Kaggle Submission Name: Jacquelyn Du

I. Exploratory Data Analysis

First looking at the attrition column itself, it's clear that there is a class imbalance. Out of the 1340 observations in the training dataset, approximately only 11% have actually left the company. This class imbalance will be addressed via resampling methods in the model tuning selection later. All of the potential predictors are examined both visually as well as using some form of hypothesis tests to determine if there are any statistically significant differences between each predictor and attrition.

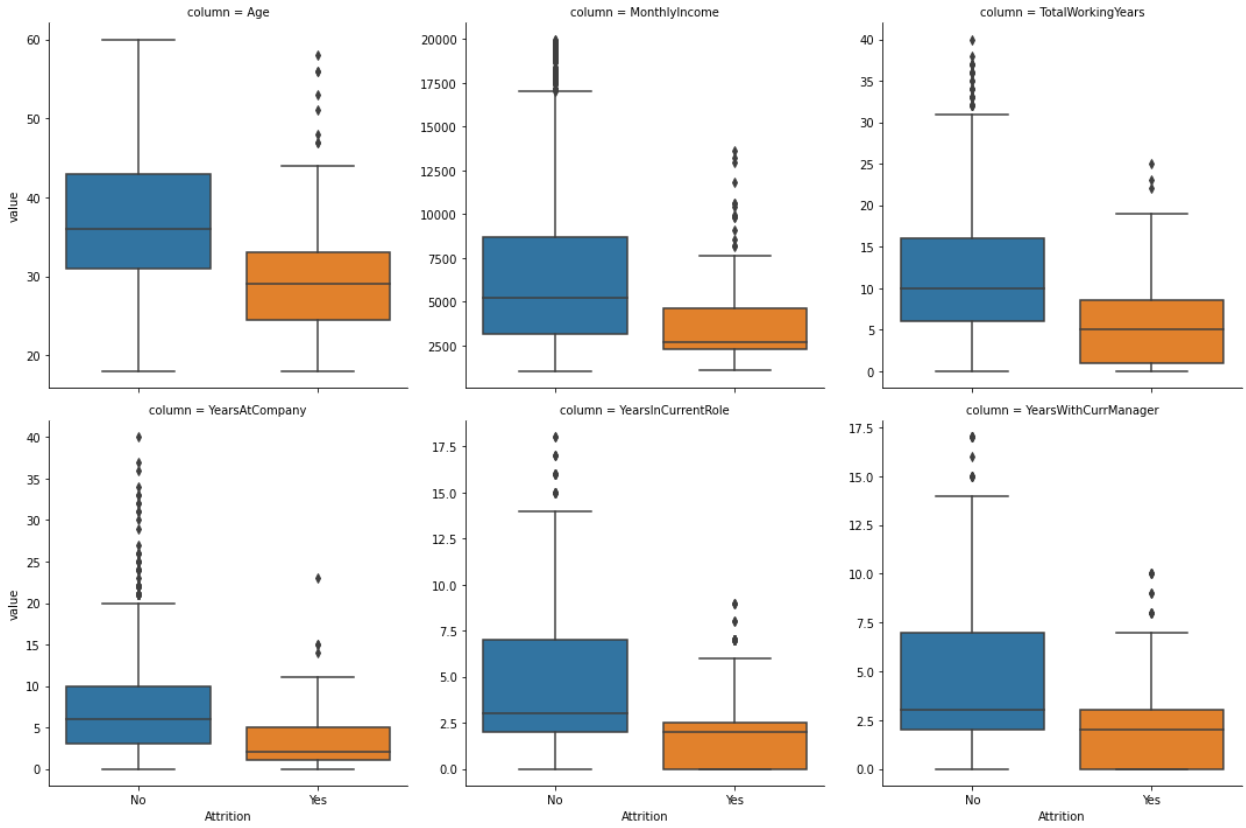
The first part of this section explores the numerical predictors. Of the fifteen numerical predictors, eight predictors were found to have significant differences in attrition using a two-sample t-test. Below shows the table of p-values for each of these predictors, which indicates the probability of observing as extreme a difference in means between attrition or not, assuming that the expected difference is zero. Those that have a significant p-value have asterisks.

Fig. 1 - Two-sample t-test results for numeric predictors (* 0.05, **0.01, ***<0.001)

Predictor	**P-value**	**Predictor**	**P-value**
Age	1.316e-22 ***	Training Times Last Year	0.076
Daily Rate	0.043 *	Years at Company	1.584e-26 ***
Distance from Home	0.00019 ***	Years in Current Role	2.063e-23 ***
Hourly Rate	0.193	Years since Last Promotion	0.0001 ***
Monthly Income	1.130e-27 ***	Years with Current Manager	3.537e-18 ***
Monthly Rate	0.222	Total Working Years	8.909e-32 ***
Number of Companies Worked	0.886	Percent Salary Hike	0.773

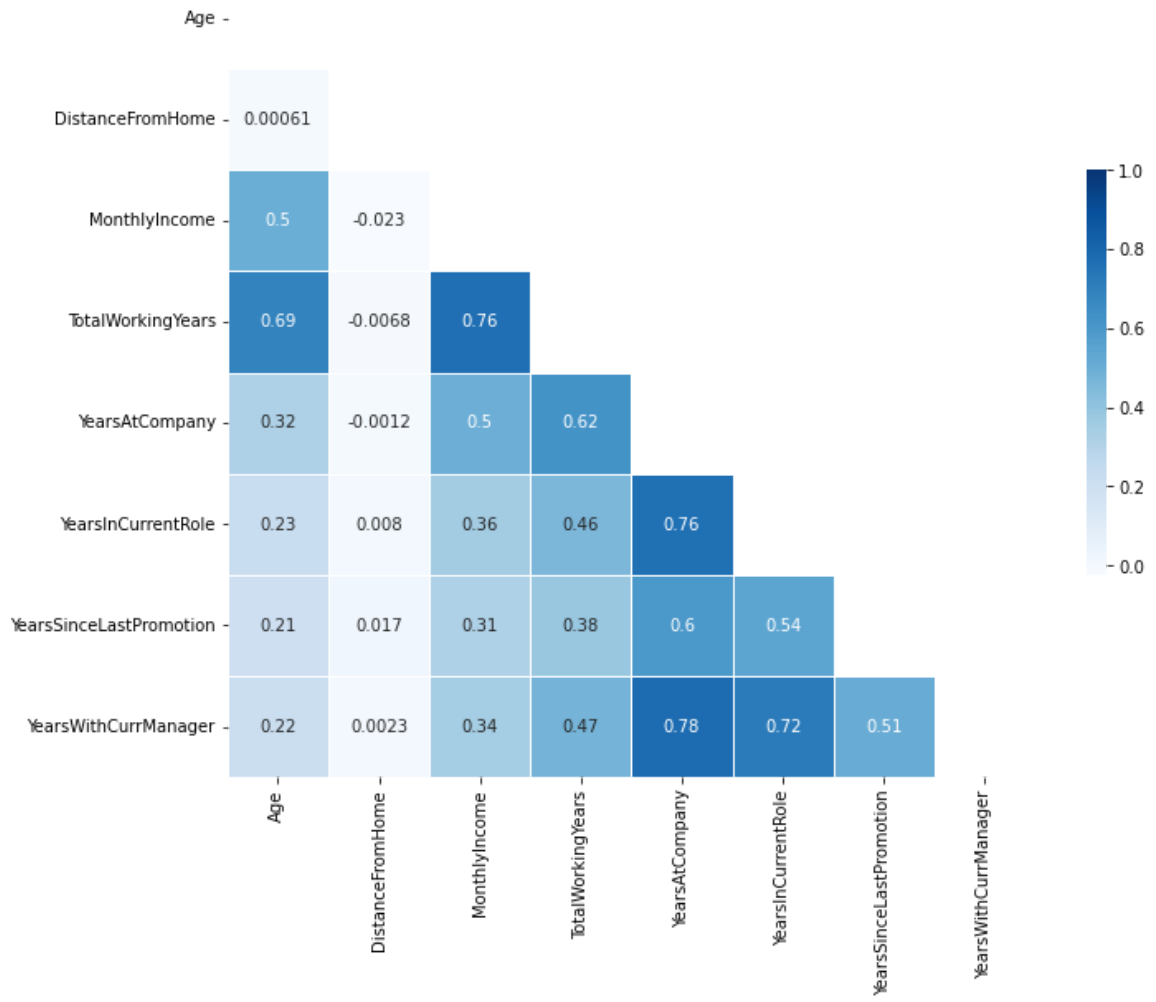
In the below sample of boxplots, we can see a clear difference in distribution of the predictor between the two levels of attrition. Only six out of the eight significant predictors are shown for simplicity's sake.

Fig. 2 - Boxplots of sampled numeric predictors by Attrition



Then, the correlation matrix is calculated for the eight significant numeric predictors. As shown below, there are a few predictors that are relatively positively correlated - mostly those in terms of years worked, either at the company, in the role, or overall working years. The rest are only weakly positively correlated; the exception is distance from home, which is very weakly positively and negatively correlated with the other predictors).

Fig. 3 - Correlation Matrix of Significant Numeric Predictors



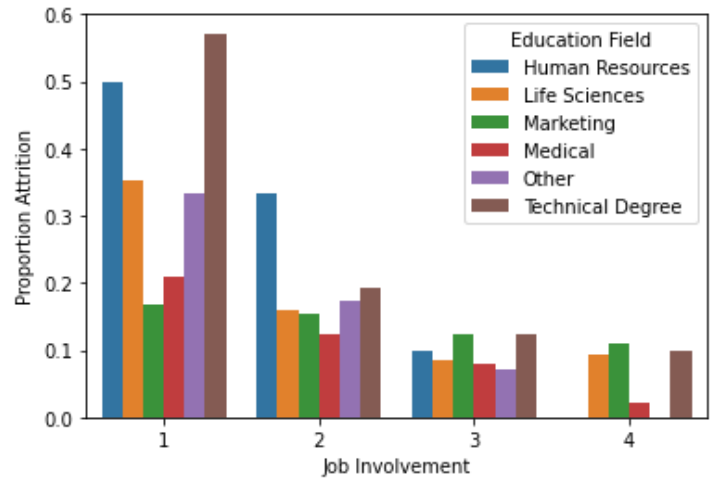
Now looking at the categorical predictors, feature selection was made based on the result of a F-test. The F-test would determine whether there was any significant difference between the levels of the specified categorical predictor in terms of attrition. If the F-test proved statistically significant, pairwise step-down t-tests were conducted between all pairs of levels to determine which pairs are significantly different from each other. The p-values of these pairwise t-tests were adjusted using the Holm's method to preserve the overall power at the 0.05 level.

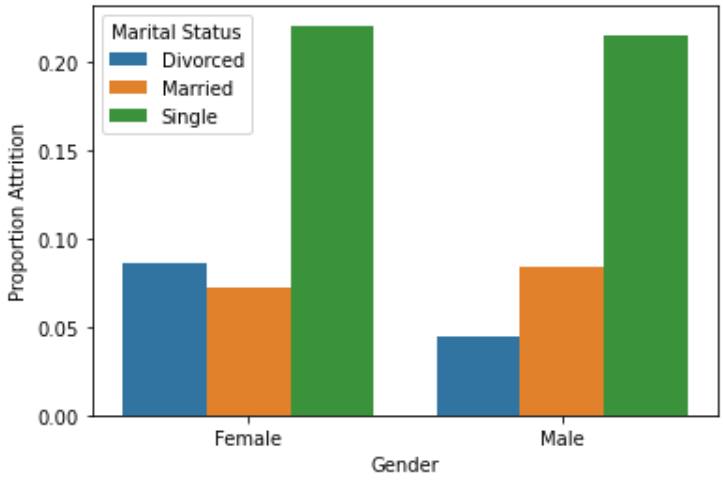
Out of the sixteen categorical predictors, nine predictors were very significant and 'Job Satisfaction' was weakly significant (p-value of 0.04) at the 0.05 level. Upon examining the pairwise t-tests for 'Job Satisfaction', there were no significant differences between any of the levels, so this predictor was excluded from further analysis. All except one of the highly significant predictors have almost all significant differences between pairs - the exception is 'Work Life Balance' where being in category 1 was the only level that differed from the others. The table below aggregates the F-test results for each predictor; asterisks signify significant results.

Fig. 4 - F-test Results for Categorical Predictors by Attrition (* 0.05, **0.01, ***<0.001)

Predictor	**P-value**	**Predictor**	**P-value**
Business Travel	0.002 **	Job Role	2.205e-08 ***
Department	0.05 *	Job Satisfaction	0.048 *
Education Level	0.164	Marital Status	1.442e-13 ***
Education Field	0.159	Work Life Balance	0.0001 ***
Environment Satisfaction	0.000011 ***	Over Time	8.578e-35 ***
Gender	0.674	Performance Rating	0.777
Job Involvement	9.482e-10 ***	Relationship Satisfaction	0.509
Job Level	3.435e-23 ***	Shift	2.428e-13 ***

Fig. 5 & 6 - Two-way Bar-charts of Categorical Predictors by Proportion of Attrition





The above barcharts show that even when there is no significant difference across levels of one particular variable with regards to Attrition, that doesn't necessarily mean that it doesn't differ when you interact it with a different term. From the F-test results, we see that education and gender are not considered significant. But when we interact gender with marital status, and education field with job involvement, we see a difference in the 'insignificant' variable based on the levels of the interacted variable. For example, gender itself has no visible difference between male and female, but we can see that divorced females are much more likely to leave a company relative to divorced males. This might be evidence for the case of including predictors beyond just those that are on the surface significant.

II. Data Preparation

The original dataset is then subsetted based on the results of the previous section. The numerical predictors are left in their original units and not standardized because only tree-based methods are used, which are robust to different scales. The categorical predictors are processed using one-hot encoding. Note that while initially, the data was subsetted to include only the significant predictors from this section, the performance was poor so the entire dataset is then included for training which improved scores. These two subsets together comprise the predictors for the data to be used in model training, and the boolean values of the Attrition column are used as the true labels.

III. Initial Model Selection

For models, I chose to tune and select the best performing out of a Random Forest classifier, Gradient Boosting classifier, or AdaBoost classifier (all from the sklearn package).
I chose tree-based ensemble methods since I am familiar with working with all of them, and they are pretty robust in the sense that it makes no strong assumptions on the data, both in the distribution of the outcome variable and in the relationships among the predictors. Given that the goal is predictive accuracy, I leaned towards using these ensemble methods over using something more interpretable like decision trees or penalized logistic regression, especially since these methods might require more careful feature engineering or domain knowledge.

In terms of how each individual algorithm is optimized:

(i). Random Forest
The random forest algorithm is a tree-based ensemble method in which many decision trees are trained independently, and the algorithm averages over the trees in the forest to improve generalization. For each tree, only a subset of the predictors is considered at each split - the exact number of predictors in consideration is tuned in this analysis. The reason only a subset of predictors is considered is to decorrelate across the trees - if we allowed all predictors to be considered, naturally all the trees in the forest would resemble each other. This way, we can perturb the subsets of data to allow for individual trees to learn the nuances of some predictors.

While individual trees in the forest might classify poorly, to predict labels, in classification the majority vote is taken across all trees. Since each tree is trained on a different subset, random forests tend to have good predictive accuracy over having just one decision tree classifier.

Within each tree, each split is made greedily in the sense that the algorithm will choose the predictor in the subset that minimizes the loss function at that node. The loss function that it minimizes is Gini impurity in this analysis, which characterizes the distribution of classes in a given node. The formula is shown below where each p_i represents the proportion of data points with label i

$$Gini(D) = 1 - \sum_{i=1}^{k} p_i^2$$

In this analysis, the number of trees and the depth of each tree are also tuned. Therefore, the random forest classifier will fit the specified number of trees where each tree is fit to the specified depth. Tuning both of these parameters prevents the algorithm from overfitting.

(ii). Gradient Boosting
The gradient boosting classifier differs from random forest primarily in that the former fits its weak classifiers additively, while random forest fits them independently. While the way that each decision tree is constructed similarly as before (i.e. in a greedy optimization way with gini impurity based on the specified constraints for number of features and maximum depth), the difference is that each new tree is constructed in a way to optimize an overall gradient descent. This is to say that rather than averaging each tree's classification at the end of training like in random forest, each additional tree in gradient boosting is fitted so that the overall loss of the algorithm is minimized in each subsequent step. In this analysis, the log loss function is used, which is the default loss function for classification.

(iii). AdaBoost
The AdaBoost algorithm, like the name suggests, is a form of gradient boosting in which each weak classifier is a decision tree with only one split and the loss function is the exponential loss function. While gradient boosting can also tune for these parameters, in this analysis, the learning rate (or the contribution of each classifier to prediction) and the number of classifiers are specifically tuned. (The specific ranges and methodology will be discussed in a later section.)

IV. Resampling Methods

Another aspect of the model pipeline that needed to be selected was the resampling method. Most online resources referenced using the imblearn package which had functions to resample the data accordingly. I selected to use two combined over and undersampling methods, manually using SMOTE and randomly undersampling as well as the available function that combinines SMOTE and Tomek Links method. I chose to use a combination method because

acting alone, oversampling tends to bias the algorithm to overfit to the training data as it effectively creates more data using the existing training data. On the other hand, since the training dataset is relatively small, undersampling the majority class to balance the classes would reduce the training dataset too small to have efficient cross validation.

SMOTE stands for Synthetic Minority Over-sampling Technique. This method over-samples the data by taking the k-nearest neighbors of points to then artificially simulate new data points. The number of neighbors in the analysis are tuned between 3 and 5 points.

The randomly undersampling method will randomly sample points of the majority class to exclude from the resulting dataset. Combined, the data is first oversampled using SMOTE so that the ratio of classes now equals 0.3 (from ~0.11). Then, the majority class is then undersampled so that the number of observations for the classes are equal. These ratio parameters are manually set and not tuned simply for computational efficiency. Because this is my first time working with these methods, I wanted to keep it simple and focus more of the hyperparameter tuning on the actual algorithms themselves.

The Tomek Links method undersamples the majority class by finding and removing the nearest neighbor (i.e. the Tomek Link) to each sampled point. Therefore in this combined method, which exists as a function in the imblearn package as SMOTETomek, the dataset's minority class is first oversampled using SMOTE as described above and then the majority class is undersampled using the Tomek Links method.

V. Hyperparameter Tuning

The hyperparameters of the models were tuned via grid search cross validation using repeated stratified k-folds (with 5 folds and 10 repeats). I used stratified because of class imbalance, and I used stratified to increase robustness in the performance across the cross validation since repeated splits would reduce any variance resulting from any one split. Both resampling methods were tuned for the number of neighbors the SMOTE algorithm would use (two options were given, k=3 and k=5).

A imblearn Pipeline for each resampling method was used to combine the resampling and model fitting step (this was because I wasn't able to figure out how to both tune for different resampling methods and models in one pipeline, so I created a separate pipeline for each resampling method). I used the same random state in creating the stratified folds to preserve which splits are used across pipelines.

The specific hyperparameters of the models tuned are as follows:

a. For Random Forest
    i. The number of trees in the forest were tuned for every 25th value ranging between 25 and 250
    ii. The maximum depth of each tree for values between 1 and 5

iii. The maximum number of features to be considered at each split - two provided parameter options were given, either the base 2 log or the square root of the number of total features

b. For Gradient Boosting, the exact same hyperparameters were tuned across the same range as that for random forest above
c. For Adaboost
    i. The learning rate, or the weight of contribution for each classifier is tuned for values 0.001, 0.001, 0.01, 0.1, and 1.0.
    ii. The maximum number of estimators trained for every 10th value between the values of 10 and 50.

VI. Model Training

The total time for model training was 5310 seconds, or approximately 88 minutes.

Fig. 7 - Best mean (F1 / AUC) score for each sampling method and model training combination

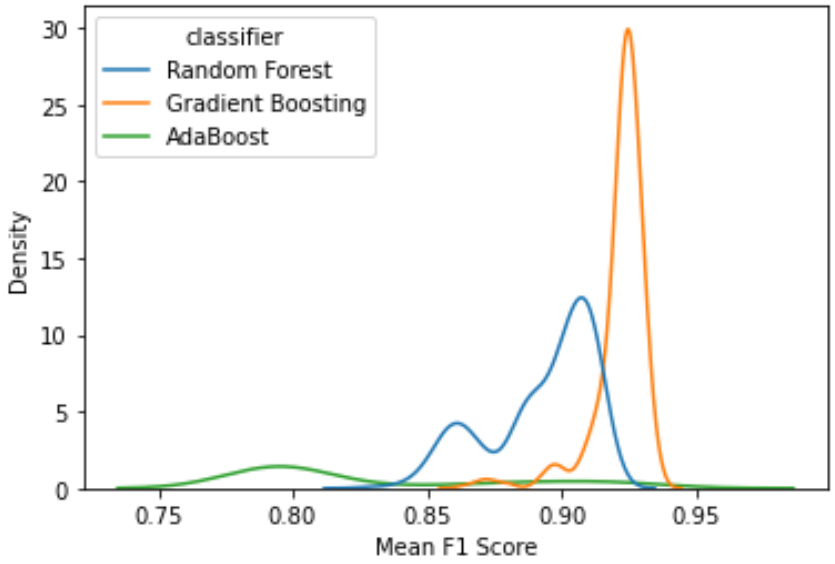
	Random Forest	Gradient Boosting	AdaBoost
Smote Tomek, k=3	0.9150 / 0.9264	0.9347 / 0.9516	0.9257 / 0.9388
Smote Tomek, k=5	0.9144 / 0.9251	**0.9353** / 0.9524	0.9229 / 0.9322
Smote & RUS, k=3	0.9153 / 0.9271	0.9342 / 0.9523	0.9241 / 0.9353
Smote & RUS, k=5	0.9168 / 0.9272	0.9342 / **0.9529**	0.9262 / 0.9386

The table above is constructed by finding the max F1 or AUC score for each resampling method and model. (Note that the parameters based on the two scoring metrics for a given cell might not be the same, but I will choose to pick a final model based on only the F1 metric). The maximum score of each metric across all cells are shown in bold. I show both metrics just to see how the model does across the two metrics - while AUC is commonly used, I wanted to see if F1 performance differed dramatically since that metric factors in false negatives in the Recall term, and is also the evaluation metric of this assignment.

We can see that across columns, or across models, the best Gradient Boosting algorithm tends to perform better than its Random Forest and AdaBoost counterparts for a given sampling method. Across rows, or across resampling methods, we see that the Smote Tomek is not definitively better between the number of neighbors. But, for Smote and Randomly Undersampling, it appears that when we use 5 neighbors, the performance tends to be slightly better. (I also tried training on 7 neighbors, but that did not improve performance - to keep the following analysis pared down, I evaluate performance on these k=3,5)

In order to assess the models specifically, I aggregate the different resampling methods and plot the density plots of the mean F1 scores. These mean F1 scores are the validation (are out of sample) scores when conducting cross validation tuning. They are averages across splits for a given fold. We can see that Adaboost has really high variation in its performance with a comparatively uniform distribution peaking around 0.80. The Random Forest does much better than Adaboost - it has a bi-modal distribution, presumably a peak for the two distinct resampling methods at approximately 0.87 and 0.92. By far, the gradient boosting method does the best, with a very concentrated distribution around 0.93. The high concentration suggests that its performance does not vary much across the resampling methods.

Fig. 8 - Histogram of F1 test scores across models and resampling methods



Since the gradient boosting model performs the best, I examine the tuned hyperparameters solely for this model. The three graphs each represent one tuned hyperparameter, and the different colored curves are for the four sampling method options (two unique methods with a tuned parameter). We can see that as the number of estimators increases, the validation F1 scores increase and then plateaus towards the higher end - this makes sense as the larger the model, the better the performance but the more it starts to overfit. For the maximum depth, we see a sharp peak at 3, and trees in the ensemble with greater depth tend to overfit and perform poorly. Finally, for the number of features to consider for each split, there does not appear to be much drastic difference between the options. Note that the graph limits on the y-axis are very zoomed in to show nuances. Maybe using square root instead of log base 2 is slightly better, but it varies by sampling method and if so, it is only marginally better.

Fig. 9 - Tuning of Number of Estimators for Gradient Boosting Classifier

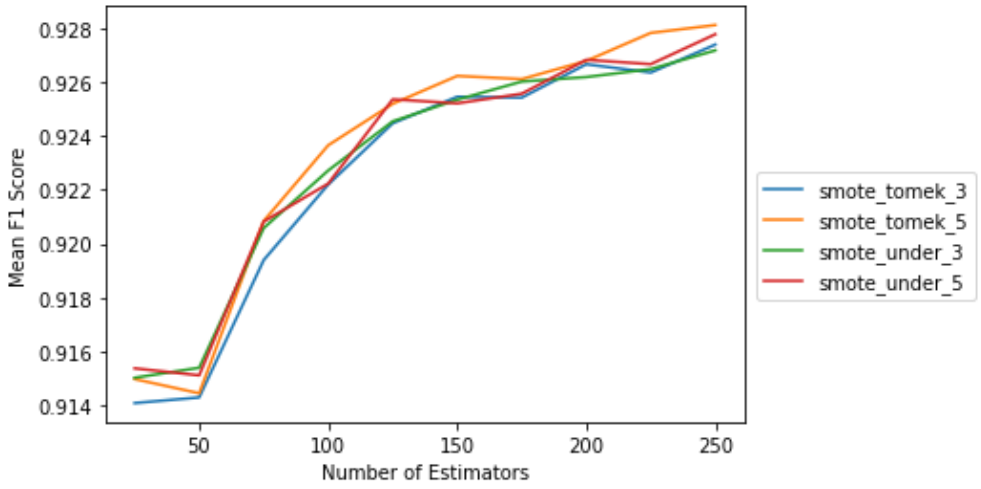


Fig. 10 - Tuning of Maximum Depth for Gradient Boosting Classifier

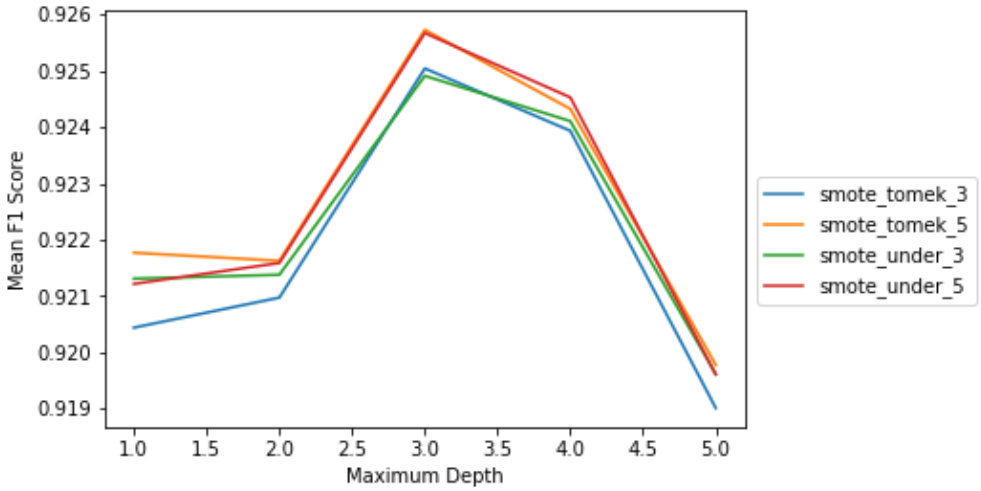
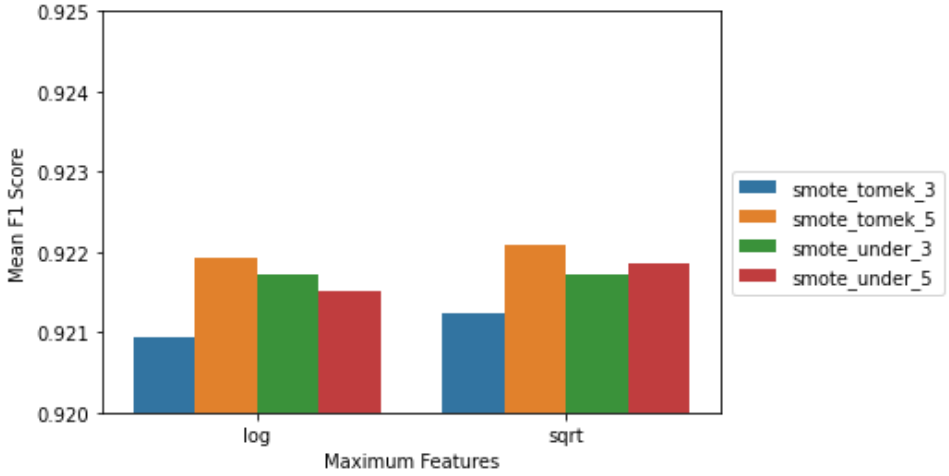


Fig. 11 - Tuning of Maximum Features for Gradient Boosting Classifier



VII. Final Model Selection

Using the highest average F1 score from cross validation of hyperparameters, the best overall model was using the SMOTE Tomek resampling method with 5 neighbors with a Gradient Boosting classifier with parameters maximum depth of 2, maximum features using log base 2 of total number of feature, and 250 maximum number of estimators.

I then fit this final best model on the entire training set. After applying the same data processing to the testing set, I make predictions on attrition based on the best model on the test set. These predictions are then made to fit the format of the submission file, and then stored into a csv to be then uploaded onto Kaggle.

VIII. Conclusion

In retrospect, I would have liked to spend more time exploring feature engineering and feature selection. I chose to first use a subset of significant predictors initially. When the performance was bad, I thought it was based on model design. After deducting that the poor performance was due to subsetting the data, I chose to use the entire training dataset instead. If I had more time, I would choose to see if different subsets would improve performance. In terms of predictive accuracy, including unimportant or correlated predictors would not affect these ensemble methods, but the efficiency and interpretability would be improved.

Additionally, implementing different resampling methods or improving the way that I implemented it would be another focus of mine. At the very least, tuning the ratios of classes for sampling methods would be something that I would further explore.

I found it difficult in trying to implement complex methods and then having to backtrack and debug how components could be altered to improve performance. I tried using simpler models to test performance but then when adding the resampling, I wasn't sure if I did something simple it would inherently affect performance. Gaining more experience in processing data via the sklearn pipeline was very helpful, and I definitely want to continue so that I can more efficiently train models in the future.