CONTEXT

In the ever-evolving landscape of business,understanding and effectively catering to the diverse needs of customers is paramount.The customer segmentation project endeavors to leverage advanced analytics and machine learning techniques to classify and group customers based on their behavior,preferences, and characteristics. By doing so, businesses can tailor their strtegies to specific customer segments, optimizing marketing efforts, enhancing customer satisfaction, and ultimately maximizing overall business performance.

Project Scope

The project scope encompasses the analysis of customer data, including purchase history, demographics, and interaction patterns, to develop a sophisticated customer segmentation model. The system aims to identify meaningful customer segments that will empower businesses to customize their offerings, improve customer engagement, and drive targeted marketing campaigns for more impactful result.

Objective

The primary objective of this project is to create an effective customer segmentation system that goes beyond traditional demographic classifications. By incorporating machine learning algorithms, the system will analyze diverse data points to identify naunced customer segments. This will enable businesses to tailor their products, services, and marketing strategies to specific customer needs, fostering stronger customer relationships and increasing overall business success.

Let's start coding!

Importing necessary libraries

In [1]:
```
# this will help in making python code more structured automatically (good coding p
# % load_ext nb_black

# Libraries to help with reading and manipulating data
import pandas as pd
import numpy as np

# libraries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Remove the limit for the number of displayed columns
pd.set_option("display.max_columns", None)
#sets the limit for the number of displayed rows
pd.set_option("display.max_rows", 200)

# to compute distances
from scipy.spatial.distance import cdist, pdist

# to perform k-means clustering and compute silhouette scores
from sklearn.cluster import KMeans
```

```
from sklearn.metrics import silhouette_score

# to visualize the elbow curve and silhouette scores
from yellowbrick.cluster import KElbowVisualizer, SilhouetteVisualizer

# to perform hierarchical clustering, compute cophenetic correlation, and create de
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage, cophenet

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# loading the dataset
data = pd.read_excel(r"C:\Users\omats\OneDrive\Desktop\Sterling E-Commerce Data.xls
```

In [3]:
```
data.shape
```

Out[3]:
```
(283083, 19)
```

Observation

- The dataset has 283083 rows and 19 columns

In [31]:
```
# viewing the first 5 rows of the data
data.head()
```

Out[31]:

	Category	City	County	Cust Id	Customer Since	Date of Order	Full Name	Gender	Item Id	Ord
0	Health & Sports	Bode	Humboldt	112285	2008-02-11	2022-08-07	Renaud, Maudie	F	880913	100547
1	Men's Fashion	Belleville	St. Clair	112386	2005-06-23	2022-08-08	Shimp, Mariela	F	881493	100548
2	Men's Fashion	Belleville	St. Clair	112386	2005-06-23	2022-08-08	Shimp, Mariela	F	881492	100548
3	Computing	Young America	Carver	112501	2013-09-15	2022-08-18	Doiron, Latrina	F	886067	100551
4	Entertainment	Young America	Carver	112501	2013-09-15	2022-08-20	Doiron, Latrina	F	886878	100551

In [32]:
```
data.tail()
```

Out[32]:

	Category	City	County	Cust Id	Customer Since	Date of Order	Full Name	Gender	Item Id	O
283078	Women's Fashion	Burkettsville	Mercer	81251	2013-10-15	2021-12-30	Kester, Apolonia	F	700522	1004
283079	Women's Fashion	Burkettsville	Mercer	81251	2013-10-15	2021-12-30	Kester, Apolonia	F	700518	1004
283080	Women's Fashion	Burkettsville	Mercer	81251	2013-10-15	2021-12-30	Kester, Apolonia	F	700520	1004
283081	Women's Fashion	Burkettsville	Mercer	81251	2013-10-15	2021-12-30	Kester, Apolonia	F	700517	1004
283082	Women's Fashion	Burkettsville	Mercer	81251	2013-10-15	2021-12-30	Kester, Apolonia	F	700519	1004

In [6]:
```
# copying the data to another variables to aviod any changes to original data
df = data.copy()
```

In [7]:
```
# fixing columns names
df.columns =[c.replace(" ", "_") for c in df.columns] # replace empty space betweer
```

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 283083 entries, 0 to 283082
Data columns (total 19 columns):
 #   Column          Non-Null Count   Dtype         
---  ------          --------------   -----         
 0   Category        283083 non-null  object        
 1   City            283083 non-null  object        
 2   County          283083 non-null  object        
 3   Cust_Id         283083 non-null  int64         
 4   Customer_Since  283083 non-null  datetime64[ns]
 5   Date_of_Order   283083 non-null  datetime64[ns]
 6   Full_Name       283083 non-null  object        
 7   Gender          283083 non-null  object        
 8   Item_Id         283083 non-null  int64         
 9   Order_Id        283078 non-null  float64       
 10  Payment_Method  283083 non-null  object        
 11  Place_Name      283083 non-null  object        
 12  Ref_Num         283083 non-null  int64         
 13  Region          283083 non-null  object        
 14  State           283083 non-null  object        
 15  User_Name       283083 non-null  object        
 16  Zip             283083 non-null  int64         
 17  Qty_Ordered     283083 non-null  int64         
 18  Total           283083 non-null  float64       
dtypes: datetime64[ns](2), float64(2), int64(5), object(10)
memory usage: 41.0+ MB
```

Observation

- The data in the columns are both Object and numeric.
- It shows there is an element of missing values

In [9]:
```
# checking for missing values
df.isnull().sum()
```

Out[9]:
```
Category          0
City              0
County            0
Cust_Id           0
Customer_Since    0
Date_of_Order     0
Full_Name         0
Gender            0
Item_Id           0
Order_Id          5
Payment_Method    0
Place_Name        0
Ref_Num           0
Region            0
State             0
User_Name         0
Zip               0
Qty_Ordered       0
Total             0
dtype: int64
```

Observations

- Only one column has missing numbers , which is the Order id column

In [10]:
```
#dropping off missing data points
data.dropna(inplace=True)
```

In [11]:
```
data.isnull().sum()
```

Out[11]:
```
Category          0
City              0
County            0
Cust Id           0
Customer Since    0
Date of Order     0
Full Name         0
Gender            0
Item Id           0
Order Id          0
Payment Method    0
Place Name        0
Ref Num           0
Region            0
State             0
User Name         0
Zip               0
Qty Ordered       0
Total             0
dtype: int64
```

In [12]:
```
# checking the number of unique values in each column
df.nunique()
```

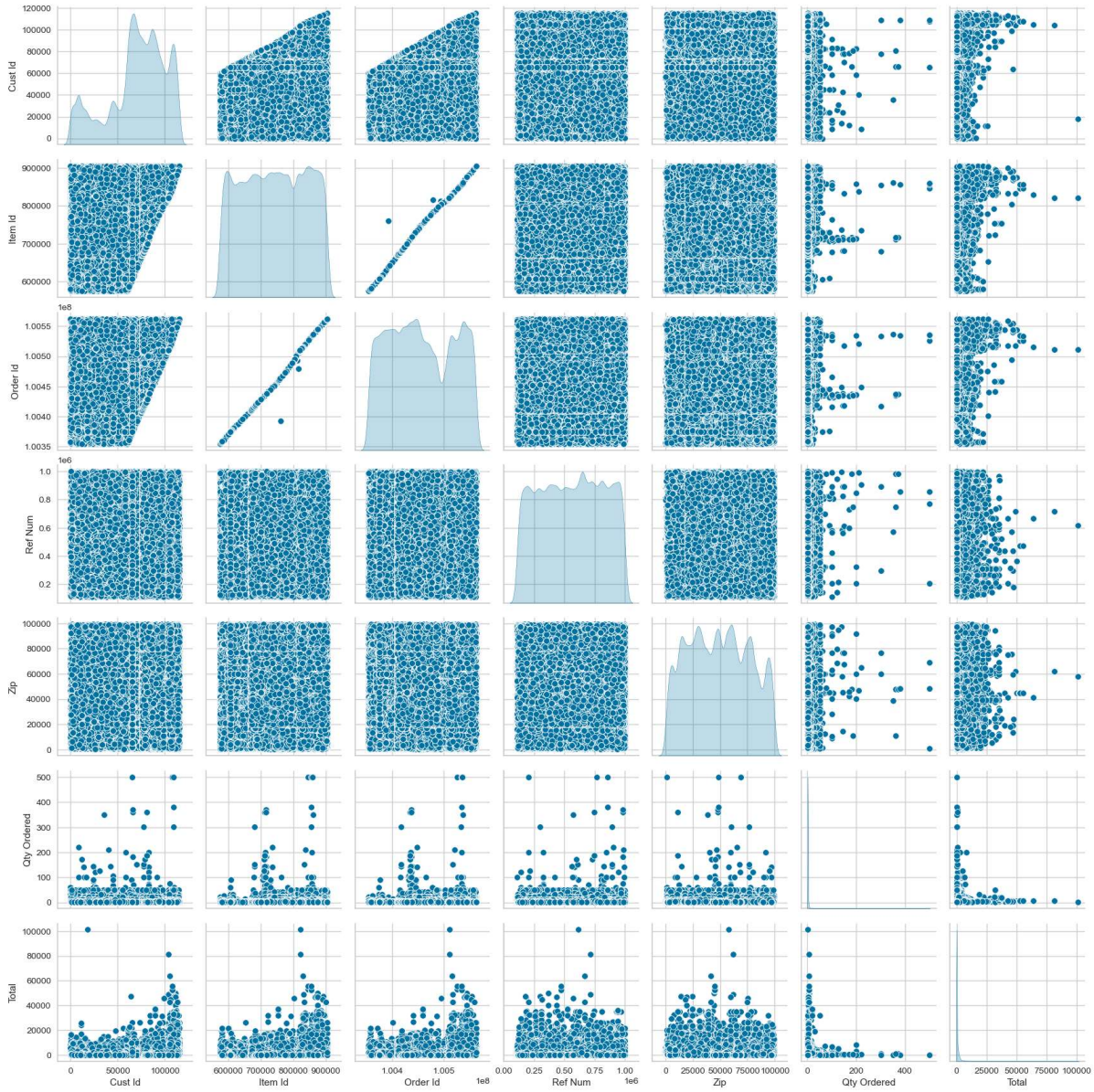
Out[12]:
```
Category             15
City              15668
County             2518
Cust_Id           63646
Customer_Since    11629
Date_of_Order       365
Full_Name         63610
Gender                2
Item_Id          283083
Order_Id         199329
Payment_Method       13
Place_Name        15668
Ref_Num           61505
Region                4
State                49
User_Name         63407
Zip               33440
Qty_Ordered          72
Total             23588
dtype: int64
```

In [13]:
```
# check for duplicated values
df.duplicated().sum()
```

Out[13]:
```
0
```

In [14]:
```
sns.pairplot(data, diag_kind='kde')
```

Out[14]:
```
<seaborn.axisgrid.PairGrid at 0x23563ed85d0>
```



Observation

- There are no duplicated values

In [15]:
```
df.Category.value_counts()
```

Out[15]:
```
Mobiles & Tablets     60954
Men's Fashion         40183
Appliances            32693
Women's Fashion       28003
Others                25913
Beauty & Grooming     17723
Entertainment         17120
Superstore            14815
Home & Living         13815
Health & Sports        8347
Computing              8025
Soghaat                7189
Kids & Baby            6421
School & Education     1078
Books                   804
Name: Category, dtype: int64
```

In [16]:
```
df.Customer_Since.value_counts()
```

Out[16]:
```
2005-11-30    2536
2017-02-08     774
2015-05-17     659
2017-06-11     541
2016-12-25     491
              ...
1981-08-01       1
1993-02-13       1
1989-01-01       1
1983-12-21       1
1992-06-10       1
Name: Customer_Since, Length: 11629, dtype: int64
```

In [17]:
```
df.Date_of_Order.value_counts()
```

Out[17]:
```
2021-12-20    13522
2021-12-27    13042
2021-12-21     7154
2022-04-30     6207
2021-12-28     5284
              ...
2022-07-20      133
2022-09-25      129
2021-10-20      122
2022-09-30       99
2022-09-24       92
Name: Date_of_Order, Length: 365, dtype: int64
```

In [18]:
```
df.Cust_Id.value_counts()
```

Out[18]:
```
85775    2524
87724     707
96927     608
65910     436
39707     397
         ...
80756       1
86561       1
73594       1
85809       1
88688       1
Name: Cust_Id, Length: 63646, dtype: int64
```

In [ ]:

In [ ]:

In [19]:
```
# statistical analysis of the data
data.describe().T
```

Out[19]:

	count	mean	std	min	25%	50%	
Cust Id	283078.0	7.010640e+04	30215.281047	4.0	5.664000e+04	74320.0	9.237100
Item Id	283078.0	7.417451e+05	95664.051170	574769.0	6.598972e+05	742468.5	8.260758
Order Id	283078.0	1.004570e+08	60909.919565	100354677.0	1.004047e+08	100451836.5	1.005134
Ref Num	283078.0	5.611037e+05	256099.906291	111127.0	3.410710e+05	565623.0	7.821822
Zip	283078.0	4.914744e+04	27235.638638	210.0	2.626400e+04	48808.0	7.200475
Qty Ordered	283078.0	3.008238e+00	4.565207	1.0	2.000000e+00	2.0	3.000000
Total	283078.0	8.162323e+02	1986.176881	0.0	4.990000e+01	149.8	8.000000

-### Observation

- The average quantity ordered is 3 while the median is 2 and the maximum is 5
- The total goods is 283083

In [20]:
```
data.describe(exclude=["int64", "float64"]).T
```

Out[20]:

	count	unique	top	freq	first	last
Category	283078	15	Mobiles & Tablets	60952	NaT	NaT
City	283078	15668	Dekalb	2525	NaT	NaT
County	283078	2518	Jefferson	3510	NaT	NaT
Customer Since	283078	11629	2005-11-30 00:00:00	2536	1978-11-04	2017-07-28
Date of Order	283078	365	2021-12-20 00:00:00	13522	2021-10-01	2022-09-30
Full Name	283078	63610	Gonzalez, Joel	2524	NaT	NaT
Gender	283078	2	M	144292	NaT	NaT
Payment Method	283078	13	cod	101745	NaT	NaT
Place Name	283078	15668	Dekalb	2525	NaT	NaT
Region	283078	4	South	103481	NaT	NaT
State	283078	49	TX	17510	NaT	NaT
User Name	283078	63407	jugonzalez	2524	NaT	NaT

Observations

- There are 15 categories of goods sold with the top good being mobiles and Tablets
- There are 15668 cities where their goods are sold and Dekalb is where they have the most goods
- Goods are sold in 2518 counties with Jefferson being the highest of 3510
- There are 11629 customers since 1978
- There are male and female customers with the male being the highest of 144295
- The customers prepfered Cash On Delivery (COD)

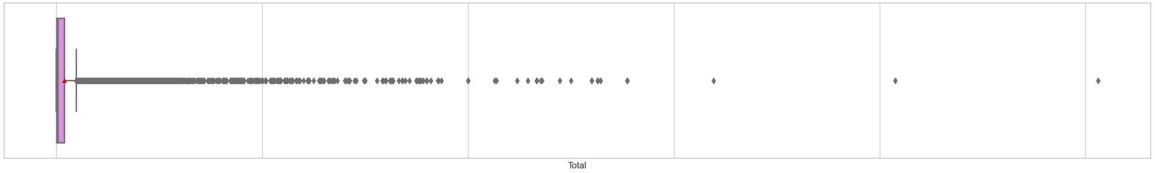
- The city with the highest visit to the mall is Dekalb
- There are for regions with the south reion has the most frequency
- There are 49 state where goods are sold with TX having the most frequency

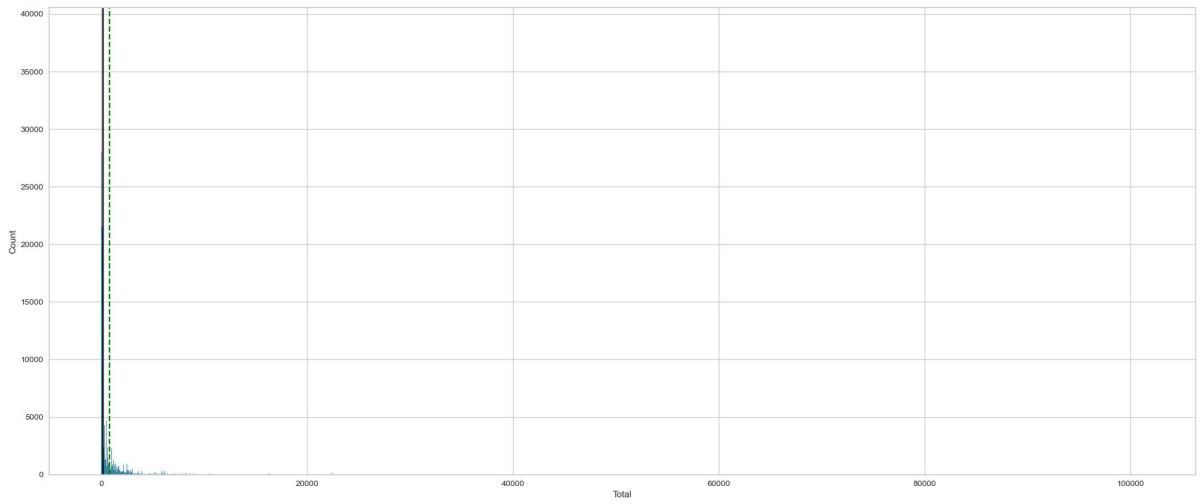
EXPLORATORY DATA ANALYSIS

UNIVBARIATE ANALYSIS

In [21]:
```
# While doing univariate analysis of numerical variables we want to study their cer
# Let us write a function that will help us create boxplot and histogram for any in
# This function takes the numerical column as the input and retrurns the boxplots a
# Let see if this help us write faster and clearer code.

def histogram_boxplot(feature, figsize=(25,15), bins = None):
    """ Boxplot and histogram combined
    feature: 1-d feature array
    figsize: size of fig (default (20,12))
    bins: number of bins (default None / auto)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(nrows = 2, # Number of rows of the subpl
                                           sharex = True, # x-axis will be shared am
                                           gridspec_kw = {"height_ratios": (.25, .75
                                           figsize = figsize
                                           )# creating the 2 subplots
    sns.boxplot(x=feature, ax=ax_box2, showmeans=True, color='violet') # boxplot wi
    sns.histplot(x=feature, kde=F, ax=ax_hist2, bins=bins, palette="winter") if bin
    ax_hist2.axvline(np.mean(feature), color='green', linestyle='--')# Add mean to
    ax_hist2.axvline(np.median(feature), color='black', linestyle='-') #Add median
```

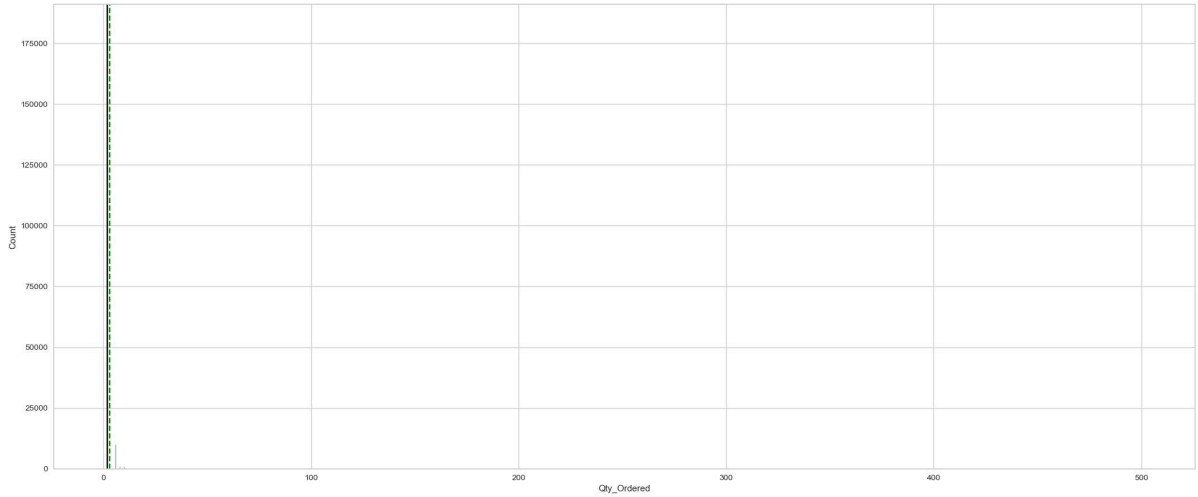
In [22]:
```
# check for outliers
histogram_boxplot(df['Total'])
```





In [ ]:
```
```

In [23]:
```
# check for outliers
histogram_boxplot(df['Qty_Ordered'])
```

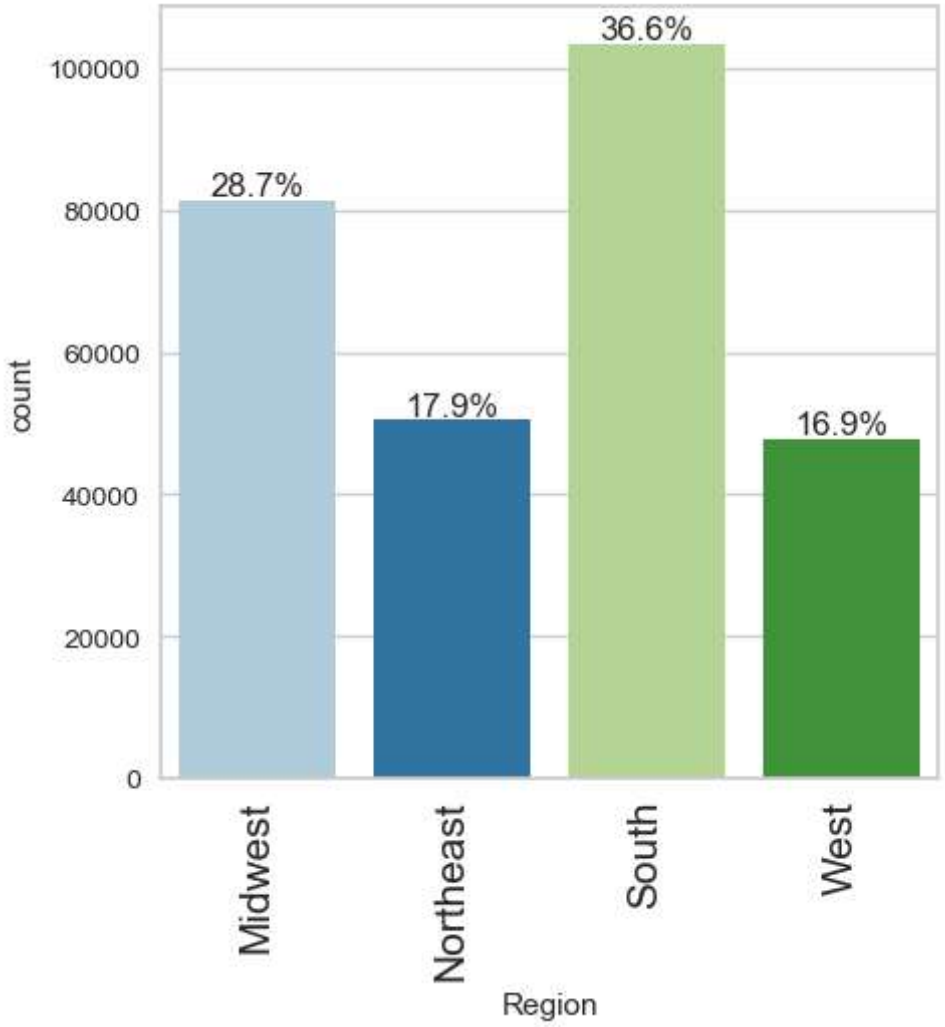


In [ ]:

In [24]:
```
# function to create labeled barplots

def labeled_barplot(data, feature, perc=False, n=None):
    """

    Barplot with p[ercentage at the top

    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of count (default is False)
    n: display the top n cvategory levels (default is None, i.e., display all level
    """

    total = len(data[feature]) # Length of the column
    count = data[feature]. nunique()
    if n is None:
        plt.figure(figsize=(count + 1, 5))
    else:
        plt.figure(figsize=(n + 1, 5))

    plt.xticks(rotation=90, fontsize=15)
    ax = sns.countplot(
        data=data,
    x=feature,
    palette="Paired",
    order=data[feature].value_counts().index[:n].sort_values(),
    )

    for p in ax.patches:
        if perc == True:
            label = "{:.1f}%".format(
                100 * p.get_height() / total
            ) # percentage of each class of the category
        else:
            label = p.get_height() #count of each level of the category
        x = p.get_x() + p.get_width() /2 # width of the plot
        y = p.get_height() # height of the plot
```

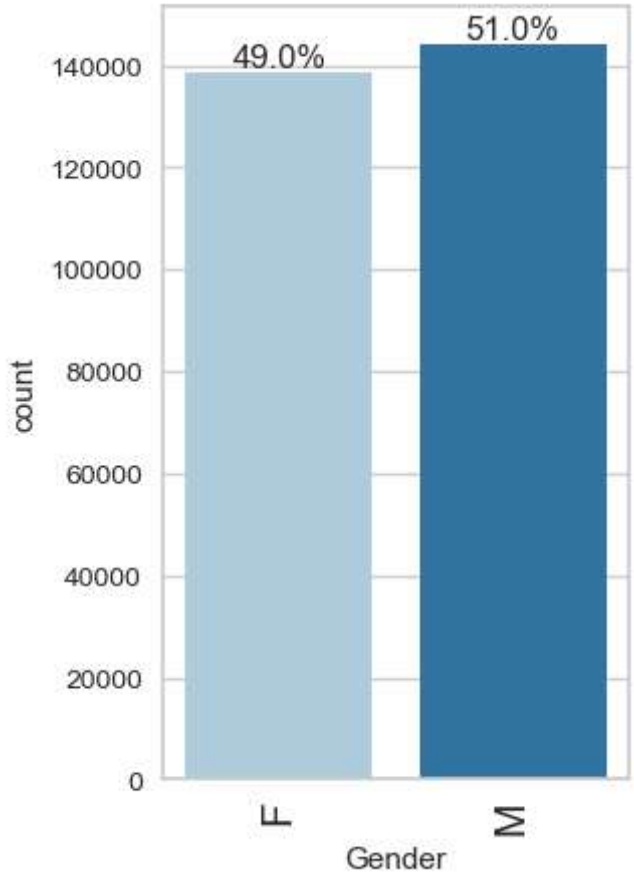
```
        ax.annotate(
          label,
          (x,y),
          ha="center",
          va="center",
          size=12,
          xytext=(0,5),
          textcoords="offset points",
        ) # annotate the percentage
    plt.show # show the plot
```

In [25]:
```
labeled_barplot(df, "Region", perc=True)
```



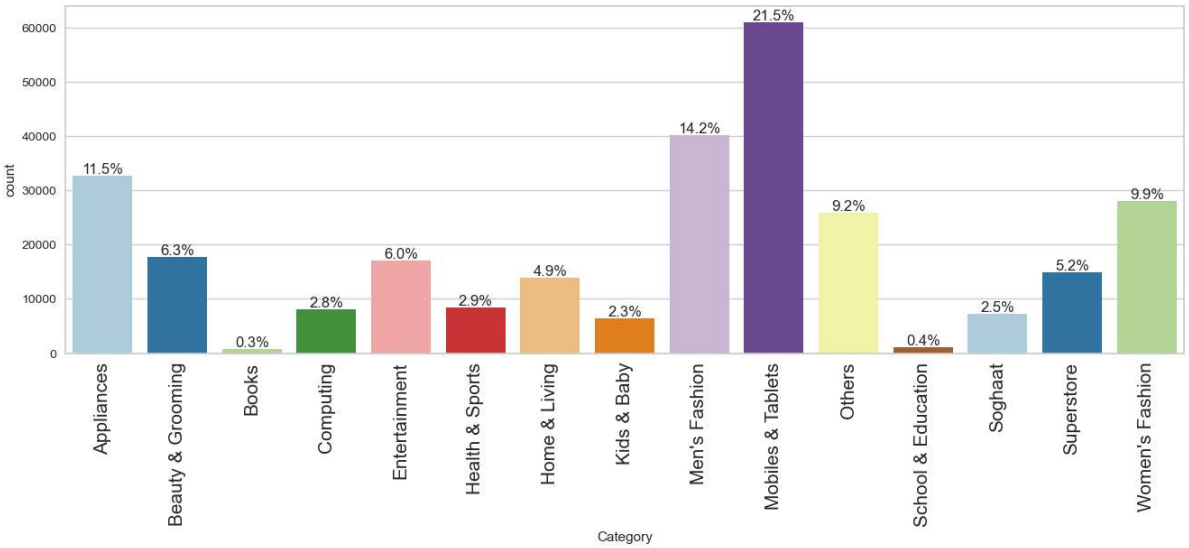
In [ ]:
```
```

In [26]:
```
labeled_barplot(df, "Gender", perc=True)
```



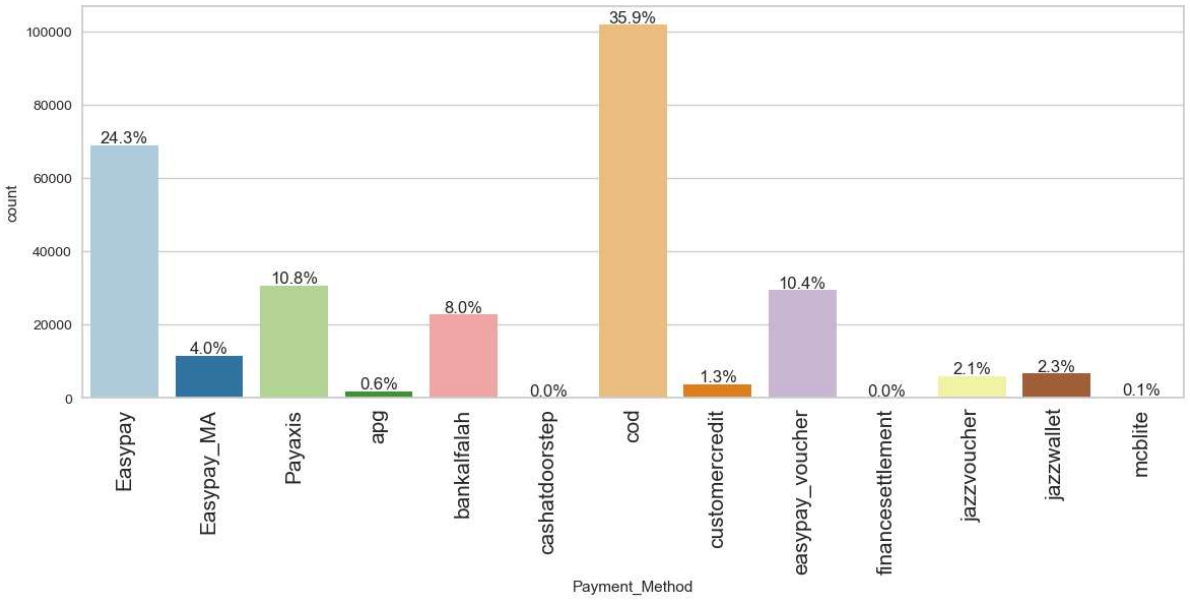
In [ ]:

In [27]:
```
labeled_barplot(df, "Category", perc=True)
```



In [ ]:

In [28]:
```
labeled_barplot(df, "Payment_Method", perc=True)
```

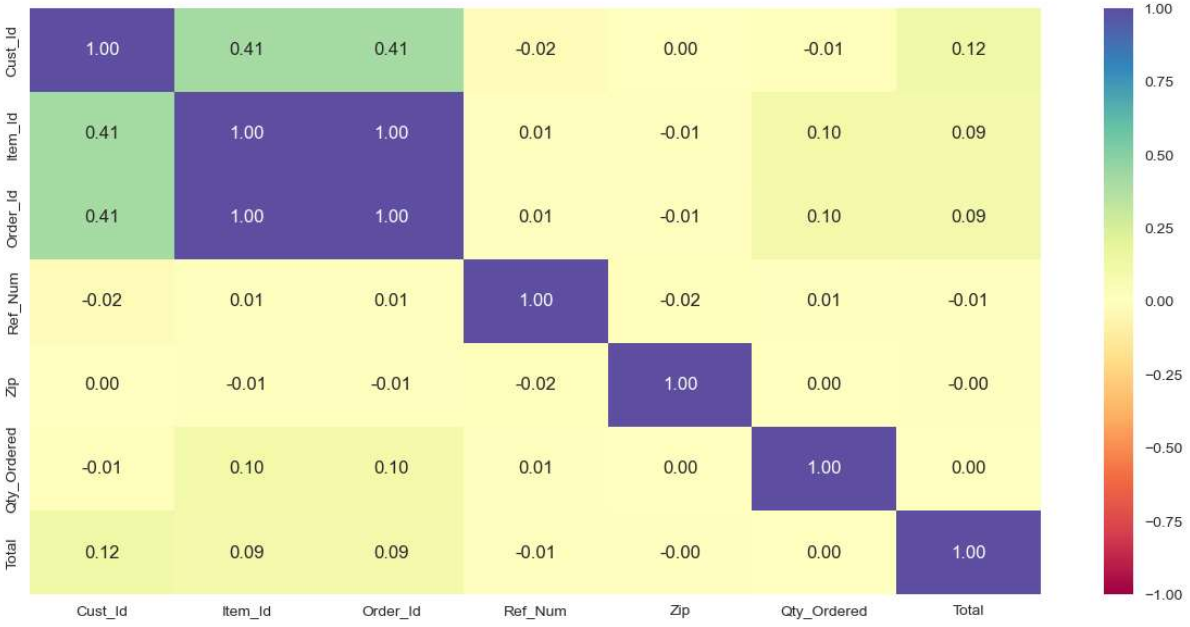


In [ ]:

BIVARIATE ANALYSIS

Let's look at the correlations.

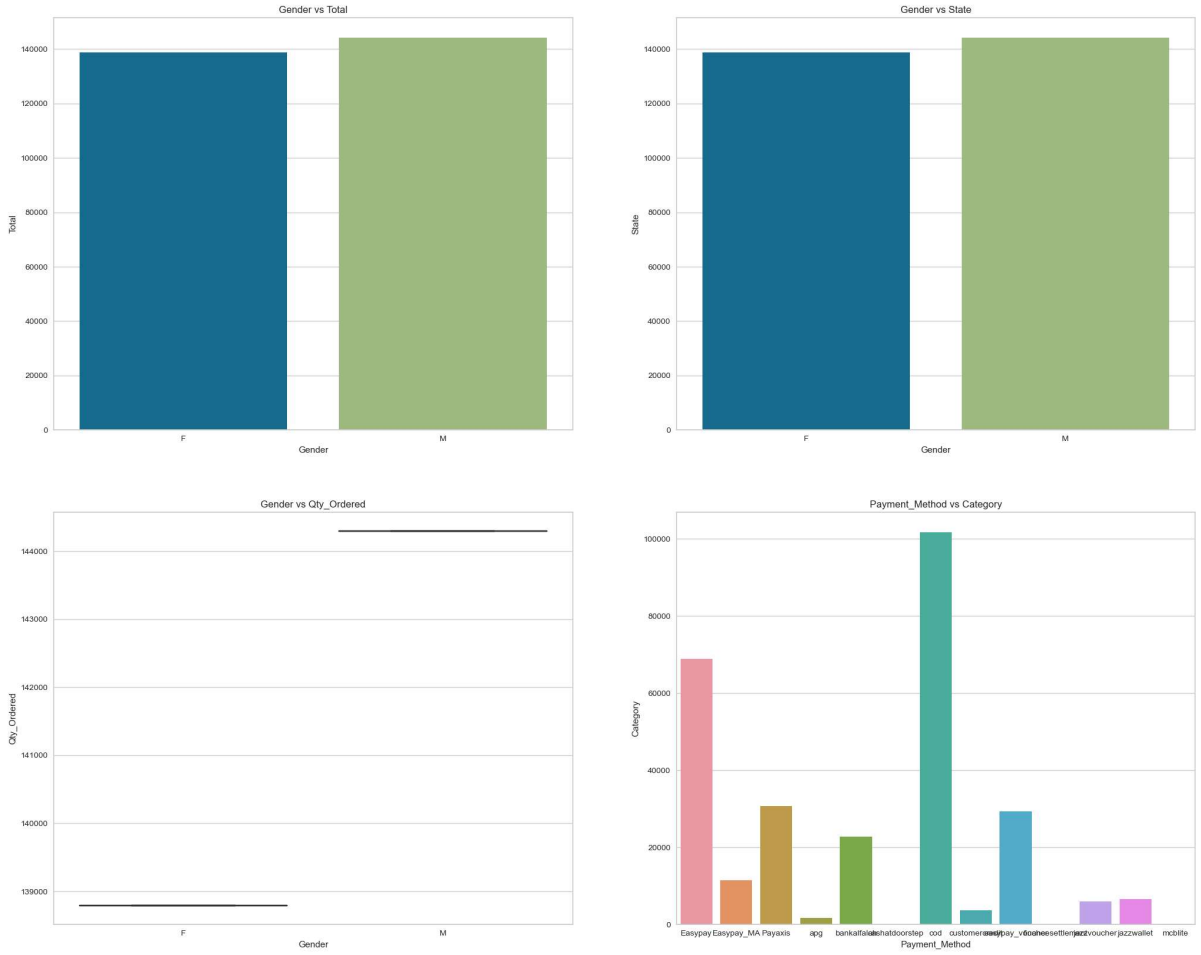
In [29]:
```
plt.figure(figsize=(15,7))
sns.heatmap(
    df.corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral"
)
plt.show()
```



In [ ]:

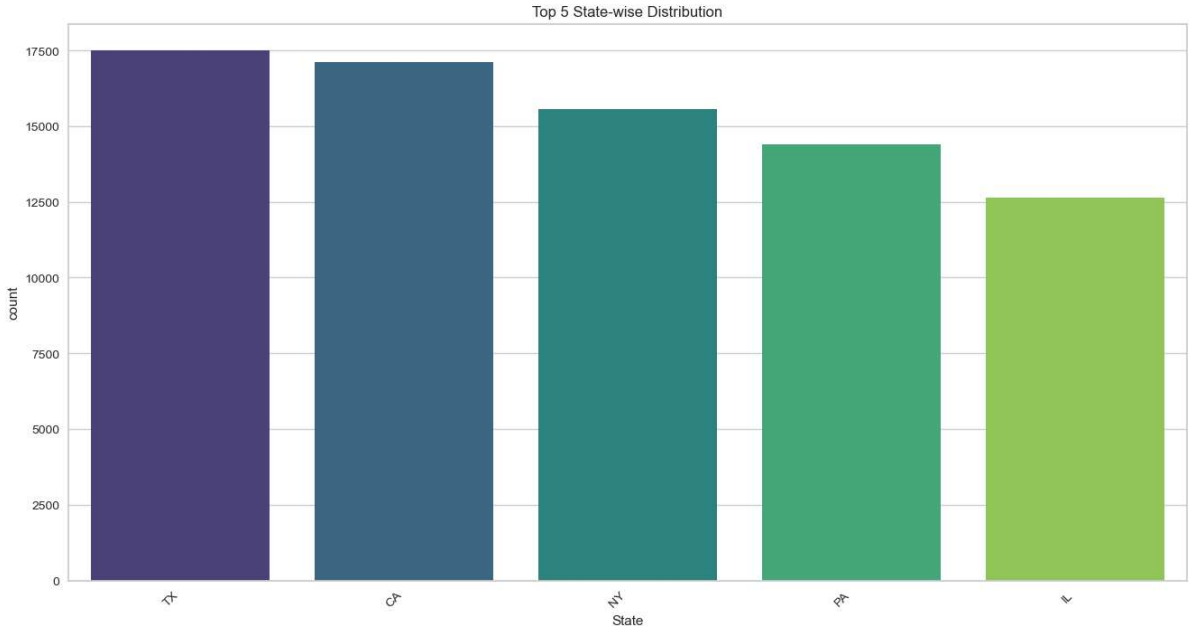
In [34]:
```
fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(25,20))

Gender_Total = df.groupby("Gender")["Total"].count().reset_index()
sns.barplot(x="Gender", y="Total",  data=Gender_Total, ax=axs[0,0])
axs[0,0].set_title("Gender vs Total");

Gender_Monetary = df.groupby("Gender")["State"].count().reset_index()
```

```
sns.barplot(x="Gender", y="State", data=Gender_Monetary, ax=axs[0,1])
axs[0,1].set_title("Gender vs State");

Gender_Qty_Ordered = df.groupby("Gender")["Qty_Ordered"].count().reset_index()
sns.boxplot(x="Gender", data=Gender_Qty_Ordered, y="Qty_Ordered", ax=axs[1,0])
axs[1,0].set_title("Gender vs Qty_Ordered");

Payment_Method_Category = df.groupby("Payment_Method")["Category"].count().reset_ir
sns.barplot(x="Payment_Method", data=Payment_Method_Category, y="Category", ax=axs[
axs[1,1].set_title("Payment_Method vs Category");
```



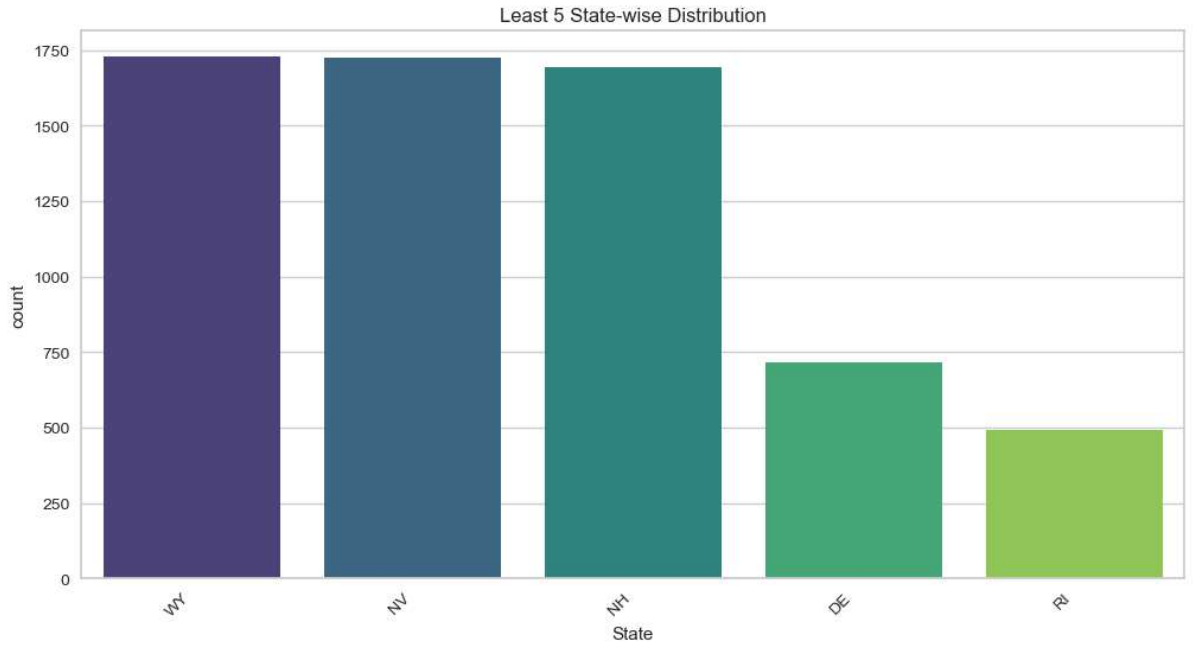
In [ ]:

In [37]:
```
### State-wise Distribution
# Get the top 5 states based on customer count
top_5_states = data['State'].value_counts().head(5).index
# Filter the DataFrame for the top 15 states
data_top_5_states = data[data['State'].isin(top_5_states)]
# Bar chart for top 5 state-wise distribution
plt.figure(figsize=(16, 8))
sns.countplot(x='State', data=data_top_5_states, palette='viridis', order=top_5_sta
plt.title('Top 5 State-wise Distribution')
plt.xticks(rotation=45, ha='right')
plt.show()
```



In [ ]:

In [38]:
```
# Get the least 5 states based on customer count
least_5_states = data['State'].value_counts().tail(5).index
# Filter the DataFrame for the least 5 states
data_least_5_states = data[data['State'].isin(least_5_states)]
# Bar chart for least 5 state-wise distribution
plt.figure(figsize=(12, 6))
sns.countplot(x='State', data=data_least_5_states, palette='viridis', order=least_5
plt.title('Least 5 State-wise Distribution')
plt.xticks(rotation=45, ha='right')
plt.show()
```



In [ ]:

In [49]:
```
# Calculate the average of numerical variables
avg_qty_ordered = df['Qty_Ordered'].mean()
avg_total_sales = df['Total'].mean()

print(f'Average Qty_Ordered: {avg_qty_ordered}')
print(f'Average Total Sales: {avg_total_sales}')
```

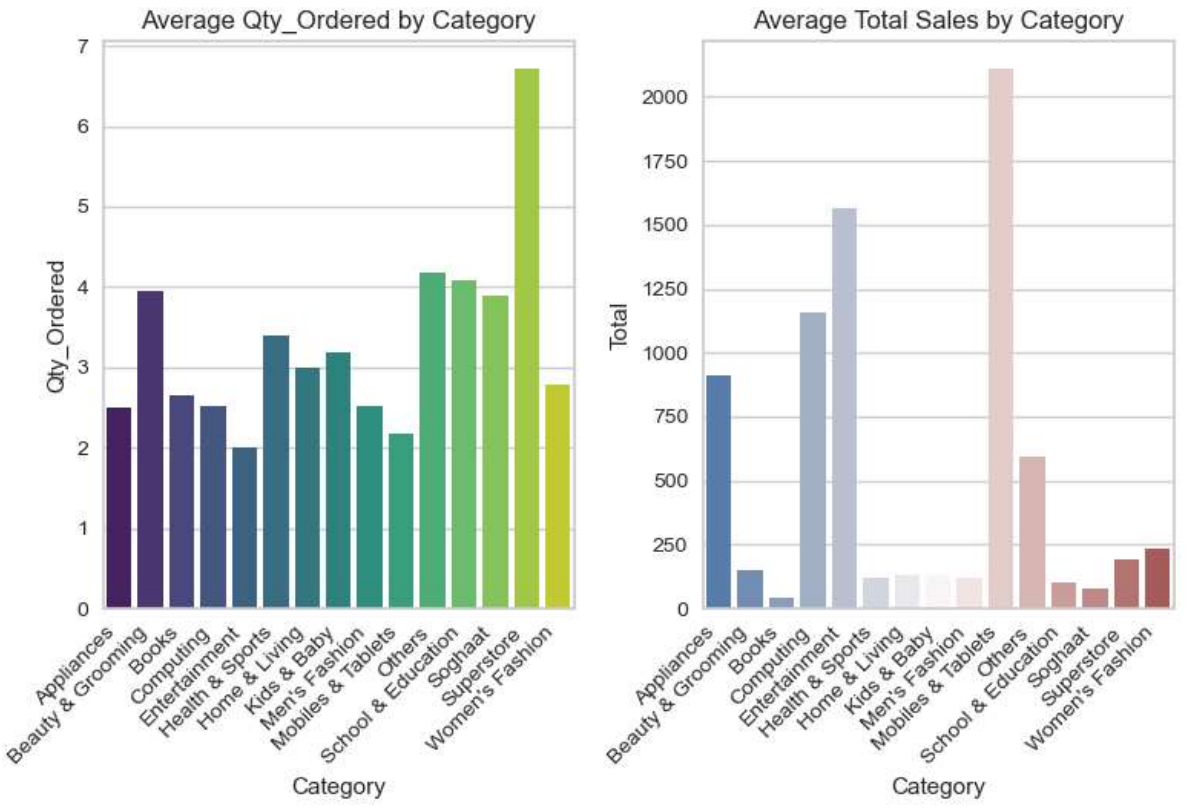
```
Average Qty_Ordered: 3.008223736501309
Average Total Sales: 816.2307119361458
```

In [50]:
```
# Calculate average values by category
avg_values_by_category = df.groupby('Category')[['Qty_Ordered', 'Total']].mean().re
# Bar chart for average quantity ordered and total sales by category
plt.figure(figsize=(12, 6))
```

Out[50]:
```
<Figure size 1200x600 with 0 Axes>
```

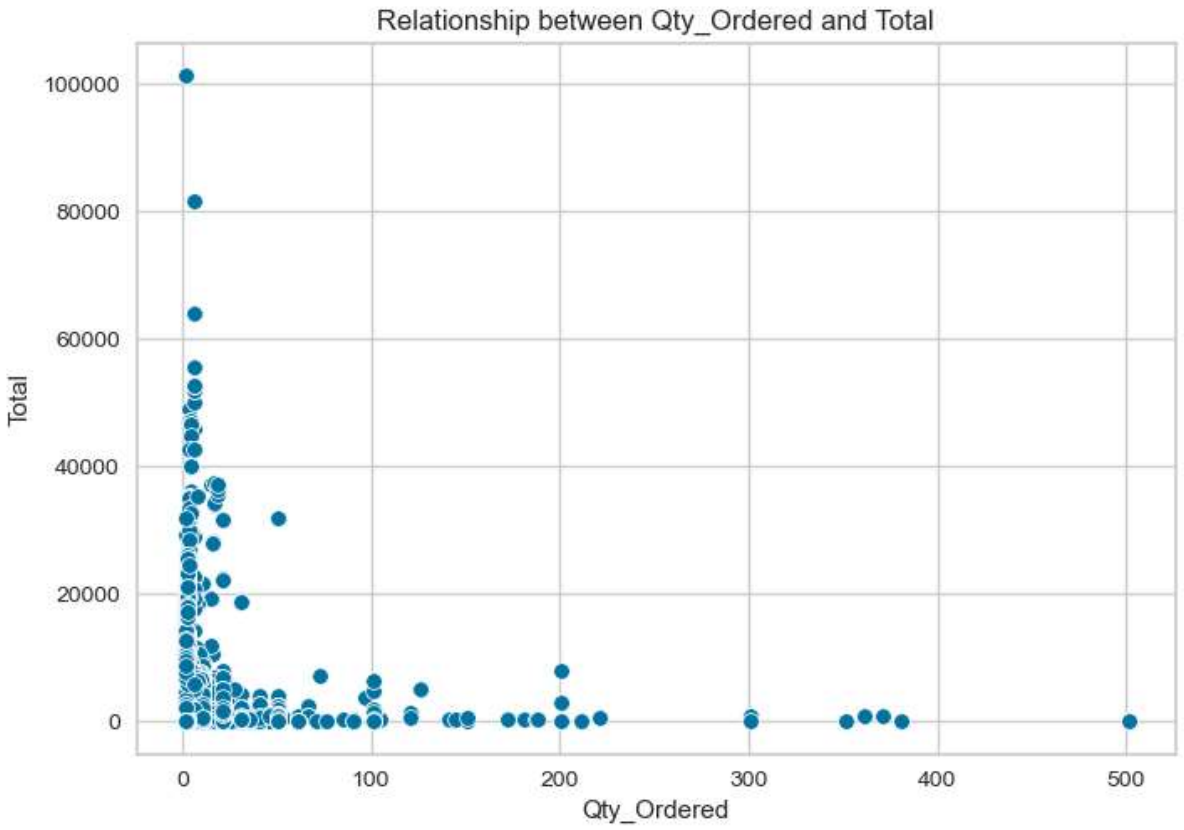
```
<Figure size 1200x600 with 0 Axes>
```

In [48]:
```
# Bar chart for Average Qty Ordered
plt.subplot(1, 2, 1)
sns.barplot(x='Category', y='Qty_Ordered', data=avg_values_by_category, palette='vi
plt.title('Average Qty_Ordered by Category')
plt.xticks(rotation=45, ha='right')
# Bar chart for Average Total Sales
plt.subplot(1, 2, 2)
sns.barplot(x='Category', y='Total', data=avg_values_by_category, palette='vlag')
plt.title('Average Total Sales by Category')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



In [ ]:
```
```

In [109…
```
sns.scatterplot(x='Qty_Ordered', y='Total', data=df)
plt.xlabel('Qty_Ordered')
plt.ylabel('Total')
plt.title('Relationship between Qty_Ordered and Total')
plt.show()
```

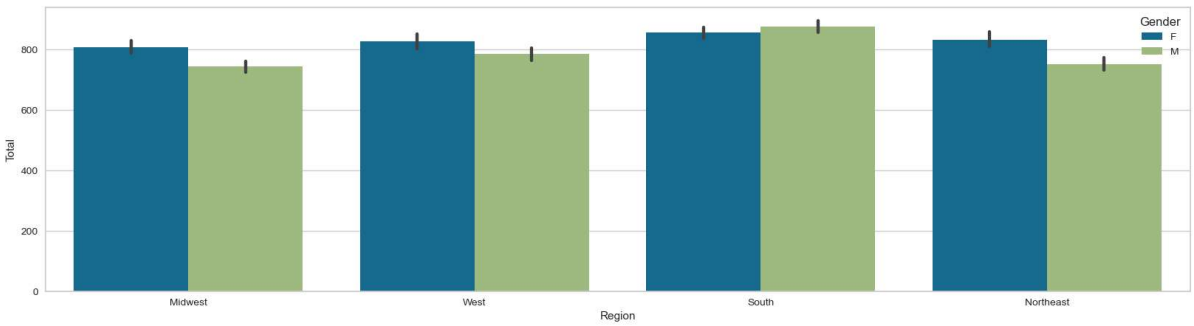


In [ ]:

MULTIVARIATE ANALYSIS

In [106…
```
plt.figure(figsize=(20,5))
sns.barplot(x='Region', y='Total', data=df, hue='Gender')
```

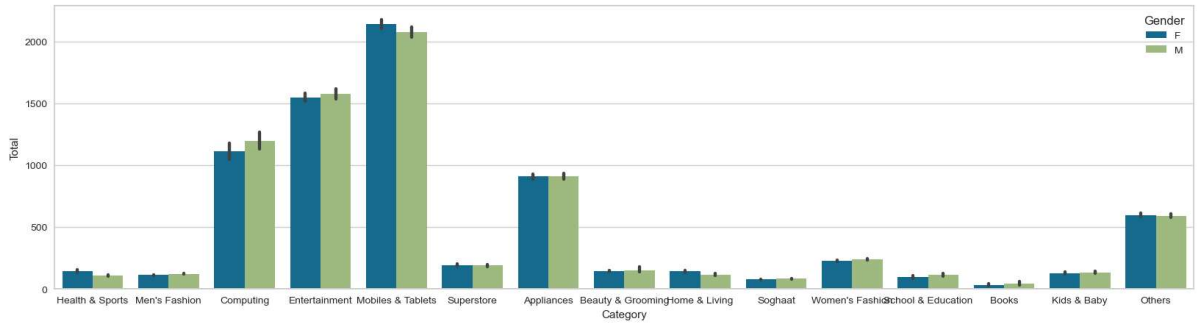
Out[106]:
```
<Axes: xlabel='Region', ylabel='Total'>
```



In [ ]:

In [107…
```
plt.figure(figsize=(20,5))
sns.barplot(x='Category', y='Total', data=df, hue='Gender')
```

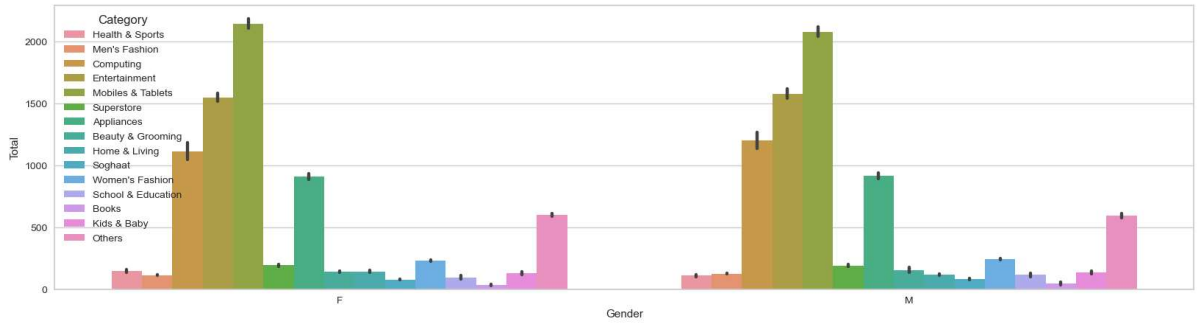
Out[107]:
```
<Axes: xlabel='Category', ylabel='Total'>
```



In [ ]:

In [108…
```
plt.figure(figsize=(20,5))
sns.barplot(x='Gender', y='Total', data=df, hue='Category')
```

Out[108]:
```
<Axes: xlabel='Gender', ylabel='Total'>
```



In [ ]:

In [ ]:

FEATURE ENGINEERING

In [52]:
```
# finding out each cx rfm values

#Recency
day='2022-10-01'
day=pd.to_datetime(day)
df['Date_of_Order']=pd.to_datetime(df["Date_of_Order"])

recency=df.groupby(["Cust_Id"]).agg({"Date_of_Order":lambda x:((day-x.max()).days)}
recency
```

Out[52]:

Cust_Id	Date_of_Order
4	2
15	232
16	323
20	2
21	240
...	...
115322	1
115323	1
115324	1
115325	1
115326	1

63646 rows × 1 columns

In [ ]:

```
```

In [53]:

```
# Frequency
freq=df.drop_duplicates(subset="Order_Id").groupby(["Cust_Id"])[["Order_Id"]].count
```

In [54]:

```
freq.head()
```

Out[54]:

Cust_Id	Order_Id
4	29
15	3
16	3
20	11
21	1

In [55]:

```
#Monetary
df["total"]=df["Qty_Ordered"]*df["Category"]
```

In [56]:

```
money=df.groupby(["Cust_Id"])[["Total"]].sum()
money
```

Out[56]:

Cust_Id	Total
4	27394.190
15	216.800
16	11868.899
20	28719.018
21	105.000
...	...
115322	209.600
115323	4419.900
115324	39.900
115325	89.900
115326	3559.900

63646 rows × 1 columns

In [ ]:

```
```

In [58]:

```
RFM =pd.concat([recency,freq,money], axis=1)
recency.columns=["Recency"]
freq.columns=["Frequency"]
money.columns=["Monetary"]
RFM
```

Out[58]:

Cust_Id	Recency	Frequency	Monetary
4	2	29	27394.190
15	232	3	216.800
16	323	3	11868.899
20	2	11	28719.018
21	240	1	105.000
...	...	...	...
115322	1	2	209.600
115323	1	1	4419.900
115324	1	1	39.900
115325	1	2	89.900
115326	1	1	3559.900

63646 rows × 3 columns

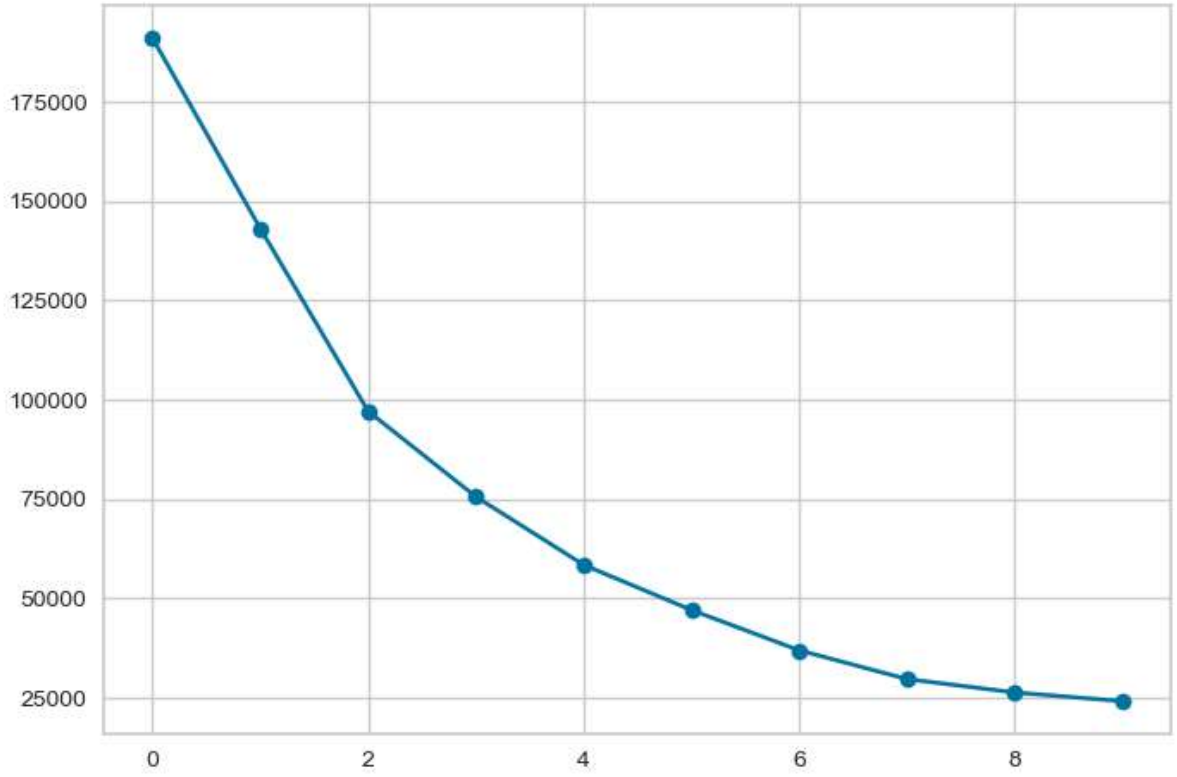
In [59]:

```
from sklearn.preprocessing import StandardScaler
```

```
scaler=StandardScaler()
scaled=scaler.fit_transform(RFM)
```

In [60]:
```
from sklearn.cluster import KMeans

inertia=[]
for i in np.arange(1,11):
    kmeans=KMeans(n_clusters=i)
    kmeans.fit(scaled)
    inertia.append(kmeans.inertia_)

plt.plot(inertia,marker="o")
```

Out[60]:
```
[<matplotlib.lines.Line2D at 0x2350512d410>]
```



In [86]:
```
kmeans = KMeans(n_clusters=3)
kmeans.fit(scaled)
RFM["Clusters"]=(kmeans.labels_+1)
```

In [87]:
```
RFM
```

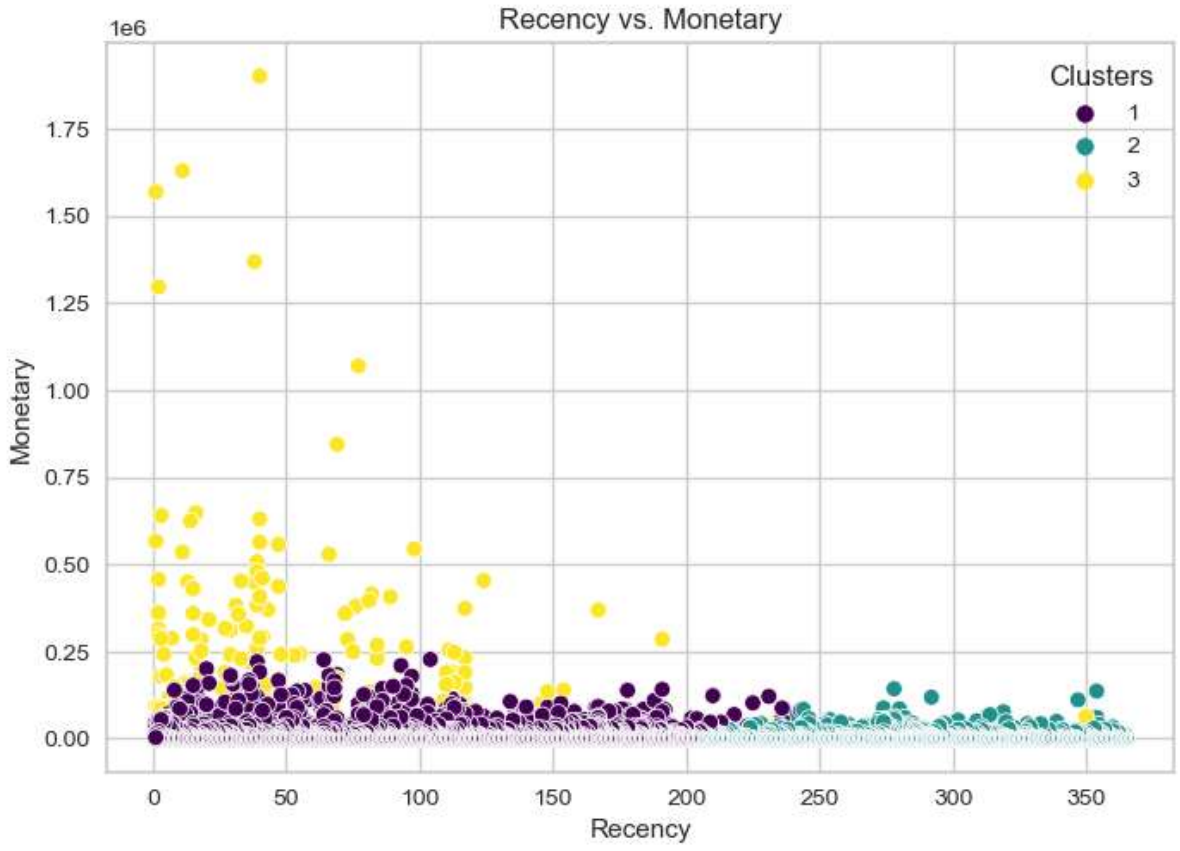
Out[87]:

Cust_Id	Recency	Frequency	Monetary	Clusters
4	2	29	27394.190	1
15	232	3	216.800	2
16	323	3	11868.899	2
20	2	11	28719.018	1
21	240	1	105.000	2
...	...	...	...	...
115322	1	2	209.600	1
115323	1	1	4419.900	1
115324	1	1	39.900	1
115325	1	2	89.900	1
115326	1	1	3559.900	1

63646 rows × 4 columns

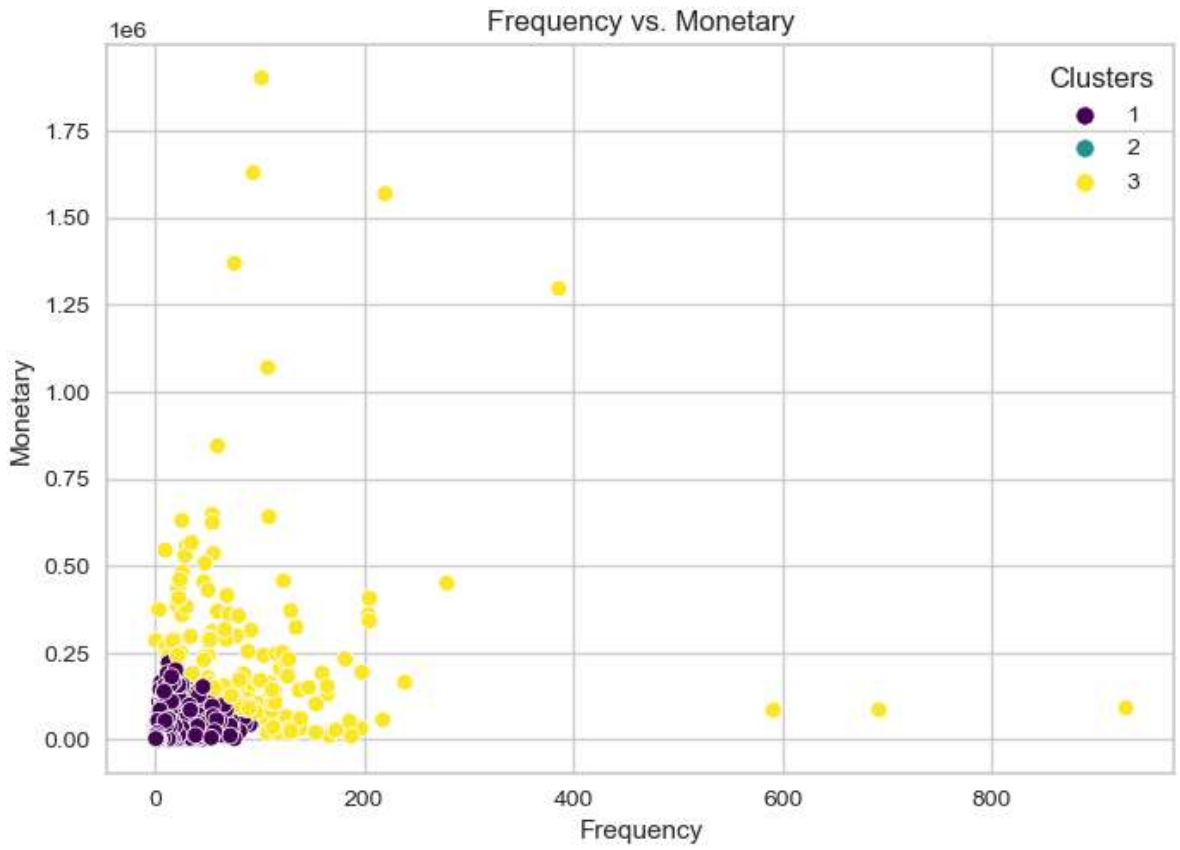
In [ ]:

In [88]:
```
sns.scatterplot(data=RFM, x='Recency', y='Monetary', hue='Clusters', palette='viric
plt.title('Recency vs. Monetary')
plt.xlabel('Recency')
plt.ylabel('Monetary')
plt.show()
```



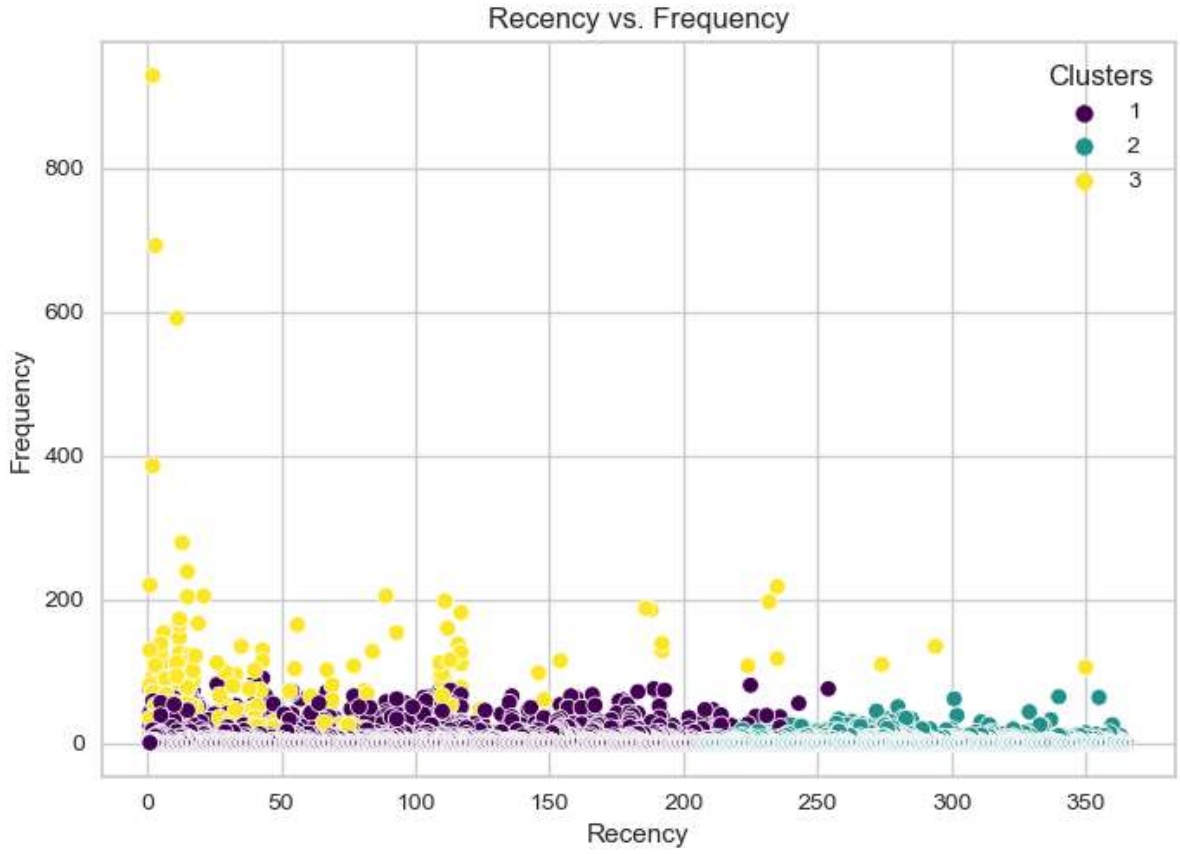
In [ ]:

In [89]:
```
sns.scatterplot(data=RFM, x='Frequency', y='Monetary', hue='Clusters', palette='vir
plt.title('Frequency vs. Monetary')
plt.xlabel('Frequency')
plt.ylabel('Monetary')
plt.show()
```



In [ ]:
```
```

In [90]:
```
sns.scatterplot(data=RFM, x='Recency', y='Frequency', hue='Clusters', palette='viri
plt.title('Recency vs. Frequency')
plt.xlabel('Recency')
plt.ylabel('Frequency')
plt.show()
```



In [91]:
```
group=RFM.groupby(["Clusters"])["Recency","Frequency","Monetary"].mean()
group
```

Out[91]:

Clusters	Recency	Frequency	Monetary
1	125.948176	3.428584	3752.510996
2	281.486718	2.200178	1971.621907
3	59.707792	104.493506	277608.084147

In [102…
```
def func(row):
    if row["Clusters"]==1:
        return 'Gold'
    elif row["Clusters"]==2:
        return 'Silver'
    else:
        return 'Platinum'
```

In [103…
```
RFM["Conditions"]=RFM.apply(func,axis=1)
RFM
```

Out[103]:

Cust_Id	Recency	Frequency	Monetary	Clusters	Conditions
4	2	29	27394.190	1	Gold
15	232	3	216.800	2	Silver
16	323	3	11868.899	2	Silver
20	2	11	28719.018	1	Gold
21	240	1	105.000	2	Silver
...	...	...	...	...	...
115322	1	2	209.600	1	Gold
115323	1	1	4419.900	1	Gold
115324	1	1	39.900	1	Gold
115325	1	2	89.900	1	Gold
115326	1	1	3559.900	1	Gold

63646 rows × 5 columns

In [ ]:

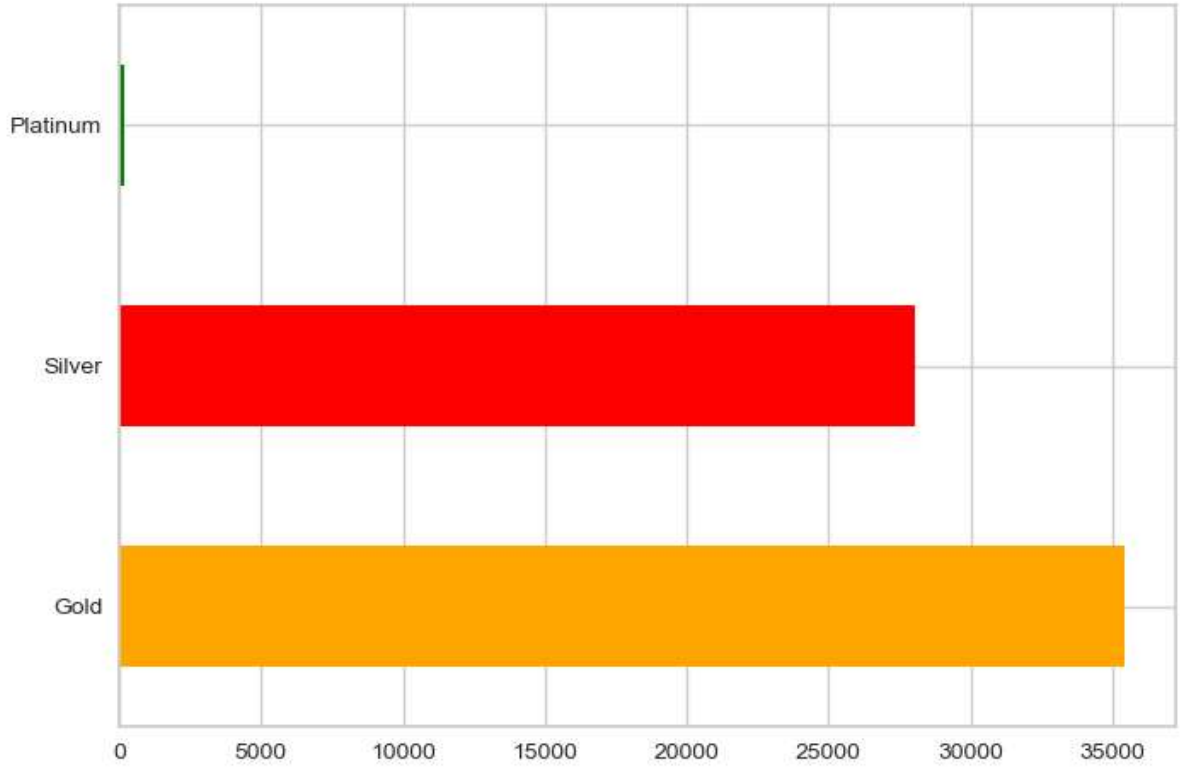
In [104…
```
result=RFM["Conditions"].value_counts()
result
```

Out[104]:
```
Gold        35447
Silver      28045
Platinum      154
Name: Conditions, dtype: int64
```

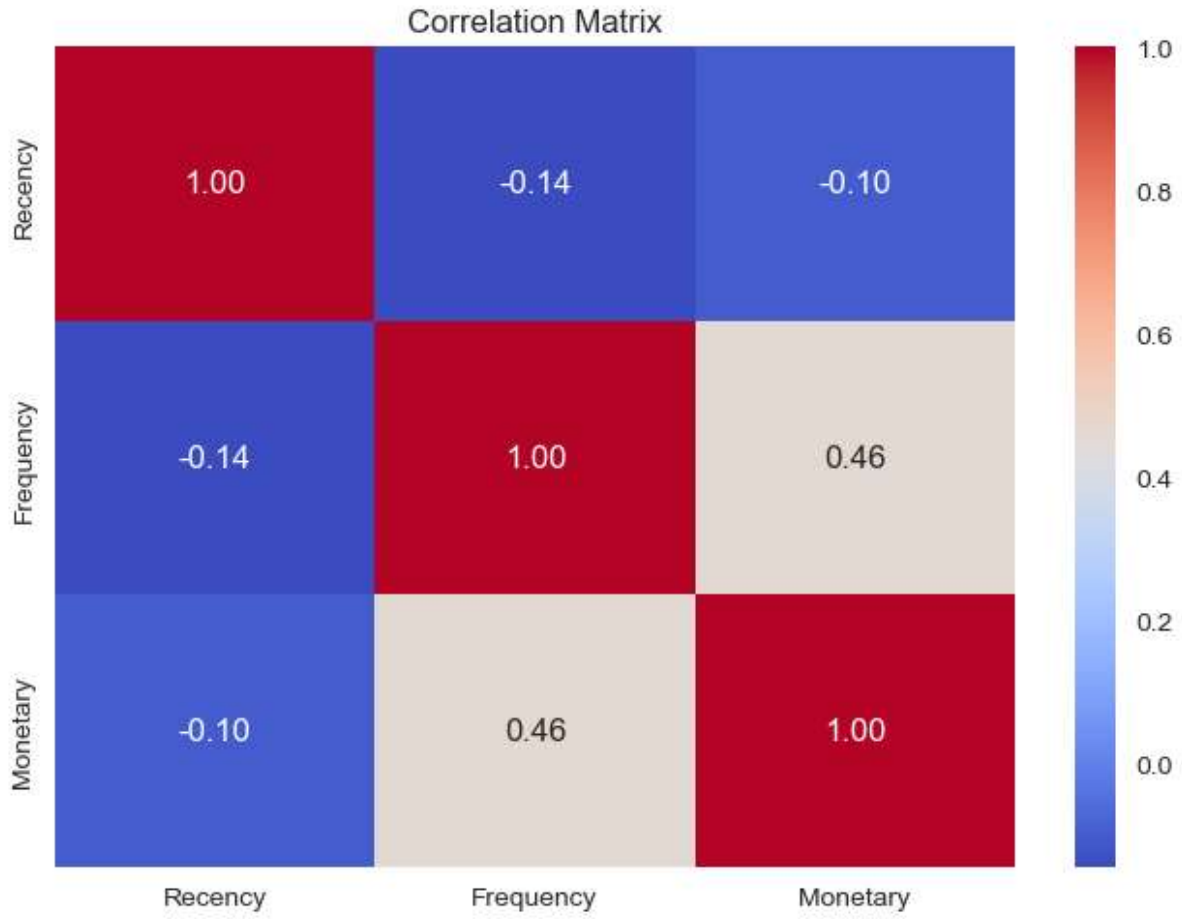
In [ ]:

In [105…
```
result.plot(kind='barh', color=["Orange", "red","green"])
```

Out[105]:
```
<Axes: >
```



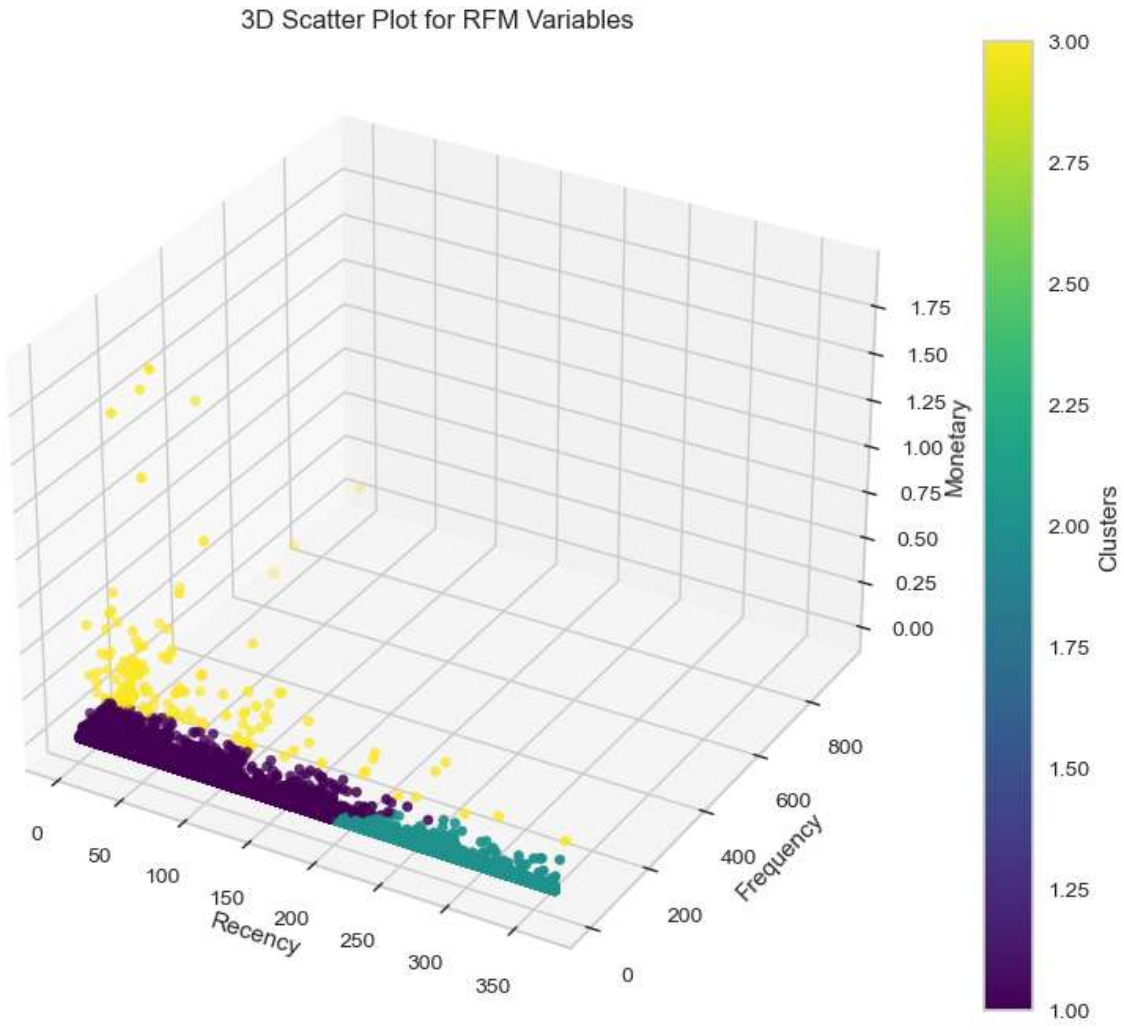
In [97]:
```
corr_matrix = RFM[['Recency', 'Frequency', 'Monetary']].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```



In [ ]:

In [101…

```
# Multivariate analysis with a 3D scatter plot
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(RFM['Recency'], RFM['Frequency'], RFM['Monetary'], c=RFM['Clus
ax.set_xlabel('Recency')
ax.set_ylabel('Frequency')
ax.set_zlabel('Monetary')
ax.set_title('3D Scatter Plot for RFM Variables')
fig.colorbar(scatter, ax=ax, label='Clusters')
plt.show()
```



In [ ]:

KEY INSIGHTS

Cluster 1 (GOLD)

- RECENCY : 126 days
- FREQUENCY : 3.4
- MONETARY : 3753

Cluster 2 (Silver)

- RECENCY : 281 days
- FREQUENCY : 2.2

- MONETARY : 1972

Cluster 3 (Platinum)

- RECENCY : 60
- FREQUENCY : 104.5
- MONETARY : 277608

CONCLUSION

This RFM customer segmentation analysis helps business allocate resources more effectively by providing a roadmap for strategic decision-making, enabling organizations to tailor marketing efforts, improve customer experiences, and maximize the value of each segment. By implementing targeted strategies for Silver, Gold, and Platinum customers, and helps to foster customer loyalty, drive revenue growth, and position your brand as a leader in the market.

RECOMMENDATIONS

Based on the key insights, we propose the following strategies for business growth:

- Regional Targeting: Given the regional distribution, focus marketing efforts on the South and Midwest regions, where a significant customer base exists.The model identify these region where they have the most loyal customer Tailor promotions and campaigns to resonate with the preferences of customers in these regions.
- Payment Method Optimization:Recognize the dominance of "cod" transactions and EasyPay. Consider incentivizing the use of other payment methods to diversify and streamline the payment process for both the business and customers.

Based on the average of each cluster, here key insight and recommendations for each segment:

GOLD Segment (Cluster 1):

Recency: Customers in this segment have a moderate recency score (126 days), indicating recent visits. Ensure to maintain their engagement by providing personalized recommendations, exclusive offers, or early access to new products.

Frequency: The frequency is relatively low (3.4 times), suggesting that they make occasional purchases. Leverage this by introducing loyalty programs, referral bonuses, or special discounts for repeat purchases to foster customer loyalty. Focus on maintaining the loyalty of this segment through personalized offers and exclusive deals, given their moderate recency but higher frequency and monetary value!

Monetary: The monetary value is high ($3,753), implying they spent a significant amount. Capitalize on this by offering premium products, VIP access, or personalized services to enhance their shopping experience.

The monetary value is high ($3,911.09), implying they spent a significant amount. Capitalize on this by offering premium products, VIP access, or personalized services to enhance their shopping experience.

SILVER Segment (Cluster 2)

Recency: Customers in this segment have a high recency score (281 days), indicating that they haven't visited the platform recently. It's recommended to implement targeted re-engagement strategies such as personalized promotions, discounts, or reminders to encourage them to return and make a purchase. With a higher recency compared to the other segments, consider targeted promotions or loyalty programs to re-engage this group.

Frequency: The frequency is high (2.2 times), suggesting they make occasional purchase. Consider offering loyalty programs, exclusive deals, or product recommendations based on their past purchases to increase their engagement and encourage more frequent transactions. Craft compelling campaigns to convert occasional customers into regular ones. Highlighting the potential benefits and value propositions can be effective.

Monetary: The monetary value is moderate ($1,972), so efforts can be made to upsell or cross-sell to increase the average order value. Special promotions or bundled deals might be effective in boosting their spending.

PLATINUM Segment (Cluster 3)

Recency: Customers in this segment have a very low recency score (60 days), indicating they have visited the platform recently. Implement targeted campaigns, personalized promotions, or exclusive offers to engage them.

Frequency: The frequency is extremely high (104.5 times), indicating they made frequent purchases in the past. Investigate reasons for the drop in frequency and tailor marketing efforts to reignite their interest, such as personalized recommendations or special incentives for returning customers. Build a close relationship with these top-tier customers through exclusive communications, seeking feedback, and involving them in brand initiatives.

Monetary: The monetary value is exceptionally high ($277,608), suggesting they have a significant lifetime value. Develop personalized loyalty programs, exclusive perks, or premium services to maintain their high spending and enhance their overall satisfaction. Prioritize VIP treatment for this high-value segment. Consider premium services, early access, or personalized experiences to enhance their loyalty further.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: