PROJECT REPORT
TEAM 7
BUAN 6341 – APPLIED MACHINE LEARNING

Team 7 Members

Khushali Shah

Shreyas Nellore

Anala Peddi Reddy

Mohanish Pradeep

Vishal Karur

Duc Hoang

I. Background and Motivation:

Background knowledge: Credit card fraud can be described as a form of identity theft where someone else's credit card or account is accessed without their permission. After getting access to credit card information there may be numerable unauthorized purchases or even cash advances without the card holder's knowledge or consent. The credit card theft can be actual physical possession of someone else's card or even virtual form of it in the form of account takeover, a cloned card or card-not present theft. In the later case, credit card account information may be obtained in various ways like through hacking, phishing or even purchases made on the dark web. In order to detect such credit card fraud transactions, financial institutions use AI and machine learning techniques. In this project, we will be demonstrating how classification supervised machine learning models that are discrete in nature (with an outcome of y=1 or 0) can be used to analyze the data and find whether a given credit card transaction is fraud or not using Python.

Motivation: One of the motivation behind choosing this topic was to put the supervised machine learning techniques learned in this class to practical application. Another factor of motivation was personal experience of being a victim of debit card fraud in real life in a foreign country as an international student for some of our group members.

II. Data:

Data Description: The credit card fraud detection dataset, available on Kaggle, spans over a 2-year period from 1 January 2019 to 31 December 2020 and comprises 1000 customers and 800 merchants. With 22 variables, it includes both legitimate and fraudulent credit card transactions. The dataset, split into a training set with 1,296,675 observations and a test set with 555,719 observations, offers an ideal platform for developing and testing machine learning models to identify fraudulent activities. Valuable for algorithm development, the dataset's diverse features and substantial volume make it a valuable resource for practitioners and enthusiasts alike. The Kaggle link provides access for further exploration.
https://www.kaggle.com/datasets/kartik2112/fraud-detection?select=fraudTrain.csv

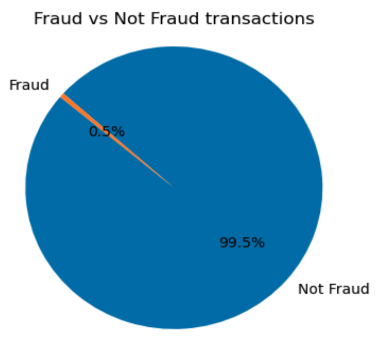
Out of the 22 variables, the key variables used in our model after performing exploratory data analysis were : **cc_num** - Credit Card Number of Customer, **amt** - Amount of Transaction, **zip** - Zip of Credit Card Holder, **lat** - Latitude Location of Credit Card Holder, **long** - Longitude Location of Credit Card Holder, **city_pop** - Credit Card Holder's City Population, **unix_time** - UNIX Time of transaction, **merch_lat** - Latitude Location of Merchant, **merch_long** - Longitude Location of Merchant, **is_fraud** - Fraud Flag – this is the target class variable.

Feature Engineering: In the quest to enhance the efficacy of our credit card fraud detection model, key feature engineering strategies were implemented. The introduction of the 'Age' column, derived from the

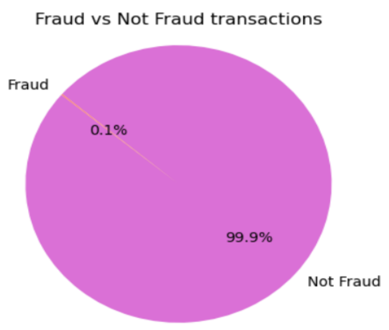
'Date of Birth' feature, provides a temporal dimension to the dataset, potentially aiding in identifying patterns associated with different age groups. Simultaneously, a novel 'Card Type' column was incorporated, extracting information from the first digit of the credit card number. This intelligent categorization represents a significant enrichment, where the first digit, for instance, '4' signifies Visa, '5' indicates Mastercard, '6' corresponds to Discover, and so forth. To ensure the dataset's compatibility with machine learning algorithms, gender underwent one-hot encoding as a categorical variable, while credit card type was subjected to label encoding for effective model interpretation. To standardize and scale the 'Amount' and 'City Population' features, both Min-max and Standard Scaling techniques were employed. These meticulous feature engineering aims to fortify our fraud detection model with enriched, standardized, and provider-specific inputs for heightened accuracy and robust predictions.

Exploratory Data Analysis:

Fraud vs non-fraud with outliers



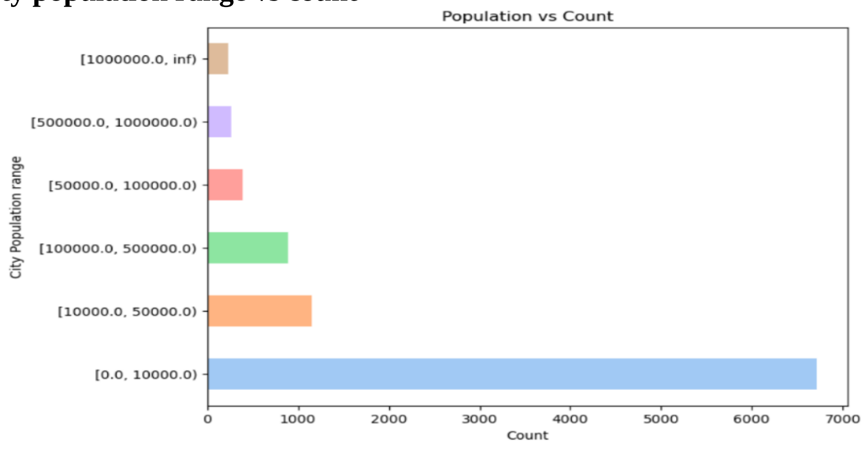
Fraud vs non-fraud without outliers



The presented Pie-chart provides a clear visual representation, underscoring the pivotal role of including outliers in our analysis. Notably, the incorporation of outliers contributes substantially to the observed rise in the proportion of fraudulent transactions.

By retaining these outliers, we ensure that our credit card fraud detection model is exposed to diverse patterns and anomalies, enhancing its adaptability and robustness to real-world scenarios.

City population range vs count

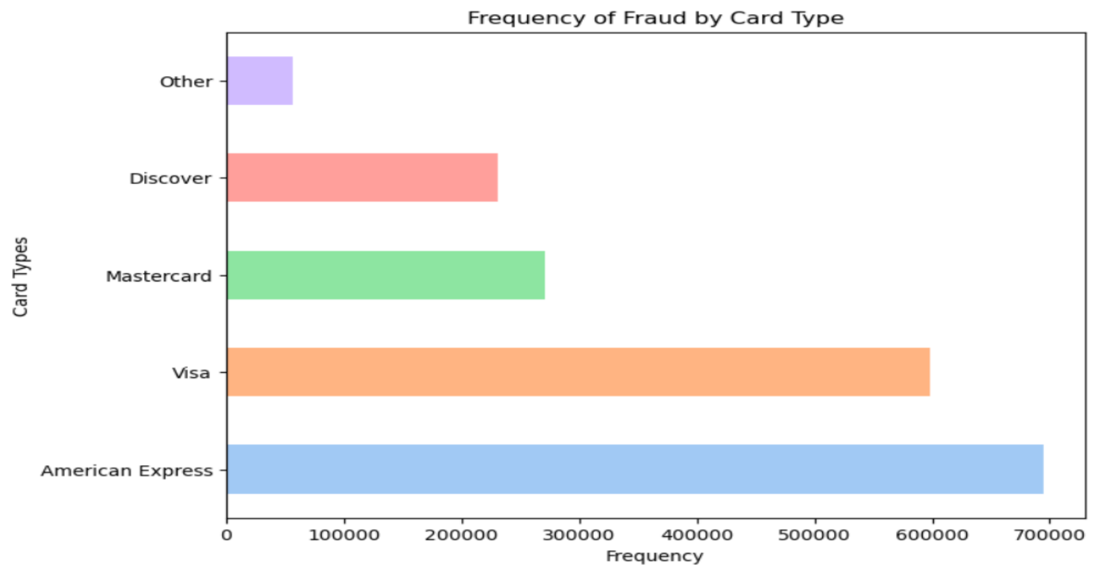


The graph depicting city population ranges and their corresponding counts highlights a predominant trend, with approximately

70% of the total observations originating from cities with populations under 10,000. Following closely, around 12% of individuals are associated with cities having populations below 50,000.

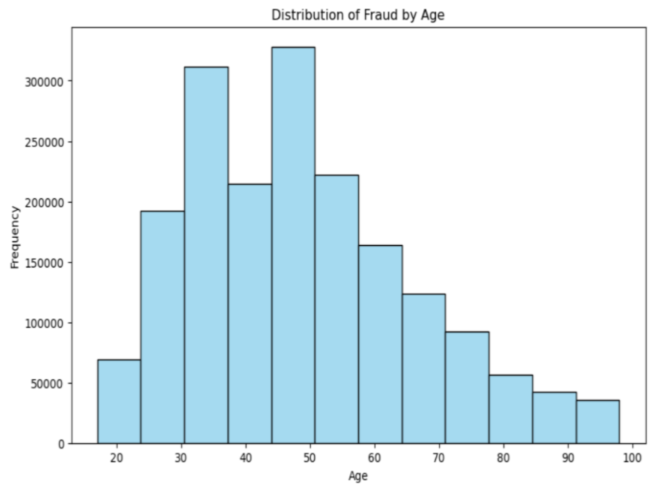
Notably, outliers are observed, representing roughly 2.5% of the total records, where individuals hail from cities exceeding a population of one million. This succinctly underscores the distribution pattern, emphasizing the concentration of observations in smaller population centers and the presence of noteworthy outliers in larger urban areas.

Frequency of fraud by card type



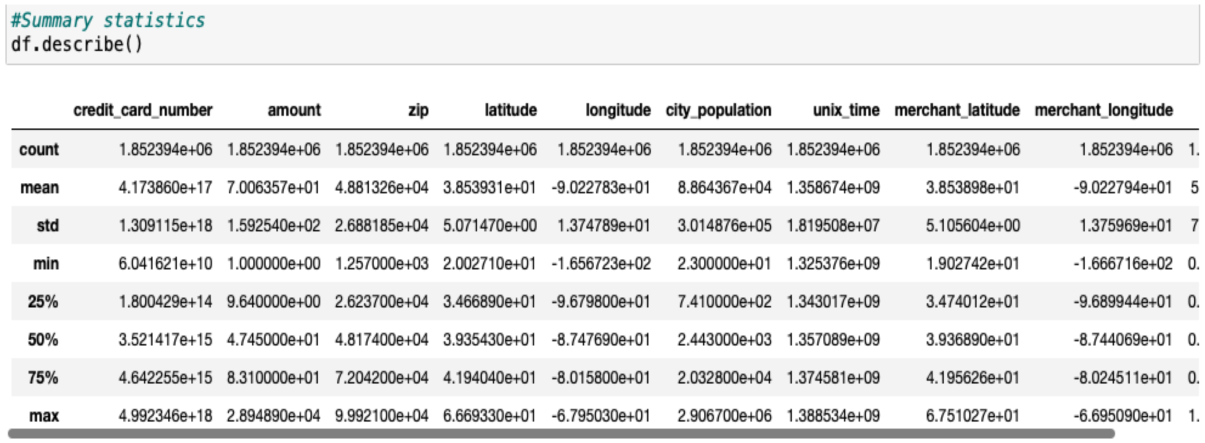
American Express users emerge as the predominant cohort within the fraudulent dataset, constituting approximately 37.5% of the records, while Visa cardholders closely follow, comprising around 32%. Other card types represent a marginal share, accounting for a mere 3% of the fraudulent dataset.

Distribution of Fraud by Age:



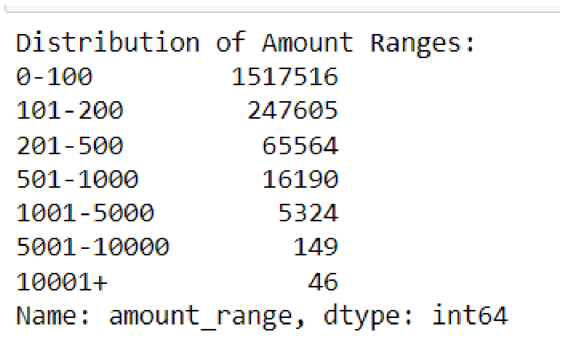
The histogram exhibits a distinctive right-skewed pattern with an elongated tail, implying that the age distribution is skewed towards higher values. Notably, the age group most susceptible to fraud appears to center around 50 years, as evidenced by a peak frequency exceeding 300,000 instances. However, beyond the age of 60, there is a discernible decline in frequency, indicating a diminishing occurrence of fraud in older age groups.

Preprocessing:



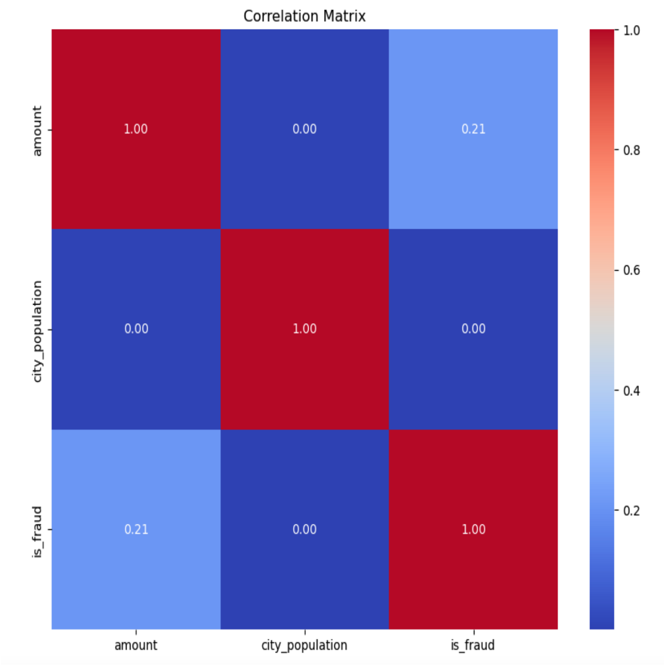
```
#Summary statistics
df.describe()
```

	credit_card_number	amount	zip	latitude	longitude	city_population	unix_time	merchant_latitude	merchant_longitude	
count	1.852394e+06	1.852394e+06	1.852394e+06	1.852394e+06	1.852394e+06	1.852394e+06	1.852394e+06	1.852394e+06	1.852394e+06	1.
mean	4.173860e+17	7.006357e+01	4.881326e+04	3.853931e+01	-9.022783e+01	8.864367e+04	1.358674e+09	3.853898e+01	-9.022794e+01	5
std	1.309115e+18	1.592540e+02	2.688185e+04	5.071470e+00	1.374789e+01	3.014876e+05	1.819508e+07	5.105604e+00	1.375969e+01	7
min	6.041621e+10	1.000000e+00	1.257000e+03	2.002710e+01	-1.656723e+02	2.300000e+01	1.325376e+09	1.902742e+01	-1.666716e+02	0.
25%	1.800429e+14	9.640000e+00	2.623700e+04	3.466890e+01	-9.679800e+01	7.410000e+02	1.343017e+09	3.474012e+01	-9.689944e+01	0.
50%	3.521417e+15	4.745000e+01	4.817400e+04	3.935430e+01	-8.747690e+01	2.443000e+03	1.357089e+09	3.936890e+01	-8.744069e+01	0.
75%	4.642255e+15	8.310000e+01	7.204200e+04	4.194040e+01	-8.015800e+01	2.032800e+04	1.374581e+09	4.195626e+01	-8.024511e+01	0.
max	4.992346e+18	2.894890e+04	9.992100e+04	6.669330e+01	-6.795030e+01	2.906700e+06	1.388534e+09	6.751027e+01	-6.695090e+01	1.



```
Distribution of Amount Ranges:
0-100          1517516
101-200         247605
201-500          65564
501-1000         16190
1001-5000         5324
5001-10000         149
10001+              46
Name: amount_range, dtype: int64
```

The initial examination revealed the absence of null values, indicating the completeness of the dataset. Further scrutiny for duplicate entries yielded a clean dataset without any instances of redundant records. To assess the presence of outliers, the 'Amount' column was specifically examined. The data reveals a substantial majority of transactions falling within the range of 0 to 100 Dollars, Notably, there are occurrences in the upper ranges (e.g., 5001-10000 and 10001+), albeit with significantly fewer instances, indicating the presence of transactions considered as outliers. When we do exploratory analysis of data the ones without outliers performed much poorer than the ones with outliers so we have kept it in the Dataset for further processing.

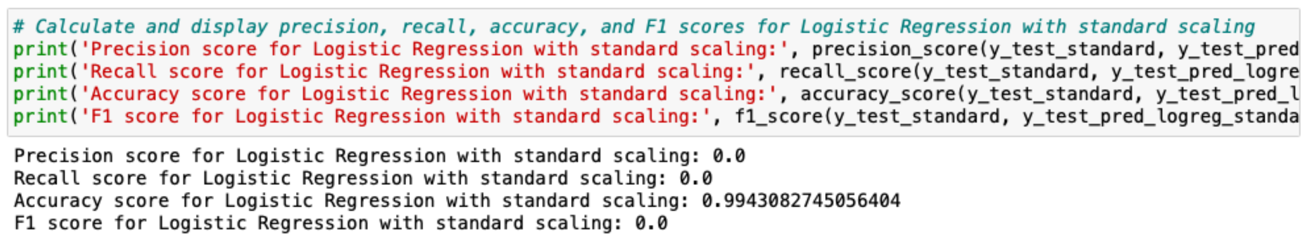


A crucial aspect of our analysis involved investigating the relationships within the dataset through correlation analysis. Specifically, we examined the correlations between the 'Amount' column, reflecting transaction amounts, the 'City Population' column, indicating the population of the cardholder's city, and the 'Is_Fraud' column, which serves as our target variable indicating potential fraud. This exploration aimed to unveil any discernible patterns or dependencies between transaction amounts, city population, and instances of fraud.

III. Model and Performance Evaluation

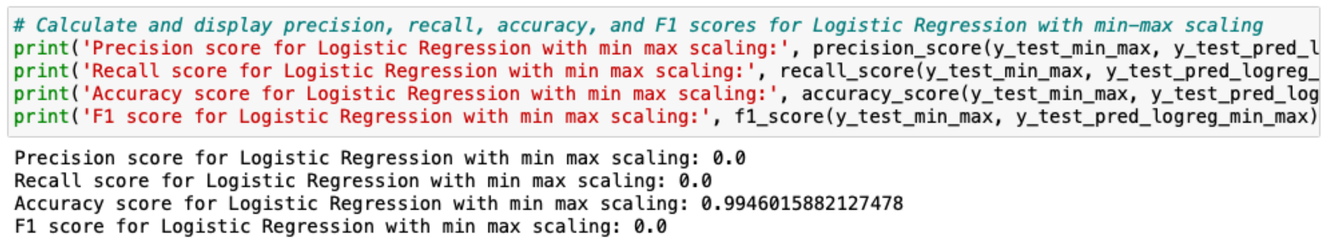
Logistic Regression (Without Upsampling)

We first performed Logistic Regression on the dataset scaled using both standard and minmax scaler



```
# Calculate and display precision, recall, accuracy, and F1 scores for Logistic Regression with standard scaling
print('Precision score for Logistic Regression with standard scaling:', precision_score(y_test_standard, y_test_pred
print('Recall score for Logistic Regression with standard scaling:', recall_score(y_test_standard, y_test_pred_logre
print('Accuracy score for Logistic Regression with standard scaling:', accuracy_score(y_test_standard, y_test_pred_l
print('F1 score for Logistic Regression with standard scaling:', f1_score(y_test_standard, y_test_pred_logreg_standa
```

```
Precision score for Logistic Regression with standard scaling: 0.0
Recall score for Logistic Regression with standard scaling: 0.0
Accuracy score for Logistic Regression with standard scaling: 0.9943082745056404
F1 score for Logistic Regression with standard scaling: 0.0
```



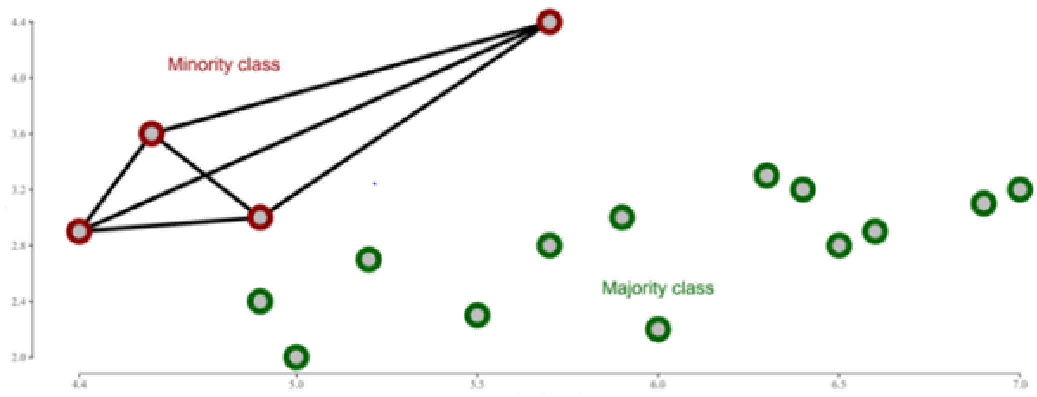
```
# Calculate and display precision, recall, accuracy, and F1 scores for Logistic Regression with min-max scaling
print('Precision score for Logistic Regression with min max scaling:', precision_score(y_test_min_max, y_test_pred_l
print('Recall score for Logistic Regression with min max scaling:', recall_score(y_test_min_max, y_test_pred_logreg_
print('Accuracy score for Logistic Regression with min max scaling:', accuracy_score(y_test_min_max, y_test_pred_log
print('F1 score for Logistic Regression with min max scaling:', f1_score(y_test_min_max, y_test_pred_logreg_min_max)
```

```
Precision score for Logistic Regression with min max scaling: 0.0
Recall score for Logistic Regression with min max scaling: 0.0
Accuracy score for Logistic Regression with min max scaling: 0.9946015882127478
F1 score for Logistic Regression with min max scaling: 0.0
```

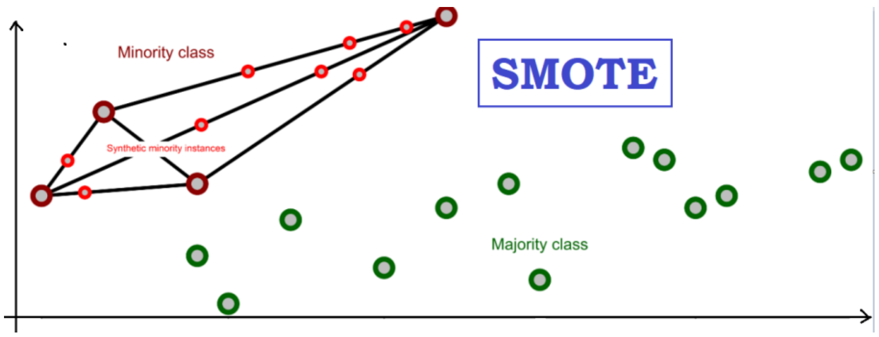
Looking at the performance metrics for this model it can be clearly understood that although the accuracy of both the models are fairly high 99.4%, all the other three metrics precision, recall and F1 score are zero. This can be attributed to the fact that the dataset is imbalanced as discussed earlier in EDA. Therefore, an oversampling technique SMOTE can be applied to overcome this imbalance.

SMOTE (Synthetic Minority Oversampling Technique) (Reference 1: Genesis, 2018)

Synthetic Minority Over-sampling Technique, or SMOTE for short, is a preprocessing technique used to address a class imbalance in a dataset.

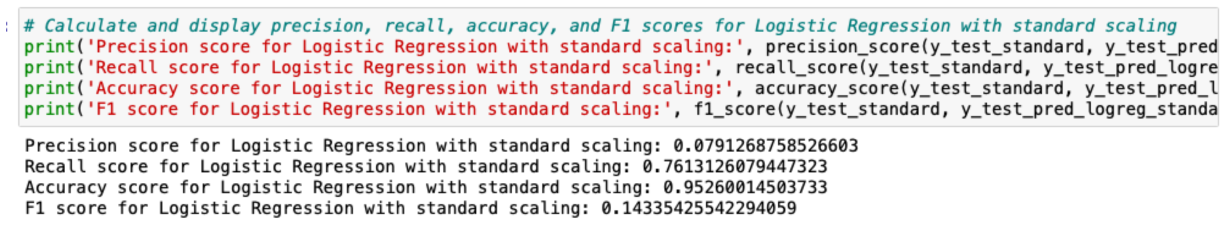


Specifically, a random example from the minority class is first chosen. Then nearest neighbors for that example are found. A randomly selected neighbor is chosen, and a synthetic example is created at a randomly selected point between the two examples in feature space as shown in the screenshot below.



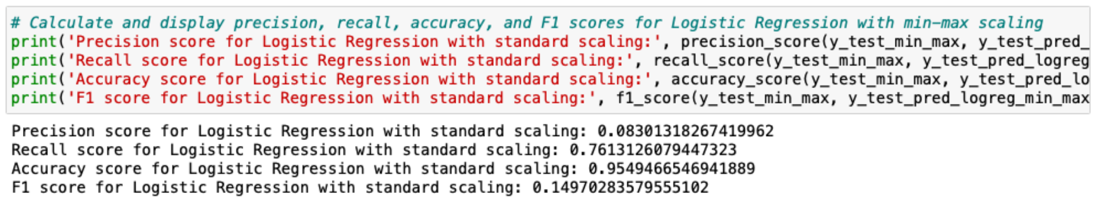
Logistic Regression (With Upsampling)

Running the Logistic regression again on the upsampled data provides us with a much better model and the improved performance metrics are a sign of a much better model. Although the accuracy has dropped from 99% to 95% approximately, this is still a much better model as the other scores precision, accuracy and f1 have improved.



```
# Calculate and display precision, recall, accuracy, and F1 scores for Logistic Regression with standard scaling
print('Precision score for Logistic Regression with standard scaling:', precision_score(y_test_standard, y_test_pred
print('Recall score for Logistic Regression with standard scaling:', recall_score(y_test_standard, y_test_pred_logre
print('Accuracy score for Logistic Regression with standard scaling:', accuracy_score(y_test_standard, y_test_pred_l
print('F1 score for Logistic Regression with standard scaling:', f1_score(y_test_standard, y_test_pred_logreg_standa
```

```
Precision score for Logistic Regression with standard scaling: 0.0791268758526603
Recall score for Logistic Regression with standard scaling: 0.7613126079447323
Accuracy score for Logistic Regression with standard scaling: 0.95260014503733
F1 score for Logistic Regression with standard scaling: 0.14335425542294059
```

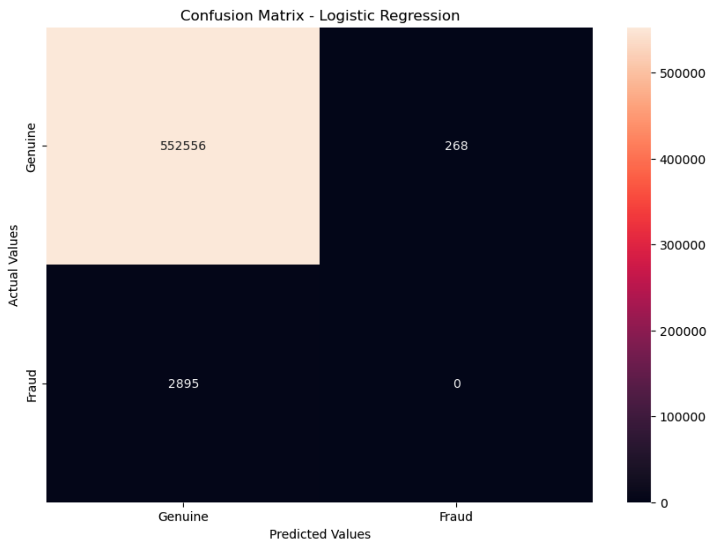


```
# Calculate and display precision, recall, accuracy, and F1 scores for Logistic Regression with min-max scaling
print('Precision score for Logistic Regression with standard scaling:', precision_score(y_test_min_max, y_test_pred_
print('Recall score for Logistic Regression with standard scaling:', recall_score(y_test_min_max, y_test_pred_logreg
print('Accuracy score for Logistic Regression with standard scaling:', accuracy_score(y_test_min_max, y_test_pred_lo
print('F1 score for Logistic Regression with standard scaling:', f1_score(y_test_min_max, y_test_pred_logreg_min_max
```

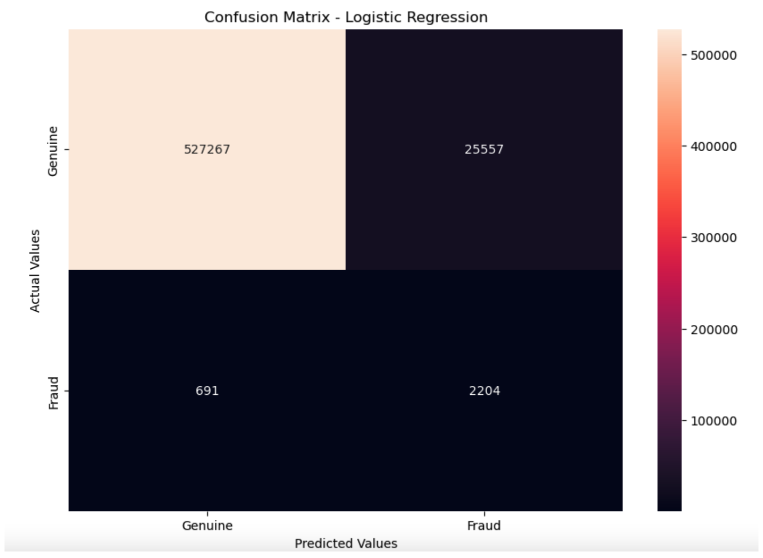
```
Precision score for Logistic Regression with standard scaling: 0.08301318267419962
Recall score for Logistic Regression with standard scaling: 0.7613126079447323
Accuracy score for Logistic Regression with standard scaling: 0.9549466546941889
F1 score for Logistic Regression with standard scaling: 0.14970283579555102
```

Confusion Matrix Comparison for Logistic Regression with standard scale

Without Upsampling



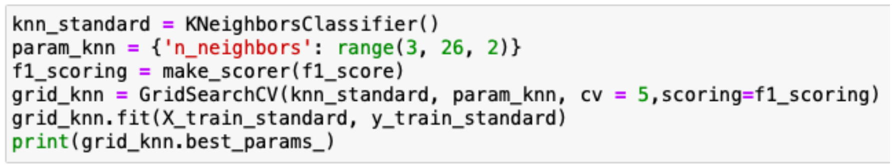
With Upsampling



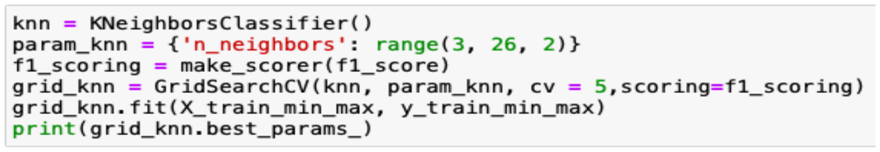
The models further discussed in this project wouldtherefore be trained on the upsampled data.

KNN

The KNN model is implemented and tuned using GridSearchCV to find out the optimal k value. A cross validation value of 5 is being used along with k being all odd numbers between 3 and 25 (included). The scoring parameter for GridSearchCV is set to F1 score. The KNN model is tuned with GridSearchCV for data upsampled using SMOTE and scaled using standardized and min max scaler.



```
knn_standard = KNeighborsClassifier()
param_knn = {'n_neighbors': range(3, 26, 2)}
f1_scoring = make_scorer(f1_score)
grid_knn = GridSearchCV(knn_standard, param_knn, cv = 5,scoring=f1_scoring)
grid_knn.fit(X_train_standard, y_train_standard)
print(grid_knn.best_params_)
```

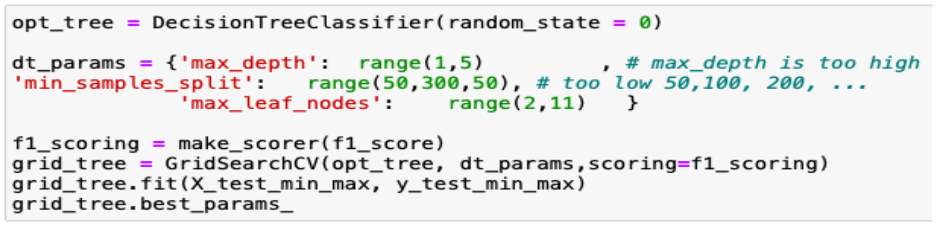


```
knn = KNeighborsClassifier()
param_knn = {'n_neighbors': range(3, 26, 2)}
f1_scoring = make_scorer(f1_score)
grid_knn = GridSearchCV(knn, param_knn, cv = 5,scoring=f1_scoring)
grid_knn.fit(X_train_min_max, y_train_min_max)
print(grid_knn.best_params_)
```

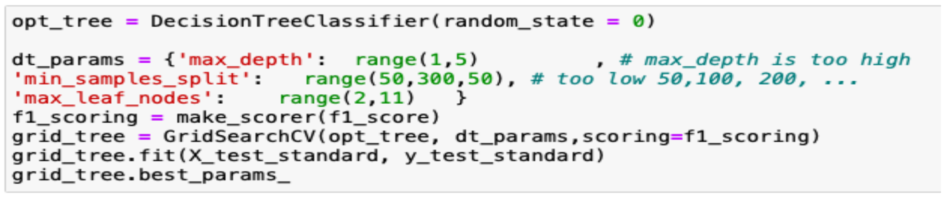
The optimal k value for both the above models is 3 for data normalized using standard scalar and 9 for data normalized using min max scalar.

Decision Tree

The Decision Tree model is implemented and tuned using GridSearchCV to find out the optimal max_depth, min_samples_split and max_leaf_nodes value. A cross validation value of 5 is being used along with max_depth being all numbers between 1 and 5, min_samples_split being all numbers between 50 and 300 with increments of 50, max_leaf_nodes being all numbers between 2 and 11.The scoring parameter for GridSearchCV is set to F1 score. The following are Decision Tree implementations tuned with GridSearchCV for data upsampled using SMOTE and scaled using standardized and min max scaler.



```
opt_tree = DecisionTreeClassifier(random_state = 0)

dt_params = {'max_depth':  range(1,5)          , # max_depth is too high
'min_samples_split':   range(50,300,50), # too low 50,100, 200, ...
              'max_leaf_nodes':     range(2,11)    }

f1_scoring = make_scorer(f1_score)
grid_tree = GridSearchCV(opt_tree, dt_params,scoring=f1_scoring)
grid_tree.fit(X_test_min_max, y_test_min_max)
grid_tree.best_params_
```

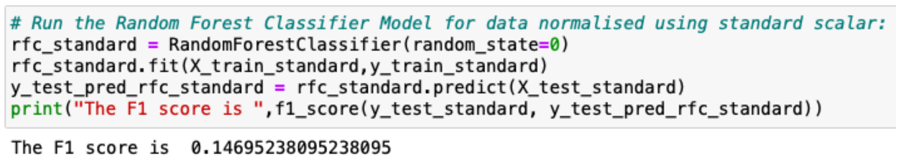


```
opt_tree = DecisionTreeClassifier(random_state = 0)

dt_params = {'max_depth':  range(1,5)          , # max_depth is too high
'min_samples_split':   range(50,300,50), # too low 50,100, 200, ...
'max_leaf_nodes':     range(2,11)    }
f1_scoring = make_scorer(f1_score)
grid_tree = GridSearchCV(opt_tree, dt_params,scoring=f1_scoring)
grid_tree.fit(X_test_standard, y_test_standard)
grid_tree.best_params_
```

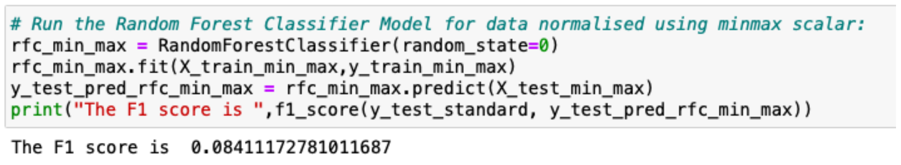
The optimal k value for both the above models are max_depth= 3, max_leaf_nodes= 6, min_samples_split= 50 for both data normalized using standard scalar and using min max scalar.

Random Forest

The following are Random Forest implementations for data upsampled using SMOTE and scaled using standardized and min max scaler.



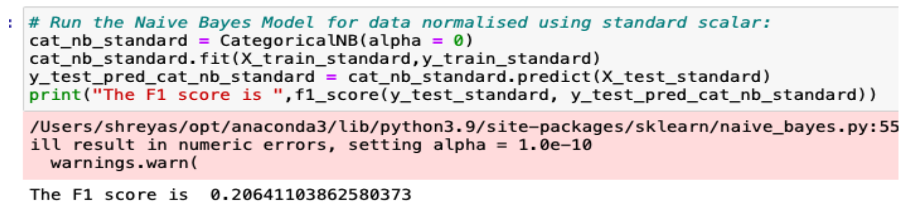
```
# Run the Random Forest Classifier Model for data normalised using standard scalar:
rfc_standard = RandomForestClassifier(random_state=0)
rfc_standard.fit(X_train_standard,y_train_standard)
y_test_pred_rfc_standard = rfc_standard.predict(X_test_standard)
print("The F1 score is ",f1_score(y_test_standard, y_test_pred_rfc_standard))

The F1 score is  0.14695238095238095
```



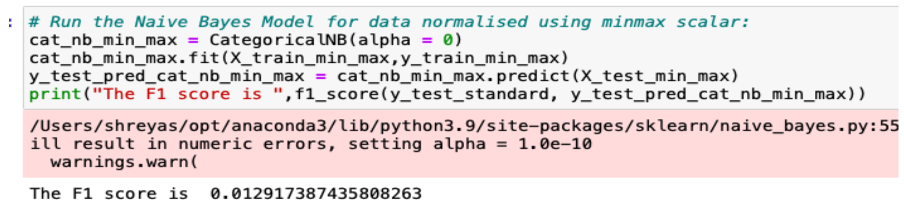
```
# Run the Random Forest Classifier Model for data normalised using minmax scalar:
rfc_min_max = RandomForestClassifier(random_state=0)
rfc_min_max.fit(X_train_min_max,y_train_min_max)
y_test_pred_rfc_min_max = rfc_min_max.predict(X_test_min_max)
print("The F1 score is ",f1_score(y_test_standard, y_test_pred_rfc_min_max))

The F1 score is  0.08411172781011687
```

Naïve Bayes

The following are Naive Bayes implementations for data upsampled using SMOTE and scaled using standardized and min max scaler.



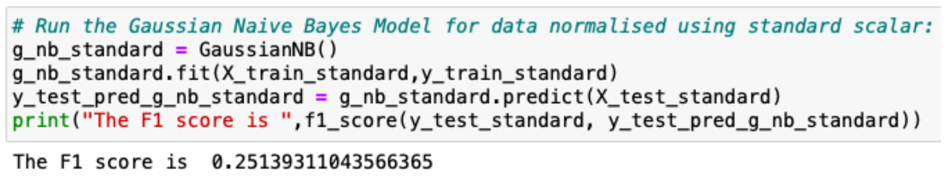
```
# Run the Naive Bayes Model for data normalised using standard scalar:
cat_nb_standard = CategoricalNB(alpha = 0)
cat_nb_standard.fit(X_train_standard,y_train_standard)
y_test_pred_cat_nb_standard = cat_nb_standard.predict(X_test_standard)
print("The F1 score is ",f1_score(y_test_standard, y_test_pred_cat_nb_standard))

/Users/shreyas/opt/anaconda3/lib/python3.9/site-packages/sklearn/naive_bayes.py:55
ill result in numeric errors, setting alpha = 1.0e-10
  warnings.warn(

The F1 score is  0.20641103862580373
```



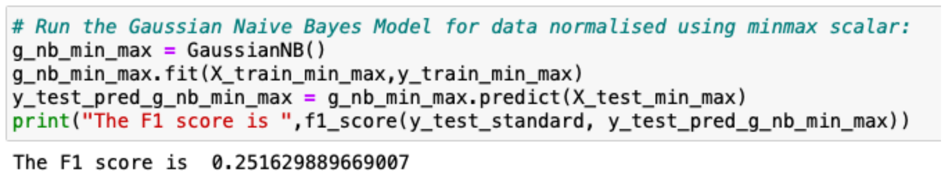
```
# Run the Naive Bayes Model for data normalised using minmax scalar:
cat_nb_min_max = CategoricalNB(alpha = 0)
cat_nb_min_max.fit(X_train_min_max,y_train_min_max)
y_test_pred_cat_nb_min_max = cat_nb_min_max.predict(X_test_min_max)
print("The F1 score is ",f1_score(y_test_standard, y_test_pred_cat_nb_min_max))

/Users/shreyas/opt/anaconda3/lib/python3.9/site-packages/sklearn/naive_bayes.py:55
ill result in numeric errors, setting alpha = 1.0e-10
  warnings.warn(

The F1 score is  0.012917387435808263
```

Gaussian Naïve Bayes

The following are Gaussian Naive Bayes implementations for data upsampled using SMOTE and scaled using standardized and min max scaler.



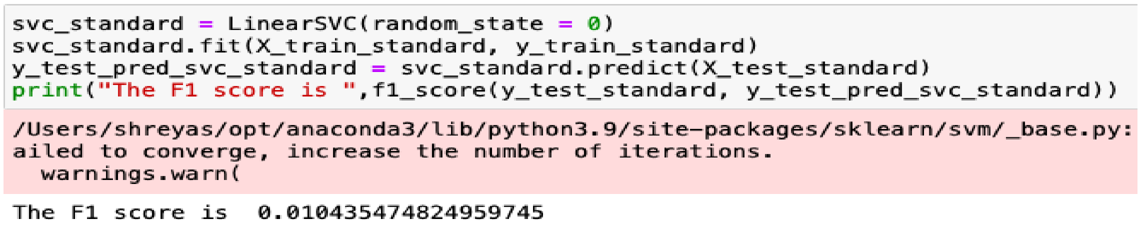
```
# Run the Gaussian Naive Bayes Model for data normalised using standard scalar:
g_nb_standard = GaussianNB()
g_nb_standard.fit(X_train_standard,y_train_standard)
y_test_pred_g_nb_standard = g_nb_standard.predict(X_test_standard)
print("The F1 score is ",f1_score(y_test_standard, y_test_pred_g_nb_standard))

The F1 score is  0.25139311043566365
```



```
# Run the Gaussian Naive Bayes Model for data normalised using minmax scalar:
g_nb_min_max = GaussianNB()
g_nb_min_max.fit(X_train_min_max,y_train_min_max)
y_test_pred_g_nb_min_max = g_nb_min_max.predict(X_test_min_max)
print("The F1 score is ",f1_score(y_test_standard, y_test_pred_g_nb_min_max))

The F1 score is  0.251629889669007
```

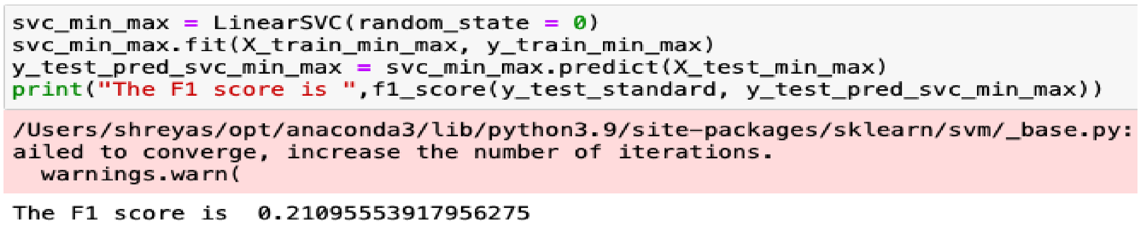
SVM

The following are are SVM implementations for data upsampled using SMOTE and scaled using standardized and min max scaler.



```
svc_standard = LinearSVC(random_state = 0)
svc_standard.fit(X_train_standard, y_train_standard)
y_test_pred_svc_standard = svc_standard.predict(X_test_standard)
print("The F1 score is ",f1_score(y_test_standard, y_test_pred_svc_standard))
```

```
/Users/shreyas/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:
ailed to converge, increase the number of iterations.
  warnings.warn(
The F1 score is  0.010435474824959745
```

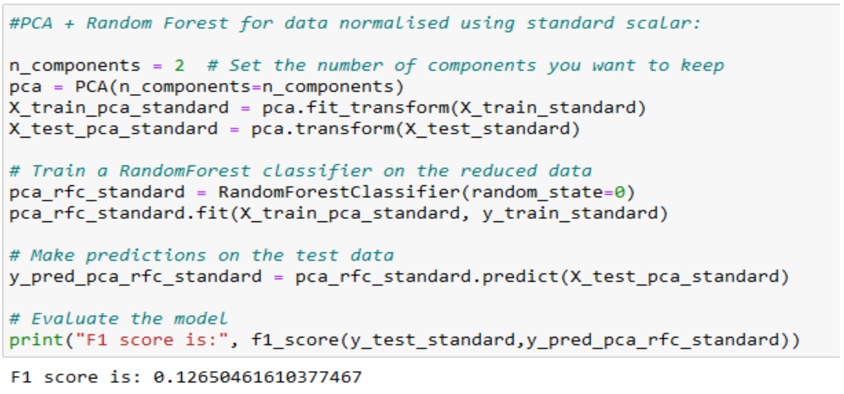


```
svc_min_max = LinearSVC(random_state = 0)
svc_min_max.fit(X_train_min_max, y_train_min_max)
y_test_pred_svc_min_max = svc_min_max.predict(X_test_min_max)
print("The F1 score is ",f1_score(y_test_standard, y_test_pred_svc_min_max))
```

```
/Users/shreyas/opt/anaconda3/lib/python3.9/site-packages/sklearn/svm/_base.py:
ailed to converge, increase the number of iterations.
  warnings.warn(
The F1 score is  0.21095553917956275
```

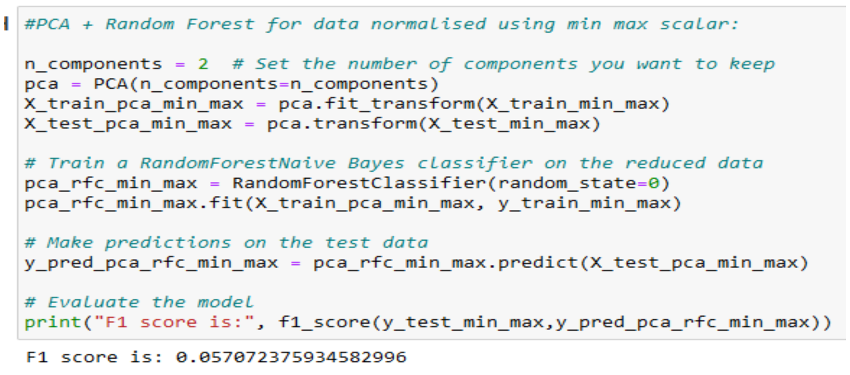
Model Extension:

Besides trying the classification supervised models learned in this class, we decided to try a model extension by combining the supervised and unsupervised machine learning techniques together to see if provided better model performance. For this, we combined the PCA (unsupervised) model with Random Forest (supervised) model using the following implementations and scaled them using both standard and min-max scalar.



```
#PCA + Random Forest for data normalised using standard scalar:

n_components = 2  # Set the number of components you want to keep
pca = PCA(n_components=n_components)
X_train_pca_standard = pca.fit_transform(X_train_standard)
X_test_pca_standard = pca.transform(X_test_standard)

# Train a RandomForest classifier on the reduced data
pca_rfc_standard = RandomForestClassifier(random_state=0)
pca_rfc_standard.fit(X_train_pca_standard, y_train_standard)

# Make predictions on the test data
y_pred_pca_rfc_standard = pca_rfc_standard.predict(X_test_pca_standard)

# Evaluate the model
print("F1 score is:", f1_score(y_test_standard,y_pred_pca_rfc_standard))
```

```
F1 score is: 0.12650461610377467
```



```
#PCA + Random Forest for data normalised using min max scalar:

n_components = 2  # Set the number of components you want to keep
pca = PCA(n_components=n_components)
X_train_pca_min_max = pca.fit_transform(X_train_min_max)
X_test_pca_min_max = pca.transform(X_test_min_max)

# Train a RandomForestNaive Bayes classifier on the reduced data
pca_rfc_min_max = RandomForestClassifier(random_state=0)
pca_rfc_min_max.fit(X_train_pca_min_max, y_train_min_max)

# Make predictions on the test data
y_pred_pca_rfc_min_max = pca_rfc_min_max.predict(X_test_pca_min_max)

# Evaluate the model
print("F1 score is:", f1_score(y_test_min_max,y_pred_pca_rfc_min_max))
```

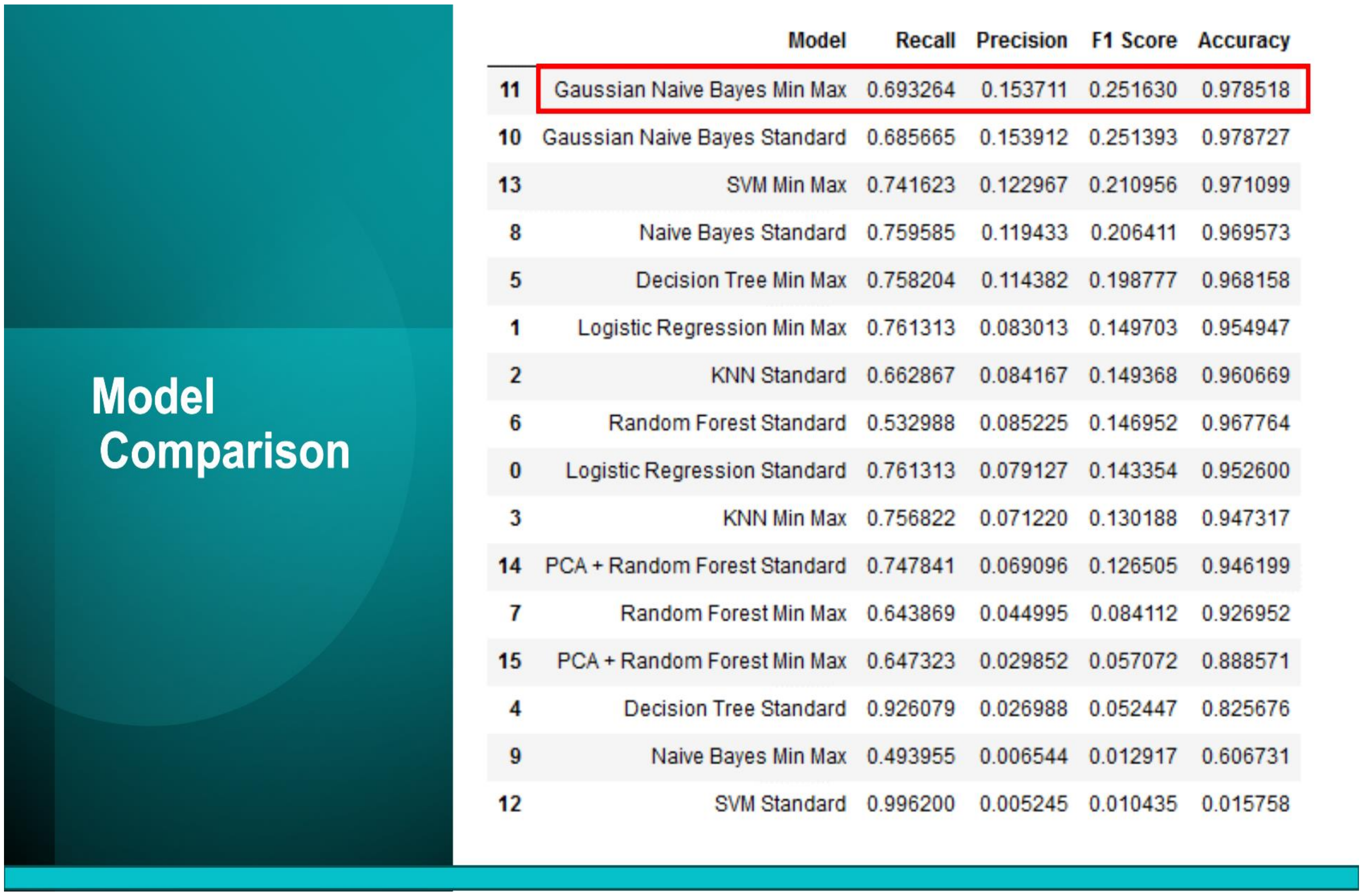
```
F1 score is: 0.057072375934582996
```

Performance Metrics:

In developing the models, we initially incorporated precision, recall, accuracy, and F1 Score as the primary performance metrics. The rationale behind including accuracy lies in its simplicity and its ability to provide an overall measure of model correctness. However, we were mindful of the fact that accuracy may not be the sole determinant of success in fraud detection due to the inherent class imbalance, where fraudulent transactions are often rarely compared to legitimate ones. Precision was selected to minimize false positives, ensuring that legitimate transactions are not falsely flagged as fraudulent, which could inconvenience cardholders. Recall is equally crucial to capture as many fraudulent transactions as possible, mitigating the risk of missing potential threats. The F1 Score, as a harmonic mean of precision and recall, strikes a balance between minimizing both false positives and false negatives, offering a comprehensive evaluation of the model's performance. Hence, each model was tuned with scoring parameter set as F-1 score.

Performance Evaluation:

The performance of different models used in this project including the extensions is as seen below where the Gaussian Naïve Bayes Model with Min Max Scaling performs the best with highest F1 score of 0.251630.



Model Comparison

	Model	Recall	Precision	F1 Score	Accuracy
11	Gaussian Naive Bayes Min Max	0.693264	0.153711	0.251630	0.978518
10	Gaussian Naive Bayes Standard	0.685665	0.153912	0.251393	0.978727
13	SVM Min Max	0.741623	0.122967	0.210956	0.971099
8	Naive Bayes Standard	0.759585	0.119433	0.206411	0.969573
5	Decision Tree Min Max	0.758204	0.114382	0.198777	0.968158
1	Logistic Regression Min Max	0.761313	0.083013	0.149703	0.954947
2	KNN Standard	0.662867	0.084167	0.149368	0.960669
6	Random Forest Standard	0.532988	0.085225	0.146952	0.967764
0	Logistic Regression Standard	0.761313	0.079127	0.143354	0.952600
3	KNN Min Max	0.756822	0.071220	0.130188	0.947317
14	PCA + Random Forest Standard	0.747841	0.069096	0.126505	0.946199
7	Random Forest Min Max	0.643869	0.044995	0.084112	0.926952
15	PCA + Random Forest Min Max	0.647323	0.029852	0.057072	0.888571
4	Decision Tree Standard	0.926079	0.026988	0.052447	0.825676
9	Naive Bayes Min Max	0.493955	0.006544	0.012917	0.606731
12	SVM Standard	0.996200	0.005245	0.010435	0.015758

IV: Conclusion:

Results:

Various models learned in this class were tried to determine the best model for our case that included major supervised models as well as a model extension with a combination of supervised and unsupervised model. Gaussian Naïve model with min-max scaler performed the best in our case with the highest F1 score of 0.251630 suggests a balance between precision and recall as both false positives and false negatives are important in our case because if an algorithm detected no fraud when there is an actual fraud that will create most loss, similarly if it detects fraud when it is not fraud is wastage of resources.

Insights:

The following insights were derived from our exploratory data analysis:

1. Age Bracket and Fraud: People in their fifties are most subject to fraud
2. Credit Card and Fraud: American Express cards are associated with the highest number of fraud cases.
3. Gender Disparities in Fraud: Females experience slightly lower total losses ($2,402,746.68) compared to males ($2,718,666.71). Males account for over 53% of the total amount lost.
4. Population Dynamics: Over 70% of fraudulent cases occur in areas with a population less than 10,000.12% of fraud instances are reported in cities with populations ranging from 10,000 to 50,000 indicating cities with lesser population are more subject to credit card fraud.
5. Cities and Fraud: Cities such as Dallas and Houston, experience a significant number of fraud instances (39 and 36 instances, respectively). New York City records the highest number of fraud instances (35 instances). These insights provide valuable context for understanding the demographics, credit card affiliations, and geographic locations associated with credit card fraud.

Applications:

Besides detecting credit card fraud, the machine learning algorithms can be further expanded to address broader applications such as financial transactions, AI-powered fraud detection, and the implementation of robust cybersecurity protocols.

Financial Transactions

Gaussian Naive Bayes with Min-max scaling can be expanded to detect debit card frauds and other banking frauds. Some of the techniques used for this would be customer profiling using profiles of typical customer behavior to identity deviations. Another technique used would be tokenization where sensitive card information is replaced with unique identifier token to reduce risk of data breaches.

Cybersecurity Protocols

Here Robust cybersecurity protocols are implemented to protect against data breaches and unauthorized access. Some techniques used for implementation are data encryption where sensitive data is encrypted both in transit and at rest and it remains secure and unreadable without proper decryption keys. Another technique used is biometric authentication where fingerprint or facial recognition are used for added layer of security. One more technique used is geolocation tracking where transaction location is compared with cardholder's usual locations and a farther location would trigger alert.

Artificial Intelligence

Al powered fraud detection techniques could be used for advanced fraud detection, leveraging Natural Language Processing and Deep Learning. Blockchain technology can also be implemented where blockchain is used for secure and transparent transaction records.

Limitations:

While our models were tuned setting F-1 score as scoring parameter using both standard and min-max scaling, if equipped with higher performing systems, we would have been able to prune the decision tree model even further to enhance the model performance. With the current system, even with using google colab and running it over a period of over two days, the model was still running and not completed. So, we were limited from pruning the decision tree model even further.

Another aspect that could be done differently in our project was not excluding variables even if they did not perform well during explanatory data analysis but could have logical meaning. For example, understanding the distance between the customer’s residential location and the zipcode where the card was used. If they were far, it should trigger an alert for a fraud transaction. While we were limited this time due to time and system constraints, it could be a future scope where the model can be further enhanced.

Conclusion:

In conclusion, after evaluating various models for credit card fraud detection, Gaussian Naïve Bayes with min-max scaling stands out as the optimal choice. The F-1 score proves to be a more effective performance measure, accounting for both false negatives and false positives in this context. Utilizing the up-sampling technique with SMOTE yields significantly improved results. While combining supervised and unsupervised techniques provides an extension, it doesn't enhance the overall model performance. In the future there is a scope for expanding the machine learning algorithms to address broader applications such as financial transactions, AI-powered fraud detection, and the implementation of robust cybersecurity protocols.

APPENDIX

Appendix – More Performance Evaluation besides F-1 Score

KNN

The confusion matrix and performance metrics for KNN models with standard and minmax scaling are as follows:

```python
# Calculate and display the confusion matrix for KNN with standard scaling
confusion_matrix(y_test_standard,y_test_pred_knn_standard)
```

```
array([[531943,  20881],
       [   976,   1919]])
```

```python
# Calculate and display precision, recall, accuracy, and F1 scores for KNN with standard scaling
print('Precision score for KNN with standard scaling:', precision_score(y_test_standard,y_test_pred_knn_standard))
print('Recall score for KNN with standard scaling:', recall_score(y_test_standard,y_test_pred_knn_standard))
print('Accuracy score for KNN with standard scaling:', accuracy_score(y_test_standard,y_test_pred_knn_standard))
print('F1 score for KNN with standard scaling:', f1_score(y_test_standard,y_test_pred_knn_standard))
```

```
Precision score for KNN with standard scaling: 0.08416666666666667
Recall score for KNN with standard scaling: 0.66286701208981
Accuracy score for KNN with standard scaling: 0.9606689711886763
F1 score for KNN with standard scaling: 0.14936758124148666
```

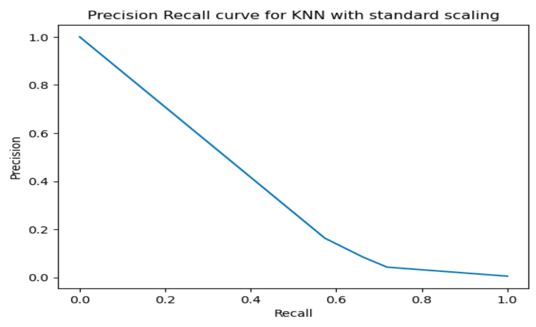
```python
confusion_matrix(y_test_min_max,y_test_pred_knn_min_max)
```

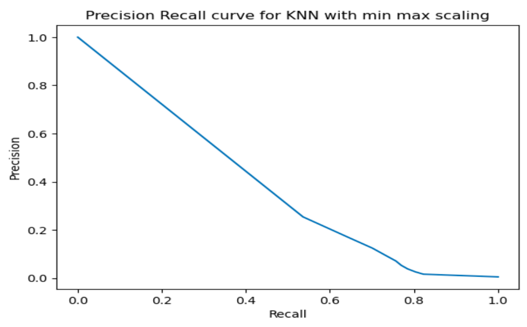
```
array([[524251,  28573],
       [   704,   2191]])
```

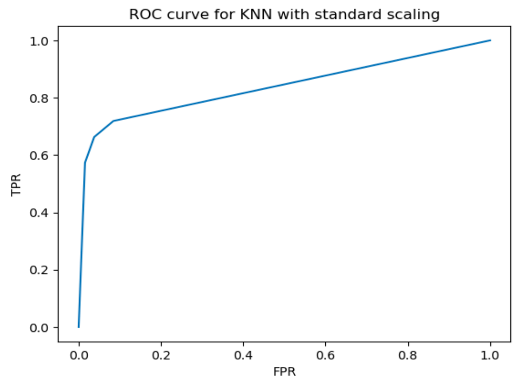
```python
print('Precision score with minmax scaling:', precision_score(y_test_min_max,y_test_pred_knn_min_max))
print('Recall score with minmax scaling:', recall_score(y_test_min_max,y_test_pred_knn_min_max))
print('Accuracy score with minmax scaling:', accuracy_score(y_test_min_max,y_test_pred_knn_min_max))
print('F1 score with minmax scaling:', f1_score(y_test_min_max,y_test_pred_knn_min_max))
```

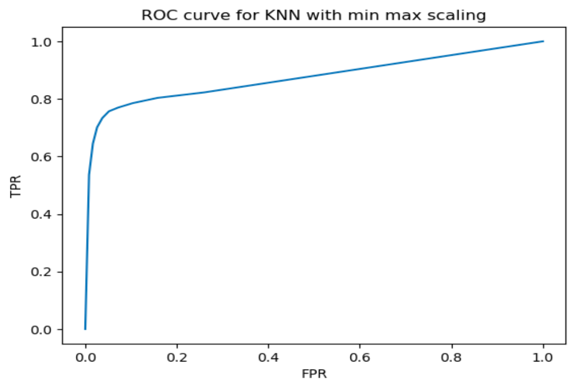
```
Precision score with minmax scaling: 0.07121960733324666
Recall score with minmax scaling: 0.7568221070811745
Accuracy score with minmax scaling: 0.9473168993682058
F1 score with minmax scaling: 0.13018806262812324
```

The Precision Recall and AUC curves of the 2 models are as follows:



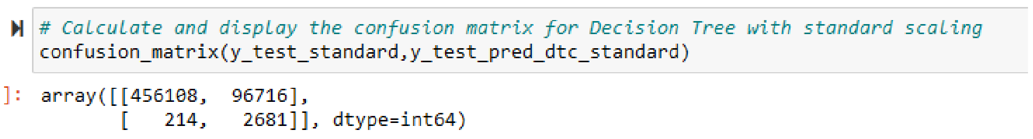






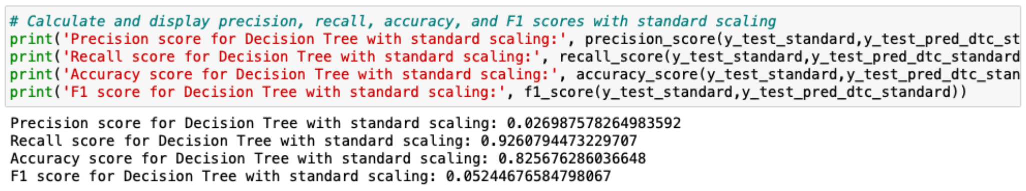
Decision Tree

The confusion matrix and performance metrics for Decision Tree models with standard and minmax scaling are as follows:



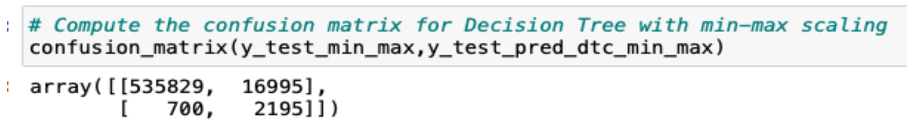
```
# Calculate and display the confusion matrix for Decision Tree with standard scaling
confusion_matrix(y_test_standard,y_test_pred_dtc_standard)
```

```
]: array([[456108,  96716],
          [   214,   2681]], dtype=int64)
```



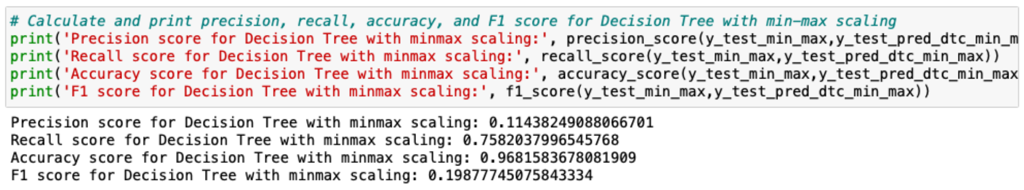
```
# Calculate and display precision, recall, accuracy, and F1 scores with standard scaling
print('Precision score for Decision Tree with standard scaling:', precision_score(y_test_standard,y_test_pred_dtc_st
print('Recall score for Decision Tree with standard scaling:', recall_score(y_test_standard,y_test_pred_dtc_standard
print('Accuracy score for Decision Tree with standard scaling:', accuracy_score(y_test_standard,y_test_pred_dtc_stan
print('F1 score for Decision Tree with standard scaling:', f1_score(y_test_standard,y_test_pred_dtc_standard))
```

```
Precision score for Decision Tree with standard scaling: 0.026987578264983592
Recall score for Decision Tree with standard scaling: 0.9260794473229707
Accuracy score for Decision Tree with standard scaling: 0.825676286036648
F1 score for Decision Tree with standard scaling: 0.05244676584798067
```



```
# Compute the confusion matrix for Decision Tree with min-max scaling
confusion_matrix(y_test_min_max,y_test_pred_dtc_min_max)
```

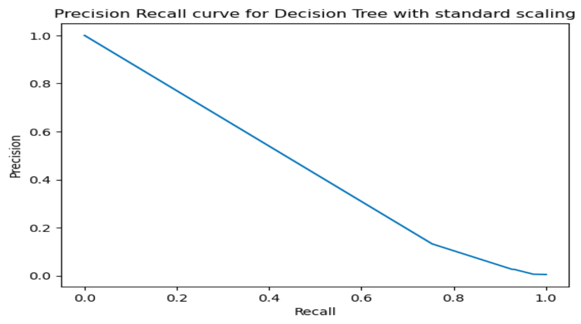
```
array([[535829,  16995],
       [   700,   2195]])
```

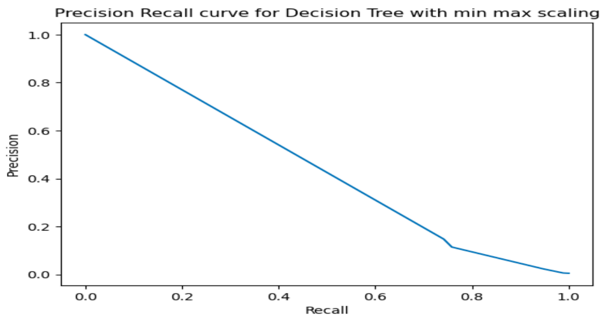


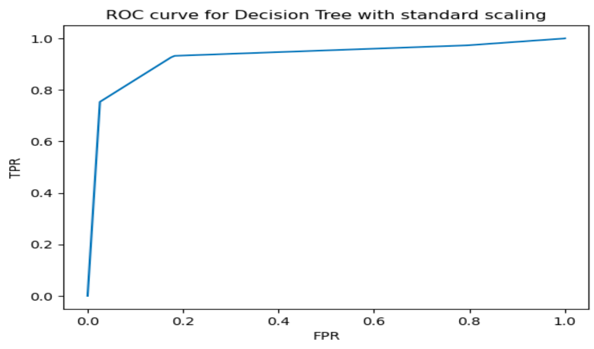
```
# Calculate and print precision, recall, accuracy, and F1 score for Decision Tree with min-max scaling
print('Precision score for Decision Tree with minmax scaling:', precision_score(y_test_min_max,y_test_pred_dtc_min_m
print('Recall score for Decision Tree with minmax scaling:', recall_score(y_test_min_max,y_test_pred_dtc_min_max))
print('Accuracy score for Decision Tree with minmax scaling:', accuracy_score(y_test_min_max,y_test_pred_dtc_min_max
print('F1 score for Decision Tree with minmax scaling:', f1_score(y_test_min_max,y_test_pred_dtc_min_max))
```

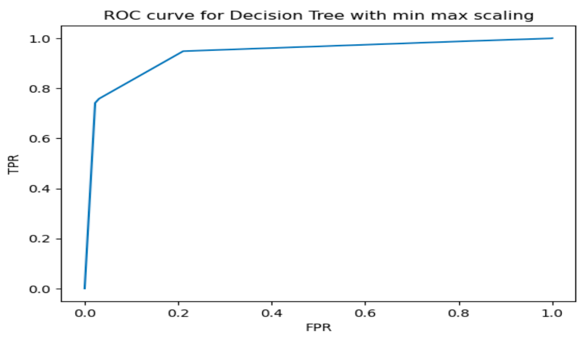
```
Precision score for Decision Tree with minmax scaling: 0.11438249088066701
Recall score for Decision Tree with minmax scaling: 0.7582037996545768
Accuracy score for Decision Tree with minmax scaling: 0.9681583678081909
F1 score for Decision Tree with minmax scaling: 0.19877745075843334
```

The Precision Recall and AUC curves of the 2 models are as follows:









Random Forest

The confusion matrix and performance metrics for Random Forest models with standard and minmax scaling are as follows:

```
# Calculate and display the confusion matrix for Random Forest with standard scaling
confusion_matrix(y_test_standard,y_test_pred_rfc_standard)
```

```
array([[536262,  16562],
       [  1352,   1543]])
```

```
# Calculate and display precision, recall, accuracy, and F1 scores with standard scaling
print('Precision score for Random Forest with standard scaling:', precision_score(y_test_standard,y_test_pred_rfc_st
print('Recall score for Random Forest with standard scaling:', recall_score(y_test_standard,y_test_pred_rfc_standard
print('Accuracy score for Random Forest with standard scaling:', accuracy_score(y_test_standard,y_test_pred_rfc_stan
print('F1 score for Random Forest with standard scaling:', f1_score(y_test_standard,y_test_pred_rfc_standard))
```

```
Precision score for Random Forest with standard scaling: 0.08522507594587131
Recall score for Random Forest with standard scaling: 0.5329879101899827
Accuracy score for Random Forest with standard scaling: 0.9677642837477214
F1 score for Random Forest with standard scaling: 0.14695238095238095
```

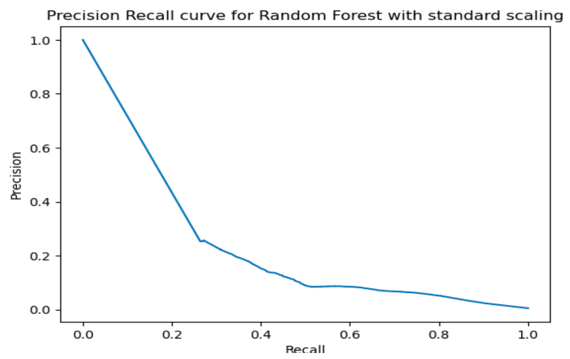
```
# Compute the confusion matrix for Random Forest with min-max scaling
confusion_matrix(y_test_min_max,y_test_pred_rfc_min_max)
```

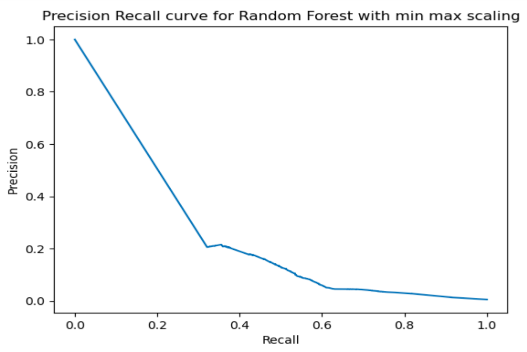
```
array([[513261,  39563],
       [  1031,   1864]])
```

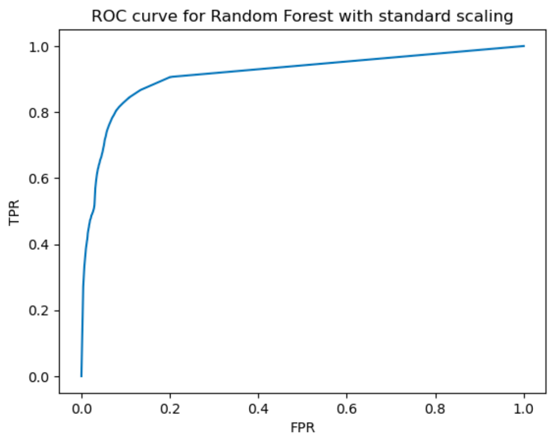
```
# Calculate and print precision, recall, accuracy, and F1 score for Random Forest with min-max scaling
print('Precision score for Random Forest with minmax scaling:', precision_score(y_test_min_max,y_test_pred_rfc_min_m
print('Recall score for Random Forest with minmax scaling:', recall_score(y_test_min_max,y_test_pred_rfc_min_max))
print('Accuracy score for Random Forest with minmax scaling:', accuracy_score(y_test_min_max,y_test_pred_rfc_min_max
print('F1 score for Random Forest with minmax scaling:', f1_score(y_test_min_max,y_test_pred_rfc_min_max))
```

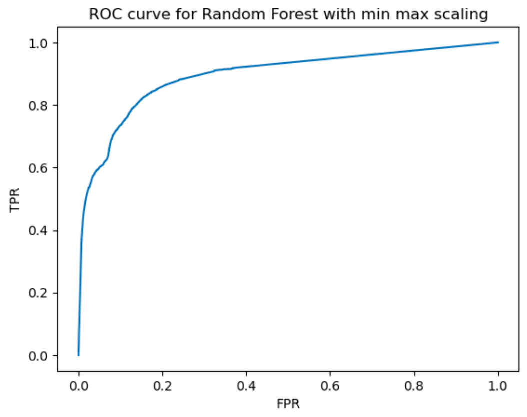
```
Precision score for Random Forest with minmax scaling: 0.04499481014797113
Recall score for Random Forest with minmax scaling: 0.6438687392055268
Accuracy score for Random Forest with minmax scaling: 0.9269522906360949
F1 score for Random Forest with minmax scaling: 0.08411172781011687
```

The Precision Recall and AUC curves of the 2 models are as follows:









Naïve Bayes

The confusion matrix and performance metrics for Naïve Bayes models with standard and minmax scaling are as follows:

```
# Calculate and display the confusion matrix for Naive Bayes with standard scaling
confusion_matrix(y_test_standard,y_test_pred_cat_nb_standard)
```

```
array([[536611,  16213],
       [   696,   2199]])
```

```
# Calculate and display precision, recall, accuracy, and F1 scores with standard scaling
print('Precision score for Naive Bayes with standard scaling:', precision_score(y_test_standard,y_test_pred_cat_nb_s
print('Recall score for Naive Bayes with standard scaling:', recall_score(y_test_standard,y_test_pred_cat_nb_standar
print('Accuracy score for Naive Bayes with standard scaling:', accuracy_score(y_test_standard,y_test_pred_cat_nb_sta
print('F1 score for Naive Bayes with standard scaling:', f1_score(y_test_standard,y_test_pred_cat_nb_standard))
```

```
Precision score for Naive Bayes with standard scaling: 0.11943297849228764
Recall score for Naive Bayes with standard scaling: 0.7595854922279792
Accuracy score for Naive Bayes with standard scaling: 0.9695727516964509
F1 score for Naive Bayes with standard scaling: 0.20641103862580373
```

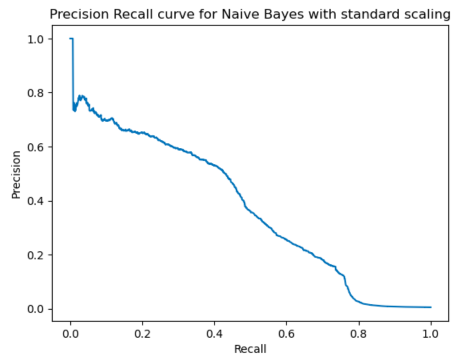
```
# Calculate and display the confusion matrix for Naive Bayes with min max scaling
confusion_matrix(y_test_min_max,y_test_pred_cat_nb_min_max)
```

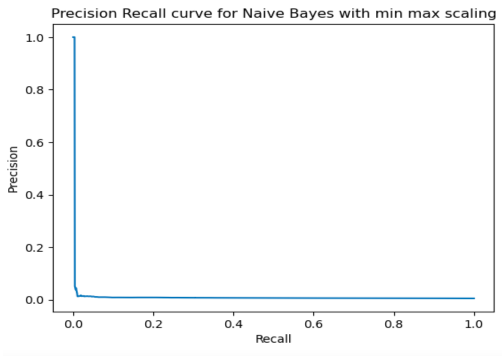
```
array([[335742, 217082],
       [  1465,   1430]])
```

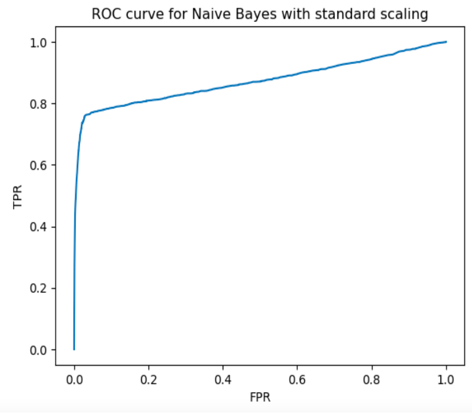
```
# Calculate and display precision, recall, accuracy, and F1 scores with min max scaling
print('Precision score for Naive Bayes with min max scaling:', precision_score(y_test_min_max,y_test_pred_cat_nb_min
print('Recall score for Naive Bayes with min max scaling:', recall_score(y_test_min_max,y_test_pred_cat_nb_min_max))
print('Accuracy score for Naive Bayes with min max scaling:', accuracy_score(y_test_min_max,y_test_pred_cat_nb_min_m
print('F1 score for Naive Bayes with min max scaling:', f1_score(y_test_min_max,y_test_pred_cat_nb_min_max))
```

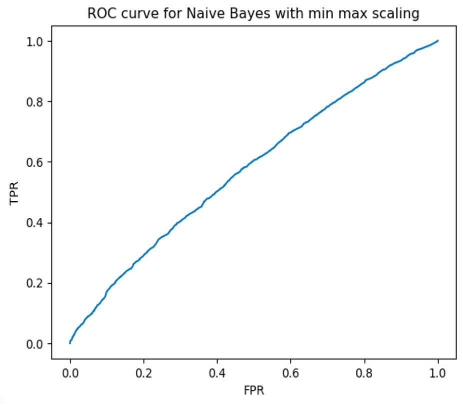
```
Precision score for Naive Bayes with min max scaling: 0.006544263015303507
Recall score for Naive Bayes with min max scaling: 0.49395509499913644
Accuracy score for Naive Bayes with min max scaling: 0.6067310997104651
F1 score for Naive Bayes with min max scaling: 0.012917387435808263
```

The Precision Recall and AUC curves of the 2 models are as follows:



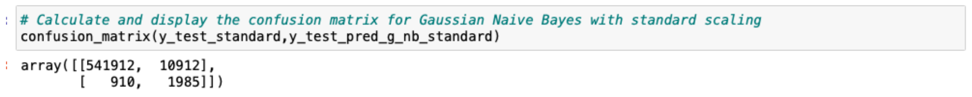






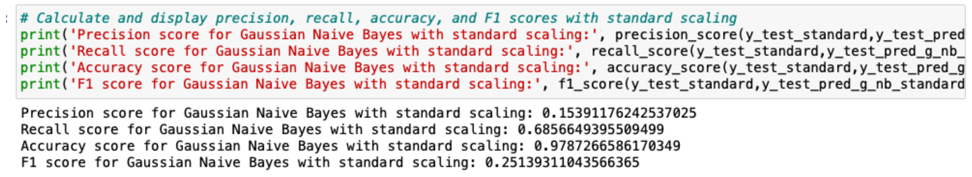
Gaussian Naïve Bayes

The confusion matrix and performance metrics for Gaussian Naïve Bayes models with standard and minmax scaling are as follows:



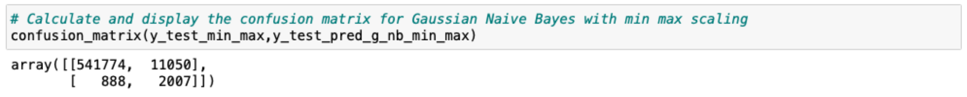
```
# Calculate and display the confusion matrix for Gaussian Naive Bayes with standard scaling
confusion_matrix(y_test_standard,y_test_pred_g_nb_standard)
```

```
array([[541912,  10912],
       [   910,   1985]])
```



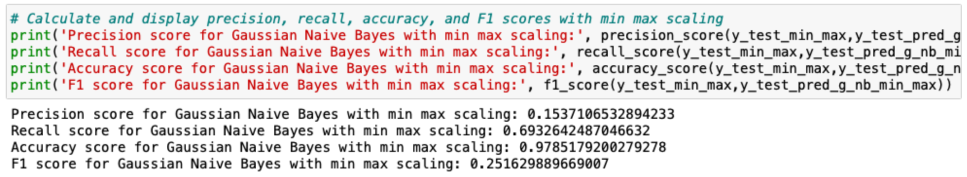
```
# Calculate and display precision, recall, accuracy, and F1 scores with standard scaling
print('Precision score for Gaussian Naive Bayes with standard scaling:', precision_score(y_test_standard,y_test_pred
print('Recall score for Gaussian Naive Bayes with standard scaling:', recall_score(y_test_standard,y_test_pred_g_nb_
print('Accuracy score for Gaussian Naive Bayes with standard scaling:', accuracy_score(y_test_standard,y_test_pred_g
print('F1 score for Gaussian Naive Bayes with standard scaling:', f1_score(y_test_standard,y_test_pred_g_nb_standard
```

```
Precision score for Gaussian Naive Bayes with standard scaling: 0.15391176242537025
Recall score for Gaussian Naive Bayes with standard scaling: 0.6856649395509499
Accuracy score for Gaussian Naive Bayes with standard scaling: 0.9787266586170349
F1 score for Gaussian Naive Bayes with standard scaling: 0.25139311043566365
```



```
# Calculate and display the confusion matrix for Gaussian Naive Bayes with min max scaling
confusion_matrix(y_test_min_max,y_test_pred_g_nb_min_max)
```

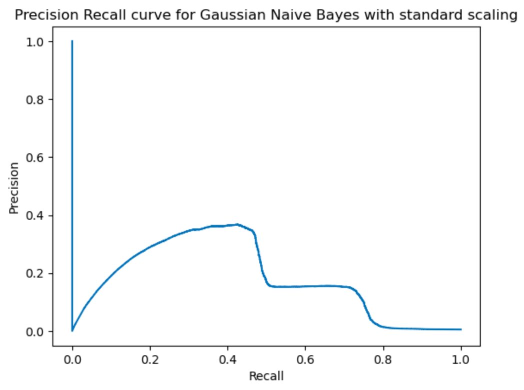
```
array([[541774,  11050],
       [   888,   2007]])
```

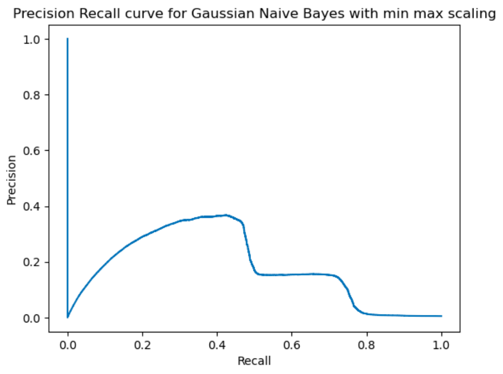


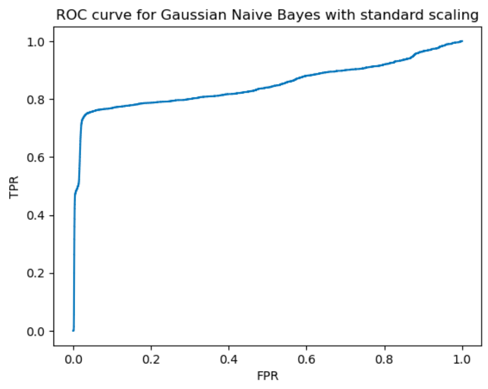
```
# Calculate and display precision, recall, accuracy, and F1 scores with min max scaling
print('Precision score for Gaussian Naive Bayes with min max scaling:', precision_score(y_test_min_max,y_test_pred_g
print('Recall score for Gaussian Naive Bayes with min max scaling:', recall_score(y_test_min_max,y_test_pred_g_nb_mi
print('Accuracy score for Gaussian Naive Bayes with min max scaling:', accuracy_score(y_test_min_max,y_test_pred_g_n
print('F1 score for Gaussian Naive Bayes with min max scaling:', f1_score(y_test_min_max,y_test_pred_g_nb_min_max))
```

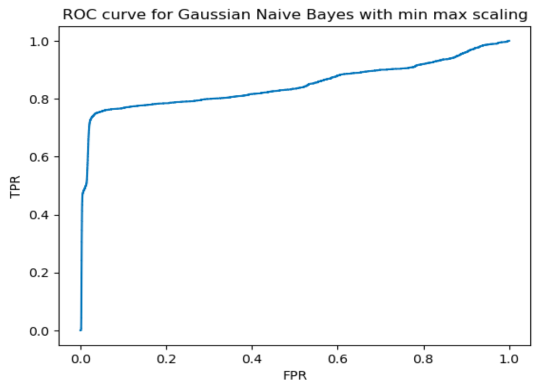
```
Precision score for Gaussian Naive Bayes with min max scaling: 0.1537106532894233
Recall score for Gaussian Naive Bayes with min max scaling: 0.6932642487046632
Accuracy score for Gaussian Naive Bayes with min max scaling: 0.9785179200279278
F1 score for Gaussian Naive Bayes with min max scaling: 0.251629889669007
```

The Precision Recall and AUC curves of the 2 models are as follows:



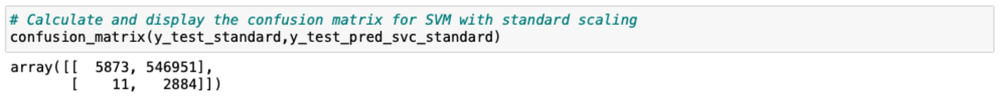




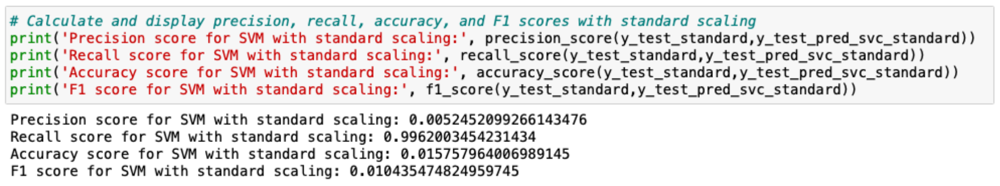


SVM

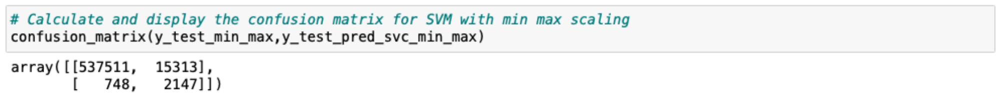
The confusion matrix and performance metrics for SVM models with standard and minmax scaling are as follows:



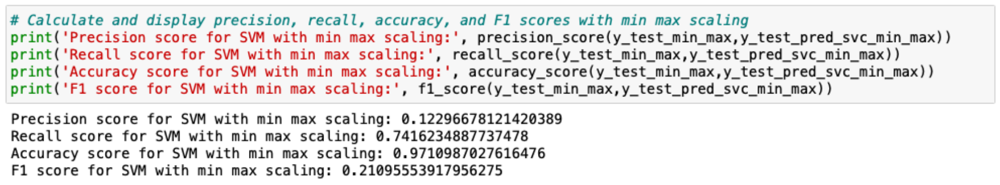
```
# Calculate and display the confusion matrix for SVM with standard scaling
confusion_matrix(y_test_standard,y_test_pred_svc_standard)

array([[  5873, 546951],
       [    11,   2884]])
```



```
# Calculate and display precision, recall, accuracy, and F1 scores with standard scaling
print('Precision score for SVM with standard scaling:', precision_score(y_test_standard,y_test_pred_svc_standard))
print('Recall score for SVM with standard scaling:', recall_score(y_test_standard,y_test_pred_svc_standard))
print('Accuracy score for SVM with standard scaling:', accuracy_score(y_test_standard,y_test_pred_svc_standard))
print('F1 score for SVM with standard scaling:', f1_score(y_test_standard,y_test_pred_svc_standard))

Precision score for SVM with standard scaling: 0.0052452099266143476
Recall score for SVM with standard scaling: 0.9962003454231434
Accuracy score for SVM with standard scaling: 0.015757964006989145
F1 score for SVM with standard scaling: 0.010435474824959745
```



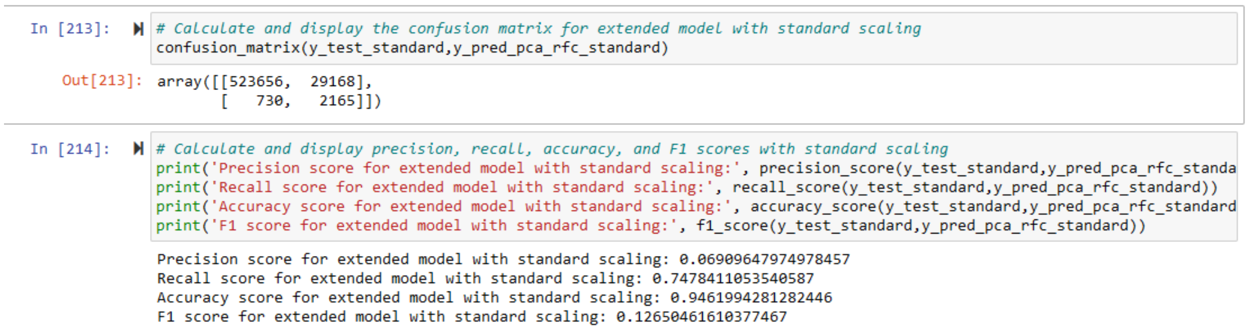
```
# Calculate and display the confusion matrix for SVM with min max scaling
confusion_matrix(y_test_min_max,y_test_pred_svc_min_max)

array([[537511,  15313],
       [   748,   2147]])
```



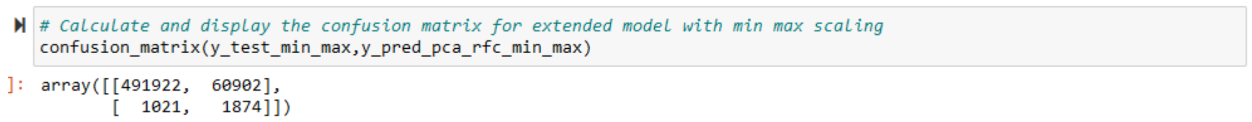
```
# Calculate and display precision, recall, accuracy, and F1 scores with min max scaling
print('Precision score for SVM with min max scaling:', precision_score(y_test_min_max,y_test_pred_svc_min_max))
print('Recall score for SVM with min max scaling:', recall_score(y_test_min_max,y_test_pred_svc_min_max))
print('Accuracy score for SVM with min max scaling:', accuracy_score(y_test_min_max,y_test_pred_svc_min_max))
print('F1 score for SVM with min max scaling:', f1_score(y_test_min_max,y_test_pred_svc_min_max))

Precision score for SVM with min max scaling: 0.12296678121420389
Recall score for SVM with min max scaling: 0.7416234887737478
Accuracy score for SVM with min max scaling: 0.9710987027616476
F1 score for SVM with min max scaling: 0.21095553917956275
```

Extension - Unsupervised + Supervised

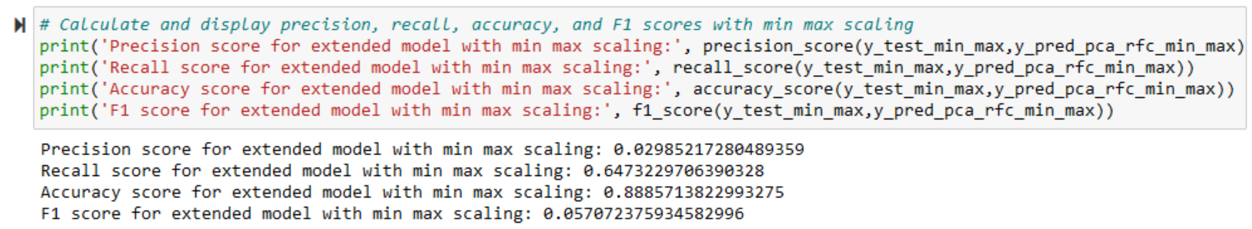
The confusion matrix and performance metrics for the extension using PCA and Random Forest models together with standard and minmax scaling are as follows:



```
In [213]:  # Calculate and display the confusion matrix for extended model with standard scaling
           confusion_matrix(y_test_standard,y_pred_pca_rfc_standard)

Out[213]:  array([[523656,  29168],
                  [   730,   2165]])

In [214]:  # Calculate and display precision, recall, accuracy, and F1 scores with standard scaling
           print('Precision score for extended model with standard scaling:', precision_score(y_test_standard,y_pred_pca_rfc_standa
           print('Recall score for extended model with standard scaling:', recall_score(y_test_standard,y_pred_pca_rfc_standard))
           print('Accuracy score for extended model with standard scaling:', accuracy_score(y_test_standard,y_pred_pca_rfc_standard
           print('F1 score for extended model with standard scaling:', f1_score(y_test_standard,y_pred_pca_rfc_standard))

           Precision score for extended model with standard scaling: 0.06909647974978457
           Recall score for extended model with standard scaling: 0.7478411053540587
           Accuracy score for extended model with standard scaling: 0.9461994281282446
           F1 score for extended model with standard scaling: 0.12650461610377467
```

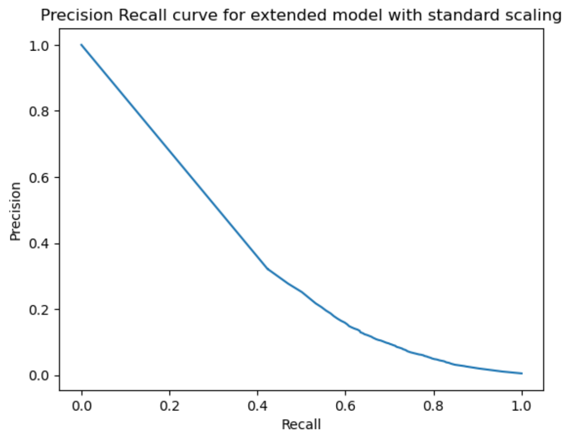


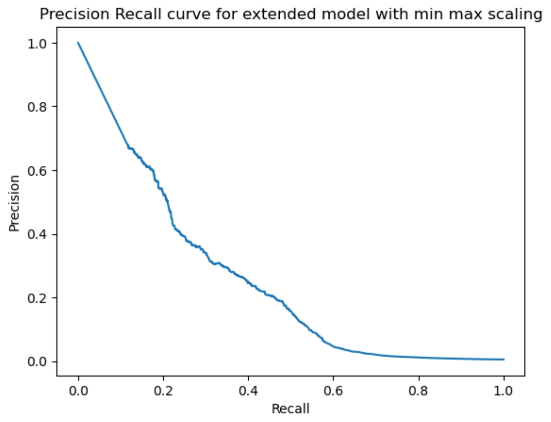
```
# Calculate and display the confusion matrix for extended model with min max scaling
confusion_matrix(y_test_min_max,y_pred_pca_rfc_min_max)

]: array([[491922,  60902],
          [  1021,   1874]])
```

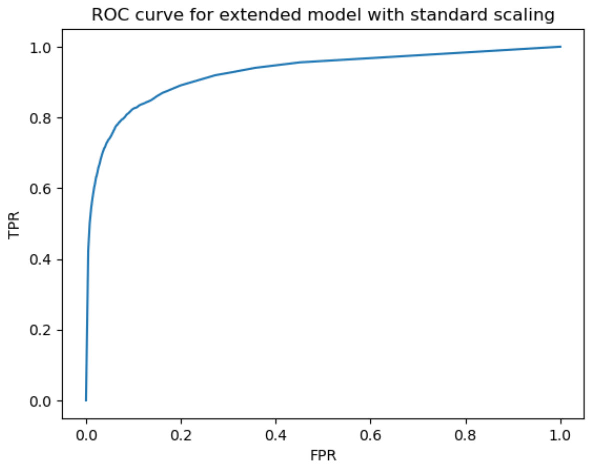


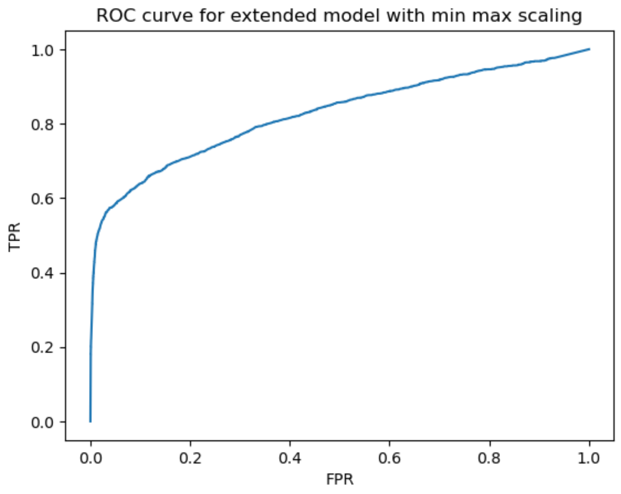
```
# Calculate and display precision, recall, accuracy, and F1 scores with min max scaling
print('Precision score for extended model with min max scaling:', precision_score(y_test_min_max,y_pred_pca_rfc_min_max)
print('Recall score for extended model with min max scaling:', recall_score(y_test_min_max,y_pred_pca_rfc_min_max))
print('Accuracy score for extended model with min max scaling:', accuracy_score(y_test_min_max,y_pred_pca_rfc_min_max))
print('F1 score for extended model with min max scaling:', f1_score(y_test_min_max,y_pred_pca_rfc_min_max))

Precision score for extended model with min max scaling: 0.02985217280489359
Recall score for extended model with min max scaling: 0.6473229706390328
Accuracy score for extended model with min max scaling: 0.8885713822993275
F1 score for extended model with min max scaling: 0.057072375934582996
```

The Precision Recall and AUC curves of the 2 models are as follows:









References:

1. “SMOTE (Synthetic Minority Oversampling Technique) by Genesis, June 26, 2018

https://www.fromthegenesis.com/smote-synthetic-minority-oversampling-technique/