Credit Card Fraud Detection with Isolation Forest
Detailed Code Report

Dhruv Patel

Introduction

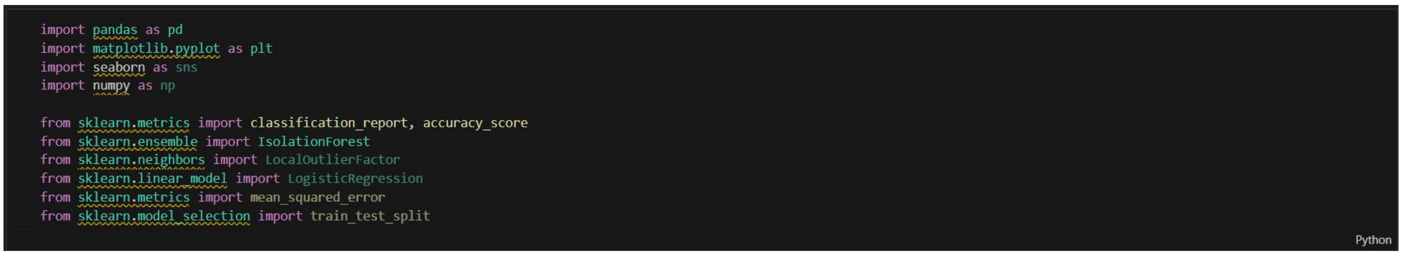
Credit card fraud is one of the most critical problem in any industry. The problem of credit card fraud affects both consumers and financial organizations. To avoid financial losses, it's essential to spot fraudulent transactions. Using Python code with isolation forest and other few important libraries, we will detect the credit card fraud.

Structure of Code

There are different parts of the code which are-

1. **Importing Libraries:**

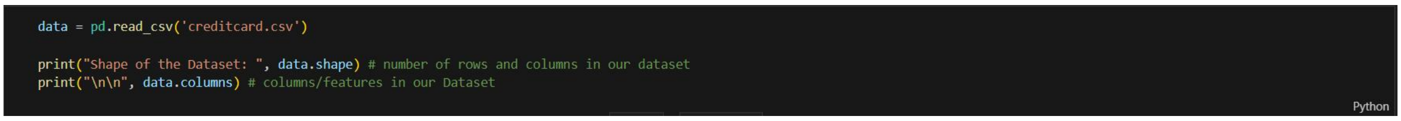
I imported libraries like numpy, pandas, matplotlib, seaborn which are useful for the data visualization and numerical operations. I also used sklearn library for using model evaluation metric, data processing and ensemble methods.



```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

from sklearn.metrics import classification_report, accuracy_score
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
```
Python

2. **Loading Data Set:**

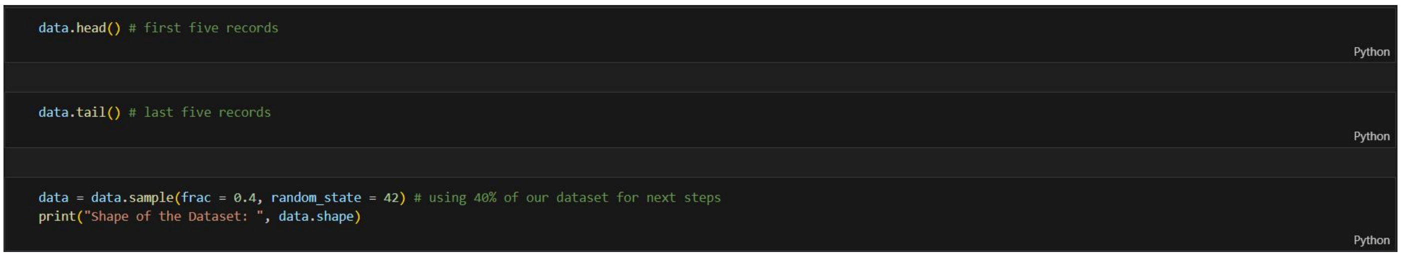
There is already a data set for credit card transactions is available online. I loaded the **creditcard.csv** data using Pandas DataFrame using **pd.read_csv**.



```python
data = pd.read_csv('creditcard.csv')

print("Shape of the Dataset: ", data.shape) # number of rows and columns in our dataset
print("\n\n", data.columns) # columns/features in our Dataset
```
Python

3. **Data Analysis:**

The dataset's shape (number of rows and columns) and column names are printed by the code. The first five and last five records are shown via the **data.head() and data.tail()** functions, respectively. To speed up processing, data is down-sampled so that just **40% of the dataset (data.sample) is used.**



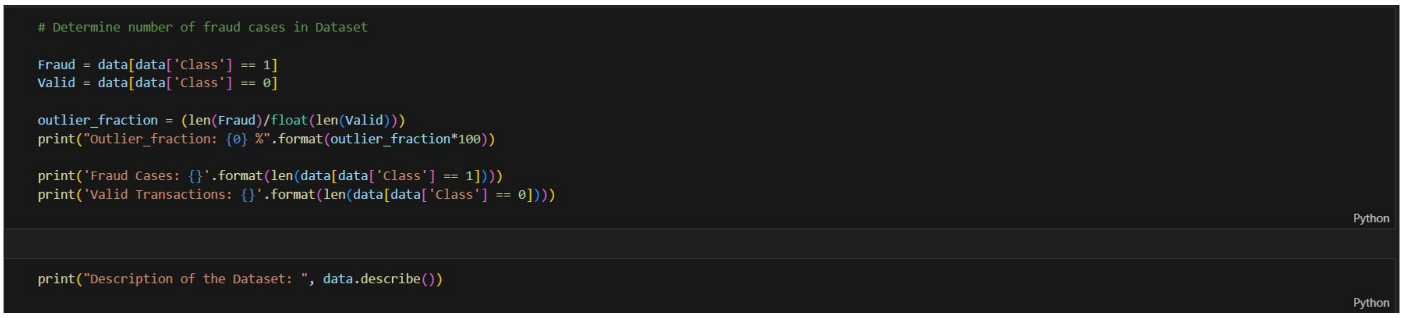
```python
data.head() # first five records
```
Python

```python
data.tail() # last five records
```
Python

```python
data = data.sample(frac = 0.4, random_state = 42) # using 40% of our dataset for next steps
print("Shape of the Dataset: ", data.shape)
```
Python

4. **Fraud and Valid Transactions:**

The 'Class' column is used to calculate the number of valid transactions and fraudulent transactions. The ratio of fraud instances to legitimate cases is represented by the outlier fraction, which is calculated.

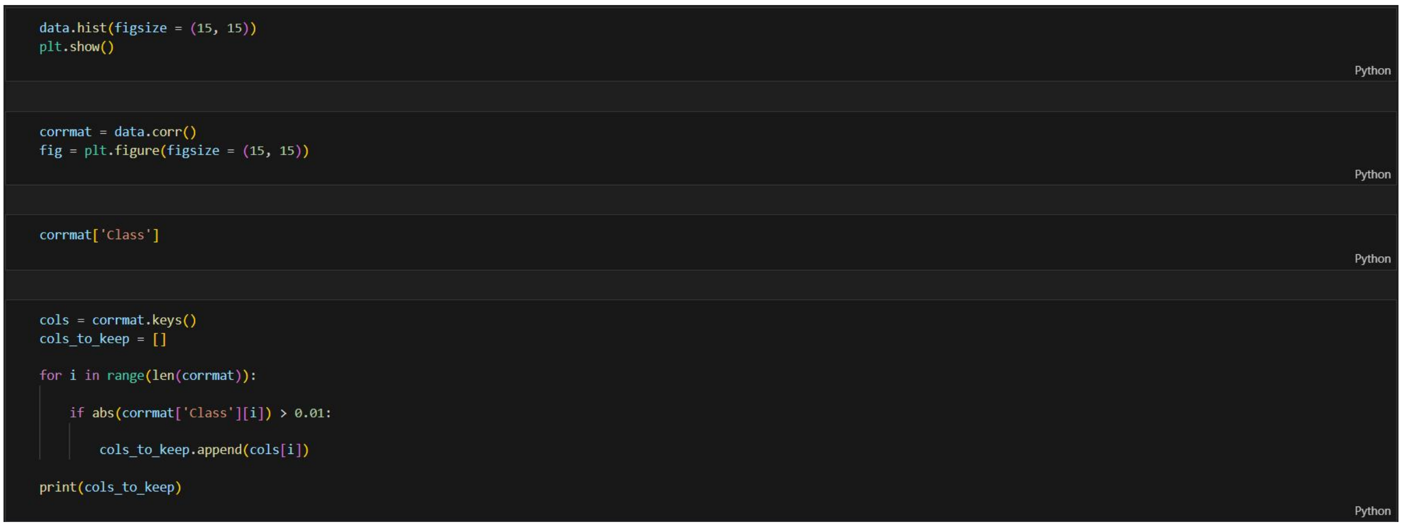


```python
# Determine number of fraud cases in Dataset

Fraud = data[data['Class'] == 1]
Valid = data[data['Class'] == 0]

outlier_fraction = (len(Fraud)/float(len(Valid)))
print("Outlier_fraction: {0} %".format(outlier_fraction*100))

print('Fraud Cases: {}'.format(len(data[data['Class'] == 1])))
print('Valid Transactions: {}'.format(len(data[data['Class'] == 0])))
```

```python
print("Description of the Dataset: ", data.describe())
```

5. Data Visualization:

The algorithm creates histograms for various features using **data.hist()** to visualize data distribution. A **correlation matri**x, or correlation, is produced to comprehend how different variables relate to one another. Features chosen for further analysis have a correlation coefficient with the 'Class' column greater than **0.01 (or less than -0.01).**



```python
data.hist(figsize = (15, 15))
plt.show()
```

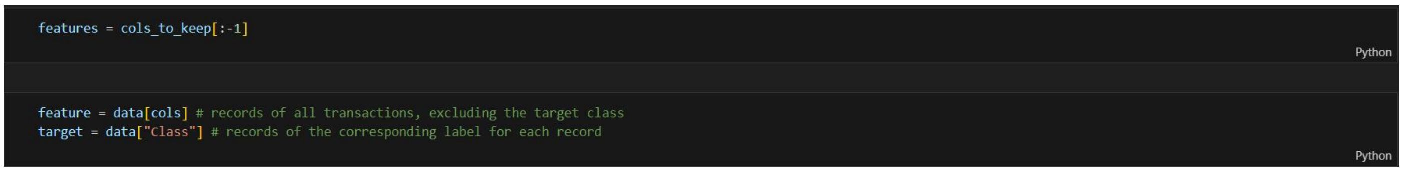
```python
corrmat = data.corr()
fig = plt.figure(figsize = (15, 15))
```

```python
corrmat['Class']
```

```python
cols = corrmat.keys()
cols_to_keep = []

for i in range(len(corrmat)):

    if abs(corrmat['Class'][i]) > 0.01:

        cols_to_keep.append(cols[i])

print(cols_to_keep)
```

6. Data Preparation:

The '**features**' variable contains the selected features. With the exception of the target class ('Class'), all transaction records are contained in the 'feature' variable. The labels ('Class') for each record are contained in the 'target' variable.

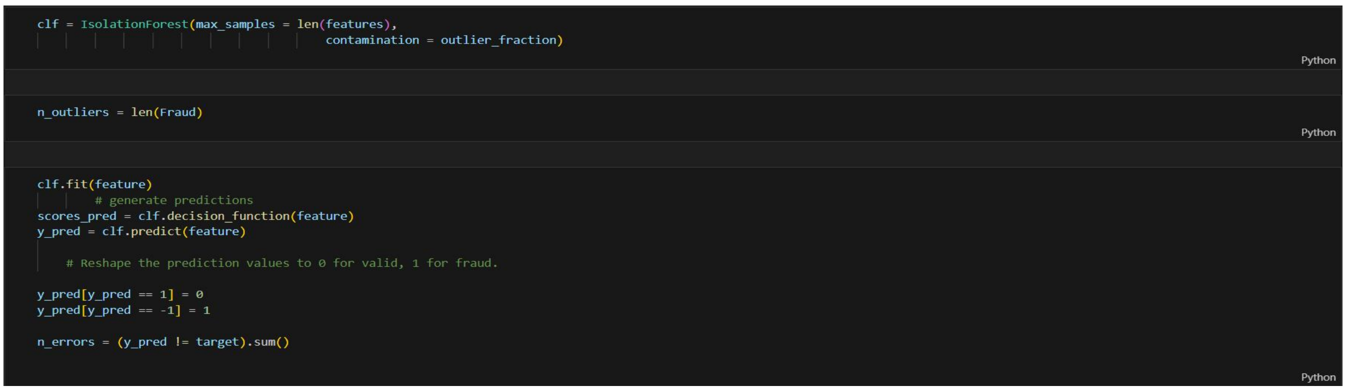


```python
features = cols_to_keep[:-1]
```

```python
feature = data[cols] # records of all transactions, excluding the target class
target = data["Class"] # records of the corresponding label for each record
```

7. Isolation Forest Algorithm:

The algorithm used to find anomalies is called Isolation Forest. The initialization of IsolationForest includes parameters like contamination (the anticipated percentage of outliers in the dataset) and max_samples (the maximum number of samples to draw when creating a tree). The features are used to train the model using **clf.fit**. Utilizing **clf.decision_function** and **clf.predict**, predictions are produced.



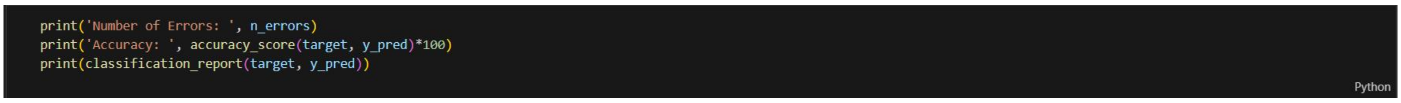
```python
clf = IsolationForest(max_samples = len(features),
                                        contamination = outlier_fraction)
```

```python
n_outliers = len(Fraud)
```

```python
clf.fit(feature)
        # generate predictions
scores_pred = clf.decision_function(feature)
y_pred = clf.predict(feature)

    # Reshape the prediction values to 0 for valid, 1 for fraud.

y_pred[y_pred == 1] = 0
y_pred[y_pred == -1] = 1

n_errors = (y_pred != target).sum()
```

8. Performance and Accuracy Evaluation:

Reformed predictions now provide 0 to legitimate transactions and 1 to fraudulent ones. Comparing y_pred against the actual labels yields the number of prediction errors. Using **accuracy_score**, accuracy is calculated. With the help of **classification_report**, a classification report is produced that includes precision, recall, an F1-score, and support for both classes (fraud and legitimate transactions).



```python
print('Number of Errors: ', n_errors)
print('Accuracy: ', accuracy_score(target, y_pred)*100)
print(classification_report(target, y_pred))
```

Reason Behind Choosing Isolation Forest:

The Isolation Forest algorithm is chosen for credit card fraud detection for several reasons:

- **Anomaly Detection:** Isolation Forest is well-suited for finding infrequent fraudulent transactions in an unbalanced dataset because it is built for anomaly identification.
- **Efficiency:** It is capable of processing big datasets and is computationally efficient. I could use linear regression or markov's algorithm but it is unable to process too big data at same time. So the accuracy will be less as compared to isolation forest.
- **Non-linear Separability:** Unlike othe algorithms, Isolation Forest is capable of handling non-linearly separable data, which is frequently the case in fraud detection.
- **Scalability:** It works well with high-dimensional data, making it suitable for credit card transaction datasets with numerous features. If there would be a case to add new attributes the it can manage easily.

Conclusion:

In this detailed code report, we examined a Python script for Isolation Forest algorithm-based credit card fraud detection. The code includes data pre-processing, feature selection, model training, and performance evaluation. We also looked into why I chose Isolation Forest over other algorithms. Isolation Forest is chosen for its efficiency and ability to handle imbalanced and high-dimensional datasets. This code acts as a helpful base for creating effective fraud detection systems in the financial sector. Further improvements could involve hyper-parameter tuning and exploring other anomaly detection algorithms for comparison.

Source Code: https://drive.google.com/file/d/13Xj9ro5CZVWY1oXwj0Y3hE899JppoXqb/view?usp=sharing