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

1.1 Data Types and Basic Operations

Every programming language has **different data types** that define the nature of stored values. The fundamental types in **Python and R** include:

Data Type	Python Example	R Example	Description
Integer (int)	`x = 10`	`x <- 10`	Whole numbers
Float (double in R)	`y = 3.14`	`y <- 3.14`	Decimal numbers
String (chr in R)	`name = "Alice"`	`name <- "Alice"`	Text data
Boolean (logical in R)	`is_valid = True`	`is_valid <- TRUE`	True or False
List (Vector in R)	`numbers = [1, 2, 3]`	`numbers <- c(1, 2, 3)`	Ordered collection of values
Dictionary (Named List in R)	`person = {"name": "Bob", "age": 25}`	`person <- list(name="Bob", age=25)`	Key-value storage

1.2 Variable Declaration

In both Python and R, variables do not need explicit declaration.

1.2.1 Python Code

```
x = 10              # Integer
y = 3.14            # Float
name = "Alice"      # String
is_valid = True     # Boolean
```

1.2.2 R Code

```r
x <- 10            # Integer
y <- 3.14          # Numeric
name <- "Alice"    # Character
is_valid <- TRUE   # Logical
```

1.3 Basic Data Manipulation

Operations on variables such as reassignment, updating values, and basic computations.

1.3.1 Python Code

```python
x = x + 5                       # Updating value
name = name + " Johnson"        # String concatenation
print(name)                     # Output: Alice Johnson
```

1.3.2 R Code

```r
x <- x + 5                      # Updating value
name <- paste(name, "Johnson")  # String concatenation
print(name)                     # Output: Alice Johnson
```

1.4 Basic Arithmetic Operations

Arithmetic operations are fundamental for numerical computations.

Operation	Python Example	R Example	Description
Addition (+)	`a + b`	`a + b`	Adds two numbers
Subtraction (-)	`a - b`	`a - b`	Subtracts one number from another
Multiplication (*)	`a * b`	`a * b`	Multiplies two numbers
Division (/)	`a / b`	`a / b`	Divides one number by another
Exponentiation (**) / (^ in R)	`a ** 2`	`a ^ 2`	Raises a number to a power
Modulo (%)	`a % b`	`a %% b`	Returns remainder of division

1.4.1 Python Code

```python
a = 10
b = 3
print(a + b)   # Addition
```

```
print(a - b)   # Subtraction
print(a * b)   # Multiplication
print(a / b)   # Division
print(a ** 2)  # Exponentiation
print(a % b)   # Modulo
```

1.4.2 R Code

```
a <- 10
b <- 3
print(a + b)   # Addition
print(a - b)   # Subtraction
print(a * b)   # Multiplication
print(a / b)   # Division
print(a ^ 2)   # Exponentiation
print(a %% b)  # Modulo
```

1.5 String Operations

String operations are used to manipulate text data.

Operation	Python Example	R Example	Description
Uppercase	`text.upper()`	`toupper(text)`	Convert to uppercase
Lowercase	`text.lower()`	`tolower(text)`	Convert to lowercase
Length	`len(text)`	`nchar(text)`	Get string length
Substring	`text[0:5]`	`substr(text, 1, 5)`	Extract part of string

1.5.1 Python Code

```
text = "Hello, World!"   # Variable declaration
print(text.upper())      # Convert to uppercase
print(text.lower())      # Convert to lowercase
print(len(text))         # Get string length
print(text[0:5])         # Slicing (first 5 characters)
```

1.5.2 R Code

```r
text <- "Hello, World!"   # variable declaration
print(toupper(text))      # Convert to uppercase
print(tolower(text))      # Convert to lowercase
print(nchar(text))        # Get string length
print(substr(text, 1, 5)) # Extract first 5 characters
```

1.6 Logical and Comparison Operators

Logical and comparison operators help in decision-making.

Operator	Description	Python Example	R Example
Equal (==)	Checks if values are equal	`x == y`	`x == y`
Not Equal (!=)	Checks if values are different	`x != y`	`x != y`
Greater (>)	Checks if left value is greater	`x > y`	`x > y`
Less (<)	Checks if left value is smaller	`x < y`	`x < y`
AND (and in Python, & in R)	Both conditions must be true	`(x > 5) and (y < 10)`	`(x > 5) & (y < 10)`
OR (or in Python, | in R)	At least one condition is true	`(x > 5) or (y > 10)`	`(x > 5) | (y > 10)`

1.6.1 Python Code

```python
x = 10
y = 5
print(x == y)                # False
print(x != y)                # True
print(x > y)                 # True
print(x < y)                 # False
print((x > 5) and (y < 10))  # True
print((x > 5) or (y > 10))   # True
```

1.6.2 R Code

```r
x <- 10
y <- 5
```

```
print(x == y)               # FALSE
print(x != y)               # TRUE
print(x > y)                # TRUE
print(x < y)                # FALSE
print((x > 5) & (y < 10))   # TRUE
print((x > 5) | (y > 10))   # TRUE
```

1.7 Data Type Conversion

Data type conversion allows changing one data type to another.

Conversion	Python Example	R Example	Description
String to Integer	`int("100")`	`as.numeric("100")`	Convert text to number
Integer to String	`str(100)`	`as.character(100)`	Convert number to text
Float to Integer	`int(10.5)`	`as.integer(10.5)`	Convert decimal to whole number

1.7.1 Python Code

```
num_str = "100"
num = int(num_str)              # Convert string to integer
print(num + 10)                 # Output: 110
```

```
num_float = 10.5
num_int = int(num_float)        # Convert float to integer
print(num_int)                  # Output: 10
```

1.7.2 R Code

```
num_str <- "100"
num <- as.numeric(num_str)      # Convert string to number
print(num + 10)                 # Output: 110

num_float <- 10.5
num_int <- as.integer(num_float) # Convert float to integer
print(num_int)                  # Output: 10
```

1.8 Practicum

1.8.1 Identifying Data Types

Determine the data types of the following variables in Python and R:

```
# R
 a = 42
 b = 3.14
 c = "Hello"
 d = FALSE
 e = [1, 2, 3]
 f = {"name": "Alice", "age": 25}
```

Questions:

1. Identify the data type of each variable above.
2. Print the data type of each variable using `type()` (Python) and `class()` (R).

1.8.2 Variables and Data Manipulation

Create the following variables in **Python** and **R**:

```
# R
 x = 20
 y = 5
 text1 = "Data"
 text2 = "Science"
```

Questions:

1. Update the value of `x` by adding `10`.
2. Concatenate `text1` and `text2` into `"Data Science"`.
3. Convert the concatenated text to uppercase.

1.8.3 Arithmetic Operations

Given the following variables:

```
# R
 a = 15
 b = 4
```

Questions:

1. Compute the sum, difference, product, division, and modulo of `a` and `b`.
2. Compute `a` raised to the power of `b`.
3. Create a new variable `c = a / b` and convert it to an **integer**.

1.8.4 String Operations

Given the following text:

```
# R
 text = "Hello, Data Science!"
```

Questions:

1. Extract the **first 5 characters** from the text.
2. Count the number of characters in the text.
3. Convert the text to lowercase.

1.8.5 Comparison and Logical Operators

Given the following variables:

```
# R
 x = 7
 y = 10
```

Questions:

1. Check if `x` is greater than `y`.
2. Check if `x` is less than or equal to `y`.
3. Check if `x` is **not equal to** `y`.
4. Evaluate the expression `(x > 5) AND (y < 20)`.

1.8.6 Data Type Conversion

Given the following variables:

```
# R
 num_str = "123"
 num_float = 45.67
```

Questions:

1. Convert `num_str` to an integer and add `10`.
2. Convert `num_float` to an integer.
3. Convert the converted `num_float` back to a string.

1.9 Bonus Challenge

Create an interactive program that asks the user to input:

1. Name
2. Age
3. Hometown

Then, print the output as follows:

```
"Hello [Name], you are [Age] years old and from [Hometown]."
```

Happy coding! If you need any help, feel free to ask **Chat-GPT**.

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:

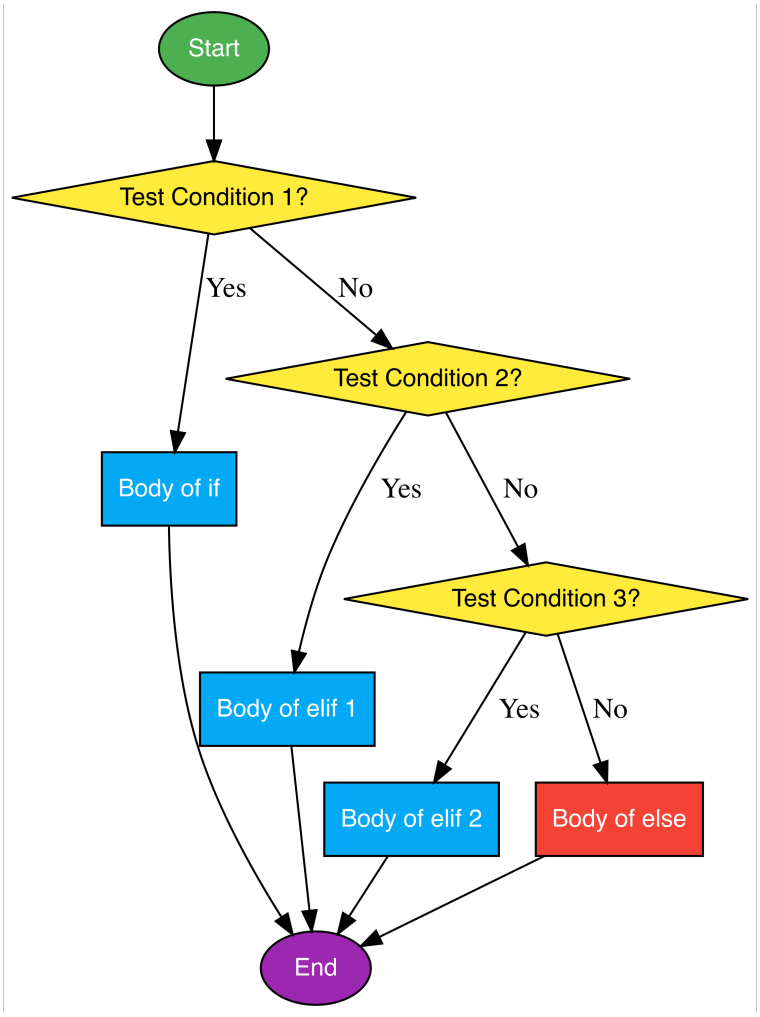
There are three main types of control flow statements:

- 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

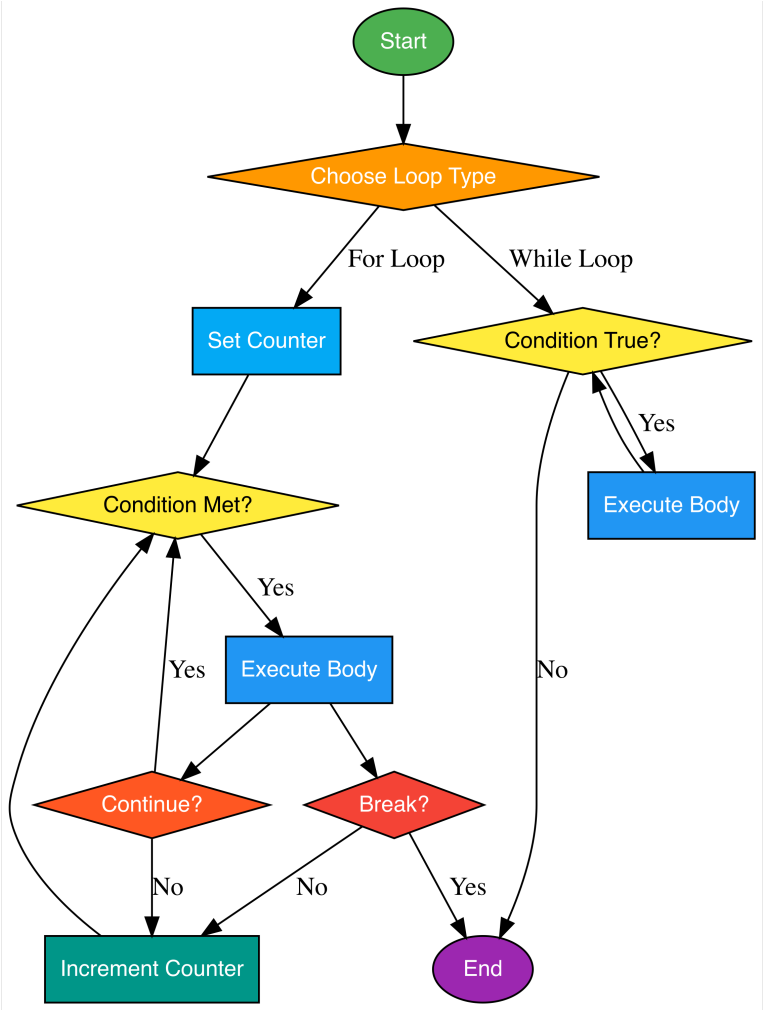
```r
age <- as.integer(readline("Enter your age: "))

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

```python
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

```r
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

```python
#  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))
```

```r
#  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

```python
#  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)
```

```python
#  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	83939	Staff	3920	32	5	Finance	Houston
1	55025	Staff	3123	28	1	HR	Phoenix
2	46158	Director	13755	56	24	Finance	New York
3	31642	Staff	4066	30	5	HR	New York
4	20805	Staff	4957	26	3	Operations	Los Angeles
5	35976	Supervisor	7646	37	9	HR	Phoenix
6	81021	Manager	10824	47	13	Operations	Los Angeles
7	60257	Manager	10450	41	13	HR	Houston
8	67114	Manager	8551	47	12	Finance	Phoenix
9	40801	Staff	3615	34	1	Sales	Houston
10	10850	Staff	4441	32	4	HR	Chicago
11	83955	Director	13855	40	17	Marketing	Phoenix
12	27905	Director	14975	46	23	HR	Phoenix
13	25507	Supervisor	7712	28	6	HR	Phoenix
14	89359	Director	14886	50	16	IT	Chicago
15	73803	Manager	9174	43	12	Sales	Los Angeles
16	40046	Director	13348	50	21	Marketing	Chicago
17	53086	Supervisor	5884	37	6	Marketing	New York
18	98668	Supervisor	6819	41	10	Marketing	Los Angeles
19	93462	Staff	3281	33	2	Marketing	Chicago

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      79262 Supervisor   7889  33          6  Marketing     Chicago
2      26128   Director  12955  47         23 Operations     Phoenix
3      57233 Supervisor   5226  33          7         IT     Phoenix
4      76745   Director  13605  47         23    Finance     Houston
5      86396      Staff   3033  35          4      Sales Los Angeles
6      87183   Director  12744  47         22      Sales     Chicago
7      20318    Manager   8257  40         12    Finance     Houston
8      96319 Supervisor   6861  33          8    Finance     Houston
9      49275      Staff   4377  35          2         HR     Chicago
10     78707   Director  12661  51         20         IT    New York
11     68434 Supervisor   7863  40         10      Sales Los Angeles
12     56545   Director  13020  47         18         IT     Houston
13     76727 Supervisor   5054  31          9         IT     Chicago
14     19813 Supervisor   5510  36          8         IT     Phoenix
15     92424    Manager  10251  36         12  Marketing     Phoenix
16     77115   Director  12742  48         19         IT     Houston
17     12683 Supervisor   5954  29          6         IT     Chicago
18     92411      Staff   4575  36          4    Finance     Phoenix
19     65660 Supervisor   7044  39          7 Operations     Phoenix
20     26632    Manager   9730  36         11      Sales     Houston
```

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	55968.9
median	54055.5
variance	7.13998e+08
std_dev	26720.7
min	10850
q1	34892.5
q3	81750.5
max	98668

Summary Statistics for 'Position'

Statistic	Value
count	20
unique	4
mode	Staff

Category	Frequency
Staff	7
Director	5
Supervisor	4
Manager	4

Summary Statistics for 'Salary'

Statistic	Value
count	20
mean	8264.1
median	7679
variance	1.74603e+07
std_dev	4178.55
min	3123
q1	4347.25
q3	11455
max	14975

Summary Statistics for 'Age'

Statistic	Value
count	20
mean	38.9
median	38.5
variance	73.2526
std_dev	8.55878
min	26
q1	32
q3	46.25
max	56

Summary Statistics for 'Experience'

Statistic	Value
count	20
mean	10.15
median	9.5
variance	52.1342
std_dev	7.2204
min	1
q1	4.75
q3	13.75
max	24

Summary Statistics for 'Department'

Statistic	Value
count	20
unique	6
mode	HR

```
| Category    |   Frequency |
|:------------|------------:|
| HR          |           7 |
| Marketing   |           5 |
| Finance     |           3 |
| Operations  |           2 |
| Sales       |           2 |
| IT          |           1 |

### Summary Statistics for 'Location'

| Statistic   | Value   |
|:------------|:--------|
| count       | 20      |
| unique      | 5       |
| mode        | Phoenix |


| Category    |   Frequency |
|:------------|------------:|
| Phoenix     |           6 |
| Los Angeles |           4 |
| Chicago     |           4 |
| Houston     |           3 |
| New York    |           3 |
```

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)
```

Pernyataan	Kategori	Nirgambar	Matematika Geometri	Pengantar Bidang Studi Desain Produk Industri	Desain Produk I	Teknologi Presentasi I	Bahan dan Proses Produksi	Semantika Produk II	Desain Produk III	Ergonomi Dasar I	Desain Produk II	Metodologi Desain I	Workshop Material	Pemodelan Digital I	Desain Produk V	Kolokium Kerja Profesi	Pra Tugas Akhir	Pengantar Pemasaran Produk	Manajemen Desain Lingkungan	Pengantar Manajemen Sarana Tugas Akhir
A	Reliability	4	4	4	4	3	3	3	4	4	4	4	4	4	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
C	Reliability	4	4	4	3	4	3	3	4	4	4	4	4	4	3	3	3	3	3	3
D	Reliability	4	4	4	4	4	3	3	4	4	4	4	4	4	3	3	3	3	3	3
E	Reliability	4	4	4	3	4	3	3	4	4	4	4	4	4	3	3	3	3	3	3
F	Reliability	4	4	4	3	3	3	3	4	4	4	4	4	4	3	3	3	3	3	3
G	Responsiveness	4	4	4	3	3	3	3	4	4	4	3	4	4	3	3	3	3	3	3
H	Assurance	4	4	4	3	3	3	3	4	4	4	3	4	4	3	4	3	3	3	4
I	Assurance	4	4	4	3	3	3	3	4	4	4	4	4	4	3	3	3	3	3	4
J	Empathy	4	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	4	4	3	4	4	3	3	3	3	3	3
L	Tangible	4	3	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

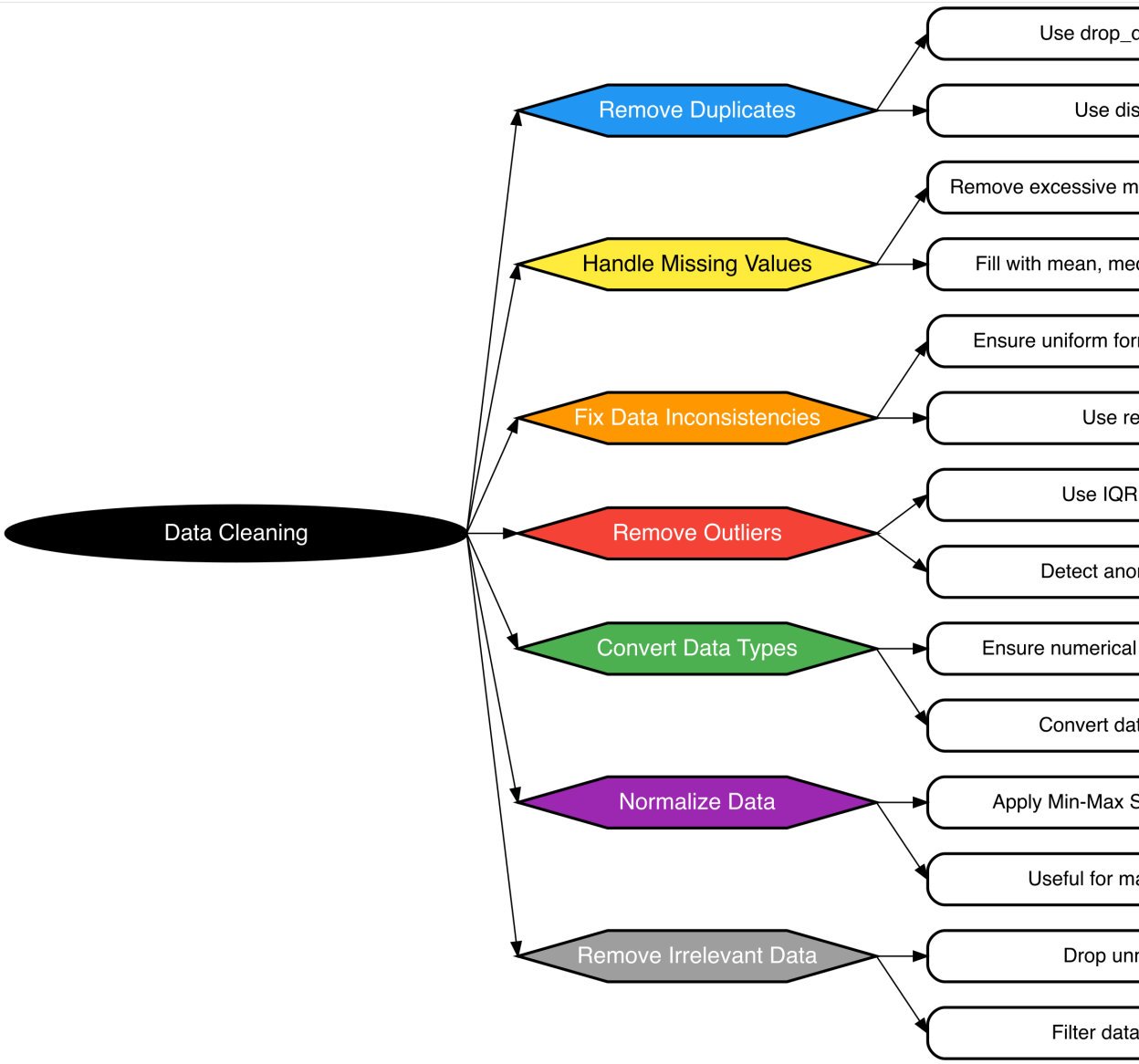
Data Cleaning

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 2022-05-01 15:22:04.191954  824.835708  ...    31.604697                  3.0
1 2022-05-02 15:22:04.191954  793.086785  ...    29.708164                  5.0
2 2022-05-03 15:22:04.191954  832.384427  ...    26.989298                  3.0
3 2022-05-04 15:22:04.191954  876.151493  ...    31.016866                  5.0
4 2022-05-05 15:22:04.191954  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))
```

```
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 2022-04-30  868.5479        102     244      60 14442.79         Positive
2 2022-05-01  771.7651        115     287      50 14765.80          neutral
3 2022-05-02  818.1564         99     208      77 14457.90         Positive
4 2022-05-03   50.0000         90     467     107 14033.63         NEGATIVE
5 2022-05-04  820.2134        130     305     148 14881.60         NEGATIVE
6 2022-05-05  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

```
# 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

```
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 19112  868.5479        102     244      60 14442.79         Positive
2 19113  771.7651        115     287      50 14765.80          neutral
3 19114  818.1564         99     208      77 14457.90         Positive
4 19115   50.0000         90     467     107 14033.63         NEGATIVE
5 19116  820.2134        130     305     148 14881.60         NEGATIVE
6 19117  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 19112  868.5479        102     244      60 14442.79         Positive
2 19113  771.7651        115     287      50 14765.80          neutral
3 19114  818.1564         99     208      77 14457.90         Positive
4 19115   50.0000         90     467     107 14033.63         NEGATIVE
5 19116  820.2134        130     305     148 14881.60         NEGATIVE
6 19117  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 19112  868.5479        102     244      60 14442.79         Positive
2 19113  771.7651        115     287      50 14765.80          neutral
3 19114  818.1564         99     208      77 14457.90         Positive
4 19115   50.0000         90     467     107 14033.63         NEGATIVE
5 19116  820.2134        130     305     148 14881.60         NEGATIVE
6 19117  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 2022-05-01 15:22:04.191954  ...                      True
1 2022-05-02 15:22:04.191954  ...                     False
2 2022-05-03 15:22:04.191954  ...                     False
3 2022-05-04 15:22:04.191954  ...                      True
4 2022-05-05 15:22:04.191954  ...                      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)
```

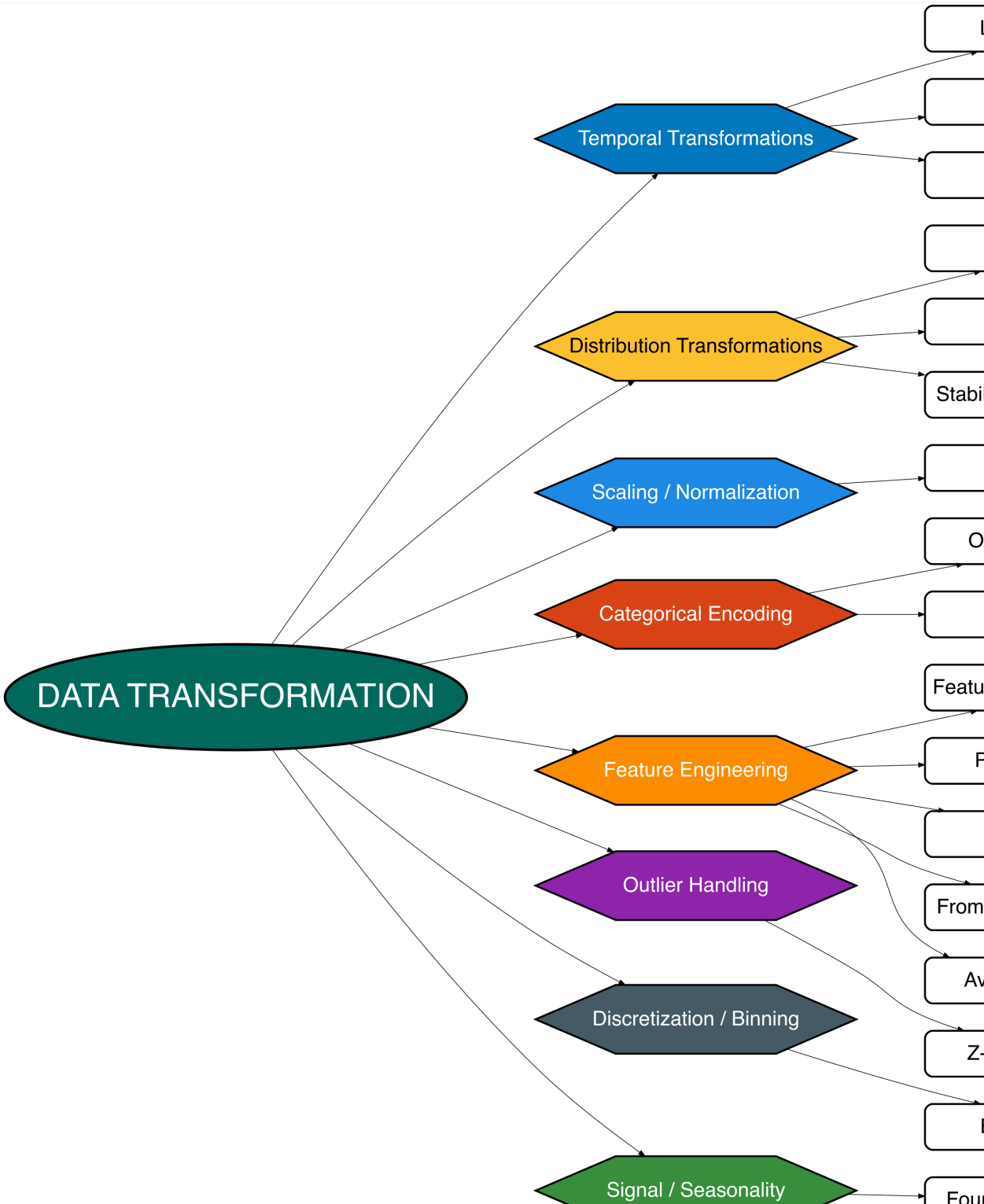
```
  Market_SentimentNEGATIVE Market_Sentimentneutral Market_SentimentPositive
1                        0                       0                        1
2                        0                       1                        0
3                        0                       0                        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 critical step in the data analysis process, aimed at converting raw data into more useful and meaningful information. This process involves a variety of techniques used to adjust the format of the data, improve its quality, and convert it into a form that is easier to analyze and understand. Data transformation also plays a key role in enhancing the accuracy and effectiveness of analytical models, especially in the context of handling large and complex datasets.

In the following mind map, we will explore several data transformation techniques that can be used to improve data quality and generate better insights. The mind map covers major categories such as temporal transformations, distribution transformations, normalization, categorical encoding, feature engineering, and more, each with important subcomponents that aid in the data processing workflow.


DATA TRANSFORMATION
Temporal Transformations
Distribution Transformations
Scaling / Normalization
Categorical Encoding
Feature Engineering
Outlier Handling
Discretization / Binning
Signal / Seasonality

Dummy Dataset

Below is the dummy dataset related to business that has been created using R (via Python for the demo). You can also generate it yourself in R using packages like `dplyr`, `lubridate`, and `stringi`. This dataset simulates transaction data from a retail business with key elements:

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

Copy CSV PDF Print Search:

	Transaction_ID	Transaction_Date	Customer_ID	Product_Category	Product_ID	Quantity	Unit_Price	Discount	Region	Sales_Channel	Total_Price
1	7zmPHxF7XfN9	2021-07-14	BAl3Y7yxev	Clothing	P0370	2	15.18	0	North	Online	30.36
2	y4bCY9pKTBWU	2020-11-16	TYY0h5C190	Electronics	P0185	5	10.22	0.15	West	Offline	43.44
3	8k0B7XX19Ykf	2023-03-22	nUX640AaXg	Home	P0443	3	17.74	0.05	West	Online	50.56
4	l8ahQz5YNOKz	2023-01-02	sBZyUSJLEP	Home	P0035	6	28.3	0.22	North	Offline	132.44
5	kmufgw8wx5qk	2023-06-05	GMfVH2ZWNX	Groceries	P0375	3	11.91	0.13	North	Offline	31.09
6	aI0KADT0mn7C	2023-03-15	YxqAmfTU9M	Clothing	P0447	1	5.43	0.07	North	Offline	5.05
7	zu0iP1OBFuNl	2021-09-25	ifYw95qmoL	Groceries	P0345	2	13.37	0.3	West	Online	18.72
8	l0Bz1iXOUzUy	2020-02-18	F5zL0GjDJh	Clothing	P0193	5	6.56	0.23	South	Online	25.26
9	kgU3mL1c8BdA	2023-02-25	Zm9ROmGygf	Clothing	P0114	4	18.23	0.27	South	Offline	53.23
10	MDp09wloF78Q	2023-08-19	fgldeSSYSL	Home	P0095	1	11.94	0.09	South	Online	10.87
11	M8Wk6MDt6qq1	2020-01-24	y5n7T2vCfd	Electronics	P0309	2	21.06	0.2	South	Offline	33.7
12	lH96iSOc84fn	2020-12-21	SS9WdTh6Gp	Books	P0402	4	9.38	0.01	North	Online	37.14
13	NnorQ5sHloaf	2024-06-12	9l5PBRrmgl	Clothing	P0135	3	15.61	0.17	South	Online	38.87
14	dEbUZoU5Sbu2	2020-06-13	A03t0cAGgC	Clothing	P0312	1	9.17	0.15	South	Offline	7.79
15	o5Gqu5Y2GBZ4	2021-09-13	7hcyxq0KSE	Groceries	P0498	1	16.31	0.24	West	Online	12.4
16	RYXFAjCauCpJ	2024-04-04	A9pnVHddZb	Electronics	P0098	2	18.49	0.09	East	Offline	33.65

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

```
# (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()
head(Tempral)
```

```
# A tibble: 6 x 14
  Transaction_ID Transaction_Date Customer_ID Product_Category Product_ID
  <chr>          <date>           <chr>       <chr>            <chr>
1 1PaWvGiAokHB   2023-02-09       02kNipUny9  Groceries        P0253
2 IYlgOMAGwsUw   2020-11-30       04npfK5VJa  Books            P0113
3 OkgKWZOpEp6Z   2020-03-18       0NwgTyo7P2  Clothing         P0445
4 ddFsMFlQOa2J   2023-05-19       0PlvCaxPuS  Electronics      P0298
5 VOCzzxDOxQps   2020-07-07       0S9qOEuCr9  Clothing         P0353
6 FyRCpwyOqKWK   2021-12-02       0dEDsy4fWM  Groceries        P0175
# i 9 more variables: Quantity <int>, Unit_Price <dbl>, Discount <dbl>,
#   Region <chr>, Sales_Channel <chr>, Total_Price <dbl>, Lag_Quantity <int>,
#   Diff_Quantity <int>, RollingMean_3 <lgl>
```

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 = .))
  )

# View the first few rows
head(Extract)
```

```
# A tibble: 6 x 24
  Transaction_ID Transaction_Date Customer_ID Product_Category Product_ID
  <chr>          <date>           <chr>       <fct>            <chr>
1 7zmPHxF7XfN9   2021-07-14       BAl3Y7yxev  Clothing         P0370
2 y4bCY9pKTBWU   2020-11-16       TYYOh5C190  Electronics      P0185
3 8k0B7XX19Ykf   2023-03-22       nUX640AaXg  Home             P0443
4 l8ahQz5YNOKz   2023-01-02       sBZyUSJLEP  Home             P0035
5 kmufgw8wx5qk   2023-06-05       GMfVH2ZWNX  Groceries        P0375
6 aI0KADT0mn7C   2023-03-15       YxqAmfTU9M  Clothing         P0447
# i 19 more variables: Quantity <int>, Unit_Price <dbl>, Discount <dbl>,
#   Region <fct>, Sales_Channel <chr>, Total_Price <dbl>, Day_of_Week <chr>,
#   Month <ord>, Year <dbl>, Is_Weekend <dbl>, RegionEast <dbl>,
#   RegionNorth <dbl>, RegionSouth <dbl>, RegionWest <dbl>,
#   Product_CategoryBooks <dbl>, Product_CategoryClothing <dbl>,
#   Product_CategoryElectronics <dbl>, Product_CategoryGroceries <dbl>,
#   Product_CategoryHome <dbl>
```

6.1.3 Cumulative Values

```
Tempral3 <- data_bisnis %>%
  group_by(Customer_ID) %>%
  mutate(
    Cumulative_Quantity = cumsum(Quantity),
    Cumulative_Spend = cumsum(Total_Price),
    Cumulative_AvgPrice = cummean(Unit_Price)
  ) %>%
  ungroup()
head(Tempral3)
```

```
# A tibble: 6 x 14
  Transaction_ID Transaction_Date Customer_ID Product_Category Product_ID
  <chr>          <date>           <chr>       <chr>            <chr>
1 7zmPHxF7XfN9   2021-07-14       BAl3Y7yxev  Clothing         P0370
2 y4bCY9pKTBWU   2020-11-16       TYYOh5C190  Electronics      P0185
3 8k0B7XX19Ykf   2023-03-22       nUX640AaXg  Home             P0443
4 l8ahQz5YNOKz   2023-01-02       sBZyUSJLEP  Home             P0035
5 kmufgw8wx5qk   2023-06-05       GMfVH2ZWNX  Groceries        P0375
6 aI0KADT0mn7C   2023-03-15       YxqAmfTU9M  Clothing         P0447
# i 9 more variables: Quantity <int>, Unit_Price <dbl>, Discount <dbl>,
#   Region <chr>, Sales_Channel <chr>, Total_Price <dbl>,
#   Cumulative_Quantity <int>, Cumulative_Spend <dbl>,
#   Cumulative_AvgPrice <dbl>
```

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)
  )
head(Log)
```

```
# A tibble: 6 x 13
  Transaction_ID Transaction_Date Customer_ID Product_Category Product_ID
  <chr>          <date>           <chr>       <chr>            <chr>
1 7zmPHxF7XfN9   2021-07-14       BAl3Y7yxev  Clothing         P0370
2 y4bCY9pKTBWU   2020-11-16       TYY0h5C190  Electronics      P0185
3 8k0B7XX19Ykf   2023-03-22       nUX640AaXg  Home             P0443
4 l8ahQz5YNOKz   2023-01-02       sBZyUSJLEP  Home             P0035
5 kmufgw8wx5qk   2023-06-05       GMfVH2ZWNX  Groceries        P0375
6 aI0KADT0mn7C   2023-03-15       YxqAmfTU9M  Clothing         P0447
# i 8 more variables: Quantity <int>, Unit_Price <dbl>, Discount <dbl>,
#   Region <chr>, Sales_Channel <chr>, Total_Price <dbl>,
#   Safe_Total_Price <dbl>, Log_Total_Price <dbl>
```

6.2.2 Box-Cox

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

```
# A tibble: 6 x 13
  Transaction_ID Transaction_Date Customer_ID Product_Category Product_ID
  <chr>          <date>           <chr>       <chr>            <chr>
1 7zmPHxF7XfN9   2021-07-14       BAl3Y7yxev  Clothing         P0370
```

```
2 y4bCY9pKTBWU   2020-11-16       TYY0h5C190  Electronics      P0185
3 8k0B7XX19Ykf   2023-03-22       nUX640AaXg  Home             P0443
4 l8ahQz5YNOKz   2023-01-02       sBZyUSJLEP  Home             P0035
5 kmufgw8wx5qk   2023-06-05       GMfVH2ZWNX  Groceries        P0375
6 aI0KADT0mn7C   2023-03-15       YxqAmfTU9M  Clothing         P0447
# i 8 more variables: Quantity <int>, Unit_Price <dbl>, Discount <dbl>,
#   Region <chr>, Sales_Channel <chr>, Total_Price <dbl>, YeoJ_Quantity <dbl>,
#   YeoJ_TotalPrice <dbl>
```

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)
  )
head(data_std)
```

```
# A tibble: 6 x 13
  Transaction_ID Transaction_Date Customer_ID Product_Category Product_ID
  <chr>          <date>           <chr>       <chr>            <chr>
1 7zmPHxF7XfN9   2021-07-14       BAl3Y7yxev  Clothing         P0370
2 y4bCY9pKTBWU   2020-11-16       TYY0h5C190  Electronics      P0185
3 8k0B7XX19Ykf   2023-03-22       nUX640AaXg  Home             P0443
4 l8ahQz5YNOKz   2023-01-02       sBZyUSJLEP  Home             P0035
5 kmufgw8wx5qk   2023-06-05       GMfVH2ZWNX  Groceries        P0375
6 aI0KADT0mn7C   2023-03-15       YxqAmfTU9M  Clothing         P0447
# i 8 more variables: Quantity <int>, Unit_Price <dbl>, Discount <dbl>,
#   Region <chr>, Sales_Channel <chr>, Total_Price <dbl>,
#   Quantity_Std <dbl[,1]>, Unit_Price_Std <dbl[,1]>
```

```r
# Min-Max Normalization
data_norm <- data_bisnis %>%
  mutate(
```

```
    Quantity_Norm = (Quantity - min(Quantity)) / (max(Quantity) - min(Quantity)),
    Total_Price_Norm = (Total_Price - min(Total_Price)) / (max(Total_Price) - min(Total_Pr
  )
head(data_norm)
```

```
# A tibble: 6 x 13
  Transaction_ID Transaction_Date Customer_ID Product_Category Product_ID
  <chr>          <date>           <chr>       <chr>            <chr>
1 7zmPHxF7XfN9   2021-07-14       BAl3Y7yxev  Clothing         P0370
2 y4bCY9pKTBWU   2020-11-16       TYY0h5C190  Electronics      P0185
3 8k0B7XX19Ykf   2023-03-22       nUX640AaXg  Home             P0443
4 l8ahQz5YNOKz   2023-01-02       sBZyUSJLEP  Home             P0035
5 kmufgw8wx5qk   2023-06-05       GMfVH2ZWNX  Groceries        P0375
6 aI0KADT0mn7C   2023-03-15       YxqAmfTU9M  Clothing         P0447
# i 8 more variables: Quantity <int>, Unit_Price <dbl>, Discount <dbl>,
#   Region <chr>, Sales_Channel <chr>, Total_Price <dbl>, Quantity_Norm <dbl>,
#   Total_Price_Norm <dbl>
```

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

```
library(fastDummies)
one_hot <- dummy_cols(data_bisnis,
                          select_columns = c("Region",
                                             "Sales_Channel",
                                             "Product_Category"),
                          remove_first_dummy = TRUE,
                         remove_selected_columns = TRUE)
head(one_hot)
```

```
# A tibble: 6 x 16
  Transaction_ID Transaction_Date Customer_ID Product_ID Quantity Unit_Price
```

```
  <chr>          <date>           <chr>       <chr>         <int>      <dbl>
1 7zmPHxF7XfN9   2021-07-14       BAl3Y7yxev  P0370             2      15.2
2 y4bCY9pKTBWU   2020-11-16       TYYOh5C190  P0185             5      10.2
3 8k0B7XX19Ykf   2023-03-22       nUX640AaXg  P0443             3      17.7
4 l8ahQz5YNOKz   2023-01-02       sBZyUSJLEP  P0035             6      28.3
5 kmufgw8wx5qk   2023-06-05       GMfVH2ZWNX  P0375             3      11.9
6 aI0KADT0mn7C   2023-03-15       YxqAmfTU9M  P0447             1       5.43
# i 10 more variables: Discount <dbl>, Total_Price <dbl>, Region_North <int>,
#   Region_South <int>, Region_West <int>, Sales_Channel_Online <int>,
#   Product_Category_Clothing <int>, Product_Category_Electronics <int>,
#   Product_Category_Groceries <int>, Product_Category_Home <int>
```

6.4.2 Frequency Encoding

```r
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)
  )
head(Frequency)
```

```
# A tibble: 6 x 13
  Transaction_ID Transaction_Date Customer_ID Product_Category Product_ID
  <chr>          <date>           <chr>       <chr>            <chr>
1 7zmPHxF7XfN9   2021-07-14       BAl3Y7yxev  Clothing         P0370
2 y4bCY9pKTBWU   2020-11-16       TYYOh5C190  Electronics      P0185
3 8k0B7XX19Ykf   2023-03-22       nUX640AaXg  Home             P0443
4 l8ahQz5YNOKz   2023-01-02       sBZyUSJLEP  Home             P0035
5 kmufgw8wx5qk   2023-06-05       GMfVH2ZWNX  Groceries        P0375
6 aI0KADT0mn7C   2023-03-15       YxqAmfTU9M  Clothing         P0447
# i 8 more variables: Quantity <int>, Unit_Price <dbl>, Discount <dbl>,
#   Region <chr>, Sales_Channel <chr>, Total_Price <dbl>, Region_freq <dbl>,
#   Product_Category_freq <dbl>
```

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.

Key strategies include:

- **Mathematical combinations:** Creating interaction terms (e.g., price × quantity) or ratios (e.g., discount / price).

- **Domain-driven insights:** Building features based on domain knowledge (e.g., efficiency, customer value).
- **Statistical summaries:** Aggregating features by group (e.g., average purchase per customer).
- **Parsing IDs or Text:** Extracting patterns or categories from strings.

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.

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

Feature_Eng <- data_bisnis %>%
  mutate(
    Price_per_Unit = Total_Price / Quantity,
    ID_Prefix = substr(Product_ID, 1, 2),
    ID_Length = nchar(Product_ID),
    Discount_Level = ntile(Discount, 4)
  ) %>%
  group_by(Region) %>%
  mutate(
    Avg_Quantity_Region = mean(Quantity),
    Sum_Sales_Region = sum(Total_Price)
  ) %>%
  ungroup()
head(Feature_Eng)
```

```
# A tibble: 6 x 17
  Transaction_ID Transaction_Date Customer_ID Product_Category Product_ID
  <chr>          <date>           <chr>       <chr>            <chr>
1 7zmPHxF7XfN9   2021-07-14       BAl3Y7yxev  Clothing         P0370
2 y4bCY9pKTBWU   2020-11-16       TYY0h5C190  Electronics      P0185
3 8k0B7XX19Ykf   2023-03-22       nUX640AaXg  Home             P0443
4 l8ahQz5YNOKz   2023-01-02       sBZyUSJLEP  Home             P0035
5 kmufgw8wx5qk   2023-06-05       GMfVH2ZWNX  Groceries        P0375
6 aI0KADT0mn7C   2023-03-15       YxqAmfTU9M  Clothing         P0447
# i 12 more variables: Quantity <int>, Unit_Price <dbl>, Discount <dbl>,
#   Region <chr>, Sales_Channel <chr>, Total_Price <dbl>, Price_per_Unit <dbl>,
#   ID_Prefix <chr>, ID_Length <int>, Discount_Level <int>,
#   Avg_Quantity_Region <dbl>, Sum_Sales_Region <dbl>
```

6.5.1 Interaction Features

```
# Interaction between Quantity and Unit_Price
data_interaction <- data_bisnis %>%
  mutate(
```

```
    Price_Impact = Quantity * Unit_Price
  )
head(data_interaction)
```

```
# A tibble: 6 x 12
  Transaction_ID Transaction_Date Customer_ID Product_Category Product_ID
  <chr>          <date>           <chr>       <chr>            <chr>
1 7zmPHxF7XfN9   2021-07-14       BAl3Y7yxev  Clothing         P0370
2 y4bCY9pKTBWU   2020-11-16       TYYOh5C190  Electronics      P0185
3 8k0B7XX19Ykf   2023-03-22       nUX640AaXg  Home             P0443
4 l8ahQz5YNOKz   2023-01-02       sBZyUSJLEP  Home             P0035
5 kmufgw8wx5qk   2023-06-05       GMfVH2ZWNX  Groceries        P0375
6 aI0KADT0mn7C   2023-03-15       YxqAmfTU9M  Clothing         P0447
# i 7 more variables: Quantity <int>, Unit_Price <dbl>, Discount <dbl>,
#   Region <chr>, Sales_Channel <chr>, Total_Price <dbl>, Price_Impact <dbl>
```

6.5.2 Ratio Features

```
# Discount to Price Ratio
data_ratio <- data_bisnis %>%
  mutate(
    Discount_Ratio = Discount / (Unit_Price + 1e-5)  # Avoid division by zero
  )
head(data_ratio)
```

```
# A tibble: 6 x 12
  Transaction_ID Transaction_Date Customer_ID Product_Category Product_ID
  <chr>          <date>           <chr>       <chr>            <chr>
1 7zmPHxF7XfN9   2021-07-14       BAl3Y7yxev  Clothing         P0370
2 y4bCY9pKTBWU   2020-11-16       TYYOh5C190  Electronics      P0185
3 8k0B7XX19Ykf   2023-03-22       nUX640AaXg  Home             P0443
4 l8ahQz5YNOKz   2023-01-02       sBZyUSJLEP  Home             P0035
5 kmufgw8wx5qk   2023-06-05       GMfVH2ZWNX  Groceries        P0375
6 aI0KADT0mn7C   2023-03-15       YxqAmfTU9M  Clothing         P0447
# i 7 more variables: Quantity <int>, Unit_Price <dbl>, Discount <dbl>,
#   Region <chr>, Sales_Channel <chr>, Total_Price <dbl>, Discount_Ratio <dbl>
```

6.5.3 Group Aggregation

```
# Total sales by each Customer
customer_sales <- data_bisnis %>%
  group_by(Customer_ID) %>%
  summarise(
    Avg_Spent = mean(Total_Price),
    Max_Spent = max(Total_Price),
    Transaction_Count = n()
  )

# Join with main data
```

```
data_bisnis <- left_join(data_bisnis, customer_sales, by = "Customer_ID")
head(data_bisnis)
```

```
# A tibble: 6 x 14
  Transaction_ID Transaction_Date Customer_ID Product_Category Product_ID
  <chr>          <date>           <chr>       <chr>            <chr>
1 7zmPHxF7XfN9   2021-07-14       BAl3Y7yxev  Clothing         P0370
2 y4bCY9pKTBWU   2020-11-16       TYY0h5C190  Electronics      P0185
3 8k0B7XX19Ykf   2023-03-22       nUX640AaXg  Home             P0443
4 l8ahQz5YNOKz   2023-01-02       sBZyUSJLEP  Home             P0035
5 kmufgw8wx5qk   2023-06-05       GMfVH2ZWNX  Groceries        P0375
6 aI0KADT0mn7C   2023-03-15       YxqAmfTU9M  Clothing         P0447
# i 9 more variables: Quantity <int>, Unit_Price <dbl>, Discount <dbl>,
#   Region <chr>, Sales_Channel <chr>, Total_Price <dbl>, Avg_Spent <dbl>,
#   Max_Spent <dbl>, Transaction_Count <int>
```

6.5.4 Rank Transformation

```
data_ranked <- data_bisnis %>%
  mutate(
    Sales_Rank = rank(-Total_Price) # Higher total price gets lower rank number
  )
head(data_ranked)
```

```
# A tibble: 6 x 15
  Transaction_ID Transaction_Date Customer_ID Product_Category Product_ID
  <chr>          <date>           <chr>       <chr>            <chr>
1 7zmPHxF7XfN9   2021-07-14       BAl3Y7yxev  Clothing         P0370
2 y4bCY9pKTBWU   2020-11-16       TYY0h5C190  Electronics      P0185
3 8k0B7XX19Ykf   2023-03-22       nUX640AaXg  Home             P0443
4 l8ahQz5YNOKz   2023-01-02       sBZyUSJLEP  Home             P0035
5 kmufgw8wx5qk   2023-06-05       GMfVH2ZWNX  Groceries        P0375
6 aI0KADT0mn7C   2023-03-15       YxqAmfTU9M  Clothing         P0447
# i 10 more variables: Quantity <int>, Unit_Price <dbl>, Discount <dbl>,
#   Region <chr>, Sales_Channel <chr>, Total_Price <dbl>, Avg_Spent <dbl>,
#   Max_Spent <dbl>, Transaction_Count <int>, Sales_Rank <dbl>
```

6.5.5 Text Cleaning & Feature Creation

```
# Create numeric suffix from Product_ID
data_text <- data_bisnis %>%
  mutate(
    Product_Num = as.numeric(gsub("P", "", Product_ID))
  )
head(data_text)
```

```
# A tibble: 6 x 15
  Transaction_ID Transaction_Date Customer_ID Product_Category Product_ID
  <chr>          <date>           <chr>       <chr>            <chr>
```

```
1 7zmPHxF7XfN9   2021-07-14       BAl3Y7yxev  Clothing         P0370
2 y4bCY9pKTBWU   2020-11-16       TYY0h5C190  Electronics      P0185
3 8k0B7XX19Ykf   2023-03-22       nUX640AaXg  Home             P0443
4 l8ahQz5YNOKz   2023-01-02       sBZyUSJLEP  Home             P0035
5 kmufgw8wx5qk   2023-06-05       GMfVH2ZWNX  Groceries        P0375
6 aI0KADT0mn7C   2023-03-15       YxqAmfTU9M  Clothing         P0447
# i 10 more variables: Quantity <int>, Unit_Price <dbl>, Discount <dbl>,
#   Region <chr>, Sales_Channel <chr>, Total_Price <dbl>, Avg_Spent <dbl>,
#   Max_Spent <dbl>, Transaction_Count <int>, Product_Num <dbl>
```

6.5.6 Cumulative Features

```
data_cumulative <- data_bisnis %>%
  arrange(Customer_ID, Transaction_Date) %>%
  group_by(Customer_ID) %>%
  mutate(
    Cumulative_Sales = cumsum(Total_Price)
  )
head(data_cumulative)
```

```
# A tibble: 6 x 15
# Groups:   Customer_ID [6]
  Transaction_ID Transaction_Date Customer_ID Product_Category Product_ID
  <chr>          <date>           <chr>       <chr>            <chr>
1 1PaWvGiAokHB   2023-02-09       02kNipUny9  Groceries        P0253
2 IYlg0MAGwsUw   2020-11-30       04npfK5VJa  Books            P0113
3 0kgKWZOpEp6Z   2020-03-18       0NwgTyo7P2  Clothing         P0445
4 ddFsMFlQOa2J   2023-05-19       0PlvCaxPuS  Electronics      P0298
5 VOCzzxDOxQps   2020-07-07       0S9qOEuCr9  Clothing         P0353
6 FyRCpwyOqKWK   2021-12-02       0dEDsy4fWM  Groceries        P0175
# i 10 more variables: Quantity <int>, Unit_Price <dbl>, Discount <dbl>,
#   Region <chr>, Sales_Channel <chr>, Total_Price <dbl>, Avg_Spent <dbl>,
#   Max_Spent <dbl>, Transaction_Count <int>, Cumulative_Sales <dbl>
```

6.6 Outlier Handeling

Outlier handling refers to the identification and treatment of data points that significantly deviate from other observations. Outliers can distort statistical analyses and model predictions if not managed properly.

Common techniques include:

- **Z-Score:** Identifies how many standard deviations a point is from the mean.
- **Interquartile Range (IQR):** Flags values that fall far outside the middle 50% of data.
- **Winsorizing:** Limits extreme values by capping them at a certain percentile.

Deciding whether to keep, transform, or remove outliers depends on their cause and the goals of the analysis. Proper handling ensures that models are not overly influenced by anomalous data, leading to more stable and accurate outcomes.

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

# IQR method for detecting and removing outliers
Q1 <- quantile(data_bisnis$Total_Price, 0.25)
Q3 <- quantile(data_bisnis$Total_Price, 0.75)
IQR_val <- Q3 - Q1
data_outliers <- data_bisnis %>%
  filter(Total_Price > (Q1 - 1.5 * IQR_val) & Total_Price < (Q3 + 1.5 * IQR_val))

head(data_outliers)
```

```
# A tibble: 6 x 14
  Transaction_ID Transaction_Date Customer_ID Product_Category Product_ID
  <chr>          <date>           <chr>       <chr>            <chr>
1 7zmPHxF7XfN9   2021-07-14       BAl3Y7yxev  Clothing         P0370
2 y4bCY9pKTBWU   2020-11-16       TYY0h5C190  Electronics      P0185
3 8k0B7XX19Ykf   2023-03-22       nUX640AaXg  Home             P0443
4 kmufgw8wx5qk   2023-06-05       GMfVH2ZWNX  Groceries        P0375
5 aI0KADT0mn7C   2023-03-15       YxqAmfTU9M  Clothing         P0447
6 zu0iP1OBFuNI   2021-09-25       ifYw95qmoL  Groceries        P0345
# i 9 more variables: Quantity <int>, Unit_Price <dbl>, Discount <dbl>,
#   Region <chr>, Sales_Channel <chr>, Total_Price <dbl>, Avg_Spent <dbl>,
#   Max_Spent <dbl>, Transaction_Count <int>
```

6.7 Discretization

Discretization, also known as binning, is the process of converting continuous numerical variables into categorical intervals or "bins." This technique simplifies data, improves model performance (especially for tree-based models), and enhances interpretability.

Common approaches include:

- **Equal-width binning:** Divides the range of values into intervals of equal size.
- **Equal-frequency binning:** Ensures each bin contains roughly the same number of observations.
- **Custom bins:** Created based on domain knowledge or statistical thresholds (e.g., age groups, income brackets).

Binning is especially useful when the exact values of a variable are less important than the range they fall into. It can also help deal with non-linearity and reduce the impact of outliers.

```r
binning <- data_bisnis %>%
  mutate(
    Price_Level = cut(Unit_Price,
                      breaks = quantile(Unit_Price, probs = c(0, 0.33, 0.66, 1), na.rm = T
                      labels = c("Low", "Medium", "High"),
                      include.lowest = TRUE)
```

```
  )
head(binning)
```

```
# A tibble: 6 x 15
  Transaction_ID Transaction_Date Customer_ID Product_Category Product_ID
  <chr>          <date>           <chr>       <chr>            <chr>
1 7zmPHxF7XfN9   2021-07-14       BAl3Y7yxev  Clothing         P0370
2 y4bCY9pKTBWU   2020-11-16       TYYOh5C190  Electronics      P0185
3 8k0B7XX19Ykf   2023-03-22       nUX640AaXg  Home             P0443
4 l8ahQz5YNOKz   2023-01-02       sBZyUSJLEP  Home             P0035
5 kmufgw8wx5qk   2023-06-05       GMfVH2ZWNX  Groceries        P0375
6 aI0KADT0mn7C   2023-03-15       YxqAmfTU9M  Clothing         P0447
# i 10 more variables: Quantity <int>, Unit_Price <dbl>, Discount <dbl>,
#   Region <chr>, Sales_Channel <chr>, Total_Price <dbl>, Avg_Spent <dbl>,
#   Max_Spent <dbl>, Transaction_Count <int>, Price_Level <fct>
```

6.8 Seasonality

Seasonality and signal decomposition involve identifying recurring patterns or cyclical behaviors in time-series data. These patterns can significantly enhance predictive models, especially in fields like sales forecasting, weather prediction, or traffic analysis.

One powerful tool for detecting seasonality is the Fourier Transform, which converts time-based signals into frequency components, revealing hidden periodic trends.

Applications include:

- Extracting weekly, monthly, or yearly patterns
- Smoothing or denoising time series
- Creating seasonal features for machine learning models

By isolating and leveraging these seasonal signals, analysts can improve forecast accuracy and understand underlying business cycles.

```
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)
  )

head(Seasonality)
```

```
# A tibble: 6 x 19
  Transaction_ID Transaction_Date Customer_ID Product_Category Product_ID
  <chr>          <date>           <chr>       <chr>            <chr>
1 7zmPHxF7XfN9   2021-07-14       BAl3Y7yxev  Clothing         P0370
2 y4bCY9pKTBWU   2020-11-16       TYYOh5C190  Electronics      P0185
3 8k0B7XX19Ykf   2023-03-22       nUX640AaXg  Home             P0443
```

```
4 l8ahQz5YNOKz   2023-01-02       sBZyUSJLEP  Home             P0035
5 kmufgw8wx5qk   2023-06-05       GMfVH2ZWNX  Groceries        P0375
6 aI0KADT0mn7C   2023-03-15       YxqAmfTU9M  Clothing         P0447
# i 14 more variables: Quantity <int>, Unit_Price <dbl>, Discount <dbl>,
#   Region <chr>, Sales_Channel <chr>, Total_Price <dbl>, Avg_Spent <dbl>,
#   Max_Spent <dbl>, Transaction_Count <int>, Year <dbl>, DayOfYear <dbl>,
#   DaysInYear <dbl>, sin_year <dbl>, cos_year <dbl>
```

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

```r
# Ensure all required packages are installed
packages <- c("dplyr", "stringi", "lubridate", "DT")
new_packages <- packages[!(packages %in% installed.packages()[, "Package"])]
if(length(new_packages)) install.packages(new_packages)

# Load libraries
library(dplyr)
library(stringi)
library(lubridate)
library(DT)

# Create a complex dummy weather dataset for data transformation
set.seed(42)
n <- 500

dates <- seq.Date(from = as.Date("2020-01-01"), to = as.Date("2024-12-31"), by = "day")
sample_dates <- sample(dates, n, replace = TRUE)

month <- month(sample_dates)
season <- case_when(
  month %in% c(11, 12, 1, 2) ~ "Rainy Season",
  month %in% c(6, 7, 8, 9) ~ "Dry Season",
  TRUE ~ "Transitional Season"
)

humidity <- round(runif(n, 60, 100), 1)
temperature <- round(rnorm(n, mean = 27, sd = 3), 1)
rainfall <- round(rgamma(n, shape = 2, rate = 0.2), 1)
```

```
wind_speed <- round(runif(n, 1, 15), 1)

weather_data <- tibble(
  Observation_ID = stri_rand_strings(n, 12),
  Date = sample_dates,
  Location = sample(c("Jakarta", "Bandung", "Surabaya", "Medan", "Makassar"), n, replace = TR
  Season = season,
  Temperature = temperature,
  Humidity = humidity,
  Rainfall = rainfall,
  Wind_Speed = wind_speed
)

# Show interactive table with download buttons
datatable(
  weather_data,
  extensions = 'Buttons',
  options = list(
    dom = 'Bfrtip',
    buttons = c('copy', 'csv', 'excel', 'pdf', 'print'),
    scrollY = "400px",
    scrollCollapse = TRUE,
    paging = FALSE
  ),
  caption = htmltools::tags$caption(
    style = 'caption-side: top; text-align: left;
             font-size: 18px; font-weight: bold;'
  ),
  class = 'stripe hover compact'
)
```

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	qpTzXirxU7F8	2021-02-13	Makassar	Rainy Season	31.1	86.9	10.8	12.1

Showing 1 to 500 of 500 entries

6.9.2 General Health

```
# Ensure all required packages are installed
packages <- c("dplyr", "stringi", "lubridate", "DT")
new_packages <- packages[!(packages %in% installed.packages()[, "Package"])]
if(length(new_packages)) install.packages(new_packages)
```

```r
# Load libraries
library(dplyr)
library(stringi)
library(lubridate)
library(DT)

# Create a complex dummy health dataset for data transformation
set.seed(42)
n <- 500

dates <- seq.Date(from = as.Date("2020-01-01"), to = as.Date("2024-12-31"), by = "day")
sample_dates <- sample(dates, n, replace = TRUE)

# Simulate health parameters
age <- sample(18:80, n, replace = TRUE)
bmi <- round(rnorm(n, mean = 25, sd = 4), 1)  # BMI (Body Mass Index)
blood_pressure <- round(rnorm(n, mean = 120, sd = 15), 1)  # Systolic BP
cholesterol <- round(rnorm(n, mean = 200, sd = 40), 1)  # Cholesterol (mg/dL)
glucose <- round(rnorm(n, mean = 90, sd = 20), 1)  # Glucose (mg/dL)
heart_rate <- round(rnorm(n, mean = 75, sd = 10), 1)  # Heart Rate (bpm)

# Simulate location and health condition
location <- sample(c("Jakarta", "Bandung", "Surabaya", "Medan", "Makassar"), n, replace =
health_condition <- sample(c("Healthy", "Hypertension", "Diabetes", "Obesity", "Cardiovasc

# Simulate seasonal impact on health
season <- case_when(
  month(sample_dates) %in% c(11, 12, 1, 2) ~ "Rainy Season",
  month(sample_dates) %in% c(6, 7, 8, 9) ~ "Dry Season",
  TRUE ~ "Transitional Season"
)

# Create the health dataset
health_data <- tibble(
  Patient_ID = stri_rand_strings(n, 12),
  Date = sample_dates,
  Age = age,
  BMI = bmi,
  Blood_Pressure = blood_pressure,
  Cholesterol = cholesterol,
  Glucose = glucose,
  Heart_Rate = heart_rate,
  Location = location,
  Health_Condition = health_condition,
  Season = season
)

# Show interactive table with download buttons
```

```
datatable(
  health_data,
  extensions = 'Buttons',
  options = list(
    dom = 'Bfrtip',
    buttons = c('copy', 'csv', 'excel', 'pdf', 'print'),
    scrollY = "400px",
    scrollCollapse = TRUE,
    paging = FALSE
  ),
  caption = htmltools::tags$caption(
    style = 'caption-side: top; text-align: left;
             font-size: 18px; font-weight: bold;'
  ),
  class = 'stripe hover compact'
)
```

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	5PBRrmglA03t	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

6.9.3 Financial Market

```
# Ensure all required packages are installed
packages <- c("dplyr", "stringi", "lubridate", "DT")
new_packages <- packages[!(packages %in% installed.packages()[, "Package"])]
if(length(new_packages)) install.packages(new_packages)

# Load libraries
library(dplyr)
library(stringi)
library(lubridate)
library(DT)

# Create a complex dummy financial market dataset for data transformation
set.seed(42)
n <- 500

# Dates from 2020-01-01 to 2024-12-31
dates <- seq.Date(from = as.Date("2020-01-01"), to = as.Date("2024-12-31"), by = "day")
sample_dates <- sample(dates, n, replace = TRUE)
```

```
# Simulate financial parameters
stock_price <- round(runif(n, 100, 1500), 2)  # Stock price (in USD)
volume_traded <- sample(1000:1000000, n, replace = TRUE)  # Trading volume
market_cap <- round(stock_price * volume_traded, 2)  # Market cap (in USD)
pe_ratio <- round(rnorm(n, mean = 15, sd = 5), 2)  # Price-to-Earnings ratio
dividend_yield <- round(runif(n, 0, 10), 2)  # Dividend Yield (%)
return_on_equity <- round(rnorm(n, mean = 12, sd = 3), 2)  # Return on Equity (%)

# Simulate stock market sector and performance
sector <- sample(c("Technology", "Finance", "Healthcare", "Energy", "Consumer Goods"), n,
performance <- sample(c("Positive", "Negative", "Stable"), n, replace = TRUE)

# Create the financial market dataset
financial_data <- tibble(
  Stock_ID = stri_rand_strings(n, 12),
  Date = sample_dates,
  Stock_Price = stock_price,
  Volume_Traded = volume_traded,
  Market_Cap = market_cap,
  PE_Ratio = pe_ratio,
  Dividend_Yield = dividend_yield,
  Return_on_Equity = return_on_equity,
  Sector = sector,
  Performance = performance
)

# Show interactive table with download buttons
datatable(
  financial_data,
  extensions = 'Buttons',
  options = list(
    dom = 'Bfrtip',
    buttons = c('copy', 'csv', 'excel', 'pdf', 'print'),
    scrollY = "400px",
    scrollCollapse = TRUE,
    paging = FALSE
  ),
  caption = htmltools::tags$caption(
    style = 'caption-side: top; text-align: left;
             font-size: 18px; font-weight: bold;'
  ),
  class = 'stripe hover compact'
)
```

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	XgdcIQcFxt8V	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

Part III

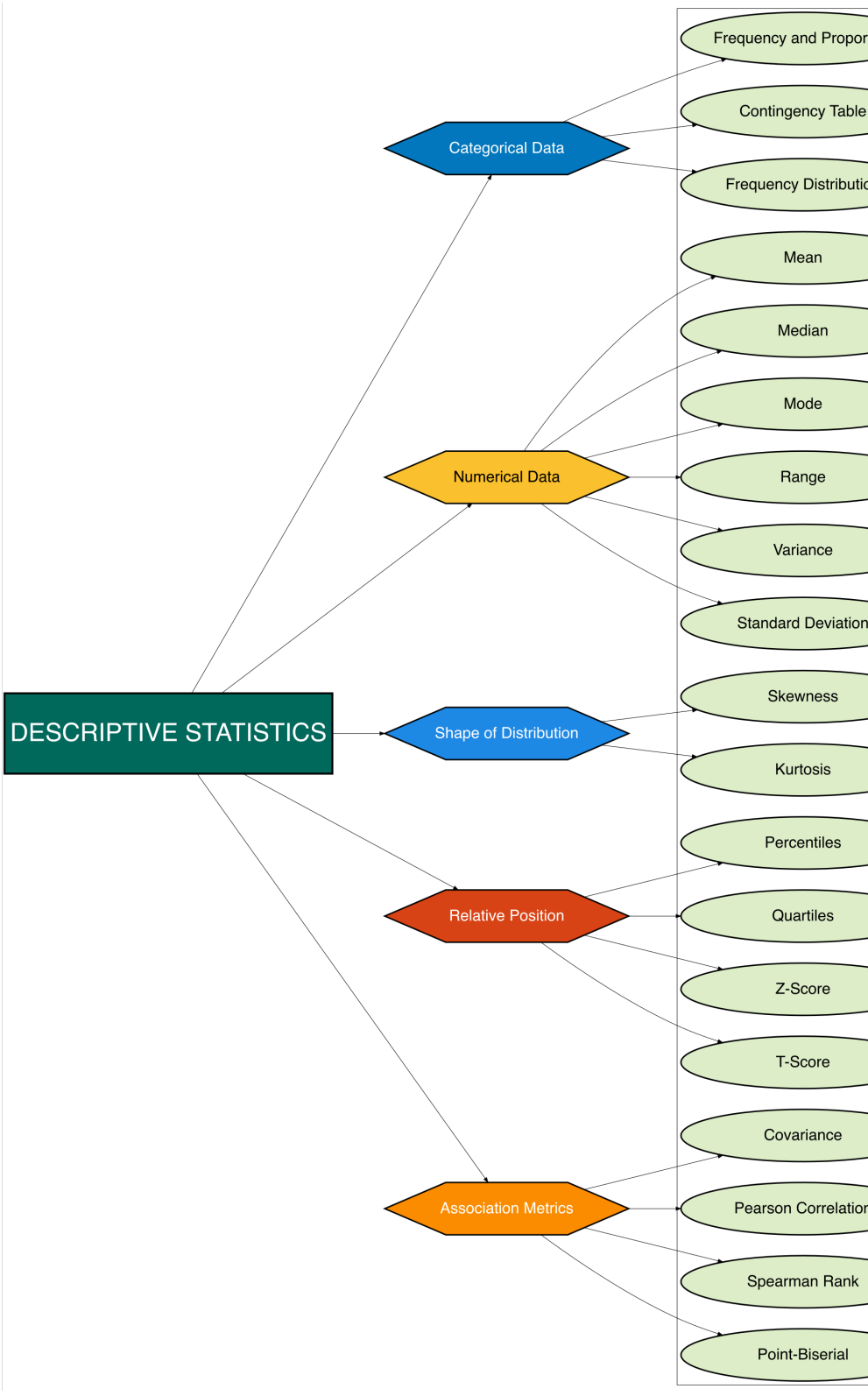
EDA

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 of Descriptive Statistics 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. The mind map is designed to provide a clearer understanding and practical application of descriptive statistical techniques in data analysis.


DESCRIPTIVE STATISTICS
Categorical Data
Numerical Data
Shape of Distribution
Relative Position
Association Metrics
Frequency and Propor
Contingency Table
Frequency Distributio
Mean
Median
Mode
Range
Variance
Standard Deviation
Skewness
Kurtosis
Percentiles
Quartiles
Z-Score
T-Score
Covariance
Pearson Correlation
Spearman Rank
Point-Biserial

In data analysis, there is often a distinction between categorical and numerical data, each requiring different descriptive methods. Descriptive statistics for categorical data involves calculating frequency and proportion, using contingency tables, and frequency distributions to summarize qualitative data. On the other hand, descriptive statistics for numerical data focuses on computing measures like the mean, median, mode, range, variance, and standard deviation to provide a numerical summary of the data distribution.

It is crucial to understand the shape of the distribution of data through metrics such as skewness and kurtosis, which offer deeper insights into the data distribution and the potential presence of outliers. Additionally, relative position measures like percentiles, quartiles, and z-scores help in understanding how data points are positioned relative to others in the dataset.

7.1 Categorical Data

7.1.1 Frequency and Proportion

- Definition of frequency
- Calculating proportion in categorical data

7.1.2 Contingency Table

- Definition and use of contingency tables
- Analyzing relationships between categorical variables

7.1.3 Frequency Distribution

- Creating and interpreting frequency distributions
- Visualizing frequency distributions using tables and bar charts

7.2 Numerical Data

7.2.1 Mean

- Definition and formula of mean
- Advantages and disadvantages of using the mean

7.2.2 Median

- Definition and how to calculate the median
- Median in skewed data distributions

7.2.3 Mode

- Definition and how to calculate the mode
- Mode in data with multiple identical values

7.2.4 Range

- How to calculate the range
- Use of range in data analysis

7.2.5 Variance

- Definition of variance and its relationship to standard deviation
- Variance in data distributions

7.2.6 Standard Deviation

- Definition and formula of standard deviation
- Interpretation of standard deviation values

7.3 Shape of Distribution

7.3.1 Skewness

- Definition and types: positive, negative, and zero skew
- Formula and interpretation
- Effects of skewness on mean and median

7.3.2 Kurtosis

- Definition and types: leptokurtic, platykurtic, mesokurtic
- Formula and interpretation
- Importance in risk and outlier analysis

7.4 Relative Position

7.4.1 Percentiles

- Definition and how to calculate
- Usage in performance measurement and threshold setting

7.4.2 Quartiles

- Q1, Q2 (median), Q3 definitions
- Application in boxplots and IQR calculation

7.4.3 Z-Score

- Standardized score relative to mean and standard deviation
- Use in comparing data points across distributions

7.4.4 T-Score

- Definition and formula
- Comparison with Z-score in small samples

7.5 Association Metrics

7.5.1 Covariance

- Meaning and direction of linear relationship
- Limitations in interpretation without normalization

7.5.2 Pearson Correlation Coefficient

- Strength and direction of linear relationship
- Range [-1, 1] and interpretation

7.5.3 Spearman's Rank Correlation

- Non-parametric correlation measure
- Suitable for ordinal data and monotonic relationships

7.5.4 Point-Biserial Correlation

- Used when one variable is continuous and one is binary
- Application in psychological and social sciences

Chapter 8

Descriptive Visualizations

8.1 Categorical Data

8.1.1 Bar Chart

- Used to display frequencies or proportions of categorical variables
- Each bar represents a category with its height corresponding to frequency
- Easily compares values across different categories

8.1.2 Pie Chart

- Represents proportions as slices of a circle
- Useful for showing part-to-whole relationships
- Less effective for comparing similar values

8.1.3 Frequency Table

- Tabular summary of frequency and proportion
- Can be accompanied by bar chart or pie chart for visual representation
- Enhances understanding of distribution in categorical data

8.2 Numerical Data

8.2.1 Histogram

- Displays the distribution of a continuous numeric variable
- Bins group values into intervals with bar heights representing frequency
- Useful for identifying skewness, modality, and spread

8.2.2 Boxplot (Box-and-Whisker Plot)

- Visualizes median, quartiles, and potential outliers
- Shows distribution shape and variability
- Ideal for comparing multiple numeric groups

8.2.3 Density Plot

- Smoothed version of a histogram using a kernel function
- Visualizes the probability distribution of a continuous variable
- Useful for overlaying multiple distributions

8.2.4 Line Chart

- Typically used for time series or ordered data
- Shows trends, patterns, or changes over intervals
- Each point connected with lines to indicate progression

8.3 Multivariate Visualizations

8.3.1 Scatter Plot

- Displays relationship between two numeric variables
- Shows trends, clusters, and possible outliers
- Basis for correlation and regression analysis

8.3.2 Bubble Chart

- Extension of scatter plot with size of bubbles representing a third variable
- Adds one more dimension of data insight
- Useful in business and demographic visualizations

8.3.3 Faceted Plots

- Create multiple subplots by splitting data based on a categorical variable
- Useful for comparing distributions or relationships across categories
- Often used with histograms, scatter plots, or boxplots

8.3.4 Heatmap

- Color-coded matrix showing values of two variables (often categorical or correlation)
- Commonly used for visualizing correlation matrices or frequency tables
- Colors represent magnitude or intensity

8.4 Distribution Shape and Outliers

8.4.1 Histogram and Density Plot

- Enhances distribution interpretation by showing both bar and smoothed curve
- Ideal for understanding shape, central tendency, and spread

8.4.2 Boxplot with Outlier Labels

- Explicitly marks outliers beyond whiskers
- Helps in detecting unusual data points

8.4.3 Violin Plot

- Merges boxplot and density plot
- Reveals detailed distribution shape while showing summary statistics

Chapter 9

Interactive Visualizations

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)