Introduction

Employee attrition costs employers by reducing productivity and efficiency while training new employees. Employees on the other hand face stagnation both financially or professionally if keeping the status quo. Understanding and balancing the interests of both parties are paramount to have a better functioning society with improved quality of life. As the relationship can be mutually beneficial, researchers have been looking into such data, typically gathered from human resources information systems (HRIS). Unfortunately, such systems are typically underfunded due to the priority of business first and foremost[2]. The purpose of this project is to perform some exploratory analysis of the IBM HR Analytics Employee Attrition and Performance dataset, looking for any potential insights or discoveries regarding probable causes for employee attrition. Ideally, an accurate model that is capable of predicting "at-risk-of-attrition" employees is possible, pointing out areas that the company can improve upon while maintaining a good relationship with the employees. Python3 will be the language of choice for this project, as there are various APIs that are well suited for analytics as well as data visualization to make use of.

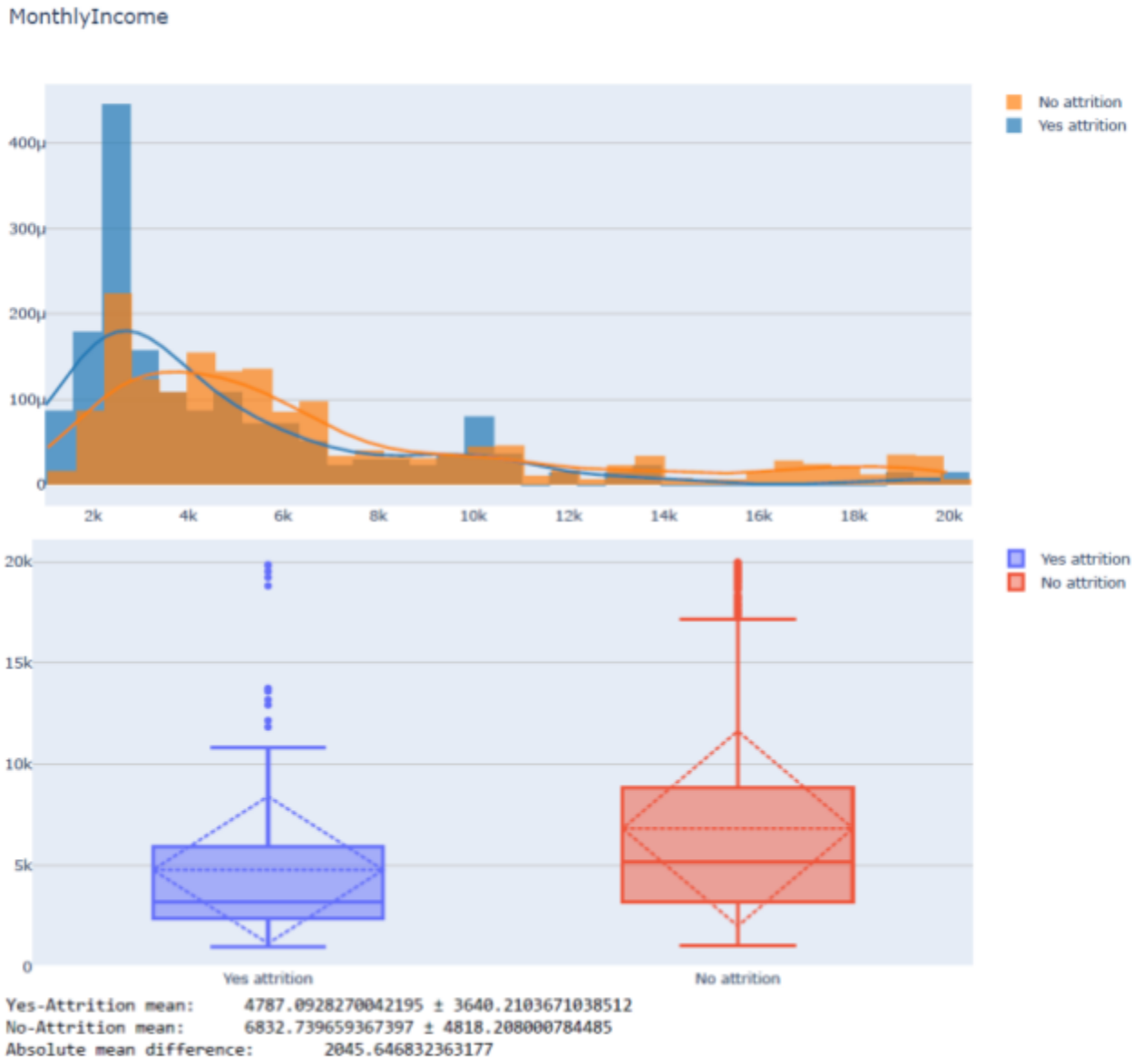
Implementation

I.First Iteration

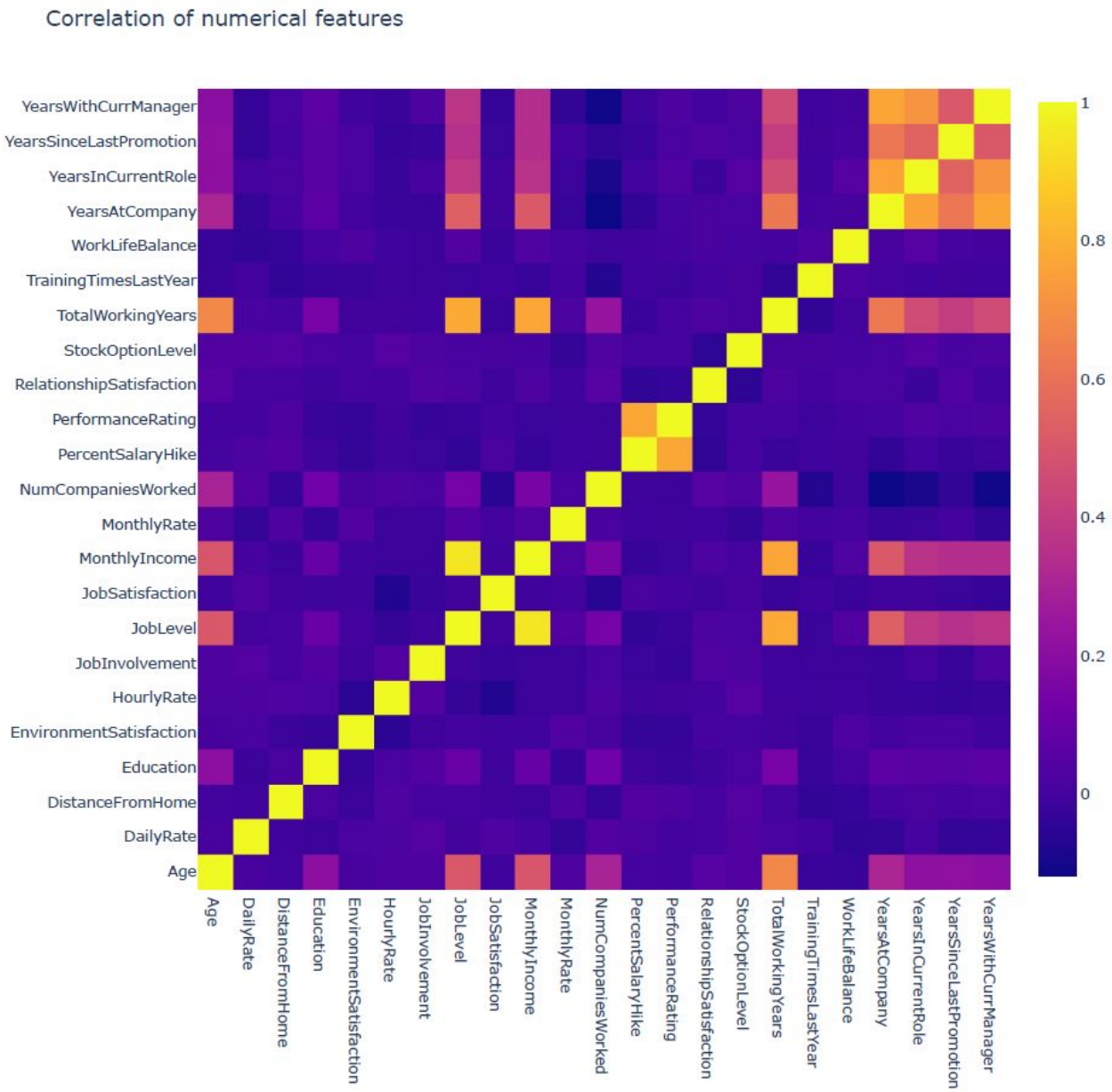
The first step of action as with any machine learning applications is to understand the dataset at hand before jumping into deriving information from it. The dataset may contain a multitude of problems ranging from missing fields, corrupted data, or inconsistent data to name a few. As this dataset does not have any null values, looking at the data types contained shows numerical and categorical data types to work with. The label we're looking for is the "Attrition" column, leaving us with 34 features to learn from to potentially predict attrition. Looking at the class distribution, this dataset has an imbalanced class - high chance of causing an inaccurate model, with only 16.1% in the Yes-attrition class, and the remaining 83.9% in the No-attrition class.

With that in mind, we'll be looking at how each class differs against some of its features to point out potential areas that may indicate attrition better than others. Some notable

features included the age, monthly income, and years worked at the company. Other features such as distance from home, job satisfaction, or environment satisfaction surprisingly did not have much differences between the Yes-attrition and no-attrition classes.



Before going ahead and training a model, we can also check the features for possible redundancy through a correlation matrix. If there are any features that are highly correlated with another, it is a candidate for elimination to improve computational efficiency.



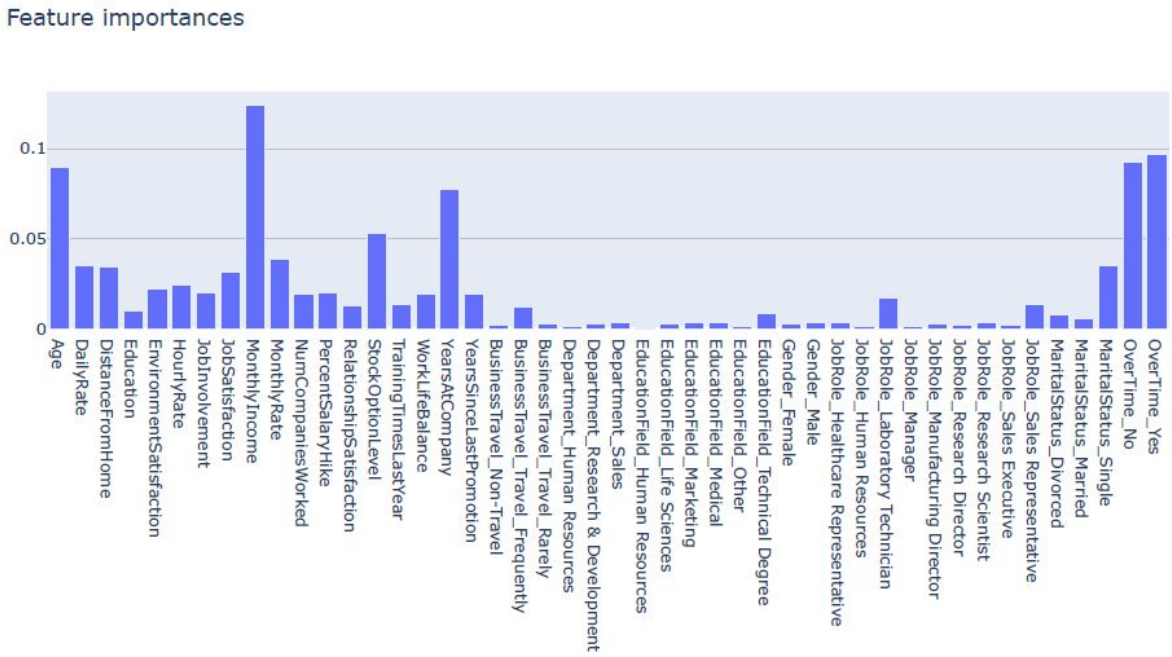
Some of the higher correlated features included years with the current manager, years since last promotion, years in the current role, job level. Intuitively, they do more or less refer to the same type of data. With an initial look from the data, we can now train a model, in which the RandomForest is chosen. It's an ensemble technique that generally does well in accurately classifying even the general cases.

We can evaluate the performance of the model by using a confusion matrix, and the results from the first iteration of the random forest shows the effects of the imbalanced dataset.

```
[[369   0]
 [ 70   2]]
              precision    recall  f1-score   support

          No       0.84      1.00      0.91       369
         Yes       1.00      0.03      0.05        72

   micro avg       0.84      0.84      0.84       441
   macro avg       0.92      0.51      0.48       441
weighted avg       0.87      0.84      0.77       441
```



The first iteration's model places the most important feature as MonthlyIncome, with other factors such as age, years worked at the company, and overtime also contributing to attrition. The confusion matrix shows the model did exceptionally well for the No-attrition case, but horrible in the Yes-attrition case, an artifact from the imbalanced training set.

II. Second Iteration

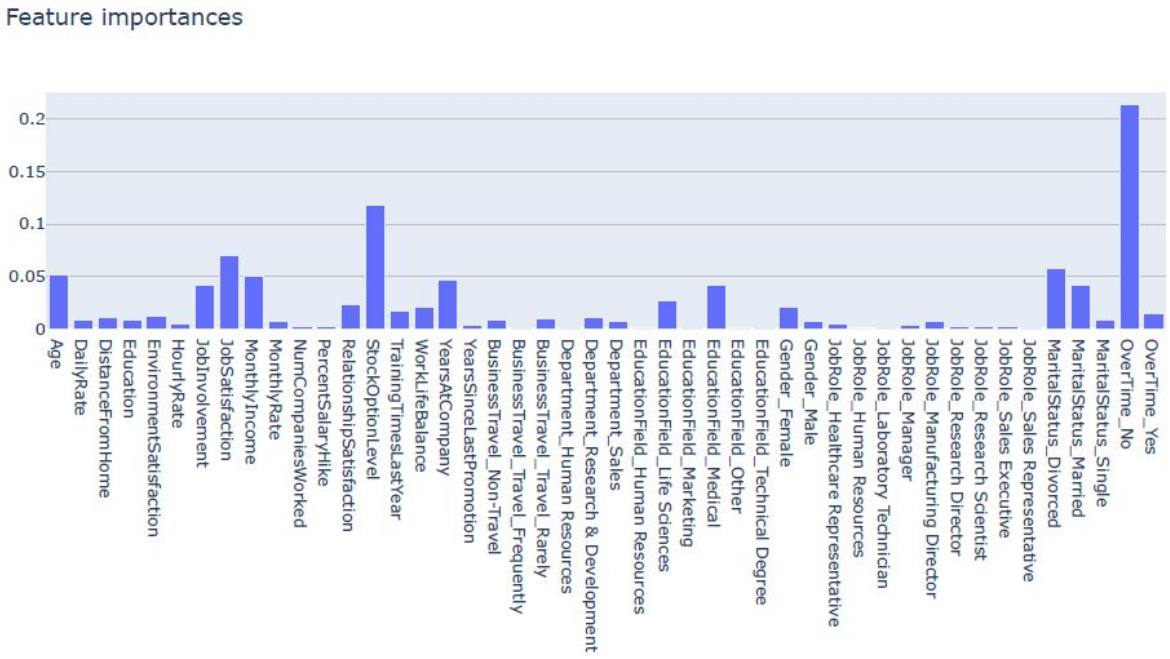
A method of dealing with imbalanced class is to oversample the minority class to match that the majority class. A straightforward approach would be taking samples from the minority class with replacement, essentially filling up duplicates to even out the class balance. The better approach is Synthetic Minority Over-sampling Technique (SMOTE), in which synthetic samples of the minority class is created instead of duplicating samples. For

this iteration, we will use SMOTE and train the model again and evaluate to see how this affects its performance.

```
[[338  31]
 [ 39  33]]
              precision    recall  f1-score   support

          No       0.90      0.92      0.91       369
         Yes       0.52      0.46      0.49        72

   micro avg       0.84      0.84      0.84       441
   macro avg       0.71      0.69      0.70       441
weighted avg       0.83      0.84      0.84       441
```



The second model's performance has drastically improved its Yes-attrition classification, while suffering a loss of accuracy for its No-attrition. However, the model is overall more accurate than the previous iteration, while the feature importance has also drastically changed, with the most "OverTime" becoming the most important feature contributing to attrition classification.

III. Third Iteration

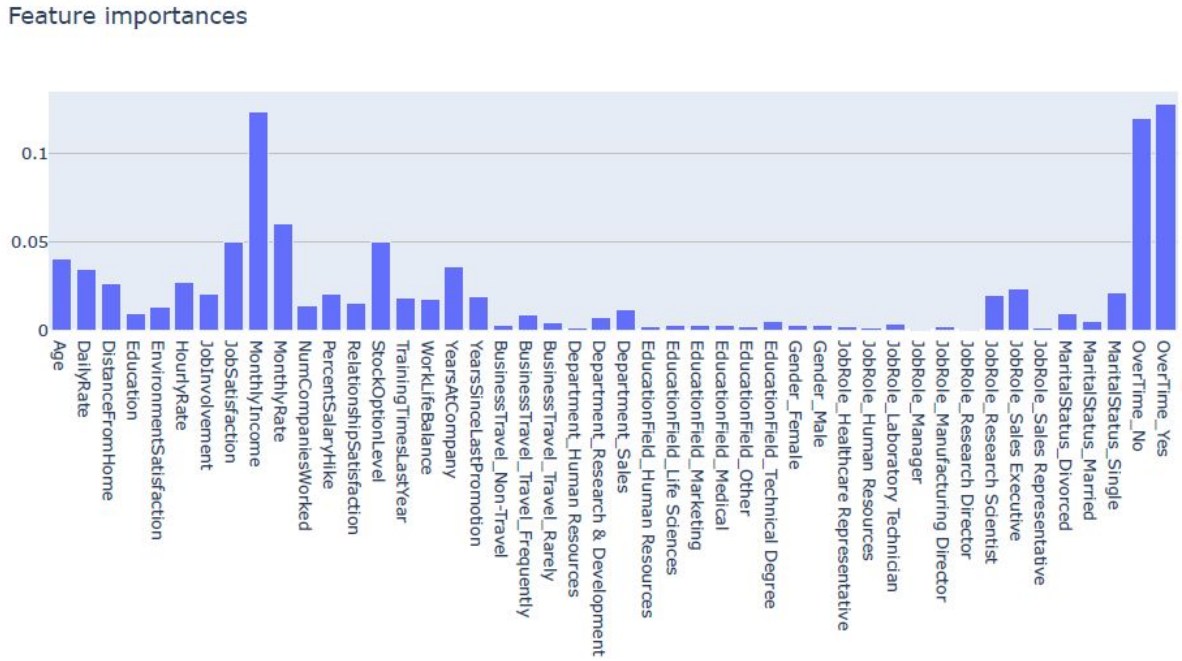
Looking to further improve the accuracy of the model, we will perform undersampling of the majority class instead, another method of dealing with imbalanced class. Essentially, we are dropping samples from the majority class to match the minority class. This will drop

the number of training samples to work with, but does preserve the training data being authentic samples of data.

```
[[162 207]
 [ 18  54]]
              precision    recall  f1-score   support

          No       0.90      0.44      0.59       369
         Yes       0.21      0.75      0.32        72

   micro avg       0.49      0.49      0.49       441
   macro avg       0.55      0.59      0.46       441
weighted avg       0.79      0.49      0.55       441
```



This iteration's most important feature is still "overtime" and monthly income following closely. While the model performs decently well for Yes-attrition class, it suffers from poor accuracy for the No-attrition class.

Conclusion

From basic exploratory data analysis, we can see that there are features such as monthly income, that do produce noticeable difference between the Yes-attrition and no-attrition classes. Keeping in mind the imbalanced classes though, the model shows different kinds of features of importance, with overtime consistently being moderately to the most important feature that contributes to attrition.

While this project did manage to build a model that is 84% accurate from the second iteration, there is plenty of room for improvement. Future work for improvement includes improved feature engineering for model training - providing more relevant features to focus on, and also tweaking the technique used to train the model - such as limiting the maximum depth of the trees in an ensemble as a form of gradient boosting.

References

[1] "IBM HR Analytics Employee Attrition & Performance." Kaggle.Com, 2017, www.kaggle.com/pavansubhasht/ibm-hr-analytics-attrition-dataset.

[2] Rohit, Punnoose & Ajit, Pankaj. (2016). Prediction of Employee Turnover in Organizations using Machine Learning Algorithms. International Journal of Advanced Research in Artificial Intelligence. 5. 10.14569/IJARAI.2016.050904.