CS 671 Kaggle Competition Writeup

Evan Glas

Abstract

The following report documents my approach towards the CS 671, Theory and Algorithms of Machine Learning, final Kaggle competition. In brief, the goal of my work was to build a binary classifier that predicts employee attrition on a tabular dataset of approximately 40 columns and 1300 rows. The objective of the competition was to maximize my classifier's F1 score on a given test set. The Kaggle competition revealed a given classifier's performance on only half of the test set before the submission deadline, discouraging overfitting the test set through excessive submissions. This report outlines my work throughout this project, including the techniques I employed, insights I generated, and mistakes I made along this process. This report is divided into the following sections: exploratory data analysis, models, training, hyperparameter selection, mistakes/lessons learned, predictive accuracy, and an appendix of relevant code. My final model, a cross-validated logistic regression model trained on up-sampled data, achieved an F1 score of 0.8.

Exploratory Data Analysis

Upon loading the dataset, the first step I took was to gain a sense of the shape, and quality, and distribution of the data. The training dataset consists of 1340 rows and 35 columns, while the test dataset consists of 340 rows and 35 columns. Of the 35 columns, 9 columns correspond to categorical features and the remaining 26 correspond to numerical features. The data appears to be of high quality, as there are no NaN values, all columns hold consistent datatypes, there are no extreme outliers in any column, and all values for each feature have a consistent format.

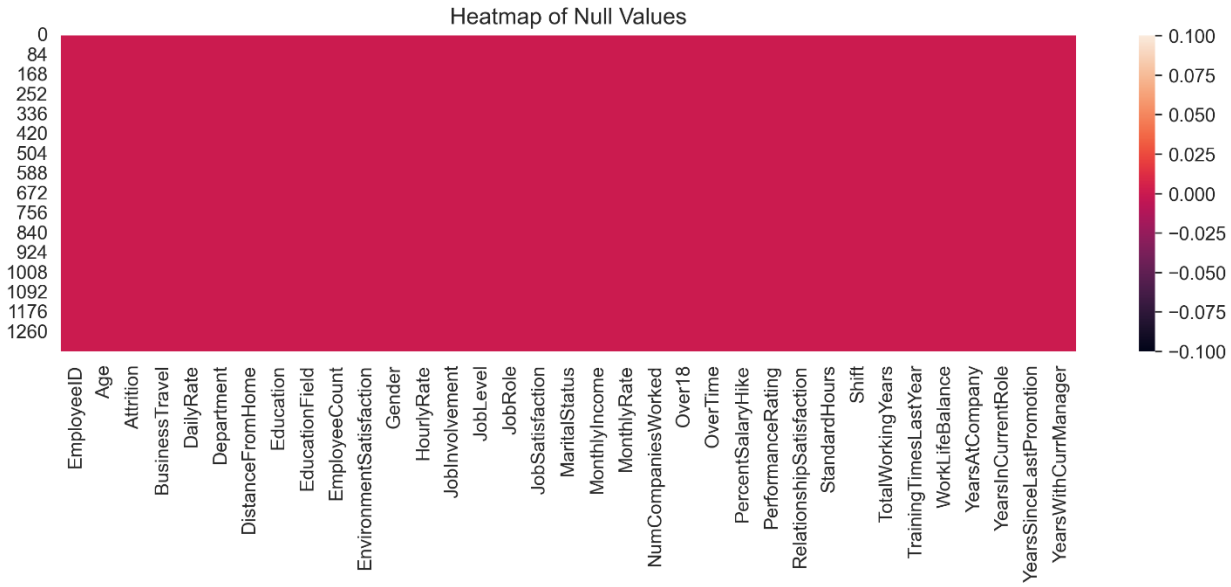


Figure 1: heatmap demonstrates there are no null values across the entire training dataset

To determine the distribution of values across each feature, I constructed bar charts for each of the 9 categorical features and histograms for each of the 26 numerical features (see following pages).

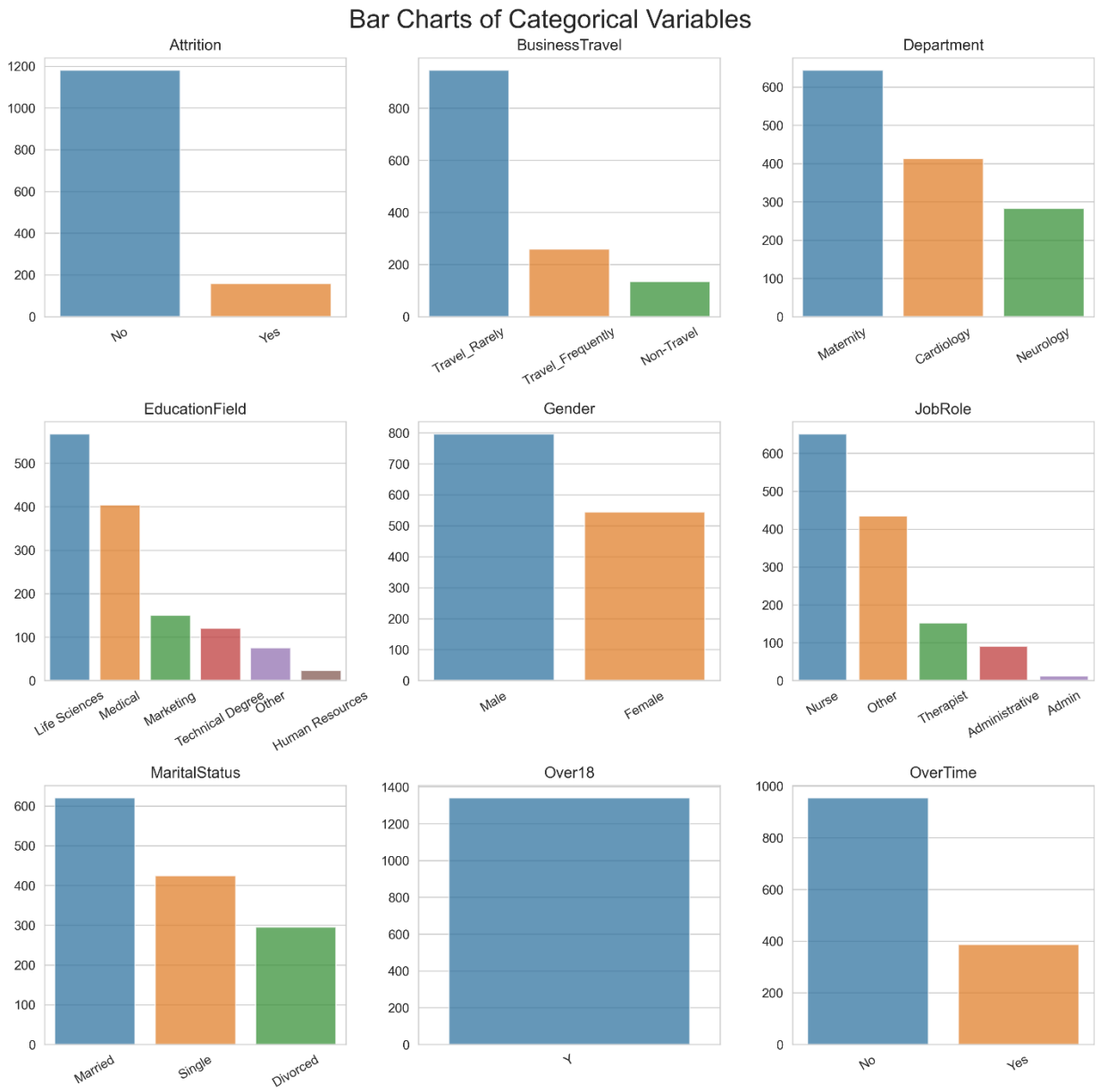


Figure 2: Bar charts of all 9 categorical variables in the training data set

I drew several observations from the above bar charts of the categorical variables. I noticed the distribution of the target variable, Attrition, is highly unbalanced, with about 1150 entries with value "No" to 150 entries with value "Yes". This fact would inform my approach to adjusting point weights in models as well as my implementation of up-sampling. Further, I chose to drop the "Over18" column during preprocessing as this feature only contains entries with value "Y", rendering it useless in a predictive model. The remaining categorical features are generally distributed across multiple different values, and there are no categorical features (besides "Over18") for which one class overtly dominates the distribution. I chose to represent each of these features using leave-one-out one-hot encodings to avoid one-to-one collinearity between one-hot features. That is, a I represented a categorical variable with *n* possible values with *n-1* one-hot columns.

Figure 3: Histograms of numerical variables

I also drew multiple observations from the above histograms of each of the 26 numerical variables. Two features, “StandardHours” and “EmployeeCount”, consist entirely of one value, and I thus chose to drop those columns during preprocessing. These histograms also demonstrate that all features fall within outwardly reasonable ranges. That is, each of the histograms spans a reasonable range of values for its corresponding feature (e.g. “Age” falls within 20 to 60, which would make sense in the

context of an employee attrition dataset). Further, I noticed multiple features which take on discrete values which may, in effect, function more as categorical variables. For example, the "Shift" variable takes on values [0,1,2,3]. Without associated documentation, these values could be interpreted in disparate ways. The underlying meaning of these values also do not necessarily follow a clear increasing order from 0 to 3, and the distance between such values cannot be assumed to be "linear". As such, when I eventually applied linear models to the dataset, I chose to represent such ambiguous numerical features ("Education", "EnvironmentSatisfaction", "JobInvolvement", "JobLevel", "JobSatisfaction", "PerformanceRating", "RelationshipSatisfaction", "Shift", "WorkLifeBalance") with one-hot encodings. Finally, I noticed the "EmployeeID" variable appeared to follow a somewhat random distribution across its range. Since an employee ID may be an arbitrarily assigned identifier, I examined the correlation between "EmployeeID" and every other variable. I discovered that "EmployeeID" is not highly correlated with any other feature, and especially not the target variable, and I consequently chose to drop this feature from my processed dataset.

I then attempted to gauge a preliminary understanding of the high-dimensional shape and structure of the data using several methods which project the dataset onto two dimensions. First, I implemented a principal component analysis on a standardized version of the training data. In the below graph, it is apparent that many of the samples positive for attrition are concentrated within a small two-dimensional region of the first two principal components. Some of these points are visibly separated from negative samples and would thus likely be easily distinguished even with a linear model. However, most of the Attrition-positive samples are scattered among negative samples. This would suggest that it may be necessary to include additional features/principal components to provide further dimensions with which the positive/negative samples may be distinguished.

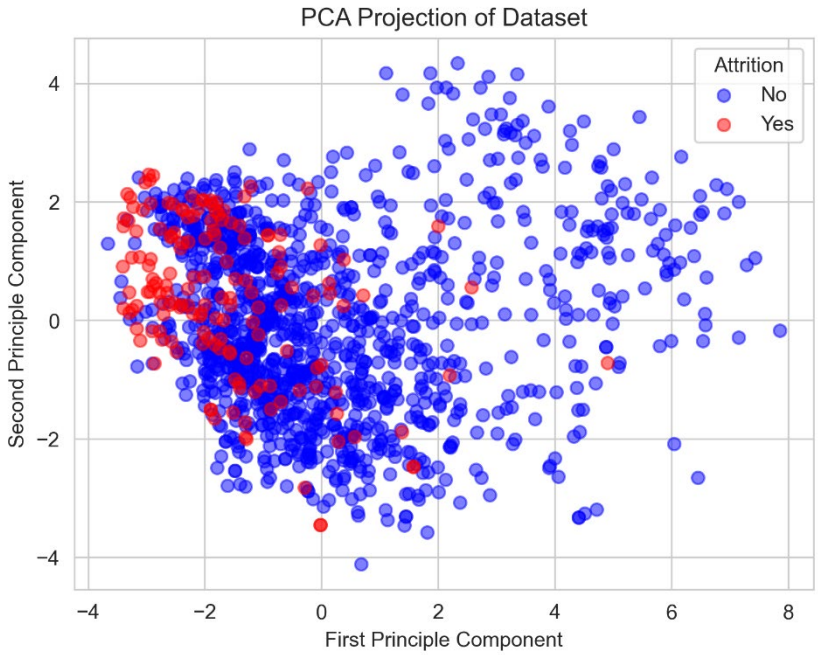


Figure 4: First two principal components of training data

I also plotted the data using t-distributed stochastic neighbor embedding (TSNE) [1] and Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP) [2]. Both TSNE and

UMAP are dimensionality-reduction algorithms which aim to preserve the local and global relationships among datapoints in a lower dimensional space. That is, both algorithms aim to generate a mapping in which similar points in high-dimensional space remain close to each other in the projected space.

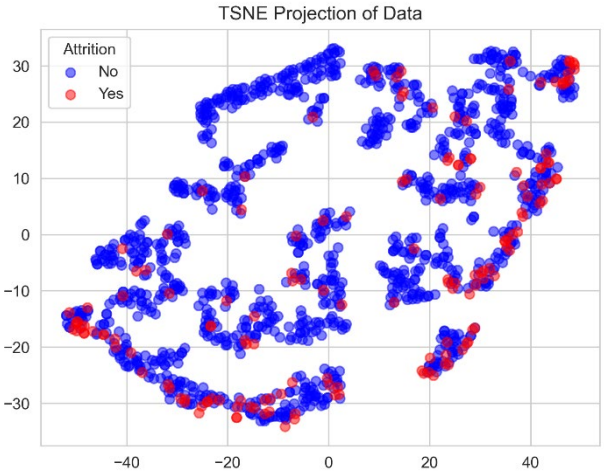


Figure 5: TSNE Projection of Training Data with Perplexity 30

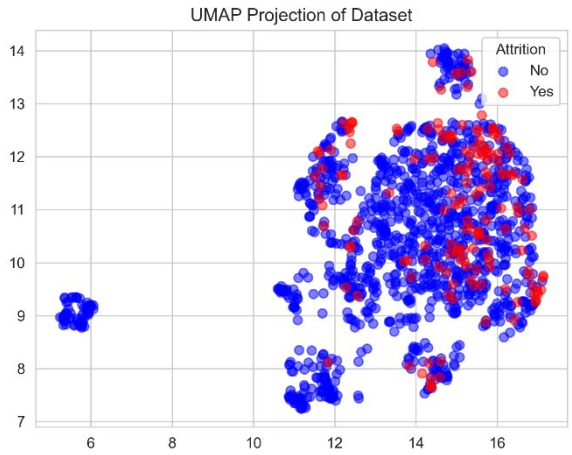


Figure 6: UMAP Projection of Training Data with Num Neighbors 15

In the above figures, it is apparent that neither TSNE nor UMAP generated a mapping which clearly separated the positive and negative samples. However, as is this case for the PCA projection, both projections appeared to have delineated pockets within the high-dimensional space in which positive samples are more likely to arise. For instance, in the TSNE projection, positive points are most highly concentrated in the outer right and bottom clusters. In the UMAP projection, positive points are clustered towards the right side of the central cluster of points. These projections would thus imply that positive samples do share, at least some, similarity which each other, and would provide evidence to support that a classifier may, at least partially, differentiate them from the negative samples.

I then determined a rough, preliminary, estimate for the relative predicted power of each variable by plotting ROC curves along the values for each feature individually. That is, I generated an ROC curve for each feature by sweeping a threshold across the unaltered values of each feature.

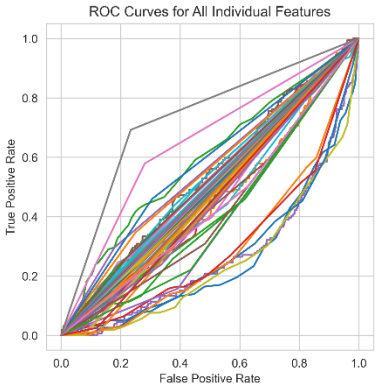


Figure 7: ROC Curves for All Features

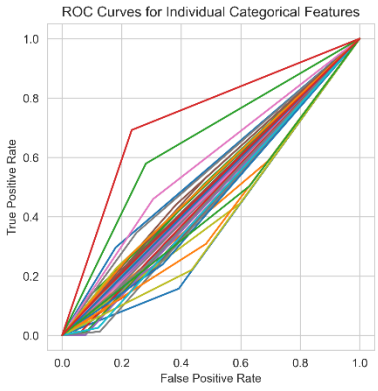


Figure 8: ROC Curves for Categorical Features

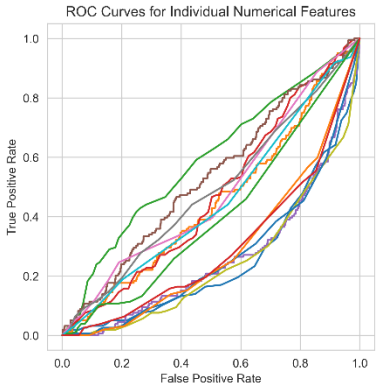


Figure 9: ROC Curves for Numerical Features

From the above curves, it is apparent the un-altered features provide a wide range of predictive power. Most features appear to offer to offer little to no individual predictive utility (when left unaltered), hence the clump of curves that stay close to the diagonal of the graph. However, a low AUC in the above graphs does not imply the feature does not predict employee attrition, as the above curves may fail to capture non-liner relationships between feature values and attrition likelihood. A select few features display a high degree of deviation from the central diagonal and may thus be construed to be highly important towards a predictive model. It may also be noted that many of the stronger ROC curves among the numerical features appear very similar to each other, which may suggest those features are correlated. I then chose to graph the 10 features with the highest AUC. In the below charts, it is apparent that the feature AUC values are highly right-skewed.

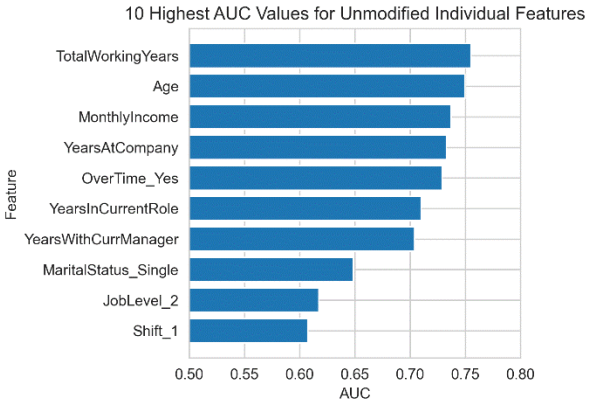


Figure 11: Graph of Individual Features with Highest AUC Score

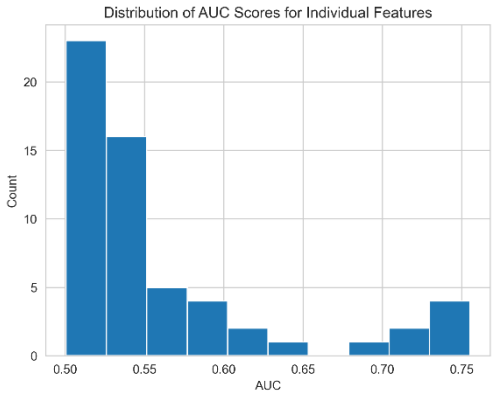


Figure 10: Histogram of Individual Feature AUC

Most features possess an AUC score below 0.6, although a small cluster of features possess AUC scores ranging from 0.7 to 0.75. The similarity of AUC values across these best-performing features again suggested the possibility these features were correlated. I believe the feature names supported this hypothesis, as many of the best-performing features would appear to display strong relationships. For instance, age can reasonably be associated with total working years, monthly income, and job level. Similarly, years in current role would likely be highly related to years with current manager. I then formally determined the correlations and visualized them with the below heatmap.

As suspected, many of the features with highest individual AUC scores display a high correlation with each other. For instance, Total Working Years displays high correlations (> 0.6) with Age, Job Level, Monthly Rate, and Years At Company, all of which are among the six best-performing features. As expected, the dummy variables generated from the same original feature also display high degrees of correlation with each other, which would potentially cause problems when constructing linear classifiers.

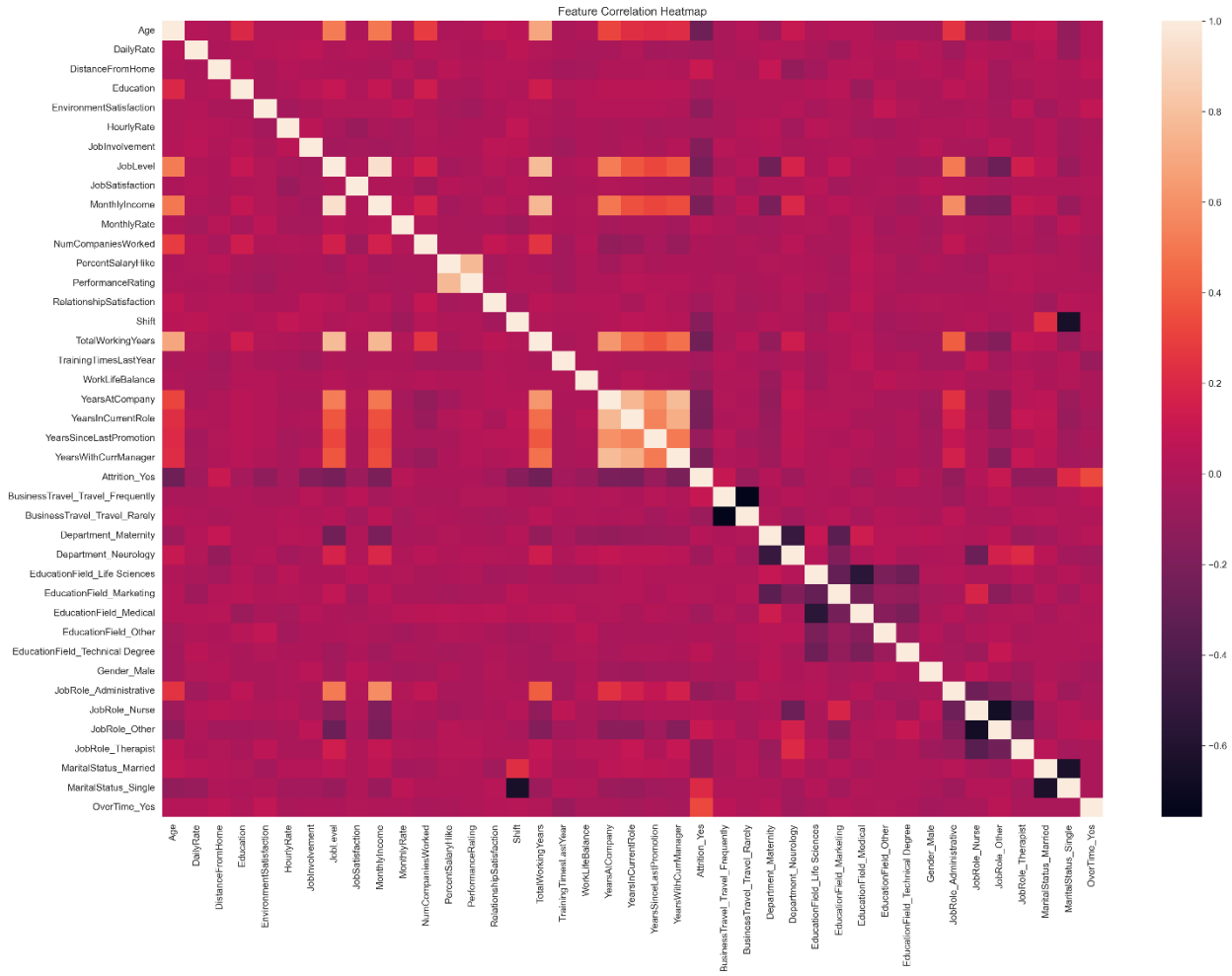


Figure 12: Heatmap of Correlations Between Each Pair of Features

Feature Engineering and Data Processing

I completed various data processing steps for each model I implemented. First, for all models, I converted categorical (and the above "ambiguous" variables for the logistic regression) to dummy variables through leave-one-out one-hot-encodings. After applying the one-hot encodings, the original 31 "good" features increased to 58 features in my final training set. Next, I addressed the imbalance in the target variable by using two up-sampling techniques to increase representation of the attrition-positive samples. First, I tried random up-sampling, which simply generates a balanced training set by repeatedly drawing samples randomly from the minority class and adding them to the new training set until the desired classes are balanced. I also implemented Synthetic Minority Over-Sampling Technique (SMOTE), which artificially generates new samples in the minority class. SMOTE generates new samples by considering the *k* closest neighbors around a given point and then placing a new point at the midpoint along the line between the given point and one of these *k* neighbors. After implementing either of these algorithms, my final dataset now consisted of an equal number of attrition-positive and attrition-negative samples (1181 of each). This way, my models would not simply "learn" the distribution of positive to negative samples in the training data. Finally, for my logistic regression model, I scaled the input data using Scikit-learn's Standard Scaler package to achieve consistent model weights during training.

Models

I chose to use three different algorithms to build classifiers: logistic regression, AdaBoost, and XGBoost. I first chose to use a logistic regression given its simplicity, ease of training, and highly interpretable results. A logistic regression also possesses a very simple parameterization, as one parameter, C, determines the regularization strength of the model. There are likely many logistic regression packages available on the internet, however, I chose Scikit-learn's Logistic Regression package [3] as I am most familiar with this implementation. This package offers several adjustable hyperparameters, including the type of regularization penalty, whether to fit an intercept, the desired solver, and maximum number of training iterations. In my model, I left most parameters to their default values, as I was not concerned with model runtime (our training set was sufficiently small), I sought to use l2 regularization (I did not seek a sparse result), and I chose to fit an intercept.

I then chose to implement AdaBoost as the logistic regression still left room to improve on its training F1 score. Since AdaBoost produces a non-linear decision function, it would potentially be able to achieve greater accuracy on the training data while hopefully still generalizing to the test data. AdaBoost also possesses a simple parameterization, involving just the choice of estimator, number of estimators, and the learning rate. I again used Scikit-learn's implementation of AdaBoost [3], which itself implements the AdaBoost-SAMME algorithm [4]. I used this package as I had used it previously in a homework assignment and I was not concerned with the solver's ability to handle the size of the training data. Scikit-learn's implementation includes each of the above parameters as adjustable options.

Finally, I trained an XGBoost classifier [5]. XGBoost is a package that implements gradient boosting to construct a classifier/regressor [6]. I chose to use this package as I have familiarity with XGBoost and the algorithm is known for strong performance on tabular datasets. XGBoost also runs quickly (although this would likely not be a concern anyway given the small size of our dataset), and from my experience, is relatively easy to implement. Although XGBoost does not have as similar a parameterization as a logistic regression or AdaBoost model, I felt confident in my ability to achieve strong performance by cross validating several key parameters of the model rather than all the given options.

[1] Y. Freund, R. Schapire, "A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting", 1995.
[2] Zhu, H. Zou, S. Rosset, T. Hastie, "Multi-class AdaBoost", 2009.

Training

I trained the Scikit-learn Logistic Regression classifier using the fit() function with the default solver, "lbfgs". LGBFS, or limited-memory Broyd-Fletcher-Goldfarb-Shanno algorithm [7]. The LGBFS algorithm is a quasi-Newton optimization method that approximates the GBFMS algorithm in order use less memory. A quasi-Newton method is one in which the hessian matrix of a given function week seek to minimize is not calculated in full (as in Newton's method). Instead, quasi-static methods form an approximation of the hessian to save computing time. The LGBFS algorithm's approximation of the Hessian matrix requires significantly less memory than storing the true Hessian matrix through a unique approximation formula outlined in [7]. Ultimately, LGBFS generally yields fast and efficient convergence, and in my case, produced the training time of any model tested (0.02 seconds)

I trained the Scikit-learn AdaBoost classifier using the fit() function with the default solver, "SAMME.R" [4]. SAMME.R, or Stagewise Additive Modeling using Multi-class Exponential loss function

(Real) is similar to the standard AdaBoost algorithm, except it offers a functionality for multi-class classification. This application did not rely on that functionality, however, as this project involved binary classification. If I had used the SAMME algorithm (rather than SAMME.R), it would have then reduced to standard AdaBoost [4]. SAMME.R differs from SAMME by using real-valued probability/confidence estimates for each point to update the model, whereas SAMME uses discrete prediction values to update the model. Ultimately, SAMME.R yields generally strong results on a wide range of data, and potentially converges faster than SAMME given its used of real-valued confidence estimates rather than discrete predictions to update the model during training. When I trained my AdaBoost model, SAMME.R did take the longest of all three models, producing a wall time of 0.262 seconds.

I trained the XGBoost classifier using the fit() function using the default training algorithm. XGBoost is a package for implementing gradient tree boosting. Tree boosting is an algorithm which build an additive model by iteratively constructing trees. The model updates itself by considering the gradient at each leaf node. XGBoost offers functionality for both exact tree boosting as well as variations which approximate an exact algorithm. In my case, as I chose default settings, the classifier chose to use either an exact or approximation version of gradient boosting according to a heuristic. XGBoost as a package also contains a number of optimizations which help the algorithm run quickly on a wide variety of data. In my case, the algorithm ran in 0.082 seconds, faster than AdaBoost, but slower than logistic regression.

Model	Training Wall Time on Available Test Data (seconds)
Logistic Regression	0.020
AdaBoost	0.262
XGBoost	0.082

Hyperparameter Selection

For the Logistic Regression, I used 5-fold cross validation to determine the optimal C parameter, which controls the model regularization strength. I also considered multiple values of SMOTE's k-

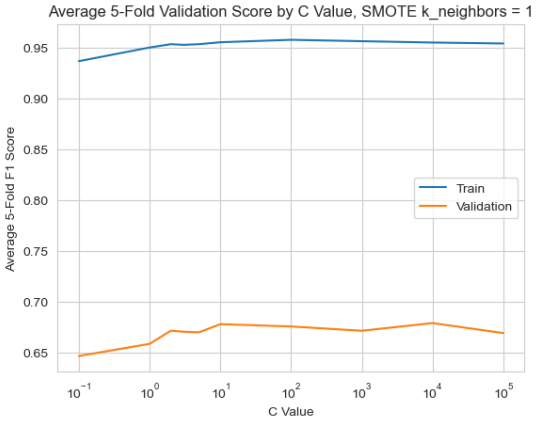


Figure 15: Logistic Regression C Cross Validation k_neighbors = 1

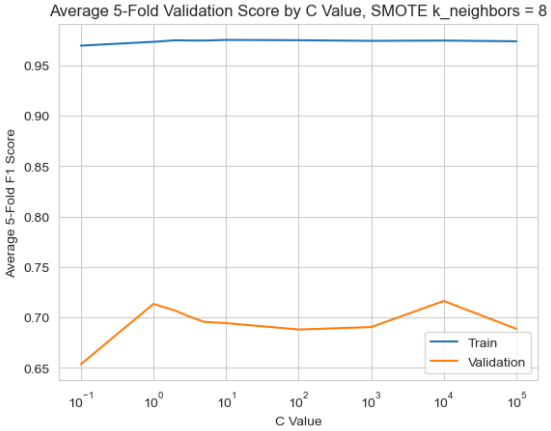


Figure 16: Logistic regression C cross validation k_neighbors = 8

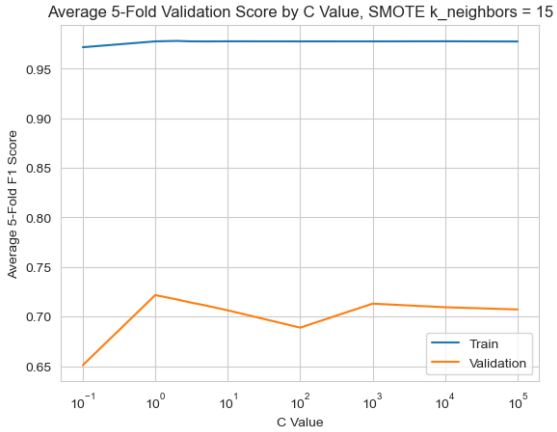


Figure 14: Logistic regression C cross validation k_neighbors = 15

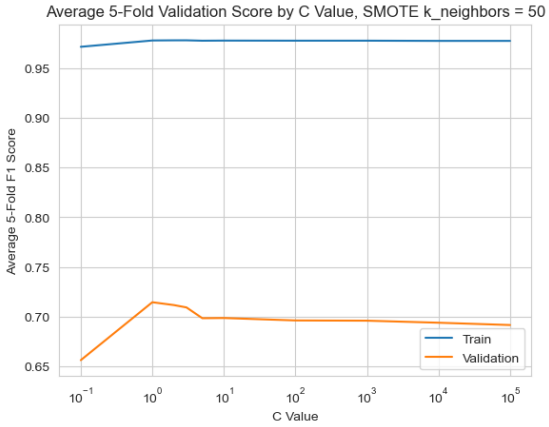


Figure 13:Figure 13: Logistic regression C cross validation k_neighbors = 50

neighbors parameter, which determines the number of closest neighbors SMOTE considers when generating artificial points. In theory, a higher k-neighbors value would produce artificial points that, on average, stray farther from each parent point. A higher k-neighbors value may then reduce the likelihood the model overfits to the training data as artificial points would span a broader range, and would collectively be more dissimilar to the original training set. I also tested standard random up-sampling by testing a k-neighbors value of 1 (i.e. SMOTE would only consider one point when generating a new sample).

I ultimately chose to set the k_neighbors parameter to 8 on my final logistic regression with a c-value of 1, as consistent with the highest validation score across all of the validation folds. From the above graphs, it is apparent that the logistic regression model performed extremely similarly across most values of C (except for the smallest tested values). Most values of k_neighbors also achieved similar results, except a k_neighbor value of 1 (corresponding to standard random sampling) performed

noticeably worse. I used the best k_neighbors value of 8 as found in these tests to build my subsequent AdaBoost and XGBoost models.

For the AdaBoost model, I implemented 5-fold grid-search cross validation on the n_estimators and learning rate parameters. The n_estimators parameter controls the maximum number estimators generated before the training process terminates. The learning rate parameter determines the relative contribution of each subsequent estimator during training. In theory, a lower n_estimators value and a lower learning rate value would lead to more conservative models.

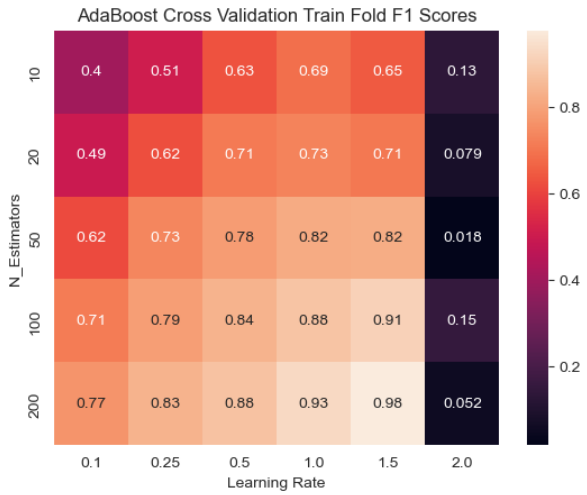


Figure 17: AdaBoost cross validation training f1 scores

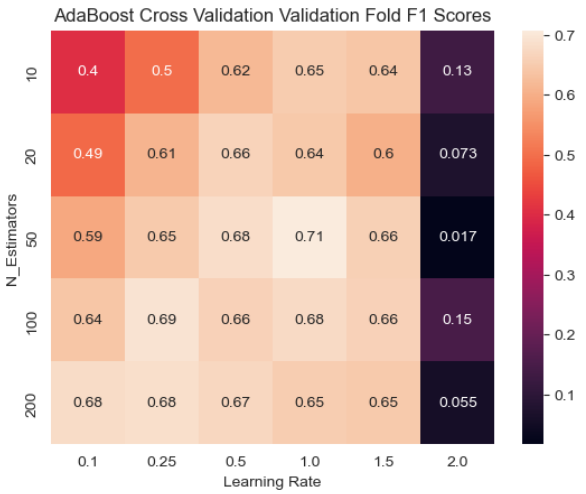


Figure 18: AdaBoost cross validation validation f1 scores

Over grid search cross validation, the AdaBoost model achieved the best validation score with a learning rate of 1 and an n_estimators value of 50. As expected, the model achieved higher training score on higher n_estimator values and higher learning_rate values (until a learning rate of 2.0, at which point the model performs significantly worse). However, it appears the best validation value fell in the middle of tested learning rate values (1.0) and n_estimator values (50), perhaps striking a balance between model bias and variance.

For the XGBoost model, I implemented 5-fold grid-search cross validation on the max-depth and gamma parameters. Max-depth determines the maximum depth of each tree generated by the XGBoost algorithm. Gamma determines the necessary reduction in loss for a split to take place within a given tree. In theory, decreasing max-depth and increasing gamma, respectively, would lead to more conservative models. After determining these best two parameters through an exhaustive grid search, I tested multiple values across many of the other XGBoost parameters in a “line-search” like fashion in which I manually changed a given parameter while holding the rest constant until I arrived at an apparent optimal validation score. My methodology in determining the remaining best parameters was thus less scientific, however, I was able to further increase the validation accuracy on my final XGBoost model using this methodology. Example parameters I tuned using this method include max_pos_weight, which effectively determines the minimum number of training points in each tree leaf, eta, the model learning rate, and subsample, which determines the size of a random sample on which each XGBoost model trains.

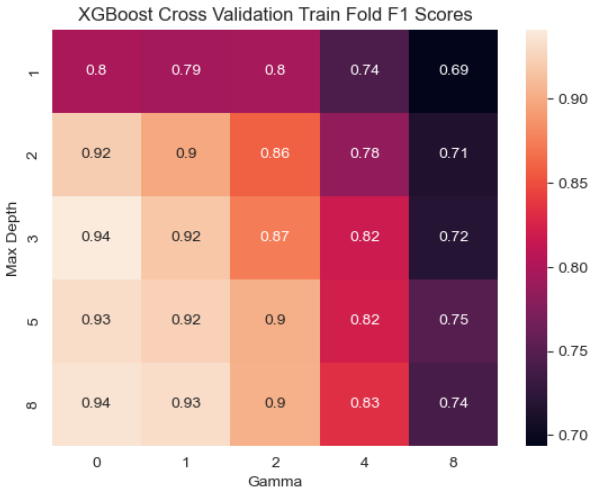


Figure 20: XGBoost cross validation training f1 scores

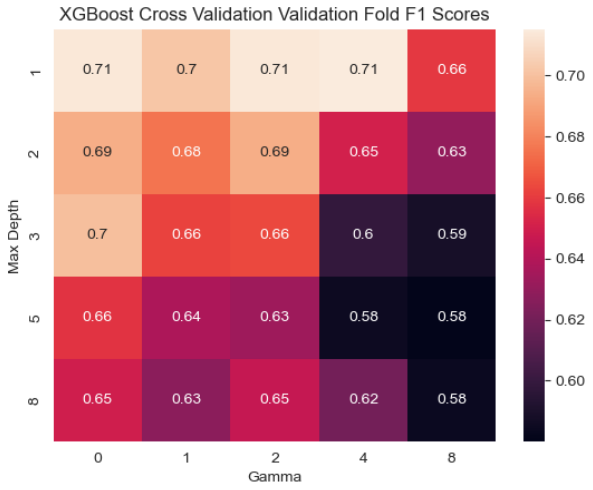


Figure 19: XGBoost cross validation validation f1 scores

Over grid search cross validation, the XGBoost model achieves higher training f1 scores on lower values of gamma and higher max_depth values, as expected. The XGBoost model performed best on the validation sets when gamma was set to 1 and the max depth was set to 1, meaning the best XGBoost model consisted entirely of 1-split decision stumps. From my additional testing, I found the following values worked well or some of the remaining parameters: eta: 0.2, scale_pos_weight: 5, reg_lambda: 4, min_child_weight: 20, subsample: 0.5, n_estimators: 200.

Data Splits

When completing k-fold cross-validation for each model, I attempted to create a train-validation splits which reflected the performance of the model on test data as closely as possible. I used 5-fold cross validation, which split the entire original training data into 5 equally sized folds. For each split, I used four folds as training data and the remaining fold for validation data. I then up-sampled the training data using SMOTE to achieve balanced class labels (I did not up-sample the validation data so as to achieve a more accurate representation of the test data). For the logistic regression model, I also scaled the data before up-sampling it. I scaled the training fold and validation folds separately so that now information was "leaked" from the training data to the validation data. I let the outer loop of my cross-validation script loop through the parameters I wanted to test for each model, while the inner fold iterated through the 5-folds of the data. For each inner loop, I calculated both the training validation f1 scores of the model. For each outer loop, I found the average of each of the inner loop scores and appended these values to an array of average training and validation scores for each value of the hyperparameter I was cross-validating. I then created plots of the training and validation scores for each hyperparameter I tuned.

Errors and Mistakes

Throughout this project, I made several mistakes subsequent realizations. One of the first mistakes I made was to not resample my data before training my models. At first, I achieved extremely low f1-scores that fell well-below the baseline on the Kaggle leaderboard. I did not initially understand the cause of the low scores, as I felt I was training the models correctly on the training data. I quickly realized that my models were likely "learning" the distribution of the training data given the highly

unbalanced 'Attrition" target feature. As a result, although my models would likely have produced highly accurate predictions, even on the test data, their f1-scores would likely be low given a tendency to misclassify attrition-positive samples as attrition-negative (meaning recall of my model would have likely been very low). I first attempted to solve this problem by adjusting the data weights so that the models considered positive samples with weight inversely proportional to their representation in the dataset. Adjusting the model weights led to immediate improvement in my f1 scores. In addition to applying weights to the data, I then tried up-sampling techniques as described above. I discovered that resampling seemed to perform even better than just weighting the data points. I believe SMOTE may have outperformed data weighting by generating artificial points which were not identical to the original datapoints, which could perhaps reduce my models' likelihood of overfitting the training data.

Another mistake I made was to forget to scale my data in various situations. Originally, I proceeded with completely unscaled data when building my models. Although unscaled data was likely not a problem for tree-based models, I believe my used of unscaled data led to poor performance on my logistic regression model. When I had first tested a logistic regression in my exploratory data analysis, I believe I also failed to up-sample the training data. My resulting f1 score was so low compared to tree-based methods that I dismissed the logistic regression as a viable model in this project until much later in my work. After I made the corrections to up-sample and scale my data, the logistic regression performed better than any other model I had tested.

I was very surprised to see the logistic regression perform so well, and I initially thought the high-validation scores compared to the tree-based models reflected a mistake in my code. However, upon submitting logistic regression predictions on the test data, I found the model was in fact performing better. I am unsure why the logistic regression performed so well in this competition, however, upon examining the feature importances across each model type, I believe one reason could be the fact that the decision tree models tended to "ignore" the less predictive features. While the logistic regression assigned at least some weight to every feature in the dataset, the tree-based models were not forced to split on any given feature. I believe the tree-based models then concentrated their splits around the most highly predictive features, which, as shown above, may have been highly correlated with each other. In effect, I think the tree-based models were losing out on information that could be drawn from the less-predictive, but potentially uncorrelated, features in the training data. As a result, since the logistic regression considered all features, it was better able to distinguish positive and negative samples despite only being able to construct a linear decision function. I think the nature of the data itself may have also contributed to this result, as I do not believe many variables held a clearly non-linear (or at least non-monotonic) relationship to the target variable. Especially after assigning one-hot encodings to the categorical and "ambiguous" features, I think a logistic regression would thus have been highly capable of fitting the training data.

Predictive Accuracy

I submitted predictions on Kaggle for each of the models trained above, achieving varying f1 scores on the available test data.

Model	Best F1 Score on 5-Fold CV	Best F1 Score on ½ of Test Data
Logistic Regression	0.72	0.75
AdaBoost	0.71	0.60
XGBoost	0.71	0.67

I found the models' performance on the available test data to be surprising in how variable each of the models performed despite having similar best 5-fold cross validation values. For instance, the logistic regression model significantly outperformed both the best AdaBoost and XGBoost models on the test data, despite only having a 5-fold cross validation f1 score that was 0.01 greater. Given this difference, there is a possibility my logistic regression model performed better on the available test data by chance and would thus potentially have lower results on the other half of the test data. However, I do not think my logistic regression "overfit" the test data, as the submitted model had a high regularization parameter, and upon submitting slightly tweaked iterations of a similar logistic regression model, it still achieved relatively high performance. I also examined confusion matrices for a train-validation split of 0.8 to 0.2 for each of the models.

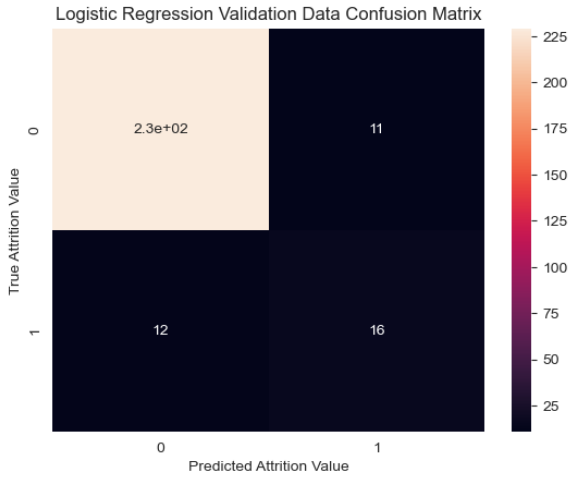


Figure 23: Logistic regression validation set confusion matrix

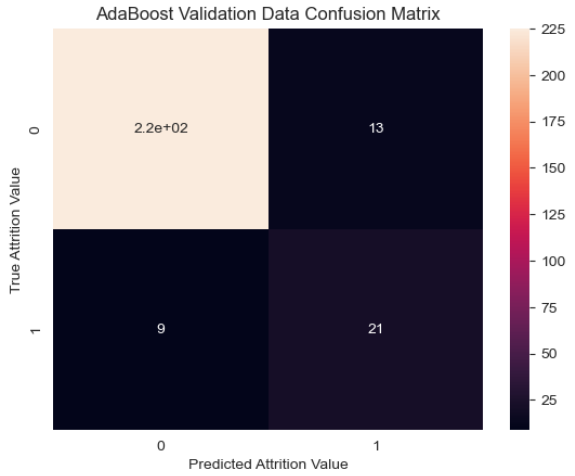


Figure 22: AdaBoost validation set confusion matrix

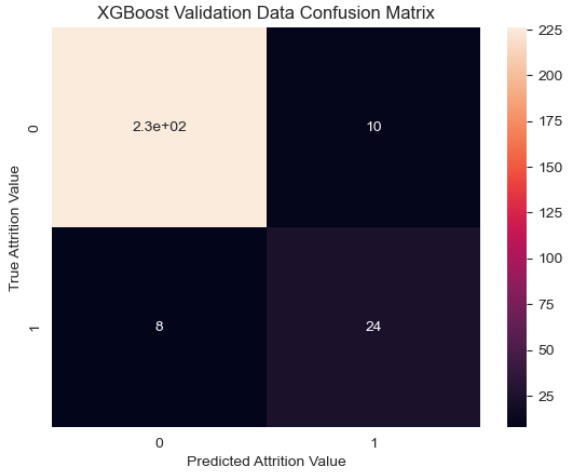


Figure 21: XGBoost validatoin set confusion matrix

From the above matrices, it is visible that each model produced a proportionally similar number of false positive and false negative predictions. Since the goal of this project is to maximize f1 score, this is thus a desirable output (so that the harmonic mean of precision and recall would be maximized). It should be noted, however that each of the validation sets used to generate these matrices potentially

contained a different number of positive and negative samples, meaning the absolute number of predictions in each of the categories in the above matrices are not directly comparable.

Out of curiosity as to how my models were functioning, I examined the feature importance/coefficient values across each feature for each model on the test data (cont. on page 15).

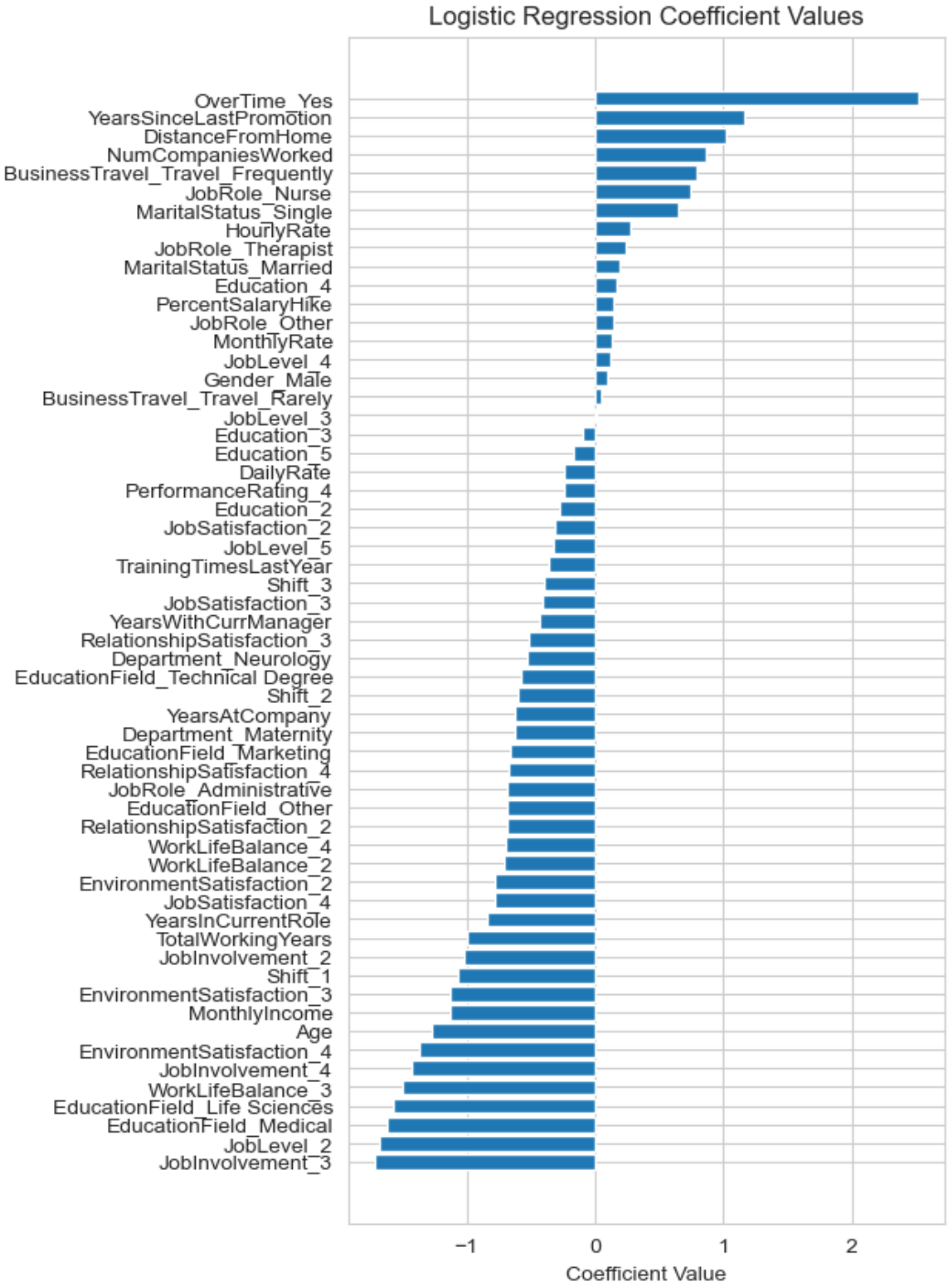


Figure 24: Logistic regression coefficient values on available test data

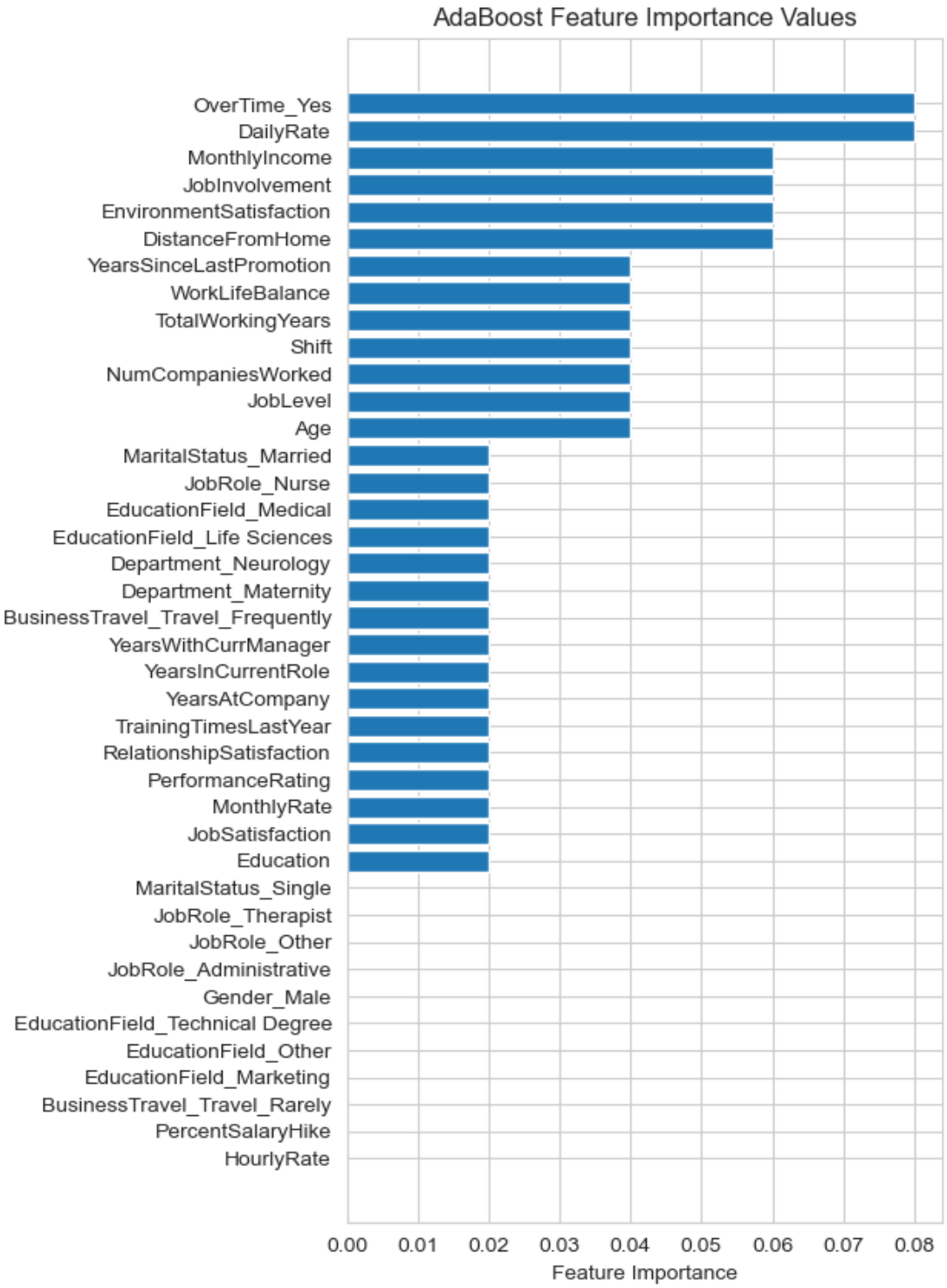


Figure 25: AdaBoost feature importance values on available test data

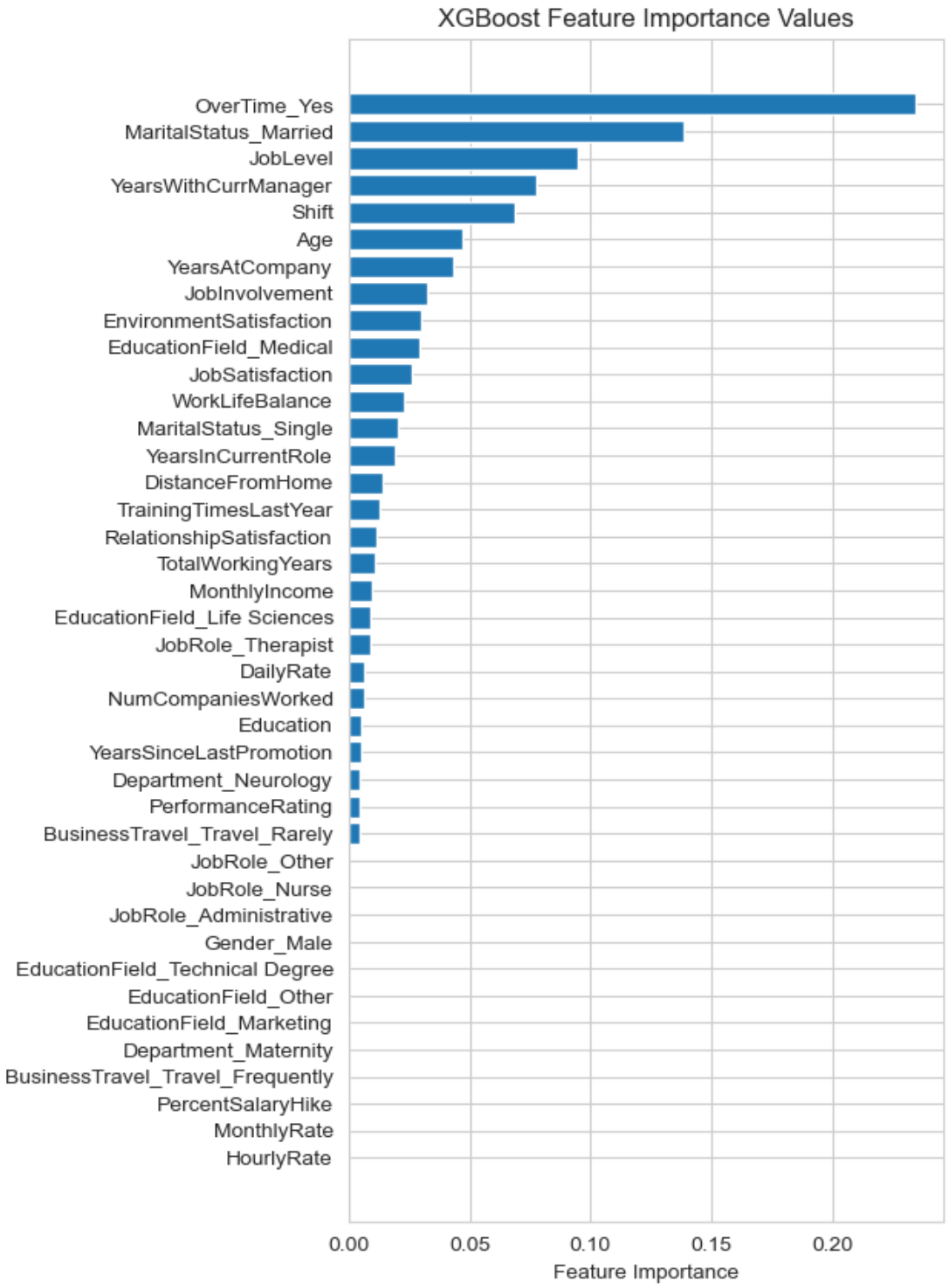


Figure 26: XGBoost feature importance values on available test data

From the above charts, it is apparent that each model assigns variable importance to each feature. The AdaBoost and XGBoost model are most similar in their high dependence on a select few key features, many of which overlap between the two models. The logistic regression, however, places some coefficient on almost every feature and tends to assign coefficient values such that the features are

considered more equally in the final model. All three models agreed on "OverTime_Yes" as the feature with highest importance but varied in their ranking of further feature importances.

References

[1] L. Maaten and G. Hinton, "Visualizing Data using t-SNE," *Journal of Machine Learning Research,* vol. 9, no. 86, pp. 2579-2605, 2008. [Online]. Available: http://jmlr.org/papers/v9/vandermaaten08a.html.

[2] L. McInnes, J. Healy, and J. Melville, *UMAP: Uniform Manifold Approximation and Projection for Dimension Redu ction*. arXiv.

[3] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

[4] T. Hastie, S. Rosset, J. Zhu, and H. Zou, "Multi-class AdaBoost," *Statistics and Its Interface,* vol. 2, no. 3, pp. 349-360, 2009-01-01 2009, doi: 10.4310/sii.2009.v2.n3.a8.

[5] T. Chen and C. Guestrin, "XGBoost," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016-08-13 2016: ACM, doi: 10.1145/2939672.2939785.

[6] "dmlc/xgboost." GitHub. https://github.com/dmlc/xgboost (accessed 12/9/2022, 2022).

[7] J. Nocedal, "Updating quasi-Newton matrices with limited storage," *Mathematics of Computation,* vol. 35, no. 151, pp. 773-782, 1980-01-01 1980, doi: 10.1090/s0025-5718-1980-0572855-7.

Code

Below are five notebooks I used along this project. Here is a table summarizing the contents of each notebook (in order they are attached below):

File Name	Contents
EDA Code.ipynb	Contains exploratory data analysis as well as some initial attempts training models
Ambiguous Variable One Hot Code.ipynb	Creates dummy variables for the "ambiguous" variables I discussed in the above exploratory data analysis section. Repeats much of the contents of EDA Code.ipynb except with modified dataset with ambiguous feature dummy variables. Also tests XGBoost model with PCA.
Logistic Regression Code.ipynb	Trains a logistic regression using the dataset where categorical and ambiguous features have been encoded as one-hot dummy variables. Runs 5-fold cross validation and generates predictions.
AdaBoost Code.ipynb	Trains an AdaBoost classifier using the dataset where categorical features have been encoded as one-hot dummy variables. Runs 5-fold cross validation and generates predictions.
XGBoost Modelling Code.ipynb	Trains an XGBoost classifier using the dataset where categorical features have been encoded as one-hot dummy variables. Runs 5-fold cross validation and generates predictions.

In [55]:
```
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
from xgboost import XGBClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.manifold import TSNE
import umap
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import warnings
warnings.filterwarnings("ignore")
plt.rcParams['savefig.dpi'] = 300
```

In [56]:
```
# Load in data, remove useless features, assign dummy variables to categorical features
train = pd.read_csv('train.csv')
drop_features = ['Over18', 'EmployeeCount', 'StandardHours', 'EmployeeID']
good_features = [c for c in train.columns if c not in drop_features]
train_dummies = pd.get_dummies(train[good_features], drop_first=True)
train_y = train_dummies.Attrition_Yes
train_x = train_dummies.loc[:, train_dummies.columns != 'Attrition_Yes']
x_train, x_val, y_train, y_val = train_test_split(train_x, train_y, test_size = 0.25)

numerical_features = train[good_features].select_dtypes(include=np.number).columns
categorical_features = [f for f in x_train.columns if f not in numerical_features]

test_x_original = pd.read_csv('test.csv')
test_features = [c for c in good_features if c != 'Attrition']
test_x = pd.get_dummies(test_x_original[test_features], drop_first=True)
```

In [57]:
```
# Look at shape of training data
train.shape
```

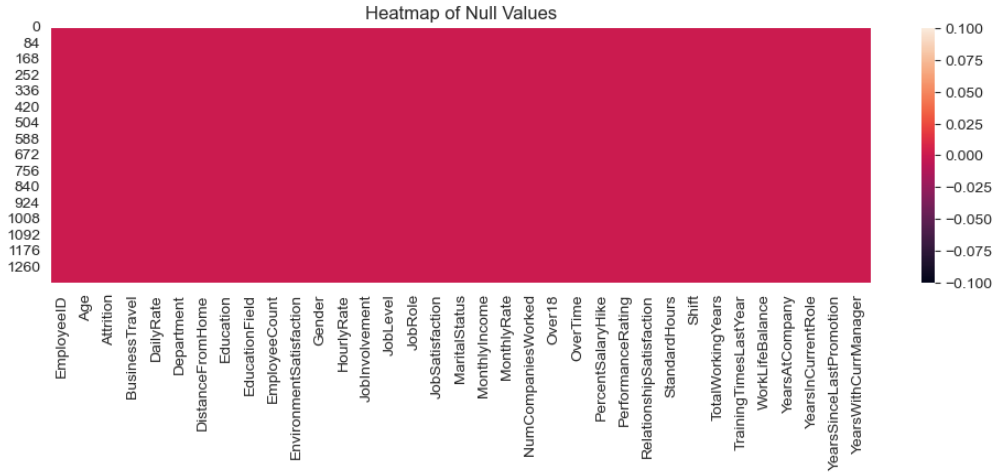
Out[57]:
```
(1340, 35)
```

In [58]:
```
# Look at shape of test data
test_x.shape
```

Out[58]:
```
(336, 40)
```

In [59]:
```
# Create heatmap of null values
plt.figure(figsize=(12,3))
sns.heatmap(train.isna())
plt.title('Heatmap of Null Values')
# plt.savefig('Graphs/NaNheatmap.png', bbox_inches='tight')
```

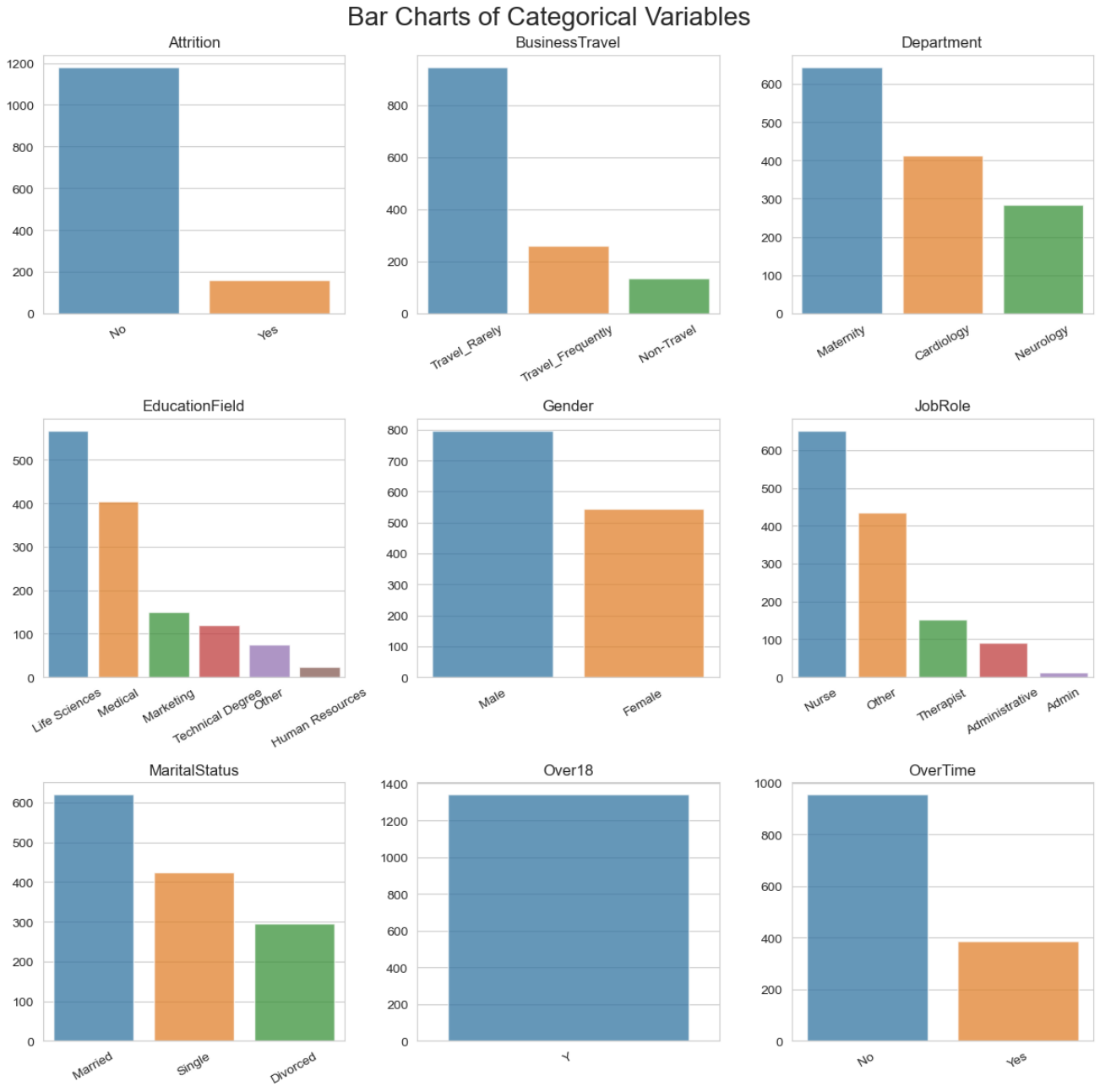
Out[59]:
```
Text(0.5, 1.0, 'Heatmap of Null Values')
```



In [60]:
```
# Create histograms for each numerical variable in training data
num_histograms = plt.figure(figsize=(12,14))
# cat_barplots.tight_layout(pad=100)
plt.suptitle('Histograms of Numerical Variables', fontsize=20)
for i, col in enumerate(train.select_dtypes(include='number').columns):
    plt.subplot(7,4,i+1)
    plt.title(col)
    sns.histplot(train[col])
    plt.xticks(rotation=30)

plt.tight_layout()

# plt.savefig('Graphs/NumHistograms2.png')
```

In [61]:

```
# Create bar charts for each categorical variable in training data
cat_barplots = plt.figure(figsize=(12,12))
plt.suptitle("Bar Charts of Categorical Variables", fontsize=20)
# cat_barplots.tight_layout(pad=100)
for i, col in enumerate(train.select_dtypes(exclude='number').columns):
    plt.subplot(3,3,i+1)
    plt.title(col)
    col_counts = train[col].value_counts()
    sns.barplot(x=col_counts.index, y=col_counts.values, alpha=0.75)
    plt.xticks(rotation=30)

plt.subplots_adjust(hspace=0.5)
plt.tight_layout()
# plt.savefig('Graphs/BarChartsCat.png')
```

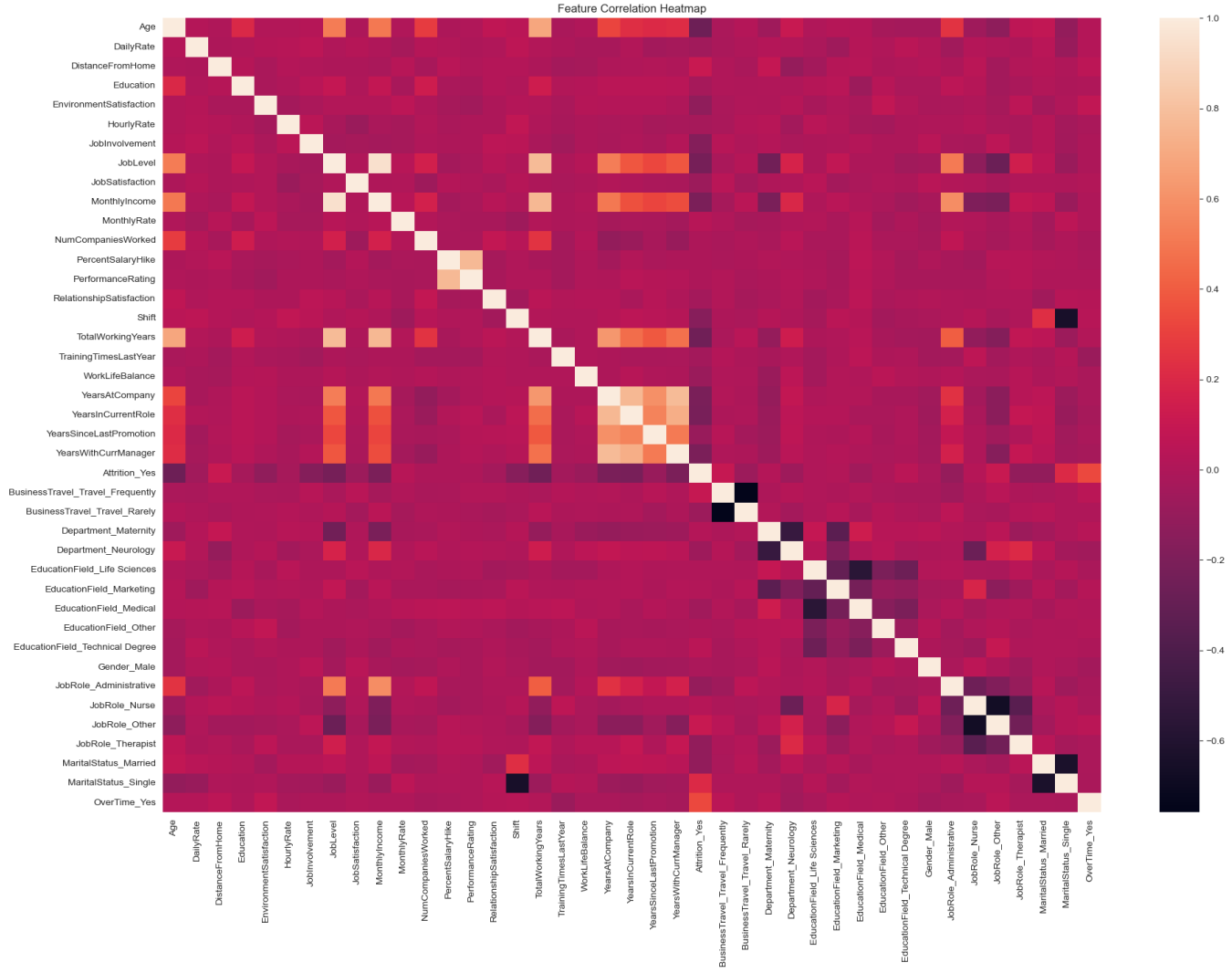


In [62]:
```
# Look at correlations to target variable
train_dummies.corr().Attrition_Yes.abs().sort_values(ascending=False)
```

Out[62]:
```
Attrition_Yes                       1.000000
OverTime_Yes                        0.327144
Age                                 0.269277
TotalWorkingYears                   0.254779
JobLevel                            0.226127
YearsInCurrentRole                  0.216750
YearsAtCompany                      0.213344
YearsWithCurrManager                0.207276
MonthlyIncome                       0.207201
MaritalStatus_Single                0.206861
JobInvolvement                      0.167779
Shift                               0.157736
EnvironmentSatisfaction             0.121397
JobRole_Therapist                   0.116692
MaritalStatus_Married               0.113704
DistanceFromHome                    0.112645
JobRole_Other                       0.106040
JobRole_Administrative              0.099041
BusinessTravel_Travel_Frequently    0.095071
YearsSinceLastPromotion             0.087319
WorkLifeBalance                     0.084726
BusinessTravel_Travel_Rarely        0.077239
JobSatisfaction                     0.074069
Department_Neurology                0.065472
DailyRate                           0.055333
EducationField_Medical              0.055000
TrainingTimesLastYear               0.049233
EducationField_Technical Degree     0.045432
RelationshipSatisfaction            0.041260
Education                           0.035685
HourlyRate                          0.033830
MonthlyRate                         0.033675
JobRole_Nurse                       0.031187
Department_Maternity                0.021178
EducationField_Marketing            0.016113
Gender_Male                         0.011517
EducationField_Other                0.009028
PercentSalaryHike                   0.008080
EducationField_Life Sciences        0.008042
PerformanceRating                   0.007747
NumCompaniesWorked                  0.004108
Name: Attrition_Yes, dtype: float64
```

In [63]:
```
# Visualize correlation matrix of training data
plt.figure(figsize=(22,15))
sns.heatmap(train_dummies.corr())
plt.title('Feature Correlation Heatmap')
# plt.savefig('Graphs/correlationheatmap.png',bbox_inches='tight')
```

Out[63]:
```
Text(0.5, 1.0, 'Feature Correlation Heatmap')
```

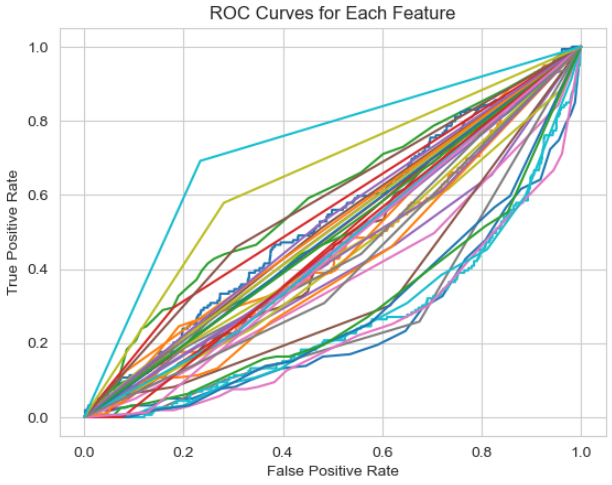


In [64]:
```
train_dummies.corr().Attrition_Yes.sort_values(ascending=False)[:20].index.values
```

Out[64]:
```
array(['Attrition_Yes', 'OverTime_Yes', 'MaritalStatus_Single',
       'DistanceFromHome', 'JobRole_Other',
       'BusinessTravel_Travel_Frequently',
       'EducationField_Technical Degree', 'MonthlyRate', 'JobRole_Nurse',
       'Department_Maternity', 'EducationField_Marketing',
       'EducationField_Life Sciences', 'NumCompaniesWorked',
       'PerformanceRating', 'PercentSalaryHike', 'EducationField_Other',
       'Gender_Male', 'HourlyRate', 'Education',
       'RelationshipSatisfaction'], dtype=object)
```

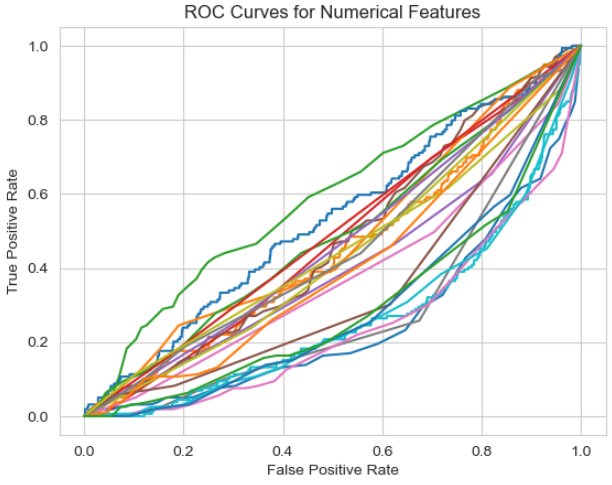
In [65]:
```
# Plot roc curves for all features
auc_scores = []
for i, feature in enumerate(x_train.columns):
    fpr_train, tpr_train, thresholds_train = roc_curve(train_y, train_x[feature])
    auc_scores.append((feature, roc_auc_score(train_y, train_x[feature])))
    plt.plot(fpr_train, tpr_train, label=feature)
plt.title('ROC Curves for Each Feature')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
# plt.legend()
```

Out[65]:
```
Text(0, 0.5, 'True Positive Rate')
```



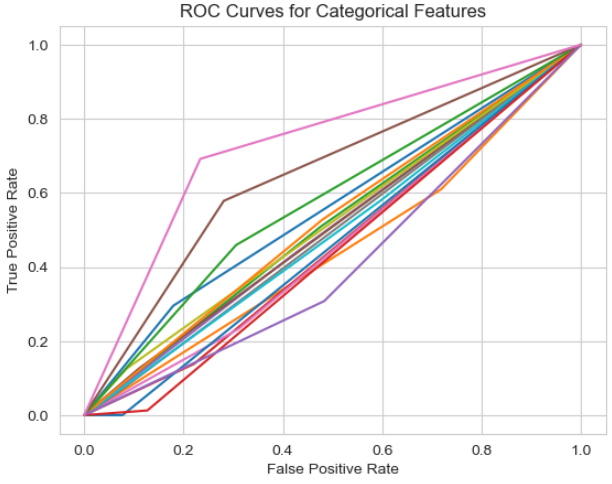
In [66]:
```
# Plot ROC curves for numerical features
for i, feature in enumerate(numerical_features):
    fpr_train, tpr_train, thresholds_train = roc_curve(train_y, train_x[feature])
    plt.plot(fpr_train, tpr_train, label=feature)
plt.title('ROC Curves for Numerical Features')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
# plt.savefig('Graphs/FeatureROCNum.png')
# plt.legend()
```

Out[66]:
```
Text(0, 0.5, 'True Positive Rate')
```



In [67]:
```
# Plot ROC curves for categorical features
for i, feature in enumerate(categorical_features):
    fpr_train, tpr_train, thresholds_train = roc_curve(train_y, train_x[feature])
    plt.plot(fpr_train, tpr_train, label=feature)
plt.title('ROC Curves for Categorical Features')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
# plt.savefig('Graphs/FeatureROCCat.png')
# plt.legend()
```

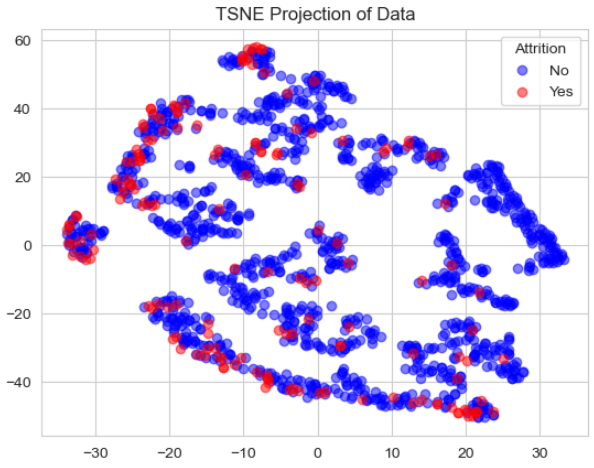
Out[67]:
```
Text(0, 0.5, 'True Positive Rate')
```



In [68]:
```
auc_scores.sort(key=lambda x: abs(x[1]-0.5), reverse=True)
predictive_features = [x[0] for x in auc_scores[:20]]
```

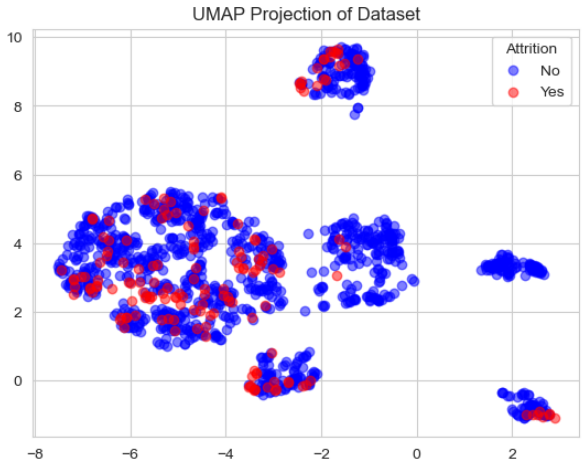
In [69]:
```
# Visualize data using TSNE
x_tsne = TSNE(perplexity=30).fit_transform(train_x)
plt.scatter(x=x_tsne[train_y == 0][:,0], y=x_tsne[train_y == 0][:,1], c='blue', alpha=0.5, label='No')
plt.scatter(x=x_tsne[train_y == 1][:,0], y=x_tsne[train_y == 1][:,1], c='red', alpha=0.5, label='Yes')
plt.legend(title='Attrition')
plt.title("TSNE Projection of Data")
# plt.savefig('Graphs/tsne2dp30.png')
```

Out[69]:
```
Text(0.5, 1.0, 'TSNE Projection of Data')
```



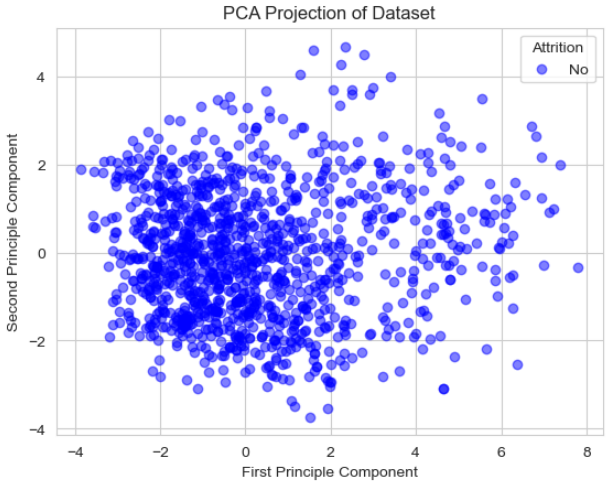
In [70]:
```
# Visualize data using UMAP
x_train_scaled = StandardScaler().fit_transform(train_x)
x_umap = umap.UMAP().fit_transform(x_train_scaled)
plt.scatter(x=x_umap[train_y == 0][:,0], y=x_umap[train_y == 0][:,1], c='blue', alpha=0.5, label='No')
plt.scatter(x=x_umap[train_y == 1][:,0], y=x_umap[train_y == 1][:,1], c='red', alpha=0.5, label='Yes')
plt.legend(title='Attrition')
plt.title("UMAP Projection of Dataset")
# plt.savefig('Graphs/umap2dpdefault.png')
```

Out[70]:
```
Text(0.5, 1.0, 'UMAP Projection of Dataset')
```



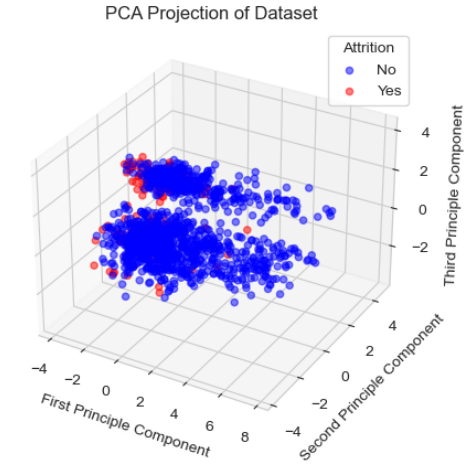
In [71]:
```
# Visualize data using PCA
x_pca = PCA(n_components=2).fit_transform(x_train_scaled)
plt.xlabel('First Principle Component')
plt.ylabel('Second Principle Component')
plt.scatter(x=x_pca[train_y == 0][:,0], y=x_pca[train_y == 0][:,1], c='blue', alpha=0.5, label='No')
# plt.scatter(x=x_pca[train_y == 1][:,0], y=x_pca[train_y == 1][:,1], c='red', alpha=0.5, label='Yes')
plt.legend(title='Attrition')
plt.title("PCA Projection of Dataset")
# plt.savefig('Graphs/pca2dNo.png')
```

Out[71]:
```
Text(0.5, 1.0, 'PCA Projection of Dataset')
```



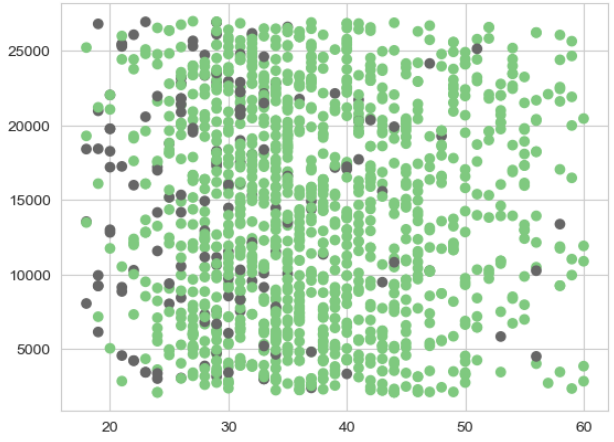
In [72]:
```
# Visualize data in 3D using PCA
x_pca = PCA(n_components=3).fit_transform(x_train_scaled)
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.scatter(x_pca[train_y == 0][:,0], x_pca[train_y == 0][:,1], x_pca[train_y==0][:,2], c='blue', alpha=0.5, label='No')
ax.scatter(x_pca[train_y == 1][:,0], x_pca[train_y == 1][:,1], x_pca[train_y==1][:,2], c='red', alpha=0.5, label='Yes')
plt.xlabel('First Principle Component')
plt.ylabel('Second Principle Component')
ax.set_zlabel('Third Principle Component')
# plt.scatter(x=x_pca[train_y == 1][:,0], y=x_pca[train_y == 1][:,1], c='red', alpha=0.5, label='Yes')
plt.legend(title='Attrition')
plt.title("PCA Projection of Dataset")
# plt.savefig('Graphs/pca3dBoth.png')
```

Out[72]:
```
Text(0.5, 0.92, 'PCA Projection of Dataset')
```



In [73]:
```
# Scatter plot of Age vs. Monthly Rate colored by Attrition Value
plt.scatter(x=train_x.Age, y=train_x.MonthlyRate, c=train_y, cmap='Accent')
```

Out[73]:
```
<matplotlib.collections.PathCollection at 0x21972817220>
```



In [74]:
```
# Fit logistic regression model to training data
model = LogisticRegression(class_weight='balanced').fit(x_train, y_train)
f1_score(model.predict(x_val), y_val)
```

Out[74]:
```
0.33566433566433573
```

In [75]:
```
# Fit random forest model to training data
rf_model = RandomForestClassifier(class_weight='balanced', max_depth=6).fit(x_train, y_train)
f1_score(rf_model.predict(x_val), y_val)
```

Out[75]:
```
0.5675675675675675
```

In [76]:
```
# Fit XGBoost model to training data
scaling = y_train.value_counts()[0] / y_train.value_counts()[1]
xgb_model = XGBClassifier(scale_pos_weight=scaling, max_depth=5).fit(x_train[predictive_features], y_train)
f1_score(xgb_model.predict(x_val[predictive_features]), y_val)
```

Out[76]:
```
[20:00:22] WARNING: C:\Windows\Temp\abs_557yfx631l\croots\recipe\xgboost-split_1659548953302\work\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
0.6052631578947368
```

In [77]:
```
# Create initial XGBoost predictions to compare result with baseline on Kaggle leaderboard
submission6 = pd.DataFrame(xgb_model.predict(test_x[predictive_features])).rename(columns={0: 'Predicted'})
submission6.index.rename('Id', inplace=True)
submission6.to_csv('submission6.csv')
```

In [78]:
```
# Look at XGBoost model feature importances
xgb_model.get_booster().get_score(importance_type='total_gain')
```

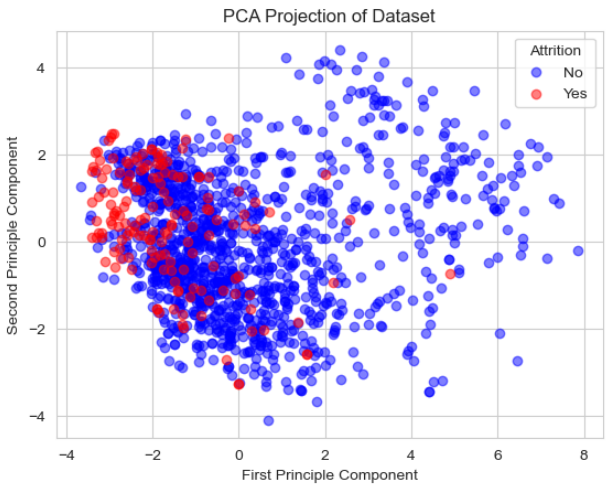
Out[78]:
```
{'TotalWorkingYears': 479.4785461425781,
 'Age': 663.6758422851562,
 'MonthlyIncome': 396.1846618652344,
 'YearsAtCompany': 308.6967468261719,
 'OverTime_Yes': 869.488037109375,
 'JobLevel': 74.30020141601562,
 'YearsInCurrentRole': 72.86554718017578,
 'YearsWithCurrManager': 57.48765182495117,
 'Shift': 182.5340118408203,
 'MaritalStatus_Single': 107.9872817993164,
 'JobInvolvement': 193.6339111328125,
 'EnvironmentSatisfaction': 187.33905029296875,
 'YearsSinceLastPromotion': 98.68278503417969,
 'DistanceFromHome': 466.9065856933594,
 'MaritalStatus_Married': 40.28858947753906,
 'JobRole_Other': 30.742328643798828,
 'JobSatisfaction': 105.19133758544922,
 'BusinessTravel_Travel_Frequently': 42.11652374267578,
 'JobRole_Therapist': 22.300079345703125,
 'WorkLifeBalance': 228.75448608398438}
```

In [25]:
```
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
from xgboost import XGBClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.manifold import TSNE
import umap
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib as mpl
from sklearn.utils import resample
import warnings
warnings.filterwarnings("ignore")
plt.rcParams['savefig.dpi'] = 300
```

In [14]:
```
# Load in dataset while taking into account ambigious features
train = pd.read_csv('train.csv')
drop_features = ['Over18', 'EmployeeCount', 'StandardHours', 'EmployeeID']
good_features = [c for c in train.columns if c not in drop_features]
ambiguous_features = ['Education', 'EnvironmentSatisfaction', 'JobInvolvement', 'JobLevel', 'JobSatisfaction', 'PerformanceRating', 'RelationshipSatisfaction', 'Shift', '

numerical_features = list(set(train[good_features].select_dtypes(include=np.number).columns) - set(ambiguous_features))
categorical_features = [f for f in train.columns if f not in numerical_features and f not in drop_features]
catx_features = [f for f in categorical_features if f != 'Attrition']

train_dummies = pd.get_dummies(train[good_features], columns=categorical_features, drop_first=True)
# amb_train_dummies = pd.get_dummies(train_dummies, columns=ambiguous_features, drop_first=True)

train_y = train_dummies.Attrition_Yes
train_x = train_dummies.loc[:, train_dummies.columns != 'Attrition_Yes']
x_train, x_val, y_train, y_val = train_test_split(train_x, train_y, test_size = 0.25)

catxt_features = [f for f in train_x.columns if '_' in f]
numxt_features = [f for f in train_x if f not in catxt_features]

test_x_original = pd.read_csv('test.csv')
test_features = [c for c in good_features if c != 'Attrition']
test_cat = [c for c in categorical_features if c != 'Attrition']
test_x = pd.get_dummies(test_x_original[test_features], columns=test_cat, drop_first=True)
```

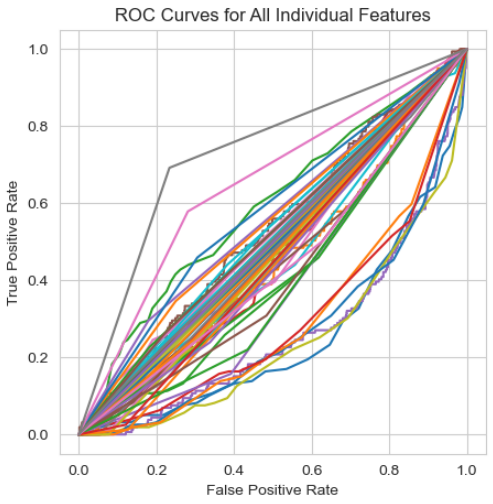
In [15]:
```
# Visualize data using PCA
x_train_scaled = StandardScaler().fit_transform(train_x)
x_pca = PCA(n_components=2).fit_transform(x_train_scaled)
plt.xlabel('First Principle Component')
plt.ylabel('Second Principle Component')
plt.scatter(x=x_pca[train_y == 0][:,0], y=x_pca[train_y == 0][:,1], c='blue', alpha=0.5, label='No')
plt.scatter(x=x_pca[train_y == 1][:,0], y=x_pca[train_y == 1][:,1], c='red', alpha=0.5, label='Yes')
plt.legend(title='Attrition')
plt.title("PCA Projection of Dataset")
# plt.savefig('Graphs/pcaall.png',bbox_inches='tight')
```

Out[15]:
```
Text(0.5, 1.0, 'PCA Projection of Dataset')
```



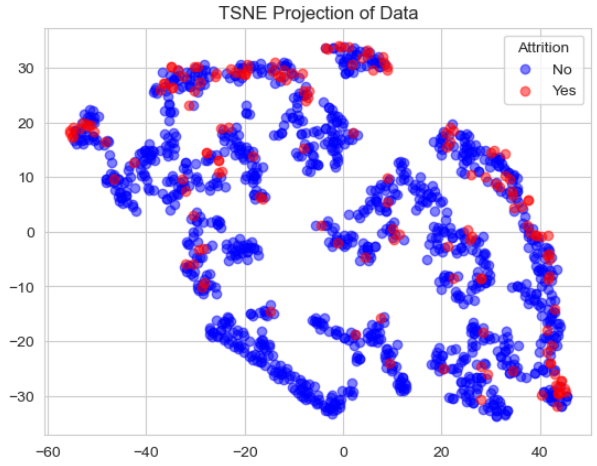
In [16]:
```
# Plot ROC Curves
auc_scores = []
plt.figure(figsize=(5,5))
for i, feature in enumerate(train_x.columns):
    fpr_train, tpr_train, thresholds_train = roc_curve(train_y, train_x[feature])
    auc_scores.append((feature, roc_auc_score(train_y, train_x[feature])))
    plt.plot(fpr_train, tpr_train, label=feature)
plt.title('ROC Curves for All Individual Features')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
# plt.savefig('Graphs/AllROC.png', bbox_inches='tight')
```

Out[16]:
```
Text(0, 0.5, 'True Positive Rate')
```



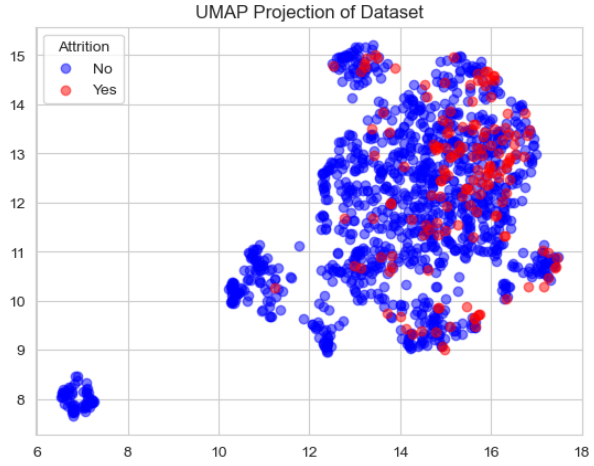
In [17]:
```
# Visualize data using TSNE
x_tsne = TSNE(perplexity=30).fit_transform(train_x)
plt.scatter(x=x_tsne[train_y == 0][:,0], y=x_tsne[train_y == 0][:,1], c='blue', alpha=0.5, label='No')
plt.scatter(x=x_tsne[train_y == 1][:,0], y=x_tsne[train_y == 1][:,1], c='red', alpha=0.5, label='Yes')
plt.legend(title='Attrition')
plt.title("TSNE Projection of Data")
# plt.savefig('Graphs/tsne2dp30.png')
```

Out[17]:
```
Text(0.5, 1.0, 'TSNE Projection of Data')
```



In [18]:
```
# Visualize data using UMAP
x_train_scaled = StandardScaler().fit_transform(train_x)
x_umap = umap.UMAP(n_neighbors=15).fit_transform(x_train_scaled)
plt.scatter(x=x_umap[train_y == 0][:,0], y=x_umap[train_y == 0][:,1], c='blue', alpha=0.5, label='No')
plt.scatter(x=x_umap[train_y == 1][:,0], y=x_umap[train_y == 1][:,1], c='red', alpha=0.5, label='Yes')
plt.legend(title='Attrition')
plt.title("UMAP Projection of Dataset")
# plt.savefig('Graphs/umap2dpdefault.png')
```

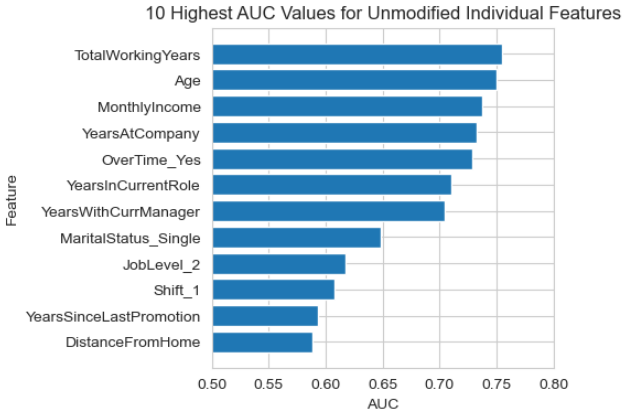
Out[18]:
```
Text(0.5, 1.0, 'UMAP Projection of Dataset')
```



In [19]:
```
# Sort AUC scores, get sorted list of feature names and AUC values
auc_scores.sort(key=lambda x: max(0.5-x[1], x[1]-0.5), reverse=False)
auc_features = [x[0] for x in auc_scores]
auc_f_scores = [0.5 + max(0.5-x[1], x[1]-0.5) for x in auc_scores]
```

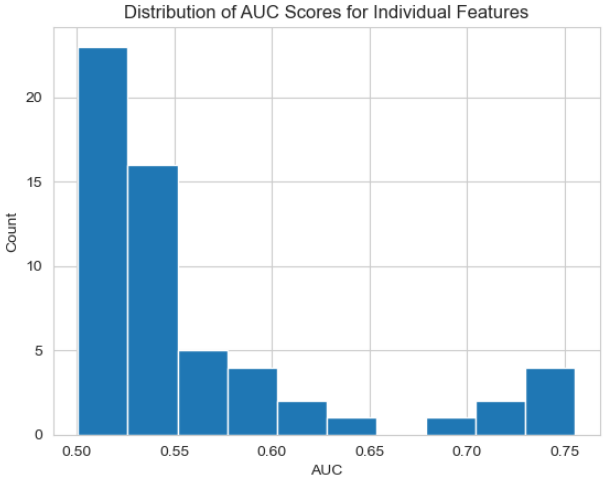
In [20]:
```
# Bar chart of individual features with greatest AUC values
plt.figure(figsize=(4,4))
plt.barh(auc_features[-12:], auc_f_scores[-12:])
plt.xlim(0.5,0.8)
plt.title('10 Highest AUC Values for Unmodified Individual Features')
plt.xlabel('AUC')
plt.ylabel('Feature')
# plt.savefig('Graphs/highauc.png', bbox_inches='tight')
```

Out[20]:
```
Text(0, 0.5, 'Feature')
```



In [21]:
```
# Histogram of AUC values for individual features
plt.hist(auc_f_scores)
plt.title('Distribution of AUC Scores for Individual Features')
plt.xlabel('AUC')
plt.ylabel('Count')
# plt.savefig('Graphs/auchistogram.png', bbox_inches='tight')
```

Out[21]:
```
Text(0, 0.5, 'Count')
```



In [22]:
```
# Create random up-sample training data
xyr = pd.concat([x_train, y_train], axis=1)
random_upsample = resample(xyr.loc[xyr.Attrition_Yes == 1], n_samples=len(xyr.loc[xyr.Attrition_Yes == 0]))
upsampled_df = pd.concat([random_upsample, xyr.loc[xyr.Attrition_Yes == 0]])
xr_train = upsampled_df.drop(columns='Attrition_Yes')
yr_train = upsampled_df.Attrition_Yes
```

In [23]:
```
# Generate PCA on training data
xr_train_scaled = StandardScaler().fit_transform(xr_train)
x_val_scaled = StandardScaler().fit_transform(x_val)

xr_train_pca = PCA(n_components=5).fit_transform(xr_train_scaled)
x_val_pca = PCA(n_components=5).fit_transform(x_val_scaled)
```

In [27]:
```
# Try to fit XGBoost model using PCA components
xgb_r_model = XGBClassifier(max_depth=7, verbosity=0).fit(xr_train_pca, yr_train)
f1_score(xgb_r_model.predict(x_val_pca), y_val)
```

Out[27]:
```
0.2365591397849462
```

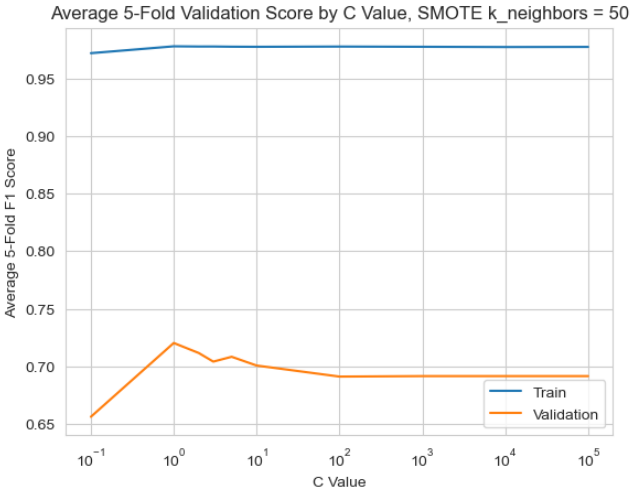
In [40]:
```
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegressionCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
from xgboost import XGBClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from imblearn.over_sampling import SMOTENC
from sklearn.utils import resample
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler
import time
import warnings
warnings.filterwarnings("ignore")
```

In [41]:
```
# Load in data, create train, validation split, get categorical and ambiguous features, scale data, use SMOTENC to upsample data
train = pd.read_csv('train.csv')
drop_features = ['Over18', 'EmployeeCount', 'StandardHours', 'EmployeeID']
good_features = [c for c in train.columns if c not in drop_features]
ambiguous_features = ['Education', 'EnvironmentSatisfaction', 'JobInvolvement', 'JobLevel', 'JobSatisfaction', 'PerformanceRating', 'RelationshipSatisfaction', 'Shift', '

numerical_features = list(set(train[good_features].select_dtypes(include=np.number).columns) - set(ambiguous_features))
categorical_features = [f for f in train.columns if f not in numerical_features and f not in drop_features]
catx_features = [f for f in categorical_features if f != 'Attrition']

train_dummies = pd.get_dummies(train[good_features], columns=categorical_features, drop_first=True)
# amb_train_dummies = pd.get_dummies(train_dummies, columns=ambiguous_features, drop_first=True)

train_y = train_dummies.Attrition_Yes
train_x = train_dummies.loc[:, train_dummies.columns != 'Attrition_Yes']
x_train, x_val, y_train, y_val = train_test_split(train_x, train_y, test_size = 0.2)

catxt_features = [f for f in train_x.columns if '_' in f]
numxt_features = [f for f in train_x if f not in catxt_features]

test_x_original = pd.read_csv('test.csv')
test_features = [c for c in good_features if c != 'Attrition']
test_cat = [c for c in categorical_features if c != 'Attrition']
test_x = pd.get_dummies(test_x_original[test_features], columns=test_cat, drop_first=True)
test_x_scaled = StandardScaler().fit_transform(test_x)

x_cat = [train_x.columns.get_loc(c) for c in train_x.columns if '_' in c]

train_x_scaled = StandardScaler().fit_transform(x_train)
val_scaled = StandardScaler().fit_transform(x_val)

train_xf_scaled = StandardScaler().fit_transform(train_x)

smotef = SMOTENC(categorical_features=x_cat, k_neighbors=7)
xf_sr_train, yf_sr_train = smotef.fit_resample(train_xf_scaled, train_y)
```

In [42]:
```
# Conduct 5-fold cross validation on C parameter, store result
Cs = [0.1, 1, 2, 3, 5, 10, 100, 1000, 10000, 100000]
kfold = KFold(n_splits=5)
k_train_scores = []
k_val_scores = []
for c in Cs:
    train_scores = []
    val_scores = []
    for tr_idx, val_idx in kfold.split(train_x):
        xt_scaled= StandardScaler().fit_transform(train_x.iloc[tr_idx])
        xv_scaled = StandardScaler().fit_transform(train_x.iloc[val_idx])
        xt_up_scaled, yt_up = SMOTENC(categorical_features=x_cat, k_neighbors=50).fit_resample(xt_scaled, train_y.iloc[tr_idx])
        log_reg = LogisticRegression(C=c, class_weight={0:0.5, 1:0.5}).fit(xt_up_scaled, yt_up)
        # print(f1_score(log_reg.predict(xv_scaled), train_y[val_idx]))
        train_scores.append(f1_score(log_reg.predict(xt_up_scaled), yt_up))
        val_scores.append(f1_score(log_reg.predict(xv_scaled), train_y.iloc[val_idx]))
    k_train_scores.append(np.mean(train_scores))
    k_val_scores.append(np.mean(val_scores))
```

In [43]:
```
# Plot cross validation train and validation f1 scores across tested C values
plt.plot(Cs,k_train_scores, label='Train')
plt.plot(Cs,k_val_scores, label='Validation')
plt.legend()
plt.xscale('log')
plt.xlabel('C Value')
plt.ylabel('Average 5-Fold F1 Score')
plt.title('Average 5-Fold Validation Score by C Value, SMOTE k_neighbors = 50')
# plt.savefig('Graphs/LogRegCVk50.png', bbox_inches='tight')
```

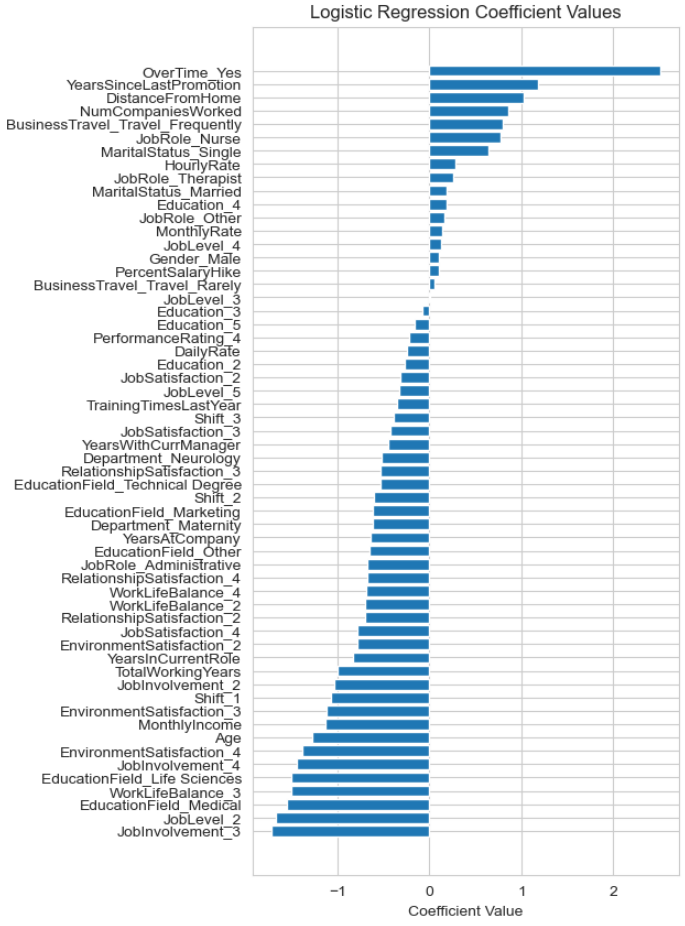
Out[43]:
```
Text(0.5, 1.0, 'Average 5-Fold Validation Score by C Value, SMOTE k_neighbors = 50')
```



In [44]:
```
# Make bar chart of coefficient values
xt_scaled= StandardScaler().fit_transform(train_x)
xt_up_scaled, yt_up = SMOTENC(categorical_features=x_cat, k_neighbors=15).fit_resample(xt_scaled, train_y)
cur = time.time()
lg_reg = LogisticRegression(C=3).fit(xt_up_scaled, yt_up)
print(time.time() - cur)
plt.figure(figsize=(5,10))
f_i = list(zip(train_x.columns, lg_reg.coef_[0].tolist()))
f_i.sort(key=lambda x: x[1], reverse=False)
s_c = [x[0] for x in f_i]
s_f = [x[1] for x in f_i]
plt.barh(s_c, s_f)
plt.title('Logistic Regression Coefficient Values')
plt.xlabel('Coefficient Value')
# plt.savefig('Graphs/LGCoef.png', bbox_inches='tight')
```

```
0.03200888633728027
```

Out[44]:
```
Text(0.5, 0, 'Coefficient Value')
```

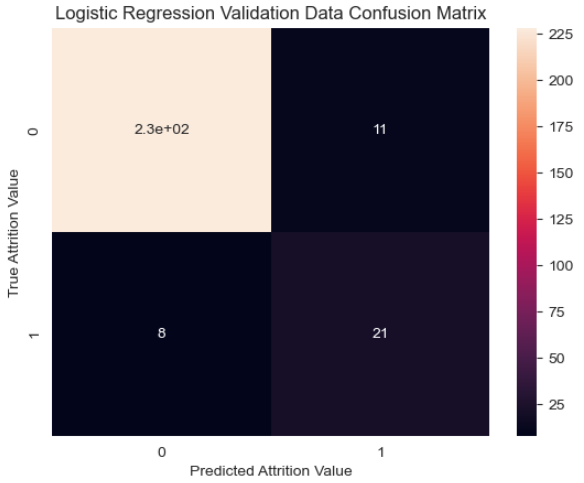


In [45]:
```
# Generate confusion matrix on validation data
xt_scaled = StandardScaler().fit_transform(x_train)
xv_scaled = StandardScaler().fit_transform(x_val)
xt_up_scaled, yt_up = SMOTENC(categorical_features=x_cat, k_neighbors=15).fit_resample(xt_scaled, y_train)
lr_cv = LogisticRegression(C=10, class_weight={0:0.5, 1:0.5}).fit(xt_up_scaled, yt_up)
print('Train f1: ', f1_score(lr_cv.predict(xt_up_scaled), yt_up))
print('Val f1: ', f1_score(lr_cv.predict(xv_scaled), y_val))
sns.heatmap(confusion_matrix(y_pred=lr_cv.predict(xv_scaled), y_true=y_val), annot=True)
```

```
plt.title('Logistic Regression Validation Data Confusion Matrix')
plt.xlabel('Predicted Attrition Value')
plt.ylabel('True Attrition Value')
# plt.savefig('Graphs/LRCM.png', bbox_inches='tight')
```

```
Train f1:  0.9786552828175026
Val f1:  0.6885245901639345
```

Out[45]:
```
Text(52.722222222222214, 0.5, 'True Attrition Value')
```



In [46]:
```
# Generate predictions from Logistic regression model
lr = LogisticRegression(C=10000, class_weight={0:0.5, 1:0.5}).fit(xf_sr_train, yf_sr_train)
submission14 = pd.DataFrame(lr.predict(test_x_scaled)).rename(columns={0: 'Predicted'})
# submission14.index.rename('Id', inplace=True)
# submission14.to_csv('submission15.csv')
```

In [32]:
```
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
from xgboost import XGBClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from imblearn.over_sampling import SMOTENC
from sklearn.utils import resample
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import AdaBoostClassifier
import time
import warnings
warnings.filterwarnings("ignore")
```

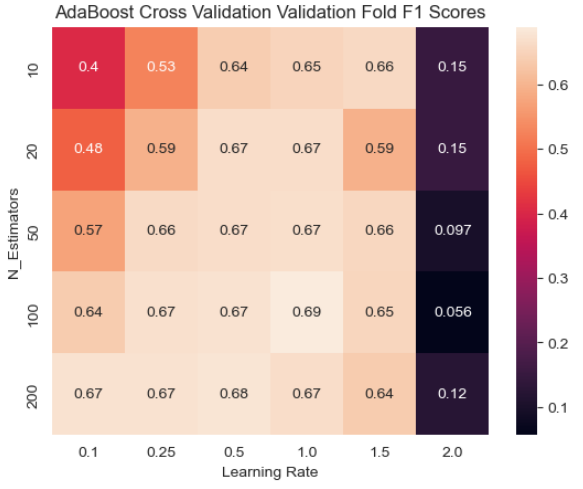
In [33]:
```
# Load in data, create train, validation split, get categorical variables when using SMOTENC
train = pd.read_csv('train_clean.csv', index_col=[0])
test = pd.read_csv('test_clean.csv', index_col=[0])
xf_train = train.loc[:,train.columns != 'Attrition_Yes']
yf_train = train.Attrition_Yes
x_train, x_val, y_train, y_val = train_test_split(train.loc[:,train.columns != 'Attrition_Yes'], train.Attrition_Yes, test_size = 0.2)

x_cat = [x_train.columns.get_loc(c) for c in x_train.columns if '_' in c]
```

In [34]:
```
smote = SMOTENC(categorical_features=x_cat)
xf_sr_train, yf_sr_train = smote.fit_resample(x_train, y_train)
```

In [35]:
```
# Conduct 5-fold cross validation on n_estimators, learning_rate parameters, store result
n_estimators = [10, 20, 50, 100, 200]
learning_rates = [0.1, 0.25, 0.5, 1, 1.5, 2]

kfold = KFold(n_splits=5)
n_est_train_scores = []
n_est_val_scores = []
for n in n_estimators:
    lr_train_scores = []
    lr_val_scores = []
    for lr in learning_rates:
        train_scores = []
        val_scores = []
        for tr_idx, val_idx in kfold.split(xf_train):
            xt_up, yt_up = SMOTENC(categorical_features=x_cat, k_neighbors=15).fit_resample(xf_train.iloc[tr_idx], yf_train.iloc[tr_idx])
            ab_model = AdaBoostClassifier(n_estimators=n, learning_rate=lr).fit(xt_up, yt_up)
            # print(f1_score(ab_model.predict(xf_train.iloc[val_idx]), yf_train.iloc[val_idx]))
            train_scores.append(f1_score(ab_model.predict(xf_train.iloc[tr_idx]), yf_train.iloc[tr_idx]))
            val_scores.append(f1_score(ab_model.predict(xf_train.iloc[val_idx]), yf_train.iloc[val_idx]))
        lr_train_scores.append(np.mean(train_scores))
        lr_val_scores.append(np.mean(val_scores))
    n_est_train_scores.append(lr_train_scores)
    n_est_val_scores.append(lr_val_scores)
```

In [36]:
```
# Visualize validation f1 scores from cross validation
ab_train_scores = pd.DataFrame(n_est_train_scores)
ab_val_scores = pd.DataFrame(n_est_val_scores)

ab_val_scores.columns = learning_rates
ab_val_scores.index = n_estimators

ab_train_scores.columns = learning_rates
ab_train_scores.index = n_estimators

sns.heatmap(ab_val_scores, annot=True)
plt.ylabel('N_Estimators')
plt.xlabel('Learning Rate')
plt.title('AdaBoost Cross Validation Validation Fold F1 Scores')
# plt.savefig('Graphs/ABCVVal.png', bbox_inches='tight')
```

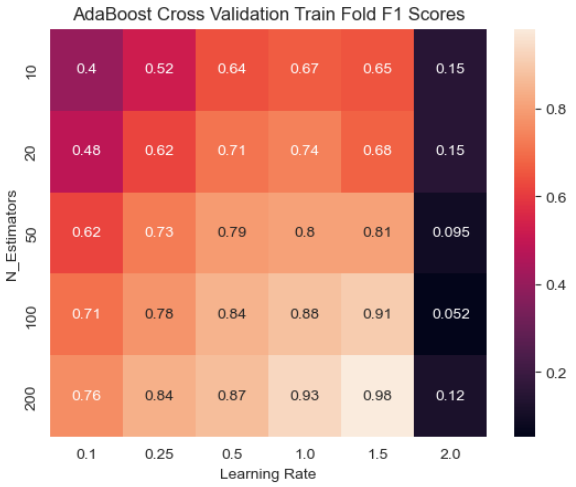
Out[36]:
```
Text(0.5, 1.0, 'AdaBoost Cross Validation Validation Fold F1 Scores')
```



In [37]:
```
# Visualize train f1 scores from cross validation
sns.heatmap(ab_train_scores,annot=True)
```

```
plt.ylabel('N_Estimators')
plt.xlabel('Learning Rate')
plt.title('AdaBoost Cross Validation Train Fold F1 Scores')
# plt.savefig('Graphs/ABCVTrain.png', bbox_inches='tight')
```

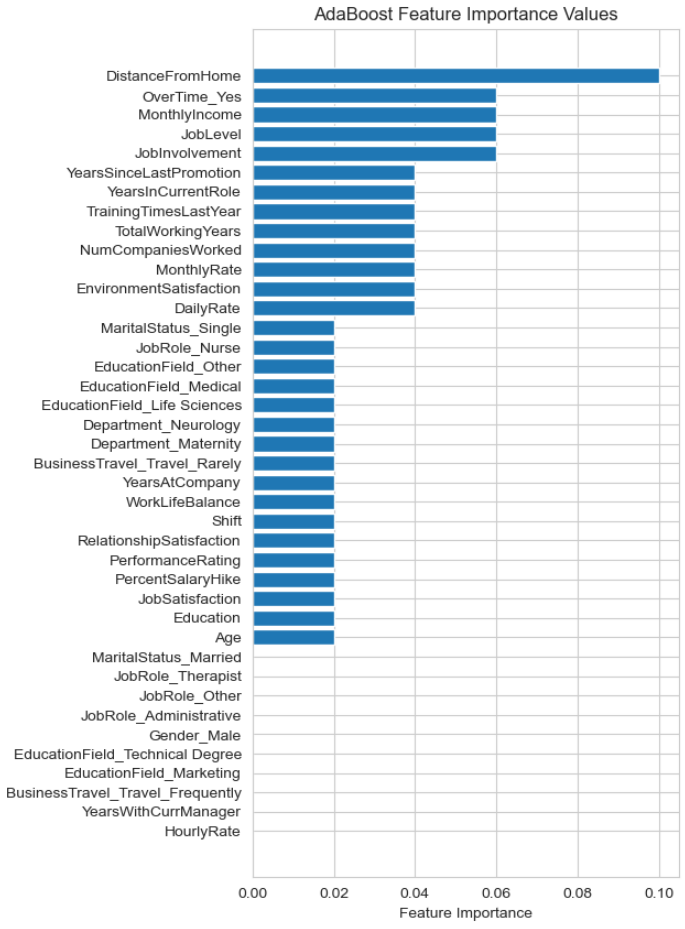
Out[37]:
```
Text(0.5, 1.0, 'AdaBoost Cross Validation Train Fold F1 Scores')
```



In [38]:
```
# Make bar chart of feature importance values
smote = SMOTENC(categorical_features=x_cat, k_neighbors=8)
x_sr_train, y_sr_train = smote.fit_resample(xf_train, yf_train)
cur = time.time()
cv_ab = AdaBoostClassifier(n_estimators=50, learning_rate=1).fit(x_sr_train, y_sr_train)
print(time.time() - cur)
plt.figure(figsize=(5,10))
f_i = list(zip(x_sr_train.columns, cv_ab.feature_importances_))
f_i.sort(key=lambda x: x[1], reverse=False)
s_c = [x[0] for x in f_i]
s_f = [x[1] for x in f_i]
plt.barh(s_c, s_f)
plt.title('AdaBoost Feature Importance Values')
plt.xlabel('Feature Importance')
# plt.savefig('Graphs/ABFI.png', bbox_inches='tight')
```

```
0.16730332374572754
```

Out[38]:
```
Text(0.5, 0, 'Feature Importance')
```

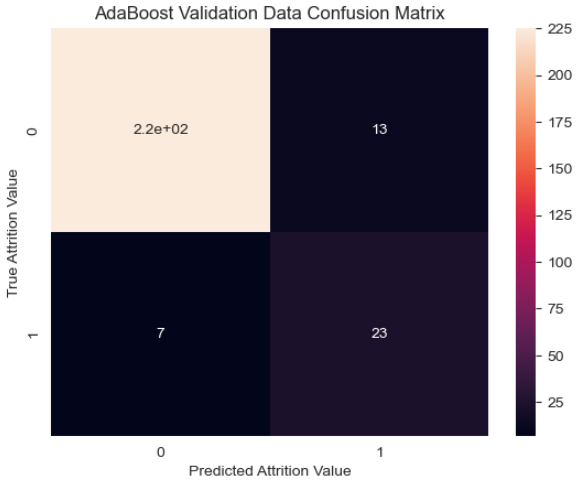


In [39]:
```
# Generate confusion matrix on validation data
smote = SMOTENC(categorical_features=x_cat, k_neighbors=5)
```

```
xsr_train, ysr_train = smote.fit_resample(x_train, y_train)
ab_cv_model = AdaBoostClassifier(n_estimators=50, learning_rate=1).fit(xsr_train, ysr_train)
print('Train f1: ', f1_score(ab_cv_model.predict(xsr_train), ysr_train))
print('Val f1: ', f1_score(ab_cv_model.predict(x_val), y_val))
sns.heatmap(confusion_matrix(y_pred=ab_cv_model.predict(x_val), y_true=y_val), annot=True)
plt.title('AdaBoost Validation Data Confusion Matrix')
plt.xlabel('Predicted Attrition Value')
plt.ylabel('True Attrition Value')
# plt.savefig('Graphs/ABCM.png', bbox_inches='tight')
```

```
Train f1:  0.9687665431445209
Val f1:  0.696969696969697
```

Out[39]:
```
Text(52.722222222222214, 0.5, 'True Attrition Value')
```



In [40]:
```
# Generate submissions using best model as determined through cross validation
smote = SMOTENC(categorical_features=x_cat, k_neighbors=10)
xf_sr_train, yf_sr_train = smote.fit_resample(xf_train, yf_train)
ab_predictions_model = AdaBoostClassifier(n_estimators=50, learning_rate=1).fit(xf_sr_train, yf_sr_train)
submission18 = pd.DataFrame(ab_predictions_model.predict(test)).rename(columns={0: 'Predicted'})
submission18.index.rename('Id', inplace=True)
submission18.to_csv('submission18.csv')
```

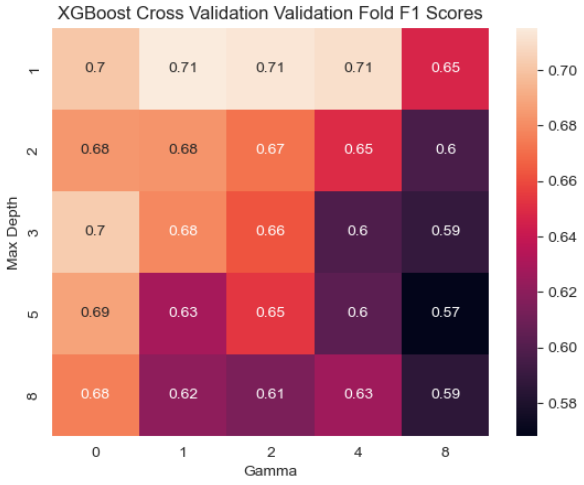
In [28]:
```
import pandas as pd
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
from xgboost import XGBClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from imblearn.over_sampling import SMOTENC
from sklearn.utils import resample
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
import time
import warnings
warnings.filterwarnings("ignore")
```

In [29]:
```
# Load in data, create train, validation split, get categorical variables when suing SMOTENC
train = pd.read_csv('train_clean.csv', index_col=[0])
test = pd.read_csv('test_clean.csv', index_col=[0])
xf_train = train.loc[:,train.columns != 'Attrition_Yes']
yf_train = train.Attrition_Yes
x_train, x_val, y_train, y_val = train_test_split(train.loc[:,train.columns != 'Attrition_Yes'], train.Attrition_Yes, test_size = 0.2)

x_cat = [x_train.columns.get_loc(c) for c in x_train.columns if '_' in c]
```

In [30]:
```
# Conduct 5-fold cross validation on max_depth, gamma parameters, store result
max_depths = [1, 2, 3, 5, 8]
gammas = [0, 1, 2, 4, 8]

kfold = KFold(n_splits=5)
k_by_md_train_scores = []
k_by_gamma_val_scores = []
for md in max_depths:
    by_g_train_scores = []
    by_g_val_scores = []
    for g in gammas:
        train_scores = []
        val_scores = []
        for tr_idx, val_idx in kfold.split(xf_train):
            xt_up, yt_up = SMOTENC(categorical_features=x_cat, k_neighbors=15).fit_resample(xf_train.iloc[tr_idx], yf_train.iloc[tr_idx])
            xgb_clf = XGBClassifier(gamma=g, max_depth=md, verbosity=0).fit(xt_up.iloc[tr_idx], yt_up.iloc[tr_idx])
            # print(f1_score(xgb_clf.predict(xf_train.iloc[val_idx]), yf_train.iloc[val_idx]))
            train_scores.append(f1_score(xgb_clf.predict(xf_train.iloc[tr_idx]), yf_train.iloc[tr_idx]))
            val_scores.append(f1_score(xgb_clf.predict(xf_train.iloc[val_idx]), yf_train.iloc[val_idx]))
        by_g_train_scores.append(np.mean(train_scores))
        by_g_val_scores.append(np.mean(val_scores))
    k_by_md_train_scores.append(by_g_train_scores)
    k_by_gamma_val_scores.append(by_g_val_scores)
```

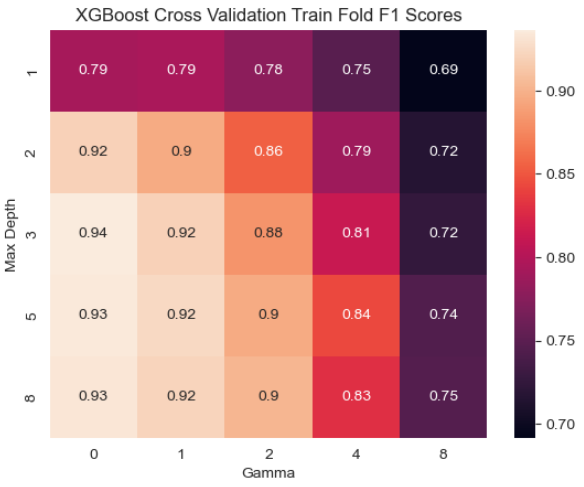
In [31]:
```
# Visualize validation f1 scores from cross validation
xgb_train_scores = pd.DataFrame(k_by_md_train_scores)
xgb_val_scores = pd.DataFrame(k_by_gamma_val_scores)

xgb_val_scores.columns = gammas
xgb_val_scores.index = max_depths

xgb_train_scores.columns = gammas
xgb_train_scores.index = max_depths

sns.heatmap(xgb_val_scores, annot=True)
plt.ylabel('Max Depth')
plt.xlabel('Gamma')
plt.title('XGBoost Cross Validation Validation Fold F1 Scores')
# plt.savefig('Graphs/XGBCVVal.png', bbox_inches='tight')
```

Out[31]:
```
Text(0.5, 1.0, 'XGBoost Cross Validation Validation Fold F1 Scores')
```



In [32]:
```
# Visualize train f1 scores from cross validation
sns.heatmap(xgb_train_scores, annot=True)
plt.ylabel('Max Depth')
plt.xlabel('Gamma')
plt.title('XGBoost Cross Validation Train Fold F1 Scores')
# plt.savefig('Graphs/XGBCVTrain.png', bbox_inches='tight')
```

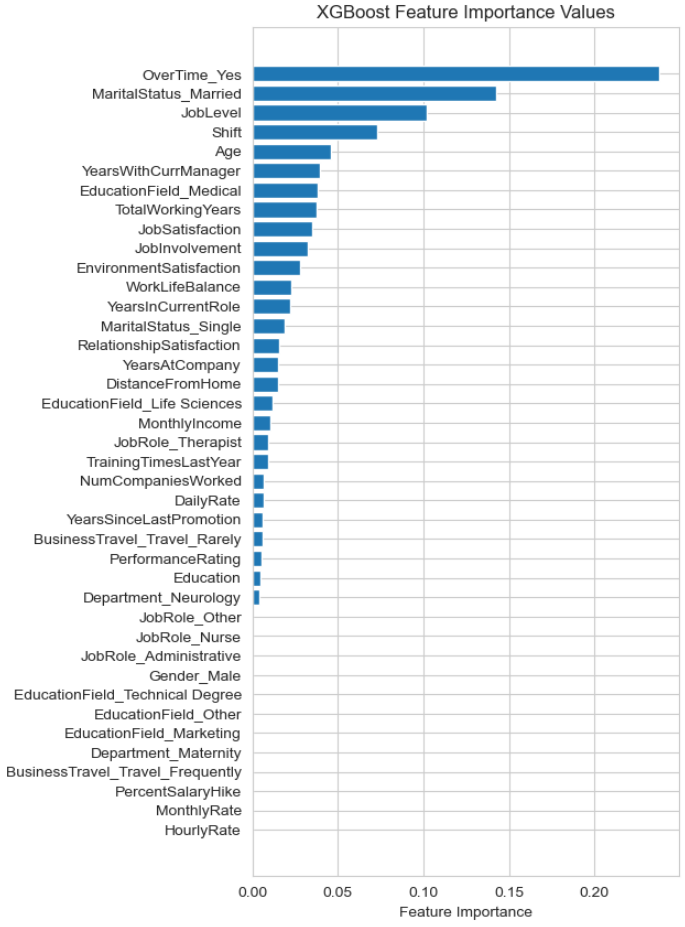
Out[32]: Text(0.5, 1.0, 'XGBoost Cross Validation Train Fold F1 Scores')



In [33]:
```
# Make bar chart of feature importance values
xt_up, yt_up = SMOTENC(categorical_features=x_cat, k_neighbors=15).fit_resample(xf_train, yf_train)
cur = time.time()
xgb_clf = XGBClassifier(gamma=1, max_depth=1).fit(xt_up, yt_up)
print(time.time() - cur)
plt.figure(figsize=(5,10))
f_i = list(zip(xt_up.columns, xgb_clf.feature_importances_))
f_i.sort(key=lambda x: x[1], reverse=False)
s_c = [x[0] for x in f_i]
s_f = [x[1] for x in f_i]
plt.barh(s_c, s_f)
plt.title('XGBoost Feature Importance Values')
plt.xlabel('Feature Importance')
# plt.savefig('Graphs/XGBoostFI.png', bbox_inches='tight')
```

```
0.07808208465576172
```

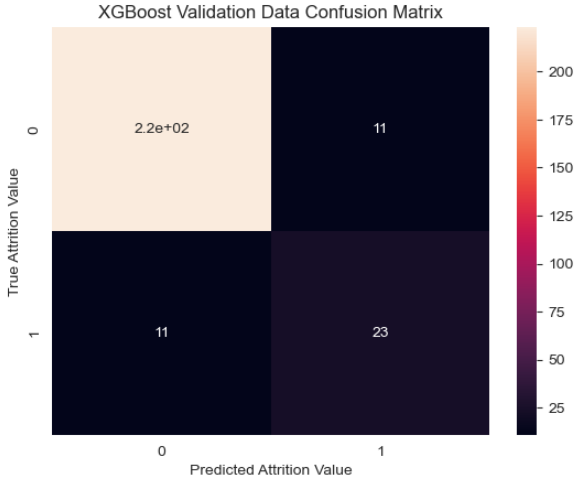
Out[33]: Text(0.5, 0, 'Feature Importance')



In [34]:
```
# Generate confusion matrix on validation data
smote = SMOTENC(categorical_features=x_cat, k_neighbors=5)
xsr_train, ysr_train = smote.fit_resample(x_train, y_train)
xgb_sr_model = XGBClassifier(n_estimators= 200, gamma=4, max_depth=1, min_child_weight=20, reg_lambda = 7, subsample=0.3, eta=0.2).fit(xsr_train, ysr_train)
print('Train f1: ', f1_score(xgb_sr_model.predict(xsr_train), ysr_train))
print('Val f1: ', f1_score(xgb_sr_model.predict(x_val), y_val))
sns.heatmap(confusion_matrix(y_pred=xgb_sr_model.predict(x_val), y_true=y_val), annot=True)
```

```
plt.title('XGBoost Validation Data Confusion Matrix')
plt.xlabel('Predicted Attrition Value')
plt.ylabel('True Attrition Value')
plt.savefig('Graphs/XGBCM.png', bbox_inches='tight')
```

```
Train f1:  0.9522309711286089
Val f1:  0.6764705882352942
```



In [35]:
```
# Generate submissions using best model as determined through cross validation
smote = SMOTENC(categorical_features=x_cat, k_neighbors=10)
xf_sr_train, yf_sr_train = smote.fit_resample(xf_train, yf_train)
xgb_predictions_model = XGBClassifier(n_estimators= 200, gamma=4, max_depth=1, min_child_weight=20, reg_lambda = 7, subsample=0.3, eta=0.2).fit(xf_sr_train, yf_sr_train)
submission18 = pd.DataFrame(xgb_predictions_model.predict(test)).rename(columns={0: 'Predicted'})
submission18.index.rename('Id', inplace=True)
submission18.to_csv('submission18.csv')
```