202051029-section1-ankur-shukla

October 22, 2023

[16]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[30]:
```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

1 New Section

[17]:
```
import pandas as pd
df = pd.read_excel("/content/drive/MyDrive/RetailPulseAssignmentData.
↪xlsx",sheet_name=['Year 2009-2010','Year 2010-2011'])
```

[18]:
```
# Lets
df1=df['Year 2009-2010']
# df2=df['Year 2010-2011']
```

1.0.1 Around 5 lakh data point we have

[19]:
```
# Check the shape of the data
print(df1.shape)
```

```
(525461, 8)
```

Description and Country seems to have object type . May required to to change to other dtype to include in analysis

[20]:
```
# Check the data types of each column
print(df1.dtypes)
```

```
Invoice                 object
StockCode               object
Description             object
Quantity                 int64
```

```
InvoiceDate    datetime64[ns]
Price                 float64
Customer ID           float64
Country                object
dtype: object
```

0.005% of data missing in description column .

[21]:
```python
# Check for any missing values
print(df1.isnull().sum())
```

```
Invoice             0
StockCode           0
Description      2928
Quantity            0
InvoiceDate         0
Price               0
Customer ID    107927
Country             0
dtype: int64
```

[22]:
```python
# Look at the summary statistics
print(df1.describe())
```

```
            Quantity          Price    Customer ID
count  525461.000000  525461.000000  417534.000000
mean       10.337667       4.688834   15360.645478
std       107.424110     146.126914    1680.811316
min     -9600.000000  -53594.360000   12346.000000
25%         1.000000       1.250000   13983.000000
50%         3.000000       2.100000   15311.000000
75%        10.000000       4.210000   16799.000000
max     19152.000000   25111.090000   18287.000000
```

1.0.2 Key Point Concluded

- 417534 unique customers in the data *mean Quantity is 10.34, but the standard deviation is large (107.42) indicating there is high variability in the quantities purchased* maximum quantity is 19,152 showing there are some very large orders *mean Price is £4.69. The large standard deviation (146.13) and wide range (-£53,594 to £25,111) indicates there is high variability in the prices* total count of 525461 transactions is larger than the number of unique customers, indicating customers are making multiple purchases over time

[97]:
```python
# Look at the first 5 rows
df1.head(10)
```

[97]:
```
  Invoice StockCode                          Description  Quantity  \
0  489434     85048  15CM CHRISTMAS GLASS BALL 20 LIGHTS        12
1  489434    79323P                   PINK CHERRY LIGHTS        12
```

```
2  489434    79323W                    WHITE CHERRY LIGHTS        12
3  489434     22041          RECORD FRAME 7" SINGLE SIZE          48
4  489434     21232        STRAWBERRY CERAMIC TRINKET BOX         24
5  489434     22064             PINK DOUGHNUT TRINKET POT         24
6  489434     21871                   SAVE THE PLANET MUG         24
7  489434     21523   FANCY FONT HOME SWEET HOME DOORMAT          10
8  489435     22350                             CAT BOWL          12
9  489435     22349        DOG BOWL , CHASING BALL DESIGN         12

          InvoiceDate  Price  Customer ID         Country
0 2009-12-01 07:45:00   6.95      13085.0  United Kingdom
1 2009-12-01 07:45:00   6.75      13085.0  United Kingdom
2 2009-12-01 07:45:00   6.75      13085.0  United Kingdom
3 2009-12-01 07:45:00   2.10      13085.0  United Kingdom
4 2009-12-01 07:45:00   1.25      13085.0  United Kingdom
5 2009-12-01 07:45:00   1.65      13085.0  United Kingdom
6 2009-12-01 07:45:00   1.25      13085.0  United Kingdom
7 2009-12-01 07:45:00   5.95      13085.0  United Kingdom
8 2009-12-01 07:46:00   2.55      13085.0  United Kingdom
9 2009-12-01 07:46:00   3.75      13085.0  United Kingdom
```

- Quantity mean (10.34) is greater than the median (3), indicating the distribution is likely right skewed.
- positive skew suggests there are more low quantity transactions than high, but some very large outliers on the high end
- mean (£4.69) is greater than the median (£2.10), indicating a right skewed distribution.
- more low priced transactions than high priced, but some expensive outliers extending the right tail.

[99]:

```python
# Combine the two DataFrames into one DataFrame
df3 = pd.concat([df['Year 2009-2010'], df['Year 2010-2011']], ignore_index=True)
```

[100]:

```python
df3.shape
df3.head(10)
```

[100]:

```
  Invoice StockCode                          Description  Quantity  \
0  489434     85048  15CM CHRISTMAS GLASS BALL 20 LIGHTS        12
1  489434    79323P                   PINK CHERRY LIGHTS        12
2  489434    79323W                  WHITE CHERRY LIGHTS        12
3  489434     22041         RECORD FRAME 7" SINGLE SIZE         48
4  489434     21232       STRAWBERRY CERAMIC TRINKET BOX        24
5  489434     22064           PINK DOUGHNUT TRINKET POT         24
6  489434     21871                  SAVE THE PLANET MUG        24
7  489434     21523  FANCY FONT HOME SWEET HOME DOORMAT         10
8  489435     22350                            CAT BOWL         12
9  489435     22349       DOG BOWL , CHASING BALL DESIGN        12
```

```
          InvoiceDate  Price  Customer ID         Country
0 2009-12-01 07:45:00   6.95      13085.0  United Kingdom
1 2009-12-01 07:45:00   6.75      13085.0  United Kingdom
2 2009-12-01 07:45:00   6.75      13085.0  United Kingdom
3 2009-12-01 07:45:00   2.10      13085.0  United Kingdom
4 2009-12-01 07:45:00   1.25      13085.0  United Kingdom
5 2009-12-01 07:45:00   1.65      13085.0  United Kingdom
6 2009-12-01 07:45:00   1.25      13085.0  United Kingdom
7 2009-12-01 07:45:00   5.95      13085.0  United Kingdom
8 2009-12-01 07:46:00   2.55      13085.0  United Kingdom
9 2009-12-01 07:46:00   3.75      13085.0  United Kingdom
```

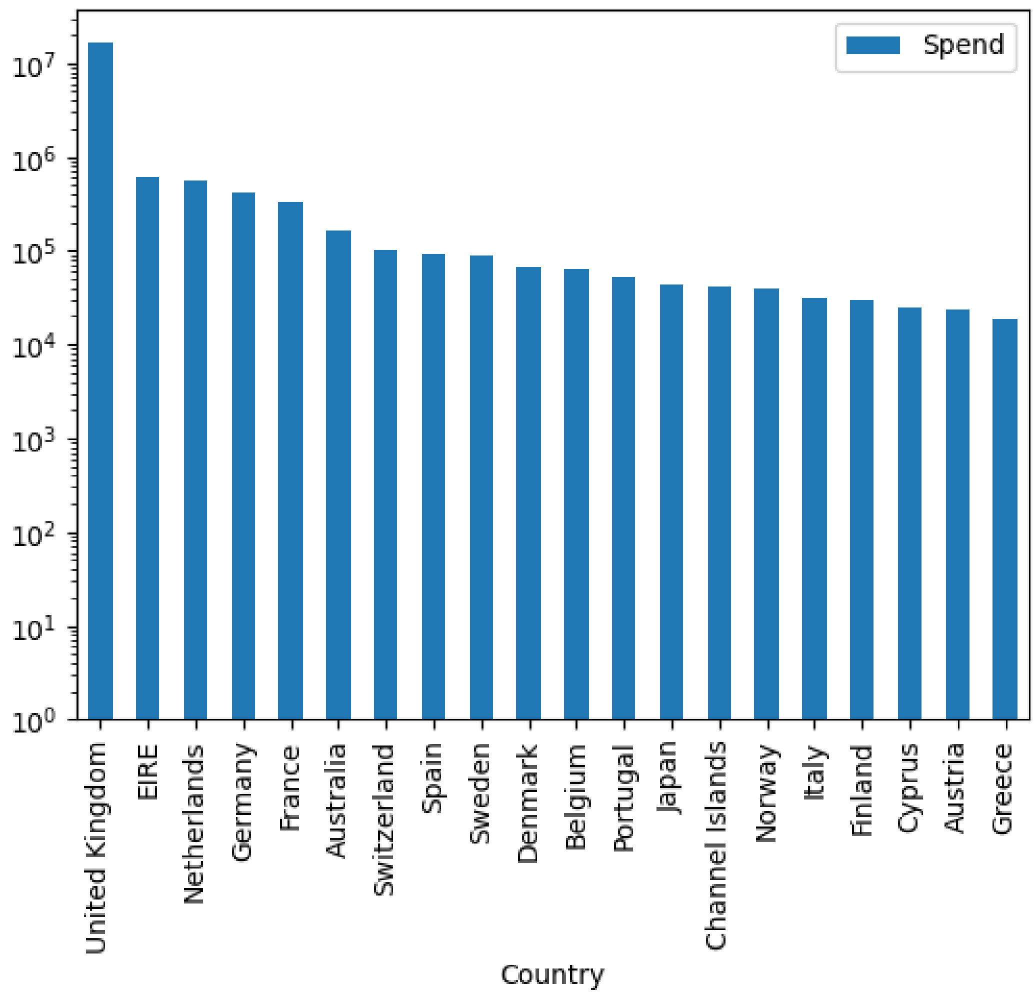
[108]:
```
# Total money spent by customers from each country
df3['Total'] = df3['Price'] * df3['Quantity']
temp = df3.groupby('Country')['Total'].sum()
tempDF = pd.DataFrame({'Country':temp.index, 'Spend':temp.values})
tempDF = tempDF.sort_values('Spend', ascending=False)
tempDF = tempDF.iloc[:20]
tempDF.head()
```

[108]:

	Country	Spend
40	United Kingdom	1.638258e+07
11	EIRE	6.155196e+05
26	Netherlands	5.485249e+05
15	Germany	4.179886e+05
14	France	3.281918e+05

[109]:
```
tempDF.plot(x='Country', y='Spend', kind='bar', log=True)
```

[109]:
```
<Axes: xlabel='Country'>
```

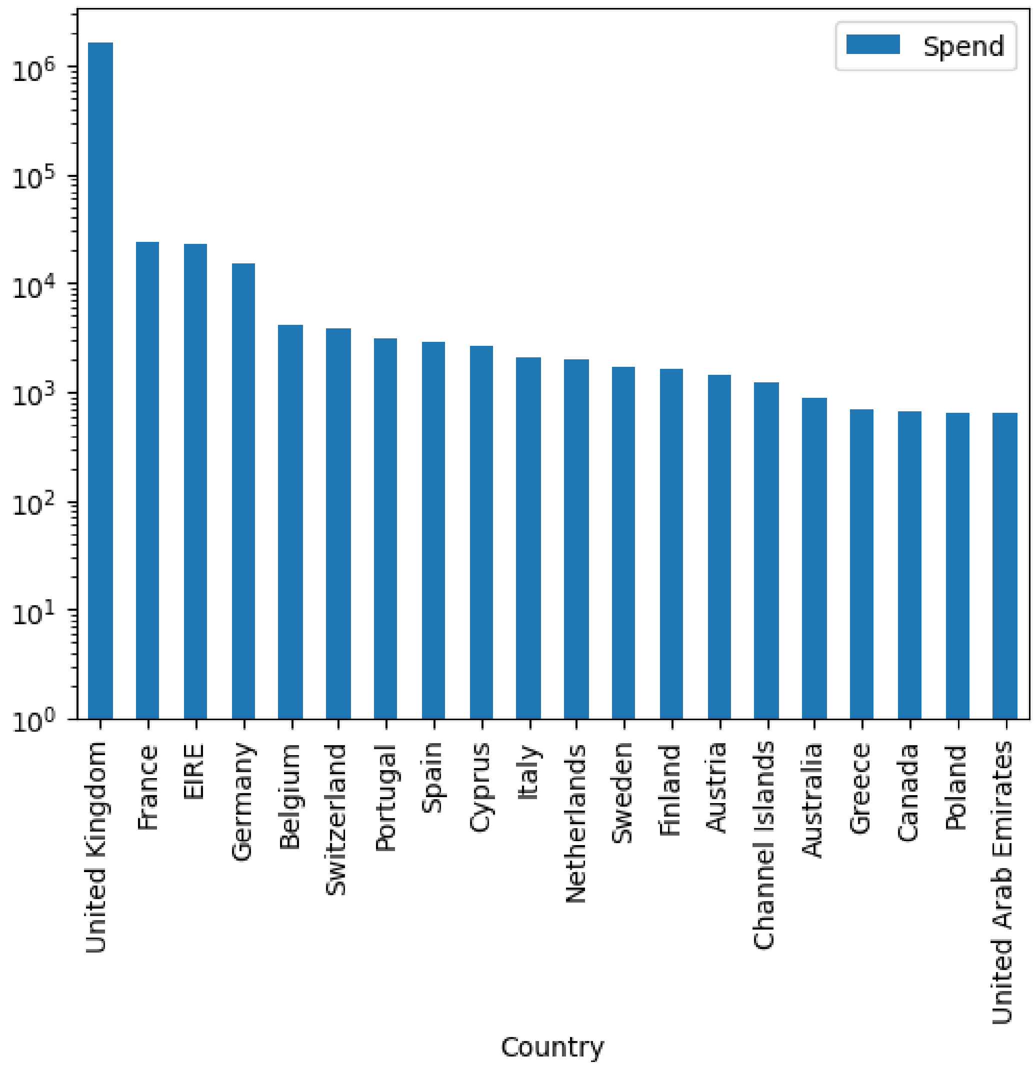


1.0.3 Country like UK , EIRE , Netherland Germany bring most of sales .

1.0.4 Rest of other country also bring consistant result

[110]:
```
# Total money spent by customers from each country
df3['Total'] = df3['Price'] / df3['Quantity']
temp = df3.groupby('Country')['Total'].sum()
tempDF = pd.DataFrame({'Country':temp.index, 'Spend':temp.values})
tempDF = tempDF.sort_values('Spend', ascending=False)
tempDF = tempDF.iloc[:20]
tempDF.head()
tempDF.plot(x='Country', y='Spend', kind='bar', log=True)
```

[110]:
```
<Axes: xlabel='Country'>
```



1.0.5 Surprisingly France , Belgium along with UK , Germany bring high value sales . (price/quantity)

1.1 Q1 Can customers be segmented into different categories? If yes then perform analysis on the same and also propose categories. If no, then explain why?

[ ]:

```
```

[42]:

```
df3['Price per Quantity'] = df3['Price'] / df3['Quantity']
# Select the relevant columns
data = df3[['Customer ID', 'Price', 'Price per Quantity']]
data=data[data['Price'] >= 0 ]
data=data[data['Price per Quantity']>=0]
```

1.1.1 There is negetive price value . It can be assumed as returned item . So customer can be divided into 2 categories . One who did not return purchased item and one who return

[47]:
```
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data[['Price']])
```

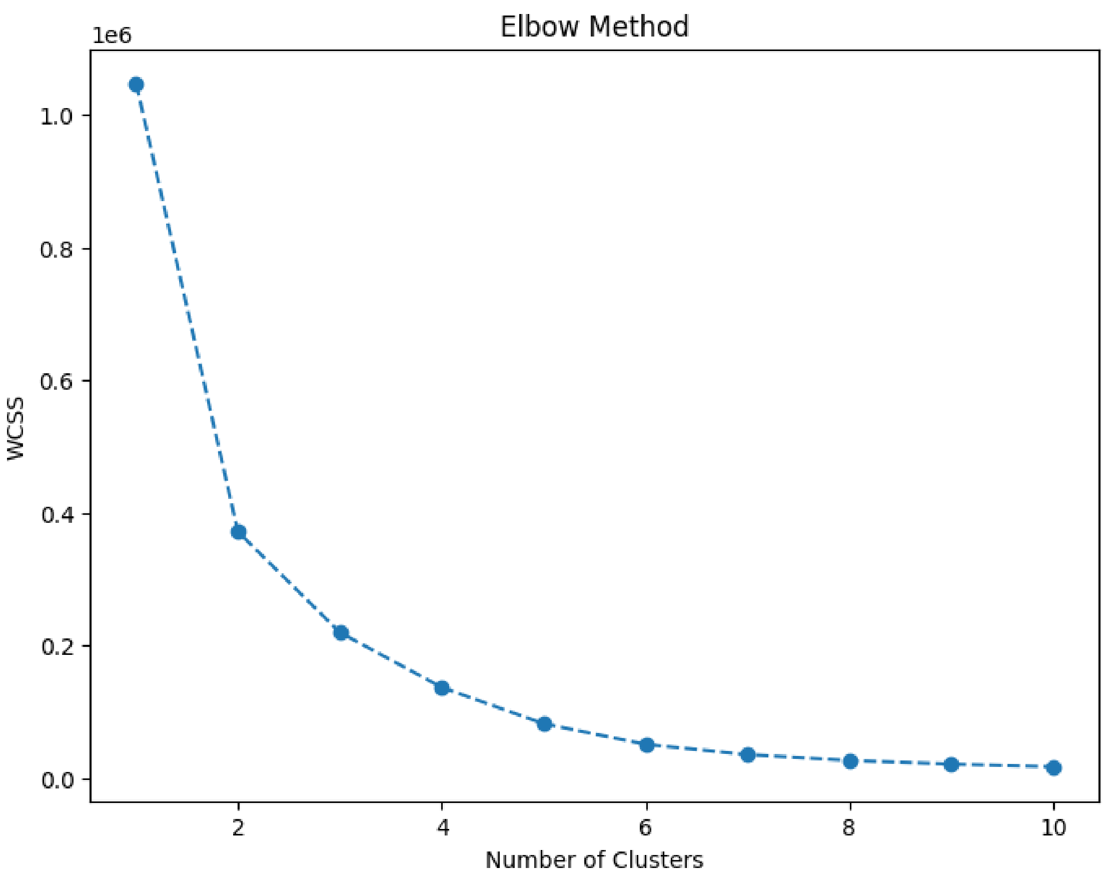
1.1.2 here doing clustering of consumer who purchased but not returned

[48]:
```
wcss = []  # Within-cluster sum of squares

for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=0)
    kmeans.fit(scaled_data)
    wcss.append(kmeans.inertia_)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

[49]:
```
# Plot the elbow method graph
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), wcss, marker='o', linestyle='--')
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```



1.1.3 From elbow method it seems to have 4 or 5 group clustering can be done on the basis of price and price per quantity

[52]:

```
num_clusters = 4
kmeans = KMeans(n_clusters=num_clusters, init='k-means++', random_state=0)
data['Cluster'] = kmeans.fit_predict(scaled_data)

# Calculate cluster centers
cluster_centers = scaler.inverse_transform(kmeans.cluster_centers_)

# Define the price ranges for each cluster
cluster_ranges = []

for i in range(num_clusters):
    if i == 0:
        lower_bound = 0
    else:
        lower_bound = cluster_centers[i - 1][0]
    upper_bound = cluster_centers[i][0]
    cluster_ranges.append((lower_bound, upper_bound))

# Visualize the clusters
plt.figure(figsize=(8, 6))
plt.scatter(data['Price'], [0] * len(data), c=data['Cluster'], cmap='rainbow')
plt.title('Clustered Customers Based on Price')
plt.xlabel('Price')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

[53]:
```
cluster_ranges
```

[53]:
```
[(0, 3.633138547736626),
 (3.633138547736626, 9045.30705882353),
 (9045.30705882353, 1453.9028888888895),
 (1453.9028888888895, 25111.09)]
```

1.1.4 we see our consumer are divided in 4 catecogry of purchase price

[54]:
```
cluster_counts = data['Cluster'].value_counts()
print(cluster_counts)
```

```
0    1047675
2        180
1         17
3          1
Name: Cluster, dtype: int64
```

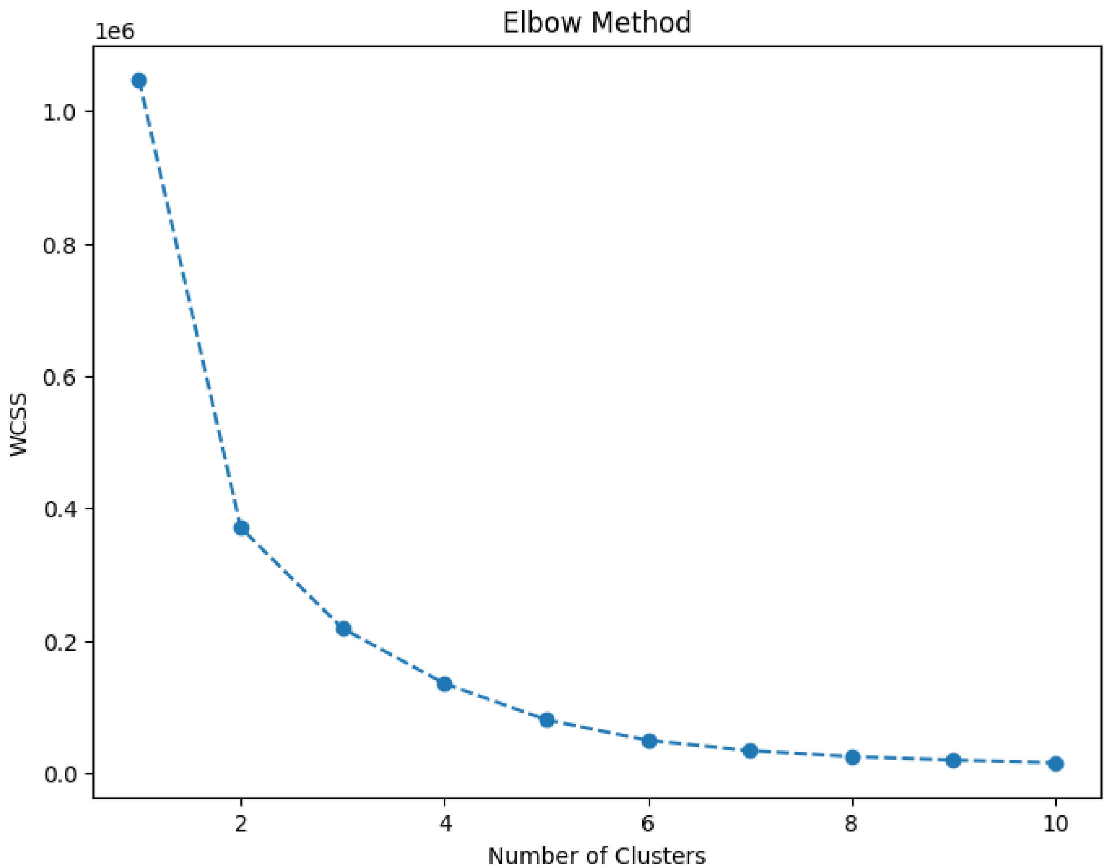
1.1.5 so out most of consumer lie in $0 to $3.63 price purchase

[55]:

```
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data[['Price per Quantity']])
wcss = []  # Within-cluster sum of squares

for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=0)
    kmeans.fit(scaled_data)
    wcss.append(kmeans.inertia_)

# Plot the elbow method graph
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), wcss, marker='o', linestyle='--')
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

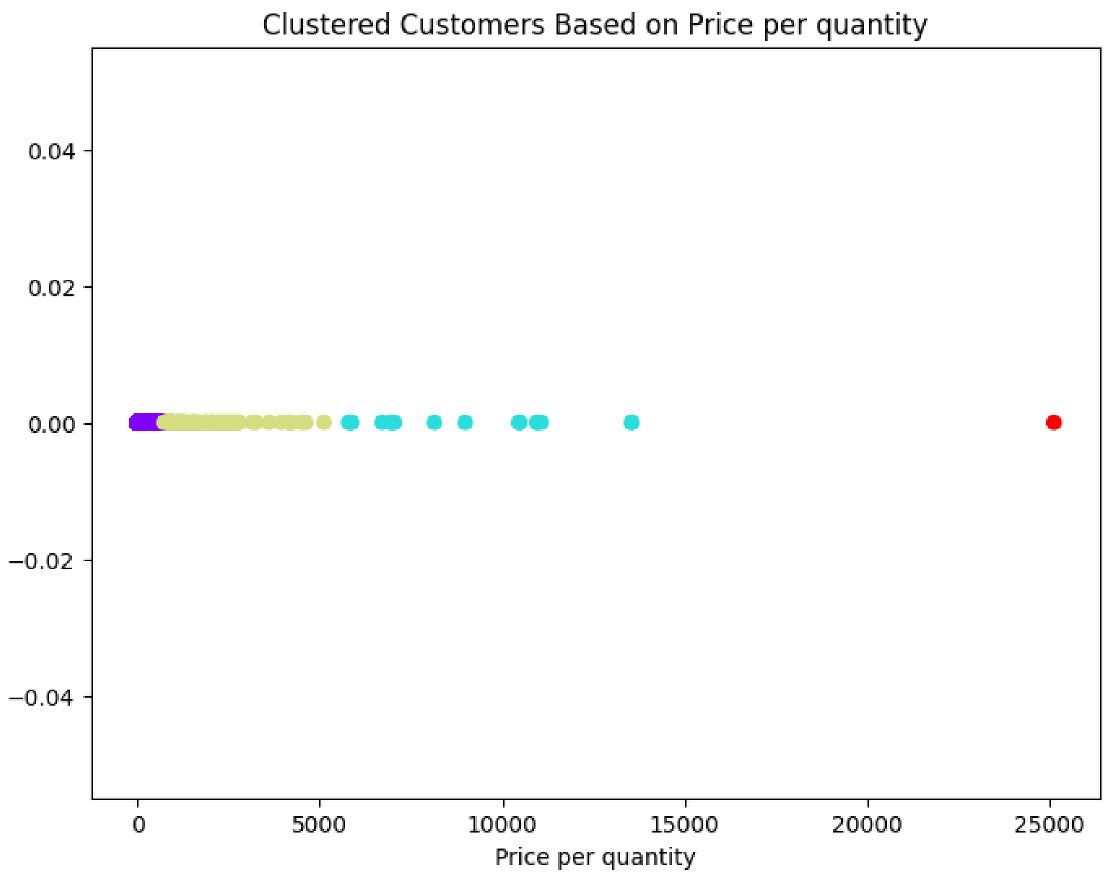


1.1.6 on the basis of price per quantity also it seems to have 4 cluster

[56]:
```
num_clusters = 4
kmeans = KMeans(n_clusters=num_clusters, init='k-means++', random_state=0)
data['Cluster'] = kmeans.fit_predict(scaled_data)
```

```
# Calculate cluster centers
cluster_centers = scaler.inverse_transform(kmeans.cluster_centers_)

# Define the price ranges for each cluster
cluster_ranges = []

for i in range(num_clusters):
    if i == 0:
        lower_bound = 0
    else:
        lower_bound = cluster_centers[i - 1][0]
    upper_bound = cluster_centers[i][0]
    cluster_ranges.append((lower_bound, upper_bound))

# Visualize the clusters
plt.figure(figsize=(8, 6))
plt.scatter(data['Price per Quantity'], [0] * len(data), c=data['Cluster'],␣
 ↪cmap='rainbow')
plt.title('Clustered Customers Based on Price per quantity')
plt.xlabel('Price per quantity')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

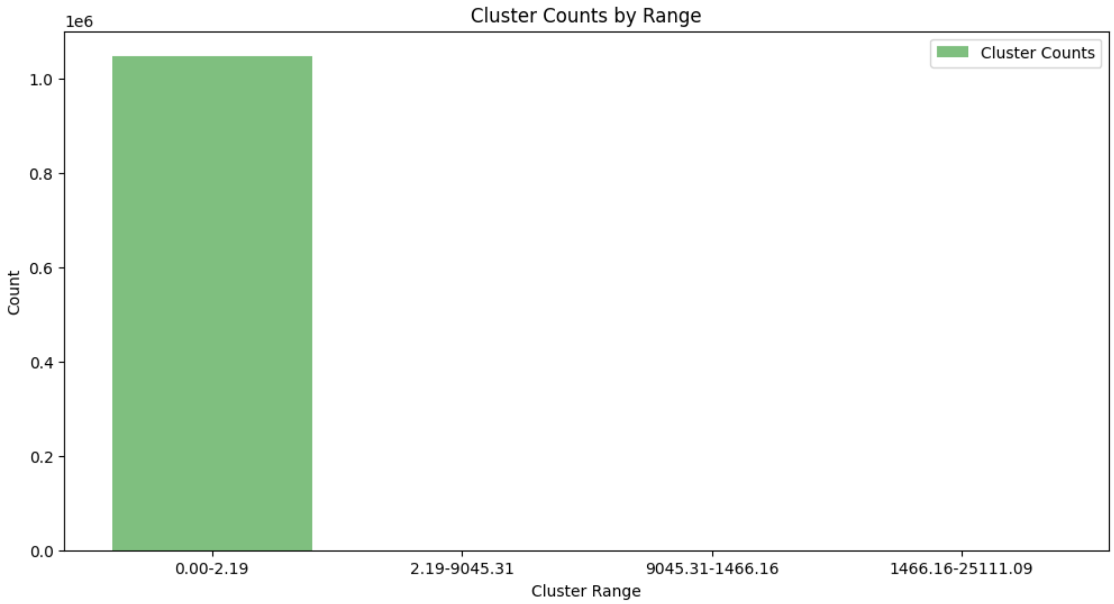


[57]:
```
print(cluster_ranges)
cluster_counts = data['Cluster'].value_counts()
print(cluster_counts)
```

```
[(0, 2.194496127646506), (2.194496127646506, 9045.30705882353),
(9045.30705882353, 1466.163785310734), (1466.163785310734, 25111.09)]
0    1047678
2        177
1         17
3          1
Name: Cluster, dtype: int64
```

[68]:
```
# Plot cluster counts with cluster ranges on the x-axis
lower_bounds = [range[0] for range in cluster_ranges]
upper_bounds = [range[1] for range in cluster_ranges]

# Plot cluster counts with cluster ranges on the x-axis
plt.figure(figsize=(12, 6))
```

```
plt.bar(range(len(cluster_counts)), cluster_counts, alpha=0.5, color='g',␣
↪align='center', label='Cluster Counts')
plt.xlabel('Cluster Range')
plt.ylabel('Count')
plt.title('Cluster Counts by Range')
plt.xticks(range(len(cluster_counts)), [f'{lb:.2f}-{ub:.2f}' for lb, ub in␣
↪zip(lower_bounds, upper_bounds)])
plt.legend()

plt.show()
```



1.1.7 From above 2 clustering result , Most of transaction occur in item value 0 to 3.6 and per quantitiy sales occur is of 0 to 2.19

[ ]:

```

```

1.2 Q2 How would you define a loyal customer?

[76]:

```
print("Total consumer entry in trasaction :",len(df3['Customer ID']))
print("Unique consumer :",df3['Customer ID'].nunique())
```

```
Total consumer entry in trasaction : 1067371
Unique consumer : 5942
```

1.2.1 This means that there is consumer how make more number of transaction throughtout this 2 year. This can be one parameter of loyal consumer

1.2.2 Must also consider value of transaction the make . For now let us define it as number of visit.

[ ]:

```

```

[80]:

```
df_positive_price = df3[df3['Price'] >= 0]

# Group by 'Customer ID' and collect unique invoices
temp = df_positive_price.groupby(['Customer ID'])['Invoice'].unique()

temp = pd.DataFrame({'CustomerID': temp.index, 'Invoice': temp.values})
temp['CustomerID'] = temp['CustomerID'].astype(int)
# Calculate the number of returns for each customer
temp['Returns'] = [len(invoices) for invoices in temp['Invoice']]
temp_sorted = temp.sort_values(by='Returns', ascending=False)
temp_sorted.head(10)
```

[80]:

	CustomerID	Invoice	Returns
2565	14911	[489520, 490542, 490687, 490972, C490997, C491…	510
402	12748	[490362, C490748, 491759, 492224, 492423, 4925…	365
5495	17841	[489875, 490302, 490711, 490714, C491632, 4916…	289
2965	15311	[489514, 489733, 490301, 490504, 490817, 49112…	270
2260	14606	[490024, C490507, 490512, 490990, C491430, 491…	259
743	13089	[489877, 490694, 491670, 491828, C492213, C492…	247
1810	14156	[489546, 489550, 490127, C490721, C490934, C49…	202
2181	14527	[489792, C490269, 490286, 490941, 491192, 4917…	190
2300	14646	[489889, 489890, 490964, 492248, 492724, 49414…	164
1348	13694	[490320, 490324, 490327, 490725, 490726, 49181…	164

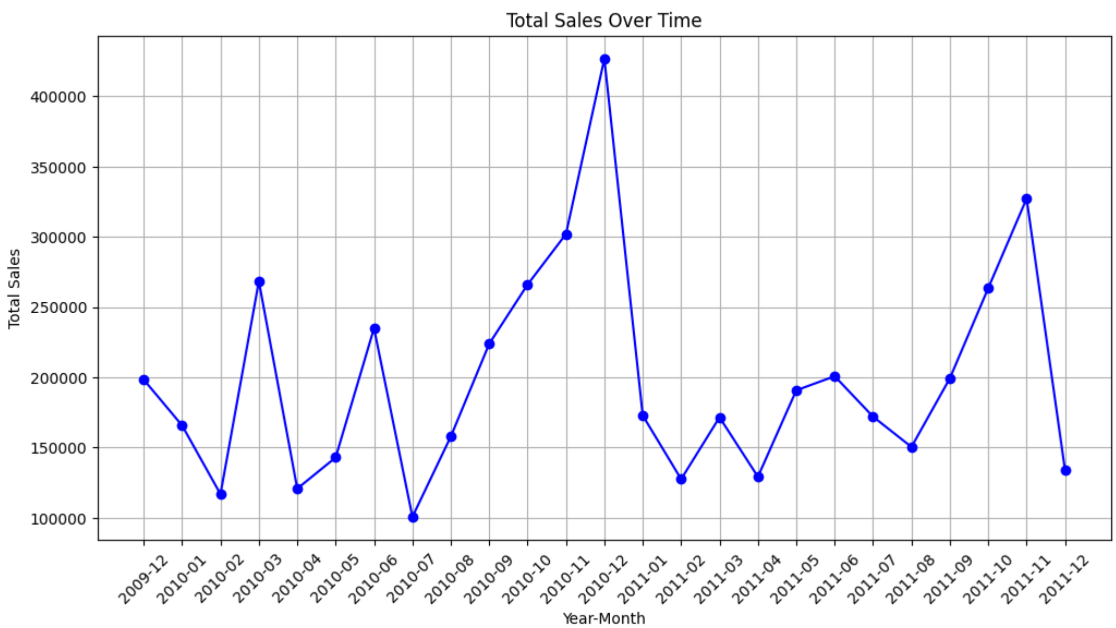
1.2.3 These are top 10 repeated costomer who come make make positive transaction and Hence can be treated as loyal consumer

1.3 Q3 What is the most popular time of year based on this sales data?

[83]:

```
# Convert 'InvoiceDate' to a datetime object
# Convert 'InvoiceDate' to a datetime object
df3['InvoiceDate'] = pd.to_datetime(df3['InvoiceDate'])

# Extract the year and month from 'InvoiceDate' for grouping
df3['YearMonth'] = df3['InvoiceDate'].dt.to_period('M')

# Group by 'YearMonth' and calculate the total sales for each month
monthly_sales = df3.groupby('YearMonth')['Price'].sum()

# Convert 'YearMonth' values to strings for plotting
```

```
monthly_sales.index = monthly_sales.index.strftime('%Y-%m')

# Plot the total sales over time
plt.figure(figsize=(12, 6))
plt.plot(monthly_sales.index, monthly_sales.values, marker='o', linestyle='-',␣
↪color='b')
plt.title('Total Sales Over Time')
plt.xlabel('Year-Month')
plt.ylabel('Total Sales')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```



1.3.1 From this its clear that number of transaction (so sales) increase in month of November than in usual

[ ]:

```
```

1.4 Q4 Is there any seasonality in data? Explain with supportive evidence?

1.4.1 Monthly Seasonality

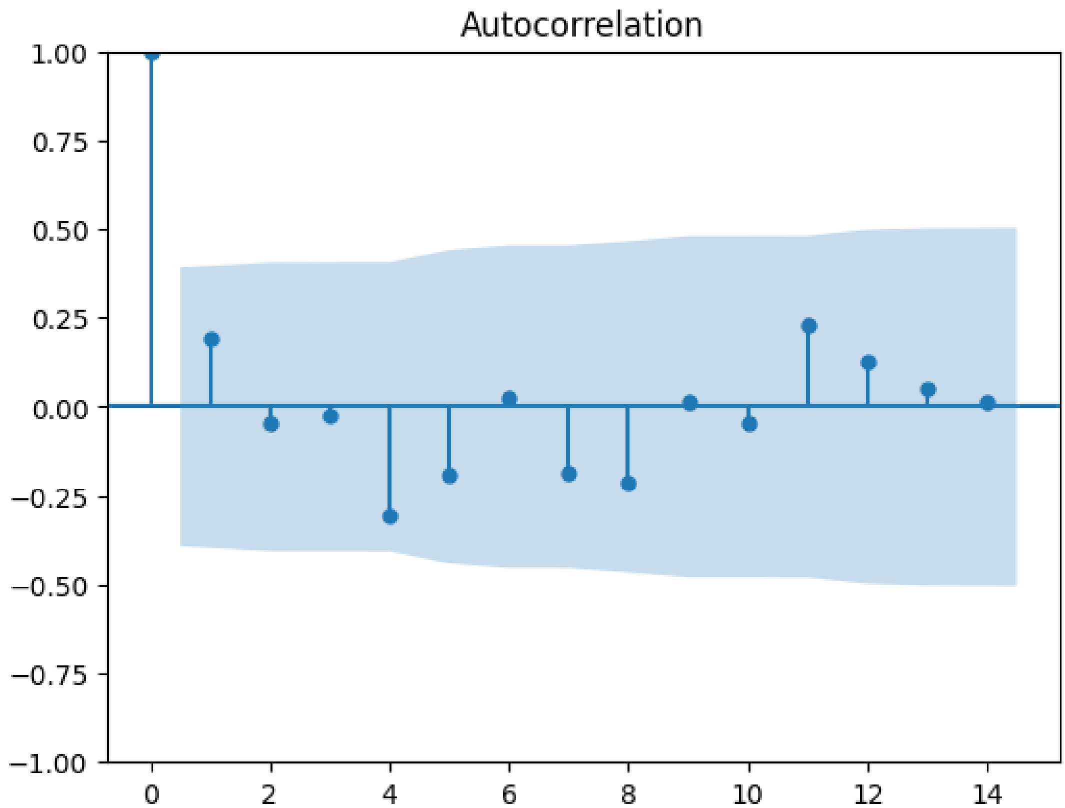
[91]:

```
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.tsa.seasonal import seasonal_decompose
```

```
from statsmodels.graphics.tsaplots import plot_acf



# Convert 'InvoiceDate' to a datetime object
df3['InvoiceDate'] = pd.to_datetime(df3['InvoiceDate'])

# Set 'InvoiceDate' as the index
df3.set_index('InvoiceDate', inplace=True)

# Resample data to a monthly frequency and sum the sales and prices for each␣
↪month
monthly_sales = df3.resample('M').sum()

# Perform seasonal decomposition to check for seasonality
result = seasonal_decompose(monthly_sales['Price'], model='additive')
result.plot()
plt.show()

# Plot the autocorrelation function (ACF) to identify seasonality
plot_acf(monthly_sales['Price'])
plt.show()
```

```
<ipython-input-91-b089ad153e17>:16: FutureWarning: The default value of
numeric_only in DataFrameGroupBy.sum is deprecated. In a future version,
numeric_only will default to False. Either specify numeric_only or select only
columns which should be valid for the function.
  monthly_sales = df3.resample('M').sum()
```


Price
400000
200000
Trend
210000
205000
200000
Seasonal
200000
0
Resid
10000
0
−10000
2010-01
2010-04
2010-07
2010-10
2011-01
2011-04
2011-07
2011-10

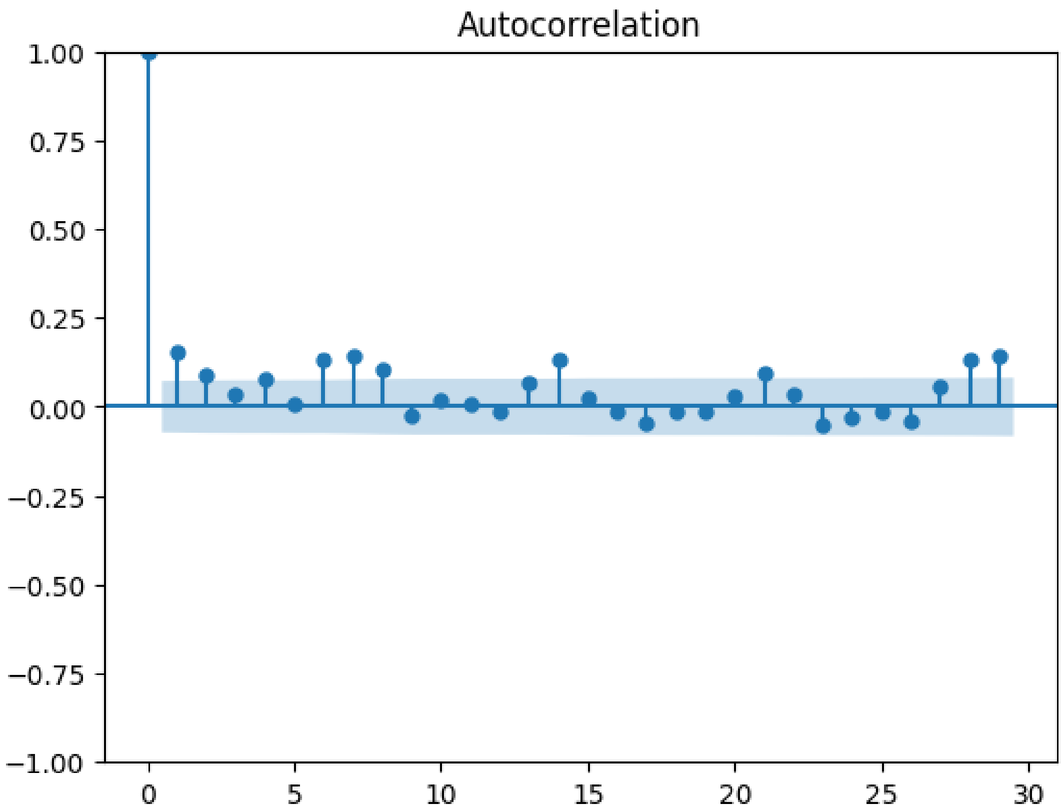


1.4.2 Daily Seasonality

[93]:
```
# Set 'InvoiceDate' as the index
# df3.set_index('InvoiceDate', inplace=True)

# Resample data to a daily frequency and sum the sales and prices for each day
daily_sales = df3.resample('D').sum()

# Perform seasonal decomposition to check for daily seasonality
result = seasonal_decompose(daily_sales['Price'], model='additive')
result.plot()
plt.show()

# Plot the autocorrelation function (ACF) to identify daily seasonality
plot_acf(daily_sales['Price'])
plt.show()
```

```
<ipython-input-93-ff934be8043a>:5: FutureWarning: The default value of
numeric_only in DataFrameGroupBy.sum is deprecated. In a future version,
numeric_only will default to False. Either specify numeric_only or select only
columns which should be valid for the function.
```

```
daily_sales = df3.resample('D').sum()
```

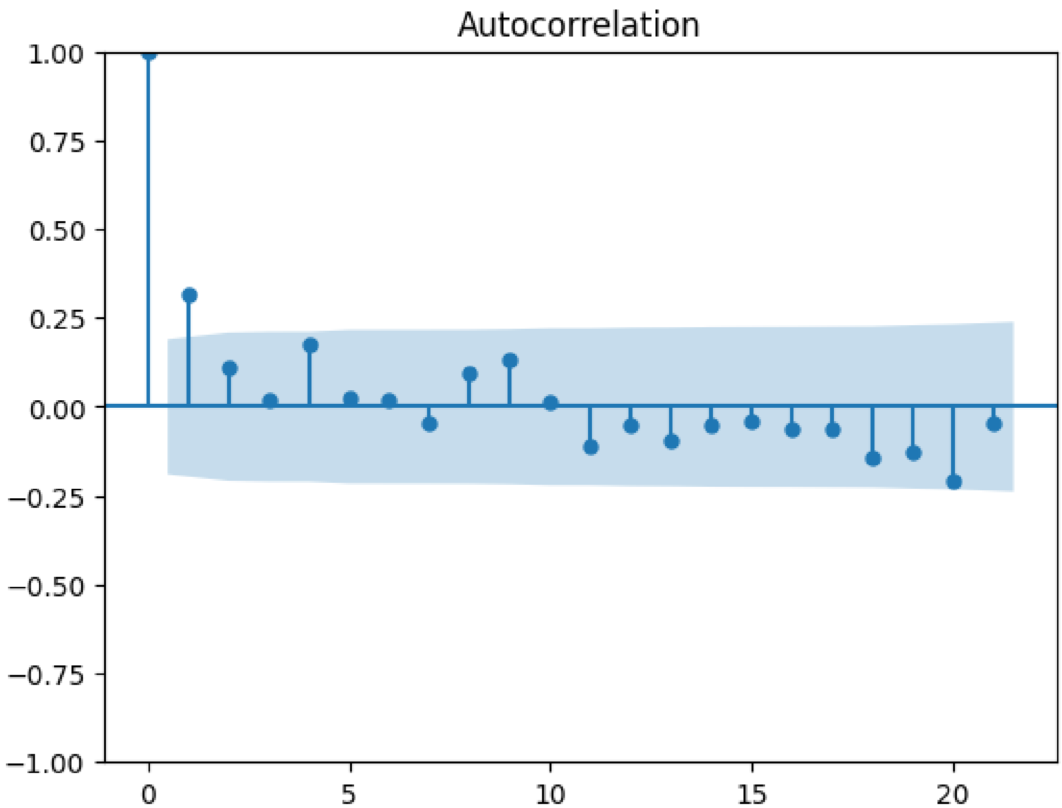


1.4.3 Weekly Seasonality

[94]:
```
weekly_sales = df3.resample('W-SUN').sum()

# Perform seasonal decomposition to check for weekly seasonality
result = seasonal_decompose(weekly_sales['Price'], model='additive')
result.plot()
plt.show()

# Plot the autocorrelation function (ACF) to identify weekly seasonality
plot_acf(weekly_sales['Price'])
plt.show()
```

```
<ipython-input-94-af74ad15295c>:1: FutureWarning: The default value of
numeric_only in DataFrameGroupBy.sum is deprecated. In a future version,
numeric_only will default to False. Either specify numeric_only or select only
columns which should be valid for the function.
  weekly_sales = df3.resample('W-SUN').sum()
```


Price
200000
0
Trend
48000
46000
44000
Seasonal
200000
0
Resid
10000
0
−10000
2010-01
2010-04
2010-07
2010-10
2011-01
2011-04
2011-07
2011-10



1.4.4 seasonal autocorrelation plot would show spikes at lags equal to the period of the seasonality.

1.4.5 NO clear seasonality is observerd on daily weekly and monthly plot

[ ]:

```

```

[ ]:

```

```

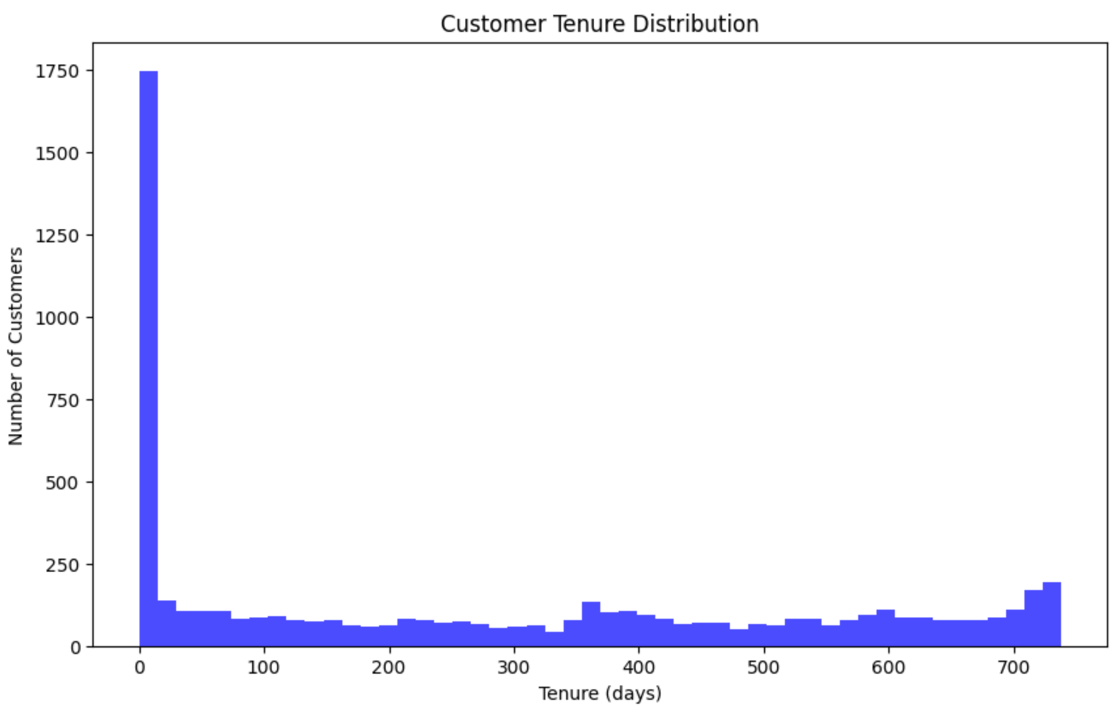
[105]:

```
import pandas as pd
import matplotlib.pyplot as plt


# Convert 'InvoiceDate' to a datetime object
df3['InvoiceDate'] = pd.to_datetime(df3['InvoiceDate'])

# Group data by 'Customer ID' and calculate customer tenure
customer_tenure = df3.groupby('Customer ID')['InvoiceDate'].agg(['min', 'max'])
customer_tenure['Tenure'] = (customer_tenure['max'] - customer_tenure['min']).
    ↪dt.days
print(customer_tenure.head(10))
```

```
                                   min                  max  Tenure
Customer ID
12346.0     2009-12-14 08:34:00 2011-01-18 10:17:00     400
12347.0     2010-10-31 14:20:00 2011-12-07 15:52:00     402
12348.0     2010-09-27 14:59:00 2011-09-25 13:13:00     362
12349.0     2009-12-04 12:49:00 2011-11-21 09:51:00     716
12350.0     2011-02-02 16:01:00 2011-02-02 16:01:00       0
12351.0     2010-11-29 15:23:00 2010-11-29 15:23:00       0
12352.0     2010-11-12 10:20:00 2011-11-03 14:37:00     356
12353.0     2010-10-27 12:44:00 2011-05-19 17:47:00     204
12354.0     2011-04-21 13:11:00 2011-04-21 13:11:00       0
12355.0     2010-05-21 11:59:00 2011-05-09 13:49:00     353
```

[106]:

```
# Plot a histogram of customer tenure
plt.figure(figsize=(10, 6))
plt.hist(customer_tenure['Tenure'], bins=50, color='blue', alpha=0.7)
plt.title('Customer Tenure Distribution')
plt.xlabel('Tenure (days)')
plt.ylabel('Number of Customers')
plt.show()
```



[107]:

```
mean_tenure = customer_tenure['Tenure'].mean()
median_tenure = customer_tenure['Tenure'].median()
```

```python
print("Mean Tenure:", mean_tenure)
print("Median Tenure:", median_tenure)
```

```
Mean Tenure: 275.3963312016156
Median Tenure: 225.0
```

1.4.6 Median of tenure is 225 . This means most of costomer engagement is of 225 days .

```
[ ]:
```

202051029-section2-ankur-shukla

October 22, 2023

1 Section 2 - Coding

In this section we will load and manipulate "unconventional" data files - for which you will be required to create a simple loading functionality and then be able to process and query a data file.

There is a "section2_data.txt" file attached to this IPython notebook. The data file contains few rows from a computer vision dataset. Each row corresponds to a frame in a video and contains some metadata and annotations over it.

The following notebook will pose some questions about reading and processing this data.

Feel free to use any existing python library to answer the questions.

[144]:
```
!head /kaggle/input/data-kirana/section2_data.txt
```


```
{"_i": 0, "frame": "frame_000.png", "video": "video000", "value": 39, "labels":
["bird"]}
{"_i": 1, "frame": "frame_001.png", "video": "video000", "value": 33, "labels":
["frog", "dog"]}
{"_i": 2, "frame": "frame_002.png", "video": "video000", "value": 25, "labels":
["panda", "panda"]}
{"_i": 3, "frame": "frame_003.png", "video": "video000", "value": 28, "labels":
["dog", "dog"]}
{"_i": 4, "frame": "frame_004.png", "video": "video000", "value": 16, "labels":
["cat"]}
{"_i": 5, "frame": "frame_005.png", "video": "video000", "value": 32, "labels":
["bird", "frog", "bird"]}
{"_i": 6, "frame": "frame_006.png", "video": "video000", "value": 35, "labels":
["bird", "dog"]}
{"_i": 7, "frame": "frame_000.png", "video": "video001", "value": 25, "labels":
["dog", "bird"]}
{"_i": 8, "frame": "frame_001.png", "video": "video001", "value": 16, "labels":
["dog", "panda", "bird"]}
{"_i": 9, "frame": "frame_002.png", "video": "video001", "value": 23, "labels":
["panda"]}
```


1.1 Section 1 - Design a data loader

Design a data structure, that give a file path `"section2_data.txt"`, it will read and parse the contents of the file above.

Q1 - Design the data structure with the following properties:

- The data structure is either callable or indexable. It will accepts a single parameter, as integer, and return the parsed contents of the row corresponding to the given index.
- The data structure needs to return the number of rows in the file (and in memory) when called with the python command `len(my_data_struct)`

[145]:
```
class Data_Struct_ank:
    def __init(self, file_path):
        self.data = []
        with open(file_path, 'r') as file:
            for line in file:
                row = json.loads(line.strip())  # Parse each line as JSON
                self.data.append(row)

    def __call__(self, index):
        if 0 <= index < len(self.data):
            return self.data[index]
        else:
            raise IndexError("Custom error (Ankur Shukla): Index out of range")

    def __len__(self):
        return len(self.data)


    def __getitem__(self, index):
        return self.data[index]

    def __iter__(self):
        self.current_index = 0
        return self

    def __next__(self):
        if self.current_index < len(self.data):
            result = self.data[self.current_index]
            self.current_index += 1
            return result
        else:
            raise StopIteration
```

[146]:
```
my_data_struct = DataLoader("/kaggle/input/data-kirana/section2_data.txt")
```

[147]:
```
row = my_data_struct(0)
print(row)
num_rows = len(my_data_struct)
print(f"Number of rows: {num_rows}")
```

```
['{"_i": 0, "frame": "frame_000.png", "video": "video000", "value": 39,
```

```
"labels": ["bird"]}']
Number of rows: 51
```

Q2 - Prove that you can initialize the reader and then calculate its length `len(reader)` and print the 26th and 43rd elements of the dataset.

[148]:
```
## YOUR SOLUTION
num_rows = len(my_data_struct)
print(f"Number of rows: {num_rows}")
```

```
Number of rows: 51
```

1.1.1 Printing the 26th and 43rd elements of the dataset (indices are 0-based)

[149]:
```
if num_rows >= 26:
    row_26 = my_data_struct(25)  # 26th element
    print("26th Element:", row_26)
```

```
26th Element: ['{"_i": 25, "frame": "frame_003.png", "video": "video003",
"value": 24, "labels": ["panda"]}']
```

[150]:
```
if num_rows >= 43:
    row_43 = my_data_struct(42)  # 43rd element
    print("43rd Element:", row_43)
```

```
43rd Element: ['{"_i": 42, "frame": "frame_002.png", "video": "video004",
"value": 32, "labels": ["panda", "bird", "cat"]}']
```

1.1.2 For index pass greater than 51 must throw error as in code

[151]:
```
# row_43 = my_data_struct(52)

# output :
#    ␣
 ↪---------------------------------------------------------------------------
# IndexError                                Traceback (most recent call last)
# Cell In[21], line 1
# ----> 1 row_43 = my_data_struct(52)

# Cell In[14], line 15, in DataLoader.__call__(self, index)
#      13     return self.data[index]
#      14 else:
# ---> 15     raise IndexError(" Coustom error (Ankur Shukla) : Index out of␣
 ↪range")

# IndexError:  Coustom error (Ankur Shukla) : Index out of range
```

1.2 Section 2 - Process the data

Q1 - Write an algorithm that will generate a dictionary with key/value pairs, where the keys are the name of each unique video in the dataset and the value is the number of frames of that video.

[152]:
```
### YOUT SOLUTION
```

[153]:
```
def video_frame_count(my_data_struct):

    video_frame_count = {}
    for i in range(len(my_data_struct)):

# Iterate through the data structure to process each row
        for row in my_data_struct(i):


            data_dict = json.loads(row)
            video_name = data_dict['video']

    # Check if the video_name is already in the dictionary, if not, add it with␣
 ↪the value as 1
            if video_name not in video_frame_count:
                video_frame_count[video_name] = 1
            else:

        # Increment the count for the existing video_name
                 video_frame_count[video_name] += 1
    return video_frame_count
```

[ ]:
```

```

[154]:
```
data_dict=video_frame_count(my_data_struct)
print(data_dict)
```

```
{'video000': 7, 'video001': 10, 'video002': 5, 'video003': 18, 'video004': 11}
```

[ ]:
```

```

Q2 - Write an algorithm that will generate a dictionary with key/value pairs, where the keys are the name of each unique video in the dataset and the value is the sum of the `value` field of all the frames containing a `dog`.

[155]:
```
my_data_struct(0)
```

[155]:
```
['{"_i": 0, "frame": "frame_000.png", "video": "video000", "value": 39,
"labels": ["bird"]}']
```

[156]:
```python
### YOUR SOLUTION
import json
def video_value_sum_with_dog(my_data_struct):
    video_value_sum = {}


    for i in range(len(my_data_struct)):
        row = my_data_struct(i)

        # Iterate through the rows
        for row_str in row:
            data_dict = json.loads(row_str)
            video_name = data_dict['video']
            value = data_dict['value']

            # Check if 'dog' is in the 'labels' field
            if 'dog' in data_dict['labels']:
                # Check if the video_name is already in the dictionary, if not,␣
 ↪add it with the value as 0
                if video_name not in video_value_sum:
                    video_value_sum[video_name] = 0
                # Add the 'value' to the existing sum
                video_value_sum[video_name] += value

    return video_value_sum
```

[157]:
```python
video_value_sum_with_dog(my_data_struct)
```

[157]:
```
{'video000': 96,
 'video001': 69,
 'video002': 91,
 'video003': 129,
 'video004': 49}
```

Q3 - Last, create an algorithm that returns a dictionary with the count of each of the animal types in the dataset, excluding occurrences in `video003` and rows where the value is odd.

[158]:
```python
### YOUR SOLUTION
def animal_count(my_data_struct):
    animal_type_count = {}

    # Iterate through the data structure
    for i in range(len(my_data_struct)):
        row = my_data_struct(i)
```

```python
        for row_str in row:
            data_dict = json.loads(row_str)
            video_name = data_dict['video']
            value = data_dict['value']
            animal_types = data_dict['labels']

            # Exclude 'video003' and rows with odd 'value'
            if video_name != 'video003' and value % 2 == 0:
                for animal_type in animal_types:
                    # Increment the count for each animal type
                    if animal_type not in animal_type_count:
                        animal_type_count[animal_type] = 1
                    else:
                        animal_type_count[animal_type] += 1

    return animal_type_count
```

[159]:
```python
animal_count(my_data_struct)
```

[159]: {'dog': 10, 'cat': 7, 'bird': 6, 'frog': 8, 'panda': 4}

[ ]:
```python

```