MSDS 696 Data Science Practicum II:

Final Report: Classification Predictive Models for Credit Card Fraud Detection

Babawale Olatunji

Regis University

Authors Note

This Final Report was prepared for MSDS 696 taught by Prof. Aiman Gannous

Classification Predictive Models for Credit Card Fraud Detection

1. Introduction

Credit Card Fraud Detection is a challenging task. This is due to several features that should be examined for accurate and timely detection of Credit Card Fraudulent Transactions. It is imperative that credit card companies including banking, finance, retail, and e-commerce companies can identify fraudulent credit card transactions so that customers are not charged for unauthorized transactions. According to legal dictionary, "Credit Card Fraud is the unauthorized use of an individual's credit card information to make purchases, or to remove funds from the cardholder's account".

The purpose of this data science practicum project is to develop a machine learning classification predictive model that can best be leveraged in credit card fraud detection. The second motivation behind this project is to create a classification machine learning model that can be utilized by credit card companies to significantly prevent the loss of billions of dollars to credit card fraudulent transactions.

2. Data

2.1 Presentation of the data

This project used Kaggle data set located at https://www.kaggle.com/mlg-ulb/creditcardfraud to achieve its objectives. It contains transactions that took place in two days, where out of 284,807 transactions we have 492 fraudulent transactions. The dataset contains only numerical input variables due to Principal Component Analysis (PCA) transformation and confidentiality issues. The dataset has 31 Features including 28 PCA transformed Attributes and two attributes that are not transformed by PCA. The last feature is the target attribute of interest in this practicum project. This last feature is the Transaction Class and it can be Fraudulent or Non-Fraudulent.

2.2 Data Preparation

There are no missing values identified as NaNs/Null and zeros (0.0) in the dataset used for this project. Missing values if not dealt with would result in bias resulting from the differences between missing and complete data. Figure 1 depicts the summary count of missing values per column for this dataset. As highlighted in Figure 1, meaningful column names were created for Time, Amount and Class columns.

```
## Display the number of missing values per column
df.isnull().sum()
```

```
Time_in_Seconds         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
Transaction_Amount      0
Transaction_Class_id    0
dtype: int64
```

Figure 1: Summary count of missing values per column

2.3 The Imbalanced Transaction Class Distribution

It is imperative to highlight the significant contrast within the Transaction Class. As expected, most of the transactions are Non-Fraudulent while only very few transactions are Fraudulent Transactions. Both Figure 2 and Figure 3 depict this important contrast within the Transaction Class.

Credit card fraud detection is a binary classification predictive problem with an imbalanced class distribution.

```
In [28]: # Frequency Counts of Attribute Class
         df['Transaction_Class_id'].value_counts()

Out[28]: 0    284315
         1       492
         Name: Transaction_Class_id, dtype: int64
```

Figure 2: Frequency Counts of Credit Card Fraud Detection Transaction Class

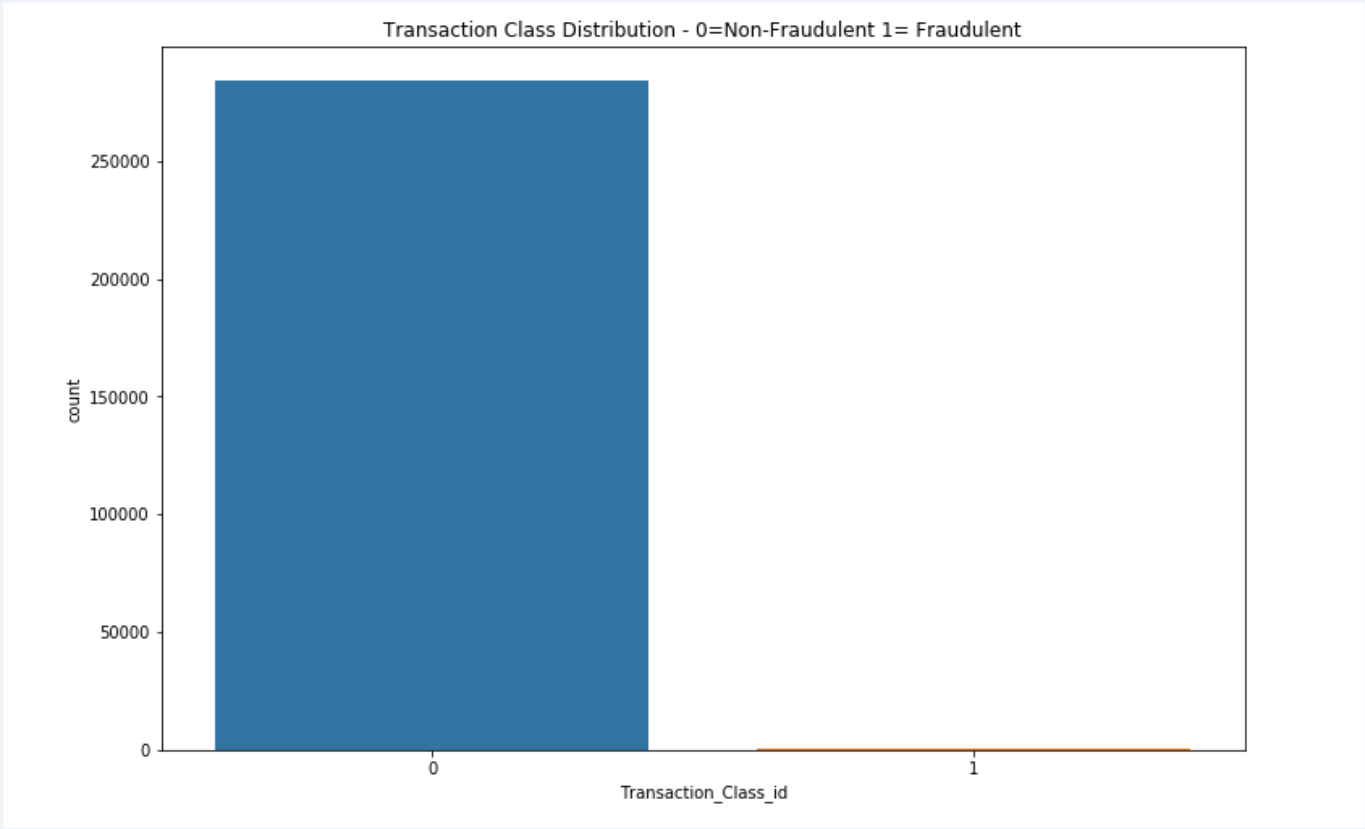


Figure 3: Credit Card Fraud Detection Transaction Class Count Plot

2.4 Exploratory Data Analysis (EDA)

Summary Statistics and Histograms of the Attributes were used to highlight the overall view of the credit card fraud detection dataset. Table 1 depicts the Summary Statistics of the dataset features.

Table 1: Summary Statistics of Credit Card Fraud Detection Attributes

```
# Descriptive or Summary statistics of numeric columns
credit_card_fraud_detection_df.describe()
```

	Time_in_Seconds	V1	V2	V3	V4	V5	V6	V7	V8	V
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+0
mean	94813.859575	3.919560e-15	5.688174e-16	-8.769071e-15	2.782312e-15	-1.552563e-15	2.010663e-15	-1.694249e-15	-1.927028e-16	-3.137024e-1
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+0
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+0
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-0
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-0
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-0
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+0

V21	V22	V23	V24	V25	V26	V27	V28	Transaction_Amount	Transaction_Class_id
48070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	284807.000000	284807.000000
537294e-16	7.959909e-16	5.367590e-16	4.458112e-15	1.453003e-15	1.699104e-15	-3.660161e-16	-1.206049e-16	88.349619	0.001727
345240e-01	7.257016e-01	6.244603e-01	6.056471e-01	5.212781e-01	4.822270e-01	4.036325e-01	3.300833e-01	250.120109	0.041527
183038e+01	-1.093314e+01	-4.480774e+01	-2.836627e+00	-1.029540e+01	-2.604551e+00	-2.256568e+01	-1.543008e+01	0.000000	0.000000
283949e-01	-5.423504e-01	-1.618463e-01	-3.545861e-01	-3.171451e-01	-3.269839e-01	-7.083953e-02	-5.295979e-02	5.600000	0.000000
945017e-02	6.781943e-03	-1.119293e-02	4.097606e-02	1.659350e-02	-5.213911e-02	1.342146e-03	1.124383e-02	22.000000	0.000000
863772e-01	5.285536e-01	1.476421e-01	4.395266e-01	3.507156e-01	2.409522e-01	9.104512e-02	7.827995e-02	77.165000	0.000000
'20284e+01	1.050309e+01	2.252841e+01	4.584549e+00	7.519589e+00	3.517346e+00	3.161220e+01	3.384781e+01	25691.160000	1.000000

```
The data is from two days transactions as highlighted by maximum Time value of 172792 seconds.
This is approximately equal to 60x60x24x2 = 172800 seconds = 2 days. Furthermore, the minimum Transaction Amount is 0.0
and maximum amount is 25691.16. The mean Transaction Amount is 88.34.
```

The data is from two days credit card transactions as highlighted by maximum time in seconds of 172792 in Table 1 above. This is approximately equal to two days in seconds. Furthermore, the minimum transaction amount is 0.0 and the maximum transaction amount 88.34.

Tables 2 and 3 further provide the summary statistics of fraudulent and non-fraudulent transactions.

Table 2: Fraudulent Transactions Summary Statistics

```
# Descriptive or Summary statistics of numeric columns
Fraudulent_Transactions.describe()
```

	Time_in_Seconds	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V:
count	492.000000	492.000000	492.000000	492.000000	492.000000	492.000000	492.000000	492.000000	492.000000	492.000000	...	492.000000	492.0000
mean	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	...	0.713588	0.0140
std	47835.365138	6.783687	4.291216	7.110937	2.873318	5.372468	1.858124	7.206773	6.797831	2.500896	...	3.869304	1.4946
min	406.000000	-30.552380	-8.402154	-31.103685	-1.313275	-22.105532	-6.406267	-43.557242	-41.044261	-13.434066	...	-22.797604	-8.8870
25%	41241.500000	-6.036063	1.188226	-8.643489	2.373050	-4.792835	-2.501511	-7.965295	-0.195336	-3.872383	...	0.041787	-0.5337
50%	75568.500000	-2.342497	2.717869	-5.075257	4.177147	-1.522962	-1.424616	-3.034402	0.621508	-2.208768	...	0.592146	0.0484
75%	128483.000000	-0.419200	4.971257	-2.276185	6.348729	0.214562	-0.413216	-0.945954	1.764879	-0.787850	...	1.244611	0.6174
max	170348.000000	2.132386	22.057729	2.250210	12.114672	11.095089	6.474115	5.802537	20.007208	3.353525	...	27.202839	8.3619

```
# Descriptive or Summary statistics of numeric columns
Fraudulent_Transactions.describe()
```

V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Transaction_Amount	Transaction_Class_id
000	492.000000	...	492.000000	492.000000	492.000000	492.000000	492.000000	492.000000	492.000000	492.000000	492.000000	492.0
636	-2.581123	...	0.713588	0.014049	-0.040308	-0.105130	0.041449	0.051648	0.170575	0.075667	122.211321	1.0
831	2.500896	...	3.869304	1.494602	1.579642	0.515577	0.797205	0.471679	1.376766	0.547291	256.683288	0.0
261	-13.434066	...	-22.797604	-8.887017	-19.254328	-2.028024	-4.781606	-1.152671	-7.263482	-1.869290	0.000000	1.0
336	-3.872383	...	0.041787	-0.533764	-0.342175	-0.436809	-0.314348	-0.259416	-0.020025	-0.108868	1.000000	1.0
508	-2.208768	...	0.592146	0.048434	-0.073135	-0.060795	0.088371	0.004321	0.394926	0.146344	9.250000	1.0
879	-0.787850	...	1.244611	0.617474	0.308378	0.285328	0.456515	0.396733	0.826029	0.381152	105.890000	1.0
208	3.353525	...	27.202839	8.361985	5.466230	1.091435	2.208209	2.745261	3.052358	1.779364	2125.870000	1.0

From the above Fraudulent Transaction Class summary statistics, the minimum Transaction Amount is 0.0, maximum Transaction Amount is 2125.87 and the mean Transaction Amount is 122.21. Hence, it can be inferred that an average of 122 is lost to Fraudulent Transaction.

Table 3: Non-Fraudulent Transactions Summary Statistics

```
# Descriptive or Summary statistics of numeric columns
Non_Fraudulent_Transactions.describe()
```

	Time_in_Seconds	V1	V2	V3	V4	V5	V6	V7	V8	
count	284315.000000	284315.000000	284315.000000	284315.000000	284315.000000	284315.000000	284315.000000	284315.000000	284315.000000	284315.000
mean	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.000987	0.004
std	47484.015786	1.929814	1.636146	1.459429	1.399333	1.356952	1.329913	1.178812	1.161283	1.089
min	0.000000	-56.407510	-72.715728	-48.325589	-5.683171	-113.743307	-26.160506	-31.764946	-73.216718	-6.290
25%	54230.000000	-0.917544	-0.599473	-0.884541	-0.850077	-0.689398	-0.766847	-0.551442	-0.208633	-0.640
50%	84711.000000	0.020023	0.064070	0.182158	-0.022405	-0.053457	-0.273123	0.041138	0.022041	-0.049
75%	139333.000000	1.316218	0.800446	1.028372	0.737624	0.612181	0.399619	0.571019	0.326200	0.598
max	172792.000000	2.454930	18.902453	9.382558	16.875344	34.801666	73.301626	120.589494	18.709255	15.594

V21	V22	V23	V24	V25	V26	V27	V28	Transaction_Amount	Transaction_Class_id
15.000000	284315.000000	284315.000000	284315.000000	284315.000000	284315.000000	284315.000000	284315.000000	284315.000000	284315.0
-0.001235	-0.000024	0.000070	0.000182	-0.000072	-0.000089	-0.000295	-0.000131	88.291022	0.0
0.716743	0.723668	0.621541	0.605776	0.520673	0.482241	0.399847	0.329570	250.105092	0.0
34.830382	-10.933144	-44.807735	-2.836627	-10.295397	-2.604551	-22.565679	-15.430084	0.000000	0.0
-0.228509	-0.542403	-0.161702	-0.354425	-0.317145	-0.327074	-0.070852	-0.052950	5.650000	0.0
-0.029821	0.006736	-0.011147	0.041082	0.016417	-0.052227	0.001230	0.011199	22.000000	0.0
0.185626	0.528407	0.147522	0.439869	0.350594	0.240671	0.090573	0.077962	77.050000	0.0
22.614889	10.503090	22.528412	4.584549	7.519589	3.517346	31.612198	33.847808	25691.160000	0.0

From the above Non-Fraudulent Transaction Class summary statistics, the minimum Transaction Amount is 0.0, maximum Transaction Amount is 25691 and the mean Transaction Amount is 88.29. Hence, the average amount for Non-Transaction is 88.29.

From Table 2 above for fraudulent transaction class summary statistics, the minimum transaction amount is 0.0, the maximum transaction amount is 2125.87 and the average transaction amount is 122.21. Hence, it can be inferred that an average of 122 is lost to fraudulent transaction. However, from Table 3 above for non-fraudulent transaction class

summary statistics, the minimum transaction amount is 0.0, the maximum transaction amount is 25691 and the mean value of non-fraudulent is 88.89. Hence, the average amount for non-fraudulent transaction is 88.29. Also, based on the difference between the minimum and maximum transaction amount, it can be concluded that non-fraudulent transaction class has a wide range. Furthermore, figures 4 and 5 showcase the top 10 distribution of fraudulent and non-fraudulent transactions.

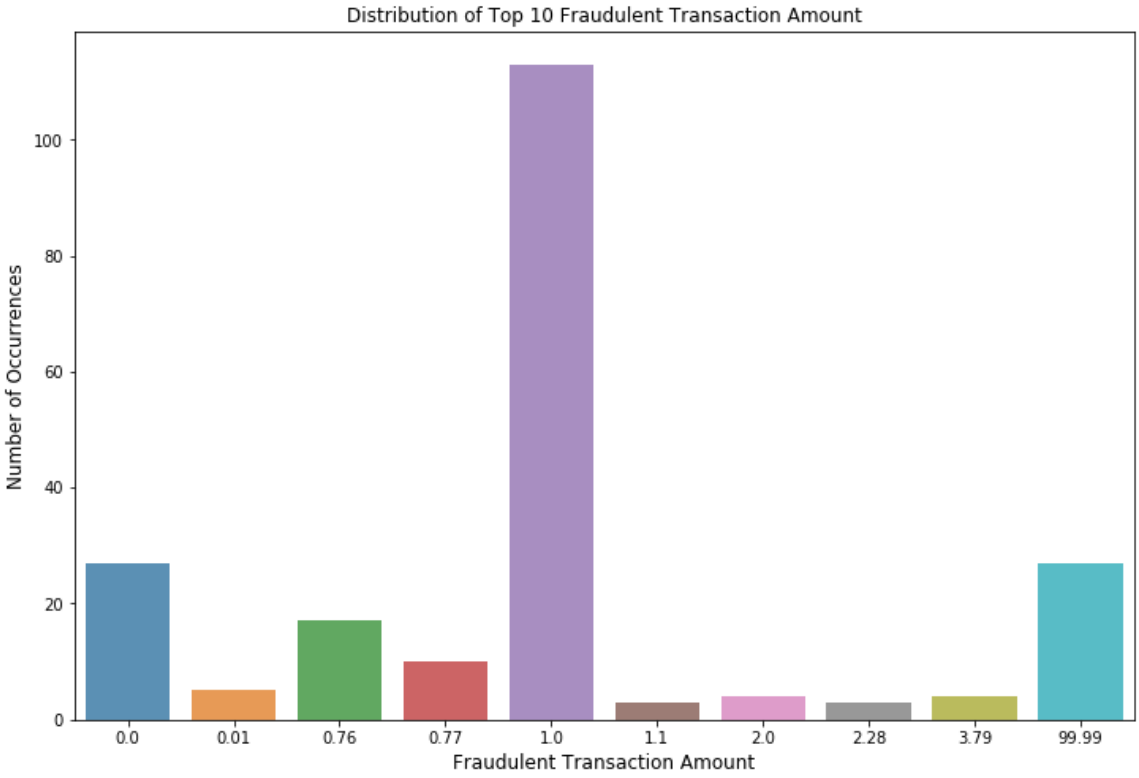


Figure 4: Distribution of Top 10 Fraudulent Transaction Amount

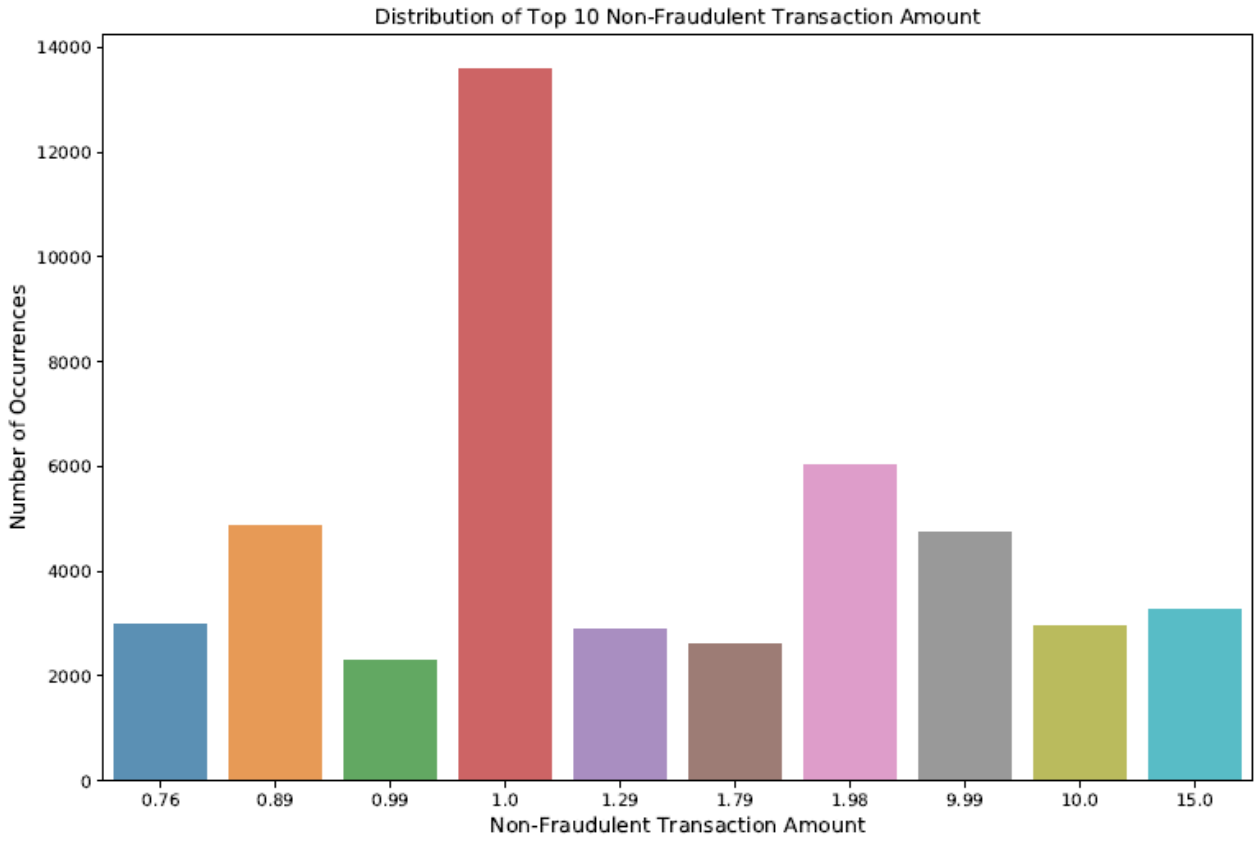


Figure 5: Distribution of Top 10 Non-Fraudulent Transaction Amount

Histograms of the dataset variables were created to visualize the distribution of the features data. Figure 6 showcases the histogram for each of the credit card fraud detection attribute. Each histogram below highlights if the distribution of data for each of the variable is symmetric, left-skewed or right-skewed.

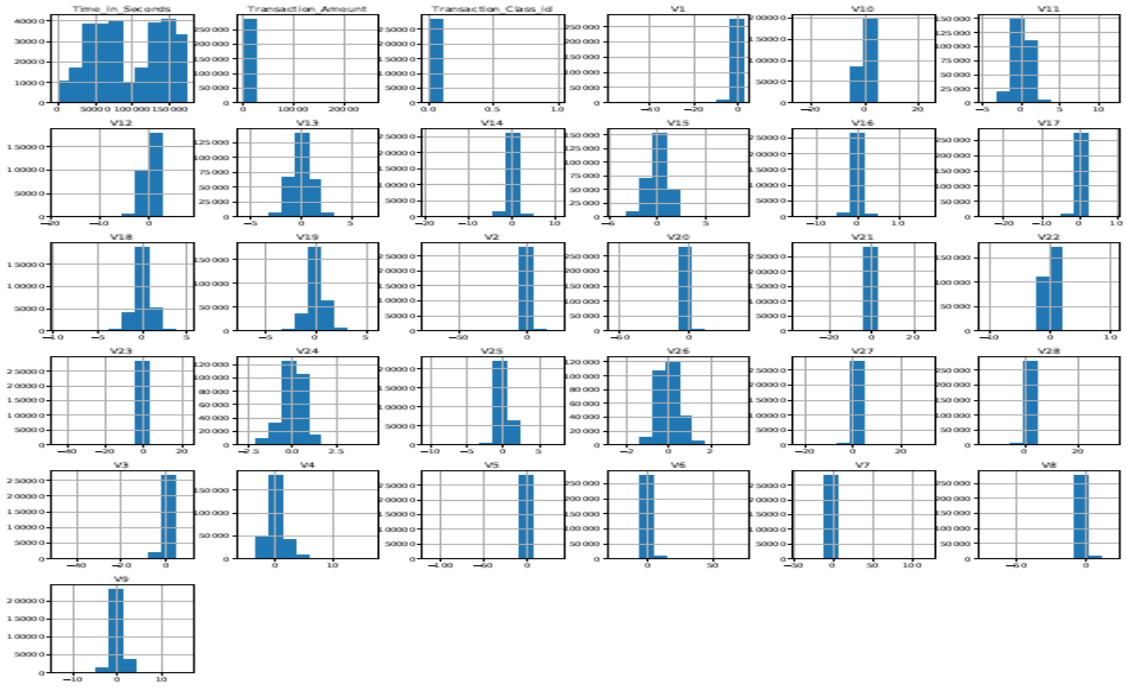


Figure 6: Histograms of Credit Card Fraud Detection Features

3. Feature Selection

The performance of a classification machine learning model is greatly affected by feature selection. It is worth noting that model performance can be negatively affected by irrelevant or partially pertinent features. The process of manually or automatically selecting those features or input that contribute the most to the target or prediction variable is called Feature Selection. It minimizes overfitting, fosters prediction accuracy and minimizes training time.

To determine the important features for the transaction class, univariate selection process was leveraged in this project to identify relevant features (independent variables) that contribute the most to the classification of transactions as either non-fraudulent or fraudulent.

```
# Display the order of importance of the features
print(featureScores.nlargest(30,'Score'))
```

	Features	Score
17	V17	33979.168593
14	V14	28695.547788
12	V12	20749.822361
10	V10	14057.979985
16	V16	11443.349428
3	V3	11014.508305
7	V7	10349.605408
11	V11	6999.355047
4	V4	5163.832114
18	V18	3584.380605
1	V1	2955.668946
9	V9	2746.600273
5	V5	2592.357929
2	V2	2393.401678
6	V6	543.510578
21	V21	465.916251
19	V19	344.990997
20	V20	114.999731
8	V8	112.548287
27	V27	88.045296
0	Time_in_Seconds	43.252998
28	V28	25.901405
24	V24	14.850932
29	Transaction_Amount	9.033345
13	V13	5.947672
26	V26	5.653653
15	V15	5.080193
25	V25	3.116062
23	V23	2.053476
22	V22	0.184706

Figure 7: Univariate Feature Selection Result

The above figure 7 result identifies features V17, V14, V12, V10 and V16 are the top five features that facilitate the classification of a transaction as either a non-fraudulent or fraudulent transaction. Feature Selection is further performed for most of the classification models developed in this project.

It is worth noting that some classifiers do have feature importance attribute. The following features importance results were produced by Decision Tree, Random Forest and Extreme Gradient Boosting Classifiers:

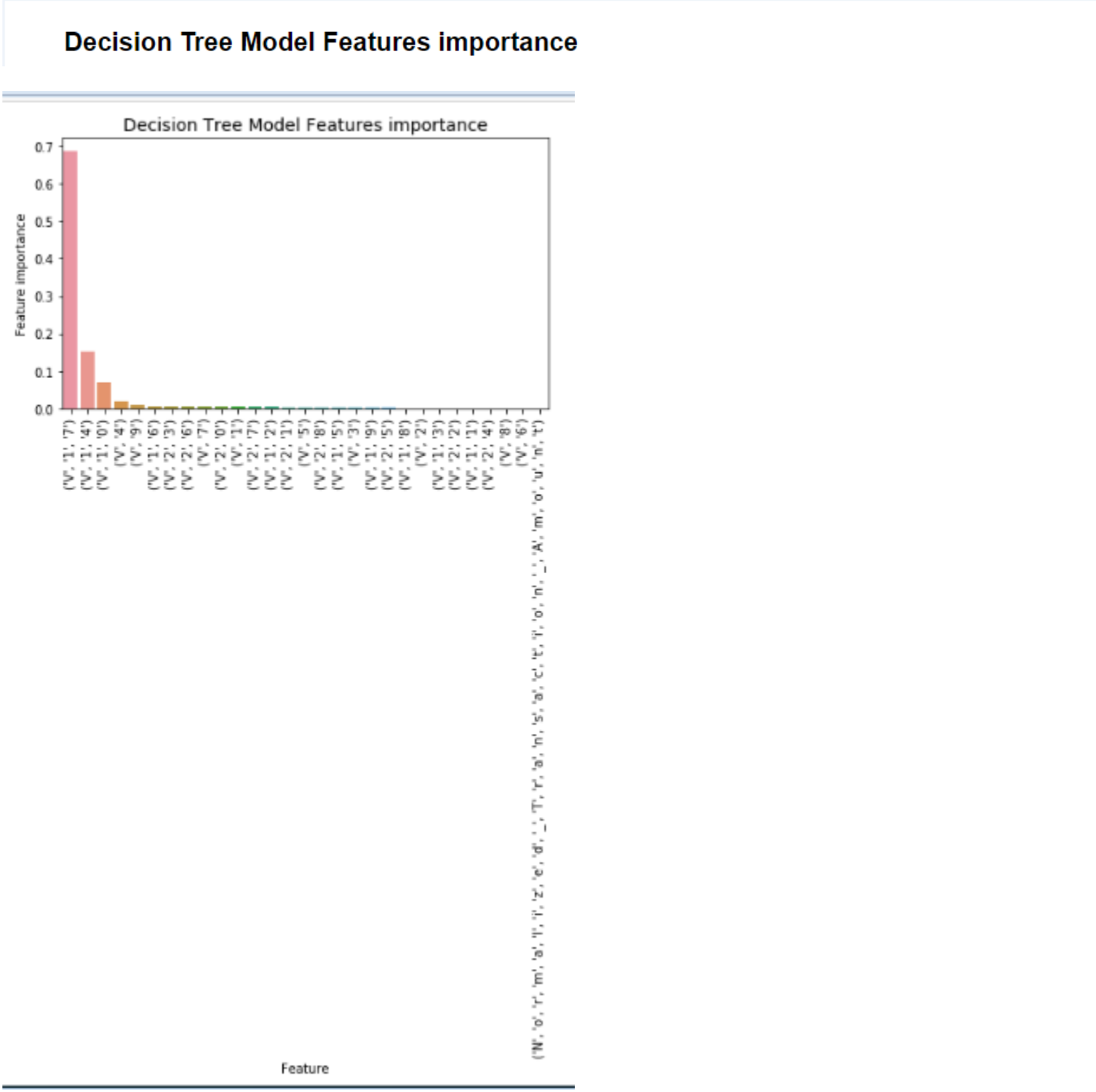


Figure 8: Decision Tree Features Importance

The above figure 8 result identifies features V17, V14, V10, V4 and V9 as the top five features that facilitated the classification of transactions as either non-fraudulent or fraudulent by the decision tree model developed for this project.

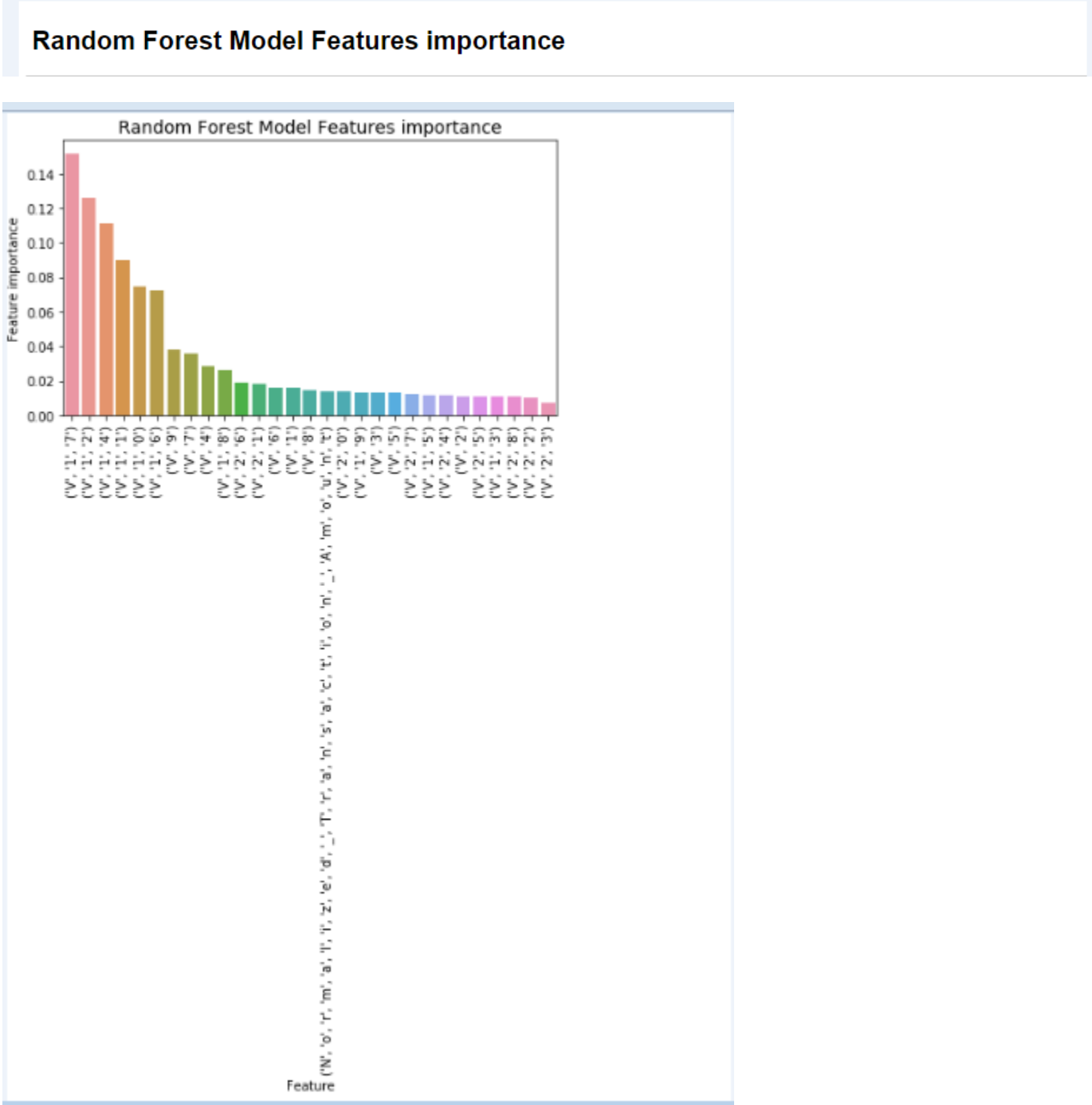


Figure 9: Random Forest Features Importance

The above result identifies features V17, V12, V14, V10 and V11 as the top five features that fostered the classification of transactions as either non-fraudulent or fraudulent by random forest model constructed for this credit card fraud detection project.

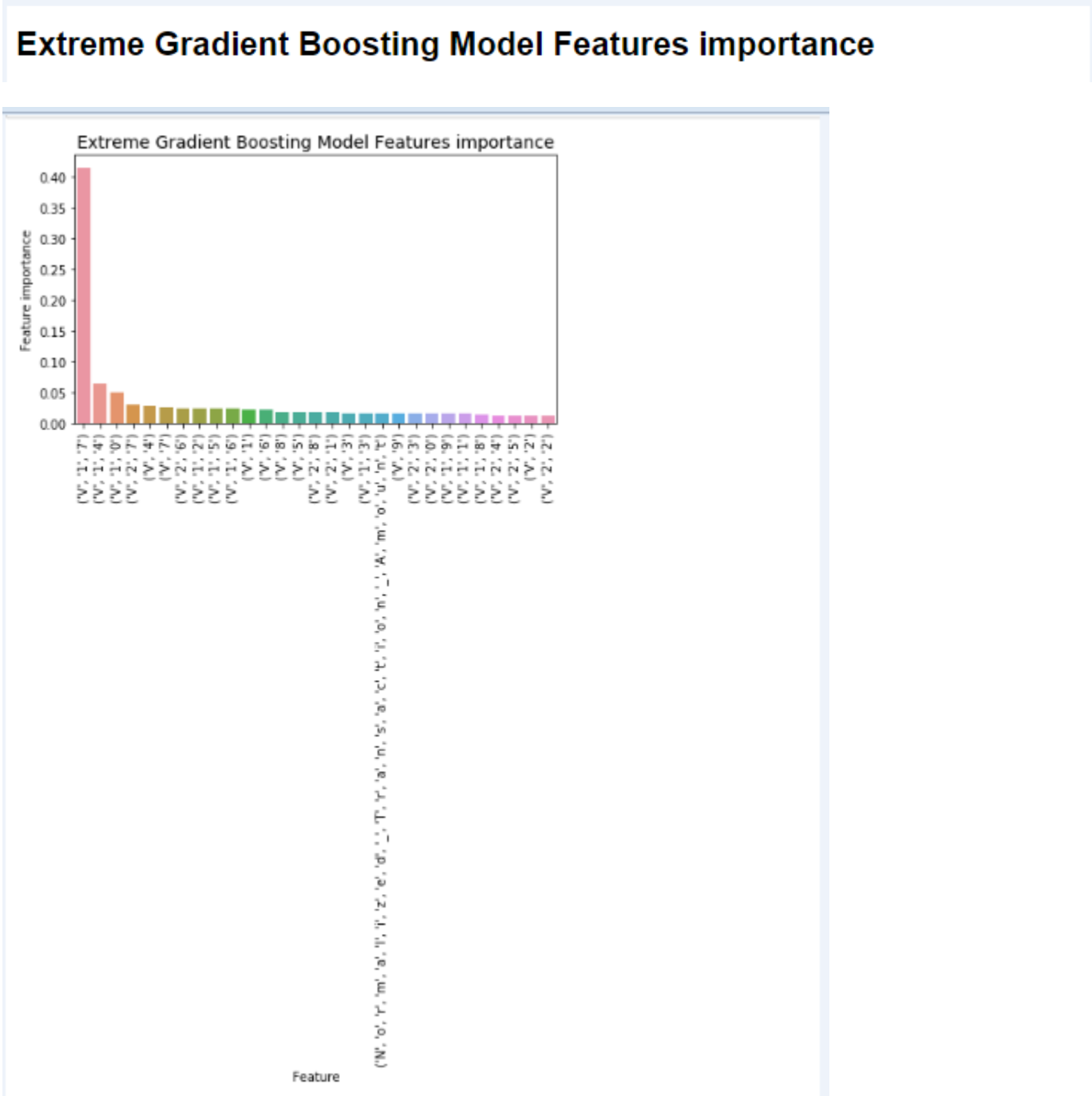


Figure 10: Extreme Gradient Boosting Features Importance

The above result identifies features V17, V14, V10, V27, and V4 as the top five features that enabled the classification of transactions as either non-fraudulent or fraudulent by the Extreme Gradient Boosting model developed for this project.

It worth noting that both Logistic Regression Classifier and KNN Classifier have no features importance attribute. Hence, there is no need to generate features importance for both classifiers.

4. Methods

4.1 Models

There are two types of supervised machine learning algorithms including Regression and Classification. To classify credit card transactions as either fraudulent or non-fraudulent, predictive classification models are used. Using Python Scikit learn library, various kinds of classification models can be developed and implemented.

For this practicum project, I chose to utilize five different Python Scikit learn classification methods/algorithms for training and building five different classification models:

- Logistic Regression: This is a classification algorithm. In this project, this algorithm is leveraged to estimate the probability of a credit card transaction class outcome as either fraudulent or non-fraudulent based on the 28 PCA transformed features and transaction amount.
- Decision Tree Classifier: This is the second classification model constructed in this project. In this project, it trains a classification model in a flowchart-like tree structure to output predicted transaction class as either fraudulent or non-fraudulent.
- Random Forest Classifier. This is the third model developed in this classification predictive project. It is an ensemble method and it combines multiple decision trees in determining the final output.
- Extreme Gradient Boosting: This is the fourth python Scikit learn library classification algorithm utilized for model construction in this project. This classification algorithm develops models in a stage-wise manner and creates a prediction model in the form of an ensemble of weak prediction models, normally decision tree.

- K Nearest Neighbors (KNN): This is the fifth classification algorithm leveraged for model construction in this project. KNN works by leveraging a number of nearest neighbors (k) to classify outcomes in a dataset.

4.2 Methods

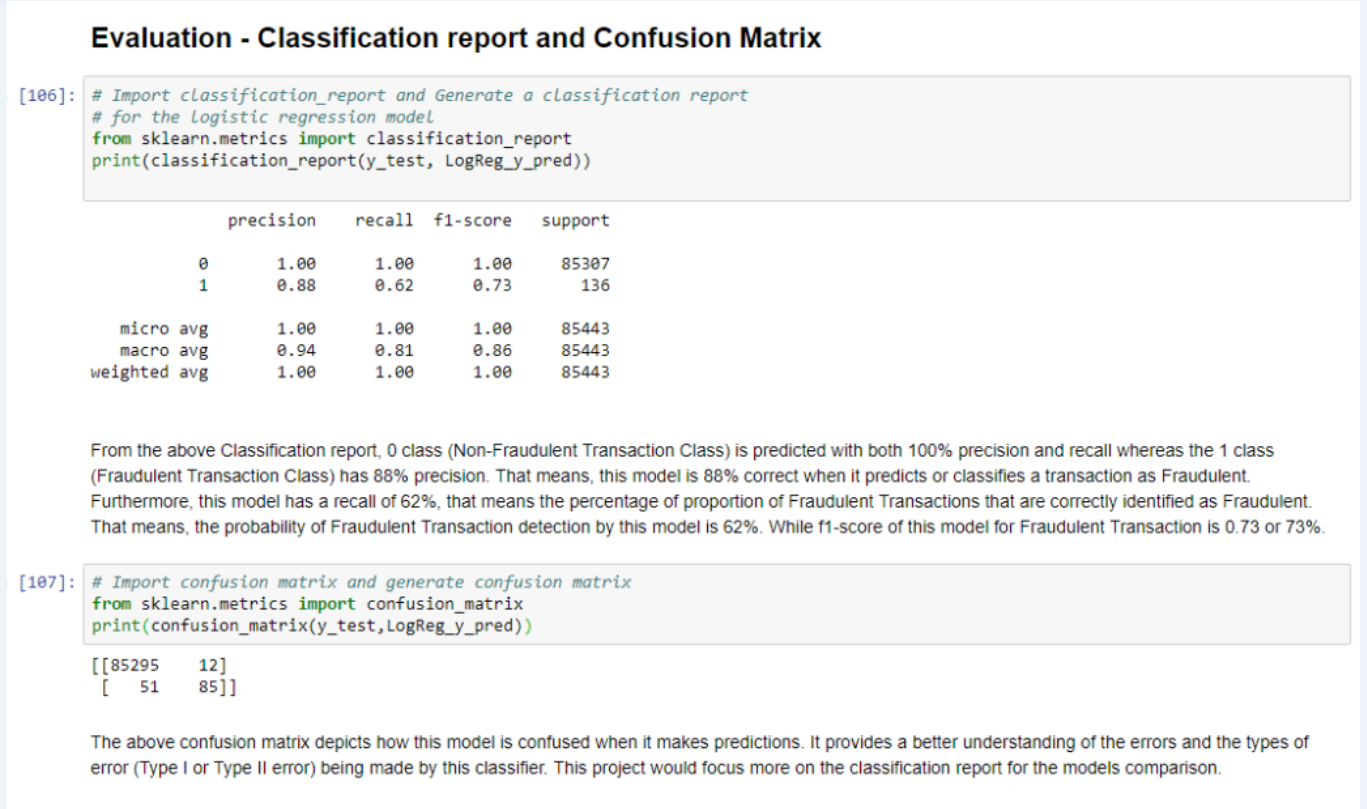
In this project the following methods were utilized in the construction of the models:

- All the five models were first constructed with a single split into training and test dataset with a 70 to 30 ratio. The models were fitted on the training dataset and the performance of the models were evaluated utilizing the test data.
- The models were developed the second time with the incorporation of 5-fold cross-validation on the dataset. This requires training the same classification model multiple (five) times leveraging different split each time.
- The models were constructed the third time utilizing both cross-validation and undersampling techniques. In this project, undersampling was performed by randomly selecting 492 observations from the non-fraudulent class to match the number of observations in the fraudulent transaction class.
- The models were constructed the fourth time utilizing both cross-validation and oversampling techniques. In this project, oversampling was performed by randomly replicating observations in fraudulent transaction class to match the number of observations in the non-fraudulent class.

5 Results

5.1 Presentation of the results

The followings are the Precision, Recall, f1-score, and Precision-Recall AUC model performance metrics generated by the five models based on a single 70:30 train to test split ratio:



Evaluation - Classification report and Confusion Matrix

```
[106]: # Import classification_report and Generate a classification report
       # for the logistic regression model
       from sklearn.metrics import classification_report
       print(classification_report(y_test, LogReg_y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85307
           1       0.88      0.62      0.73       136

   micro avg       1.00      1.00      1.00     85443
   macro avg       0.94      0.81      0.86     85443
weighted avg       1.00      1.00      1.00     85443
```

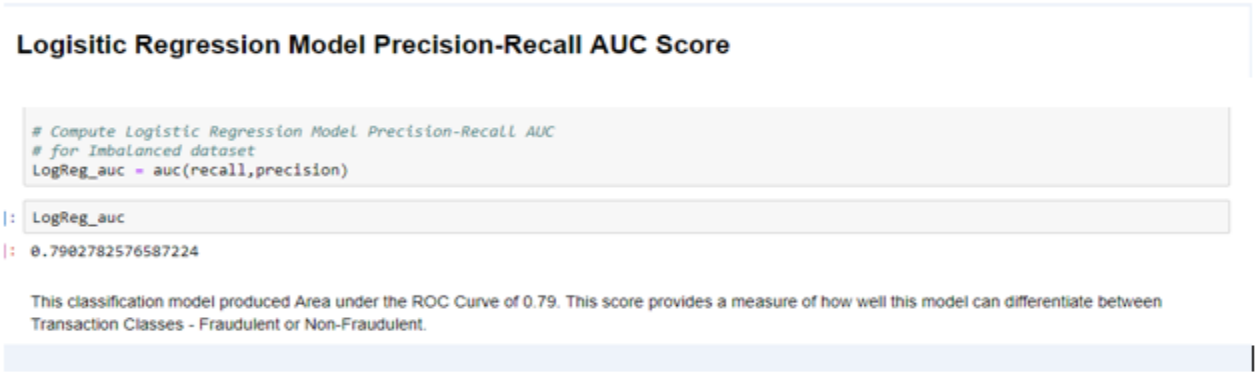
From the above Classification report, 0 class (Non-Fraudulent Transaction Class) is predicted with both 100% precision and recall whereas the 1 class (Fraudulent Transaction Class) has 88% precision. That means, this model is 88% correct when it predicts or classifies a transaction as Fraudulent. Furthermore, this model has a recall of 62%, that means the percentage of proportion of Fraudulent Transactions that are correctly identified as Fraudulent. That means, the probability of Fraudulent Transaction detection by this model is 62%. While f1-score of this model for Fraudulent Transaction is 0.73 or 73%.

```
[107]: # Import confusion matrix and generate confusion matrix
       from sklearn.metrics import confusion_matrix
       print(confusion_matrix(y_test,LogReg_y_pred))
```

```
[[85295    12]
 [   51    85]]
```

The above confusion matrix depicts how this model is confused when it makes predictions. It provides a better understanding of the errors and the types of error (Type I or Type II error) being made by this classifier. This project would focus more on the classification report for the models comparison.

Figure 11: Logistic Regression Model Evaluation Results



Logisitic Regression Model Precision-Recall AUC Score

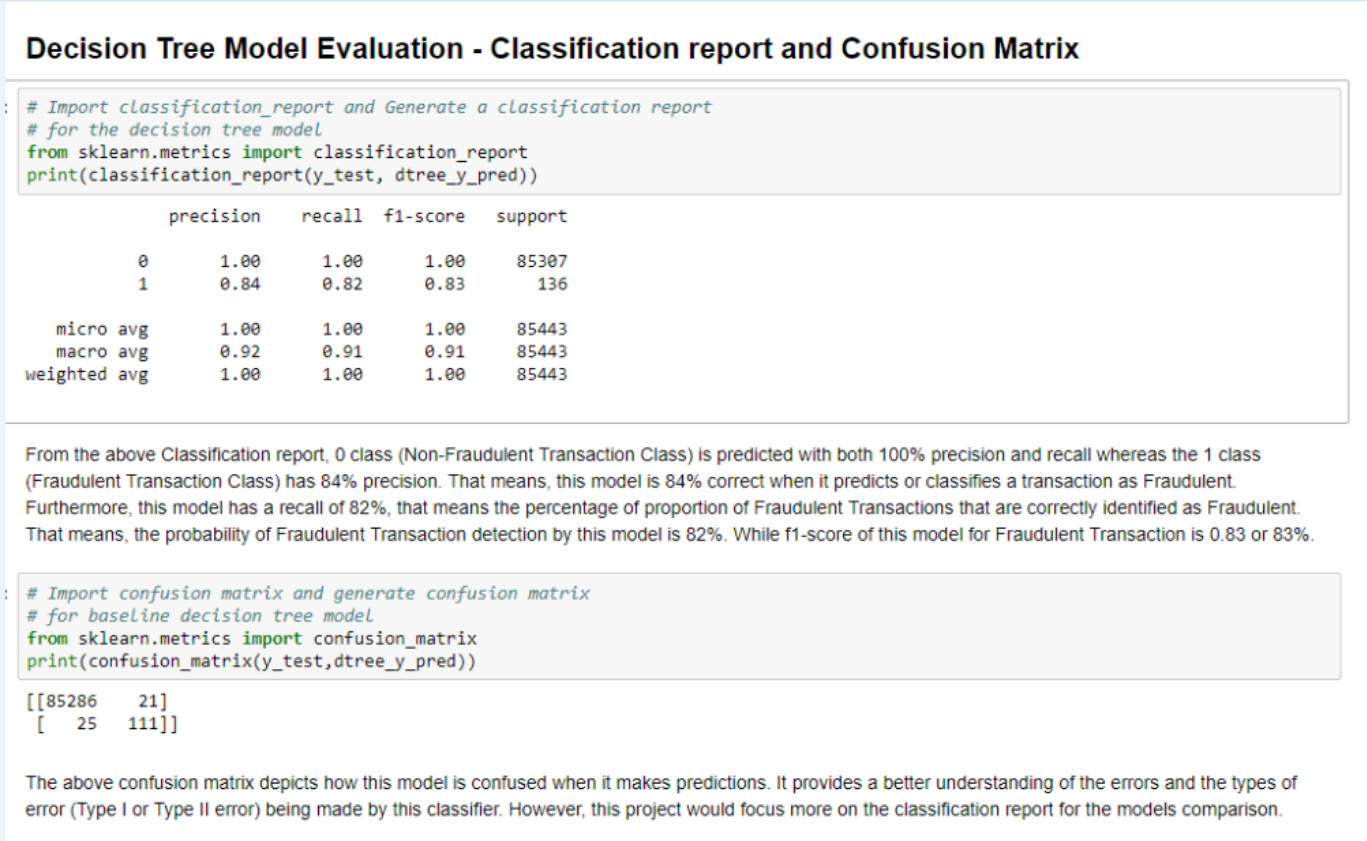
```
# Compute Logistic Regression Model Precision-Recall AUC
# for Imbalanced dataset
LogReg_auc = auc(recall,precision)
```

```
|: LogReg_auc
```

```
|: 0.7902782576587224
```

This classification model produced Area under the ROC Curve of 0.79. This score provides a measure of how well this model can differentiate between Transaction Classes - Fraudulent or Non-Fraudulent.

Figure 12: Logistic Regression Model Precision-Recall AUC Score



Decision Tree Model Evaluation - Classification report and Confusion Matrix

```
# Import classification_report and Generate a classification report
# for the decision tree model
from sklearn.metrics import classification_report
print(classification_report(y_test, dtree_y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85307
           1       0.84      0.82      0.83       136

   micro avg       1.00      1.00      1.00     85443
   macro avg       0.92      0.91      0.91     85443
weighted avg       1.00      1.00      1.00     85443
```

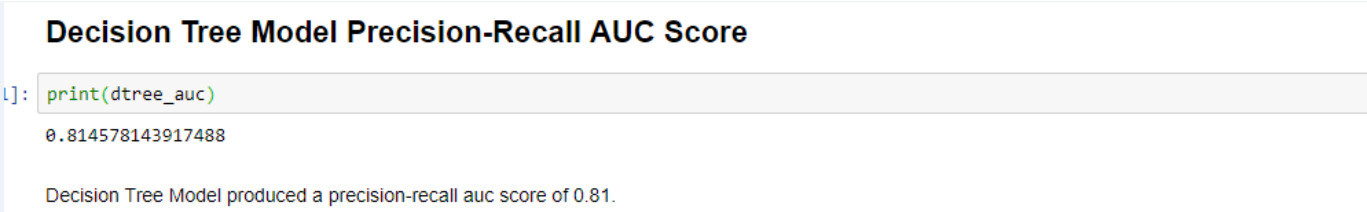
From the above Classification report, 0 class (Non-Fraudulent Transaction Class) is predicted with both 100% precision and recall whereas the 1 class (Fraudulent Transaction Class) has 84% precision. That means, this model is 84% correct when it predicts or classifies a transaction as Fraudulent. Furthermore, this model has a recall of 82%, that means the percentage of proportion of Fraudulent Transactions that are correctly identified as Fraudulent. That means, the probability of Fraudulent Transaction detection by this model is 82%. While f1-score of this model for Fraudulent Transaction is 0.83 or 83%.

```
# Import confusion matrix and generate confusion matrix
# for baseline decision tree model
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test,dtree_y_pred))
```

```
[[85286    21]
 [   25   111]]
```

The above confusion matrix depicts how this model is confused when it makes predictions. It provides a better understanding of the errors and the types of error (Type I or Type II error) being made by this classifier. However, this project would focus more on the classification report for the models comparison.

Figure 13: Decision Tree Model Evaluation Results



Decision Tree Model Precision-Recall AUC Score

```
print(dtree_auc)
```

```
0.814578143917488
```

Decision Tree Model produced a precision-recall auc score of 0.81.

Figure 14: Decision Tree Model Precision-Recall AUC Score

Random Forest Model Evaluation - Classification report and Confusion Matrix

```
# Import classification_report and Generate a classification report
# for the random forest model
from sklearn.metrics import classification_report
print(classification_report(y_test, rfc_y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85307
           1       0.94      0.82      0.88       136

   micro avg       1.00      1.00      1.00     85443
   macro avg       0.97      0.91      0.94     85443
weighted avg       1.00      1.00      1.00     85443
```

From the above Classification report, 0 class (Non-Fraudulent Transaction Class) is predicted with both 100% precision and recall whereas the 1 class (Fraudulent Transaction Class) has 94% precision. That means, this model is 94% correct when it predicts or classifies a transaction as Fraudulent. Furthermore, this model has a recall of 82%, that means the percentage or proportion of Fraudulent Transactions that are correctly identified as Fraudulent. That means, the probability of Fraudulent Transaction detection by this model is 82%. While f1-score of this model for Fraudulent Transaction is 0.88 or 88%.

```
# Import confusion matrix and generate confusion matrix
# for baseline random forest model
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test,rfc_y_pred))
```

```
[[85300     7]
 [   24   112]]
```

The above confusion matrix depicts how this model is confused when it makes predictions. It provides a better understanding of the errors and the types of error (Type I or Type II error) being made by this classifier. This project would focus more on the classification report for the models comparison.

Figure 15: Random Forest Model Evaluation Model Results

Random Forest Model Precision-Recall AUC Score

```
# Display Random Forest Model Precision-Recall AUC Score
print(rfc_auc)
```

```
0.872407816648124
```

Random Forest Model produced a precision-recall auc score of 0.87 for the imbalanced dataset.

Figure 16: Random Forest Model Precision-Recall AUC Score

Extreme Gradient Boosting Evaluation - Classification report and Confusion Matrix

```
# Import classification_report and Generate a classification report
# for the Extreme Gradient Boosting model
from sklearn.metrics import classification_report
print(classification_report(y_test, xgb_y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85307
           1       0.95      0.82      0.88       136

   micro avg       1.00      1.00      1.00     85443
   macro avg       0.97      0.91      0.94     85443
weighted avg       1.00      1.00      1.00     85443
```

From the above Classification report, 0 class (Non-Fraudulent Transaction Class) is predicted with both 100% precision and recall whereas the 1 class (Fraudulent Transaction Class) has 95% precision. That means, this model is 95% correct when it predicts or classifies a transaction as Fraudulent. Furthermore, this model has a recall of 82%, that means the percentage or proportion of Fraudulent Transactions that are correctly identified as Fraudulent. That means, the probability of Fraudulent Transaction detection by this model is 82%. While f1-score of this model for Fraudulent Transaction is 0.88 or 88%.

```
# Import confusion matrix and generate confusion matrix
# for baseline Extreme Gradient Boosting model
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test,xgb_y_pred))
```

```
[[85301     6]
 [   25   111]]
```

he above confusion matrix depicts how this model is confused when it makes predictions. It provides a better understanding of the errors and the types of error (Type I or Type II error) being made by this classifier. This project would focus more on the classification report for the models comparison.

Figure 17: Extreme Gradient Boosting Model Evaluation Model Results

Extreme Gradient Boosting Model Precision-Recall AUC Score

```
# Display Extreme Gradient Boosting Model Precision-Recall AUC Score
print(xgb_auc)
```

```
0.8949761241404558
```

Extreme Gradient Boosting Model produced a precision-recall auc score of 0.89.

Figure 18: Extreme Gradient Boosting Model Precision-Recall AUC Score

KNN Model Evaluation - Classification report and Confusion Matrix

```
# Import classification_report and Generate a classification report
# for the knn model
from sklearn.metrics import classification_report
print(classification_report(y_test, knn_y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85307
           1       0.92      0.74      0.82       136

   micro avg       1.00      1.00      1.00     85443
   macro avg       0.96      0.87      0.91     85443
weighted avg       1.00      1.00      1.00     85443
```

From the above Classification report, 0 class (Non-Fraudulent Transaction Class) is predicted with both 100% precision and recall whereas the 1 class (Fraudulent Transaction Class) has 92% precision. That means, this model is 92% correct when it predicts or classifies a transaction as Fraudulent. Furthermore, this model has a recall of 74%, that means the percentage or proportion of Fraudulent Transactions that are correctly identified as Fraudulent. That means, the probability of Fraudulent Transaction detection by this model is 74%. While f1-score of this model for Fraudulent Transaction is 0.82 or 82%.

```
# Import confusion matrix and generate confusion matrix
# for baseline knn model
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test,knn_y_pred))
```

```
[[85298     9]
 [   35   101]]
```

The above confusion matrix depicts how this model is confused when it makes predictions. It provides a better understanding of the errors and the types of error (Type I or Type II error) being made by this classifier. This project would focus more on the classification report for the models comparison.

Figure 19: KNN Model Evaluation Model Results

K Nearest Neighbors (KNN) Model Precision-Recall AUC Score

```
# Display K Nearest Neighbors Model Precision-Recall AUC Score
print(knn_auc)
```

```
0.8797679375127065
```

K Nearest Neighbors(KNN) Model produced a precision-recall auc score of 0.88.

Figure 20: KNN Model Precision-Recall AUC Score

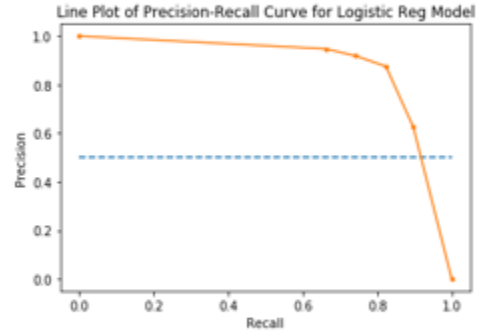
Table 4: Classification Models' Results for Train-Test Single Split Comparison

Performance Metrics	Logistic Regression Model	Decision Tree Model	Random Forest Model	Extreme Gradient Boosting Model	KNN Model
Precision	0.88	0.84	0.94	0.95	0.92
Recall	0.62	0.82	0.82	0.82	0.74
F1-Score	0.73	0.83	0.88	0.88	0.82
PR-AUC	0.79	0.81	0.87	0.89	0.88

From the results in Table 4 above, Extreme Gradient Boosting model based on a single 70/30 train to test ratio split produced precision score of 0.95, Recall score of 0.82, F1-Score of 0.88 and Precision-Recall AUC score of 0.89 for the Fraudulent Transaction Class. Extreme Gradient Boosting model produced the best metrics among the five models developed for this single train-test split.

The followings are the Precision-Recall Curves depicting the five models Precision-Recall Curves:

Precision-Recall Curve for Logistic Regression Model

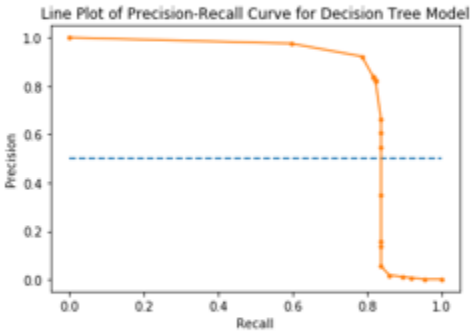



```
<Figure size 432x288 with 0 Axes>
```

The above Precision-Recall Curve is a visual representation of Logistic Regression Classifier performance. It shows the precision/recall for each threshold in comparison to a no skill model.

Figure 21: Logistic Regression Model Precision-Recall Curve

Decision Tree Model Precision-Recall AUC Curve

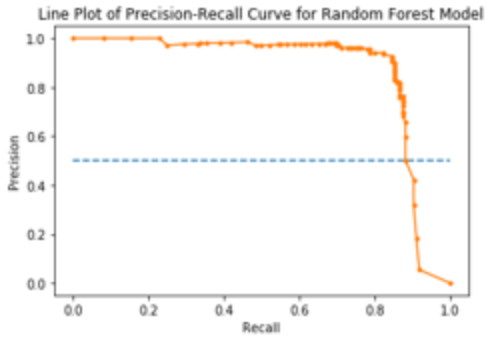



```
<Figure size 432x288 with 0 Axes>
```

The above Precision-Recall Curve is a visual representation of Decision Tree Model performance. It shows the precision/recall for each threshold in comparison to a no skill model.

Figure 22: Decision Tree Model Precision-Recall AUC Curve

Random Forest Model Precision-Recall Curve

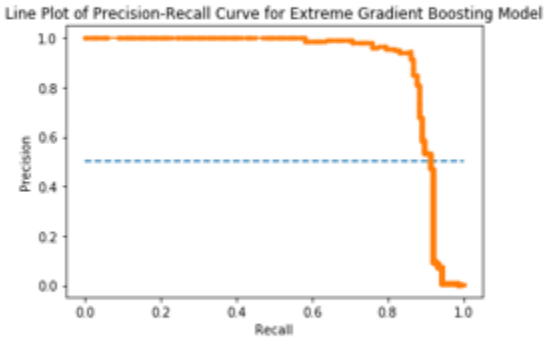



```
<Figure size 432x288 with 0 Axes>
```

The above Precision-Recall Curve is a visual representation of Random Forest Model performance. It shows the precision/recall for each threshold in comparison to a no skill model.

Figure 23: Random Forest Model Precision-Recall AUC Curve

Extreme Gradient Boosting Model Precision-Recall Curve




```
<Figure size 432x288 with 0 Axes>
```

The above Precision-Recall Curve is a visual representation of Extreme Gradient Boosting Model performance. It shows the precision/recall for each threshold in comparison to a no skill model.

Figure 24: Extreme Gradient Boosting Model Precision-Recall Curve

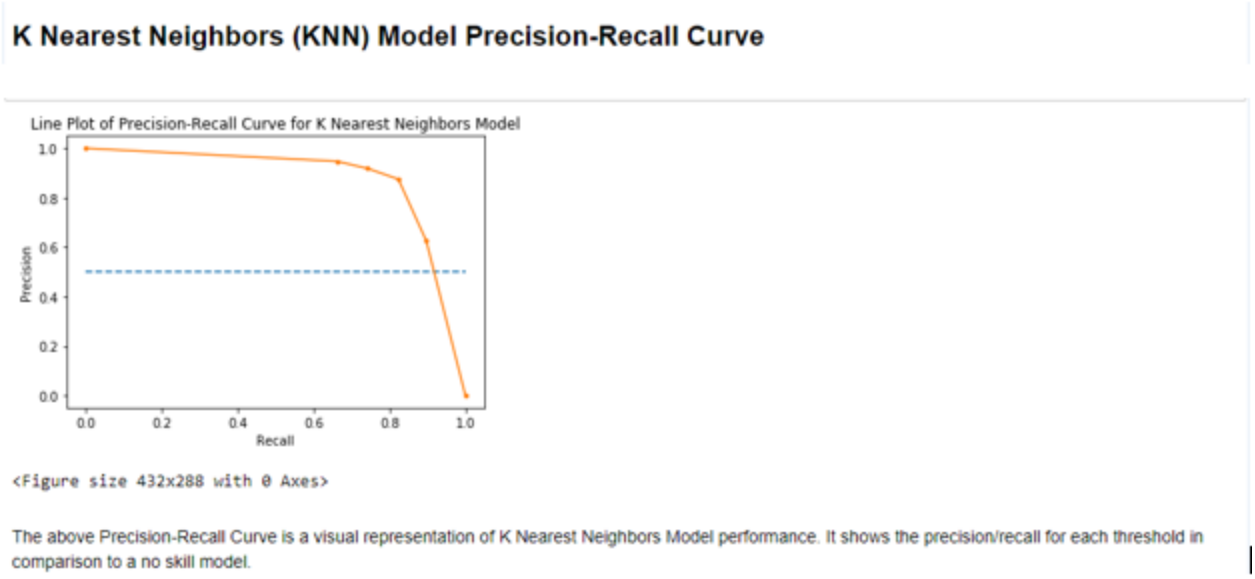


Figure 25: K Nearest Neighbors Model Precision-Recall Curve

A Precision-Recall curve is essentially a plot of the precision on the y-axis and the recall on the x-axis for different thresholds. Precision-Recall Curve is appropriate to use since we are dealing with a highly imbalanced class between non-fraudulent majority class and fraudulent minority class.

Extreme Gradient Boosting model produced the best Precision -Recall Curve among the five precision-recall curves.

Cross-Validation

The train-test split based on one split is known as classical approach. The results above are based on this approach. In this project, the technique of cross-validation is further applied to the splitting of data. This project utilized 5-fold splits in cross validation. Cross-Validation would reveal consistency in performance of the machine learning classification model and data or reveal inconsistency for further investigation.

The followings are the accuracy, precision, recall, f1-score and roc-auc model evaluation metrics generated by the five models when more than one split is done in cross validation:

Logistic Regression Model with Cross-Validation Performance Metrics

```
# Compute Logistic Regression Model with Cross Validation Performance Metrics Mean and Standard Deviation
# Display the Evaluation Metrics for Logistic Regression Classifier with Cross-Validation
print ("Average Evaluation Metrics of Logistic Regression Classifier with 5-fold Cross-Validation:")
print('Accuracy: %0.3f (+/- %0.3f)' % (LogReg_cross_val_accuracy.mean(), LogReg_cross_val_accuracy.std()))
print('Precision: %0.3f (+/- %0.3f)' % (LogReg_cross_val_precision.mean(), LogReg_cross_val_precision.std()))
print('Recall: %0.3f (+/- %0.3f)' % (LogReg_cross_val_recall.mean(), LogReg_cross_val_recall.std()))
print('F1-Score: %0.3f (+/- %0.3f)' % (LogReg_cross_val_f1.mean(), LogReg_cross_val_f1.std()))
print('roc-auc: %0.3f (+/- %0.3f)' % (LogReg_cross_val_roc_auc.mean(), LogReg_cross_val_roc_auc.std()))
```

```
Average Evaluation Metrics of Logistic Regression Classifier with 5-fold Cross-Validation:
Accuracy: 0.999 (+/- 0.000)
Precision: 0.916 (+/- 0.102)
Recall: 0.609 (+/- 0.151)
F1-Score: 0.712 (+/- 0.080)
roc-auc: 0.975 (+/- 0.011)
```

The Logistic Regression model with cross-validation produced precision score of 0.916, recall score of 0.609, F1-Score of 0.712 and ROC-AUC of 0.975 for this classification task.

Figure 26: Logistic Regression Model with Cross-Validation Evaluation Results

Decision Tree Model with Cross-Validation Performance Metrics

```
# Compute Decision Tree Model with Cross Validation Performance Metrics Mean and Standard Deviation
# Display the Evaluation Metrics for Decision Tree Classifier with Cross-Validation
print ("Average Evaluation Metrics of Decision Tree Classifier with 5-fold Cross-Validation:")
print('Accuracy: %0.3f (+/- %0.3f)' % (dtree_cross_val_accuracy.mean(), dtree_cross_val_accuracy.std()))
print('Precision: %0.3f (+/- %0.3f)' % (dtree_cross_val_precision.mean(), dtree_cross_val_precision.std()))
print('Recall: %0.3f (+/- %0.3f)' % (dtree_cross_val_recall.mean(), dtree_cross_val_recall.std()))
print('F1-Score: %0.3f (+/- %0.3f)' % (dtree_cross_val_f1.mean(), dtree_cross_val_f1.std()))
print('roc-auc: %0.3f (+/- %0.3f)' % (dtree_cross_val_roc_auc.mean(), dtree_cross_val_roc_auc.std()))
```

```
Average Evaluation Metrics of Decision Tree Classifier with 5-fold Cross-Validation:
Accuracy: 0.999 (+/- 0.000)
Precision: 0.839 (+/- 0.110)
Recall: 0.713 (+/- 0.084)
F1-Score: 0.763 (+/- 0.056)
roc-auc: 0.912 (+/- 0.025)
```

The decision tree model with cross-validation produced precision score of 0.839, recall score of 0.713, F1-Score of 0.763 and ROC-AUC of 0.912 for this classification task.

Figure 27: Decision Tree Model with Cross-Validation Evaluation Results

Random Forest Model with Cross-Validation Performance Metrics

```
# Compute Random Forest Model with Cross Validation Performance Metrics Mean and Standard Deviation
# Display the Evaluation Metrics for Random Forest Classifier with Cross-Validation
print ("Average Evaluation Metrics of Random Forest Classifier with 5-fold Cross-Validation:")
print('Accuracy: %0.3f (+/- %0.3f)' % (rfc_cross_val_accuracy.mean(), rfc_cross_val_accuracy.std()))
print('Precision: %0.3f (+/- %0.3f)' % (rfc_cross_val_precision.mean(), rfc_cross_val_precision.std()))
print('Recall: %0.3f (+/- %0.3f)' % (rfc_cross_val_recall.mean(), rfc_cross_val_recall.std()))
print('F1-Score: %0.3f (+/- %0.3f)' % (rfc_cross_val_f1.mean(), rfc_cross_val_f1.std()))
print('roc-auc: %0.3f (+/- %0.3f)' % (rfc_cross_val_roc_auc.mean(), rfc_cross_val_roc_auc.std()))
```

```
Average Evaluation Metrics of Random Forest Classifier with 5-fold Cross-Validation:
Accuracy: 0.999 (+/- 0.000)
Precision: 0.892 (+/- 0.122)
Recall: 0.764 (+/- 0.072)
F1-Score: 0.815 (+/- 0.058)
roc-auc: 0.945 (+/- 0.020)
```

The random forest model with cross-validation produced precision score of 0.892, recall score of 0.764, F1-Score of 0.815 and ROC-AUC of 0.945 for this classification task.

Figure 28: Random Forest Model with Cross Validation Evaluation Results

Extreme Gradient Boosting Model with Cross-Validation Performance Metrics

```
# Compute Extreme Gradient Boosting Model with Cross Validation Performance Metrics Mean and Standard Deviation
# Display the Evaluation Metrics for Extreme Gradient Boosting Classifier with Cross-Validation
print ("Average Evaluation of Extreme Gradient Boosting Classifier with 5-fold Cross-Validation:")
print('Accuracy: %0.3f (+/- %0.3f)' % (xgb_cross_val_accuracy.mean(), xgb_cross_val_accuracy.std()))
print('Precision: %0.3f (+/- %0.3f)' % (xgb_cross_val_precision.mean(), xgb_cross_val_precision.std()))
print('Recall: %0.3f (+/- %0.3f)' % (xgb_cross_val_recall.mean(), xgb_cross_val_recall.std()))
print('F1-Score: %0.3f (+/- %0.3f)' % (xgb_cross_val_f1.mean(), xgb_cross_val_f1.std()))
print('roc-auc: %0.3f (+/- %0.3f)' % (xgb_cross_val_roc_auc.mean(), xgb_cross_val_roc_auc.std()))
```

```
Average Evaluation of Extreme Gradient Boosting Classifier with 5-fold Cross-Validation:
Accuracy: 0.999 (+/- 0.000)
Precision: 0.852 (+/- 0.136)
Recall: 0.782 (+/- 0.073)
F1-Score: 0.809 (+/- 0.078)
roc-auc: 0.979 (+/- 0.017)
```

Extreme Gradient Boosting model with cross-validation produced precision score of 0.852, recall score of 0.782, F1-Score of 0.809 and ROC-AUC of 0.979 for this binary classification task.

Figure 29: Extreme Gradient Boosting Model with Cross Validation Evaluation Results

KNN Model with Cross-Validation Performance Metrics

```
# Compute K Nearest Neighbors (KNN) Model with Cross Validation Performance Metrics Mean and Standard Deviation
# Display the Evaluation Metrics for K Nearest Neighbors (KNN) Classifier with Cross-Validation
print ("Average Evaluation of K Nearest Neighbors (KNN) Classifier with 5-fold Cross-Validation:")
print('Accuracy: %0.3f (+/- %0.3f)' % (knn_cross_val_accuracy.mean(), knn_cross_val_accuracy.std()))
print('Precision: %0.3f (+/- %0.3f)' % (knn_cross_val_precision.mean(), knn_cross_val_precision.std()))
print('Recall: %0.3f (+/- %0.3f)' % (knn_cross_val_recall.mean(), knn_cross_val_recall.std()))
print('F1-Score: %0.3f (+/- %0.3f)' % (knn_cross_val_f1.mean(), knn_cross_val_f1.std()))
print('roc-auc: %0.3f (+/- %0.3f)' % (knn_cross_val_roc_auc.mean(), knn_cross_val_roc_auc.std()))
```

```
Average Evaluation of K Nearest Neighbors (KNN) Classifier with 5-fold Cross-Validation:
Accuracy: 0.999 (+/- 0.000)
Precision: 0.863 (+/- 0.147)
Recall: 0.699 (+/- 0.100)
F1-Score: 0.764 (+/- 0.093)
roc-auc: 0.901 (+/- 0.036)
```

K Nearest Neighbors (KNN) model with cross-validation produced precision score of 0.863, recall score of 0.699, F1-Score of 0.764 and ROC-AUC of 0.901 for this binary classification task.

Figure 30: KNN Model with Cross Validation Evaluation Results

Table 5: Classification Models' Results for 5-folds Cross-Validation Comparison

Performance Metrics	Logistic Regression Model	Decision Tree Model	Random Forest Model	Extreme Gradient Boosting Model	KNN Model
Accuracy	0.999	0.999	0.999	0.999	0.999
Precision	0.916	0.839	0.892	0.852	0.863
Recall	0.609	0.713	0.764	0.782	0.699
F1-Score	0.712	0.763	0.815	0.809	0.764
ROC-AUC	0.975	0.912	0.945	0.979	0.901

From the results in Table 5 above, Logistic Regression Model produced the best precision score, the best recall score was produced by Extreme Gradient Boosting and the best F1-Score was produced by Random Forest Model with the incorporation of cross validation to the models.

Resampling

Resampling techniques including undersampling and oversampling were incorporated to the models to deal with the issue of imbalanced transaction class

distribution. It is worth noting that imbalanced class distribution would bias the classification predictive models toward the non-fraudulent transaction class, the majority class.

In this project, undersampling was performed by randomly selecting 492 observations from the non-fraudulent class to match the number of observations in the fraudulent transaction class. Hence, we have a balanced dataset.

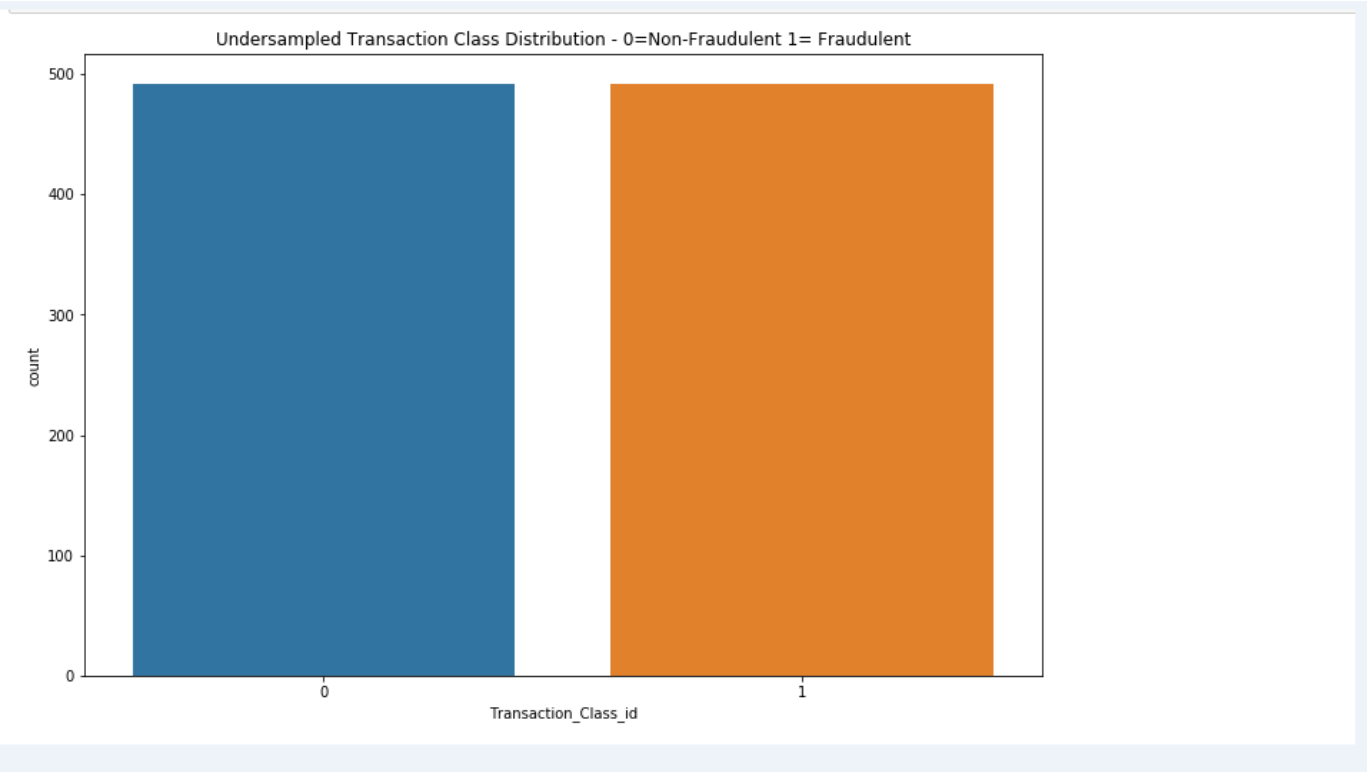


Figure 31: Undersampled Transaction Class Distribution

The followings are the accuracy, precision, recall, f1-score and roc-auc model evaluation metrics generated by the five models with the incorporation of both cross validation and undersampling techniques:

Logistic Regression Model with Cross-Validation and Undersampling Performance Metrics

```
[30]: # Compute Logistic Regression Model with Cross Validation and undersampling Performance Metrics Mean and Standard Deviation
      # Display the Evaluation Metrics for Logistic Regression Classifier with Cross-Validation and undersampling
      print ("Average Evaluation Metrics of Logistic Regression Classifier with Cross-Validation and undersampling:")
      print('Accuracy: %0.3f (+/- %0.3f)' % (LogReg_undersampled_accuracy.mean(), LogReg_undersampled_accuracy.std()))
      print('Precision: %0.3f (+/- %0.3f)' % (LogReg_undersampled_precision.mean(), LogReg_undersampled_precision.std()))
      print('Recall: %0.3f (+/- %0.3f)' % (LogReg_undersampled_recall.mean(), LogReg_undersampled_recall.std()))
      print('F1-Score: %0.3f (+/- %0.3f)' % (LogReg_undersampled_f1.mean(), LogReg_undersampled_f1.std()))
      print('roc-auc: %0.3f (+/- %0.3f)' % (LogReg_undersampled_roc_auc.mean(), LogReg_undersampled_roc_auc.std()))
```

```
Average Evaluation Metrics of Logistic Regression Classifier with Cross-Validation and undersampling:
Accuracy: 0.939 (+/- 0.017)
Precision: 0.968 (+/- 0.014)
Recall: 0.908 (+/- 0.037)
F1-Score: 0.937 (+/- 0.019)
roc-auc: 0.978 (+/- 0.009)
```

This model with cross-validation and undersampling produced an average accuracy score of 0.939, precision score of 0.968, recall score of 0.908, F1-Score of 0.937 and ROC-AUC of 0.978.

Figure 32: Logistic Regression Model with Cross Validation and Undersampling Results

Decision Tree Model with Cross-Validation and Undersampling Performance Metrics

```
[ ]: # Compute Decision Tree Model with Cross Validation and undersampling Performance Metrics Mean and Standard Deviation
     # Display the Evaluation Metrics for Decision Tree Classifier with Cross-Validation and undersampling
     print ("Average Evaluation Metrics of Decision Tree Classifier with Cross-Validation and Undersampling:")
     print('Accuracy: %0.3f (+/- %0.3f)' % (dtree_undersampled_accuracy.mean(), dtree_undersampled_accuracy.std()))
     print('Precision: %0.3f (+/- %0.3f)' % (dtree_undersampled_precision.mean(), dtree_undersampled_precision.std()))
     print('Recall: %0.3f (+/- %0.3f)' % (dtree_undersampled_recall.mean(), dtree_undersampled_recall.std()))
     print('F1-Score: %0.3f (+/- %0.3f)' % (dtree_undersampled_f1.mean(), dtree_undersampled_f1.std()))
     print('roc-auc: %0.3f (+/- %0.3f)' % (dtree_undersampled_roc_auc.mean(), dtree_undersampled_roc_auc.std()))
```

```
Average Evaluation Metrics of Decision Tree Classifier with Cross-Validation and Undersampling:
Accuracy: 0.909 (+/- 0.019)
Precision: 0.915 (+/- 0.025)
Recall: 0.904 (+/- 0.044)
F1-Score: 0.909 (+/- 0.020)
roc-auc: 0.931 (+/- 0.029)
```

This model with cross-validation and undersampling produced an average accuracy score of 0.909, precision score of 0.915, recall score of 0.904, F1-Score of 0.909 and ROC-AUC of 0.931.

Figure 33: Decision Tree Model with Cross Validation and Undersampling Results

Random Forest Model with Cross-Validation and Undersampling Performance Metrics

```
# Compute Random Forest Model with Cross Validation and undersampling Performance Metrics Mean and Standard Deviation
# Display the Evaluation Metrics for Random Forest Classifier with Cross-Validation and undersampling
print ("Average Evaluation Metrics of Random Forest Classifier with Cross-Validation and Undersampling:")
print('Accuracy: %0.3f (+/- %0.3f)' % (rfc_undersampled_accuracy.mean(), rfc_undersampled_accuracy.std()))
print('Precision: %0.3f (+/- %0.3f)' % (rfc_undersampled_precision.mean(), rfc_undersampled_precision.std()))
print('Recall: %0.3f (+/- %0.3f)' % (rfc_undersampled_recall.mean(), rfc_undersampled_recall.std()))
print('F1-Score: %0.3f (+/- %0.3f)' % (rfc_undersampled_f1.mean(), rfc_undersampled_f1.std()))
print('roc-auc: %0.3f (+/- %0.3f)' % (rfc_undersampled_roc_auc.mean(), rfc_undersampled_roc_auc.std()))
```

```
Average Evaluation Metrics of Random Forest Classifier with Cross-Validation and Undersampling:
Accuracy: 0.938 (+/- 0.020)
Precision: 0.974 (+/- 0.005)
Recall: 0.900 (+/- 0.041)
F1-Score: 0.935 (+/- 0.022)
roc-auc: 0.978 (+/- 0.013)
```

This model with cross-validation and undersampling produced an average accuracy score of 0.938, precision score of 0.974, recall score of 0.900, F1-Score of 0.935 and ROC-AUC of 0.978.

Figure 34: Random Forest Model with Cross Validation and Undersampling Results

Extreme Gradient Boosting with Cross-Validation and Undersampling Performance Metrics

```
# Compute Extreme Gradient Boosting Model with Cross Validation and undersampling Performance Metrics Mean and Standard Deviatio
# Display the Evaluation Metrics for Extreme Gradient Boosting Classifier with Cross-Validation and undersampling
print ("Average Evaluation Metrics of Extreme Gradient Boosting Classifier with Cross-Validation and Undersampling:")
print('Accuracy: %0.3f (+/- %0.3f)' % (xgb_undersampled_accuracy.mean(), xgb_undersampled_accuracy.std()))
print('Precision: %0.3f (+/- %0.3f)' % (xgb_undersampled_precision.mean(), xgb_undersampled_precision.std()))
print('Recall: %0.3f (+/- %0.3f)' % (xgb_undersampled_recall.mean(), xgb_undersampled_recall.std()))
print('F1-Score: %0.3f (+/- %0.3f)' % (xgb_undersampled_f1.mean(), xgb_undersampled_f1.std()))
print('roc-auc: %0.3f (+/- %0.3f)' % (xgb_undersampled_roc_auc.mean(), xgb_undersampled_roc_auc.std()))
```

```
Average Evaluation Metrics of Extreme Gradient Boosting Classifier with Cross-Validation and Undersampling:
Accuracy: 0.936 (+/- 0.020)
Precision: 0.963 (+/- 0.011)
Recall: 0.906 (+/- 0.038)
F1-Score: 0.934 (+/- 0.022)
roc-auc: 0.976 (+/- 0.012)
```

This model with cross-validation and undersampling produced an average accuracy score of 0.936, precision score of 0.963, recall score of 0.906, F1-Score of 0.934 and ROC-AUC of 0.976.

Figure 35: Extreme Gradient Boosting with Cross Validation and Undersampling Results

KNN Model with Cross-Validation and Undersampling Performance Metrics

```
# Compute K Nearest Neighbors Model with Cross Validation and undersampling Performance Metrics Mean and Standard Deviation
# Display the Evaluation Metrics for K Nearest Neighbors Classifier with Cross-Validation and undersampling
print ("Average Evaluation Metrics of K Nearest Neighbors Classifier with Cross-Validation and Undersampling:")
print('Accuracy: %0.3f (+/- %0.3f)' % (knn_undersampled_accuracy.mean(), knn_undersampled_accuracy.std()))
print('Precision: %0.3f (+/- %0.3f)' % (knn_undersampled_precision.mean(), knn_undersampled_precision.std()))
print('Recall: %0.3f (+/- %0.3f)' % (knn_undersampled_recall.mean(), knn_undersampled_recall.std()))
print('F1-Score: %0.3f (+/- %0.3f)' % (knn_undersampled_f1.mean(), knn_undersampled_f1.std()))
print('roc-auc: %0.3f (+/- %0.3f)' % (knn_undersampled_roc_auc.mean(), knn_undersampled_roc_auc.std()))
```

```
Average Evaluation Metrics of K Nearest Neighbors Classifier with Cross-Validation and Undersampling:
Accuracy: 0.931 (+/- 0.017)
Precision: 0.985 (+/- 0.013)
Recall: 0.876 (+/- 0.043)
F1-Score: 0.926 (+/- 0.021)
roc-auc: 0.961 (+/- 0.018)
```

This model with cross-validation and undersampling produced an average accurary score of 0.931, precision score of 0.985, recall score of 0.876, F1-Score of 0.926 and ROC-AUC of 0.961.

Figure 36: KNN Model with Cross Validation and Undersampling Results

Table 6: Classification Models' Results for Cross-Validation and Undersampling Comparison

Performance Metrics	Logistic Regression Model	Decision Tree Model	Random Forest Model	Extreme Gradient Boosting Model	KNN Model
Accuracy	0.939	0.909	0.938	0.936	0.931
Precision	0.968	0.915	0.974	0.963	0.985
Recall	0.908	0.904	0.900	0.906	0.876
F1-Score	0.937	0.909	0.935	0.934	0.926
ROC-AUC	0.978	0.931	0.978	0.976	0.961

From the results in Table 6 above, Logistic Regression Model produced the best overall performance metrics. Hence, it is the best model with the incorporation of both cross validation and undersampling to the models.

Furthermore, in this project, oversampling was performed by randomly replicating observations in fraudulent transaction class to match the number of observations in the non-fraudulent class. Hence, we have a balanced dataset.

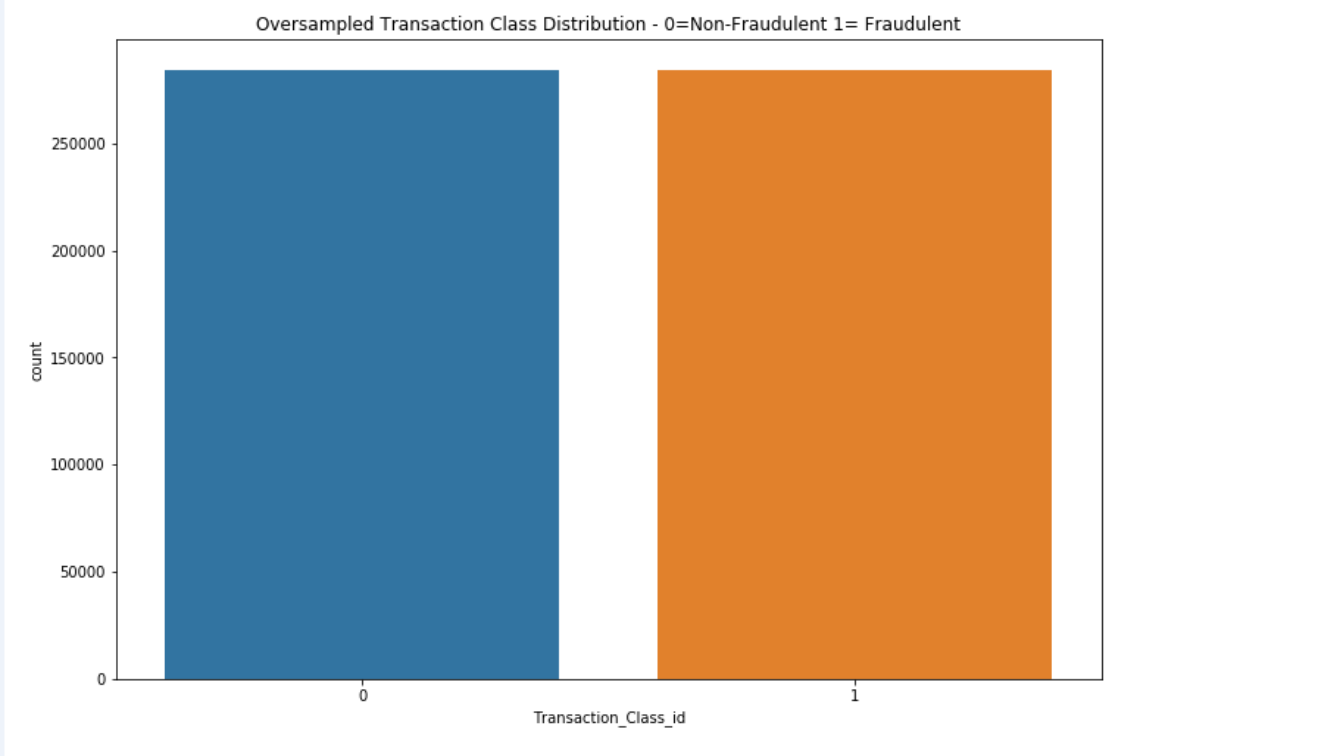


Figure 37: Oversampled Transaction Class Distribution

The followings are the accuracy, precision, recall, f1-score and roc-auc model evaluation metrics generated by the five models with the incorporation of both cross validation and oversampling techniques:

Logistic Regression Model with Cross-Validation and Oversampling Performance Metrics

```
# Compute Logistic Regression Model with Cross Validation and Oversampling Performance Metrics Mean and Standard Deviation
# Display the Evaluation Metrics for Logistic Regression Classifier with Cross-Validation and Oversampling
print ("Average Evaluation Metrics of Logistic Regression Classifier with Cross-Validation and oversampling:")
print('Accuracy: %0.3f (+/- %0.3f)' % (LogReg_oversampled_accuracy.mean(), LogReg_oversampled_accuracy.std()))
print('Precision: %0.3f (+/- %0.3f)' % (LogReg_oversampled_precision.mean(), LogReg_oversampled_precision.std()))
print('Recall: %0.3f (+/- %0.3f)' % (LogReg_oversampled_recall.mean(), LogReg_oversampled_recall.std()))
print('F1-Score: %0.3f (+/- %0.3f)' % (LogReg_oversampled_f1.mean(), LogReg_oversampled_f1.std()))
print('roc-auc: %0.3f (+/- %0.3f)' % (LogReg_oversampled_roc_auc.mean(), LogReg_oversampled_roc_auc.std()))
```

```
Average Evaluation Metrics of Logistic Regression Classifier with Cross-Validation and oversampling:
Accuracy: 0.947 (+/- 0.003)
Precision: 0.970 (+/- 0.008)
Recall: 0.922 (+/- 0.002)
F1-Score: 0.945 (+/- 0.003)
roc-auc: 0.985 (+/- 0.001)
```

This model with cross-validation and oversampling produced an average accuracy score of 0.947, precision score of 0.970, recall score of 0.922, F1-Score of 0.945 and ROC-AUC of 0.985.

Figure 38: Logistic Regression Model with Cross Validation and Oversampling Results

Decision Tree Model with Cross-Validation and Oversampling Performance Metrics

```
# Compute Decision Tree Model with Cross Validation and Oversampling Performance Metrics Mean and Standard Deviation
# Display the Evaluation Metrics for Decision Tree Classifier with Cross-Validation and Oversampling
print ("Average Evaluation Metrics of Decision Tree Classifier with Cross-Validation and oversampling:")
print('Accuracy: %0.3f (+/- %0.3f)' % (dtree_oversampled_accuracy.mean(), dtree_oversampled_accuracy.std()))
print('Precision: %0.3f (+/- %0.3f)' % (dtree_oversampled_precision.mean(), dtree_oversampled_precision.std()))
print('Recall: %0.3f (+/- %0.3f)' % (dtree_oversampled_recall.mean(), dtree_oversampled_recall.std()))
print('F1-Score: %0.3f (+/- %0.3f)' % (dtree_oversampled_f1.mean(), dtree_oversampled_f1.std()))
print('roc-auc: %0.3f (+/- %0.3f)' % (dtree_oversampled_roc_auc.mean(), dtree_oversampled_roc_auc.std()))
```

```
Average Evaluation Metrics of Decision Tree Classifier with Cross-Validation and oversampling:
Accuracy: 0.960 (+/- 0.008)
Precision: 0.952 (+/- 0.018)
Recall: 0.968 (+/- 0.014)
F1-Score: 0.960 (+/- 0.008)
roc-auc: 0.995 (+/- 0.002)
```

This model with cross-validation and oversampling produced an average accuracy score of 0.960, precision score of 0.952, recall score of 0.968, F1-Score of 0.960 and ROC-AUC of 0.995.

Figure 39: Decision Tree Model with Cross Validation and Oversampling Results

Random Forest Model with Cross-Validation and Oversampling Performance Metrics

```
# Compute Random Forest Model with Cross Validation and Oversampling Performance Metrics Mean and Standard Deviation
# Display the Evaluation Metrics for Random Forest Classifier with Cross-Validation and Oversampling
print ("Average Evaluation Metrics of Random Forest Classifier with Cross-Validation and oversampling:")
print('Accuracy: %0.3f (+/- %0.3f)' % (rfc_oversampled_accuracy.mean(), rfc_oversampled_accuracy.std()))
print('Precision: %0.3f (+/- %0.3f)' % (rfc_oversampled_precision.mean(), rfc_oversampled_precision.std()))
print('Recall: %0.3f (+/- %0.3f)' % (rfc_oversampled_recall.mean(), rfc_oversampled_recall.std()))
print('F1-Score: %0.3f (+/- %0.3f)' % (rfc_oversampled_f1.mean(), rfc_oversampled_f1.std()))
print('roc-auc: %0.3f (+/- %0.3f)' % (rfc_oversampled_roc_auc.mean(), rfc_oversampled_roc_auc.std()))
```

```
Average Evaluation Metrics of Random Forest Classifier with Cross-Validation and oversampling:
Accuracy: 1.000 (+/- 0.000)
Precision: 1.000 (+/- 0.000)
Recall: 1.000 (+/- 0.000)
F1-Score: 1.000 (+/- 0.000)
roc-auc: 1.000 (+/- 0.000)
```

This model when combined with both cross-validation and oversampling produced all round perfect performance metrics for this binary classification task.

Figure 40: Random Forest Model with Cross Validation and Oversampling Results

Extreme Gradient Boosting with Cross-Validation and Oversampling Performance Metrics

```
# Compute Extreme Gradient Boosting Model with Cross Validation and Oversampling Performance Metrics Mean and Standard Deviation
# Display the Evaluation Metrics for Extreme Gradient Boosting Classifier with Cross-Validation and Oversampling
print ("Average Evaluation Metrics of Extreme Gradient Boosting Classifier with Cross-Validation and oversampling:")
print('Accuracy: %0.3f (+/- %0.3f)' % (xgb_oversampled_accuracy.mean(), xgb_oversampled_accuracy.std()))
print('Precision: %0.3f (+/- %0.3f)' % (xgb_oversampled_precision.mean(), xgb_oversampled_precision.std()))
print('Recall: %0.3f (+/- %0.3f)' % (xgb_oversampled_recall.mean(), xgb_oversampled_recall.std()))
print('F1-Score: %0.3f (+/- %0.3f)' % (xgb_oversampled_f1.mean(), xgb_oversampled_f1.std()))
print('roc-auc: %0.3f (+/- %0.3f)' % (xgb_oversampled_roc_auc.mean(), xgb_oversampled_roc_auc.std()))
```

```
Average Evaluation Metrics of Extreme Gradient Boosting Classifier with Cross-Validation and oversampling:
Accuracy: 1.000 (+/- 0.000)
Precision: 1.000 (+/- 0.000)
Recall: 1.000 (+/- 0.000)
F1-Score: 1.000 (+/- 0.000)
roc-auc: 1.000 (+/- 0.000)
```

This model when combined with both cross-validation and oversampling produced all round perfect performance metrics for this binary classification task.

Figure 41: Extreme Gradient Boosting with Cross Validation and Oversampling Results

KNN Model with Cross-Validation and Oversampling Performance Metrics

```
# Compute K Nearest Neighbors (KNN) Model with Cross Validation and Oversampling Performance Metrics Mean and Standard Deviation
# Display the Evaluation Metrics for Random Forest Classifier with Cross-Validation and Oversampling
print ("Average Evaluation Metrics of Random Forest Classifier with Cross-Validation and oversampling:")
print('Accuracy: %0.3f (+/- %0.3f)' % (knn_oversampled_accuracy.mean(), knn_oversampled_accuracy.std()))
print('Precision: %0.3f (+/- %0.3f)' % (knn_oversampled_precision.mean(), knn_oversampled_precision.std()))
print('Recall: %0.3f (+/- %0.3f)' % (knn_oversampled_recall.mean(), knn_oversampled_recall.std()))
print('F1-Score: %0.3f (+/- %0.3f)' % (knn_oversampled_f1.mean(), knn_oversampled_f1.std()))
print('roc-auc: %0.3f (+/- %0.3f)' % (knn_oversampled_roc_auc.mean(), knn_oversampled_roc_auc.std()))
```

```
Average Evaluation Metrics of Random Forest Classifier with Cross-Validation and oversampling:
Accuracy: 0.999 (+/- 0.001)
Precision: 0.998 (+/- 0.002)
Recall: 1.000 (+/- 0.000)
F1-Score: 0.999 (+/- 0.001)
roc-auc: 0.999 (+/- 0.001)
```

This model with cross-validation and oversampling produced precision score of 0.998, recall score of 1.000, F1-Score of 0.999 and ROC-AUC of 0.999 for this binary classification task.

Figure 42: KNN Model with Cross Validation and Oversampling Results

Table 7: Classification Models' Results for Cross-Validation and Oversampling Comparison

Performance Metrics	Logistic Regression Model	Decision Tree Model	Random Forest Model	Extreme Gradient Boosting Model	KNN Model
Accuracy	0.947	0.960	1.000	1.000	0.999
Precision	0.970	0.952	1.000	1.000	0.998
Recall	0.922	0.968	1.000	1.000	1.000
F1-Score	0.945	0.960	1.000	1.000	0.999
ROC-AUC	0.985	0.995	1.000	1.000	0.999

From the Table 7 above, all the models produced their best performance results with the incorporation of both cross validation and oversampling. Both Random Forest and Extreme Gradient Boosting models produced perfect performance results. Hence, we have a tie between the two models.

6 Conclusions

- In this project, five classification predictive models leveraging Linear Regression, Decision Tree, Random Forest, Extreme Gradient Boosting and K Nearest Neighbors (KNN) were developed for credit card fraud detection.
- Most of the transactions in the dataset are Non-Fraudulent while only very few transactions are Fraudulent Transactions. Hence, the dataset has an imbalanced transaction class distribution.
- For the imbalanced data with single 70/30 train to test ratio split, Extreme Gradient Boosting model produced the best result with precision score of 0.95, Recall score of 0.82, F1-Score of 0.88 and Precision-Recall AUC score of 0.89 for the Fraudulent Transaction Class.
- Extreme Gradient Boosting model also produced the best Precision-Recall curve among the five models' precision-recall curves for the imbalanced data.

- For the balanced data with cross validation and undersampling, Logistic Regression Model produced the best result with accuracy of 0.939, precision score of 0.968, recall score of 0.908, F1-Score of 0.937 and ROC-AUC of 0.978.
- Balanced dataset was produced through resampling techniques including undersampling and oversampling.
- For the balanced data with cross validation and undersampling, Logistic Regression Model produced the best result with accuracy of 0.939, precision score of 0.968, recall score of 0.908, F1-Score of 0.937 and ROC-AUC of 0.978.
- For the balanced data with cross validation and oversampling, all the five models produced their best performance results in comparison with results obtained from the single split, cross-validation and cross validation with undersampling.
- For balanced data with cross validation and oversampling, both random forest and extreme gradient boosting models produced perfect performance results. Hence, there is a tie between the two models.
- From the results obtained from this project, my recommended solution for this credit card fraud detection is to go with balanced data leveraging both cross validation and oversampling with Random Forest model as my first choice and Extreme Gradient Boosting model as my second choice.

7 Future Work

Future work on this project would involve utilizing unsupervised learning methods to perform credit card fraud detection. It would be interesting to see how the performance of unsupervised learning methods would compare with the supervised learning methods' results obtained in this project.

Furthermore, I would like to learn if better performance results can be obtained with the utilization of other resampling techniques including SMOTE (Synthetic Minority Over-sampling Technique) combined with cross-validation.

References

Brownlee J. May 2016. Metrics To Evaluate Machine Learning Algorithms in Python. Retrieved from https://machinelearningmastery.com/metrics-evaluate-machine-learning-algorithms-python/

Brownlee J. August 2018. How and When to Use ROC Curves and Precision-Recall Curves for Classification in Python. Retrieved from

https://www.researchgate.net/publication/332631570_A_Comparison_Study_of_Credit_Card_Fraud_Detection_Supervised_versus_Unsupervised/link/5cc6b8894585156cd7b9aa0c/download

https://stackoverflow.com/questions.

https://www.kaggle.com/mlg-ulb/creditcardfraud/discussion

https://www.kaggle.com/mlg-ulb/creditcardfraud/kernels

https://www.researchgate.net/publication/40227011_Credit_card_fraud_and_detection_techniques_A_review