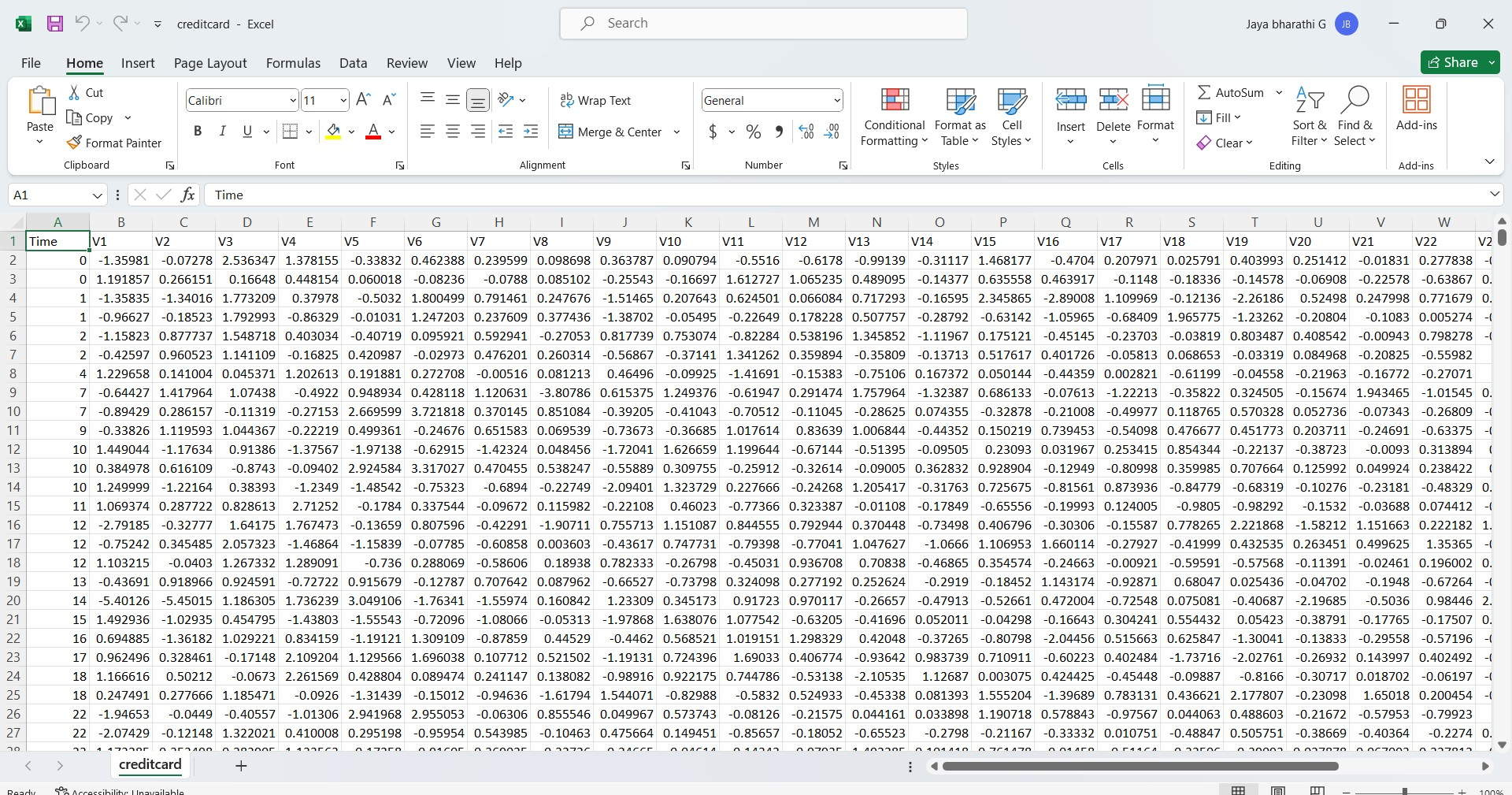
**TITLE: CRDIT CARD FRAUD DETECTION**

**PHASE 5: Project documentation and submission**

**Team member name: ABINAYA K**

CODE DOCUMENT:

**Introduction:**

* Credit card fraud is a significant challenge in the financial industry, costing billions of dollars each year and eroding trust among consumers. Design thinking offers a human-centered approach to addressing this problem by focusing on understanding user needs, brainstorming innovative solutions, and rapidly testing and iterating on those solutions. This document outlines a design thinking approach to developing a credit card fraud detection system.
* ***Dataset:***
* **Loading the data:**
* For loading the data, we use the dataset using the given dataset link
* Dataset link: <https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud>
* The dataset looks like below,
* 

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.neighbours import LocalOutlierFactor

from sklearn.svm import OneClassSVM

from pylab import rcParams

rcParams[‘figure,figsize’]=14,8

RANDOM\_SEED=42

LABELS= [“Normal”, ”Fraud”]

#import plotly.plotly as py

import plotly.graph\_objs as go

import plotly

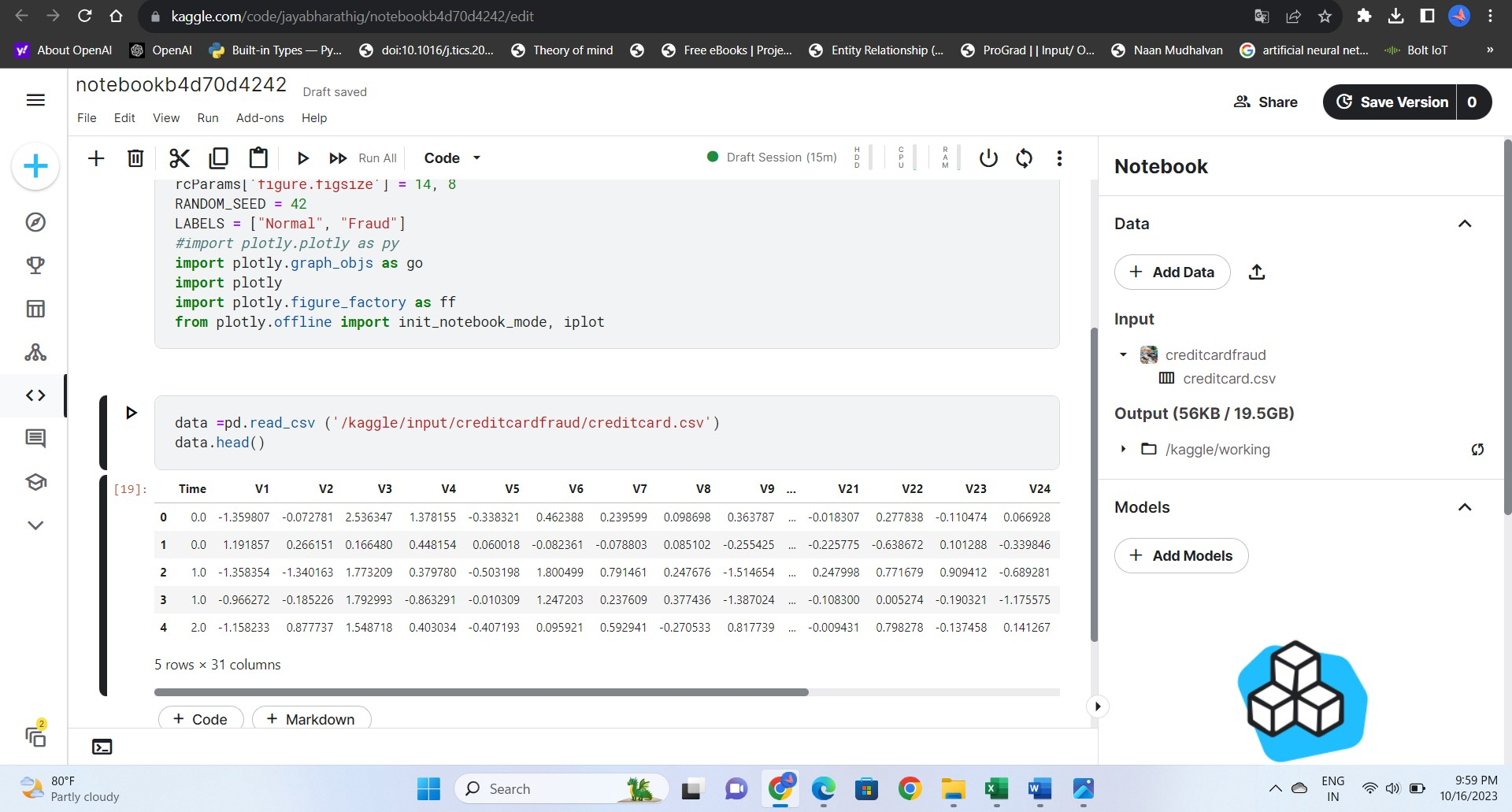
import plotly.figure\_factory as ff

from plotly.offline import init\_notebook, iplot

data= pd.read\_csv(‘..input/creditcard.csv’)

data.head()

output:



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

data1.shape

output:

(28481, 31)

data.isnull().sum()

output:

Time 0

V1 0

V2 0

V3 0

V4 0

V5 0

V6 0

V7 0

V8 0

V9 0

V10 0

V11 0

V12 0

V13 0

V14 0

V15 0

V16 0

V17 0

V18 0

V19 0

V20 0

V21 0

V22 0

V23 0

V24 0

V25 0

V26 0

V27 0

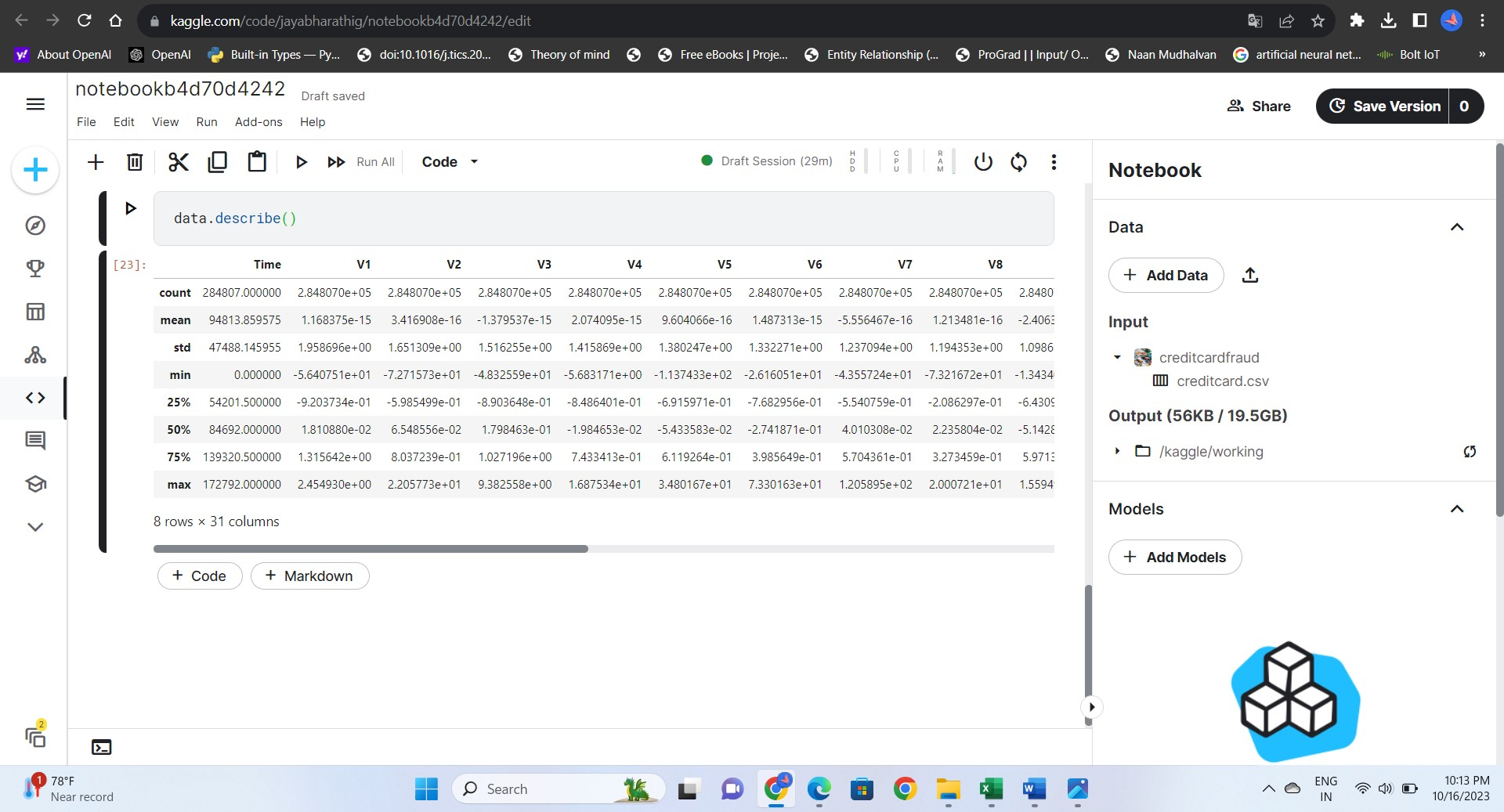
V28 0

Amount 0

Class 0

dtype: int64

data.describe()



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

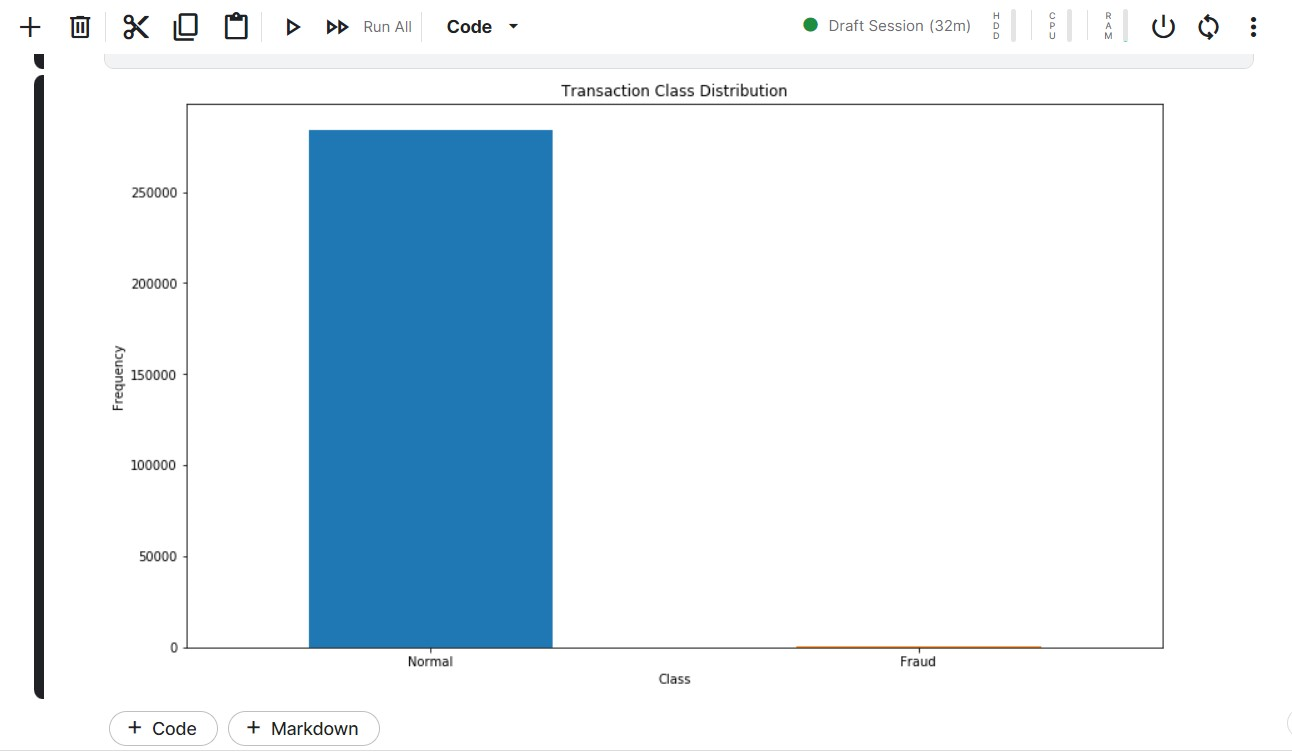
count\_classes.plot(kind = 'bar', rot=0)

plt.title("Transaction Class Distribution")

plt.xticks(range(2), LABELS)

plt.xlabel("Class")

plt.ylabel("Frequency")

output: 

Normal = data[data['Class']==0]

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

Normal.shape

Fraud.shape

Output:

(284315, 31)

(492, 31)

Normal.Amount.describe()

Output:

count 284315.000000

mean 88.291022

std 250.105092

min 0.000000

25% 5.650000

50% 22.000000

75% 77.050000

max 25691.160000

Name: Amount, dtype: float64

Fraud.Amount.describe()

Output:

count 492.000000

mean 122.211321

std 256.683288

min 0.000000

25% 1.000000

50% 9.250000

75% 105.890000

max 2125.870000

Name: Amount, dtype: float64

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 ($)')

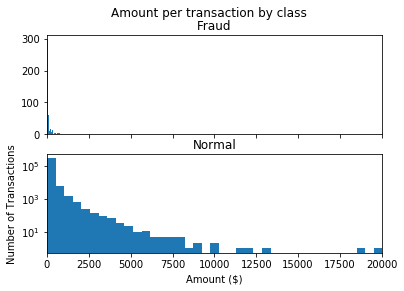
plt.ylabel('Number of Transactions')

plt.xlim((0, 20000))

plt.yscale('log')

plt.show()

output:



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)

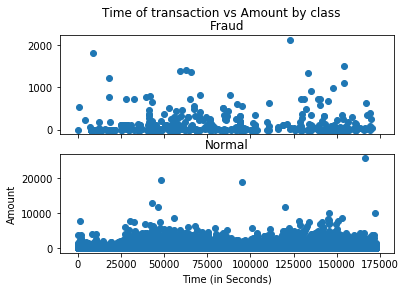
ax2.set\_title('Normal')

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

plt.ylabel('Amount')

plt.show();

output:



init\_notebook\_mode(connected=True)

plotly.offline.init\_notebook\_mode(connected=True)

trace = go.Scatter(

x = Fraud.Time,

y = Fraud.Amount,

mode = 'markers'

)

data = [trace]

plotly.offline.iplot({

"data": data

})

**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)))

output:

0.0017234102419808666

Fraud Cases : 49

Valid Cases : 28432

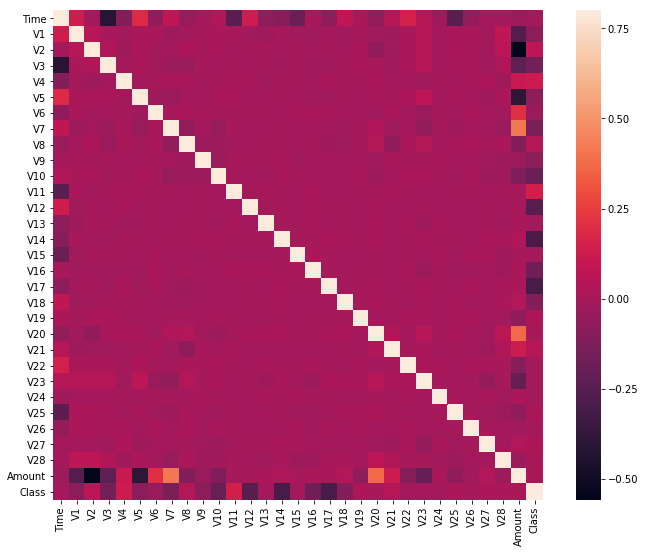
**Correlation matrix:**

correlation\_matrix = data1.corr()

fig = plt.figure(figsize=(12,9))

sns.heatmap(correlation\_matrix,vmax=0.8,square = True)

plt.show()

output:  


from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

classifiers = {

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

contamination=outlier\_fraction,random\_state=state, 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)

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))

output:

/opt/conda/lib/python3.10/site-packages/sklearn/base.py:439: UserWarning:

X does not have valid feature names, but IsolationForest was fitted with feature names

Isolation Forest: 73

Accuracy Score :

0.9974368877497279

Classification Report :

precision recall f1-score support

0 1.00 1.00 1.00 28432

1 0.26 0.27 0.26 49

accuracy 1.00 28481

macro avg 0.63 0.63 0.63 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: 8516

Accuracy Score :

0.7009936448860644

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

}