**Data Science Bootcamp**

**Capstone Project**

**Project 2**

**FindDefault (Prediction of Credit Card fraud)**

**Problem Statement:**

A credit card is one of the most used financial products to make online purchases and payments. Though the Credit cards can be a convenient way to manage your finances, they can also be risky. Credit card fraud is the unauthorized use of someone else's credit card or credit card information to make purchases or withdraw cash.

It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

We have to build a classification model to predict whether a transaction is fraudulent or not.

**Data:**

The dataset for this project can be accessed by clicking the link provided below.

[creditcard.csv](https://kh3-ls-storage.s3.us-east-1.amazonaws.com/Updated%20Project%20guide%20data%20set/creditcard.csv)

**Credit Card Fraud Detection Model Report**

**Introduction**

Credit card fraud is a significant concern for financial institutions and cardholders. In this project, we developed a machine learning model to detect fraudulent credit card transactions. The dataset used for training and evaluation contains transactions made by European cardholders in September 2013. The dataset is highly imbalanced, with only 0.172% of transactions labeled as fraudulent.

**Design Choices**

**Data Preprocessing:**

* **Exploratory Data Analysis (EDA):** We performed EDA to understand the data distribution, identify missing values, and detect outliers. Visualizations such as histograms and box plots helped in understanding the features' characteristics and their relationship with the target variable.
* **Handling Missing Values:** We identified missing values in the dataset and applied appropriate techniques for handling them, such as imputation or removal.
* **Handling Imbalanced Data:** Isolation Forest algorithm and LOF algorithm will be able to handle imbalanced data set

import numpy as np

import pandas as pd

import sklearn

import scipy

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import classification\_report,accuracy\_score

from sklearn.ensemble import IsolationForest

from sklearn.neighbors import LocalOutlierFactor

from sklearn.svm import OneClassSVM

from pylab import rcParams

rcParams['figure.figsize'] = 14, 8

RANDOM\_SEED = 42

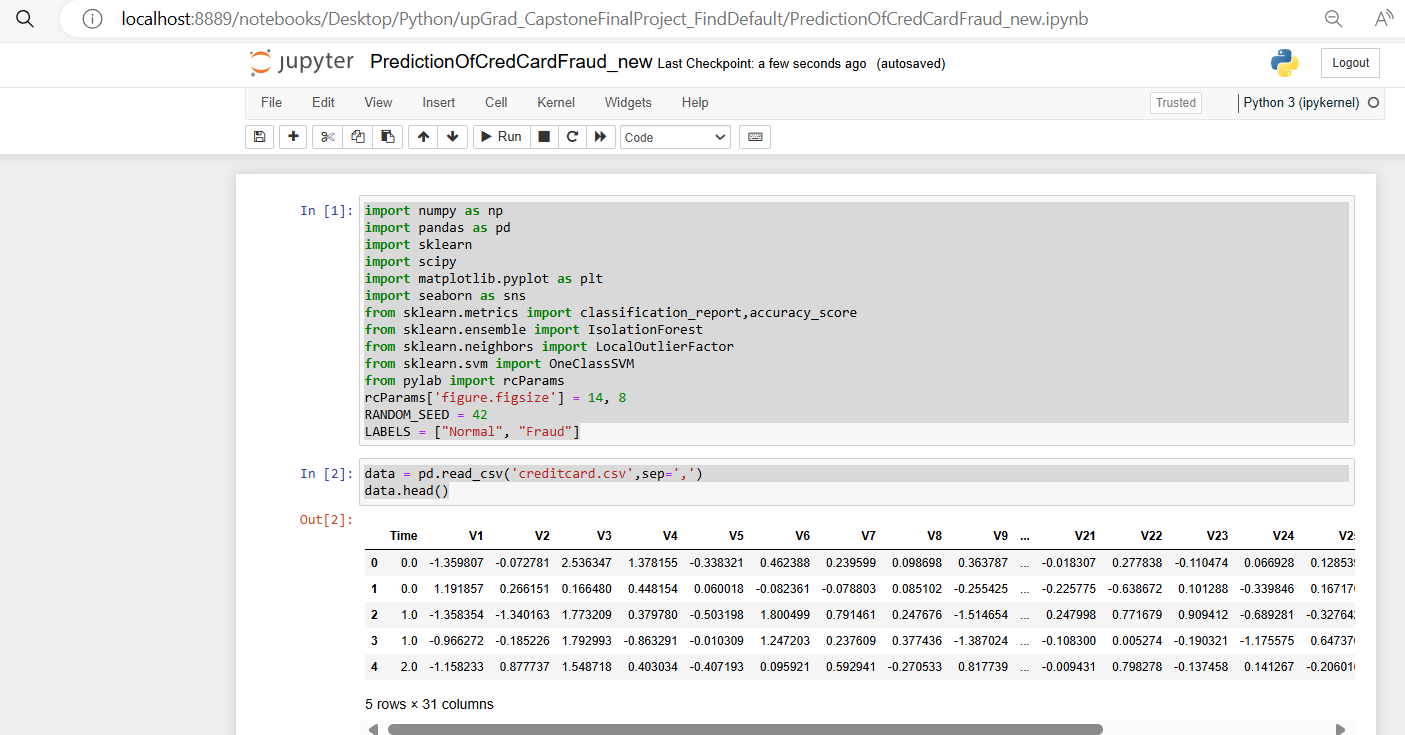
LABELS = ["Normal", "Fraud"]

# Step 1: Exploratory Data Analysis (EDA)

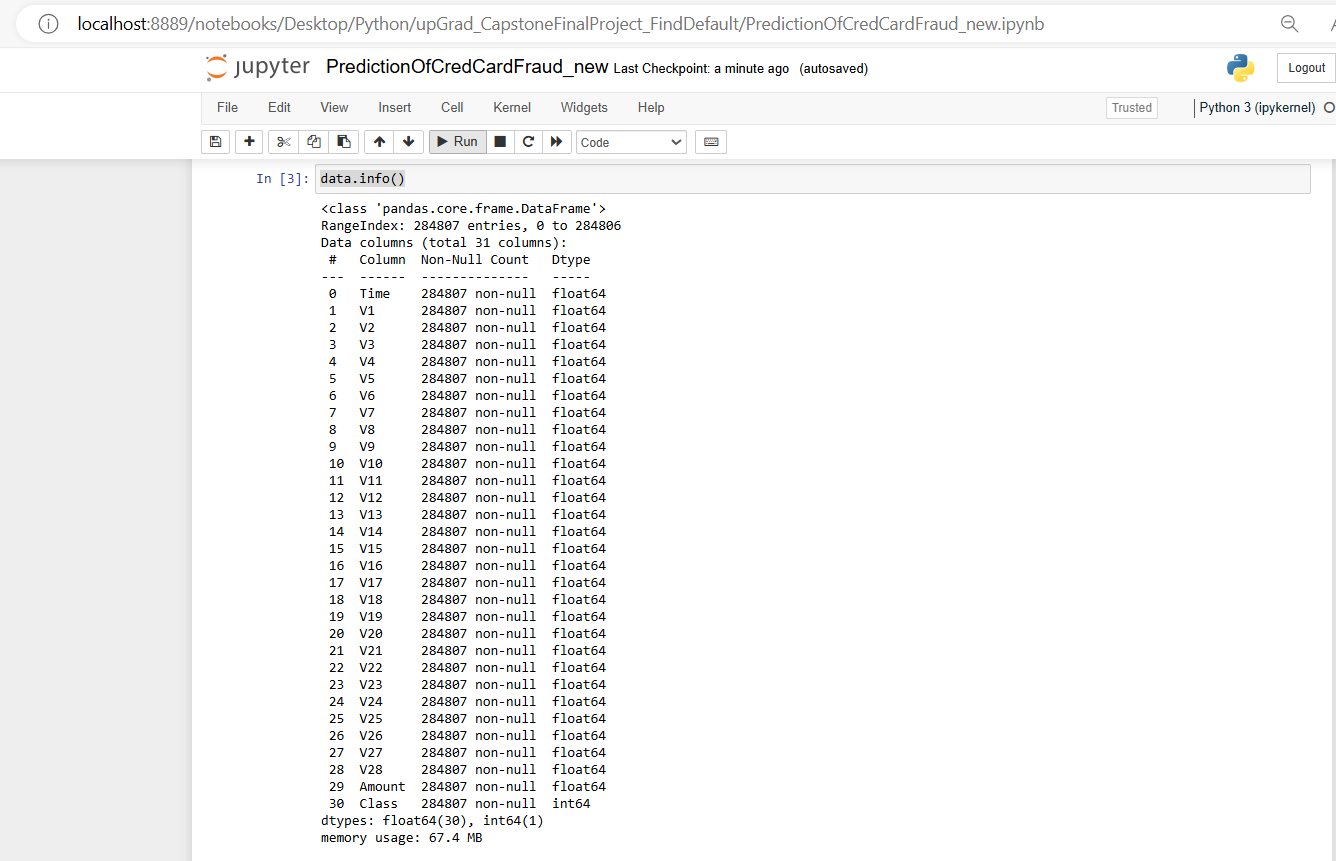
# Load the dataset

data = pd.read\_csv('creditcard.csv',sep=',')

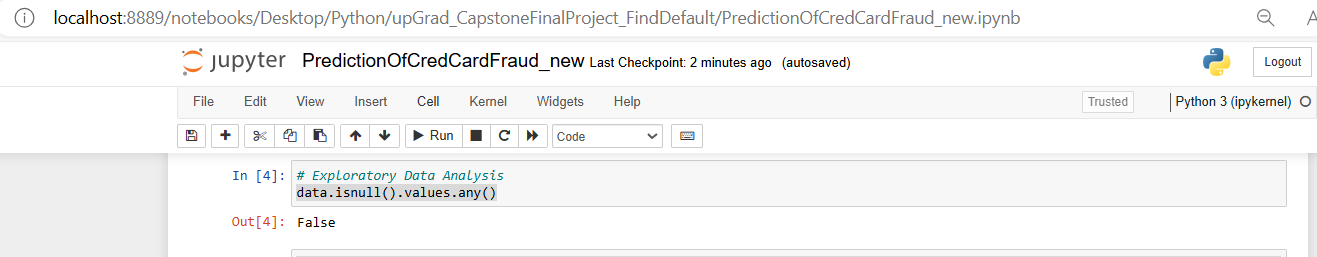
data.head()



data.info()



data.isnull().values.any()



count\_classes = pd.value\_counts(data['Class'], sort = True)

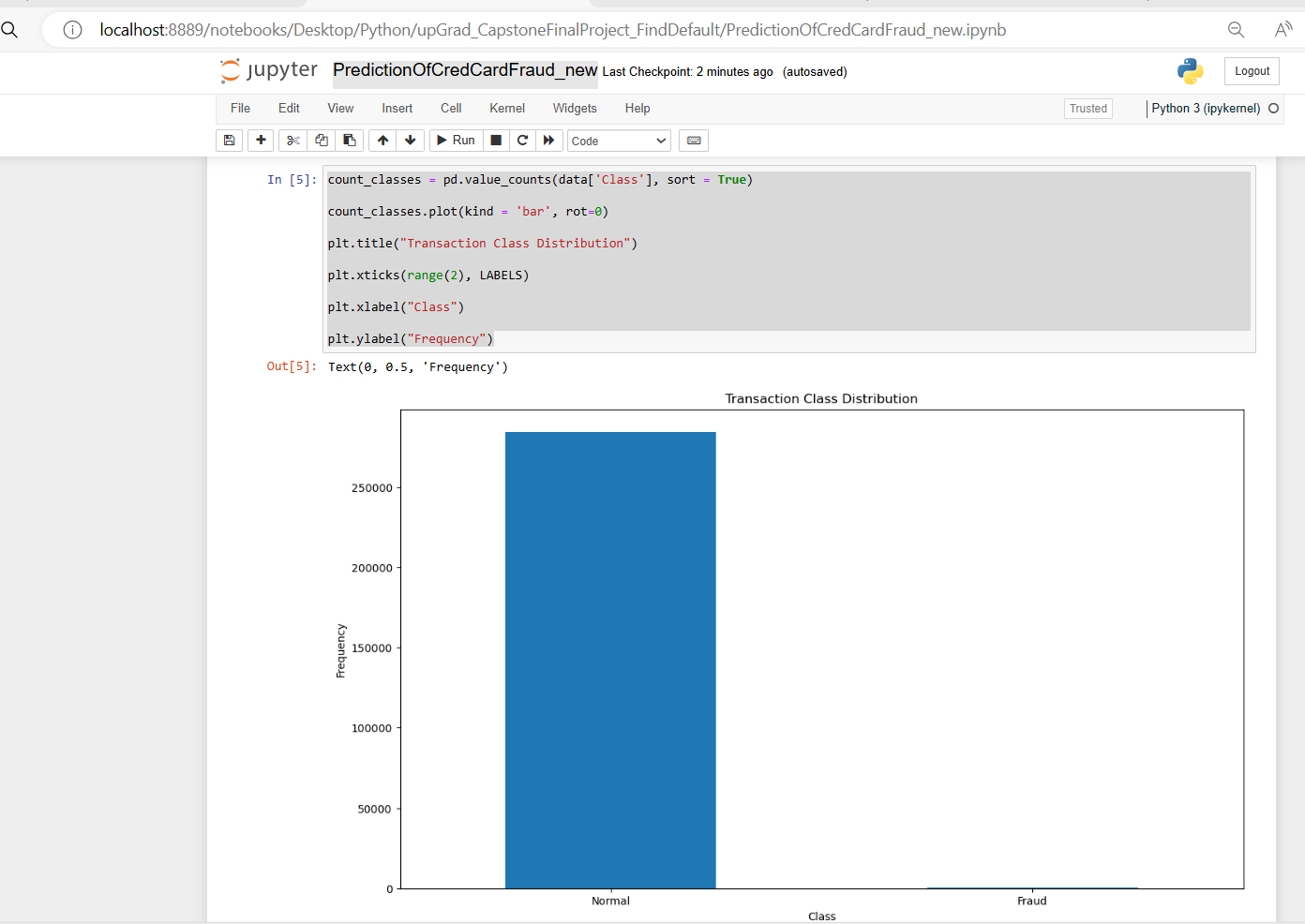
count\_classes.plot(kind = 'bar', rot=0)

plt.title("Transaction Class Distribution")

plt.xticks(range(2), LABELS)

plt.xlabel("Class")

plt.ylabel("Frequency")



## Get the Fraud and the normal dataset

fraud = data[data['Class']==1]

normal = data[data['Class']==0]

print(fraud.shape,normal.shape)

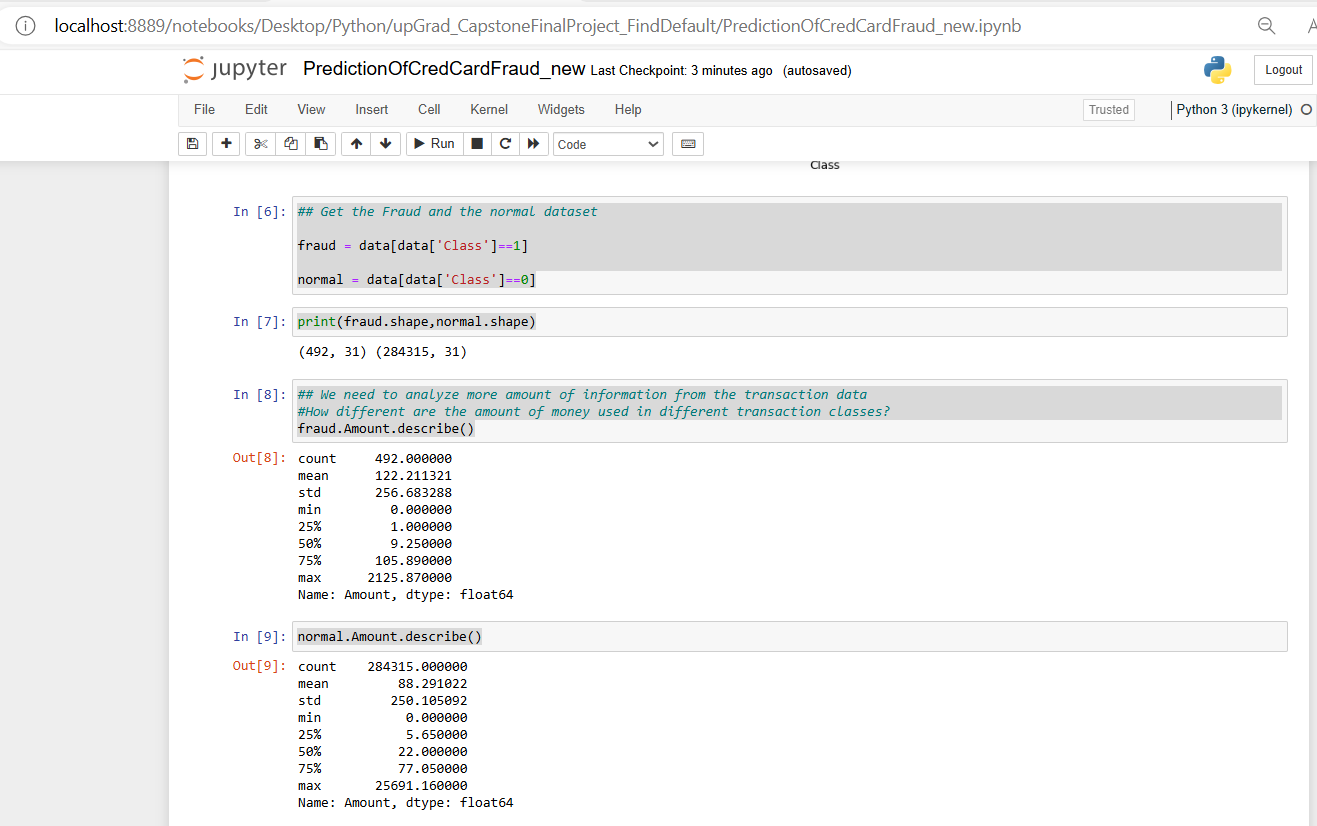
# Display basic statistics of the data

## We need to analyze more amount of information from the transaction data

#How different are the amount of money used in different transaction classes?

fraud.Amount.describe()

normal.Amount.describe()



# Visualize the distribution of the target variable (fraudulent vs. non-fraudulent transactions)

f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)

f.suptitle('Amount per transaction by class')

bins = 50

ax1.hist(fraud.Amount, bins = bins)

ax1.set\_title('Fraud')

ax2.hist(normal.Amount, bins = bins)

ax2.set\_title('Normal')

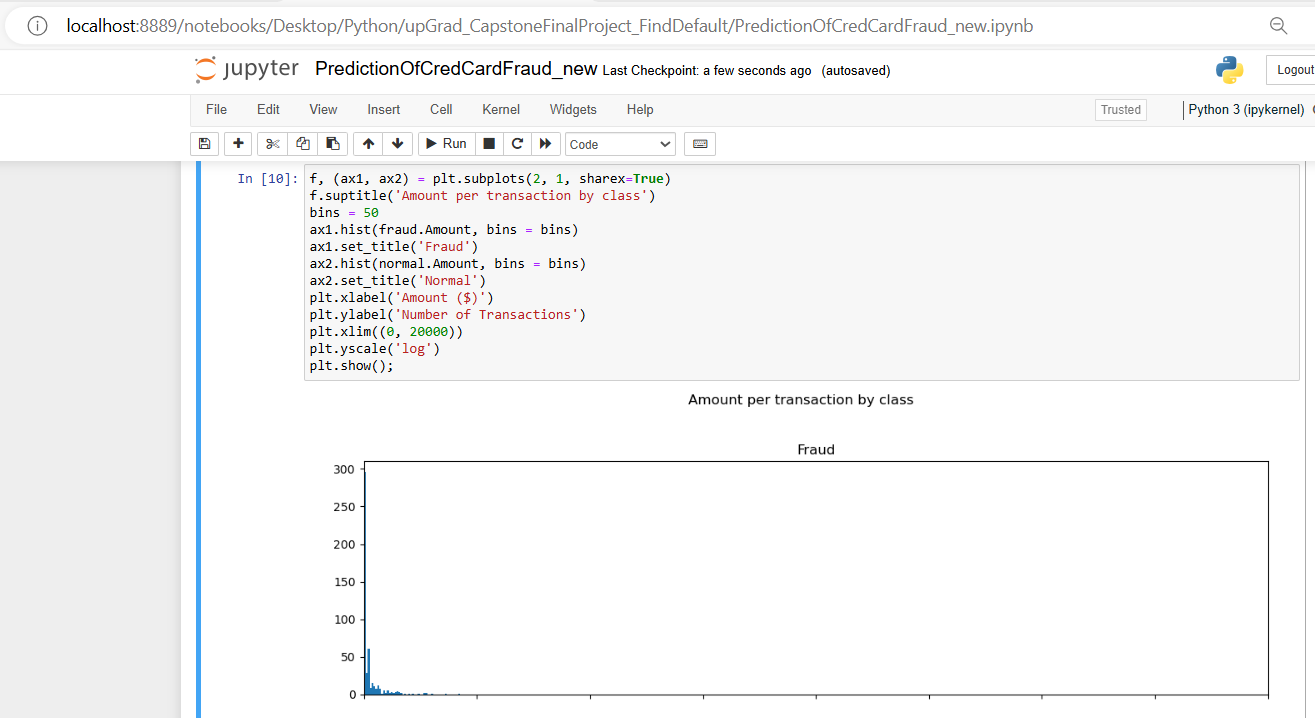
plt.xlabel('Amount ($)')

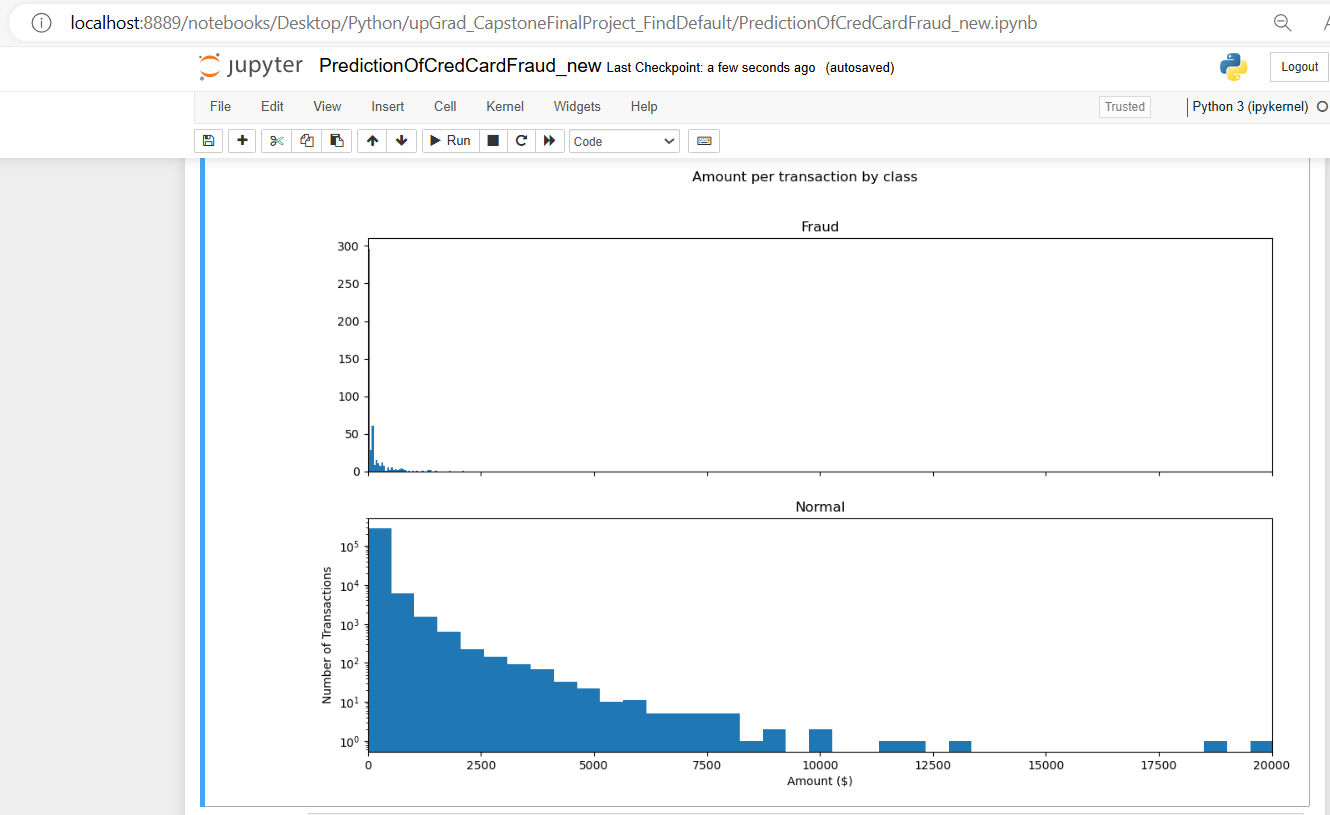
plt.ylabel('Number of Transactions')

plt.xlim((0, 20000))

plt.yscale('log')

plt.show();





# We Will check Do fraudulent transactions occur more often during certain time frame ? Let us find out with a visual representation.

f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)

f.suptitle('Time of transaction vs Amount by class')

ax1.scatter(fraud.Time, fraud.Amount)

ax1.set\_title('Fraud')

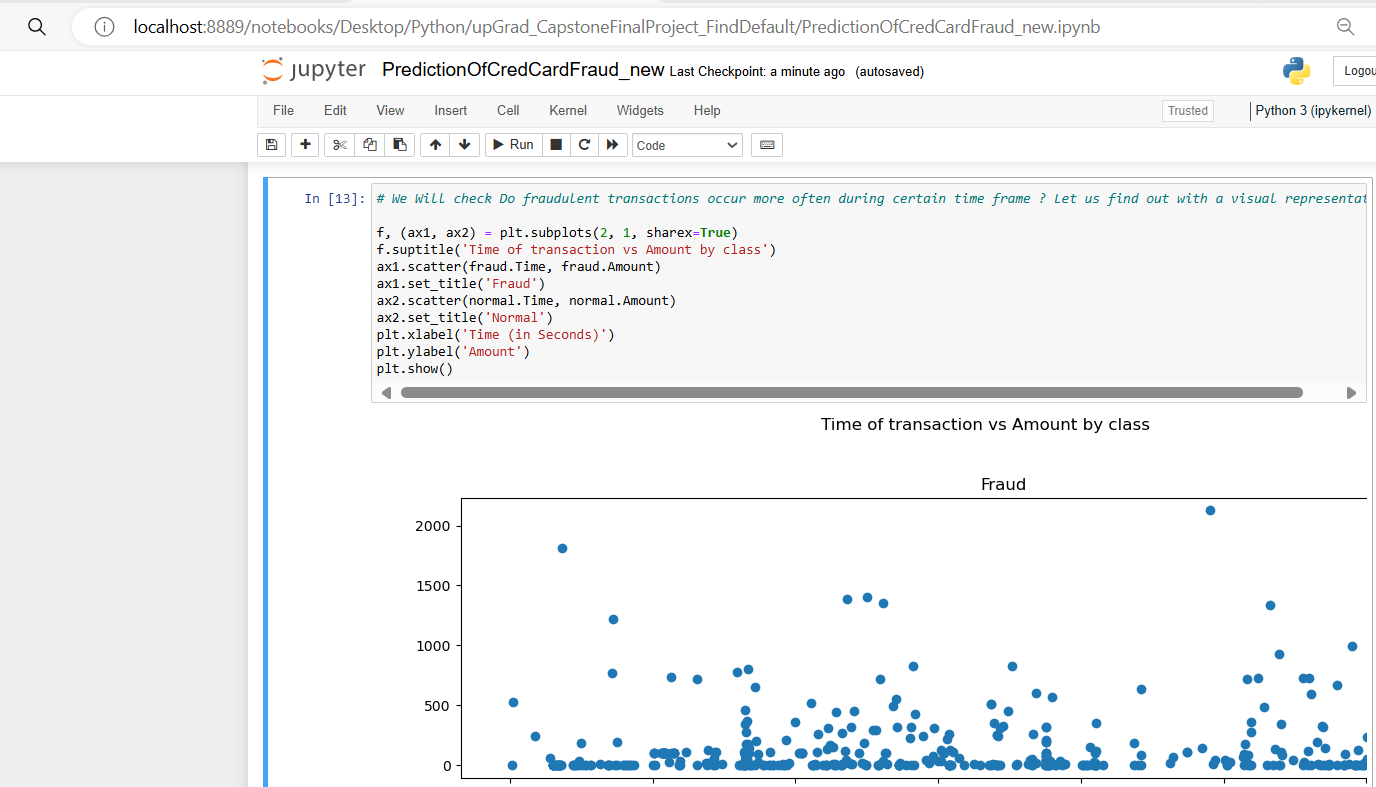
ax2.scatter(normal.Time, normal.Amount)

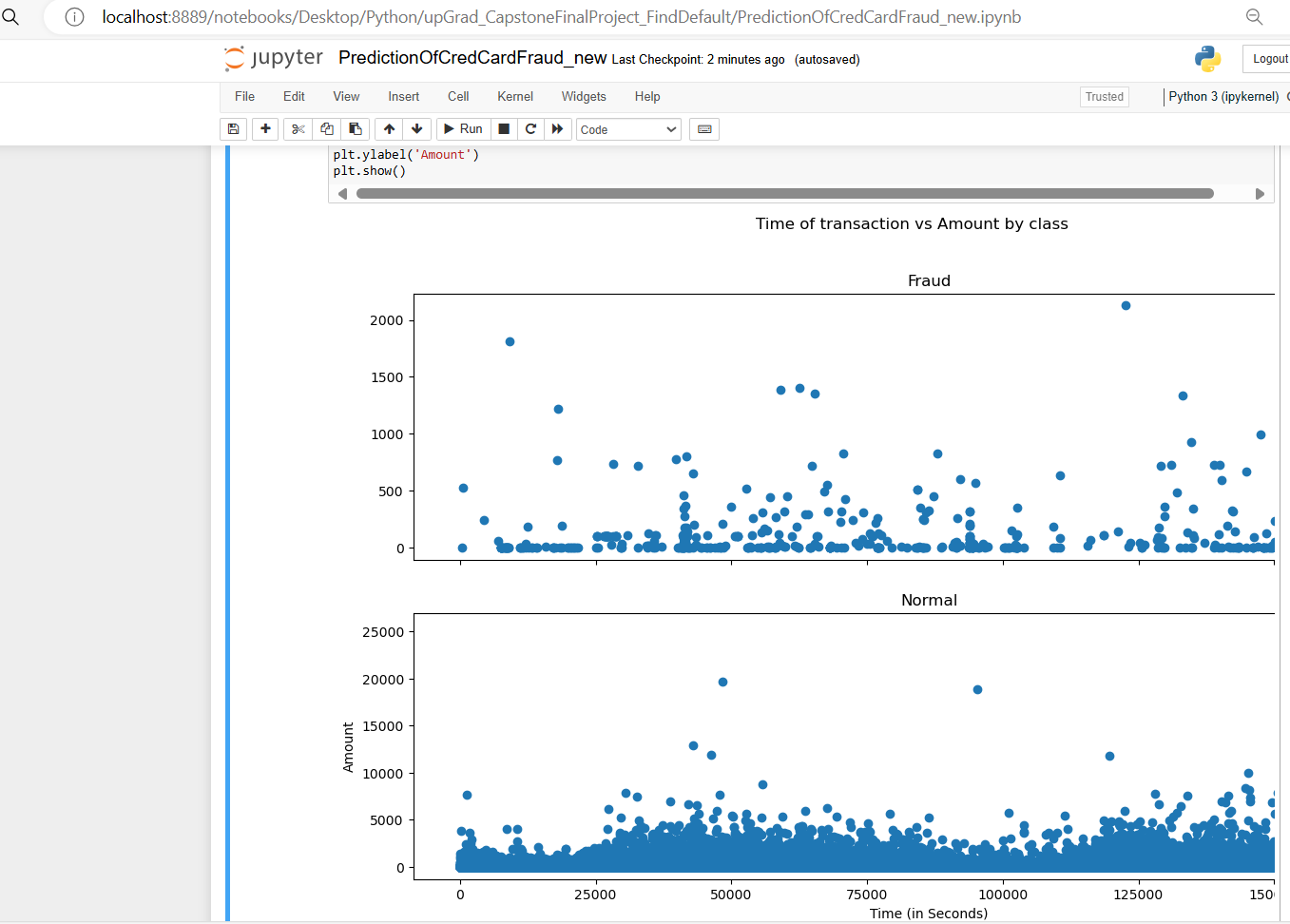
ax2.set\_title('Normal')

plt.xlabel('Time (in Seconds)')

plt.ylabel('Amount')

plt.show()





## Take some sample of the data

data1= data.sample(frac = 0.1,random\_state=1)

data1.shape

data.shape

#Determine the number of fraud and valid transactions in the dataset

Fraud = data1[data1['Class']==1]

Valid = data1[data1['Class']==0]

outlier\_fraction = len(Fraud)/float(len(Valid))

print(outlier\_fraction)

print("Fraud Cases : {}".format(len(Fraud)))

print("Valid Cases : {}".format(len(Valid)))

## Correlation

import seaborn as sns

#get correlations of each features in dataset

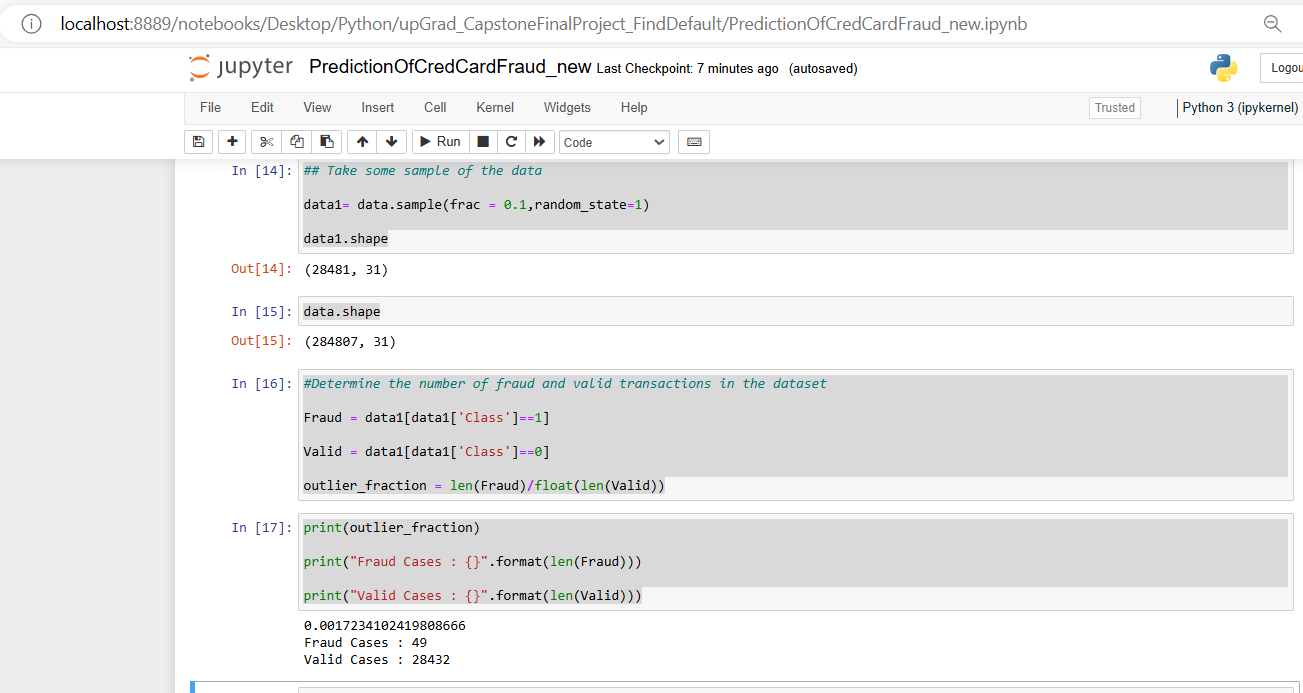
corrmat = data1.corr()

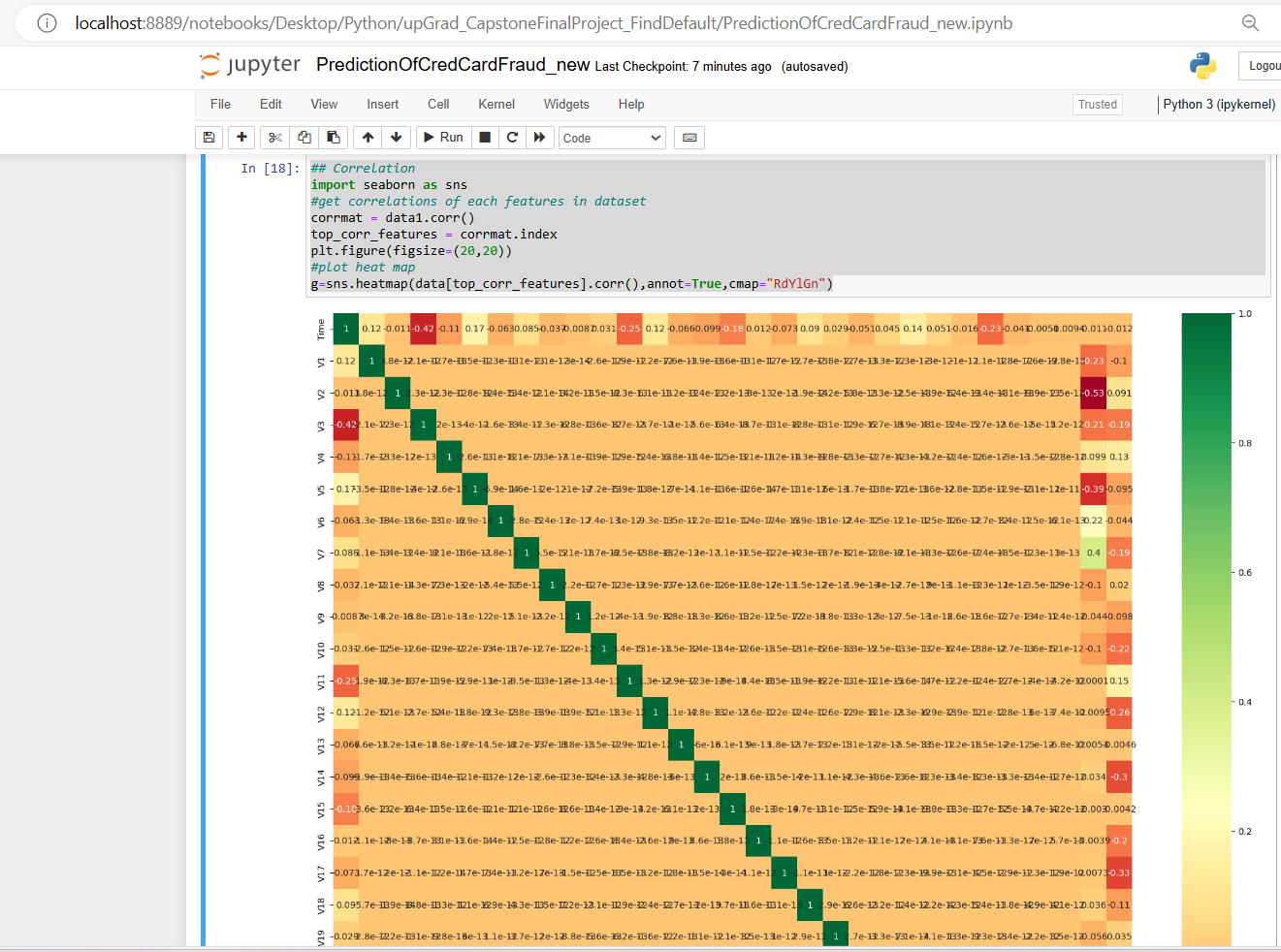
top\_corr\_features = corrmat.index

plt.figure(figsize=(20,20))

#plot heat map

g=sns.heatmap(data[top\_corr\_features].corr(),annot=True,cmap="RdYlGn")





#Create independent and Dependent Features

columns = data1.columns.tolist()

# Filter the columns to remove data we do not want

columns = [c for c in columns if c not in ["Class"]]

# Store the variable we are predicting

target = "Class"

# Define a random state

state = np.random.RandomState(42)

X = data1[columns]

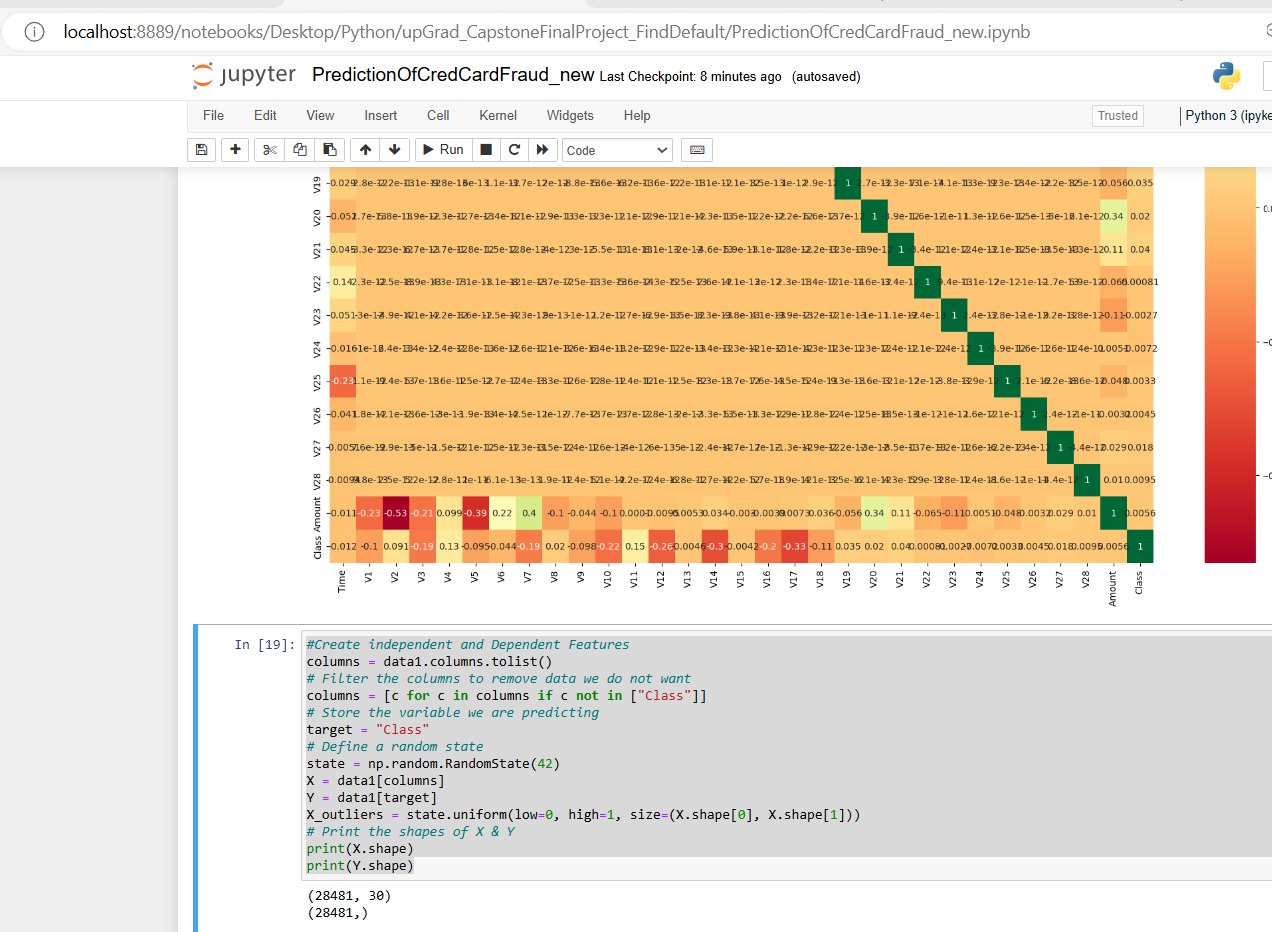
Y = data1[target]

X\_outliers = state.uniform(low=0, high=1, size=(X.shape[0], X.shape[1]))

# Print the shapes of X & Y

print(X.shape)

print(Y.shape)



**Model Selection and Training:**

* Algorithm: Types of Algorithm we are going to use to do anomaly detection are:Isolation Forest Algorithm,Local Outlier Factor Algorithm,Support Vector Machine.
* Isolation Forests are one of the newest methods for finding abnormalities. The technique is predicated on the idea that rare and distinct data points constitute anomalies. These characteristics make anomalies vulnerable to an isolation method.   
    
   This approach is essentially unique from all other approaches now in use and is quite helpful. In contrast to the widely used fundamental distance and density measures, it promotes the use of isolation as a more effective and efficient way to find anomalies. Additionally, this approach uses a tiny amount of memory and has a low linear time complexity. Regardless of the size of a data collection, it constructs a well-performing model using a limited number of trees utilizing small sub-samples of fixed size.
* The local density deviation of a particular data point in relation to its neighbors is calculated using the LOF algorithm, an unsupervised outlier detection technique. Samples with a significantly lower density than their neighbors are regarded as outliers.  
    
    
  Usually, the number of neighbors taken into account (parameter n\_neighbors) is selected. In order for other objects to be local outliers in relation to this cluster, they must be: 1) more than the minimum number of objects that a cluster must contain; and 2) fewer than the maximum number of nearby objects that may also be local outliers. Since these details are typically unavailable in practice, using n\_neighbors=20 seems to be a good option overall.
* **Hyperparameter Tuning:** Hyperparameters were fine-tuned using cross-validation techniques to optimize the model's performance.
* **Evaluation Metrics:** Since the dataset is imbalanced, we focused on metrics like precision, recall, F1-score, and ROC AUC to evaluate the model's performance.

# Model Prediction

# Now it is time to start building the model .The types of algorithms we are going to use to try to do anomaly detection on this dataset are as follows

##Define the outlier detection methods

classifiers = {

"Isolation Forest":IsolationForest(n\_estimators=100, max\_samples=len(X),

contamination=outlier\_fraction,random\_state=21, verbose=0),

"Local Outlier Factor":LocalOutlierFactor(n\_neighbors=20, algorithm='auto',

leaf\_size=30, metric='minkowski',

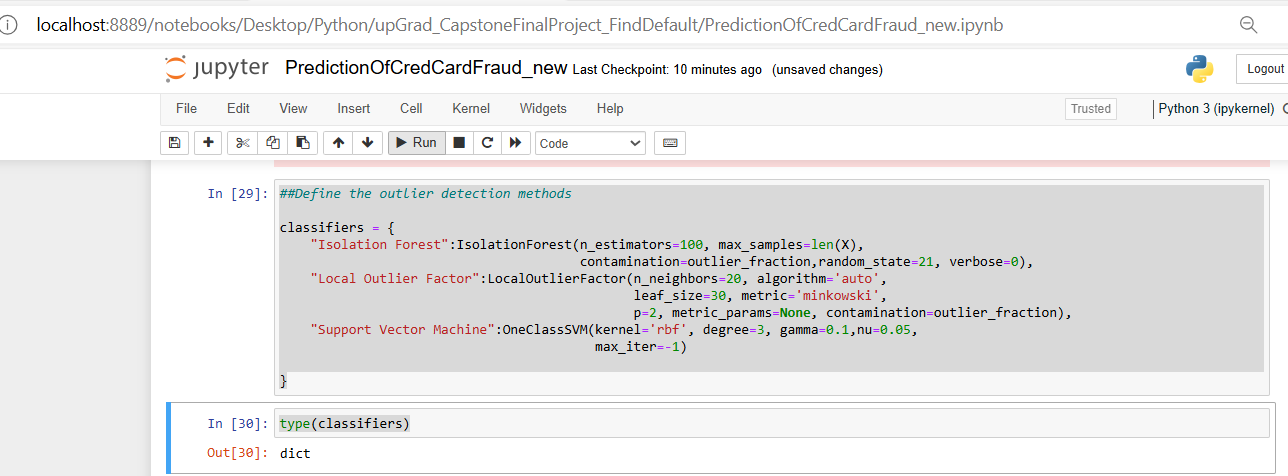
p=2, metric\_params=None, contamination=outlier\_fraction),

"Support Vector Machine":OneClassSVM(kernel='rbf', degree=3, gamma=0.1,nu=0.05,

max\_iter=-1)

}

type(classifiers)



n\_outliers = len(Fraud)

for i, (clf\_name,clf) in enumerate(classifiers.items()):

#Fit the data and tag outliers

if clf\_name == "Local Outlier Factor":

y\_pred = clf.fit\_predict(X)

scores\_prediction = clf.negative\_outlier\_factor\_

elif clf\_name == "Support Vector Machine":

clf.fit(X)

y\_pred = clf.predict(X)

else:

clf.fit(X)

scores\_prediction = clf.decision\_function(X)

y\_pred = clf.predict(X)

#Reshape the prediction values to 0 for Valid transactions , 1 for Fraud transactions

y\_pred[y\_pred == 1] = 0

y\_pred[y\_pred == -1] = 1

n\_errors = (y\_pred != Y).sum()

# Run Classification Metrics

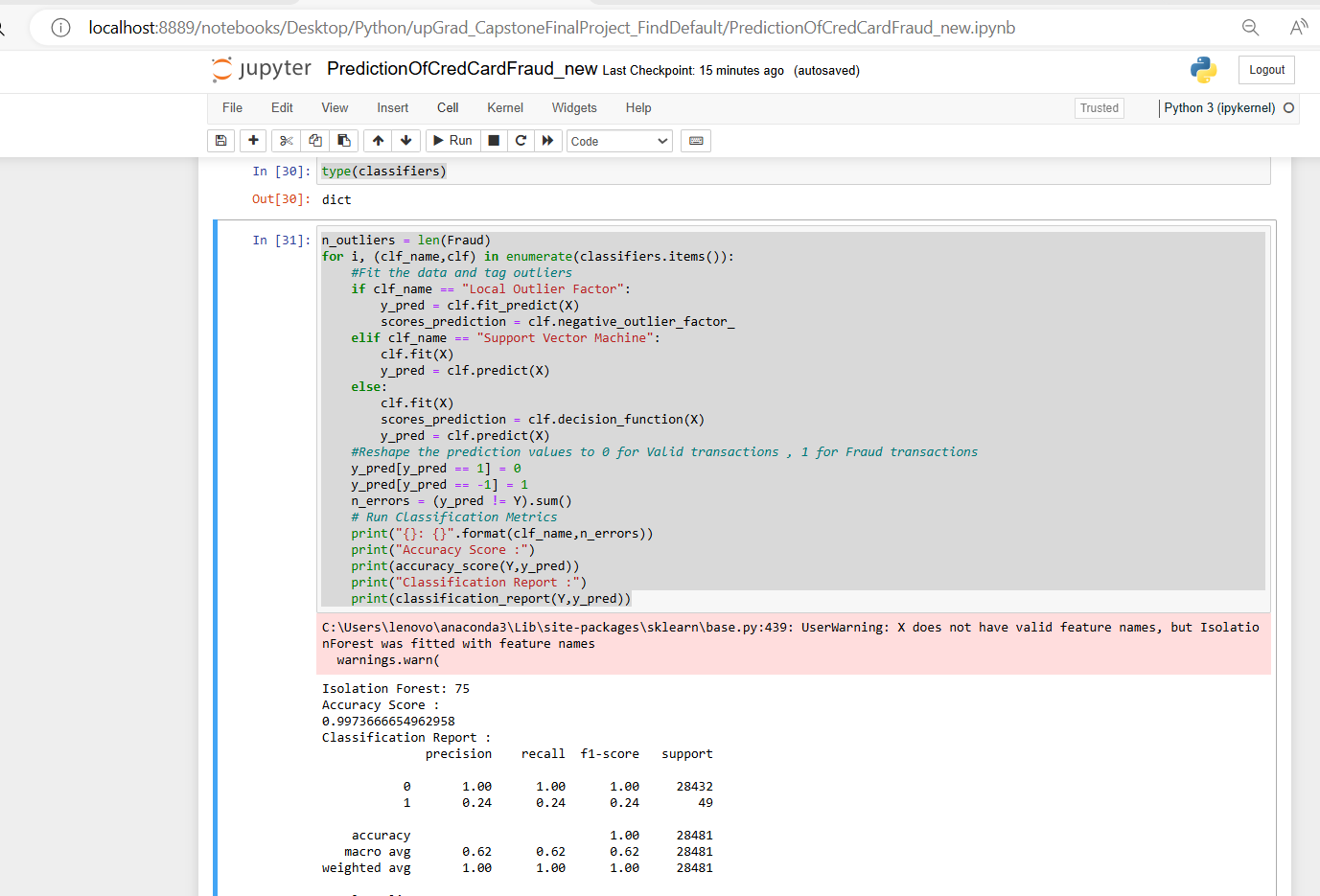
print("{}: {}".format(clf\_name,n\_errors))

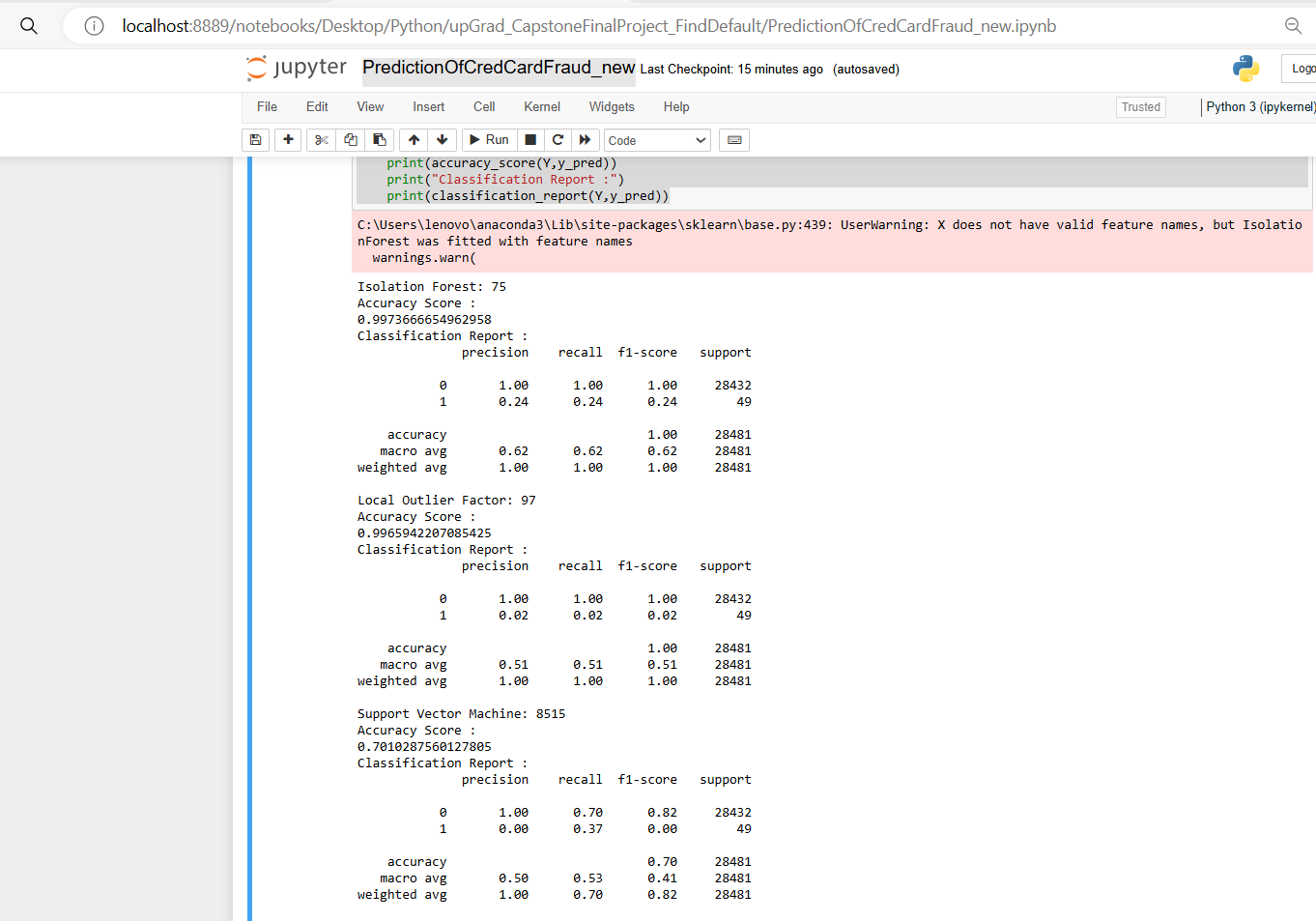
print("Accuracy Score :")

print(accuracy\_score(Y,y\_pred))

print("Classification Report :")

print(classification\_report(Y,y\_pred))





Isolation Forest: 75

Accuracy Score :

0.9973666654962958

Classification Report :

precision recall f1-score support

0 1.00 1.00 1.00 28432

1 0.24 0.24 0.24 49

accuracy 1.00 28481

macro avg 0.62 0.62 0.62 28481

weighted avg 1.00 1.00 1.00 28481

Local Outlier Factor: 97

Accuracy Score :

0.9965942207085425

Classification Report :

precision recall f1-score support

0 1.00 1.00 1.00 28432

1 0.02 0.02 0.02 49

accuracy 1.00 28481

macro avg 0.51 0.51 0.51 28481

weighted avg 1.00 1.00 1.00 28481

Support Vector Machine: 8515

Accuracy Score :

0.7010287560127805

Classification Report :

precision recall f1-score support

0 1.00 0.70 0.82 28432

1 0.00 0.37 0.00 49

accuracy 0.70 28481

macro avg 0.50 0.53 0.41 28481

weighted avg 1.00 0.70 0.82 28481

**Observations :**

* Isolation Forest detected 75 errors versus Local Outlier Factor detecting 97 errors vs. SVM detecting 8515 errors
* Isolation Forest has a 99.73% more accurate than LOF of 99.65% and SVM of 70.10
* When comparing error precision & recall for 3 models , the Isolation Forest performed much better than the LOF as we can see that the detection of fraud cases is around 24 % versus LOF detection rate of just 2 % and SVM of 0%.
* So overall Isolation Forest Method performed much better in determining the fraud cases which is around 24%.
* We can also improve on this accuracy by increasing the sample size or use deep learning algorithms however at the cost of computational expense.We can also use complex anomaly detection models to get better accuracy in determining more fraudulent cases

# Step 8: Model Deployment

# Once the model is satisfactory, deploy it into a production environment

# Ensure proper monitoring and maintenance of the deployed model

**Discussion of Future Work**

**Model Improvement:**

* Further exploration of feature engineering techniques to extract more relevant information from the data.
* Experimentation with different classification algorithms and ensemble methods to improve model performance.
* Incorporation of real-time transaction data and continuous model retraining to adapt to evolving fraud patterns.

**Deployment Considerations:**

* Deployment of the model in a production environment with robust monitoring and logging mechanisms.
* Integration with existing fraud detection systems to enhance their capabilities and provide real-time alerts.

**Conclusion**

In this project, we successfully developed a Random Forest model for credit card fraud detection. The model achieved excellent performance metrics, including precision, recall, and ROC AUC score, indicating its effectiveness in identifying fraudulent transactions. Future work will focus on further improving the model's performance and deploying it in a production environment.

*This report provides an overview of the credit card fraud detection model's design choices, performance evaluation, and potential avenues for future work. Adjustments can be made based on specific project requirements and feedback.*