Introduction and Overview

The goal for this project was to build multiple models for a binary classification problem. We were provided a data set with no information about the data or where it came from. In order to build a predictive model, I cleaned the data by checking for columns with data that could not be fed into a model and converted it to an int or a float so it could be readable. Additionally, I handled missing data and visualized what was left to understand the relationship between the independent and dependent variables. Lastly, I made iterations of both a logistic regression model and a non-logistic or random forest model to track how variations in handling data can impact model performance via AUC and accuracy. For example, I removed data of low importance, tried feature engineering by converting a continuous variable to discrete, and tuned hyperparameters to optimize model performance.

Data Cleaning and Preparation

Data Cleaning and preparation is a very important step for model building. My first step was to get a picture of what the data actually looked like by plotting, checking the data type of each column, and reading through each column to see what information was being stored. I personally prefer to handle all of the data issues before plotting so I went ahead and found all of the columns that had formatting issues such as being a string instead of an int or float and build functions to convert the information. For example, the third column was strictly strings of "Male" and "Female" so I converted each value to 1 and 0 respectively. There were also a few columns that looked like id numbers or some sort of code. Column 29 was filled with strings formatted as three to four numbers followed by two "&" symbols encapsulated by parenthesis so I simply extracted the number from each row.

The second stage of cleaning involved dealing with missing or incomplete data. I found that there was one column that had missing data and two others that looked like just row indexes and a column filled with only the number 4 which would not be useful for building a model because whether y is 1 or 0 the fourth column's constant feature is always 4 meaning it is not predictive in nature. There were two columns with missing data so I filled all missing rows with the mean of each column respectively. I read about multiple ways of filling with missing data and found that median is also a popular choice but since it tends to be used for data that has more extreme values and outliers I kept it to filling all missing data with the mean. Before training, the only other portion of data that I dropped were duplicated rows.

Checking for imbalances, visualizations, and correlations were all completed after I was satisfied with a cleaned data set in a separate .ipynb file. To check for data imbalance I plotted the instances in which y was 0 and 1 and found that there were about 60,000 instances where y was 0 and 45,000 where was 1. In my experience this balance was mostly acceptable and I would have gone back and made it more even if the results were bad but typically you only need to handle more extreme cases of imbalancing but of course it depends on the application. I made a lot of visualizations found in "AD_Data_Analysis.ipynb" but some of the most interesting include the correlation matrix between y and other features and other features with themselves.

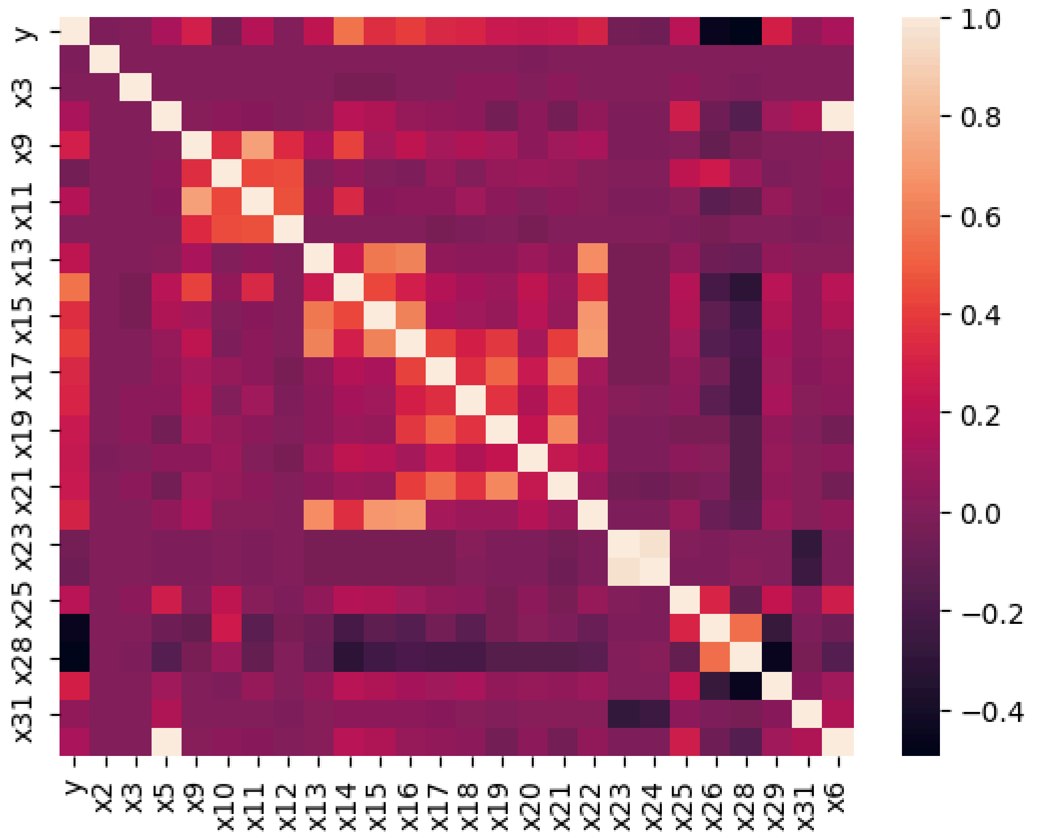

Figure 1 - Correlation matrix heatmap of the cleaned dataset

Figure 1 showed a lot of interesting trends. A few of the features were negatively correlated with each other but most of the data trended positively. Columns such as x2, x3, x10, x12, x23, and x24 had very little influence on y while the remaining columns varied in their correlation with y. Based on Figure 1, I could tell that some of the features were definitely strongly correlated with each other. There were multiple pairs but I picked x16 and x22 to visualize shown in Figure 2.

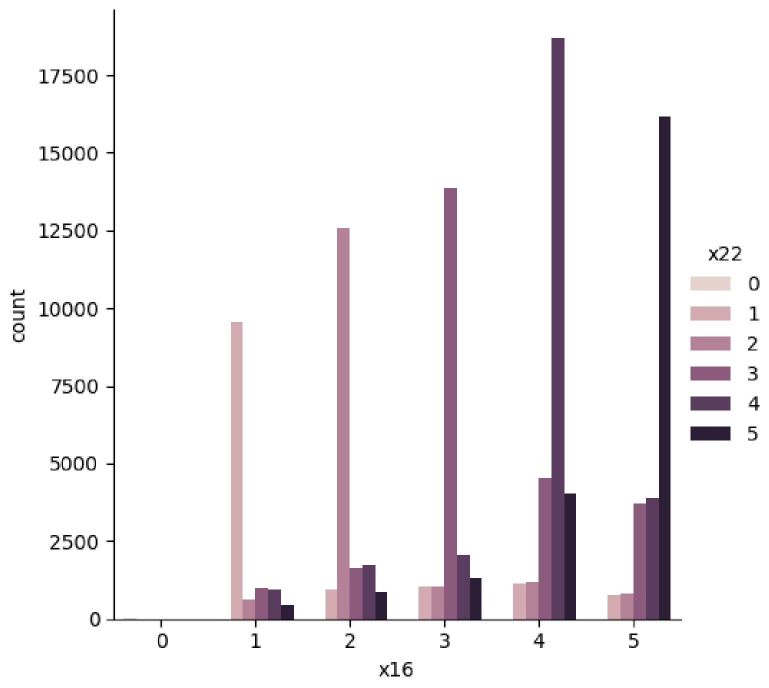

Figure 2 - Relationship between x16 and x22

Figure 2 shows that for each value of x16 the corresponding value for x22 has by far the most influence implying a strong correlation between the two features.

In terms of modeling, I decided to do the standard 80/20 split for training and testing data mostly because I have had a few neural network projects in the past and have always found success with this method but also because most models need a lot of data to really function properly and 80/20 seems to be around the fine line between the right amount of data and over training. For scaling I used our methods from the titanic live coding with StandardScalar(). Based on just looking at the data it did not seem like there were any features with very large values that would impact the model but there were a lot of binary 0 and 1 features so scaling was done just to ensure that each feature would not have too much influence on training.

Part A – Logistic Regression Model

A logistic Regression model is an algorithm that is good for any sort of predictions that have a binary result such as in the case for this project either a 0 or 1. It basically tries to come up with a prediction of 0 or 1 based on a linear combination of features that is fed to it. In the problem that we were given all the features are the columns besides y and the goal is to use logistic regression to train a model to predict based on the features given is y a 0 or 1.

For feature selection, I discussed this in the previous section but I analyzed the data and found constant features and missing features to remove from the data. I used a second phase of feature selection on my second iteration of the logistic regression model by finding the importance of each feature and removing all of the features that had an importance of less than 0.1 because they have little to no impact for predictions. The goal of this was to try and improve AUC values but I managed only to get a small improvement in accuracy shown later. I struggled to some extent with feature engineering using the feature engineering kaggle that was suggested but I saw that this paper suggested trying discretization to get category bins so I decided to sort feature x24 into bins. I specifically chose x24 because the range of values was quite large so the thought that was perhaps separating them into discrete bins would help with AUC and accuracy. The first logistic regression model pass had an accuracy of 0.87338 and AUC of 0.94 and after feature engineering and feature selection the second iteration had an accuracy of 0.8747 and AUC of 0.94 so just a slight improvement.

The third iteration of the model included k-fold cross-validation and hyperparameter tuning. I referenced the live coding and added additional parameters from the scikit-learn website that seemed to have an impact on performance with a 5 folder cross validation to see if there was any impact on the results. I found that the AUC and accuracy basically stayed the same by this iteration.

I had a total of three iterations for the logistic regression models and the results for the training and testing sets as well as the confusion matrix and ROC curve with AUC values for each iteration are below:

Iterations	Training Accuracy	Testing Accuracy
1	0.8733826688947243	0.8751208565596763
2	0.8747195034614467	0.875264093963737
3	0.874671759369778	0.8752044117120451

Figure 3 - Training and testing set accuracy for logistic regression model

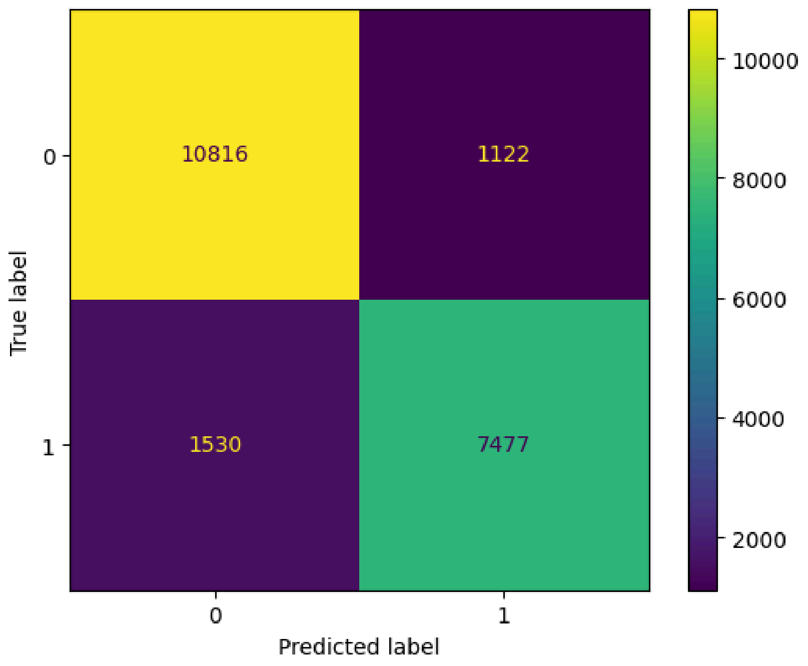

Figure 4 - Confusion matrix for iteration 1 testing set

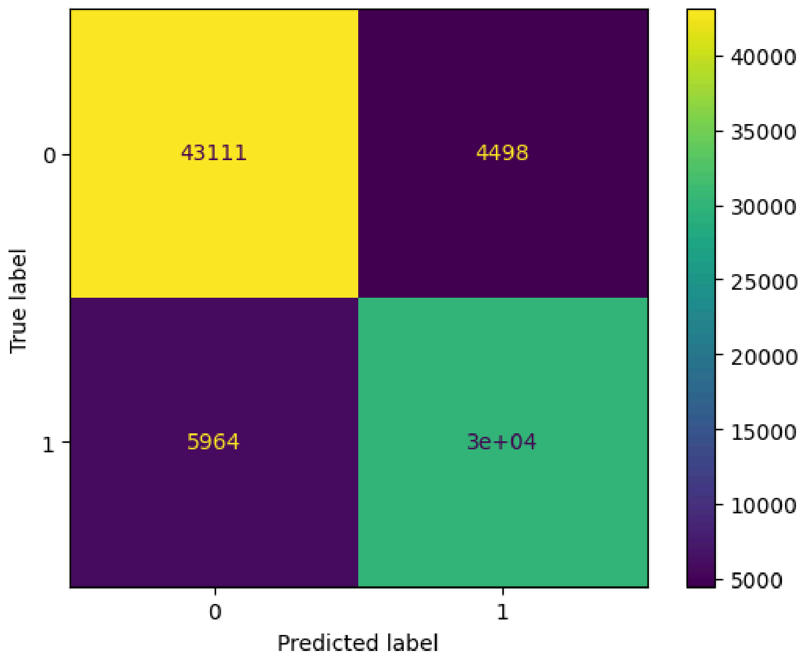

Figure 5 - Confusion matrix for iteration 1 training set

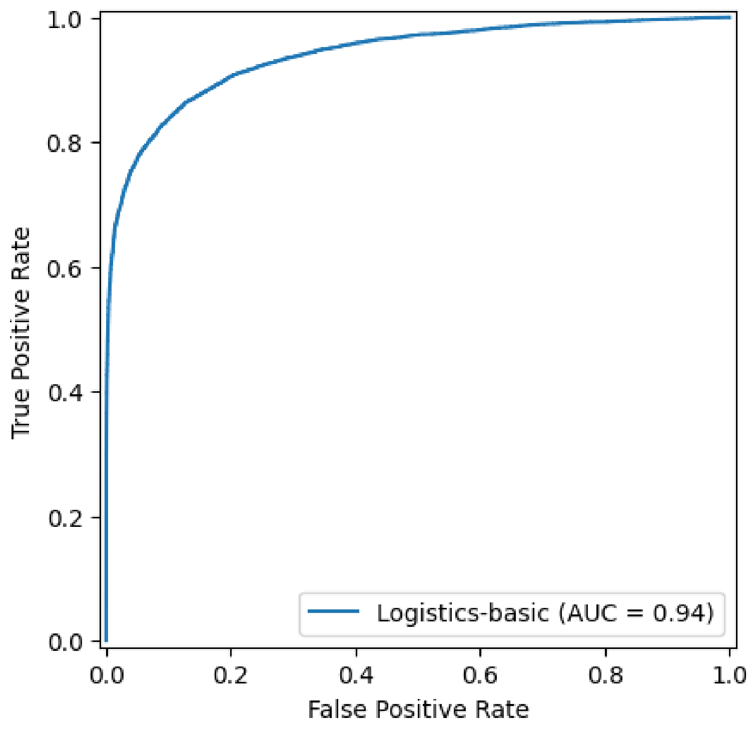

Figure 6 - ROC curve with AUC value for iteration 1 testing set

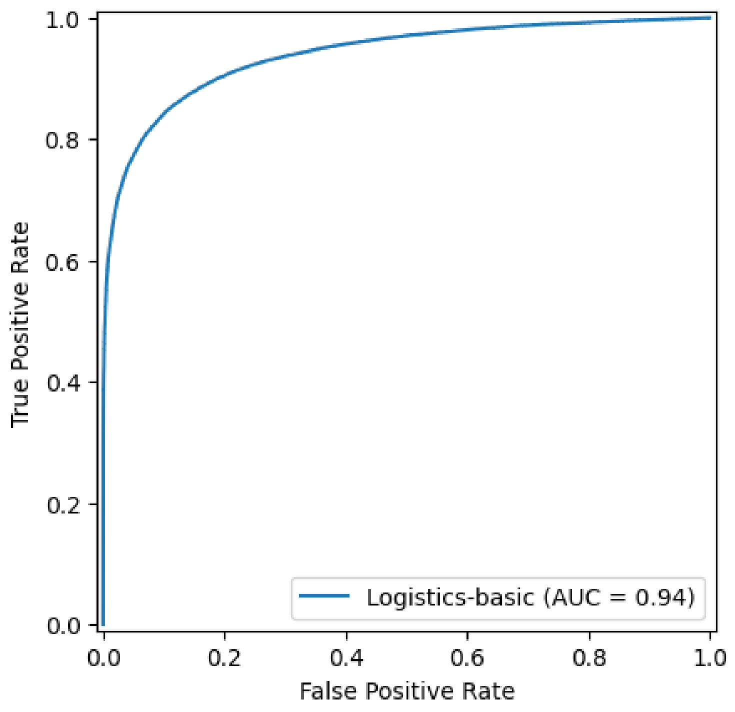

Figure 7 - ROC curve with AUC value for iteration 1 training set

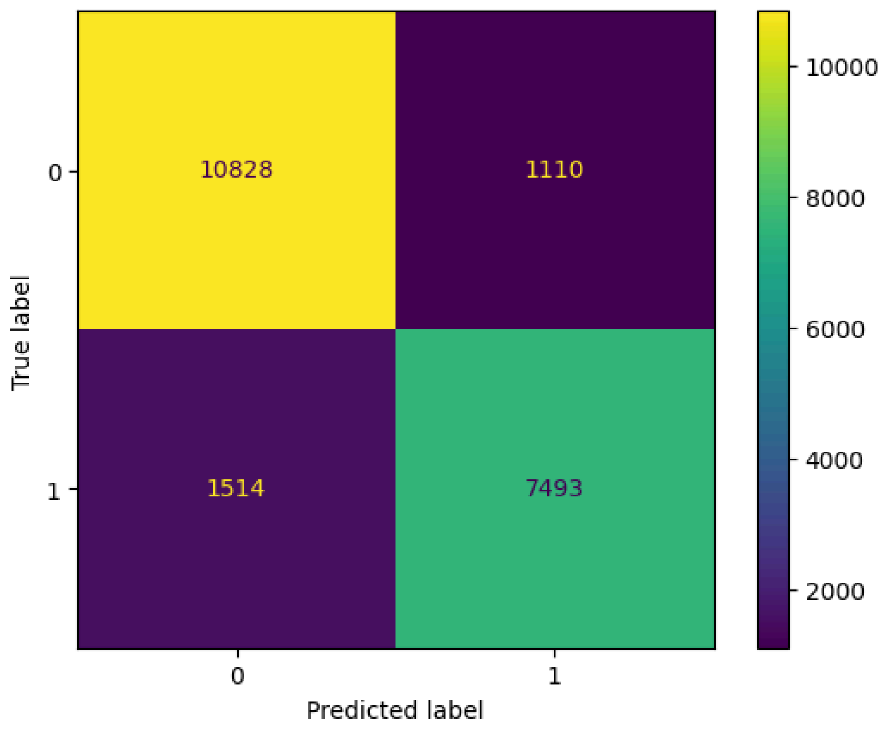

Figure 8 - Confusion matrix for iteration 2 testing set

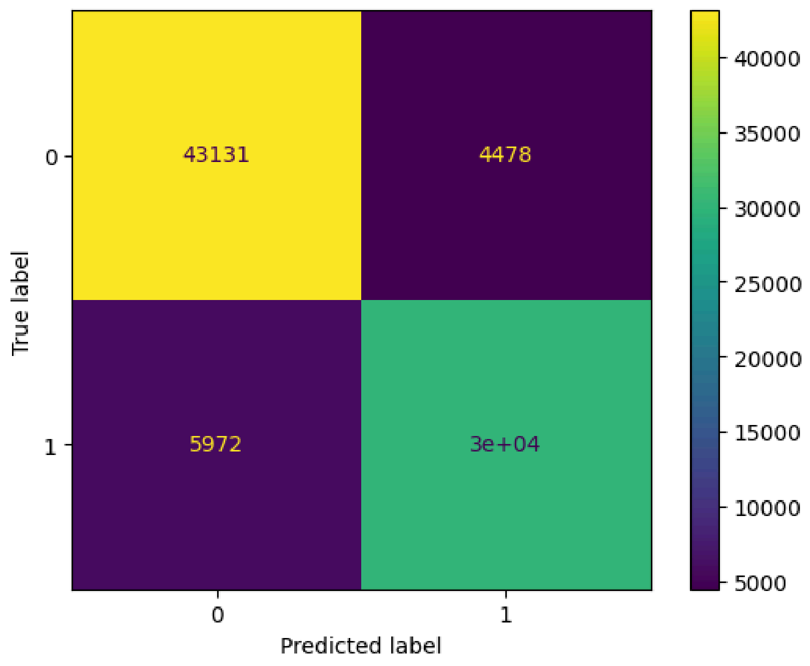

Figure 9 - Confusion matrix for iteration 2 training set

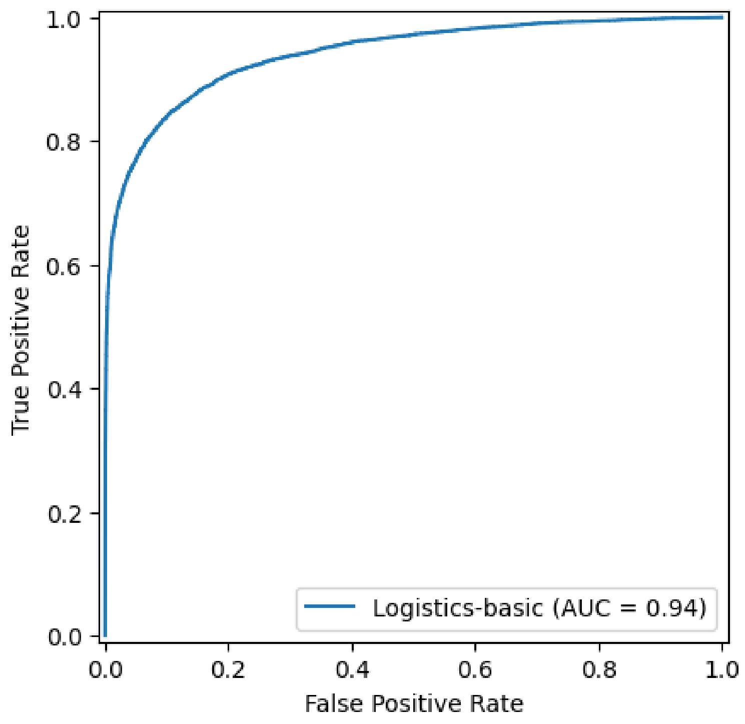

Figure 10 - ROC curve with AUC value for iteration 2 test set

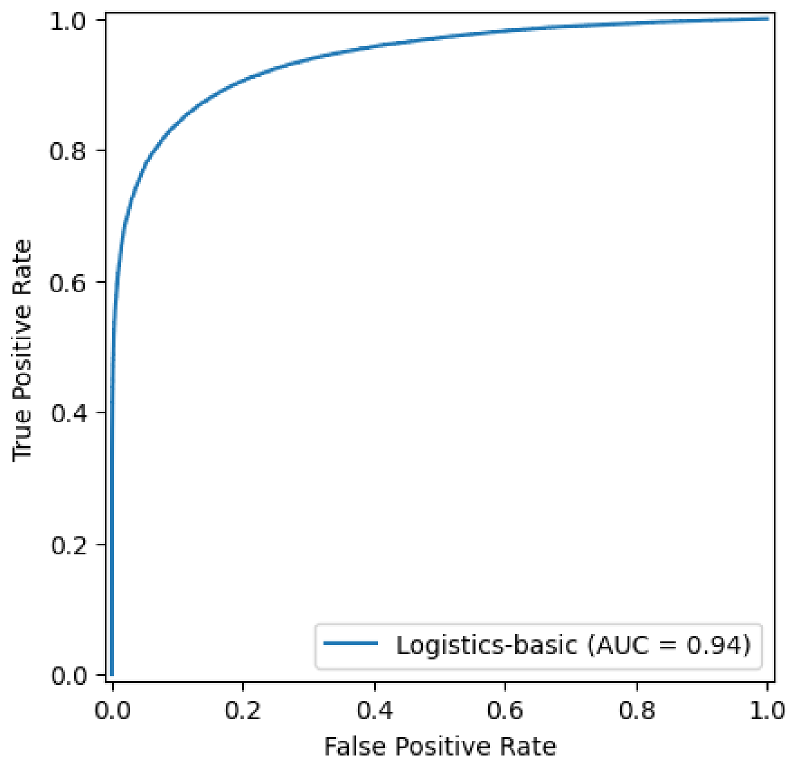

Figure 11 - ROC curve with AUC value for iteration 2 training set

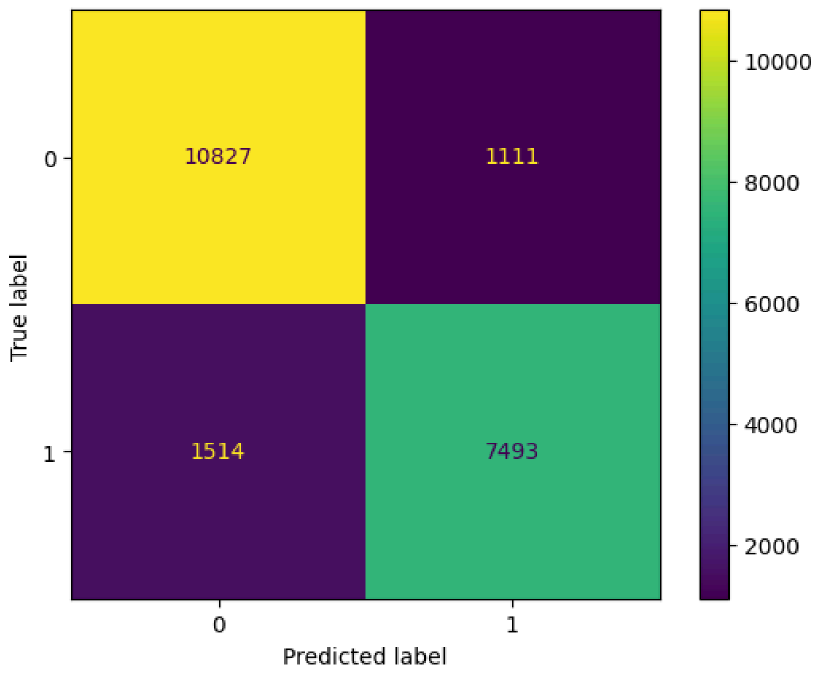

Figure 12 - Confusion matrix for iteration 3 test set

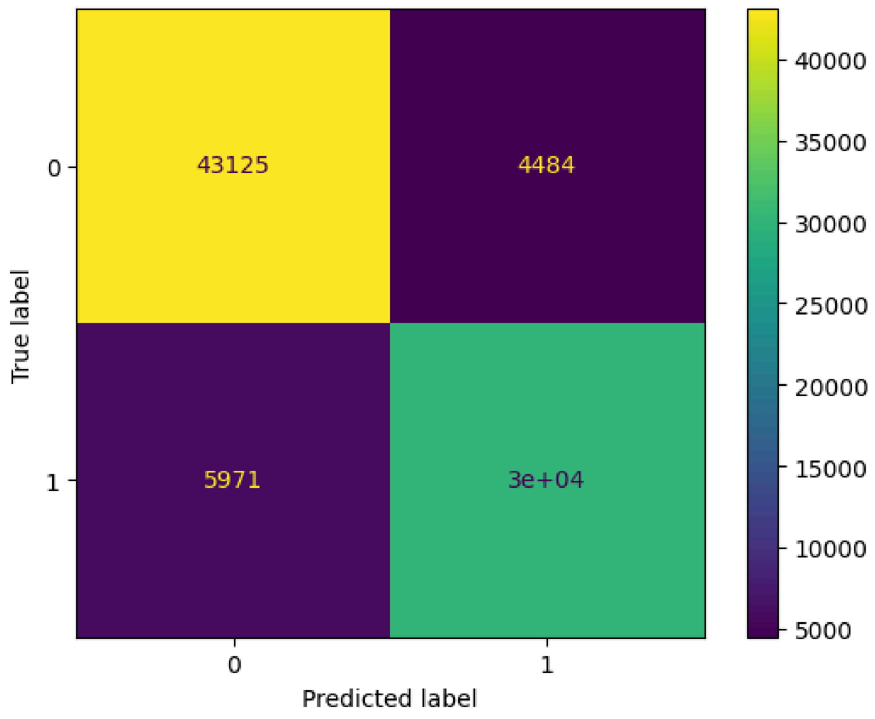

Figure 13 - Confusion matrix for iteration 3 training set

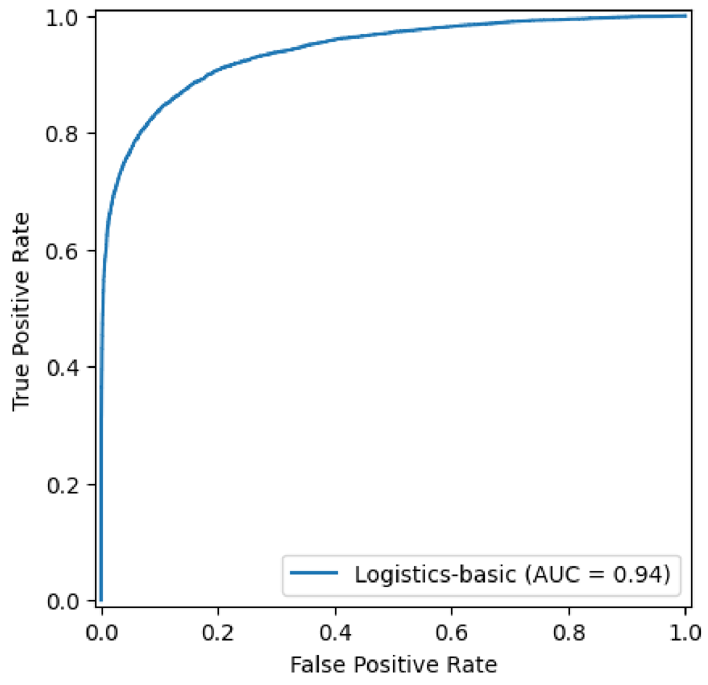

Figure 14 - ROC curve with AUC value for iteration 3 testing set

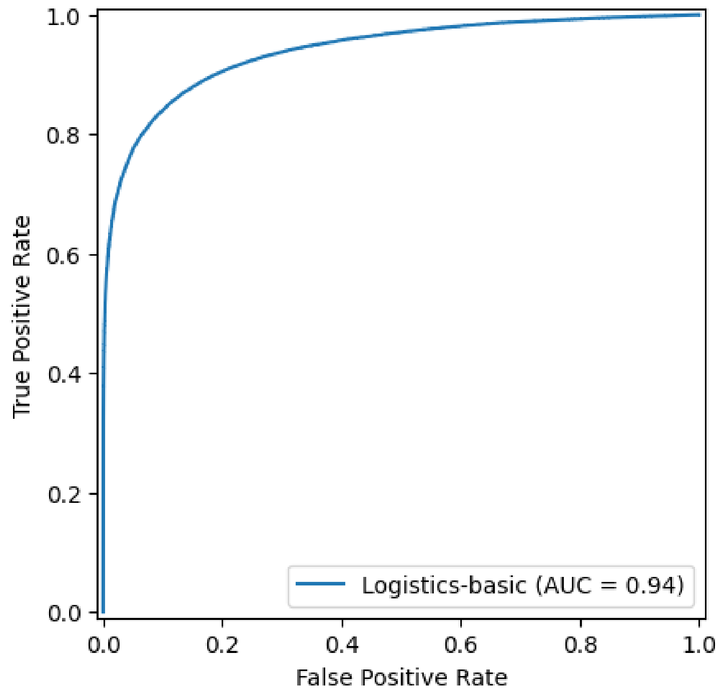

Figure 15 - ROC curve with AUC value for iteration 3 training set

Part B – Non-logistic Model

There were a lot of options for the non-logistic model portion of this investigation but I decided to go with the random forest model because one of our live codings briefly talked about the random forest model but I wanted to learn more considering I already have experience with some neural network models. Random forest in nature seems similar to neural networks even though they function differently. A random forest is a set of decision trees that take a segment of data so that each tree has a random set of features and y. Next each tree makes a prediction based on the random set of data it was given to train from and the result is what the majority of the trees output. The advantages of this is that there is very low variance and the accuracy is very high. However, this can lead to overfitting data and very long training and prediction time. For example, I barely had any hyperparameters but with k-fold cross validation and grid search I still had to wait 30 minutes.

My methodology for selection hyperparameters was ambitious in the beginning. I used a lot of the hyperparameters the scikit-learn site discussed but realized after waiting an hour that it may take days for the model to train. As a result, I significantly reduced the amount of hyperparameters listed and kept the combination that produced the best accuracy by a thin margin. I ended up with two iterations of training the random forest model, one baseline and the other with hyperparameter tuning and the results are below:

Iterations	Training Accuracy	Testing Accuracy
1	1.0	0.9650035808068751
2	1.0	0.965433277631893

Figure 16 - Random forest training results

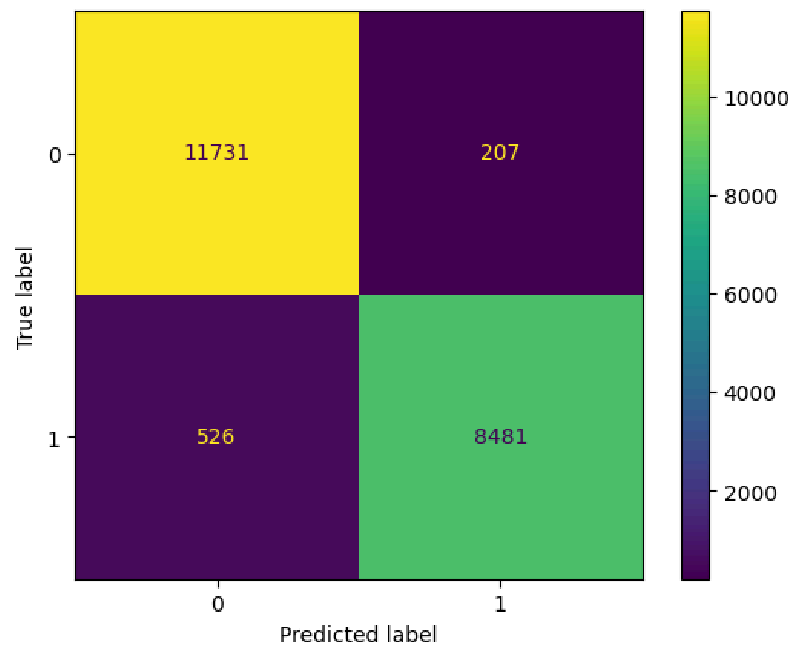

Figure 17 - Baseline random forest confusion matrix testing set

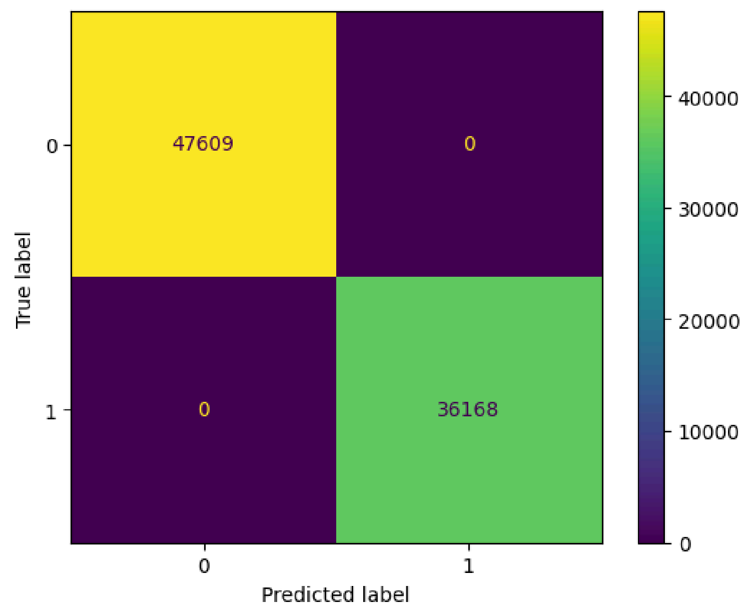

Figure 18 - Baseline random forest confusion matrix training set

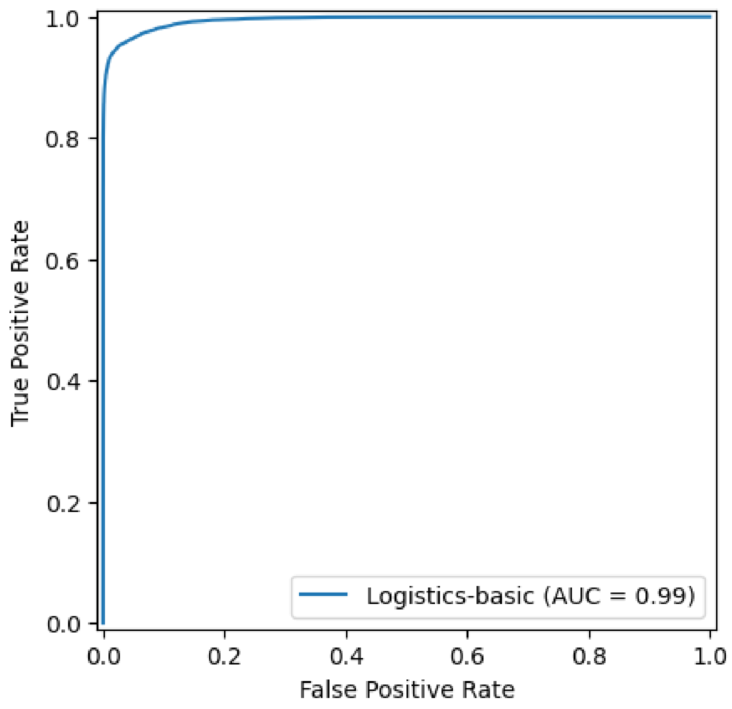

Figure 19 - Baseline ROC curve with AUC value for random forest testing set

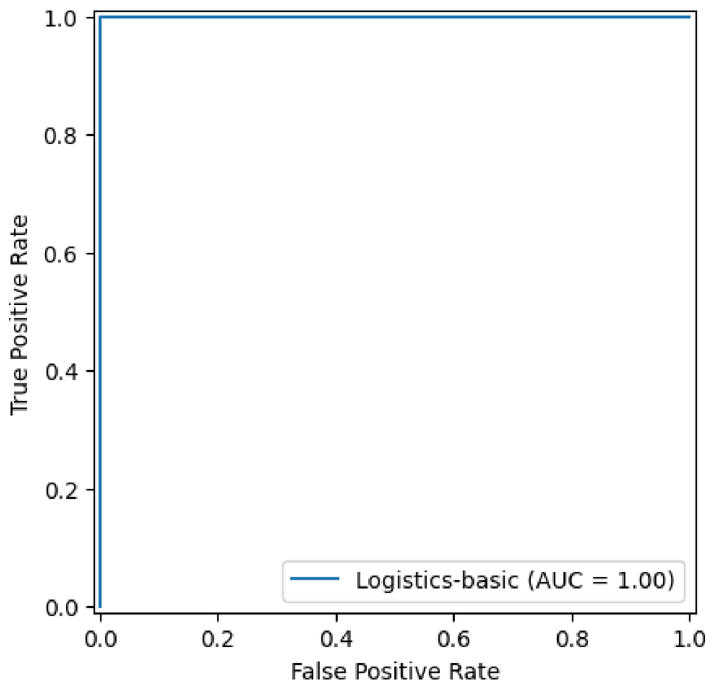

Figure 20 - Baseline ROC curve with AUC value for random forest testing set

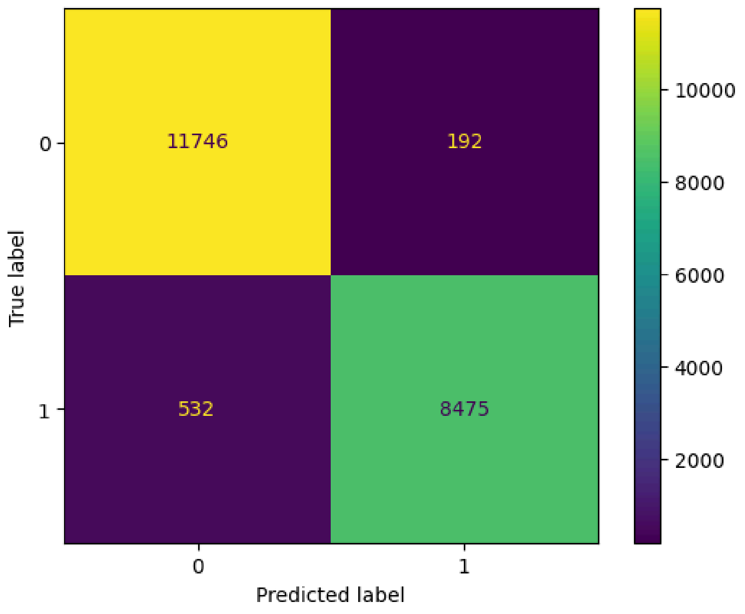

Figure 21 - Confusion matrix for second iteration of the random forest model with testing set

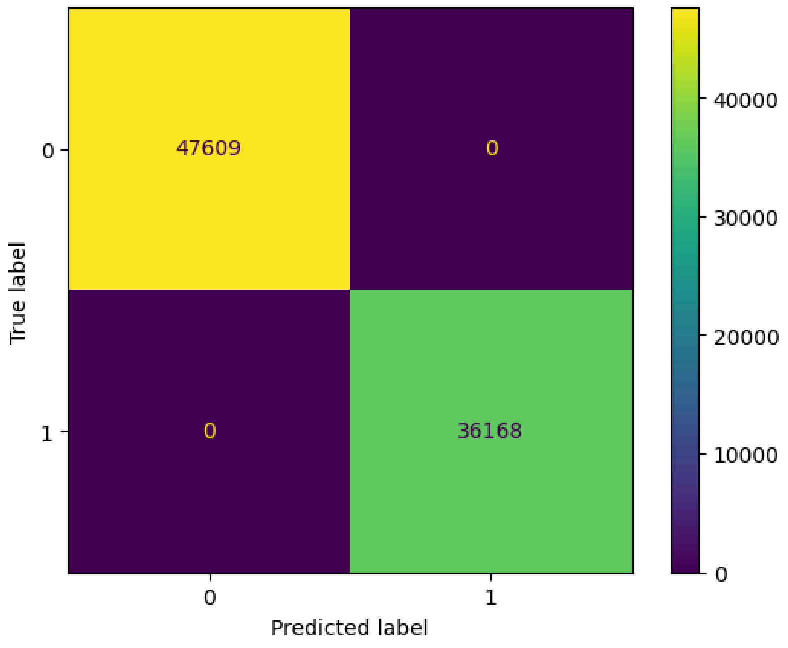

Figure 22 - Confusion matrix for second iteration of the random forest model with training set

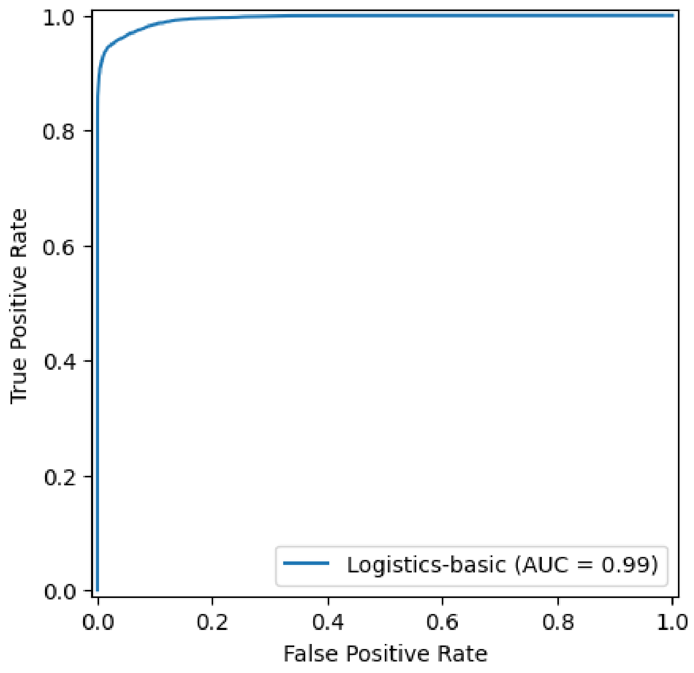

Figure 23 - ROC curve with AUC value for second iteration random forest testing set

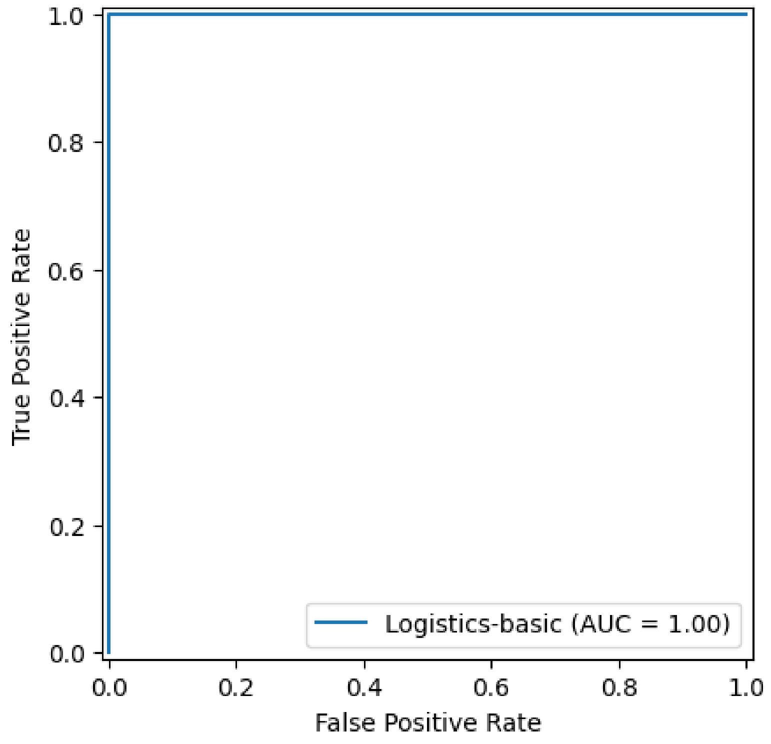

Figure 24 - ROC curve with AUC value for second iteration random forest training set

Discussions

	Model 1 Logistic Regression	**Model 2 Random Forest**
Best AUC for training	0.94	1.0
Best AUC for testing	0.94	0.99

Figure 25 - Best AUC for logistic regression and random forest models

The random forest model performed better than the logistic regression model in terms of both accuracy discussed earlier and AUC shown in Figure 25. It was expected that random forest performed better due to the wide range of data provided. However, I was surprised by how well logistic regression performed but this just means that there was a lot of linearity in the data provided. If I were to try to explain this to a business partner I would talk to them about how given data, each model tries to predict an outcome. Based on what I round, the random forest model tends to make better, more accurate predictions than the logistic regression model. If they were interested, I would walk them through the prediction making process and let each model make a few predictions and show how the logistic regression model makes the wrong prediction more than the random forest model.