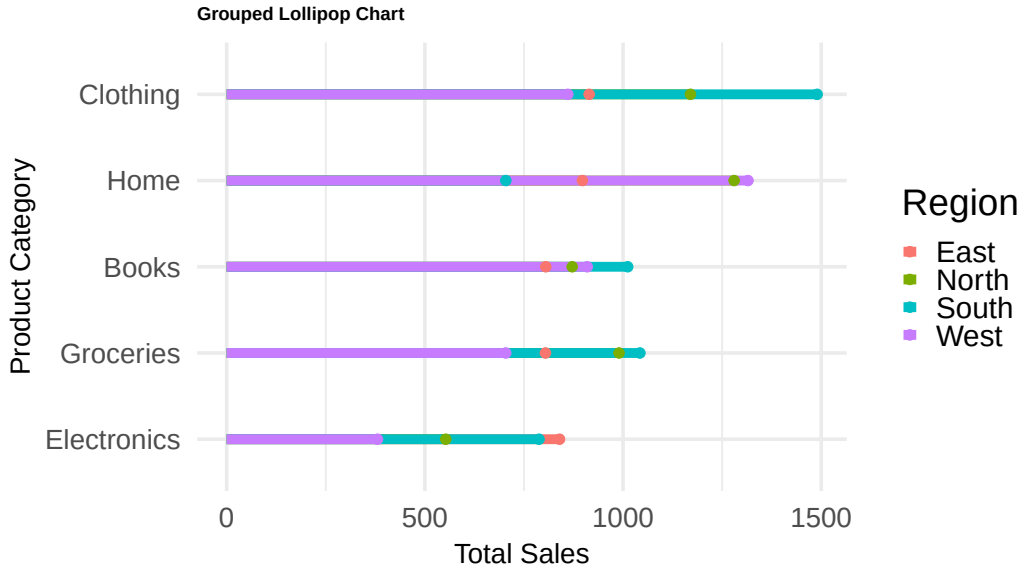
A Lollipop Chart is a clean and visually appealing alternative to bar charts, used to display and compare values across categories. It combines the simplicity of a dot plot with a line that connects each dot to the baseline, making it easier to perceive the magnitude of each value.

Key Characteristics of a Lollipop Chart:

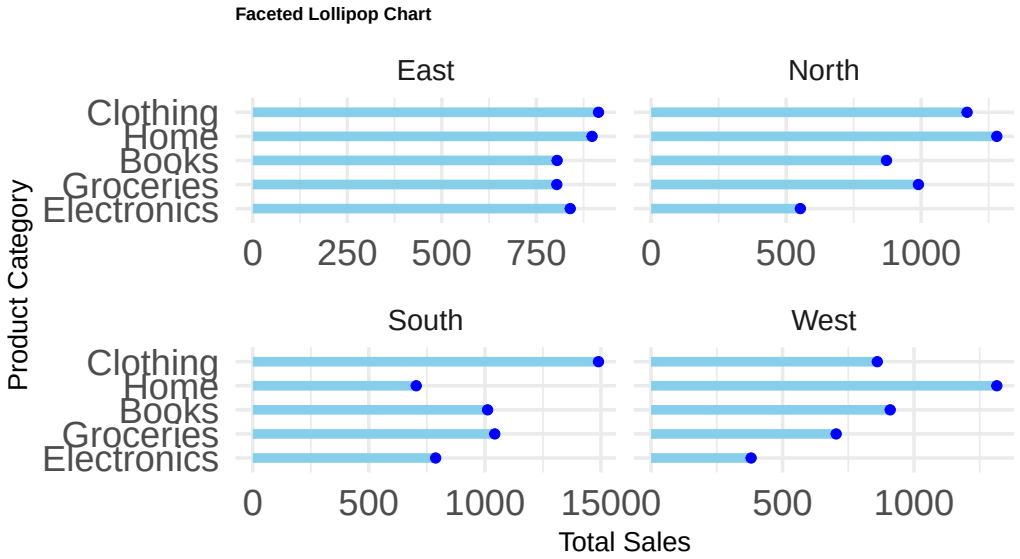
- Minimalist and clear: It emphasizes individual data points with dots, connected by thin lines to a baseline, reducing visual clutter compared to traditional bar charts.
- Easy value comparison: The length of the "stick" and the position of the dot allow quick comparison of category values.
- Highlights exact data points: Unlike bar charts, the dot makes it easier to focus on the precise value rather than the area of a bar.
- Useful for ranked or ordered data: Lollipop charts work well when categories are ordered by magnitude or rank, enhancing the visual storytelling.
- Customizable aesthetics: Colors, dot sizes, and line thickness can be adjusted to improve readability or highlight specific categories.
- Good for datasets with moderate number of categories: It maintains clarity without overcrowding, suitable for datasets where bar charts might become visually heavy.
- Visually engaging: Its sleek design is popular in dashboards and reports for conveying data in an elegant, intuitive way.
- Supports grouping or faceting: Can be combined with grouping variables or facets to compare subcategories within each category.

R Code (Lollipop Chart)

```r
# ================================
# 1. Load Required Libraries
# ================================
library(ggplot2)
library(dplyr)

# ================================
# 2. Prepare Data
# ================================
data_bisnis <- read.csv("data/bab8/data_bisnis.csv", stringsAsFactors = FALSE)

# Summarize total sales by Product_Category and Region
sales_grouped <- data_bisnis %>%
  group_by(Product_Category, Region) %>%
  summarise(Total_Sales = sum(Total_Price, na.rm = TRUE), .groups = "drop")

# ================================
# 3. Grouped Lollipop Chart
# ================================
ggplot(sales_grouped, aes(x = Total_Sales, y = reorder(Product_Category, Total_Sales), col
  geom_segment(aes(x = 0, xend = Total_Sales, y = Product_Category, yend = Product_Categor
  geom_point(size = 5) +
  labs(
    title = "Grouped Lollipop Chart",
    x = "Total Sales",
    y = "Product Category"
  ) +
  theme_minimal() +
  theme_minimal(base_size = 40) +
  theme(
    axis.text = element_text(size = 30),
    axis.title = element_text(size = 30),
    plot.title = element_text(size = 20, face = "bold")
  )
```




```
# ==============================
# 4. Faceted Lollipop Chart
# ==============================
ggplot(sales_grouped, aes(x = Total_Sales, y = reorder(Product_Category, Total_Sales))) +
  geom_segment(aes(x = 0, xend = Total_Sales, y = Product_Category, yend = Product_Category),
  geom_point(color = "blue", size = 5) +
  facet_wrap(~ Region, scales = "free_x") +
  labs(
    title = "Faceted Lollipop Chart",
    x = "Total Sales",
    y = "Product Category"
  ) +
  theme_minimal() +
  theme_minimal(base_size = 40) +
  theme(
    axis.text = element_text(size = 40),
    axis.title = element_text(size = 30),
    plot.title = element_text(size = 20, face = "bold")
  )
```



Python Code (Lollipop Chart)

```
# Your task
```

8.3.5 Heatmap

A Heatmap is a graphical representation of data where individual values contained in a matrix are represented as colors. It is widely used to visualize complex data by encoding data values as varying colors, making patterns, correlations, and outliers easier to detect.

Key Characteristics of a Heatmap:

- Color-Coded Values: Each cell's color intensity corresponds to the magnitude of the value it represents, often using a gradient from low to high.
- Matrix Format: Typically displayed as a grid where rows and columns correspond to categorical variables or dimensions.
- Pattern Recognition: Helps identify clusters, trends, and anomalies in large datasets quickly.
- Flexible Applications: Commonly used in fields like genomics, finance, marketing, and business analytics to visualize correlations, sales performance, or activity levels.
- Interactive Variants: Often paired with interactivity (zoom, hover details) in dashboards for deeper exploration.
- Customization: Color palettes, scales, and clustering methods can be adjusted for clarity and emphasis.
- Relationship Focused: Unlike simple bar charts, heatmaps show relationships between two categorical variables by their aggregated values.

R Code (Heatmap)

```
# Your task
```

Python Code (Heatmap)

```
# Your task
```

8.4 Relationship

Relationship data visualization refers to graphical methods designed to explore and present the connections, associations, or interactions between two or more variables or entities. These visualizations help reveal patterns, trends, correlations, or networks that may not be obvious from raw data alone.

8.4.1 Scatter Plot

A scatter plot is a fundamental data visualization tool used to display the relationship between two continuous variables. Each point on the plot represents an observation with its position determined by the values of the two variables.

Key Characteristics of a Scatter Plot:

- Bivariate relationship: Shows how one variable changes with respect to another.
- Detects correlation: Helps identify positive, negative, or no correlation.
- Outlier detection: Reveals unusual data points.
- Supports additional aesthetics: Color, size, and shape can represent more variables for multidimensional insights.
- Common in exploratory data analysis (EDA): Provides a quick visual summary of data relationships.

R Code (Scatter)

```
# Your task
```

Python Code (Scatter)

```
# Your task
```

8.4.2 Bubble Chart

A Bubble Chart is an extension of the scatter plot that visualizes three dimensions of data. It plots points like a scatter plot but uses the size of each bubble to represent a third variable, allowing richer insight into relationships between variables.

R Code (Scatter)

```
# Your task
```

Python Code (Scatter)

```
# Your task
```

8.4.3 Correlation Matrix

A Correlation Matrix is a table showing correlation coefficients between many variables. Each cell in the matrix represents the correlation between two variables, typically measured by Pearson's correlation coefficient. It helps you quickly understand relationships, patterns, and dependencies in multivariate data.

Key Characteristics:

- Values range from -1 to +1:
- +1 means perfect positive correlation,
- -1 means perfect negative correlation,
- 0 means no linear correlation.
- Useful for detecting multicollinearity and feature relationships in data analysis.
- Often visualized with a heatmap or colored matrix, where colors indicate strength and direction of correlations.
- Important in fields like finance, health sciences, and machine learning feature selection.

R Code (Correlation)

```
# Your task
```

Python Code (Correlation)

```
# Your task
```

8.5 Time Series

Time Series Visualization is the graphical representation of data points collected or recorded at successive time intervals. It is essential for identifying trends, seasonal patterns, cycles, and anomalies over time.

Key Characteristics:

- Temporal Ordering: Data points are ordered by time (seconds, minutes, days, months, years).
- Trend Detection: Helps identify long-term increase or decrease in data.
- Seasonality: Reveals repeating patterns or cycles within fixed periods.
- Anomalies: Detects outliers or unusual changes.
- Common Plots: Line charts, area charts, and stacked graphs.
- Multivariate Time Series: Multiple related time series can be plotted using facets or colors to compare.

8.5.1 Line Chart

A Line Chart is one of the most common and intuitive ways to visualize data points connected by straight lines, especially effective for showing trends over time.

R Code (Line Chart)

```
# Your task
```

Python Code (Line Chart)

```
# Your task
```

8.5.2 Area Chart

An Area Chart is a variation of the line chart where the area between the line and the axis is filled with color. It emphasizes the magnitude of values over time and is useful to visualize cumulative quantities or proportions

R Code (Line Chart)

```
# Your task
```

Python Code (Line Chart)

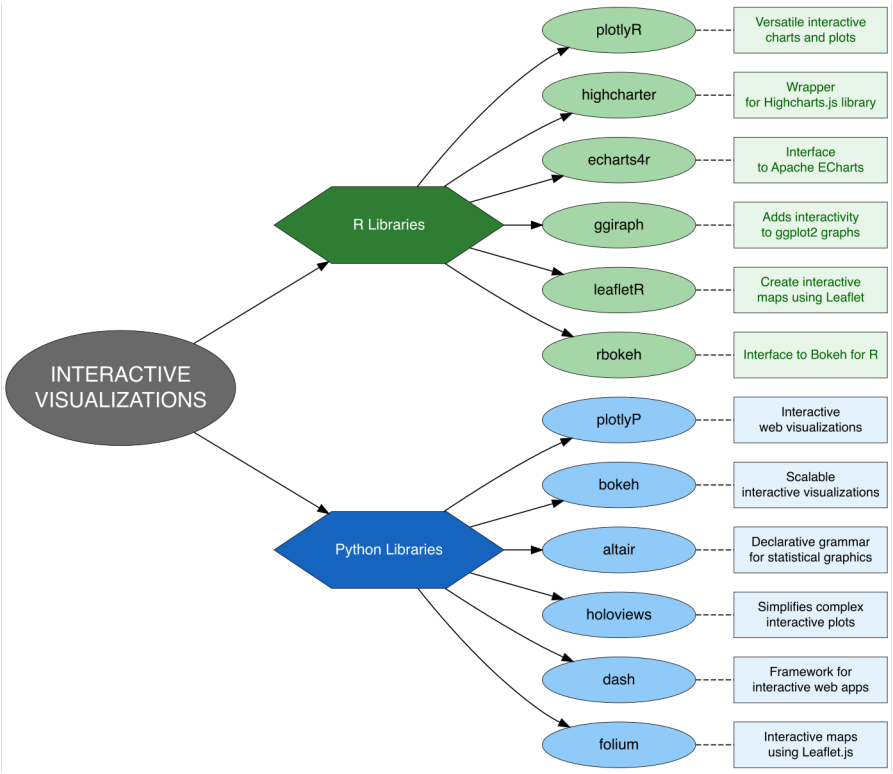
```
# Your task
```

Chapter 9

Interactive Visualizations

In the world of data visualization, interactivity has become essential for deeper insight and user engagement. While plotly is one of the most popular libraries for interactive graphics in R or Python, there are several powerful alternatives that offer similar—and in some cases, more specialized—capabilities.

These libraries provide web-based, dynamic, and responsive visualizations ideal for dashboards, exploratory data analysis, and communication of complex insights. Depending on the context (e.g., time series, maps, statistical charts), different tools may offer better performance, flexibility, or aesthetics.



Let say you have dataset like this:

Copy CSV PDF Print Search:

Transformed Business Dataset

	Transaction_ID	Transaction_Date	Customer_ID	Product_Category	Product_ID	Quantity	Unit_Price	Discount	Region	Sales_Channel	Delivery_Time
1	7zmPHxF7XfN9	2021-07-14	BAl3Y7yxev	Clothing	P0370	2	15.18	0	North	Online	
2	y4bCY9pKTBWU	2020-11-16	TYY0h5C190	Electronics	P0185	5	10.22	0.15	West	Offline	
3	8k0B7XX19Ykf	2023-03-22	nUX640AaXg	Home	P0443	3	17.74	0.05	West	Online	
4	l8ahQz5YNOKz	2023-01-02	sBZyUSJLEP	Home	P0035	6	28.3	0.22	North	Offline	
5	kmufgw8wx5qk	2023-06-05	GMfVH2ZWNX	Groceries	P0375	3	11.91	0.13	North	Offline	
6	al0KADT0mn7C	2023-03-15	YxqAmfTU9M	Clothing	P0447	1	5.43	0.07	North	Offline	
7	zu0iP1OBFuNl	2021-09-25	ifYw95qmoL	Groceries	P0345	2	13.37	0.3	West	Online	
8	l0Bz1iXOUzUy	2020-02-18	F5zL0GjDJh	Clothing	P0193	5	6.56	0.23	South	Online	
9	kgU3mL1c8BdA	2023-02-25	Zm9ROmGygf	Clothing	P0114	4	18.23	0.27	South	Offline	
10	MDp09wloF78Q	2023-08-19	fgldeSSYSL	Home	P0095	1	11.94	0.09	South	Online	
11	M8Wk6MDt6qq1	2020-01-24	y5n7T2vCfd	Electronics	P0309	2	21.06	0.2	South	Offline	
12	lH96iSOc84fn	2020-12-21	SS9WdTh6Gp	Books	P0402	4	9.38	0.01	North	Online	
13	NnorQ5sHloaf	2024-06-12	9l5PBRrmgl	Clothing	P0135	3	15.61	0.17	South	Online	
14	dEbUZoU5Sbu2	2020-06-13	A03t0cAGgC	Clothing	P0312	1	9.17	0.15	South	Offline	
15	o5Gqu5Y2GBZ4	2021-09-13	7hcyxq0KSE	Groceries	P0498	1	16.31	0.24	West	Online	
16	RYXFAjCauCpJ	2024-04-04	A9pnVHddZb	Electronics	P0098	2	18.49	0.09	East	Offline	

Showing 1 to 500 of 500 entries

9.1 R Libraries

R offers a rich ecosystem of libraries for creating interactive and dynamic visualizations, enabling deeper data exploration and engagement. Here are some popular options beyond the widely-used plotly:

9.1.1 plotly

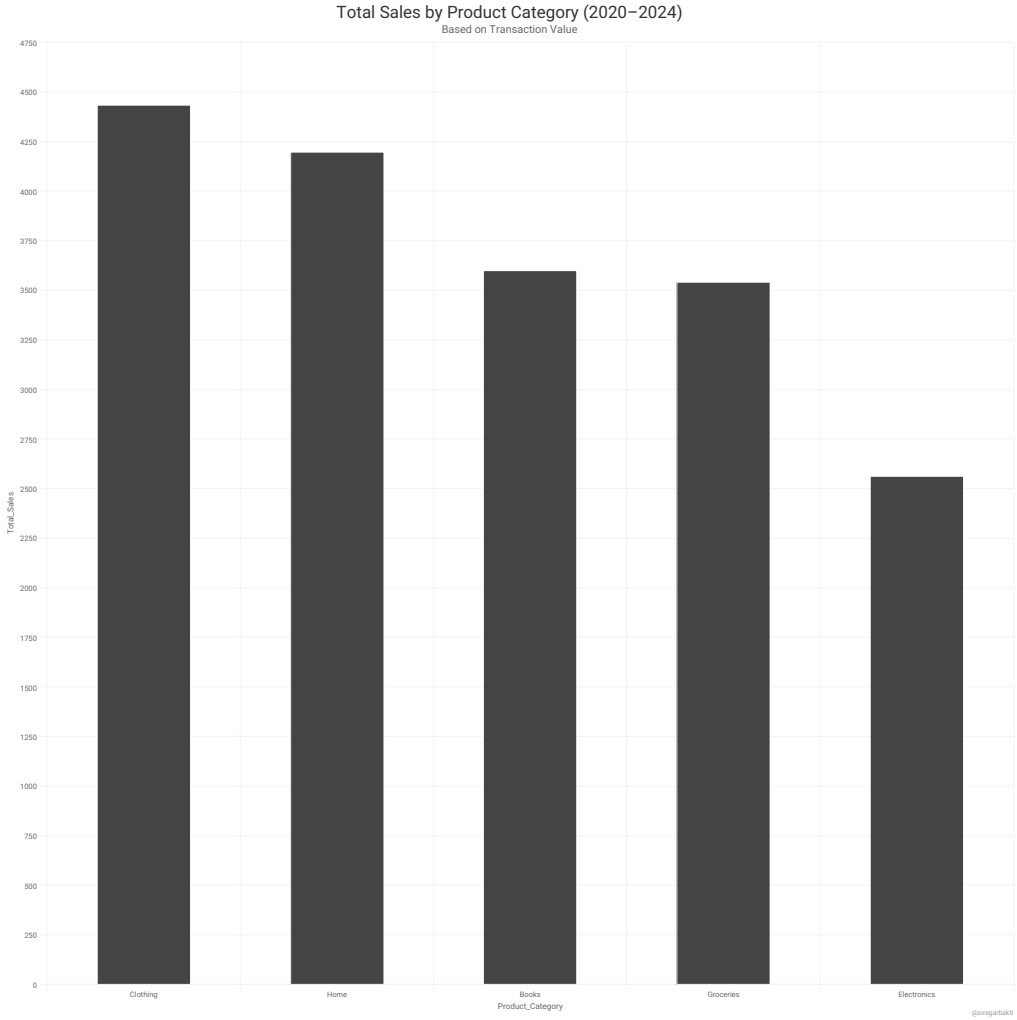
A versatile library for creating interactive, web-based graphs that support zooming, panning, and hover tooltips.

9.1.2 highcharter

A wrapper around the popular Highcharts.js library, known for highly customizable and polished visualizations suitable for dashboards.

```
## Required libraries
library(dplyr)        # Used for data wrangling: grouping, summarizing, etc.
library(highcharter)  # Used for creating interactive charts
library(viridis)      # Provides color palettes with good perceptual properties

# Create a color palette based on the number of product categories
custom_colors <- viridis::turbo(n=length(unique(data_bisnis$Product_Category)))

# Begin visualization pipeline
data_bisnis %>%

# Step 1: Group the dataset by Product Category
group_by(Product_Category) %>%

# Step 2: Calculate total sales (sum of Total_Price) for each category
summarise(Total_Sales = sum(Total_Price, na.rm = TRUE)) %>%

# Step 3: Sort the result in descending order of total sales
arrange(desc(Total_Sales)) %>%
```

```
# Step 4: Create a column chart using highcharter
hchart('column',                    # Specify chart type
    hcaes(
      x = Product_Category,         # X-axis = product category
      y = Total_Sales,              # Y-axis = total sales value
      color = custom_colors         # Apply custom colors to each bar
    )
  ) %>%

# Step 5: Apply a Google-themed style
hc_add_theme(hc_theme_google()) %>%

# Step 6: Add a tooltip showing the total sales when hovering over a bar
hc_tooltip(pointFormat = '<b>Total Sales: </b> Rp {point.y:,.0f}<br>') %>%

# Step 7: Add a main title to the chart
hc_title(
  text = 'Total Sales by Product Category (2020-2024)',
    style = list(fontSize = '25px', fontWeight = 'bold')
) %>%

# Step 8: Add a subtitle below the main title
hc_subtitle(
    text = 'Based on Transaction Value',
    style = list(fontSize = '16px')
  ) %>%

# Step 9: Add credits to the bottom-right corner of the chart
hc_credits(
  enabled = TRUE,
  text = '<a href="https://www.linkedin.com/in/dsciencelabs/" target="_blank">@siregarbakti</
  style = list(fontSize = '10px', fontWeight = 'bold')
)
```



9.1.3 echarts4r

Based on Apache ECharts, it excels at rendering large datasets with beautiful, flexible, and responsive charts.

9.1.4 ggiraph

Enhances ggplot2 plots by adding interactivity such as tooltips and hover effects, integrating seamlessly with familiar syntax.

9.1.5 leaflet

Specializes in interactive geospatial visualizations and mapping, perfect for spatial data applications.

9.1.6 rbokeh

An R interface to the Python Bokeh library, designed for elegant and interactive browser-based plots.

9.2 Python Libraries

9.2.1 plotly

Python offers the same interactive charting capabilities as in R. Built on plotly.js, it allows users to create dynamic, web-based visualizations with ease.

9.2.2 Bokeh

Designed for creating interactive and scalable visualizations for the web. Excellent for streaming data and linked plots.

9.2.3 Altair

A declarative statistical visualization library based on the Vega and Vega-Lite grammars, focused on simplicity and concise syntax.

9.2.4 HoloViews

Built on top of Bokeh and Matplotlib, it simplifies building complex interactive plots with minimal code.

9.2.5 Dash

Developed by the Plotly team, Dash is a framework for building interactive web applications with Python, ideal for creating full-featured dashboards.

9.2.6 Folium

For interactive maps and geospatial visualization, built on Leaflet.js.

Choosing the right tool depends on your familiarity with the syntax, the type of data, and the intended audience or platform (e.g., static report vs. dynamic dashboard).

Part IV

Applied DSP

Chapter 10

Automation Report

10.1 Report Overview

- **Purpose**: Summary of what the automated report covers (e.g., sales trend analysis, model performance monitoring).
- **Frequency**: Daily, weekly, monthly, or on-demand.
- **Data Source**: Databases, APIs, spreadsheets, or other integrated sources.

10.2 Data Summary

- **Last Updated**: `YYYY-MM-DD HH:MM`
- **Total Records Processed**: `XXXX`
- **Data Quality Checks**:
  - Missing Values: `X%`
  - Duplicate Records: `X entries`
  - Outliers Detected: `X cases`

10.3 Descriptive Statistics

10.3.1 Categorical Variables

- Frequency Tables
- Top Categories by Volume

10.3.2 Numerical Variables

- Mean, Median, Mode
- Range, Variance, Standard Deviation
- Skewness, Kurtosis

10.4 Visualizations

- **Time Series Plot**: Trends over time

- **Bar Charts**: Category comparison
- **Boxplots**: Distribution and outliers
- **Heatmaps**: Correlation matrix

10.5 Alerts & Thresholds

- Any metric exceeding predefined limits
- Notification status: Sent / Not Sent

10.6 Recommendations

- Suggestions based on data trends or model results
- Actions to take (e.g., retrain model, investigate outliers)