WPI

MIS - 502 (Data management for Analytics)

Final Project: Bike Sales Data Analysis

By Ishan Ratnakar Deshpande

Data Set Description:

The dataset comprises information related to bike sales transactions, including various attributes such as date, day, month, and year of the transaction. It also includes details about the customer, such as age, age group, gender, country, and state. Additionally, the dataset provides insights into the product details, including product category, sub-category, specific product, order quantity, unit cost, unit price, profit, cost, and revenue.

This dataset offers valuable information for analyzing bike sales trends, customer demographics, and profitability across different regions and product categories. Researchers and analysts can utilize this dataset to explore patterns, identify key factors influencing sales, and make data-driven decisions to optimize marketing strategies, inventory management, and product offerings in the bike industry.

Column Information:

1. Date: Date of the bike sales transaction.
2. Day: Day of the week corresponding to the transaction date.
3. Month: Month of the transaction date.
4. Year: Year of the transaction date.
5. Customer_Age: Age of the customer involved in the transaction.
6. Age_Group: Categorized age group of the customer (e.g., young adult, middle-aged, senior).
7.Customer_Gender: Gender of the customer (e.g., male, female, other).
8. Country: Country where the transaction took place.
9. State: State or region within the country where the transaction occurred.
10. Product_Category: Category of the bike product (eg Bikes, Accessories, Clothing).
11. Sub_Category: Subcategory of the bike product (e.g., Bike Racks, Bike Stands, Cleaners, Helmets, etc).
12. Product: Specific model or type of bike product.
13. Order_Quantity: Quantity of bikes ordered in the transaction.
14. Unit_Cost: Cost of each bike unit.
15. Unit_Price: Price of each bike unit.
16. Profit: Profit generated from the transaction.
17. Cost: Total cost incurred for the transaction.
18. Revenue: Total revenue generated from the transaction.

Number of Records: 113036
Size of the file: 14.4 MB

Potential Outcomes:

1. Sales Trends: Identify trends in bike sales over time (daily, monthly, yearly) to understand patterns and seasonality.
2. Customer Demographics: Analyze customer age groups, genders, and geographical locations to target specific demographics for marketing campaigns.
3. Product Performance: Evaluate the performance of different bike categories and sub-categories to identify best-selling products and optimize inventory.
4. Profitability Analysis: Assess profitability metrics such as profit margins, revenue, and costs to optimize pricing strategies and improve profitability.
5. Geographical Insights: Explore sales distribution across different countries and states to identify regions with high demand and potential market expansion opportunities.
6. Customer Behavior: Analyze order quantities and purchase frequency to understand customer behavior and preferences.
7. Seasonal Effects: Investigate seasonal effects on bike sales to plan inventory management and marketing strategies accordingly.
8. Correlation Analysis: Explore correlations between variables such as customer age, product categories, and sales performance to uncover insights and make data-driven decisions.
9. Forecasting: Use historical sales data to forecast future sales trends and demand, aiding in inventory planning and resource allocation.
10. Customer Segmentation: Segment customers based on demographics and purchasing behavior to personalize marketing efforts and enhance customer satisfaction.

Data Wrangling:

1] The dataset was checked for duplicate values. A total of 1000 duplicate entries were found.

CODE:

main.py

```python
import pandas as pd
import seaborn as sn
import numpy as np
import matplotlib.pyplot as plt
desired_width = 320
pd.set_option('display.width', desired_width)
pd.set_option('display.max_columns', 18)
df = pd.read_csv(r'C:\Users\ishan\OneDrive\PowerBI Dashboards\Sales.csv')
print(df.duplicated().sum())
duplicated_records_all = df[df.duplicated(keep=False)]
print(duplicated_records_all.head(6))
```

OUTPUT:

```
C:\Users\ishan\PycharmProjects\final_project\.venv\Scripts\python.exe C:\Users\ishan\PycharmProjects\final_project\main.py
1000
           Date  Day      Month  Year  Customer_Age       Age_Group Customer_Gender    Country           State Product_Category      Sub_Category                Product  Or
1020  12/19/2013   19   December  2013            22     Youth (<25)               M  Australia  New South Wales      Accessories       Bike Stands  All-Purpose Bike Stand
1021  12/19/2013   19   December  2013            22     Youth (<25)               M  Australia  New South Wales      Accessories       Bike Stands  All-Purpose Bike Stand
1090   9/30/2015   30  September  2015            42  Adults (35-64)               F  Australia         Victoria      Accessories  Bottles and Cages    Mountain Bottle Cage
1091   9/30/2015   30  September  2015            42  Adults (35-64)               F  Australia         Victoria      Accessories  Bottles and Cages    Mountain Bottle Cage
1092  10/24/2013   24    October  2013            42  Adults (35-64)               F  Australia         Victoria      Accessories  Bottles and Cages    Mountain Bottle Cage
1093  10/24/2013   24    October  2013            42  Adults (35-64)               F  Australia         Victoria      Accessories  Bottles and Cages    Mountain Bottle Cage

Process finished with exit code 0
```

Removal of the duplicate records

CODE:

main.py

```python
import pandas as pd
import seaborn as sn
import numpy as np
import matplotlib.pyplot as plt
desired_width = 320
pd.set_option('display.width', desired_width)
pd.set_option('display.max_columns', 18)
df = pd.read_csv(r'C:\Users\ishan\OneDrive\PowerBI Dashboards\Sales.csv')
#print(df.duplicated().sum())
duplicated_records_all = df[df.duplicated(keep=False)]
#print(duplicated_records_all.head(6))
deduplicated_df = df.drop_duplicates()
print(deduplicated_df.duplicated().sum())
df_new=deduplicated_df
print(df_new.info())

```

OUTPUT:

```
C:\Users\ishan\PycharmProjects\final_project\.venv\Scripts\python.exe C:\Users\ishan\PycharmProjects\final_project\main.py
0
<class 'pandas.core.frame.DataFrame'>
Index: 112036 entries, 0 to 113035
Data columns (total 18 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   Date              112036 non-null  object
 1   Day               112036 non-null  int64
 2   Month             112036 non-null  object
 3   Year              112036 non-null  int64
 4   Customer_Age      112036 non-null  int64
 5   Age_Group         112036 non-null  object
 6   Customer_Gender   112036 non-null  object
 7   Country           112036 non-null  object
 8   State             112036 non-null  object
 9   Product_Category  112036 non-null  object
 10  Sub_Category      112036 non-null  object
 11  Product           112036 non-null  object
 12  Order_Quantity    112036 non-null  int64
 13  Unit_Cost         112036 non-null  int64
 14  Unit_Price        112036 non-null  int64
 15  Profit            112036 non-null  int64
 16  Cost              112036 non-null  int64
 17  Revenue           112036 non-null  int64
dtypes: int64(9), object(9)
memory usage: 16.2+ MB
None
```

At the outset, the dataset contained 113,036 records, out of which 1,000 were identified as duplicates. Following the removal of these duplicate records, the dataset was left with a total of 112,036 unique records.

2] Datatype of the date column is object. It should be changed to the date datatype

CODE:

```
df_new['Date']=pd.to_datetime(df_new['Date'])
print(type(df_new.info()))
```

OUTPUT:

```
<class 'pandas.core.frame.DataFrame'>
Index: 112036 entries, 0 to 113035
Data columns (total 18 columns):
 #   Column            Non-Null Count   Dtype         
---  ------            --------------   -----         
 0   Date              112036 non-null  datetime64[ns]
 1   Day               112036 non-null  int64         
 2   Month             112036 non-null  object        
 3   Year              112036 non-null  int64         
 4   Customer_Age      112036 non-null  int64         
 5   Age_Group         112036 non-null  object        
 6   Customer_Gender   112036 non-null  object        
 7   Country           112036 non-null  object        
 8   State             112036 non-null  object        
 9   Product_Category  112036 non-null  object        
 10  Sub_Category      112036 non-null  object        
 11  Product           112036 non-null  object        
 12  Order_Quantity    112036 non-null  int64         
 13  Unit_Cost         112036 non-null  int64         
 14  Unit_Price        112036 non-null  int64         
 15  Profit            112036 non-null  int64         
 16  Cost              112036 non-null  int64         
 17  Revenue           112036 non-null  int64         
dtypes: datetime64[ns](1), int64(9), object(8)
memory usage: 16.2+ MB
<class 'NoneType'>
```

The provided code successfully converted the data type of the Date column from "object" to "datetime".

3] Checking the null records in each column

CODE:

```
null_counts = df_new.isnull().sum()
result = pd.DataFrame({'Column': null_counts.index, 'Null Count': null_counts.values})
print(result)
```

OUTPUT:

	Column	Null Count
0	Date	0
1	Day	0
2	Month	0
3	Year	0
4	Customer_Age	0
5	Age_Group	0
6	Customer_Gender	0
7	Country	0
8	State	0
9	Product_Category	0
10	Sub_Category	0
11	Product	0
12	Order_Quantity	0
13	Unit_Cost	0
14	Unit_Price	0
15	Profit	0
16	Cost	0
17	Revenue	0

The output reveals that the dataset does not contain any null values.

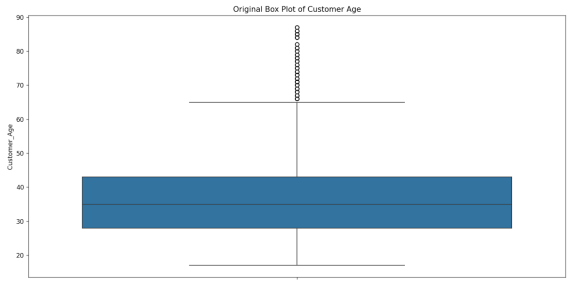
4] Outlier detection and removal

A] Customer Age - Outlier Detection

CODE:

```
sn.boxplot(y='Customer_Age', data=df_new)
plt.title(f'Original Box Plot of Customer Age')
plt.show()
```

OUTPUT:

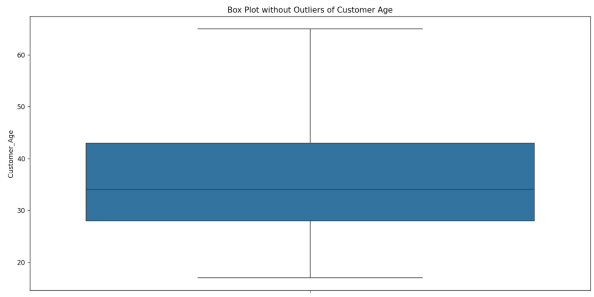


Customer Age - Outlier Removal

CODE:

```
df1=df_new[df_new['Customer_Age']<66]
sn.boxplot(y='Customer_Age', data=df1)
plt.title(f'Box Plot without Outliers of Customer Age')
plt.show()
df_new = df_new[df_new['Customer_Age'] < 66]
```

OUTPUT:

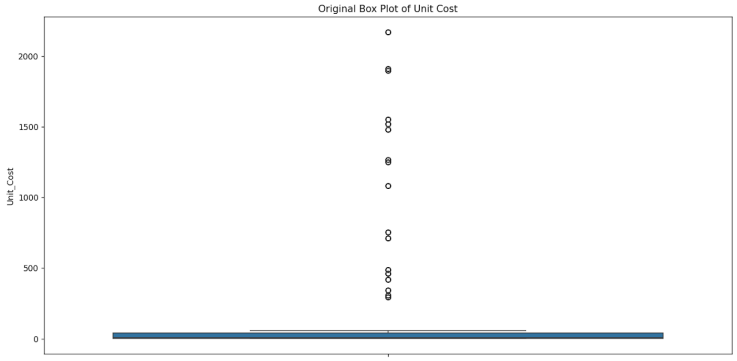


B] Unit Cost - Outlier Detection

CODE:

```
sn.boxplot(y='Unit_Cost', data=df_new)
plt.title(f'Original Box Plot of Unit Cost')
plt.show()
```

OUTPUT:

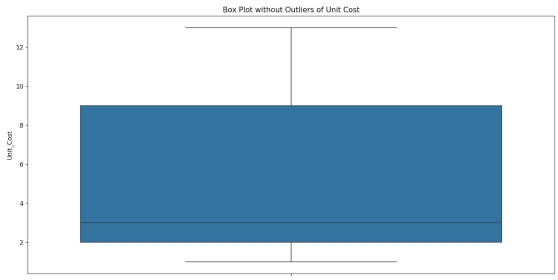


Unit Cost - Outlier Removal

CODE:

```
df2=df_new[df_new['Unit_Cost']<21]
sn.boxplot(y='Unit_Cost', data=df2)
plt.title(f'Box Plot without Outliers of Unit Cost')
plt.show()
df_new = df_new[df_new['Unit_Cost'] < 21]
```

OUTPUT:

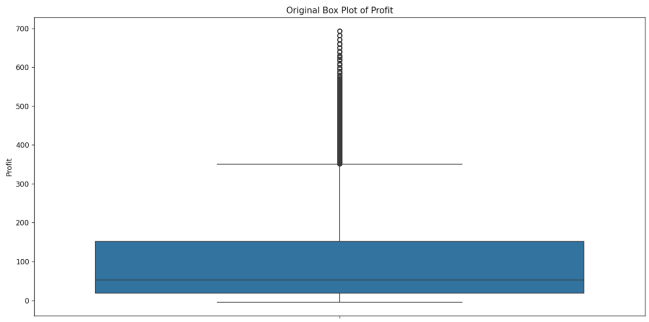


C] Profit - Outlier Detection

CODE:

```
sn.boxplot(y='Profit', data=df_new)
plt.title(f'Original Box Plot of Profit')
plt.show()
```

OUTPUT:

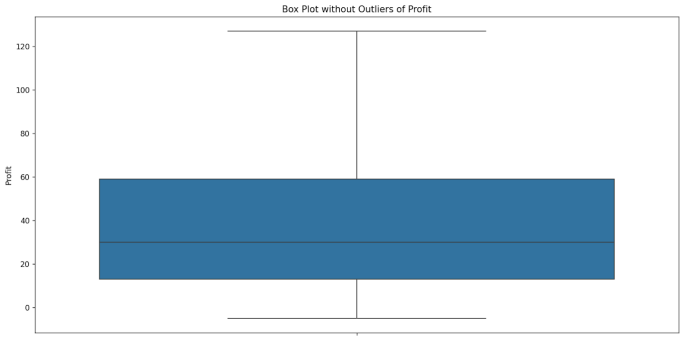


Profit - Outlier Removal

CODE:

```
df3=df_new[df_new['Profit']<128]
sn.boxplot(y='Profit', data=df3)
plt.title(f'Box Plot without Outliers of Profit')
plt.show()
df_new = df_new[df_new['Profit'] < 128]
```

OUTPUT:

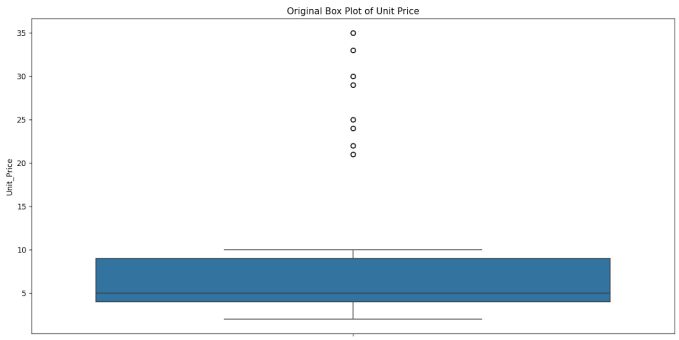


D] Unit Price - Outlier Detection

CODE:

```
sn.boxplot(y='Unit_Price', data=df_new)
plt.title(f'Original Box Plot of Unit Price')
plt.show()
```

OUTPUT:

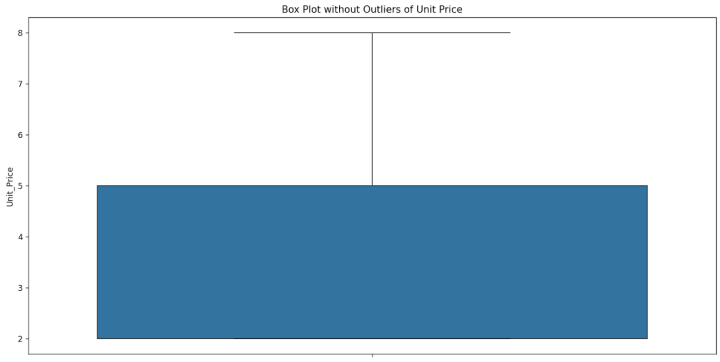


Unit Price - Outlier Removal

CODE:

```
df4=df_new[df_new['Unit_Price']<9]
sn.boxplot(y='Unit_Price', data=df4)
plt.title(f'Box Plot without Outliers of Unit Price')
plt.show()
df_new = df_new[df_new['Unit_Price'] < 9]
```

OUTPUT:

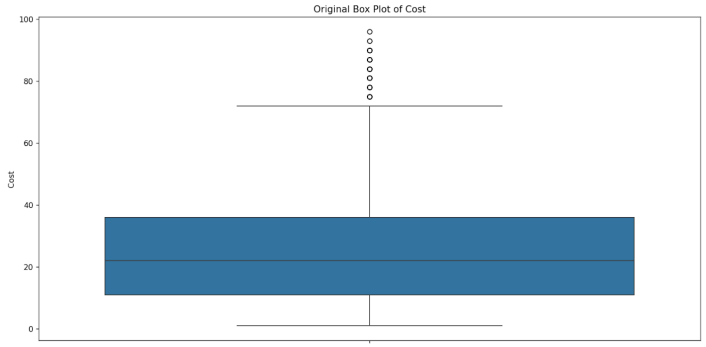


E] Cost - Outlier Detection

CODE:

```
sn.boxplot(y='Cost', data=df_new)
plt.title(f'Original Box Plot of Cost')
plt.show()
```

OUTPUT:

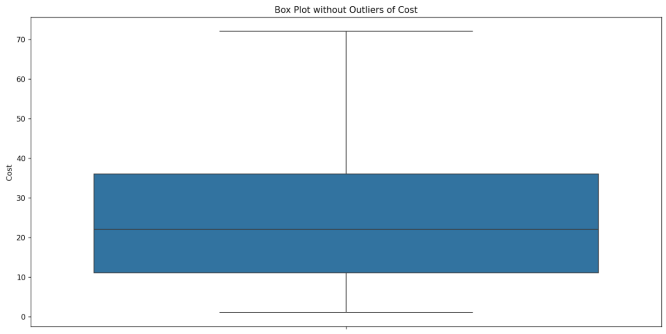


Cost - Outlier Removal

CODE:

```
df5=df_new[df_new['Cost']<75]
sn.boxplot(y='Cost', data=df5)
plt.title(f'Box Plot without Outliers of Cost')
plt.show()
df_new = df_new[df_new['Cost'] < 75]
```

OUTPUT:

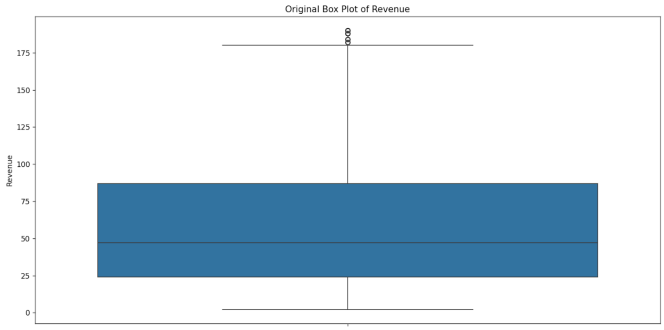


F] Revenue - Outlier Detection

CODE:

```
sn.boxplot(y='Revenue', data=df_new)
plt.title(f'Original Box Plot of Revenue')
plt.show()
```

OUTPUT:

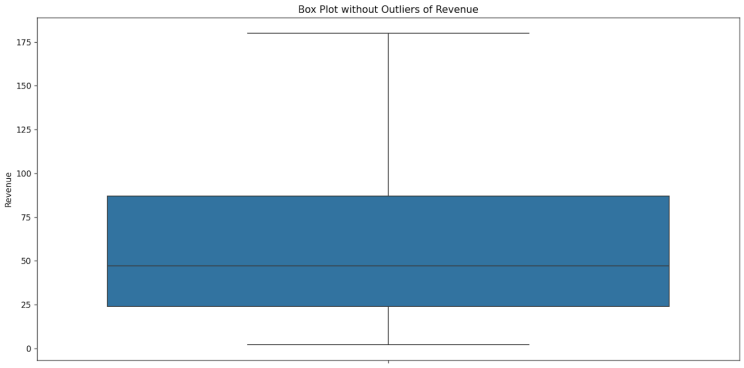


Revenue - Outlier Removal

CODE:

```
df6=df_new[df_new['Revenue']<182]
sn.boxplot(y='Revenue', data=df6)
plt.title(f'Box Plot without Outliers of Revenue')
plt.show()
df_new = df_new[df_new['Revenue'] < 182]
```

OUTPUT:



After removing all the outliers, the number of records are reduced.

Data after outlier removal

CODE:

```
print(df_new.info())
```

OUTPUT:

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_new['Date']=pd.to_datetime(df_new['Date'])
<class 'pandas.core.frame.DataFrame'>
Index: 36555 entries, 1056 to 108373
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype         
---  ------            --------------  -----         
 0   Date              36555 non-null  datetime64[ns]
 1   Day               36555 non-null  int64         
 2   Month             36555 non-null  object        
 3   Year              36555 non-null  int64         
 4   Customer_Age      36555 non-null  int64         
 5   Age_Group         36555 non-null  object        
 6   Customer_Gender   36555 non-null  object        
 7   Country           36555 non-null  object        
 8   State             36555 non-null  object        
 9   Product_Category  36555 non-null  object        
 10  Sub_Category      36555 non-null  object        
 11  Product           36555 non-null  object        
 12  Order_Quantity    36555 non-null  int64         
 13  Unit_Cost         36555 non-null  int64         
 14  Unit_Price        36555 non-null  int64         
 15  Profit            36555 non-null  int64         
 16  Cost              36555 non-null  int64         
 17  Revenue           36555 non-null  int64         
dtypes: datetime64[ns](1), int64(9), object(8)
memory usage: 5.3+ MB
None
```

The total number of data entries is reduced to 36555.

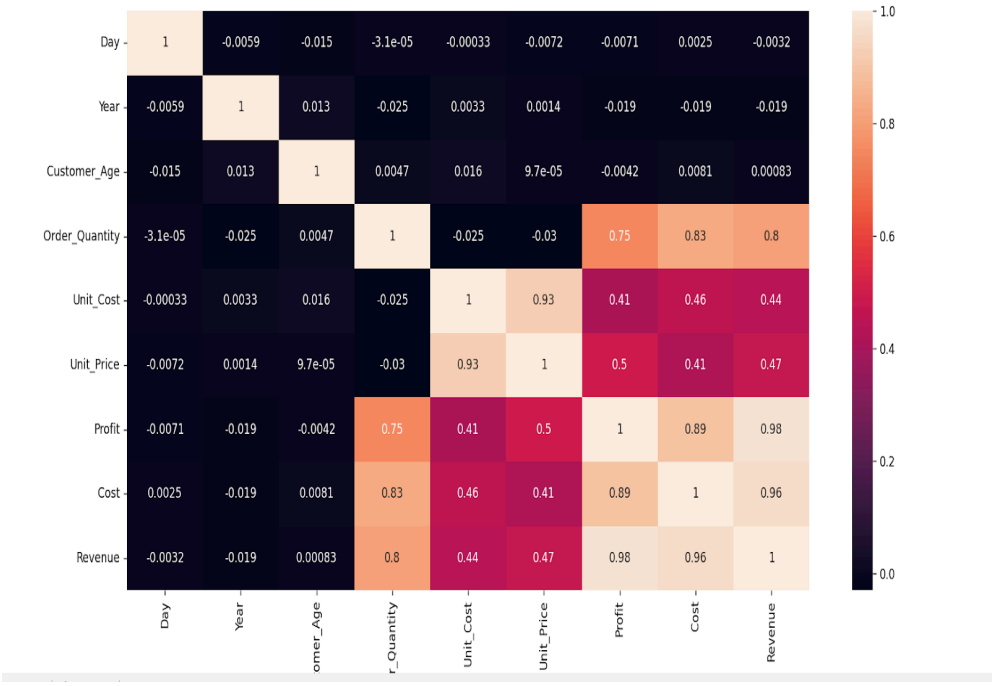
Data Mining:

1] Correlation Matrix

CODE:

```
numeric_columns = df_new.select_dtypes(include=['number']).columns
dataplot = sn.heatmap(df_new[numeric_columns].corr(), annot=True)
plt.show()
```

OUTPUT:



The correlation matrix indicates that the metrics Profit (0.75), Cost (0.83), and Revenue (0.8) exhibit strong positive correlations with Order Quantity. This suggests that as Order Quantity increases, there is a corresponding increase in these three metrics.

2] Profit Prediction using Linear Regression

CODE:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
desired_width = 400
pd.set_option('display.width', desired_width)
pd.set_option('display.max_columns', 20)
X = df_new[['Order_Quantity', 'Unit_Cost', 'Unit_Price', 'Cost', 'Revenue']]
y = df_new.Profit
lg = LinearRegression()
X_train, X_test, y_train, y_test = train_test_split( *arrays: X,y)
lg.fit(X_train, y_train)
print(lg.score(X_test, y_test))
y_prime = lg.predict(X_test)
print(y_prime, np.array(y_test))
print('MAE = ', mean_absolute_error(y_test, y_prime))
print('MSE =', mean_squared_error(y_test, y_prime))
print('RMSE = ', np.sqrt(mean_squared_error(y_test, y_prime)))
```

OUTPUT:

The Linear Regression model achieves a perfect accuracy of 100%, indicating its suitability for predicting Profit using the following metrics: Order Quantity, Unit Cost, Unit Price, Cost, and Revenue.

Example of Prediction using linear regression.

CODE:

```
Order_Quantity_1=20
Unit_Cost_1=59
Unit_Price_1=170
Cost_1=Unit_Cost_1*Order_Quantity_1
Revenue_1=2500
new_data=[[Order_Quantity_1,Unit_Cost_1,Unit_Price_1,Cost_1,Revenue_1]]
profit_value = lg.predict(new_data)
print("Predicted profit value in dollars:", profit_value)
```

OUTPUT:

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_new['Date']=pd.to_datetime(df_new['Date'])
Predicted profit value in dollars: [1320.]
C:\Users\ishan\anaconda3\Lib\site-packages\sklearn\base.py:439: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
```

When the unit cost is $59 and the unit price is $170, resulting in a cost of $10,030 (calculated as the product of unit cost and unit price), and a revenue of $2,500, the profit obtained is $1,320.

3] Profit Class Prediction Using Logistic Regression

CODE:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
desired_width = 400
pd.set_option('display.width', desired_width)
pd.set_option('display.max_columns', 20)
X = df_new[['Order_Quantity', 'Unit_Cost', 'Unit_Price', 'Cost', 'Revenue']]
y = df_new.Profit
df_new['Profit_Class'] = df_new.Profit.apply(lambda x: 'low' if x < 11 else 'high')
y2 = df_new.Profit_Class
log = LogisticRegression()
X_train, X_test, y2_train, y2_test = train_test_split( *arrays: X,y2)
print(X_train.shape, y2_train.shape, X_test.shape, y2_test.shape)
print(log.fit(X_train, y2_train))
print(log.score(X_test, y2_test))
y2_pred = log.predict(X_test)
print(y2_pred, np.array(y2_test))
print(confusion_matrix(y2_test, y2_pred))
```

OUTPUT:

```
LogisticRegression()
1.0
['high' 'high' 'high' ... 'high' 'low' 'high'] ['high' 'high' 'high' ... 'high' 'low' 'high']
[[7169    0]
 [   0 1970]]

Process finished with exit code 0
```

The Logistic Prediction model has an accuracy of 100%, indicating that it is perfectly suitable for predicting the Profit using the following metrics: Order Quantity, Unit Cost, Unit Price, Cost, and Revenue.

Moreover, we can conclude the following points from the confusion matrix:

High Accuracy: The model has high accuracy because it correctly predicted a large number of instances (TN + TP) out of the total instances.

Perfect Precision and Recall: The model achieved perfect precision (no false positives) and perfect recall (no false negatives), indicating that it performed exceptionally well in both identifying positive instances and avoiding misclassification of negative instances.

Excellent Performance: Overall, the model's performance seems excellent based on these values. It effectively differentiated between the two classes without making any misclassifications.

4] K Means Clustering

CODE:

```
import pandas as pd
import seaborn as sn
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
desired_width = 320
df_kmeans = df_new[['Order_Quantity', 'Unit_Cost', 'Unit_Price', 'Profit', 'Cost', 'Revenue']]
k_groups=KMeans(n_clusters=5, random_state=0).fit(df_kmeans)
print(k_groups.labels_)
print(len(k_groups.labels_), df_kmeans.shape)
print(k_groups.cluster_centers_)
print(k_groups.cluster_centers_[0])
df_kmeans['cluster']=k_groups.labels_
print(df_kmeans.head(3))
print(df_kmeans.groupby('cluster').mean())
```

OUTPUT:

cluster	Order_Quantity	Unit_Cost	Unit_Price	Profit	Cost	Revenue
0	26.573842	2.090010	5.321187	75.800917	54.660232	130.461149
1	15.130706	1.422445	3.449913	19.581844	18.677627	38.259471
2	16.296036	1.782928	4.674872	37.405691	27.401535	64.807225
3	5.003004	1.450085	3.555122	6.989186	6.371082	13.360268
4	21.909647	1.890824	4.991843	55.952314	40.448941	96.401255

Silhouette Coefficient

CODE:

```
from sklearn.metrics import silhouette_score
df_silhouette=df_kmeans.drop('cluster',axis=1)
for i in range (3,10):
    k_groups=KMeans(n_clusters=i).fit(df_silhouette)
    labels=k_groups.labels_
    print('K Groups = ', i, 'Silhouette Coefficient = ', silhouette_score(df_silhouette, labels))
```

OUTPUT:

```
K Groups =  3 Silhouette Coefficient =  0.5202140765693988
K Groups =  4 Silhouette Coefficient =  0.4916442033636758
K Groups =  5 Silhouette Coefficient =  0.44827756822140685
K Groups =  6 Silhouette Coefficient =  0.4318731616242389
K Groups =  7 Silhouette Coefficient =  0.44088652966014585
K Groups =  8 Silhouette Coefficient =  0.4254585054309554
K Groups =  9 Silhouette Coefficient =  0.40233678407388906
```

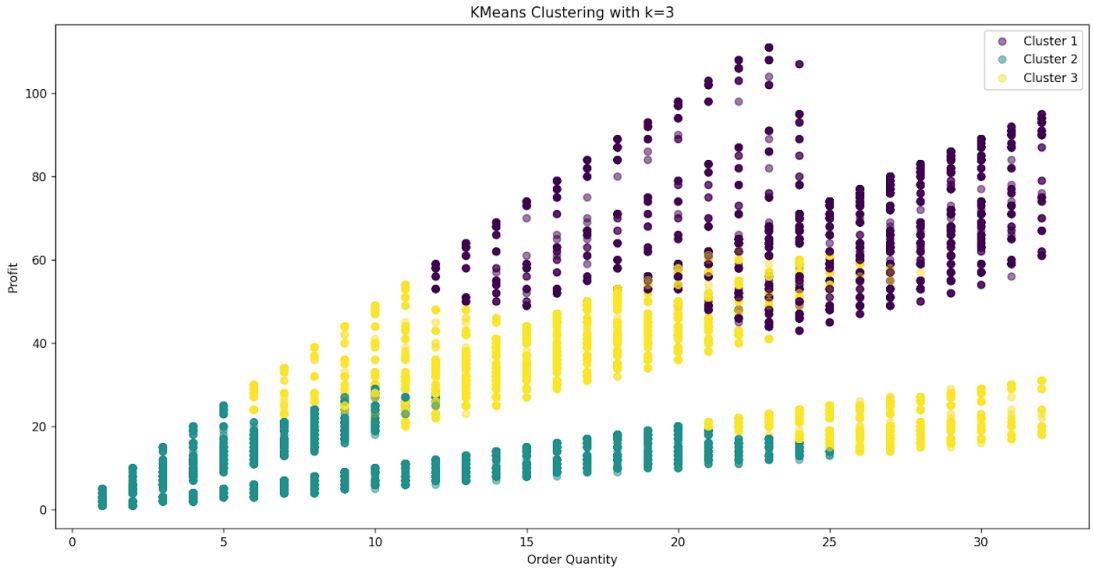
The Silhouette coefficient indicates that the clustering performance is highest when the number of clusters, k, is set to 3, with a value of 0.52.

Clustering with k=3

CODE:

```
import pandas as pd
import seaborn as sn
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
# Assuming df_new is your dataframe containing the data
df_kmeans = df_new[['Order_Quantity', 'Unit_Cost', 'Unit_Price', 'Profit', 'Cost', 'Revenue']]
# Perform KMeans clustering with k=3
k_groups = KMeans(n_clusters=3, random_state=0).fit(df_kmeans)
# Add cluster labels to the dataframe
df_kmeans['cluster'] = k_groups.labels_ + 1  # Adjusting cluster labels to start from 1
# Calculate silhouette score
silhouette_avg = silhouette_score(df_kmeans.drop('cluster', axis=1), k_groups.labels_)
print("For n_clusters =", 3, "The average silhouette_score is :", silhouette_avg)
# Visualize the clusters
plt.figure(figsize=(10, 6))
scatter = plt.scatter(df_kmeans['Order_Quantity'], df_kmeans['Profit'], c=df_kmeans['cluster'], cmap='viridis', alpha=0.5)
plt.xlabel('Order Quantity')
plt.ylabel('Profit')
plt.title('KMeans Clustering with k=3')
# Customizing legend
legend1 = plt.legend( *args: *scatter.legend_elements(), title="Clusters")
#plt.gca().add_artist(legend1)
# Customizing cluster legend labels
legend_labels = ['Cluster 1', 'Cluster 2', 'Cluster 3']
plt.legend(handles=scatter.legend_elements()[0], labels=legend_labels)
plt.show()
```

OUTPUT:



5] SVR, SVC, Prediction check (Support Vector Machine):

CODE:

```
import pandas as pd
from sklearn.svm import SVC, SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
desired_width = 400
pd.set_option('display.width', desired_width)
pd.set_option('display.max_columns', 20)
df_1=df_new
X = df_1[['Order_Quantity', 'Unit_Cost', 'Unit_Price', 'Cost', 'Revenue']]
X1 = X
y = df_1.Profit
df_1['profit_class'] = df_1.Profit.apply(lambda x: 'low' if x < 11 else 'high')
y2 = df_1.profit_class
X_train, X_test, y_train, y_test = train_test_split( *arrays: X,y)
X1_train, X1_test, y2_train, y2_test = train_test_split( *arrays: X1,y2)
svr_reg = SVR()
svc_class = SVC()
svr_reg.fit(X_train, y_train)
print('svr_score = ', svr_reg.score(X_test, y_test))
svc_class.fit(X1_train, y2_train)
print('svc_score = ', svc_class.score(X1_test, y2_test))
y2_pred = svc_class.predict(X_test)
print(confusion_matrix(y2_test, y2_pred))
```

OUTPUT:

```
svr_score =  0.9998533334244853
svc_score =  0.9914651493598862
[[5595 1562]
 [1562  420]]

Process finished with exit code 0
```

SVR Score (Support Vector Machine Regression): The SVR model achieved a high score of 0.9998, indicating strong performance in predicting continuous values. This high score suggests that the SVR model is fitting the data very well, with minimal error.

SVC Score (Support Vector Machine Classification): The SVC model achieved a score of 0.9914, indicating strong performance in classifying the data into distinct categories. This high score suggests that the SVC model is effective in distinguishing between the classes in the dataset.

Confusion Matrix: The confusion matrix provides a more detailed view of the performance of the SVC model. It shows that out of 8367 instances:

5595 instances were correctly classified as belonging to the first class.
420 instances were correctly classified as belonging to the second class.
1562 instances were incorrectly classified as belonging to the first class, but actually belong to the second class.
1562 instances were incorrectly classified as belonging to the second class, but actually belong to the first class.
Based on these results:

The SVR model appears to be highly accurate in predicting continuous values, achieving a near-perfect score.
The SVC model also performs well, with a high classification score. However, there seems to be some confusion between the classes, as indicated by the confusion matrix.

Overall, both models demonstrate strong performance, but further analysis may be needed to address misclassifications and improve the SVC model's performance, especially in correctly identifying instances from both classes.

6] Principal Component Analysis (PCA)

CODE:

```
X = df_new[['Order_Quantity','Unit_Cost','Unit_Price','Cost','Revenue']]
X1 = X
pca = PCA(4)
X_transformed = pca.fit_transform(X)
y = df_new.Profit
y1 = df_new.Profit
lg = LinearRegression()
X_train, X_test, y_train, y_test = train_test_split( *arrays: X_transformed, y)
X1_train, X1_test, y1_train, y1_test = train_test_split( *arrays: X1, y1)
print(X_transformed.shape)
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
lg.fit(X_train, y_train)
print('PCA = ', lg.score(X_test, y_test))
lg.fit(X1_train,y1_train)
print('non-PCA = ', lg.score(X1_test, y1_test))
```

OUTPUT:

```
df_new['Date'] = pd.to_datetime(df_new['Date'])
(36555, 4)
(27416, 4) (27416,) (9139, 4) (9139,)
PCA =  0.9999999334034099
non-PCA =  1.0

Process finished with exit code 0
```

PCA (Principal Component Analysis): The score obtained using features transformed by PCA is approximately 0.9999999334034099.

Non-PCA: The score obtained using the original features (without PCA transformation) is 1.0.

Comparing these scores, it appears that the linear regression model performed slightly better (with a score of 1.0) when trained on the original features without PCA transformation compared to when trained on the features transformed by PCA.

Here's what you can conclude from this:

Effectiveness of PCA: PCA is a dimensionality reduction technique used to reduce the number of features while preserving most of the variance in the data. The fact that the score obtained with PCA is close to 1 suggests that PCA was effective in retaining much of the information in the original features.

Model Performance: Despite the effectiveness of PCA, the linear regression model performed slightly better when trained on the original features. This suggests that the additional variance retained in the original features, which was discarded during the PCA transformation, may have been beneficial for the linear regression model's predictive performance in this specific case.

Trade-off: While PCA can help reduce the dimensionality of the data and potentially improve computational efficiency and mitigate multicollinearity, it may not always lead to improved model performance. The choice between using PCA-transformed features and original features depends on the specific dataset, the complexity of the problem, and the performance requirements of the model.

7] K Nearest Neighbors – Regression and Classification

CODE:

```
X = df_new[['Order_Quantity','Unit_Cost','Unit_Price','Cost','Revenue']]
X1 = X
y = df_new.Profit
df_new['profit_class'] = df_new.Profit.apply(lambda x: 'low' if x < 11 else 'high')
y2 = df_new.profit_class
X_train, X_test, y_train, y_test = train_test_split( *arrays: X,y)
X1_train, X1_test, y2_train, y2_test = train_test_split( *arrays: X1,y2)
knn_reg = KNeighborsRegressor()
knn_class = KNeighborsClassifier()
knn_reg.fit(X_train, y_train)
print('knn_reg score = ', knn_reg.score(X_test, y_test))
knn_class.fit(X1_train, y2_train)
print('knn_class score = ', knn_class.score(X1_test, y2_test))
y2_pred = knn_class.predict(X_test)
print(confusion_matrix(y2_test, y2_pred))
```

OUTPUT:

```
knn_reg score =  0.9999562759770516
knn_class score =  0.9996717365138418
[[5529 1587]
 [1578  445]]

Process finished with exit code 0
```

KNN Regression Score: The score obtained for KNN regression is approximately 0.9999562759770516. This indicates that the KNN regression model performed very well, with a high degree of accuracy in predicting the target variable.

KNN Classification Score: The score obtained for KNN classification is approximately 0.9996717365138418. Similarly, this score indicates that the KNN classification model performed exceptionally well, with a high accuracy rate in classifying the data points into their respective classes.

Confusion Matrix: The confusion matrix provides a detailed breakdown of the model's performance by showing the counts of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) predictions. From the confusion matrix:

True Positives (TP): 445
True Negatives (TN): 5529
False Positives (FP): 1587
False Negatives (FN): 1578

The high count of true positives and true negatives compared to false positives and false negatives indicates that the model made accurate predictions for both classes.

Data Visualization

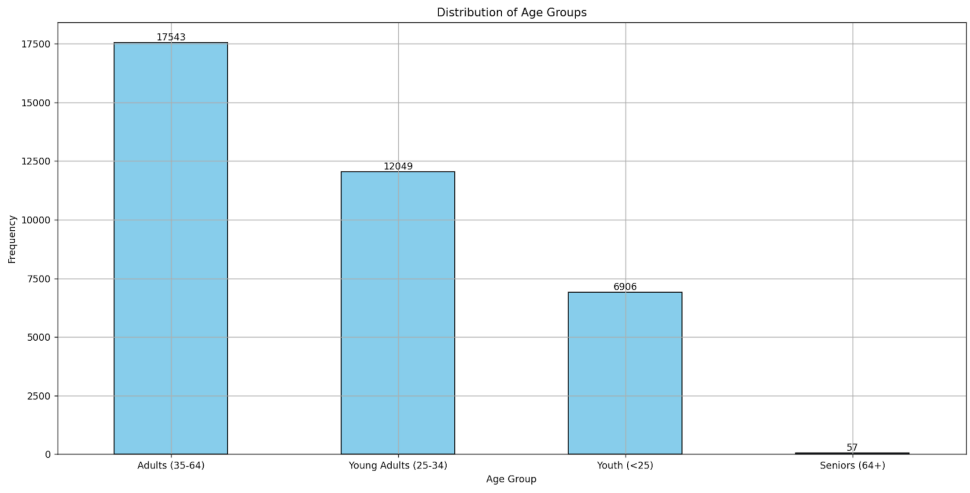
Managerial Questions that can be solved using data visualization.

1] What is the distribution of the customer age group?

CODE:

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

plt.figure(figsize=(8,6))
age_group_counts=df_new['Age_Group'].value_counts()
age_group_counts.plot(kind='bar', color='skyblue', edgecolor='black')
plt.title('Distribution of Age Groups')
plt.xlabel('Age Group')
plt.ylabel('Frequency')
plt.grid(True)
plt.xticks(rotation=360)

for i, count in enumerate(age_group_counts):
    plt.text(i, count+0.2, str(count), ha='center', va='bottom')
plt.tight_layout()
plt.show()
```

OUTPUT:



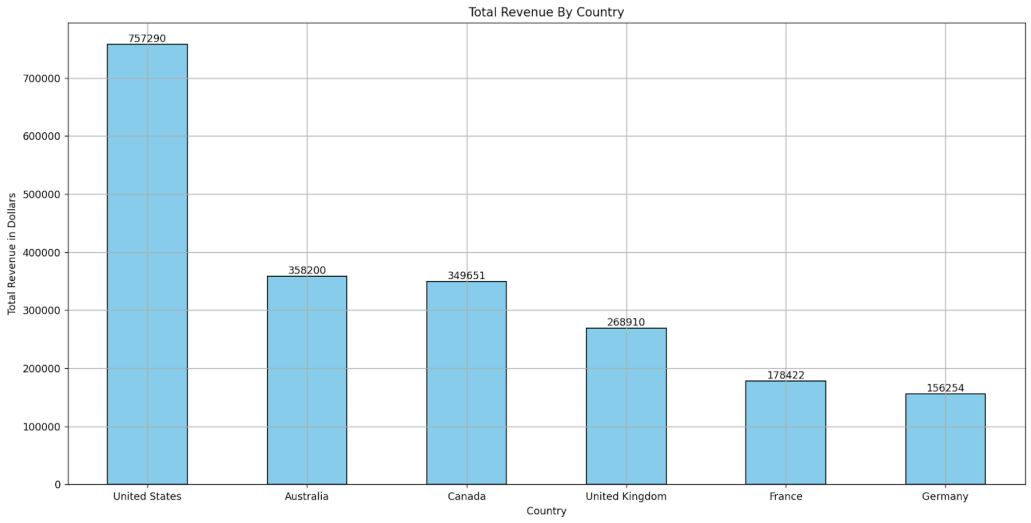
This data visualization suggests that the majority of customers fall within the adult age range (35-64), with 17,543 individuals in this group. Young adults (25-34) make up the next largest segment with 12,049 customers. The youth category (<25) has 6,906 customers, indicating a smaller but still significant portion of the customer base. Seniors (64+) represent the smallest segment, with only 57 customers.

2] How much revenue does each country contribute?

CODE:

```
plt.figure(figsize=(8,6))
Total_Revenue=df_new.groupby('Country')['Revenue'].sum().sort_values(ascending=False)
Total_Revenue.plot(kind='bar', color='skyblue', edgecolor='black')
plt.title('Total Revenue By Country')
plt.xlabel('Country')
plt.ylabel('Total Revenue in Dollars')
plt.grid(True)
plt.xticks(rotation=360)

for i, sum in enumerate(Total_Revenue):
    plt.text(i, sum + 0.2, str(sum), ha='center', va='bottom')
plt.tight_layout()
plt.show()
```

OUTPUT:



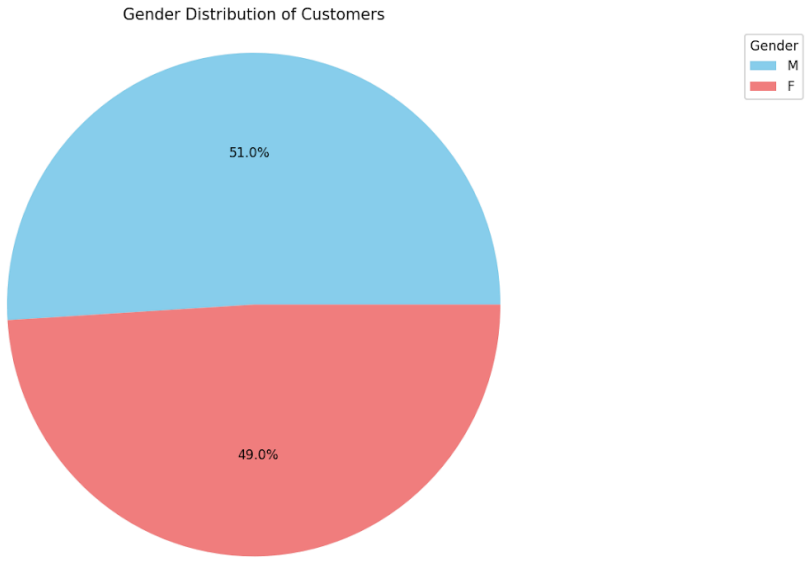
This visualization displays the total revenue generated by each country. The data reveals that the United States leads in revenue generation, followed by Australia, Canada, the United Kingdom, France, and Germany.

3] What is the gender distribution of customers?

CODE:

```
gender_counts=df_new['Customer_Gender'].value_counts()

plt.figure(figsize=(8,6))
plt.pie(gender_counts, autopct='%1.1f%%', colors=['skyblue','lightcoral'])
plt.title('Gender Distribution of Customers')
plt.axis('equal')
plt.legend(title='Gender', loc='upper right', labels=gender_counts.index)
plt.show()
```

OUTPUT:



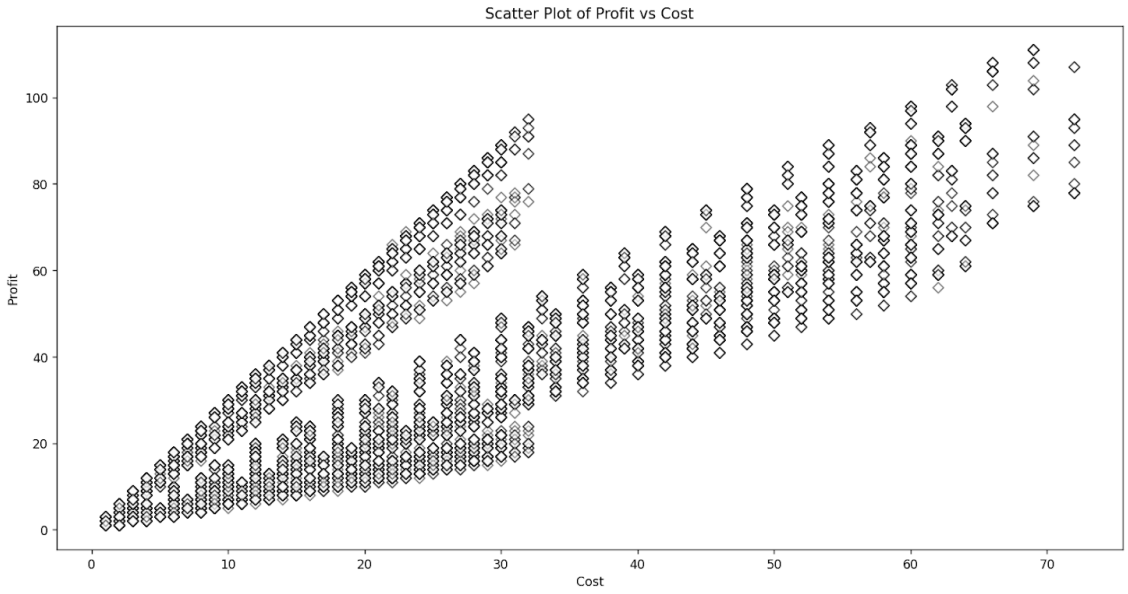
The pie chart illustrates that among all customers, 51% are male and 49% are female.

4] How does the relationship between profit and cost impact our business's financial performance and decision-making processes?

CODE:

```
x=df_new['Cost']
y=df_new['Profit']
plt.figure(figsize=(8,6))
plt.scatter(x,y,color='white',alpha=0.5, edgecolor='black', marker='D', linewidth=1)
plt.title('Scatter Plot of Profit vs Cost')
plt.xlabel('Cost')
plt.ylabel('Profit')
plt.show()
```

OUTPUT:



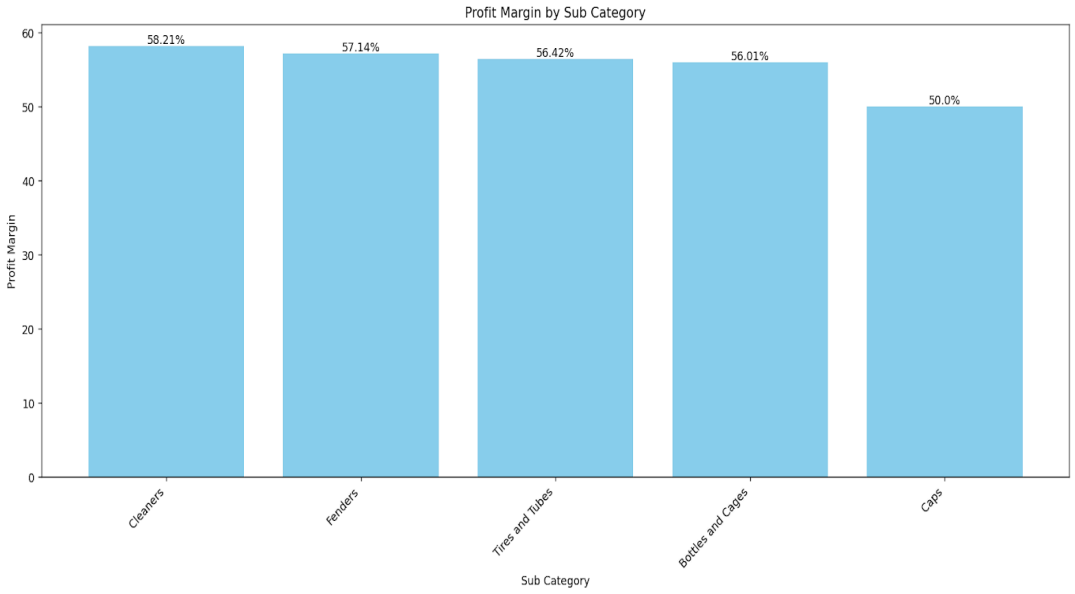
The scatter plot suggests that there is a positive correlation between cost and profit, indicating that higher costs are generally associated with higher profits. However, it's notable that the majority of customers tend to fall within a cost price range of 10-30.

5] What is the profit margin for each Sub category

CODE:

```
# Calculate total profit and revenue for each product category
category_profit_revenue = df_new.groupby('Sub_Category')[['Profit', 'Revenue']].sum()

# Calculate profit margin
category_profit_revenue['Profit_Margin'] = (category_profit_revenue['Profit'] / category_profit_revenue['Revenue'])*100

# Sort the result by profit margin in descending order
category_profit_margin_sorted = category_profit_revenue.sort_values(by='Profit_Margin', ascending=False)

# Create a bar chart
plt.figure(figsize=(10, 6))
bars = plt.bar(category_profit_margin_sorted.index, category_profit_margin_sorted['Profit_Margin'], color='skyblue')

# Add profit margin values on top of each bar
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, f'{round(yval, 2)}%', ha='center', va='bottom')

plt.title('Profit Margin by Sub Category')
plt.xlabel('Sub Category')
plt.ylabel('Profit Margin')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

OUTPUT:



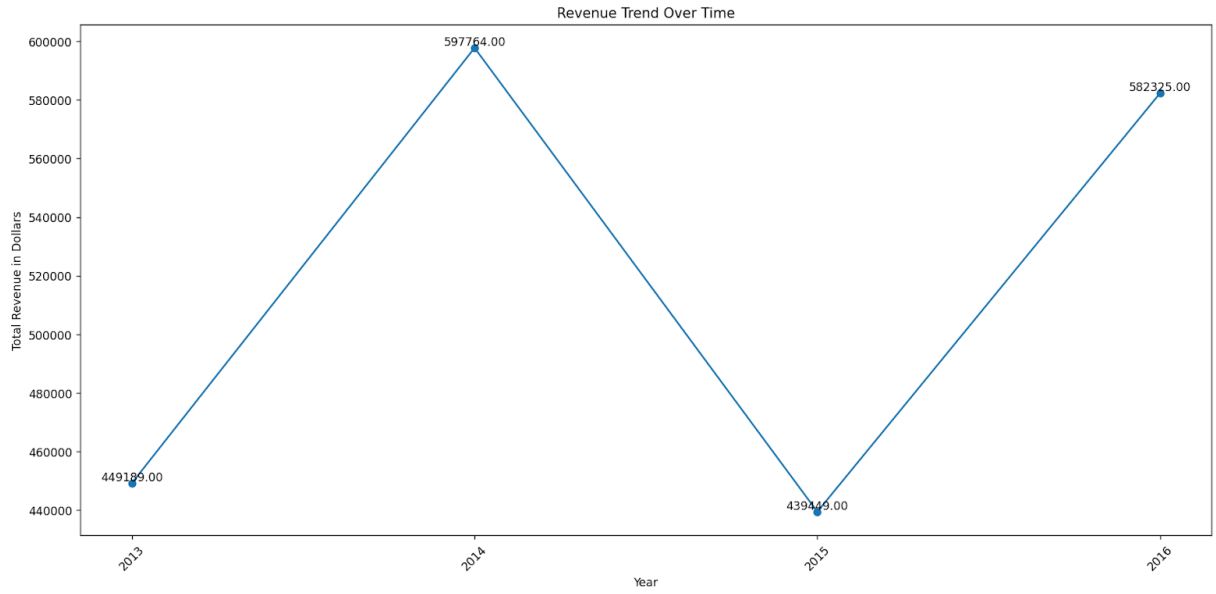
The bar plot reveals that every subcategory boasts a profit margin of at least 50%. Notably, the Cleaners category exhibits the highest profit margin at 58.21%, closely followed by Fenders at 57.14% and Tires and Tubes at 56.42%.

6] How does revenue change over time? Are there any noticeable trends or fluctuations in sales performance?

CODE:

```
df_new['Year'] = df_new['Year'].astype(str)
revenue_trend = df_new.groupby('Year')['Revenue'].sum()

# Plotting
plt.figure(figsize=(10, 6))
plt.plot( *args: revenue_trend.index, revenue_trend.values, marker='o', linestyle='-')
plt.title('Revenue Trend Over Time')
plt.xlabel('Year')
plt.ylabel('Total Revenue in Dollars')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
for i, revenue in enumerate(revenue_trend.values):
    plt.text(revenue_trend.index[i], revenue,  s: f'{revenue:.2f}', ha='center', va='bottom')
plt.show()
```

OUTPUT:



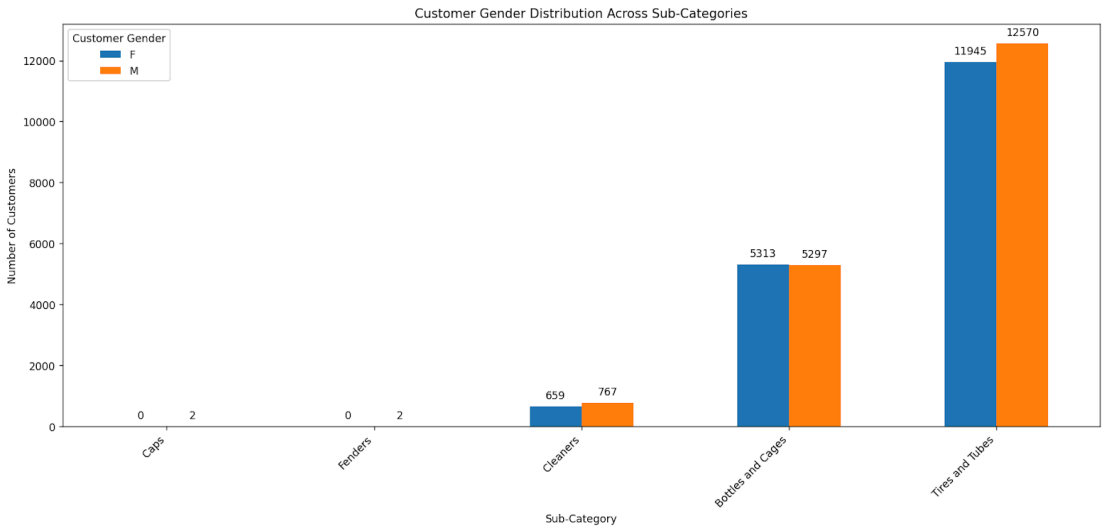
In 2013, the total revenue was $449,189.
In 2014, the total revenue increased to $597,764.
In 2015, the total revenue decreased to $439,449.
In 2016, the total revenue increased again to $582,325.
The revenue trend shows fluctuations over the years, with an initial increase from 2013 to 2014, followed by a decrease in 2015, and then another increase in 2016. This suggests that there may be seasonal or cyclical patterns influencing the revenue performance.

7] How is the customer gender distributed across different sub categories? Are there gender-specific preferences?

CODE:

```
# Group data by sub-category and customer gender and count the number of occurrences
gender_distribution = df_new.groupby(['Sub_Category', 'Customer_Gender']).size().unstack(fill_value=0)

gender_distribution['Total'] = gender_distribution.sum(axis=1)

# Sort the dataframe by the total number of customers in ascending order
gender_distribution_sorted = gender_distribution.sort_values(by='Total')

# Remove the 'Total' column
gender_distribution_sorted.drop(columns='Total', inplace=True)
# Plotting
ax=gender_distribution_sorted.plot(kind='bar', stacked=False, figsize=(10, 6))
plt.title('Customer Gender Distribution Across Sub-Categories')
plt.xlabel('Sub-Category')
plt.ylabel('Number of Customers')
plt.xticks(rotation=45, ha='right')
plt.legend(title='Customer Gender')
plt.tight_layout()

for p in ax.patches:
    ax.annotate(str(int(p.get_height())), (p.get_x() + p.get_width() / 2, p.get_height()),
                ha='center', va='bottom', xytext=(0, 5), textcoords='offset points')

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

OUTPUT:



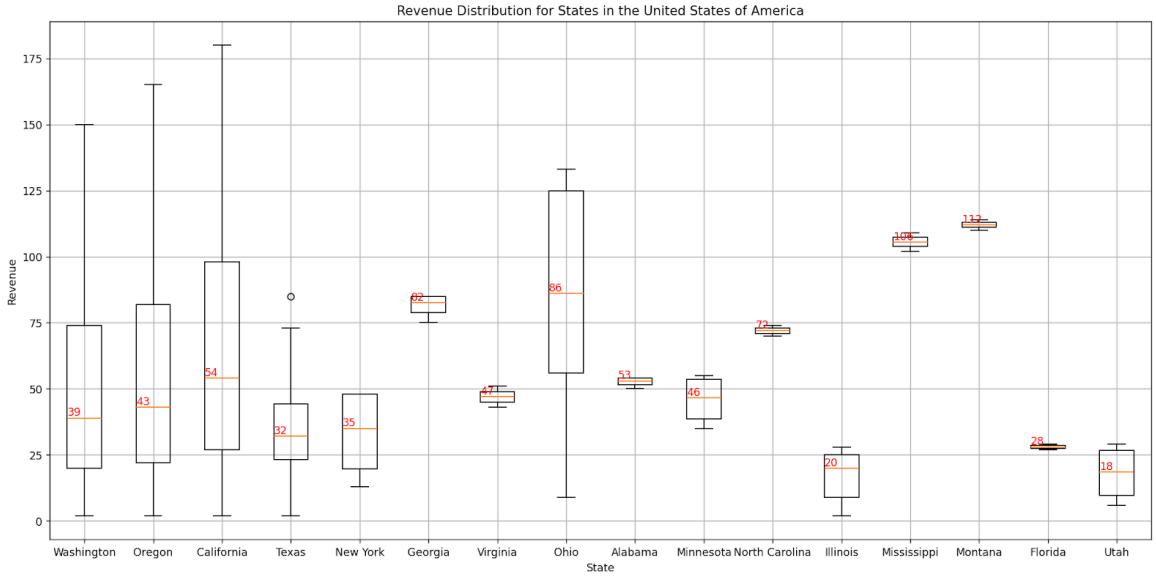
In the 'Caps' and 'Fenders' sub-categories, the number of male customers is 2 each, while there are no female customers.
In the 'Cleaners' sub-category, there are 767 male customers and 659 female customers. The number of male customers is slightly higher than the number of female customers.
In the 'Bottles and Cages' sub-category, there are 5,297 male customers and 5,313 female customers. The number of female customers is slightly higher than the number of male customers.
In the 'Tires and Tubes' sub-category, there are 12,570 male customers and 11,945 female customers. The number of male customers is significantly higher than the number of female customers.

8] What is the distribution of revenue among states in the United States of America?

CODE:

```
df_usa = df_new[df_new['Country'] == 'United States']

# Group data by state and extract revenue for each state
revenue_by_state = [df_usa[df_usa['State'] == state]['Revenue'].values for state in df_usa['State'].unique()]

# Plotting
plt.figure(figsize=(12, 8))
boxplot=plt.boxplot(revenue_by_state, labels=df_usa['State'].unique(), vert=True)
plt.title('Revenue Distribution for States in the United States of America')
plt.xlabel('State')
plt.ylabel('Revenue')
plt.grid(True)
plt.tight_layout()

for median in boxplot['medians']:
    plt.text(median.get_xdata()[0], median.get_ydata()[0], s: f'{median.get_ydata()[0]:.0f}',
             ha='left', va='bottom', color='red')
plt.show()
```

OUTPUT:



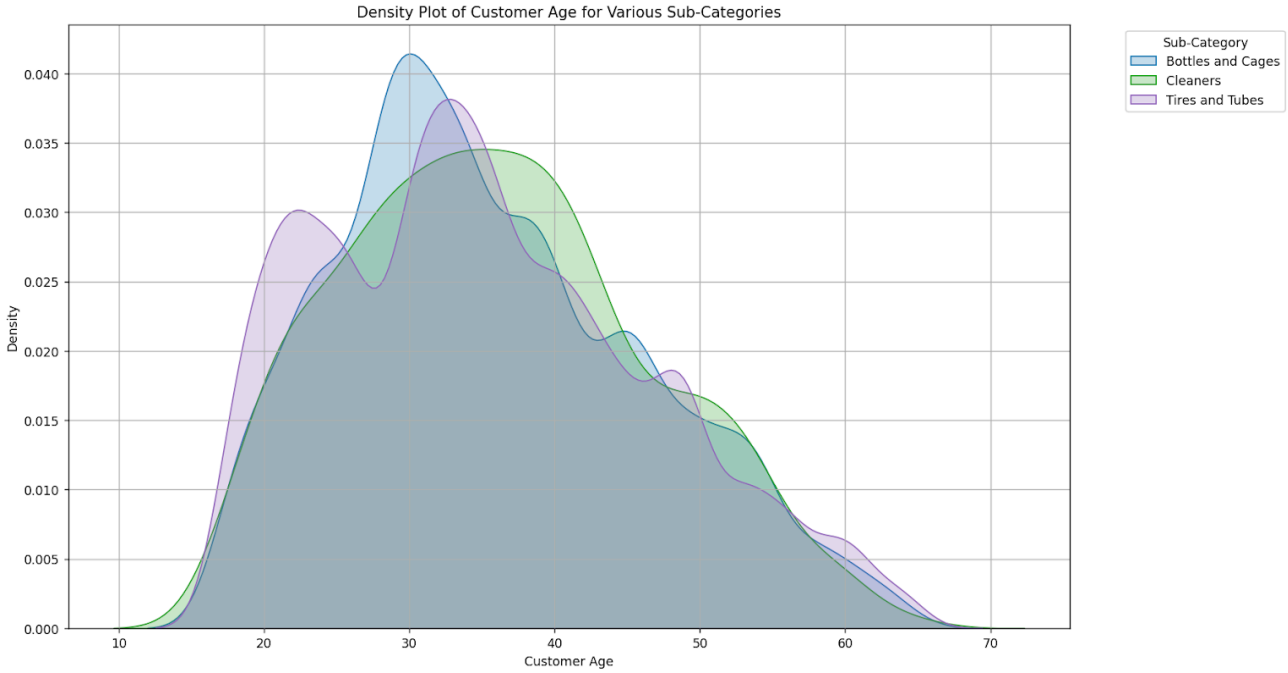
- **States like Mississippi, Montana, and North Carolina have the highest median revenue, with values of $106, $112, and $72, respectively.**
- **Other states such as Georgia, Virginia, and Alabama also have relatively high median revenues, with values of $82, $47, and $53, respectively.**
- **On the other hand, states like Utah, Illinois, and Florida have lower median revenues, with values of $18, $20, and $28, respectively.**
- **The remaining states fall within the range of median revenues between $32 and $54.**
- **From this analysis, we can observe significant variations in median revenue among different states. States with higher median revenues may indicate stronger economic activity or higher consumer spending, while states with lower median revenues may have less robust economic conditions or smaller consumer markets.**

9] What is the density distribution of customer age across various sub-categories?

CODE:

```
sub_category_groups = df_new.groupby('Sub_Category')

# Plot density plot for each sub-category
plt.figure(figsize=(12, 8))
for sub_category, data in sub_category_groups:
    sn.kdeplot(data=data, x='Customer_Age', label=sub_category, fill=True)

plt.title('Density Plot of Customer Age for Various Sub-Categories')
plt.xlabel('Customer Age')
plt.ylabel('Density')
plt.legend(title='Sub-Category', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.tight_layout()
plt.show()
```

OUTPUT:



In Bottles and Cages, the highest density of customer age is observed at 30, with a decreasing density as age increases.

In Cleaners, the peak density of customer age occurs at 35, followed by a decrease in density as age increases.

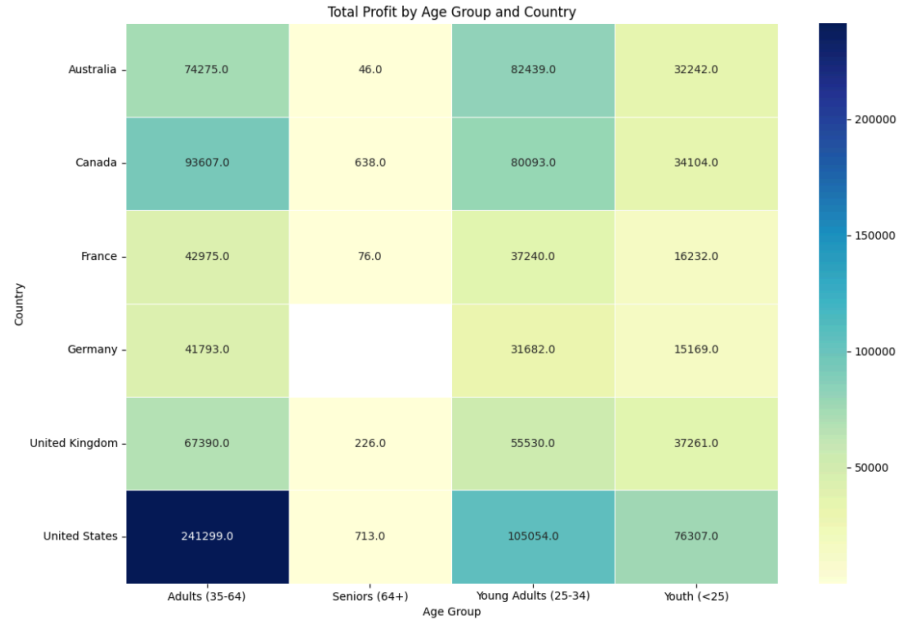
For Tires and Tubes, the highest density of customer age is found in the 35 age group, while the second-highest density is observed in the 22-23 age group.

10] What is the distribution of total profit among different age groups within each country?

CODE:

```
pivot_table = df_new.pivot_table(index='Country', columns='Age_Group', values='Profit', aggfunc='sum')

# Create a heatmap
plt.figure(figsize=(12, 8))
sn.heatmap(pivot_table, cmap='YlGnBu', annot=True, fmt=".1f", linewidths=.5)
plt.title('Total Profit by Age Group and Country')
plt.xlabel('Age Group')
plt.ylabel('Country')
plt.tight_layout()
plt.show()
```

OUTPUT:



The analysis of the heatmap reveals the following insights:

In the United States, the highest profit of $241,299 is observed in the Adults age group, while the lowest profit of $76,307 is seen in the Youth age group.
In the United Kingdom, the highest profit of $67,390 is recorded in the Adults age group, with the lowest profit of $37,261 in the Youth age group.
France shows the highest profit of $42,975 in the Adults age group, contrasting with the lowest profit of $16,232 in the Youth age group.
Germany exhibits the highest profit of $41,793 in the Adults age group, while the lowest profit of $15,169 is noted in the Youth age group.
Canada indicates the highest profit of $93,607 in the Adults age group, with the lowest profit of $34,104 observed in the Youth age group.
Australia displays the highest profit of $82,439 in the Young Adults age group, with the lowest profit of $32,242 in the Youth age group.

Summary:

The bike sales dataset comprised 113,036 records and underwent several techniques:

1] Data Wrangling:

Duplicate records were identified and removed, resulting in the elimination of approximately 1,000 duplicates and reducing the dataset to 112,036 entries. The date column, initially in object data type, was converted to datetime. Null values were checked and found to be absent. Outlier detection was conducted across numeric columns (Customer Age, Unit Cost, Profit, Unit Price, Cost, Revenue), and outliers were subsequently removed, reducing the dataset to 36,555 entries.

2] Data Mining:

A correlation matrix was generated for all numerical columns, visualized on a heatmap. Profit Prediction was executed using a Linear Regression model based on Order Quantity, Unit Cost, Unit Price, Cost, and Revenue, yielding a perfect accuracy of 100%. Logistic Regression was employed to predict Profit Class (High and Low) based on the same columns. K Means Clustering was performed with silhouette analysis, indicating the highest clustering performance at n=3 clusters (52%). Support Vector Machine Regression and Classification models were developed, yielding high scores (svr score = 0.9998, svc score = 0.9914). Principal Component Analysis (PCA) was utilized, showing a slight decrease in linear regression accuracy (99.99%) compared to the non-PCA model (100%). K Nearest Neighbors Regression and Classification were implemented, displaying high scores (knn_reg score = 0.9999, knn_class score = 0.9996).

3] Data Visualization:

Ten managerial questions were addressed using various charts. Insights on Profit, Revenue, Total Number of Customers, and Gender Distribution by Country, State, and Age Groups were obtained through bar graphs, grouped bar graphs, scatter plots, density plots, heatmaps, box plots, line charts, and pie charts.

Link of the dataset:

https://www.kaggle.com/datasets/sadiqshah/bike-sales-in-europe