In Summary;

In this project, I developed a Python-based solution to detect fraudulent credit card transactions using machine learning techniques. The process began by importing essential libraries such as pandas, numpy, and machine learning frameworks like scikit-learn and XGBoost. After exploring and preprocessing the dataset, I analyzed the distribution of transactions to identify and separate fraudulent and legitimate ones, creating a specialized dataset for focused analysis. I calculated the percentage of fraudulent transactions from the original dataset, providing valuable context about the dataset's imbalance, a common challenge in fraud detection.

To build and evaluate predictive models, I split the data into training and testing sets. Several classification algorithms, including Logistic Regression, Random Forest Classifier, and XGBoost, were applied to test their effectiveness. Performance metrics such as the F1 Score were used to measure the balance between precision and recall. Among the models tested, the XGBoost model delivered the best results, offering superior prediction accuracy and reliability for detecting fraudulent transactions. This project demonstrates the effectiveness of advanced machine learning models in addressing real-world challenges like fraud detection, emphasizing the importance of data preparation and model evaluation in achieving accurate predictions.

* Import necessay libraries.

Data Processing

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Machine Learning Dependencies

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.feature_selection import RFE
from sklearn.datasets import load_iris
from xgboost import XGBClassifier

import random
from deap import base, creator, tools, algorithms
```

* Import the dataset.

```
data =
pd.read_csv('C:\\Users\\Asus\\OneDrive\\Masaüstü\\website\\python\\PREDICTION\\credit
_card_detection\\creditcard.csv')
print('No of rows: ', data.shape[0])
print('No of columns: ', data.shape[1])
```

* Check the first columns.

```
data.head(3)
```

* Check the column names.

```
data.columns
```

* Drop the time column.

```
data.drop(['Time'], axis = 1, inplace=True)
```

* Information about the dataset.

```
data.info()
```

* Null values.

```
data.isnull().sum()
```

There is no null values.

* Describe the dataset.

```
data.describe()
```

* Create the new dataset.

```
data_fraud = data[data.Class ==1]
data_true = data[data.Class ==0]
data_true = data_true.sample(frac = 0.5)
data_new = pd.concat([data_true, data_fraud])
data_new = data.reset_index(drop=True)
data_new.shape
```

* See the new dataset nulls.

```
data_new.isna().sum()
```

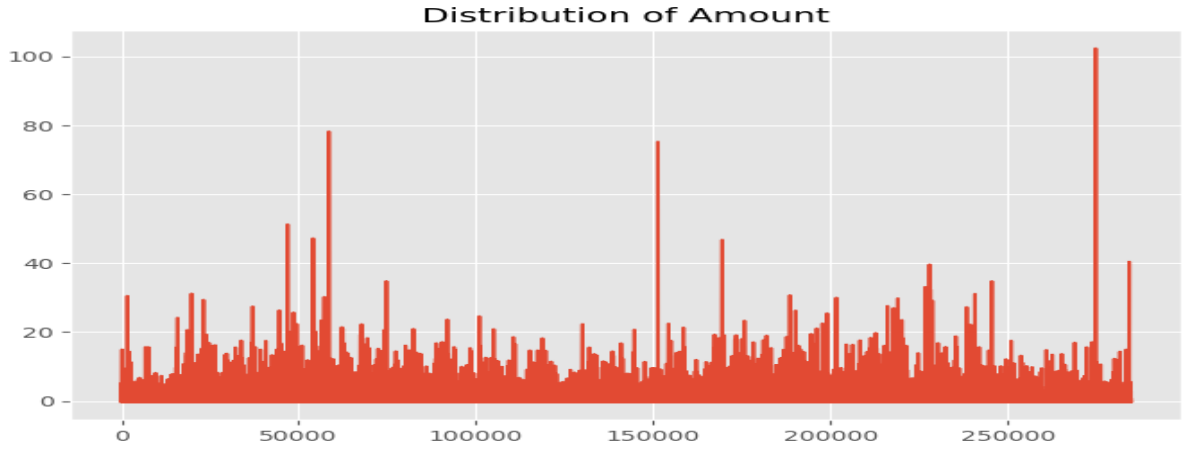
No null values.

* Describe the amount column.

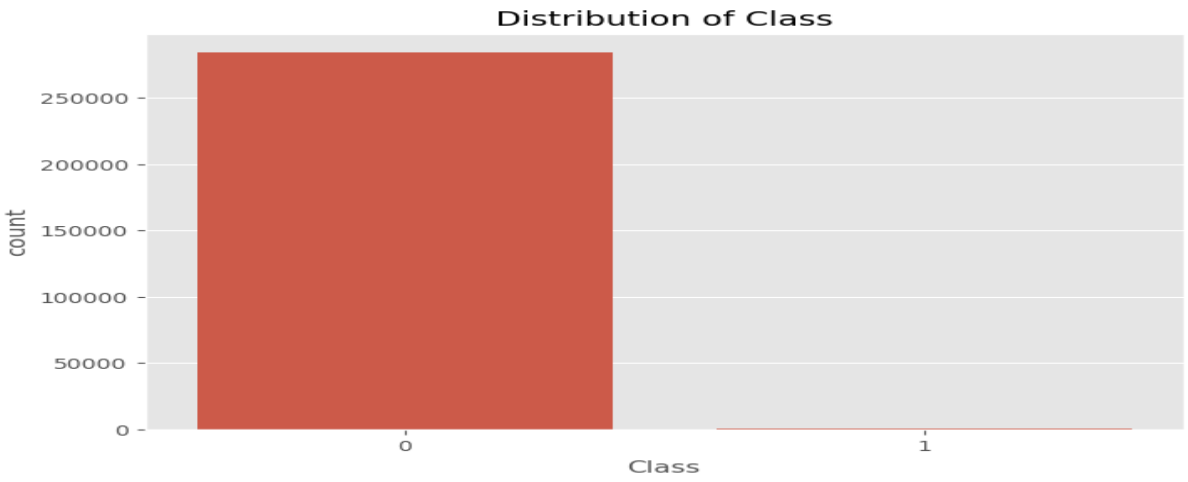
```
data_new['Amount'].describe()

data_new.Amount

with plt.style.context(('ggplot')):
    plt.figure(figsize=(8,5))
    plt.title('Distribution of Amount')
    plt.plot(data_new['Amount'])
    plt.show()
```




```
with plt.style.context(('ggplot')):
    plt.figure(figsize=(8,5))
    plt.title('Distribution of Class')
    sns.countplot(data = data_new, x =data['Class'])
    plt.show()
```



* Percentage of fraud transaction.

```
fraud_per = round((len(data_fraud)/len(data_true))*100,2)
fraud_per
```

* Scaling the amount column with standart scaling process.

```
sc = StandardScaler()
data_new.Amount = sc.fit_transform(data_new.Amount.values.reshape(-1,1))
```

* Amount column after transofrmation.

```
data_new.Amount
```

* Drop duplicates.

```
data_new.drop_duplicates(inplace=True)
data_new.shape
```

* Train - Test Split

```
x = data_new.drop(['Class'], axis =1)
y = data_new['Class']
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.8)
```

* Logistic Regression

```
model_log = LogisticRegression()
model_log.fit(x_train, y_train)
y_pred_log = model_log.predict(x_test)
print(f'The accuracy of the model is {round(model_log.score(x_test, y_test)*100,2)}%')
```

* F1 Score

```
print(f'F1 - Score of the model is {round(f1_score(y_test, y_pred_log),2)}')
with plt.style.context(('ggplot')):
   sns.heatmap(confusion_matrix(y_test, y_pred_log), annot = True)
```

* Random Forest Classifier

```
model = RandomForestClassifier()
model.fit(x_train, y_train)
y_pred_random = model.predict(x_test)
print(f'The accuracy of the model is {round(model.score(x_test, y_test)*100,2)}%')
with plt.style.context(('ggplot')):
   sns.heatmap(confusion_matrix(y_test, y_pred_random), annot = True)
   # right corner value increased, it means that it is better result.
print(f'The accuracy of the model is {round(f1_score(y_test, y_pred_random),2)}')
estimator_log = LogisticRegression()
from sklearn.feature_selection import RFECV


estimator_log = LogisticRegression()



# Use RFECV to automatically select the optimal number of features
    rfecv = RFECV(estimator=estimator_log, step=1, cv=5, scoring='accuracy')


# Fit RFECV
    rfecv.fit(x_train, y_train)


# Make predictions
    y_log_rfecv = rfecv.predict(x_test)
```

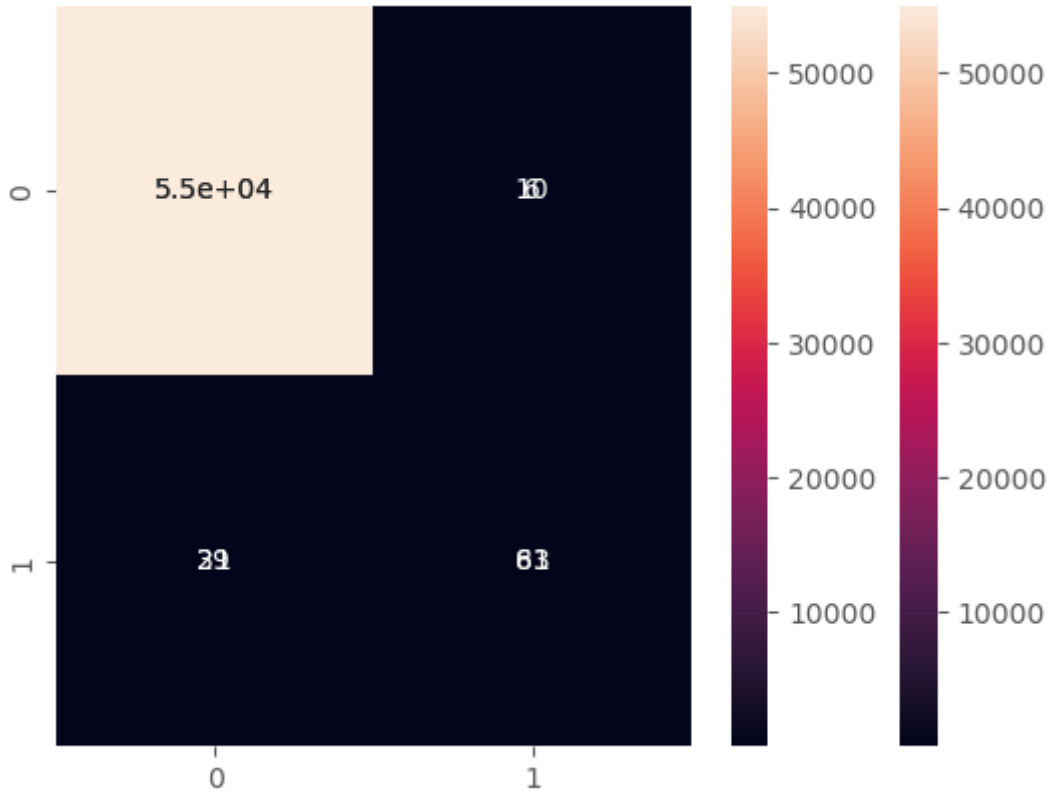
* Random forest gave better result than logistic regression.

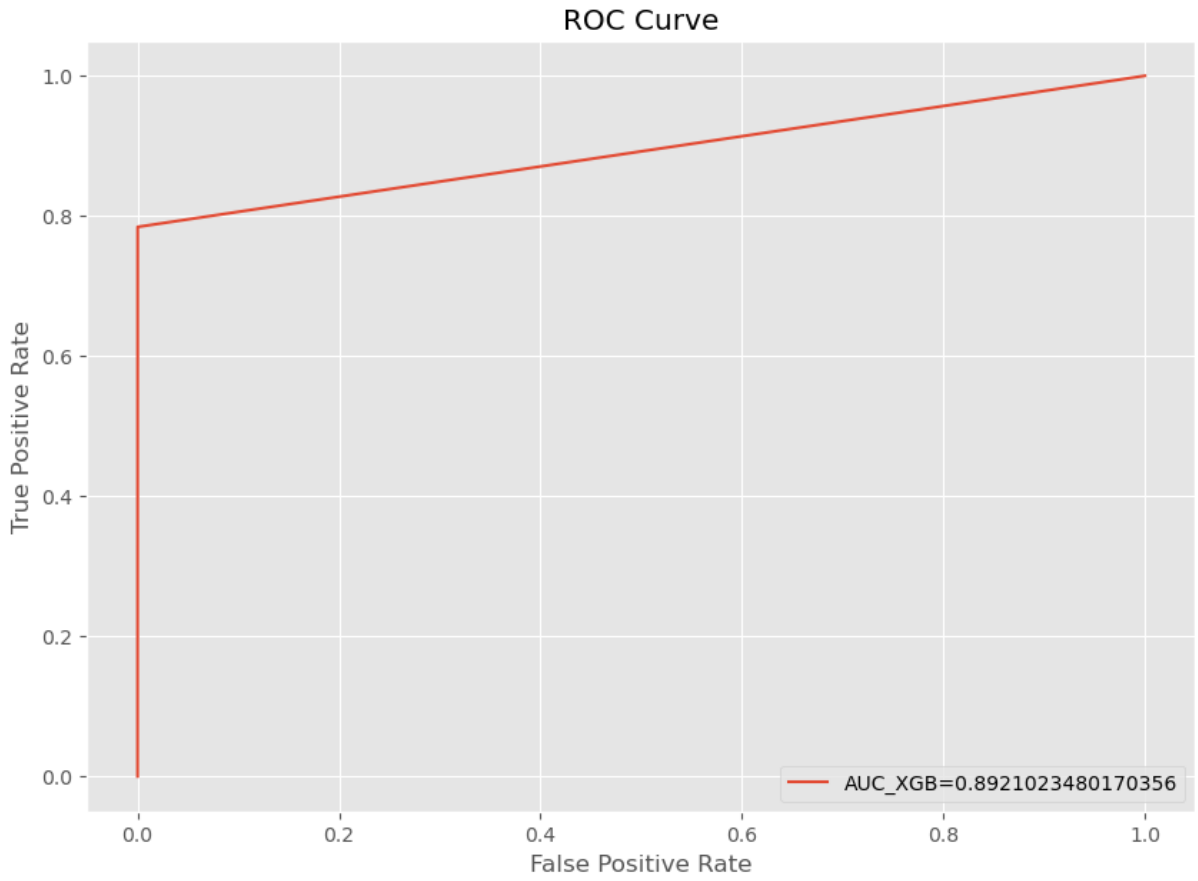
```
# Applying Xgboost Model
    model_xgb = XGBClassifier(max_depth =2)
    model_xgb.fit(x_train, y_train)
    y_pred_xgb = model_xgb.predict(x_test)
    print(f'The accuracy of the model is {round(model_xgb.score(x_test, y_test)*100,2)}%')
```

* To see which model is best suitable = Generate ROC - AUC Curve

```
# ROC - AUC for XGBoost Model
        fpr, trp, _ = metrics.roc_curve(y_test, y_pred_xgb)
        auc = metrics.roc_auc_score(y_test, y_pred_xgb)


# Create ROC Curve
        with plt.style.context(('ggplot')):
          plt.figure(figsize=(10,7))
          plt.title('ROC Curve')
          plt.plot(fpr, trp, label = 'AUC_XGB='+str(auc))
          plt.ylabel('True Positive Rate')
          plt.xlabel('False Positive Rate')
          plt.legend(loc=4)
          plt.show()
```





* Checking Performance for all models.

Xgbbost

```
fpr_xgb, tpr_xgb, _ = metrics.roc_curve(y_test, y_pred_xgb)
auc_xgb = metrics.roc_auc_score(y_test, y_pred_xgb)
```

Logistic - Regression

```
fpr_log, tpr_log, _ = metrics.roc_curve(y_test, y_pred_log)
auc_log = metrics.roc_auc_score(y_test, y_pred_log)
```

Random - Forest

```
fpr_rand, trp_rand, _ = metrics.roc_curve(y_test, y_pred_random)
auc_rand = metrics.roc_auc_score(y_test, y_pred_random)
```

Ga + Logistic

```
fpr_log_ga, trp_log_ga, _ = metrics.roc_curve(y_test, y_log_rfecv)
auc_log_ga = metrics.roc_auc_score(y_test, y_log_rfecv)
```

```
with plt.style.context(('ggplot')):
    plt.figure(figsize=(10,7))
    plt.title('ROC Curve')
    plt.plot(fpr_xgb, tpr_xgb, label ='AUC_XBG='+str(auc_xgb))
    plt.plot(fpr_log, tpr_log, label ='AUC_Logistic_Regression='+str(auc_log))
    plt.plot(fpr_rand, trp_rand, label ='AUC_Random_Forest='+str(auc_rand))
    plt.plot(fpr_log_ga, trp_log_ga, label ='AUC_ga_log='+str(auc_log_ga))
```

* As a result, XGB is the best.