CREDIT CARD FRAUD DETECTION

By Ozlem Kilickaya

CREDIT CARD FRAUD DETECTION

Programming Language: Python

Steps are given as follows.

1. Perform Exploratory Data Analysis (EDA) on our dataset
2. Apply different Machine Learning algorithms to our dataset
3. Train and evaluate our models on the dataset and then, choose the best.

Step 1. Perform Exploratory Data Analysis (EDA)

There are a total of 284,807 transactions with only 492 of them being fraud.

```
import pandas as pd

from collections import Counter

import itertools


# Load the csv file


dataframe = pd.read_csv("./Desktop/DataFlair/credit_card_fraud_detection/creditcard.csv")

dataframe.head()
```

Out[2]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V2
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.1285
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.1671
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.3276
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.6473
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.2060

5 rows × 31 columns

There are no null or NaN values in our dataset.

```
In [5]: dataframe["Amount"].describe()
```

```
Out[5]: count    284807.000000
        mean         88.349619
        std         250.120109
        min           0.000000
        25%           5.600000
        50%          22.000000
        75%          77.165000
        max       25691.160000
        Name: Amount, dtype: float64
```

CREDIT CARD FRAUD DETECTION

By Ozlem Kilickaya

non_fraud = len(dataframe[dataframe.Class == 0])

fraud = len(dataframe[dataframe.Class == 1])

fraud_percent = (fraud / (fraud + non_fraud)) * 100

print("Number of Genuine transactions: ", non_fraud)

print("Number of Fraud transactions: ", fraud)

print("Percentage of Fraud transactions: {:.4f}".format(fraud_percent))

import matplotlib.pyplot as plt

labels = ["Genuine", "Fraud"]

count_classes = dataframe.value_counts(dataframe['Class'], sort= True)

count_classes.plot(kind = "bar", rot = 0)

plt.title("Visualization of Labels")

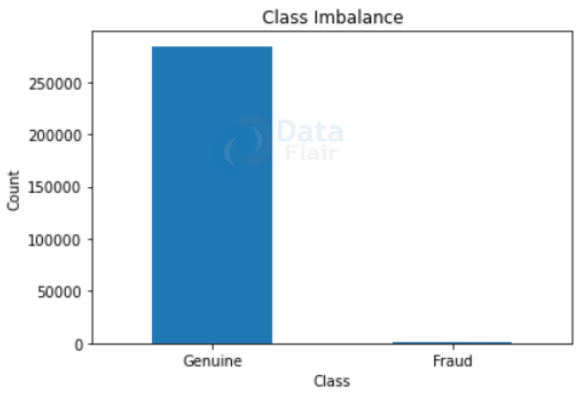
plt.ylabel("Count")

plt.xticks(range(2), labels)

plt.show()

```
In [7]: # Visualize the "Labels" column in our dataset

        labels = ["Genuine", "Fraud"]
        count_classes = dataframe.value_counts(dataframe['Class'], sort= True)
        count_classes.plot(kind = "bar", rot = 0)
        plt.title("Visualization of Labels")
        plt.ylabel("Count")
        plt.xticks(range(2), labels)
        plt.show()
```



CREDIT CARD FRAUD DETECTION

By Ozlem Kilickaya

We can observe that the genuine transactions are over 99%! This is not good.

Therefore, scaling techniques on the “Amount” feature to transform the range of values are applied. The original “Amount” column is dropped and a new column with the scaled values is added. “Time” column is dropped as it is irrelevant.

```
import numpy as np

from sklearn.preprocessing import StandardScaler


scaler = StandardScaler()

dataframe["NormalizedAmount"] =
scaler.fit_transform(dataframe["Amount"].values.reshape(-1, 1))

dataframe.drop(["Amount", "Time"], inplace= True, axis= 1)


Y = dataframe["Class"]

X = dataframe.drop(["Class"], axis= 1)
```

Split credit card data:

```
from sklearn.model_selection import train_test_split


(train_X, test_X, train_Y, test_Y) = train_test_split(X, Y, test_size= 0.3, random_state= 42)


print("Shape of train_X: ", train_X.shape)

print("Shape of test_X: ", test_X.shape)
```

```
Shape of train_X:  (199364, 29)
Shape of test_X:  (85443, 29)
```

CREDIT CARD FRAUD DETECTION

By Ozlem Kilickaya

Step 2. Apply Machine Learning Algorithms to Credit Card Dataset

Random Forest and Decision Tree Classifiers are used. They are present in the sklearn package in the form of RandomForestClassifier() and DecisionTreeClassifier() respectively.

```
from sklearn.ensemble import RandomForestClassifier

from sklearn.tree import DecisionTreeClassifier

#Decision Tree

decision_tree = DecisionTreeClassifier()

# Random Forest

random_forest = RandomForestClassifier(n_estimators= 100)
```

Step 3. Train and Evaluate our Models on the Dataset

```
decision_tree.fit(train_X, train_Y)

predictions_dt = decision_tree.predict(test_X)

decision_tree_score = decision_tree.score(test_X, test_Y) * 100


random_forest.fit(train_X, train_Y)

predictions_rf = random_forest.predict(test_X)

random_forest_score = random_forest.score(test_X, test_Y) * 100


print("Random Forest Score: ", random_forest_score)

print("Decision Tree Score: ", decision_tree_score)
```

CREDIT CARD FRAUD DETECTION

By Ozlem Kilickaya

```
RF:  99.96254813150287
DT:  99.92275552122467
```

The Random Forest classifier has better performance over Decision Tree classifier.

After this, a function is created to print the metrics: accuracy, precision, recall, and f1-score.

from sklearn.metrics import accuracy_score, precision_score, confusion_matrix, recall_score, f1_score

def metrics(actuals, predictions):

    print("Accuracy: {:.5f}".format(accuracy_score(actuals, predictions)))

    print("Precision: {:.5f}".format(precision_score(actuals, predictions)))

    print("Recall: {:.5f}".format(recall_score(actuals, predictions)))

    print("F1-score: {:.5f}".format(f1_score(actuals, predictions)))

Confusion matrix and evaluation metrics are obtained for decision tree model.

confusion_matrix_dt = confusion_matrix(test_Y, predictions_dt.round())

print("Confusion Matrix - Decision Tree")

print(confusion_matrix_dt)

plot_confusion_matrix(confusion_matrix_dt, classes=[0, 1], title= "Confusion Matrix - Decision Tree")

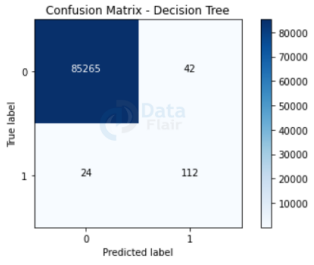
```
In [22]: # Plot confusion matrix for Decision Trees

confusion_matrix_dt = confusion_matrix(test_Y, predictions_dt.round())
print("Confusion Matrix - Decision Tree")
print(confusion_matrix_dt)

Confusion Matrix - Decision Tree
[[85265    42]
 [   24   112]]

In [18]: plot_confusion_matrix(confusion_matrix_dt, classes=[0, 1], title= "Confusion Matrix - Decision Tree")

Confusion matrix, without normalization
```



CREDIT CARD FRAUD DETECTION

By Ozlem Kilickaya

print("Evaluation of Decision Tree Model")

print()

metrics(test_Y, predictions_dt.round())

```
In [25]: print("Evaluation of Decision Tree Model")
         print()
         metrics(test_Y, predictions_dt.round())

         Evaluation of Decision Tree Model

         Accuracy: 0.99923
         Precision: 0.72727
         Recall: 0.82353
         F1-score: 0.77241
```

Confusion matrix and the evaluation metrics of our Random Forest model are obtained.

confusion_matrix_rf = confusion_matrix(test_Y, predictions_rf.round())

print("Confusion Matrix - Random Forest")

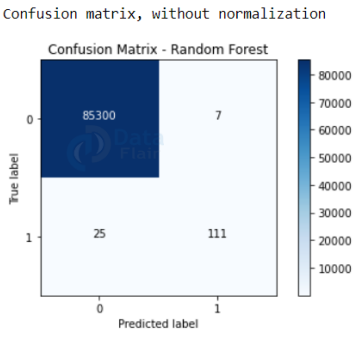
print(confusion_matrix_rf)

plot_confusion_matrix(confusion_matrix_rf, classes=[0, 1], title= "Confusion Matrix - Random Forest")

```
In [20]: # Plot confusion matrix for Random Forests

         confusion_matrix_rf = confusion_matrix(test_Y, predictions_rf.round())
         print("Confusion Matrix - Random Forest")
         print(confusion_matrix_rf)

         Confusion Matrix - Random Forest
         [[85300     7]
          [   25   111]]
```



CREDIT CARD FRAUD DETECTION

By Ozlem Kilickaya

print("Evaluation of Random Forest Model")

print()

metrics(test_Y, predictions_rf.round())

```
In [26]: print("Evaluation of Random Forest Model")
         print()
         metrics(test_Y, predictions_rf.round())

         Evaluation of Random Forest Model

         Accuracy: 0.99963
         Precision: 0.94068
         Recall: 0.81618
         F1-score: 0.87402
```

Class Imbalance Issue

The Random Forest model works better than Decision Trees. However, this dataset has class imbalance problem. The genuine (not fraud) transactions are more than 99% with the credit card fraud transactions constituting 0.17%.

If we train our model without taking care of the imbalance issues, it predicts the label with higher importance given to genuine transactions (as there is more data about them) and hence obtains more accuracy. The class imbalance problem can be solved by various techniques. Oversampling is one of them.

Oversample the minority class is one of the approaches to address the imbalanced datasets. The easiest solution entails doubling examples in the minority class, even though these examples contribute no new data to the model.

Instead, new examples may be generated by replicating existing ones. The Synthetic Minority Oversampling Technique, or SMOTE for short, is a method of data augmentation for the minority class.

It is present in the imblearn package.

CREDIT CARD FRAUD DETECTION

By Ozlem Kilickaya

```
from imblearn.over_sampling import SMOTE

X_resampled, Y_resampled = SMOTE().fit_resample(X, Y)

print("Resampled shape of X: ", X_resampled.shape)

print("Resampled shape of Y: ", Y_resampled.shape)

value_counts = Counter(Y_resampled)

print(value_counts)

(train_X, test_X, train_Y, test_Y) = train_test_split(X_resampled, Y_resampled, test_size= 0.3, random_state= 42)
```

As the Random Forest algorithm performed better than the Decision Tree algorithm, we will apply the Random Forest algorithm to our resampled data.

```
rf_resampled = RandomForestClassifier(n_estimators = 100)

rf_resampled.fit(train_X, train_Y)

predictions_resampled = rf_resampled.predict(test_X)

random_forest_score_resampled = rf_resampled.score(test_X, test_Y) * 100
```

Let’s visualize the predictions of our model and plot the confusion matrix.

```
cm_resampled = confusion_matrix(test_Y, y_predict.round())

print("Confusion Matrix - Random Forest")

print(cm_resampled)

plot_confusion_matrix(cm_resampled, classes=[0, 1], title= "Confusion Matrix - Random Forest After Oversampling")
```

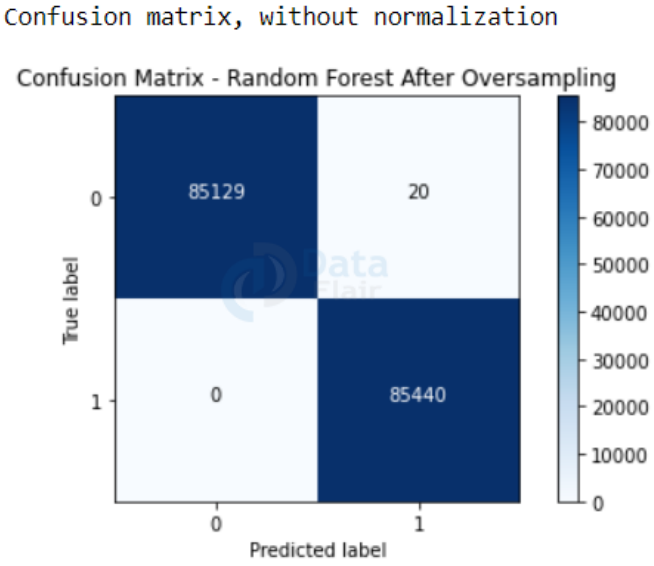
CREDIT CARD FRAUD DETECTION

By Ozlem Kilickaya

```
In [36]: # Visualize the confusion matrix

cm_resampled = confusion_matrix(test_Y, y_predict.round())
print("Confusion Matrix - Random Forest")
print(cm_resampled)
```

```
Confusion Matrix - Random Forest
[[85129    20]
 [    0 85440]]
```



print("Evaluation of Random Forest Model")

print()

metrics(test_Y, predictions_resampled.round())

```
Evaluation of Random Forest Model

Accuracy: 0.99989
Precision: 0.99978
Recall: 1.00000
F1-score: 0.99989
```

Now, it is clearly evident that our model performed much better than our previous Random Forest classifier without oversampling.