Capstone Project on Customer Electronics Sales Data Analysis

Name: Mihir Milind Jade

Languages Used: Python

Library used: Numpy, Pandas, Matplotlib, Seaborn, Sckit-learn`

Overview

In today's competitive consumer electronics market, understanding consumer behavior and predicting purchasing intent are crucial for driving sales and enhancing customer satisfaction. This dataset provides a detailed view of customer demographics, product performance, and purchasing patterns, making it anexcellent resource for analysis and predictive modeling.

Objective

The objective of this analysis is to understand customer behavior and product performance in the competitive consumer electronics market. By examining demographics, product preferences, pricing and purchasing patterns, the study aims to identify key factors influencing purchase intent and satisfaction. Additionally a predictive model will be developed to forecast purchase intent, enabling businesses to make data-driven decisions, optimize marketing strategies, and improve customer engagement.

Loading the Data

```
# importing library
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv(r'consumer_electronics_sales_data.csv')
df
```

```
      ProductID ProductCategory  ProductBrand  ProductPrice
CustomerAge  \
0          5874     Smartphones  Other Brands    312.949668
18
1          5875   Smart Watches       Samsung    980.389404
35
2          5876         Tablets       Samsung   2606.718293
63
```

```
3          5877     Smartphones       Samsung    870.395450
63
4          5878         Tablets          Sony   1798.955875
57
...         ...             ...           ...          ...           .
..
8995      14869   Smart Watches       Samsung   1041.149163
36
8996      14870     Smartphones       Samsung   1485.694311
57
8997      14871      Headphones       Samsung   2887.369597
28
8998      14872         Tablets            HP   1490.453964
38
8999      14873     Smartphones          Sony   2315.583087
62

      CustomerGender  PurchaseFrequency  CustomerSatisfaction
PurchaseIntent
0                  0                  2                     1
0
1                  1                  7                     2
1
2                  0                  1                     5
1
3                  1                 10                     3
1
4                  0                 17                     3
0
...              ...                ...                   ...
...
8995               1                 16                     4
0
8996               0                  5                     1
1
8997               0                 18                     4
0
8998               0                  4                     2
1
8999               0                 15                     2
1

[9000 rows x 9 columns]
```

```
df.shape
```

```
(9000, 9)
```

Observation

- There are total 9000 rows and 9 columns in our dataset.

Overview of Dataset

ProductID: Unique identifier for the product.

ProductCategory: Category of the product (e.g, Smartphones, Tabets).

ProductBrand: Brand of the product.

ProductPrice: Price of the product.

CustomerAge: Age of the customer.

CustomerGender: Gender of the Customer (binary: 0 = female, 1 = Male).

PurchaseFrequency: Frequency of purchase by the customer.

CustomerSatisfaction: Satisfaction level of customer (scale: 1-5).

PurchaseIntent: Indicator of the intent to purchase (binary: 0 = No, 1 = Yes).

```
df.columns

# observation
    # The target column is 'PurchaseIntent'
```

```
Index(['ProductID', 'ProductCategory', 'ProductBrand', 'ProductPrice',
       'CustomerAge', 'CustomerGender', 'PurchaseFrequency',
       'CustomerSatisfaction', 'PurchaseIntent'],
      dtype='object')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9000 entries, 0 to 8999
Data columns (total 9 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   ProductID             9000 non-null   int64  
 1   ProductCategory       9000 non-null   object 
 2   ProductBrand          9000 non-null   object 
 3   ProductPrice          9000 non-null   float64
 4   CustomerAge           9000 non-null   int64  
 5   CustomerGender        9000 non-null   int64  
 6   PurchaseFrequency     9000 non-null   int64  
 7   CustomerSatisfaction  9000 non-null   int64  
 8   PurchaseIntent        9000 non-null   int64  
dtypes: float64(1), int64(6), object(2)
memory usage: 632.9+ KB
```

Observation

- There are no missing values in dataset.
- The data has 9000 entries.
- There are 7 numerical and 2 categorical column.

```
df.describe()
```

	ProductID	ProductPrice	CustomerAge	CustomerGender
count	9000.000000	9000.000000	9000.000000	9000.000000
mean	10373.500000	1527.429195	43.347000	0.508889
std	2598.220545	829.900898	15.055084	0.499949
min	5874.000000	100.376358	18.000000	0.000000
25%	8123.750000	809.165014	30.000000	0.000000
50%	10373.500000	1513.024577	43.000000	1.000000
75%	12623.250000	2244.415520	56.000000	1.000000
max	14873.000000	2999.852253	69.000000	1.000000

	PurchaseFrequency	CustomerSatisfaction	PurchaseIntent
count	9000.000000	9000.000000	9000.000000
mean	10.054667	2.996000	0.566444
std	5.461328	1.405301	0.495593
min	1.000000	1.000000	0.000000
25%	5.000000	2.000000	0.000000
50%	10.000000	3.000000	1.000000
75%	15.000000	4.000000	1.000000
max	19.000000	5.000000	1.000000

Checking Null Values

```
df.isnull().sum()
```

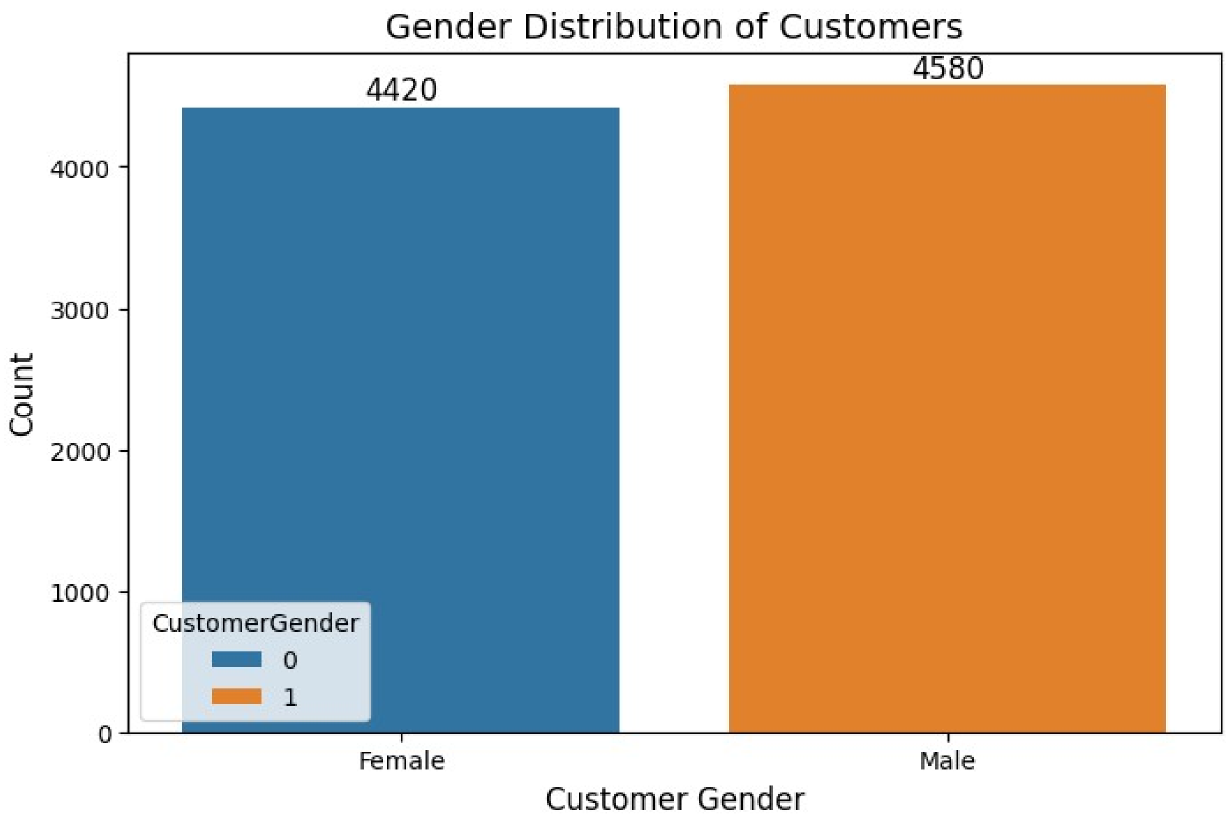
```
ProductID                0
ProductCategory          0
ProductBrand             0
ProductPrice             0
CustomerAge              0
CustomerGender           0
PurchaseFrequency        0
CustomerSatisfaction     0
PurchaseIntent           0
dtype: int64
```

Exploratory Data Analysis

Distribution of Gender

```
plt.figure(figsize=(8,5))
ax = sns.countplot(data=df,x='CustomerGender', hue='CustomerGender')
```

```
for container in ax.containers:
    ax.bar_label(container, label_type='edge', fontsize=12)
plt.title("Gender Distribution of Customers",fontsize=14)
plt.xlabel('Customer Gender',fontsize=12)
plt.ylabel('Count',fontsize=12)
plt.xticks([0,1],['Female','Male'])
plt.show()
```

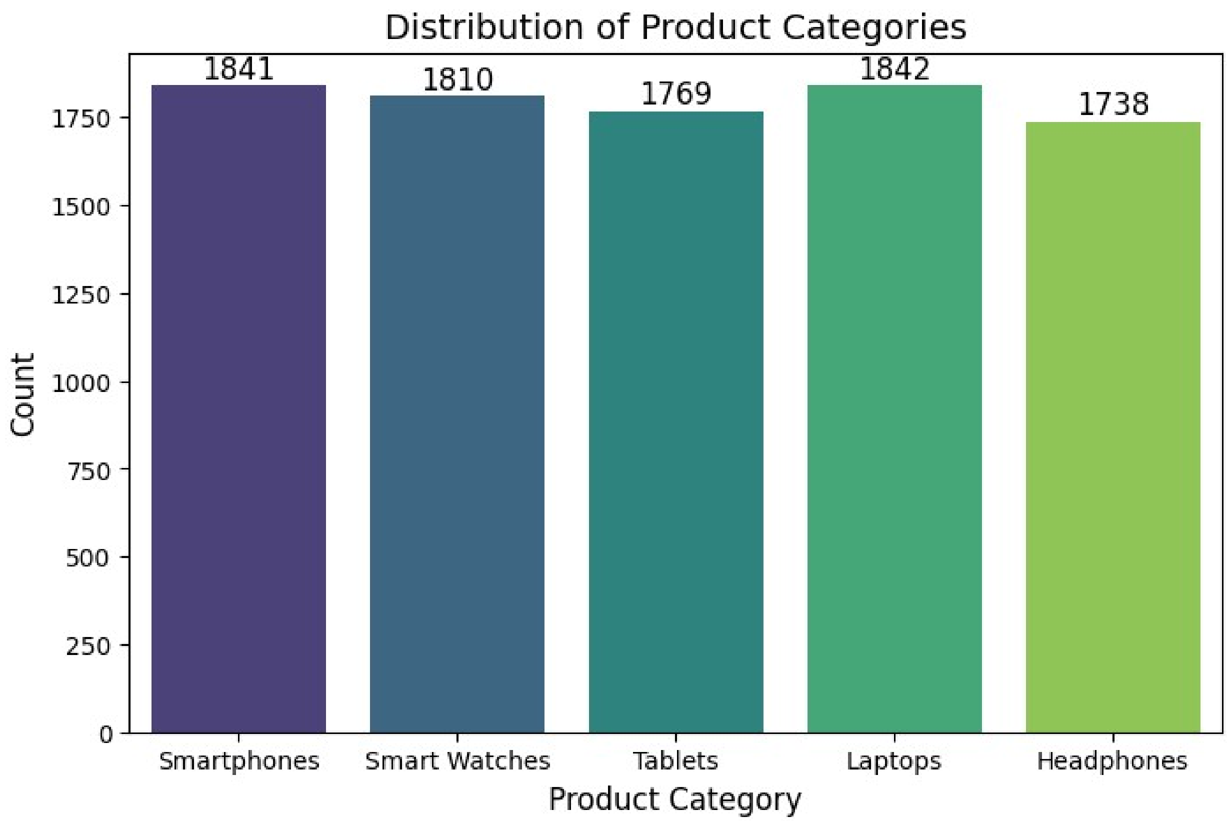


Observation

- There are total 4580 male customers and 4420 female customers.
- Male customers are more than female customers in the dataset.

Distribution of Product Category

```
plt.figure(figsize=(8,5))
ax = sns.countplot(data=df,x='ProductCategory',
hue='ProductCategory',palette='viridis')
for container in ax.containers:
    ax.bar_label(container, label_type='edge', fontsize=12)
plt.title("Distribution of Product Categories",fontsize=14)
plt.xlabel('Product Category',fontsize=12)
plt.ylabel('Count',fontsize=12)
plt.show()
```

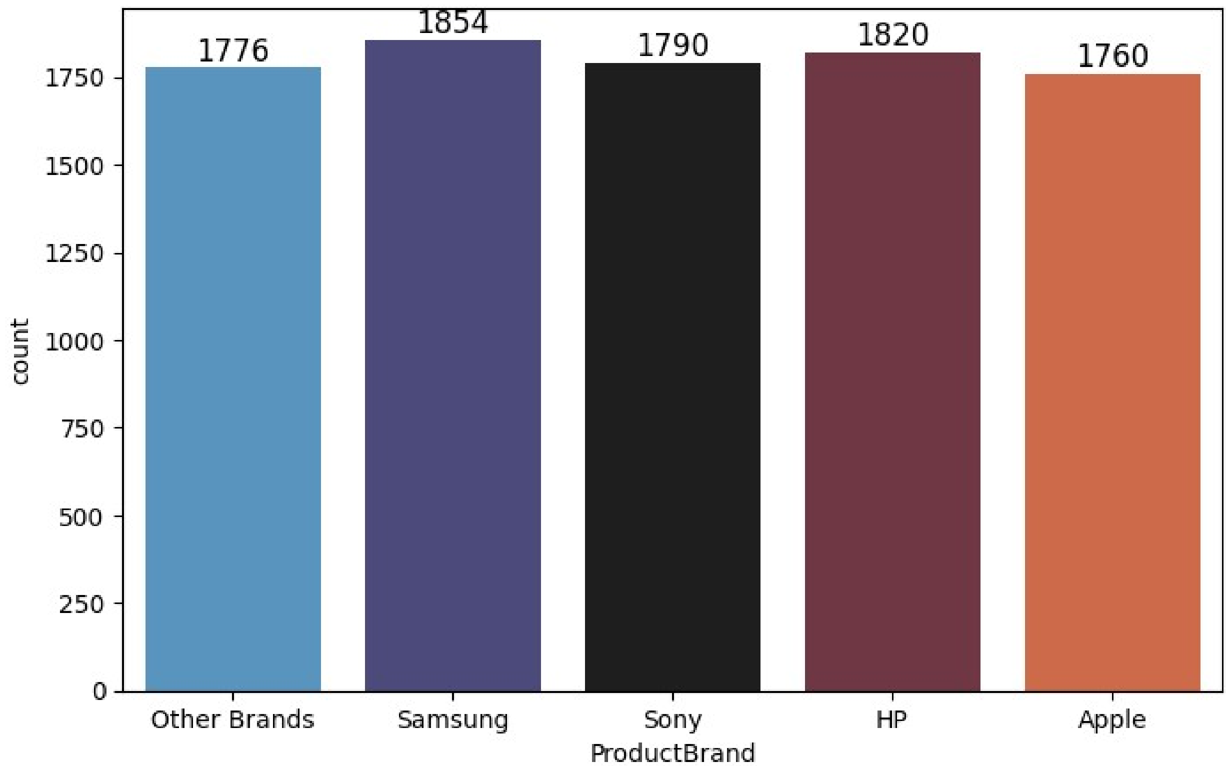


Observation

- Most customers purchase Laptops i.e 1842 and Smartphones i.e 1841.
- Headphones are the less selling product in the store.

Frequency of Product Brands

```
plt.figure(figsize=(8,5))
ax = sns.countplot(data=df,x='ProductBrand',
hue='ProductBrand',palette='icefire')
for container in ax.containers:
    ax.bar_label(container, label_type='edge', fontsize=12)
```

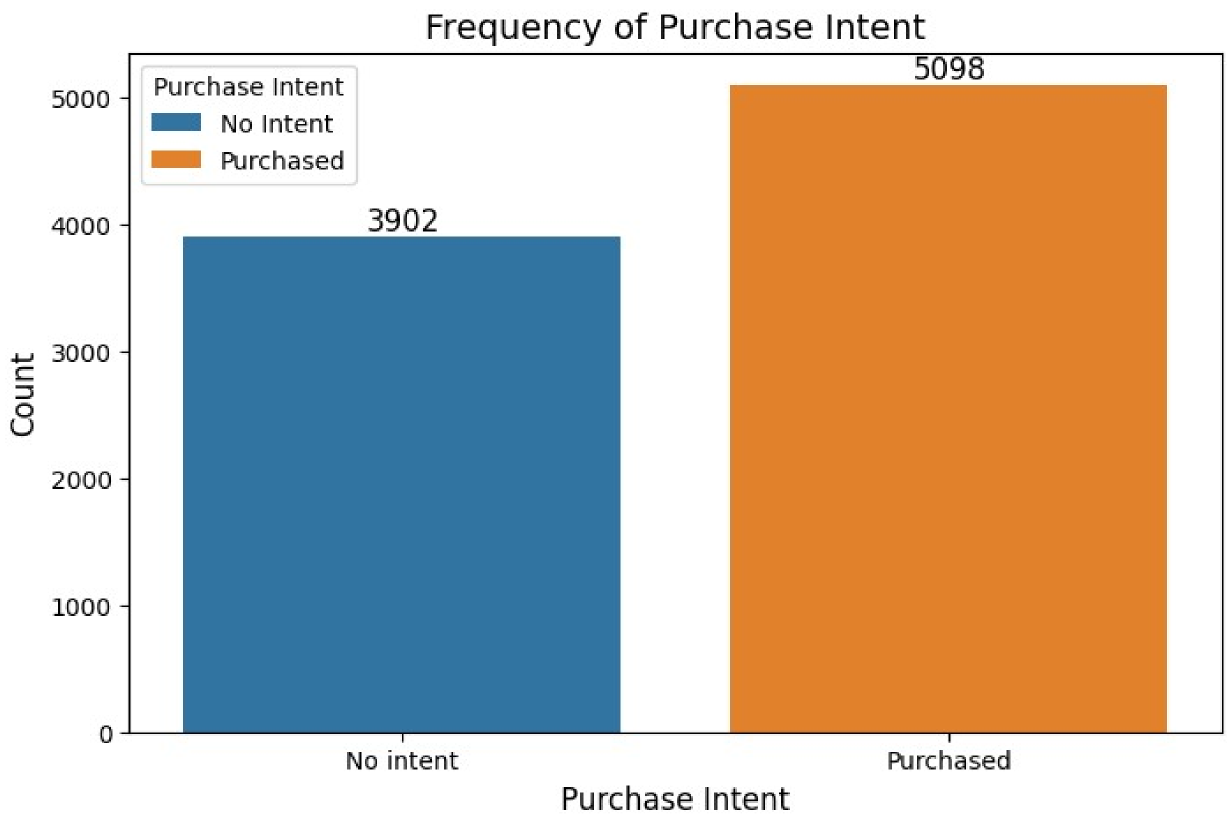


Observation

- Samsung(1854) leads the market, followed by HP(1820).
- Apple(1760) has a relatively lower presence compared to leading brand like Samsung and Hp.

Frequency of Purchase Intent

```
plt.figure(figsize=(8,5))
ax = sns.countplot(data=df,x='PurchaseIntent', hue='PurchaseIntent')
for container in ax.containers:
    ax.bar_label(container, label_type='edge', fontsize=12)
plt.xticks([0,1],['No intent','Purchased'])
plt.title("Frequency of Purchase Intent",fontsize=14)
plt.xlabel('Purchase Intent',fontsize=12)
plt.ylabel('Count',fontsize=12)
ax.legend(labels=['No Intent', 'Purchased'], title='Purchase Intent')
plt.show()
```

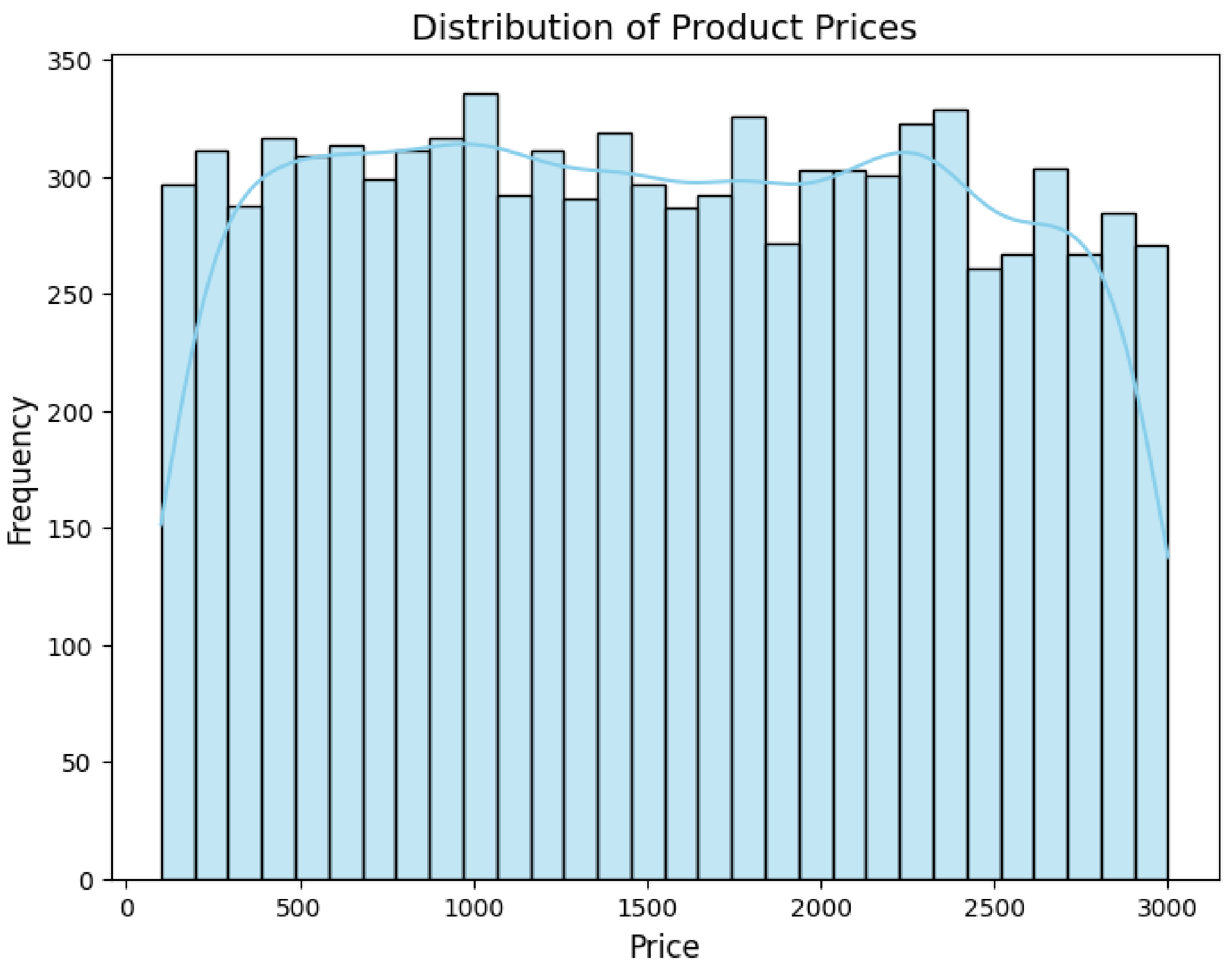


Observation

- Higher number of customers has intent to purchase with 5,098 instances compared to no intent with instances 3,092.
- The balance between purchase and non-purchase intent is relatively good, with a slight skew towards purchase intent.

Distribution of Product Prices

```
plt.figure(figsize=(8,6))
sns.histplot(df['ProductPrice'],bins=30,kde=True,color='skyblue',edgec
olor='black')
plt.title('Distribution of Product Prices',fontsize=14)
plt.xlabel('Price',fontsize=12)
plt.ylabel('Frequency',fontsize=12)
plt.show()
```



Observation

- The distribution of product prices appear to be relatively uniform with only slight fluctuations in the KDE curve.
- This indicates that the prices are spreaded evenly across the range, with no significant peaks.
- This could indicate a competitive market with similar

Distribution of Customer Satisfaction

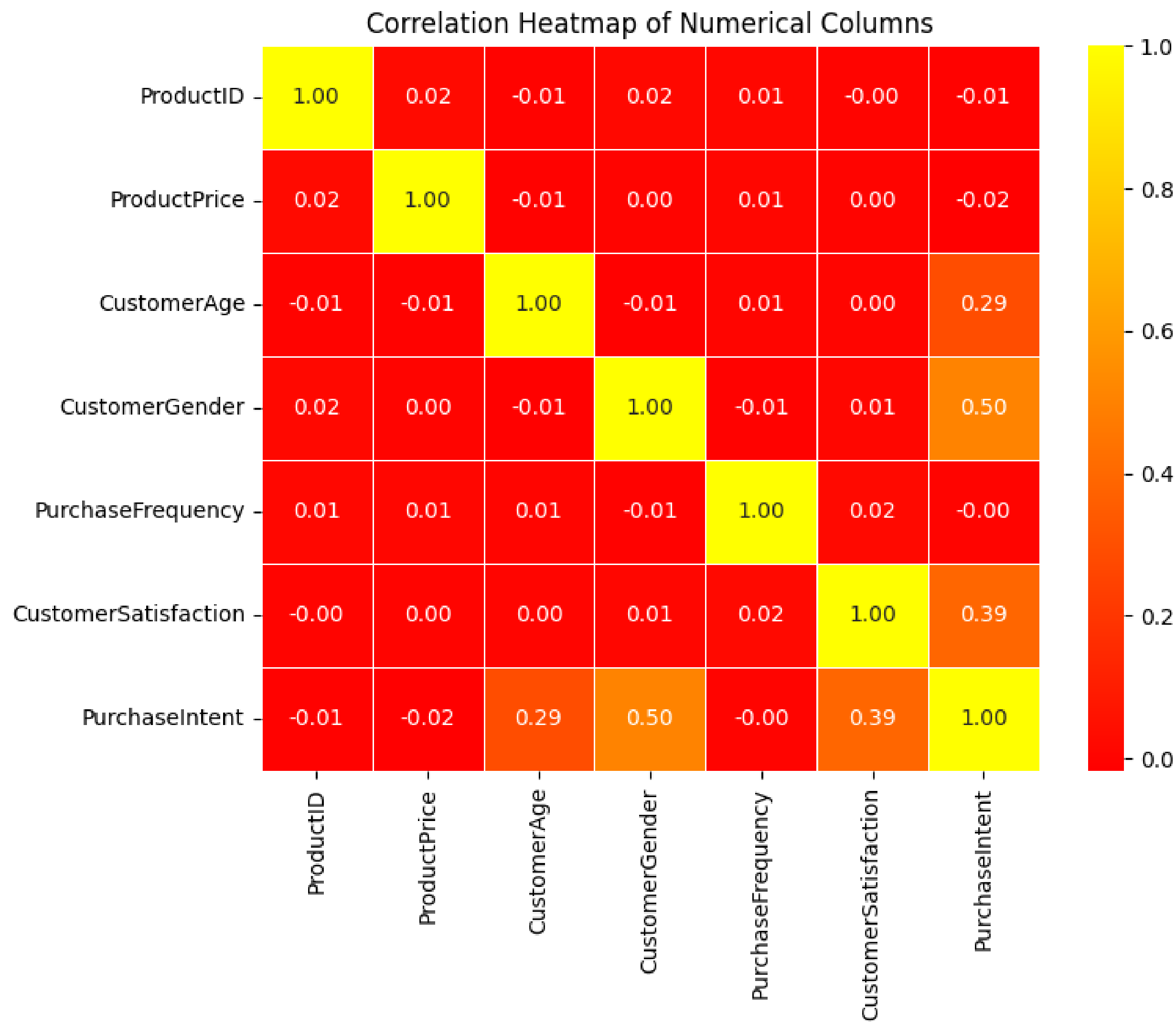
```
plt.figure(figsize=(8,5))
ax =
sns.countplot(data=df,x='CustomerSatisfaction',hue='CustomerSatisfacti
on',palette='Greens')
for container in ax.containers:
    ax.bar_label(container,label_type='edge',fontsize=12)
plt.title('Distribution of Customer Satisfaction',fontsize=14)
plt.xlabel('Customer Satisfaction Level',fontsize=12)
plt.ylabel('Frequency',fontsize=12)
plt.show()
```

Observation

- Highest number of Customers i.e 1848 customers have given rating of 3 and 1765 customers have given rating of 5.
- This graph is also relatively uniform.

Correlation Matrix

```
num_cols = df.select_dtypes(include=['number'])
corr_matrix = num_cols.corr()
plt.figure(figsize=(8,6))
sns.heatmap(corr_matrix,annot=True,cmap='autumn',fmt='.2f',
linewidths=0.5)
plt.title("Correlation Heatmap of Numerical Columns")
plt.show()
```



Preprocessing Steps

Dropping Unnecessary Column

```
# Dropping 'ProductID' column as it is not necessary
df.drop('ProductID', axis=1, inplace=True)
df.head()

  ProductCategory  ProductBrand  ProductPrice  CustomerAge
CustomerGender  \
0     Smartphones  Other Brands    312.949668           18
0
1   Smart Watches       Samsung    980.389404           35
1
2         Tablets       Samsung   2606.718293           63
0
3     Smartphones       Samsung    870.395450           63
```

```
1
4         Tablets         Sony   1798.955875          57
0

   PurchaseFrequency  CustomerSatisfaction  PurchaseIntent
0                  2                     1               0
1                  7                     2               1
2                  1                     5               1
3                 10                     3               1
4                 17                     3               0
```

```
df.head()
```

```
  ProductCategory  ProductBrand  ProductPrice  CustomerAge
CustomerGender  \
0     Smartphones  Other Brands    312.949668           18
0
1   Smart Watches       Samsung    980.389404           35
1
2         Tablets       Samsung   2606.718293           63
0
3     Smartphones       Samsung    870.395450           63
1
4         Tablets          Sony   1798.955875           57
0

   PurchaseFrequency  CustomerSatisfaction  PurchaseIntent
0                  2                     1               0
1                  7                     2               1
2                  1                     5               1
3                 10                     3               1
4                 17                     3               0
```

Checking Skewness

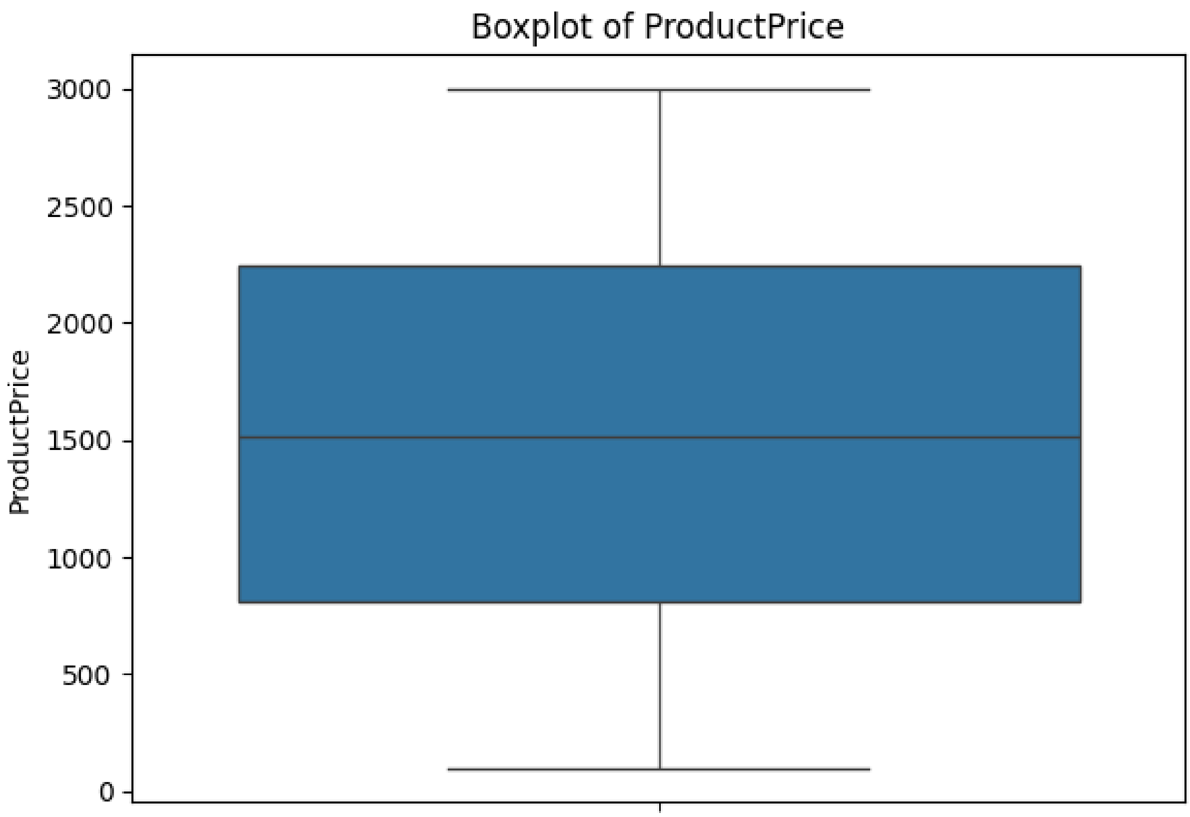
```
df[['ProductPrice','CustomerAge','PurchaseFrequency','CustomerSatisfac
tion','PurchaseIntent','CustomerGender']].skew()
```

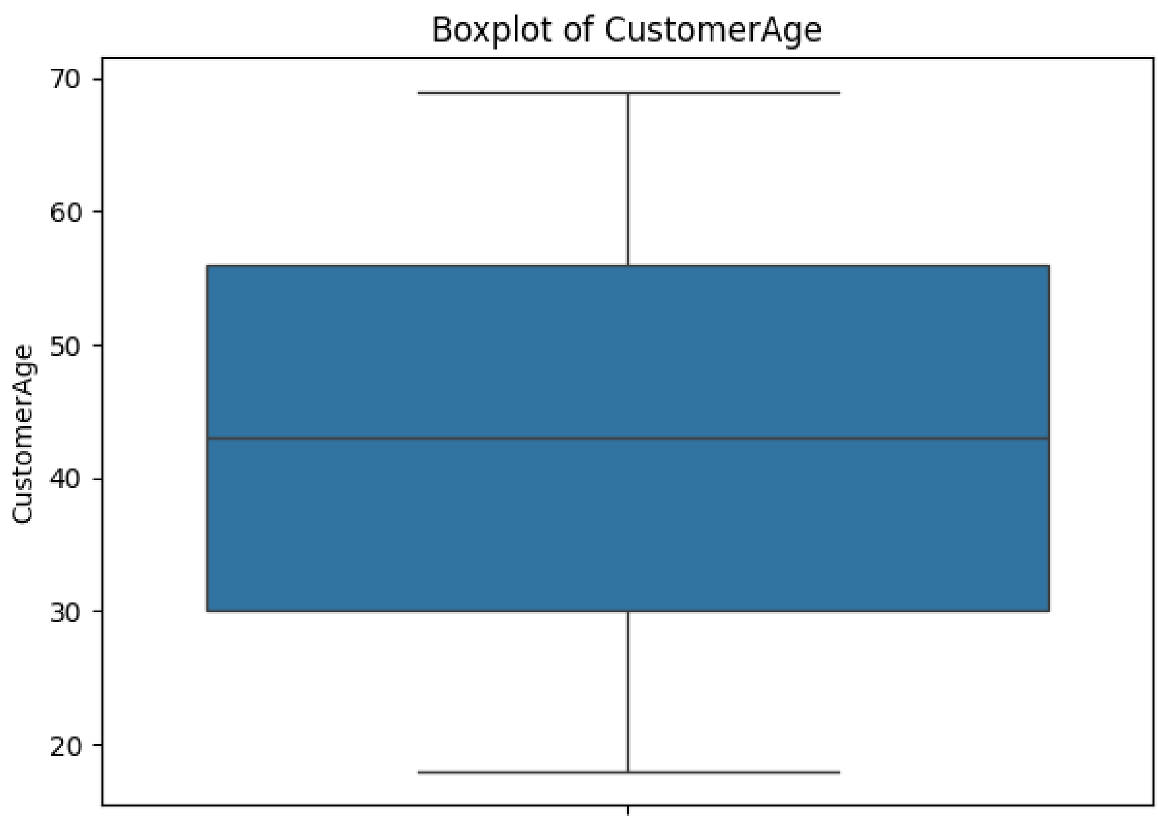
```
ProductPrice            0.029077
CustomerAge             0.003520
PurchaseFrequency      -0.001468
CustomerSatisfaction    0.004696
PurchaseIntent         -0.268201
CustomerGender         -0.035567
dtype: float64
```

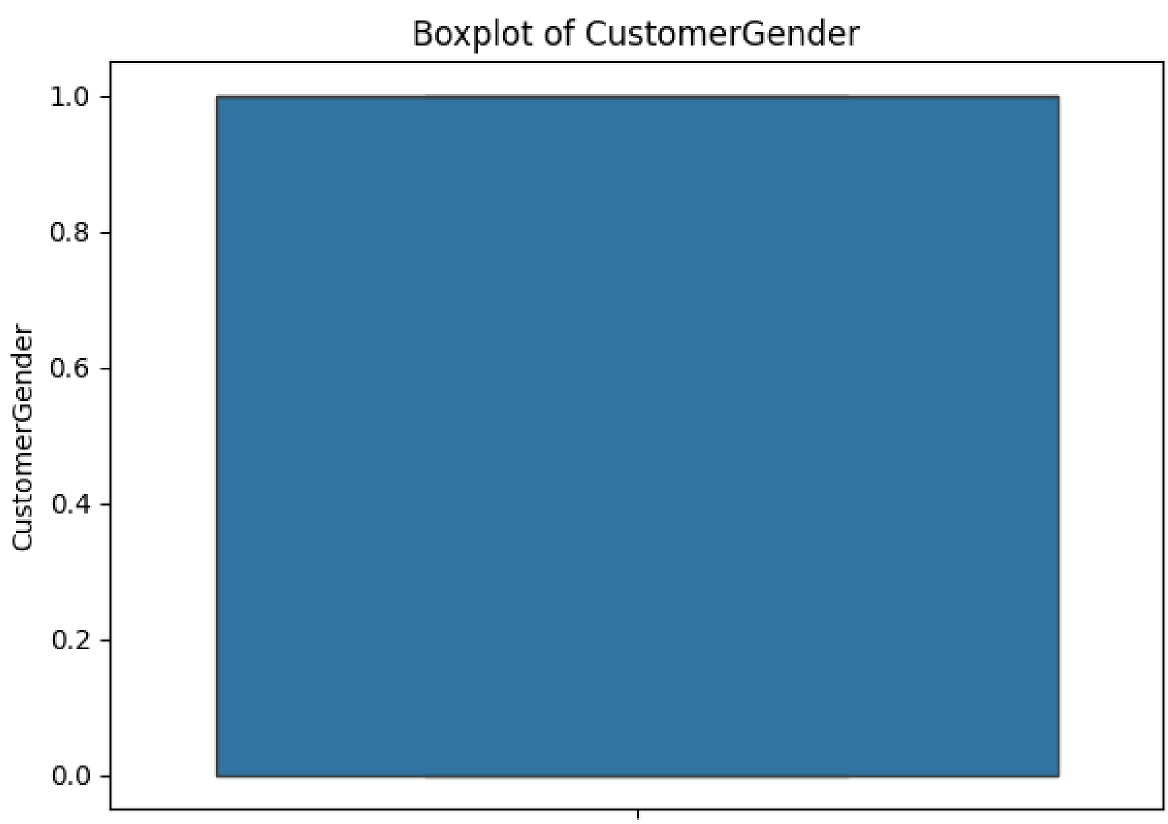
Checking Outliers

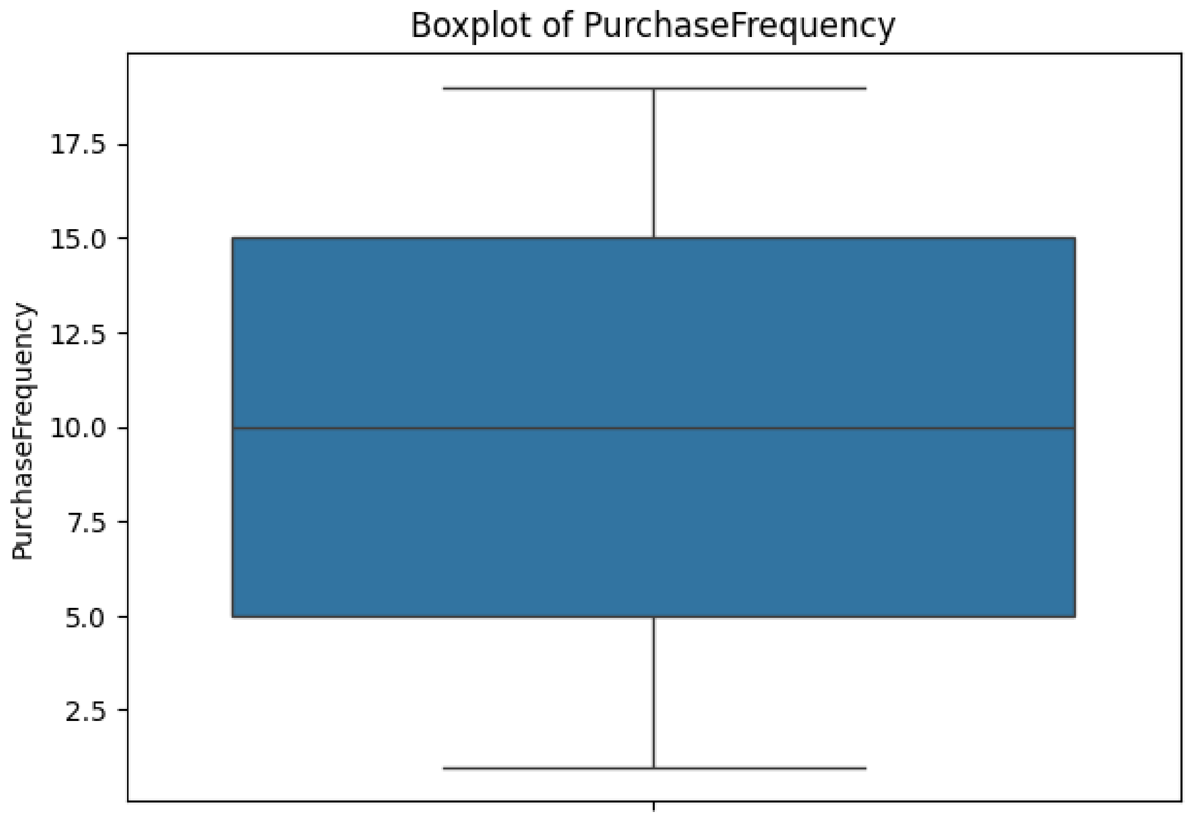
```
num_cols = df.select_dtypes(include=['number'])
for i in num_cols:
    plt.figure(figsize=(7,5))
```

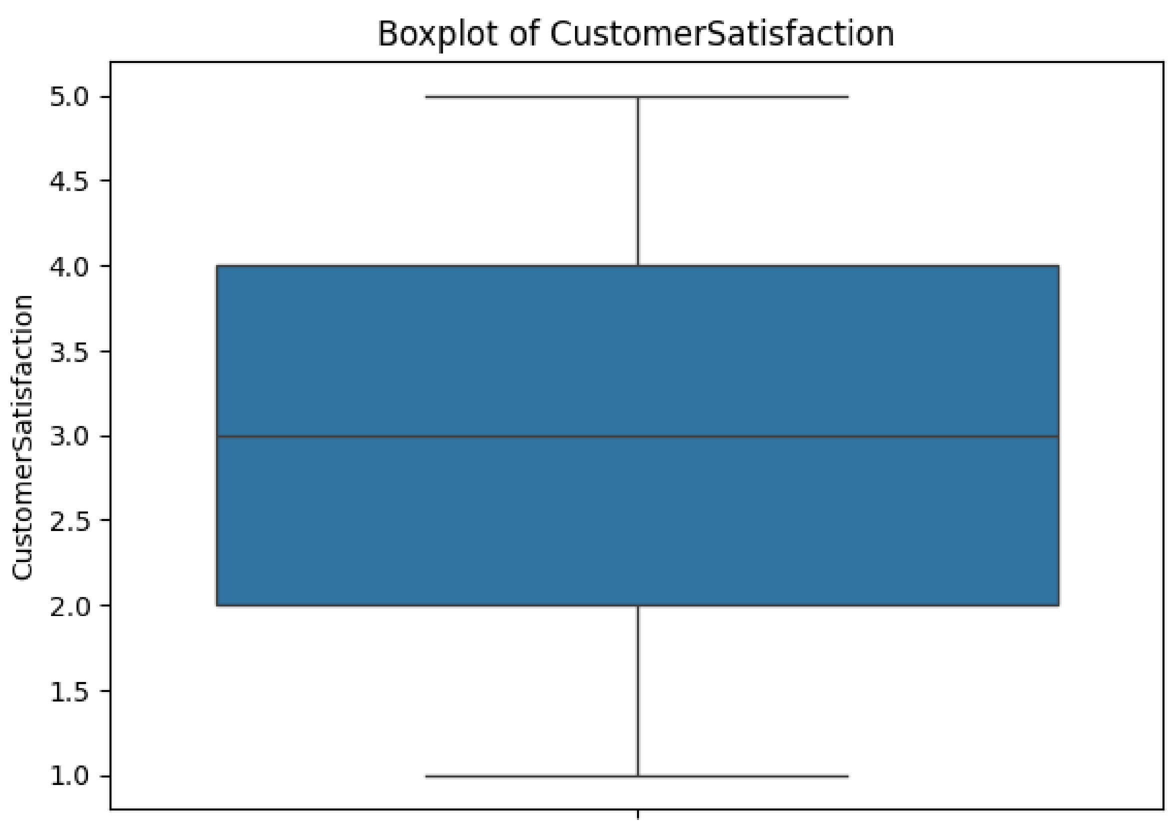
```
    sns.boxplot(data=df,y=i)
    plt.title(f"Boxplot of {i}")
    plt.ylabel(i)
```

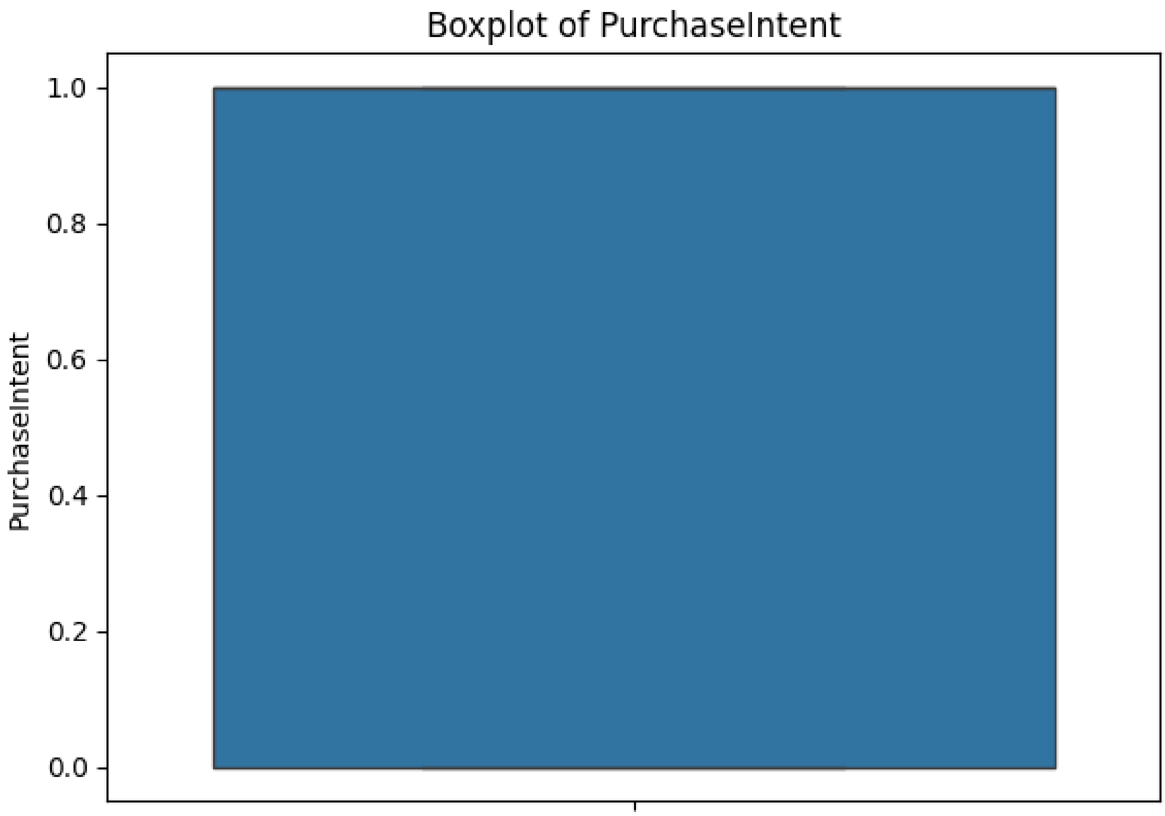



Boxplot of CustomerAge
70
60
50
40
30
20
CustomerAge


Boxplot of CustomerGender
1.0
0.8
0.6
0.4
0.2
0.0
CustomerGender


Boxplot of PurchaseFrequency
PurchaseFrequency
17.5
15.0
12.5
10.0
7.5
5.0
2.5


Boxplot of CustomerSatisfaction
CustomerSatisfaction
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0



Observation

- There is no outlier present in the dataset

Converting Categorical column to Numerical

```
df.select_dtypes(include=['object','category']).columns

Index(['ProductCategory', 'ProductBrand'], dtype='object')

df['ProductCategory'].unique()

array(['Smartphones', 'Smart Watches', 'Tablets', 'Laptops',
'Headphones'],
      dtype=object)

df['ProductBrand'].unique()

array(['Other Brands', 'Samsung', 'Sony', 'HP', 'Apple'],
dtype=object)

# importing library

from sklearn.preprocessing import LabelEncoder

LE = LabelEncoder()
LE
```

```
LabelEncoder()

df['ProductCategory'] = LE.fit_transform(df['ProductCategory'])
df['ProductBrand'] = LE.fit_transform(df['ProductBrand'])

df.head()
```

```
   ProductCategory  ProductBrand  ProductPrice  CustomerAge  
CustomerGender  \
0                3             2    312.949668           18   
0   
1                2             3    980.389404           35   
1   
2                4             3   2606.718293           63   
0   
3                3             3    870.395450           63   
1   
4                4             4   1798.955875           57   
0   

   PurchaseFrequency  CustomerSatisfaction  PurchaseIntent  
0                  2                     1               0  
1                  7                     2               1  
2                  1                     5               1  
3                 10                     3               1  
4                 17                     3               0  
```

Training and Testing

```
x = df.drop('PurchaseIntent',axis=1)
x
```

```
      ProductCategory  ProductBrand  ProductPrice  CustomerAge  \
0                   3             2    312.949668           18   
1                   2             3    980.389404           35   
2                   4             3   2606.718293           63   
3                   3             3    870.395450           63   
4                   4             4   1798.955875           57   
...               ...           ...           ...          ...   
8995                2             3   1041.149163           36   
8996                3             3   1485.694311           57   
8997                0             3   2887.369597           28   
8998                4             1   1490.453964           38   
8999                3             4   2315.583087           62   

      CustomerGender  PurchaseFrequency  CustomerSatisfaction  
0                  0                  2                     1  
1                  1                  7                     2  
2                  0                  1                     5  
3                  1                 10                     3  
```

```
4                  0                 17                      3
...              ...                ...                    ...
8995               1                 16                      4
8996               0                  5                      1
8997               0                 18                      4
8998               0                  4                      2
8999               0                 15                      2

[9000 rows x 7 columns]
```

```
y = df['PurchaseIntent']
y
```

```
0       0
1       1
2       1
3       1
4       0
       ..
8995    0
8996    1
8997    0
8998    1
8999    1
Name: PurchaseIntent, Length: 9000, dtype: int64
```

```
# importiong library

from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test =
train_test_split(x,y,test_size=0.3,random_state=42)
```

Logistic Regression

```
# importing library
from sklearn.linear_model import LogisticRegression

LR = LogisticRegression()
LR

LogisticRegression()

LR.fit(x_train,y_train)

C:\Users\jadem\anaconda3\Lib\site-packages\sklearn\linear_model\
_logistic.py:469: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
```

```
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

LogisticRegression()
```

Model Prediction

```
y_pred = LR.predict(x_test)
y_pred

array([1, 1, 1, ..., 1, 1, 1])

y_test

7940    1
1162    1
582     1
4081    0
8412    0
       ..
6764    0
1450    1
7461    1
6505    1
4927    1
Name: PurchaseIntent, Length: 2700, dtype: int64

error = y_test - y_pred
error

7940    0
1162    0
582     0
4081    0
8412    0
       ..
6764    0
1450    0
7461    0
6505    0
4927    0
Name: PurchaseIntent, Length: 2700, dtype: int64
```

Training Score

```
LR.score(x_train,y_train)
```

```
0.8188888888888889
```

Testing Score

```
LR.score(x_test,y_test)

0.8140740740740741
```

Accuracy of Model

```
# importing library
from sklearn.metrics import
accuracy_score,confusion_matrix,classification_report

acc = accuracy_score(y_pred,y_test)*100
print(f"Accuracy of model using Logistic Regression is {acc:.2f}%")

Accuracy of model using Logistic Regression is 81.41%

print('Confusion Matrix')
confusion_matrix(y_test,y_pred)

Confusion Matrix

array([[ 882,  306],
       [ 196, 1316]])

print('Classification Report')
print(classification_report(y_pred,y_test))

Classification Report
              precision    recall  f1-score   support

           0       0.74      0.82      0.78      1078
           1       0.87      0.81      0.84      1622

    accuracy                           0.81      2700
   macro avg       0.81      0.81      0.81      2700
weighted avg       0.82      0.81      0.82      2700
```

KNN Classifier

```
from sklearn.neighbors import KNeighborsClassifier

KNN = KNeighborsClassifier(n_neighbors=6)
KNN

KNeighborsClassifier(n_neighbors=6)

KNN.fit(x_train,y_train)

KNeighborsClassifier(n_neighbors=6)
```

Model Prediction

```
y_pred = KNN.predict(x_test)
y_pred
```

```
array([1, 1, 1, ..., 0, 1, 1])
```

```
y_test
```

```
7940    1
1162    1
582     1
4081    0
8412    0
       ..
6764    0
1450    1
7461    1
6505    1
4927    1
Name: PurchaseIntent, Length: 2700, dtype: int64
```

```
error = y_test - y_pred
error
```

```
7940    0
1162    0
582     0
4081    0
8412    0
       ..
6764   -1
1450    1
7461    1
6505    0
4927    0
Name: PurchaseIntent, Length: 2700, dtype: int64
```

Training Score

```
KNN.score(x_train,y_train)
```

```
0.7377777777777778
```

Testing Score

```
KNN.score(x_test,y_test)
```

```
0.6066666666666667
```

Accuracy of Model

```
accr = accuracy_score(y_pred,y_test)*100
print(f"Accuracy of Model using KNN Classifier {accr:.2f}%")
```

```
Accuracy of Model using KNN Classifier 60.67%
```

```
print("Confusion Matrix")
confusion_matrix(y_test,y_pred)
```

```
Confusion Matrix
```

```
array([[726, 462],
       [600, 912]])
```

```
print("Classification Report")
print(classification_report(y_test,y_pred))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.55      0.61      0.58      1188
           1       0.66      0.60      0.63      1512

    accuracy                           0.61      2700
   macro avg       0.61      0.61      0.60      2700
weighted avg       0.61      0.61      0.61      2700
```

Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
```

```
DTC = DecisionTreeClassifier()
DTC
```

```
DecisionTreeClassifier()
```

```
DTC.fit(x_train,y_train)
```

```
DecisionTreeClassifier()
```

Model Prediction

```
y_pred = DTC.predict(x_test)
y_pred
```

```
array([1, 1, 1, ..., 1, 1, 1])
```

```
y_test
```

```
7940    1
1162    1
582     1
4081    0
8412    0
       ..
6764    0
1450    1
7461    1
6505    1
4927    1
Name: PurchaseIntent, Length: 2700, dtype: int64
```

```
error = y_test - y_pred
error
```

```
7940    0
1162    0
582     0
4081    0
8412    0
       ..
6764    0
1450    0
7461    0
6505    0
4927    0
Name: PurchaseIntent, Length: 2700, dtype: int64
```

Training Score

```
DTC.score(x_train,y_train)
```

```
1.0
```

Testing Score

```
DTC.score(x_test,y_test)
```

```
0.8951851851851852
```

Accuracy of Model

```
acc = accuracy_score(y_test,y_pred)*100
print(f"Accuracy score of Model using Decision Tree Classifier
{acc:.2f}%")
```

```
Accuracy score of Model using Decision Tree Classifier 89.52%
```

```
print("Confusion Matrix")
confusion_matrix(y_test,y_pred)
```

```
Confusion Matrix
```

```
array([[1043,  145],
       [ 138, 1374]])
```

```
print("Classification Report")
print(classification_report(y_test,y_pred))
```

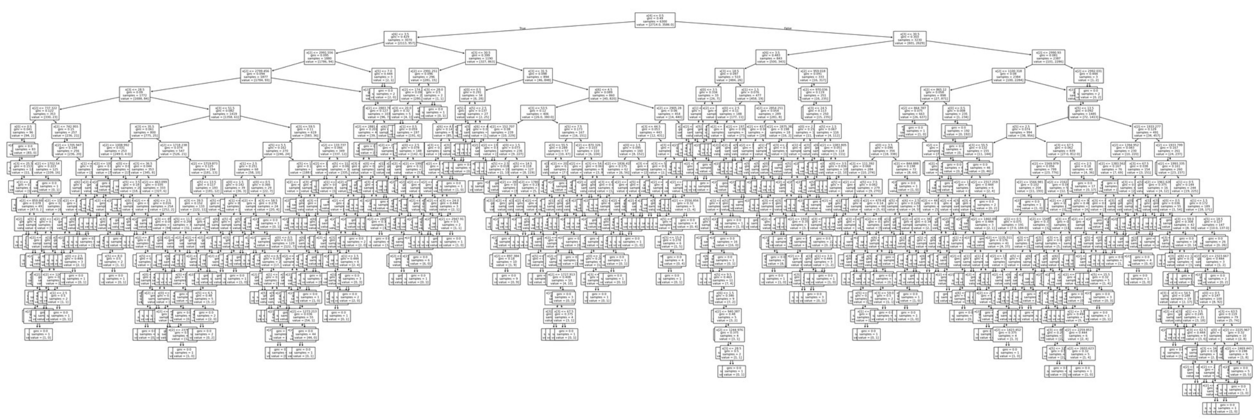
```
Classification Report
              precision    recall  f1-score   support

           0       0.88      0.88      0.88      1188
           1       0.90      0.91      0.91      1512

    accuracy                           0.90      2700
   macro avg       0.89      0.89      0.89      2700
weighted avg       0.90      0.90      0.90      2700
```

```
# Plotting Tree
from sklearn import tree

plt.figure(figsize=(60,20))
tree.plot_tree(DTC,fontsize = 8)
plt.show()
```



Random Forest Classifier

```
# importing library

from sklearn.ensemble import RandomForestClassifier

RFC = RandomForestClassifier(n_estimators=100,random_state=42)
RFC

RandomForestClassifier(random_state=42)

RFC.fit(x_train,y_train)

RandomForestClassifier(random_state=42)
```

Model Prediction

```
y_pred = RFC.predict(x_test)
y_pred

array([1, 1, 1, ..., 1, 1, 1])

y_test

7940    1
1162    1
582     1
4081    0
8412    0
       ..
6764    0
1450    1
7461    1
6505    1
4927    1
Name: PurchaseIntent, Length: 2700, dtype: int64

error = y_test - y_pred
error

7940    0
1162    0
582     0
4081    0
8412    0
       ..
6764    0
1450    0
7461    0
6505    0
4927    0
Name: PurchaseIntent, Length: 2700, dtype: int64
```

Training Score

```
RFC.score(x_train,y_train)
```

```
1.0
```

Testing Score

```
RFC.score(x_test,y_test)
```

```
0.9540740740740741
```

Accuracy of Model

```
acc = accuracy_score(y_test,y_pred)*100
print(f"Accuracy score of Model using Decision Tree Classifier
{acc:.2f}%")
```

```
Accuracy score of Model using Decision Tree Classifier 95.41%
```

```
print("Confusion Matrix")
confusion_matrix(y_test,y_pred)
```

```
Confusion Matrix

array([[1112,   76],
       [  48, 1464]])
```

```
print("Classification Report")
print(classification_report(y_test,y_pred))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.96      0.94      0.95      1188
           1       0.95      0.97      0.96      1512

    accuracy                           0.95      2700
   macro avg       0.95      0.95      0.95      2700
weighted avg       0.95      0.95      0.95      2700
```

```
dict = {'lr':0.81,
       'KNN':0.61,
       'DTC':0.90,
```

```
       'RFC':85.95}
dict
```

```
{'lr': 0.81, 'KNN': 0.61, 'DTC': 0.9, 'RFC': 85.95}
```

Observation

- After performing all Classification Algorithms.
- **Decision tree** and **Random Forest** show best accuracy score.
- **Random Forest** gives the best accuracy score of **95.41%**.
- After Random forest, **Decision Tree** gives the best accuracy score of **89.52%**.
- So we can say that **Random Forest** is best for this dataset.