Data exploration:

We started by seeing how many null values we had for each feature. Features that had a high percentage of nulls we decided to drop such as Own_House_Age, Client Occupation, Social Circle Default, and all 3 Score_Source. We also dropped ID and other identifying features as they do not relate to our task of classification.

To determine which features we want to use to build our model, we calculated the correlation between the different features and our target variable. Compared to Assignment 1, these correlation values were much lower, so we selected a threshold of greater than abs(0.02) to select our features.

After selecting our features, we split the data into a training and testing set with a 80/20 split. Originally, the dataset came with separate train and test datasets, however, since this data was used for a kaggle competition, the testing dataset did not include a Default column. Thus, we only looked at the training set and split this data into training and testing so we would be able to evaluate our models. Following this, we standardized the data as we had a large number of features and each feature has a different range. Thus, standardizing the data prevents any one feature from disproportionately affecting the model. After splitting the data and scaling, we somehow ended up with some more null values in our data set, so we removed those as well.

We also checked the distribution of "1"s and "0"s in the default columns and found that there was an overwhelming number of "0"s compared to "1"s. We initially decided to leave it as is, however the models performed quite poorly in predicting default = 1 which we suspected was due to the over representation of 0s in the data. We decided to implement the oversampling technique, SMOTE, to help with this issue.

Model 1 - Plain Decision Tree:

For the Decision Tree model, the following hyperparameters were tested:

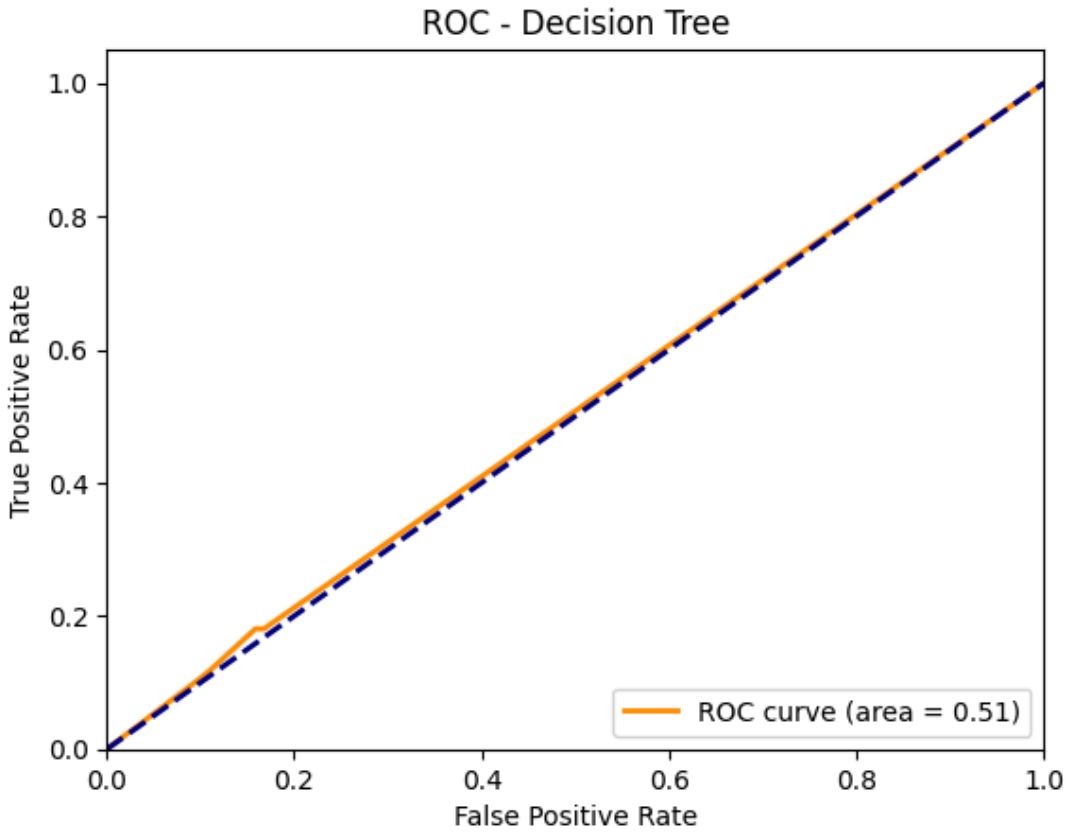
```
'max_depth': [10, 20, 50],
'min_samples_split': [2, 5, 10, 15],
'min_samples_leaf': [1, 2, 5],
'max_features': [None, 'sqrt', 'log2']
```

Per the Grid Search Algorithm, the parameters that created the best model included:

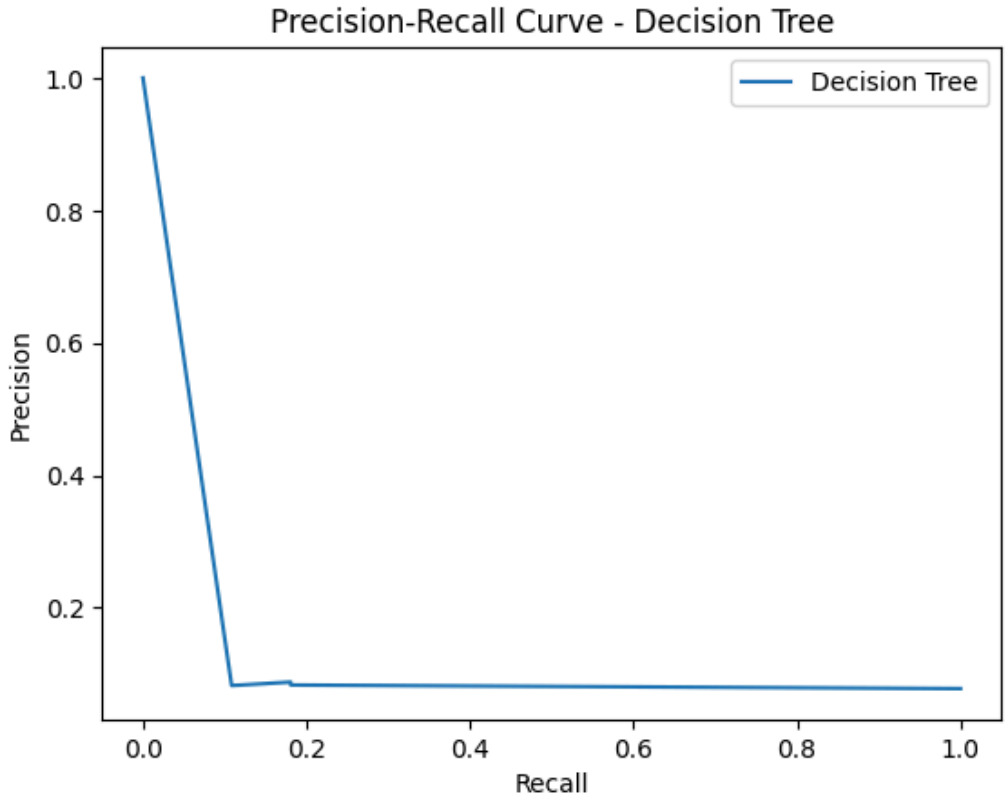
```
'max_depth': 50,
'max_features': None,
'min_samples_leaf': 2,
'min_samples_split': 2
```

	precision	recall	f1-score	support
0	0.92	0.89	0.91	983
1	0.08	0.12	0.10	83
accuracy			0.83	1066
macro avg	0.50	0.50	0.50	1066
weighted avg	0.86	0.83	0.84	1066

The classification report shows that even with using an oversampling technique to balance the default class, there still is an imbalance between them. The model has an accuracy of 0.83, however this is



misleading as the model is unable to accurately predict the "1" class.



The ROC curve and precision-recall curve further illustrates that the model is performing poorly. The area under the ROC curve is marginally performing better than randomly guessing. Perhaps more complex ensemble methods will help to address this in our future models.

** This is a visualization of the tree produced by the Decision Tree model. Because the grid search indicated that using all the features gave the best model, this image is quite large.

Model 2 - Random Forest:

Parameters tested: 'max_depth': [5, 7, 9],
'n_estimators': [50, 100, 200],
'max_features': ['sqrt', 'log2']

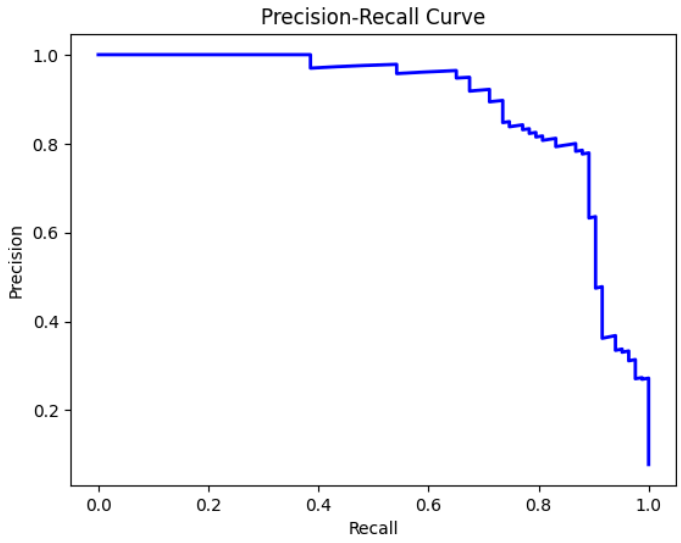
Adding more parameters made the runtime take too long so we settled for this.

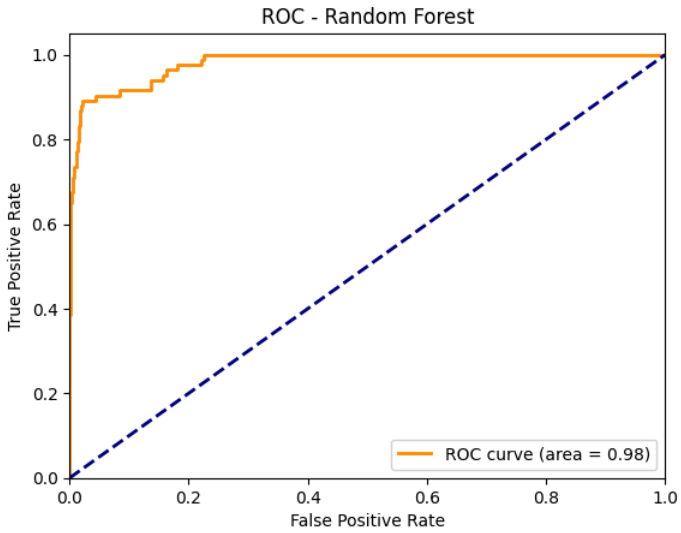
Best Parameters using GridSearch: 'max_depth': 5,

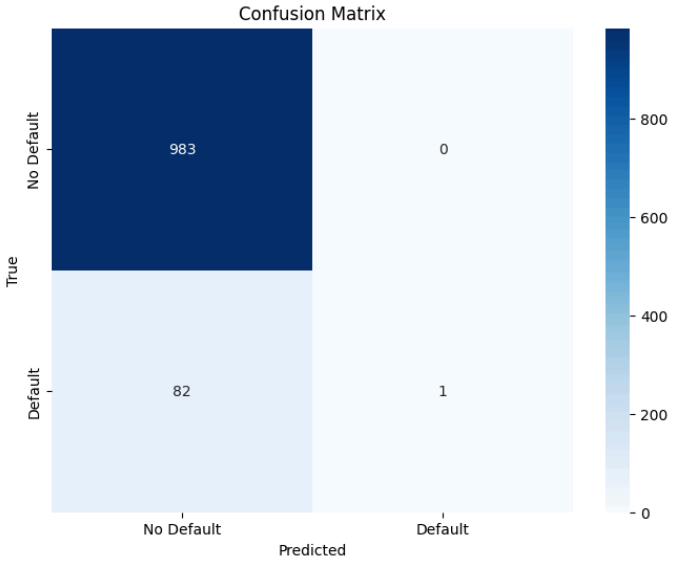
```
                                    'max_features': 'sqrt',
                                    'n_estimators': 50
```

```
------------------Random Forest Model Report-----------------------
accuracy = 0.9258536585365854
              precision    recall  f1-score   support

           0       0.93      1.00      0.96     10367
           1       1.00      0.08      0.15       908

    accuracy                           0.93     11275
   macro avg       0.96      0.54      0.55     11275
weighted avg       0.93      0.93      0.90     11275
```

This model turned out to perform fairly well. Not only does it have an accuracy of .93. It has the highest f1-scores out of all the other models. The ROC curve has a great shape and large area as well as the Precision Curve. Although this is the best model in the report the problem is that the model still does not predict True Negatives well.


Precision-Recall Curve
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0


ROC - Random Forest
True Positive Rate
False Positive Rate
ROC curve (area = 0.98)
0.0
0.2
0.4
0.6
0.8
1.0


Confusion Matrix
No Default
Default
True
983
0
82
1
No Default
Default
Predicted
800
600
400
200
0

Model 3 - AdaBoost:

For the AdaBoost Model, the following hyperparameters were tested

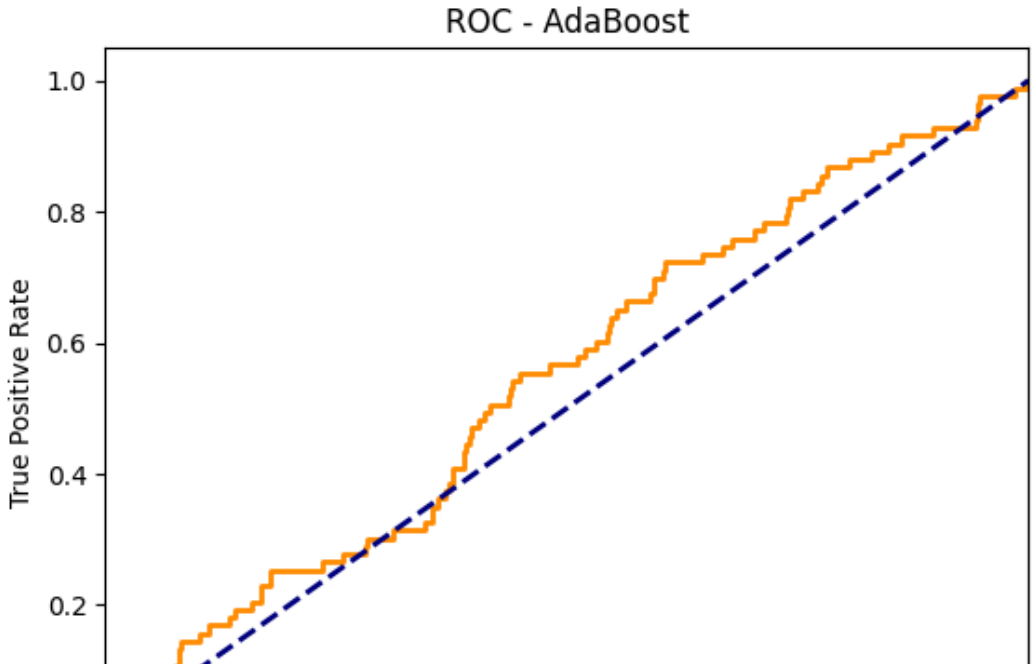
'n_estimators': [50, 100, 200],
'learning_rate': [0.01, 0.1, 1]

The Grid Search algorithm found the best model hyperparameters to be:

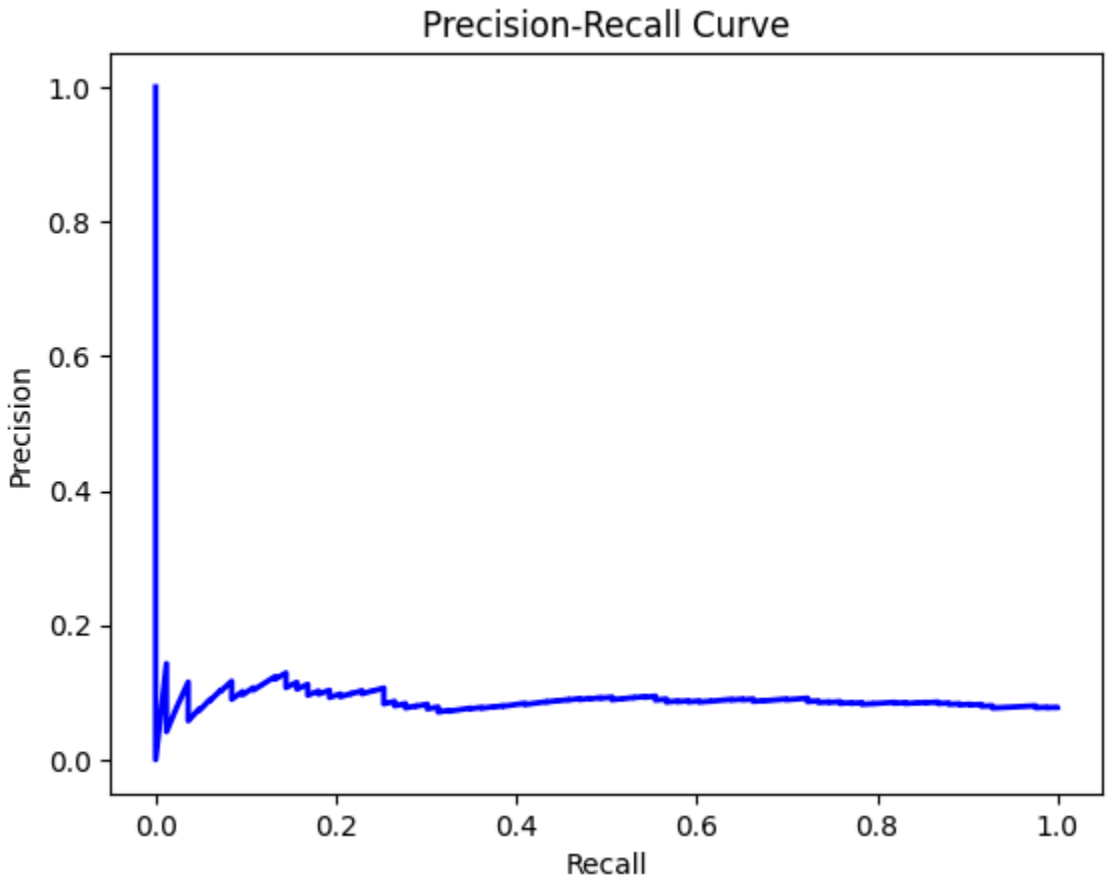
'learning_rate': 1,
'n_estimators': 200

	precision	recall	f1-score	support
0	0.92	0.94	0.93	983
1	0.11	0.08	0.09	83
accuracy			0.87	1066
macro avg	0.52	0.51	0.51	1066
weighted avg	0.86	0.87	0.87	1066

Between the Decision Tree Classifier and this classifier, the overall accuracy improved from 0.83 to 0.87. Furthermore, the f-1 for class 0 improved slightly from 0.91 to 0.93. However, the AdaBoost model still does not perform well for class 1. The f-1 score reduces from 0.1 with the Decision Tree to 0.09 when using AdaBoost.



In addition, the area under the ROC curve being 0.55 indicates that this model is slightly more effective at classifying different classes than randomly guessing. That is not an ideal result for our model, though it is a slight improvement from the Decision Tree model.



Model 4 - XGBoost

For the XGBoost Model, the following hyperparameters were tested

'n_estimators': [100, 200],
'learning_rate': [0.01, 0.1, 0.3],
'max_depth': [3, 5, 7]

The Grid Search algorithm found the best model hyperparameters to be:

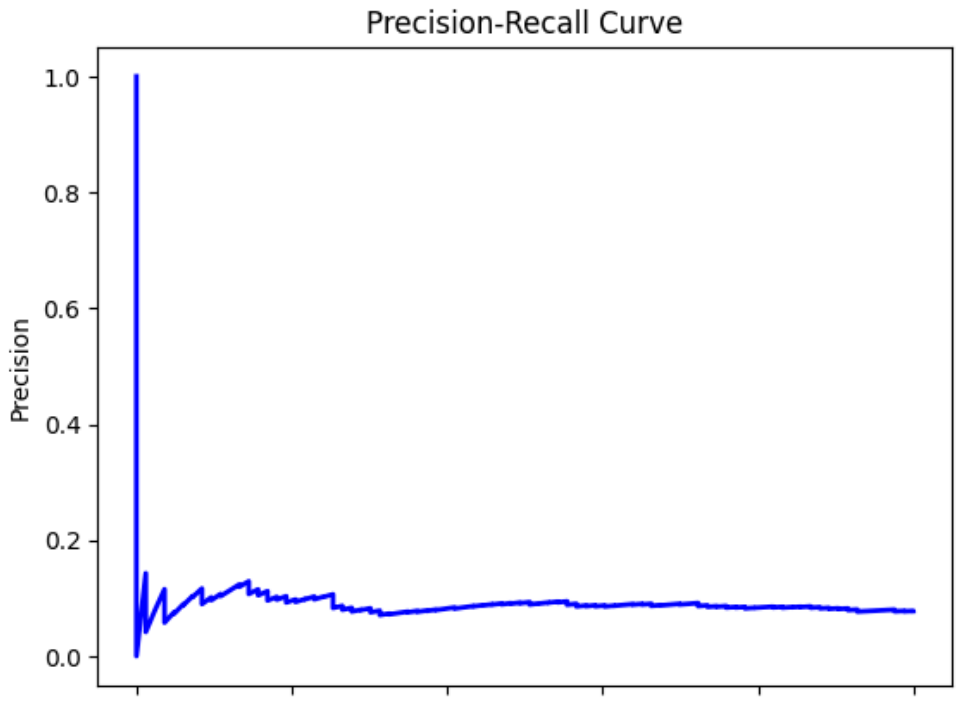
'learning_rate': 0.3,

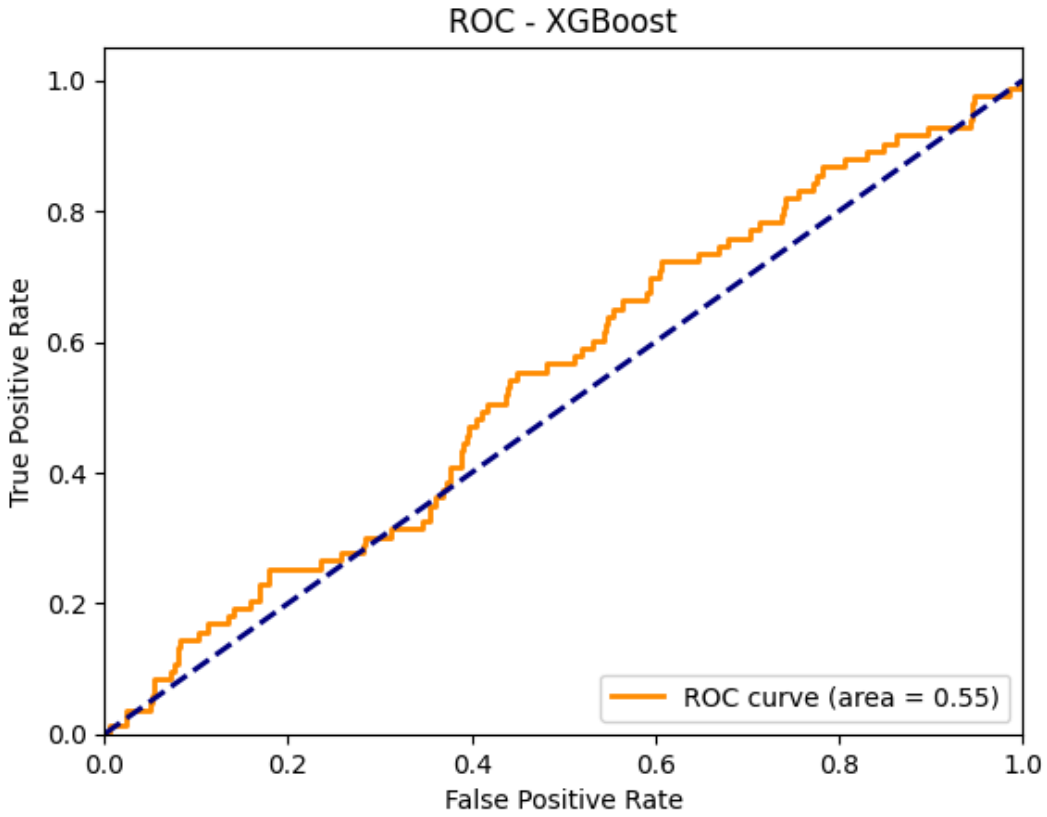
'max_depth': 7,
'n_estimators': 200

	precision	recall	f1-score	support
0	0.92	0.99	0.96	983
1	0.22	0.02	0.04	83
accuracy			0.92	1066
macro avg	0.57	0.51	0.50	1066
weighted avg	0.87	0.92	0.89	1066

Again, we can see that XGBoost improves the f-1 score for the 0 class, however the 1 class score has reduced. Furthermore, the area under the ROC curve is the same as that from the AdaBoost model, indicating that this model is no more effective at predicting the classes than AdaBoost - which did not perform well.

We had initially run the models looking at the data without oversampling the class 1 data. However, in those cases the precision and recall were practically 0. We thought this was caused by the significant class imbalance in our training data and thought that using this oversampling technique (SMOTE) would solve this issue. However, these results, while an improvement, still do not seem to be accurate at classifying the class 1 data. In our data exploration, we found that the features were not highly correlated with the target to begin with. The highest correlation between a feature and the target was 0.07, which is quite low. The lack of meaningful correlation between the features and the target, could be a reason for the poor model performance in this case even after rebalancing the data.





Conclusion:
The models are severely limited by our machines and the imbalance of the Target variable. Even after oversampling we still did not have enough data for actual Defaults to predict it accurately. Perhaps we can try other sampling methods in the future or look at other feature engineering methods. Of the models tested, random forest performed the best.