MINI PROJECT-II

(2020-21)

CREDIT CARD FRAUD DETECTION FINAL REPORT

INSTITUTE OF ENGINEERING AND TECHNOLOGY

SUBMITTED TO:

MR.VINAY AGARWAL

(ASSISTANT PROFESSOR)

` SUBMITTED BY:

NITIN KUMAR SINGH(181500434)

HARSHVARDHAN SINGH(181500264)

ABSTRACT

The rapid growth in E-Commerce industry has lead to an exponential increase in the use of credit cards for online purchases and consequently they has been surge in the fraud related to it .In recent years, For banks has become very difficult for detecting the fraud in credit card system. Machine learning plays a vital role for detecting the credit card fraud in the transactions. For predicting these transactions banks make use of various machine learning methodologies, past data has been collected and new features are been used for enhancing the predictive power. The performance of fraud detecting in credit card transactions is greatly affected by the sampling approach on data-set, selection of variables and detection techniques used. This paper investigates the performance of logistic regression, decision tree and random forest for credit card fraud detection. Dataset of credit card transactions is collected from kaggle and it contains a total of 2,84,808 credit card transactions of a European bank data set. It considers fraud transactions as the “positive class” and genuine ones as the “negative class” .The data set is highly imbalanced, it has about 0.172% of fraud transactions and the rest are genuine transactions. The author has been done oversampling to balance the data set, which resulted in 60% of fraud transactions and 40% genuine ones. The three techniques are applied for the dataset and work is implemented in R language. The performance of the techniques is evaluated for different variables based on sensitivity, specificity, accuracy and error rate. The result shows of accuracy for logistic regression, Decision tree and random forest classifier are 90.0, 94.3, 95.5 respectively. The comparative results show that the Random forest performs better than the logistic regression and decision tree technique

Department of computer Engineering and Applications
GLA University, Mathura
17 km. Stone NH#2, Mathura-Delhi Road, P.O. – Chaumuha,
Mathura – 281406

Declaration

This is certify that Nitin Kumar Singh and Harshvardhan Singh student of B.Tech(CSE) 3rd year has successfully Completed the MINI PROJECT named CREDIT CARD FRAUD DETECTION under the Guidance of Mr.Vinay Agarwal During 2020-21.

Signature:

Vinay Agarwal

(Mentor)

ACKNOWLEDGEMENT

It gives us a great sense of pleasure to present the synopsis of the B.Tech Mini Project (CREDIT CARD FRAUD DETECTION) Undertaken during B.Tech 3rd year. This project in itself is going to acknowledgement to the inspiration, drive and technical assistance will be contributed to it by many individuals.

We owe special debt of gratitude to Mr.Vinay Agarwal, Assistant Professor for providing with an encouraging platform to develop this Project, which thus helped us in shaping our abilities towards a constructive goal and for his constant support and guidelines to our work. His sincerity, thoroughness and perserverance is been a constant source of motivation for us. We believe that he will shower us with all his extensively experienced ideas and insightful comments at different stages of project and taught us about the latest industry-oriented technologies.

We also do not like to miss the opportunity to acknowledge the contribution of all faculty members of department for their guidance and contribution.

Nitin Kumar Singh(181500434)

Harshvardhan Singh(181500264)

INTRODUCTION

1.1-OVERVIEW

By applying the python libraries and macine learning algorithms we produce the model of credit card fraud detection by visualizing the Fraud in credit card transaction.

1.2-GENERAL INTRODUCTION OF THE TOPIC

'Fraud' in credit card transactions is unauthorized and unwanted usage of an account by someone other than the owner of that account. Necessary prevention measures can be taken to stop this abuse and the behaviour of such fraudulent practices can be studied to minimize it and protect against similar occurrences in the future.In other words, Credit Card Fraud can be defined as a case where a person uses someone else’s credit card for personal reasons while the owner and the card issuing authorities are unaware of the fact that the card is being used. Fraud detection involves monitoring the activities of populations of users in order to estimate, perceive or avoid objectionable behaviour, which consist of fraud, intrusion, and defaulting.

This is a very relevant problem that demands the attention of communities such as machine learning and data science where the solution to this problem can be automated. This problem is particularly challenging from the perspective of learning, as it is characterized by various factors such as class imbalance. The number of valid transactions far outnumber fraudulent ones. Also, the transaction patterns often change their statistical properties over the course of time.

HARDWARE REQUIREMENTS:

PROCESSOR USED:-Intel Pentium or above

OPERATING SYSTEM:-Win 7 or above

RAM:-4GB or above

HARDWARE DEVICES:-Computer or Laptop System

HARD DISK:-256GB or above

SOFTWARE REQUIREMENTS:

TECHNOLOGY USED:Numpy,Pandas,Scikit-learn library

LANGUAGE USED:PYTHON

USER INTERFACE DESIGN- JUPYTER NOTEBOOK

2-PROBLEM DEFINATION

To design and develop model which will provide the user details of ecommerce environment and detect credit card fraud at a time of payment which will be done using machine learning algorithm.to recognize fraudulent credit card transactions so that the customers of credit card companies are not charged for items that they did not purchase

3-OBJECTIVE:

The objective of credit card fraud detection are reduce looses due to payment fraud for both merchants and issuing a bank and increase revenue opportunities for merchants. Fraud represent significant financial risk to the merchant and issuing the bank to reduce fraud chip and pin technology.

1. The model used must be simple and fast enough to detect the anomaly and classify it as a fraudulent transaction as quickly as possible.
2. Imbalance can be dealt with by properly using some methods which we will talk about in the next paragraph
3. For protecting the privacy of the user the dimensionality of the data can be reduced.
4. A more trustworthy source must be taken which double-check the data, at least for training the model.

5. The model used must be simple and fast enough to detect the anomaly and classify it as a fraudulent transaction as quickly as possible.
6. Imbalance can be dealt with by properly using some methods which we will talk about in the next paragraph
7. For protecting the privacy of the user the dimensionality of the data can be reduced.
8. A more trustworthy source must be taken which double-check the data, at least for training the model.

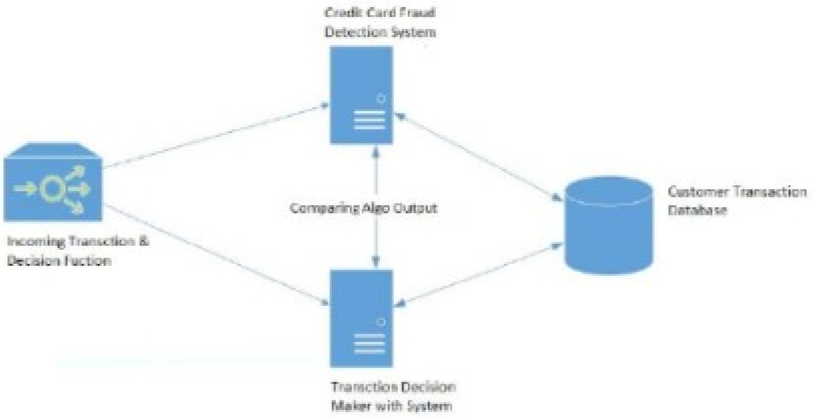
3. SOFTWARE DESIGN

A software design document (SDD) is a written description of a software product, that a software designer writes in order to give a software development team overall guidance to the architecture of the software project. An SDD usually accompanies an architecture diagram with pointers to detailed feature specifications of smaller pieces of the design. Practically, a design document is required to coordinate a large team under a single vision. A design document needs to be a stable reference, outlining all parts of the software and how they will work. The document is commanded to give a fairly complete description, while maintaining a high-level view of the software.

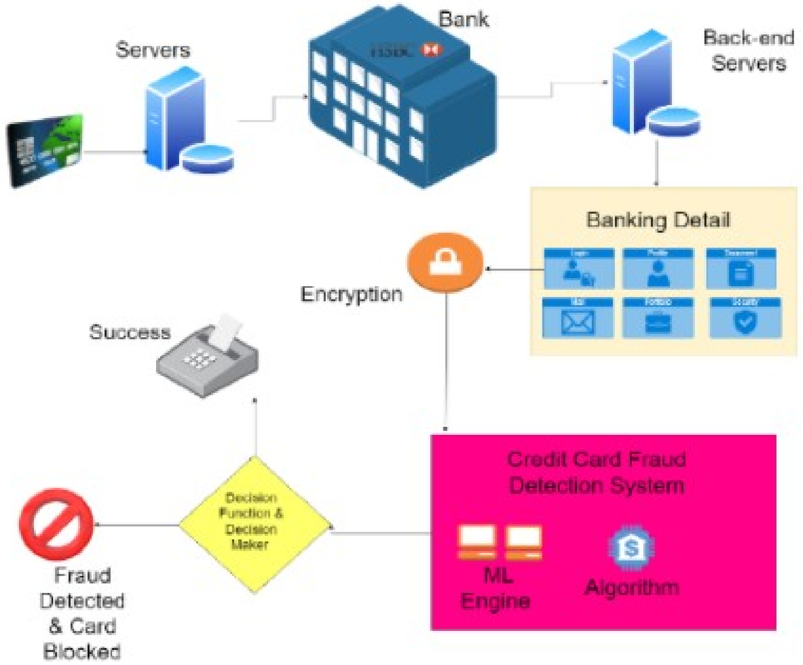
III. METHODOLOGY

The approach that this paper proposes, uses the latest machine learning algorithms to detect anomalous activities, called outliers.

The basic rough architecture diagram can be represented with the following figure:



When looked at in detail on a larger scale along with real life elements, the full architecture diagram can be represented as follows.

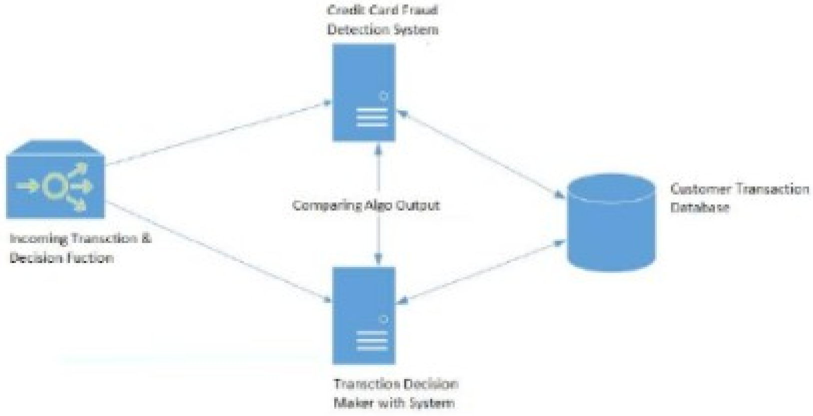

Servers
Bank
Back-end Servers
Banking Detail
Encryption
Success
Credit Card Fraud Detection System
ML Engine
Algorithm
Decision Function & Decision Maker
Fraud Detected & Card Blocked

First of all, we obtained our dataset from Kaggle, a data analysis website which provides datasets.

Inside this dataset, there are 31 columns out of which 28 are named as v1-v28 to protect sensitive data.

The other columns represent Time, Amount and Class. Time shows the time gap between the first transaction and the following one. Amount is the amount of money transacted. Class 0 represents a valid transaction and 1 represents a fraudulent one.

We plot different graphs to check for inconsistencies in the dataset and to visually comprehend it:


Credit Card Fraud Detection System
Incoming Transction & Decision Fuction
Comparing Algo Output
Customer Transaction Database
Transction Decision Maker with System

4. Testing

Introduction:

The implementation phase of software development is concerned with translating design specification into source code. The preliminary goal of implementation is to write source code and internal documentation so that conformance of the code to its specifications can be easily verified, and so that debugging, testing and modifications are eased. This goal can be achieved by making the source code as clear and straightforward as possible. Simplicity, clarity and elegance are the hallmark of good programs, obscurity, cleverness, and complexity are indications of inadequate design and misdirected thinking.

Source code clarity is enhanced by structured coding techniques, by good coding style, by, appropriate supporting documents, by good internal comments, and by feature provided in modern programming languages.

Terms in Testing Fundamental:

1. Error

2. Fault

3. Failure

4. Functional Test

Functional test cases involve exercising the code with Nominal input values for which expected results are known; as well as boundary values (minimum values, maximum values and values on and just outside the functional boundaries) and special values.

5. Performance Test

Performance testing determines the amount of execution time spent in various parts of the unit, program throughput, response time, and device utilization by the program unit.

6. Stress Test

Stress test are those designed to intentionally break the unit. A great deal can be learned about the strengths and limitations of a program by examining the manner in which a program unit breaks.

7. Structure Test

Structure tests are concerned with exercising the internal logic of a program and traversing particular execution paths. Some authors refer collectively to functional performance and stress testing as “black box” testing

Test Case	Description	Prediction	Result
1	Import required modules and packages	installed	OK
2	CSV FILE	Saved successfully	OK
3	Model prepare	successfully	OK
4	Regression analysis	Run successfully	OK

IMPLEMENTATION-

This idea is difficult to implement in real life because it requires the cooperation from banks, which aren't willing to share information due to their market competition, and also due to legal reasons and protection of data of their users. Therefore, we looked up some reference papers which followed similar approaches and gathered results. As stated in one of these reference papers:
"This technique was applied to a full application data set supplied by a German bank in 2006. For banking confidentiality reasons, only a summary of the results obtained is presented below. After applying this technique, the level 1 list encompasses a few cases but with a high probability of being fraudsters.
All individuals mentioned in this list had their cards closed to avoid any risk due to their high-risk profile. The condition is more complex for the other list. The level 2 list is still restricted adequately to be checked on a case by case basis. Credit and collection officers considered that half of the cases in this list could be considered as suspicious fraudulent behaviour. For the last list and the largest, the work is equitably heavy. Less than a third of them are suspicious.
In order to maximize the time efficiency and the overhead charges, a possibility is to include a new element in the query; this element can be the five first digits of the phone numbers, the email address, and the password.

queries can be applied to the level 2 list and level 3 list.".

```
import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

url = 'creditcard.csv'

creditcard = pd.read_csv(url)

creditcard
creditcard.shape

creditcard.columns

creditcard['Class'].value_counts()

sns.set_style('whitegrid')

sns.FacetGrid(creditcard, hue = 'Class', size = 6).map(plt.scatter, 'Time',
'Amount').add_legend()

plt.show()

sns.set_style('whitegrid')

sns.FacetGrid(creditcard, hue = 'Class', size = 6).map(plt.scatter, 'Amount',
'Time').add_legend()

plt.show()

FilteredData = creditcard[['Time', 'Amount', 'Class']]

FilteredData
```

```
FilteredData.shape

FilteredData['Class'].value_counts()

plt.close()

sns.set_style('whitegrid')

sns.pairplot(FilteredData, hue = 'Class', size = 5)

plt.show()

countLess = 0

countMore = 0

for i in range(284806):

    if(FilteredData.iloc[i]['Amount'] < 2500):

        countLess = countLess + 1

    else:

        countMore = countMore + 1

print(countLess)

print(countMore)
```

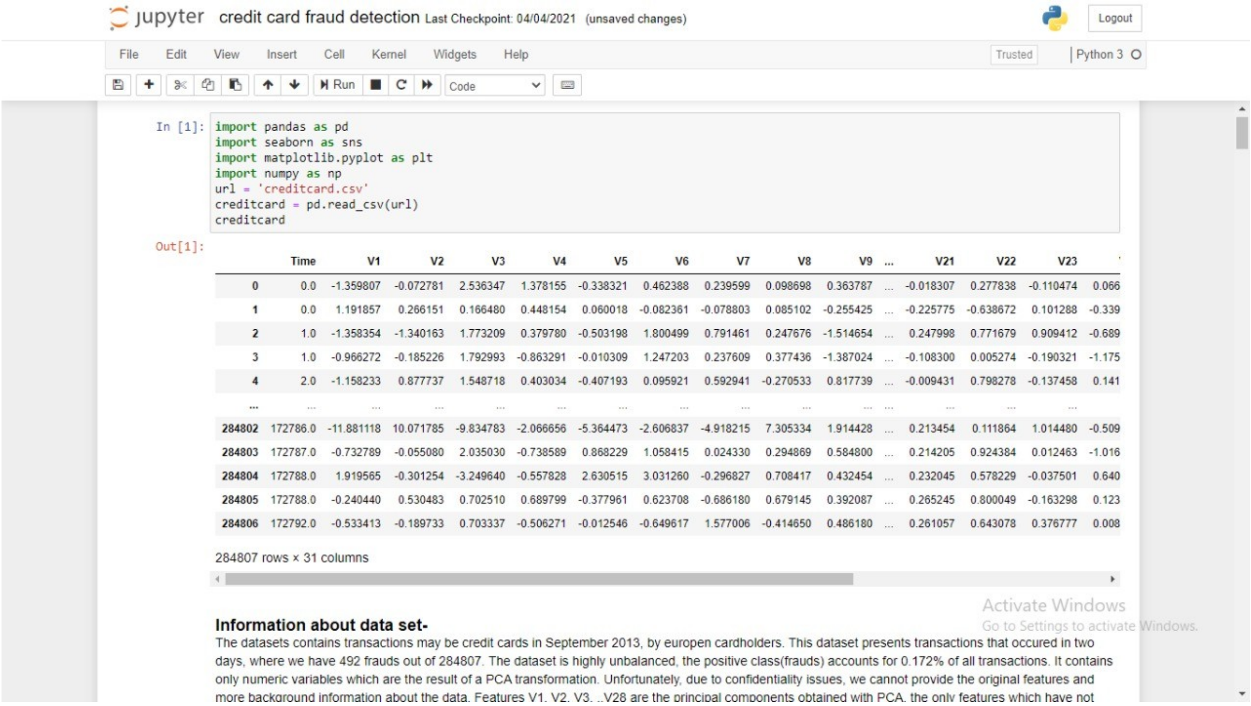
```
percentage = (countLess / 284807) * 100

percentage

class0 = 0

class1 = 0

for i in range(284806):

    if(FilteredData.iloc[i]['Amount'] < 2500):

        if(FilteredData.iloc[i]['Class'] == 0):

            class0 = class0 + 1

        else:

            class1 = class1 + 1

print(class0)

print(class1)

FilteredData['Class'].value_counts()

creditcard_genuine = FilteredData.loc[FilteredData['Class'] == 0]

creditcard_fraud = FilteredData.loc[FilteredData['Class'] == 1]

plt.plot(creditcard_genuine['Time'], np.zeros_like(creditcard_genuine['Time']), "o")

plt.plot(creditcard_fraud['Time'], np.zeros_like(creditcard_fraud['Time']), "o")

plt.show()

plt.plot(creditcard_genuine['Amount'], np.zeros_like(creditcard_genuine['Amount']), "o")

plt.plot(creditcard_fraud['Amount'], np.zeros_like(creditcard_fraud['Amount']), "o")
```

```
sns.FacetGrid(FilteredData, hue = 'Class', size = 10).map(sns.distplot,
'Time').add_legend()

plt.show()

sns.FacetGrid(FilteredData, hue = 'Class', height = 10).map(sns.distplot,
'Amount').add_legend()

plt.show()

counts, bin_edges = np.histogram(FilteredData['Amount'], bins = 10, density =
True)

pdf = counts / (sum(counts))

print("pdf = ", pdf)

print("\n")

print("counts = ", counts)

print("\n")

print("Bin edges = ", bin_edges)

print('Means:')

print('Mean of transaction amount of genuine transactions: ',
np.mean(creditcard_genuine['Amount']))

print('Mean of transaction amount of fraud transactions: ',
np.mean(creditcard_fraud['Amount']))

print('Standard Deviation:')

print('Std-Deviation of transaction amount of genuine transactions: ',
np.std(creditcard_genuine['Amount']))

print('Std-Deviation of transaction amount of fraud transactions: ',
np.std(creditcard_fraud['Amount']))
```

```
print('Median:')
print('Median of transaction amount of genuine transactions: ', np.median(creditcard_genuine['Amount']))
print('Median of transaction amount of fraud transactions: ', np.median(creditcard_fraud['Amount']))
print('Quantiles:')
print(np.percentile(creditcard_genuine['Amount'], np.arange(0, 100, 25)))
print(np.percentile(creditcard_fraud['Amount'], np.arange(0, 100, 25)))
sns.boxplot(x = 'Class', y = 'Time', data = creditcard)
plt.show()
sns.boxplot(x = 'Class', y = 'Amount', data = creditcard)
plt.ylim(0, 5000)
plt.show()
from scipy import spatial
sampleData = creditcard.head(20000) #Sample the data from original data so as to save the computation time
samples = creditcard.loc[30401:30500] #taking samples of size 100 from index 30401 to 30500
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) {
    return false;
}
frame = []
```

```
for i in range(30401, 30501):
    t1 = samples.loc[i]
    c = samples.loc[i]['Class']
    for j in range(20000):
        t2 = sampleData.loc[j]
        classLabel = creditcard.loc[j]['Class']
        similarity = 1 - spatial.distance.cosine(t1, t2)
        frame.append([classLabel, similarity, j])
    df = pd.DataFrame(frame, columns = ['Class', 'Similarity', 'Transaction ID'])
    df_sorted = df.sort_values('Similarity', ascending = False)
    print('Top 10 transactions having highest similarity with transaction ID = ' + str(i)
+ ' and class = ' + str(c) + ':')
    print(df_sorted.iloc[:10])
    print("\n")
    frame = []
```

SCREENSHOTS



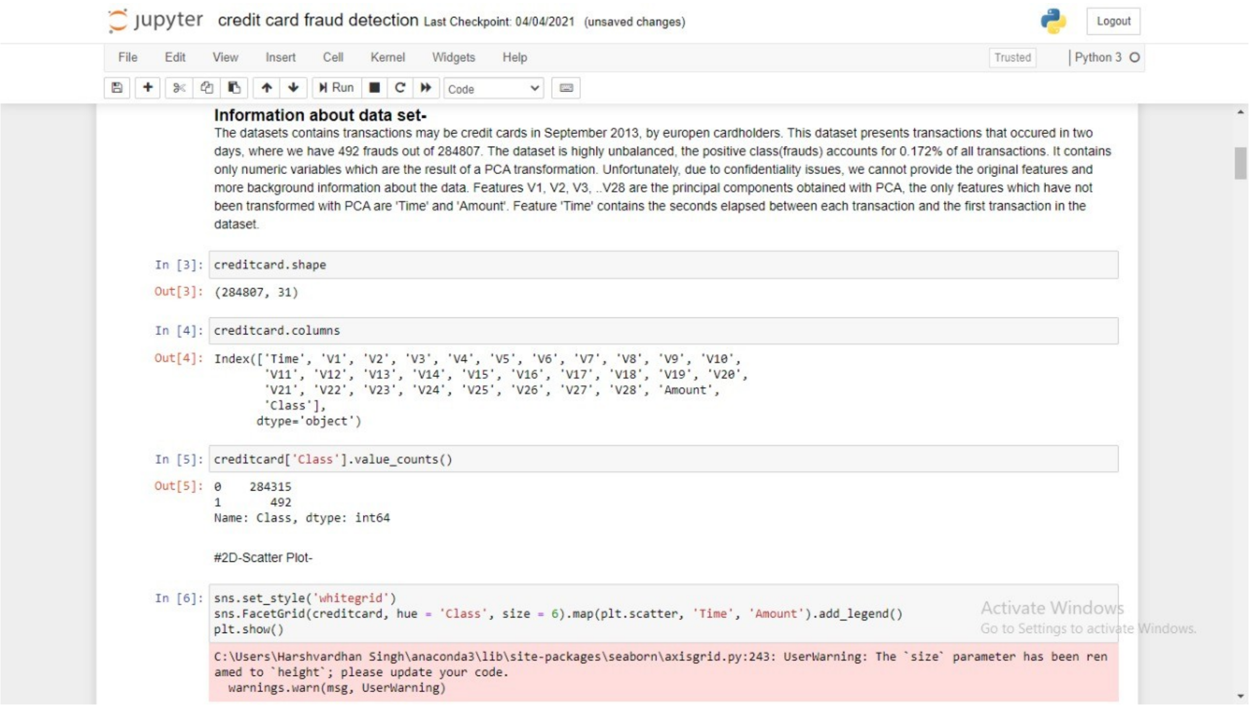
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
url = 'creditcard.csv'
creditcard = pd.read_csv(url)
creditcard
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	
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	0.066
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	-0.339
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	-0.689
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	-1.175
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	0.141
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014480	-0.509
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.012463	-1.016
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.037501	0.640
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.163298	0.123
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.376777	0.008

284807 rows × 31 columns

Information about data set-

The datasets contains transactions may be credit cards in September 2013, by europen cardholders. This dataset presents transactions that occured in two days, where we have 492 frauds out of 284807. The dataset is highly unbalanced, the positive class(frauds) accounts for 0.172% of all transactions. It contains only numeric variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, V3, ..V28 are the principal components obtained with PCA, the only features which have not


jupyter credit card fraud detection Last Checkpoint: 04/04/2021 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 ○
Run
Code
Activate Windows
Go to Settings to activate Windows.

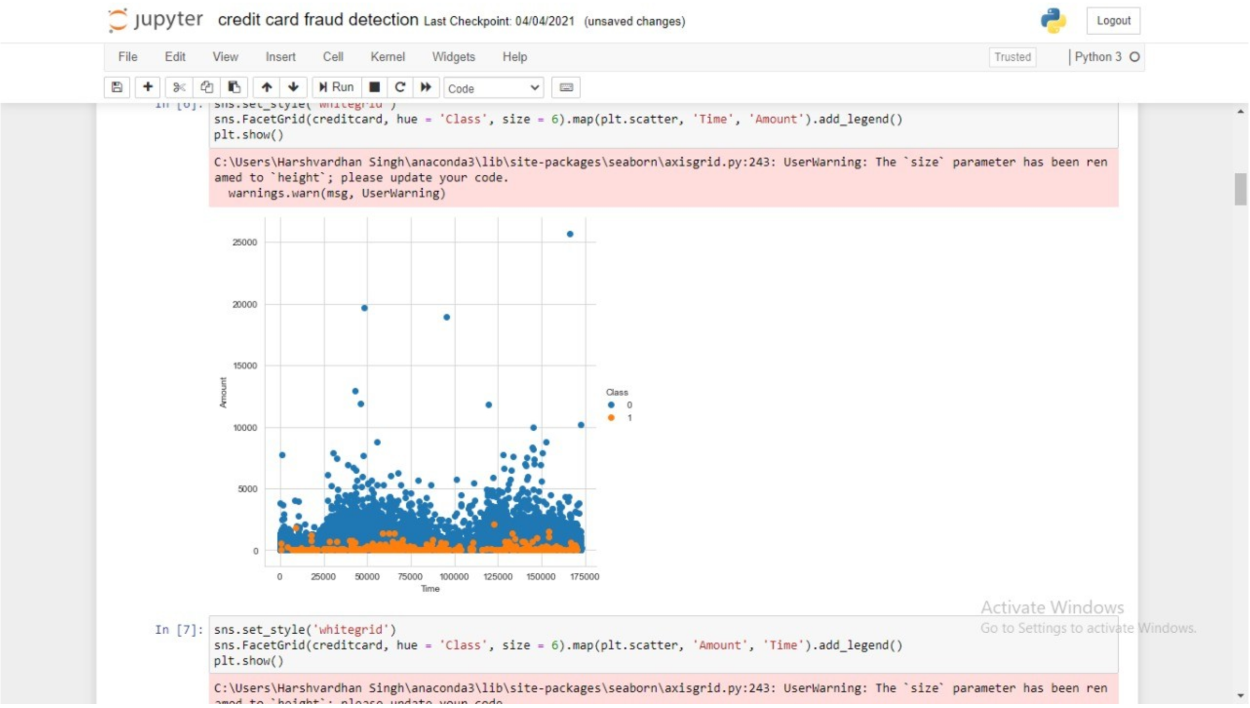
Information about data set-

The datasets contains transactions may be credit cards in September 2013, by europen cardholders. This dataset presents transactions that occured in two days, where we have 492 frauds out of 284807. The dataset is highly unbalanced, the positive class(frauds) accounts for 0.172% of all transactions. It contains only numeric variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, V3, ..V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset.

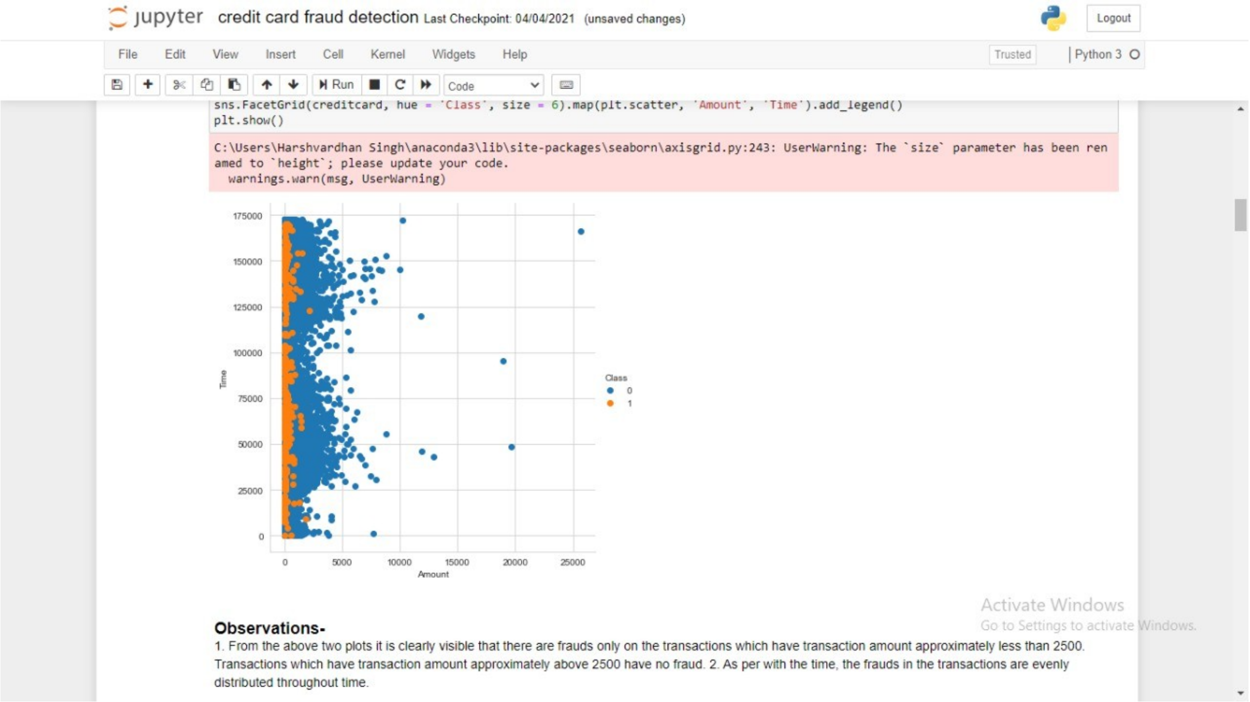
```
In [3]: creditcard.shape

Out[3]: (284807, 31)

In [4]: creditcard.columns

Out[4]: Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
               'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
               'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
               'Class'],
              dtype='object')

In [5]: creditcard['Class'].value_counts()

Out[5]: 0    284315
        1       492
        Name: Class, dtype: int64
```

#2D-Scatter Plot-

```
In [6]: sns.set_style('whitegrid')
        sns.FacetGrid(creditcard, hue = 'Class', size = 6).map(plt.scatter, 'Time', 'Amount').add_legend()
        plt.show()

        C:\Users\Harshvardhan Singh\anaconda3\lib\site-packages\seaborn\axisgrid.py:243: UserWarning: The `size` parameter has been ren
        amed to `height`; please update your code.
          warnings.warn(msg, UserWarning)
```


jupyter credit card fraud detection Last Checkpoint: 04/04/2021 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 ○
Run
Code
25000
20000
15000
10000
5000
0
Amount
0
25000
50000
75000
100000
125000
150000
175000
Time
Class
● 0
● 1
Activate Windows
Go to Settings to activate Windows.

```
In [6]: sns.set_style('whitegrid')
        sns.FacetGrid(creditcard, hue = 'Class', size = 6).map(plt.scatter, 'Time', 'Amount').add_legend()
        plt.show()

        C:\Users\Harshvardhan Singh\anaconda3\lib\site-packages\seaborn\axisgrid.py:243: UserWarning: The `size` parameter has been ren
        amed to `height`; please update your code.
          warnings.warn(msg, UserWarning)

In [7]: sns.set_style('whitegrid')
        sns.FacetGrid(creditcard, hue = 'Class', size = 6).map(plt.scatter, 'Amount', 'Time').add_legend()
        plt.show()

        C:\Users\Harshvardhan Singh\anaconda3\lib\site-packages\seaborn\axisgrid.py:243: UserWarning: The `size` parameter has been ren
        amed to `height`; please update your code.
```

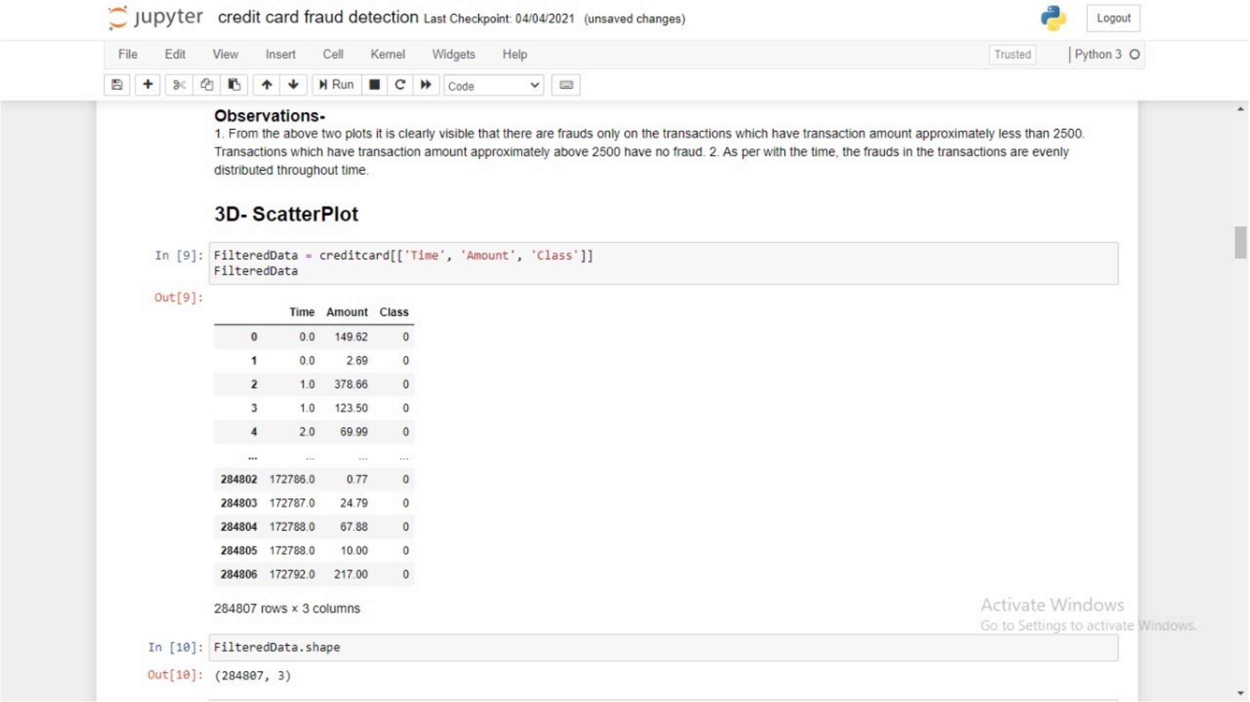


```
sns.FacetGrid(creditcard, hue = 'Class', size = 6).map(plt.scatter, 'Amount', 'Time').add_legend()
plt.show()
```

```
C:\Users\Harshvardhan Singh\anaconda3\lib\site-packages\seaborn\axisgrid.py:243: UserWarning: The `size` parameter has been ren
amed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```

Observations-

1. From the above two plots it is clearly visible that there are frauds only on the transactions which have transaction amount approximately less than 2500. Transactions which have transaction amount approximately above 2500 have no fraud. 2. As per with the time, the frauds in the transactions are evenly distributed throughout time.


jupyter
credit card fraud detection Last Checkpoint: 04/04/2021 (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
Activate Windows
Go to Settings to activate Windows.

Observations-

1. From the above two plots it is clearly visible that there are frauds only on the transactions which have transaction amount approximately less than 2500. Transactions which have transaction amount approximately above 2500 have no fraud. 2. As per with the time, the frauds in the transactions are evenly distributed throughout time.

3D- ScatterPlot

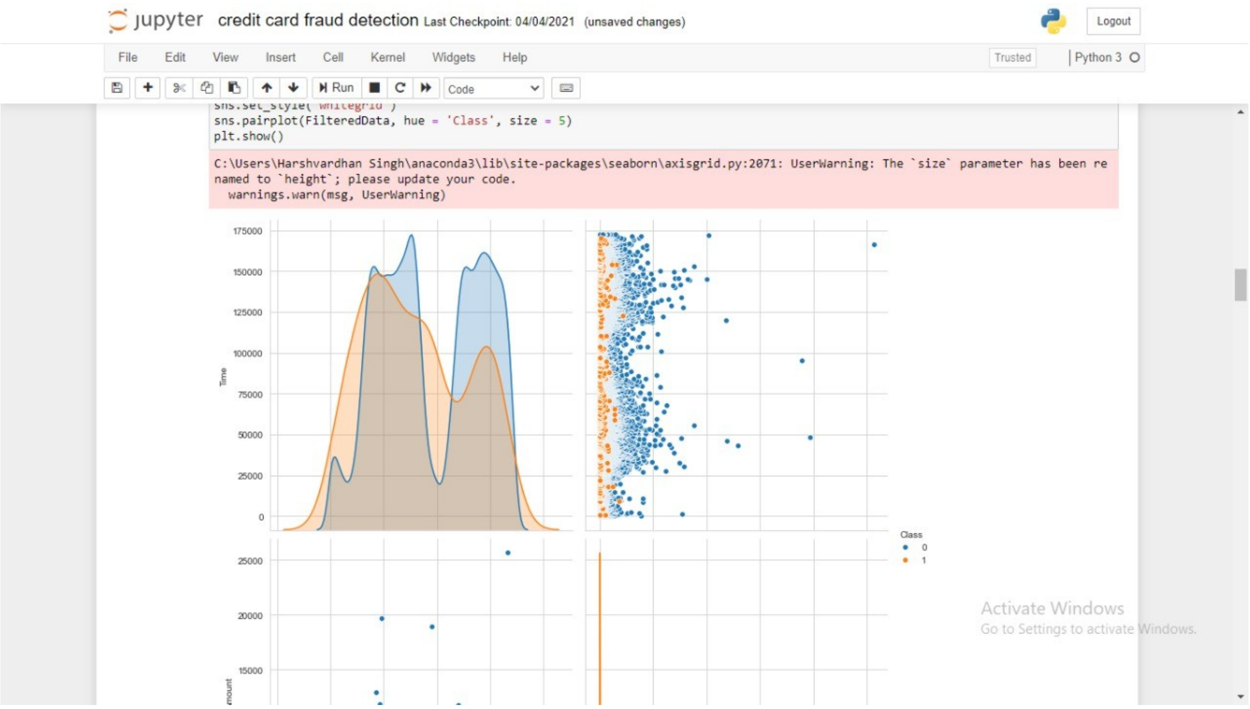
```
In [9]: FilteredData = creditcard[['Time', 'Amount', 'Class']]
        FilteredData
```

Out[9]:

	Time	Amount	Class
0	0.0	149.62	0
1	0.0	2.69	0
2	1.0	378.66	0
3	1.0	123.50	0
4	2.0	69.99	0
...	...	...	...
284802	172786.0	0.77	0
284803	172787.0	24.79	0
284804	172788.0	67.88	0
284805	172788.0	10.00	0
284806	172792.0	217.00	0

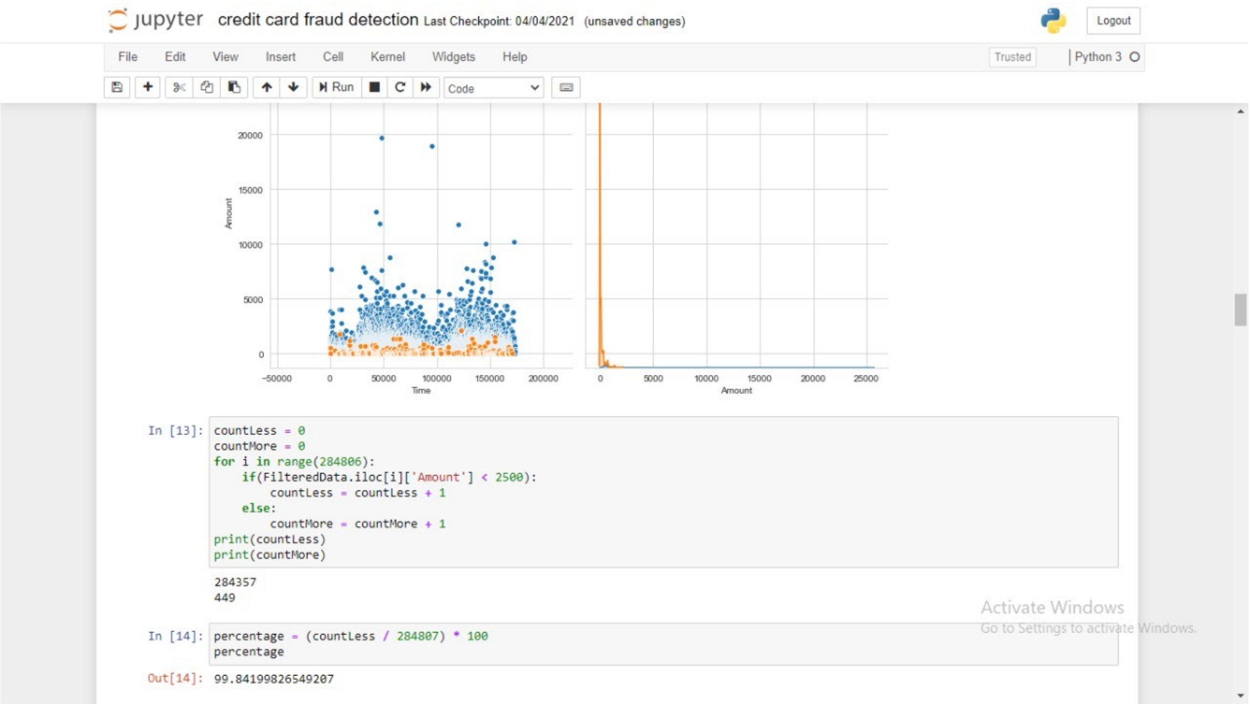
284807 rows × 3 columns

```
In [10]: FilteredData.shape
Out[10]: (284807, 3)
```

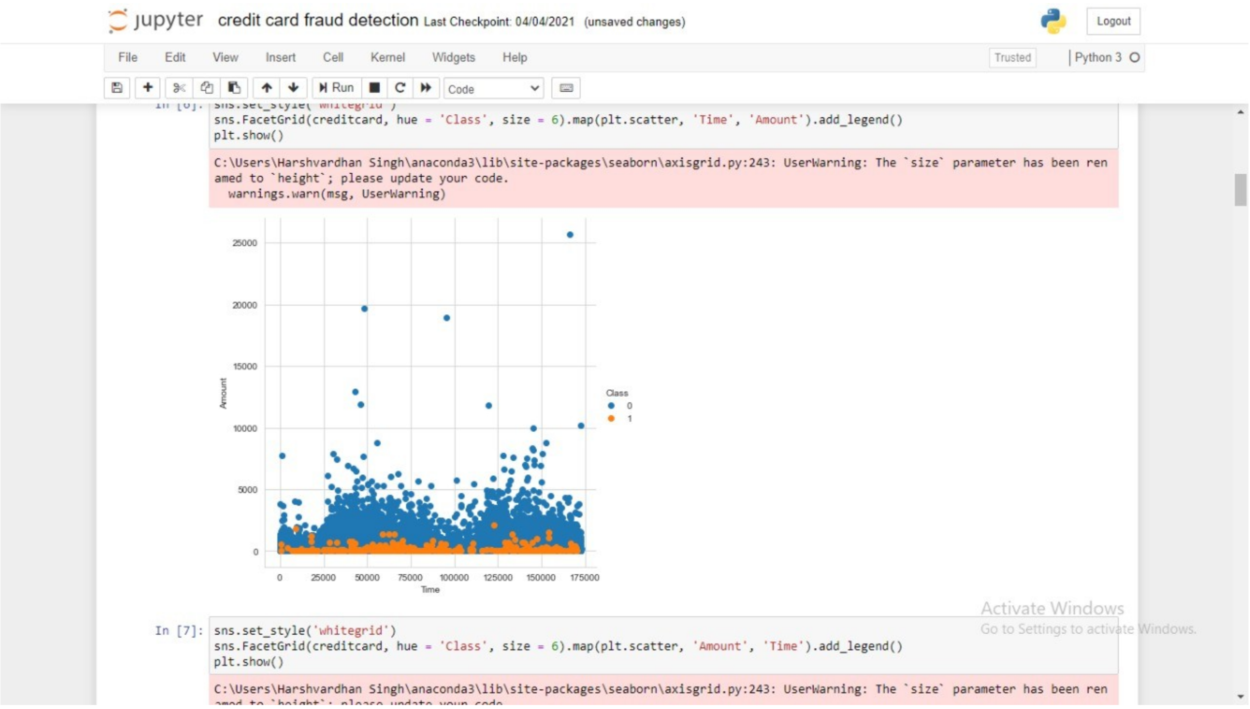


```
sns.set_style('whitegrid')
sns.pairplot(FilteredData, hue = 'Class', size = 5)
plt.show()
```

```
C:\Users\Harshvardhan Singh\anaconda3\lib\site-packages\seaborn\axisgrid.py:2071: UserWarning: The `size` parameter has been re
named to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```

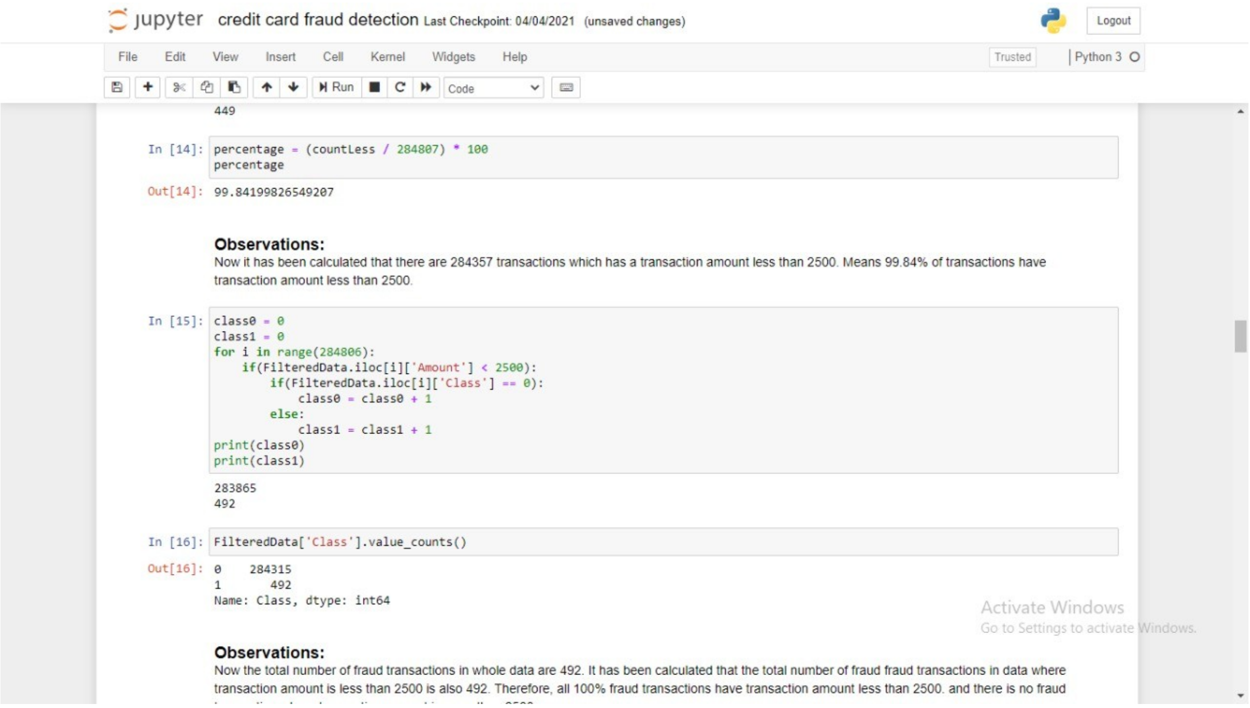


```
In [13]: countLess = 0
         countMore = 0
         for i in range(284806):
             if(FilteredData.iloc[i]['Amount'] < 2500):
                 countLess = countLess + 1
             else:
                 countMore = countMore + 1
         print(countLess)
         print(countMore)

         284357
         449

In [14]: percentage = (countLess / 284807) * 100
         percentage

Out[14]: 99.84199826549207
```



```
In [6]: sns.set_style('whitegrid')
        sns.FacetGrid(creditcard, hue = 'Class', size = 6).map(plt.scatter, 'Time', 'Amount').add_legend()
        plt.show()

C:\Users\Harshvardhan Singh\anaconda3\lib\site-packages\seaborn\axisgrid.py:243: UserWarning: The `size` parameter has been ren
amed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```

```
In [7]: sns.set_style('whitegrid')
        sns.FacetGrid(creditcard, hue = 'Class', size = 6).map(plt.scatter, 'Amount', 'Time').add_legend()
        plt.show()

C:\Users\Harshvardhan Singh\anaconda3\lib\site-packages\seaborn\axisgrid.py:243: UserWarning: The `size` parameter has been ren
```


jupyter
credit card fraud detection Last Checkpoint: 04/04/2021 (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
Activate Windows
Go to Settings to activate Windows.

```
449
```

```
In [14]: percentage = (countLess / 284807) * 100
         percentage

Out[14]: 99.84199826549207
```

Observations:

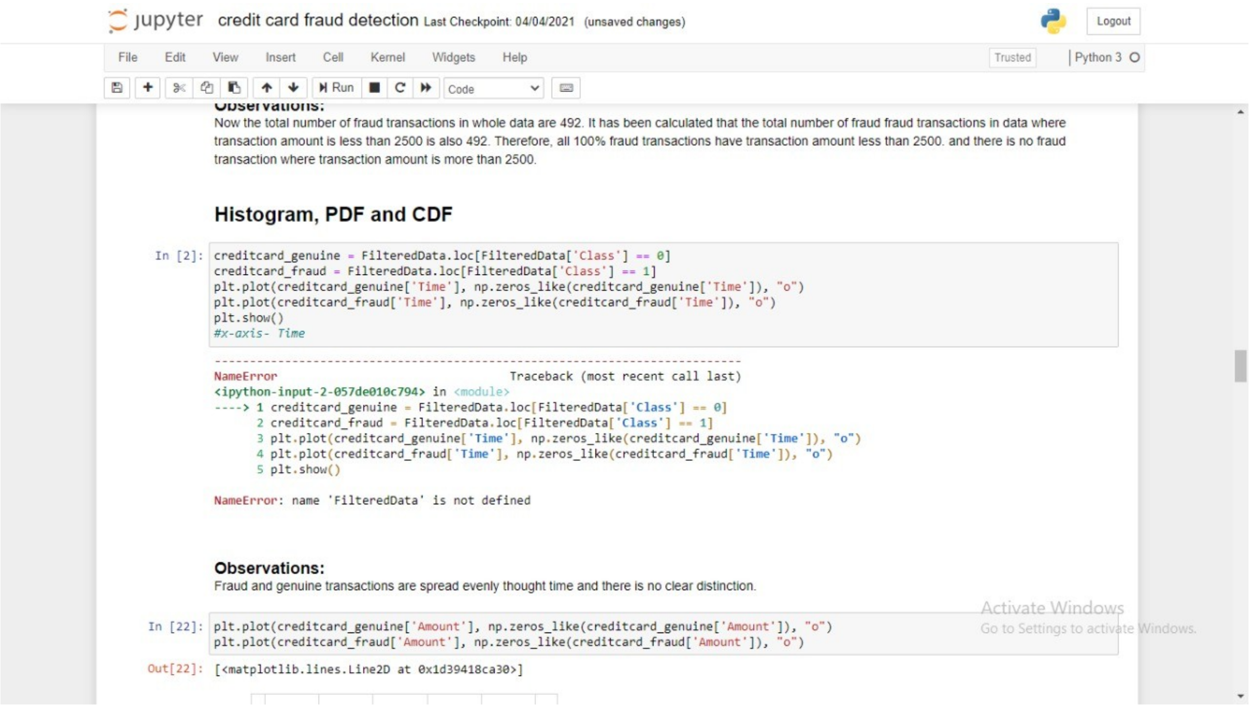
Now it has been calculated that there are 284357 transactions which has a transaction amount less than 2500. Means 99.84% of transactions have transaction amount less than 2500.

```
In [15]: class0 = 0
         class1 = 0
         for i in range(284806):
             if(FilteredData.iloc[i]['Amount'] < 2500):
                 if(FilteredData.iloc[i]['Class'] == 0):
                     class0 = class0 + 1
                 else:
                     class1 = class1 + 1
         print(class0)
         print(class1)

         283865
         492
```

```
In [16]: FilteredData['Class'].value_counts()

Out[16]: 0    284315
         1       492
         Name: Class, dtype: int64
```

Observations:

Now the total number of fraud transactions in whole data are 492. It has been calculated that the total number of fraud fraud transactions in data where transaction amount is less than 2500 is also 492. Therefore, all 100% fraud transactions have transaction amount less than 2500. and there is no fraud



Observations:

Now the total number of fraud transactions in whole data are 492. It has been calculated that the total number of fraud fraud transactions in data where transaction amount is less than 2500 is also 492. Therefore, all 100% fraud transactions have transaction amount less than 2500. and there is no fraud transaction where transaction amount is more than 2500.

Histogram, PDF and CDF

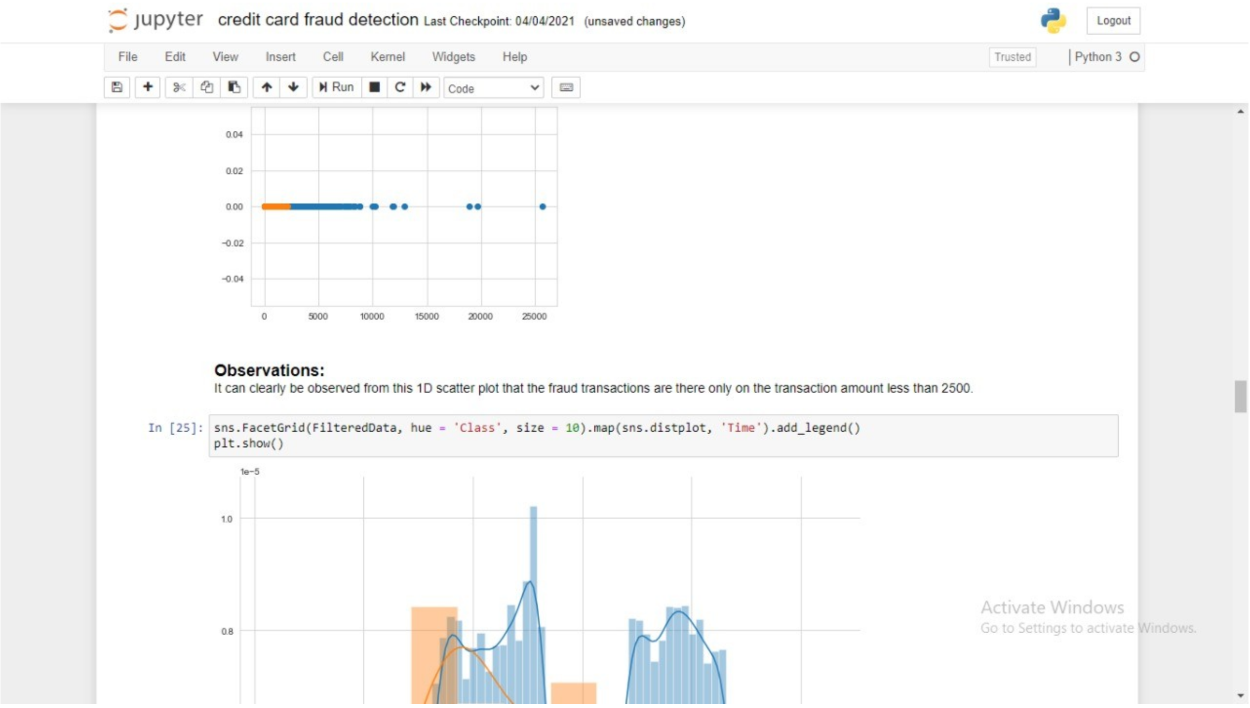
```
In [2]: creditcard_genuine = FilteredData.loc[FilteredData['Class'] == 0]
        creditcard_fraud = FilteredData.loc[FilteredData['Class'] == 1]
        plt.plot(creditcard_genuine['Time'], np.zeros_like(creditcard_genuine['Time']), "o")
        plt.plot(creditcard_fraud['Time'], np.zeros_like(creditcard_fraud['Time']), "o")
        plt.show()
        #x-axis- Time
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-057de010c794> in <module>
----> 1 creditcard_genuine = FilteredData.loc[FilteredData['Class'] == 0]
      2 creditcard_fraud = FilteredData.loc[FilteredData['Class'] == 1]
      3 plt.plot(creditcard_genuine['Time'], np.zeros_like(creditcard_genuine['Time']), "o")
      4 plt.plot(creditcard_fraud['Time'], np.zeros_like(creditcard_fraud['Time']), "o")
      5 plt.show()

NameError: name 'FilteredData' is not defined
```

Observations:

Fraud and genuine transactions are spread evenly thought time and there is no clear distinction.

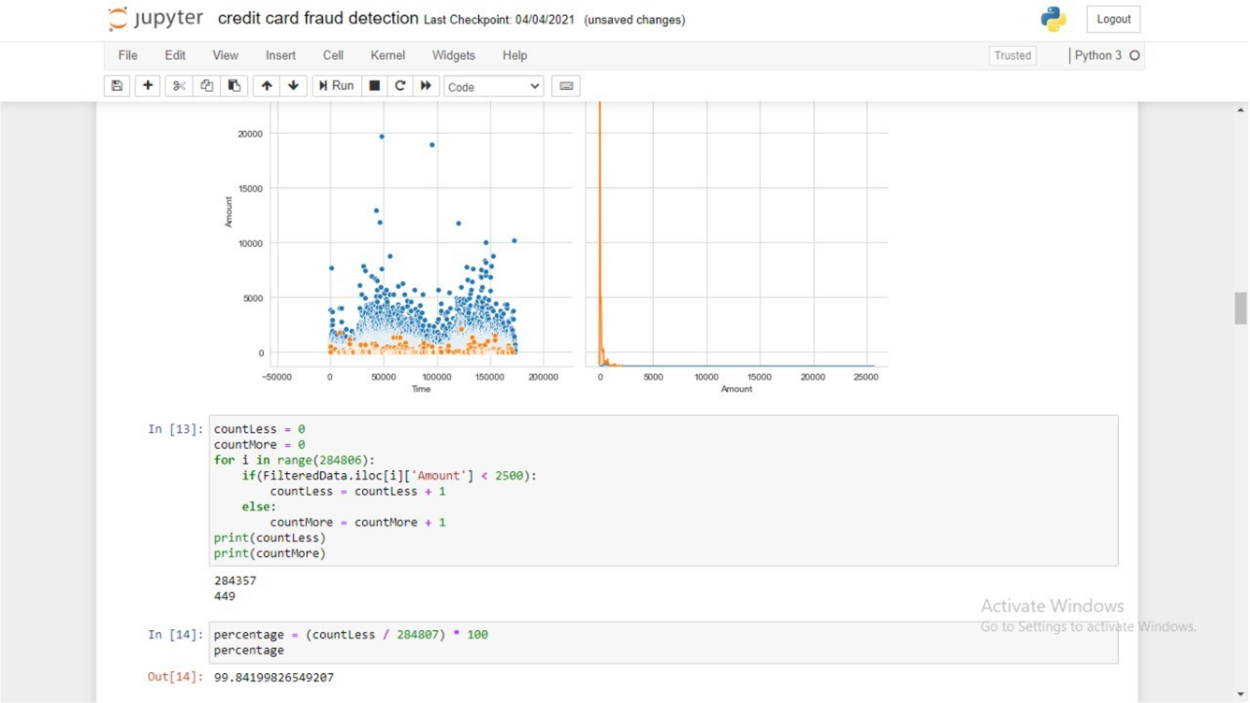
```
In [22]: plt.plot(creditcard_genuine['Amount'], np.zeros_like(creditcard_genuine['Amount']), "o")
         plt.plot(creditcard_fraud['Amount'], np.zeros_like(creditcard_fraud['Amount']), "o")

Out[22]: [<matplotlib.lines.Line2D at 0x1d39418ca30>]
```



Observations:

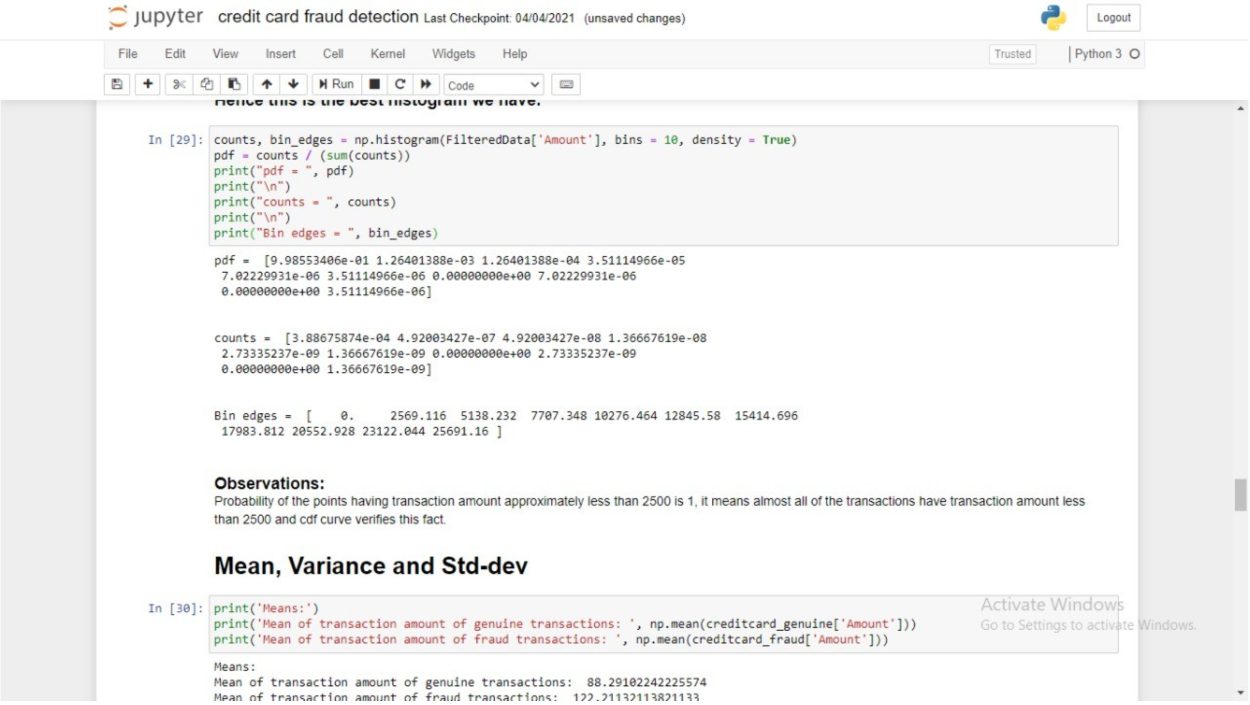
It can clearly be observed from this 1D scatter plot that the fraud transactions are there only on the transaction amount less than 2500.

```
In [25]: sns.FacetGrid(FilteredData, hue = 'Class', size = 10).map(sns.distplot, 'Time').add_legend()
         plt.show()
```



```
In [13]: countLess = 0
         countMore = 0
         for i in range(284806):
             if(FilteredData.iloc[i]['Amount'] < 2500):
                 countLess = countLess + 1
             else:
                 countMore = countMore + 1
         print(countLess)
         print(countMore)

         284357
         449
```

```
In [14]: percentage = (countLess / 284807) * 100
         percentage

Out[14]: 99.84199826549207
```


jupyter
credit card fraud detection Last Checkpoint: 04/04/2021 (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
Activate Windows
Go to Settings to activate Windows.

Hence this is the best histogram we have.

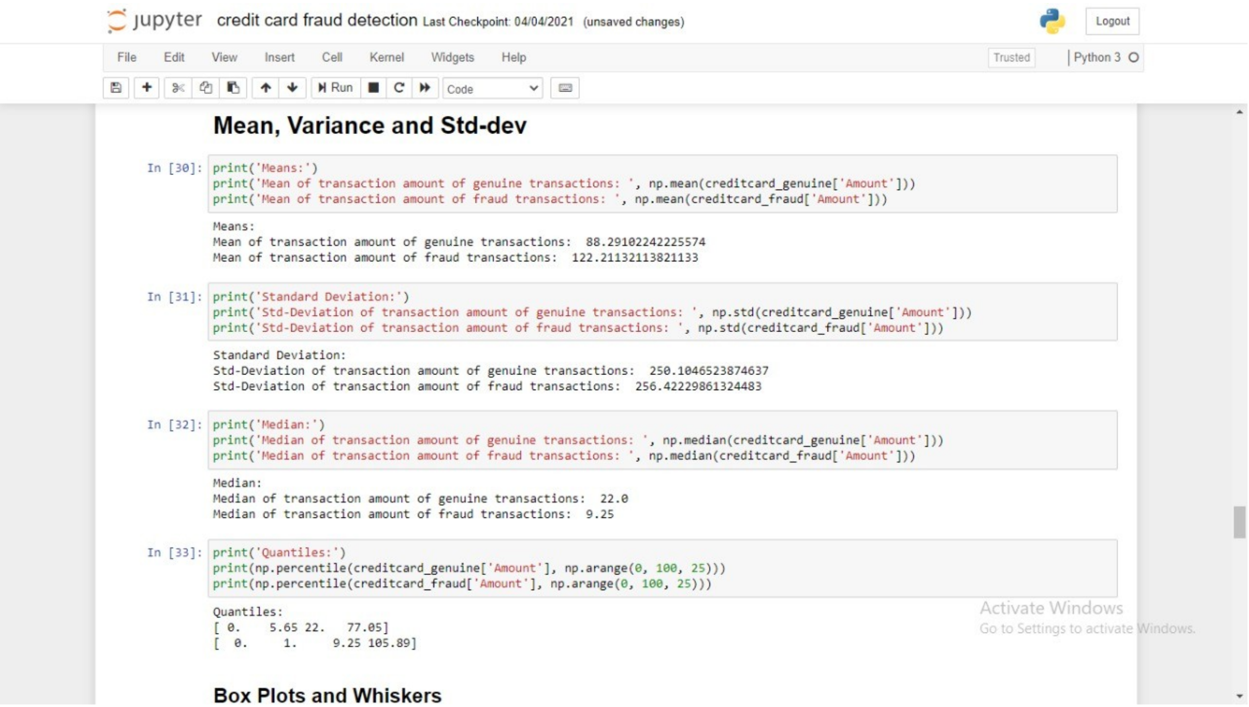
```
In [29]: counts, bin_edges = np.histogram(FilteredData['Amount'], bins = 10, density = True)
         pdf = counts / (sum(counts))
         print("pdf = ", pdf)
         print("\n")
         print("counts = ", counts)
         print("\n")
         print("Bin edges = ", bin_edges)

         pdf =  [9.98553406e-01 1.26401388e-03 1.26401388e-04 3.51114966e-05
          7.02229931e-06 3.51114966e-06 0.00000000e+00 7.02229931e-06
          0.00000000e+00 3.51114966e-06]


         counts =  [3.88675874e-04 4.92003427e-07 4.92003427e-08 1.36667619e-08
          2.73335237e-09 1.36667619e-09 0.00000000e+00 2.73335237e-09
          0.00000000e+00 1.36667619e-09]


         Bin edges =  [    0.      2569.116  5138.232  7707.348 10276.464 12845.58  15414.696
          17983.812 20552.928 23122.044 25691.16 ]
```

Observations:

Probability of the points having transaction amount approximately less than 2500 is 1, it means almost all of the transactions have transaction amount less than 2500 and cdf curve verifies this fact.

Mean, Variance and Std-dev

```
In [30]: print('Means:')
         print('Mean of transaction amount of genuine transactions: ', np.mean(creditcard_genuine['Amount']))
         print('Mean of transaction amount of fraud transactions: ', np.mean(creditcard_fraud['Amount']))

         Means:
         Mean of transaction amount of genuine transactions:  88.29102242225574
         Mean of transaction amount of fraud transactions:  122.21132113821133
```


jupyter credit card fraud detection Last Checkpoint: 04/04/2021 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Activate Windows
Go to Settings to activate Windows.

Mean, Variance and Std-dev

In [30]:
```
print('Means:')
print('Mean of transaction amount of genuine transactions: ', np.mean(creditcard_genuine['Amount']))
print('Mean of transaction amount of fraud transactions: ', np.mean(creditcard_fraud['Amount']))
```

```
Means:
Mean of transaction amount of genuine transactions:  88.29102242225574
Mean of transaction amount of fraud transactions:  122.21132113821133
```

In [31]:
```
print('Standard Deviation:')
print('Std-Deviation of transaction amount of genuine transactions: ', np.std(creditcard_genuine['Amount']))
print('Std-Deviation of transaction amount of fraud transactions: ', np.std(creditcard_fraud['Amount']))
```

```
Standard Deviation:
Std-Deviation of transaction amount of genuine transactions:  250.1046523874637
Std-Deviation of transaction amount of fraud transactions:  256.42229861324483
```

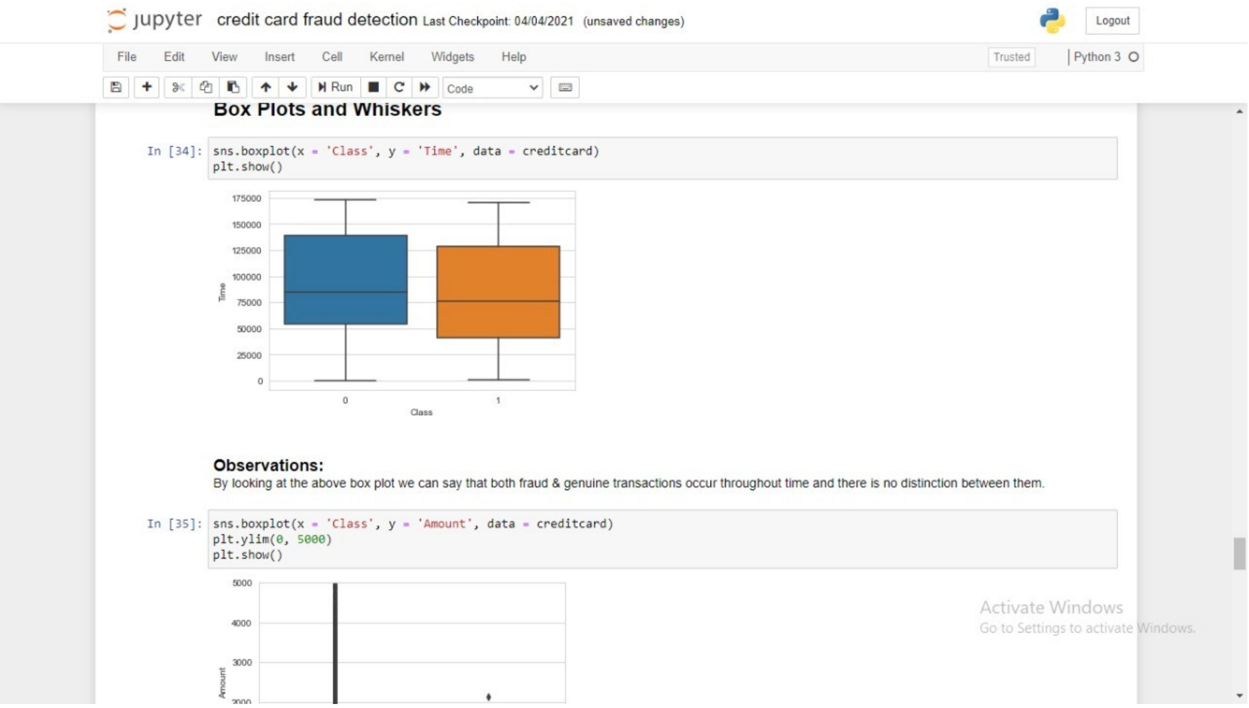
In [32]:
```
print('Median:')
print('Median of transaction amount of genuine transactions: ', np.median(creditcard_genuine['Amount']))
print('Median of transaction amount of fraud transactions: ', np.median(creditcard_fraud['Amount']))
```

```
Median:
Median of transaction amount of genuine transactions:  22.0
Median of transaction amount of fraud transactions:  9.25
```

In [33]:
```
print('Quantiles:')
print(np.percentile(creditcard_genuine['Amount'], np.arange(0, 100, 25)))
print(np.percentile(creditcard_fraud['Amount'], np.arange(0, 100, 25)))
```

```
Quantiles:
[ 0.     5.65 22.    77.05]
[  0.      1.      9.25 105.89]
```

Box Plots and Whiskers

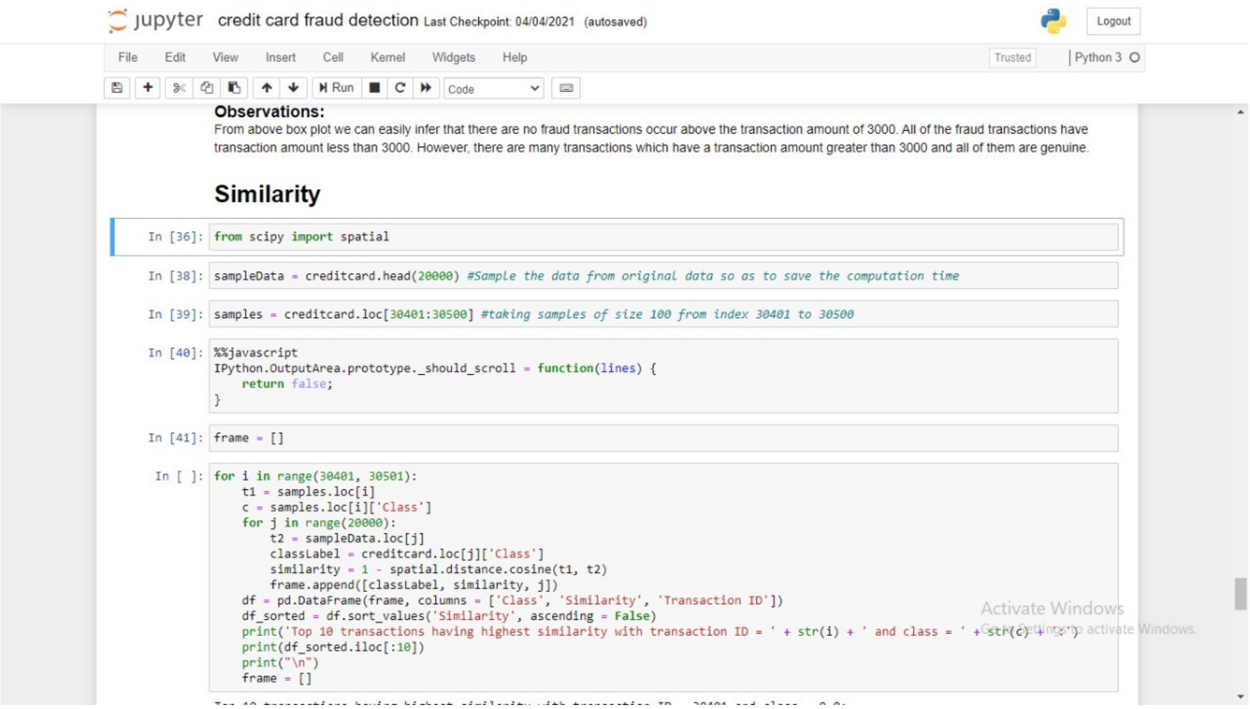

jupyter credit card fraud detection Last Checkpoint: 04/04/2021 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
175000
150000
125000
100000
75000
50000
25000
0
Time
0
1
Class
5000
4000
3000
2000
Amount
Activate Windows
Go to Settings to activate Windows.

Box Plots and Whiskers

In [34]:
```
sns.boxplot(x = 'Class', y = 'Time', data = creditcard)
plt.show()
```

Observations:
By looking at the above box plot we can say that both fraud & genuine transactions occur throughout time and there is no distinction between them.

In [35]:
```
sns.boxplot(x = 'Class', y = 'Amount', data = creditcard)
plt.ylim(0, 5000)
plt.show()
```


jupyter credit card fraud detection Last Checkpoint: 04/04/2021 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Activate Windows
Go to Settings to activate Windows.

Observations:

From above box plot we can easily infer that there are no fraud transactions occur above the transaction amount of 3000. All of the fraud transactions have transaction amount less than 3000. However, there are many transactions which have a transaction amount greater than 3000 and all of them are genuine.

Similarity

```
In [36]: from scipy import spatial
```

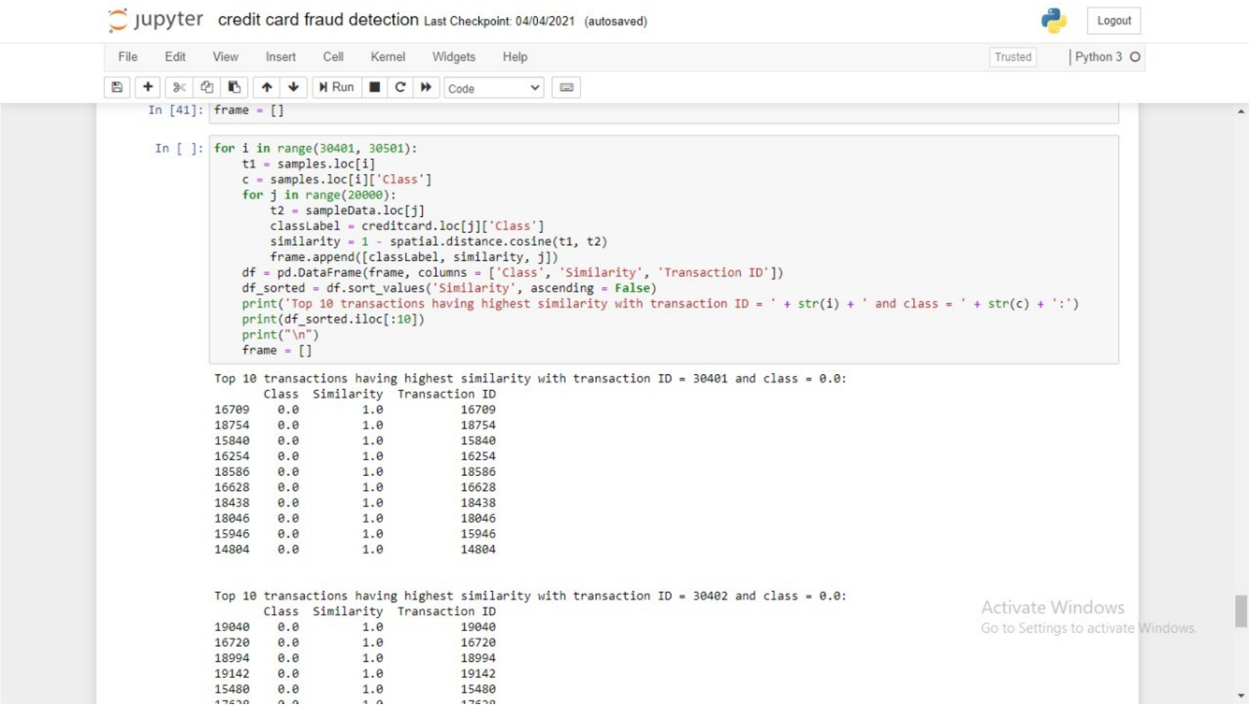
```
In [38]: sampleData = creditcard.head(20000) #Sample the data from original data so as to save the computation time
```

```
In [39]: samples = creditcard.loc[30401:30500] #taking samples of size 100 from index 30401 to 30500
```

```
In [40]: %%javascript
         IPython.OutputArea.prototype._should_scroll = function(lines) {
             return false;
         }
```

```
In [41]: frame = []
```

```
In [ ]: for i in range(30401, 30501):
            t1 = samples.loc[i]
            c = samples.loc[i]['Class']
            for j in range(20000):
                t2 = sampleData.loc[j]
                classLabel = creditcard.loc[j]['Class']
                similarity = 1 - spatial.distance.cosine(t1, t2)
                frame.append([classLabel, similarity, j])
            df = pd.DataFrame(frame, columns = ['Class', 'Similarity', 'Transaction ID'])
            df_sorted = df.sort_values('Similarity', ascending = False)
            print('Top 10 transactions having highest similarity with transaction ID = ' + str(i) + ' and class = ' + str(c) + ':')
            print(df_sorted.iloc[:10])
            print("\n")
            frame = []
```


jupyter credit card fraud detection Last Checkpoint: 04/04/2021 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Activate Windows
Go to Settings to activate Windows.

```
In [41]: frame = []
```

```
In [ ]: for i in range(30401, 30501):
            t1 = samples.loc[i]
            c = samples.loc[i]['Class']
            for j in range(20000):
                t2 = sampleData.loc[j]
                classLabel = creditcard.loc[j]['Class']
                similarity = 1 - spatial.distance.cosine(t1, t2)
                frame.append([classLabel, similarity, j])
            df = pd.DataFrame(frame, columns = ['Class', 'Similarity', 'Transaction ID'])
            df_sorted = df.sort_values('Similarity', ascending = False)
            print('Top 10 transactions having highest similarity with transaction ID = ' + str(i) + ' and class = ' + str(c) + ':')
            print(df_sorted.iloc[:10])
            print("\n")
            frame = []
```

```
Top 10 transactions having highest similarity with transaction ID = 30401 and class = 0.0:
       Class  Similarity  Transaction ID
16709    0.0         1.0           16709
18754    0.0         1.0           18754
15840    0.0         1.0           15840
16254    0.0         1.0           16254
18586    0.0         1.0           18586
16628    0.0         1.0           16628
18438    0.0         1.0           18438
18046    0.0         1.0           18046
15946    0.0         1.0           15946
14804    0.0         1.0           14804


Top 10 transactions having highest similarity with transaction ID = 30402 and class = 0.0:
       Class  Similarity  Transaction ID
19040    0.0         1.0           19040
16720    0.0         1.0           16720
18994    0.0         1.0           18994
19142    0.0         1.0           19142
15480    0.0         1.0           15480
```

FUTURE OF PROJECT:

Beyond the technical approach of this problem it is important to note the important role and the challenges regulation around the world.The heterogeneity across regulatory frameworks in different countries poses great challenges for many industries to detect fraud. For instance, in countries where Electronic privacy laws are too strict it is harder to gather data, detect fraudulent patterns, and thus track and identify fraudsters. To learn more about the specific tools that are in the process of being implemented to combat fraud.

CONCLUSION

Online transaction has increased greatly and due to its simple and fast approach customer prefers it over cash payment. Companies and organization involve in ecommerce provide attractive discounts and offers for online payment. This has made customers to switch from the cash mode to online mode.

REFERENCES:

www.youtube.com

www.geeeksforgeeks.com

www.analyticvidhya.com

www.tutorialspoint.com

www.learnpython.org

www.simplileaen.com

www.edurekha.com

BOOK-

:Hands–On Machine Learning with Scikit–Learn and TensorFlow 2e: Concepts, Tools, and Techniques to Build Intelligent Systems
: Machine Learning for Absolute Beginners

FACULTY GUILDELINE:

MR.VINAY AGARWAL

(ASSISTANT PROFESSOR)