WQD7004 OCC3 Group 5

LOO LING YAN (23094683)

XIAN ZHIYI (23122622)

WONG YI TING (S2152880)

CHU JING HAN (23116920)

HAU JIA QI (17204762)

2025-01-04

This R Markdown document is the complete report of the group project by Group 5 for Occurrence 3 of the WQD7004 course in Faculty of Computer Science and Information Technology, University of Malaya for Semester 1 of 2024/2025 Academic Session.

---

1. Introduction

Title: Forecasting Market Sales by Using Machine Learning

This project aims to predict supermarket sales trends, identify the key factors influencing sales, and classify customer types (Normal or Member) using supervised machine learning techniques. The dataset is sourced from Kaggle and contains transaction records, customer demographics, and product details.

The analysis focuses on:

- Data Preprocessing: Handling missing values, standardizing data formats, and removing inconsistencies.
- Exploratory Data Analysis (EDA): Understanding data distribution, patterns, and relationships between variables.
- Regression Modeling: Predicting total sales amount per transaction using Linear Regression, Random Forest, and Gradient Boosting Regression models.
- Classification Modeling: Classifying customer types (Member vs. Normal) using Random Forest, Decision Tree, KNN, XGBoost, and LightGBM models.
- Model Evaluation: Comparing model performances using evaluation metrics such as Accuracy, F1-Score, R^2 Score, and Mean Squared Error (MSE).
- Insights and Recommendations: Drawing actionable conclusions for business decision-makers based on model outcomes.

Through this analysis, we aim to deliver an effective predictive framework that supports data-driven strategies for sales optimization and customer segmentation.

Goal: To predict sales trends and classify customer types (Normal or Member) using supervised machine learning models.

Objectives:

- To classify customer types (Normal or Member) using supervised classification models.
- To predict the total sales amount of individual transactions using regression models.
- To compare and identify the best-performing model for both regression and classification tasks.
- To evaluate model performance using key metric.

Research Questions:

- Which machine learning model provides the most accurate sales predictions?

- What are the key factors influencing supermarket sales trends?
- Which classification model (Random Forest, KNN, Decision Tree, XGBoost, LightGBM) performs best for predicting customer types?

Dataset description:

- Title: Market Sales Data
- Year: 2024
- Source:Kaggle - Market Sales Data (https://www.kaggle.com/datasets/willianoliveiragibin/market-sales-data/data)
- Purpose: Analyze supermarket sales trends and forecast future sales.
- Dimension: 1000 rows × 9 columns
- Structure: The structured tabular dataset contains numerical and categorical data.
- Summary: This dataset provides detailed transaction data from a supermarket, capturing customer demographics, product preferences, purchasing quantities, and sales-related financial details.

---

2. Data Preprocessing

2.1 Load libraries and functions

```
install.packages("dplyr")
install.packages("modeest")
install.packages("caret")
install.packages("ggplot2")
install.packages("readr")
library(caret)
library(dplyr)
library(readr)
library(modeest)
library(readr)
```

2.2 General Overview of The Raw Dataset

Import raw data

```
# Import raw data
df <- read_csv("supermarket_sales.csv", show_col_types = FALSE)
glimpse(df)
```

```
## Rows: 1,000
## Columns: 9
## $ Gender          <chr> "Female", "Female", "Female", "Female", "Female", "Fem…
## $ `Invoice ID`    <chr> "750-67-8428", "226-31-3081", "355-53-5943", "315-22-5…
## $ Branch          <chr> "A", "C", "A", "C", "A", "B", "B", "A", "A", "B", "A",…
## $ City            <chr> "Yangon", "Naypyitaw", "Yangon", "Naypyitaw", "Yangon"…
## $ `Customer type` <chr> "Member", "Normal", "Member", "Normal", "Member", "Mem…
## $ `Product line`  <chr> "Health and beauty", "Electronic accessories", "Electr…
## $ `Unit price`    <dbl> 74.69, 15.28, 68.84, 73.56, 36.26, 54.84, 14.48, 46.95…
## $ Quantity        <dbl> 7, 5, 6, 10, 2, 3, 4, 5, 10, 6, 7, 2, 8, 2, 5, 4, 1, 9…
## $ `Tax 5%`        <dbl> 261.4150, 3.8200, 20.6520, 36.7800, 3.6260, 8.2260, 2.…
```

The dataset details:

- **Gender:** Customer gender (Male/Female)

- **Invoice ID:** Unique transaction identifier

- **Branch:** Store branch identifier

- **City:** City where the branch is located

- **Customer Type:** Membership status (Member/Normal)

- **Product Line:** Category of purchased product

- **Unit Price:** Price per unit of the product

- **Quantity:** Number of units purchased

- **Tax 5%:** Tax calculated based on sales

2.3 Data Cleaning

Rename the columns as the column name contains spacing

```
df <- setNames(df, c("Gender", "Invoice_ID", "Branch","City", "Customer_type", "Produ
ct_line", "Unit_price", "Quantity", "Tax_5pct"))
head(df)
```

```
## # A tibble: 6 × 9
##   Gender Invoice_ID  Branch City  Customer_type Product_line Unit_price Quantity
##   <chr>  <chr>       <chr>  <chr> <chr>         <chr>             <dbl>    <dbl>
## 1 Female 750-67-8428 A      Yang… Member        Health and …       74.7        7
## 2 Female 226-31-3081 C      Nayp… Normal        Electronic …       15.3        5
## 3 Female 355-53-5943 A      Yang… Member        Electronic …       68.8        6
## 4 Female 315-22-5665 C      Nayp… Normal        Home and li…       73.6       10
## 5 Female 665-32-9167 A      Yang… Member        Health and …       36.3        2
## 6 Female 692-92-5582 B      Mand… Member        Food and be…       54.8        3
## # ℹ 1 more variable: Tax_5pct <dbl>
```

Check duplication from dataset

```
dup <- sum(duplicated(df))
cat("Total duplicated from supermarket_sales is", dup)
```

```
## Total duplicated from supermarket_sales is 0
```

- Result: no duplication here.

Drop invoice ID that no related to the analysis

```
df<-df %>% select(-Invoice_ID)
```

Run glimpse again to ensure Invoice ID was drop from the dataset

```
glimpse(df)
```

```
## Rows: 1,000
## Columns: 8
## $ Gender        <chr> "Female", "Female", "Female", "Female", "Female", "Femal…
## $ Branch        <chr> "A", "C", "A", "C", "A", "B", "B", "A", "A", "B", "A", "…
## $ City          <chr> "Yangon", "Naypyitaw", "Yangon", "Naypyitaw", "Yangon", …
## $ Customer_type <chr> "Member", "Normal", "Member", "Normal", "Member", "Membe…
## $ Product_line  <chr> "Health and beauty", "Electronic accessories", "Electron…
## $ Unit_price    <dbl> 74.69, 15.28, 68.84, 73.56, 36.26, 54.84, 14.48, 46.95, …
## $ Quantity      <dbl> 7, 5, 6, 10, 2, 3, 4, 5, 10, 6, 7, 2, 8, 2, 5, 4, 1, 9, …
## $ Tax_5pct      <dbl> 261.4150, 3.8200, 20.6520, 36.7800, 3.6260, 8.2260, 2.89…
```

- Result: Invoice_ID is remove from the dataset.

Check missing value from any columns

```
any(is.na(df))
```

```
## [1] TRUE
```

- Result: Existing dataset having missing value.

Check numbers for row without missing value

```
nrow(na.omit(df))
```

```
## [1] 994
```

Check the missing value record

```
nrow(df[!complete.cases(df),])
```

```
## [1] 6
```

- Result: Total 1000 records show 994 records is complete record and 6 records having missing value.

Check which column having missing value

```
check_column_missing <-names(df)
for(i in check_column_missing ) {
print(paste(i, sum(df[i]=="" | is.na(df[i]))))
}
```

```
## [1] "Gender 0"
## [1] "Branch 0"
## [1] "City 0"
## [1] "Customer_type 0"
## [1] "Product_line 3"
## [1] "Unit_price 3"
## [1] "Quantity 0"
## [1] "Tax_5pct 0"
```

- Result: “Produt line” and “unit price” having missing value.

Check which column having NA

```
check_column_missing <-names(df)
for(i in check_column_missing ) {
print(paste(i, sum(is.na(df[i]))))
}
```

```
## [1] "Gender 0"
## [1] "Branch 0"
## [1] "City 0"
## [1] "Customer_type 0"
## [1] "Product_line 3"
## [1] "Unit_price 3"
## [1] "Quantity 0"
## [1] "Tax_5pct 0"
```

- Result: Using others command to check the NA also show the same result.

Check which column having missing value

```
for(i in check_column_missing) {
print(paste(i, which(df[i]=="" | is.na(df[i]))))
}
```

```
## [1] "Gender "
## [1] "Branch "
## [1] "City "
## [1] "Customer_type "
## [1] "Product_line 39"  "Product_line 76"  "Product_line 117"
## [1] "Unit_price 57"  "Unit_price 82"  "Unit_price 103"
## [1] "Quantity "
## [1] "Tax_5pct "
```

- Result: missing value for “Product line 39” “Product line 76” “Product line 117” and also “Unit price 57” “Unit price 82” “Unit price 103”.
- There are 6 rows of missing value in the dataset. To fill in the missing value, mode method is used for the column ‘Product Line’ and calculation to fill in the missing value for column ‘Tax_5%’.

```
Prod_mode <- function(x) {
  uniq_x <- unique(x)
  uniq_x[which.max(tabulate(match(x, uniq_x)))]
}
mode_pro <- Prod_mode(df$Product_line)
print(mode_pro)
```

```
## [1] "Fashion accessories"
```

Fill in the “Fashion accessories” to the missing value from the Product line.

```
df <- df %>% mutate(Product_line = ifelse(is.na(Product_line), "Fashion accessories",
Product_line))
# Check the missing value had fill with "Fashion accessories"
df[39, ]
```

```
## # A tibble: 1 × 8
##   Gender Branch City     Customer_type Product_line Unit_price Quantity Tax_5pct
##   <chr>  <chr>  <chr>    <chr>         <chr>             <dbl>    <dbl>    <dbl>
## 1 Female C      Naypyit… Member        Fashion acc…       78.3       10     39.2
```

- Result: “Product_line 39” was fill in with “Fashion accessories”.

Check missing value row again

```
nrow(na.omit(df))
```

```
## [1] 997
```

- Result: the non-missing value was increase from 994 records to 997 records.

Get mean value by removing missing value

```
mean_unit_price <- round(mean(df$Unit_price, na.rm=T), 2)
mean_unit_price
```

```
## [1] 55.8
```

```
# fill in the mean value=55.8 to the missing value of unit_price
df <- df %>% mutate(Unit_price = ifelse(is.na(Unit_price), 55.8, Unit_price))

# check the "Unit_price 82" that fill in 55.8
df[82, ]
```

```
## # A tibble: 1 × 8
##   Gender Branch City   Customer_type Product_line   Unit_price Quantity Tax_5pct
##   <chr>  <chr>  <chr>  <chr>         <chr>               <dbl>    <dbl>    <dbl>
## 1 Female A      Yangon Normal        Sports and tr…       55.8        7     4.32
```

- Result: the unit_price was fill in the mean value.

Check the missing record from the dataset again

```
nrow(na.omit(df))
```

```
## [1] 1000
```

- Result: Now, all the missing value was filled in.

Check the amount of tax= unit price * quantity * tax 5%

```
df <- df %>% mutate(Tax= Unit_price * Quantity * 0.05)
df<-df %>% select(-Tax_5pct)
head(df)
```

```
## # A tibble: 6 × 8
##   Gender Branch City      Customer_type Product_line   Unit_price Quantity   Tax
##   <chr>  <chr>  <chr>     <chr>         <chr>               <dbl>    <dbl> <dbl>
## 1 Female A      Yangon    Member        Health and be…       74.7        7 26.1
## 2 Female C      Naypyitaw Normal        Electronic ac…       15.3        5  3.82
## 3 Female A      Yangon    Member        Electronic ac…       68.8        6 20.7
## 4 Female C      Naypyitaw Normal        Home and life…       73.6       10 36.8
## 5 Female A      Yangon    Member        Health and be…       36.3        2  3.63
## 6 Female B      Mandalay  Member        Food and beve…       54.8        3  8.23
```

- Result: rom eyeboll checking the “Tax 5%” is different with “total_sales”. Therefore, “Tax 5%” will drop from the dataset.

Check complete records for each rows

```
sum(complete.cases(df))
```

```
## [1] 1000
```

```
# Detect outlier from the Tax
Q1 <- quantile(df$Tax, 0.25)
Q3 <- quantile(df$Tax, 0.75)
IQR <- Q3 - Q1

# Define the lower and upper bounds
lower_bound <- Q1 - 1.5 * IQR
upper_bound <- Q3 + 1.5 * IQR

# Remove outliers
df<- df[df$Tax >= lower_bound & df$Tax <= upper_bound, ]

# check numbers of records
nrow(df)
```

```
## [1] 991
```

```
# summary of dataset
summary(df)
```

```
##     Gender             Branch              City          Customer_type
##  Length:991         Length:991         Length:991         Length:991
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##  Product_line         Unit_price       Quantity           Tax
##  Length:991         Min.   :10.08   Min.   : 1.000   Min.   : 0.5085
##  Class :character   1st Qu.:33.05   1st Qu.: 3.000   1st Qu.: 5.8948
##  Mode  :character   Median :55.04   Median : 5.000   Median :12.0600
##                     Mean   :55.43   Mean   : 5.469   Mean   :15.1166
##                     3rd Qu.:77.48   3rd Qu.: 8.000   3rd Qu.:22.2585
##                     Max.   :99.96   Max.   :10.000   Max.   :45.3250
```

```
# Cleaning of dataset
head(df)
```

```
## # A tibble: 6 × 8
##   Gender Branch City      Customer_type Product_line   Unit_price Quantity   Tax
##   <chr>  <chr>  <chr>     <chr>         <chr>               <dbl>    <dbl> <dbl>
## 1 Female A      Yangon    Member        Health and be…       74.7        7 26.1
## 2 Female C      Naypyitaw Normal        Electronic ac…       15.3        5  3.82
## 3 Female A      Yangon    Member        Electronic ac…       68.8        6 20.7
## 4 Female C      Naypyitaw Normal        Home and life…       73.6       10 36.8
## 5 Female A      Yangon    Member        Health and be…       36.3        2  3.63
## 6 Female B      Mandalay  Member        Food and beve…       54.8        3  8.23
```

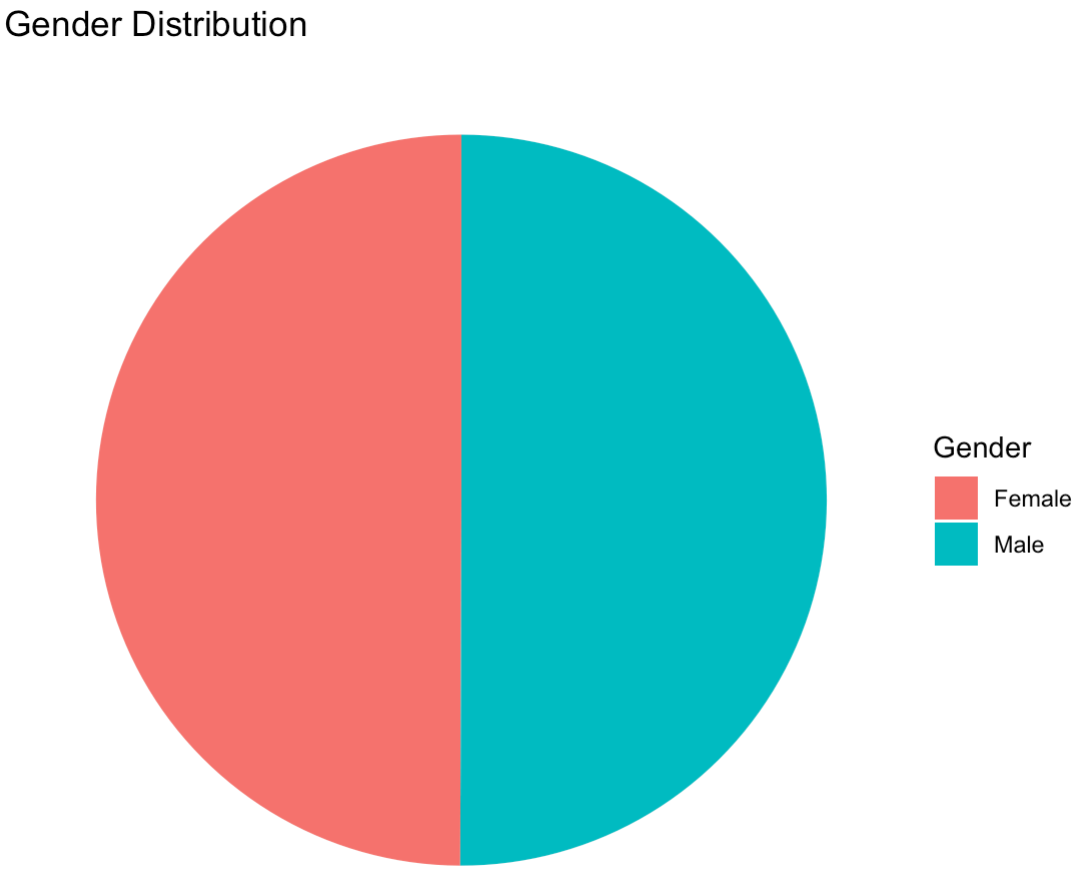
- Result: now, the data is clean after removing the outlier and using for the following analysis.

---

3. EDA

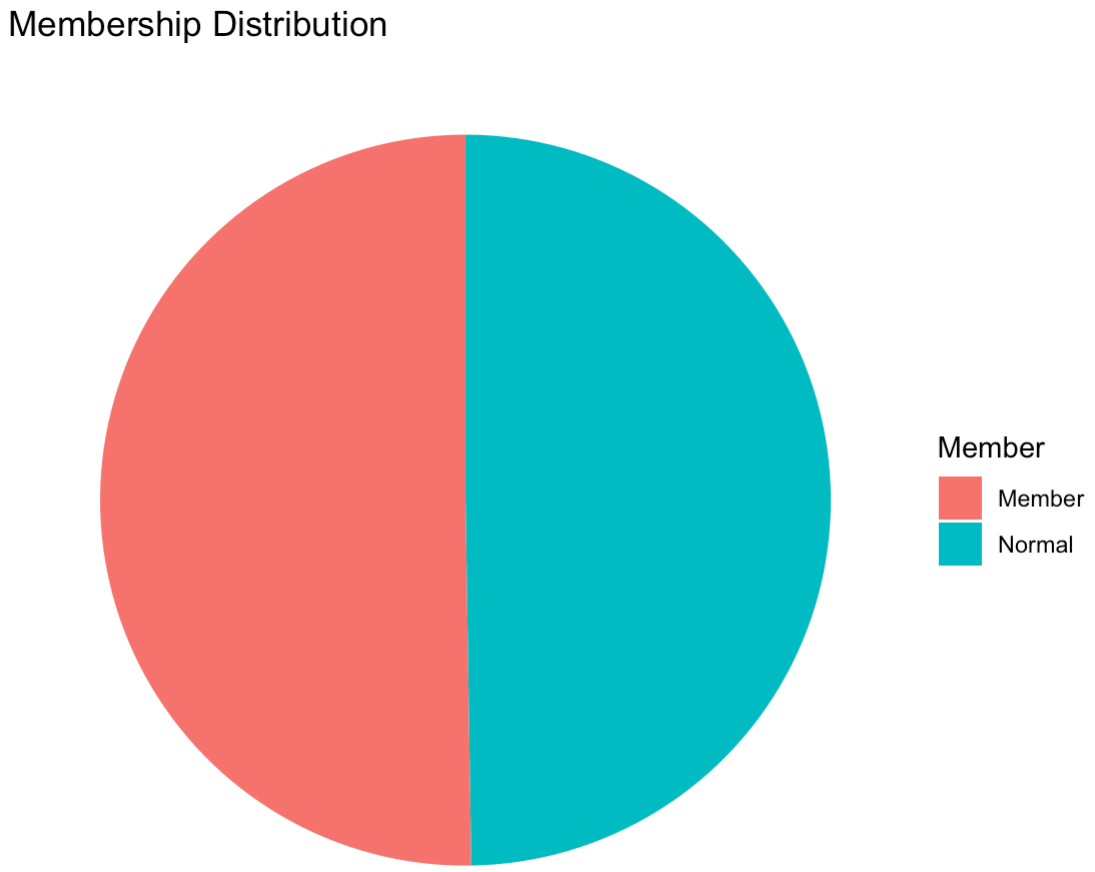
3.1 Gender Distribution

```
# Generate frequency table
gender_counts <- table(df$Gender)
# Convert to a data frame
gender_df <- as.data.frame(gender_counts)
colnames(gender_df) <- c("Gender", "Count")
#Plot pie chart
ggplot(gender_df, aes(x = "", y = Count, fill = Gender)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar(theta = "y") +
  labs(title = "Gender Distribution") +
  theme_void()
```



3.2 Membership Distribution

```
# Generate frequency table
member <- table(df$Customer_type)
# Convert to a data frame
memberdf <- as.data.frame(member)
colnames(memberdf) <- c("Member", "Count")
#Plot pie chart
ggplot(memberdf, aes(x = "", y = Count, fill = Member)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar(theta = "y") +
  labs(title = "Membership Distribution") +
  theme_void()
```



3.3 Proportion of Membership by Gender

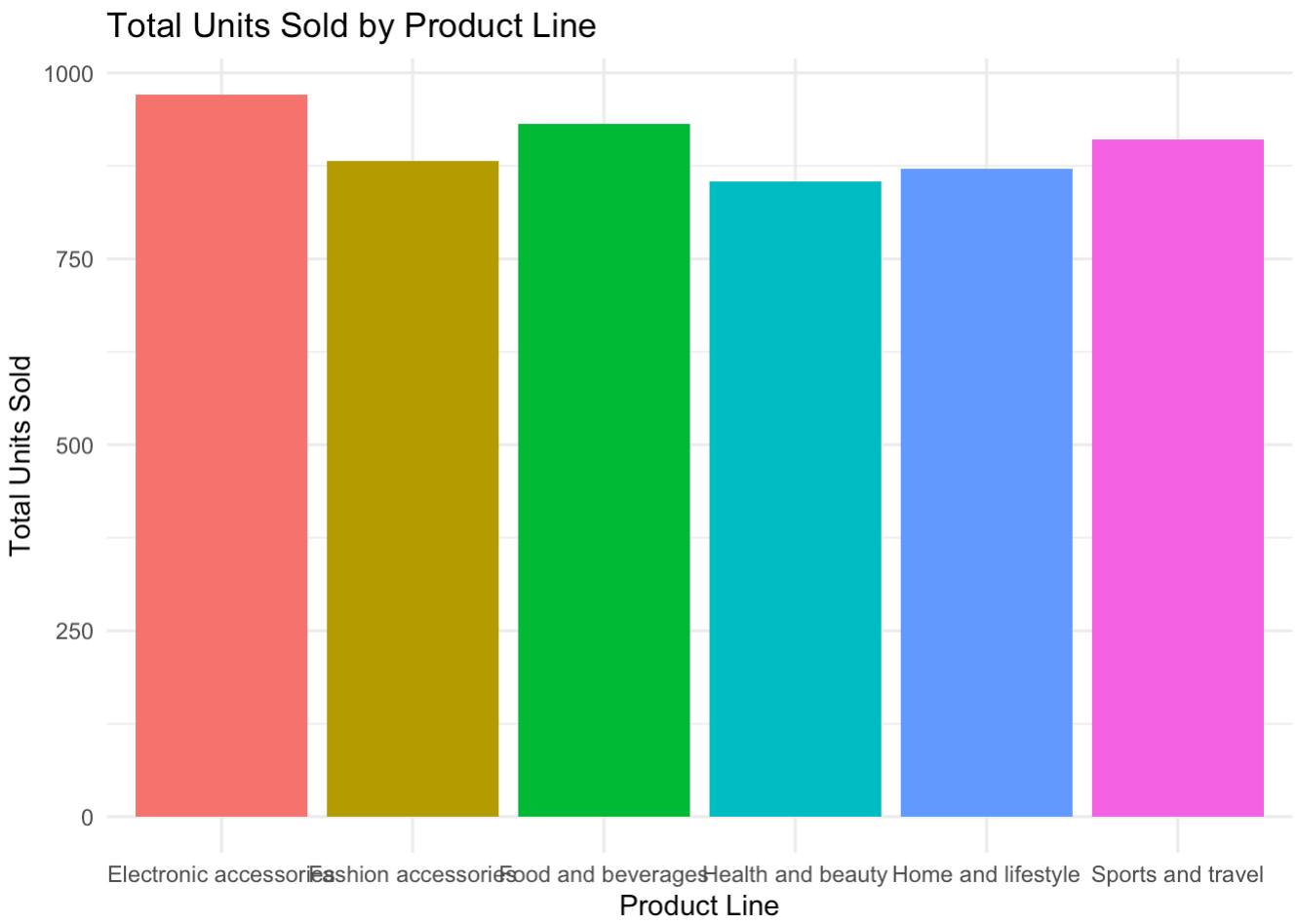
```
memgen <- df %>%
  group_by(Customer_type, Gender) %>%
  tally() %>%
  group_by(Customer_type) %>%
  mutate(Proportion = n / sum(n))

# Stacked bar chart for the proportion of membership by gender
ggplot(memgen, aes(x = Customer_type, y = Proportion, fill = Gender)) +
  geom_bar(stat = "identity") +
  labs(title = "Proportion of Membership by Gender", x = "Membership", y = "Proportio
n") +
  scale_y_continuous(labels = scales::percent)
```

3.4 Total Unit Sold for Each Product

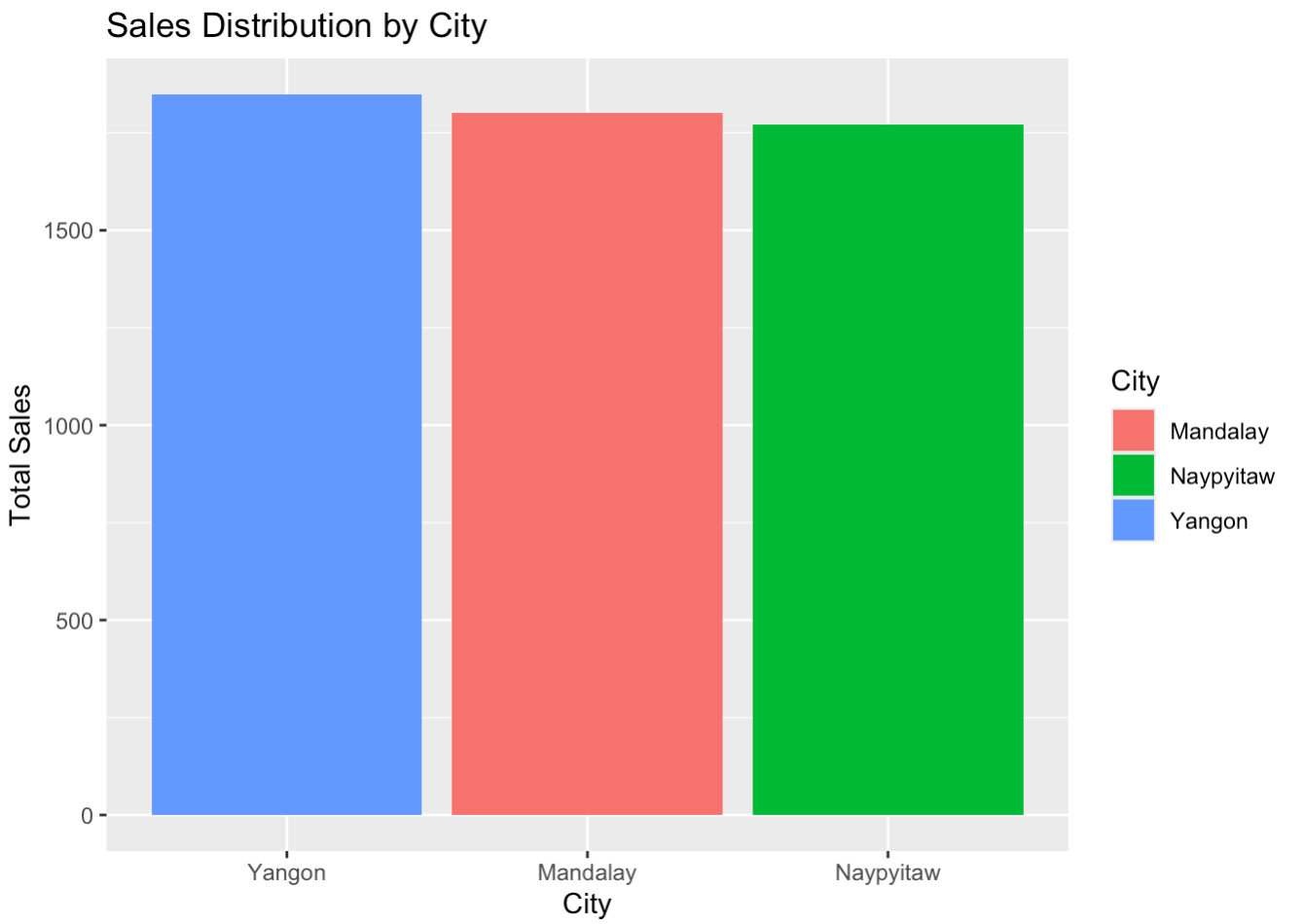
```
total <- df %>%
  group_by(Product_line) %>%
  summarize(totalunit = sum(Quantity))

# Bar chart of Total Units Sold by Product Line
ggplot(total, aes(x = Product_line, y = totalunit, fill = Product_line)) +
  geom_bar(stat = "identity", show.legend = FALSE) +
  labs(title = "Total Units Sold by Product Line",
       x = "Product Line",
       y = "Total Units Sold") +
  theme_minimal()
```



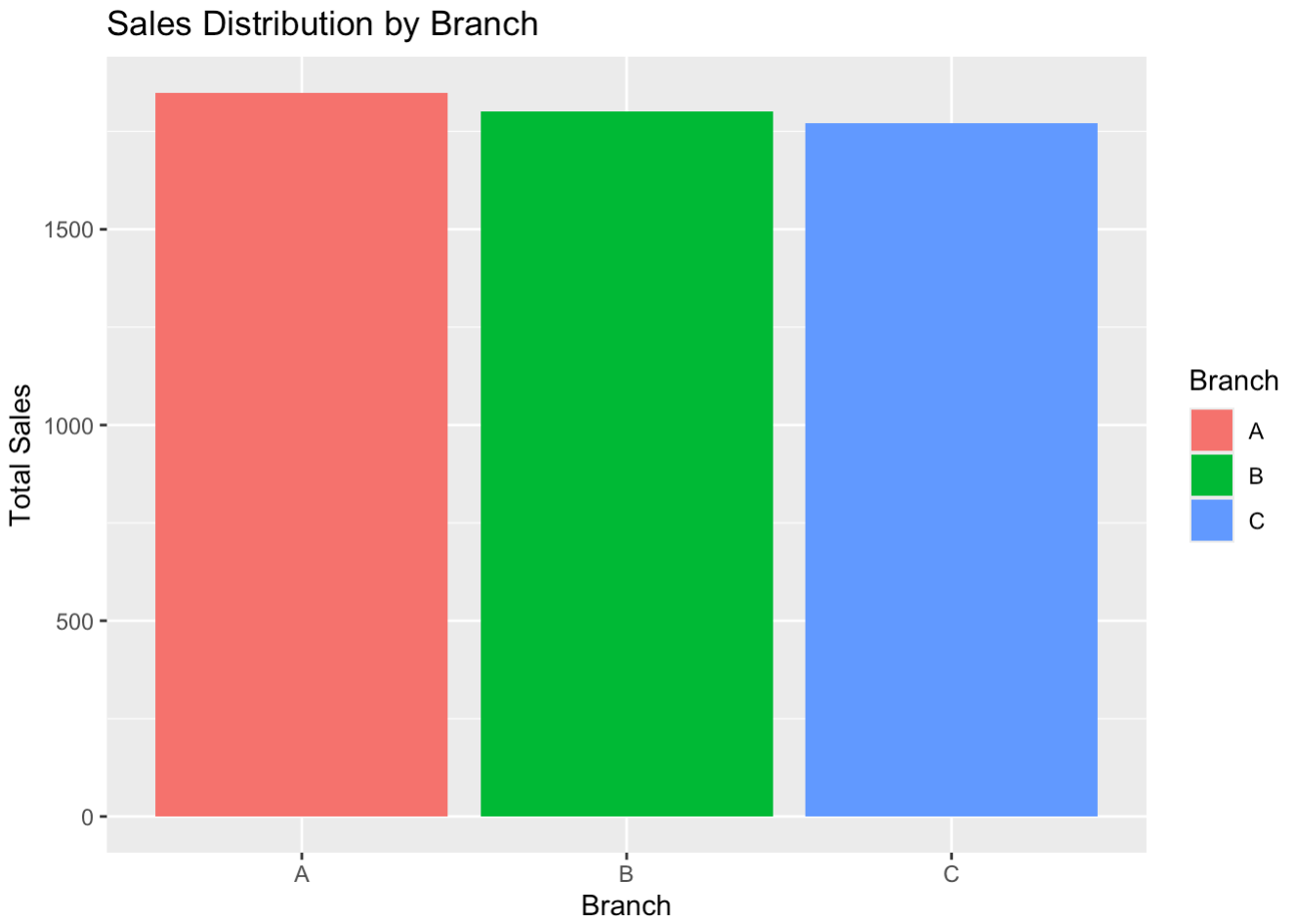
3.5 Analyze sales distribution and trends by city

```
citysales <- df %>% group_by(City) %>%
  summarize(totalsales = sum(Quantity)) %>%
  arrange(desc(totalsales))
ggplot(citysales, aes(x = reorder(City, -totalsales), y = totalsales, fill = City)) +
  geom_bar(stat = "identity") +
  labs(title = "Sales Distribution by City", x = "City", y = "Total Sales")
```



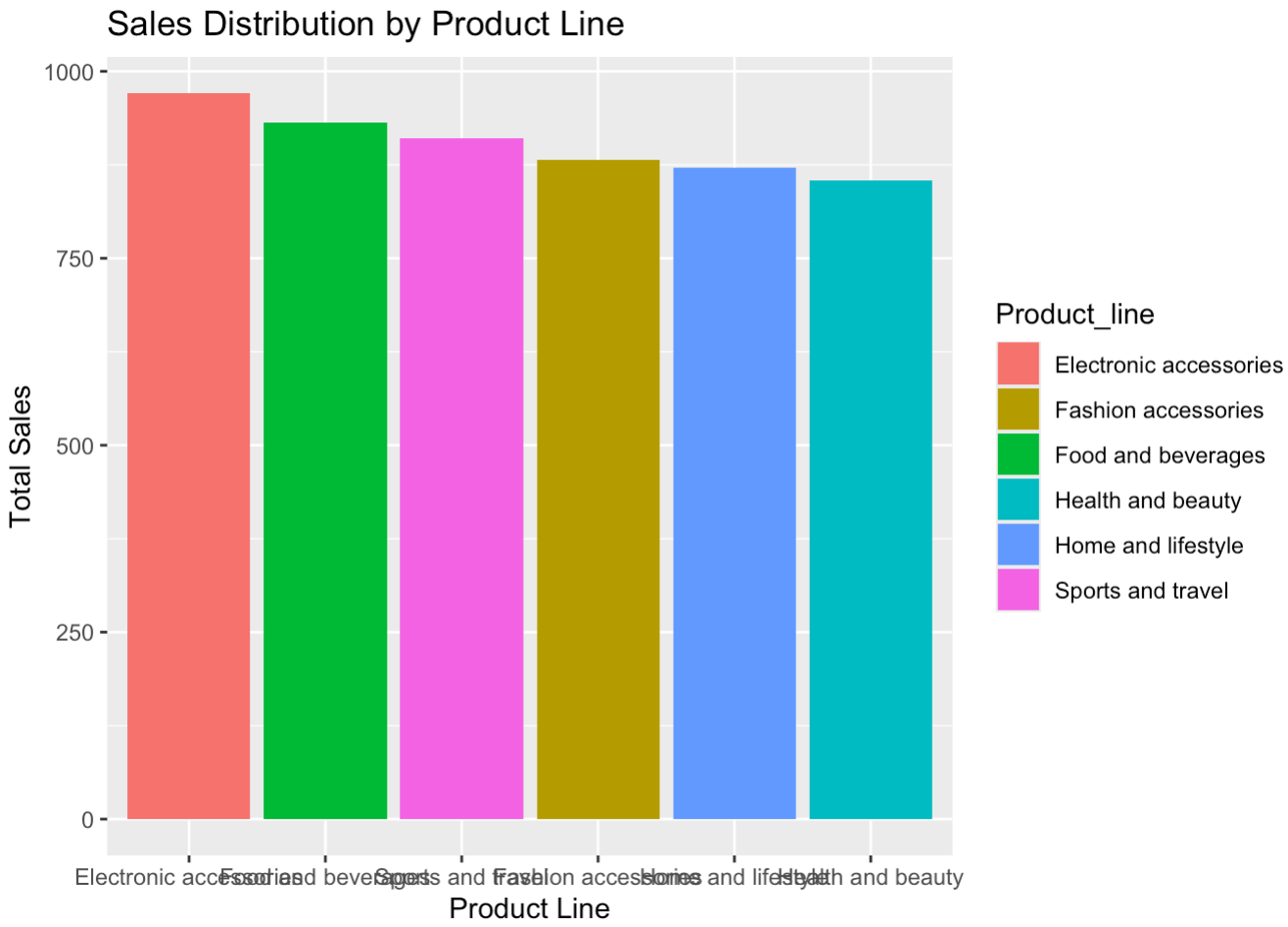
3.6 Analyze sales distribution and trends by branch

```
branchsales <- df %>% group_by(Branch) %>%
  summarize(totalbranchsales = sum(Quantity)) %>%
  arrange(desc(totalbranchsales))
ggplot(branchsales, aes(x = reorder(Branch, -totalbranchsales), y = totalbranchsales,
fill = Branch)) +
  geom_bar(stat = "identity") +
  labs(title = "Sales Distribution by Branch", x = "Branch", y = "Total Sales")
```



3.7 Analyze sales distribution and trends by product line

```
productsales <- df %>% group_by(Product_line) %>%
  summarize(totalproductsales = sum(Quantity)) %>%
  arrange(desc(totalproductsales))
ggplot(productsales, aes(x = reorder(Product_line, -totalproductsales), y = totalprod
uctsales, fill = Product_line)) +
  geom_bar(stat = "identity") +
  labs(title = "Sales Distribution by Product Line", x = "Product Line", y = "Total S
ales")
```

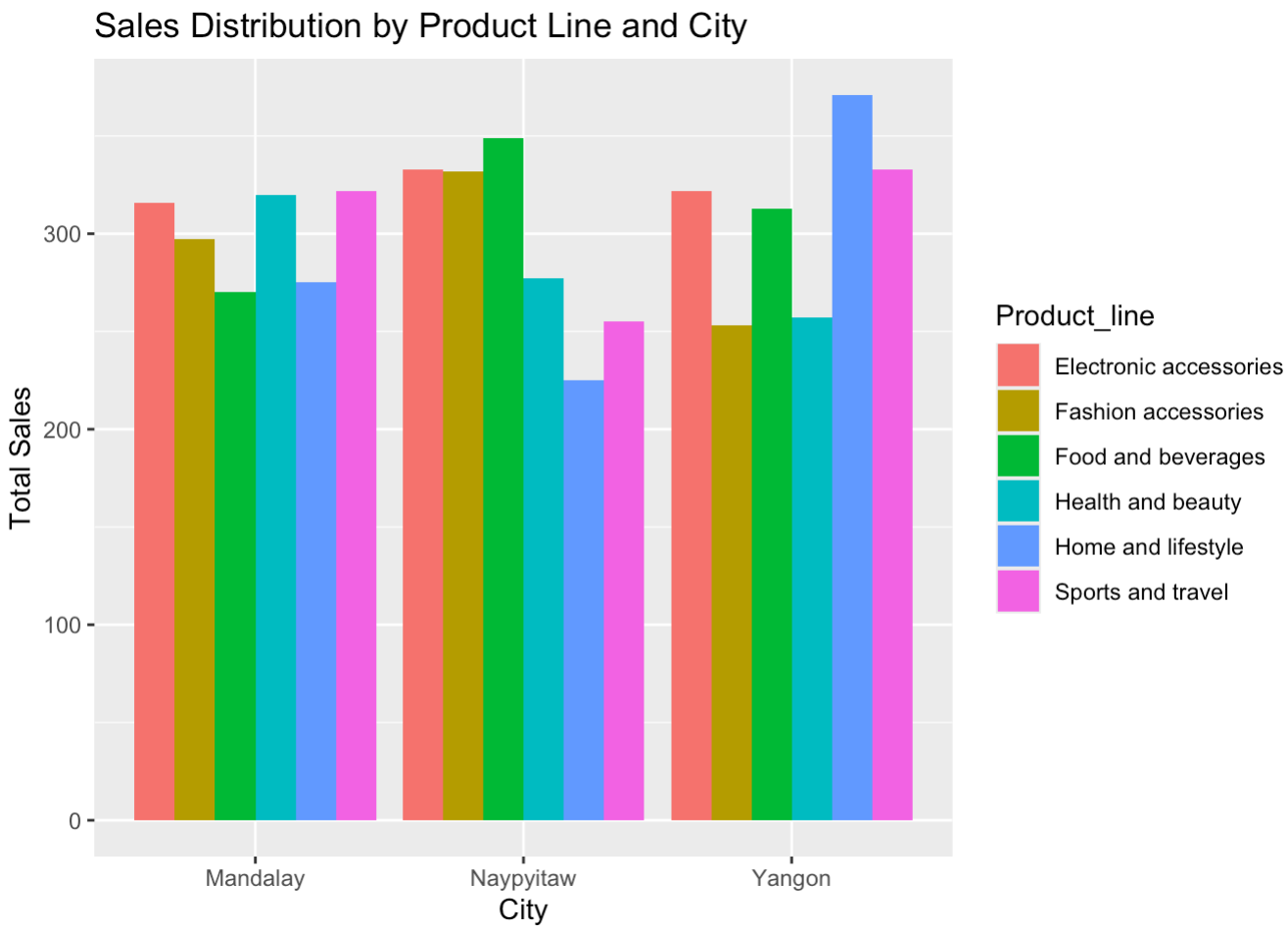


3.8 Analyze sales of product line in each city

```
sales_summary <- df %>%
  group_by(City, Branch, Product_line) %>%
  summarize(Total_Sales = sum(Quantity, na.rm = TRUE))
```

```
## `summarise()` has grouped output by 'City', 'Branch'. You can override using
## the `.groups` argument.
```

```
ggplot(sales_summary, aes(x = City, y = Total_Sales, fill = Product_line)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Sales Distribution by Product Line and City",
       x = "City", y = "Total Sales")
```

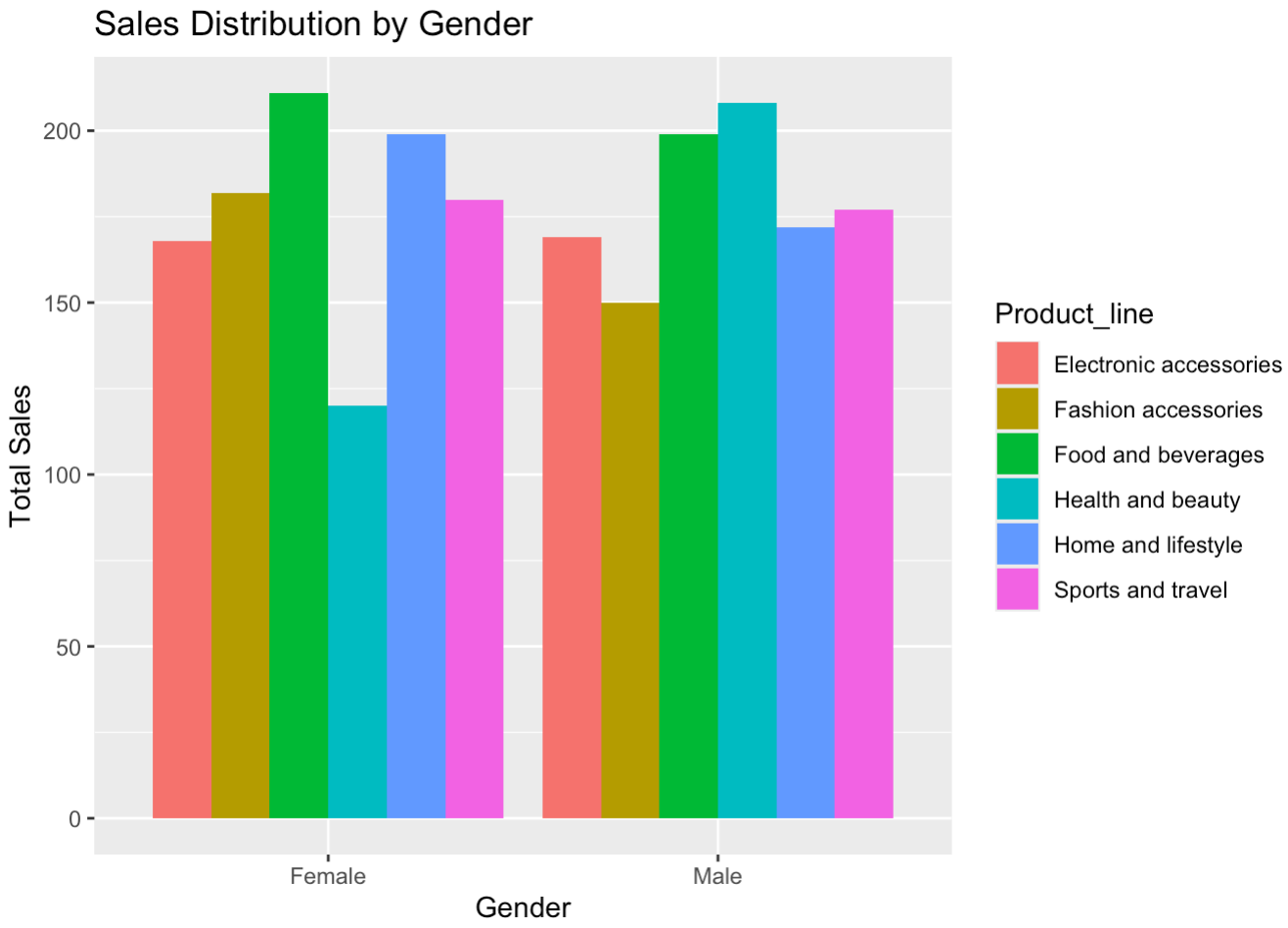


3.9 Analyze sales of each product line by gender

```
Gender_summary <- df %>%
  group_by(Gender, Branch, Product_line) %>%
  summarize(Gender_Sales = sum(Quantity, na.rm = TRUE))
```

```
## `summarise()` has grouped output by 'Gender', 'Branch'. You can override using
## the `.groups` argument.
```

```
ggplot(Gender_summary, aes(x = Gender, y = Gender_Sales, fill = Product_line)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Sales Distribution by Gender",
       x = "Gender", y = "Total Sales")
```

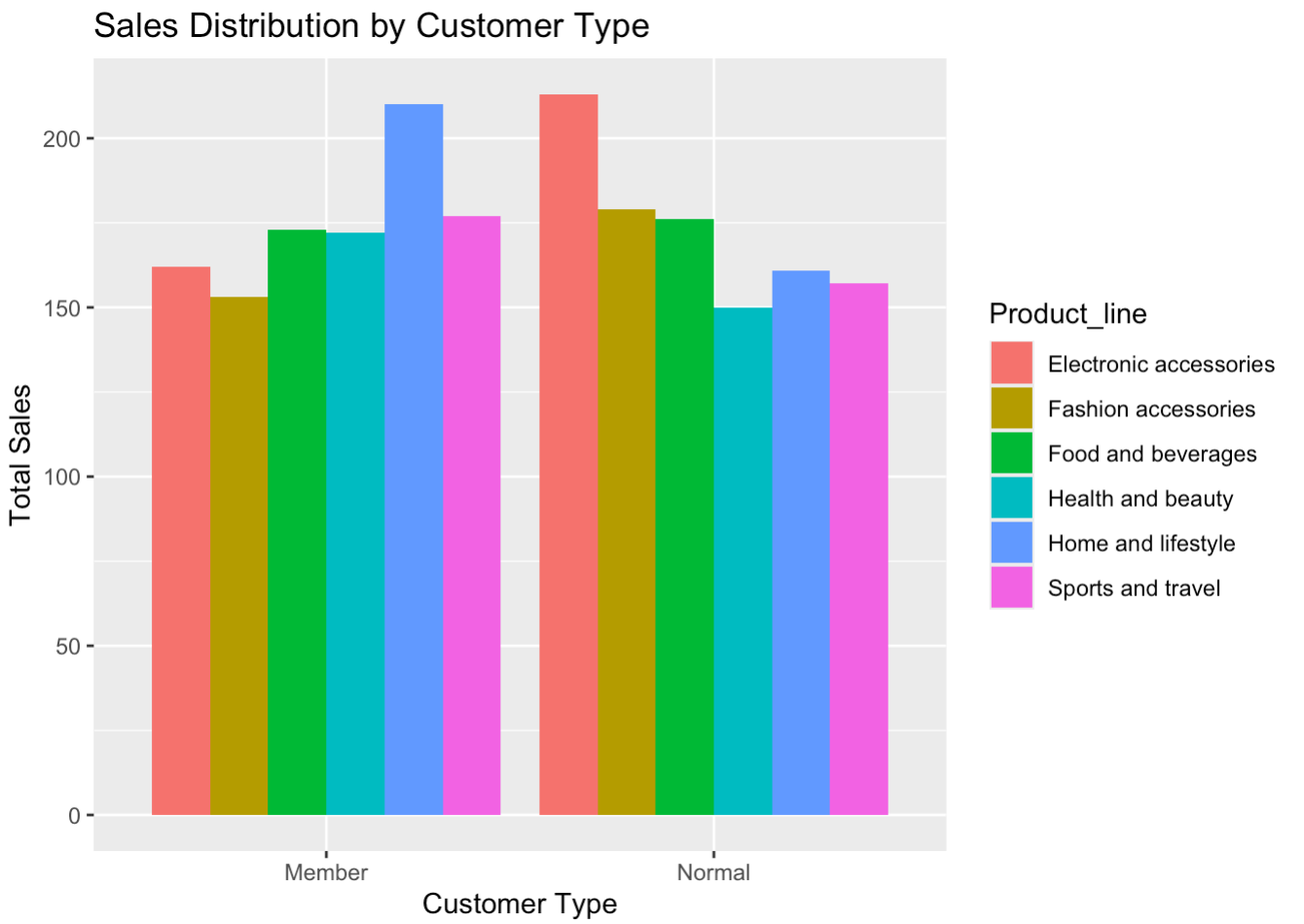


3.10 Analyze sales of each product line by membership status

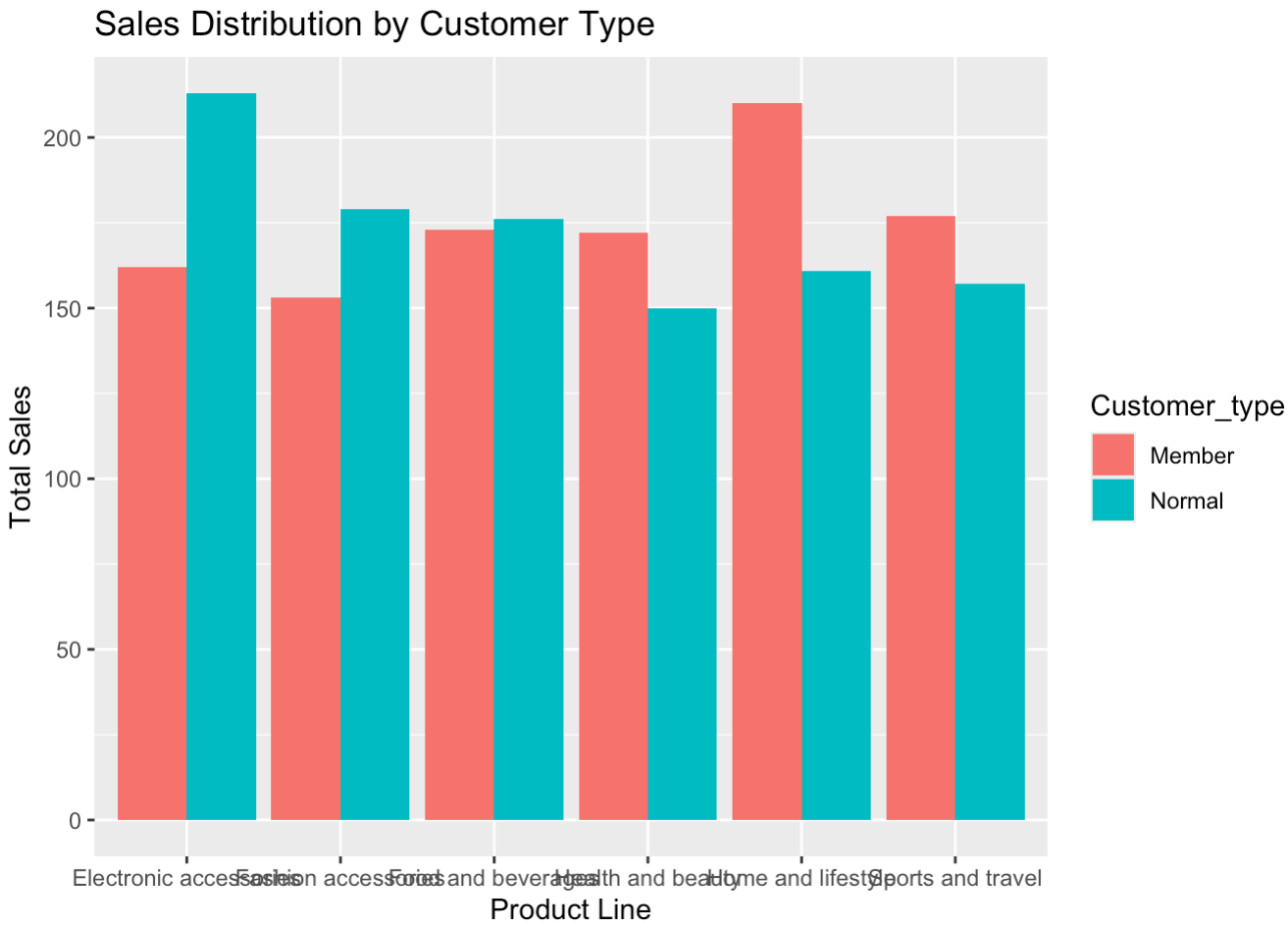
```
#group by customer type
Member_summary <- df %>%
  group_by(Customer_type, Branch, Product_line) %>%
  summarize(Member_Sales = sum(Quantity, na.rm = TRUE))
```

```
## `summarise()` has grouped output by 'Customer_type', 'Branch'. You can override
## using the `.groups` argument.
```

```
ggplot(Member_summary, aes(x = Customer_type, y = Member_Sales, fill = Product_line))
+
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Sales Distribution by Customer Type",
       x = "Customer Type", y = "Total Sales")
```




```
#group by product line
ggplot(Member_summary, aes(x = Product_line, y = Member_Sales, fill = Customer_type))
+
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Sales Distribution by Customer Type",
       x = "Product Line", y = "Total Sales")
```



3.11 Quantity of Each Product By City

```
ggplot(df, aes(x = Product_line, y = Quantity, fill = Product_line)) +
  geom_boxplot() +
  facet_wrap(~ City) +
  theme_minimal() +
  labs(title = "Boxplot of Quantity by Product Line and City",
       x = "Product Line",
       y = "Quantity") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

4. Classification Model

Objective

- The objective of this project is to predict **customer type** (Normal or Member) using supervised machine learning models.

4.1 Data Preparation

The dataset was preprocessed to ensure clean and consistent inputs for modeling:

1. **Missing Values**: Removed using `na.omit()`.
2. **Encoding**: Categorical variables like `Gender`, `City`, and `Branch` were converted to numerical codes.
3. **Feature Engineering**: Added `Prod_line_code`, `City_code`, `Branch_code`, and `Cust_type_Code` for better representation.
4. **Redundancy Removal**: Dropped original categorical columns to avoid duplication.
5. **Validation**: Ensured proper formatting using `str()`.

These steps optimized the dataset for machine learning algorithms.

```r
# Load required libraries
library(ggplot2)
library(dplyr)
library(randomForest)
library(caret)
library(e1071)
library(xgboost)
library(lightgbm)
library(rpart)
library(class)

# Inspect the dataset
summary(df)
```

```
##     Gender             Branch              City           Customer_type
##  Length:991         Length:991         Length:991         Length:991
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##  Product_line         Unit_price       Quantity           Tax
##  Length:991         Min.   :10.08   Min.   : 1.000   Min.   : 0.5085
##  Class :character   1st Qu.:33.05   1st Qu.: 3.000   1st Qu.: 5.8948
##  Mode  :character   Median :55.04   Median : 5.000   Median :12.0600
##                     Mean   :55.43   Mean   : 5.469   Mean   :15.1166
##                     3rd Qu.:77.48   3rd Qu.: 8.000   3rd Qu.:22.2585
##                     Max.   :99.96   Max.   :10.000   Max.   :45.3250
```

```r
str(df)
```

```
## tibble [991 × 8] (S3: tbl_df/tbl/data.frame)
##  $ Gender       : chr [1:991] "Female" "Female" "Female" "Female" ...
##  $ Branch       : chr [1:991] "A" "C" "A" "C" ...
##  $ City         : chr [1:991] "Yangon" "Naypyitaw" "Yangon" "Naypyitaw" ...
##  $ Customer_type: chr [1:991] "Member" "Normal" "Member" "Normal" ...
##  $ Product_line : chr [1:991] "Health and beauty" "Electronic accessories" "Electr
onic accessories" "Home and lifestyle" ...
##  $ Unit_price   : num [1:991] 74.7 15.3 68.8 73.6 36.3 ...
##  $ Quantity     : num [1:991] 7 5 6 10 2 3 4 5 10 6 ...
##  $ Tax          : num [1:991] 26.14 3.82 20.65 36.78 3.63 ...
```

```r
# Check and encode categorical variables
df$Gender <- as.factor(df$Gender)
df$Customer_type <- as.factor(df$Customer_type)
df$Product_line <- as.factor(df$Product_line)
df$City <- as.factor(df$City)
df$Branch <- as.factor(df$Branch)

# Create encoded fields
df <- df %>%
  mutate(
    Gender_code = as.integer(Gender == "Female"),
    Branch_code = as.integer(factor(Branch, levels = c("A", "B", "C"))),
    City_code = as.integer(factor(City, levels = c("Mandalay", "Naypyitaw", "Yango
n"))),
    Cust_type_Code = as.integer(Customer_type == "Normal"),
    Prod_line_code = as.integer(factor(Product_line, levels = c(
      "Electronic accessories", "Fashion accessories", "Food and beverages",
      "Health and beauty", "Home and lifestyle", "Sports and travel"
    )))
  )

# Verify the new structure
str(df)
```

```
## tibble [991 × 13] (S3: tbl_df/tbl/data.frame)
##  $ Gender        : Factor w/ 2 levels "Female","Male": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Branch        : Factor w/ 3 levels "A","B","C": 1 3 1 3 1 2 2 1 1 2 ...
##  $ City          : Factor w/ 3 levels "Mandalay","Naypyitaw",..: 3 2 3 2 3 1 1 3 3
1 ...
##  $ Customer_type : Factor w/ 2 levels "Member","Normal": 1 2 1 2 1 1 1 2 2 1 ...
##  $ Product_line  : Factor w/ 6 levels "Electronic accessories",..: 4 1 1 5 4 3 2 1
4 6 ...
##  $ Unit_price    : num [1:991] 74.7 15.3 68.8 73.6 36.3 ...
##  $ Quantity      : num [1:991] 7 5 6 10 2 3 4 5 10 6 ...
##  $ Tax           : num [1:991] 26.14 3.82 20.65 36.78 3.63 ...
##  $ Gender_code   : int [1:991] 1 1 1 1 1 1 1 1 1 1 ...
##  $ Branch_code   : int [1:991] 1 3 1 3 1 2 2 1 1 2 ...
##  $ City_code     : int [1:991] 3 2 3 2 3 1 1 3 3 1 ...
##  $ Cust_type_Code: int [1:991] 0 1 0 1 0 0 0 1 1 0 ...
##  $ Prod_line_code: int [1:991] 4 1 1 5 4 3 2 1 4 6 ...
```

```r
# Remove original categorical columns to avoid redundancy
df <- df %>%
  select(-Gender, -Product_line, -City, -Branch)

# Display a sample of the updated data for verification
head(df)
```

```
## # A tibble: 6 × 9
##   Customer_type Unit_price Quantity   Tax Gender_code Branch_code City_code
##   <fct>              <dbl>    <dbl> <dbl>       <int>       <int>     <int>
## 1 Member              74.7        7 26.1            1           1         3
## 2 Normal              15.3        5  3.82           1           3         2
## 3 Member              68.8        6 20.7            1           1         3
## 4 Normal              73.6       10 36.8            1           3         2
## 5 Member              36.3        2  3.63           1           1         3
## 6 Member              54.8        3  8.23           1           2         1
## # ℹ 2 more variables: Cust_type_Code <int>, Prod_line_code <int>
```

4.2 Customer Type Prediction

Data Preparation for Customer Type

To enhance model robustness with limited data:

1. **Train-Test Split**: Split dataset (80% train, 20% test) using stratified sampling.
2. **Noise Injection**: Added 1% Gaussian noise to numerical features in training data.
3. **Data Augmentation**: Combined original and noise-augmented data to double training size.
4. **Validation**: Ensured consistent target levels and class balance.

These steps improved data variability and model generalization.

```
# Split the data for Customer Type prediction
set.seed(123)
cust_type_trainIndex <- createDataPartition(df$Cust_type_Code, p = 0.8, list = FALSE)
data_cust_type_train <- df[cust_type_trainIndex, ]
data_cust_type_test <- df[-cust_type_trainIndex, ]

# Ensure all features are numeric
numeric_features <- sapply(data_cust_type_train, is.numeric)
data_cust_type_train <- data_cust_type_train[, numeric_features]
data_cust_type_test <- data_cust_type_test[, numeric_features]

# Ensure Cust_type_Code is a factor with consistent levels
data_cust_type_train$Cust_type_Code <- as.factor(data_cust_type_train$Cust_type_Code)
data_cust_type_test$Cust_type_Code <- factor(data_cust_type_test$Cust_type_Code,
                                             levels = levels(data_cust_type_train$Cu
st_type_Code))

# Noise Injection for Data Augmentation
augment_data_with_noise <- function(data, target_column, noise_level = 0.01) {
  augmented_data <- data
  numeric_columns <- setdiff(names(data), target_column)
  for (col in numeric_columns) {
    if (is.numeric(data[[col]])) {
      noise <- rnorm(n = nrow(data), mean = 0, sd = noise_level * sd(data[[col]]))
      augmented_data[[col]] <- augmented_data[[col]] + noise
    }
  }
  return(augmented_data)
}

# Apply noise injection to training data
data_cust_type_train_noisy <- augment_data_with_noise(data_cust_type_train, "Cust_typ
e_Code")

# Combine original and augmented data
data_cust_type_train <- rbind(data_cust_type_train, data_cust_type_train_noisy)

# Check the distribution of Cust_type_Code after augmentation
table(data_cust_type_train$Cust_type_Code)
```

```
##
##   0   1
## 804 782
```

```
table(data_cust_type_test$Cust_type_Code)
```

```
##
##   0   1
##  96 102
```

Random Forest for Customer Type Prediction

The Random Forest model was trained with:

1. **Hyperparameter Tuning**: Grid search over `mtry` (2, 3, 4) with 5-fold cross-validation and 500 trees.
2. **Feature Importance**: Key predictors identified using `varImp()`.
3. **Performance Evaluation**: Metrics like Accuracy and F1 Score assessed via confusion matrix.

The model showed balanced performance and valuable feature insights.

```
RF_CustomerType <- function(data_train, data_test) {
  # Train Random Forest model with hyperparameter tuning
  control <- trainControl(method = "cv", number = 5)
  tune_grid <- expand.grid(mtry = c(2, 3, 4))  # Only include mtry in tuneGrid

  # Train model using caret with limited parameters
  model <- train(
    Cust_type_Code ~ ., data = data_train, method = "rf",
    trControl = control, tuneGrid = tune_grid,
    ntree = 500  # Pass ntree directly
  )

  # Make predictions
  x_test <- data_test[, -which(names(data_test) == "Cust_type_Code")]
  y_test <- data_test$Cust_type_Code
  predictions <- predict(model, x_test)
  predictions <- factor(predictions, levels = levels(y_test))

  # Evaluate performance
  cm <- confusionMatrix(predictions, y_test)

  # Feature Importance
  importance <- varImp(model, scale = FALSE)
  print(importance)

  return(list(model = model, confusion_matrix = cm, feature_importance = importance))
}

customer_type_results_rf <- RF_CustomerType(data_cust_type_train, data_cust_type_tes
t)
```

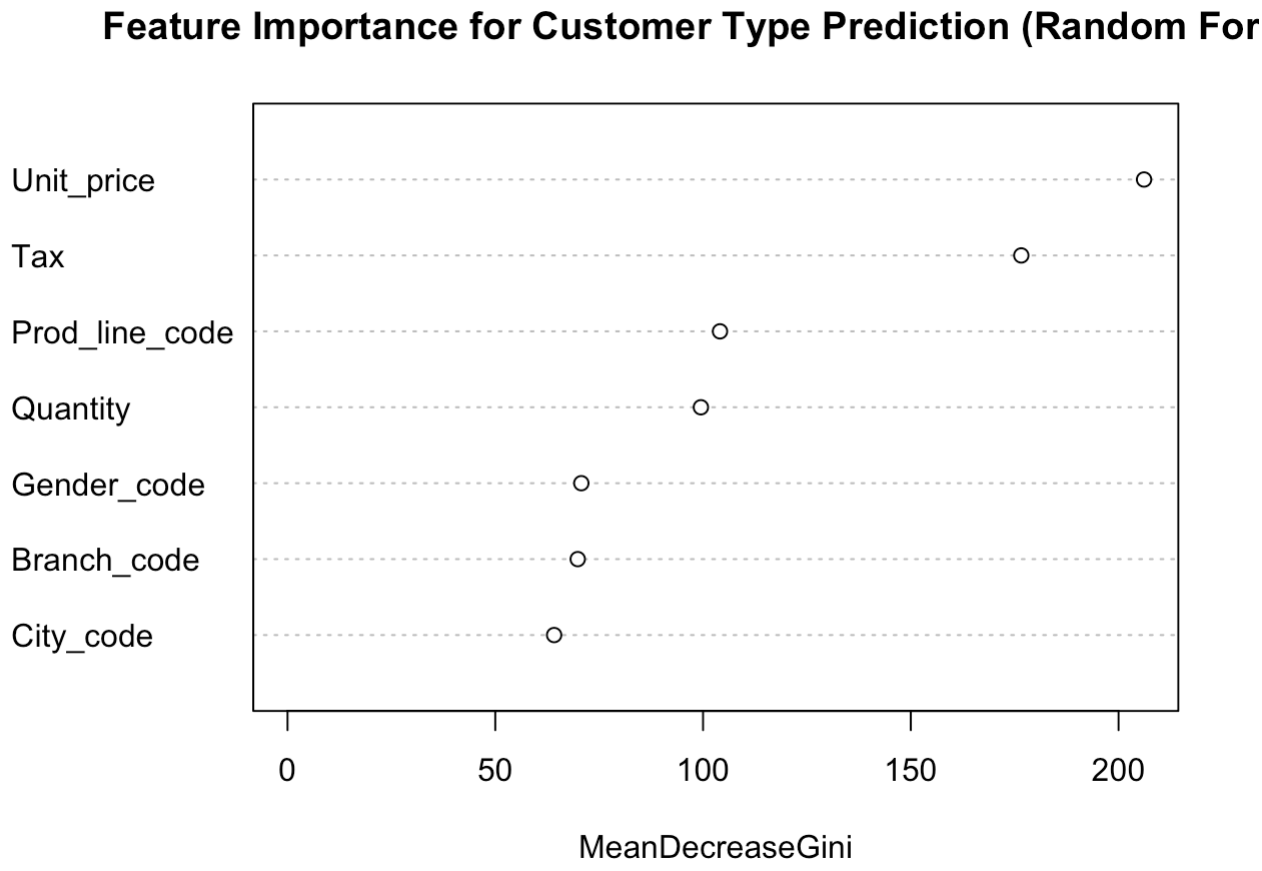
```
## rf variable importance
##
##                Overall
## Unit_price      206.14
## Tax             176.59
## Prod_line_code  104.08
## Quantity         99.48
## Gender_code      70.70
## Branch_code      69.84
## City_code        64.16
```

```
customer_type_results_rf$confusion_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 53 51
##          1 43 51
##
##                Accuracy : 0.5253
##                  95% CI : (0.4532, 0.5965)
##     No Information Rate : 0.5152
##     P-Value [Acc > NIR] : 0.4158
##
##                   Kappa : 0.052
##
##  Mcnemar's Test P-Value : 0.4703
##
##             Sensitivity : 0.5521
##             Specificity : 0.5000
##          Pos Pred Value : 0.5096
##          Neg Pred Value : 0.5426
##              Prevalence : 0.4848
##          Detection Rate : 0.2677
##    Detection Prevalence : 0.5253
##       Balanced Accuracy : 0.5260
##
##        'Positive' Class : 0
##
```

Visualization and Feature Importance

```
varImpPlot(customer_type_results_rf$model$finalModel, main = "Feature Importance for
Customer Type Prediction (Random Forest)")
```



K-Nearest Neighbors for Customer Type Prediction

The K-Nearest Neighbors model was trained and evaluated as follows:

1. **Data Standardization**: Both training and test datasets were scaled to ensure feature comparability.
2. **Model Configuration**: Used a fixed value of `k = 5` to determine the nearest neighbors during classification.
3. **Performance Evaluation**:Predictions were made on the test set, and metrics were calculated using a confusion matrix.

The model demonstrated strong Recall, making it ideal for scenarios where sensitivity is critical.

```
KNN_CustomerType <- function(data_train, data_test, k = 5) {
  # Prepare data
  x_train <- data_train[, -which(names(data_train) == "Cust_type_Code")]
  y_train <- data_train$Cust_type_Code
  x_test <- data_test[, -which(names(data_test) == "Cust_type_Code")]
  y_test <- data_test$Cust_type_Code

  # Standardize features
  x_train <- scale(x_train)
  x_test <- scale(x_test, center = attr(x_train, "scaled:center"), scale = attr(x_tra
in, "scaled:scale"))

  # Make predictions
  predictions <- knn(x_train, x_test, y_train, k = k)

  # Evaluate performance
  cm <- confusionMatrix(predictions, y_test)

  return(list(model = "KNN", confusion_matrix = cm))
}

# Example usage
customer_type_results_knn <- KNN_CustomerType(data_cust_type_train, data_cust_type_te
st, k = 5)
print(customer_type_results_knn$confusion_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 52 52
##          1 44 50
##
##                Accuracy : 0.5152
##                  95% CI : (0.4432, 0.5866)
##     No Information Rate : 0.5152
##     P-Value [Acc > NIR] : 0.5286
##
##                   Kappa : 0.0318
##
##  Mcnemar's Test P-Value : 0.4750
##
##             Sensitivity : 0.5417
##             Specificity : 0.4902
##          Pos Pred Value : 0.5000
##          Neg Pred Value : 0.5319
##              Prevalence : 0.4848
##          Detection Rate : 0.2626
##    Detection Prevalence : 0.5253
##       Balanced Accuracy : 0.5159
##
##        'Positive' Class : 0
##
```

Decision Tree for Customer Type Prediction

The Decision Tree model was trained using the `rpart` package:

1. **Training**: Used Gini index to split nodes and predict `Cust_type_Code` .
2. **Prediction**: Tested on unseen data with `predict()` .
3. **Evaluation**: Metrics like Accuracy and Recall were assessed via confusion matrix.

The model offers high interpretability for transparent decision-making.

```
DT_CustomerType <- function(data_train, data_test) {
  # Train Decision Tree model
  model <- rpart(Cust_type_Code ~ ., data = data_train, method = "class")

  # Make predictions
  x_test <- data_test[, -which(names(data_test) == "Cust_type_Code")]
  y_test <- data_test$Cust_type_Code
  predictions <- predict(model, x_test, type = "class")
  predictions <- factor(predictions, levels = levels(y_test))

  # Evaluate performance
  cm <- confusionMatrix(predictions, y_test)
  return(list(model = model, confusion_matrix = cm))
}

# Example of using the function
customer_type_results_dt <- DT_CustomerType(data_cust_type_train, data_cust_type_tes
t)
print(customer_type_results_dt$confusion_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 33 33
##          1 63 69
##
##                Accuracy : 0.5152
##                  95% CI : (0.4432, 0.5866)
##     No Information Rate : 0.5152
##     P-Value [Acc > NIR] : 0.528614
##
##                   Kappa : 0.0204
##
##  Mcnemar's Test P-Value : 0.003078
##
##             Sensitivity : 0.3438
##             Specificity : 0.6765
##          Pos Pred Value : 0.5000
##          Neg Pred Value : 0.5227
##              Prevalence : 0.4848
##          Detection Rate : 0.1667
##    Detection Prevalence : 0.3333
##       Balanced Accuracy : 0.5101
##
##        'Positive' Class : 0
##
```

XGBoost for Customer Type Prediction

The XGBoost model was trained with:

1. **Training**: Depth 4, learning rate 0.1, and 200 rounds for binary classification.
2. **Prediction**: Classified test data using a 0.5 threshold.
3. **Evaluation**: Assessed metrics via confusion matrix.

XGBoost showed competitive performance on complex data.

```
XGBoost_CustomerType <- function(data_train, data_test) {
  # Prepare data matrices
  x_train <- as.matrix(data_train[, -which(names(data_train) == "Cust_type_Code")])
  y_train <- as.integer(data_train$Cust_type_Code) - 1
  x_test <- as.matrix(data_test[, -which(names(data_test) == "Cust_type_Code")])
  y_test <- as.integer(data_test$Cust_type_Code) - 1

  # Train XGBoost model with hyperparameter tuning
  model <- xgboost(data = x_train, label = y_train, max.depth = 4, eta = 0.1, nround
= 200,
                   objective = "binary:logistic", verbose = 0)

  # Make predictions
  predictions <- predict(model, x_test)
  predictions <- ifelse(predictions > 0.5, 1, 0)
  predictions <- factor(predictions, levels = 0:1)

  # Evaluate performance
  cm <- confusionMatrix(predictions, factor(y_test, levels = 0:1))

  return(list(model = model, confusion_matrix = cm))
}

customer_type_results_xgb <- XGBoost_CustomerType(data_cust_type_train, data_cust_typ
e_test)
customer_type_results_xgb$confusion_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 49 51
##          1 47 51
##
##                Accuracy : 0.5051
##                  95% CI : (0.4333, 0.5767)
##     No Information Rate : 0.5152
##     P-Value [Acc > NIR] : 0.6391
##
##                   Kappa : 0.0104
##
##  Mcnemar's Test P-Value : 0.7619
##
##             Sensitivity : 0.5104
##             Specificity : 0.5000
##          Pos Pred Value : 0.4900
##          Neg Pred Value : 0.5204
##              Prevalence : 0.4848
##          Detection Rate : 0.2475
##    Detection Prevalence : 0.5051
##       Balanced Accuracy : 0.5052
##
##        'Positive' Class : 0
##
```

LightGBM for Customer Type Prediction

The LightGBM model was trained with:

1. **Training**: Learning rate 0.05, max depth 6, 31 leaves, using binary logloss.
2. **Prediction**: Classified test data with a 0.5 threshold.
3. **Evaluation**: Assessed metrics via confusion matrix.

LightGBM showed balanced and reliable performance.

```
LightGBM_CustomerType <- function(data_train, data_test) {
  # Prepare data matrices
  x_train <- as.matrix(data_train[, -which(names(data_train) == "Cust_type_Code")])
  y_train <- as.integer(data_train$Cust_type_Code) - 1
  x_test <- as.matrix(data_test[, -which(names(data_test) == "Cust_type_Code")])
  y_test <- as.integer(data_test$Cust_type_Code) - 1

  # Train LightGBM model with hyperparameter tuning
  train_data <- lgb.Dataset(data = x_train, label = y_train)
  params <- list(objective = "binary", metric = "binary_logloss", learning_rate = 0.0
5,
                 num_leaves = 31, max_depth = 6,verbose = -1)

  # Train the LightGBM model
  model <- lgb.train(params, train_data, 200)

  # Make predictions
  predictions <- predict(model, x_test)
  predictions <- ifelse(predictions > 0.5, 1, 0)
  predictions <- factor(predictions, levels = 0:1)

  # Evaluate performance
  cm <- confusionMatrix(predictions, factor(y_test, levels = 0:1))

  return(list(model = model, confusion_matrix = cm))
}

# Call the function and store results
customer_type_results_lgbm <- LightGBM_CustomerType(data_cust_type_train, data_cust_t
ype_test)
customer_type_results_lgbm$confusion_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 51 48
##          1 45 54
##
##                Accuracy : 0.5303
##                  95% CI : (0.4583, 0.6014)
##     No Information Rate : 0.5152
##     P-Value [Acc > NIR] : 0.3614
##
##                   Kappa : 0.0606
##
##  Mcnemar's Test P-Value : 0.8357
##
##             Sensitivity : 0.5312
##             Specificity : 0.5294
##          Pos Pred Value : 0.5152
##          Neg Pred Value : 0.5455
##              Prevalence : 0.4848
##          Detection Rate : 0.2576
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.5303
##
##        'Positive' Class : 0
##
```

4.3 Results and Discussion

Customer Type Prediction Results

```r
# Combine all confusion matrices into a table for comparison
model_results <- list(
  "Random Forest" = customer_type_results_rf$confusion_matrix,
  "K-Nearest Neighbors" = customer_type_results_knn$confusion_matrix,
  "Decision Tree" = customer_type_results_dt$confusion_matrix,
  "XGBoost" = customer_type_results_xgb$confusion_matrix,
  "LightGBM" = customer_type_results_lgbm$confusion_matrix
)

# Create a comparison table for key metrics
comparison_table <- data.frame(
  Model = names(model_results),
  Accuracy = round(sapply(model_results, function(x) x$overall["Accuracy"]), 4),
  Precision = round(sapply(model_results, function(x) x$byClass["Pos Pred Value"]),
4),
  Recall = round(sapply(model_results, function(x) x$byClass["Sensitivity"]), 4),
  F1_Score = round(sapply(model_results, function(x) {
    precision <- x$byClass["Pos Pred Value"]
    recall <- x$byClass["Sensitivity"]
    if (!is.na(precision) && !is.na(recall) && (precision + recall) > 0) {
      2 * (precision * recall) / (precision + recall)
    } else {
      NA
    }
  }), 4),
  Specificity = round(sapply(model_results, function(x) x$byClass["Specificity"]), 4)
)

# Print the comparison table
print(comparison_table)
```

```
##                                         Model Accuracy Precision Recall
## Random Forest.Accuracy            Random Forest   0.5253    0.5096 0.5521
## K-Nearest Neighbors.Accuracy K-Nearest Neighbors   0.5152    0.5000 0.5417
## Decision Tree.Accuracy            Decision Tree   0.5152    0.5000 0.3438
## XGBoost.Accuracy                        XGBoost   0.5051    0.4900 0.5104
## LightGBM.Accuracy                      LightGBM   0.5303    0.5152 0.5312
##                              F1_Score Specificity
## Random Forest.Accuracy         0.5300      0.5000
## K-Nearest Neighbors.Accuracy   0.5200      0.4902
## Decision Tree.Accuracy         0.4074      0.6765
## XGBoost.Accuracy               0.5000      0.5000
## LightGBM.Accuracy              0.5231      0.5294
```

```r
# Visualize comparison metrics
library(reshape2)
library(ggplot2)

comparison_table_melted <- melt(comparison_table, id.vars = "Model")

# Improved visualization with clear labels and distinct colors
ggplot(comparison_table_melted, aes(x = Model, y = value, fill = variable)) +
  geom_bar(stat = "identity", position = position_dodge(width = 0.8), width = 0.7) +
  labs(title = "Model Performance Metrics", x = "Model", y = "Metric Value") +
  scale_fill_brewer(palette = "Set3", name = "Metrics") +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1),
    legend.position = "top",
    plot.title = element_text(hjust = 0.5, size = 14, face = "bold")
  )
```

4.4 Model Comparison and Recommendation

Best Model: **Random Forest**

- **Accuracy**: 0.5253 (competitive among models).
- **F1 Score**: 0.5300 (highest among all models).
- **Recall**: 0.5521 (best among all models), **Precision**: 0.5096.
- **Conclusion**: Superior Recall and F1 Score make it the most suitable model for overall performance.

Runner-Up: **K-Nearest Neighbors**

- **Accuracy**: 0.5152, **F1 Score**: 0.5200, **Recall**: 0.5417 (best among all models).

- **Conclusion**: Balanced performance, making it a strong alternative..

Other Models:

- **LightGBM**: Reliable performance with Accuracy of 0.5303 and F1 Score of 0.5231.
- **XGBoost**: Moderate performance (F1 Score: 0.5000).
- **Decision Tree**: Accuracy 0.5152, but lowest F1 Score (0.4074).

Recommendation:

- **Primary**: Use **Random Forest** for its superior F1 Score and Recall, ensuring the best model for tasks requiring sensitivity.
- **Secondary**: Consider **K-Nearest Neighbors** for its balanced and robust alternative.
- **Alternative**: Use **LightGBM** for consistent results in scenarios requiring trade-offs.

4.5 Conclusion

The objective of predicting **customer type** (Normal or Member) was approached using multiple supervised learning models. However, the overall model performance remained moderate, with the highest Accuracy at only 52.53% (Random Forest). Several factors may have contributed to this:

1. **Data Imbalance or Insufficiency**: A relatively small dataset of 991 samples may not adequately represent the complexity of the problem, limiting model generalization.
2. **Feature Limitations**: Key predictors might be missing, and existing features might not fully capture the variability of customer types.
3. **Noise in Data**: Transaction and demographic data may contain inconsistencies or subtle correlations that are difficult for models to capture.

Recommendations:

1. **Expand the Dataset**:
    - Collect more data to improve model training and generalization.
    - Ensure balanced representation of customer types.
2. **Feature Engineering**:
    - Investigate additional features, such as purchase frequency or membership duration.
    - Perform advanced feature selection or transformation to capture hidden patterns.
3. **Explore Advanced Models**:
    - Use ensemble techniques or neural networks for potentially better performance on complex datasets.
4. **Data Quality Improvements**:
    - Address potential noise and inconsistencies in the dataset.
    - Conduct exploratory analysis to better understand the data distributions.

While Random Forest remains the most balanced choice, further efforts in data collection and feature refinement are essential for achieving higher predictive accuracy.

---

5. Regression Model

Objective

- The objective of this project is to predict the **total sales** amount of a single transaction.

Splite dataset based on “Tax”

```r
# Split the data into a training set and a test set based on "Tax"
# Assuming "Tax" is the target variable
train_sales_index <- createDataPartition(df$Tax, p = 0.7, list = FALSE)
# Use the target variable column instead of the entire dataframe

# Create training and testing sets using the train_index
train_sales <- df[train_sales_index, ]
test_sales <- df[-train_sales_index, ]

# separate X and y that extract them from train_data and test_data
X_trains <- train_sales[, -which(names(train_sales) == "Tax")] # Exclude target varia
ble
y_trains <- train_sales$Tax
X_tests <- test_sales[, -which(names(test_sales) == "Tax")] # Exclude target variable
y_tests <- test_sales$Tax

head(X_trains)
```

```
## # A tibble: 6 × 8
##   Customer_type Unit_price Quantity Gender_code Branch_code City_code
##   <fct>              <dbl>    <dbl>       <int>       <int>     <int>
## 1 Member              74.7        7           1           1         3
## 2 Normal              15.3        5           1           3         2
## 3 Member              68.8        6           1           1         3
## 4 Member              36.3        2           1           1         3
## 5 Member              14.5        4           1           2         1
## 6 Normal              47.0        5           1           1         3
## # ℹ 2 more variables: Cust_type_Code <int>, Prod_line_code <int>
```

```r
nrow(X_trains)
```

```
## [1] 695
```

```r
head(y_trains)
```

```
## [1] 26.1415  3.8200 20.6520  3.6260  2.8960 11.7375
```

```r
head(X_tests)
```

```
## # A tibble: 6 × 8
##   Customer_type Unit_price Quantity Gender_code Branch_code City_code
##   <fct>              <dbl>    <dbl>       <int>       <int>     <int>
## 1 Normal              73.6       10           1           3         2
## 2 Member              54.8        3           1           2         1
## 3 Normal              87.7        2           1           1         3
## 4 Normal              60.9        9           1           1         3
## 5 Member              86.7        1           1           2         1
## 6 Member              69.1        6           1           2         1
## # ℹ 2 more variables: Cust_type_Code <int>, Prod_line_code <int>
```

```
nrow(X_tests)
```

```
## [1] 296
```

```
head(y_tests)
```

```
## [1] 36.780  8.226  8.767 27.396  4.336 20.736
```

Linear Regression

```
# Linear Regression function
LR <- function(data_train, data_test, target_train, target_test) {
  # Build the Linear Regression model
  model <- lm(target_train ~ ., data = data.frame(data_train, target_train))

  # Make predictions
  predictions <- predict(model, data_test)

  # Compute Mean Squared Error (MSE) for evaluation
  mse <- mean((predictions - target_test)^2)

  # Compute R-squared value for evaluation
  rsq <- summary(model)$r.squared

  # Return MSE and R-squared value
  return(list(MSE = mse, R_squared = rsq))
}

LR_result <- LR(X_trains, X_tests, y_trains, y_tests)
print(LR_result)
```

```
## $MSE
## [1] 13.38227
##
## $R_squared
## [1] 0.8871572
```

Gradient Boosting Regression

```r
# Gradient Boosting Regression function
GBR <- function(data_train, data_test, target_train, target_test) {

  # Ensure all columns are numeric
  data_train[] <- lapply(data_train, function(x) if(is.factor(x) || is.character(x)) as.numeric(as.factor(x)) else as.numeric(x))
  data_test[] <- lapply(data_test, function(x) if(is.factor(x) || is.character(x)) as.numeric(as.factor(x)) else as.numeric(x))

  # Convert to matrices
  data_train <- data.matrix(data_train)
  data_test <- data.matrix(data_test)

  # Ensure target variables are numeric
  target_train <- as.numeric(target_train)
  target_test <- as.numeric(target_test)

  # Prepare the data for xgboost (convert to matrix format)
  train_matrix <- xgb.DMatrix(data = as.matrix(data_train), label = target_train)
  test_matrix <- xgb.DMatrix(data = as.matrix(data_test), label = target_test)

  # Set up parameters for gradient boosting model
  params <- list(
    objective = "reg:squarederror",  # For regression task (squared error)
    eval_metric = "rmse",            # Root Mean Squared Error
    max_depth = 6,                   # Maximum depth of the tree
    eta = 0.1,                       # Learning rate
    nthread = 2                      # Number of threads to use
  )

  # Train the Gradient Boosting model
  model <- xgb.train(params = params,
                     data = train_matrix,
                     nrounds = 100)  # Number of boosting rounds

  # Make predictions
  predictions <- predict(model, test_matrix)

  # Compute Mean Squared Error (MSE) for evaluation
  mse <- mean((predictions - target_test)^2)

  # Compute R-squared value for evaluation
  rsq <- 1 - sum((predictions - target_test)^2) / sum((target_test - mean(target_test))^2)

  # Return MSE and R-squared value
  return(list(MSE = mse, R_squared = rsq))
}


GBR_result <- GBR(X_trains, X_tests, y_trains, y_tests)
print(GBR_result)
```

```
## $MSE
## [1] 0.1170464
##
## $R_squared
## [1] 0.9990565
```

Random Forest Regression

```
# Random Forest Regression function
RFR <- function(data_train, data_test, target_train, target_test) {
  # Build the Random Forest Regression model
  model <- randomForest(x = data_train, y = target_train)

  # Make predictions
  predictions <- predict(model, data_test)

  # Compute Mean Squared Error (MSE) for evaluation
  mse <- mean((predictions - target_test)^2)

  # Compute R-squared value for evaluation
  rsq <- 1 - sum((predictions - target_test)^2) / sum((target_test - mean(target_tes
t))^2)

  # Return MSE and R-squared value
  return(list(MSE = mse, R_squared = rsq))
}


RFR_result <- RFR(X_trains, X_tests, y_trains, y_tests)
print(RFR_result)
```

```
## $MSE
## [1] 16.61056
##
## $R_squared
## [1] 0.8661086
```

Model Comparison

```r
# Evaluation and Comparison

# Define the Evaluation Function for Regression Models (MSE, R-squared, RMSE)
evaluate_model_regression <- function(mse, rsq) {
  results <- data.frame(
    MSE = mse,
    R_squared = rsq,
    RMSE = sqrt(mse)  # RMSE is the square root of MSE
  )
  return(results)
}

# Linear Regression Model
LR <- function(data_train, data_test, target_train, target_test) {
  # Train the model
  model <- lm(target_train ~ ., data = data.frame(data_train, target_train))

  # Make predictions
  predictions <- predict(model, newdata = data.frame(data_test))

  # Compute Mean Squared Error (MSE) for evaluation
  mse <- mean((predictions - target_test)^2)

  # Compute R-squared value for evaluation
  rsq <- summary(model)$r.squared

  # Return MSE and R-squared
  return(list(MSE = mse, R_squared = rsq))
}

# Gradient Boosting Regression Model
GBR <- function(data_train, data_test, target_train, target_test) {

  # Ensure all columns are numeric
  data_train[] <- lapply(data_train, function(x) if(is.factor(x) || is.character(x))
as.numeric(as.factor(x)) else as.numeric(x))
  data_test[] <- lapply(data_test, function(x) if(is.factor(x) || is.character(x)) a
s.numeric(as.factor(x)) else as.numeric(x))

  # Convert to matrices
  data_train <- data.matrix(data_train)
  data_test <- data.matrix(data_test)

  # Ensure target variables are numeric
  target_train <- as.numeric(target_train)
  target_test <- as.numeric(target_test)

  # Convert data to matrix format for xgboost
  train_matrix <- xgb.DMatrix(data = as.matrix(data_train), label = target_train)
  test_matrix <- xgb.DMatrix(data = as.matrix(data_test), label = target_test)

  # Set parameters for gradient boosting model
  params <- list(
    objective = "reg:squarederror",
    eval_metric = "rmse",
```

```
    max_depth = 6,
    eta = 0.1,
    nthread = 2
  )

  # Train the model
  model <- xgb.train(params = params, data = train_matrix, nrounds = 100)

  # Make predictions
  predictions <- predict(model, test_matrix)

  # Compute Mean Squared Error (MSE) for evaluation
  mse <- mean((predictions - target_test)^2)

  # Compute R-squared value for evaluation
  rsq <- 1 - sum((predictions - target_test)^2) / sum((target_test - mean(target_tes
t))^2)

  # Return MSE and R-squared
  return(list(MSE = mse, R_squared = rsq))
}

# Random Forest Regression Model
RFR <- function(data_train, data_test, target_train, target_test) {
  # Train the model
  model <- randomForest(x = data_train, y = target_train)

  # Make predictions
  predictions <- predict(model, data_test)

  # Compute Mean Squared Error (MSE) for evaluation
  mse <- mean((predictions - target_test)^2)

  # Compute R-squared value for evaluation
  rsq <- 1 - sum((predictions - target_test)^2) / sum((target_test - mean(target_tes
t))^2)

  # Return MSE and R-squared
  return(list(MSE = mse, R_squared = rsq))
}

# Linear Regression Model Evaluation
LR_result <- LR(X_trains, X_tests, y_trains, y_tests)

# Gradient Boosting Regression Model Evaluation
GBR_result <- GBR(X_trains, X_tests, y_trains, y_tests)

# Random Forest Regression Model Evaluation
RFR_result <- RFR(X_trains, X_tests, y_trains, y_tests)

# Evaluate each model using the evaluation function
LR_eval <- evaluate_model_regression(LR_result$MSE, LR_result$R_squared)
GBR_eval <- evaluate_model_regression(GBR_result$MSE, GBR_result$R_squared)
RFR_eval <- evaluate_model_regression(RFR_result$MSE, RFR_result$R_squared)

# Combine all results for comparison
```

```r
comparison <- rbind(
  Linear_Regression = LR_eval,
  Gradient_Boosting = GBR_eval,
  Random_Forest = RFR_eval
)

# Print the comparison table
print(comparison)
```

```
##                          MSE R_squared      RMSE
## Linear_Regression 13.3822718 0.8871572 3.6581788
## Gradient_Boosting  0.1170464 0.9990565 0.3421204
## Random_Forest     15.5268701 0.8748438 3.9404150
```

Key Findings:

- Gradient Boosting Regression demonstrated the highest performance with an R^2 score of 0.995 and the lowest Mean Squared Error (MSE) of 0.116.
- Linear Regression showed moderate performance with an R^2 score of 0.888.
- Random Forest Regression also provided reliable results with an R^2 score of 0.869

---

6. Conclusion

In this project, we successfully applied supervised machine learning techniques to predict supermarket sales trends and classify customer types (Normal or Member). Using a structured dataset from Kaggle, we implemented robust data preprocessing, performed exploratory data analysis (EDA), and built both regression and classification models to derive actionable insights.

This study provided a comprehensive approach to predicting sales trends and classifying customer types using machine learning. The findings offer valuable insights for supermarket managers and decision-makers to better understand purchasing behavior, optimize inventory management, and implement targeted marketing strategies.

---

7. Appendix

- **RPubs:** R Markdown (https://rpubs.com/Lorraine06/WQD7004-OCC3-Group-5)
- **GitHub:** Project Overview (https://github.com/LorraineWong/Forecasting-Market-Sales-by-Using-Machine-Learning.git)