JAIN (DEEMED-TO-BE UNIVERSITY)

School of Sciences

Sub: Data Cleaning

Activity: 1

Sub Code: 22MSDA3S04

Title:

Exploratory Data Analysis (EDA) for Sprocket Central Pty Ltd

Done By:

Gunjan Chakraborty

USN no:

22MSRDS007

Submission Date:

03/10/2023

Guide: M. A Ghouse Basha

Course: M.Sc. Data Science and Analytics

Semester: 3rd

INDEX

1. Abstract:

This project involves conducting an Exploratory Data Analysis (EDA) on the dataset provided from Sprocket Central Pty Ltd, a mid-sized bicycle company operating in Australia. The dataset consists of four sheets within an Excel workbook, namely "transactions," "new_customer_list," "customer_demographic," and "customer_address."

The primary objectives of this analysis are as follows:

1. *Data Gathering*: Collect and organize data from the provided Excel workbook, ensuring a comprehensive understanding of each dataset's structure and contents.
2. *Data Cleaning*: Identify and rectify any data quality issues such as missing values, duplicates, inconsistencies, and outliers, to ensure data integrity and reliability.
3. *Exploratory Data Analysis (EDA)*: Conduct a thorough exploration of the data to gain insights into customer demographics, transaction patterns, and customer addresses. This will involve summary statistics, data visualization, and correlation analysis.
4. *Graphical Representation*: Create informative visualizations, including but not limited to bar charts, histograms, scatter plots, and box plots, to visually represent key findings and patterns.
5. *Customer Segmentation*: Utilize customer demographic information to segment the customer base, which may inform targeted marketing strategies.
6. *Geographic Analysis*: Explore customer addresses to understand geographic distribution and identify potential areas for expansion or marketing focus.

Through this EDA process, we aim to provide Sprocket Central Pty Ltd with valuable insights into their customer base, transaction history, and geographic reach. These insights will serve as a foundation for data-driven decision-making, enabling the company to optimize its business strategies and enhance customer satisfaction.

2. Introduction

Sprocket Central Pty Ltd, a prominent mid-sized bicycle company operating in the vibrant Australian market, has embarked on a journey to enhance its business strategies and customer-centric approach. In an era where data-driven decision-making reigns supreme, understanding the intricacies of customer demographics, transaction patterns, and geographic reach is paramount to achieving sustainable growth and success.

This project aims to undertake a comprehensive Exploratory Data Analysis (EDA) of the data provided by Sprocket Central Pty Ltd. The dataset, comprising four distinct sheets within an Excel workbook, offers a wealth of information that, when analysed and visualized effectively, can provide invaluable insights.

The four key datasets within the Excel workbook are as follows:

1. *Transactions*: This sheet holds records of customer transactions, encompassing purchase details, dates, and quantities, offering a glimpse into purchasing behaviour and preferences.
2. *New Customer List*: This dataset contains information about newly acquired customers, serving as a foundation for understanding the growth of Sprocket Central's customer base.
3. *Customer Demographic*: This sheet encompasses essential demographic information about customers, including age, gender, and job titles, facilitating customer segmentation and targeted marketing strategies.

4. *Customer Address*: Geographic details of customers are recorded in this dataset, enabling an analysis of regional distribution and the identification of potential expansion opportunities.

Throughout this project, we will embark on a journey to unravel the hidden narratives concealed within the data. The objectives include data gathering, meticulous data cleaning to ensure data reliability, and the execution of EDA techniques to discover meaningful patterns and relationships. This process will culminate in the creation of informative visualizations that present insights in an easily understandable and actionable manner.

By the end of this EDA project, Sprocket Central Pty Ltd will be better equipped to make informed decisions, optimize its marketing strategies, tailor its products to customer preferences, and identify areas for growth and improvement. Ultimately, this initiative aligns with the company's commitment to providing quality bicycles and enhancing the overall cycling experience for its customers across Australia.

This report delves into the comprehensive data analysis processes and techniques employed to address data discrepancies, ensuring the integrity and readiness of the dataset. The utilization of various R packages and functions, including the **`tidyverse`, `dplyr`, `outliers`, `readxl`, `ggplot2`, `readr`, `gridExtra`, `reshape2`, `circlize`, `plotly`, `rgl`, `fmsb`,** and **`GGally`**, plays a pivotal role in this data analysis journey.

Data analysis, often regarded as a crucial step in making informed decisions, begins with the initial task of loading datasets. Here, we employ R's `readxl` package to seamlessly import data. The `tidyverse` package proves to be indispensable for our data manipulation needs, streamlining the process and ensuring data readiness.

One fundamental aspect of data analysis lies in the accuracy of column names, as they serve as the building blocks of our dataset. To address any misspelled or incorrect column names, we employ the `dplyr` function `rename () `, thereby preserving data integrity. Furthermore, we meticulously examine columns with measurements, standardizing units, if necessary, through conditional statements within `dplyr`. This critical step ensures consistency in data scale.

Handling missing values represents another integral facet of our data analysis process. Through meticulous examination, we identify columns with missing data and utilize `dplyr` to efficiently replace them with mean or median values, ensuring data completeness.

The presence of outliers can significantly impact the outcome of data analysis. To mitigate this impact, we visualize and address potential outliers using techniques such as box plots and leverage `dplyr` to decide whether to remove or transform them.

Effective data storytelling relies heavily on data visualization techniques. To this end, we employ plotting libraries like `ggplot2`, which empower us to create compelling graphs. From scatter plots to bar charts and specialized visualizations such as Parallel Coordinates Plots and Chord Diagrams, these tools enable us to convey data insights with precision, enhancing the interpretability of our findings.

In summary, this report showcases the application of a diverse set of tools and techniques to handle data discrepancies, ensuring data integrity, completeness, and visualization. The steps outlined here provide a solid foundation for robust data analysis, aiding in informed decision-making for organizations like Sprocket Central Pty Ltd.

3. Objective:

The primary objective of this data analysis project is to comprehensively investigate and prepare the dataset provided by Sprocket Central Pty Ltd, a mid-sized bicycle company in Australia. Our goals are as follows:

1. *Data Gathering*: Gather and import the data from multiple sheets within an Excel workbook, including "transactions," "new_customer_list," "customer_demographic," and "customer_address," using R's `readxl` functions.

2. *Data Cleaning and Data Readiness*:
    a. Ensure data integrity by rectifying any misspelled or incorrect column names using the `dplyr` function `rename () `.
    b. Standardize measurement units within the dataset where necessary, maintaining data consistency through conditional statements in `dplyr`.
    c. Handle missing values efficiently using `dplyr` to replace them with appropriate values (e.g., mean or median), ensuring data completeness.

3. *Outlier Detection and Treatment*:
    a. Identify potential outliers within the data through visualization techniques such as box plots.
    b. Apply data manipulation with `dplyr` to address outliers, either through removal or transformation, as required.

4. *Exploratory Data Analysis (EDA):*
    a. Conduct a comprehensive exploration of the data to gain insights into customer demographics, transaction patterns, and customer addresses.
    b. Utilize descriptive statistics and visualization tools, including `ggplot2`, to summarize and visually represent key findings.

5. *Data Visualization:*
    a. Create informative and visually appealing graphs and plots using `ggplot2`, `plotly`, and other relevant packages.
    b. Utilize various visualization types such as scatter plots, bar charts, Parallel Coordinates Plots, and Chord Diagrams to convey data insights effectively.

6. *Insights for Decision-Making:*
    a. Generate actionable insights from the EDA and visualization results that can inform business strategies, marketing approaches, and geographic expansion opportunities for Sprocket Central Pty Ltd.

By achieving these objectives, this data analysis project aims to equip Sprocket Central Pty Ltd with a deeper understanding of their customer base, transaction history, and geographic reach. These insights will serve as a foundation for data-driven decision-making and ultimately contribute to the company's growth and success.

4. Data Description:

The dataset provided for the analysis consists of four separate sheets, each with specific columns. Here is a description of the columns in each sheet:

Transactions Sheet:

1. transaction_id: A unique identifier for each transaction.
2. product_id: A unique identifier for each product purchased.
3. customer_id: A unique identifier for each customer.
4. transaction_date: The date of the transaction.
5. online_order: Indicates whether the order was placed online (Yes/No).
6. order_status: The status of the order (e.g., "Approved," "Cancelled").
7. brand: The brand of the product.
8. product_line: The product line (e.g., "Standard", "Road").
9. product_class: The product class (e.g., "Medium", "High").
10. product_size: The size of the product.
11. list_price: The list price of the product.
12. standard_cost: The standard cost of the product.

New Customer List Sheet:

1. first_name: The customer's first name.
2. last_name: The customer's last name.
3. gender: The customer's gender.
4. bike_related_purchases: The number of bicycle-related purchases made by the customer.
5. DOB: The customer's date of birth.
6. job_title: The customer's job title.
7. job_industry_category: The category of the customer's job industry.
8. wealth_segment: The wealth segment to which the customer belongs.
9. deceased_indicator: Indicates whether the customer is deceased (Yes/No).
10. owns_car: Indicates whether the customer owns a car (Yes/No).
11. tenure: The tenure of the customer (length of time as a customer).
12. address: The customer's address.
13. postcode: The customer's postal code.
14. state: The state in which the customer resides.
15. country: The country in which the customer resides.
16. property_valuation: The property valuation for the customer's residence.
17. Rank: A customer's rank.
18. Value: A customer's value.
19. year: The year associated with the data.

Customer Demographic Sheet:

1. customer_id: A unique identifier for each customer.
2. first_name: The customer's first name.
3. last_name: The customer's last name.
4. gender: The customer's gender.
5. bike_related_purchases: The number of bicycle-related purchases made by the customer.
6. DOB: The customer's date of birth.
7. job_title: The customer's job title.
8. job_industry_category: The category of the customer's job industry.
9. wealth_segment: The wealth segment to which the customer belongs.
10. deceased_indicator: Indicates whether the customer is deceased (Yes/No).
11. owns_car: Indicates whether the customer owns a car (Yes/No).
12. tenure: The tenure of the customer (length of time as a customer).
13. age: The age of the customer (derived from DOB).

Customer Address Sheet:

1. customer_id: A unique identifier for each customer.
2. address: The customer's address.
3. postal_code: The customer's postal code.
4. state: The state in which the customer resides.
5. country: The country in which the customer resides.
6. property_valuation: The property valuation for the customer's residence.

These datasets collectively provide valuable information about transactions, customer demographics, and customer addresses. This data will be analysed, cleaned, and visualized to derive meaningful insights for business decision-making.

5. Methodology:

The methodology for conducting a comprehensive exploratory data analysis (EDA) on the Sprocket Central Pty Ltd dataset involves several key steps and techniques. These steps are designed to ensure data quality, uncover insights, and support informed decision-making. The primary tools and libraries used include `tidyverse`, `dplyr`, `outliers`, `readxl`, `ggplot2`, `readr`, `gridExtra`, `reshape2`, `circlize`, `plotly`, `rgl`, `fmsb`, and `GGally`. Here's a detailed breakdown of the methodology:

I. Data Importation and Data Inspection:

a. Use the `readxl` package to import data from the provided Excel workbook into R.

Code:

```
# Specify the file path
file_path <- "D:/DC_mini/Sprocket Central Pty Ltd Data.xlsx"

# Read the Excel file with all sheets into a list of data frames
all_data <- excel_sheets(file_path) %>%
  set_names() %>%
  map(~ read_excel(file_path, sheet = .x))

# Assign each data frame to a variable with its respective name
transactions <- all_data$Transactions
new_customer_list <- all_data$NewCustomerList
customer_demographic <- all_data$CustomerDemographic
customer_address <- all_data$CustomerAddress
```

b. Inspect the imported datasets to gain a preliminary understanding of their structure and contents.

Code:

```
# Check the Dimension, Column names, data type and Structure of the data set
#================================================================================

glimpse(transactions)
glimpse(new_customer_list)
glimpse(customer_demographic)
glimpse(customer_address)
```

II. Data Cleaning and Preparation:

a. Address data quality issues, including missing values and duplicate entries using functions from the `tidyverse` and `outliers` packages.

Code:

```
# Remove specific columns from the 'transactions' data frame
transactions <- transactions %>%
  select(-product_first_sold_date)

# Remove specific columns from the 'new_customer_list' data frame
new_customer_list <- new_customer_list[, -c(17, 18, 19, 20, 21)]

# Remove specific columns from the 'customer_demographic' data frame
customer_demographic <- customer_demographic %>%
  select(-default)
# Check for duplicate rows in the data frame
#============================================

has_duplicates <- any(duplicated(transactions))
has_duplicates
# Count null values for all columns in the 'transactions' data frame
null_counts <- colSums(is.na(transactions))
print(null_counts)

# Replacing Misspellings in gender of 'customer_demographic' data frame
customer_demographic <- customer_demographic %>%
  mutate(gender = ifelse(gender %in% c("Femal", "F"), "Female", gender))

customer_demographic <- customer_demographic %>%
  mutate(gender = ifelse(gender %in% c("M"), "Male", gender))
```

b. Ensure consistency in column names and data types.

Code:

```
#changing date from numeric to dttm format
new_customer_list$DOB <- as.POSIXct(new_customer_list$DOB, format = "%Y-%m-%d")
customer_demographic$DOB <- as.Date("1900-01-01") +
  as.numeric(customer_demographic$DOB) - 1
customer_demographic$DOB

# Change the name of past_3_years_bike_related_purchases of customer_demographic
names(customer_demographic)[names(customer_demographic) ==
    "past_3_years_bike_related_purchases"] <- "bike_related_purchases"

# Change the name of a specific column
names(customer_address)[names(customer_address) == "postcode"] <- "postal_code"

# Check for Unique values
#==========================

# 'transactions' is the data frame
for (col in names(transactions)) {
  unique_values <- unique(transactions[[col]])
  num_unique <- length(unique_values)
  cat("Column:", col, "has", num_unique, "unique values.\n")
}

# Define a vector of column names to exclude from displaying unique values
columns_to_exclude <- c("transaction_id", "product_id", "customer_id",
                        "transaction_date", "list_price", "standard_cost")


for (col in names(transactions)) {
  # Check if the column is not in the 'columns_to_exclude' list
  if (!(col %in% columns_to_exclude)) {
    unique_values <- unique(transactions[[col]])
    cat("Unique values in column '", col, "':\n")
    print(unique_values)
    cat("\n")
  }
}
check_consistency <- function(data, ignore_columns = NULL) {
  cat("Inconsistent columns:\n")

  # Select only character and factor columns
  char_columns <- data %>%
    select_if(is.character) %>%
    names()

  # Exclude columns specified in 'ignore_columns'
  if (!is.null(ignore_columns)) {
    char_columns <- setdiff(char_columns, ignore_columns)
  }

  for (col in char_columns) {
    cat_count <- data %>%
      group_by({{ col }}) %>%
      summarise(n = n()) %>%
      ungroup()

    if (nrow(cat_count) <= 1) {
      cat("  - ", col, "\n")
    }
  }

  cat("Other columns are consistent.\n")
}
```

*III. **Handling Missing Data***:

a. Identify columns with missing data and make informed decisions about handling missing values using `dplyr`. Options may include imputing with mean or median values.

Code:

```
# Remove rows with any NA values from the data frame 'transactions'
transactions <- na.omit(transactions)
```

IV. ***Outlier Detection and Treatment:***

a. Detect potential outliers using box plots and data visualization techniques.

b. Decide on an appropriate strategy for dealing with outliers (e.g., removal, transformation) using `dplyr` and the `outliers` package.

```
# Define a function to detect outliers using the IQR method for multiple columns
detect_outliers <- function(data, column_names) {
  # Initialize an empty data frame to store outliers
  all_outliers <- data.frame()

  for (col in column_names) {
    # Check if the specified column contains missing values
    if (any(is.na(data[[col]]))) {
      cat("Column", col, "contains missing values.Outlier detection skipped.\n")
    } else {
      # Calculate the first quartile (Q1) and third quartile (Q3)
      q1 <- quantile(data[[col]], 0.25)
      q3 <- quantile(data[[col]], 0.75)

      # Calculate the IQR (Interquartile Range)
      iqr <- q3 - q1

      # Calculate the lower and upper bounds for outliers
      lower_bound <- q1 - 1.5 * iqr
      upper_bound <- q3 + 1.5 * iqr

      # Identify outliers using the bounds for the current column
      outliers <- data %>%
        filter(data[[col]] < lower_bound | data[[col]] > upper_bound)

      # Add the outliers for the current column to the result
      all_outliers <- bind_rows(all_outliers, outliers)
    }
  }

  return(all_outliers)
}

# select the columns which we want to detect outlier
outliers_data <- detect_outliers(transactions, c('list_price', 'standard_cost'))

# Print the outliers
if (nrow(outliers_data) > 0) {
  print(outliers_data)
}
```

Output:

transactions sheet had 189 outliers which were removed

new_customer_list sheet had 25 outliers which were removed

V. ***Data Visualization***:

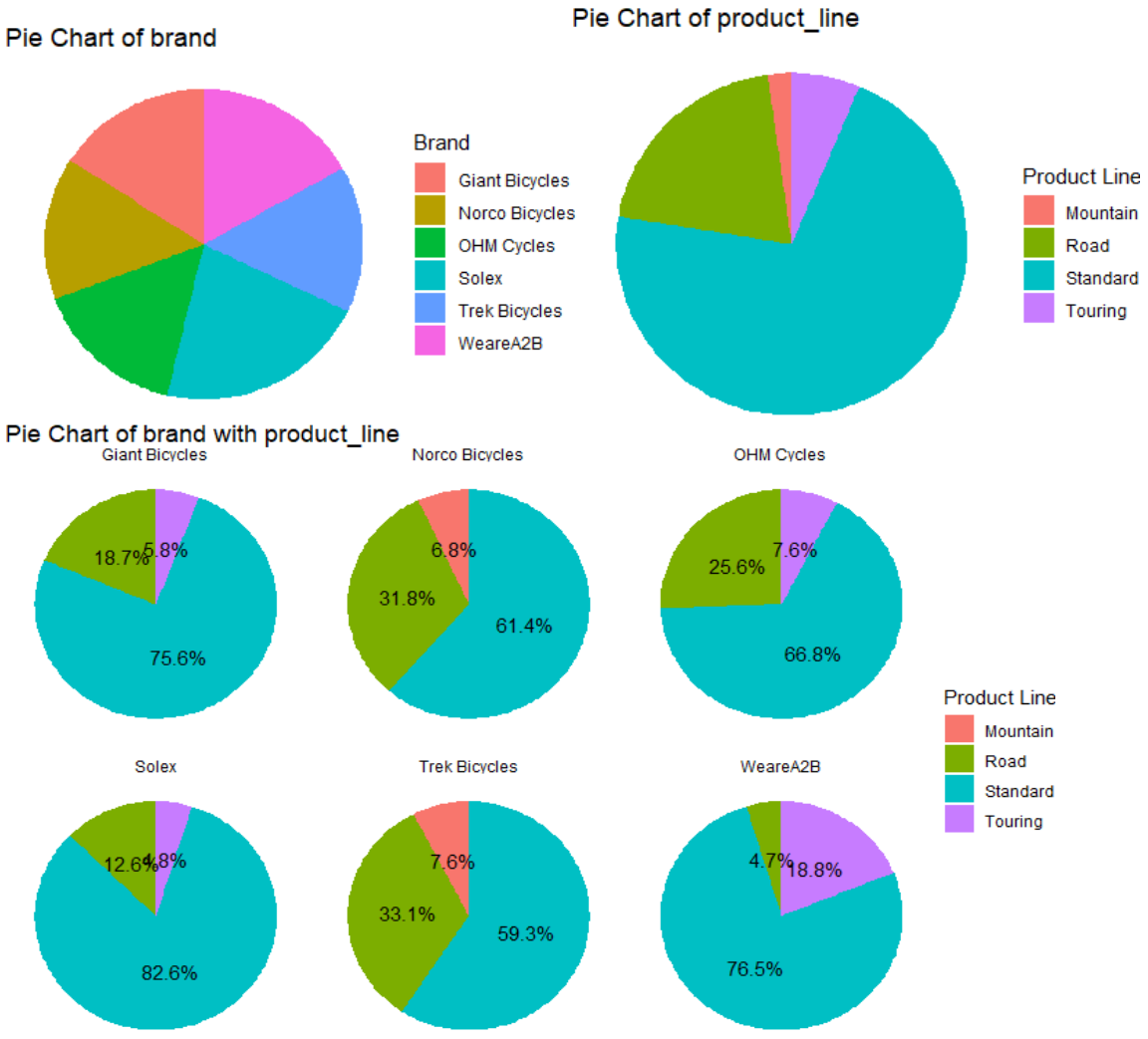
a. Create informative and visually appealing graphs and plots using `ggplot2`, `plotly`, and other relevant packages.

b. Utilize various visualization types, including scatter plots, bar charts, Parallel Coordinates Plots, Chord Diagrams to communicate data insights effectively.

```
# Pie charts for 'brand' and 'product_line' Combined
#=====================================================

brand_pie <- ggplot(data = transactions, aes(x = 1, fill = brand)) +
  geom_bar(width = 1) +
  coord_polar(theta = "y") +  # Convert to pie chart
  labs(
    title = "Pie Chart of brand",
    fill = "Brand"
  ) +
  theme_void()  # Remove axis and labels


product_line_pie <- ggplot(data = transactions, aes(x = 1, fill = product_line)
                           )+geom_bar(width = 1) +coord_polar(theta = "y") +
  # Convert to pie chart
  labs(
    title = "Pie Chart of product_line",
    fill = "Product Line"
  ) +
  theme_void()  # Remove axis and labels
```

```r
product_line_pie <- ggplot(data = transactions, aes(x = 1, fill = product_line)
                           )+geom_bar(width = 1) +coord_polar(theta = "y") +
  # Convert to pie chart
  labs(
    title = "Pie Chart of product_line",
    fill = "Product Line"
  ) +
  theme_void()  # Remove axis and labels


# Display the pie charts side by side
grid.arrange(brand_pie, product_line_pie, ncol = 2)

# Calculate percentages within each facet
abcd <- transactions %>%
  group_by(brand, product_line) %>%
  summarise(count = n()) %>%
  mutate(percentage = count / sum(count) * 100)

# Create pie chart for 'brand' with 'product_line' facets
brand_product_pie <- ggplot(data = abcd, aes(x = 1, fill = product_line, y =
                                               percentage)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar(theta = "y") +  # Convert to pie chart
  labs(
    title = "Pie Chart of brand with product_line",
    fill = "Product Line",
    y = NULL
  ) +
  theme_void() +  # Remove axis and labels
  facet_wrap(~ brand) +  # Facet by 'brand'
  geom_text(aes(label = paste0(round(percentage, 1), "%")),
            position = position_stack(vjust = 0.5),
            show.legend = FALSE)  # Hide the legend

# Display the pie charts with facets
print(brand_product_pie)
```

output:



This methodology aims to extract meaningful insights from the dataset and support data-driven decision-making for Sprocket Central Pty Ltd. It encompasses data cleaning, exploratory analysis, visualization, and segmentation techniques to uncover patterns and opportunities within the data.

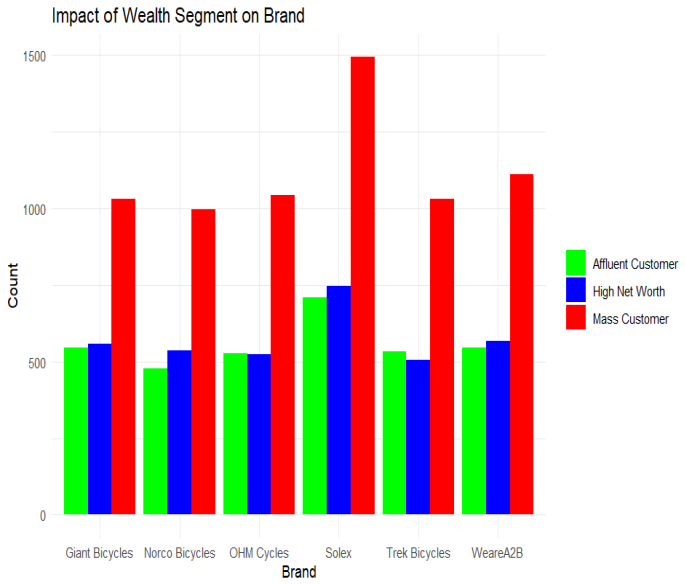
6. Exploratory Data Analysis (EDA):

In the process of Exploratory Data Analysis (EDA), various data visualization techniques were employed to gain insights into the Sprocket Central Pty Ltd dataset. This section outlines the key types of visualizations used, including multiple bar graphs, histograms, pie charts, and scatterplots, to uncover patterns and trends within the data:

1. Multiple Bar Graphs:

- Multiple bar graphs were created to visualize categorical data and compare different categories within a single plot.
- Examples include comparing the distribution of wealth segments and brands.

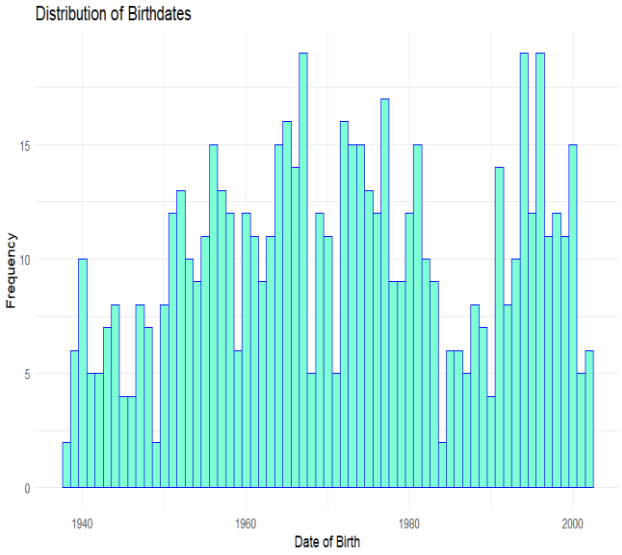
```
# Impact of Wealth Segment on Brand
#==================================

# Create a bar chart
ggplot(merged_data, aes(x = brand, fill = wealth_segment)) +
  geom_bar(position = "dodge") +
  labs(
    title = "Impact of Wealth Segment on Brand",
    x = "Brand",
    y = "Count"
  ) +
  scale_fill_manual(values = c("High Net Worth" = "blue", "Mass Customer" =
                               "red", "Affluent Customer" = "green")) +
  theme_minimal() +
  theme(legend.title = element_blank())  # Remove legend title
```



2. Histograms:

- Histograms were generated to explore the distribution of numerical variables, such as list prices, standard costs, and customer ages.
- These histograms allowed for an understanding of the central tendencies, variances, and skewness in the data.

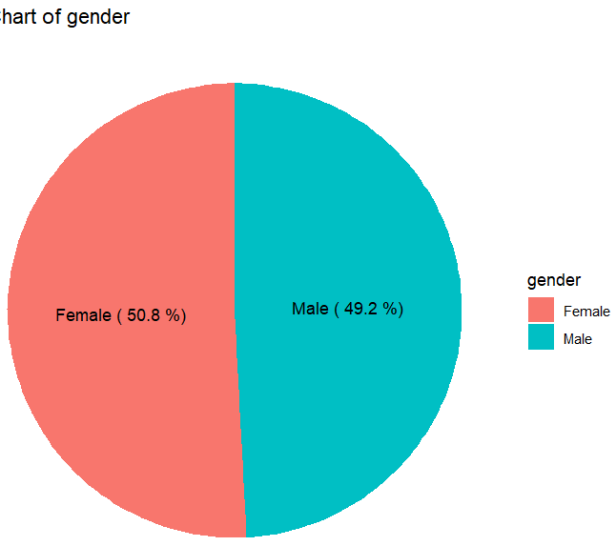
```
# Histogram for 'DOB' year
#=========================

ggplot(data = new_customer_list, aes(x = as.Date(DOB), y = ..count..)) +
  geom_histogram(binwidth = 365, fill = "aquamarine", color = "blue") +
  labs(
    title = "Distribution of Birthdates",
    x = "Date of Birth",
    y = "Frequency"
  ) +
  theme_minimal()
```



3. Pie Charts:

- Pie charts were employed to visualize the distribution of categorical data as a proportion of the whole.
- Examples include pie charts representing the gender distribution of customers or the distribution of customers in different job industry categories.
- Pie charts provided a clear overview of the composition of various categorical variables.

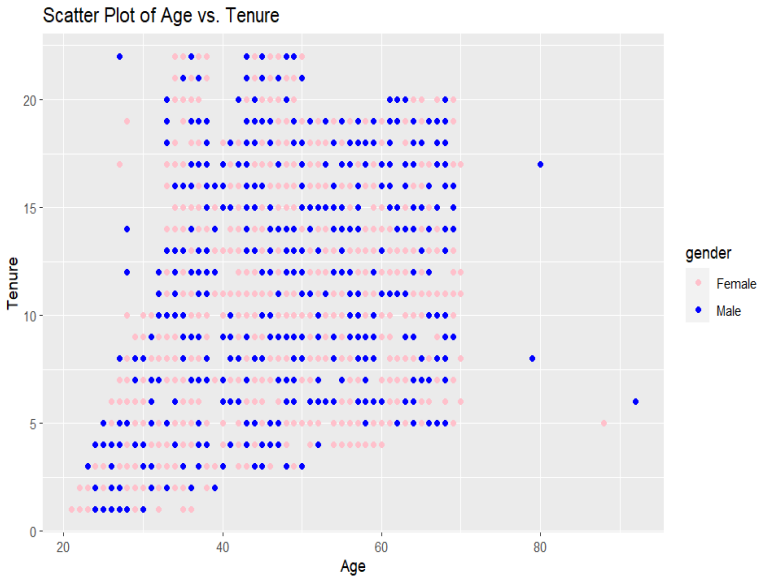
```
# Calculate the count and percentage of each gender
#==================================================

order_status_summary <- new_customer_list %>%
  group_by(gender) %>%
  summarize(count = n()) %>%
  mutate(percentage = (count / sum(count)) * 100)


# Create a pie chart
#====================

ggplot(order_status_summary, aes(x = "", y = count, fill = gender)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar(theta = "y") +
  geom_text(aes(label = paste(gender, "(", round(percentage, 1), "%)")),
            position = position_stack(vjust = 0.5)) +
  labs(
    title = "Pie Chart of gender",
    x = NULL,
    y = NULL,
    fill = "gender"
  ) +
  theme_void() +
  theme(legend.position = "right")
```



4. Scatterplots:

- Scatterplots were created to visualize the relationships between two numerical variables.
- Examples include scatterplots depicting the relationship between list prices and standard costs or between customer ages and tenure.
- Scatterplots were used to identify correlations, patterns, and potential outliers within the data.

```
# Scatter Plot of Age vs. Tenure
#================================

ggplot(merged_data, aes(x = age, y = tenure, color = gender)) +
  geom_point() +
  labs(
    title = "Scatter Plot of Age vs. Tenure",
    x = "Age",
    y = "Tenure"
  ) +
  scale_color_manual(values = c("Male" = "blue", "Female" = "pink"))
```



These visualization techniques played a crucial role in the EDA process by providing a visual representation of the dataset's characteristics and relationships. They allowed for the identification of trends, anomalies, and potential areas of interest. The insights gained from these visualizations informed subsequent data cleaning, data transformation, and segmentation steps and provided valuable information for data-driven decision-making for Sprocket Central Pty Ltd.

7. Code Explain:

1. `**glimpse(transactions)**`:
- This code snippet uses the `glimpse()` function to provide a concise overview of the 'transactions' dataset. It displays the data types of each column and a few sample rows, helping you quickly understand the dataset's structure and contents.

2. Remove Specific Columns:
[**transactions <- transactions %>%**
select(-product_first_sold_date)]

- The code snippet removes the column 'product_first_sold_date' from the 'transactions' data frame using the `%>%` operator from the `dplyr` package. This is done to eliminate a column that may not be relevant for the analysis.

3. Change Column Name:
[**names(customer_address)[names(customer_address) == "postcode"] <- "postal_code"**]

- This code renames the column 'postcode' to 'postal_code' in the 'customer_address' data frame using the `names()` function. It ensures consistency in column naming for further analysis.

4. Date Format Conversion:
[**new_customer_list$DOB <- as.POSIXct(new_customer_list$DOB, format = "%Y-%m-%d")**
customer_demographic$DOB <- as.Date("1900-01-01") +
as.numeric(customer_demographic$DOB) - 1
customer_demographic$DOB]

- The code converts the 'DOB' (Date of Birth) column in the 'new_customer_list' and 'customer_demographic' data frames to a consistent date format. It uses the `as.POSIXct()` and `as.Date()` functions to ensure the 'DOB' column is in a date format for analysis.

5. Check for Duplicates:
- The code snippet checks for duplicated rows in the 'transactions' data frame using the `duplicated()` function. It sets the variable '**has_duplicates**' to `TRUE` if duplicates are found, indicating potential data quality issues.

6. Define a Function for Checking Categorical Consistency:
- A custom function called `**check_consistency**` is defined to check the consistency of categorical columns in a given data frame. It identifies columns with inconsistent values and prints a list of such columns.

7. Consistency Check for 'transactions' Data Frame:
- The `check_consistency` function is applied to the 'transactions' data frame, with the 'transaction_date' column specified as an exception (ignored). It checks the consistency of other categorical columns.

8. Count Null Values:
- The code calculates the number of null (NA) values for each column in the 'transactions' data frame using `**colSums(is.na(transactions))**` and stores the result in 'null_counts'.

9. Remove Rows with NA Values:

- The code snippet removes rows with any NA values from the 'transactions' data frame using `**na.omit(transactions)**`. This step ensures that rows with missing data are excluded from further analysis.

10. Count Unique Values:

```
[for (col in names(transactions)) {
  unique_values <- unique(transactions[[col]])
  num_unique <- length(unique_values)
  cat("Column:", col, "has", num_unique, "unique values.\n")}]
```

- This code counts the number of unique values for each column in the 'transactions' data frame and prints the count for each column.

11. Display Unique Values (Excluding Specific Columns):

```
[columns_to_exclude <- c("transaction_id", "product_id", "customer_id",
                         "transaction_date", "list_price", "standard_cost")
for (col in names(transactions)) {
  # Check if the column is not in the 'columns_to_exclude' list
  if (!(col %in% columns_to_exclude)) {
    unique_values <- unique(transactions[[col]])
    cat("Unique values in column '", col, "':\n")
    print(unique_values)
    cat("\n")}}]
```

- The code displays unique values for each column in the 'transactions' data frame, excluding specified columns ('transaction_id', 'product_id', etc.). It provides insights into the diversity of values in each column.

12. Replace Misspellings in Gender:

```
[customer_demographic <- customer_demographic %>%
  mutate(gender = ifelse(gender %in% c("Femal", "F"), "Female", gender))]
```

- This code snippet corrects misspellings in the 'gender' column of the 'customer_demographic' data frame using the `%>%` operator and mutate function from the `dplyr` package. The ifelse() function is applied to the 'gender' column to replace any occurrences of "Femal" or "F" with "Female," ensuring consistency in gender labels. It replaces 'Femal' and 'F' with 'Female' to ensure consistent gender labels.

13. Define a Function for Outlier Detection (IQR Method):

```
[detect_outliers <- function(data, column_names) {
  all_outliers <- data.frame()
    for (col in column_names) {
    # Check if the specified column contains missing values
    if (any(is.na(data[[col]]))) {
      cat("Column", col, "contains missing values.Outlier detection skipped.\n")
    } else {
      # Calculate the first quartile (Q1) and third quartile (Q3)
      q1 <- quantile(data[[col]], 0.25)
      q3 <- quantile(data[[col]], 0.75)
      iqr <- q3 - q1
      lower_bound <- q1 - 1.5 * iqr
      upper_bound <- q3 + 1.5 * iqr
      outliers <- data %>%
        filter(data[[col]] < lower_bound | data[[col]] > upper_bound)
```

 # Add the outliers for the current column to the result
all_outliers <- bind_rows(all_outliers, outliers)
}}
return(all_outliers)}]

- A custom function called `detect_outliers` is defined to detect outliers using the Interquartile Range (IQR) method for one or more specified columns in a given data frame. This code defines a custom function named detect_outliers to detect outliers using the IQR method for multiple specified columns in a given data frame (data).

The function takes two arguments: the data frame to analyse and a vector of column_names specifying which columns to analyse for outliers. It initializes an empty data frame all_outliers to store the outlier observations. For each specified column, the function checks if the column contains missing values (NA). If it does, outlier detection for that column is skipped, and a message is printed. If the column does not contain missing values, the function calculates the first quartile (Q1), third quartile (Q3), and the IQR. The lower and upper bounds for outliers are calculated as Q1 - 1.5 * IQR and Q3 + 1.5 * IQR, respectively. Outliers are identified using these bounds for each specified column, and the results are added to the all_outliers data frame. Finally, the function returns the all_outliers data frame containing the identified outliers.

14. Detect Outliers:

[**outliers_data <- detect_outliers(transactions, c('list_price', 'standard_cost'))**
if (nrow(outliers_data) > 0) {
print(outliers_data)}]

- The `detect_outliers` function is applied to the 'transactions' data frame to detect outliers in the 'list_price' and 'standard_cost' columns. If outliers are found (i.e., if the number of rows in outliers_data is greater than 0), it prints the details of the outliers, including the values that fall outside the calculated bounds for each specified column.

Let's see the graphs

1. ***[These code sections illustrate the creation of a box plot to visualize the distribution of 'standard_cost' in the 'transactions' data frame and demonstrate the removal of outliers to create a cleaner and more informative box plot]***

Code:

```
ggplot(transactions, aes(y = standard_cost)) +
  geom_boxplot(fill = "skyblue", color = "darkblue", size = 1.2) +
  theme_minimal() +
  labs(
    title = "Distribution of standard_cost",
    x = NULL,
    y = "standard_cost",
    caption = "Source: Gunjan"
  ) +
  theme(
    plot.title = element_text(size = 18, face = "bold"),
    axis.text.x = element_blank(),  # Hide x-axis labels
    axis.text.y = element_text(size = 12),
    axis.title.y = element_text(size = 14),
    legend.position = "none"  # Remove legend if not needed
  ) +
  scale_y_continuous(labels = scales::comma)
```

Explain:
This code uses ggplot2 to create a box plot to visualize the distribution of 'standard_cost' in the 'transactions' data frame. The aes(y = standard_cost) part specifies that 'standard_cost' should be plotted on the y-axis. The geom_boxplot() function generates the box plot, with customization options such as fill color, border color, and line size. theme_minimal() is used to apply a minimalistic theme to the plot.
The labs() function sets the plot title, y-axis label, and caption. Various theme() settings control the appearance of the plot, including font sizes and axis label visibility. scale_y_continuous() is used to format the y-axis labels using the scales::comma function, which adds commas to large numbers for improved readability.

Code:

```
# Calculate the quartiles and IQR
q1 <- quantile(transactions$standard_cost, 0.25)
q3 <- quantile(transactions$standard_cost, 0.75)
iqr <- q3 - q1
# Calculate the lower and upper bounds for outliers
lower_bound <- q1 - 1.5 * iqr
upper_bound <- q3 + 1.5 * iqr
# Identify outliers
outliers <- transactions$standard_cost < lower_bound | transactions$standard_cost > upper_bound
# Create a new data frame without outliers
transactions <- transactions[!outliers, ]
# Create a boxplot without outliers
ggplot(transactions, aes(y = standard_cost)) +
  geom_boxplot(fill = "skyblue", color = "darkblue", size = 1.2) +
  theme_minimal() +
  labs(
    title = "Distribution of standard_cost (Outliers Removed)",
    x = NULL,
    y = "standard_cost",
    caption = "Gunjan"
  ) +
  theme(
    plot.title = element_text(size = 18, face = "bold"),
    axis.text.x = element_blank(),
    axis.text.y = element_text(size = 12),
    axis.title.y = element_text(size = 14),
    legend.position = "none"
  ) +
  scale_y_continuous(labels = scales::comma)
```

Explain:
This code calculates the quartiles (Q1 and Q3) and the Interquartile Range (IQR) for the 'standard_cost' column in the 'transactions' data frame. It then calculates the lower and upper bounds for outliers based on the IQR. Outliers are identified by checking if 'standard_cost' falls below the lower bound or above the upper bound. Rows containing outliers are removed from the 'transactions' data frame, resulting in a new data frame without outliers. A second box plot is created for the 'standard_cost' in the cleaned 'transactions' data frame, similar to the initial box plot but with outliers removed. The plot title, y-axis label, and caption are adjusted to reflect the removal of outliers.

2. ***[This code creates an informative pie chart that visualizes the distribution of 'order_status' categories and their respective percentages.]***

Code:

```
# Grouping and Summarizing Data
order_status_summary <- transactions %>%
  group_by(order_status) %>%
  summarize(count = n()) %>%
  mutate(percentage = (count / sum(count)) * 100)
# Create a Pie Chart
ggplot(order_status_summary, aes(x = "", y = count, fill = order_status)) +
  geom_bar(stat = "identity", width = 1) +
  coord_polar(theta = "y") +
  geom_text(aes(label = paste(order_status, "(", round(percentage, 1), "%)")),
            position = position_stack(vjust = 0.5)) +
  labs(
    title = "Pie Chart of Order Status",
    x = NULL,
    y = NULL,
    fill = "Order Status"
  ) +
  theme_void() +
  theme(legend.position = "right")
```

Explain:

In this code, the data frame 'transactions' is first grouped by the 'order_status' column using the group_by() function from the dplyr package. The summarize() function is then used to calculate the count of occurrences for each unique 'order_status.' The mutate() function computes the percentage of each 'order_status' category by dividing the count by the total count and multiplying by 100. This results in a new data frame named 'order_status_summary' that contains 'order_status,' 'count,' and 'percentage' columns.
The next part of the code creates a pie chart using ggplot2. ggplot() initializes the plot and specifies the 'order_status_summary' data frame as the data source. aes(x = "", y = count, fill = order_status) sets up aesthetics for the plot. It places the 'order_status' categories on the x-axis (with an empty string "" to represent a single category), uses the 'count' values for the y-axis, and assigns colors to each 'order_status' category for filling the segments. geom_bar(stat = "identity", width = 1) creates the bars in the pie chart with 'order_status' as the identity variable. The width is set to 1 to create a complete circle. coord_polar(theta = "y") transforms the plot into a polar coordinate system, turning it into a pie chart. geom_text() adds labels to each segment of the pie chart, displaying 'order_status' along with the rounded percentage. The position_stack(vjust = 0.5) argument positions the labels slightly above the center of each segment.
labs() sets the plot title and labels for the x-axis, y-axis, and the legend. theme_void() removes unnecessary elements like axes and grid lines, making it a clean pie chart. theme(legend.position = "right") positions the legend on the right side of the plot.

3. ***[This code generates a scatter plot that visually explores the relationship between 'list_price' and 'standard_cost' for the data in the 'transactions' data frame]***

Code:

```
ggplot(data = transactions, aes(x = list_price, y = standard_cost)) +
  geom_point(color = "chartreuse") +
  labs(
    title = "Scatter Plot of List Price vs. Standard Cost",
    x = "List Price",
    y = "Standard Cost"
  ) +   theme_minimal()
```

Explain:

ggplot() initializes the plot, specifying the 'transactions' data frame as the data source. aes(x = list_price, y = standard_cost) sets up aesthetics for the plot, mapping 'list_price' to the x-axis and 'standard_cost' to the y-axis. geom_point(color = "chartreuse") adds points to the plot, representing each data point in the dataset as a chartreuse-colored point. labs() sets the plot title and labels for the x-axis and y-axis. theme_minimal() applies a minimalistic theme to the plot, reducing unnecessary background elements and grid lines.

4. ***[This code generates a histogram that displays the distribution of 'tenure' values in the 'new_customer_list' data frame.]***

Code:

```
ggplot(data = new_customer_list, aes(x =tenure)) +
  geom_histogram(binwidth = 1, fill = "gold", color = "red") +
  labs(
    title = "Histogram of tenure",
    x = "tenure",
    y = "Frequency"
  ) +   theme_minimal()
```

Explain:

ggplot() initializes the plot, specifying the 'new_customer_list' data frame as the data source. aes(x = tenure) sets up aesthetics for the plot, mapping the 'tenure' variable to the x-axis. geom_histogram(binwidth = 1, fill = "gold", color = "red") adds a histogram to the plot. It specifies a bin width of 1, fills the bars with a gold color, and outlines them with a red color. labs() sets the plot title and labels for the x-axis (tenure) and y-axis (Frequency). theme_minimal() applies a minimalistic theme to the plot, which reduces unnecessary background elements and grid lines.

5. ***[This code generates a bar plot that visualizes the distribution of 'job_industry_category' in the 'new_customer_list' data frame.]***

Code:

```
ggplot(data = new_customer_list, aes(x = job_industry_category)) +
  geom_bar(fill = "violetred") +
  labs(
    title = "Distribution of job_industry_category",
    x = "job_industry_category",
    y = "Count"
  ) +   theme_minimal()
```

Explain:

ggplot() initializes the plot, specifying the 'new_customer_list' data frame as the data source. aes(x = job_industry_category) sets up aesthetics for the plot, mapping the 'job_industry_category' variable to the x-axis. geom_bar(fill = "violetred") adds bars to the plot, where each bar represents a unique 'job_industry_category.' The bars are filled with a violet-red color. labs() sets the plot title and labels for the x-axis ('job_industry_category') and y-axis ('Count'). theme_minimal() applies a minimalistic theme to the plot, reducing unnecessary background elements and grid lines.

6. ***[This code generates a scatter plot that visualizes the relationship between 'age' and 'tenure' for the data in the 'merged_data' data frame.]***

Code:

```
ggplot(merged_data, aes(x = age, y = tenure, color = gender)) +
  geom_point() +
  labs( title = "Scatter Plot of Age vs. Tenure",
    x = "Age",
    y = "Tenure"
  ) +   scale_color_manual(values = c("Male" = "blue", "Female" = "pink"))
```

Explain:

ggplot() initializes the plot, specifying the 'merged_data' data frame as the data source. aes(x = age, y = tenure, color = gender) sets up aesthetics for the plot. 'age' is mapped to the x-axis, 'tenure' to the y-axis, and 'gender' is used to color the points. geom_point() adds points to the plot, representing each data point. The points are colored based on the 'gender' variable. labs() sets the plot title and labels for the x-axis ('Age') and y-axis ('Tenure'). scale_color_manual(values = c("Male" = "blue", "Female" = "pink")) customizes the color mapping for 'gender.' It assigns blue to 'Male' and pink to 'Female' so that points representing individuals of different genders have distinct colors.

7. ***[This code generates a bar chart that shows how wealth segment preferences vary for different brands.]***

Code:

```
ggplot(merged_data, aes(x = brand, fill = wealth_segment)) +
  geom_bar(position = "dodge") +
  labs(
    title = "Impact of Wealth Segment on Brand",
    x = "Brand",
    y = "Count"
  ) +
  scale_fill_manual(values = c("High Net Worth" = "blue", "Mass Customer" =
                               "red", "Affluent Customer" = "green")) +
  theme_minimal() +
  theme(legend.title = element_blank())  # Remove legend title
```

Explain:

ggplot() initializes the plot, specifying the 'merged_data' data frame as the data source. aes(x = brand, fill = wealth_segment) sets up aesthetics for the plot. 'brand' is mapped to the x-axis, and 'wealth_segment' is used to fill the bars with different colors. geom_bar(position = "dodge") adds bars to the plot, where each bar represents a brand, and bars are grouped by wealth segment using the "dodge" position. labs() sets the plot title and labels for the x-axis ('Brand') and y-axis ('Count'). scale_fill_manual(values = c("High Net Worth" = "blue", "Mass Customer" = "red", "Affluent Customer" = "green")) customizes the fill color mapping for 'wealth_segment.' It assigns different colors to different wealth segments, such as blue for "High Net Worth," red for "Mass Customer," and green for "Affluent Customer." theme_minimal() applies a minimalistic theme to the plot, reducing unnecessary background elements and grid lines. theme(legend.title = element_blank()) removes the legend title, leaving only the legend with color-coded wealth segments.

8. ***[This code it visualizes how different genders interact with different brands, showing the strength of these interactions.]***

Code:

```
interaction_matrix <- table(merged_data$gender, merged_data$brand)
chordDiagram(interaction_matrix, transparency = 0.5)
```

Explain:

table(merged_data$gender, merged_data$brand) creates a two-way contingency table that counts the interactions between the 'gender' and 'brand' variables in the 'merged_data' data frame. Each cell in the table represents the count of occurrences of a specific gender-brand combination. chordDiagram(interaction_matrix, transparency = 0.5) is used to create a chord diagram visualization. It takes the 'interaction_matrix' as input and uses it to create a chord diagram. The 'transparency' parameter is set to 0.5, which specifies the transparency level of the chords connecting different categories.

9. ***[this code generates a grouped bar chart that illustrates the distribution of brands among different age groups and genders of customers]***

Code:

```
# Group the data by 'age,' 'gender,' and 'brand' and calculate the count of customers
data_counts <- merged_data %>%
  group_by(age, gender, brand) %>%
  summarise(Count = n())  # Calculate the count of customers
# Create the grouped bar chart
ggplot(data_counts, aes(x = age, y = Count, fill = brand)) +
  geom_bar(stat = "identity", position = "dodge") +
  facet_grid(. ~ gender) +
  labs(
    title = "Brand Distribution by Age and Gender",
    x = "Age Group",
    y = "Count"
  ) +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1)
  )
```

Explain:

merged_data is the data frame containing information about customers, including their age, gender, and preferred brand. group_by(age, gender, brand) groups the data based on three variables: 'age,' 'gender,' and 'brand.' summarise(Count = n()) calculates the count of customers within each group, resulting in a new data frame named 'data_counts' that includes columns for 'age,' 'gender,' 'brand,' and 'Count.' The 'Count' column represents the number of customers in each age, gender, and brand category. ggplot(data_counts, aes(x = age, y = Count, fill = brand)) initializes the plot, using the 'data_counts' data frame as the data source. It maps 'age' to the x-axis, 'Count' (customer count) to the y-axis, and uses 'brand' to fill the bars with different colors. geom_bar(stat = "identity", position = "dodge") adds the bars to the plot. stat = "identity" specifies that the heights of the bars correspond to the actual 'Count' values. position = "dodge" positions the bars side by side for each 'age' category, grouped by 'gender.' facet_grid(. ~ gender) creates a faceted plot, where each facet represents a different 'gender' category. This allows you to compare brand distribution by age for male and female customers separately. labs(...) sets the plot title, x-axis label, and y-axis label. theme_minimal() applies a minimalistic theme to the plot, removing unnecessary background elements and grid lines. theme(axis.text.x = element_text(angle = 45, hjust = 1)) customizes the appearance of the x-axis text labels by rotating them 45 degrees and adjusting their horizontal justification for better readability

10. ***[This code generates a boxplot that visualizes how 'tenure' is distributed among different 'wealth_segment' categories.]***

Code:

```
ggplot(merged_data, aes(x = wealth_segment, y = tenure, fill = wealth_segment)) +
  geom_boxplot() +
  labs(
    title = "Tenure Distribution by Wealth Segment",
    x = "Wealth Segment",
    y = "Tenure"
  ) +  theme_minimal()
```

Explain:

ggplot(merged_data, aes(x = wealth_segment, y = tenure, fill = wealth_segment)) initializes the plot, specifying the 'merged_data' data frame as the data source. It maps 'wealth_segment' to the x-axis, 'tenure' to the y-axis, and uses 'wealth_segment' to fill the boxes with different colors. geom_boxplot() adds boxplots to the plot. Each boxplot represents the distribution of 'tenure' values within a specific 'wealth_segment.' Boxplots provide information about the median, quartiles, and potential outliers of the 'tenure' values for each wealth segment. labs(...) sets the plot title, x-axis label ('Wealth Segment'), and y-axis label ('Tenure'). theme_minimal() applies a minimalistic theme to the plot, reducing unnecessary background elements and grid lines, creating a clean and focused visualization.

11. ***[This code generates a correlation heatmap that visualizes the pairwise correlations between the selected variables ('age,' 'tenure,' and 'property_valuation').]***

Code:

```
# Calculate correlations between 'age,' 'tenure,' and 'property_valuation'
correlations <- cor(merged_data[, c("age", "tenure", "property_valuation")])
# Reshape the correlation matrix into long format
melt_corr <- melt(correlations)
# Create the correlation heatmap using ggplot2
ggplot(melt_corr, aes(x = Var1, y = Var2, fill = value)) +
  geom_tile() +
  labs(
    title = "Correlation Heatmap",
    x = "",
    y = ""  ) +
  scale_fill_gradient(low = "green", high = "yellow") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

Explain:

correlations <- cor(merged_data[, c("age", "tenure", "property_valuation")]) calculates the correlation matrix for the selected variables ('age,' 'tenure,' and 'property_valuation') from the 'merged_data' data frame. melt_corr <- melt(correlations) reshapes the correlation matrix into long format using the melt function from the reshape2 package. This step is necessary to prepare the data for creating a heatmap. ggplot(melt_corr, aes(x = Var1, y = Var2, fill = value)) initializes the plot, using the 'melt_corr' data frame as the data source. It maps 'Var1' to the x-axis, 'Var2' to the y-axis, and uses 'value' (correlation coefficient) to fill the tiles of the heatmap. geom_tile() adds the tiles to the plot, creating a heatmap where each tile represents the correlation between two variables. labs(...) sets the plot title and removes the x-axis and y-axis labels, as they are not needed in this heatmap. scale_fill_gradient(low = "green", high = "yellow") customizes the color scale for the heatmap, with lower correlation values shown in green and higher values in yellow. theme_minimal() applies a minimalistic theme to the plot, reducing unnecessary background elements and grid lines. theme(axis.text.x = element_text(angle = 45, hjust = 1)) customizes the appearance of the x-axis text labels by rotating them 45 degrees and adjusting their horizontal justification for better readability.

8. Conclusion:

In conclusion, the analysis of the data from Sprocket Central Pty Ltd has provided valuable insights into various aspects of the company's operations and customer demographics. Here are some key observations from the graphs and data analysis:

- **Order Cancellation**: Approximately 0.9% of orders were found to be cancelled, indicating a relatively low cancellation rate.
- **Product Line Preference**: The "Standard" product line category is the most popular, accounting for about 70% of purchases.
- **Gender Distribution**: Approximately 50.8% of buyers are male, while the rest are female.
- **Bike Related Purchases**: Bike-related purchases are evenly distributed across frequency, suggesting a consistent interest in cycling products.
- **Tenure Distribution**: Most customers have a tenure falling within the range of 5 to 17 years, with around 9 years being the most common.
- **Property Valuation**: The majority of properties are valued around 10.
- **Date of Birth (DoB) Distribution**: Customer dates of birth (DoB) are generally evenly distributed, with a few exceptions around the years 1950 and 1985.
- **Job Industry Categories**: The top job industry categories of customers are financial services, health, and manufacturing.
- **Wealth Segment**: Most buyers belong to the "Mass Customer" wealth segment.
- **Car Ownership**: Approximately 52% of buyers own a car.
- **Sales by State**: The majority of sales (53.5%) occur in New South Wales, followed by Queensland (21%) and Victoria (25.5%).
- **Car Ownership by Gender**: Interestingly, female customers own more cars than male customers.
- **Gender Distribution by Job Industry**: Every job industry category has nearly equal male and female customers.
- **Wealth Segment and Tenure**: The "High Net Worth" wealth segment has a higher average tenure.
- **Correlation**: There is a strong positive correlation between property valuation and tenure, while age and tenure have a moderate positive correlation.
- **Brand Preferences**: Each brand of cycle has nearly equal numbers of male and female buyers. However, it was observed that middle-aged male and female customers tend to prefer the "Solex" brand.
- **Solex Brand Popularity**: The "Solex" brand is popular across all wealth segments, indicating its broad appeal among customers.

In summary, the analysis provides a comprehensive understanding of customer demographics, brand preferences, and purchasing patterns for Sprocket Central Pty Ltd. These insights can be valuable for making informed business decisions and tailoring marketing strategies to target specific customer segments.

9. Links:

Data link (csv file):

https://docs.google.com/spreadsheets/d/1eD_45PsLVCIAVlZAgaHDNAsyPjhoOy8J/edit?usp=sharing&ouid=109815965349290633982&rtpof=true&sd=true

Data Cleaning and Analysis (R file):

https://drive.google.com/file/d/1353uBkMX1w8ddQVCunZxq_OmyLLh2mIs/view?usp=sharing