BIA

BOSTON INSTITUTE OF ANALYTICS ®

CREDIT CARD FRAUD DETECTION

BY
Royston Rodrigues

What is Credit Card Fraud?!

"AN INCLUSIVE TERM FOR FRAUD COMMITTED USING A PAYMENT CARD, SUCH AS A CREDIT CARD OR DEBIT CARD."

EXAMPLES

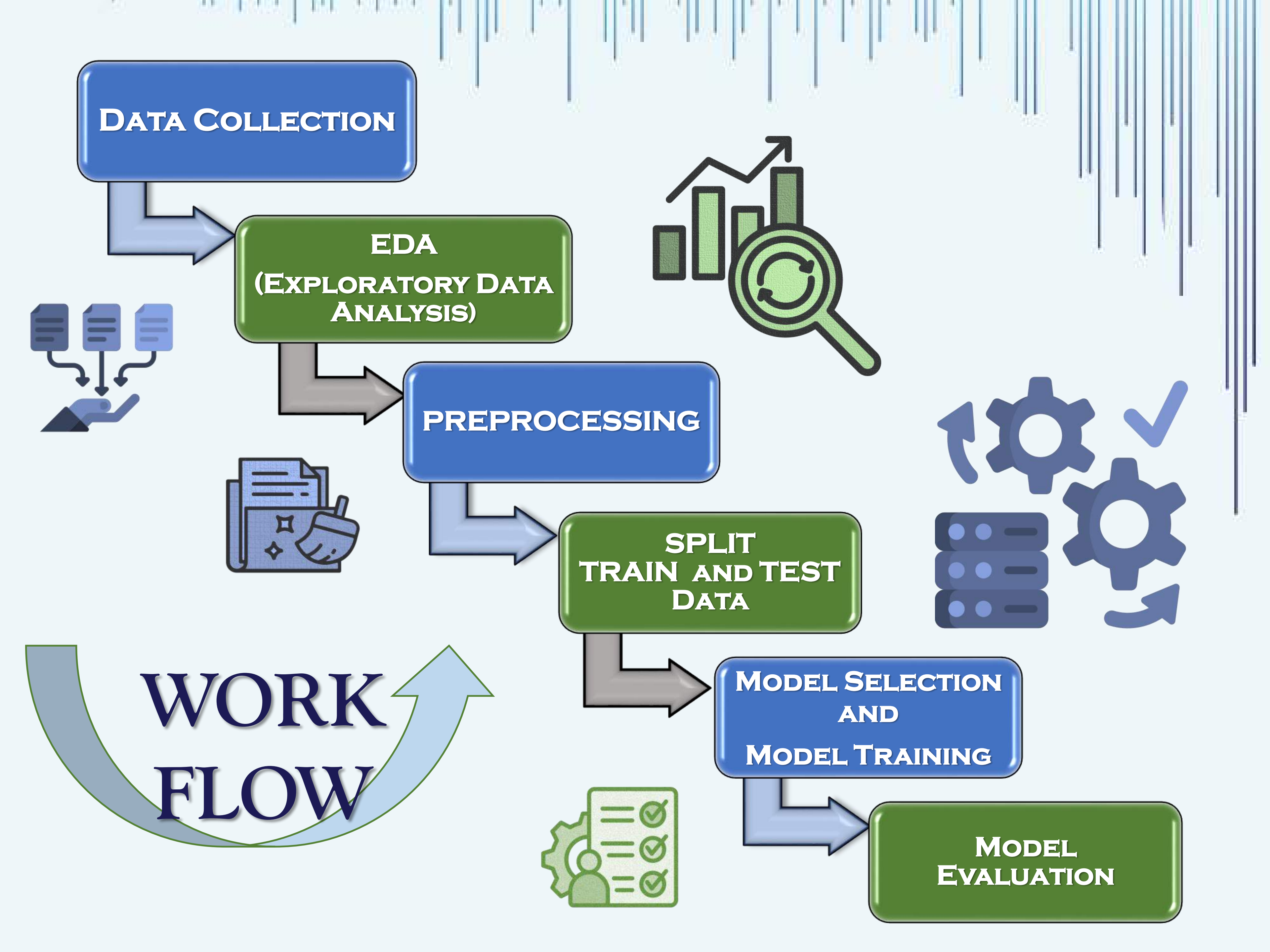
- Phishing
- Data Breaches
- Account takeover
- Hacking
- Skimming

Why is Model important?!

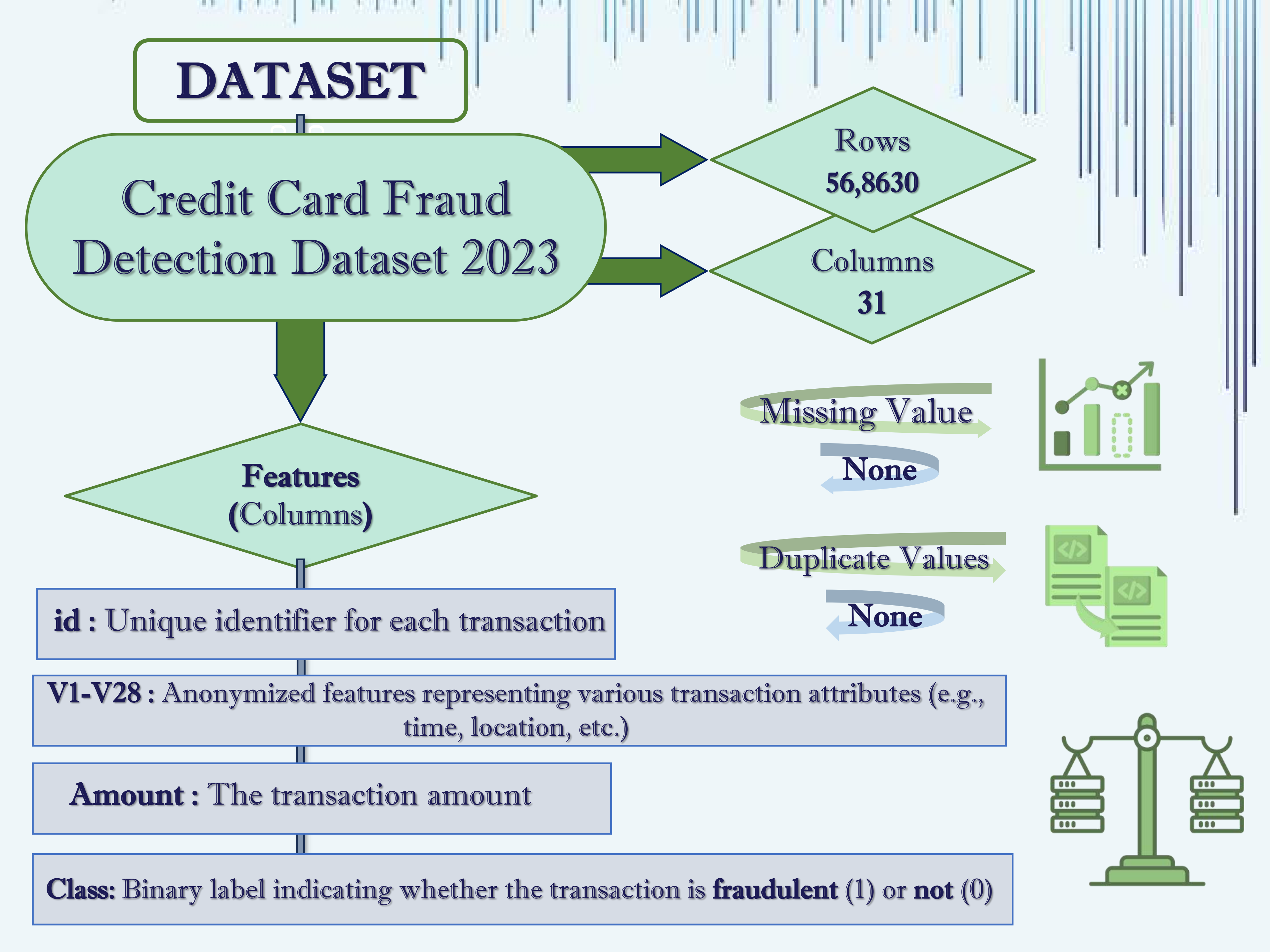
"TO PREDICT WHETHER THE TRANSACTIONS ARE **FRAUDULENT** OR **NOT**"

What will be the impact of the Model?!

ENHANCED SECURITY, REDUCED FINANCIAL LOSSES, AND IMPROVED CUSTOMER TRUST



Data Collection
EDA
(Exploratory Data Analysis)
Preprocessing
Split
Train and Test
Data
Model Selection
and
Model Training
Model
Evaluation
WORK
FLOW


DATASET
Credit Card Fraud Detection Dataset 2023
Rows
56,8630
Columns
31
Features (Columns)
Missing Value
None
Duplicate Values
None
id : Unique identifier for each transaction
V1-V28 : Anonymized features representing various transaction attributes (e.g., time, location, etc.)
Amount : The transaction amount
Class: Binary label indicating whether the transaction is fraudulent (1) or not (0)

EDA

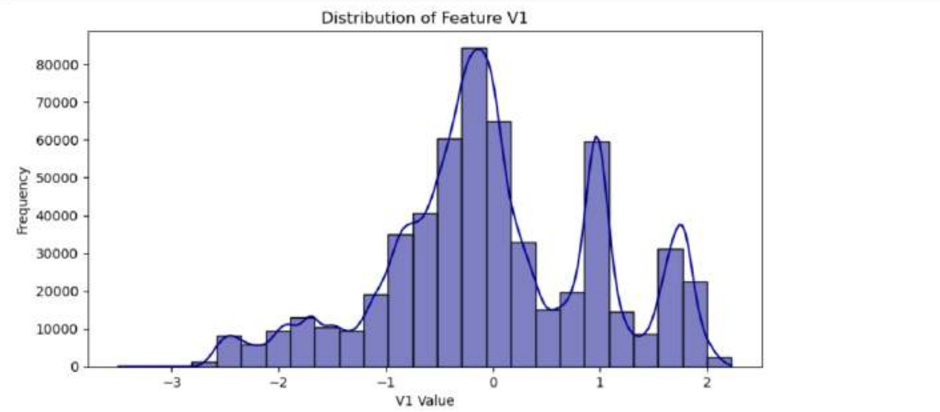
Exploratory Data Analysis

Exploratory Data Analysis is a data analysis approach that involves summarizing main characteristics of a dataset, often using visual methods, to understand its structure, identify patterns, and uncover potential relationships

df.info()	Gives a summary of the dataset including column names, data types, non-null values, and memory usage
df.dtypes	Returns the data type of each feature of a dataset.
df.head()	Displays the **first** five rows of a dataset
df.tail()	Displays the **last** five rows of dataset.
df.columns	Returns a list of column names in the dataset.
df.describe()	Gives descriptive statistics of a dataset

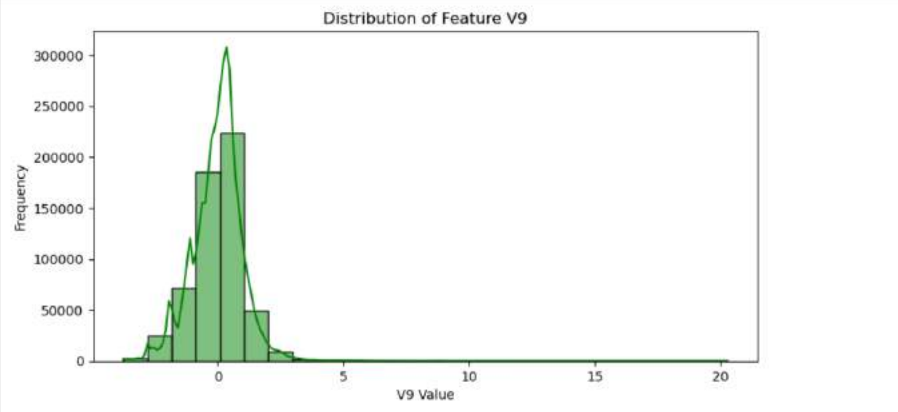
Observing the Distribution of Feature V1

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



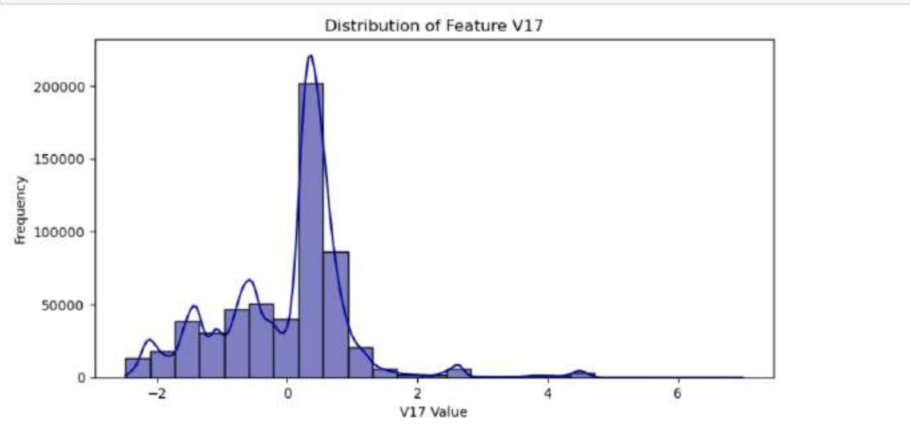
Observing the Distribution of Feature V9

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



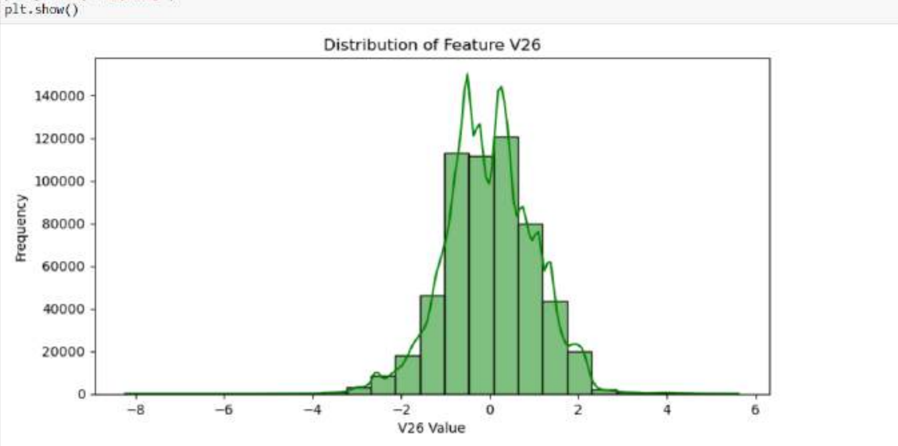
Observing the Distribution of Feature V17

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



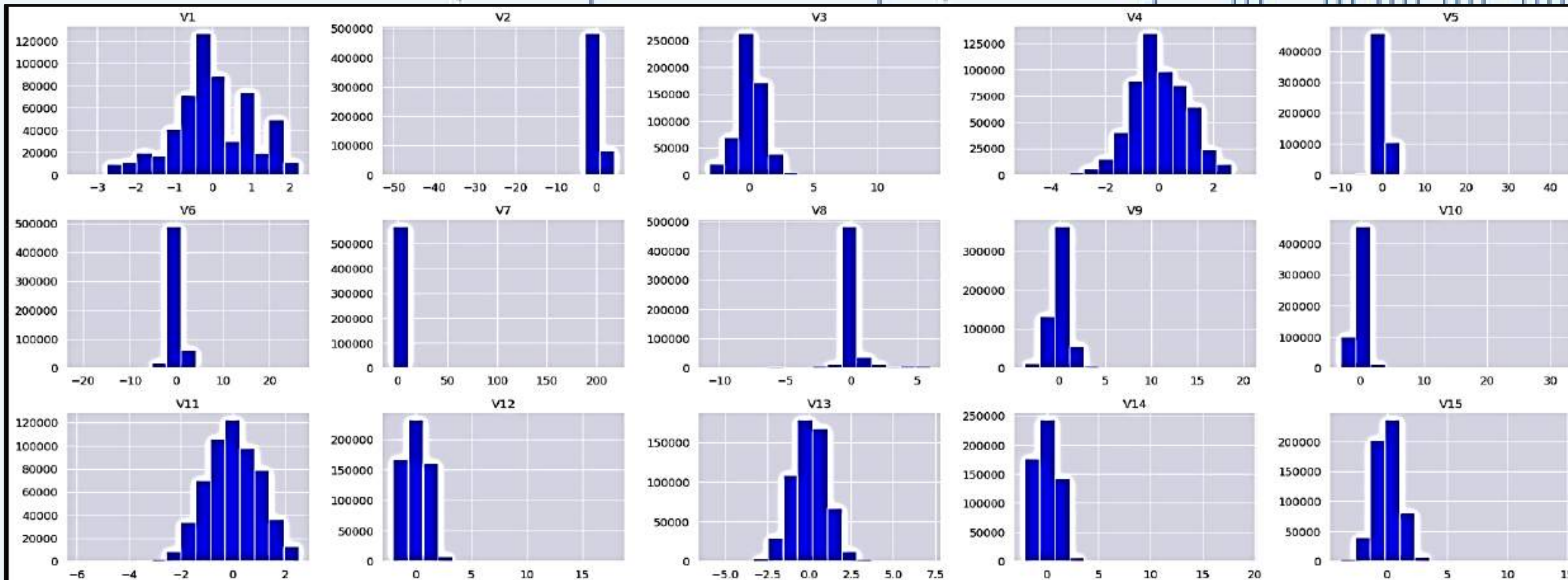
Observing the Distribution of Feature V26

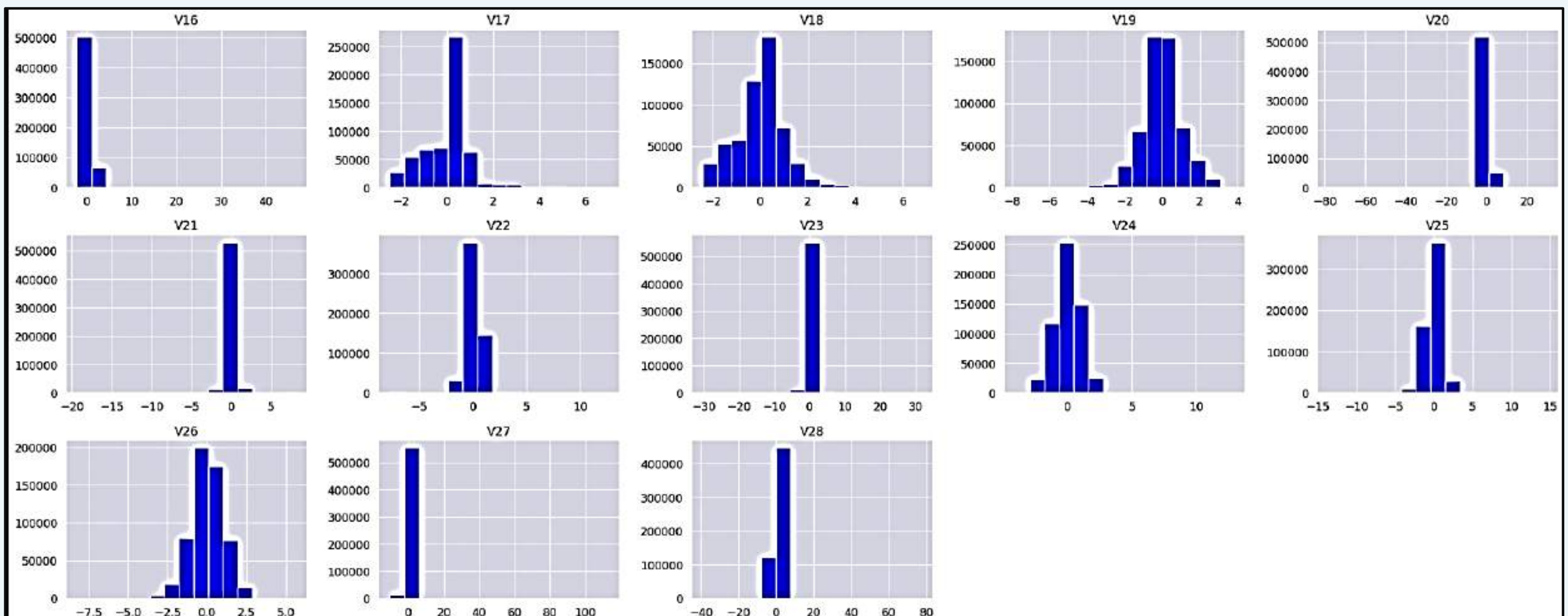
```
plt.figure(figsize=(9, 4.5))
sns.histplot(credit_card['V26'], bins=25, kde=True, color='green')
plt.title('Distribution of Feature V26')
plt.xlabel('V26 Value')
plt.ylabel('Frequency')
plt.show()
```



Observing the Distribution of Features

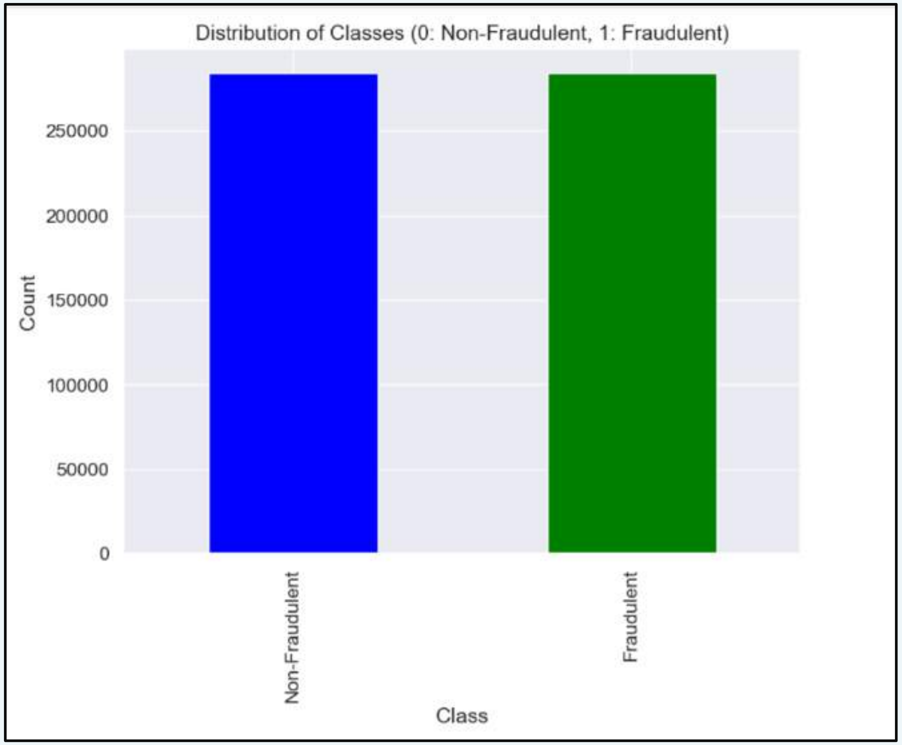
V1 – V15





V16 – V28

Observation of the Classes
0:Non fraudulent
1:Fradulent



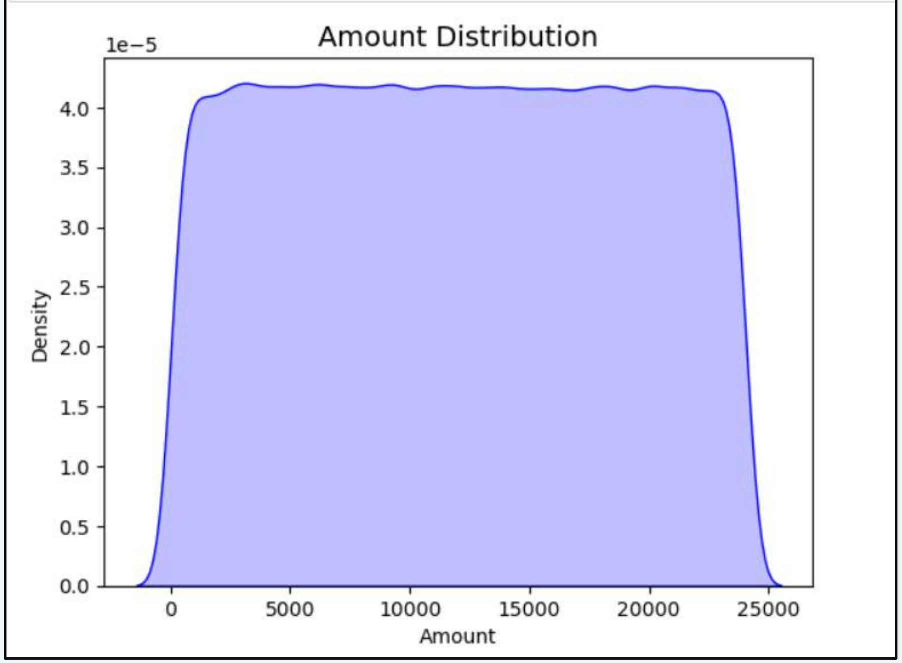
The feature 'Class' is evenly *Balanced*

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

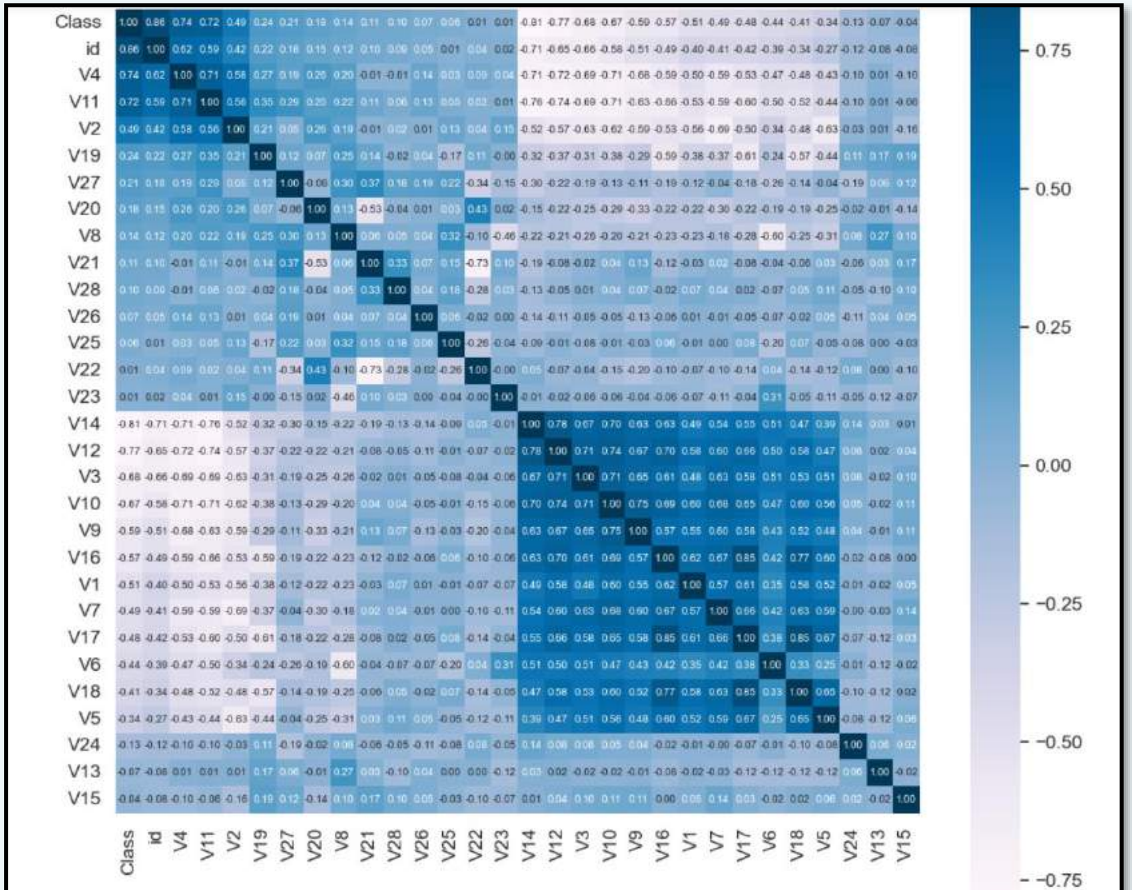
```
Class
0    284315
1    284315
Name: count, dtype: int64
```

Observation of Distribution of Amount

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



```python
# Observing the Correlation between features using a heatmap
corrmat = credit_card[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)
```




```python
# 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')
```

```python
# Pulling the highest correlated feature to the feature 'Class'
corrmat = credit_card.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')
```

Heatmap to Understand the correlation between features

Dividing dataset into “ *x* ” and “ *y* ”

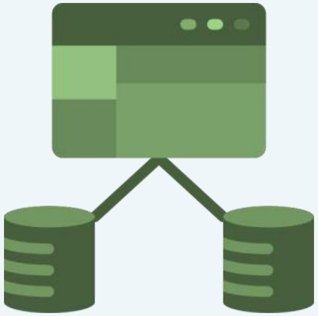
```
# Split the data into features (X) and target (y).
x = credit_card.drop(['id','Class'],axis=1)
y = credit_card.Class
```

Standardize the feature data (x)

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

Dividing dataset into Training Data and Testing Data

```
# 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)
```



MODEL SELECTION AND TRAINING

RandomForestClassifier

Model

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()

rf.fit(X_train, y_train)

▾ RandomForestClassifier
RandomForestClassifier()

y_pred_rf = rf.predict(X_test)
```

Classification report

```
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: 99.97977595272849 %
```

Support Vector Machine (SVM)

Model

```
from sklearn.svm import SVC

clf = SVC()

clf.fit(X_train, y_train)

▾ SVC
SVC()

y_pred_svm = clf.predict(X_test)
```

Classification report

```
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 %
```

MODEL SELECTION AND TRAINING

Logistic Regression

Model

```
from sklearn.linear_model import LogisticRegression

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

▾ LogisticRegression
LogisticRegression()

y_pred_reg =  reg.predict(X_test)
```

Classification report

```
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 %
```

Gradient Boosting Classifier (XGBoost)

Classification report

```
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: 99.97010358229429 %
```

Model

```
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,

y_pred_xgb = xgb.predict(X_test)
```

CAMPARISION

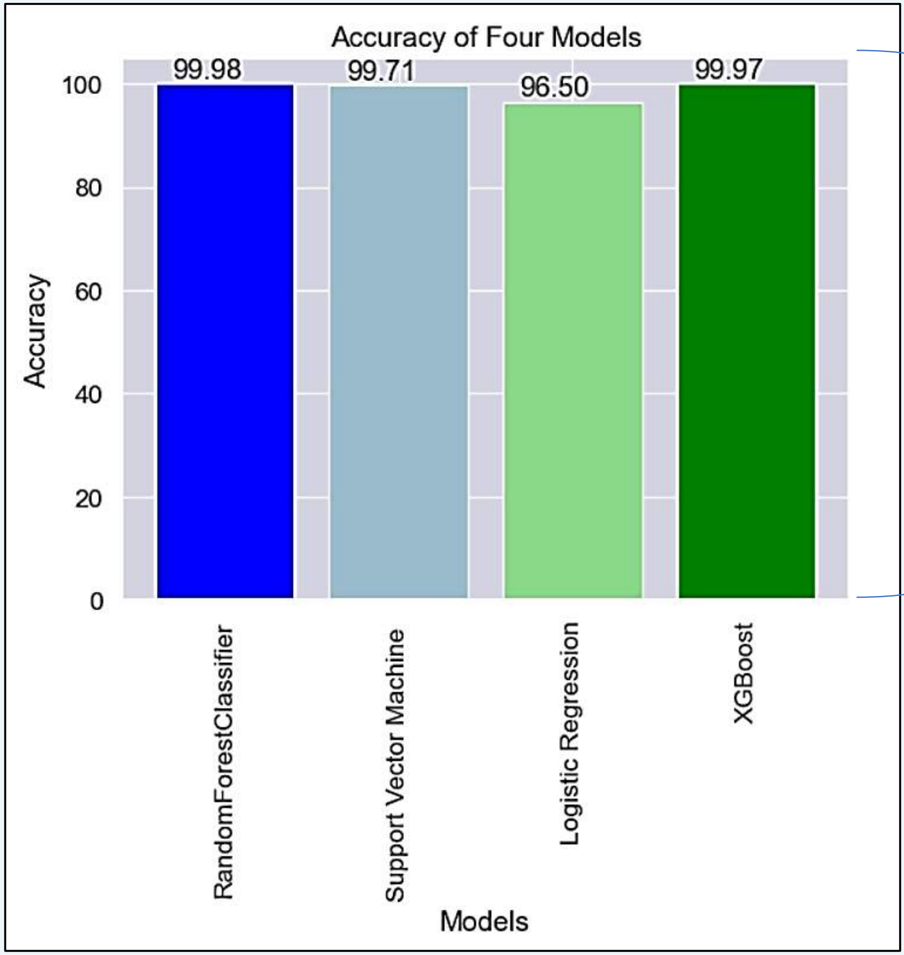
ALGORITHM	ACCURACY	CONFUSION MATRIX	CLASSIFICATTION REPORT
RandomForestClassifier	99.979776%	Randon Forest Classifier Confusion Matrix: [[56841 22] [1 56862]]	Classification Report: precision recall f1-score support 0 1.00 1.00 1.00 56863 1 1.00 1.00 1.00 56863 accuracy 1.00 113726 macro avg 1.00 1.00 1.00 113726 weighted avg 1.00 1.00 1.00 113726
Support Vector Machine (SVM)	99.708950%	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
Logistic Regression	96.495963%	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
Gradient Boosting Classifier (XGBoost)	99.970103%	XGBoost Model Confusion Matrix: [[56829 34] [0 56863]]	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

```python
model_names = ['RandomForestClassifier', 'Support Vector Machine ', 'Logistic Regression', 'XGBoost']
accuracy_values = [RandomForestClassifier, Support_Vector_Machine, Logistic_Regression, XGBoost]

bars = plt.bar(model_names, accuracy_values, color=['blue', 'lightblue', 'lightgreen', 'green'])
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Accuracy of Four Models')
plt.xticks(rotation=90)

for bar, value in zip(bars, accuracy_values):
    plt.text(bar.get_x() + bar.get_width() / 2 - 0.1, bar.get_height() + 0.01, f'{value:.2f}', ha='center', va='bottom')

plt.show()
```



This shows that out of the 4 Models, the RainForestClassifier performs the best by returning the best accuracy

Summarizing the Accuracy Scores of the 4 Models

THANK YOU

Use Your Credit Card Wisely !