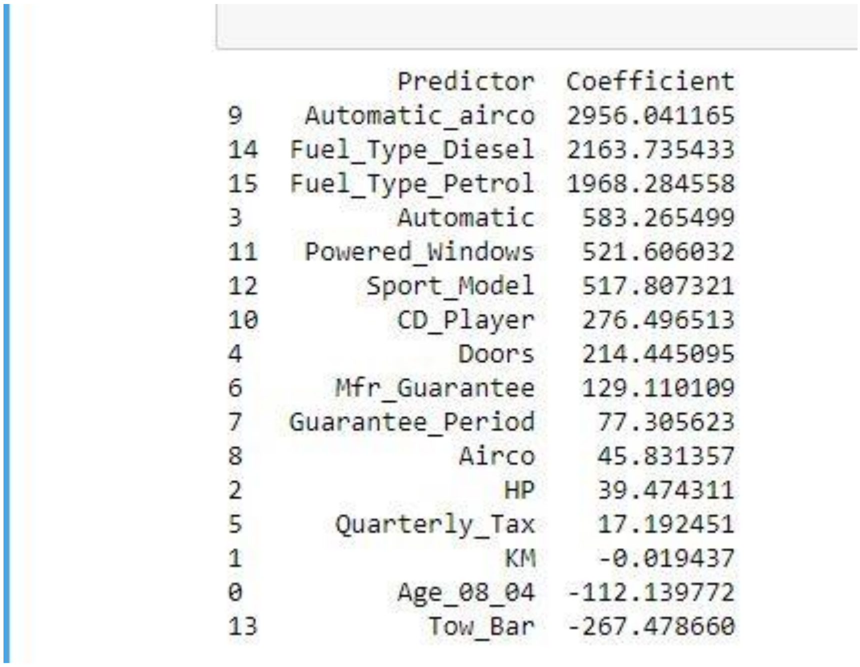
تکلیف سوم درس مبانی داده کاوی

امید رئیسی (۹۶۲۱۱۶۰۰۱۵)

۱)

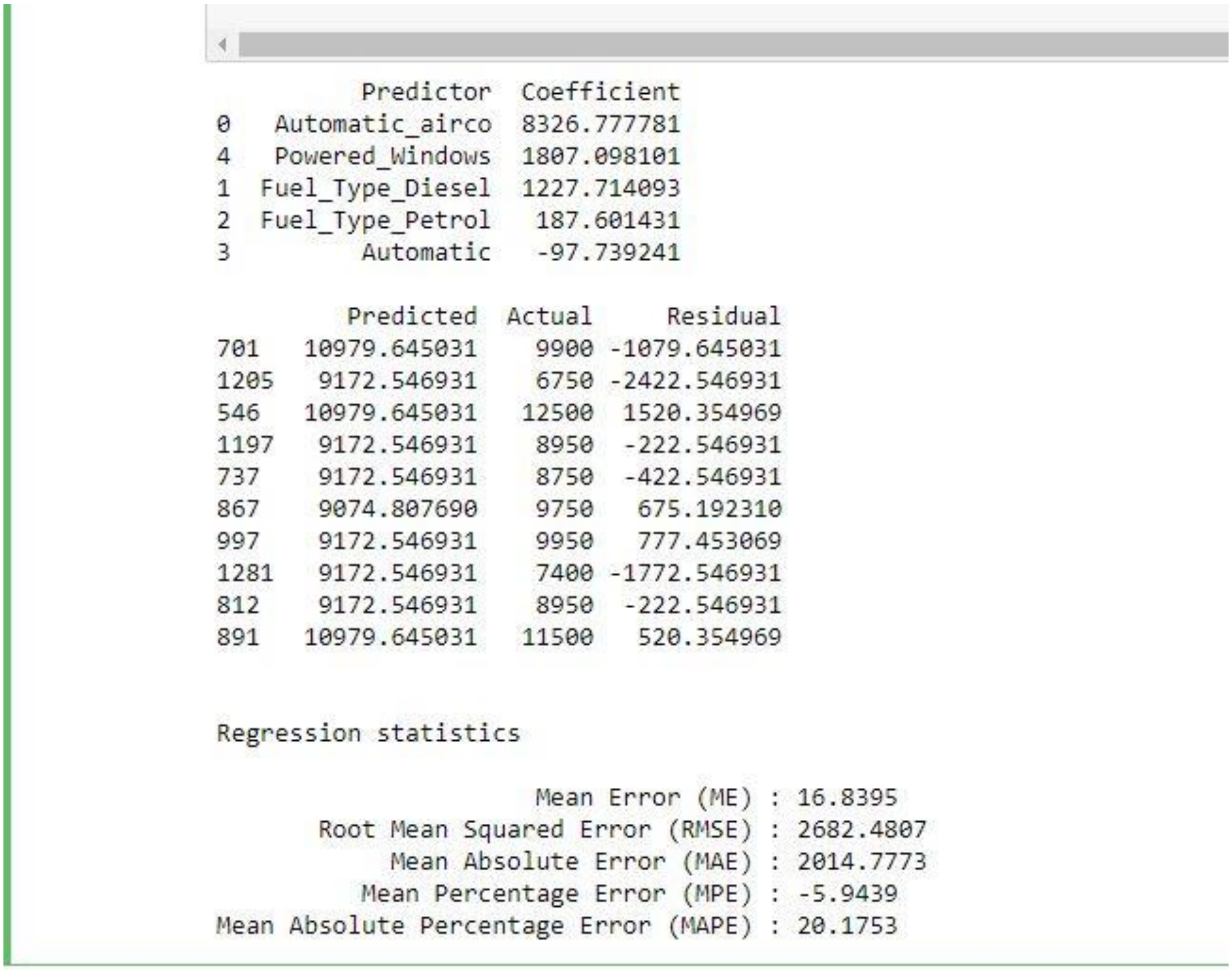
الف) همان‌طور که از شکل زیر پیداست متغیرهایی که دارای بیشترین ضرایب می‌باشند بیشترین تأثیر را در پیشبینی قیمت اتوموبیل دارند. چهار متغیری که بیشترین تأثیر را دارند عبارت‌اند از:

Automatic_airco, Fuel_Type, Automatic, Powered_Windows



```
           Predictor  Coefficient
9     Automatic_airco  2956.041165
14   Fuel_Type_Diesel  2163.735433
15   Fuel_Type_Petrol  1968.284558
3           Automatic   583.265499
11    Powered_Windows   521.606032
12        Sport_Model   517.807321
10          CD_Player   276.496513
4               Doors   214.445095
6       Mfr_Guarantee   129.110109
7    Guarantee_Period    77.305623
8               Airco    45.831357
2                  HP    39.474311
5       Quarterly_Tax    17.192451
1                  KM    -0.019437
0           Age_08_04  -112.139772
13            Tow_Bar  -267.478660
```

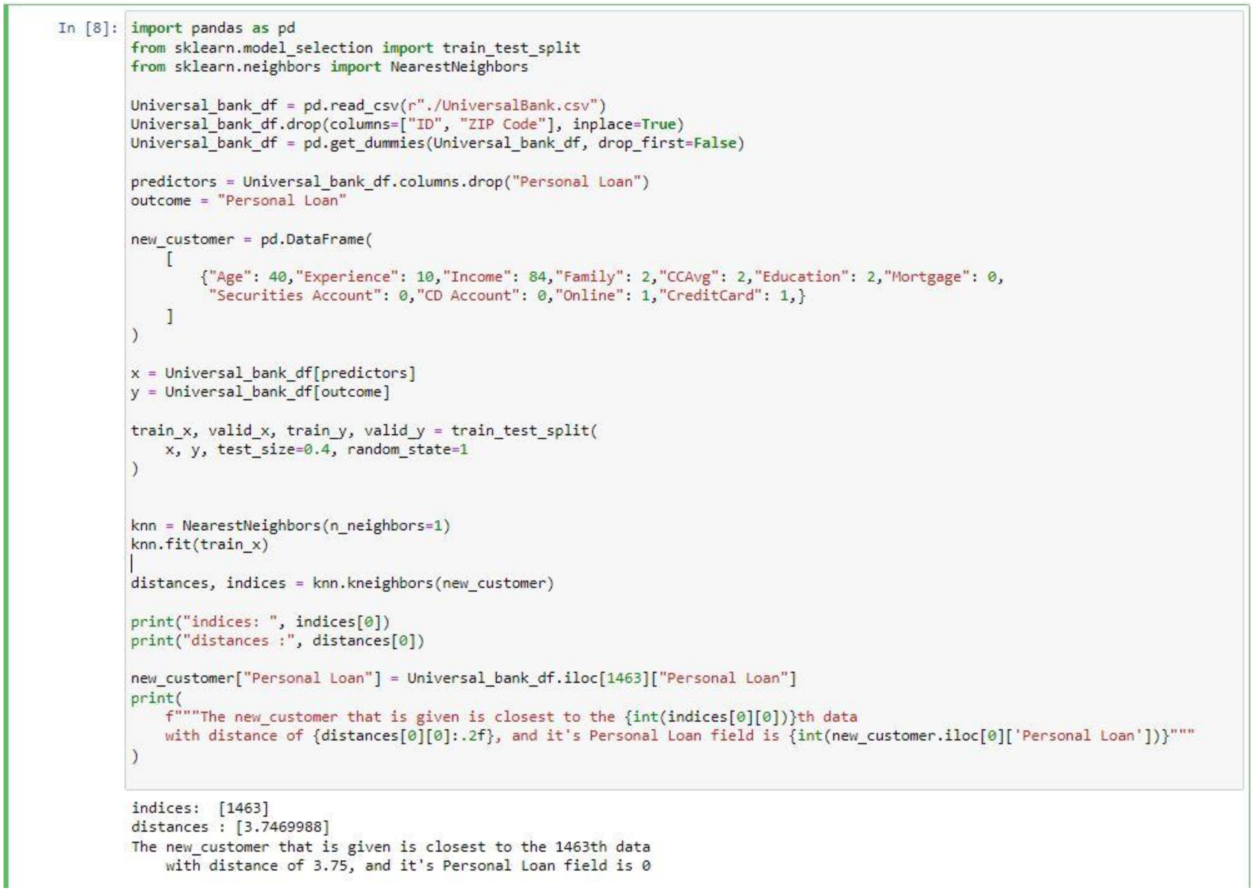
ب) با استفاده از متغیرهایی که در بالا نام بردیم دوباره مدل رگرسیون خود را می‌سازیم و سپس با استفاده از داده‌های اعتبارسنجی مدل خود را ارزیابی می‌کنیم. (به دلیل کاهش تعداد متغیرها میزان خطا بالا رفته است.)



```
          Predictor  Coefficient
0   Automatic_airco  8326.777781
4    Powered_Windows  1807.098101
1  Fuel_Type_Diesel  1227.714093
2  Fuel_Type_Petrol   187.601431
3         Automatic   -97.739241

         Predicted  Actual     Residual
701   10979.645031    9900 -1079.645031
1205   9172.546931    6750 -2422.546931
546   10979.645031   12500  1520.354969
1197   9172.546931    8950  -222.546931
737    9172.546931    8750  -422.546931
867    9074.807690    9750   675.192310
997    9172.546931    9950   777.453069
1281   9172.546931    7400 -1772.546931
812    9172.546931    8950  -222.546931
891   10979.645031   11500   520.354969


Regression statistics

                      Mean Error (ME) : 16.8395
       Root Mean Squared Error (RMSE) : 2682.4807
            Mean Absolute Error (MAE) : 2014.7773
          Mean Percentage Error (MPE) : -5.9439
Mean Absolute Percentage Error (MAPE) : 20.1753
```

۲)

الف) با توجه به اینکه **k=1** می‌باشد پس مقدار **Personal Loan** نزدیک‌ترین همسایه برای داده جدید نیز انتخاب می‌شود که مقدار آن **0** می‌باشد و این یعنی مشتری جدید هم پیشنهاد بانک را نخواد پذیرفت.

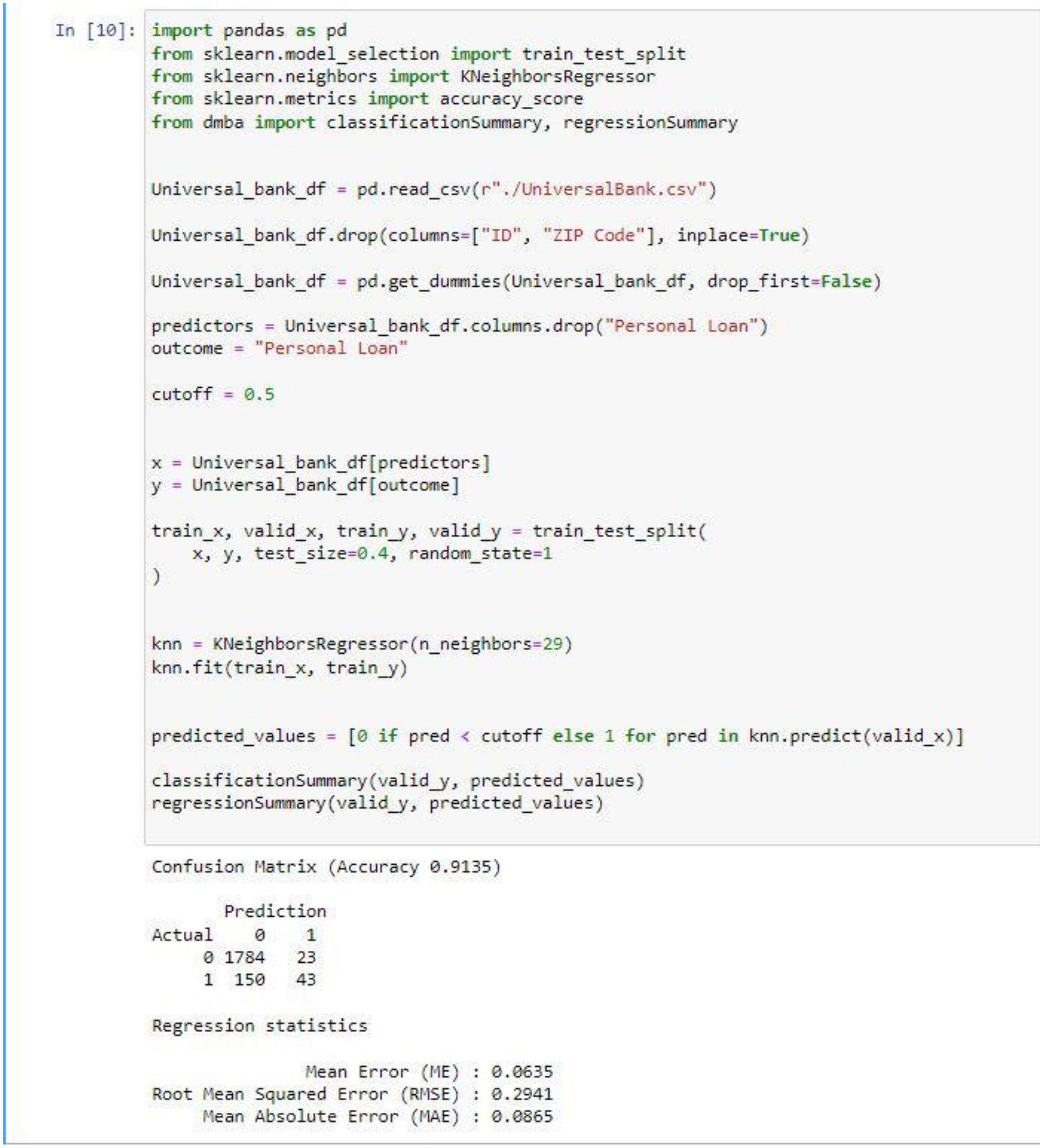


```
In [8]: import pandas as pd
        from sklearn.model_selection import train_test_split
        from sklearn.neighbors import NearestNeighbors

        Universal_bank_df = pd.read_csv(r"./UniversalBank.csv")
        Universal_bank_df.drop(columns=["ID", "ZIP Code"], inplace=True)
        Universal_bank_df = pd.get_dummies(Universal_bank_df, drop_first=False)

        predictors = Universal_bank_df.columns.drop("Personal Loan")
        outcome = "Personal Loan"

        new_customer = pd.DataFrame(
            [
                {"Age": 40,"Experience": 10,"Income": 84,"Family": 2,"CCAvg": 2,"Education": 2,"Mortgage": 0,
                 "Securities Account": 0,"CD Account": 0,"Online": 1,"CreditCard": 1,}
            ]
        )

        x = Universal_bank_df[predictors]
        y = Universal_bank_df[outcome]

        train_x, valid_x, train_y, valid_y = train_test_split(
            x, y, test_size=0.4, random_state=1
        )


        knn = NearestNeighbors(n_neighbors=1)
        knn.fit(train_x)

        distances, indices = knn.kneighbors(new_customer)

        print("indices: ", indices[0])
        print("distances :", distances[0])

        new_customer["Personal Loan"] = Universal_bank_df.iloc[1463]["Personal Loan"]
        print(
            f"""The new_customer that is given is closest to the {int(indices[0][0])}th data
            with distance of {distances[0][0]:.2f}, and it's Personal Loan field is {int(new_customer.iloc[0]['Personal Loan'])}"""
        )
```

```
indices:  [1463]
distances : [3.7469988]
The new_customer that is given is closest to the 1463th data
    with distance of 3.75, and it's Personal Loan field is 0
```

ب) برای پیدا کردن این موازنه باید مقدار بهترین **K** را برای این مسأله پیدا کنیم که طبق عکس مقدار آن برابر با **29** می‌باشد. (**k** از **1** تا **50** فرض شده است.)

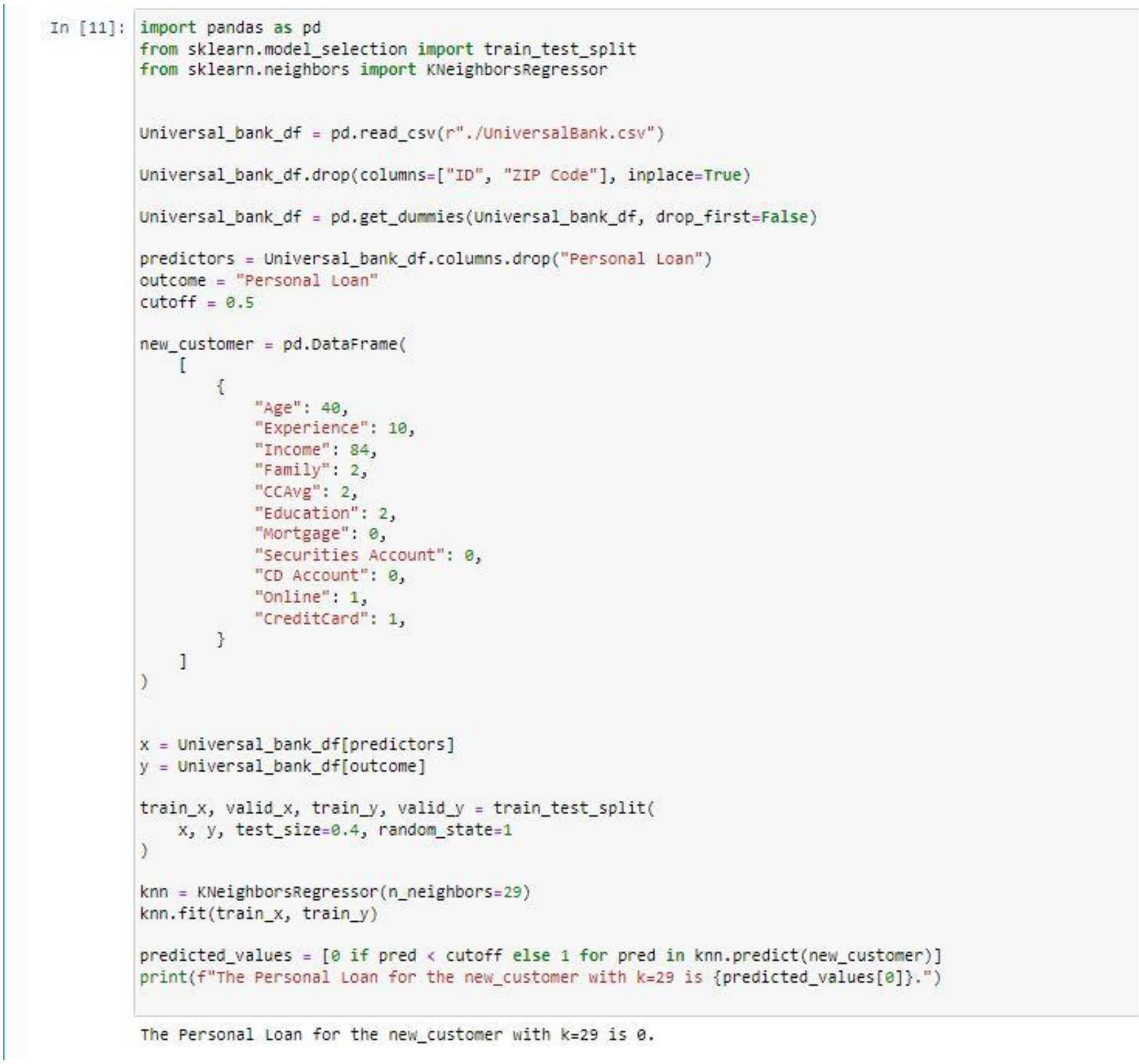
ج)



```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import accuracy_score
from dmba import classificationSummary, regressionSummary


Universal_bank_df = pd.read_csv(r"./UniversalBank.csv")

Universal_bank_df.drop(columns=["ID", "ZIP Code"], inplace=True)

Universal_bank_df = pd.get_dummies(Universal_bank_df, drop_first=False)

predictors = Universal_bank_df.columns.drop("Personal Loan")
outcome = "Personal Loan"

cutoff = 0.5


x = Universal_bank_df[predictors]
y = Universal_bank_df[outcome]

train_x, valid_x, train_y, valid_y = train_test_split(
    x, y, test_size=0.4, random_state=1
)


knn = KNeighborsRegressor(n_neighbors=29)
knn.fit(train_x, train_y)


predicted_values = [0 if pred < cutoff else 1 for pred in knn.predict(valid_x)]

classificationSummary(valid_y, predicted_values)
regressionSummary(valid_y, predicted_values)
```

```
Confusion Matrix (Accuracy 0.9135)

       Prediction
Actual    0    1
     0 1784   23
     1  150   43

Regression statistics

               Mean Error (ME) : 0.0635
Root Mean Squared Error (RMSE) : 0.2941
     Mean Absolute Error (MAE) : 0.0865
```

د) مشتری جدید با مشخصات داده شده پیشنهاد بانک را نخواهد پذیرفت.

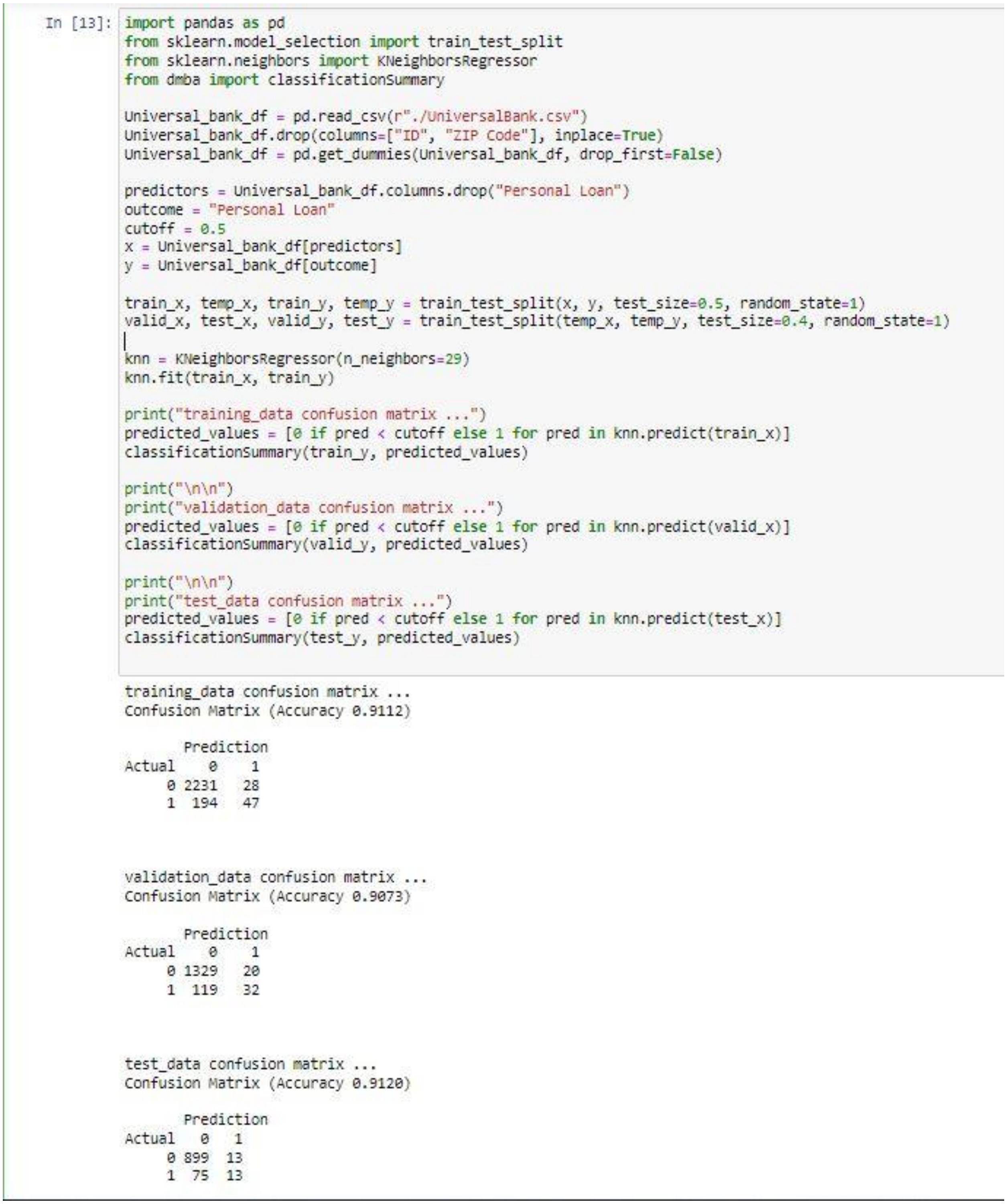


In [11]:

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor


Universal_bank_df = pd.read_csv(r"./UniversalBank.csv")

Universal_bank_df.drop(columns=["ID", "ZIP Code"], inplace=True)

Universal_bank_df = pd.get_dummies(Universal_bank_df, drop_first=False)

predictors = Universal_bank_df.columns.drop("Personal Loan")
outcome = "Personal Loan"
cutoff = 0.5

new_customer = pd.DataFrame(
    [
        {
            "Age": 40,
            "Experience": 10,
            "Income": 84,
            "Family": 2,
            "CCAvg": 2,
            "Education": 2,
            "Mortgage": 0,
            "Securities Account": 0,
            "CD Account": 0,
            "Online": 1,
            "CreditCard": 1,
        }
    ]
)


x = Universal_bank_df[predictors]
y = Universal_bank_df[outcome]

train_x, valid_x, train_y, valid_y = train_test_split(
    x, y, test_size=0.4, random_state=1
)

knn = KNeighborsRegressor(n_neighbors=29)
knn.fit(train_x, train_y)

predicted_values = [0 if pred < cutoff else 1 for pred in knn.predict(new_customer)]
print(f"The Personal Loan for the new_customer with k=29 is {predicted_values[0]}.")
```

```
The Personal Loan for the new_customer with k=29 is 0.
```

ه) با توجه به اینکه تعداد داده‌های آموزشی کاهش یافته‌اند میزان دقت در داده‌های آموزشی نیز کاهش یافته است، اما به طور کلی چون با هر بار پیشبینی داده‌های جدیدی به اضافه شده و دقت در همسایگی افزایش می‌یابد میزان خطا در داده‌های آزمایشی از میزان خطا در داده‌های آموزشی و اعتبارسنجی کمتر است و این یک ویژگی کلی از الگوریتم **KNN** است به این صورت که هر چه پیشبینی‌ها افزایش یابد میزان دقت نیز افزایش می‌یابد.

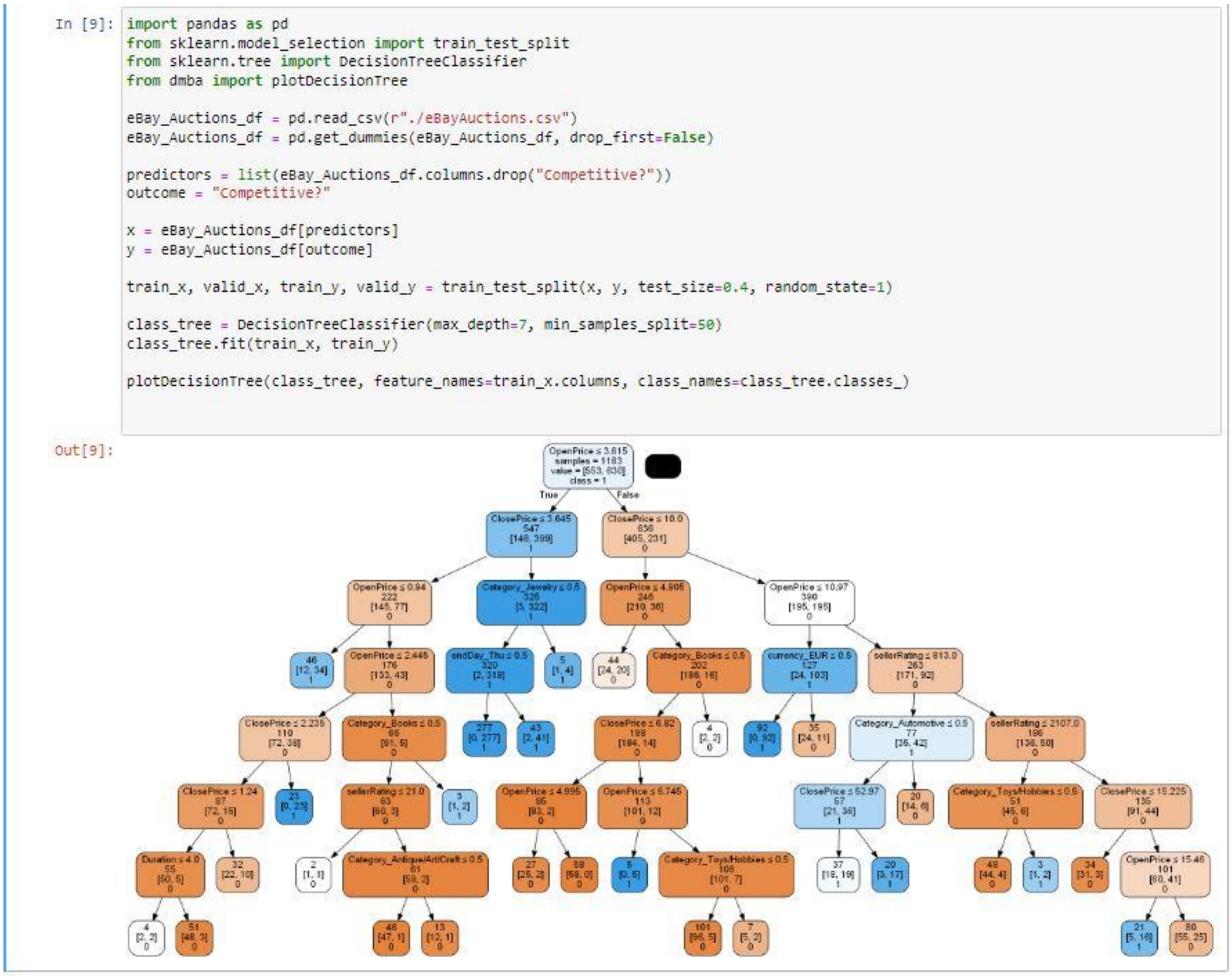


```
In [13]: import pandas as pd
         from sklearn.model_selection import train_test_split
         from sklearn.neighbors import KNeighborsRegressor
         from dmba import classificationSummary

         Universal_bank_df = pd.read_csv(r"./UniversalBank.csv")
         Universal_bank_df.drop(columns=["ID", "ZIP Code"], inplace=True)
         Universal_bank_df = pd.get_dummies(Universal_bank_df, drop_first=False)

         predictors = Universal_bank_df.columns.drop("Personal Loan")
         outcome = "Personal Loan"
         cutoff = 0.5
         x = Universal_bank_df[predictors]
         y = Universal_bank_df[outcome]

         train_x, temp_x, train_y, temp_y = train_test_split(x, y, test_size=0.5, random_state=1)
         valid_x, test_x, valid_y, test_y = train_test_split(temp_x, temp_y, test_size=0.4, random_state=1)

         knn = KNeighborsRegressor(n_neighbors=29)
         knn.fit(train_x, train_y)

         print("training_data confusion matrix ...")
         predicted_values = [0 if pred < cutoff else 1 for pred in knn.predict(train_x)]
         classificationSummary(train_y, predicted_values)

         print("\n\n")
         print("validation_data confusion matrix ...")
         predicted_values = [0 if pred < cutoff else 1 for pred in knn.predict(valid_x)]
         classificationSummary(valid_y, predicted_values)

         print("\n\n")
         print("test_data confusion matrix ...")
         predicted_values = [0 if pred < cutoff else 1 for pred in knn.predict(test_x)]
         classificationSummary(test_y, predicted_values)
```

```
training_data confusion matrix ...
Confusion Matrix (Accuracy 0.9112)

       Prediction
Actual    0    1
     0 2231   28
     1  194   47



validation_data confusion matrix ...
Confusion Matrix (Accuracy 0.9073)

       Prediction
Actual    0    1
     0 1329   20
     1  119   32



test_data confusion matrix ...
Confusion Matrix (Accuracy 0.9120)

       Prediction
Actual   0   1
     0 899  13
     1  75  13
```

۳)

الف)

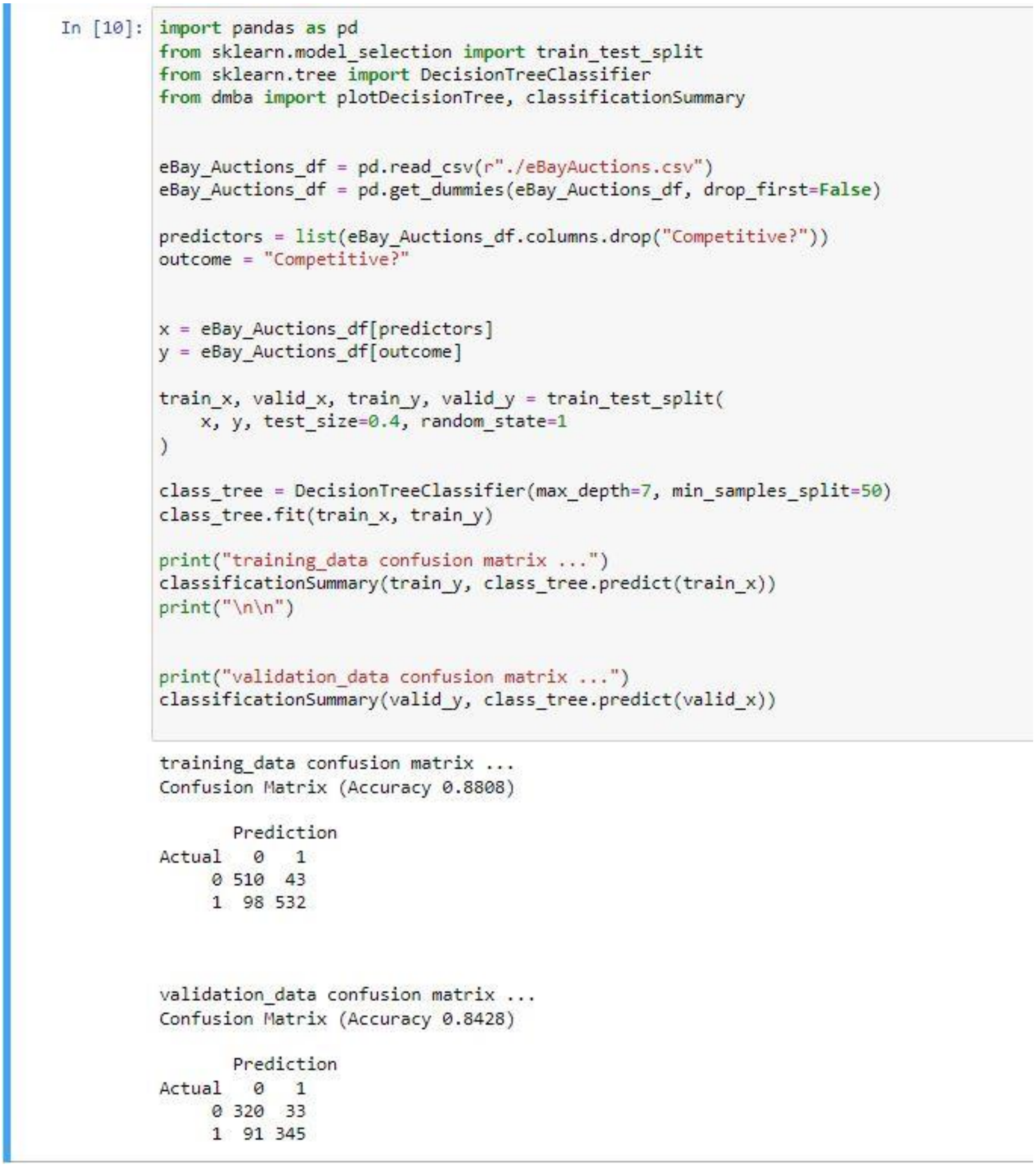


```
In [9]: import pandas as pd
        from sklearn.model_selection import train_test_split
        from sklearn.tree import DecisionTreeClassifier
        from dmba import plotDecisionTree

        eBay_Auctions_df = pd.read_csv(r"./eBayAuctions.csv")
        eBay_Auctions_df = pd.get_dummies(eBay_Auctions_df, drop_first=False)

        predictors = list(eBay_Auctions_df.columns.drop("Competitive?"))
        outcome = "Competitive?"

        x = eBay_Auctions_df[predictors]
        y = eBay_Auctions_df[outcome]

        train_x, valid_x, train_y, valid_y = train_test_split(x, y, test_size=0.4, random_state=1)

        class_tree = DecisionTreeClassifier(max_depth=7, min_samples_split=50)
        class_tree.fit(train_x, train_y)

        plotDecisionTree(class_tree, feature_names=train_x.columns, class_names=class_tree.classes_)
```

Out[9]:

OpenPrice ≤ 3.615 samples = 1183 value = [553, 630] class = 1
[redacted]
True
False
ClosePrice ≤ 3.645 547 [148, 399] 1
ClosePrice ≤ 10.0 636 [405, 231] 0
OpenPrice ≤ 0.94 222 [145, 77] 0
Category_Jewelry ≤ 0.5 325 [3, 322] 1
OpenPrice ≤ 4.905 246 [210, 36] 0
OpenPrice ≤ 10.97 390 [195, 195] 0
46 [12, 34] 1
OpenPrice ≤ 2.445 176 [133, 43] 0
endDay_Thu ≤ 0.5 320 [2, 318] 1
5 [1, 4] 1
44 [24, 20] 0
Category_Books ≤ 0.5 202 [186, 16] 0
currency_EUR ≤ 0.5 127 [24, 103] 1
sellerRating ≤ 813.0 263 [171, 92] 0
ClosePrice ≤ 2.235 110 [72, 38] 0
Category_Books ≤ 0.5 66 [61, 5] 0
277 [0, 277] 1
43 [2, 41] 1
ClosePrice ≤ 6.82 198 [184, 14] 0
4 [2, 2] 0
92 [0, 92] 1
35 [24, 11] 0
Category_Automotive ≤ 0.5 77 [35, 42] 1
sellerRating ≤ 2107.0 186 [136, 50] 0
ClosePrice ≤ 1.24 87 [72, 15] 0
23 [0, 23] 1
sellerRating ≤ 21.0 63 [60, 3] 0
3 [1, 2] 1
OpenPrice ≤ 4.995 85 [83, 2] 0
OpenPrice ≤ 6.745 113 [101, 12] 0
ClosePrice ≤ 52.97 57 [21, 36] 1
20 [14, 6] 0
Category_Toys/Hobbies ≤ 0.5 51 [45, 6] 0
ClosePrice ≤ 15.225 135 [91, 44] 0
Duration ≤ 4.0 55 [50, 5] 0
32 [22, 10] 0
2 [1, 1] 0
Category_Antique/Art/Craft ≤ 0.5 61 [59, 2] 0
27 [25, 2] 0
58 [58, 0] 0
5 [0, 5] 1
Category_Toys/Hobbies ≤ 0.5 108 [101, 7] 0
37 [18, 19] 1
20 [3, 17] 1
48 [44, 4] 0
3 [1, 2] 1
34 [31, 3] 0
OpenPrice ≤ 15.46 101 [60, 41] 0
4 [2, 2] 0
51 [48, 3] 0
48 [47, 1] 0
13 [12, 1] 0
101 [96, 5] 0
7 [5, 2] 0
21 [5, 16] 1
80 [55, 25] 0

ب) طبق شکل زیر میزان دقت برای داده‌های آموزشی و اعتبارسنجی به ترتیب **88** و **84** درصد می‌باشد که نرخ قابل قبولی نیست و می‌توان درختی با عملکرد بهتری ساخت.



```
In [10]: import pandas as pd
         from sklearn.model_selection import train_test_split
         from sklearn.tree import DecisionTreeClassifier
         from dmba import plotDecisionTree, classificationSummary


         eBay_Auctions_df = pd.read_csv(r"./eBayAuctions.csv")
         eBay_Auctions_df = pd.get_dummies(eBay_Auctions_df, drop_first=False)

         predictors = list(eBay_Auctions_df.columns.drop("Competitive?"))
         outcome = "Competitive?"


         x = eBay_Auctions_df[predictors]
         y = eBay_Auctions_df[outcome]

         train_x, valid_x, train_y, valid_y = train_test_split(
             x, y, test_size=0.4, random_state=1
         )

         class_tree = DecisionTreeClassifier(max_depth=7, min_samples_split=50)
         class_tree.fit(train_x, train_y)

         print("training_data confusion matrix ...")
         classificationSummary(train_y, class_tree.predict(train_x))
         print("\n\n")


         print("validation_data confusion matrix ...")
         classificationSummary(valid_y, class_tree.predict(valid_x))
```

```
training_data confusion matrix ...
Confusion Matrix (Accuracy 0.8808)

       Prediction
Actual   0   1
     0 510  43
     1  98 532



validation_data confusion matrix ...
Confusion Matrix (Accuracy 0.8428)

       Prediction
Actual   0   1
     0 320  33
     1  91 345
```

IF (OpenPrice <= 0.94) AND (ClosePrice <=3.645) THEN Class=0

IF (OpenPrice <= 2.445) AND (OpenPrice > 0.94) AND (ClosePrice <= 1.24) AND (Duration > 4) THEN Class=0

IF (OpenPrice <= 2.445) AND (OpenPrice > 0.94)AND (ClosePrice <= 2.235) AND (ClosePrice > 1.24) THEN Class=0

IF (OpenPrice <= 2.445) AND (OpenPrice > 0.94)AND (ClosePrice <= 3.645) AND (ClosePrice > 2.235) THEN Class=1

IF (OpenPrice <=3.615) AND (OpenPrice > 2.445)AND (ClosePrice <= 3.645) AND (Category_Books <= 0.5) AND (sellerRating >21.0) AND (endDay_fri <= 0.5) THEN Class=0

IF (OpenPrice <=3.615) AND (OpenPrice > 2.445)AND (ClosePrice <= 3.645) AND (Category_Books <= 0.5) AND (sellerRating >21.0) AND (endDay_fri > 0.5) THEN Class=0

IF (OpenPrice <=3.615) AND (OpenPrice > 2.445)AND (ClosePrice <= 3.645) AND (Category_Books > 0.5) THEN Class=1

IF (OpenPrice <=3.615) AND (ClosePrice > 3.645) AND (Category_Jewerly <= 0.5) AND (endDay_Thu <= 0.5) THEN Class=1

IF (OpenPrice <=3.615) AND (ClosePrice > 3.645) AND (Category_Jewerly <= 0.5) AND (endDay_Thu > 0.5) THEN Class=1

IF (OpenPrice <=3.615) AND (ClosePrice > 3.645) AND (Category_Jewerly > 0.5) THEN Class=1

IF (OpenPrice > 3.615) AND (ClosePrice <= 6.82) AND (Category_Books <=0.5) THEN Class=0

IF (OpenPrice > 3.615) AND (ClosePrice <= 10.0) AND (Category_Books <=0.5) AND (Close_Price > 6.82) AND (Open_Price <= 6.745)THEN Class=1

IF (OpenPrice > 3.615) AND (ClosePrice <= 10.0) AND (Category_Books <=0.5) AND (Close_Price > 6.82) THEN Class=0

IF (OpenPrice > 3.615) AND (ClosePrice > 10.0) AND (OpenPrice <= 10.97) AND (currency_EUR <= 0.5) THEN Class=1

IF (OpenPrice > 3.615) AND (ClosePrice > 10.0) AND (OpenPrice <= 10.97) AND (currency_EUR > 0.5) THEN Class=0

IF (OpenPrice > 3.615) AND (ClosePrice > 10.0) AND (sellerRating <= 813.0) AND (Category_Automative <=0.5) THEN Class=1

IF (OpenPrice > 3.615) AND (ClosePrice > 10.0) AND (sellerRating <= 813.0) AND (Category_Automative >0.5) THEN Class=0

IF (OpenPrice > 3.615) AND (ClosePrice > 10.0) AND (sellerRating > 813.0) AND (sellerRating <= 2107.0) AND (Category_Toys <= 0.5) THEN Class=0

IF (OpenPrice > 3.615) AND (ClosePrice > 10.0) AND (sellerRating > 813.0) AND (sellerRating <= 2107.0) AND (Category_Toys > 0.5) THEN Class=1

IF (OpenPrice > 3.615) AND (ClosePrice > 10.0) AND (sellerRating > 813.0) AND (ClosePrice <= 15.225) THEN Class=0

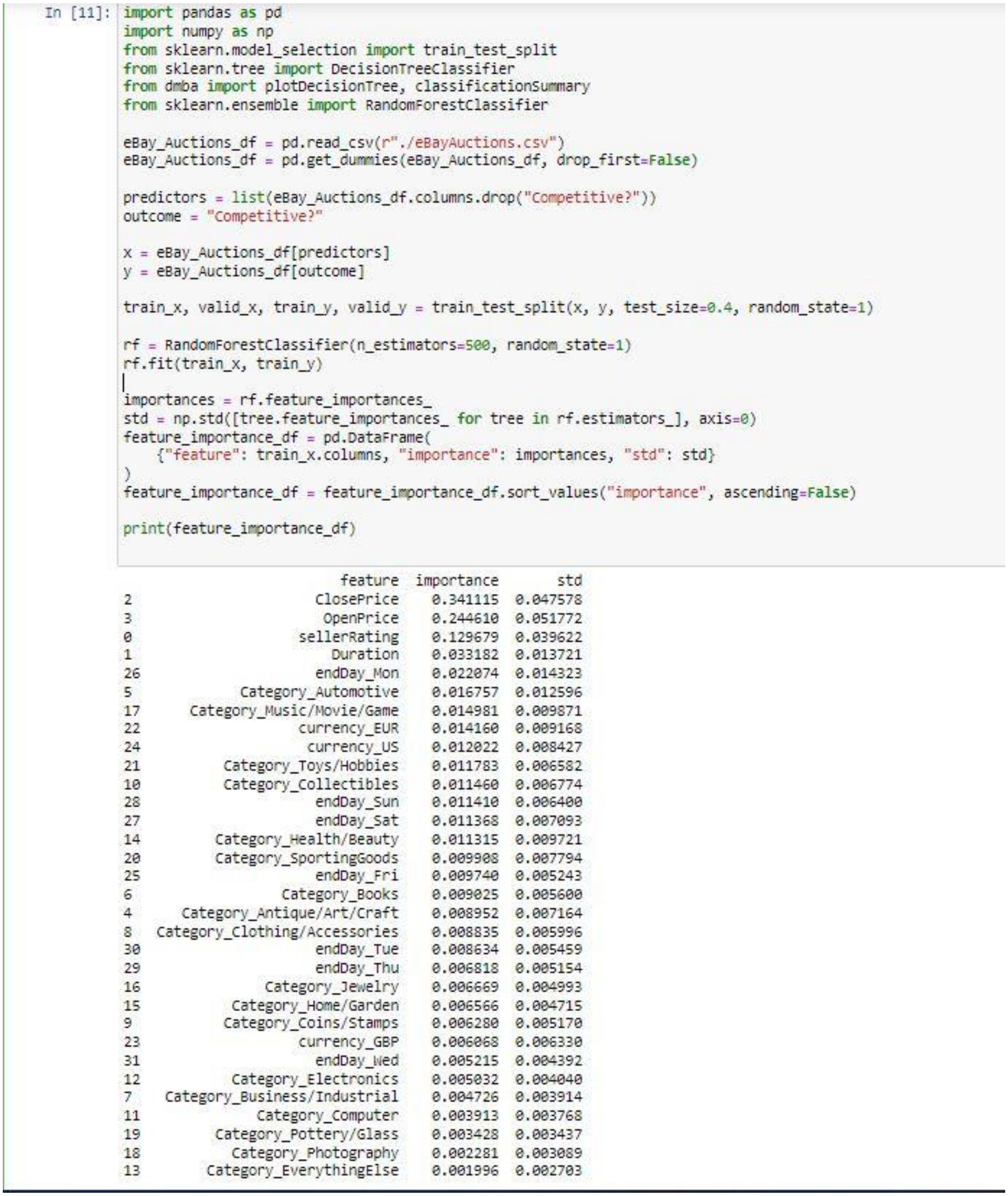
IF (OpenPrice > 3.615) AND (ClosePrice > 10.0) AND (sellerRating > 813.0) AND (OpenPrice <=15.46) THEN Class=1

IF (OpenPrice > 3.615) AND (ClosePrice > 10.0) AND (sellerRating > 813.0) THEN Class=0

ج) با توجه به درخت چند قانون زیر وجود دارد:

- اگر قیمت پایانی کمتر از **3.645** و قیمت شروع بیشتر از **0.94** و کمتر از**3.615** باشد اکثر کالاهای حراجی رقابتی نبوده‌اند.
- اگر قیمت شروع کمتر از **3.615** و قیمت نهایی بیشتر از **3.645** باشد آنگاه تقریبا تمام کالاها رقابتی بوده‌اند.
- اگر قیمت شروع بیشتر از **3.615** و قیمت نهایی کمتر از **10** باشد آنگاه تقریبا تمام کالاها رقابتی نبوده‌اند.

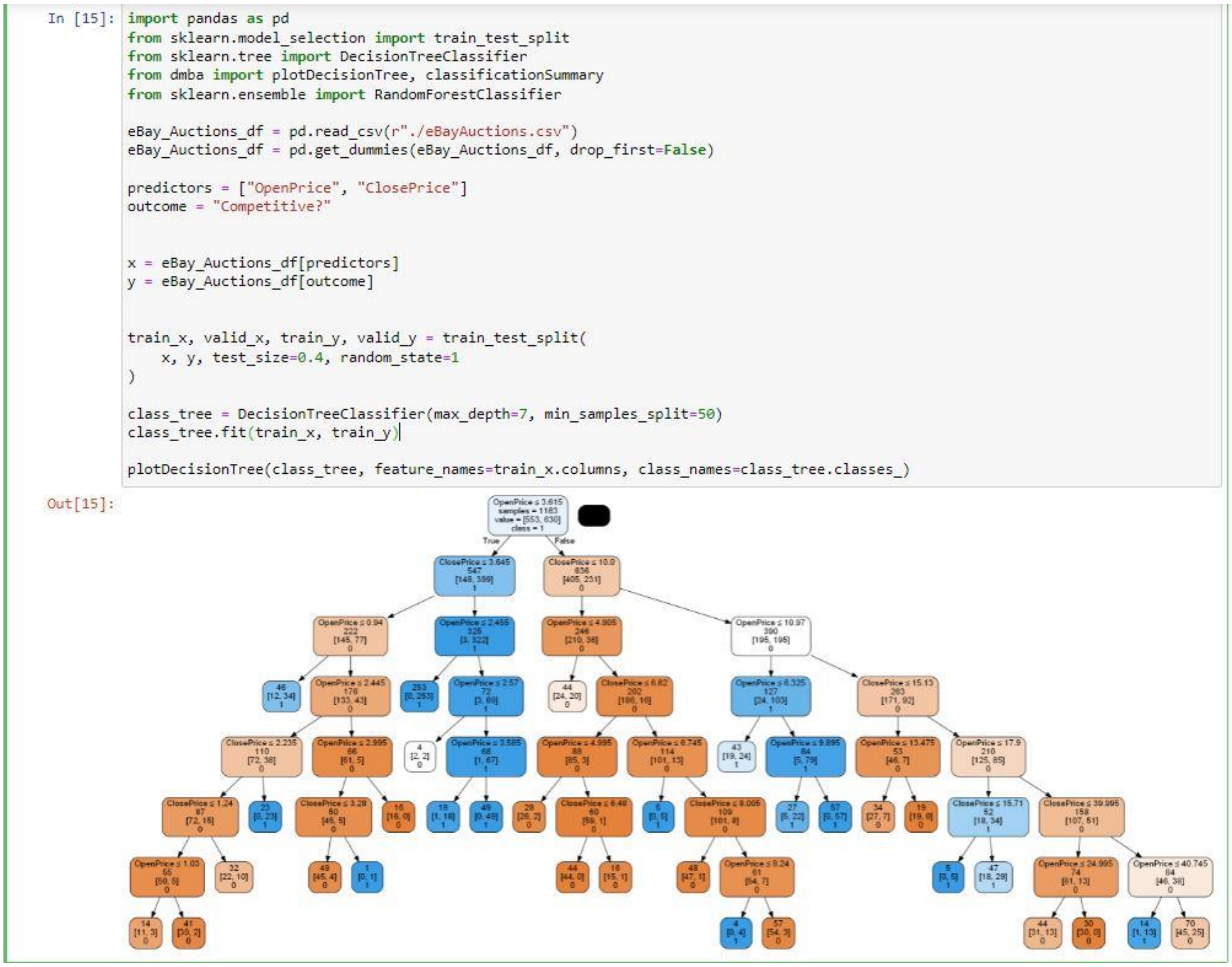
د) با استفاده از الگوریتم درخت‌های تصادفی متغیرهای که بیشترین تفکیک را ایجاد می‌کنند شناسایی کرده و سپس با استفاده از آن متغیرها دوباره درخت تصمیم خود را ایجاد می‌کنیم.



```
In [11]: import pandas as pd
         import numpy as np
         from sklearn.model_selection import train_test_split
         from sklearn.tree import DecisionTreeClassifier
         from dmba import plotDecisionTree, classificationSummary
         from sklearn.ensemble import RandomForestClassifier

         eBay_Auctions_df = pd.read_csv(r"./eBayAuctions.csv")
         eBay_Auctions_df = pd.get_dummies(eBay_Auctions_df, drop_first=False)

         predictors = list(eBay_Auctions_df.columns.drop("Competitive?"))
         outcome = "Competitive?"

         x = eBay_Auctions_df[predictors]
         y = eBay_Auctions_df[outcome]

         train_x, valid_x, train_y, valid_y = train_test_split(x, y, test_size=0.4, random_state=1)

         rf = RandomForestClassifier(n_estimators=500, random_state=1)
         rf.fit(train_x, train_y)

         importances = rf.feature_importances_
         std = np.std([tree.feature_importances_ for tree in rf.estimators_], axis=0)
         feature_importance_df = pd.DataFrame(
             {"feature": train_x.columns, "importance": importances, "std": std}
         )
         feature_importance_df = feature_importance_df.sort_values("importance", ascending=False)

         print(feature_importance_df)
```

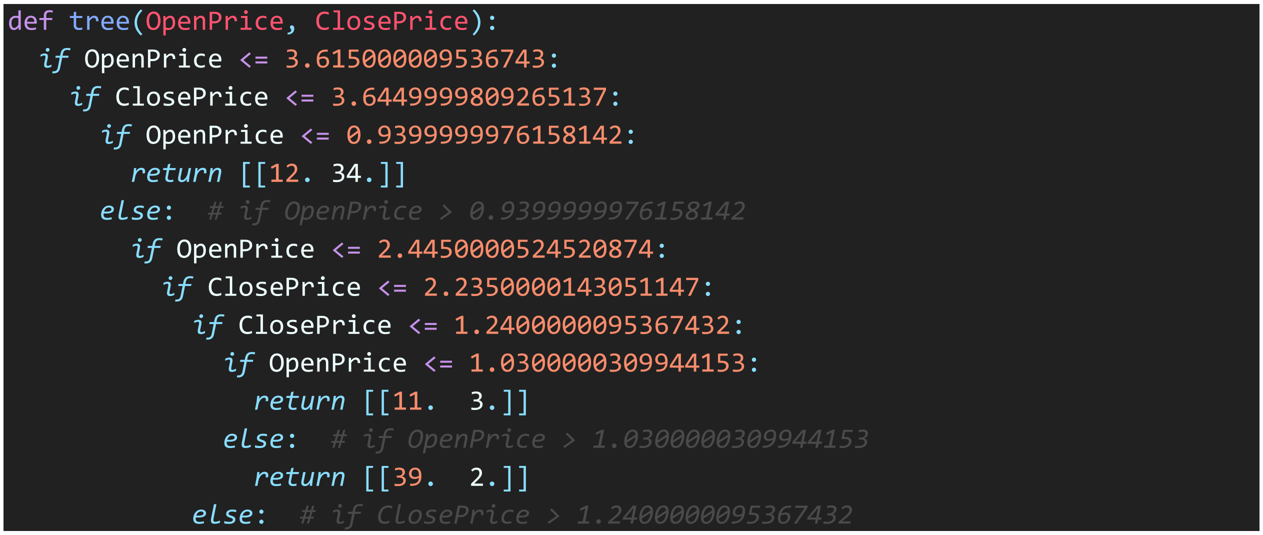
```
                          feature  importance       std
2                      ClosePrice    0.341115  0.047578
3                       OpenPrice    0.244610  0.051772
0                    sellerRating    0.129679  0.039622
1                        Duration    0.033182  0.013721
26                     endDay_Mon    0.022074  0.014323
5             Category_Automotive    0.016757  0.012596
17      Category_Music/Movie/Game    0.014981  0.009871
22                   currency_EUR    0.014160  0.009168
24                    currency_US    0.012022  0.008427
21          Category_Toys/Hobbies    0.011783  0.006582
10          Category_Collectibles    0.011460  0.006774
28                     endDay_Sun    0.011410  0.006400
27                     endDay_Sat    0.011368  0.007093
14         Category_Health/Beauty    0.011315  0.009721
20         Category_SportingGoods    0.009908  0.007794
25                     endDay_Fri    0.009740  0.005243
6                  Category_Books    0.009025  0.005600
4      Category_Antique/Art/Craft    0.008952  0.007164
8   Category_Clothing/Accessories    0.008835  0.005996
30                     endDay_Tue    0.008634  0.005459
29                     endDay_Thu    0.006818  0.005154
16               Category_Jewelry    0.006669  0.004993
15           Category_Home/Garden    0.006566  0.004715
9           Category_Coins/Stamps    0.006280  0.005170
23                   currency_GBP    0.006068  0.006330
31                     endDay_Wed    0.005215  0.004392
12           Category_Electronics    0.005032  0.004040
7    Category_Business/Industrial    0.004726  0.003914
11              Category_Computer    0.003913  0.003768
19         Category_Pottery/Glass    0.003428  0.003437
18           Category_Photography    0.002281  0.003089
13        Category_EverythingElse    0.001996  0.002703
```

همانطور که می‌بینیم متغیرهای **ClosedPrice, OpenPrice** بیشترین اهمیت را دارند.

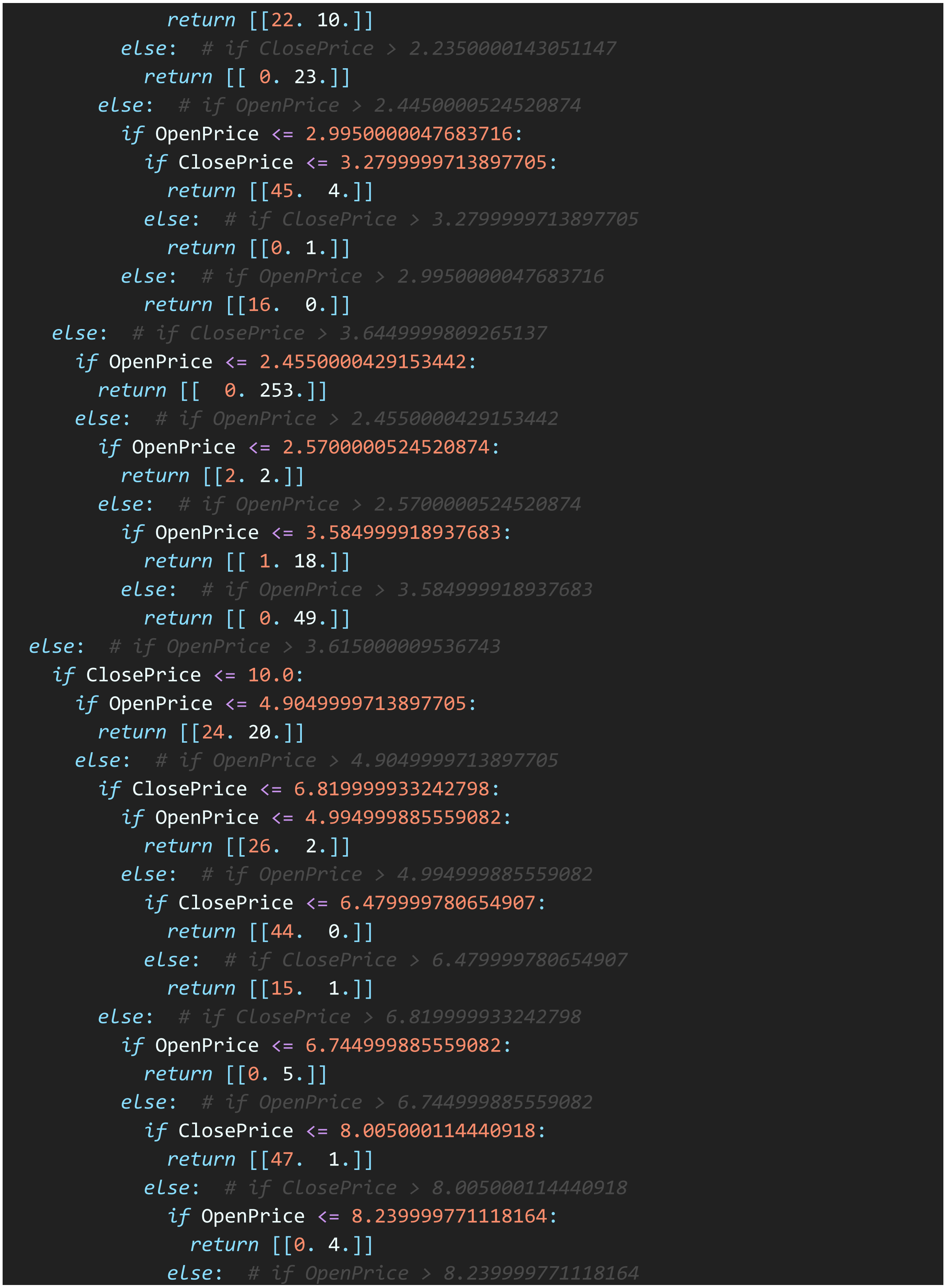


```
In [15]: import pandas as pd
         from sklearn.model_selection import train_test_split
         from sklearn.tree import DecisionTreeClassifier
         from dmba import plotDecisionTree, classificationSummary
         from sklearn.ensemble import RandomForestClassifier

         eBay_Auctions_df = pd.read_csv(r"./eBayAuctions.csv")
         eBay_Auctions_df = pd.get_dummies(eBay_Auctions_df, drop_first=False)

         predictors = ["OpenPrice", "ClosePrice"]
         outcome = "Competitive?"


         x = eBay_Auctions_df[predictors]
         y = eBay_Auctions_df[outcome]


         train_x, valid_x, train_y, valid_y = train_test_split(
             x, y, test_size=0.4, random_state=1
         )

         class_tree = DecisionTreeClassifier(max_depth=7, min_samples_split=50)
         class_tree.fit(train_x, train_y)

         plotDecisionTree(class_tree, feature_names=train_x.columns, class_names=class_tree.classes_)

Out[15]:
```

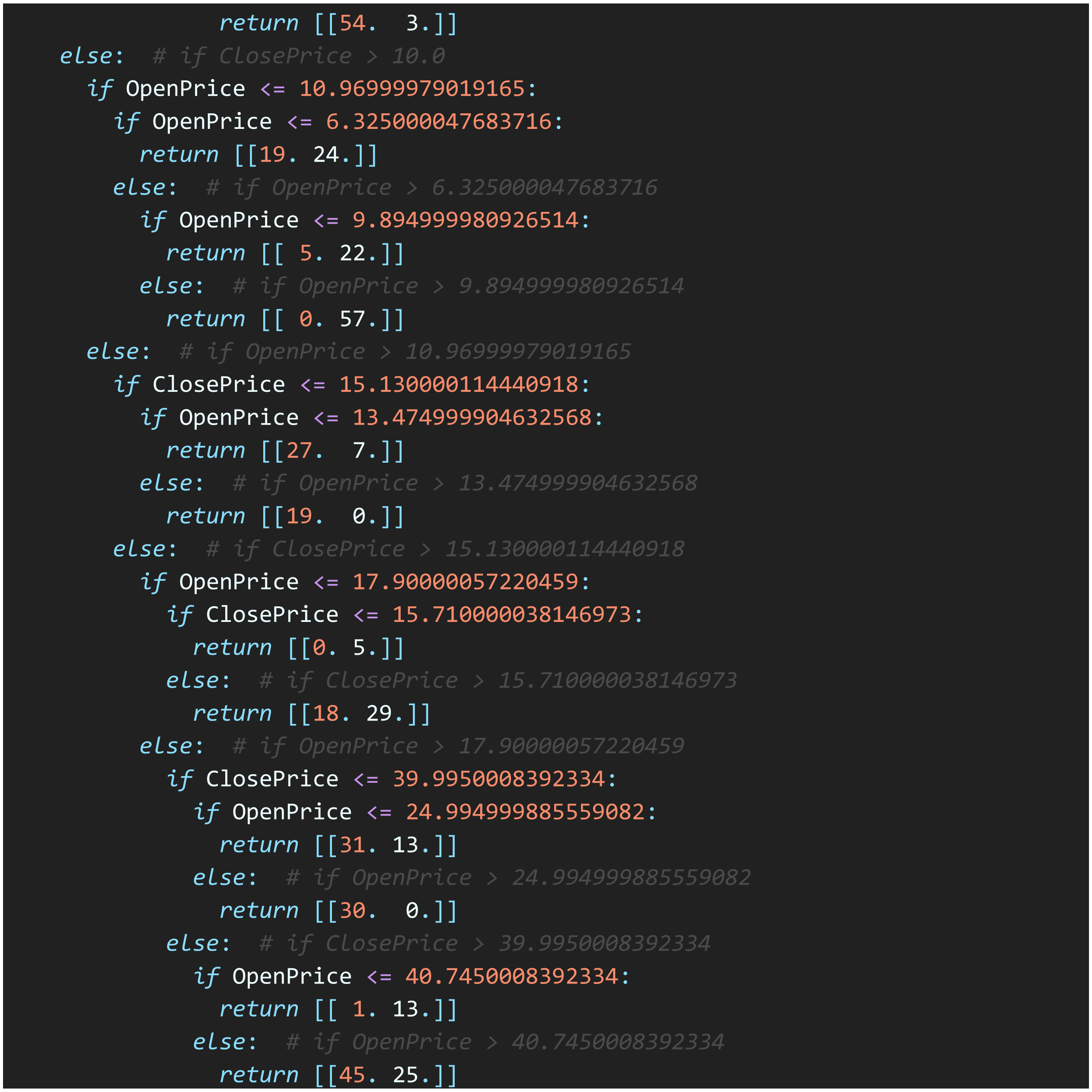
OpenPrice ≤ 3.615 samples = 1183 value = [553, 630] class = 1
[redacted]
True
False
ClosePrice ≤ 3.645 547 [148, 399] 1
ClosePrice ≤ 10.0 636 [405, 231] 0
OpenPrice ≤ 0.94 222 [145, 77] 0
OpenPrice ≤ 2.455 325 [3, 322] 1
OpenPrice ≤ 4.905 246 [210, 36] 0
OpenPrice ≤ 10.97 390 [195, 195] 0
46 [12, 34] 1
OpenPrice ≤ 2.445 176 [133, 43] 0
253 [0, 253] 1
OpenPrice ≤ 2.57 72 [3, 69] 1
44 [24, 20] 0
ClosePrice ≤ 6.82 202 [186, 16] 0
OpenPrice ≤ 6.325 127 [24, 103] 1
ClosePrice ≤ 15.13 263 [171, 92] 0
ClosePrice ≤ 2.235 110 [72, 38] 0
OpenPrice ≤ 2.995 66 [61, 5] 0
4 [2, 2] 0
OpenPrice ≤ 3.585 68 [1, 67] 1
OpenPrice ≤ 4.995 88 [85, 3] 0
OpenPrice ≤ 6.745 114 [101, 13] 0
43 [19, 24] 1
OpenPrice ≤ 9.895 84 [5, 79] 1
OpenPrice ≤ 13.475 53 [46, 7] 0
OpenPrice ≤ 17.9 210 [125, 85] 0
ClosePrice ≤ 1.24 87 [72, 15] 0
23 [0, 23] 1
ClosePrice ≤ 3.28 50 [45, 5] 0
16 [16, 0] 0
19 [1, 18] 1
49 [0, 49] 1
28 [26, 2] 0
ClosePrice ≤ 6.48 60 [59, 1] 0
5 [0, 5] 1
ClosePrice ≤ 8.005 109 [101, 8] 0
27 [5, 22] 1
57 [0, 57] 1
34 [27, 7] 0
19 [19, 0] 0
ClosePrice ≤ 15.71 52 [18, 34] 1
ClosePrice ≤ 39.995 158 [107, 51] 0
OpenPrice ≤ 1.03 55 [50, 5] 0
32 [22, 10] 0
49 [45, 4] 0
1 [0, 1] 1
44 [44, 0] 0
16 [15, 1] 0
48 [47, 1] 0
OpenPrice ≤ 8.24 61 [54, 7] 0
5 [0, 5] 1
47 [18, 29] 1
OpenPrice ≤ 24.995 74 [61, 13] 0
OpenPrice ≤ 40.745 84 [46, 38] 0
14 [11, 3] 0
41 [39, 2] 0
4 [0, 4] 1
57 [54, 3] 0
44 [31, 13] 0
30 [30, 0] 0
14 [1, 13] 1
70 [45, 25] 0



```
def tree(OpenPrice, ClosePrice):
  if OpenPrice <= 3.615000009536743:
    if ClosePrice <= 3.6449999809265137:
      if OpenPrice <= 0.9399999976158142:
        return [[12. 34.]]
      else:  # if OpenPrice > 0.9399999976158142
        if OpenPrice <= 2.4450000524520874:
          if ClosePrice <= 2.2350000143051147:
            if ClosePrice <= 1.2400000095367432:
              if OpenPrice <= 1.0300000309944153:
                return [[11.  3.]]
              else:  # if OpenPrice > 1.0300000309944153
                return [[39.  2.]]
            else:  # if ClosePrice > 1.2400000095367432
```

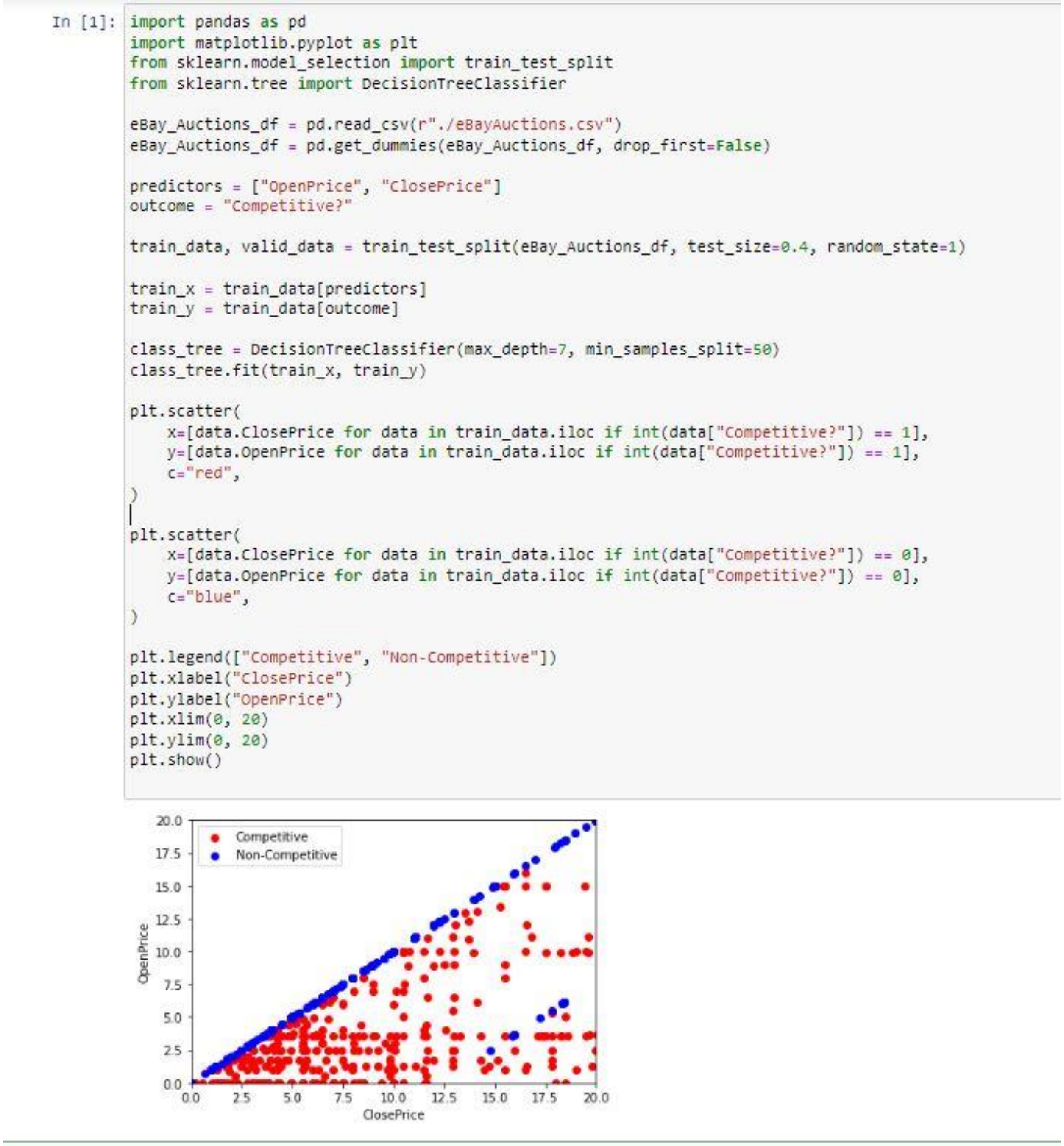


```
            return [[22. 10.]]
        else:  # if ClosePrice > 2.2350000143051147
          return [[ 0. 23.]]
      else:  # if OpenPrice > 2.4450000524520874
        if OpenPrice <= 2.9950000047683716:
          if ClosePrice <= 3.2799999713897705:
            return [[45.  4.]]
          else:  # if ClosePrice > 3.2799999713897705
            return [[0. 1.]]
        else:  # if OpenPrice > 2.9950000047683716
          return [[16.  0.]]
  else:  # if ClosePrice > 3.6449999809265137
    if OpenPrice <= 2.4550000429153442:
      return [[  0. 253.]]
    else:  # if OpenPrice > 2.4550000429153442
      if OpenPrice <= 2.5700000524520874:
        return [[2. 2.]]
      else:  # if OpenPrice > 2.5700000524520874
        if OpenPrice <= 3.584999918937683:
          return [[ 1. 18.]]
        else:  # if OpenPrice > 3.584999918937683
          return [[ 0. 49.]]
else:  # if OpenPrice > 3.615000009536743
  if ClosePrice <= 10.0:
    if OpenPrice <= 4.9049999713897705:
      return [[24. 20.]]
    else:  # if OpenPrice > 4.9049999713897705
      if ClosePrice <= 6.819999933242798:
        if OpenPrice <= 4.994999885559082:
          return [[26.  2.]]
        else:  # if OpenPrice > 4.994999885559082
          if ClosePrice <= 6.479999780654907:
            return [[44.  0.]]
          else:  # if ClosePrice > 6.479999780654907
            return [[15.  1.]]
      else:  # if ClosePrice > 6.819999933242798
        if OpenPrice <= 6.744999885559082:
          return [[0. 5.]]
        else:  # if OpenPrice > 6.744999885559082
          if ClosePrice <= 8.005000114440918:
            return [[47.  1.]]
          else:  # if ClosePrice > 8.005000114440918
            if OpenPrice <= 8.239999771118164:
              return [[0. 4.]]
            else:  # if OpenPrice > 8.239999771118164
```



```
                return [[54.  3.]]
    else:  # if ClosePrice > 10.0
      if OpenPrice <= 10.96999979019165:
        if OpenPrice <= 6.325000047683716:
          return [[19. 24.]]
        else:  # if OpenPrice > 6.325000047683716
          if OpenPrice <= 9.894999980926514:
            return [[ 5. 22.]]
          else:  # if OpenPrice > 9.894999980926514
            return [[ 0. 57.]]
      else:  # if OpenPrice > 10.96999979019165
        if ClosePrice <= 15.130000114440918:
          if OpenPrice <= 13.474999904632568:
            return [[27.  7.]]
          else:  # if OpenPrice > 13.474999904632568
            return [[19.  0.]]
        else:  # if ClosePrice > 15.130000114440918
          if OpenPrice <= 17.90000057220459:
            if ClosePrice <= 15.710000038146973:
              return [[0. 5.]]
            else:  # if ClosePrice > 15.710000038146973
              return [[18. 29.]]
          else:  # if OpenPrice > 17.90000057220459
            if ClosePrice <= 39.9950008392334:
              if OpenPrice <= 24.994999885559082:
                return [[31. 13.]]
              else:  # if OpenPrice > 24.994999885559082
                return [[30.  0.]]
            else:  # if ClosePrice > 39.9950008392334
              if OpenPrice <= 40.7450008392334:
                return [[ 1. 13.]]
              else:  # if OpenPrice > 40.7450008392334
                return [[45. 25.]]
```

۵)



```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split
        from sklearn.tree import DecisionTreeClassifier

        eBay_Auctions_df = pd.read_csv(r"./eBayAuctions.csv")
        eBay_Auctions_df = pd.get_dummies(eBay_Auctions_df, drop_first=False)

        predictors = ["OpenPrice", "ClosePrice"]
        outcome = "Competitive?"

        train_data, valid_data = train_test_split(eBay_Auctions_df, test_size=0.4, random_state=1)

        train_x = train_data[predictors]
        train_y = train_data[outcome]

        class_tree = DecisionTreeClassifier(max_depth=7, min_samples_split=50)
        class_tree.fit(train_x, train_y)

        plt.scatter(
            x=[data.ClosePrice for data in train_data.iloc if int(data["Competitive?"]) == 1],
            y=[data.OpenPrice for data in train_data.iloc if int(data["Competitive?"]) == 1],
            c="red",
        )

        plt.scatter(
            x=[data.ClosePrice for data in train_data.iloc if int(data["Competitive?"]) == 0],
            y=[data.OpenPrice for data in train_data.iloc if int(data["Competitive?"]) == 0],
            c="blue",
        )

        plt.legend(["Competitive", "Non-Competitive"])
        plt.xlabel("ClosePrice")
        plt.ylabel("OpenPrice")
        plt.xlim(0, 20)
        plt.ylim(0, 20)
        plt.show()
```