Data Science - Times Series

Description

Online shopping is, more than ever, part of people's daily lives. More than ever, using data to make decisions is the big bet of companies in the world.

Understanding the profile of the customer portfolio to generate personalized campaigns is imperative in a competitive world like today.

Furthermore, good sales planning is crucial for the supply sector chain to power the network and avoid problems such as sold out and overstocking.

Let's take on this challenge to work with a purchasing base via e-commerce that allows us to build views about customers and projections about future sales.

Goals

Customer segmentation

Construction of customer segmentation with a focus on value pyramid: Recency, Frequency and Value

Sales projection by segment

- Train a time series forecast model to forecast sales (Sales column) for the segment with the highest sales volume (across the entire history) - daily frequency
- Model validation: December 2014 (daily frequency)
- Use MAPE, SMAPE and RMSLE metrics to calculate the result in Dec 2014.

Data Environment

- conda install pandas openpyxl matplotlib seaborn scipy scikit-learn xgboost

Import Librarys

In [329]:

```
# Data Analysis Step
import datetime as dt
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

# plot parameters
%matplotlib inline
plt.rcParams['font.size'] = 14
plt.rcParams['figure.figsize'] = (9, 5)
plt.rcParams['figure.facecolor'] = '#00000000'

# outliers with mahalanobis
from scipy.spatial import distance

# Segmenetation Step
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans

# Machile Learning
import xgboost as xgb
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_error
from statsmodels.tools.eval_measures import rmse
```

In [7]:

```
# dataset from the Excel file
dataset = pd.read_excel('../data/dataset_desafio_datascience.xlsx')
df = dataset.copy()
```

Step 1 - Data Preparation and Investigation

Data Preparation

Notes:

- TARGET -> 'Sales', 'customer ID', 'segment', 'city', 'state', 'order date', 'ship date', 'profit'
- order and ship date are not in datetime format.

Understanding the data

In [10]:

```
def summary(df):
    """
    Display a summary of a DataFrame.

    This function displays the first few rows of the DataFrame, basic information about the DataFrame's struc
    and summary statistics including percentiles.

    Parameters:
    df (DataFrame): The pandas DataFrame to be summarized.

    Returns:
    None
    """

    display(f'Rows: {df.shape[0]}; Columns: {df.shape[1]}')
    display(df.head())
    print('-' * 100)
    display(df.info())
    print('-' * 100)
    display(df.describe([0.01, 0.25, 0.50, 0.75, 0.99]))
```

In [11]:

```
summary(df)
```

'Rows: 51290; Columns: 23'

	Order ID	Order Date	Ship Date	Ship Mode	customer ID	Customer Name	Segment	City	State	Country	...	Product ID	Cate
0	ES-2011-5338028	28-12-2011	01-01-2012	Second Class	18115	Mick Hernandez	Home Office	Turin	Piedmont	Italy	...	FUR-BO-10003541	Furn
1	ID-2011-24160	29-12-2011	01-01-2012	Second Class	20170	Sarah Bern	Consumer	Gold Coast	Queensland	Australia	...	TEC-PH-10002601	Techn
2	CA-2011-141313	28-12-2011	01-01-2012	Standard Class	10780	Anthony Jacobs	Corporate	Beverly	Massachusetts	United States	...	OFF-AP-10002651	Sup
3	ID-2011-24160	29-12-2011	01-01-2012	Second Class	20170	Sarah Bern	Consumer	Gold Coast	Queensland	Australia	...	FUR-BO-10002308	Furn
4	CA-2011-104738	30-12-2011	01-01-2012	Second Class	20620	Stefania Perrino	Corporate	Laredo	Texas	United States	...	TEC-PH-10002468	Techn

5 rows × 23 columns

```
----------------------------------------------------------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51290 entries, 0 to 51289
Data columns (total 23 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Order ID        51290 non-null  object
 1   Order Date      51290 non-null  object
 2   Ship Date       51290 non-null  object
 3   Ship Mode       51290 non-null  object
 4   customer ID     51290 non-null  int64
 5   Customer Name   51290 non-null  object
 6   Segment         51290 non-null  object
 7   City            51290 non-null  object
 8   State           51290 non-null  object
 9   Country         51290 non-null  object
 10  Postal Code     9994 non-null   float64
 11  Market          51290 non-null  object
 12  Region          51290 non-null  object
 13  Product ID      51290 non-null  object
 14  Category        51290 non-null  object
 15  Sub-Category    51290 non-null  object
 16  Product Name    51290 non-null  object
 17  Sales           51290 non-null  float64
 18  Quantity        51290 non-null  int64
 19  Discount        51290 non-null  float64
 20  Profit          51290 non-null  float64
 21  Shipping Cost   51290 non-null  float64
 22  Order Priority  51290 non-null  object
dtypes: float64(5), int64(2), object(16)
memory usage: 9.0+ MB
None
----------------------------------------------------------------------------------------------------
```

	customer ID	Postal Code	Sales	Quantity	Discount	Profit	Shipping Cost
count	51290.000000	9994.000000	51290.000000	51290.000000	51290.000000	51290.000000	51290.000000
mean	14037.790310	55190.379428	246.490581	3.476545	0.142908	28.610982	26.375915
std	5289.254612	32063.693350	487.565361	2.278766	0.212280	174.340972	57.296804
min	15.000000	1040.000000	0.444000	1.000000	0.000000	-6599.978000	0.000000
1%	600.000000	2149.000000	3.690000	1.000000	0.000000	-351.505650	0.200000
25%	11050.000000	23223.000000	30.758625	2.000000	0.000000	0.000000	2.610000
50%	14635.000000	56430.500000	85.053000	3.000000	0.000000	9.240000	7.790000
75%	18265.000000	90008.000000	251.053200	5.000000	0.200000	36.810000	24.450000
99%	21790.000000	98115.000000	2301.000000	11.000000	0.700000	587.359950	286.754300
max	21925.000000	99301.000000	22638.480000	14.000000	0.850000	8399.976000	933.570000

Selecting specific columns for analysis

In [12]:

```
df = df[['Order ID', 'Order Date', 'Ship Date', 'customer ID','Segment', 'City', 'State', 'Region', 'Country'
```

In [13]:

```
df.head()
```

Out[13]:

	Order ID	Order Date	Ship Date	customer ID	Segment	City	State	Region	Country	Market	Product ID	Category
0	ES-2011-5338028	28-12-2011	01-01-2012	18115	Home Office	Turin	Piedmont	South	Italy	EU	FUR-BO-10003541	Furniture
1	ID-2011-24160	29-12-2011	01-01-2012	20170	Consumer	Gold Coast	Queensland	Oceania	Australia	APAC	TEC-PH-10002601	Technology
2	CA-2011-141313	28-12-2011	01-01-2012	10780	Corporate	Beverly	Massachusetts	East	United States	US	OFF-AP-10002651	Office Supplies
3	ID-2011-24160	29-12-2011	01-01-2012	20170	Consumer	Gold Coast	Queensland	Oceania	Australia	APAC	FUR-BO-10002308	Furniture
4	CA-2011-104738	30-12-2011	01-01-2012	20620	Corporate	Laredo	Texas	Central	United States	US	TEC-PH-10002468	Technology

Normalize Date

In [14]:

```
# function, verify if is date format
def is_date_format(df, column):
    data_type = df[column].dtype

    if data_type == 'datetime64[ns]':
        print(f"The '{column}' column is in date format.")
    else:
        print(f"The '{column}' column is not in date format.")
```

In [15]:

```
is_date_format(df, 'Order Date')
```

The 'Order Date' column is not in date format.

In [16]:

```
is_date_format(df, 'Ship Date')
```

The 'Ship Date' column is not in date format.

In [17]:

```
# convert to date format
df['Order Date'] = pd.to_datetime(df['Order Date'], format='%d-%m-%Y', dayfirst=True)

is_date_format(df, 'Order Date')
df['Order Date']
```

The 'Order Date' column is in date format.

Out[17]:

```
0       2011-12-28
1       2011-12-29
2       2011-12-28
3       2011-12-29
4       2011-12-30
           ...
51285   2014-12-26
51286   2014-12-27
51287   2014-12-24
51288   2014-12-24
51289   2014-12-27
Name: Order Date, Length: 51290, dtype: datetime64[ns]
```

In [18]:

```
# convert to date format
df['Ship Date'] = pd.to_datetime(df['Ship Date'], format='%d-%m-%Y', dayfirst=True)

is_date_format(df, 'Ship Date')
df['Ship Date']
```

The 'Ship Date' column is in date format.

Out[18]:

```
0       2012-01-01
1       2012-01-01
2       2012-01-01
3       2012-01-01
4       2012-01-01
           ...
51285   2014-12-31
51286   2014-12-31
51287   2014-12-31
51288   2014-12-31
51289   2014-12-31
Name: Ship Date, Length: 51290, dtype: datetime64[ns]
```

Data Cleaning

Handling missing values

In [19]:

```
# Is there null valeus?
df.isna().sum()
```

Out[19]:

```
Order ID        0
Order Date      0
Ship Date       0
customer ID     0
Segment         0
City            0
State           0
Region          0
Country         0
Market          0
Product ID      0
Category        0
Product Name    0
Sales           0
Quantity        0
Profit          0
dtype: int64
```

Luckily I didn't need to remove null values as they are only present in the 'Postal Code' column, which will be used in the analysis.

Detecting and Removing Outliers

Box-plot

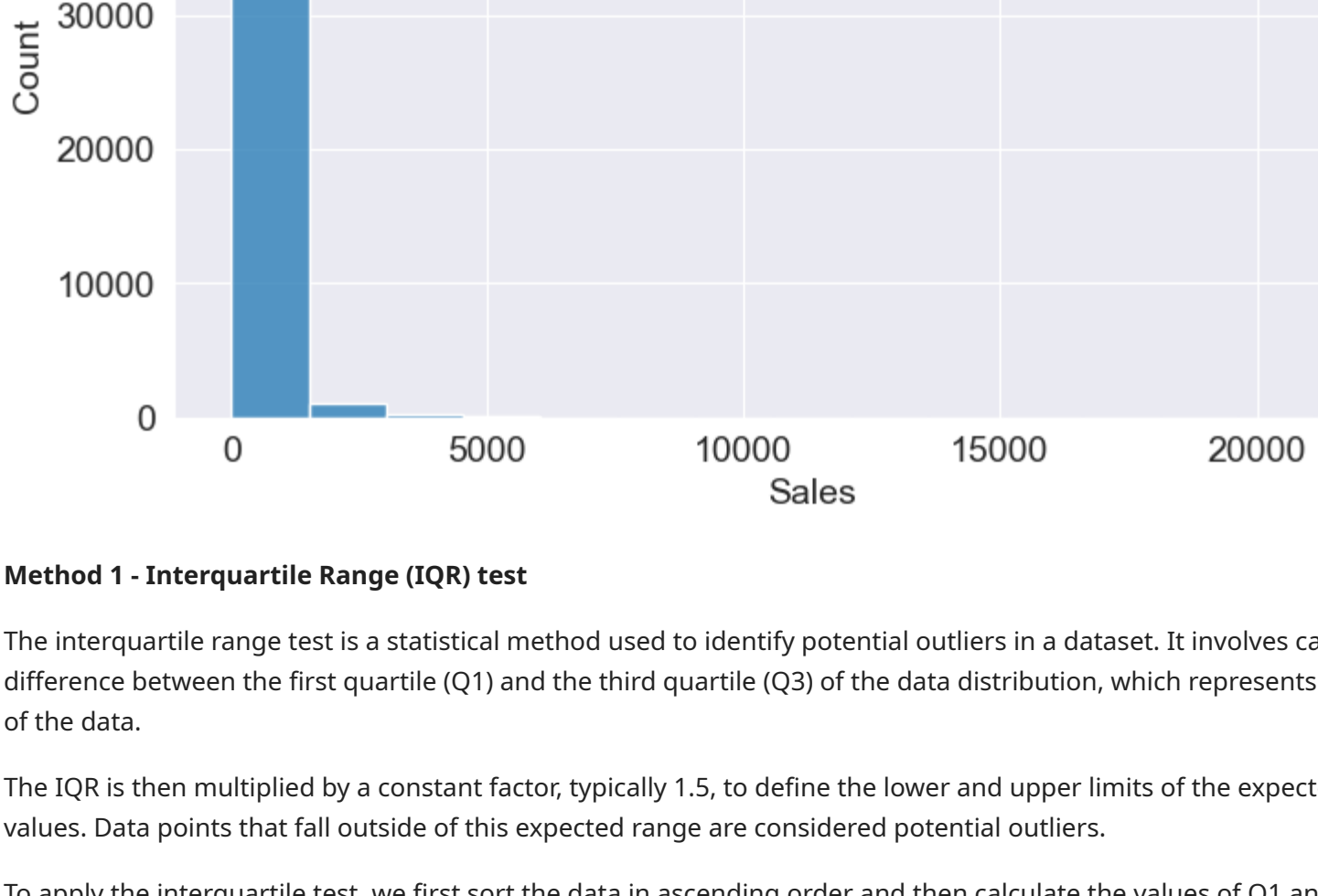
A box plot shows the distribution of the data and identify as any outliers as points that fall outside the "whiskers" of the plot.

In [25]:

```
# Box-plot function
def plot_boxplot(df, column):
    sns.boxplot(x = df[column])
    plt.show()

# Hist function
def plot_hist(df,column):
    sns.histplot(df[column], bins=15, kde=False)
    sns.set_style('darkgrid')
    plt.show()
```

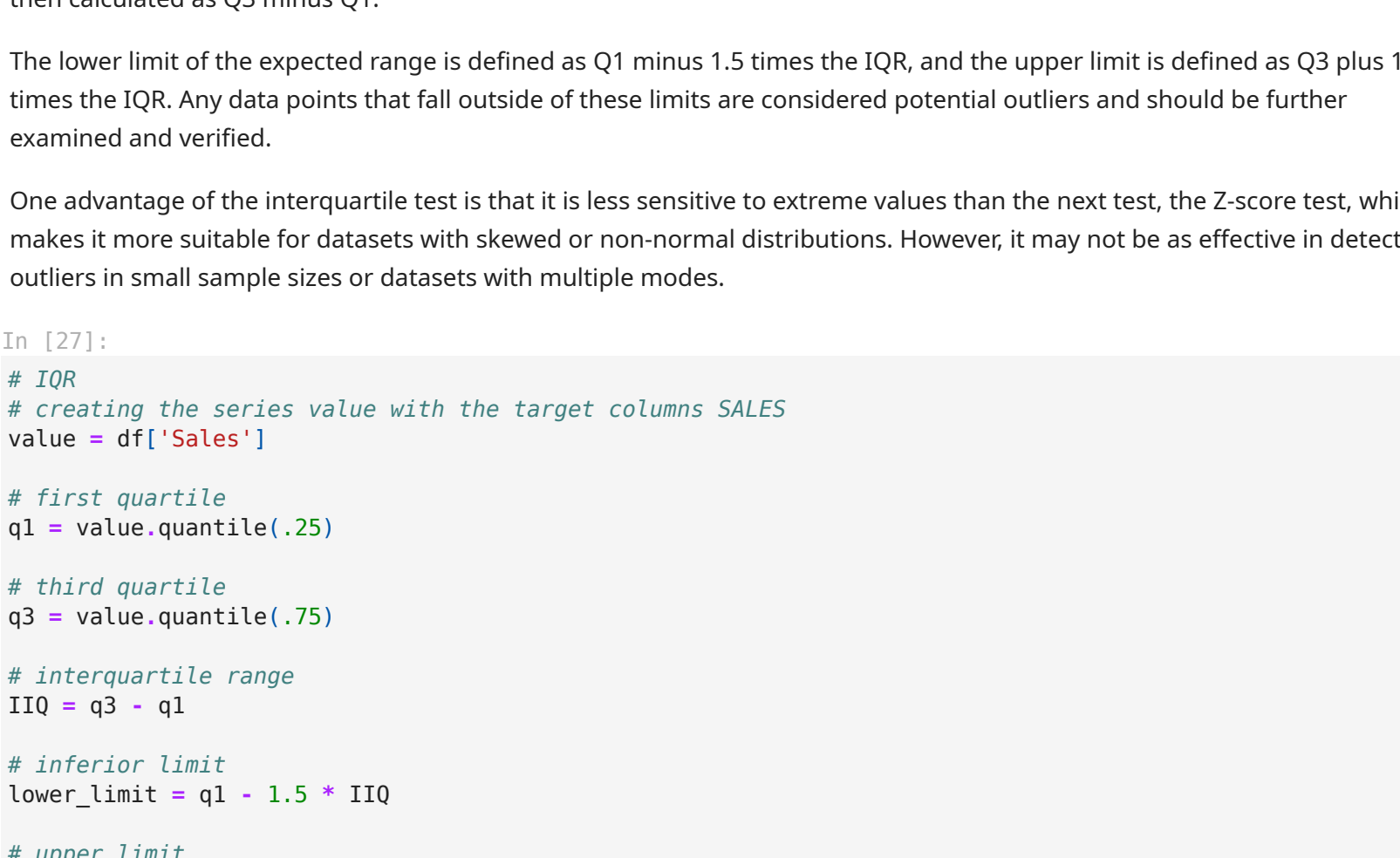
In [21]:

```
plot_boxplot(df, 'Sales')
```



In [26]:

```
plot_hist(df, 'Sales')
```



Method 1 - Interquartile Range (IQR) test

The interquartile range test is a statistical method used to identify potential outliers in a dataset. It involves calculating the difference between the first quartile (Q1) and the third quartile (Q3) of the data distribution, which represents the middle 50% of the data.

The IQR is then multiplied by a constant factor, typically 1.5, to define the lower and upper limits of the expected range of data values. Data points that fall outside of this expected range are considered potential outliers.

To apply the interquartile test, we first sort the data in ascending order and then calculate the values of Q1 and Q3. These can be calculated as the median of the lower half of the data and the median of the upper half of the data, respectively. The IQR is then calculated as Q3 minus Q1.

The lower limit of the expected range is defined as Q1 minus 1.5 times the IQR, and the upper limit is defined as Q3 plus 1.5 times the IQR. Any data points that fall outside of these limits are considered potential outliers and should be further examined and verified.

One advantage of the interquartile test is that it is less sensitive to extreme values than the next test, the Z-score test, which makes it more suitable for datasets with skewed or non-normal distributions. However, it may not be as effective in detecting outliers in small sample sizes or datasets with multiple modes.

In [27]:

```
# IQR
# creating the series value with the target columns SALES
value = df['Sales']

# first quartile
q1 = value.quantile(.25)

# third quartile
q3 = value.quantile(.75)

# interquartile range
IIQ = q3 - q1

# inferior limit
lower_limit = q1 - 1.5 * IIQ

# upper limit
upper_limit = q3 + 1.5 * IIQ


print(f'''
[Q1] -> {q1}
[Q3] -> {q3}
[IIQ] -> {IIQ:.2f}
[Q1 - 1.5 * IIQ] -> {lower_limit:.2f}
[Q3 + 1.5 * IIQ] -> {upper_limit:.2f}
''')
```

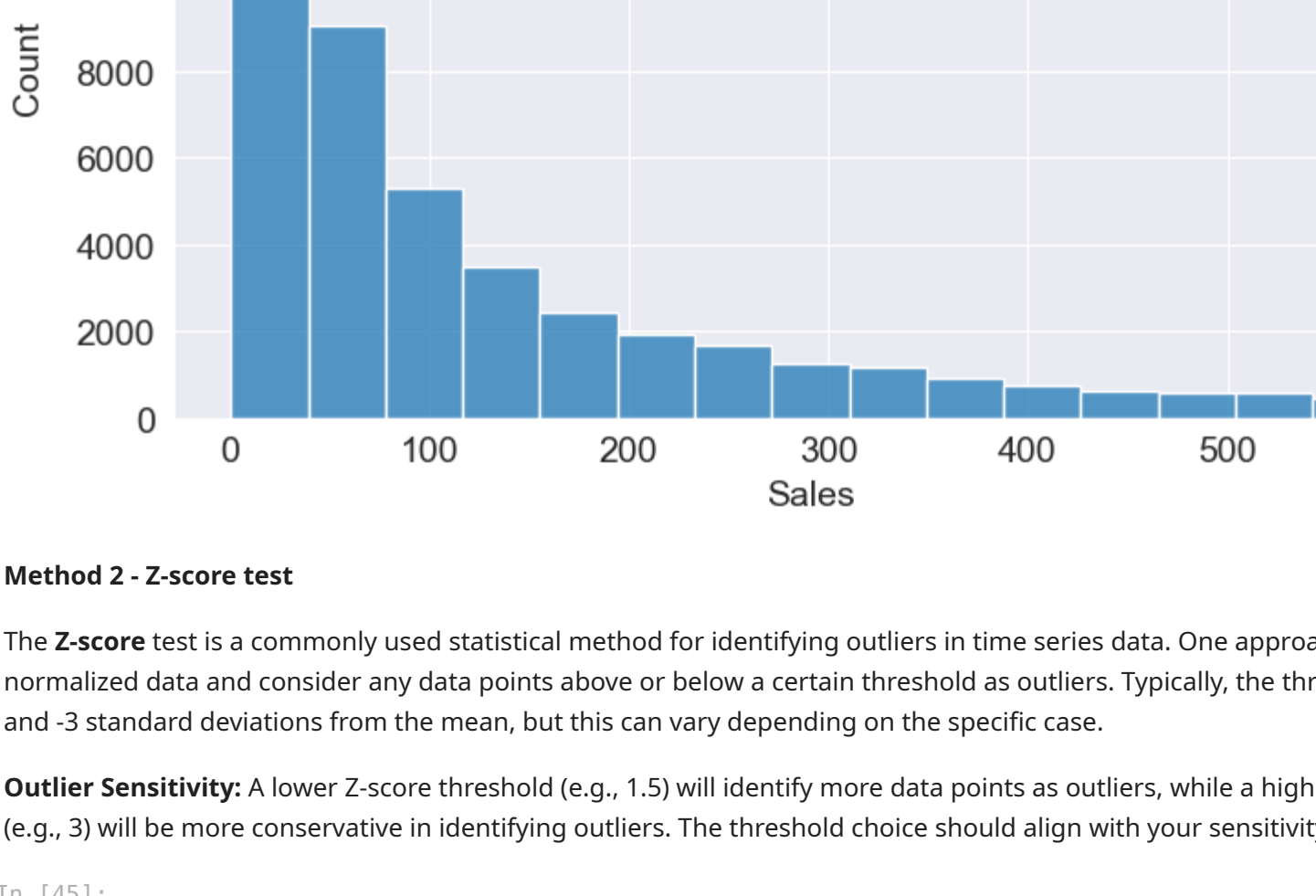
```
[Q1] -> 30.758625000000002
[Q3] -> 251.0532
[IIQ] -> 220.29
[Q1 - 1.5 * IIQ] -> -299.68
[Q3 + 1.5 * IIQ] -> 581.50
```

In [28]:

```
# applying
selection = (value >= lower_limit) & (value <= upper_limit)
df_igr = df[selection]
```

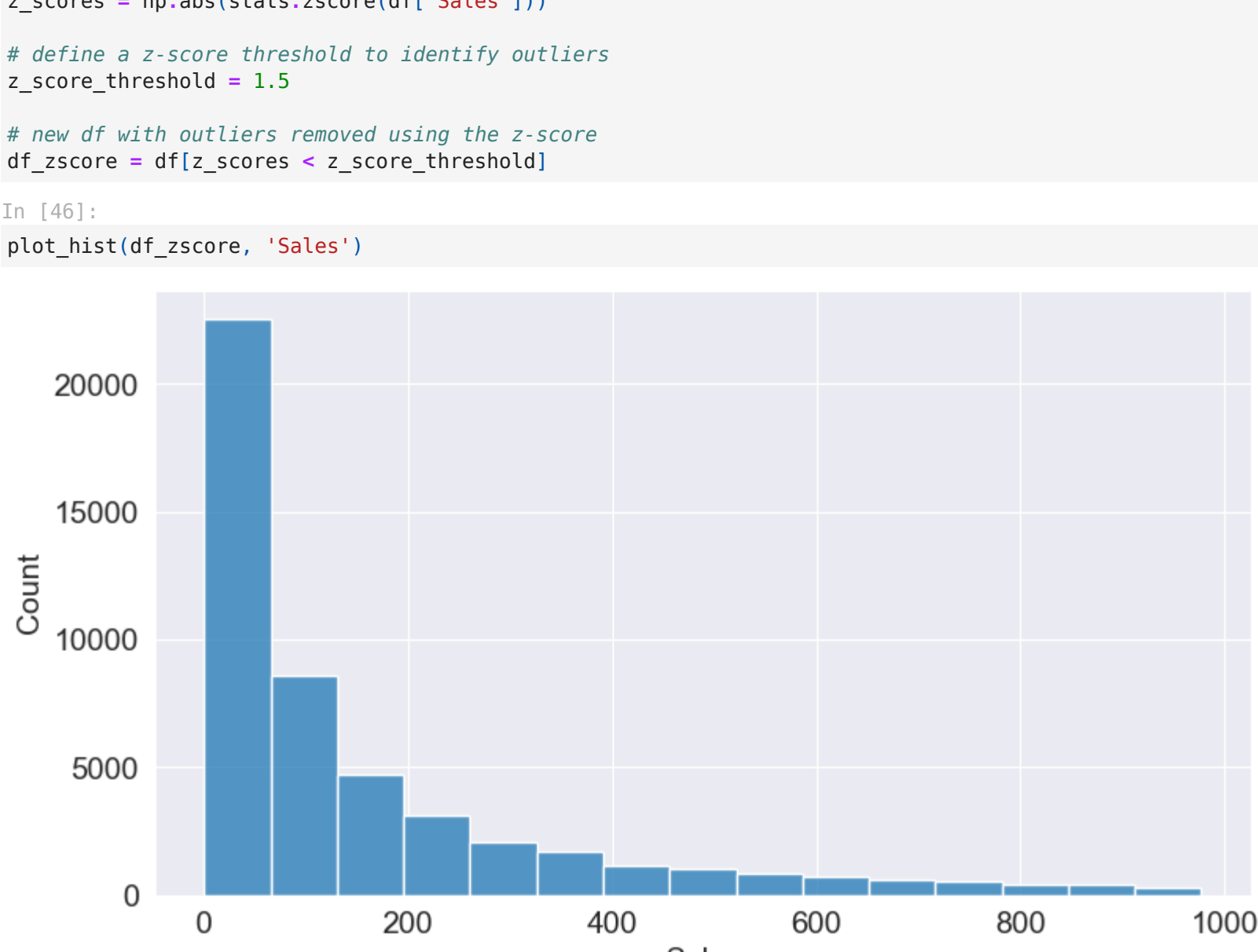
In [29]:

```
plot_boxplot(df_igr, 'Sales')
```



In [30]:

```
plot_hist(df_igr, 'Sales')
```



Method 2 - Z-score test

The **Z-score** test is a commonly used statistical method for identifying outliers in time series data. One approach is to use the normalized data and consider any data points above or below a certain threshold as outliers. Typically, the threshold is set at 3 and -3 standard deviations from the mean, but this can vary depending on the specific case.

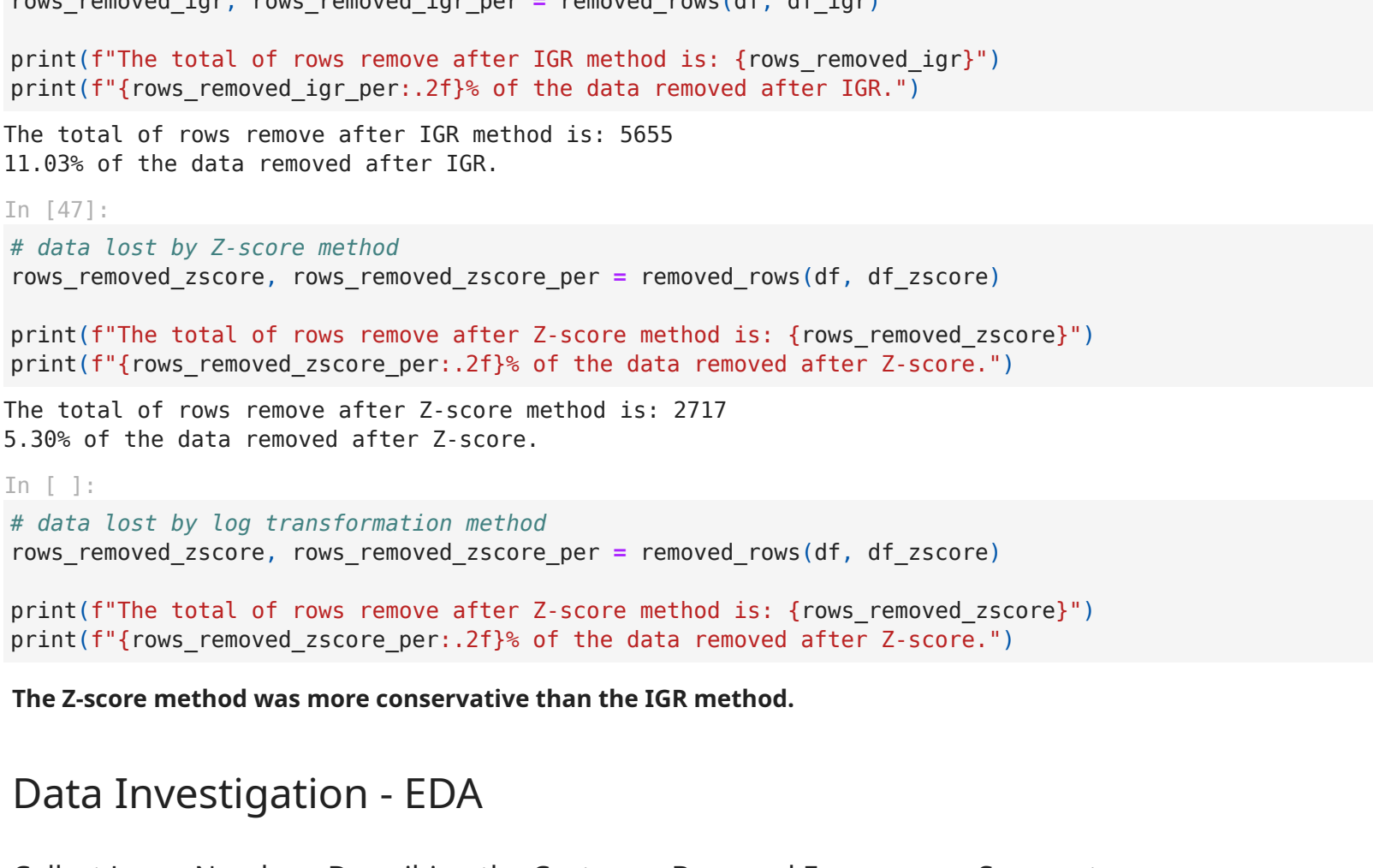
Outlier Sensitivity: A lower Z-score threshold (e.g., 1.5) will identify more data points as outliers, while a higher threshold (e.g., 3) will be more conservative in identifying outliers. The threshold choice should align with your sensitivity to outliers.

In [45]:

```
# z-scores for the 'Sales' column
z_scores = np.abs(stats.zscore(df['Sales']))

# define a z-score threshold to identify outliers
z_score_threshold = 1.5

# new df with outliers removed using the z-score
df_zscore = df[z_scores < z_score_threshold]
```

In [46]:

```
plot_hist(df_zscore, 'Sales')
```



How much of the data we lost after remove outliers?

In [33]:

```
def removed_rows(df, df_removed):
    rows_df = int(df.shape[0])
    rows_df_removed = int(df_removed.shape[0])
    rows_removed = rows_df - rows_df_removed
    rows_removed_per = (rows_removed * 100) / rows_df
    return rows_removed, rows_removed_per
```

In [34]:

```
# data lost by IGR method
rows_removed_igr, rows_removed_igr_per = removed_rows(df, df_igr)

print(f"The total of rows remove after IGR method is: {rows_removed_igr}")
print(f"{rows_removed_igr_per:.2f}% of the data removed after IGR.")
```

```
The total of rows remove after IGR method is: 5655
11.03% of the data removed after IGR.
```

In [47]:

```
# data lost by Z-score method
rows_removed_zscore, rows_removed_zscore_per = removed_rows(df, df_zscore)

print(f"The total of rows remove after Z-score method is: {rows_removed_zscore}")
print(f"{rows_removed_zscore_per:.2f}% of the data removed after Z-score.")
```

```
The total of rows remove after Z-score method is: 2717
5.30% of the data removed after Z-score.
```

In [ ]:

```
# data lost by log transformation method
rows_removed_zscore, rows_removed_zscore_per = removed_rows(df, df_zscore)

print(f"The total of rows remove after Z-score method is: {rows_removed_zscore}")
print(f"{rows_removed_zscore_per:.2f}% of the data removed after Z-score.")
```

The Z-score method was more conservative than the IGR method.

Data Investigation - EDA

Collect Large Numbers Describing the Customer Base and E-commerce Segments

In [51]:

```
df_igr.head()
```

Out[51]:

	Order ID	Order Date	Ship Date	customer ID	Segment	City	State	Region	Country	Market	Product ID	Category
1	ID-2011-24160	2011-12-29	2012-01-01	20170	Consumer	Gold Coast	Queensland	Oceania	Australia	APAC	TEC-PH-10002601	Technology
3	ID-2011-24160	2011-12-29	2012-01-01	20170	Consumer	Gold Coast	Queensland	Oceania	Australia	APAC	FUR-BO-10002308	Furniture
4	CA-2011-104738	2011-12-30	2012-01-01	20620	Corporate	Laredo	Texas	Central	United States	US	TEC-PH-10002468	Technology
5	EG-2011-590	2011-12-28	2012-01-01	5370	Consumer	Sohag	Suhaj	Africa	Egypt	Africa	OFF-SME-10003752	Office Supplies
6	CA-2011-104738	2011-12-30	2012-01-01	20620	Corporate	Laredo	Texas	Central	United States	US	TEC-PH-10000576	Technology

Specific Requests

1 - Average Delivery Time in All Countries

In [52]:

```
# Calculate 'Delivery Time' as the difference in days using .loc
df_igr.loc[:, 'Delivery Time'] = (df_igr['Ship Date'] - df_igr['Order Date']).dt.days

# Calculate the average delivery time
average_delivery_time = df_igr['Delivery Time'].mean()
print(f'Average Delivery Time in All Countries: {round(average_delivery_time)} days')
```

```
Average Delivery Time in All Countries: 4 days
```

In [53]:

```
# delivery time per country
average_delivery_time_by_country = df_igr.groupby('Country')['Delivery Time'].mean().reset_index().round()
average_delivery_time_by_country['Delivery Time'].unique()
```

Out[53]:

```
array([4., 5., 3., 2.])
```

2 - Most Profitable Customer Segment Each Year

In [54]:

```
# Extract year from Order Date
df_igr.loc[:, 'Year'] = df_igr['Order Date'].dt.year
most_profitable_segment = df_igr.groupby(['Year', 'Segment'])['Profit'].max().reset_index()

most_profitable_segment
```

Out[54]:

	Year	Segment	Profit
0	2011	Consumer	263.040
1	2011	Corporate	243.300
2	2011	Home Office	274.995
3	2012	Consumer	279.360
4	2012	Corporate	284.220
5	2012	Home Office	274.995
6	2013	Consumer	284.220
7	2013	Corporate	248.040
8	2013	Home Office	270.480
9	2014	Consumer	277.560
10	2014	Corporate	284.220
11	2014	Home Office	270.720

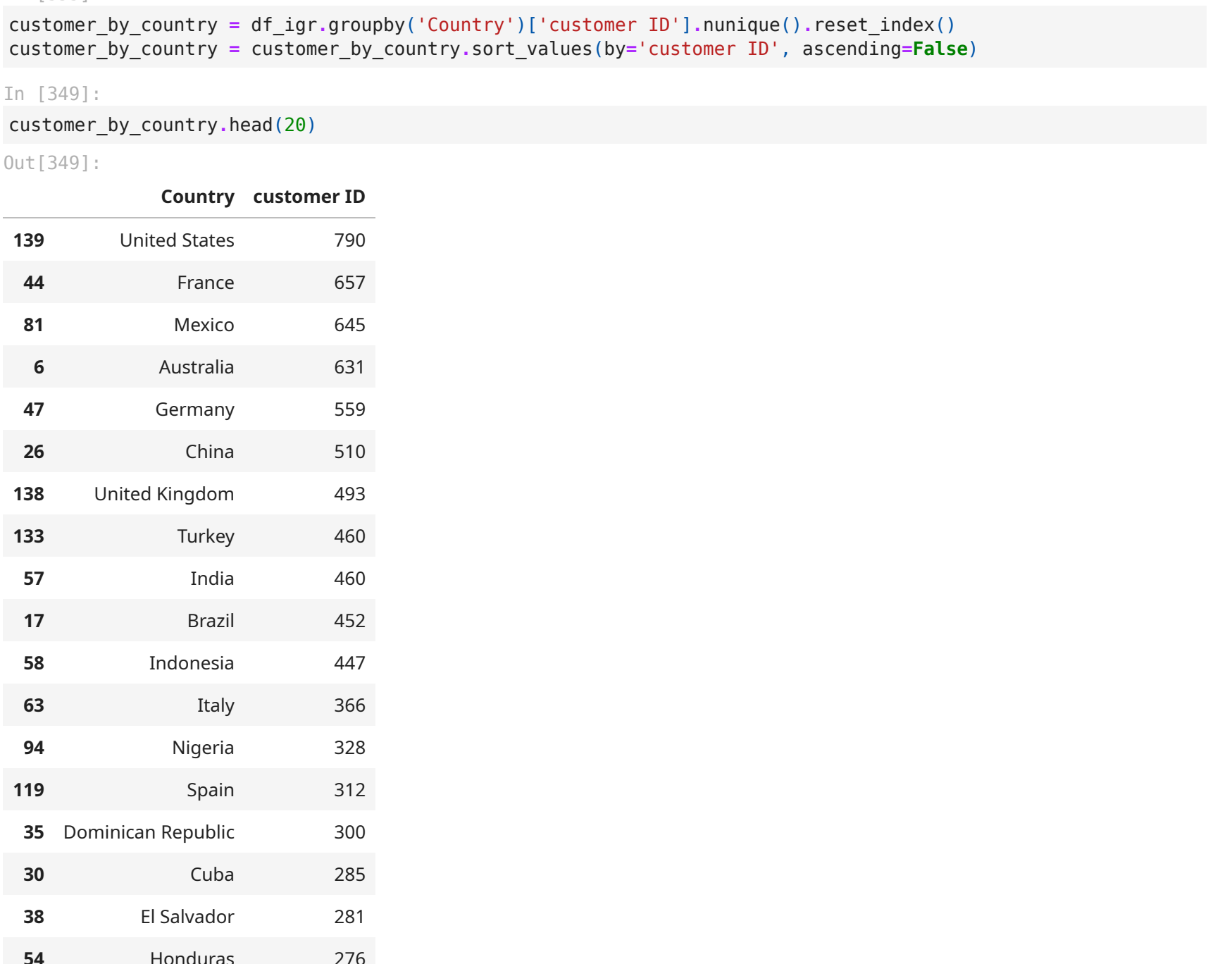
3 - Segment with the Highest Sales Volume

In [55]:

```
# barplot function for visualization
def bar_plot(x, y, data, title, xlabel, ylabel):
    plt.figure(figsize=(10, 6))
    sns.barplot(x=x, y=y, data=data, palette='viridis')
    plt.title(title, fontsize=16)
    plt.xlabel(xlabel, fontsize=12)
    plt.ylabel(ylabel, fontsize=12)
    plt.xticks(rotation=45)  # Rotate x-axis labels for better visibility
    plt.show()
```

In [56]:

```
sales_count_by_segment = df_igr.groupby('Segment')['Sales'].count().reset_index()
bar_plot('Segment', 'Sales', sales_count_by_segment, 'Sales Volume by Segment', 'Segment', 'Number of Sales')
```



4 - Customer Distribution Across Countries

In [353]:

```
customer_by_country = df_igr.groupby('Country')['customer ID'].nunique().reset_index()
customer_by_country = customer_by_country.sort_values(by='customer ID', ascending=False)
```

In [349]:

```
customer_by_country.head(20)
```

Out[349]:

	Country	customer ID
139	United States	790
44	France	657
81	Mexico	645
6	Australia	631
47	Germany	559
26	China	510
138	United Kingdom	493
133	Turkey	460
57	India	460
17	Brazil	452
58	Indonesia	447
63	Italy	366
94	Nigeria	328
119	Spain	312
35	Dominican Republic	300
30	Cuba	285
38	El Salvador	281
54	Honduras	276
101	Philippines	255
59	Iran	250

In [350]:

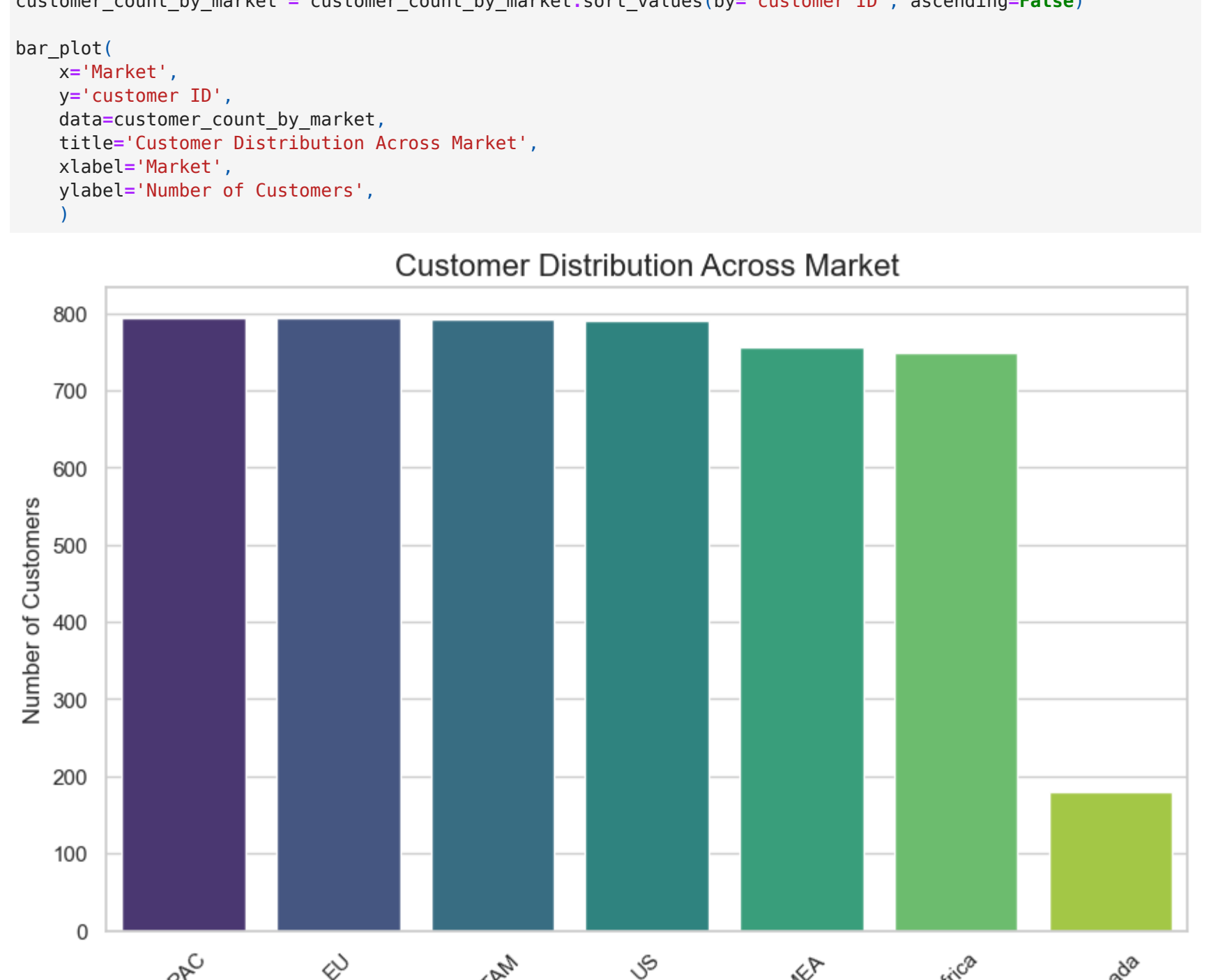
```
customer_by_country.tail()
```

Out[350]:

	Country	customer ID
118	South Sudan	1
24	Chad	1
9	Bahrain	1
39	Equatorial Guinea	1
40	Eritrea	1

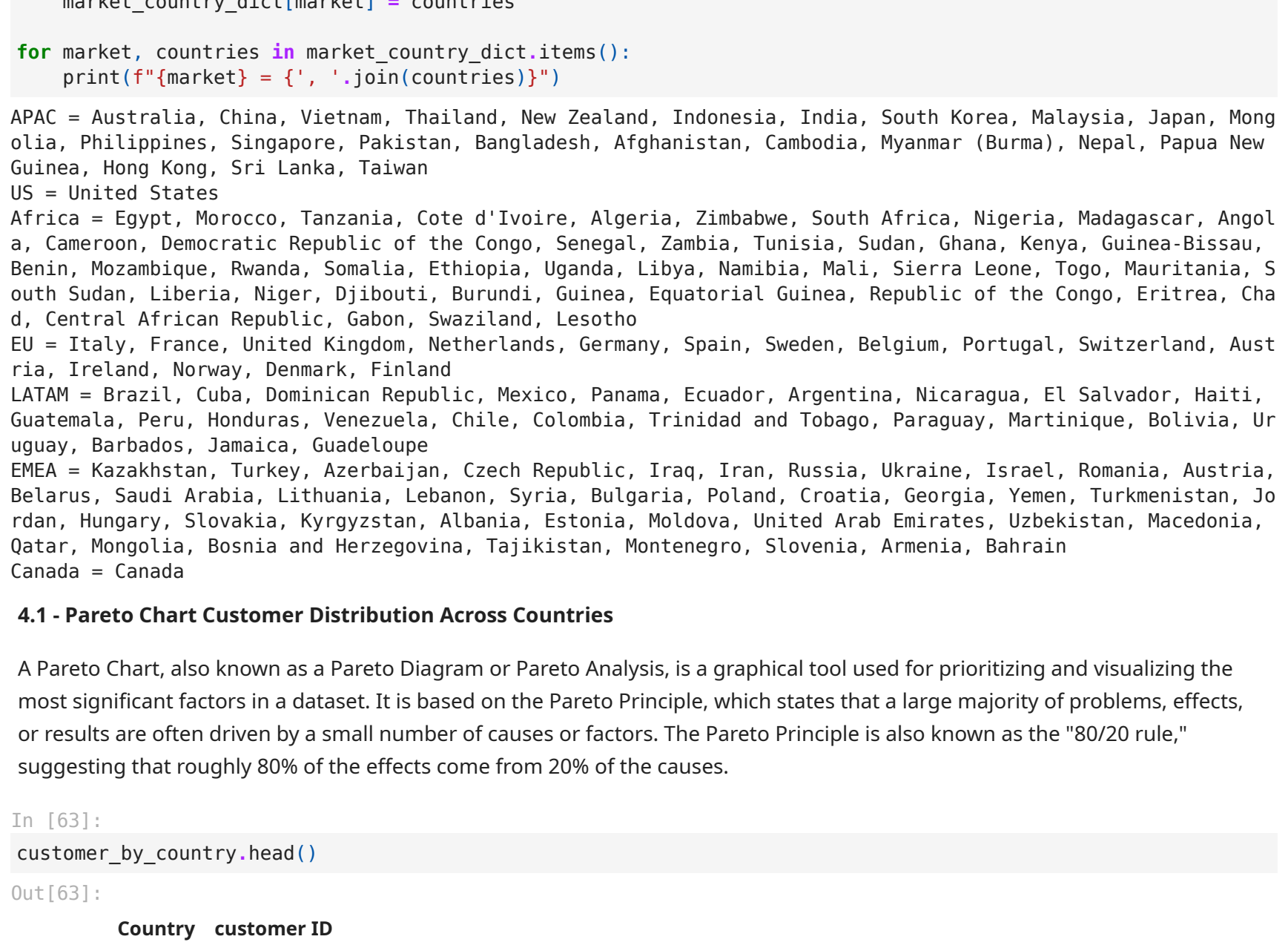
In [354]:

```python
top_10_countries = customer_by_country.head(10)

bar_plot(
    x='Country',
    y='customer ID',
    data=top_10_countries,
    title='Top 10 Countries with the Most Customers',
    xlabel='Country',
    ylabel='Number of Customers',
    )
```



In [351]:

```python
customer_count_by_market = df_igr.groupby('Market')['customer ID'].nunique().reset_index()
customer_count_by_market = customer_count_by_market.sort_values(by='customer ID', ascending=False)

bar_plot(
    x='Market',
    y='customer ID',
    data=customer_count_by_market,
    title='Customer Distribution Across Market',
    xlabel='Market',
    ylabel='Number of Customers',
    )
```



In [62]:

```python
market_country_dict = {}

for market in df_igr['Market'].unique():
    countries = df_igr[df_igr['Market'] == market]['Country'].unique().tolist()
    market_country_dict[market] = countries

for market, countries in market_country_dict.items():
    print(f"{market} = {', '.join(countries)}")
```

```
APAC = Australia, China, Vietnam, Thailand, New Zealand, Indonesia, India, South Korea, Malaysia, Japan, Mong
olia, Philippines, Singapore, Pakistan, Bangladesh, Afghanistan, Cambodia, Myanmar (Burma), Nepal, Papua New
Guinea, Hong Kong, Sri Lanka, Taiwan
US = United States
Africa = Egypt, Morocco, Tanzania, Cote d'Ivoire, Algeria, Zimbabwe, South Africa, Nigeria, Madagascar, Angol
a, Cameroon, Democratic Republic of the Congo, Senegal, Zambia, Tunisia, Sudan, Ghana, Kenya, Guinea-Bissau,
Benin, Mozambique, Rwanda, Somalia, Ethiopia, Uganda, Libya, Namibia, Mali, Sierra Leone, Togo, Mauritania, S
outh Sudan, Liberia, Niger, Djibouti, Burundi, Guinea, Equatorial Guinea, Republic of the Congo, Eritrea, Cha
d, Central African Republic, Gabon, Swaziland, Lesotho
EU = Italy, France, United Kingdom, Netherlands, Germany, Spain, Sweden, Belgium, Portugal, Switzerland, Aust
ria, Ireland, Norway, Denmark, Finland
LATAM = Brazil, Cuba, Dominican Republic, Mexico, Panama, Ecuador, Argentina, Nicaragua, El Salvador, Haiti,
Guatemala, Peru, Honduras, Venezuela, Chile, Colombia, Trinidad and Tobago, Paraguay, Martinique, Bolivia, Ur
uguay, Barbados, Jamaica, Guadeloupe
EMEA = Kazakhstan, Turkey, Azerbaijan, Czech Republic, Iraq, Iran, Russia, Ukraine, Israel, Romania, Austria,
Belarus, Saudi Arabia, Lithuania, Lebanon, Syria, Bulgaria, Poland, Croatia, Georgia, Yemen, Turkmenistan, Jo
rdan, Hungary, Slovakia, Kyrgyzstan, Albania, Estonia, Moldova, United Arab Emirates, Uzbekistan, Macedonia,
Qatar, Mongolia, Bosnia and Herzegovina, Tajikistan, Montenegro, Slovenia, Armenia, Bahrain
Canada = Canada
```

4.1 - Pareto Chart Customer Distribution Across Countries

A Pareto Chart, also known as a Pareto Diagram or Pareto Analysis, is a graphical tool used for prioritizing and visualizing the most significant factors in a dataset. It is based on the Pareto Principle, which states that a large majority of problems, effects, or results are often driven by a small number of causes or factors. The Pareto Principle is also known as the "80/20 rule," suggesting that roughly 80% of the effects come from 20% of the causes.

In [63]:

```python
customer_by_country.head()
```

Out[63]:

	Country	customer ID
139	United States	790
44	France	657
81	Mexico	645
6	Australia	631
47	Germany	559

In [357]:

```python
customer_by_country = customer_by_country.head(20)

customer_by_country['cumulative_percentage'] = (customer_by_country['customer ID'].cumsum() / customer_by_cou
customer_by_country.head(20)
```

Out[357]:

	Country	customer ID	cumulative_percentage
139	United States	790	9.021354
44	France	657	16.523924
81	Mexico	645	23.889460
6	Australia	631	31.095124
47	Germany	559	37.478589
26	China	510	43.302501
138	United Kingdom	493	48.932283
133	Turkey	460	54.185223
57	India	460	59.438164
17	Brazil	452	64.599749
58	Indonesia	447	69.704237
63	Italy	366	73.883750
94	Nigeria	328	77.629325
119	Spain	312	81.192189
35	Dominican Republic	300	84.618020
30	Cuba	285	87.872559
38	El Salvador	281	91.081421
54	Honduras	276	94.233185
101	Philippines	255	97.145141
59	Iran	250	100.000000

In [358]:

```python
fig, ax1 = plt.subplots()

# Bar plot for sales by country
ax1.bar(customer_by_country['Country'], customer_by_country['customer ID'], color='b')
ax1.set_ylabel('customer ID', color='b')
ax1.tick_params('y', colors='b')

# Create a secondary y-axis for the cumulative percentage
ax2 = ax1.twinx()
ax2.plot(customer_by_country['Country'], customer_by_country['cumulative_percentage'], color='r', marker='o',
ax2.set_ylabel('Cumulative Percentage', color='r')
ax2.axhline(y=80, color='g', linestyle='--', label='80% Line')

# Customize the chart
plt.title('Pareto Chart of Customers by Country')
ax1.set_xticklabels(customer_by_country['Country'], rotation=45, fontsize=10)
plt.grid(visible=False)

# Show the legend
fig.legend(loc='upper right', bbox_to_anchor=(0.85, 0.85))

# Display the chart
plt.show()
```

5 - Customer Profile Based on Purchasing Frequency

In [80]:

```python
df_igr.columns
```

Out[80]:

```
Index(['Order ID', 'Order Date', 'Ship Date', 'customer ID', 'Segment', 'City',
       'State', 'Region', 'Country', 'Market', 'Product ID', 'Category',
       'Product Name', 'Sales', 'Quantity', 'Profit', 'Delivery Time', 'Year'],
      dtype='object')
```

In [87]:

```python
# new df group by customer ID
customer_groups = df_igr.groupby('customer ID')

# customer profile using agregate function
customer_profile = customer_groups.agg({
    'Order ID': 'count',        # Number of orders
    'Sales': 'sum',             # Total sales
    'Profit': 'sum',            # Total profit
    'Quantity': 'sum',          # Total quantity purchased
    'Delivery Time': 'mean'     # Average delivery time
})

# rename the columns
customer_profile = customer_profile.rename(columns={
    'Order ID': 'Frequency',
    'Sales': 'Total Sales',
    'Profit': 'Total Profit',
    'Quantity': 'Total Quantity Purchased',
    'Delivery Time': 'Average Delivery Time'
})

# sort the customer profile by ppurchasing frequency
customer_profile = customer_profile.sort_values(by='Frequency', ascending=False)

# reset the index for a cleaner output
customer_profile.reset_index(inplace=True)

customer_profile
```

Out[87]:

	customer ID	Frequency	Total Sales	Total Profit	Total Quantity Purchased	Average Delivery Time
0	18850	88	11244.6880	1229.6629	298	3.988636
1	20755	82	12105.2005	914.9661	297	4.146341
2	11335	82	10569.6040	2000.7405	319	4.097561
3	15805	78	11352.7397	2240.2536	295	4.102564
4	17965	78	10132.2022	748.7027	267	4.051282
...	...	...	...	...	...	...
1585	7890	1	19.1280	-22.4820	1	0.000000
1586	6270	1	12.8520	-3.2280	1	6.000000
1587	2985	1	81.4320	-78.8880	8	6.000000
1588	7650	1	301.7100	27.1500	1	2.000000
1589	9825	1	37.4400	0.0000	4	5.000000

1590 rows × 6 columns

6 - Are Frequent Customers Contributing More Revenue and Profit?

In statistics, the Pearson correlation coefficient is a correlation coefficient that measures linear correlation between two sets of data. It is the ratio between the covariance of two variables and the product of their standard deviations

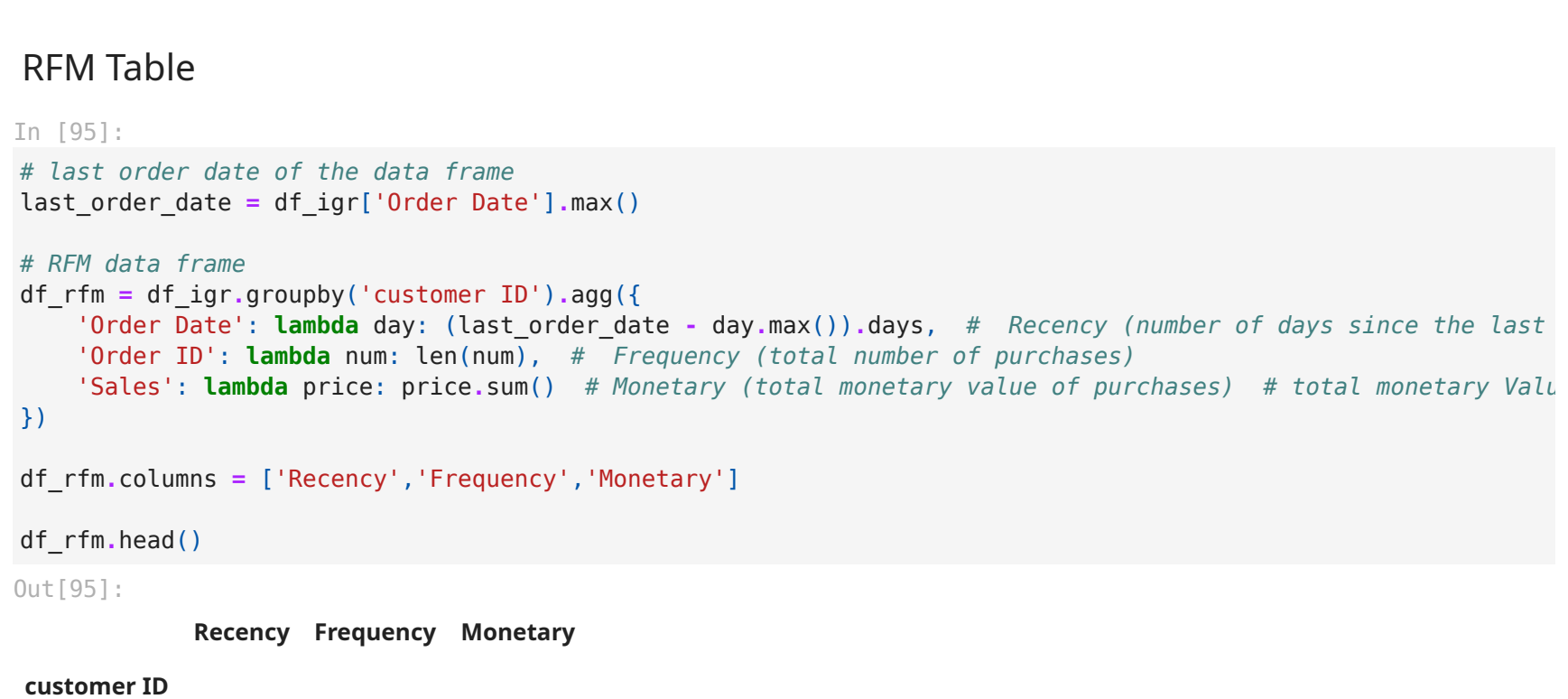
In [90]:

```python
correlation_sales = customer_profile['Frequency'].corr(customer_profile['Total Sales'])
correlation_profit = customer_profile['Frequency'].corr(customer_profile['Total Profit'])

# correlation pearson
print(correlation_sales)
print(correlation_profit)
```

```
0.9582574053987377
0.54984880601164
```

In [89]:

```python
# Create a pair of scatter plots with regression lines
plt.figure(figsize=(12, 5))

# Scatter plot for Frequency vs. Total Sales
plt.subplot(1, 2, 1)
sns.regplot(x='Frequency', y='Total Sales', data=customer_profile, color='blue')
plt.title('Frequency vs. Total Sales', fontsize=16)
plt.xlabel('Frequency', fontsize=12)
plt.ylabel('Total Sales', fontsize=12)

# Scatter plot for Frequency vs. Total Profit
plt.subplot(1, 2, 2)
sns.regplot(x='Frequency', y='Total Profit', data=customer_profile, color='green')
plt.title('Frequency vs. Total Profit', fontsize=16)
plt.xlabel('Frequency', fontsize=12)
plt.ylabel('Total Profit', fontsize=12)

plt.tight_layout()
plt.show()
```



Step 2 - Customer Segmentation

1 - Calculate Recency, Frequency, and Value:

RFM (Recency, Frequency, Monetary Value) analysis is a customer segmentation technique that uses past purchase behaviour to divide customers into groups. RFM helps divide customers into various categories or clusters to identify customers who are more likely to respond to promotions and also for future personalization services.

RECENCY (R): Days since last purchase

FREQUENCY (F): Total number of purchases

MONETARY VALUE (M): Total money this customer spent.

RFM Analysis

RFM Table

In [95]:

```python
# last order date of the data frame
last_order_date = df_igr['Order Date'].max()

# RFM data frame
df_rfm = df_igr.groupby('customer ID').agg({
    'Order Date': lambda day: (last_order_date - day.max()).days,  #  Recency (number of days since the last
    'Order ID': lambda num: len(num),  #  Frequency (total number of purchases)
    'Sales': lambda price: price.sum()  # Monetary (total monetary value of purchases)  # total monetary Valu
})

df_rfm.columns = ['Recency','Frequency','Monetary']

df_rfm.head()
```

Out[95]:

	Recency	Frequency	Monetary
customer ID			
15	412	10	1250.370
30	468	10	1477.236
45	233	13	1616.334
60	133	10	532.668
75	37	19	2582.940

RFM Score

The provided code is used to create RFM (Recency, Frequency, Monetary) scores and an overall RFM score for customer segmentation.

RFM analysis numerically scale each of these three categories for each customer 1 to 5. This is the higher the number, the better the result. The "Best" customer would receive a top score in every category.

pd.qcut() is used to perform quantile-based discretization of the Recency, Frequency, and Monetary scores into five segments each, labeled with values from 1 to 5. These segments will represent the relative ranking of customers for each of the three metrics.

In [218]:

```python
df_rfm["R_Score"]  = pd.qcut(df_rfm['Recency'], 5, [1, 2, 3, 4, 5])
df_rfm["F_Score"]  = pd.qcut(df_rfm['Frequency'],5, [1, 2, 3, 4, 5])
df_rfm["M_Score"]  = pd.qcut(df_rfm['Monetary'], 5, [1, 2, 3, 4, 5])


# sort data frame by Recency
df_rfm = df_rfm.sort_values(by='R_Score', ascending=False)
# concatenate the r, f, m score as string because is a categorical data for better understanding of the custo
df_rfm["RFM_Score"] = df_rfm["R_Score"].astype(str) + df_rfm["F_Score"].astype(str) + df_rfm["M_Score"].astyp

summary(df_rfm)
```

'Rows: 1590; Columns: 7'

	Recency	Frequency	Monetary	R_Score	F_Score	M_Score	RFM_Score
customer ID							
15	412	10	1250.370	5	1	2	512
9615	265	14	1147.926	5	2	2	522
9315	232	9	399.330	5	1	1	511
9390	270	19	2689.050	5	3	3	533
9435	216	14	2006.490	5	2	3	523

```
-----------------------------------------------------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
Index: 1590 entries, 15 to 21925
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Recency    1590 non-null   int64
 1   Frequency  1590 non-null   int64
 2   Monetary   1590 non-null   float64
 3   R_Score    1590 non-null   category
 4   F_Score    1590 non-null   category
 5   M_Score    1590 non-null   category
 6   RFM_Score  1590 non-null   object
dtypes: category(3), float64(1), int64(2), object(1)
memory usage: 67.4+ KB
None
-----------------------------------------------------------------------------------------------
```

	Recency	Frequency	Monetary
count	1590.000000	1590.000000	1590.000000
mean	87.593082	28.701258	3491.790977
std	129.897253	19.059298	2690.471999
min	0.000000	1.000000	7.173000
1%	0.000000	3.000000	145.435680
25%	14.000000	11.000000	1060.040250
50%	37.000000	25.000000	2738.931950
75%	103.000000	45.000000	5779.169600
99%	637.440000	69.000000	9573.933765
max	1206.000000	88.000000	12105.200500

2 - Segment Customers

1. **Champions:** Bought recently, buy often and spend the most
2. **Loyal customers:** Buy on a regular basis. Responsive to promotions.
3. **Potential loyalist:** Recent customers with average frequency.
4. **Recent customers:** Bought most recently, but not often.
5. **Promising:** Recent shoppers, but haven't spent much.
6. **Needs attention:** Above average recency, frequency and monetary values. May not have bought very recently though.
7. **About to sleep:** Below average recency and frequency. Will lose them if not reactivated.
8. **At risk:** Some time since they've purchased. Need to bring them back!
9. **Can't lose them:** Used to purchase frequently but haven't returned for a long time.
10. **Hibernating:** Last purchase was long back and low number of orders. May be lost.

RFM Segementation

In [226]:

```python
# calc FM_Score
df_rfm['FM_Score'] = round((df_rfm['F_Score'].astype(int) + df_rfm['M_Score'].astype(int)) / 2)

df_rfm['FM_Score'].unique()
```

Out[226]:

```
array([2., 1., 3., 4., 5.])
```

In [227]:

```python
# create the R(FM_Score)
df_rfm['R(FM)_Score'] = df_rfm['R_Score'].astype(str) + df_rfm['FM_Score'].astype(str)

# remove decimal
df_rfm['R(FM)_Score'] = df_rfm['R(FM)_Score'].str.replace('.0', '', regex=False)

df_rfm.head()
```

Out[227]:

	Recency	Frequency	Monetary	R_Score	F_Score	M_Score	RFM_Score	FM_Score	R(FM)_Score
customer ID									
15	412	10	1250.370	5	1	2	512	2.0	52
9615	265	14	1147.926	5	2	2	522	2.0	52
9315	232	9	399.330	5	1	1	511	1.0	51
9390	270	19	2689.050	5	3	3	533	3.0	53
9435	216	14	2006.490	5	2	3	523	2.0	52

In [255]:

```python
# df_rfm['R(FM)_Score'].unique()

# convert to string
df_rfm['R(FM)_Score'] = df_rfm['R(FM)_Score'].astype(str)
df_rfm['R(FM)_Score'].dtype
```

Out[255]:

```
dtype('O')
```

In [257]:

```python
# segmentation map
# regex R[1-5], FM[1-5]
seg_map = {
    r'11': 'Lost',
    r'12': 'Hibernating',
    r'13': 'About to Sleep',
    r'14': 'Promising',
    r'15': 'New Customers',
    r'21': 'At Risk',
    r'22': 'Need Attention',
    r'23': 'Potential Loyalist',
    r'24': "Can't Lose Them",
    r'25': 'Loyal Customers',
    r'31': 'Needs Attention',
    r'32': 'About to Sleep',
    r'33': 'At Risk',
    r'34': 'Promising',
    r'35': 'New Customers',
    r'41': 'At Risk',
    r'42': 'About to Sleep',
    r'43': 'Promising',
    r'44': "Can't Lose Them",
    r'45': 'Loyal Customers',
    r'51': 'Champions',
    r'52': 'Champions',
    r'53': 'Champions',
    r'54': 'Champions',
    r'55': 'Champions'
}
```

In [258]:

```python
def apply_segmentation(df, column_name, seg_map):
    """
    Apply segmentation to a DataFrame based on a given column and segmentation map.

    Parameters:
        df (DataFrame): The DataFrame to segment.
        column_name (str): The name of the column to be segmented.
        seg_map (dict): A dictionary containing regular expressions as keys and segment names as values.

    Returns:
        DataFrame: The original DataFrame with an additional 'Segment' column.
    """
    # Check if the specified column exists in the DataFrame
    if column_name not in df.columns:
        raise ValueError(f"Column '{column_name}' does not exist in the DataFrame.")

    # Apply the mapping function to create a new 'Segment' column
    df['RFM_Segment'] = df[column_name].astype(str).replace(seg_map, regex=True)

    return df
```

In [251]:

```python
df_rfm.head()
```

Out[251]:

	Recency	Frequency	Monetary	R_Score	F_Score	M_Score	RFM_Score	FM_Score	R(FM)_Score
customer ID									
15	412	10	1250.370	5	1	2	512	2.0	52
9615	265	14	1147.926	5	2	2	522	2.0	52
9315	232	9	399.330	5	1	1	511	1.0	51
9390	270	19	2689.050	5	3	3	533	3.0	53
9435	216	14	2006.490	5	2	3	523	2.0	52

In [260]:

```python
# apply function to create a new 'RFM_Class' column
df_rfm = apply_segmentation(df_rfm, 'R(FM)_Score', seg_map)

df_rfm.head()
```

Out[260]:

	Recency	Frequency	Monetary	R_Score	F_Score	M_Score	RFM_Score	FM_Score	R(FM)_Score	RFM_Segment
customer ID										
15	412	10	1250.370	5	1	2	512	2.0	52	Champions
9615	265	14	1147.926	5	2	2	522	2.0	52	Champions
9315	232	9	399.330	5	1	1	511	1.0	51	Champions
9390	270	19	2689.050	5	3	3	533	3.0	53	Champions
9435	216	14	2006.490	5	2	3	523	2.0	52	Champions

In [261]:

```python
df_rfm['RFM_Segment'].unique()
```

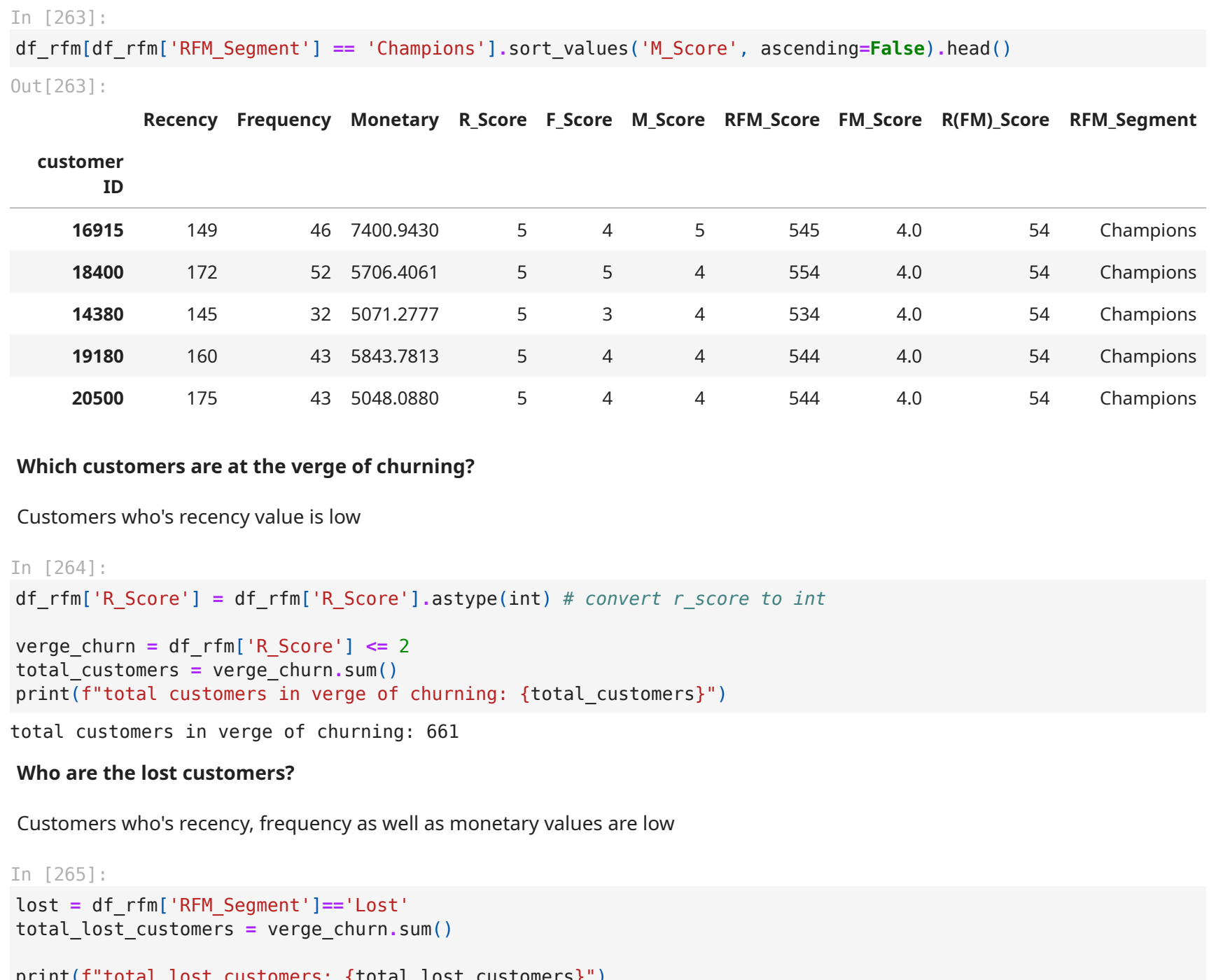
Out[261]:

```
array(['Champions', "Can't Lose Them", 'About to Sleep', 'At Risk',
       'Loyal Customers', 'Promising', 'Needs Attention', 'New Customers',
       'Potential Loyalist', 'Need Attention', 'Hibernating', 'Lost'],
      dtype=object)
```

R(FM) Segmentation readily answers

In [262]:

```python
# create a countplot
plt.figure(figsize=(10, 6))
sns.set(style="whitegrid")
ax = sns.countplot(data=df_rfm, x='RFM_Segment', order=df_rfm['RFM_Segment'].value_counts().index)

# display percentage on top of each bar
total = len(df_rfm)
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x() + p.get_width() / 2, height + 0.5, f"{height/total*100:.2f}%", ha="center")

# pie char
plt.title("Percentage of RFM Segments")
plt.xlabel("RFM Segment")
plt.ylabel("Percentage")
plt.xticks(rotation=45)
plt.show()
```



Who are my best customers?(Champions)

In [263]:

```python
df_rfm[df_rfm['RFM_Segment'] == 'Champions'].sort_values('M_Score', ascending=False).head()
```

Out[263]:

	Recency	Frequency	Monetary	R_Score	F_Score	M_Score	RFM_Score	FM_Score	R(FM)_Score	RFM_Segment
customer ID										
16915	149	46	7400.9430	5	4	5	545	4.0	54	Champions
18400	172	52	5706.4061	5	5	4	554	4.0	54	Champions
14380	145	32	5071.2777	5	3	4	534	4.0	54	Champions
19180	160	43	5843.7813	5	4	4	544	4.0	54	Champions
20500	175	43	5048.0880	5	4	4	544	4.0	54	Champions

Which customers are at the verge of churning?

Customers who's recency value is low

In [264]:

```python
df_rfm['R_Score'] = df_rfm['R_Score'].astype(int) # convert r_score to int

verge_churn = df_rfm['R_Score'] <= 2
total_customers = verge_churn.sum()
print(f"total customers in verge of churning: {total_customers}")
```

```
total customers in verge of churning: 661
```

Who are the lost customers?

Customers who's recency, frequency as well as monetary values are low

In [265]:

```python
lost = df_rfm['RFM_Segment']=='Lost'
total_lost_customers = verge_churn.sum()

print(f"total lost customers: {total_lost_customers}")
df_rfm[lost].sort_values('Recency',ascending=False).head()
```

```
total lost customers: 661
```

Out[265]:

	Recency	Frequency	Monetary	R_Score	F_Score	M_Score	RFM_Score	FM_Score	R(FM)_Score	RFM_Segment
customer ID										
11295	12	7	730.908	1	1	1	111	1.0	11	Lost
10935	11	7	487.746	1	1	1	111	1.0	11	Lost
11835	8	6	461.490	1	1	1	111	1.0	11	Lost
6765	8	5	729.090	1	1	1	111	1.0	11	Lost
810	7	7	485.007	1	1	1	111	1.0	11	Lost

Who are loyal customers?

Customers with high frequency value

In [266]:

```python
high_f = df_rfm['F_Score'] >= 4
total_loyal_customers = high_f.sum()

print(f"total of loyest customers: {total_loyal_customers}")
df_rfm[high_f].sort_values('Recency', ascending=False).head()
```

```
total of loyest customers: 627
```

Out[266]:

	Recency	Frequency	Monetary	R_Score	F_Score	M_Score	RFM_Score	FM_Score	R(FM)_Score	RFM_Segment
customer ID										
9285	254	38	4501.7580	5	4	4	544	4.0	54	Champions
20500	175	43	5048.0880	5	4	4	544	4.0	54	Champions
18400	172	52	5706.4061	5	5	4	554	4.0	54	Champions
19180	160	43	5843.7813	5	4	4	544	4.0	54	Champions
19645	150	39	3436.8463	5	4	3	543	4.0	54	Champions

All Overall view of RFM Analysis

In [267]:

```python
df_rfm[["RFM_Segment","Recency", "Frequency", "Monetary"]].groupby("RFM_Segment").agg(["mean", "count","sum"]
```

Out[267]:

	Recency			Frequency			Monetary		
	mean	count	sum	mean	count	sum	mean	count	sum
RFM_Segment									
About to Sleep	55.947137	227	12700	16.110132	227	3657	1676.953625	227	3.806685e+05
At Risk	62.786325	117	7346	13.897436	117	1626	1346.521212	117	1.575430e+05
Can't Lose Them	41.927778	180	7547	42.961111	180	7733	5405.154811	180	9.729279e+05
Champions	293.741325	317	93116	11.000000	317	3487	1112.850698	317	3.527737e+05
Hibernating	5.508475	59	325	13.186441	59	778	1311.489203	59	7.737786e+04
Lost	5.600000	15	84	6.666667	15	100	532.353600	15	7.985304e+03
Loyal Customers	35.813725	102	3653	57.519608	102	5867	7698.135336	102	7.852098e+05
Need Attention	19.241935	62	1193	13.048387	62	809	1228.157419	62	7.614576e+04
Needs Attention	40.466667	30	1214	6.966667	30	209	452.687300	30	1.358062e+04
New Customers	17.073529	136	2322	58.000000	136	7888	7820.916243	136	1.063645e+06
Potential Loyalist	20.140000	50	1007	27.760000	50	1388	3122.507861	50	1.561254e+05
Promising	29.715254	295	8766	40.993220	295	12093	5111.746810	295	1.507965e+06

Step 3 - Sales projection by segment

still working

Train a time series forecast model to forecast sales (Sales column) for the segment with the highest sales volume (across the entire

- history (daily frequency)
- Model validation: December 2014 (daily frequency)
- Use MAPE, SMAPE and RMSLE metrics to calculate the result in Dec 2014.

Duvida, Segment aqui é referente a coluna do dado original ou Segmentos relacionado a analise RFM?

Mantive o forecating na coluna Segment do dado original.

Machine Learning Model

Traditional models like ARIMA are more time spending and require more knowledge about the data.

Machine learning models are easier and faster to apply.

ALgorithm:

RandomForestRegressor for time series regression tasks such as sales forecasting to predict numeric values.

XGBoost

Data Frame Pre-processing

In [269]:

```
# create a df
df_ml = df_igr.copy()
df_ml.head()
```

Out[269]:

	Order ID	Order Date	Ship Date	customer ID	Segment	City	State	Region	Country	Market	Product ID	Category	
1	ID-2011-24160	2011-12-29	2012-01-01	20170	Consumer	Gold Coast	Queensland	Oceania	Australia	APAC	TEC-PH-10002601	Technology	Sma
3	ID-2011-24160	2011-12-29	2012-01-01	20170	Consumer	Gold Coast	Queensland	Oceania	Australia	APAC	FUR-BO-10002308	Furniture	Floa S
4	CA-2011-104738	2011-12-30	2012-01-01	20620	Corporate	Laredo	Texas	Central	United States	US	TEC-PH-10002468	Technology	Pl C - Conv
5	EG-2011-590	2011-12-28	2012-01-01	5370	Consumer	Sohag	Suhaj	Africa	Egypt	Africa	OFF-SME-10003752	Office Supplies	Sı Ca
6	CA-2011-104738	2011-12-30	2012-01-01	20620	Corporate	Laredo	Texas	Central	United States	US	TEC-PH-10000576	Technology	A

In [299]:

```
#  segment with the highest sales
highest_sales_segment = df_ml.groupby('Segment')['Sales'].sum().idxmax()
df_ml_segment = df_ml[df_ml['Segment'] == highest_sales_segment]

df_ml_segment = df_ml_segment.sort_values(by='Segment')

df_ml_segment.head()
```

Out[299]:

	Order ID	Order Date	Ship Date	customer ID	Segment	City	State	Region	Country	Market	Product ID	Cate
1	ID-2011-24160	2011-12-29	2012-01-01	20170	Consumer	Gold Coast	Queensland	Oceania	Australia	APAC	TEC-PH-10002601	Techno
34096	IN-2014-82652	2014-07-16	2014-07-21	11140	Consumer	Wollongong	New South Wales	Oceania	Australia	APAC	OFF-SU-10001488	C Sup
34095	IN-2014-82652	2014-07-16	2014-07-21	11140	Consumer	Wollongong	New South Wales	Oceania	Australia	APAC	OFF-BI-10004562	C Sup
34093	IN-2014-82652	2014-07-16	2014-07-21	11140	Consumer	Wollongong	New South Wales	Oceania	Australia	APAC	OFF-LA-10002086	C Sup
34087	CG-2014-870	2014-07-19	2014-07-21	705	Consumer	Kinshasa	Kinshasa	Africa	Democratic Republic of the Congo	Africa	OFF-BIC-10001211	C Sup

In [301]:

```
# reset index for resample('D')
df_ml_segment.set_index('Order Date', inplace=True)

df_ml_segment.head()
```

Out[301]:

	Order ID	Ship Date	customer ID	Segment	City	State	Region	Country	Market	Product ID	Category
Order Date											
2011-12-29	ID-2011-24160	2012-01-01	20170	Consumer	Gold Coast	Queensland	Oceania	Australia	APAC	TEC-PH-10002601	Technology
2014-07-16	IN-2014-82652	2014-07-21	11140	Consumer	Wollongong	New South Wales	Oceania	Australia	APAC	OFF-SU-10001488	Office Supplies
2014-07-16	IN-2014-82652	2014-07-21	11140	Consumer	Wollongong	New South Wales	Oceania	Australia	APAC	OFF-BI-10004562	Office Supplies
2014-07-16	IN-2014-82652	2014-07-21	11140	Consumer	Wollongong	New South Wales	Oceania	Australia	APAC	OFF-LA-10002086	Office Supplies
2014-07-19	CG-2014-870	2014-07-21	705	Consumer	Kinshasa	Kinshasa	Africa	Democratic Republic of the Congo	Africa	OFF-BIC-10001211	Office Supplies

In [303]:

```
# resample on daily frequency and calculate daily sales
daily_sales = df_ml_segment['Sales'].resample('D').sum()
daily_sales_df = daily_sales.reset_index()  # Reset the index to include the date as a column
daily_sales_df.columns = ['Date', 'Sales']
daily_sales_df.head()
```

Out[303]:

	Date	Sales
0	2011-01-01	763.6980
1	2011-01-02	314.2200
2	2011-01-03	1465.6262
3	2011-01-04	353.9280
4	2011-01-05	48.7800

In [308]:

```
daily_sales_df.set_index('Date', inplace=True)
daily_sales_df.head()
```

Out[308]:

	Sales
Date	
2011-01-01	763.6980
2011-01-02	314.2200
2011-01-03	1465.6262
2011-01-04	353.9280
2011-01-05	48.7800

In [310]:

```
# training and testing data sets for 2014
# as required the testing data for December of 2014, train for the rest of 2014
train_data = daily_sales_df[daily_sales_df.index < '2014-12-01']
test_data = daily_sales_df[daily_sales_df.index >= '2014-12-01']
```

In [311]:

```
train_data.shape, test_data.shape
```

Out[311]:

```
((1430, 1), (31, 1))
```

In [312]:

```
test_data # only for december
```

Out[312]:

	Sales
Date	
2014-12-01	4536.16100
2014-12-02	8693.51350
2014-12-03	4970.56520
2014-12-04	4941.87410
2014-12-05	3661.41000
2014-12-06	1401.27200
2014-12-07	828.89000
2014-12-08	4660.00010
2014-12-09	5801.84480
2014-12-10	5412.58660
2014-12-11	5189.48248
2014-12-12	4830.93040
2014-12-13	3405.40050
2014-12-14	991.62400
2014-12-15	5864.21100
2014-12-16	4550.02600
2014-12-17	2196.65710
2014-12-18	5179.34300
2014-12-19	4143.44720
2014-12-20	1104.91520
2014-12-21	17.76000
2014-12-22	3538.54120
2014-12-23	5342.80070
2014-12-24	4026.46700
2014-12-25	4306.27640
2014-12-26	7106.10940
2014-12-27	4138.38976
2014-12-28	545.20800
2014-12-29	8123.24728
2014-12-30	7813.22930
2014-12-31	2625.78100

Creating the target variable and the lagged variable (Sales)

Daily prediction: complex; The idea is to make the sale value for the previous day and predict it for the next day?

In [314]:

```
train_data['Sales']
```

Out[314]:

```
Date
2011-01-01     763.69800
2011-01-02     314.22000
2011-01-03    1465.62620
2011-01-04     353.92800
2011-01-05      48.78000
                 ...
2014-11-26    6991.84824
2014-11-27    5452.68672
2014-11-28    6714.00742
2014-11-29    1500.29100
2014-11-30    1055.88800
Name: Sales, Length: 1430, dtype: float64
```

In [315]:

```
train_data['Sales'].shift(-1)
train_data['Sales'] # lagged?
```

Out[315]:

```
Date
2011-01-01     763.69800
2011-01-02     314.22000
2011-01-03    1465.62620
2011-01-04     353.92800
2011-01-05      48.78000
                 ...
2014-11-26    6991.84824
2014-11-27    5452.68672
2014-11-28    6714.00742
2014-11-29    1500.29100
2014-11-30    1055.88800
Name: Sales, Length: 1430, dtype: float64
```

'Target' column represents the sales data for the next time period, which you want to predict.

shift(-1) is a Pandas method that shifts the values of a Series (or column) by a specified number of periods. In this case, shift(-1) shifts the 'Sales' column one period into the future.

In [317]:

```
train_data['Target'] = train_data['Sales'].shift(-1)
train_data.head()
```

Out[317]:

	Sales	Target
Date		
2011-01-01	763.6980	314.2200
2011-01-02	314.2200	1465.6262
2011-01-03	1465.6262	353.9280
2011-01-04	353.9280	48.7800
2011-01-05	48.7800	465.4602

In [318]:

```
# one row will have nan value
train_data.tail()
```

Out[318]:

	Sales	Target
Date		
2014-11-26	6991.84824	5452.68672
2014-11-27	5452.68672	6714.00742
2014-11-28	6714.00742	1500.29100
2014-11-29	1500.29100	1055.88800
2014-11-30	1055.88800	NaN

In [319]:

```
# remove this row
train_data = train_data.dropna()
train_data.tail()
```

Out[319]:

	Sales	Target
Date		
2014-11-25	8687.30604	6991.84824
2014-11-26	6991.84824	5452.68672
2014-11-27	5452.68672	6714.00742
2014-11-28	6714.00742	1500.29100
2014-11-29	1500.29100	1055.88800

Make the same thing to Test Data

In [320]:

```
test_data['Target'] = test_data['Sales'].shift(-1)
test_data = test_data.dropna()
test_data.tail()
```

Out[320]:

	Sales	Target
Date		
2014-12-26	7106.10940	4138.38976
2014-12-27	4138.38976	545.20800
2014-12-28	545.20800	8123.24728
2014-12-29	8123.24728	7813.22930
2014-12-30	7813.22930	2625.78100

Prepare for ML training and validation

In [321]:

```
X_train_data = train_data.loc[:, ['Sales']].values
y_train_data = train_data.loc[:, ['Target']].values
X_test_data= test_data.loc[:, ['Sales']].values
y_test_data = test_data.loc[:, ['Target']].values

X_train_data.shape, y_train_data.shape, X_test_data.shape, y_test_data.shape
```

Out[321]:

```
((1429, 1), (1429, 1), (30, 1), (30, 1))
```

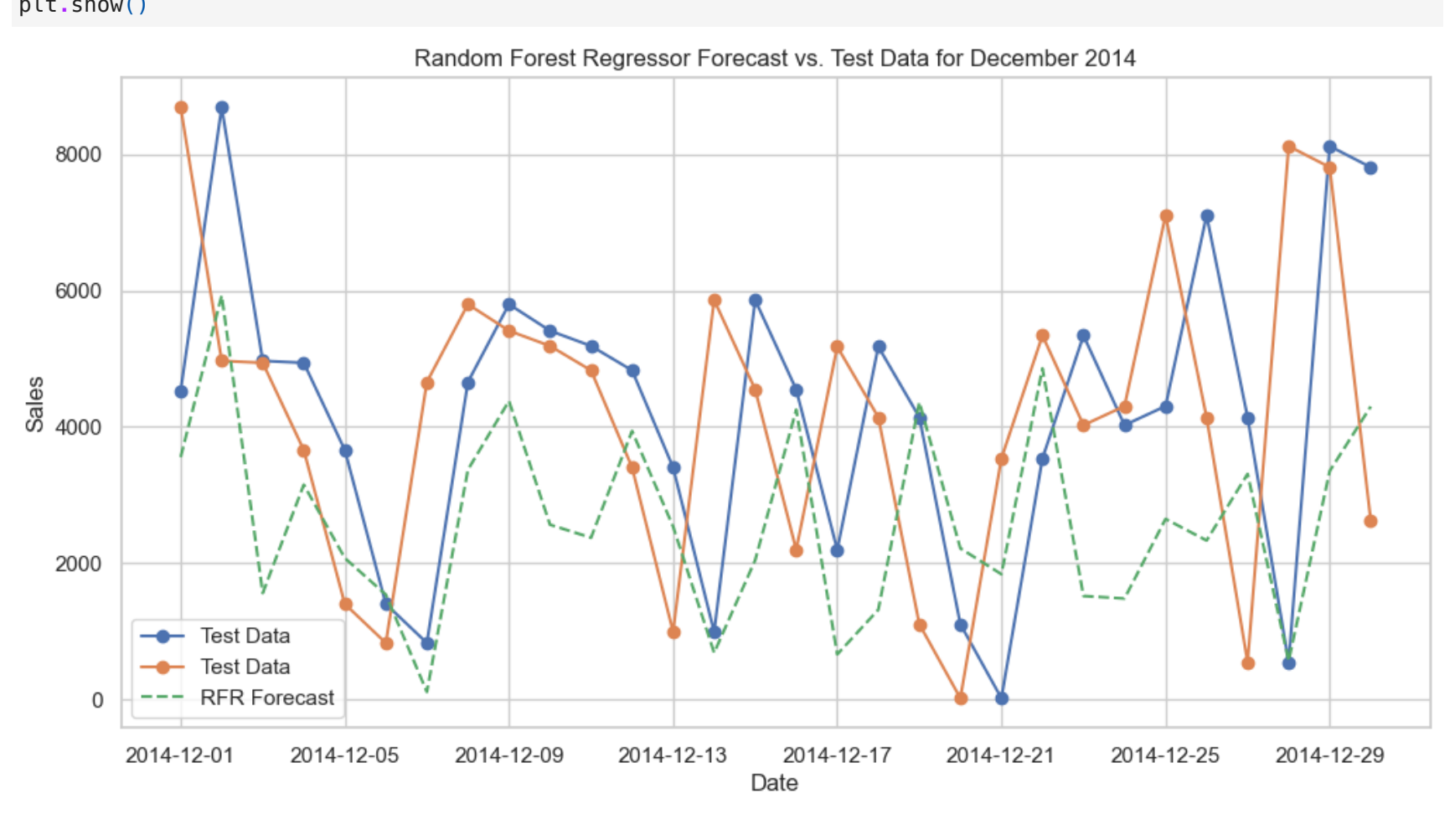
Random Forest Regressor

In [331]:

```
rf_model = RandomForestRegressor(n_estimators=100, random_state=0)
rf_model.fit(X_train_data.reshape(-1, 1), y_train_data)

# Forecast for December 2014
rf_forecast = rf_model.predict(X_test_data.reshape(-1, 1))
```

In [344]:

```
plt.figure(figsize=(12, 6))
plt.plot(test_data.index, test_data, label='Test Data', marker='o')
plt.plot(test_data.index, rf_forecast, label='RFR Forecast', linestyle='--')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.title('Random Forest Regressor Forecast vs. Test Data for December 2014')
plt.legend()
plt.grid(True)
plt.show()
```



Metrics for Forest Regressor

In [ ]:

```
# Calculate the metrics (MAE, MSE, RMSLE)
mae = mean_absolute_error(test_data[1:], rf_forecast)
mse = mean_squared_error(test_data[1:], rf_forecast)
rmsle = np.sqrt(mean_squared_log_error(test_data[1:], rf_forecast))

print(f"MAE: {mae:.2f}")
print(f"MSE: {mse:.2f}")
print(f"RMSLE: {rmsle:.4f}")
```

XGBoost Model

In [336]:

```
X_train = train_data.values[:-1].reshape(-1, 1)  # Use all data except the last data point for training
y_train = train_data.values[1:]  # Predict the next day's sales
X_train_data.shape, y_train_data.shape
```

Out[336]:

```
((1429, 1), (1429, 1))
```

In [337]:

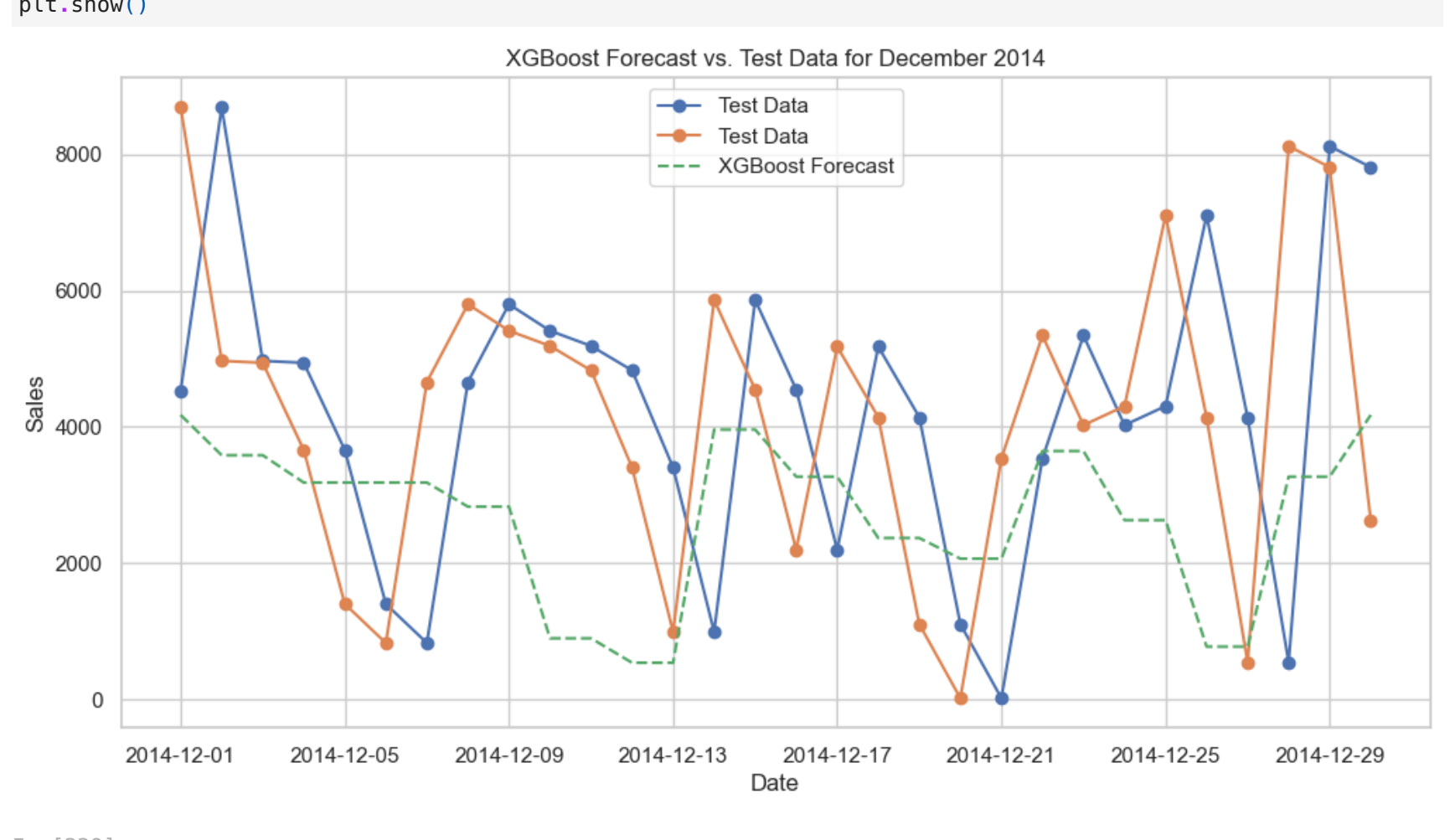
```
xgb_model = xgb.XGBRegressor(objective="reg:squarederror", n_estimators=1000)
xgb_model.fit(X_train, y_train)
```

Out[337]:

```
▾                              XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
```

In [343]:

```
plt.figure(figsize=(12, 6))
plt.plot(test_data.index, test_data, label='Test Data', marker='o')
plt.plot(test_data.index, xgb_forecast, label='XGBoost Forecast', linestyle='--')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.title('XGBoost Forecast vs. Test Data for December 2014')
plt.legend()
plt.grid(True)
plt.show()
```



In [339]:

```
# Forecast for December 2014
X_valid = test_data.values[:-1].reshape(-1, 1)  # Use all data except the last data point for validation
xgb_forecast = xgb_model.predict(X_valid)
```

In [341]:

```
# Calculate the metrics (MAE, MSE, RMSLE)
xgb_forecast = xgb_forecast[:len(test_data)]

mae = mean_absolute_error(test_data[1:], xgb_forecast)
mse = mean_squared_error(test_data[1:], xgb_forecast)
rmsle = np.sqrt(mean_squared_log_error(test_data[1:], xgb_forecast))

print(f"MAE: {mae:.2f}")
print(f"MSE: {mse:.2f}")
print(f"RMSLE: {rmsle:.4f}")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/mnt/Arquivos/Documentos/PROJETOS/MentoriaBoiTata/notebooks/desafio_leroi_merlin.ipynb Cell 134 line 4
      <a href='vscode-notebook-cell:/mnt/Arquivos/Documentos/PROJETOS/MentoriaBoiTata/notebooks/desafio_leroi
_merlin.ipynb#Y401sZmlsZQ%3D%3D?line=0'>1</a> # Calculate the metrics (MAE, MSE, RMSLE)
      <a href='vscode-notebook-cell:/mnt/Arquivos/Documentos/PROJETOS/MentoriaBoiTata/notebooks/desafio_leroi
_merlin.ipynb#Y401sZmlsZQ%3D%3D?line=1'>2</a> xgb_forecast = xgb_forecast[:len(test_data)]
----> <a href='vscode-notebook-cell:/mnt/Arquivos/Documentos/PROJETOS/MentoriaBoiTata/notebooks/desafio_leroi
_merlin.ipynb#Y401sZmlsZQ%3D%3D?line=3'>4</a> mae = mean_absolute_error(test_data[1:], xgb_forecast)
      <a href='vscode-notebook-cell:/mnt/Arquivos/Documentos/PROJETOS/MentoriaBoiTata/notebooks/desafio_leroi
_merlin.ipynb#Y401sZmlsZQ%3D%3D?line=4'>5</a> mse = mean_squared_error(test_data[1:], xgb_forecast)
      <a href='vscode-notebook-cell:/mnt/Arquivos/Documentos/PROJETOS/MentoriaBoiTata/notebooks/desafio_leroi
_merlin.ipynb#Y401sZmlsZQ%3D%3D?line=5'>6</a> rmsle = np.sqrt(mean_squared_log_error(test_data[1:], xgb_forec
ast))

File ~/.local/lib/python3.11/site-packages/sklearn/metrics/_regression.py:196, in mean_absolute_error(y_true,
 y_pred, sample_weight, multioutput)
    141 def mean_absolute_error(
    142     y_true, y_pred, *, sample_weight=None, multioutput="uniform_average"
    143 ):
    144     """Mean absolute error regression loss.
    145 
    146     Read more in the :ref:`User Guide <mean_absolute_error>`.
   (...)
    194     0.85...
    195     """
--> 196     y_type, y_true, y_pred, multioutput = _check_reg_targets(
    197         y_true, y_pred, multioutput
    198     )
    199     check_consistent_length(y_true, y_pred, sample_weight)
    200     output_errors = np.average(np.abs(y_pred - y_true), weights=sample_weight, axis=0)

File ~/.local/lib/python3.11/site-packages/sklearn/metrics/_regression.py:100, in _check_reg_targets(y_true,
 y_pred, multioutput, dtype)
     66 def _check_reg_targets(y_true, y_pred, multioutput, dtype="numeric"):
     67     """Check that y_true and y_pred belong to the same regression task.
     68 
     69     Parameters
   (...)
     98         correct keyword.
     99     """
--> 100     check_consistent_length(y_true, y_pred)
    101     y_true = check_array(y_true, ensure_2d=False, dtype=dtype)
    102     y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)

File ~/.local/lib/python3.11/site-packages/sklearn/utils/validation.py:397, in check_consistent_length(*array
s)
    395 uniques = np.unique(lengths)
    396 if len(uniques) > 1:
--> 397     raise ValueError(
    398         "Found input variables with inconsistent numbers of samples: %r"
    399         % [int(l) for l in lengths]
    400     )

ValueError: Found input variables with inconsistent numbers of samples: [29, 30]
```

In [ ]:

```
# Actual sales data for December 2014
actual_sales_december_2014 = df_daily['2014-12-01':'2014-12-31']

# Calculate MAPE, SMAPE, and RMSLE
mape = mean_absolute_error(actual_sales_december_2014, forecasted_sales) / np.mean(actual_sales_december_2014
smape = 100 * np.mean(np.abs(forecasted_sales - actual_sales_december_2014) / (np.abs(forecasted_sales) + np.
rmsle = np.sqrt(mean_squared_log_error(actual_sales_december_2014, forecasted_sales))

# Print the results
print("MAPE:", mape)
print("SMAPE:", smape)
print("RMSLE:", rmsle)
```

Forecasting with XGBoost

Bibliography

- A. Bajaj, "RFM Analysis For Successful Customer Segmentation using Python," Medium, May 2019. [Online]. Available: https://aainabajaj39.medium.com/rfm-analysis-for-successful-customer-segmentation-using-python-6291decceb4b. [Accessed: Octuber 30, 2023].
- ComunidadeDS, "Como fazer uma segmentação de clientes com RFM?," 2021, [Online]. Available: https://www.youtube.com/watch?v=CArRKwrLaeM. [Accessed: Octuber 25, 2023].
- D. Andrés, "Clean your Time Series data III: Outliers removal - ML Pills," Apr. 2023. [Online]. Available: andres2023.pdf. [Accessed: Octuber 23, 2023].
- U. Savcı, "Customer Segmentation using RFM Analysis in Python," 2022, [Online]. Available: https://medium.com/@ugursavci/customer-segmentation-using-rfm-analysis-in-python-218a3255f714. [Accessed: Octuber 24, 2023].
- U. Savcı, "Step by Step Customer Segmentation using K-Means in Python," 2022, [Online]. Available: https://medium.com/@ugursavci/step-by-step-customer-segmentation-using-k-means-and-pca-in-python-5733822295b6. [Accessed: Octuber 24, 2023].