Capstone Project

FindDefault (Prediction of Credit Card fraud)

Problem Statement:

A credit card is one of the most used financial products to make online purchases and payments. Though the Credit cards can be a convenient way to manage your finances, they can also be risky. Credit card fraud is the unauthorized use of someone else's credit card or credit card information to make purchases or withdraw cash.

It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

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.

We have to build a classification model to predict whether a transaction is fraudulent or not.

Steps Included:

1. **Exploratory Data Analysis:**
   - We started our project first by loading the .csv fie given to us. After that we checked the first five rows of the data using head() function and last five rows using tail() function.
     Here are the first five rows of the data.

```
dataset.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V2
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.12853
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.16717
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.32764
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.64737
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.20601

5 rows × 31 columns

- Then we checked that our data contains any null values or not by using **dataset.isnull().sum().**

```
dataset.isnull().sum()
```

```
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
```

- Then we converted date datatype i.e float64 to `datetime64`

8]:
```
dataset['Time'] = pd.to_datetime(dataset['Time'])
```

9]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  datetime64[ns]
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
```

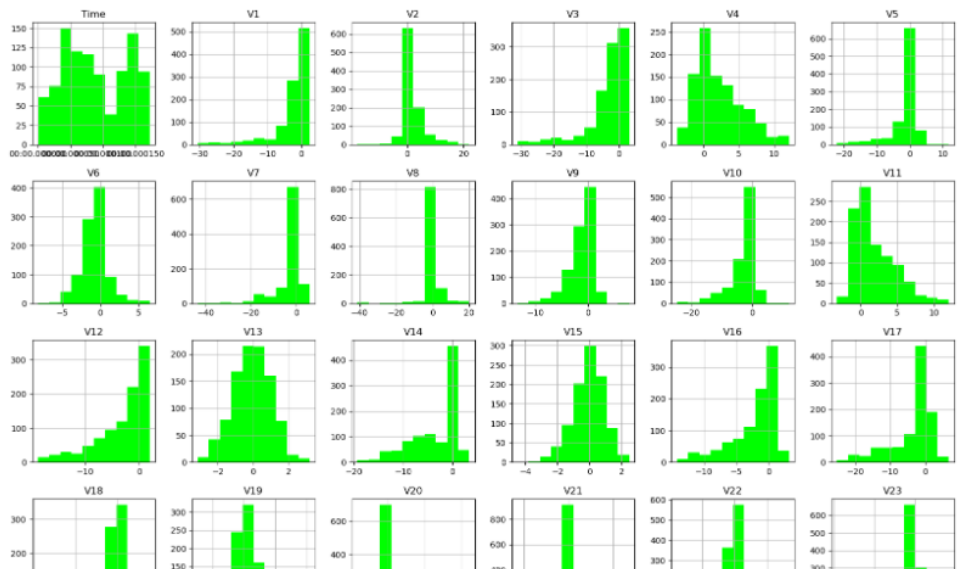
- Then we distributed the data into two parts which was Genuine cases and fraud cases. Genuine case are those whose class value is equals to 0 and fraudulent cases are those whose class value is equals to 1.

FRAUD CASES AND GENUINE CASES

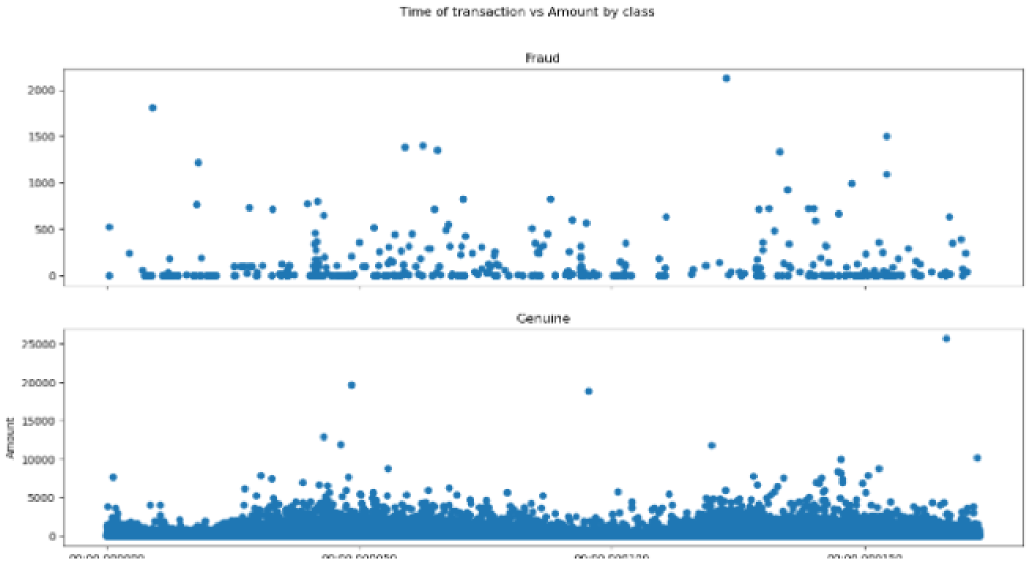
```
In [13]: fraud_cases=len(dataset[dataset['Class']==1])
         print(' Number of Fraud Cases:',fraud_cases)

          Number of Fraud Cases: 492

In [14]: non_fraud_cases=len(dataset[dataset['Class']==0])
         print('Number of Non Fraud Cases:',non_fraud_cases)

         Number of Non Fraud Cases: 284315

In [15]: fraud=dataset[dataset['Class']==1]

In [16]: fraud['Class']

Out[16]: 541       1
         623       1
         4920      1
         6108      1
         6329      1
                  ..
         279863    1
         280143    1
         280149    1
         281144    1
         281674    1
         Name: Class, Length: 492, dtype: int64
```

- Here are the few EDA performed

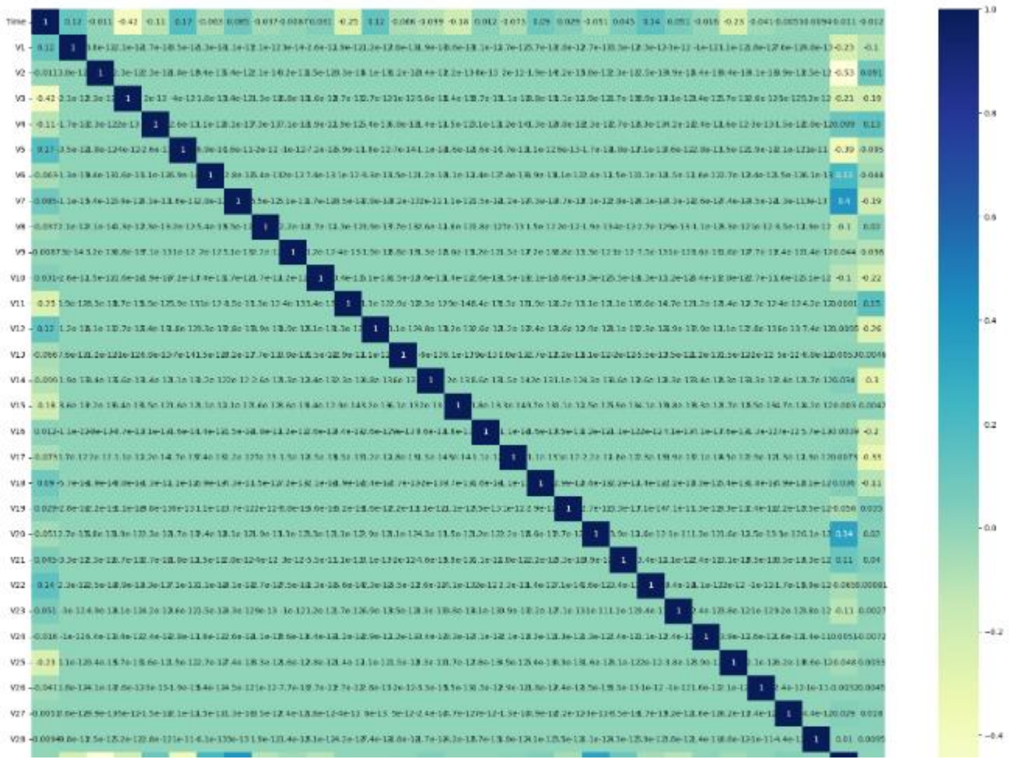
```
In [27]: new_dataset.hist(figsize=(20,20),color='lime')
         plt.show()
```



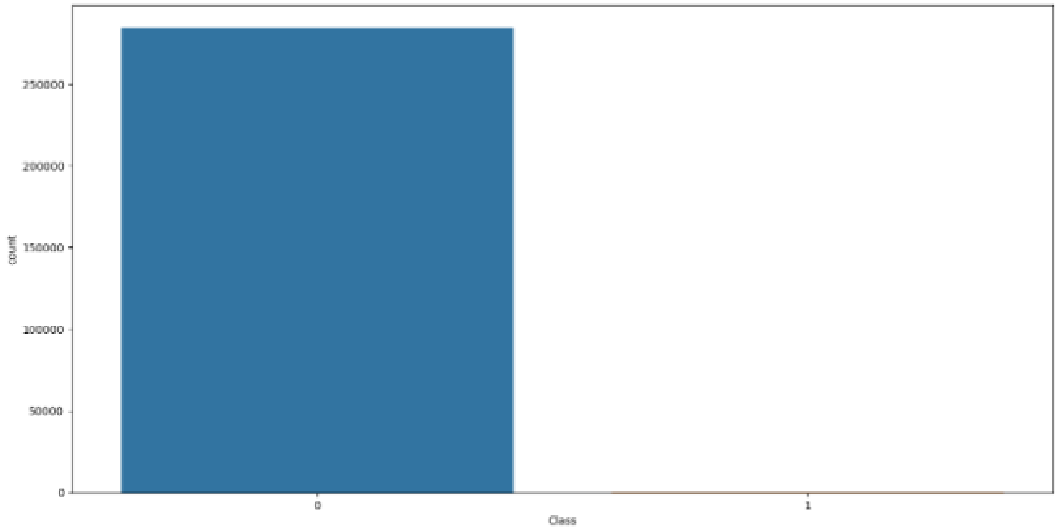
```
rcParams['figure.figsize'] = 16, 8
f,(ax1, ax2) = plt.subplots(2, 1, sharex=True)
f.suptitle('Time of transaction vs Amount by class')
ax1.scatter(fraud.Time, fraud.Amount)
ax1.set_title('Fraud')
ax2.scatter(genuine.Time, genuine.Amount)
ax2.set_title('Genuine')
plt.xlabel('Time (in Seconds)')
plt.ylabel('Amount')
plt.show()
```



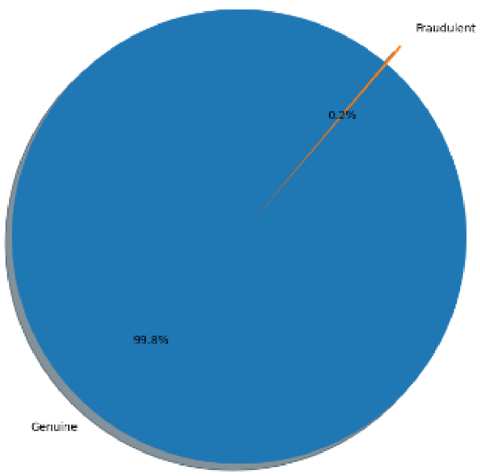
In [30]:
```
plt.figure(figsize=(24,18))
sns.heatmap(cor, cmap="YlGnBu", annot=True)
plt.show()
```



n [31]:
```
sns.countplot(x='Class', data = dataset)
plt.show()
```



In [34]:
```
labels = 'Genuine', 'Fraudulent'
sizes = [genuine_share,fraud_share]
explode = (0,0.1)
fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct = '%1.1f%%', shadow = True, startangle =50)
ax1.axis('equal')
plt.show()
```



2. Dealing with imbalance data using SMOTE Method.

SMOTE (Synthetic Minority Over-sampling Technique) is a method used in machine learning to address class imbalance, particularly in classification tasks where one class (the minority class) is significantly underrepresented compared to the other classes (the majority class or classes).

1. **Class Imbalance Problem**: In many real-world datasets, especially in areas like fraud detection, medical diagnosis, or anomaly detection, the number of instances belonging to one class (e.g., fraudulent

transactions) is much lower than the other classes (e.g., non-fraudulent transactions). This imbalance can lead to models that are biased towards the majority class and perform poorly in correctly identifying instances of the minority class.

2. **Purpose of SMOTE**: SMOTE is designed to alleviate this imbalance by oversampling the minority class. Instead of duplicating existing instances, which can lead to overfitting, SMOTE generates synthetic examples by interpolating between existing minority class instances. This is done in feature space rather than in data space.
3. **How SMOTE Works**:
   - **Identifying Minority Class Instances**: SMOTE first identifies the minority class instances in the dataset.
   - **Selecting Nearest Neighbors**: For each minority class instance, SMOTE finds its k nearest neighbors (typically using Euclidean distance) in the feature space.
   - **Generating Synthetic Instances**: Synthetic instances are then created along the line segments joining these k nearest neighbors. These synthetic instances are new, artificial samples that represent characteristics of the minority class but are not exact duplicates of existing instances.
   - **Balancing the Dataset**: By generating these synthetic instances, SMOTE increases the number of minority class samples, thereby balancing the class distribution.
4. **Implementation in Python**: SMOTE is commonly implemented in Python using libraries such as imbalanced-learn (imblearn), which provides an SMOTE class to apply the technique to your dataset. Here's a basic example of how you might use SMOTE with imbalanced-learn.

```
In [56]: from imblearn.over_sampling import SMOTE

In [57]: X_resampled, y_resampled = SMOTE().fit_resample(X, y)

In [58]: X_resampled.value_counts()

Out[58]: V1          V2          V3          V4          V5          V6          V7          V8          V9          V10         V11
         V12         V13         V14         V15         V16         V17         V18         V19         V20         V21         V22         V
         23          V24         V25         V26         V27         V28         Amount
         -13.192671  12.785971  -9.906650   3.320337  -4.801176   5.760059  -18.750889  -37.353443  -0.391540  -5.052502   4.406806
         -4.610756   -1.909488  -9.072711  -0.226074  -6.211557  -6.248145  -3.149247   0.051576  -3.493050  27.202839  -8.887017
         5.303607  -0.639435  0.263203  -0.108877  1.269566  0.939407  1.000000      3403
         -26.457745  16.497472  -30.177317  8.904157  -17.892600  -1.227904  -31.197329  -11.438920  -9.462573  -22.187089  4.419997
         -10.592305  -0.703796  -3.926207  -2.400246  -6.809890  -12.462315  -5.501051  -0.567940  2.812241  -8.755698   3.460893
         0.896538  0.254836  -0.738097  -0.966564  -7.263482  -1.324884  1.000000      1351
         -1.927453   1.827621   -7.019495   5.348303  -2.739188   -2.107219  -5.015848   1.205868   -4.382713  -8.337707   7.190306
         -9.424844   -0.223293  -12.875494  -0.071918  -6.299961  -12.719207  -3.740176  0.844060   2.172709   1.376938   -0.792017  -
         0.771414  -0.379574  0.718717  1.111151  1.277707  0.819081  512.250000      247
         -5.839192   7.151532   -12.816760  7.031115  -9.651272  -2.938427  -11.543207  4.843627   -3.494276  -13.320789  8.460244
         -17.003289  0.101557   -14.094452  0.747031  -12.661696  -18.912494  -6.626975  4.008921   0.055684   2.462056   1.054865
         0.530481  0.472670  -0.275998  0.282435  0.104886  0.254417  316.060000      244
         -10.850282  6.727466   -16.760583  8.425832  -10.252697  -4.192171  -14.077086  7.168288   -3.683242  -15.239962  8.030708
         -16.060306  0.270530   -14.952981  -0.241095  -11.866731  -15.486990  -5.748652  4.130031   -0.646818  2.541637   0.135535  -
         1.023967  0.406265  0.106593  -0.026232  -1.464630  -0.411682  78.000000      239
```

MODEL USED

1. Logistic Regression

At first we started with importing train_test_split from scikit learn and splitting our data into train and test.

```
In [62]: from sklearn.model_selection import train_test_split

In [63]: X_train,X_test,y_train,y_test=train_test_split(X_resampled,y_resampled,test_size=0.3,random_state=42)

In [67]: print(len(X_train))
         print(len(X_test))

         398041
         170589
```

Using First Model (Logistic Regression)

```
In [68]: model=LogisticRegression()

In [70]: model.fit(X_train,y_train)

Out[70]: LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

The Accuracy, Precision, Recall Score and F1 Score achieved by logistic regression are as follows:

```
In [72]: from sklearn.metrics import roc_auc_score, precision_score, recall_score, accuracy_score, f1_score

In [75]: y_pred=model.predict(X_test)
         print(y_pred)

         [0 1 0 ... 1 1 1]

In [115]: accuracy = accuracy_score(y_test,y_pred)
          precision = precision_score(y_test, y_pred)
          recall = recall_score(y_test, y_pred)
          f1 = f1_score(y_test, y_pred)

In [117]: print('Accuracy of the model', accuracy)
          print('precision of the model', precision)
          print('Recall score of the model', recall)
          print('f1_score of the model', f1)

          Accuracy of the model 0.9583032903645604
          precision of the model 0.9781809745912748
          Recall score of the model 0.9376638576779026
          f1_score of the model 0.9574939793595114
```

- Why we used Logistic Regression

Logistic regression is a powerful and flexible method for binary classification tasks that offers simplicity, interpretability, and effectiveness in a variety of practical applications. Its ability to model probabilities and handle both linear and non-linear relationships makes it a valuable tool in the machine learning toolbox.

2. Decision Tree Model

By using the trained data we applied decision tree model to our tool.By importing DecisionTreeClassifier from sklearn.tree and the Accuracy, Precision, Recall Score and F1 Score achieved by logistic regression are as follows:

Using Second Model(Decision Tree)

```
In [80]: from sklearn.tree import DecisionTreeClassifier

In [81]: dt = DecisionTreeClassifier(random_state=42)
         dt.fit(X_train, y_train)

Out[81]: DecisionTreeClassifier(random_state=42)
         In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
         On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [82]: y_pred_dt=dt.predict(X_test)
         print(y_pred_dt)

         [1 1 0 ... 1 1 1]

In [111]: accuracy = accuracy_score(y_test,y_pred_dt)
          precision = precision_score(y_test, y_pred_dt)
          recall = recall_score(y_test, y_pred_dt)
          f1 = f1_score(y_test, y_pred_dt)

In [113]: print('Accuracy of the model', accuracy)
          print('precision of the model', precision)
          print('Recall score of the model', recall)
          print('f1_score of the model', f1)

          Accuracy of the model 0.9979541471020992
          precision of the model 0.9970675172035096
          Recall score of the model 0.9988529962546816
          f1_score of the model 0.99795945811627
```

3. Random Forest Classification

By using the trained data we applied Random forest classification to our tool. By importing RandomForestClassifierfrom sklearn.ensemble and the Accuracy, Precision, Recall Score and F1 Score achieved by logistic regression are as follows:

Using Third Model (Random Forest Classification)

In [85]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [86]:
```
classifier = RandomForestClassifier(n_estimators=5, random_state = 42)
```

In [87]:
```
classifier.fit(X_train, y_train)
```

Out[87]:
```
RandomForestClassifier(n_estimators=5, random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [89]:
```
y_pred_rf = classifier.predict(X_test)
(y_pred_rf)
```

Out[89]:
```
array([1, 1, 0, ..., 1, 1, 1], dtype=int64)
```

In [118]:
```
accuracy = accuracy_score(y_test,y_pred_rf)
precision = precision_score(y_test, y_pred_rf)
recall = recall_score(y_test, y_pred_rf)
f1 = f1_score(y_test, y_pred_rf)
```

In [119]:
```
print('Accuracy of the model', accuracy)
print('precision of the model', precision)
print('Recall score of the model', recall)
print('f1_score of the model', f1)
```

```
Accuracy of the model 0.9997831044205664
precision of the model 0.9996255864836721
Recall score of the model 0.9999414794007491
f1_score of the model 0.9997835079897254
```

4. Using XG Boost

In [92]:
```
import xgboost as xgb
```

In [94]:
```
xgb_model = xgb.XGBClassifier(random_state=42)
```

In [95]:
```
xgb_model.fit(X_train, y_train)
```

Out[95]:
```
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=42, ...)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [96]:
```
y_pred_xgb=xgb_model.predict(X_test)
y_pred_xgb
```

Out[96]:
```
array([1, 1, 0, ..., 1, 1, 1])
```

In [120]:
```
accuracy = accuracy_score(y_test,y_pred_xgb)
precision = precision_score(y_test, y_pred_xgb)
recall = recall_score(y_test, y_pred_xgb)
f1 = f1_score(y_test, y_pred_xgb)
```

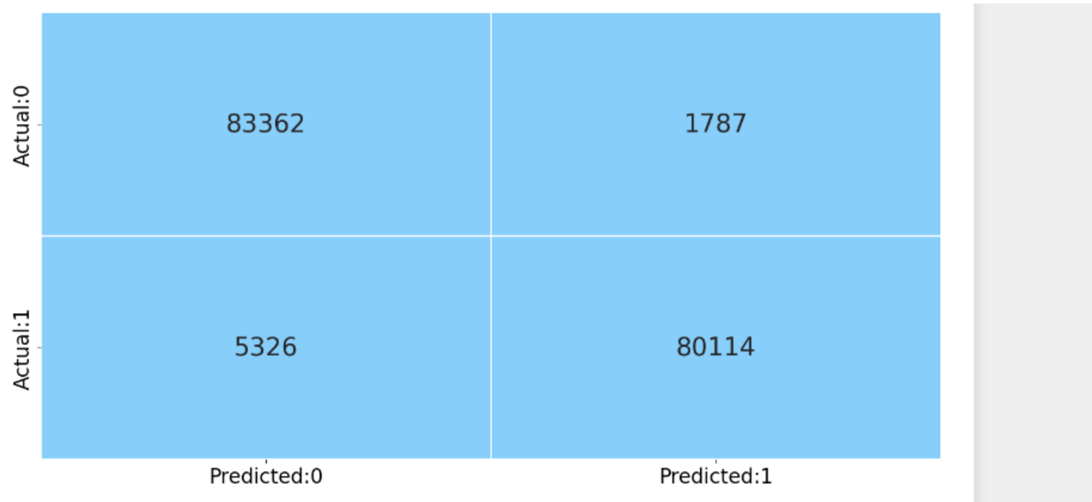
In [121]:
```
print('Accuracy of the model', accuracy)
print('precision of the model', precision)
print('Recall score of the model', recall)
print('f1_score of the model', f1)
```

```
Accuracy of the model 0.9998182766766908
precision of the model 0.9996373038808485
Recall score of the model 1.0
f1_score of the model 0.9998186190473404
```

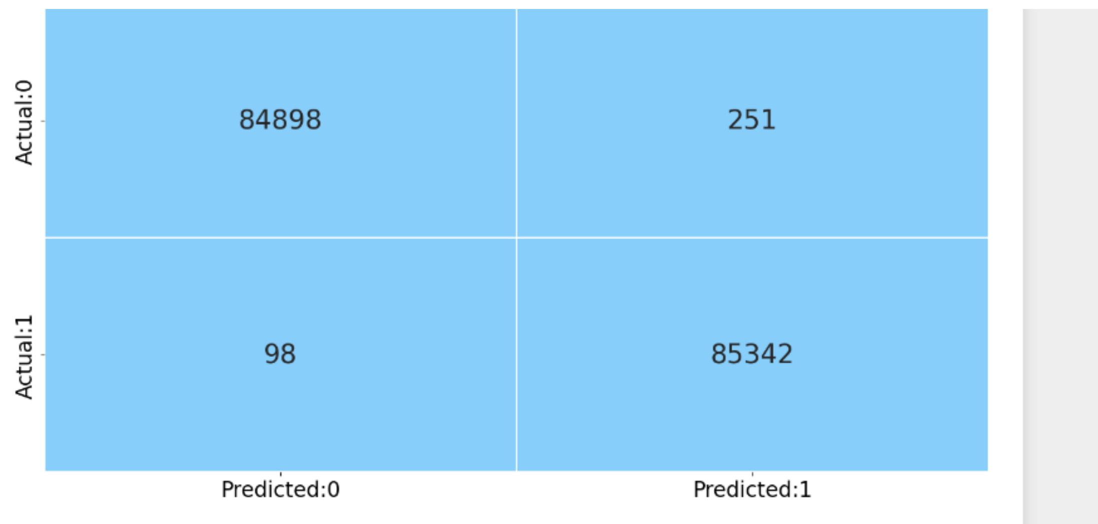
Confusion Matrix:

The confusion matrix is crucial for assessing the performance of a classification model, especially in scenarios where class distribution is imbalanced. It provides deeper insights into model strengths and weaknesses, helping to fine-tune the model or adjust decision thresholds based on specific business or application requirements.

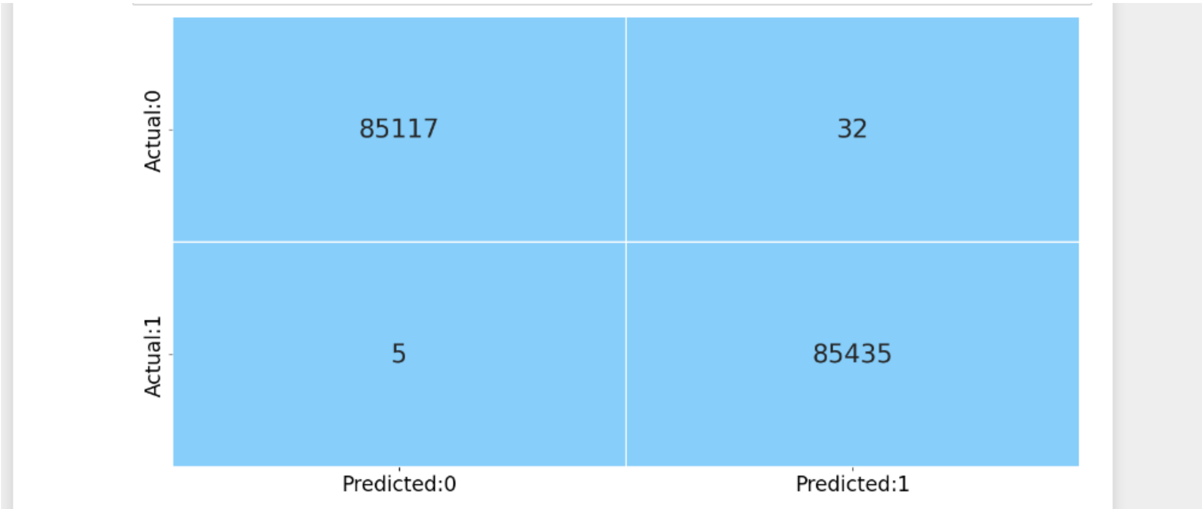
Confusion Matrix for Logistic Regression



Confusion Matrix for Decision Tree Classifier



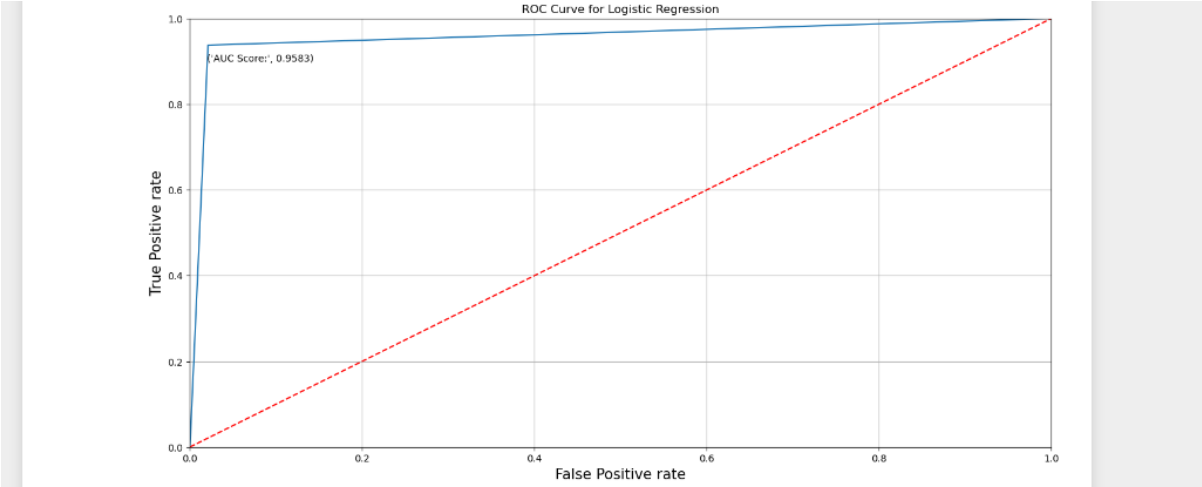
Confusion Matrix for Random Forest



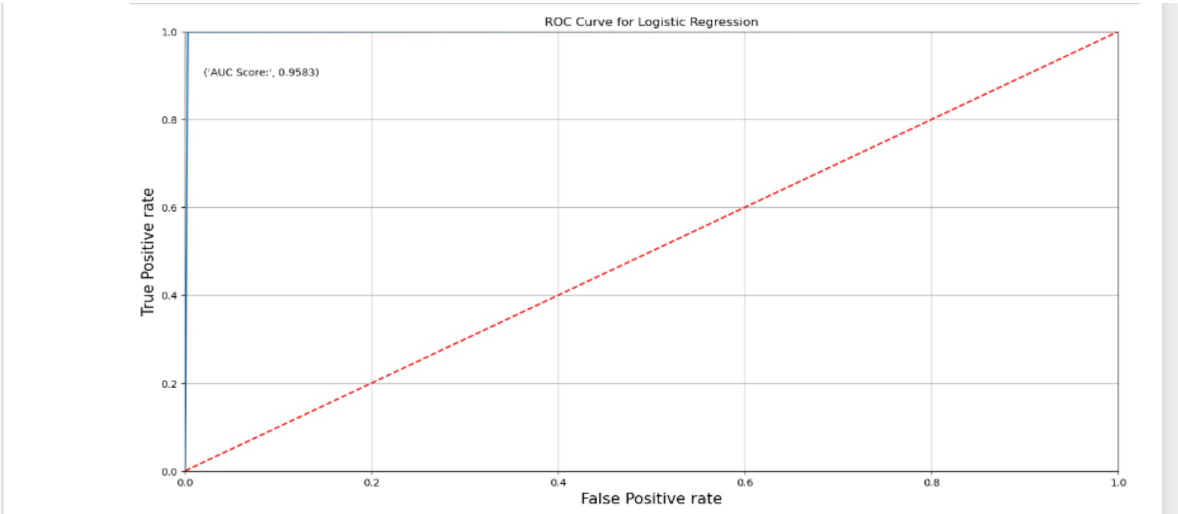
ROC Curve and AUC Curve:

ROC curves and AUC are valuable tools in evaluating and comparing binary classification models, providing insights into their discriminatory power and helping to make informed decisions about model selection and optimization.

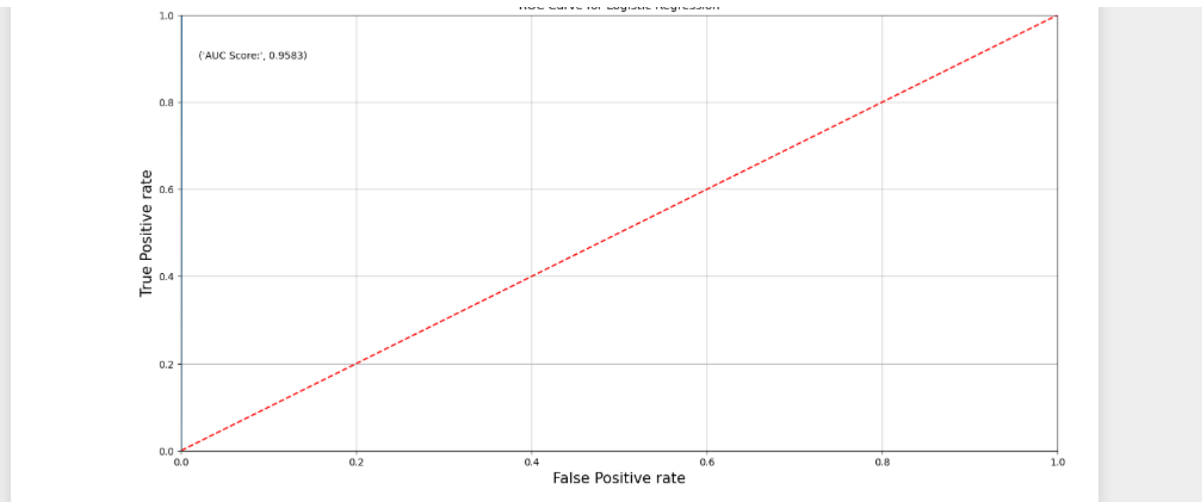
ROC Curve for Logistic Regression:



ROC Curve for Decision Tree Classifier:



ROC Curve for Decision Tree Classifier:



Here is the Score Card we achieved by applying all models.

```
In [136]: score_card
```

Out[136]:

	model_name	Accuracy Score	Precision Score	Recall Score	AUC Score	f1-score
0	Logistic Regression	0.958303	0.978181	0.937664	0.958339	0.957494
1	Decision Tree	0.997954	0.997068	0.998853	0.997953	0.997959
2	Random Forest	0.999783	0.999626	0.999941	0.999783	0.999784
3	XGBoost Classifier	0.999818	0.999637	1.000000	0.999818	0.999819
4	Logistic Regression	0.958303	0.978181	0.937664	0.958339	0.957494

Tuning our model using GridSearchCV on XGB Model

GridSearchCV is a technique in machine learning used for hyperparameter tuning, which is the process of finding the optimal hyperparameters for a model to achieve the best possible performance. It is available in Python through the GridSearchCV class provided by the sklearn.model_selection module (from scikit-learn).

How GridSearchCV Works:

1. **Hyperparameters**: Hyperparameters are parameters that are not directly learned by the model during training but are set before training. Examples include the number of trees in a random forest or the learning rate in a gradient boosting machine.
2. **Cross-Validation**: GridSearchCV performs an exhaustive search over a specified grid of hyperparameters. For each combination of hyperparameters specified in the grid, GridSearchCV trains the model using cross-validation.
3. **Grid Search**: The "grid" in GridSearchCV refers to a set of hyperparameter values that you want to try. It can be defined as a dictionary where keys are the hyperparameter names, and values are lists of hyperparameter settings to try.
4. **Cross-Validation**: GridSearchCV uses cross-validation to evaluate each combination of hyperparameters. By default, it uses 5-fold cross-validation, but this can be customized using the cv parameter.
5. **Scoring**: After fitting the models, GridSearchCV scores each model based on a scoring function (e.g., accuracy, precision, recall, etc.) provided by the user.
6. **Best Model Selection**: Once all combinations of hyperparameters have been evaluated, GridSearchCV selects the combination that has the highest cross-validation score.

Tuning the Model using GridSearchCV on XGBModel

```
[148]: from sklearn.model_selection import GridSearchCV , RandomizedSearchCV
```

```
[138]: xgb_model = xgb.XGBClassifier(random_state=42)
```

```
[140]: param_grid = {
           'n_estimators': [50, 100, 500],
           'criterion' : ['Auto', 'sqrt', 'log2'],

           'max_features': ['gini', 'antropy']
       }
```

```
[142]: gridsearch = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=5)
       gridsearch.fit(X_train, y_train)
```

```
[142]: GridSearchCV(cv=5,
                    estimator=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_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,
```

```
In [143]: print("Best Parameters :", gridsearch.best_params_)
          print("Best Accuracy :", gridsearch.best_score_)

          Best Parameters : {'criterion': 'Auto', 'max_features': 'gini', 'n_estimators': 500}
          Best Accuracy : 0.9998316755872192
```

```
In [144]: best_model = gridsearch.best_estimator_
          y_pred_best = best_model.predict(X_test)
```

```
In [146]: from sklearn.metrics import classification_report
```

```
In [147]: print("classification_report")
          print(classification_report(y_test, y_pred_best))

          classification_report
                        precision    recall  f1-score   support

                     0       1.00      1.00      1.00     85149
                     1       1.00      1.00      1.00     85440

              accuracy                           1.00    170589
             macro avg       1.00      1.00      1.00    170589
          weighted avg       1.00      1.00      1.00    170589
```

--After Tuning the data the last step is the deployment of model.

--We Used joblib for deploying the model.

Joblib-

joblib is a Python library designed to provide lightweight utilities for saving and loading Python objects. It is particularly useful for efficiently saving and loading large NumPy arrays and machine learning models, among other types of Python objects. Here are some key features and aspects of joblib:

ChatGPT

joblib is a Python library designed to provide lightweight utilities for saving and loading Python objects. It is particularly useful for efficiently saving and loading large NumPy arrays and machine learning models, among other types of Python objects. Here are some key features and aspects of joblib:

Key Features of joblib:

1. **Serialization**: joblib provides functions for serializing (saving) Python objects into files and deserializing (loading) them back into memory. This is particularly useful for saving trained machine learning models, NumPy arrays, and other complex data structures.
2. **Efficiency**: It is optimized for handling large data, such as big NumPy arrays, by using efficient binary serialization. This makes joblib well-suited for scenarios where performance and memory efficiency are critical.
3. **No Dependencies**: joblib has minimal dependencies and is part of the standard scientific Python ecosystem (often included with packages like scikit-learn). This ensures compatibility and ease of use in various Python environments.
4. **Integration with scikit-learn**: joblib is commonly used in conjunction with scikit-learn for saving and loading machine learning models trained using scikit-learn's algorithms. Many scikit-learn functions and classes utilize joblib under the hood for model persistence.

Overall, joblib is a versatile and efficient library for serializing Python objects, particularly suited for machine learning applications involving large datasets and complex models. Its simplicity and integration with popular libraries like scikit-learn make it a valuable tool in the Python ecosystem.

```
In [164]: import joblib

In [171]: joblib.dump(model, "CreditCardFraudDetection Model.pkl")
Out[171]: ['CreditCardFraudDetection Model.pkl']

In [173]: model1 = joblib.load("CreditCardFraudDetection Model.pkl")

In [178]: X.head()
Out[178]:
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V20	V21	V22	V23	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	0.251412	-0.018307	0.277838	-0.110474	0.0
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.069083	-0.225775	-0.638672	0.101288	-0.3
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.524980	0.247998	0.771679	0.909412	-0.6
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.208038	-0.108300	0.005274	-0.190321	-1.1
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	0.408542	-0.009431	0.798278	-0.137458	0.1

5 rows × 29 columns

```
In [185]: pred1 = model1.predict([[1.166616382,0.502120088,-0.067300314,2.261569239,0.428804195,0.089473517,0.24114658,0.138081705,-0.9891

In [186]: if pred1 ==0:
              print("Legit Transaction")
          else:
              print("fraud transaction")

Legit Transaction

In [191]: pred3 = model1.predict([[-2.303349568,1.75924746,-0.359744743,2.330243051,-0.821628328,-0.075787571,0.562319782,-0.399146578,

In [193]: if pred3 ==0:
              print("Legit Transaction")
          else:
              print("fraud transaction")

fraud transaction

In [ ]:
```

We applied if-else statement to find that the transaction is Legit or Fraud

```
if pred1 ==0:
    print("Legit Transaction")
else:
   print("fraud transaction")
```

Conclusion-

Over the data provided was imbalanced that we balanced using SMOTE technique, which resampled the data. After resampling and performing of EDA we proceeded to splitting the data into train and test. We applied 4 models that are Logistic regression, Random Forest, Decision Tree Classifier and XG Boost Model.

The highest accuracy we get is from XG Boost, so xgboost is the best model for this.