Credit Card Fraud Detection

Project Report

Abstract:

The purpose of this project is to detect the fraudulent transactions made by credit cards by the use of machine learning techniques, to stop fraudsters from the unauthorized usage of customers' accounts. The increase of credit card fraud is growing rapidly worldwide, which is the reason actions should be taken to stop fraudsters.

Putting a limit for those actions would have a positive impact on the customers as their money would be recovered and retrieved back into their accounts and they won't be charged for items or services that were not purchased by them which is the main goal of the project. Detection of the fraudulent transactions will be made by using three machine learning **techniques Logistic Regression, Decision Tree,** and **Random Forest Classifier** those models will be used on a credit card transaction dataset.

Dataset:

The dataset was provided from **UpGrad Capstone Project** it contains data of transactions that were made in 2013 by credit card users in Europe, in two days only. The dataset consists of 31 attributes, 284,807 rows. 28 attributes are numeric variables that due to confidentiality and privacy of the customers have been transformed using **PCA transformation**, the three remaining attributes are "Time" which contains the elapsed seconds between the first and other transactions of each attribute, "Amount" is the amount of each transaction, and the final attribute "Class" which contains binary variables where "1" is a case of fraudulent transaction, and "0" is not as case of fraudulent transaction.

The dataset used is highly imbalanced, with the majority of transactions being normal and a small fraction being fraudulent. The notebook includes data exploration, preprocessing, feature engineering, model training, evaluation, and a discussion on deployment.

Dataset: creditcard.csv

Data Exploratory and Analysis

Analyzed the dataset to understand feature relationships, identified missing values, and visualized distributions. Feature relationships were further explored through correlation matrices and feature importance scores from models. This phase helped in detecting data imbalances and selecting appropriate methods to address them.

```python
df=pd.read_csv(r"C:\Users\AK\Downloads\creditcard.csv")
```

```python
df.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  float64
 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
...
```

Distribution of Dataset using Histplot-
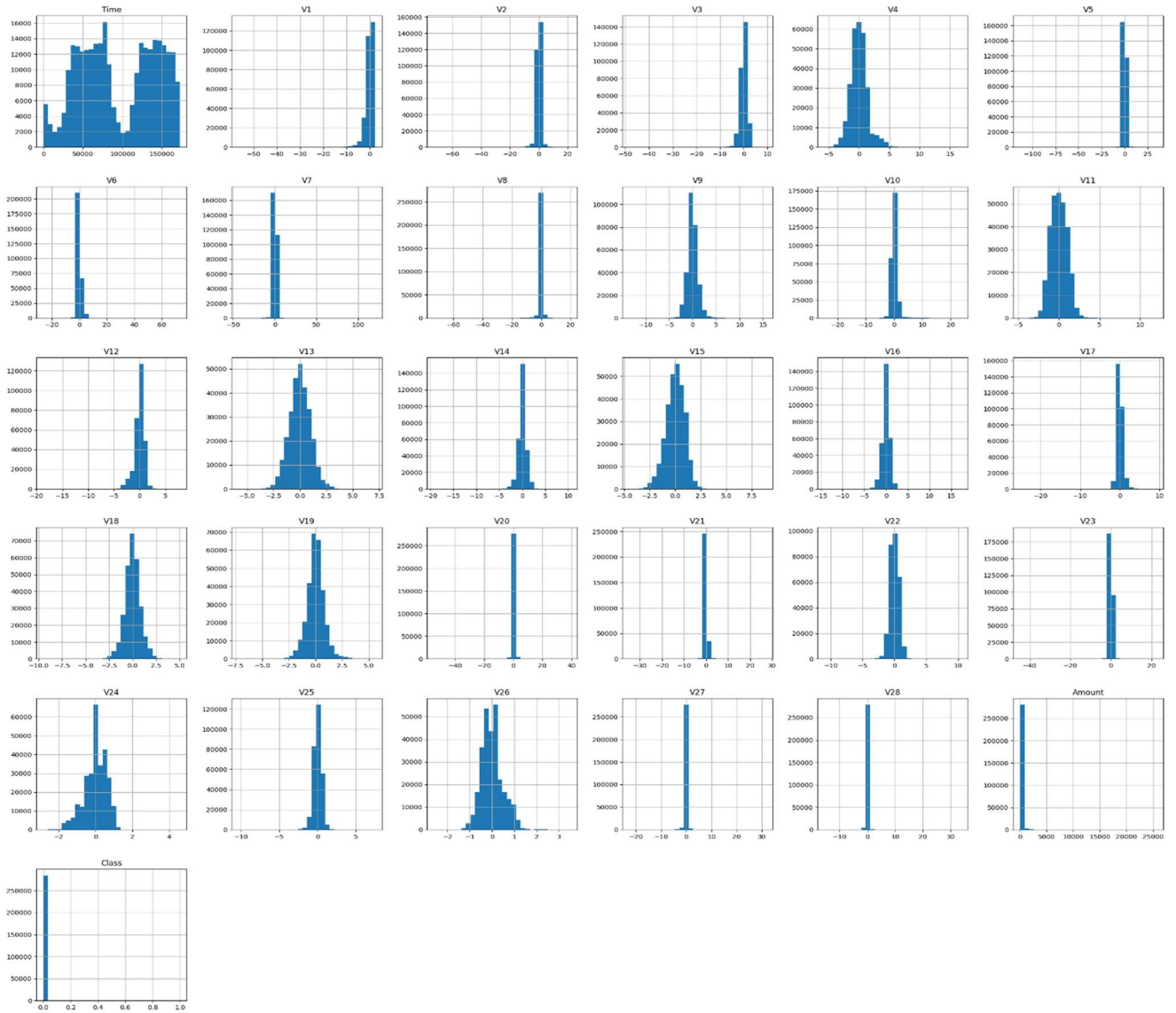
Correlation between attributes -

The correlations between all of the attributes within the dataset are presented in the figure below.
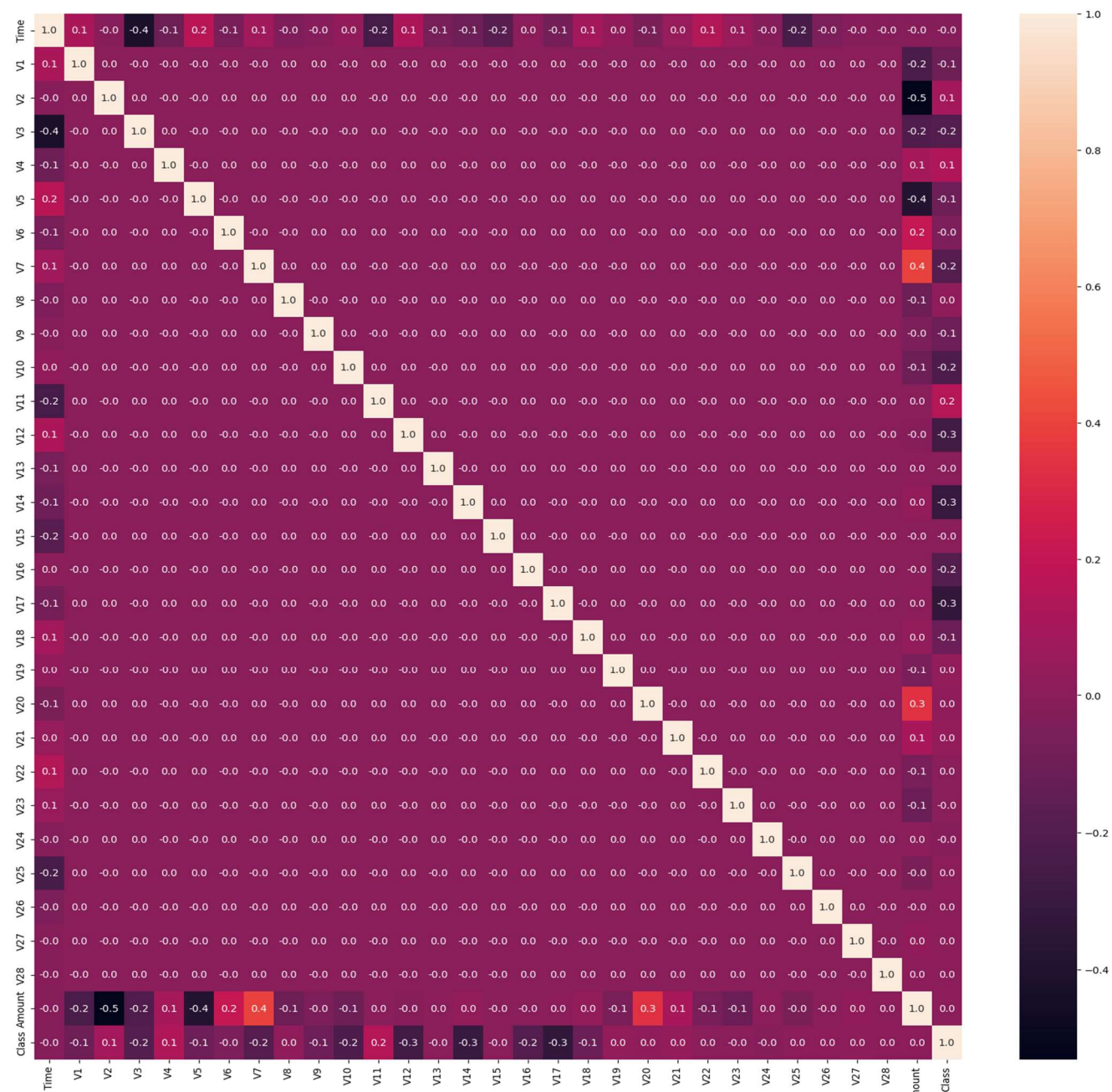
Data Preprocessing

After choosing the most suited dataset the preparation phase begins, the preparation of the dataset includes selecting the wanted attributes or variables, cleaning it by excluding Null rows, deleting duplicated variables, treating outlier if necessary, in addition to transforming data types to the wanted type, data merging can be performed as well where two or more attributes get merged. All those alterations lead to the wanted result which is to make the data ready to be modeled.

```
## Checking Missing Value
df.isna().sum()
```

```
Checking Duplicate Value
df.duplicated().any()
```

```
# Feature Scaling the Amount and Time coloumn
from sklearn.preprocessing import RobustScaler
df['Amount']=RobustScaler().fit_transform(df['Amount'].to_numpy().reshape(-1,1))
df['Time']=RobustScaler().fit_transform(df['Time'].to_numpy().reshape(-1,1))
```

Countplot of Transaction Class Distribution

Figure shows the distribution of the class after preprocessing, the red bar which contains 2,83,253 variables represents the non-fraudulent transactions, and the blue bar with 473 variables represents the fraudulent transactions.

Transaction Class Distribution
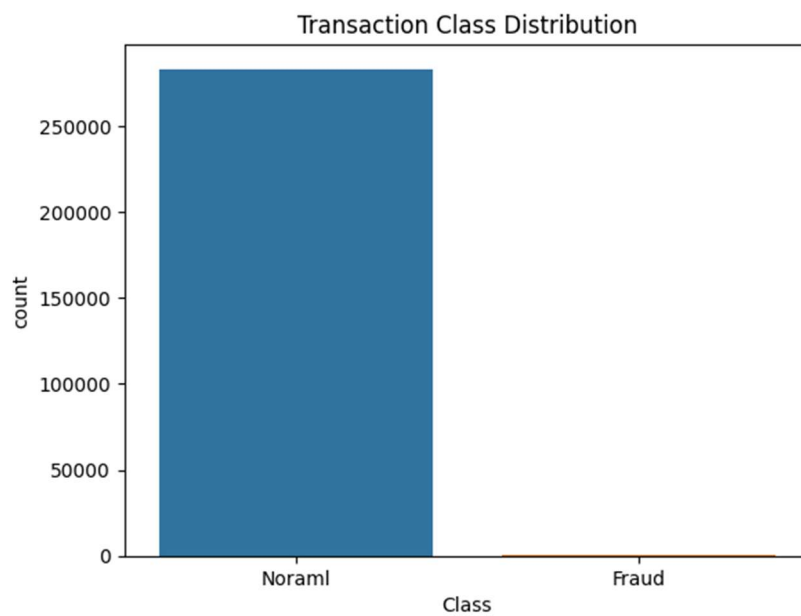
Analyzing Amount of information from the transaction data:-

Time of transaction vs Amount by class
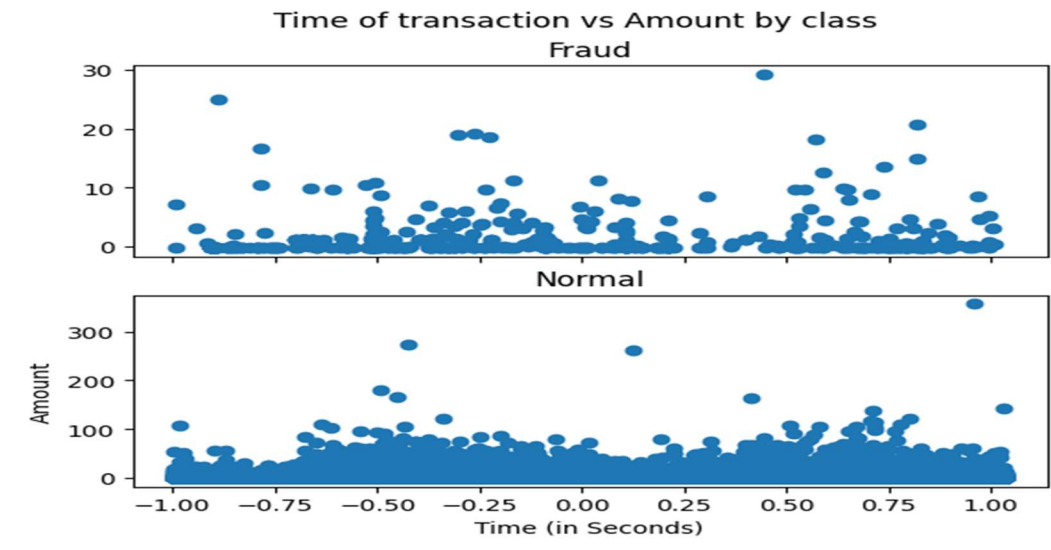
Modeling

After preprocessing the data, Three machine learning models were created in the modeling phase, Logistic Regression, Decision Tree, and Random Forest Classifier and applied on Orignial dataset. A comparison of the results will be presented later in the paper to know which technique is most suited in the credit card fraudulent transactions detection. The dataset is sectioned into a ratio of 80:20, the training set will be the 80% and remaining set will be the testing set which is the 20%.

- **Storing Features in matrix X and storing Target(response) column to y**

```
X=df.drop('Class',axis=1)
y=df['Class']
```

- **Splitting Dataset into Test and Train Dataset**

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=.2,random
_state=42)
```

```
------Logistic-Regression----
c:\New folder\lib\site-packages\sklearn\linear_model\_logistic.py:818: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
 Accuracy Score: 0.9991541254009093
Precision: 0.8888888888888888
F1-Score: 0.6666666666666667
Recall: 0.5333333333333333

------Decision-Tree----
Accuracy Score: 0.999048391076023
Precision: 0.6956521739130435
F1-Score: 0.7032967032967032
Recall: 0.7111111111111111

------Random Forest----
Accuracy Score: 0.9995594403129736
Precision: 0.9710144927536232
F1-Score: 0.8427672955974843
Recall: 0.7444444444444445
```

Observation: We got different score for each Machine Model as the the dataset is highly imbalance. So to balance data we opt **Under-sampling** and **Over-sampling** method and apply these models again and observe the result.

Under-sampling:-

Under-sampling involves reducing the number of samples from the majority class to make the class distribution more balanced.

The dataset is downsized by randomly selecting a subset of the majority class that is equal in size to the minority class, or some other desired ratio. This helps prevent the model from being biased towards the majority class.

```
#Undersampling the noraml transaction to 473 by deleting the data
normal_smple=normal.sample(n=473)
```

- Since, here we have 473 fraudulent data so we are randomly selecting a subset of Normal transaction and reducing it to 473 so that balancing the both data.

Now, we are again applying the Three Machines Model and observing the Scores.

```
[31]
...
        ------Logistic-Regression----
         Accuracy Score: 0.9421052631578948
         Precision: 0.9789473684210527
         F1-Score: 0.9441624365482234
         Recall: 0.9117647058823529

        ------Decision-Tree----
         Accuracy Score: 0.9157894736842105
         Precision: 0.9134615384615384
         F1-Score: 0.9223300970873786
         Recall: 0.9313725490196079

        ------Random Forest----
         Accuracy Score: 0.9421052631578948
         Precision: 0.989247311827957
         F1-Score: 0.9435897435897437
         Recall: 0.9019607843137255
```

<u>Over-sampling:-</u>

Oversampling involves increasing the number of samples in the minority class to make the class distribution more balanced.

The dataset is expanded by generating new synthetic samples for the minority class or by simply duplicating existing samples. **SMOTE (Synthetic Minority Over-sampling Technique)** is a popular method for creating synthetic samples.

```python
from imblearn.over_sampling import SMOTE
X_res,y_res= SMOTE().fit_resample(X,y)
```

```
▷ ∨      y_res.value_counts()
[38]

...      0    283253
         1    283253
         Name: Class, dtype: int64
```

Now, we are again applying the Three Machines Model and observing the Scores.

```
...
         ------Logistic-Regression----
          Accuracy Score: 0.9468323595346949
          Precision: 0.9732700009313588
          F1-Score: 0.9454960008685896
          Recall: 0.9192631819701261

         ------Decision-Tree----
          Accuracy Score: 0.9979788529769995
          Precision: 0.9971196740314026
          F1-Score: 0.9979872909287465
          Recall: 0.9988564190080754

         ------Random Forest----
          Accuracy Score: 0.999929392243738
          Precision: 0.9998592713775573
          F1-Score: 0.999929630737294
          Recall: 1.0
```

Observation: Here we find out that **Random Forest Classifier** outperform among the three models and gives best Accuracy, Precision, F1 and Recall score with **99.99%.**

So, we will go with over-sampling method and choose **Random Forest Classifier** and Train the data

Evaluation and Deployment

Out of all the models the model that scored the best is **Random Forest Classifier** as its accuracy is 99.99% and Recall 1%, the second best is Decision Tree and the model that scored the lowest accuracy out of all models is Logistic Regression with a score of 94.68%.

Saving Model:-

Using pickle for saving and loading the model in a pickle file **'CreditCard_Model.pkl'**

```python
import pickle

# Train the model
RF=RandomForestClassifier()
RF.fit(X_res,y_res)


# Save the model using pickle
with open('CreditCard_Model.pkl', 'wb') as f:
    pickle.dump(RF, f)
```

Flask Application

Use Flask to create a user interface for the credit card fraud detection project. The flask application allows the user to enter credit card transaction data, and the data is trained Random Forest Classifier model. Once transaction features are entered then we get the prediction on whether the transaction is legitimate or fraudulent.

```python
# app.py
import numpy as np
from flask import Flask, request, jsonify, render_template
import pickle
app = Flask(__name__)
with open('CreditCard_Model.pkl', 'rb') as f:
    model = pickle.load(f)


@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict',methods=['POST'])
def predict():
    '''
    For rendering results on HTML GUI
    '''
    int_features = [float(x) for x in request.form.values()]
    final_features = np.array(int_features).reshape(1, -1)
    prediction = model.predict(final_features)


    output = prediction[0]
    if output == 0:
        res_val = "** Normal Transaction **"
    else:
        res_val = "** Fraudulent Transaction **"

    return render_template('index.html', prediction_text='This is a :: {}'.format(res_val))



if __name__ == "__main__":
    app.run(debug=False)
```

Conclusion:

In conclusion, the main objective of this project was to find the most suited model in credit card fraud detection in terms of the machine learning techniques chosen for the project, and it was met by building the four models and finding the accuracies of them all, the best model in terms of accuracies is Random Forest Classifier which scored 99.99% I believe that using the model will help in decreasing the amount of credit card fraud and increase the customers satisfaction as it will provide them with better experience in addition to feeling secure.