```
import pandas as pd
import numpy as np
import seaborn as sns                        #visualisation
import matplotlib.pyplot as plt              #visualisation
%matplotlib inline
sns.set(color_codes=True)
pd.set_option('display.max_rows',None)
pd.set_option('display.max_columns',None)

from sklearn.preprocessing import StandardScaler    #Importing StandardScaler using sklearn library

from sklearn.model_selection import train_test_split   #To split the data in training and testing part

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score   #To generate classification report and acuracy score
```

```
df=pd.read_csv('/content/drive/MyDrive/creditcard_2023.csv')
```

```
df.head()
```

	id	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	
0	0	-0.260648	-0.469648	2.496266	-0.083724	0.129681	0.732898	0.519014	-0.130006	0.727159	0.637735	-0.987020	0.293438	-0.941386	0.
1	1	0.985100	-0.356045	0.558056	-0.429654	0.277140	0.428605	0.406466	-0.133118	0.347452	0.529808	0.140107	1.564246	0.574074	0.
2	2	-0.260272	-0.949385	1.728538	-0.457986	0.074062	1.419481	0.743511	-0.095576	-0.261297	0.690708	-0.272985	0.659201	0.805173	0.
3	3	-0.152152	-0.508959	1.746840	-1.090178	0.249486	1.143312	0.518269	-0.065130	-0.205698	0.575231	-0.752581	0.737483	0.592994	0.
4	4	-0.206820	-0.165280	1.527053	-0.448293	0.106125	0.530549	0.658849	-0.212660	1.049921	0.968046	-1.203171	1.029577	1.439310	0.

Start coding or generate with AI.

```
df.shape
```

```
(568630, 31)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 568630 entries, 0 to 568629
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   id      568630 non-null  int64  
 1   V1      568630 non-null  float64
 2   V2      568630 non-null  float64
 3   V3      568630 non-null  float64
 4   V4      568630 non-null  float64
 5   V5      568630 non-null  float64
 6   V6      568630 non-null  float64
 7   V7      568630 non-null  float64
 8   V8      568630 non-null  float64
 9   V9      568630 non-null  float64
 10  V10     568630 non-null  float64
 11  V11     568630 non-null  float64
 12  V12     568630 non-null  float64
 13  V13     568630 non-null  float64
 14  V14     568630 non-null  float64
 15  V15     568630 non-null  float64
 16  V16     568630 non-null  float64
 17  V17     568630 non-null  float64
 18  V18     568630 non-null  float64
 19  V19     568630 non-null  float64
 20  V20     568630 non-null  float64
 21  V21     568630 non-null  float64
 22  V22     568630 non-null  float64
 23  V23     568630 non-null  float64
 24  V24     568630 non-null  float64
 25  V25     568630 non-null  float64
 26  V26     568630 non-null  float64
 27  V27     568630 non-null  float64
 28  V28     568630 non-null  float64
 29  Amount  568630 non-null  float64
 30  Class   568630 non-null  int64  
dtypes: float64(29), int64(2)
memory usage: 134.5 MB
```

```
df.describe()
```

	id	V1	V2	V3	V4	V5	V6	V7
count	568630.000000	5.686300e+05	5.686300e+05	5.686300e+05	5.686300e+05	5.686300e+05	5.686300e+05	5.686300e+05
mean	284314.500000	-5.638058e-17	-1.319545e-16	-3.518788e-17	-2.879008e-17	7.997245e-18	-3.958636e-17	-3.198898e-17
std	164149.486122	1.000001e+00	1.000001e+00	1.000001e+00	1.000001e+00	1.000001e+00	1.000001e+00	1.000001e+00
min	0.000000	-3.495584e+00	-4.996657e+01	-3.183760e+00	-4.951222e+00	-9.952786e+00	-2.111111e+01	-4.351839e+00
25%	142157.250000	-5.652859e-01	-4.866777e-01	-6.492987e-01	-6.560203e-01	-2.934955e-01	-4.458712e-01	-2.835329e-01
50%	284314.500000	-9.363846e-02	-1.358939e-01	3.528579e-04	-7.376152e-02	8.108788e-02	7.871758e-02	2.333659e-01
75%	426471.750000	8.326582e-01	3.435552e-01	6.285380e-01	7.070047e-01	4.397368e-01	4.977881e-01	5.259548e-01
max	568629.000000	2.229046e+00	4.361865e+00	1.412583e+01	3.201536e+00	4.271689e+01	2.616840e+01	2.178730e+02

```
df.dtypes
```

```
id          int64
V1        float64
V2        float64
V3        float64
V4        float64
V5        float64
V6        float64
V7        float64
V8        float64
V9        float64
V10       float64
V11       float64
V12       float64
V13       float64
V14       float64
V15       float64
V16       float64
V17       float64
V18       float64
V19       float64
V20       float64
V21       float64
V22       float64
V23       float64
```

```
V24       float64
V25       float64
V26       float64
V27       float64
V28       float64
Amount    float64
Class       int64
dtype: object
```

```
df.tail()
```

	id	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	
568625	568625	-0.833437	0.061886	-0.899794	0.904227	-1.002401	0.481454	-0.370393	0.189694	-0.938153	-1.161847	1.
568626	568626	-0.670459	-0.202896	-0.068129	-0.267328	-0.133660	0.237148	-0.016935	-0.147733	0.483894	-0.210817	0.
568627	568627	-0.311997	-0.004095	0.137526	-0.035893	-0.042291	0.121098	-0.070958	-0.019997	-0.122048	-0.144495	0.
568628	568628	0.636871	-0.516970	-0.300889	-0.144480	0.131042	-0.294148	0.580568	-0.207723	0.893527	-0.080078	-0.
568629	568629	-0.795144	0.433236	-0.649140	0.374732	-0.244976	-0.603493	-0.347613	-0.340814	0.253971	-0.513556	0.

```
df.columns
```

```
Index(['id', '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')
```

```
df['Class'].value_counts()
```

```
0    284315
1    284315
Name: Class, dtype: int64
```

```
print("********* Amount Lost due to fraud:************\n")
print("Total amount lost to fraud")
print(df.Amount[df.Class == 1].sum())
print("Mean amount per fraudulent transaction")
print(df.Amount[df.Class == 1].mean().round(4))
print("Compare to normal transactions:")
print("Total amount from normal transactions")
print(df.Amount[df.Class == 0].sum())
print("Mean amount per normal transactions")
print(df.Amount[df.Class == 0].mean().round(4))
```

```
********* Amount Lost due to fraud:************

Total amount lost to fraud
3428157045.3500004
Mean amount per fraudulent transaction
12057.6018
Compare to normal transactions:
Total amount from normal transactions
3419261324.3999996
Mean amount per normal transactions
12026.3135
```

observations

1)we have 568630 rows of obervations having 31 columns

2)'class' is our output feature indicating whether the transaction is"Fraudulent"(1) or "Not Fraudulent"(0)

3)"v1-v28" anonymized features representing various transaction attributes.

4)dtype(data type) of all the features looks perfects

DATA PREPROCESSING

```
# Checking null values in the dataset
print(df.isnull().sum())
```

```
id        0
V1        0
V2        0
V3        0
```

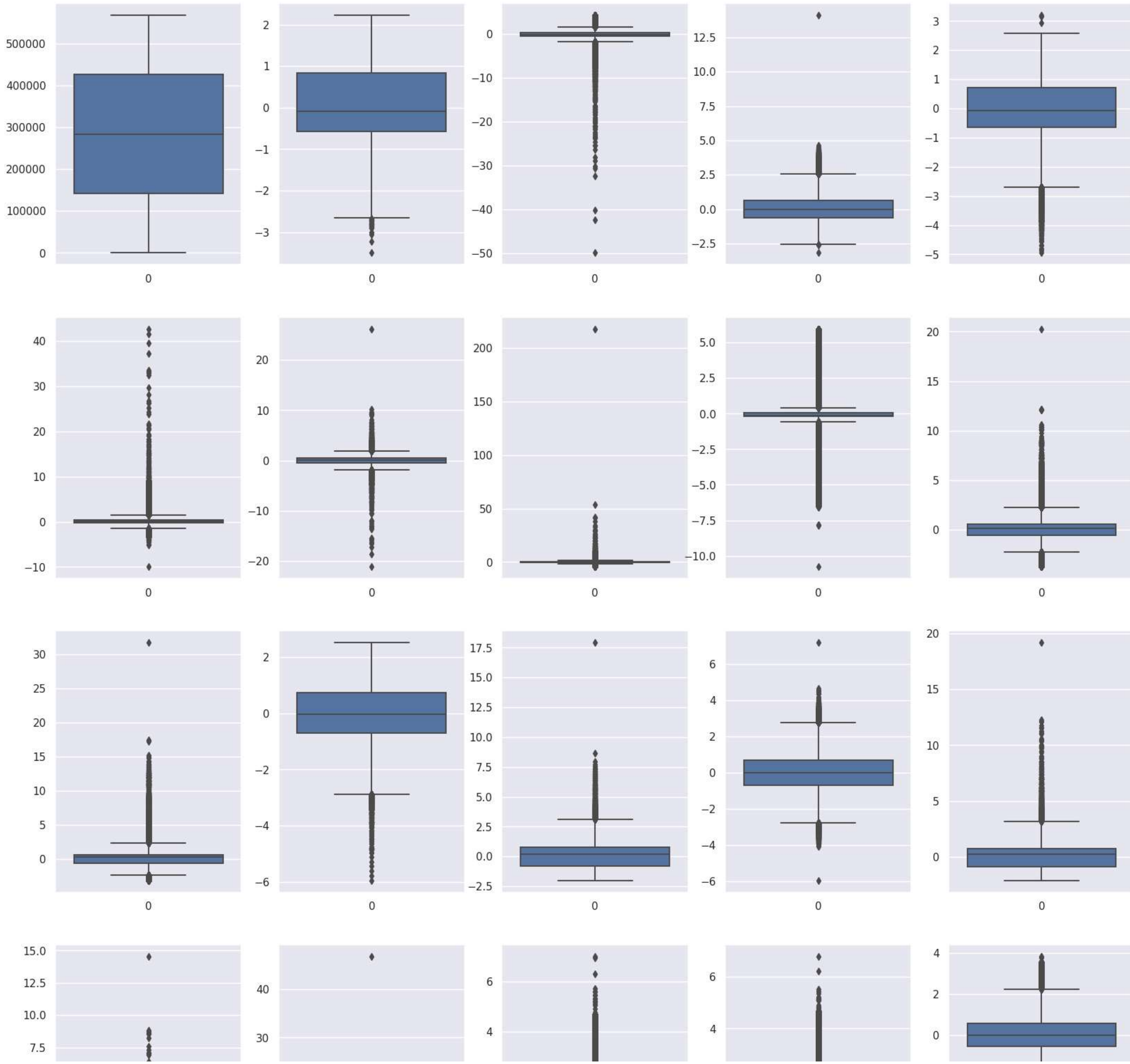
```
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

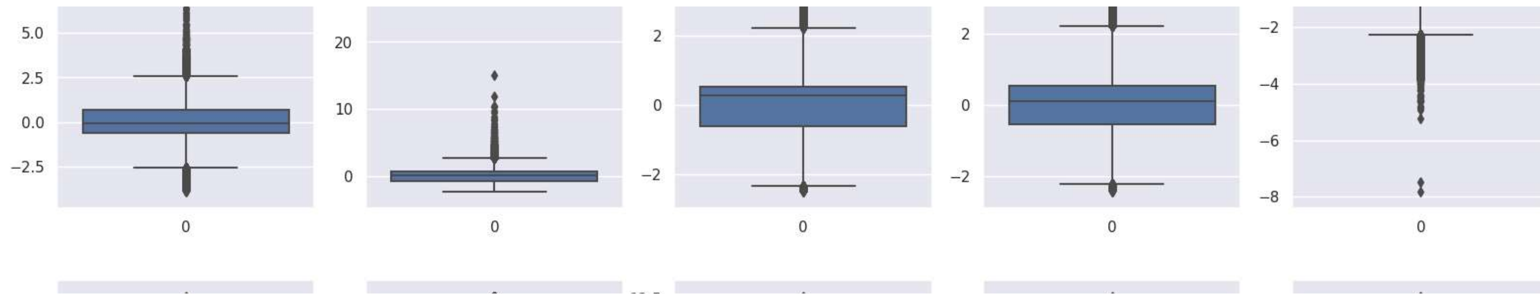
```
# Checking duplicate values in the dataset
df.duplicated().any()
```

```
False
```

Observation- No missing values No duplicates

```
plt.figure(figsize=(20, 40))
for i, col in enumerate(df.columns):
    plt.subplot(7, 5, i+1)
    sns.boxplot(df[col])
plt.show()
```

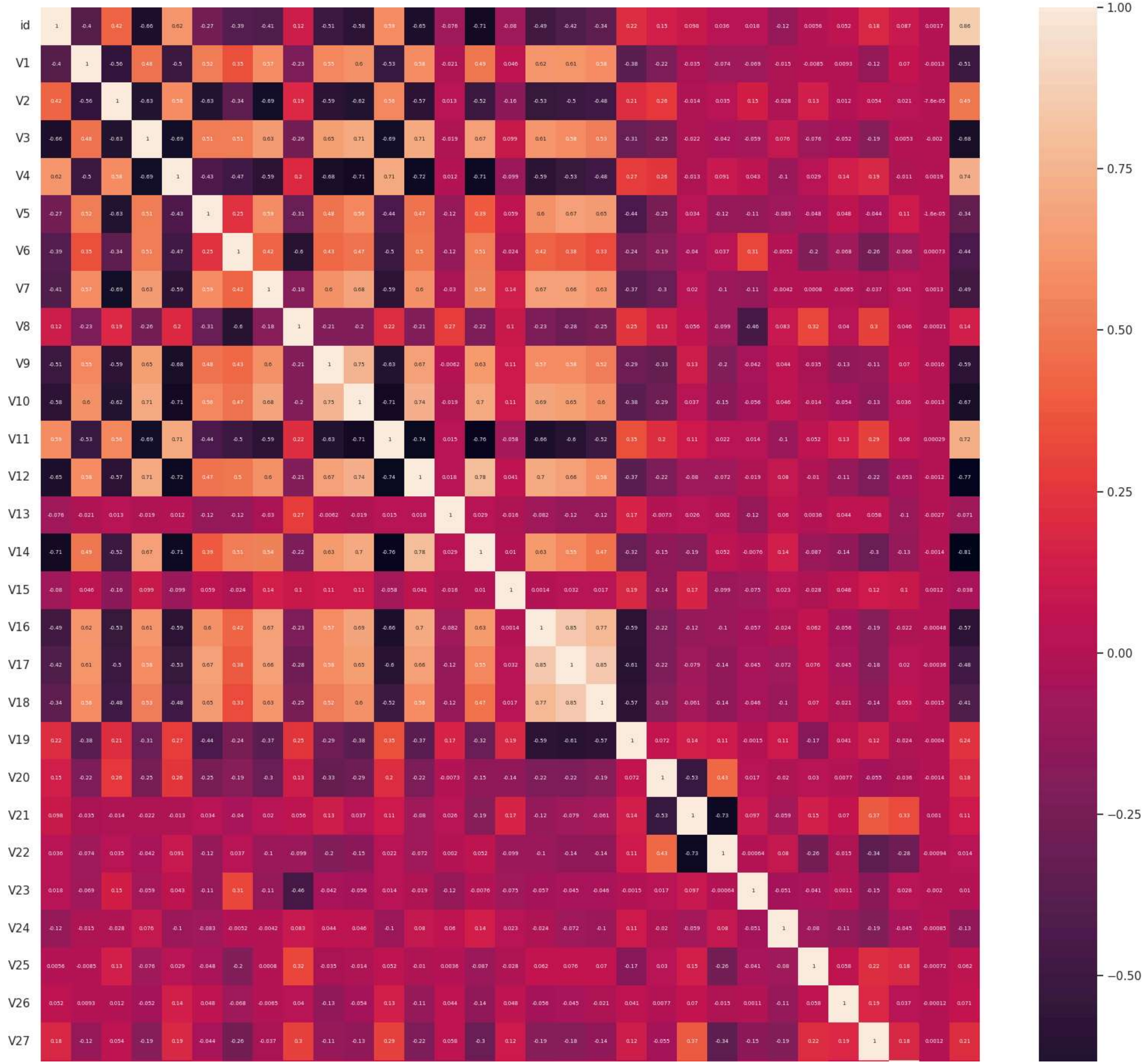

500000 400000 300000 200000 100000 0
0
2 1 0 −1 −2 −3
0
0 −10 −20 −30 −40 −50
0
12.5 10.0 7.5 5.0 2.5 0.0 −2.5
0
3 2 1 0 −1 −2 −3 −4 −5
0
40 30 20 10 0 −10
0
20 10 0 −10 −20
0
200 150 100 50 0
0
5.0 2.5 0.0 −2.5 −5.0 −7.5 −10.0
0
20 15 10 5 0
0
30 25 20 15 10 5 0
0
2 0 −2 −4 −6
0
17.5 15.0 12.5 10.0 7.5 5.0 2.5 0.0 −2.5
0
6 4 2 0 −2 −4 −6
0
20 15 10 5 0
0
15.0 12.5 10.0 7.5
40 30
6 4
6 4
4 2 0



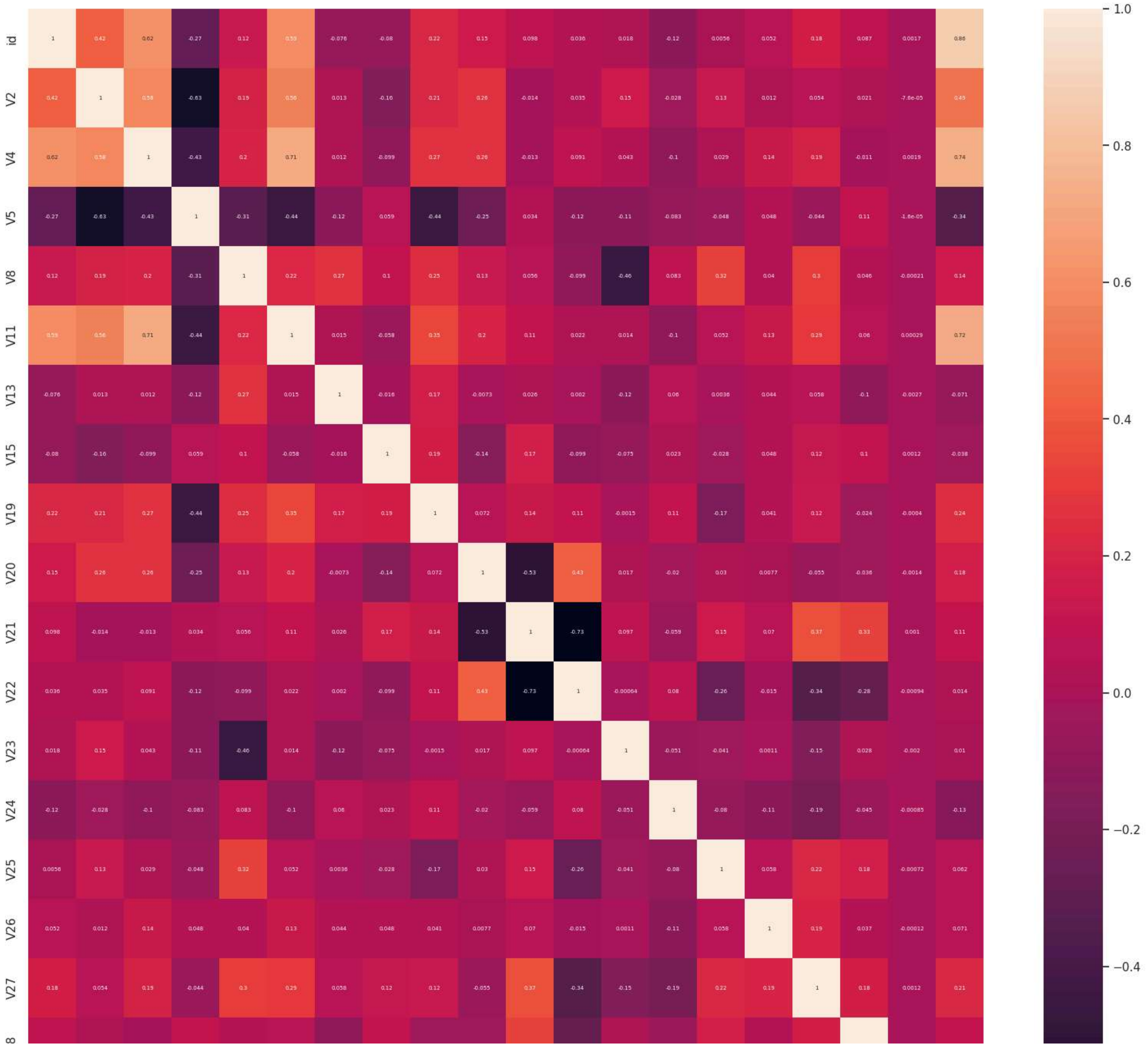

```
# Summary of the boxplots

# The boxplots show the distribution of each numerical variable in the dataset.
# The median is represented by the line in the middle of the box.
# The upper and lower quartiles are represented by the top and bottom of the box, respectively.
# The whiskers extend to the maximum and minimum values of the data, excluding outliers.
# Outliers are represented by individual points.

# The boxplots show that the numerical variables have a wide range of values.
# Some variables, such as 'Amount', 'Time', and 'V1', have a few outliers.
# Other variables, such as 'V2', 'V3', and 'V4', have no outliers.

# The boxplots can be used to identify potential outliers and to compare the distributions of different variables.
```

Double-click (or enter) to edit

```
# Doing pairplots would take massive amount of time therefore we will just do a heatmap in python
plt.figure(figsize=(20, 20))
sns.heatmap(df.corr(), annot=True, annot_kws={'size':5})
plt.show()
```

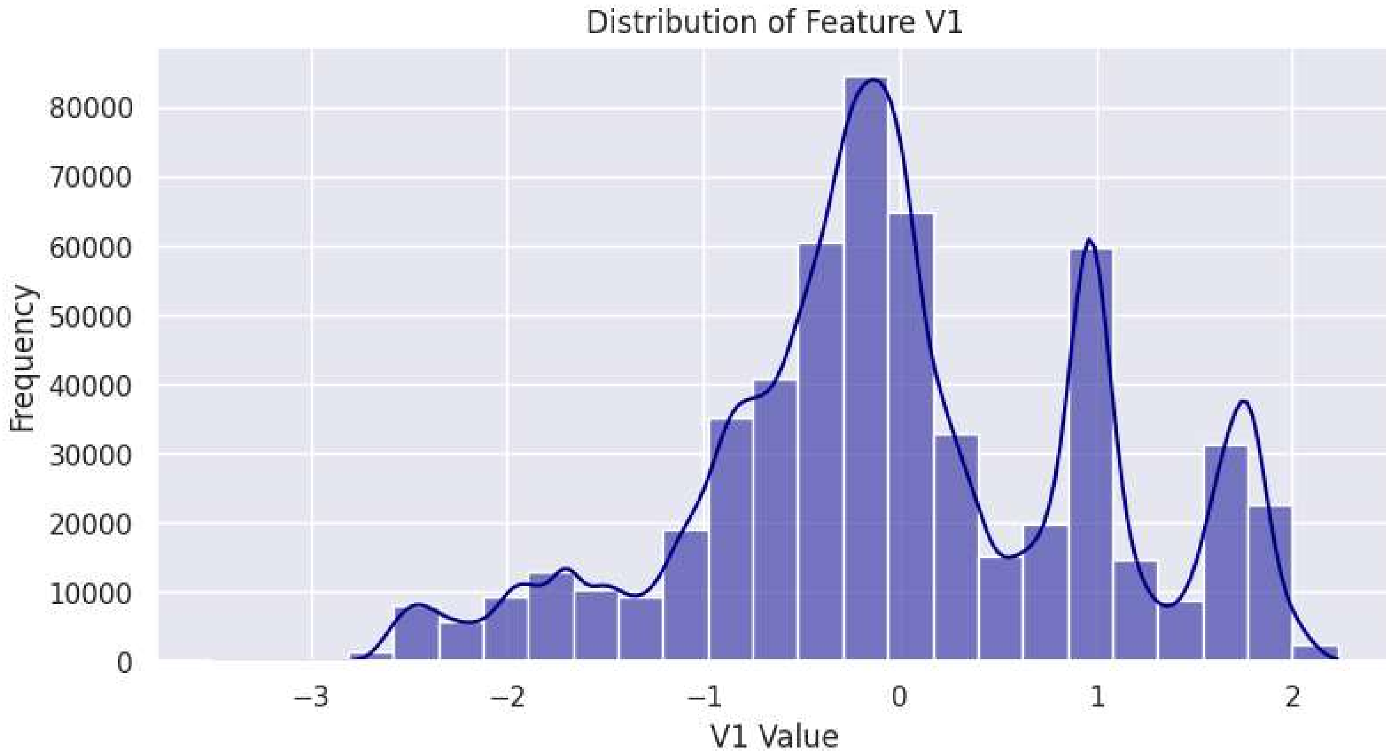

id
V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
V11
V12
V13
V14
V15
V16
V17
V18
V19
V20
V21
V22
V23
V24
V25
V26
V27
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50


```
newdf = df.drop(columns=['V1', 'V3', 'V6', 'V7', 'V9', 'V10', 'V12', 'V14', 'V16', 'V17', 'V18'])
plt.figure(figsize=(20, 20))
sns.heatmap(newdf.corr(), annot=True, annot_kws={'size':5})
plt.show()
```

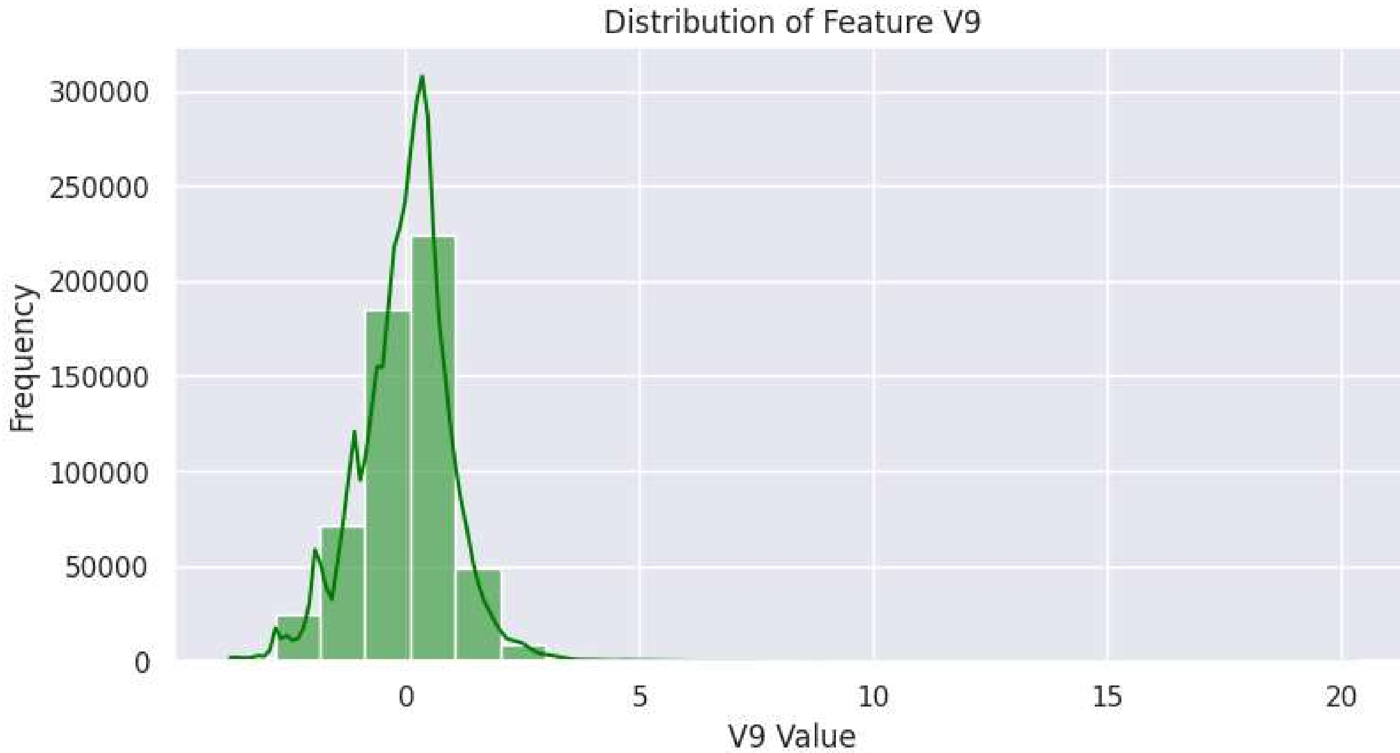


EDA (Exploratory Data Analysis)

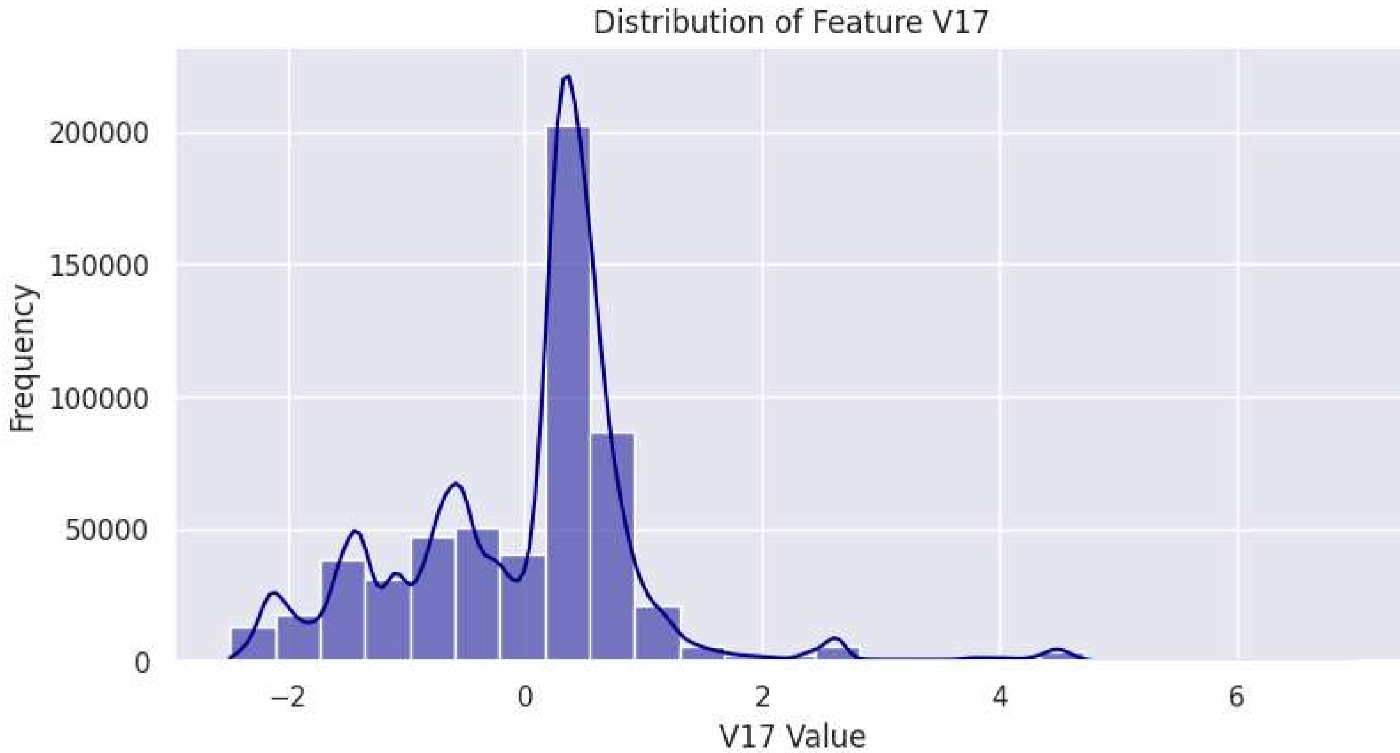
```
# Observing the Distribution of Feature V1
plt.figure(figsize=(9, 4.5))
sns.histplot(df['V1'], bins=25, kde=True, color='darkblue')
plt.title('Distribution of Feature V1')
plt.xlabel('V1 Value')
plt.ylabel('Frequency')
plt.show()
```



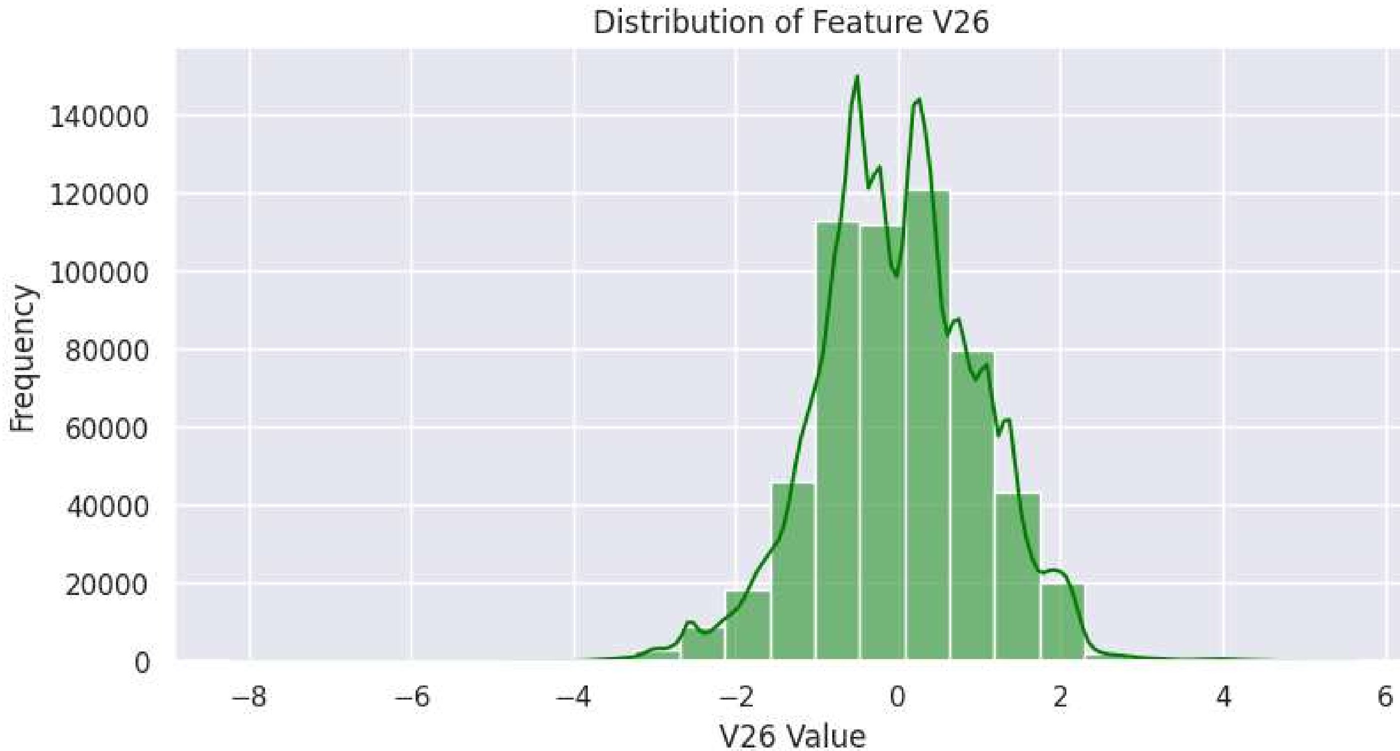

```
# Observing the Distribution of Feature V9
plt.figure(figsize=(9, 4.5))
sns.histplot(df['V9'], bins=25, kde=True, color='green')
plt.title('Distribution of Feature V9')
plt.xlabel('V9 Value')
plt.ylabel('Frequency')
plt.show()
```



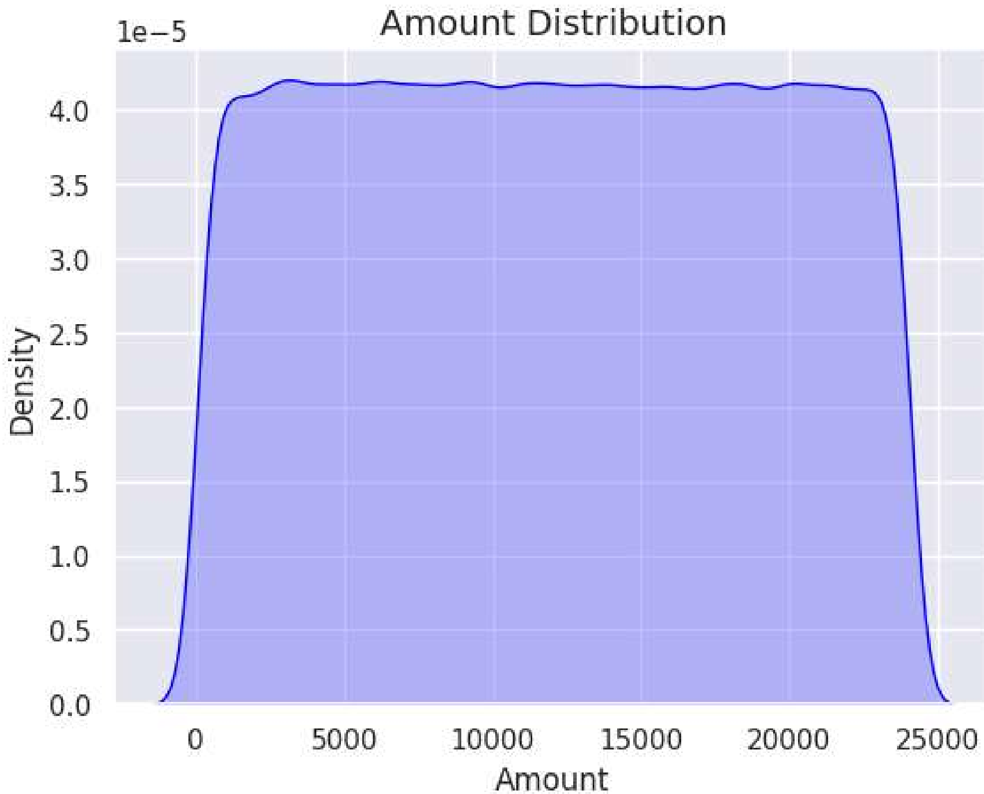

```
# Observing the Distribution of Feature V17
plt.figure(figsize=(9, 4.5))
sns.histplot(df['V17'], bins=25, kde=True, color='darkblue')
plt.title('Distribution of Feature V17')
plt.xlabel('V17 Value')
plt.ylabel('Frequency')
plt.show()
```



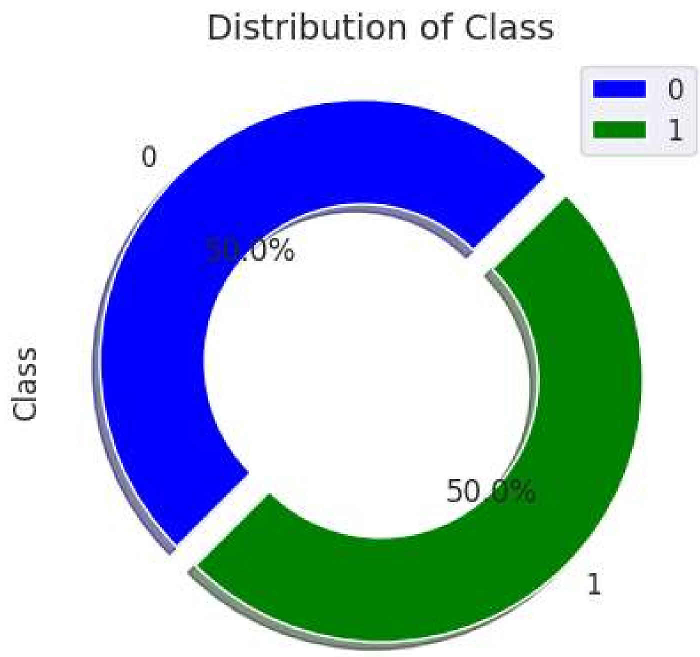

```
# Observing the Distribution of Feature V26
plt.figure(figsize=(9, 4.5))
sns.histplot(df['V26'], bins=25, kde=True, color='green')
plt.title('Distribution of Feature V26')
plt.xlabel('V26 Value')
plt.ylabel('Frequency')
plt.show()
```



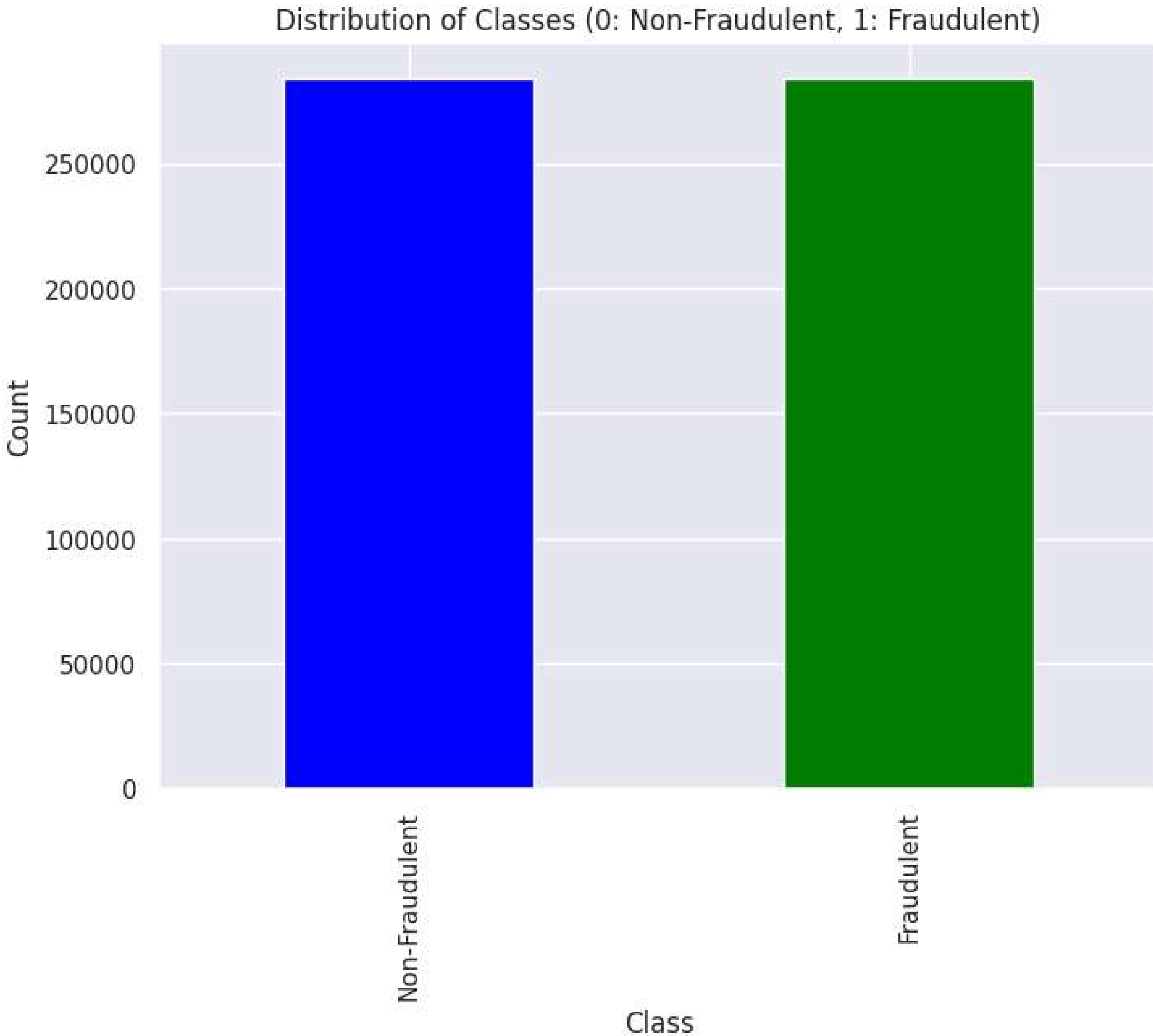

```
# Observing the Amount Disribution
sns.kdeplot(data= df['Amount'],color = 'blue', fill=True)
plt.title('Amount Distribution',size=14)
plt.show()
```




```python
# Observing the Disribution of the Feature 'Class'
colors = ['blue', 'green']
explode = [0.1, 0]
df['Class'].value_counts().plot.pie(
    explode=explode,
    autopct='%3.1f%%',
    shadow=True,
    legend=True,
    startangle=45,
    colors=colors,
    wedgeprops=dict(width=0.4)
)

plt.title('Distribution of Class',size=14)
plt.show()
```




```
# Observing the Disribution of the Feature 'Class'
plt.figure(figsize=(8, 6))
df['Class'].value_counts().plot(kind='bar', color=['blue', 'green'])
plt.title('Distribution of Classes (0: Non-Fraudulent, 1: Fraudulent)')
plt.xlabel('Class')
plt.ylabel('Count')
plt.xticks([0, 1], ['Non-Fraudulent', 'Fraudulent'])
plt.show()
```




```
# Pulling the highest correlated feature to the feature 'Class'
corrmat = df.corr()
cols = corrmat.nlargest(15,'Class')['Class'].index
cols
```

```
Index(['Class', 'id', 'V4', 'V11', 'V2', 'V19', 'V27', 'V20', 'V8', 'V21',
       'V28', 'V26', 'V25', 'V22', 'V23'],
```

```
            dtype='object')
```

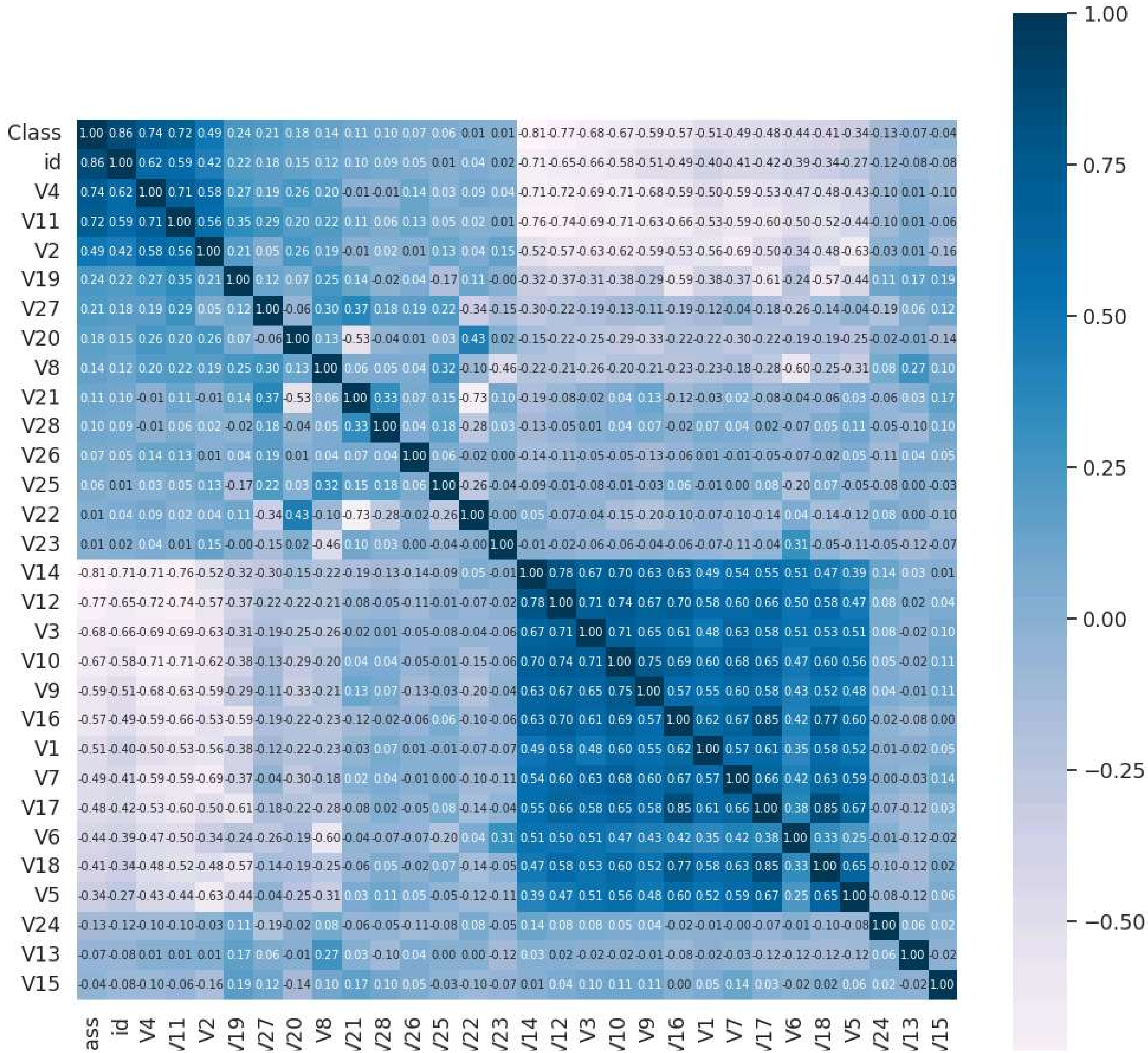
```
# Pulling the least correlated feature to the feature 'Class'
cols_negative = corrmat.nsmallest(15,'Class')['Class'].index
cols_negative
```

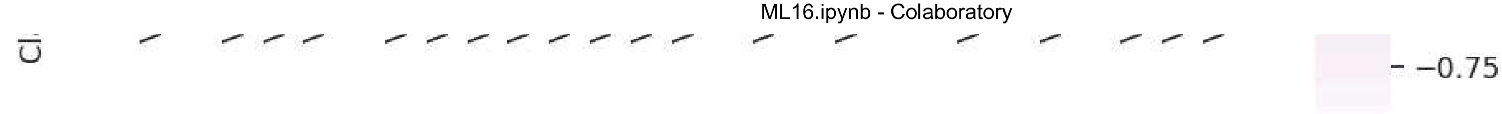
```
Index(['V14', 'V12', 'V3', 'V10', 'V9', 'V16', 'V1', 'V7', 'V17', 'V6', 'V18',
       'V5', 'V24', 'V13', 'V15'],
      dtype='object')
```

```
# Joining the two above variables in one variable 'Credit_card'
Credit_card = []
for i in cols:
    Credit_card.append (i)
for j in cols_negative:
    Credit_card.append(j)

Credit_card
```

```
['Class',
 'id',
 'V4',
 'V11',
 'V2',
 'V19',
 'V27',
 'V20',
 'V8',
 'V21',
 'V28',
 'V26',
 'V25',
 'V22',
 'V23',
 'V14',
 'V12',
 'V3',
 'V10',
 'V9',
 'V16',
 'V1',
 'V7',
 'V17',
 'V6',
 'V18',
```

```
        'V5',
        'V24',
        'V13',
        'V15']


# Observing the Correlation between features using a heatmap
corrmat = df[Credit_card].corr()
sns.set(font_scale=1.15)
f, ax = plt.subplots(figsize=(12,12))
hm = sns.heatmap(corrmat,
                 cmap='PuBu',
                 cbar=True,
                 annot=True,
                 square=True,
                 fmt='.2f',
                 annot_kws={'size': 7},
                 yticklabels=corrmat.columns,
                 xticklabels=corrmat.columns)
```


Class
id
V4
V11
V2
V19
V27
V20
V8
V21
V28
V26
V25
V22
V23
V14
V12
V3
V10
V9
V16
V1
V7
V17
V6
V18
V5
V24
V13
V15
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50



Split Data into Test and Train

```
# Split the data into features (X) and target (y).
x = df.drop(['id','Class'],axis=1)
y = df.Class


# Standardize the feature data (x)
scaler = StandardScaler()
X = scaler.fit_transform(x)
print(X)
```

```
[[-0.2606478  -0.46964845  2.49626608 ... -0.08123011 -0.15104549
   0.85844694]
 [ 0.98509973 -0.35604509  0.55805635 ... -0.24805206 -0.06451192
  -0.79636931]
 [-0.26027161 -0.94938461  1.72853778 ... -0.30025804 -0.24471823
  -1.37701093]
 ...
 [-0.31199739 -0.00409479  0.13752559 ... -0.48753975 -0.26874127
   1.66640101]
 [ 0.63687054 -0.51696952 -0.30088853 ... -0.15926926 -0.07625057
  -0.27185346]
 [-0.79514417  0.43323608 -0.64914005 ... -1.5751126   0.7229365
   1.3659619 ]]
```

```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,stratify=y)
```

Decision Tree Classifier

```
# Create a Decision Tree model

from sklearn.tree import DecisionTreeClassifier



decision_tree_model = DecisionTreeClassifier(random_state=42)



# Train the model
decision_tree_model.fit(X_train, y_train)
```

```
▾        DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

```
# Predictions on the test set
y_pred = decision_tree_model.predict(X_test)


print("Decision Tree")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nAccuracy Score:", accuracy_score(y_test, y_pred)*100,"%")
```

```
Decision Tree
Confusion Matrix:
 [[56683   180]
 [   73 56790]]

Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     56863
           1       1.00      1.00      1.00     56863

    accuracy                           1.00    113726
   macro avg       1.00      1.00      1.00    113726
weighted avg       1.00      1.00      1.00    113726
```

```
Accuracy Score: 99.77753548001337 %
```

Support Vector Machine

```
from sklearn.svm import SVC
```

```
clf = SVC()
```

```
clf.fit(X_train, y_train)
```

```
▾ SVC
SVC()
```

```
y_pred_svm = clf.predict(X_test)
```

```
print("Support Vector Machine")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_svm))
print("\nClassification Report:\n", classification_report(y_test, y_pred_svm))
print("\nAccuracy Score:", accuracy_score(y_test, y_pred_svm)*100,"%")
```

```
Support Vector Machine
Confusion Matrix:
 [[56654   209]
 [  122 56741]]

Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     56863
           1       1.00      1.00      1.00     56863

    accuracy                           1.00    113726
   macro avg       1.00      1.00      1.00    113726
weighted avg       1.00      1.00      1.00    113726
```

```
Accuracy Score: 99.70894958057085 %
```

```
Support_Vector_Machine = accuracy_score(y_test, y_pred_svm)*100
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
```

```
reg = LogisticRegression()
reg.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
y_pred_reg =  reg.predict(X_test)
```

```
print("Logistic Regression Model")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_reg))
print("\nClassification Report:\n", classification_report(y_test, y_pred_reg))
print("\nAccuracy Score:", accuracy_score(y_test, y_pred_reg)*100,"%")
```

```
Logistic Regression Model
Confusion Matrix:
 [[55593  1270]
 [ 2715 54148]]

Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.98      0.97     56863
           1       0.98      0.95      0.96     56863

    accuracy                           0.96    113726
   macro avg       0.97      0.96      0.96    113726
weighted avg       0.97      0.96      0.96    113726
```

```
Accuracy Score: 96.49596398360973 %
```

```
Logistic_Regression = accuracy_score(y_test, y_pred_reg)*100
```

Gradient Boosting Classifier(XGBoost)

```
!pip3 install xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (2.0.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.11.4)
```

```
from xgboost import XGBClassifier
```

```
xgb = XGBClassifier()
```

```
xgb.fit(X_train, y_train)
```

```
XGBClassifier

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

```
y_pred_xgb = xgb.predict(X_test)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-87-7193dc8d1e83> in <cell line: 1>()
----> 1 y_pred_xgb = xgb.predict(X_test)

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

```
print("XGBoost Model")
```