# Credit Card Fraud Detection Project

Keeping customers money safe is primary job of any bank. Any fraud with credit card end up with loss to bank in terms of money, market reputation and customer trust.

Dataset: [https://www.kaggle.com/mlg-ulb/creditcardfraud](https://www.kaggle.com/mlg-ulb/creditcardfraud)

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

## 1. Importing the Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

## 2. Loading the dataset

```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mount("/content/d
```

```
df = pd.read_csv('/content/drive/MyDrive/projectData/creditcard.csv')
df.head()
```

| | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | .. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.0 | -1.359807 | -0.072781 | 2.536347 | 1.378155 | -0.338321 | 0.462388 | 0.239599 | 0.098698 | 0.363787 | . |
| 1 | 0.0 | 1.191857 | 0.266151 | 0.166480 | 0.448154 | 0.060018 | -0.082361 | -0.078803 | 0.085102 | -0.255425 | . |
| 2 | 1.0 | -1.358354 | -1.340163 | 1.773209 | 0.379780 | -0.503198 | 1.800499 | 0.791461 | 0.247676 | -1.514654 | . |
| 3 | 1.0 | -0.966272 | -0.185226 | 1.792993 | -0.863291 | -0.010309 | 1.247203 | 0.237609 | 0.377436 | -1.387024 | . |
| 4 | 2.0 | -1.158233 | 0.877737 | 1.548718 | 0.403034 | -0.407193 | 0.095921 | 0.592941 | -0.270533 | 0.817739 | . |

5 rows × 31 columns

## 3. Data Preprocessing

```
df.shape
```

```
(284807, 31)
```

Here we have total 31 columns and 284807 rows of data.

```
df.describe()
```

| | Time | V1 | V2 | V3 | V4 | V5 | |
|---|---|---|---|---|---|---|---|
| **count** | 284807.000000 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e |
| **mean** | 94813.859575 | 1.168375e-15 | 3.416908e-16 | -1.379537e-15 | 2.074095e-15 | 9.604066e-16 | 1.487313 |
| **std** | 47488.145955 | 1.958696e+00 | 1.651309e+00 | 1.516255e+00 | 1.415869e+00 | 1.380247e+00 | 1.332271e |
| **min** | 0.000000 | -5.640751e+01 | -7.271573e+01 | -4.832559e+01 | -5.683171e+00 | -1.137433e+02 | -2.616051e |
| **25%** | 54201.500000 | -9.203734e-01 | -5.985499e-01 | -8.903648e-01 | -8.486401e-01 | -6.915971e-01 | -7.682956 |
| **50%** | 84692.000000 | 1.810880e-02 | 6.548556e-02 | 1.798463e-01 | -1.984653e-02 | -5.433583e-02 | -2.741871 |
| **75%** | 139320.500000 | 1.315642e+00 | 8.037239e-01 | 1.027196e+00 | 7.433413e-01 | 6.119264e-01 | 3.985649 |
| **max** | 172792.000000 | 2.454930e+00 | 2.205773e+01 | 9.382558e+00 | 1.687534e+01 | 3.480167e+01 | 7.330163e |

8 rows × 31 columns

### Handle missing values

```
# Cheking missing values in columns
df_missing_values = df.isnull().sum()
df_missing_values
```

```
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
```

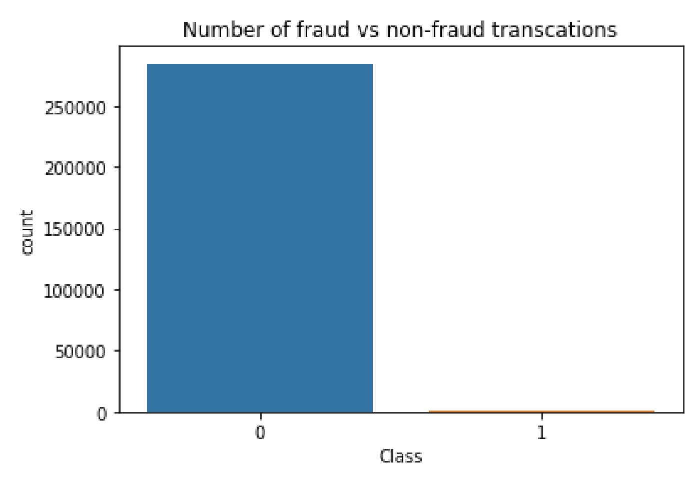
```
Class     0
dtype: int64
```

### Data distribution analysis

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

```
0    284315
1       492
Name: Class, dtype: int64
```

```
sns.countplot(x='Class', data=df)
plt.title('Number of fraud vs non-fraud transcations')
plt.show()
```




### Distribution of classes with time

```
# Creating fraud dataframe
data_fraud = df[df['Class'] == 1]
# Creating non fraud dataframe
data_non_fraud = df[df['Class'] == 0]
```

```
# Distribution plot
plt.figure(figsize=(8,5))
ax = sns.distplot(data_fraud['Time'],label='fraudt',hist=False)
ax = sns.distplot(data_non_fraud['Time'],label='non fraud',hist=False)
ax.set(xlabel='Seconds elapsed between the transction and the first transction')
plt.show()
```

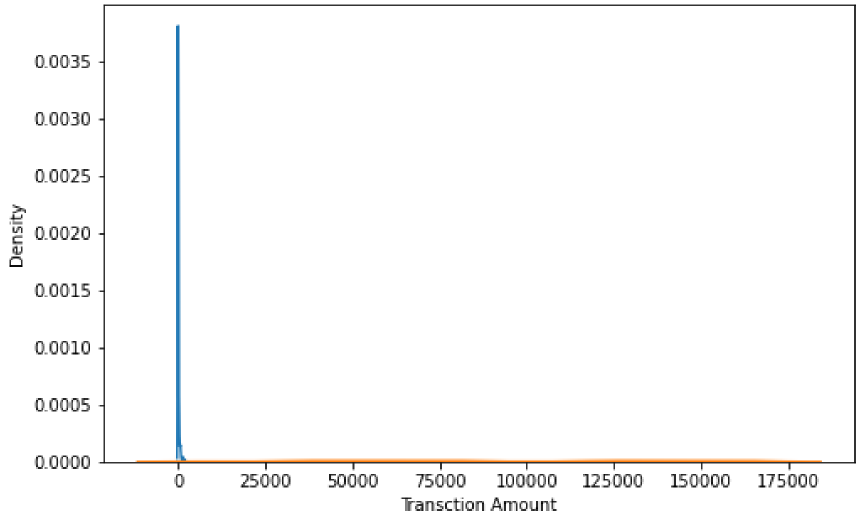
We do not see any specific pattern for the fraudulent and non-fraudulent transctions with respect to Time.

Hence, we can drop the Time column.

```
# Dropping the Time column
df.drop('Time', axis=1, inplace=True)
```

**Distribution of classes with amount**

```
# Distribution plot
plt.figure(figsize=(8,5))
ax = sns.distplot(data_fraud['Amount'],label='fraudulent',hist=False)
ax = sns.distplot(data_non_fraud['Time'],label='non fraudulent',hist=False)
ax.set(xlabel='Transction Amount')
plt.show()
```




We can see that the fraud transctions are mostly densed in the lower range of amount,

whereas the non-fraud transctions are spreaded throughout low to high range of amount.

# 4. Data Preparation for Training the Model

**Train-Test Split**

```
# Import library
from sklearn.model_selection import train_test_split
# Putting feature variables into X
X = df.drop(['Class'], axis=1)
# Putting target variable to y
y = df['Class']
# Splitting data into train and test set 80:20
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, test_size=0.2, random_state=100)
```

**Feature Scaling**

We need to scale only the Amount column as all other columns are already scaled by the PCA transformation.

```
# Standardization method
from sklearn.preprocessing import StandardScaler
# Instantiate the Scaler
scaler = StandardScaler()
```

```
# Fit the data into scaler and transform
X_train['Amount'] = scaler.fit_transform(X_train[['Amount']])
X_train.head()
```

| | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | |
|---|---|---|---|---|---|---|---|---|---|---|
| **201788** | 2.023734 | -0.429219 | -0.691061 | -0.201461 | -0.162486 | 0.283718 | -0.674694 | 0.192230 | 1.124319 | -0. |
| **179369** | -0.145286 | 0.736735 | 0.543226 | 0.892662 | 0.350846 | 0.089253 | 0.626708 | -0.049137 | -0.732566 | 0. |
| **73138** | -3.015846 | -1.920606 | 1.229574 | 0.721577 | 1.089918 | -0.195727 | -0.462586 | 0.919341 | -0.612193 | -0. |
| **208679** | 1.851980 | -1.007445 | -1.499762 | -0.220770 | -0.568376 | -1.232633 | 0.248573 | -0.539483 | -0.813368 | 0. |
| **206534** | 2.237844 | -0.551513 | -1.426515 | -0.924369 | -0.401734 | -1.438232 | -0.119942 | -0.449263 | -0.717258 | 0. |

5 rows × 29 columns

**Scale the test set**

```
# Transform the test set
X_test['Amount'] = scaler.transform(X_test[['Amount']])
X_test.head()
```

| | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | |
|---|---|---|---|---|---|---|---|---|---|---|
| **49089** | 1.229452 | -0.235478 | -0.627166 | 0.419877 | 1.797014 | 4.069574 | -0.896223 | 1.036103 | 0.745991 | -0. |
| **154704** | 2.016893 | -0.088751 | -2.989257 | -0.142575 | 2.675427 | 3.332289 | -0.652336 | 0.752811 | 1.962566 | -1. |
| **67247** | 0.535093 | -1.469185 | 0.868279 | 0.385462 | -1.439135 | 0.368118 | -0.499370 | 0.303698 | 1.042073 | -0. |
| **251657** | 2.128486 | -0.117215 | -1.513910 | 0.166456 | 0.359070 | -0.540072 | 0.116023 | -0.216140 | 0.680314 | 0. |
| **201903** | 0.558593 | 1.587908 | -2.368767 | 5.124413 | 2.171788 | -0.500419 | 1.059829 | -0.254233 | -1.959060 | 0. |

5 rows × 29 columns

# 5. Applying Models

```
# Impoting metrics
from sklearn import metrics
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_report, f1_score
```

```
results = pd.DataFrame(columns=['Model Name', 'Accuracy', 'F1-score', 'ROC'])
```

```
# ROC Curve function

def draw_roc( actual, probs ):
    fpr, tpr, thresholds = metrics.roc_curve( actual, probs,
```

```
                                                  drop_intermediate = False )
    auc_score = metrics.roc_auc_score( actual, probs )
    plt.figure(figsize=(5, 5))
    plt.plot( fpr, tpr, label='ROC curve (area = %0.2f)' % auc_score )
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate or [1 - True Negative Rate]')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()

    return None
```

***Logistic regression***

```
# Importing scikit logistic regression module
from sklearn.linear_model import LogisticRegression
# Instantiate the model with best C
logistic = LogisticRegression(C=0.01)
```

```
# Fit the model on the train set
logistic_model = logistic.fit(X_train, y_train)
```

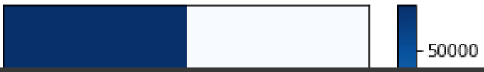
```
# Prepare results function
def display_test_results(model_name, model):

    # Prediction on the test set
    y_test_pred = model.predict(X_test)

    # Confusion matrix
    print("------------------ Confusion Matrix --------------------")
    c_matrix = metrics.confusion_matrix(y_test, y_test_pred)
    print(c_matrix)

    cm_display = ConfusionMatrixDisplay(confusion_matrix=c_matrix)
    cm_display.plot(cmap=plt.cm.Blues)
    plt.show()


    # classification_report
    print("------------------ classification_report --------------------")
    print(classification_report(y_test, y_test_pred))

    print("------------------ More Specific classification_report --------------------")
    TP = c_matrix[1,1] # true positive
    TN = c_matrix[0,0] # true negatives
    FP = c_matrix[0,1] # false positives
    FN = c_matrix[1,0] # false negatives

    # Accuracy
    print("Accuracy:-",metrics.accuracy_score(y_test, y_test_pred))

    # Sensitivity
    print("Sensitivity:-",TP / float(TP+FN))

    # Specificity
    print("Specificity:-", TN / float(TN+FP))

    # F1 score
    print("F1-Score:-", f1_score(y_test, y_test_pred))
```

```
    # Predicted probability
    y_test_pred_proba = model.predict_proba(X_test)[:,1]

    # roc_auc
    print("------------------ ROC --------------------")
    roc_auc = metrics.roc_auc_score(y_test, y_test_pred_proba)

    # Plot the ROC curve
    draw_roc(y_test, y_test_pred_proba)

    # add all metrics score in final result store
    results.loc[len(results)] = [model_name, metrics.accuracy_score(y_test, y_test_pred), f1_score(y_test, y

    return None
```

### Prediction results Using Logistic Reggression

```
display_test_results("Logistic Regression", logistic_model)
```

```
------------------ Confusion Matrix --------------------
[[56852    14]
 [   44    52]]
```




We can see that we have very good ROC on the test set 0.98, which is almost close to 1.

***XGBoost***

```
# Importing XGBoost
from xgboost import XGBClassifier
params = {'learning_rate': 0.2,
          'max_depth': 2,
          'n_estimators':200,
          'subsample':0.9,
         'objective':'binary:logistic'}
```

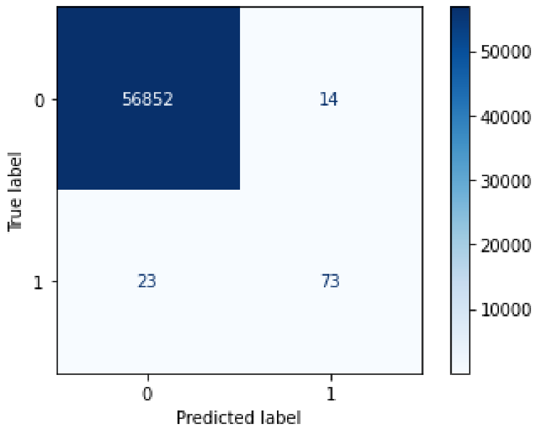
```
# fit model on training data
xgb_model = XGBClassifier(params = params)
xgb_model.fit(X_train, y_train)
```

```
XGBClassifier(params={'learning_rate': 0.2, 'max_depth': 2, 'n_estimators': 200,
                      'objective': 'binary:logistic', 'subsample': 0.9})
```

**Prediction Results Using XGBoost**

```
display_test_results("XG Boost", xgb_model)
```

```
------------------ Confusion Matrix --------------------
[[56852    14]
 [   23    73]]
```




### *Decision Tree*

```
# Importing decision tree classifier
from sklearn.tree import DecisionTreeClassifier
```

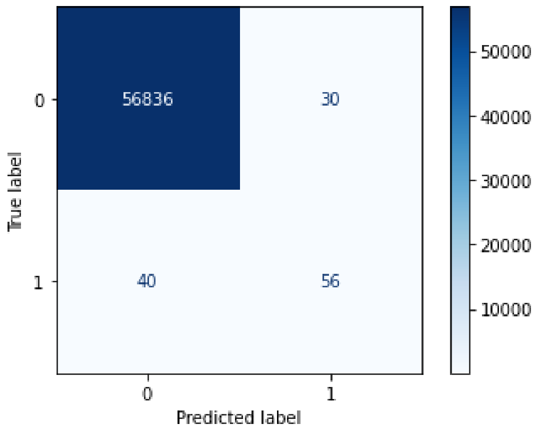
```
# Model with optimal hyperparameters
decision_tree_model = DecisionTreeClassifier(criterion = "gini",
                                  random_state = 100,
                                  max_depth=5,
                                  min_samples_leaf=100,
                                  min_samples_split=100)

decision_tree_model.fit(X_train, y_train)
```

```
DecisionTreeClassifier(max_depth=5, min_samples_leaf=100, min_samples_split=100,
                       random_state=100)
```

### Prediction Results Using Decision Tree

```
display_test_results("Decision Tree", decision_tree_model)
```

```
------------------ Confusion Matrix --------------------
[[56836    30]
 [   40    56]]
```




```
------------------ classification_report --------------------
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56866
           1       0.65      0.58      0.62        96

    accuracy                           1.00     56962
   macro avg       0.83      0.79      0.81     56962
weighted avg       1.00      1.00      1.00     56962
```

***Random Forest***

```
# Importing random forest classifier
from sklearn.ensemble import RandomForestClassifier
random_forest_model = RandomForestClassifier(bootstrap=True,
                             max_depth=5,
                             min_samples_leaf=50,
                             min_samples_split=50,
                             max_features=10,
                             n_estimators=100)
```

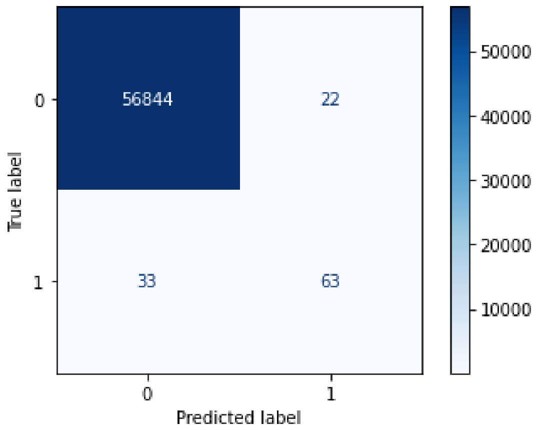
```
# Fit the model
random_forest_model.fit(X_train, y_train)
```

```
RandomForestClassifier(max_depth=5, max_features=10, min_samples_leaf=50,
                       min_samples_split=50)
```

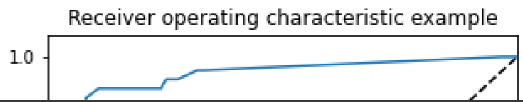
**Prediction Results Using Random forest**

```
display_test_results("Random Forest", random_forest_model)
```

```
----------------- Confusion Matrix -------------------
[[56844    22]
 [   33    63]]
```




```
----------------- classification_report -------------------
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56866
           1       0.74      0.66      0.70        96

    accuracy                           1.00     56962
   macro avg       0.87      0.83      0.85     56962
weighted avg       1.00      1.00      1.00     56962

----------------- More Specific classification_report -------------------
Accuracy:- 0.9990344440153085
Sensitivity:- 0.65625
Specificity:- 0.9996131255935005
F1-Score:- 0.6961325966850829
----------------- ROC -------------------
```




## 6. Summary

```
results.sort_values(by="ROC", ascending=False)
```

| | Model Name | Accuracy | F1-score | ROC |
|---|---|---|---|---|
| 0 | Logistic Regression | 0.998982 | 0.641975 | 0.977696 |
| 1 | XG Boost | 0.999350 | 0.797814 | 0.976238 |
| 3 | Random Forest | 0.999034 | 0.696133 | 0.961433 |
| 2 | Decision Tree | 0.998771 | 0.615385 | 0.921750 |

We can see that **XG Boost** Algorithm performing Better.

0s completed at 2:00 PM

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.