EDA REPORT

# SUPERSTORE SALES ANALYSIS

Exploratory Data Analysis and Insights from Supermarket Sales Data

**Prepared by:**
Mansoor ul Hassan

Course
Artificial Intelligence

# Table of CONTENTS

01 **Introduction**

In the dynamic world of retail, supermarkets face the challenge of understanding and leveraging sales data to boost their performance. This project focuses on analyzing supermarket sales data to uncover valuable insights that can help improve business strategies. By examining sales patterns, customer preferences, and product trends, we aim to provide actionable recommendations that can drive growth and enhance the shopping experience.

02 **Problem**

Supermarkets often struggle with several issues related to sales and inventory management. These include identifying which products are performing well, understanding seasonal trends, and managing stock levels efficiently. Without clear insights, supermarkets may face challenges like overstocking, stockouts, or misaligned promotional strategies. Addressing these problems is crucial for optimizing sales and customer satisfaction.

03 **Problem Solving**

To address these challenges, our project will employ data analysis techniques to examine sales records, customer buying patterns, and product performance. By using statistical methods and data visualization, we aim to identify key trends and factors influencing sales. This will help in understanding the root causes of issues and provide a clear picture of how to improve sales strategies and inventory management.

04 **Solution**

Our analysis will focus on several key areas: identifying top-selling products, understanding seasonal variations, and evaluating the effectiveness of promotions. We will use charts and graphs to present our findings, making it easier to grasp the data. Based on these insights, we will recommend strategies for better inventory management, targeted promotions, and improved customer engagement. Our goal is to offer practical solutions that supermarkets can implement to enhance their overall performance and profitability.

### Import Required Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.patches import Patch
from matplotlib.cm import ScalarMappable
import matplotlib.colors as mcolors
from matplotlib import colormaps as colormaps
from IPython.core.display import HTML
```

---

# Load Dataset

```
df = pd.read_csv("supermarket_sales.csv")
```

---

# Explore Dataset

```
df.shape
```

```
(1000, 17)
```

```
df.isna().sum()
```

```
Invoice ID                 0
Branch                     0
City                       0
Customer type              0
Gender                     0
Product line               0
Unit price                 0
Quantity                   0
Tax 5%                     0
Total                      0
Date                       0
Time                       0
Payment                    0
cogs                       0
gross margin percentage    0
gross income               0
Rating                     0
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Invoice ID               1000 non-null   object
 1   Branch                   1000 non-null   object
 2   City                     1000 non-null   object
 3   Customer type            1000 non-null   object
 4   Gender                   1000 non-null   object
 5   Product line             1000 non-null   object
 6   Unit price               1000 non-null   float64
 7   Quantity                 1000 non-null   int64
 8   Tax 5%                   1000 non-null   float64
 9   Total                    1000 non-null   float64
 10  Date                     1000 non-null   object
 11  Time                     1000 non-null   object
 12  Payment                  1000 non-null   object
 13  cogs                     1000 non-null   float64
 14  gross margin percentage  1000 non-null   float64
 15  gross income             1000 non-null   float64
 16  Rating                   1000 non-null   float64
dtypes: float64(7), int64(1), object(9)
memory usage: 132.9+ KB
```

```
df.head(4)
```

```
    Invoice ID Branch       City Customer type  Gender  \
0  750-67-8428      A     Yangon        Member  Female
1  226-31-3081      C  Naypyitaw        Normal  Female
2  631-41-3108      A     Yangon        Normal    Male
3  123-19-1176      A     Yangon        Member    Male

             Product line  Unit price  Quantity   Tax 5%     Total
Date  \
0       Health and beauty       74.69         7  26.1415  548.9715
1/5/2019
1  Electronic accessories       15.28         5   3.8200   80.2200
3/8/2019
2      Home and lifestyle       46.33         7  16.2155  340.5255
3/3/2019
3       Health and beauty       58.22         8  23.2880  489.0480
1/27/2019

    Time      Payment    cogs  gross margin percentage  gross income
Rating
0  13:08      Ewallet  522.83                 4.761905       26.1415
9.1
1  10:29         Cash   76.40                 4.761905        3.8200
9.6
```

```
2  13:23  Credit card  324.31                4.761905       16.2155
7.4
3  20:33      Ewallet  465.76                4.761905       23.2880
8.4
```

---

# Data Cleaning and Manipulation

- Converted the Time column to a time format.
- Dropped the Invoice ID column as it was unnecessary for the analysis.

```
df['Time'] = pd.to_datetime(df['Time'], format='%H:%M').dt.time
```

## Drop Unnessary Columns

```
df.drop(columns='Invoice ID', inplace=True)
```

---

# Data Visualization

- **Rating Across Product Lines:** Created a bar chart to visualize the sum of ratings for each product line.
- **Total Revenue:** Generated a bar chart showing the total revenue across product lines.
- **Total Tax:** Created a bar chart to represent the total tax collected across product lines.
- **Gross Income:** Visualized gross income for each product line using a bar chart.
- **City and Branch Count:** Analyzed and compared counts by city and branch, and city and gender.
- **Customer Type and Gender:** Plotted counts of customers by type and gender.
- **Gender Distribution:** Used a pie chart to show the percentage distribution of gender.
- **City Distribution:** Visualized the distribution of customers by city using a pie chart.
- **Product Line Distribution:** Showed the distribution of different product lines using a pie chart.
- **Branch Distribution:** Displayed the distribution of branches.
- **Customer Type Distribution:** Used a pie chart to depict the distribution of customer types.
- **Unit Price Distribution by Product Line:** Created a box plot to analyze the distribution of unit prices.
- **Tax Distribution by Product Line:** Intended to create a box plot for tax distribution but seems incomplete.

```
colors_bar = [
    '#1F77B4',  # Blue
    '#FF7F0E',  # Orange
```

```
    '#2CA02C',  # Green
    '#D62728',  # Red
    '#9467BD',  # Purple
    '#8C564B',  # Brown
    '#E377C2',  # Pink
    '#7F7F7F',  # Gray
    '#BCBD22',  # Lime
    '#17BECF',  # Teal
    '#FFBB78',  # Light Orange
    '#98DF8A',  # Light Green
    '#FF9896',  # Light Red
    '#C5B0D5',  # Light Purple
    '#C49C94'   # Light Brown
]

colors_xtick = [
    '#FFFFFF',  # White
    '#FF0000',  # Red
    '#00FF00',  # Green
    '#0000FF',  # Blue
    '#FF00FF',  # Magenta
    '#800000',  # Maroon
    '#808000',  # Olive
    '#008000',  # Dark Green
    '#800080',  # Purple
    '#808080',  # Gray
    '#C0C0C0',  # Silver
    '#FFC0CB',  # Pink
    '#A52A2A'   # Brown
]

colors_ytick = [
    '#FFFFFF',  # White
    '#FF00FF',  # Magenta
    '#00FF00',  # Green
    '#0000FF',  # Blue
    '#FF0000',  # Red
    '#800000',  # Maroon
    '#808000',  # Olive
    '#008000',  # Dark Green
    '#800080',  # Purple
    '#808080',  # Gray
    '#C0C0C0',  # Silver
    '#FFC0CB',  # Pink
    '#FFA500',  # Orange
    '#A52A2A',  # Brown
    '#7F7F7F',  # Medium Gray
    '#D3D3D3'   # Light Gray
]
```

```
# Define font dictionaries with color
title_font = {'fontsize': 16, 'fontweight': 'bold', 'family': 'serif',
'color': '#DC143C'}

subtitle_font = {'fontsize': 14, 'fontweight': 'light', 'family':
'serif', 'color': '#6A5ACD'}

label_font = {'fontsize': 12, 'family': 'sans-serif', 'color':
'#FFFFFF'}
```

## Rating Across Product Line

```
rating = df.groupby('Product line')['Rating'].sum() # First we sum the
rating of products

# Create the plot with specified figure size
fig, ax = plt.subplots(figsize=(10, 7))

# Create the bar chart
ax.bar(rating.index, rating.values, color=colors_bar)

# Set the background color of the figure and axes
fig.patch.set_facecolor('#000000')  # Figure background color (light
gray)
ax.set_facecolor('#F5F5F5')  # Axes background color (black)


# Set titles and labels
ax.set_title('A Detailed View Of Rating Across Product Lines',
fontdict=title_font)
ax.set_xlabel("Product Line", fontdict=label_font)
ax.set_ylabel("Rating",fontdict=label_font)

# Set the font size of x-tick labels
ax.tick_params(axis='x', labelsize=8)

# Set the color of the x-tick labels
for label, color in zip(ax.get_xticklabels(), colors_xtick):
    label.set_color(color)


# Set the color of the x-tick labels
for label, color in zip(ax.get_yticklabels(), colors_ytick):
    label.set_color(color)


# Add a figure label or caption
fig.text(0.5, -0.05, 'Figure 01: This plot shows the total rating 0f A
Product Using Bar Chart and use different colors to destingusih
between plots',
         ha='center', va='center', fontsize=12, color='white',
```

```
bbox=dict(facecolor='#1e1e1e', edgecolor='none'))


# Save the figure to a file
fig.savefig('figure1.png', bbox_inches='tight')


plt.show()
```

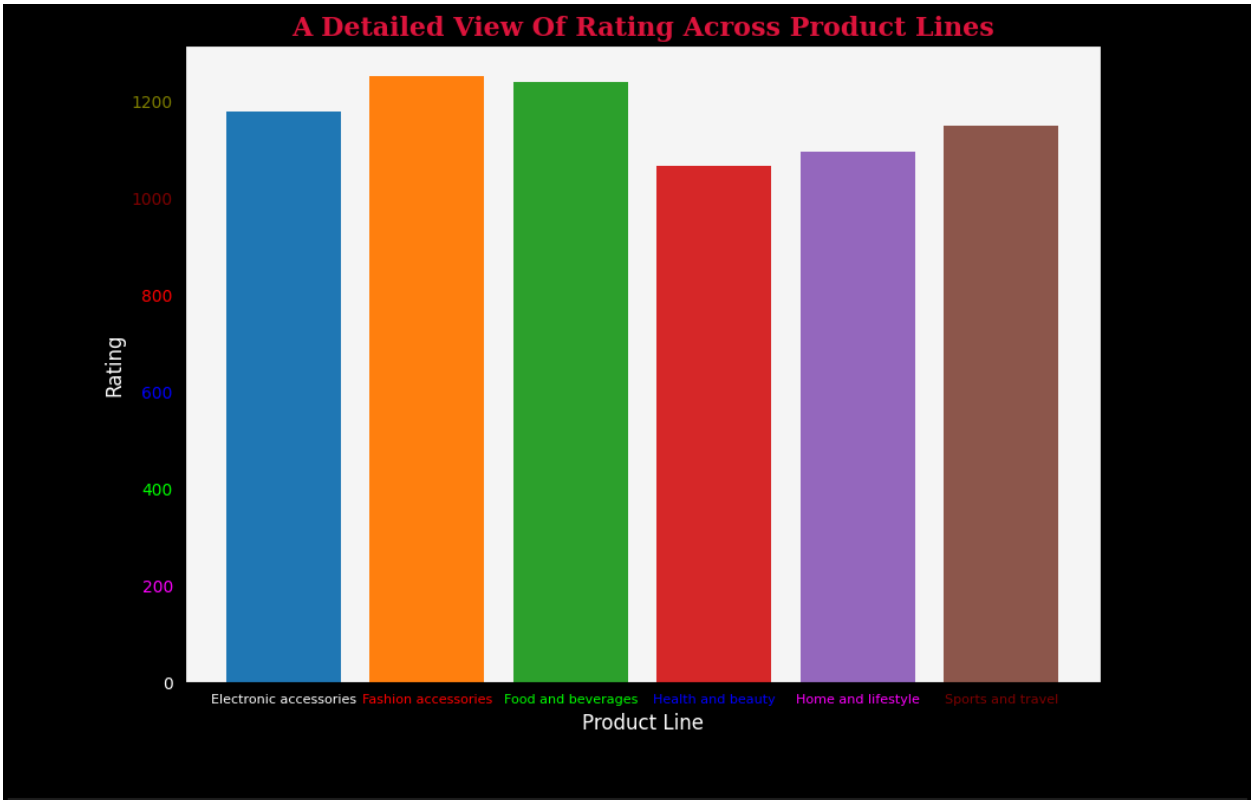


Figure 01: This plot shows the total rating 0f A Product Using Bar Chart and use different colors to destingusih between plots

```
# HTML to center the image
html = '<div style="text-align: center;"><img src="figure1.png"
/></div>'

display(HTML(html))
```

```
<IPython.core.display.HTML object>
```

## Analyze Total Revenue

```
p_revenue = df.groupby('Product line')['Total'].sum()

fig, ax = plt.subplots(figsize=(10,7))

fig.patch.set_facecolor('#f0FFF0')
```

```
ax.bar(rating.index, rating.values, color=colors_bar)
# Set the font size of x-tick labels
ax.tick_params(axis='x', labelsize=8)

ax.set_title("A Detailed View Of Total Price Across Product Line",
fontdict=title_font)
ax.set_xlabel("Prooduct Line", fontdict=label_font)
ax.set_ylabel("Total Revenue", fontdict=label_font)

for label, color in zip(ax.get_xticklabels(), colors_bar):
    label.set_color(color)

for label, color in zip(ax.get_yticklabels(), colors_ytick):
    label.set_color(color)


# Add a figure label or caption
fig.text(0.5, -0.05, 'Figure 02: This plot shows a detailed view of
Total Price Across Product Line',
         ha='center', va='center', fontsize=12, color='green',
bbox=dict(facecolor='#1e0e1e', edgecolor='none'))



# Save the figure to a file
fig.savefig('figure2.png', bbox_inches='tight')


plt.show()
```

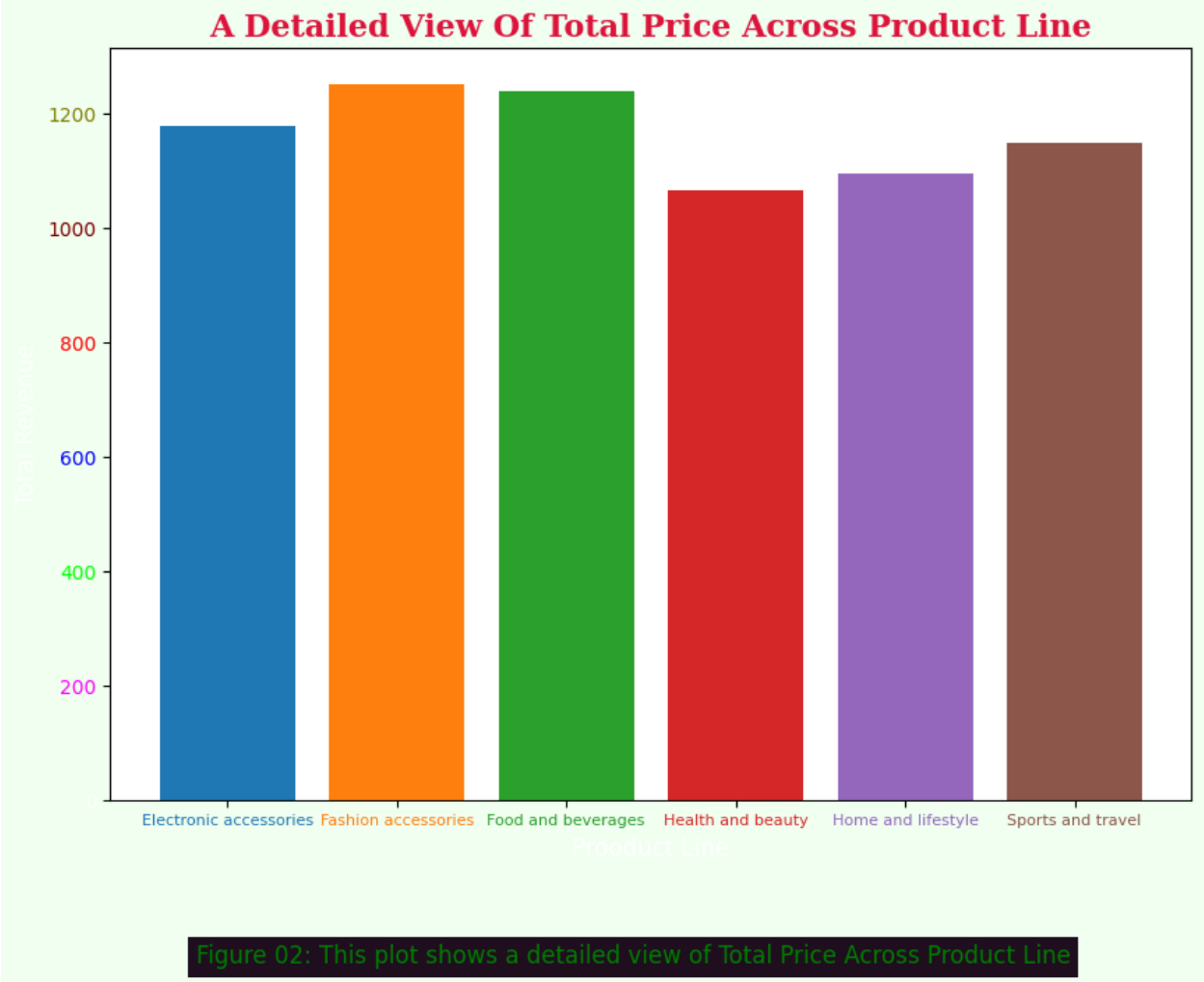


Figure 02: This plot shows a detailed view of Total Price Across Product Line

```
# HTML to center the image
html = '<div style="text-align: center;"><img src="figure2.png"
/></div>'

display(HTML(html))
```

```
<IPython.core.display.HTML object>
```

## Total Tax Across Product Line

```
p_tax = df.groupby('Product line')['Tax 5%'].sum()

fig, ax = plt.subplots(figsize=(10,7))

fig.patch.set_facecolor('#0f0F00')

ax.bar(p_tax.index, p_tax.values, color=colors_bar)
# Set the font size of x-tick labels
ax.tick_params(axis='x', labelsize=8)
```

```
ax.set_title("A Detailed View Of Total Tax Across Product Line",
fontdict=title_font)
ax.set_xlabel("Prooduct Line", fontdict=label_font)
ax.set_ylabel("Tax 5%", fontdict=label_font)

for label, color in zip(ax.get_xticklabels(), colors_xtick):
    label.set_color(color)

for label, color in zip(ax.get_yticklabels(), colors_ytick):
    label.set_color(color)


# Add a figure label or caption
fig.text(0.5, -0.05, 'Figure 03: This plot shows a detailed view of
taxes across Product Line',
         ha='center', va='center', fontsize=12, color='white',
bbox=dict(facecolor='#1e1e1e', edgecolor='none'))



# Save the figure to a file
fig.savefig('figure3.png', bbox_inches='tight')

plt.show()
```

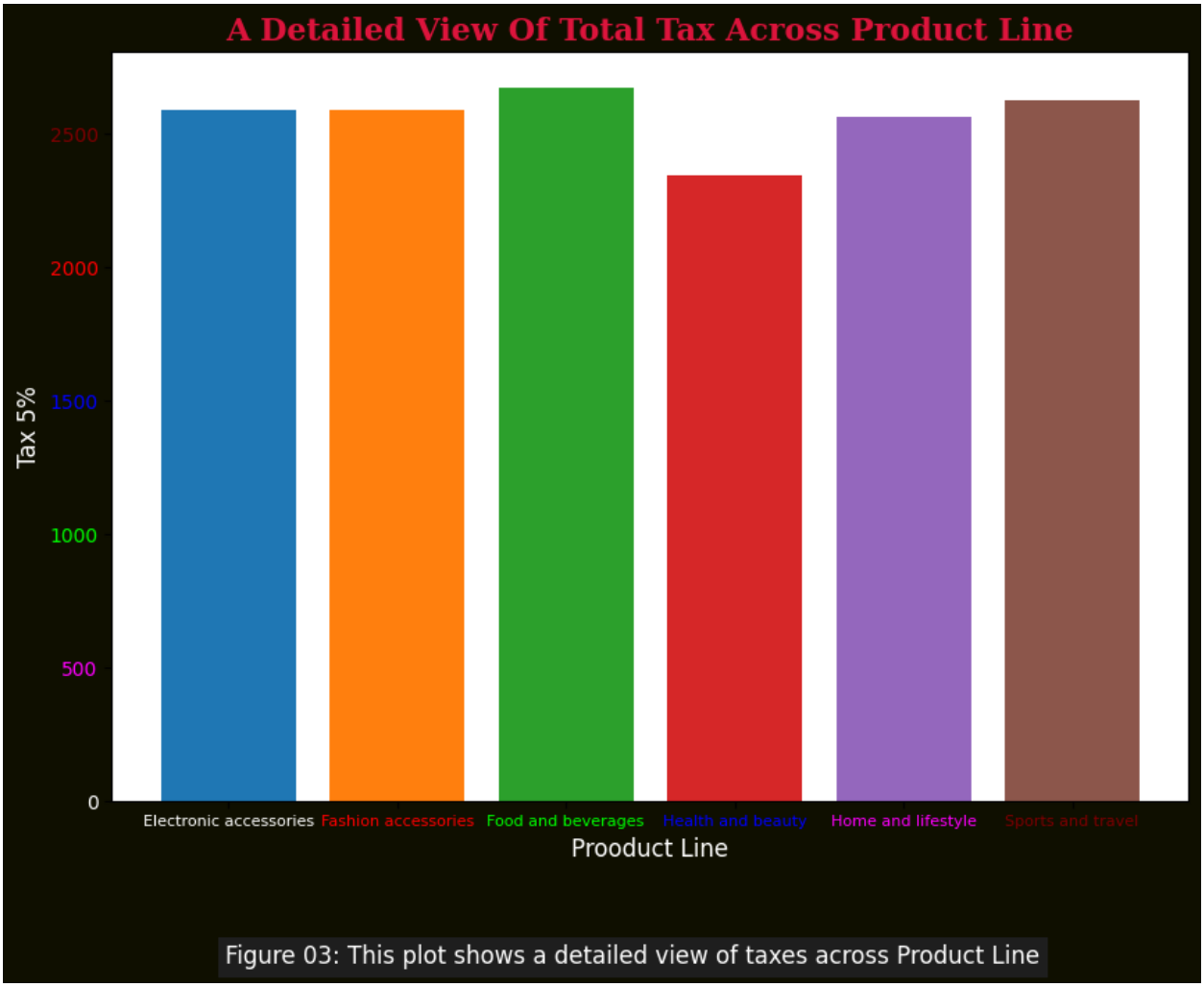


Figure 03: This plot shows a detailed view of taxes across Product Line

```
# HTML to center the image
html = '<div style="text-align: center;"><img src="figure3.png"
/></div>'

display(HTML(html))
```

```
<IPython.core.display.HTML object>
```

## Gross Income Across Product Line

```
p_gincome = df.groupby('Product line')['gross income'].sum()

fig, ax = plt.subplots(figsize=(10, 10))

fig.patch.set_facecolor('#000000')

ax.bar(p_gincome.index, p_gincome.values, color=colors_bar)
ax.tick_params(axis='x', labelsize=8)
ax.set_title("A Detailed View Of Gross Income Across Product Line",
```

```
fontdict=title_font)
ax.set_xlabel("Product Line", fontdict=label_font)
ax.set_ylabel("Gross Income", fontdict=label_font)

for label, color in zip(ax.get_xticklabels(), colors_bar):
    label.set_color(color)

for label, color in zip(ax.get_yticklabels(), colors_ytick):
    label.set_color(color)

fig.text(0.5, 0.05, 'Figure 04: This plot shows the Detailed view of
Gross Income Across Product Line using bar chart',
         ha='center', va='center', fontsize=12, color='white',
bbox=dict(facecolor='#1e1e1e', edgecolor='none'))

# Save the figure to a file
fig.savefig('figure4.png', bbox_inches='tight')

plt.show()
```

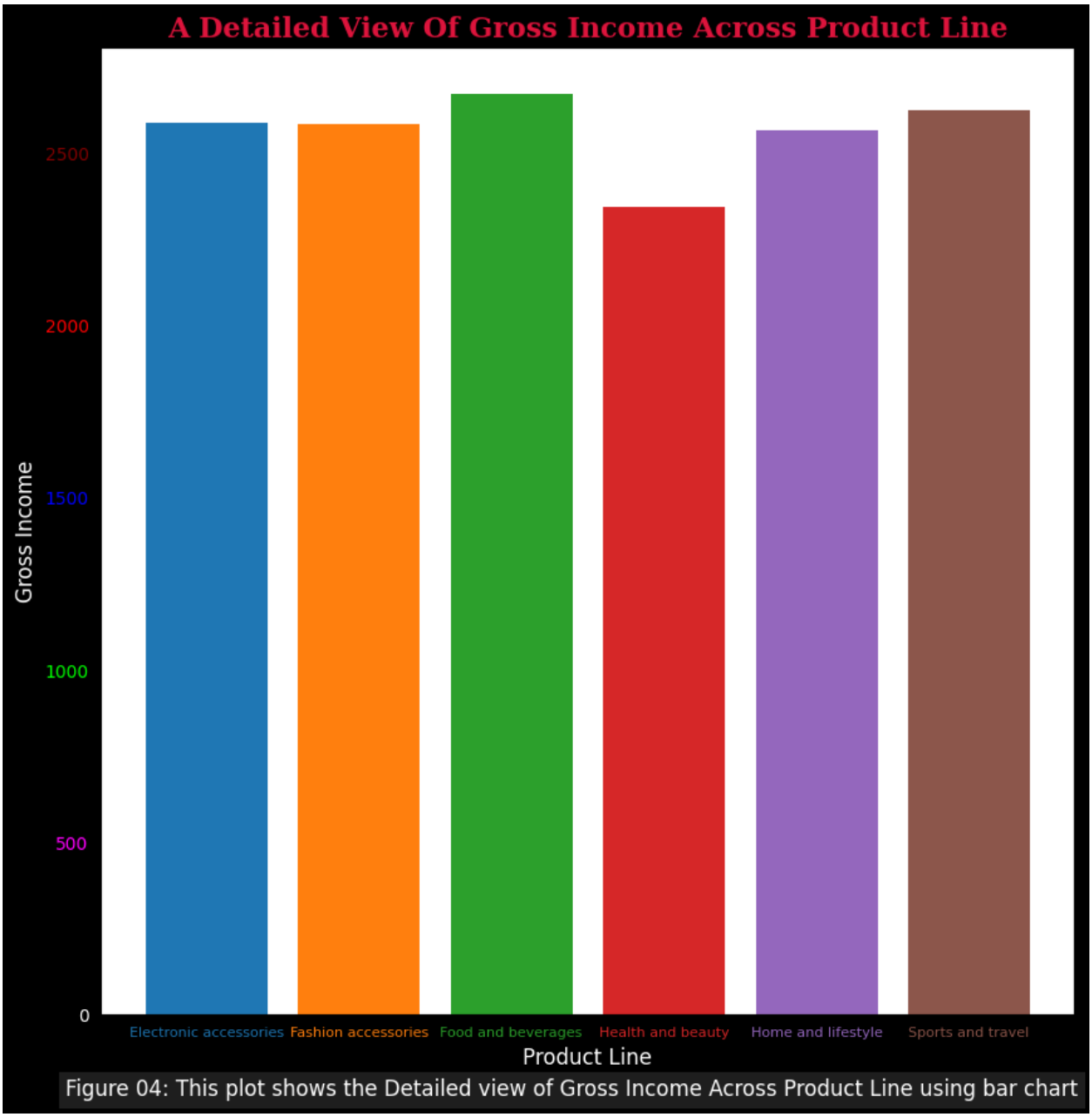

Figure 04: This plot shows the Detailed view of Gross Income Across Product Line using bar chart

```
# HTML to center the image
html = '<div style="text-align: center;"><img src="figure4.png"
/></div>'

display(HTML(html))
```

```
<IPython.core.display.HTML object>
```

## City and Branch Count

```
# Data for City and Branch count
branch_counts = df.groupby(['City', 'Branch']).size().unstack()
```

```
# Create a figure with two subplots
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(14, 6))
fig.patch.set_color('#000000')

branch_counts.plot(kind='bar', stacked=False, ax=ax[0],
edgecolor='black')
ax[0].set_title('Count by City and Branch', fontdict=subtitle_font)
ax[0].legend(title='Branch', bbox_to_anchor=(1.02, 0.5))
ax[0].set_xlabel('City', fontdict = label_font)
ax[0].set_ylabel('Count', fontdict = label_font)
for label, color in zip(ax[0].get_xticklabels(), colors_xtick):
    label.set_color(color)

for label, color in zip(ax[0].get_yticklabels(), colors_ytick):
    label.set_color(color)

# Data for City and Gender count
gender_counts = df.groupby(['City', 'Gender']).size().unstack()
gender_counts.plot(kind='bar', stacked=False, ax=ax[1],
edgecolor='black')
ax[1].set_title('Count by City and Gender', fontdict=subtitle_font)
ax[1].legend(title='Gender', bbox_to_anchor=(1.02, 0.5))
ax[1].set_xlabel('City', fontdict = label_font)
ax[1].set_ylabel('Count', fontdict = label_font)
for label, color in zip(ax[1].get_xticklabels(), colors_xtick):
    label.set_color(color)

for label, color in zip(ax[1].get_yticklabels(), colors_ytick):
    label.set_color(color)

# Add a super title for the entire figure
fig.suptitle('City Counts Analysis', fontsize=18, fontweight='bold',
family='serif', color='#DC143C')

# Adjust layout
plt.tight_layout(rect=[0, 0, 1, 1])  # Adjust rect to fit the super
title


# Add a figure label or caption
fig.text(0.5, -0.05, 'Figure 05: These plots shows a detailed analysis
of city count by using branch and gender',
         ha='center', va='center', fontsize=12, color='white',
bbox=dict(facecolor='#1e1e1e', edgecolor='none'))

# Save the figure to a file
fig.savefig('figure5.png', bbox_inches='tight')

plt.show()
```

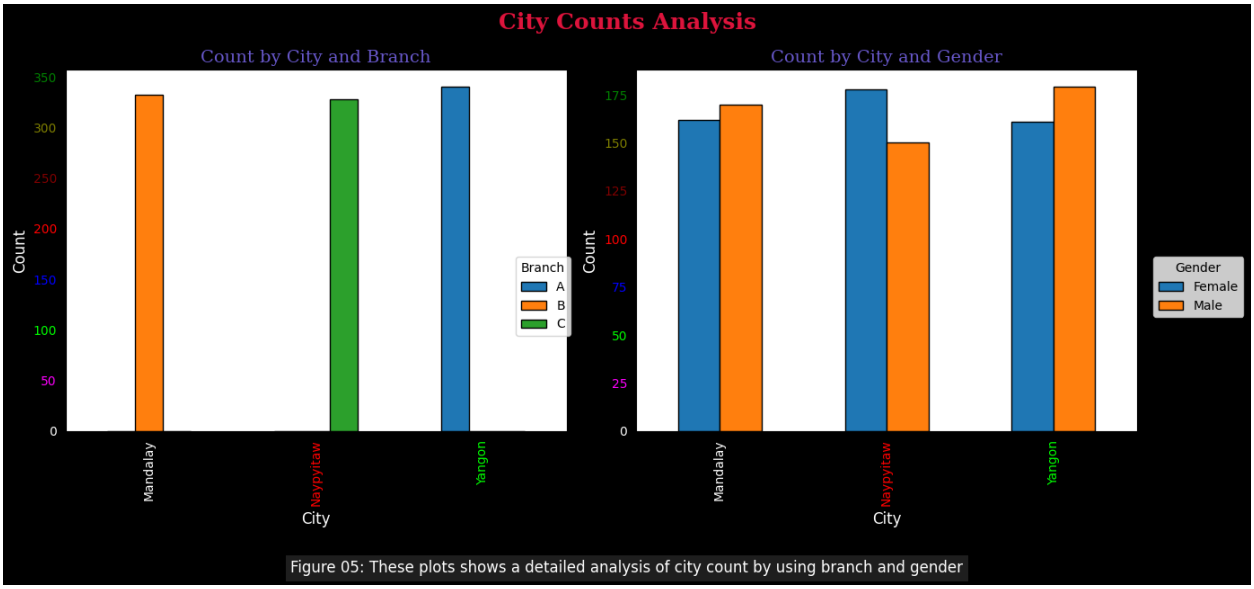


Figure 05: These plots shows a detailed analysis of city count by using branch and gender

```
# HTML to center the image
html = '<div style="text-align: center;"><img src="figure5.png"
/></div>'

display(HTML(html))
```

```
<IPython.core.display.HTML object>
```

## Customer Type and Gender

```
# Aggregate data by Customer type and Gender
data = df.groupby(['Customer type',
'Gender']).size().unstack().fillna(0)

# Get unique values for Customer type and Gender
customer_types = data.index
genders = data.columns

# Create a figure and axis
fig, ax = plt.subplots(figsize=(14, 6))
#fig.patch.set_color('#000000')  # Set background color of the figure

# Plot bars for each Gender
bar_width = 0.35
x = np.arange(len(customer_types))

for i, gender in enumerate(genders):
    counts = data[gender]
    bars = ax.bar(x + i * bar_width, counts, width=bar_width,
label=gender, color=colors_bar[i], edgecolor='red')

    # Add bar labels
```

```
    ax.bar_label(bars)

# Customize the plot
ax.set_xticks(x + bar_width / 2)
ax.set_xticklabels(customer_types)
ax.set_xlabel('Customer Type', fontdict=label_font)
ax.set_ylabel('Count', fontdict=label_font)
ax.set_title('Count by Customer Type and Gender', fontdict=title_font)
ax.legend(title='Gender', bbox_to_anchor=(1.02, 0.5))

for label, color in zip(ax.get_xticklabels(), colors_bar):
    label.set_color(color)

for label, color in zip(ax.get_yticklabels(), colors_ytick):
    label.set_color(color)


# Add a figure label or caption
fig.text(0.5, -0.05, 'Figure 06: This plot shows the Count of Customer
Type and Gender',
         ha='center', va='center', fontsize=12, color='white',
bbox=dict(facecolor='#1e1e1e', edgecolor='none'))


plt.show()
```

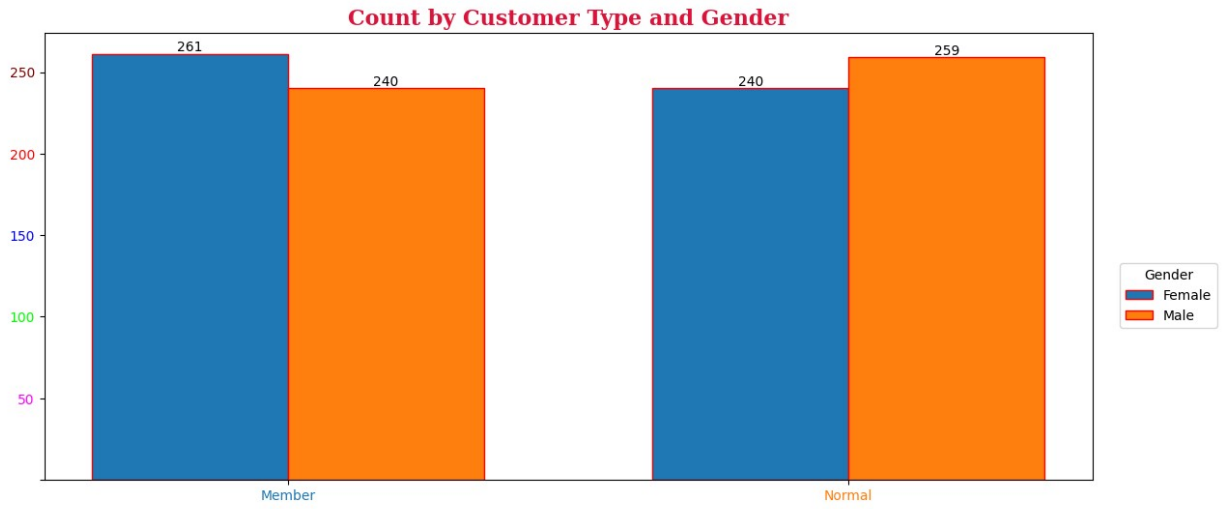


Figure 06: This plot shows the Count of Customer Type and Gender

## Percentage of Gender in Data

```
g_count = df['Gender'].value_counts()

# Create a figure with two subplots
fig, ax = plt.subplots(figsize=(9, 5))
```

```
fig.patch.set_color('#000000')

labels=g_count.index
# Pie chart on the first subplot
ax.pie(g_count, autopct='%1.1f%%', colors=colors_bar,
          startangle=140,
wedgeprops=dict(edgecolor='green'),textprops=dict(color='#ffffff'))

ax.set_title('Gender Distribution', fontdict=title_font)

# Adding a custom legend

legend_elements = [Patch(facecolor=color, edgecolor='green',
label=label) for color, label in zip(colors_bar, labels)]
# Adding a custom legend
legend = ax.legend(handles=legend_elements, title='Customer',
bbox_to_anchor=(1.02, 0.5), loc='center', fontsize='12')

# Set the color of legend text from the predefined list
for text, color in zip(legend.get_texts(), colors_bar):
    text.set_color(color)


# Add a figure label or caption
fig.text(0.53, -0.05, 'Figure 07: This Plot shows the distribution of
Gender',
         ha='center', va='center', fontsize=12, color='white',
bbox=dict(facecolor='#1e1e1e', edgecolor='none'))

fig.subplots_adjust(bottom=-0.10)  # Adjust bottom margin for caption
space

# Save the figure to a file
fig.savefig('figure7.png', bbox_inches='tight')


plt.show()
```

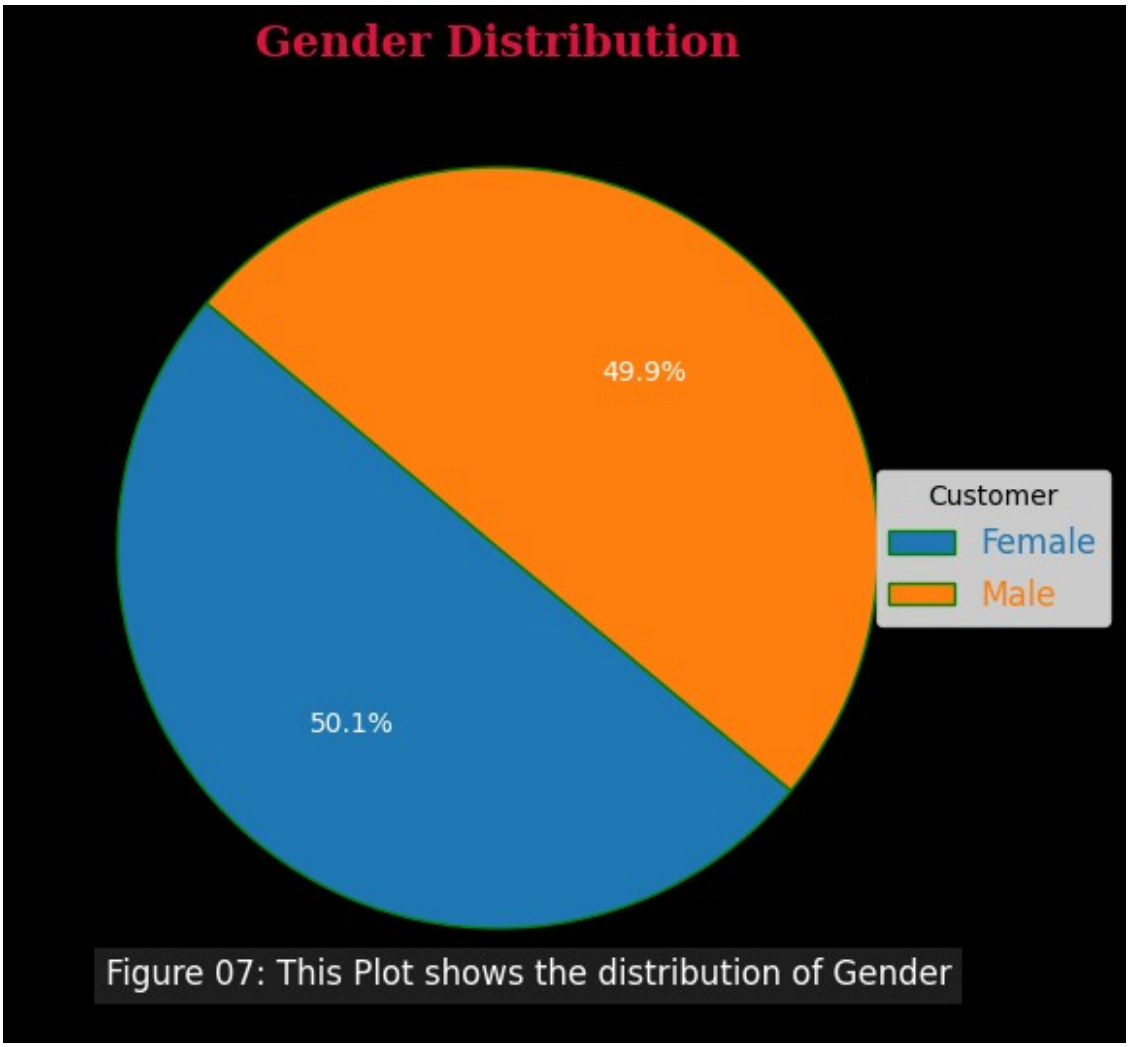


Figure 07: This Plot shows the distribution of Gender

```
# HTML to center the image
html = '<div style="text-align: center;"><img src="figure7.png"
/></div>'

display(HTML(html))
```

```
<IPython.core.display.HTML object>
```

## Percentage of cities

```
c_count = df['City'].value_counts()

# Create a figure with one subplot
fig, ax = plt.subplots(figsize=(14, 6))
fig.patch.set_color('#000000')  # Set the background color of the
figure

labels=c_count.index
# Pie chart on the subplot
```

```
wedges, texts, autotexts = ax.pie(c_count, autopct='%1.1f%%',
colors=colors_bar,
                                  startangle=140,
wedgeprops=dict(edgecolor='green'), textprops=dict(color='#ffffff'))

ax.set_title('City Distribution', fontdict=title_font)


# Adding a custom legend

legend_elements = [Patch(facecolor=color, edgecolor='green',
label=label) for color, label in zip(colors_bar, labels)]
# Adding a custom legend
legend = ax.legend(handles=legend_elements, title='Customer',
bbox_to_anchor=(1.02, 0.5), loc='center', fontsize='12')

# Set the color of legend text from the predefined list
for text, color in zip(legend.get_texts(), colors_bar):
    text.set_color(color)

# Add a figure label or caption
fig.text(0.53, -0.05, 'Figure 08: This Plot shows the distribution of
City',
         ha='center', va='center', fontsize=12, color='white',
bbox=dict(facecolor='#1e1e1e', edgecolor='none'))



fig.subplots_adjust(bottom=-0.10)  # Adjust bottom margin for caption
space

# Save the figure to a file
fig.savefig('figure8.png', bbox_inches='tight')

plt.show()
```

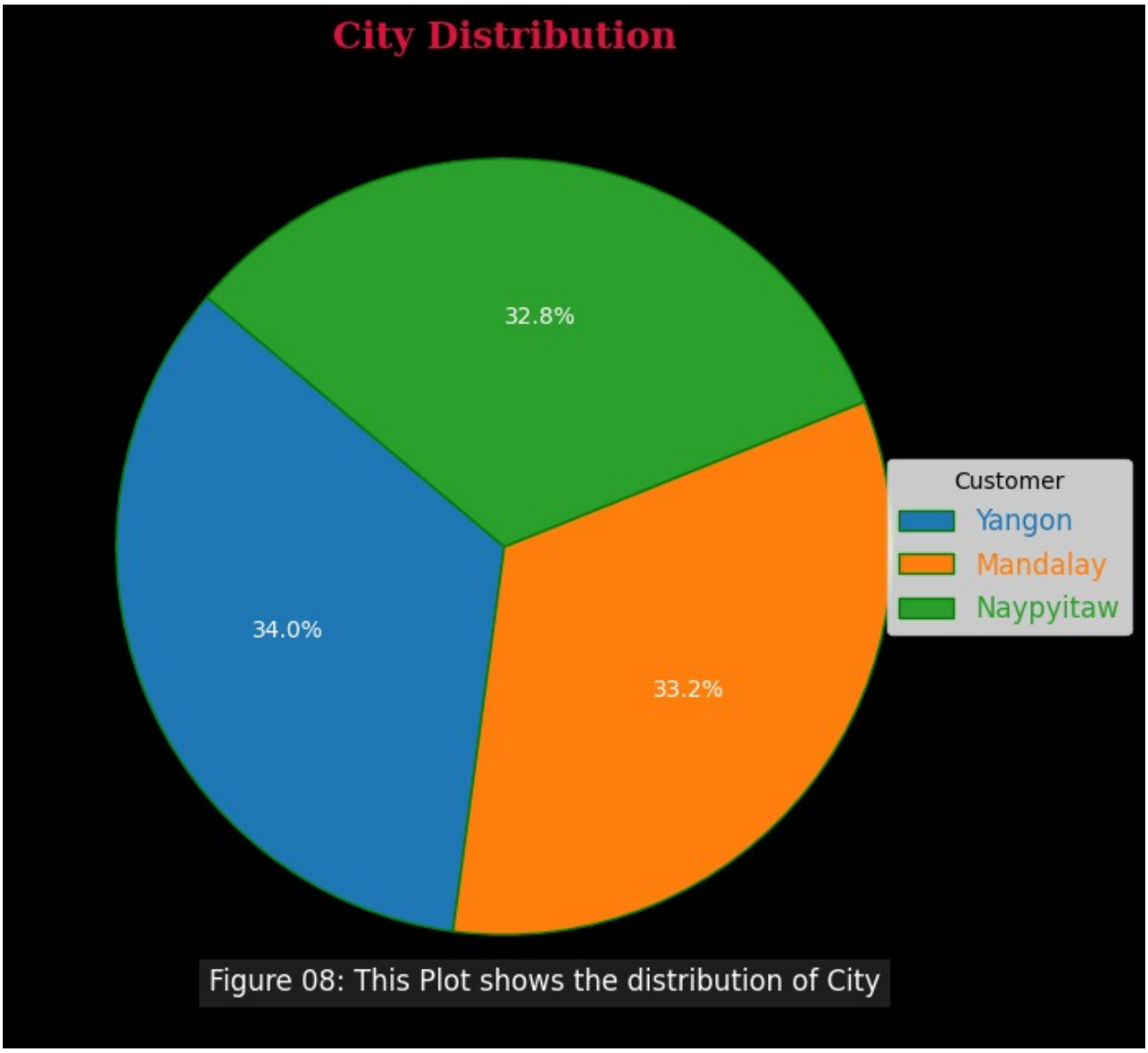


Figure 08: This Plot shows the distribution of City

```
# HTML to center the image
html = '<div style="text-align: center;"><img src="figure8.png"
/></div>'

display(HTML(html))
```

```
<IPython.core.display.HTML object>
```

## Distribution of Product Line

```
p_count = df['Product line'].value_counts()

# Create a figure with one subplot
fig, ax = plt.subplots(figsize=(14, 6))
fig.patch.set_color('#000000')  # Set the background color of the
figure
```

```
labels=p_count.index
# Pie chart on the subplot
wedges, texts, autotexts = ax.pie(p_count, autopct='%1.1f%%',
colors=colors_bar,
                                  startangle=120,
wedgeprops=dict(edgecolor='green'), textprops=dict(color='#ffffff'))

ax.set_title('Product Line Distribution', fontdict=title_font)

# Adding a custom legend

legend_elements = [Patch(facecolor=color, edgecolor='green',
label=label) for color, label in zip(colors_bar, labels)]
# Adding a custom legend
legend = ax.legend(handles=legend_elements, title='Customer',
bbox_to_anchor=(1.02, 0.5), loc='center', fontsize='12')

# Set the color of legend text from the predefined list
for text, color in zip(legend.get_texts(), colors_bar):
    text.set_color(color)

# Add a figure label or caption
fig.text(0.53, -0.05, 'Figure 09: This Plot shows the distribution of
Products',
         ha='center', va='center', fontsize=12, color='white',
bbox=dict(facecolor='#1e1e1e', edgecolor='none'))


fig.subplots_adjust(bottom=-0.10)  # Adjust bottom margin for caption
space
# Save the figure to a file
fig.savefig('figure9.png', bbox_inches='tight')


plt.show()
```

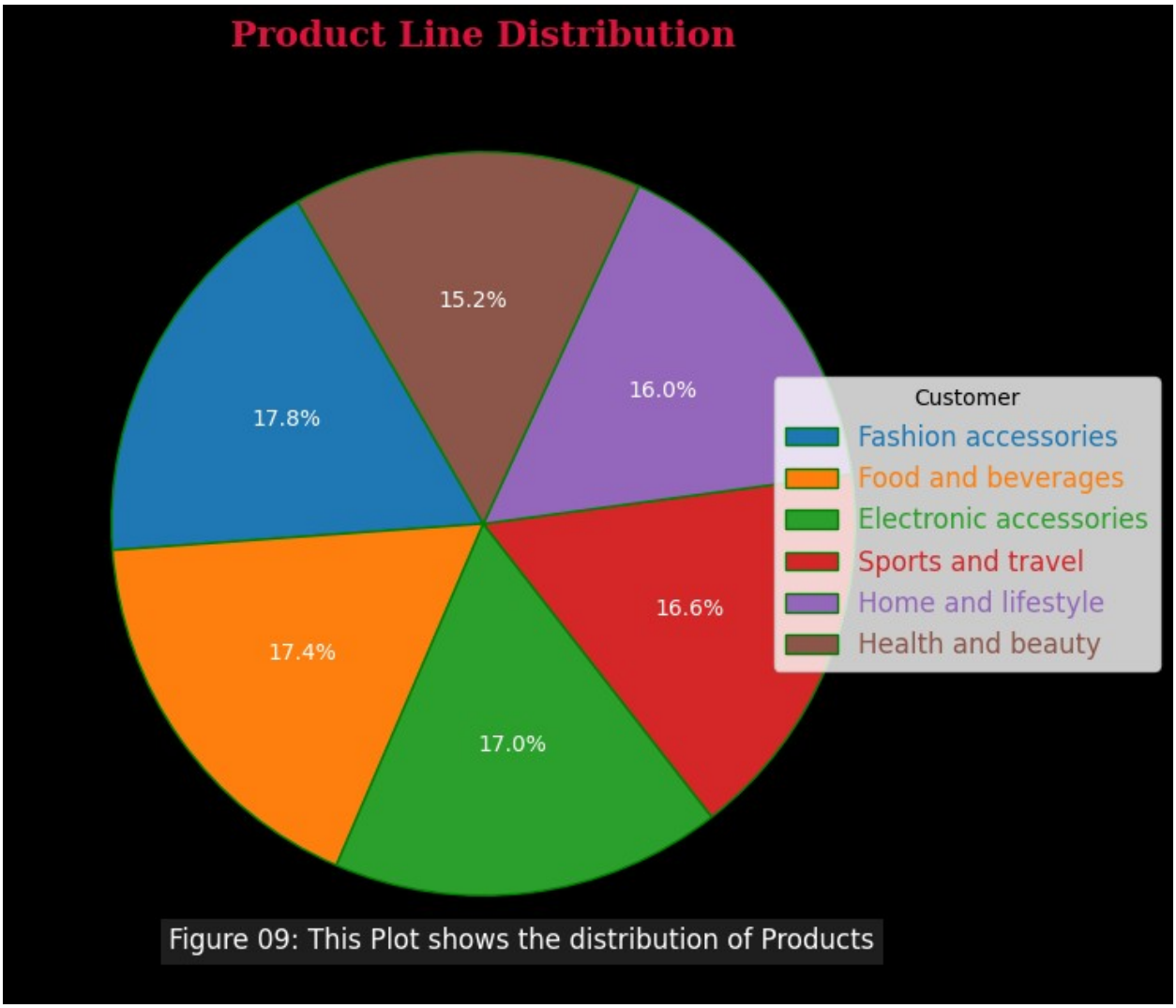

Figure 09: This Plot shows the distribution of Products

```
# HTML to center the image
html = '<div style="text-align: center;"><img src="figure9.png"
/></div>'
display(HTML(html))
```

```
<IPython.core.display.HTML object>
```

## Branch Distribution

```
b_count = df['Branch'].value_counts()

# Create a figure with one subplot
fig, ax = plt.subplots(figsize=(14, 6))
fig.patch.set_color('#000000')  # Set the background color of the
figure

labels=b_count.index
# Pie chart on the subplot
```

```
wedges, texts, autotexts = ax.pie(b_count, autopct='%1.1f%%',
colors=colors_bar,
                                  startangle=120,
wedgeprops=dict(edgecolor='green'), textprops=dict(color='#ffffff'))

ax.set_title('Branch Distribution', fontdict=title_font)

# Adding a custom legend

legend_elements = [Patch(facecolor=color, edgecolor='green',
label=label) for color, label in zip(colors_bar, labels)]
# Adding a custom legend
legend = ax.legend(handles=legend_elements, title='Customer',
bbox_to_anchor=(1.02, 0.5), loc='center', fontsize='12')

# Set the color of legend text from the predefined list
for text, color in zip(legend.get_texts(), colors_bar):
    text.set_color(color)

# Add a figure label or caption
fig.text(0.53, -0.05, 'Figure 10: This Plot shows the distribution of
Branch',
         ha='center', va='center', fontsize=12, color='white',
bbox=dict(facecolor='#1e1e1e', edgecolor='none'))


fig.subplots_adjust(bottom=-0.10)  # Adjust bottom margin for caption
space

# Save the figure to a file
fig.savefig('figure10.png', bbox_inches='tight')


plt.show()
```

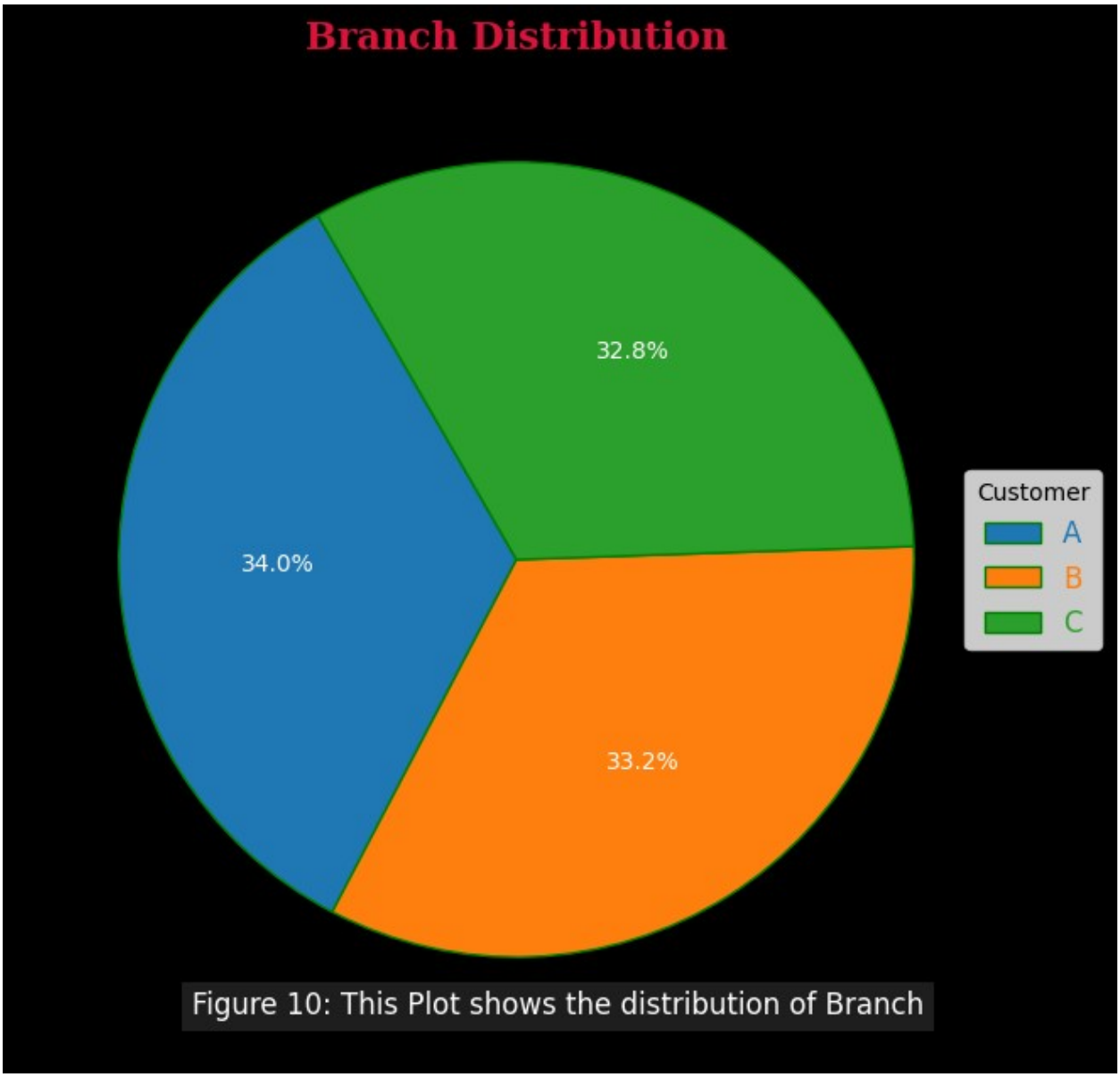


Figure 10: This Plot shows the distribution of Branch

```
# HTML to center the image
html = '<div style="text-align: center;"><img src="figure10.png"
/></div>'

display(HTML(html))
```

```
<IPython.core.display.HTML object>
```

## Customer Type Distribution

```
c_type_count = df['Customer type'].value_counts()

# Create a figure with one subplot
fig, ax = plt.subplots(figsize=(16, 6))
```

```
fig.patch.set_color('#000000')  # Set the background color of the
figure

labels=c_type_count.index
# Pie chart on the subplot
wedges, texts, autotexts = ax.pie(c_type_count, autopct='%1.1f%%',
colors=colors_bar,
                                  startangle=120,
wedgeprops=dict(edgecolor='green'), textprops=dict(color='#ffffff'))

ax.set_title('Customer Type Distribution', fontdict=title_font)


legend_elements = [Patch(facecolor=color, edgecolor='green',
label=label) for color, label in zip(colors_bar, labels)]
# Adding a custom legend
legend = ax.legend(handles=legend_elements, title='Customer',
bbox_to_anchor=(1.02, 0.5), loc='center', fontsize='12')

# Set the color of legend text from the predefined list
for text, color in zip(legend.get_texts(), colors_bar):
    text.set_color(color)

# Add a figure label or caption
fig.text(0.53, -0.05, 'Figure 11: This Plot shows the customer type
distribution',
         ha='center', va='center', fontsize=12, color='white',
bbox=dict(facecolor='#1e1e1e', edgecolor='none'))



fig.subplots_adjust(bottom=-0.10)  # Adjust bottom margin for caption
space

# Save the figure to a file
fig.savefig('figure11.png', bbox_inches='tight')


plt.show()
```

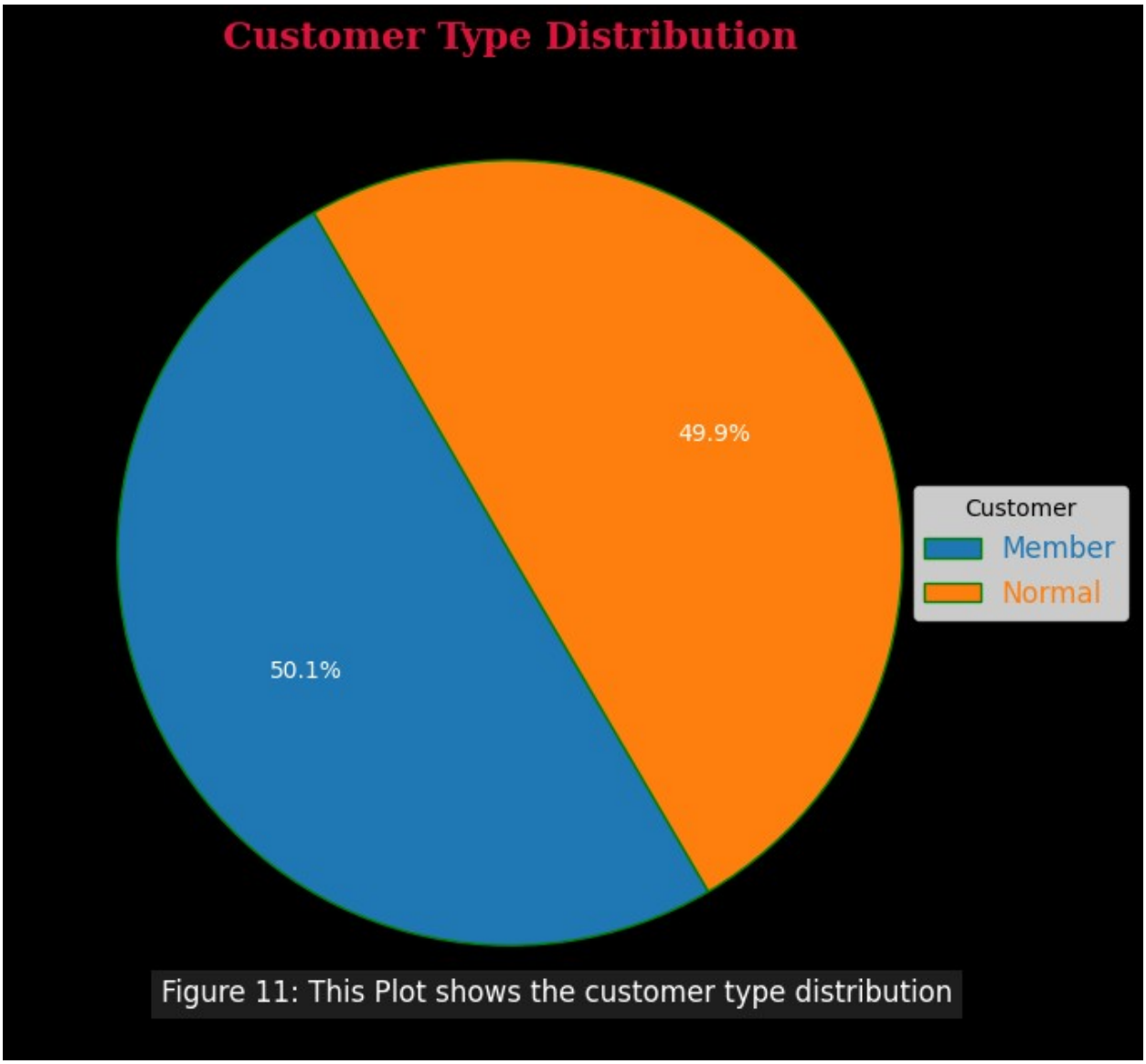


Figure 11: This Plot shows the customer type distribution

```
# HTML to center the image
html = '<div style="text-align: center;"><img src="figure11.png"
/></div>'

display(HTML(html))
```

```
<IPython.core.display.HTML object>
```

## Distribution Of unit Price By Product Line

```
# Prepare data for plotting
product_lines = df['Product line'].unique()
```

```
# Create figure and axes
fig, ax = plt.subplots(figsize=(10, 8))
fig.patch.set_color('#000000')

# Box Plot
box_data = [df[df['Product line'] == product]['Unit price'] for
product in product_lines]
# Plot each box with a different color
box_colors = colors_bar[:len(product_lines)]
for i, data in enumerate(box_data):
    boxprops = dict(facecolor=box_colors[i], color='black')
    whiskerprops = dict(color='black')
    medianprops = dict(color='red')
    ax.boxplot(data, positions=[i+1], widths=0.6, patch_artist=True,
               boxprops=boxprops, whiskerprops=whiskerprops,
medianprops=medianprops)

# Customize legend for boxplot
handles = [plt.Line2D([0], [0], color='red', lw=2, linestyle='-')]
ax.legend(handles, ['Median'], bbox_to_anchor=(1.02, 1))

# Set title and labels
ax.set_title('Distribution of Unit Prices by Product Line',
fontdict=title_font)
ax.set_xlabel('Product Line', fontdict=label_font)
ax.set_ylabel('Unit Price', fontdict=label_font)

# Customize x-ticks
ax.set_xticks(range(1, len(product_lines) + 1))
ax.set_xticklabels(product_lines, rotation=45, ha='right')


for label, color in zip(ax.get_xticklabels(), colors_bar):
    label.set_color(color)

for label, color in zip(ax.get_yticklabels(), colors_ytick):
    label.set_color(color)

# Add a figure label or caption
fig.text(0.5, -0.05, 'Figure 12: This Plot shows the distribution of
unit prices by Product Line Using Box Plot',
         ha='center', va='center', fontsize=12, color='white',
bbox=dict(facecolor='#1e1e1e', edgecolor='none'))



fig.subplots_adjust(bottom=0.20)  # Adjust bottom margin for caption
space

# Save the figure to a file
fig.savefig('figure12.png', bbox_inches='tight')
```

```
plt.tight_layout()

plt.show()
```

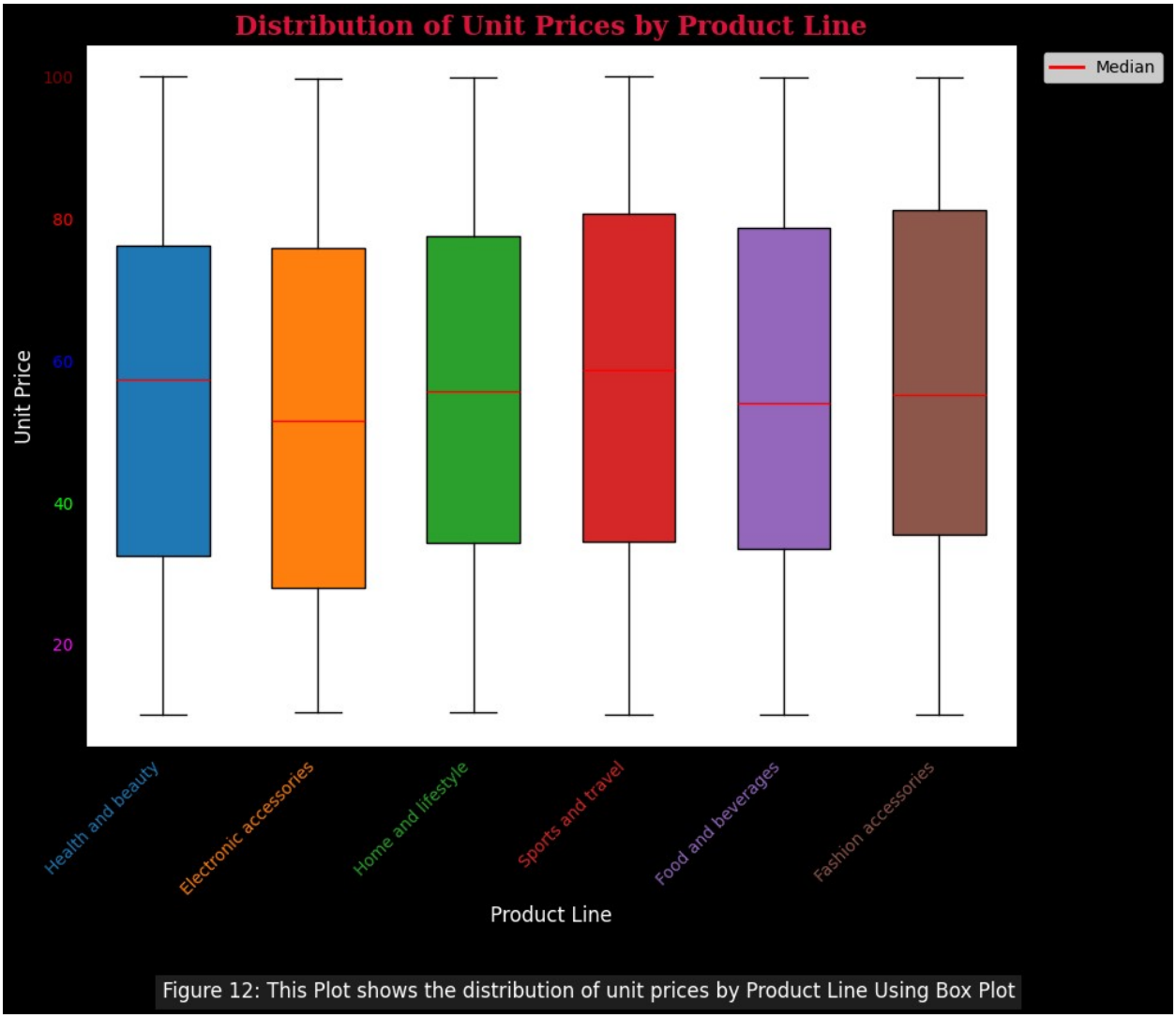


Figure 12: This Plot shows the distribution of unit prices by Product Line Using Box Plot

```
# HTML to center the image
html = '<div style="text-align: center;"><img src="figure12.png"
/></div>'
display(HTML(html))
```

```
<IPython.core.display.HTML object>
```

## Distribution Of Tax By Product Line

```
# Create figure and axes
fig, ax = plt.subplots(figsize=(10, 6))
```

```
fig.patch.set_color('#000000')

# Box Plot
box_data = [df[df['Product line'] == product]['Tax 5%'] for product in
product_lines]
# Plot each box with a different color
box_colors = colors_bar[:len(product_lines)]
for i, data in enumerate(box_data):
    boxprops = dict(facecolor=box_colors[i], color='black')
    whiskerprops = dict(color='black')
    medianprops = dict(color='red')
    ax.boxplot(data, positions=[i+1], widths=0.6, patch_artist=True,
               boxprops=boxprops, whiskerprops=whiskerprops,
medianprops=medianprops)

# Customize legend for boxplot
handles = [plt.Line2D([0], [0], color='red', lw=2, linestyle='-')]
ax.legend(handles, ['Median'], bbox_to_anchor=(1.02, 1))

# Set title and labels
ax.set_title('Distribution of Tax by Product Line',
fontdict=title_font)
ax.set_xlabel('Product Line', fontdict=label_font)
ax.set_ylabel('Tax 5%', fontdict=label_font)

# Customize x-ticks
ax.set_xticks(range(1, len(product_lines) + 1))
ax.set_xticklabels(product_lines, rotation=45, ha='right')


for label, color in zip(ax.get_xticklabels(), colors_bar):
    label.set_color(color)

for label, color in zip(ax.get_yticklabels(), colors_ytick):
    label.set_color(color)

# Add a figure label or caption
fig.text(0.5, -0.05, 'Figure 13: This Plot shows the distribution of
Tax by Product Line Using Box Plot',
         ha='center', va='center', fontsize=12, color='white',
bbox=dict(facecolor='#1e1e1e', edgecolor='none'))




fig.subplots_adjust(bottom=0.25)  # Adjust bottom margin for caption
space

# Save the figure to a file
fig.savefig('figure13.png', bbox_inches='tight')
```

```
plt.show()
```

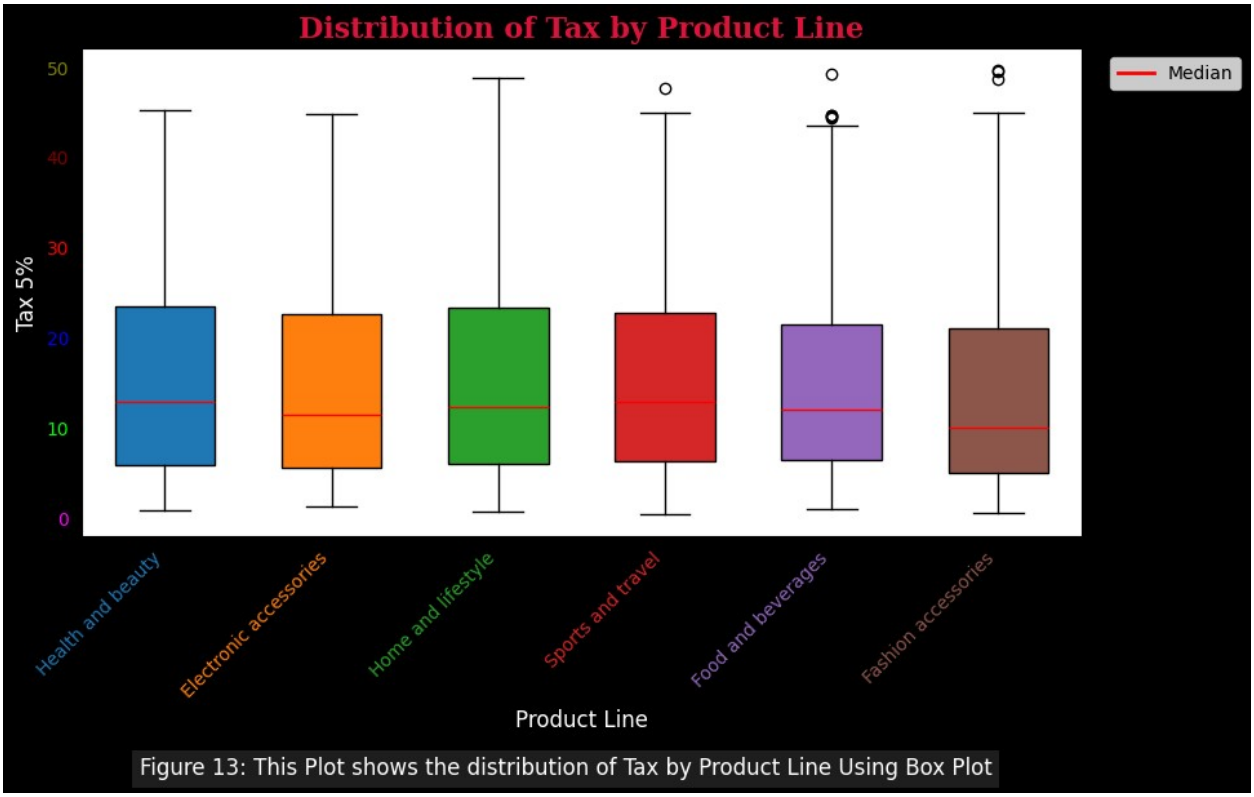


Figure 13: This Plot shows the distribution of Tax by Product Line Using Box Plot

```
# HTML to center the image
html = '<div style="text-align: center;"><img src="figure13.png"
/></div>'

display(HTML(html))
```

```
<IPython.core.display.HTML object>
```

## Distrubution Of Rating By Product Line

```
# Create figure and axes
fig, ax = plt.subplots(figsize=(10, 6))
fig.patch.set_color('#000000')

# Box Plot
box_data = [df[df['Product line'] == product]['Rating'] for product in
product_lines]
# Plot each box with a different color
box_colors = colors_bar[:len(product_lines)]
for i, data in enumerate(box_data):
    boxprops = dict(facecolor=box_colors[i], color='black')
    whiskerprops = dict(color='black')
```

```
    medianprops = dict(color='red')
    ax.boxplot(data, positions=[i+1], widths=0.6, patch_artist=True,
               boxprops=boxprops, whiskerprops=whiskerprops,
medianprops=medianprops)

# Customize legend for boxplot
handles = [plt.Line2D([0], [0], color='red', lw=2, linestyle='-')]
ax.legend(handles, ['Median'], bbox_to_anchor=(1.02, 1))

# Set title and labels
ax.set_title('Distribution of Rating by Product Line',
fontdict=title_font)
ax.set_xlabel('Product Line', fontdict=label_font)
ax.set_ylabel('Rating', fontdict=label_font)

# Customize x-ticks
ax.set_xticks(range(1, len(product_lines) + 1))
ax.set_xticklabels(product_lines, rotation=45, ha='right')


for label, color in zip(ax.get_xticklabels(), colors_bar):
    label.set_color(color)

for label, color in zip(ax.get_yticklabels(), colors_ytick):
    label.set_color(color)


# Add a figure label or caption
fig.text(0.5, -0.05, 'Figure 14: This Plot shows the distribution of
Rating by Product Line Using Box Plot',
         ha='center', va='center', fontsize=12, color='white',
bbox=dict(facecolor='#1e1e1e', edgecolor='none'))


plt.tight_layout()


fig.subplots_adjust(bottom=0.25)  # Adjust bottom margin for caption
space

# Save the figure to a file
fig.savefig('figure14.png', bbox_inches='tight')


plt.show()
```

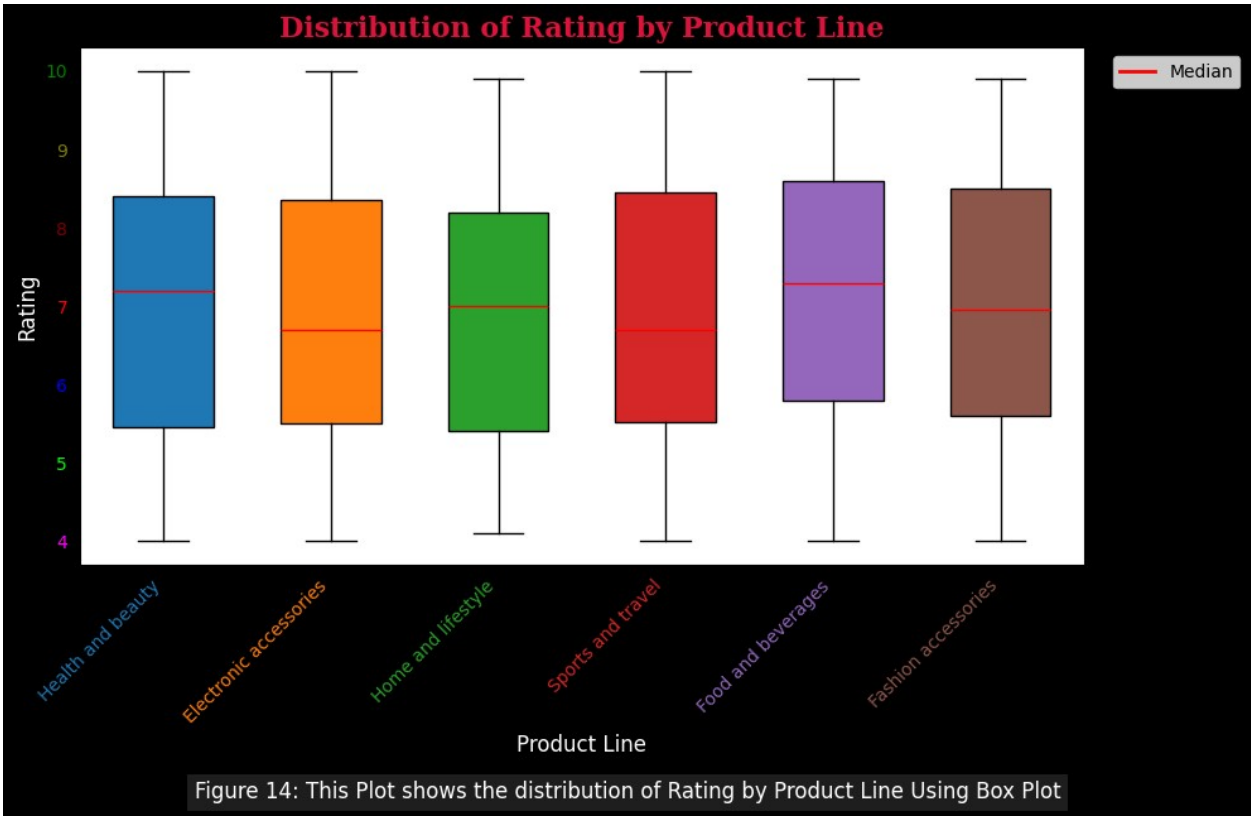


Figure 14: This Plot shows the distribution of Rating by Product Line Using Box Plot

```
# HTML to center the image
html = '<div style="text-align: center;"><img src="figure14.png"
/></div>'

display(HTML(html))
```

```
<IPython.core.display.HTML object>
```

## Distribution Of Quantity By Product Line

```
# Create figure and axes
fig, ax = plt.subplots(figsize=(10, 6))
fig.patch.set_color('#000000')

# Box Plot
box_data = [df[df['Product line'] == product]['Quantity'] for product
in product_lines]
# Plot each box with a different color
box_colors = colors_bar[:len(product_lines)]
for i, data in enumerate(box_data):
    boxprops = dict(facecolor=box_colors[i], color='black')
    whiskerprops = dict(color='black')
    medianprops = dict(color='red')
    ax.boxplot(data, positions=[i+1], widths=0.6, patch_artist=True,
               boxprops=boxprops, whiskerprops=whiskerprops,
```

```
medianprops=medianprops)

# Customize legend for boxplot
handles = [plt.Line2D([0], [0], color='red', lw=2, linestyle='-')]
ax.legend(handles, ['Median'], bbox_to_anchor=(1.02, 1))

# Set title and labels
ax.set_title('Distribution of Quantity by Product Line',
fontdict=title_font)
ax.set_xlabel('Product Line', fontdict=label_font)
ax.set_ylabel('Quantity', fontdict=label_font)

# Customize x-ticks
ax.set_xticks(range(1, len(product_lines) + 1))
ax.set_xticklabels(product_lines, rotation=45, ha='right')


for label, color in zip(ax.get_xticklabels(), colors_bar):
    label.set_color(color)

for label, color in zip(ax.get_yticklabels(), colors_ytick):
    label.set_color(color)


# Add a figure label or caption
fig.text(0.5, -0.05, 'Figure 15: This plot shows the Distribution of
Quantity by Product line using box plot.',
         ha='center', va='center', fontsize=12, color='white',
bbox=dict(facecolor='#1e1e1e', edgecolor='none'))


fig.subplots_adjust(bottom=0.20)  # Adjust bottom margin for caption
space

# Save the figure to a file
fig.savefig('figure15.png', bbox_inches='tight')


plt.show()
```

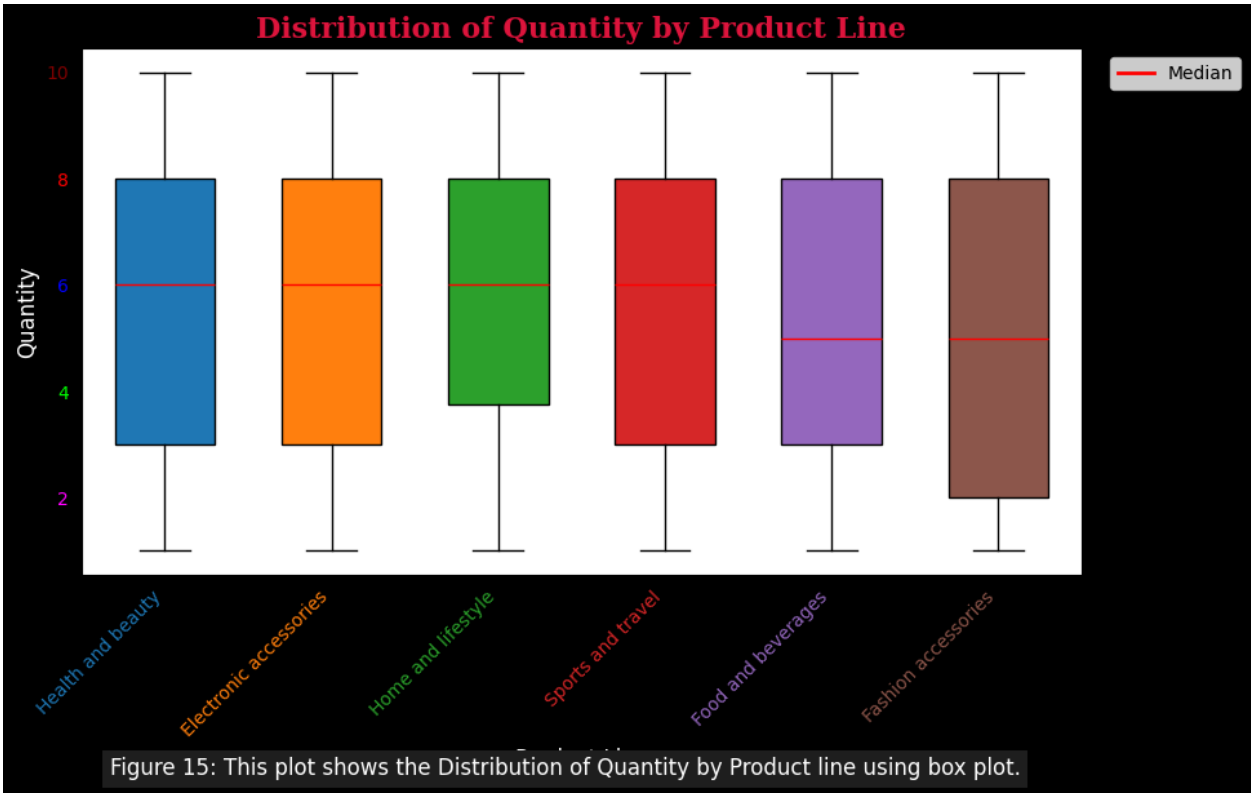

Figure 15: This plot shows the Distribution of Quantity by Product line using box plot.

```
# HTML to center the image
html = '<div style="text-align: center;"><img src="figure15.png"
/></div>'

display(HTML(html))
```

```
<IPython.core.display.HTML object>
```

## Time Based Analysis

```
# Convert 'Time' to datetime format
df['Time'] = pd.to_datetime(df['Time'], format='%H:%M:%S').dt.time

# Extract hour from the 'Time' column
df['Hour'] = df['Time'].apply(lambda x: x.hour)

# Aggregate sales by hour
hourly_rating = df.groupby('Hour')['Rating'].sum().reset_index()

# Define color gradients for the line and markers
line_color = '#1f77b4'  # Blue color for the line
marker_color = '#ff7f0e'  # Orange color for the markers

# Create figure and axes
fig, ax = plt.subplots(figsize=(10, 6))
fig.patch.set_color('#000000')  # Set figure background color
```

```
# Add color gradient to the line by plotting segments
norm = mcolors.Normalize(vmin=hourly_rating['Hour'].min(),
vmax=hourly_rating['Hour'].max())
cmap = colormaps['coolwarm']  # Use the updated method for colormap

for i in range(len(hourly_rating) - 1):
    x = hourly_rating['Hour'].iloc[i:i+2]
    y = hourly_rating['Rating'].iloc[i:i+2]
    color = cmap(norm(x.mean()))  # Get the color for this segment
    ax.plot(x, y, marker='o', linestyle='-', color=color,
            markerfacecolor=marker_color, markeredgecolor='black',
            markersize=8, linewidth=2, alpha=0.7)

# Add a color gradient colorbar
sm = plt.cm.ScalarMappable(cmap=cmap, norm=norm)
sm.set_array([])
cbar = plt.colorbar(sm, ax=ax)
cbar.set_label('Hour of the Day')

# Add title and labels
ax.set_title('Total Rating by Hour of the Day', fontdict=title_font)
ax.set_xlabel('Hour of the Day', fontdict=label_font)
ax.set_ylabel('Total Rating', fontdict=label_font)

# Customize x-ticks and y-ticks
ax.set_xticks(range(24))
ax.set_xticklabels(range(24), rotation=45, ha='right', color='white')
ax.yaxis.set_tick_params(color='white')
ax.xaxis.set_tick_params(color='white')

# Customize grid
ax.grid(True, which='both', linestyle='--', linewidth=0.7,
color='gray')

# Add data point annotations
for i, row in hourly_rating.iterrows():
    ax.text(row['Hour'], row['Rating'] + 1, str(row['Rating']),
fontsize=10, color='white', ha='center')

# Add a vertical line to highlight specific hours
highlight_hour = 19
ax.axvline(x=highlight_hour, color='red', linestyle='--',
linewidth=1.5, alpha=0.7)
ax.text(highlight_hour, max(hourly_rating['Rating']) * 0.9, f'Peak
Hour: {highlight_hour}',
        fontsize=12, color='red', ha='center', va='bottom')


for label, color in zip(ax.get_xticklabels(), colors_bar):
    label.set_color(color)
```

```
for label, color in zip(ax.get_yticklabels(), colors_ytick):
    label.set_color(color)

# Set background color for the plot area
ax.set_facecolor('#1e1e1e')

# Add a figure label or caption
fig.text(0.5, -0.05, 'Figure 16: This plot shows the total rating by
hour of the day with a color gradient representing the time of day.',
         ha='center', va='center', fontsize=12, color='white',
bbox=dict(facecolor='#1e1e1e', edgecolor='none'))

# Show plot

fig.subplots_adjust(bottom=0.20)  # Adjust bottom margin for caption
space

# Save the figure to a file
fig.savefig('figure16.png', bbox_inches='tight')

plt.ioff
```

```
<function matplotlib.pyplot.ioff() -> 'AbstractContextManager'>
```

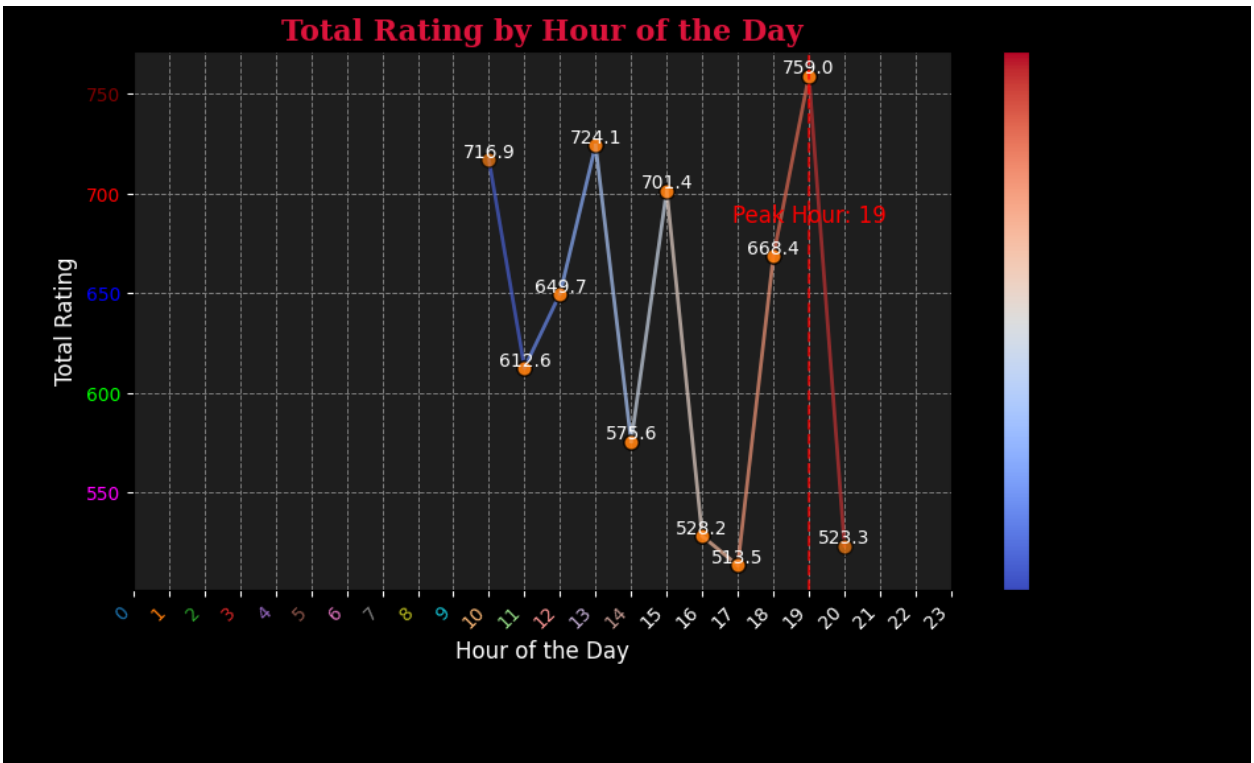


Figure 16: This plot shows the total rating by hour of the day with a color gradient representing the time of day.

```
# HTML to center the image
html = '<div style="text-align: center;"><img src="figure16.png"
/></div>'

display(HTML(html))
```

```
<IPython.core.display.HTML object>
```