**Title: CREDIT CARD FRAUD DETECTION**

PHASE 2 SUBMISSION DOCUMENT



Introduction:

* The Credit Card Fraud Detection Project is a critical endeavor aimed at developing a robust system to identify and prevent fraudulent transactions within the credit card industry.
* Credit card fraud poses a significant threat to financial institutions and cardholders, resulting in substantial financial losses and compromised data security.
* To combat this problem, advanced technologies and machine learning algorithms are harnessed to detect and thwart fraudulent activities promptly.
* The primary objective of this project is to create a sophisticated system that can effectively detect and flag potentially fraudulent credit card transactions in real-time. This includes both online and offline transactions.

Content for phase2:

Consider exploring advanced techniques like anomaly detection algorithms ,for example, isolation forest, one class SVM, and ensemble methods for improving fraud detection accuracy.

Dataset:

The dataset contains transactions made by credit cards in September 2013 by European cardholders.

It contains only numerical input variables which are the result of a PCA transformation.

Dataset link: <https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud>

**Data collection and processing:**

**Data Collection:** The project will collect transaction data from various sources, including credit card companies, financial institutions, and e-commerce platforms. This data will serve as the foundation for building fraud detection models.

**Data Preprocessing:** Data preprocessing is essential for cleaning, transforming, and normalizing the data. This step ensures that the data is suitable for analysis and model training.

**Data visualization:**

Generate summary statistics like mean, median, standard deviation, etc., for numerical features.

Create histograms, box plots, or density plots to visualize the distribution of transaction amounts and other relevant features.

Plot the distribution of fraudulent vs. non-fraudulent transactions to understand class imbalance.

**Feature engineering:**

Feature engineering involves selecting relevant features or variables that contribute to fraud detection.

It may also involve creating new features that can enhance the accuracy of the models.

**Techniques used:**

* logistic regression
* random forest
* anomaly detection

**Model development and evaluation:**

* Machine learning models, including logistic regression, decision trees, random forests, and will be developed and trained using historical transaction data to predict fraudulent activities.
* The performance of the models will be regularly assessed using metrics such as precision, recall, and F1-score. Continuous model refinement and updates will be carried out to adapt to evolving fraud patterns.

**PROGRAM:**

**Model 1: logistic regression**

In [1]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

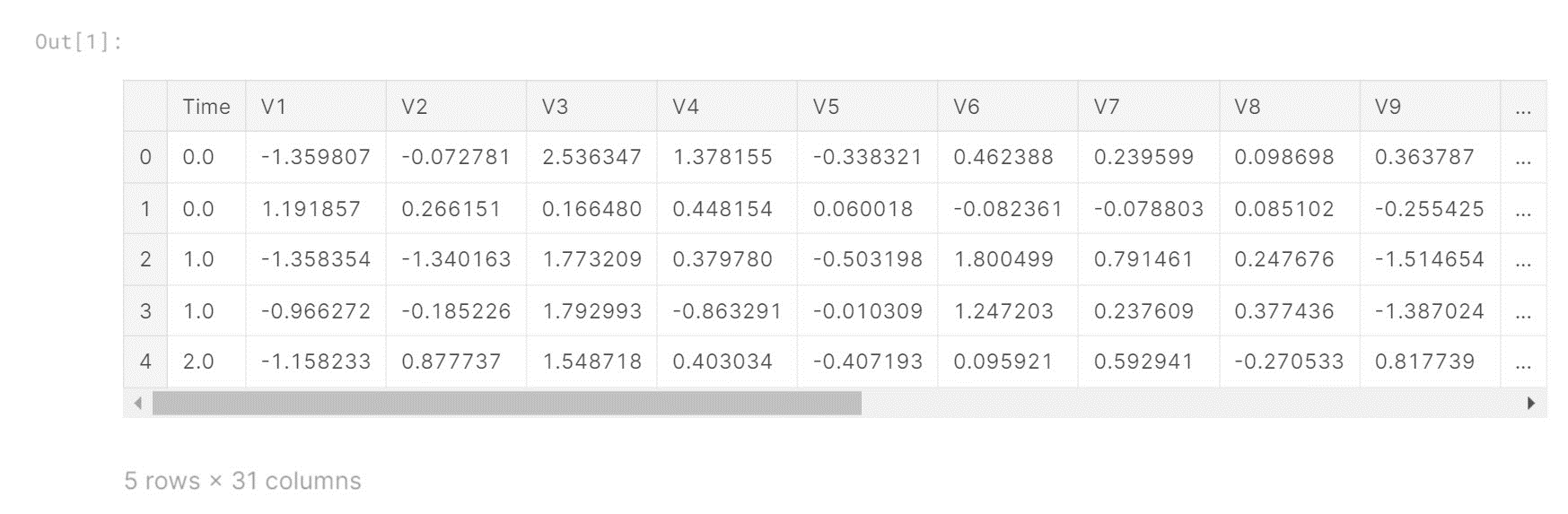
sns.set()

%matplotlib inline

df = pd.read\_csv('../input/creditcard.csv')

print(df.shape)

df.head()



In [2]:

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 284807 entries, 0 to 284806

Data columns (total 31 columns):

Time 284807 non-null float64

V1 284807 non-null float64

V2 284807 non-null float64

V3 284807 non-null float64

V4 284807 non-null float64

V5 284807 non-null float64

V6 284807 non-null float64

V7 284807 non-null float64

V8 284807 non-null float64

V9 284807 non-null float64

V10 284807 non-null float64

V11 284807 non-null float64

V12 284807 non-null float64

V13 284807 non-null float64

V14 284807 non-null float64

V15 284807 non-null float64

V16 284807 non-null float64

V17 284807 non-null float64

V18 284807 non-null float64

V19 284807 non-null float64

V20 284807 non-null float64

V21 284807 non-null float64

V22 284807 non-null float64

V23 284807 non-null float64

V24 284807 non-null float64

V25 284807 non-null float64

V26 284807 non-null float64

V27 284807 non-null float64

V28 284807 non-null float64

Amount 284807 non-null float64

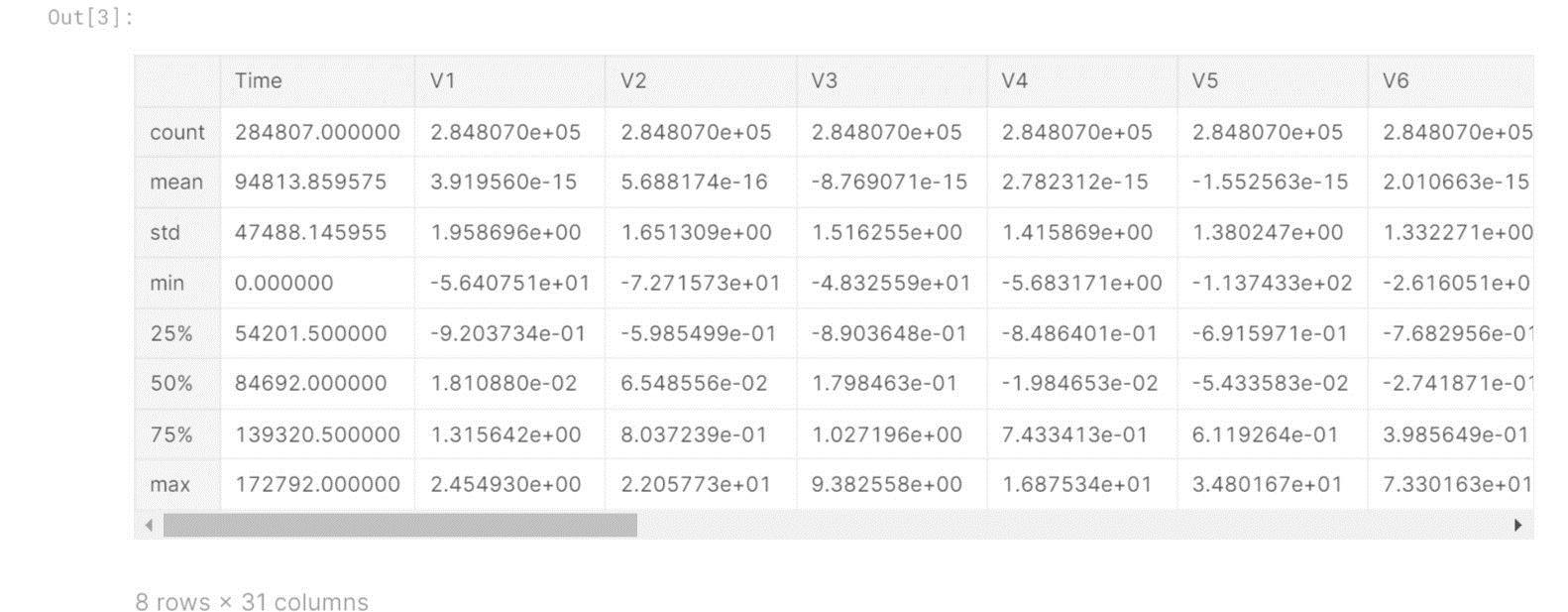
Class 284807 non-null int64

dtypes: float64(30), int64(1)

memory usage: 67.4 MB

In [3]:

df.describe()



In [4]:

class\_names = {0:'Not Fraud', 1:'Fraud'}

print(df.Class.value\_counts().rename(index = class\_names))

Not Fraud 284315

Fraud 492

Name: Class, dtype: int64

In [5]:

fig = plt.figure(figsize = (15, 12))

plt.subplot(5, 6, 1) ; plt.plot(df.V1) ; plt.subplot(5, 6, 15) ; plt.plot(df.V15)

plt.subplot(5, 6, 2) ; plt.plot(df.V2) ; plt.subplot(5, 6, 16) ; plt.plot(df.V16)

plt.subplot(5, 6, 3) ; plt.plot(df.V3) ; plt.subplot(5, 6, 17) ; plt.plot(df.V17)

plt.subplot(5, 6, 4) ; plt.plot(df.V4) ; plt.subplot(5, 6, 18) ; plt.plot(df.V18)

plt.subplot(5, 6, 5) ; plt.plot(df.V5) ; plt.subplot(5, 6, 19) ; plt.plot(df.V19)

plt.subplot(5, 6, 6) ; plt.plot(df.V6) ; plt.subplot(5, 6, 20) ; plt.plot(df.V20)

plt.subplot(5, 6, 7) ; plt.plot(df.V7) ; plt.subplot(5, 6, 21) ; plt.plot(df.V21)

plt.subplot(5, 6, 8) ; plt.plot(df.V8) ; plt.subplot(5, 6, 22) ; plt.plot(df.V22)

plt.subplot(5, 6, 9) ; plt.plot(df.V9) ; plt.subplot(5, 6, 23) ; plt.plot(df.V23)

plt.subplot(5, 6, 10) ; plt.plot(df.V10) ; plt.subplot(5, 6, 24) ; plt.plot(df.V24)

plt.subplot(5, 6, 11) ; plt.plot(df.V11) ; plt.subplot(5, 6, 25) ; plt.plot(df.V25)

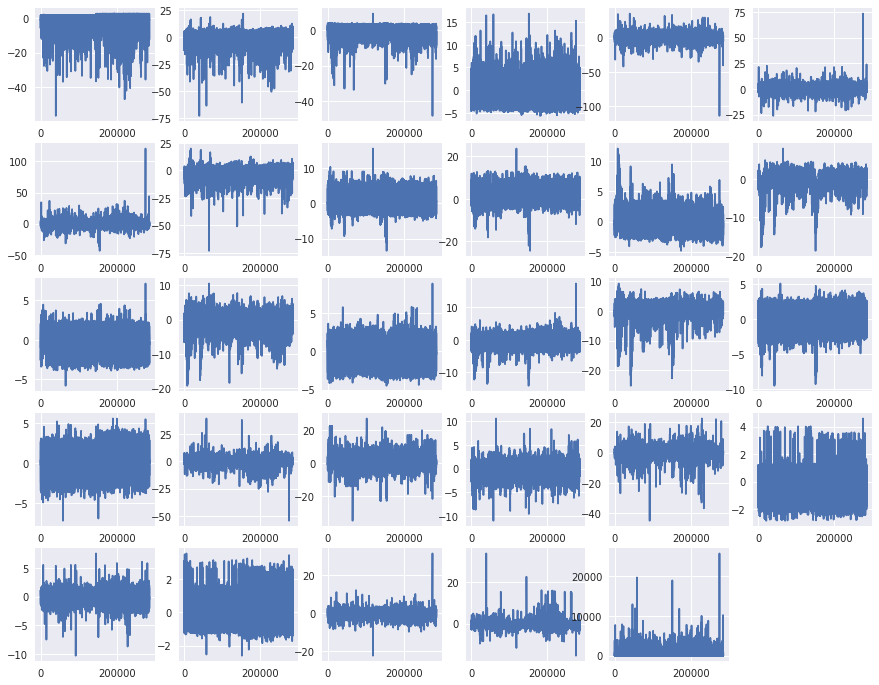
plt.subplot(5, 6, 12) ; plt.plot(df.V12) ; plt.subplot(5, 6, 26) ; plt.plot(df.V26)

plt.subplot(5, 6, 13) ; plt.plot(df.V13) ; plt.subplot(5, 6, 27) ; plt.plot(df.V27)

plt.subplot(5, 6, 14) ; plt.plot(df.V14) ; plt.subplot(5, 6, 28) ; plt.plot(df.V28)

plt.subplot(5, 6, 29) ; plt.plot(df.Amount)

plt.show()



from sklearn.cross\_validation import train\_test\_split

feature\_names = df.iloc[:, 1:30].columns

target = df.iloc[:1, 30: ].columns

print(feature\_names)

print(target)

Index(['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'],

dtype='object')

Index(['Class'], dtype='object')

In [8]:

data\_features = df[feature\_names]

data\_target = df[target]

In [9]:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data\_features, data\_target, train\_size=0.70, test\_size=0.30, random\_state=1)

print("Length of X\_train is: **{X\_train}**".format(X\_train = len(X\_train)))

print("Length of X\_test is: **{X\_test}**".format(X\_test = len(X\_test)))

print("Length of y\_train is: **{y\_train}**".format(y\_train = len(y\_train)))

print("Length of y\_test is: **{y\_test}**".format(y\_test = len(y\_test)))

Length of X\_train is: 199364

Length of X\_test is: 85443

Length of y\_train is: 199364

Length of y\_test is: 85443

In [10]:

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix

In [11]:

model = LogisticRegression()

model.fit(X\_train, y\_train.values.ravel())

Out[11]:

LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,

intercept\_scaling=1, max\_iter=100, multi\_class='ovr', n\_jobs=1,

penalty='l2', random\_state=None, solver='liblinear', tol=0.0001,

verbose=0, warm\_start=False)

In [12]:

pred = model.predict(X\_test)

In [13]:

class\_names = ['not\_fraud', 'fraud']

matrix = confusion\_matrix(y\_test, pred)

*# Create pandas dataframe*

dataframe = pd.DataFrame(matrix, index=class\_names, columns=class\_names)

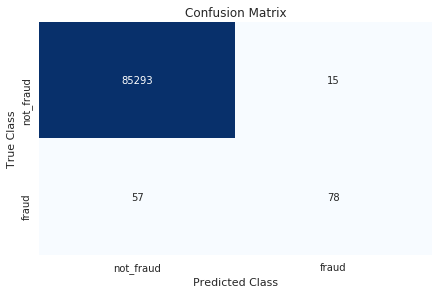
*# Create heatmap*

sns.heatmap(dataframe, annot=True, cbar=None, cmap="Blues", fmt = 'g')

plt.title("Confusion Matrix"), plt.tight\_layout()

plt.ylabel("True Class"), plt.xlabel("Predicted Class")

plt.show()



from sklearn.metrics import f1\_score, recall\_score

f1\_score = round(f1\_score(y\_test, pred), 2)

recall\_score = round(recall\_score(y\_test, pred), 2)

print("Sensitivity/Recall for Logistic Regression Model 1 : **{recall\_score}**".format(recall\_score = recall\_score))

print("F1 Score for Logistic Regression Model 1 : **{f1\_score}**".format(f1\_score = f1\_score))

Sensitivity/Recall for Logistic Regression Model 1 : 0.58

F1 Score for Logistic Regression Model 1 : 0.68

**Model 2: random forest**

import pandas as pd *#To hand with data*

import numpy as np *#To math*

import seaborn as sns *#to visualization*

import matplotlib.pyplot as plt *# to plot the graphs*

import matplotlib.gridspec as gridspec *# to do the grid of plots*

In [2]:

*#loading the data*

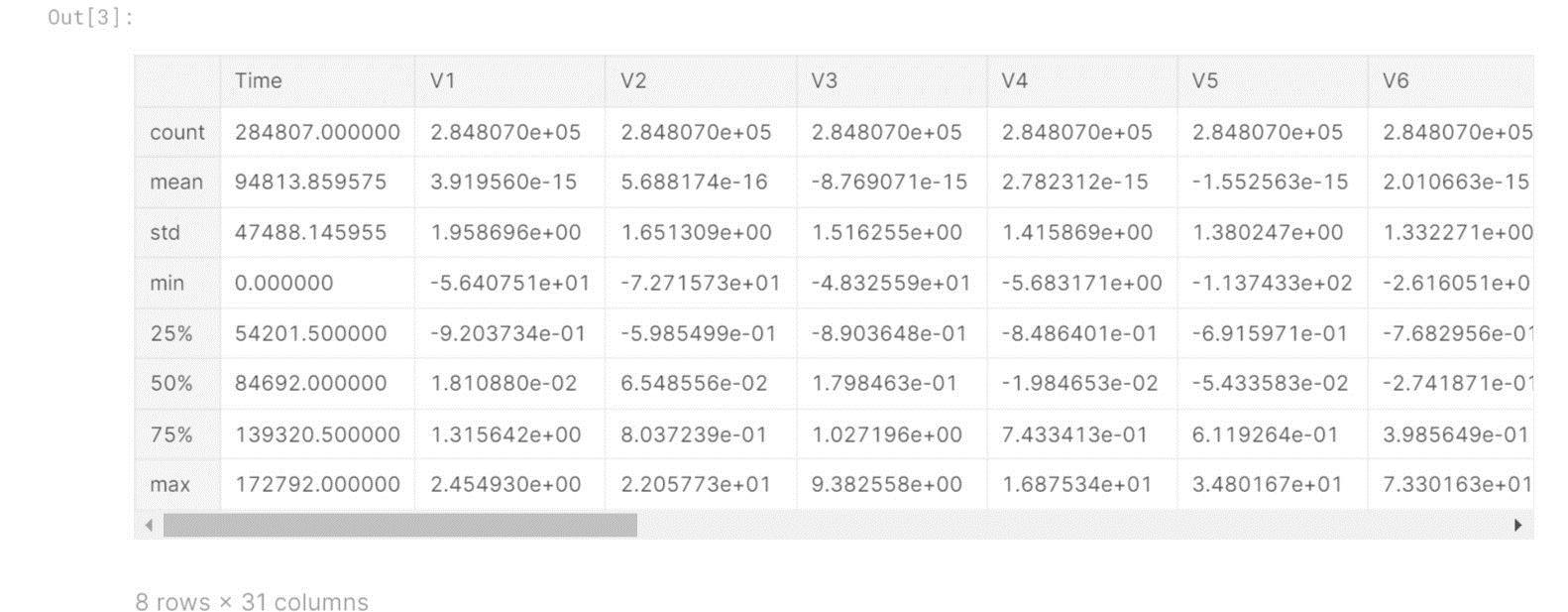
df\_credit = pd.read\_csv("../input/creditcard.csv")

In [3]:

linkcode

*#looking the how data looks*

df\_credit.head()



df\_credit.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 284807 entries, 0 to 284806

Data columns (total 31 columns):

Time 284807 non-null float64

V1 284807 non-null float64

V2 284807 non-null float64

V3 284807 non-null float64

V4 284807 non-null float64

V5 284807 non-null float64

V6 284807 non-null float64

V7 284807 non-null float64

V8 284807 non-null float64

V9 284807 non-null float64

V10 284807 non-null float64

V11 284807 non-null float64

V12 284807 non-null float64

V13 284807 non-null float64

V14 284807 non-null float64

V15 284807 non-null float64

V16 284807 non-null float64

V17 284807 non-null float64

V18 284807 non-null float64

V19 284807 non-null float64

V20 284807 non-null float64

V21 284807 non-null float64

V22 284807 non-null float64

V23 284807 non-null float64

V24 284807 non-null float64

V25 284807 non-null float64

V26 284807 non-null float64

V27 284807 non-null float64

V28 284807 non-null float64

Amount 284807 non-null float64

Class 284807 non-null int64

dtypes: float64(30), int64(1)

memory usage: 67.4 MB

In [5]:

*# The data is stardarized, I will explore them later*

*#For now I will look the "normal" columns*

df\_credit[["Time","Amount","Class"]].describe()

Out[5]:

|  | Time | Amount | Class |
| --- | --- | --- | --- |
| count | 284807.000000 | 284807.000000 | 284807.000000 |
| mean | 94813.859575 | 88.349619 | 0.001727 |
| std | 47488.145955 | 250.120109 | 0.041527 |
| min | 0.000000 | 0.000000 | 0.000000 |
| 25% | 54201.500000 | 5.600000 | 0.000000 |
| 50% | 84692.000000 | 22.000000 | 0.000000 |
| 75% | 139320.500000 | 77.165000 | 0.000000 |
| max | 172792.000000 | 25691.160000 | 1.000000 |

from imblearn.pipeline import make\_pipeline as make\_pipeline\_imb *# To do our transformation in a unique time*

from imblearn.over\_sampling import SMOTE

from sklearn.pipeline import make\_pipeline

from imblearn.metrics import classification\_report\_imbalanced

from sklearn.model\_selection import train\_test\_split

from collections import Counter

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import precision\_score, recall\_score, fbeta\_score, confusion\_matrix, precision\_recall\_curve, accuracy\_score

/opt/conda/lib/python3.6/site-packages/sklearn/ensemble/weight\_boosting.py:29: DeprecationWarning: numpy.core.umath\_tests is an internal NumPy module and should not be imported. It will be removed in a future NumPy release.

from numpy.core.umath\_tests import inner1d

In [21]:

X = df\_credit.drop(["Class"], axis=1).values *#Setting the X to do the split*

y = df\_credit["Class"].values *# transforming the values in array*

In [22]:

*# the function that we will use to better evaluate the model*

def print\_results(headline, true\_value, pred):

print(headline)

print("accuracy: **{}**".format(accuracy\_score(true\_value, pred)))

print("precision: **{}**".format(precision\_score(true\_value, pred)))

print("recall: **{}**".format(recall\_score(true\_value, pred)))

print("f2: **{}**".format(fbeta\_score(true\_value, pred, beta=2)))

*# splitting data into training and test set*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=2, test\_size=0.20)

classifier = RandomForestClassifier

*# build model with SMOTE imblearn*

smote\_pipeline = make\_pipeline\_imb(SMOTE(random\_state=4), \

classifier(random\_state=42))

smote\_model = smote\_pipeline.fit(X\_train, y\_train)

smote\_prediction = smote\_model.predict(X\_test)

*#Showing the diference before and after the transformation used*

print("normal data distribution: **{}**".format(Counter(y)))

X\_smote, y\_smote = SMOTE().fit\_sample(X, y)

print("SMOTE data distribution: **{}**".format(Counter(y\_smote)))

normal data distribution: Counter({0: 284315, 1: 492})

SMOTE data distribution: Counter({0: 284315, 1: 284315})

**Model 3: anomaly detection**

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"]

import plotly.plotly as py

import plotly.graph\_objs as go

import plotly

import plotly.figure\_factory as ff

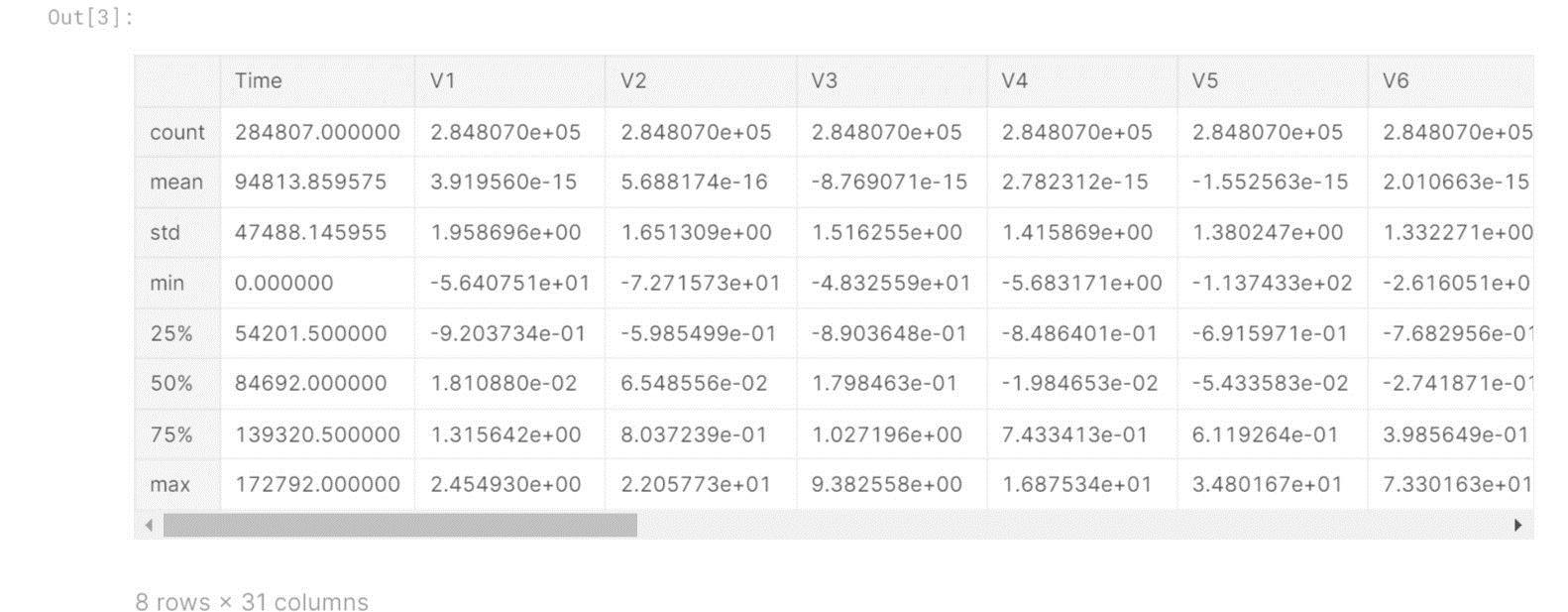
from plotly.offline import init\_notebook\_mode, iplot

In [2]:

linkcode

data = pd.read\_csv('../input/creditcard\_data.csv')

data.head()



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

data1.shape

Out[3]:

(28481, 31)

In [4]:

*# Checking the missing values*

data.isnull().sum()

Out[4]:

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

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");



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

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

In [8]:

Normal.shape

Out[8]:

(284314, 31)

In [9]:

Fraud.shape

Out[9]:

(492, 31)

In [10]:

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

Normal.Amount.describe()

Out[10]:

count 284314.000000

mean 88.290570

std 250.105416

min 0.000000

25% 5.650000

50% 22.000000

75% 77.050000

max 25691.160000

Name: Amount, dtype: float64

In [11]:

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

Fraud.Amount.describe()

Out[11]:

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

In [12]:

*#Let's have a more graphical representation of the data*

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')

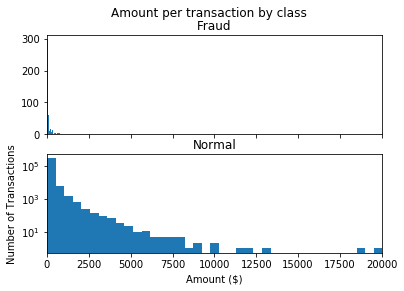
plt.xlabel('Amount ($)')

plt.ylabel('Number of Transactions')

plt.xlim((0, 20000))

plt.yscale('log')

plt.show();



*#Graphical representation of the data*

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')

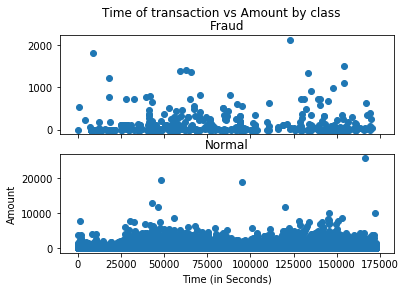
ax2.scatter(Normal.Time, Normal.Amount)

ax2.set\_title('Normal')

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

plt.ylabel('Amount')

plt.show();



In [14]:

init\_notebook\_mode(connected=True)

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

In [15]:

*# Create a trace*

trace = go.Scatter(

x = Fraud.Time,

y = Fraud.Amount,

mode = 'markers'

)

data = [trace]

plotly.offline.iplot({

"data": data

})

*#Define the outlier detection methods*

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, random\_state=state)

}

Fit the model

In [22]:

*#Fit the model*

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))

/opt/conda/lib/python3.6/site-packages/sklearn/ensemble/iforest.py:223: FutureWarning:

behaviour="old" is deprecated and will be removed in version 0.22. Please use behaviour="new", which makes the decision\_function change to match other anomaly detection algorithm API.

/opt/conda/lib/python3.6/site-packages/sklearn/ensemble/iforest.py:417: DeprecationWarning:

threshold\_ attribute is deprecated in 0.20 and will be removed in 0.22.

Isolation Forest: 69

Accuracy Score :

0.9975773322565921

Classification Report :

precision recall f1-score support

0 1.00 1.00 1.00 28434

1 0.27 0.28 0.27 47

micro avg 1.00 1.00 1.00 28481

macro avg 0.63 0.64 0.64 28481

weighted avg 1.00 1.00 1.00 28481

Local Outlier Factor: 93

Accuracy Score :

0.9967346652154068

Classification Report :

precision recall f1-score support

0 1.00 1.00 1.00 28434

1 0.02 0.02 0.02 47

micro avg 1.00 1.00 1.00 28481

macro avg 0.51 0.51 0.51 28481

weighted avg 1.00 1.00 1.00 28481

/opt/conda/lib/python3.6/site-packages/sklearn/svm/classes.py:1177: DeprecationWarning:

The random\_state parameter is deprecated and will be removed in version 0.22.

Support Vector Machine: 8411

Accuracy Score :

0.7046803131912504

Classification Report :

precision recall f1-score support

0 1.00 0.71 0.83 28434

1 0.00 0.34 0.00 47

micro avg 0.70 0.70 0.70 28481

macro avg 0.50 0.52 0.42 28481

weighted avg 1.00 0.70 0.83 2848

**Conclusion:**

In conclusion, the Credit Card Fraud Detection Project plays a pivotal role in safeguarding the financial interests of credit card users and institutions by harnessing the power of advanced technology and machine learning.

Its ultimate goal is to minimize financial losses due to fraudulent activities while maintaining the integrity and security of sensitive cardholder data.