ml-p-coffeeshopsales-prediction

May 29, 2024

ML Project Coffee Shop Sales Prediction..

[8]:
```
pip install seaborn
```

```
Requirement already satisfied: seaborn in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages
(0.13.2)Note: you may need to restart the kernel to use updated packages.

Requirement already satisfied: numpy!=1.24.0,>=1.20 in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (from
seaborn) (1.26.4)
Requirement already satisfied: pandas>=1.2 in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (from
seaborn) (2.2.2)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (from
seaborn) (3.9.0)
Requirement already satisfied: contourpy>=1.0.1 in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.2.1)
Requirement already satisfied: cycler>=0.10 in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (24.0)
Requirement already satisfied: pillow>=8 in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (10.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (3.1.2)
```

```
Requirement already satisfied: python-dateutil>=2.7 in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (from
pandas>=1.2->seaborn) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (from
pandas>=1.2->seaborn) (2024.1)
Requirement already satisfied: six>=1.5 in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (from
python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.16.0)
```

[9]:
```
pip install scipy
```

```
Requirement already satisfied: scipy in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (1.13.1)
Requirement already satisfied: numpy<2.3,>=1.22.4 in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (from
scipy) (1.26.4)
Note: you may need to restart the kernel to use updated packages.
```

[10]:
```
pip install Scikit-learn
```

```
Requirement already satisfied: Scikit-learn in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (1.5.0)
Requirement already satisfied: numpy>=1.19.5 in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (from
Scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (from
Scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (from
Scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (from
Scikit-learn) (3.5.0)
Note: you may need to restart the kernel to use updated packages.
```

[11]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

[12]:
```
df = pd.read_csv("coffee shop ml.csv")
```

[13]:
```
df
```

[13]:

	transaction_id	transaction_date	transaction_time	store_id	store_location	product_id	transaction_qty	unit_price	product_category	product_type	product_detail	size	Total bill	Month Name	Day Name	Hour	Day of Week	Month
0	114301	01-06-2023	11:33:29	3	Astoria	45	1	3.00	Tea	Brewed herbal tea	Peppermint	Large	3.0	June	Thursday	11	4	6
1	115405	02-06-2023	11:18:24	3	Astoria	45	1	3.00	Tea	Brewed herbal tea	Peppermint	Large	3.0	June	Friday	11	5	6
2	115478	02-06-2023	12:02:45	3	Astoria	45	1	3.00	Tea	Brewed herbal tea	Peppermint	Large	3.0	June	Friday	12	5	6
3	116288	02-06-2023	19:39:47	3	Astoria	45	1	3.00	Tea	Brewed herbal tea	Peppermint	Large						
4	116714	03-06-2023	12:24:57	3	Astoria	45	1	3.00	Tea	Brewed herbal tea	Peppermint	Large						
…	…	…	…	…	…	…	…	…	…	…	…	…						
149111	129465	14-06-2023	08:34:10	5	Lower Manhattan	41	4	4.25	Coffee	Barista Espresso	Cappuccino	Large						
149112	133523	17-06-2023	09:55:47	8	Hell's Kitchen	8	8	45.00	Coffee beans	Premium Beans	Civet Cat	Not Defind						
149113	133674	17-06-2023	10:41:11	8	Hell's Kitchen	8	8	45.00	Coffee beans	Premium Beans	Civet Cat	Not Defind						
149114	133744	17-06-2023	11:18:31	8	Hell's Kitchen	8	8	45.00	Coffee beans	Premium Beans	Civet Cat	Not Defind						
149115	149043	30-06-2023	11:18:31	8	Hell's Kitchen	8	8	45.00	Coffee beans	Premium Beans	Civet Cat	Not Defind						

```
3              3.0       June     Friday    19           5      6
4              3.0       June   Saturday    12           6      6
…              …         …        …   …         …    …
149111        17.0       June  Wednesday     8           3      6
149112       360.0       June   Saturday     9           6      6
149113       360.0       June   Saturday    10           6      6
149114       360.0       June   Saturday    11           6      6
149115       360.0       June     Friday    11           5      6

[149116 rows x 18 columns]
```

[14]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 149116 entries, 0 to 149115
Data columns (total 18 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   transaction_id    149116 non-null  int64
 1   transaction_date  149116 non-null  object
 2   transaction_time  149116 non-null  object
 3   store_id          149116 non-null  int64
 4   store_location    149116 non-null  object
 5   product_id        149116 non-null  int64
 6   transaction_qty   149116 non-null  int64
 7   unit_price        149116 non-null  float64
 8   product_category  149116 non-null  object
 9   product_type      149116 non-null  object
 10  product_detail    149116 non-null  object
 11  size              149116 non-null  object
 12  Total bill        149116 non-null  float64
 13  Month Name        149116 non-null  object
 14  Day Name          149116 non-null  object
 15  Hour              149116 non-null  int64
 16  Day of Week       149116 non-null  int64
 17  Month             149116 non-null  int64
dtypes: float64(2), int64(7), object(9)
memory usage: 20.5+ MB
```

1 Column names and descriptions

transaction_id : shows the details of the customers. transaction_date : shows customers transaction date. transaction_time : shows customers transaction time. store_id : shows the details no. of stores. store_location : shows location of each stores. product_id : shows details of each stores. transaction_qty : shows transaction quantity. unit_price : shows each unit price. product_category : it shows product category. product_type : it shows product type. product_detail :shows product detail. size :shows its size Small , Large , Regular. Total bill :it shows details of

total bill. Month Name :it shows month names. Day Name :it shows days names. Hour :it shows time in Hour. Day of Week :it shows week days. Month :it shows month in count.

[16]:
```
df.isnull().sum()
```

[16]:
```
transaction_id      0
transaction_date    0
transaction_time    0
store_id            0
store_location      0
product_id          0
transaction_qty     0
unit_price          0
product_category    0
product_type        0
product_detail      0
size                0
Total bill          0
Month Name          0
Day Name            0
Hour                0
Day of Week         0
Month               0
dtype: int64
```

2 Seperate the columns into x and y

[18]:
```
x = df.iloc[:,:-1]
x.shape
```

[18]:
```
(149116, 17)
```

[19]:
```
y = df.iloc[:,-1]
y.shape
```

[19]:
```
(149116,)
```

3 Encoding

[21]:
```
cat_col = x.select_dtypes(object).columns
cat_col
```

[21]:
```
Index(['transaction_date', 'transaction_time', 'store_location',
       'product_category', 'product_type', 'product_detail', 'size',
       'Month Name', 'Day Name'],
```

```
      dtype='object')
```

[22]:
```
from sklearn.preprocessing import OrdinalEncoder
oe = OrdinalEncoder()
x[cat_col] = oe.fit_transform(x[cat_col])
```

[23]:
```
x
```

[23]:

	transaction_id	transaction_date	transaction_time	store_id	\
0	114301	5.0	11609.0	3	
1	115405	11.0	11126.0	3	
2	115478	11.0	12451.0	3	
3	116288	11.0	25184.0	3	
4	116714	17.0	13090.0	3	
…	…	…	…	…	
149111	129465	83.0	4499.0	5	
149112	133523	101.0	7871.0	8	
149113	133674	101.0	9786.0	8	
149114	133744	101.0	11133.0	8	
149115	149043	177.0	11133.0	8	

	store_location	product_id	transaction_qty	unit_price	\
0	0.0	45	1	3.00	
1	0.0	45	1	3.00	
2	0.0	45	1	3.00	
3	0.0	45	1	3.00	
4	0.0	45	1	3.00	
…	…	…	…	…	
149111	2.0	41	4	4.25	
149112	1.0	8	8	45.00	
149113	1.0	8	8	45.00	
149114	1.0	8	8	45.00	
149115	1.0	8	8	45.00	

	product_category	product_type	product_detail	size	Total bill	\
0	8.0	6.0	37.0	0.0	3.0	
1	8.0	6.0	37.0	0.0	3.0	
2	8.0	6.0	37.0	0.0	3.0	
3	8.0	6.0	37.0	0.0	3.0	
4	8.0	6.0	37.0	0.0	3.0	
…	…	…	…	…	…	
149111	2.0	0.0	3.0	0.0	17.0	
149112	3.0	24.0	9.0	1.0	360.0	
149113	3.0	24.0	9.0	1.0	360.0	
149114	3.0	24.0	9.0	1.0	360.0	
149115	3.0	24.0	9.0	1.0	360.0	

```
        Month Name  Day Name  Hour  Day of Week
0              3.0       4.0    11            4
1              3.0       0.0    11            5
2              3.0       0.0    12            5
3              3.0       0.0    19            5
4              3.0       2.0    12            6
…              …         …      …             …
149111         3.0       6.0     8            3
149112         3.0       2.0     9            6
149113         3.0       2.0    10            6
149114         3.0       2.0    11            6
149115         3.0       0.0    11            5

[149116 rows x 17 columns]
```

4 Split the data into training and testing

[25]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size = 0.2,random_state =␣
↪1)
```

5 1 - Predicting the data using Logistics Regression

[27]:
```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(xtrain,ytrain)
ypred = logreg.predict(xtest)
```

[28]:
```
# Evaluate the model
from sklearn.metrics import accuracy_score,classification_report
ac = accuracy_score(ytest,ypred)
cr = classification_report(ytest,ypred)
print("Accuracy score : ",ac)
print(cr)
```

```
Accuracy score :  0.541711373390558
              precision    recall  f1-score   support

           1       0.77      0.79      0.78      3396
           2       0.50      0.19      0.27      3234
           3       0.40      0.38      0.39      4265
           4       0.50      0.50      0.50      5053
           5       0.45      0.47      0.46      6745
           6       0.62      0.78      0.69      7131
```

```
    accuracy                           0.54     29824
   macro avg       0.54      0.52      0.51     29824
weighted avg       0.53      0.54      0.53     29824
```

We have achieved an Average Accuracy of 54% which is not that good. Lets see if we can increase this accuracy by hyper tuning

6 HPT

[31]:
```
logreg = LogisticRegression(solver = "liblinear")
logreg.fit(xtrain,ytrain)
ypred = logreg.predict(xtest)
```

[32]:
```
ac = accuracy_score(ytest,ypred)
cr = classification_report(ytest,ypred)
print("Accuracy score : ",ac)
print(cr)
```

```
Accuracy score :  0.8077722639484979
              precision    recall  f1-score   support

           1       1.00      1.00      1.00      3396
           2       1.00      0.25      0.39      3234
           3       0.29      0.23      0.25      4265
           4       1.00      1.00      1.00      5053
           5       0.67      1.00      0.80      6745
           6       1.00      1.00      1.00      7131

    accuracy                           0.81     29824
   macro avg       0.83      0.75      0.74     29824
weighted avg       0.82      0.81      0.78     29824
```

#By using liblinear we get Accuracy of 81%

[34]:
```
logreg = LogisticRegression(solver = 'newton-cg')
logreg.fit(xtrain,ytrain)
ypred = logreg.predict(xtest)
```

[35]:
```
ac = accuracy_score(ytest,ypred)
cr = classification_report(ytest,ypred)
print("Accuracy score : ",ac)
print(cr)
```

```
Accuracy score :  0.9999329399141631
              precision    recall  f1-score   support
```

```
           1       1.00      1.00      1.00      3396
           2       1.00      1.00      1.00      3234
           3       1.00      1.00      1.00      4265
           4       1.00      1.00      1.00      5053
           5       1.00      1.00      1.00      6745
           6       1.00      1.00      1.00      7131

    accuracy                           1.00     29824
   macro avg       1.00      1.00      1.00     29824
weighted avg       1.00      1.00      1.00     29824
```

#By using newton-cg we get Accuracy of 100% which is good.

HPT Conclusion: liblinear and newton-cg –> newton-cg we get Accuracy of 100% which is good than liblinear

7 2 - Predicting the data using KNN Classifier

[39]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 5) # by default n_neighbors = 5
knn.fit(xtrain,ytrain)
ypred = knn.predict(xtest)
```

[40]:
```
#Evaluate the model

from sklearn.metrics import accuracy_score
ac = accuracy_score(ytest,ypred)
print(ac)
```

```
0.9999664699570815
```

#By using KNN CLASSIFIER we get 99% accuracy.

8 3 - Predicting the data using Random Forest

[43]:
```
from sklearn.ensemble import RandomForestClassifier
rc = RandomForestClassifier()
rc.fit(xtrain,ytrain)
ypred = rc.predict(xtest)
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00      3396
           2       1.00      1.00      1.00      3234
           3       1.00      1.00      1.00      4265
           4       1.00      1.00      1.00      5053
```

```
           5       1.00      1.00      1.00      6745
           6       1.00      1.00      1.00      7131

    accuracy                           1.00     29824
   macro avg       1.00      1.00      1.00     29824
weighted avg       1.00      1.00      1.00     29824
```

#By using Random Forest we get 100% of accuracy which is good.

9 4. Predicting the data using Boosting

i) • Adaboost Classifier

[47]:
```
from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier()
ada.fit(xtrain,ytrain)
ypred = ada.predict(xtest)
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00      3396
           2       0.00      0.00      0.00      3234
           3       0.57      1.00      0.73      4265
           4       1.00      1.00      1.00      5053
           5       1.00      1.00      1.00      6745
           6       1.00      1.00      1.00      7131

    accuracy                           0.89     29824
   macro avg       0.76      0.83      0.79     29824
weighted avg       0.83      0.89      0.85     29824
```

#By using AdaBoost Classifier we get 89% of accuracy

ii) • Gradient Boosting

[50]:
```
from sklearn.ensemble import GradientBoostingClassifier
gbc = GradientBoostingClassifier()
gbc.fit(xtrain,ytrain)
ypred = gbc.predict(xtest)
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00      3396
           2       1.00      1.00      1.00      3234
           3       1.00      1.00      1.00      4265
```

```
           4       1.00      1.00      1.00      5053
           5       1.00      1.00      1.00      6745
           6       1.00      1.00      1.00      7131

    accuracy                           1.00     29824
   macro avg       1.00      1.00      1.00     29824
weighted avg       1.00      1.00      1.00     29824
```

#By using Gradient Boosting algorithm we get 100% of accuracy which is good.

[51]:

10 Conclusion: Based on the above accuracy scores, we should go ahead with KNN Classifier , Random forest or Gradient boosting.