**capstone\_credit\_card\_finddefault**

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

**Problem Statement:**

The goal is to build a classification model to detect fraud in a highly imbalanced dataset of credit card transactions. We will preprocess the data, handle class imbalance with techniques like SMOTE, and evaluate models using metrics like Precision, Recall, and ROC-AUC. After training and fine-tuning, the best model will be deployed for real-time fraud detection.

***PROJECT STEPS***

1.# importing library & read data

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

import scipy.stats as stats

%matplotlib inline

credit\_card = pd.read\_csv('creditcard.csv')

2. DATA Anaysis

In Data analysis , we will Analyze to find out the below points:

1. Categorical Variables

2. Cardinality of Categorical Variables

3. Missing Values

4. Univariante analysis of all the numerical Variables

5. Distribution of the numerical Variables

6. Features Corelation

7. Outliers

8. Relationship between independent and dependent features

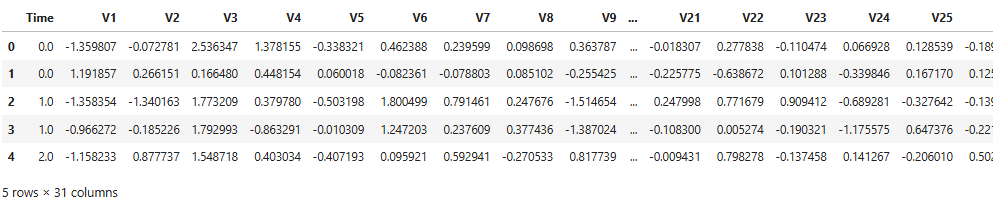
* # print shape of dataset with rowa & columns

print(credit\_card.shape)

**(284807, 31)**

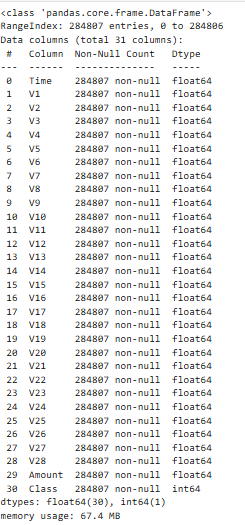
* # print top 5 records

credit\_card.head()



* # print full summary

credit\_card.info()



* Features Type

categorical\_features=[features for features in credit\_card.columns if credit\_card[features].dtypes=="0" or credit\_card[features].dtypes=="int64"]

print(categorical\_features)

**['Class']**

The dataset has a single categorical feature, 'Class', with a relatively low number of unique values, so it doesn't exhibit high cardinality. High dimensionality refers to datasets with many features, but that's not an issue here.

* Discrete Features

discrete\_feature=[feature for feature in credit\_card.columns if len(credit\_card[feature].unique())<25 and feature not in ['class']]

print("Discrete Variables Count:{}".format(len(discrete\_feature)))

**Discrete Variables Count:1**

Key point

The dataset contains one discrete variable.

* Missing Value

# step make the list of features which has missing values

features\_with\_na=[features for features in credit\_card.columns if credit\_card[features].isnull().sum()>1]

# print the missing features list

print(len(features\_with\_na))

**0**

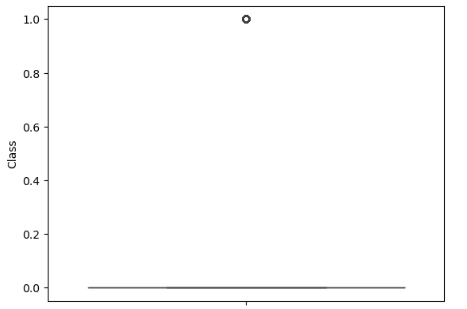
* # DATA UNBALANCE

sns.boxplot(credit\_card['Class'])

plt.show()

print('percent of fraud transaction:',len(credit\_card[credit\_card['Class']==1])/len(credit\_card['Class'])\*100,"%")

print('percent of normal transaction:',len(credit\_card[credit\_card['Class']==0])/len(credit\_card['Class'])\*100,"%")



percent of fraud transaction: 0.1727485630620034 %

percent of normal transaction: 99.82725143693798 %

The dataset is highly unbalanced, with 0.173% fraudulent transactions

* Histogram analysis

# lets analyse the continuous values by creating histograms to understand the distribution

data=credit\_card.copy()

data.drop(columns="Class",inplace = True)

for feature in data.columns:

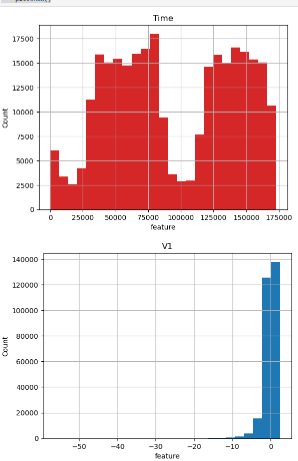
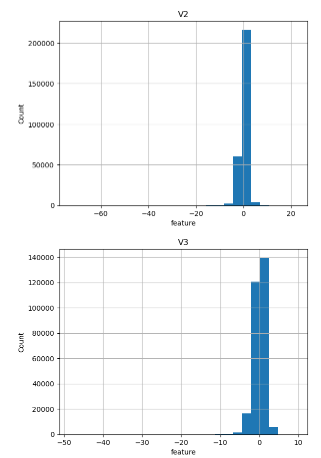
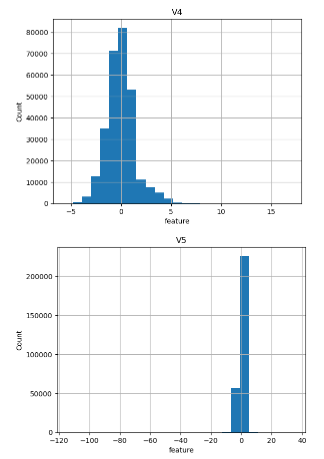
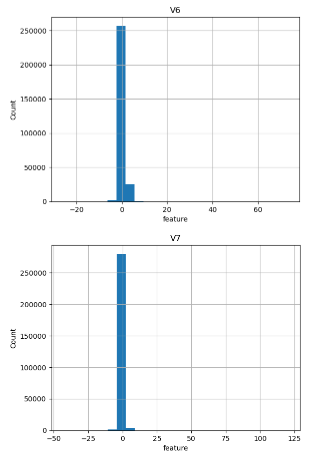
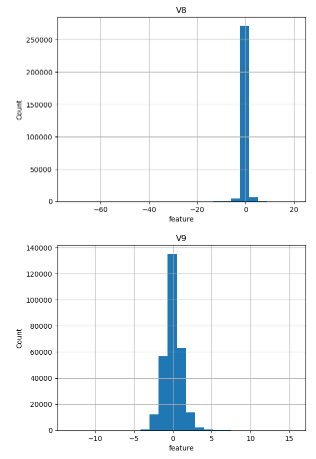
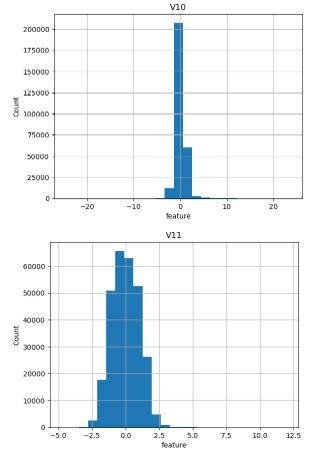
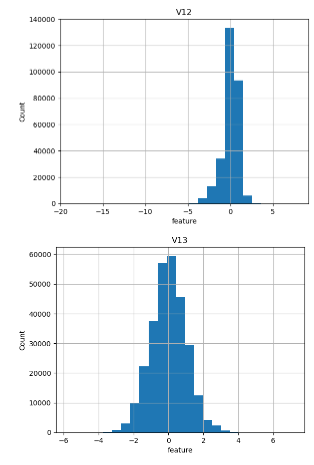
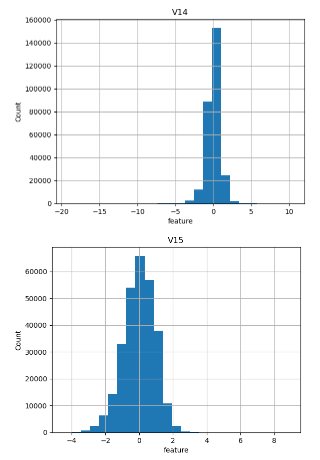
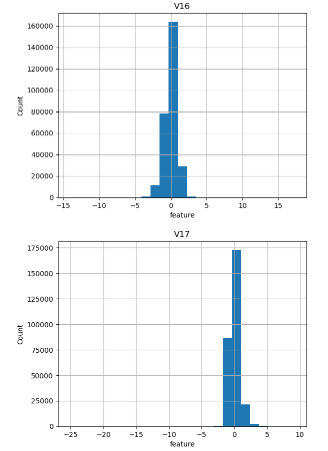
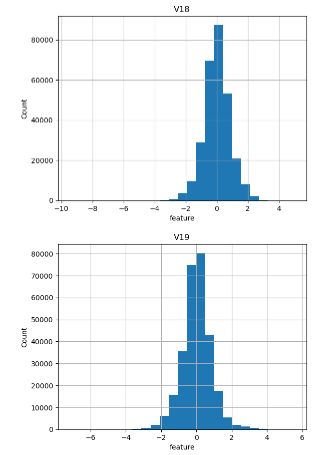
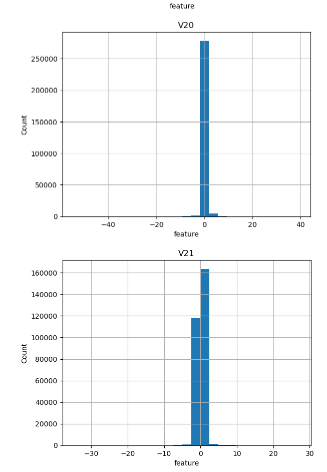
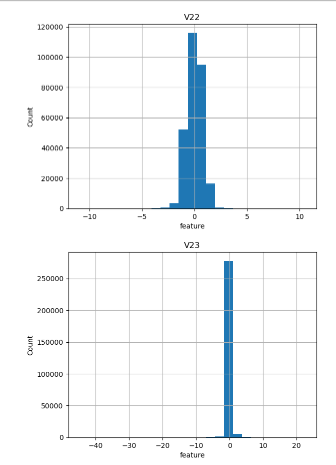
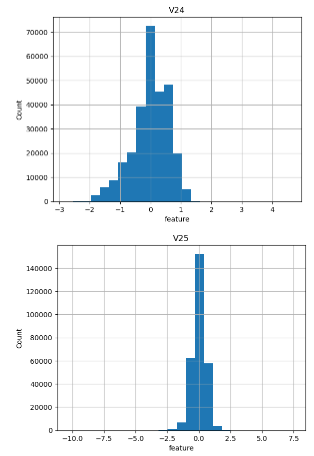
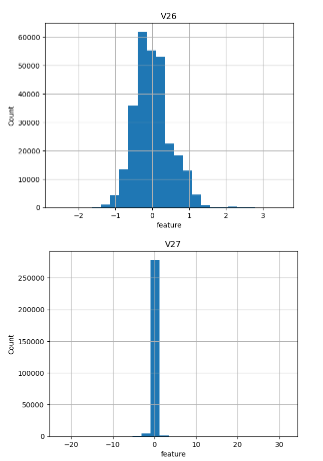
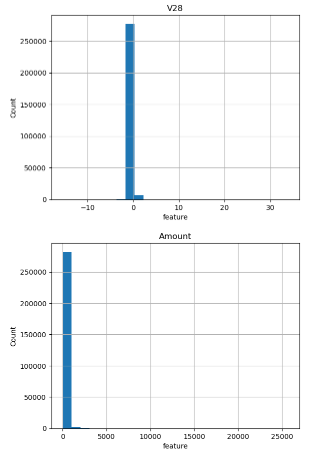
credit\_card[feature].hist(bins=25)

plt.xlabel('feature')

plt.ylabel('Count')

plt.title(feature)

plt.show()

Normalization is crucial in PCA because it is a variance-maximizing technique that projects the original data onto directions that maximize variance. Since the principal components (V1 to V28) obtained from PCA represent the directions of maximum variance, we can observe from the plot that the V1 to V28 variables are normalized. However, the "Time" and "Amount" variables were not normalized.

* #figure factory module contains dedicated functions for creating very specific types of plots

import plotly.figure\_factory as ff

from plotly import tools

from plotly.offline import download\_plotlyjs,init\_notebook\_mode,plot,iplot

class\_0 = credit\_card.loc[credit\_card['Class']==0]["Time"]

class\_1 = credit\_card.loc[credit\_card['Class']==1]["Time"]

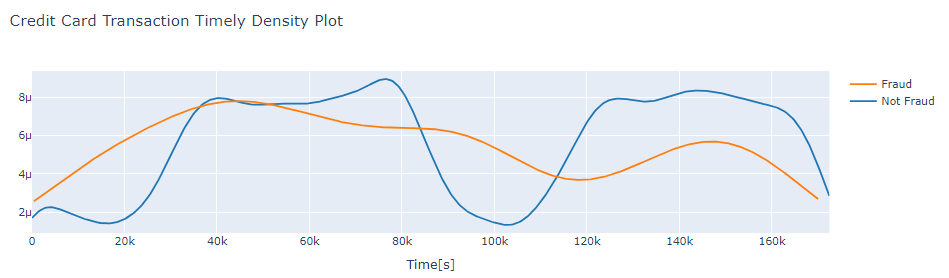
hist\_data = [class\_0,class\_1]

group\_labels = ["Not Fraud","Fraud"]

fig = ff.create\_distplot(hist\_data, group\_labels, show\_hist=False, show\_rug=False)

fig['layout'].update(title='Credit Card Transaction Timely Density Plot', xaxis=dict(title='Time[s]'))

iplot(fig,filename='dist\_only')



The credit card transaction timely density plot visualizes the distribution of "Non-Fraud" and "Fraud" transactions over a continuous time period. Fraudulent transactions are more evenly distributed across time, while valid transactions show more time-based concentration. Therefore, the "Time" feature alone cannot effectively differentiate between fraudulent and non-fraudulent transactions.

* Feature Correlation

import matplotlib.pyplot as plt

import seaborn as sns

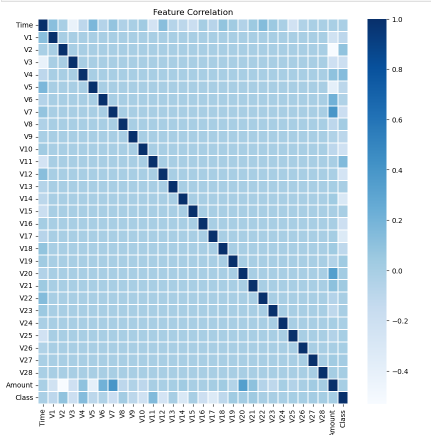
plt.figure(figsize = (10,10))

plt.title('Feature Correlation')

corr = credit\_card.corr()

sns.heatmap(corr,xticklabels=corr.columns, yticklabels=corr.columns, linewidths=.1, cmap="Blues")

plt.show()



Correlation is positive when both values increase together, and negative when one value decreases as the other increases.

The correlation values can range as follows:

* **1** indicates a perfect positive correlation (e.g., between 'Amount').
* **0** indicates no correlation (e.g., between V1-V28).
* **-1** indicates a perfect negative correlation (e.g., between 'Time' and V3, or 'Amount' and V2 & V5).
* **OUTLIER**

#transaction amount

data=credit\_card.copy()

data.drop(columns=['Class'],inplace=True)

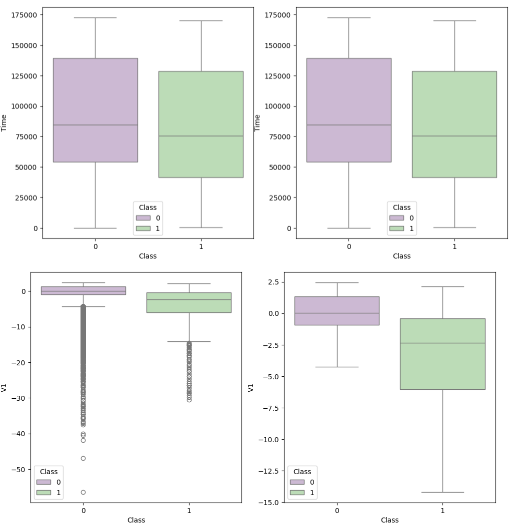
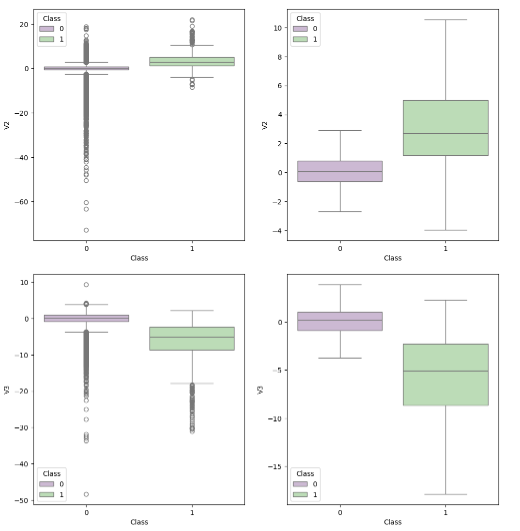
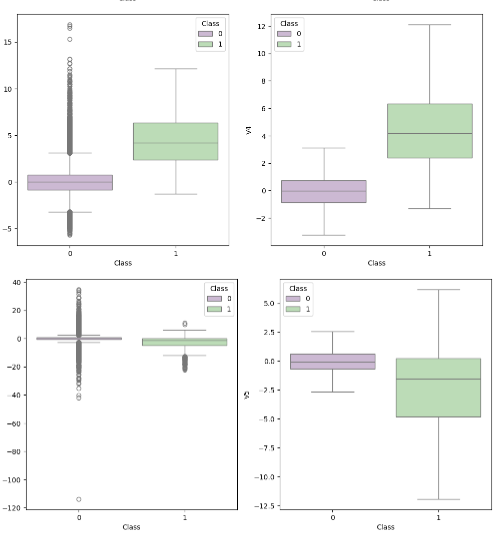
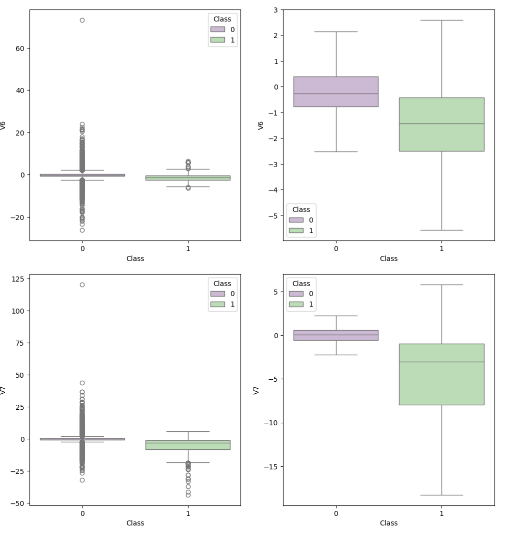
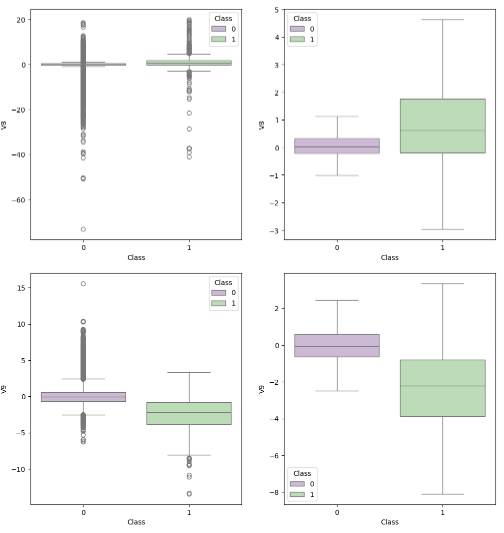
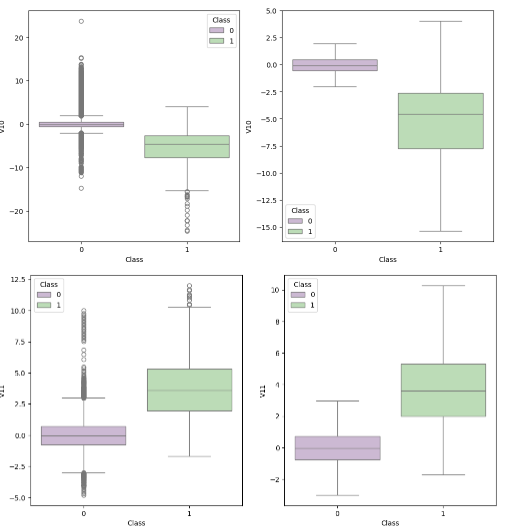
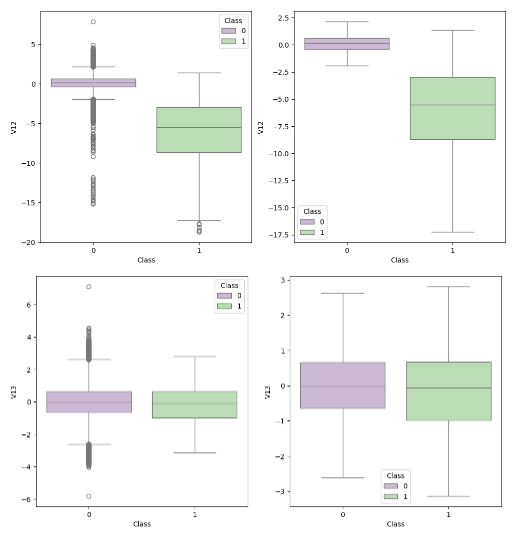
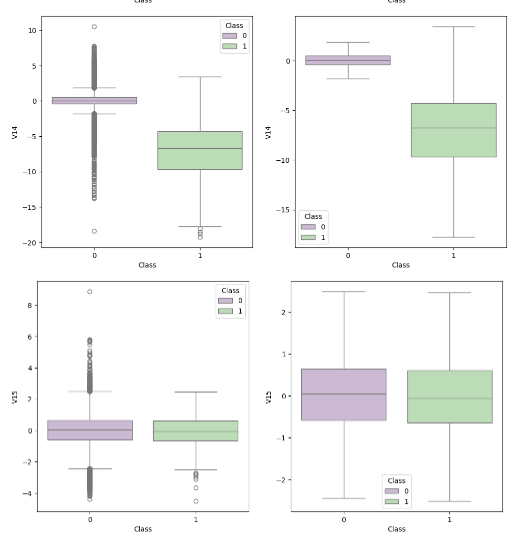
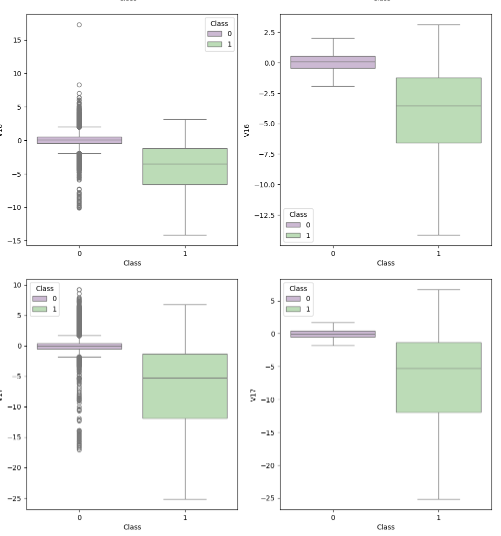
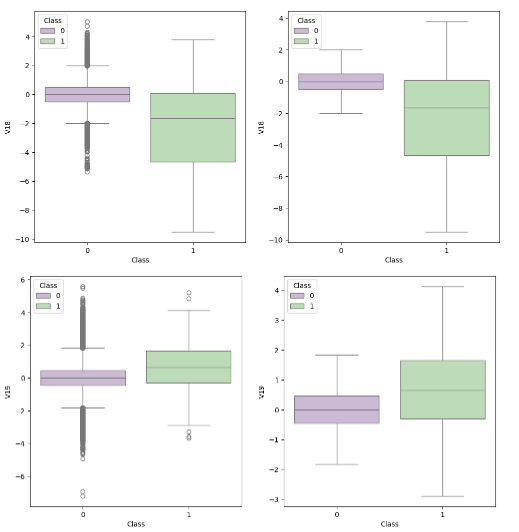
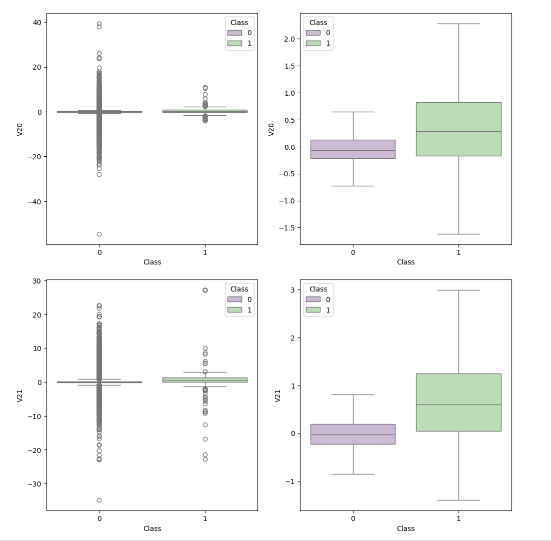
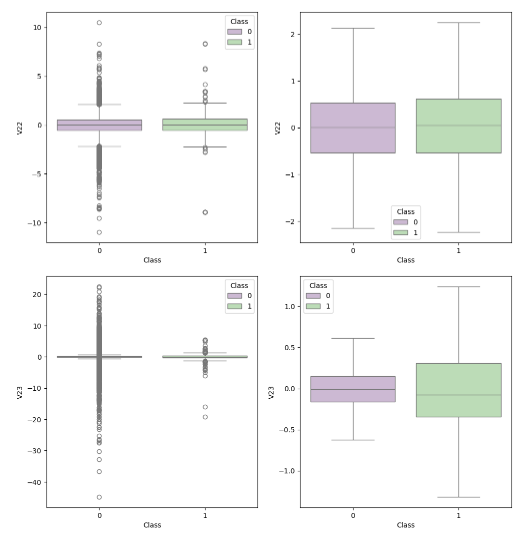
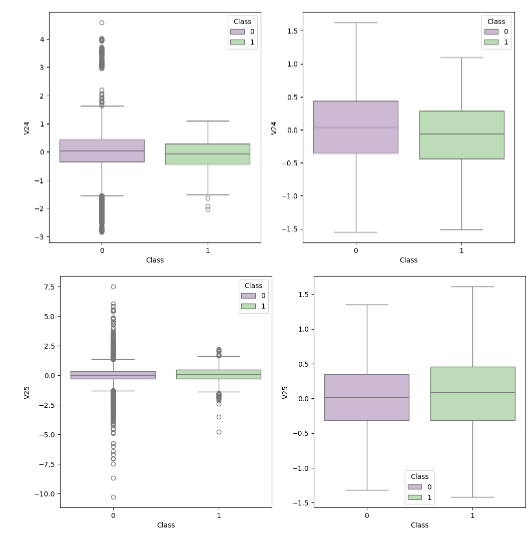
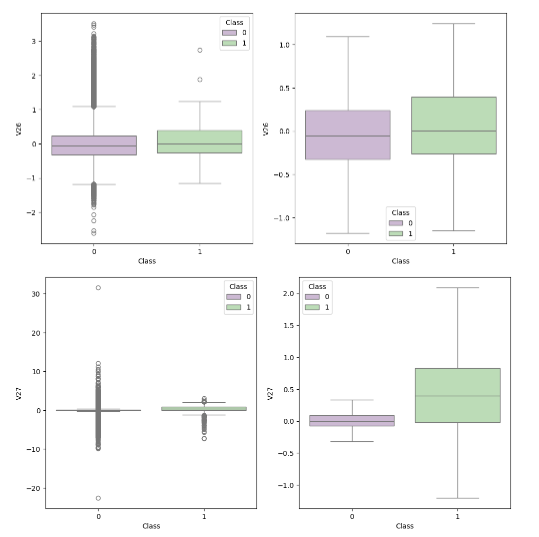
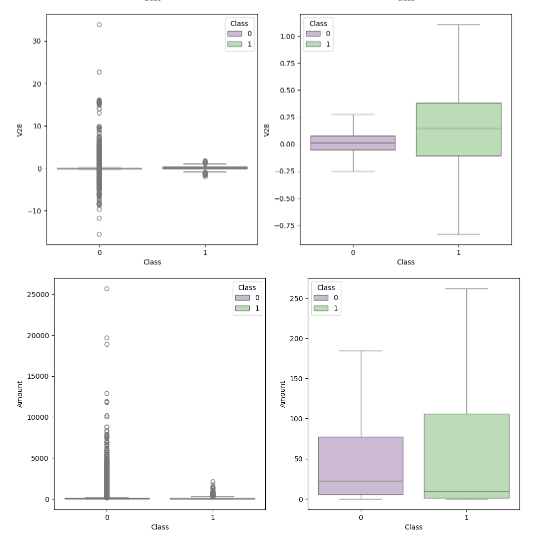
for i in data.columns:

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12,6))

s = sns.boxplot(ax = ax1, x="Class", y=i, hue="Class", data=credit\_card, palette="PRGn",showfliers=True)

s = sns.boxplot(ax = ax2, x="Class", y=i, hue="Class", data=credit\_card, palette="PRGn",showfliers=False)

plt.show();

The box plot shows that fraudulent transactions have more outliers compared to non-fraudulent transactions. Since the data is highly imbalanced, with fraudulent transactions being fewer, transforming or removing outliers could result in the loss of valuable information. Therefore, we choose to retain the outliers in the dataset as they may provide important insights for detecting fraud.

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

# Load your credit card data

credit\_card = pd.read\_csv("creditcard.csv")

# features density plot

col = ['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']

i = 0

t0 = credit\_card.loc[credit\_card["Class"]==0]

t1 = credit\_card.loc[credit\_card["Class"]==1]

sns.set\_style('whitegrid')

plt.figure()

fig,ax = plt.subplots(8,4,figsize=(16,30))

for feature in col:

i += 1

plt.subplot(8,4,i)

sns.kdeplot(t0[feature], bw\_method=0.5,label="Class = 0", color='b')

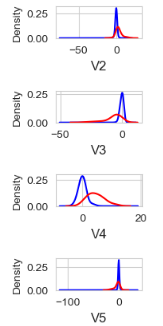
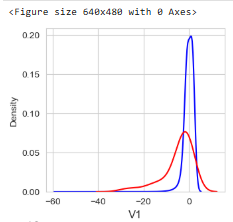
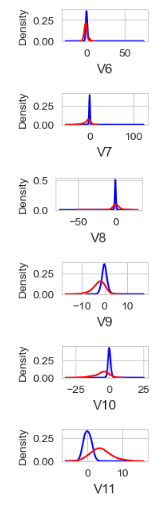
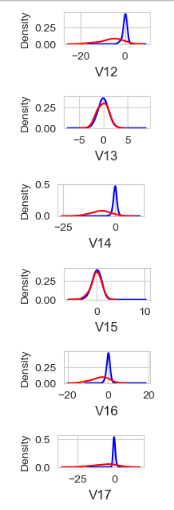
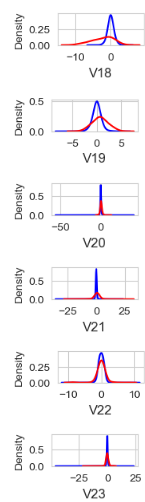
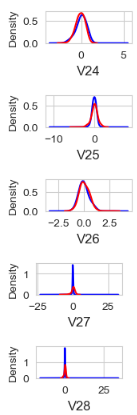
sns.kdeplot(t1[feature], bw\_method=0.5,label="Class = 1", color='r')

plt.xlabel(feature, fontsize=12)

locs, labels = plt.xticks()

plt.tick\_params(axis='both', which='major', labelsize=10)

plt.show()

Key points:

Separated distribution for Class 0 & 1 = V3, V4, V10, V11, V17-V19  
Partially Separated distribution for Class 0 & 1 =V1, V2, V7, V12, V14, V16, V18

Virtually Similar distribution for Class 0 & 1 =V13, V15, V20, V22-V28

Fairly Similar distribution for Class 0 & 1 =V5, V6, V8, V21

In general, with a few exceptions (such as "Time" and "Amount"), the features for legitimate transactions (Class = 0) are centered around 0, often with a long tail at one of the extremes. In contrast, fraudulent transactions (Class = 1) exhibit a skewed (asymmetric) distribution.

* **Skewness**

import matplotlib.pyplot as plt

import seaborn as sns

pca\_vars = ['V%i'% k for k in range(1,19)]

plt.figure(figsize=(12,4),dpi=80)

sns.barplot(x=pca\_vars, y=t0[pca\_vars].skew(),color='darkgreen')

plt.xlabel('Column')

plt.ylabel('Skewness')

plt.title('V1-V28 Skewness for class 0')

**Text(0.5, 1.0, 'V1-V28 Skewness for class 0')**

* 1. Class0

import matplotlib.pyplot as plt

import seaborn as sns

pca\_vars = ['V%i'% k for k in range(1,19)]

plt.figure(figsize=(12,4),dpi=80)

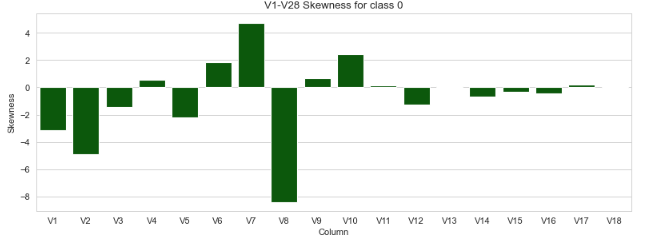
sns.barplot(x=pca\_vars, y=t0[pca\_vars].skew(),color='darkgreen')

plt.xlabel('Column')

plt.ylabel('Skewness')

plt.title('V1-V28 Skewness for class 0')

plt.show()



* 1. Class 1

import matplotlib.pyplot as plt

import seaborn as sns

plt.figure(figsize=(12,4),dpi=80)

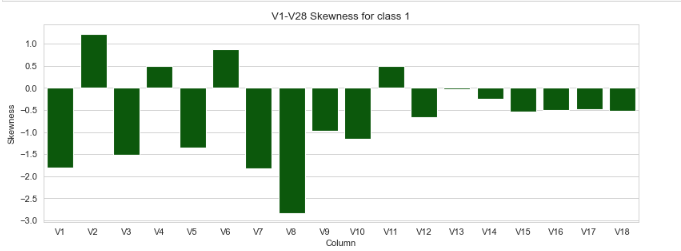
sns.barplot(x=pca\_vars, y=t1[pca\_vars].skew(),color='darkgreen')

plt.xlabel('Column')

plt.ylabel('Skewness')

plt.title('V1-V28 Skewness for class 1')

plt.show()



* **FacetGrid**

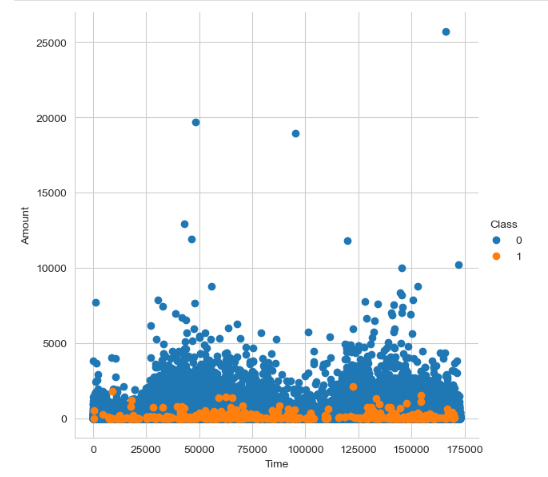
import matplotlib.pyplot as plt

import seaborn as sns

sns.set\_style("whitegrid")

sns.FacetGrid(credit\_card, hue="Class", height = 6).map(plt.scatter,"Time","Amount").add\_legend()

plt.show()



Key Points

* The plot clearly shows that fraudulent transactions occur primarily on amounts less than Rs. 2000, with very few frauds occurring on transactions above Rs. 2000.
* However, fraudulent transactions are evenly distributed over time.
* **Filter Data**

FilteredData = credit\_card[['Amount','Class']]

countLess = FilteredData[FilteredData['Amount']<2000]

CountMore = credit\_card.shape[0] - len(countLess)

percentage = round((len(countLess)/credit\_card.shape[0])\*100,2)

Class\_1 = countLess[countLess['Class']==1]

print('Total number of transaction less than 2000 is{}'.format(len(countLess)))

print('Total number of transaction more than 2000 is{}'.format(CountMore))

print('{}%of transaction having less than amount 2000'.format(percentage))

print('{}%of fraud transaction in data where less than amount 2000'.format(len('Class\_1')))

print('{} fraud transaction in data where less than amount 2000'.format(len('Class\_1')))

**Total number of transaction less than 2000 is284116**

**Total number of transaction more than 2000 is691**

**99.76%of transaction having less than amount 2000**

**7%of fraud transaction in data where less than amount 2000**

**7 fraud transaction in data where less than amount 2000**

* **BoxPlot**

import pandas as pd

import seaborn as sns

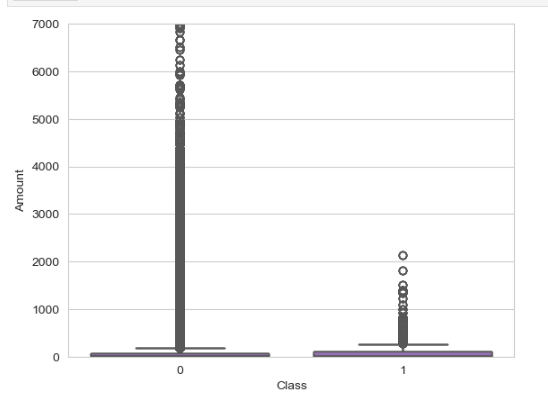
import matplotlib.pyplot as plt

credit\_card = pd.read\_csv('creditcard.csv')

sns.boxplot(x = "Class", y = "Amount", data = credit\_card)

plt.ylim(0,7000)

plt.show()



Key points

There are over 200,000 transactions with an amount less than Rs. 2000, accounting for 99.76% of the total transactions.

Among these, 7% are fraudulent, while the remaining transactions are legitimate.

* **DATA Transformation**

import pandas as pd

from sklearn.preprocessing import StandardScaler, RobustScaler

credit\_card = pd.read\_csv('creditcard.csv')

data1 = credit\_card.copy()

std\_scaler = StandardScaler()

rb\_scaler = RobustScaler()

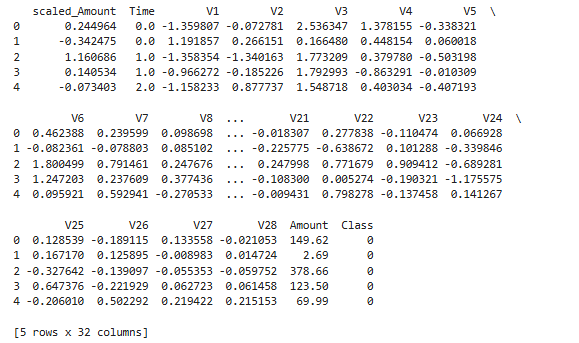
credit\_card['scaled\_Amount']= std\_scaler.fit\_transform(credit\_card['Amount'].values.reshape(-1,1))

scaled\_Amount = credit\_card['scaled\_Amount']

credit\_card.drop(['scaled\_Amount'],axis=1,inplace=True)

credit\_card.insert(0,'scaled\_Amount',scaled\_Amount)

print(credit\_card.head())



1. #Classifier Libraries

from sklearn.linear\_model import LogisticRegression , SGDClassifier

from sklearn.svm import SVC

# from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

import collections

from sklearn import metrics

from sklearn.metrics import make\_scorer, precision\_score, recall\_score, confusion\_matrix, classification\_report, matthews\_corrcoef, cohen\_kappa\_score

from sklearn.feature\_selection import SelectKBest

from sklearn.feature\_selection import f\_classif

from xgboost import XGBClassifier

from sklearn.ensemble import BaggingClassifier, GradientBoostingClassifier

Classifier={

"LogisticRegression": LogisticRegression(),

"Support Vector Classifier": SVC(),

"DecisionTreeClassifier": DecisionTreeClassifier(),

"RandomForestClassifier": RandomForestClassifier(),

"BaggingClassifier": BaggingClassifier(n\_estimators=10, random\_state=0),

"SGDClassifier": SGDClassifier(),

"GradientBoostingClassifier": GradientBoostingClassifier(),

"xgb": XGBClassifier()

}

3.

def plot(df):

fraud = df[df['Class']==1]

normal = df[df['Class']==0]

fraud.drop(['Class'],axis=1,inplace=True)

normal.drop(['Class'],axis=1,inplace=True)

fraud = fraud .set\_index('classifier')

normal = normal .set\_index('classifier')

plt.figure(figsize = (7,2))

sns.heatmap(fraud.iloc[:,:],annot=True, cmap=sns.light\_palette((210,90,60),input="husl"),linewidth=2)

plt.title('Class1')

plt.show()

plt.figure(figsize = (7,2))

sns.heatmap(normal.iloc[:,:],annot=True, cmap=sns.light\_palette((210,90,60),input="husl"),linewidth=2)

plt.title('Class0')

plt.show()

* **ROC curve**

def training(models, x, y, x\_t, y\_t):

conf = []

comp = []

rdict = {}

for key, model in models.items():

model = model.fit(x, y)

y\_pred = model.predict(x\_t)

rdict[key]= y\_pred

tn, fp, fn, tp = confusion\_matrix(y\_t, y\_pred).ravel()

precision, recall, fscore, support = metrics.precision\_recall\_fscore\_support(y\_t, y\_pred)

r1 = {'Classifier':key, 'TN':tn, 'TP':tp, 'FN':fn, 'FP':fp}

conf.append(r1)

MCC = matthews\_corrcoef(y\_t, y\_pred)

AUROC = roc\_auc\_score(y\_t, y\_pred)

Cohen\_kappa = cohen\_kappa\_score(y\_t, y\_pred)

accuracy = metrics.accuracy\_score(y\_t, y\_pred)

r2 ={'classifier':key,'matthews\_corrcoef':MCC,'Cohen\_kappa':Cohen\_kappa,'accuracy':accuracy,'Auroc':AUROC,'precision':precision[0],'recall':recall}

r3 ={'classifier':key,'matthews\_corrcoef':MCC,'Cohen\_kappa':Cohen\_kappa,'accuracy':accuracy,'Auroc':AUROC,'precision':precision[1],'recall':recall}

comp.append(r2)

comp.append(r3)

r11 = (pd.DataFrame(conf).to\_markdown())

r12 = pd.DataFrame(comp)

print(f'\n\nRoc curve \n\n')

roc\_curve(y\_t, rdict)

print(f'\n\n confusion matrics comparison \n\n')

print(r11)

print(f'\n\n Performance comparison \n\n')

plot(r12)

plt.show()

* **Classifier Prediction**

import pandas as pd

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix

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

credit\_card = pd.read\_csv('creditcard.csv')

y = credit\_card['Class']

x = credit\_card.drop('Class',axis=1)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=42)

classifier = LogisticRegression()

def training(classifiers, x\_train, y\_train, x\_test,y\_test):

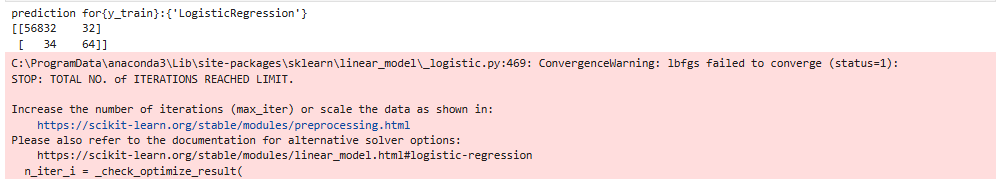
classifier.fit(x\_train,y\_train)

predictions = classifier.predict(x\_test)

print ("prediction for{y\_train}:{'LogisticRegression'}")

print(confusion\_matrix(y\_test,predictions))

training(classifier, x\_train, y\_train, x\_test,y\_test)



* **Model Training with selected features**

import os

import pandas as pd

from sklearn.feature\_selection import SelectKBest, f\_classif

credit\_card = pd.read\_csv('creditcard.csv')

y = credit\_card['Class']

x = credit\_card.drop('Class', axis=1)

bestfeatures= SelectKBest(score\_func=f\_classif, k=10)

fit = bestfeatures.fit(x,y)

dfscores = pd.DataFrame(fit.scores\_)

dfcolumns= pd.DataFrame(x.columns)

# Concat 2 different dataframe to better visualization

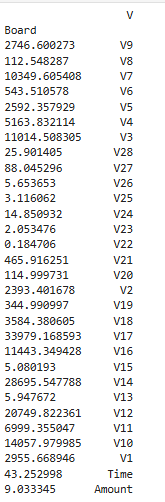
featureScores = pd.concat([dfcolumns,dfscores],axis=1)

featureScores.columns = ['V','Board']

featureScores\_df = featureScores.sort\_values(['V','Board'],ascending=[False, True])

featureScores\_df = featureScores\_df.set\_index('Board')

print(featureScores\_df)



* Index 10

import os

import pandas as pd

from sklearn.feature\_selection import SelectKBest, f\_classif

credit\_card = pd.read\_csv('creditcard.csv')

y = credit\_card['Class']

x = credit\_card.drop('Class', axis=1)

bestfeatures= SelectKBest(score\_func=f\_classif, k=10)

fit = bestfeatures.fit(x,y)

dfscores = pd.DataFrame(fit.scores\_)

dfcolumns= pd.DataFrame(x.columns)

# Concat 2 different dataframe to better visualization

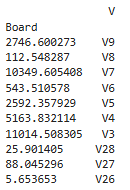
featureScores = pd.concat([dfcolumns,dfscores],axis=1)

featureScores.columns = ['V','Board']

featureScores\_df = featureScores.sort\_values(['V','Board'],ascending=[False, True])

featureScores\_df = featureScores\_df.set\_index('Board').head(10)

print(featureScores\_df)



* **CONCLUSION**

XGBoost Classifier for its high accuracy, robustness, and efficiency in handling complex classification tasks.

**1.PKL - .pkl files store serialized Python objects, allowing you to save and load data structures, models, and other objects efficiently.**

**I**

import pandas as pd

from xgboost import XGBClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix

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

credit\_card = pd.read\_csv('creditcard.csv')

y = credit\_card['Class']

x = credit\_card.drop('Class',axis=1)

col = x.columns.tolist()

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=42)

xgb = XGBClassifier()

xgb.fit(x\_train[col],y\_train)

y\_pred\_final = xgb.predict(x\_test[col])

**II**

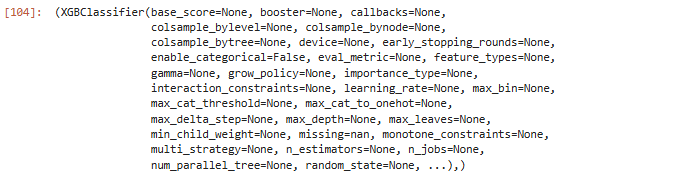
import joblib

from xgboost import XGBClassifier

joblib.dump(xgb,'final.pkl'),

clf = joblib.load('final.pkl'),

clf



**2.h5 - .h5 files store large, complex, and often scientific datasets in a structured format for efficient access and analysis.**

import h5py

credit\_card = pd.read\_csv('creditcard.csv')

with h5py.File('model.h5', 'w') as f:

f.create\_dataset('credit\_card', data=credit\_card)

with h5py.File('model.h5', 'r') as f:

loaded\_model\_data = f['credit\_card'][:]

***II***

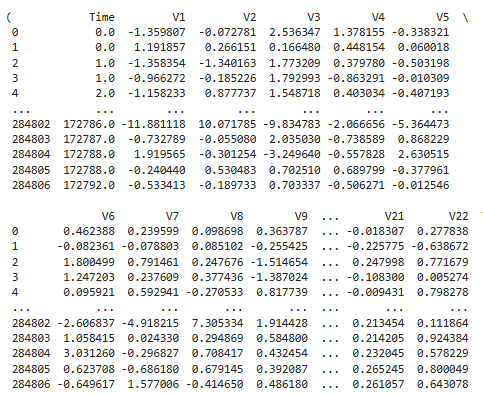
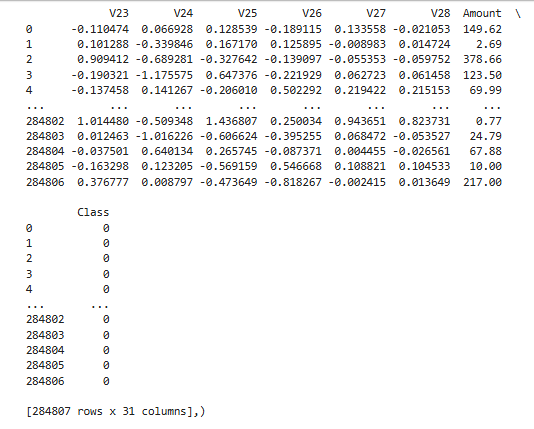
import joblib

import h5py

joblib.dump(credit\_card,'final.h5'),

clf = joblib.load('final.h5'),

clf

---------------------------------------------------------------------------------------------------------------------------------Thank You-------