Employee Attrition Rate

Prepared for: All

Prepared by: Jason McCartney, Data Scientist

March 26, 2018

Proposal number: 123-4567

SUMMARY

Objective

Employee attrition is a well known problem for most companies these days. This occurrence creates a multitude of problems. Such problems may include disruption in the company due to an unforeseen departure, wasted resources used to train individuals, and a large expenditure in recruitment. The objective of this research experiment is to identify the contributions of the most significant factors pertaining to the rate of employee attrition. Once these factors are identified, multiple models will be tested on the dataset to give the highest accuracy rate of prediction.

Goals

The goal in this experiment is to uncover research that gives employers insight into their hiring processes. With this insight, the employee attrition rate in their respective companies may be decreased due to selective hiring.

Solution

Using the Random Forest Classifier, a 3.8% accuracy rate increase was obtained in the prediction of attrition rate.

Project Outline

The experiment was performed in 3 separate steps:

- Dataset was explored and analyzed
- Dataset was modified and prepared
- Machine learning learning models were used on the dataset

IMPORTING OF PACKAGES

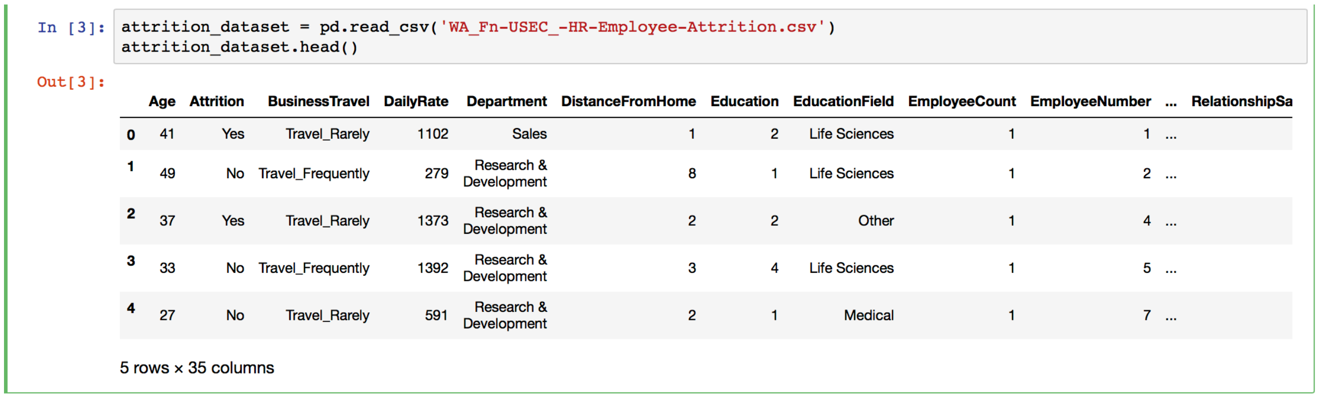
The specific python 3 environment used was created from importing the following:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# required for Plotly
import plotly.offline as py
py.init_notebook_mode(connected = True)
import plotly.graph_objs as go
import plotly.tools as tls
%matplotlib inline

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, log_loss
from imblearn.over_sampling import SMOTE
import xgboost

import warnings
warnings.filterwarnings(‘ignore')
```

EXPLORATION OF THE DATASET

The first step of the experiment entails the simple exploration of the data:

First the dataset was imported and the first 5 entries of each column were observed.



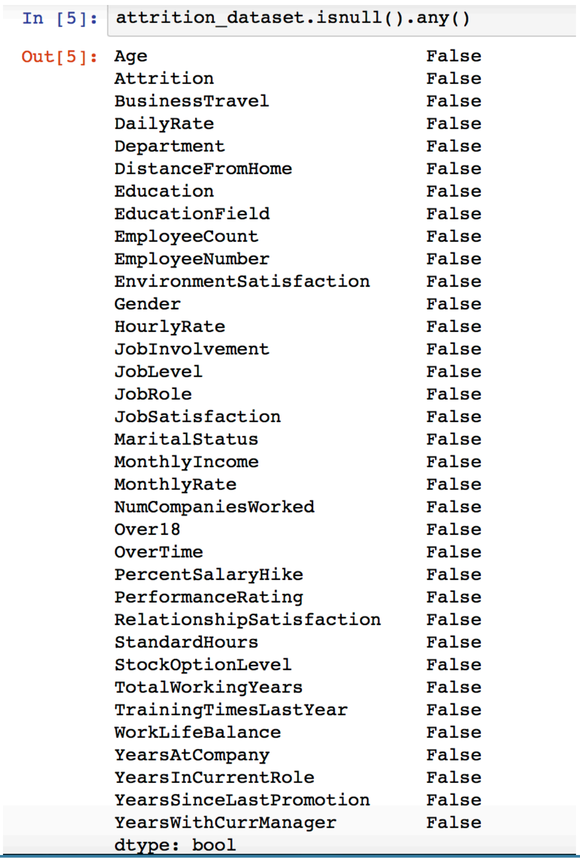
```
In [3]: attrition_dataset = pd.read_csv('WA_Fn-USEC_-HR-Employee-Attrition.csv')
        attrition_dataset.head()
```

Out[3]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumber	...	RelationshipSa
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences	1	1	...	
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences	1	2	...	
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other	1	4	...	
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sciences	1	5	...	
4	27	No	Travel_Rarely	591	Research & Development	2	1	Medical	1	7	...	

5 rows × 35 columns

Next the dataset was checked for its continuity by testing for any null values. No null values were found.

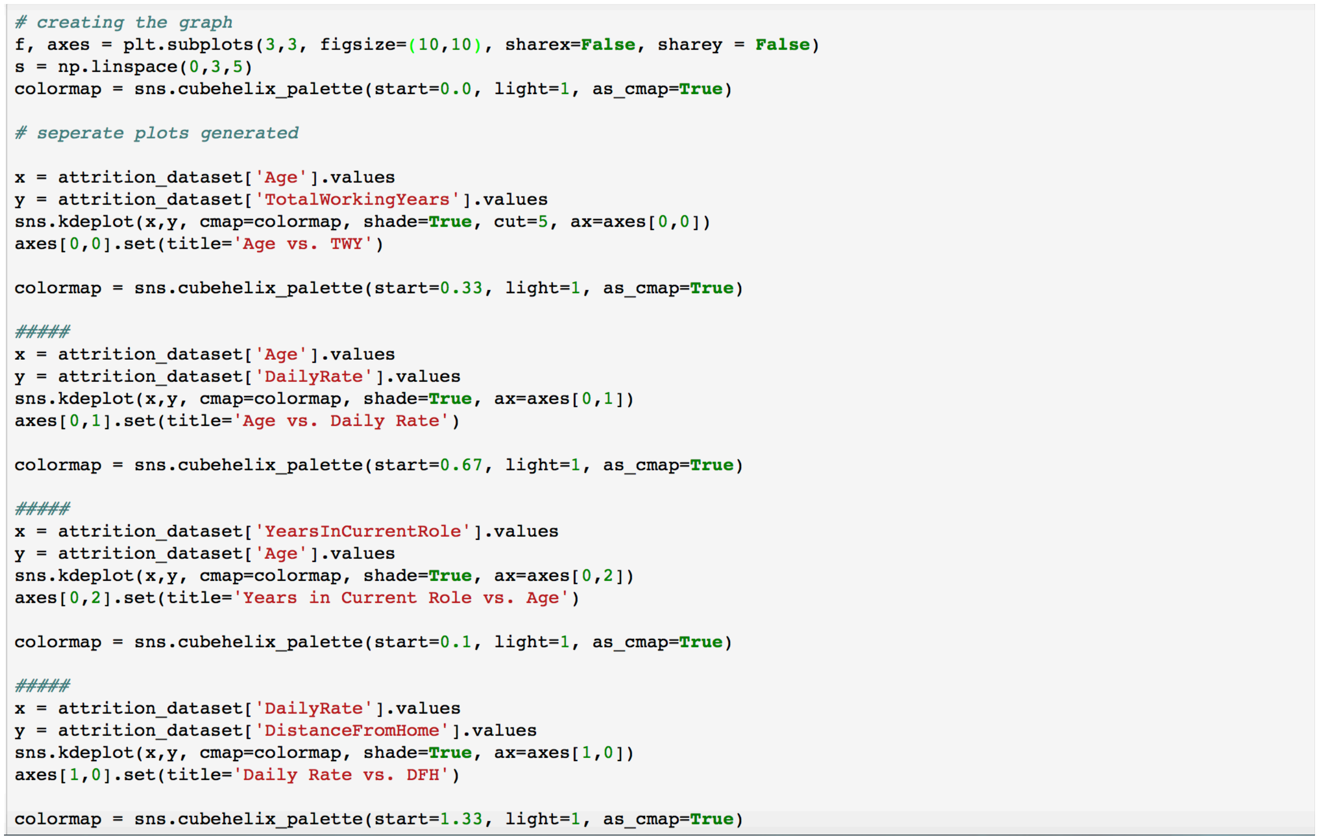


```
In [5]: attrition_dataset.isnull().any()

Out[5]: Age                         False
        Attrition                   False
        BusinessTravel              False
        DailyRate                   False
        Department                  False
        DistanceFromHome            False
        Education                   False
        EducationField              False
        EmployeeCount               False
        EmployeeNumber              False
        EnvironmentSatisfaction     False
        Gender                      False
        HourlyRate                  False
        JobInvolvement              False
        JobLevel                    False
        JobRole                     False
        JobSatisfaction             False
        MaritalStatus               False
        MonthlyIncome               False
        MonthlyRate                 False
        NumCompaniesWorked          False
        Over18                      False
        OverTime                    False
        PercentSalaryHike           False
        PerformanceRating           False
        RelationshipSatisfaction    False
        StandardHours               False
        StockOptionLevel            False
        TotalWorkingYears           False
        TrainingTimesLastYear       False
        WorkLifeBalance             False
        YearsAtCompany              False
        YearsInCurrentRole          False
        YearsSinceLastPromotion     False
        YearsWithCurrManager        False
        dtype: bool
```

Data Distributions

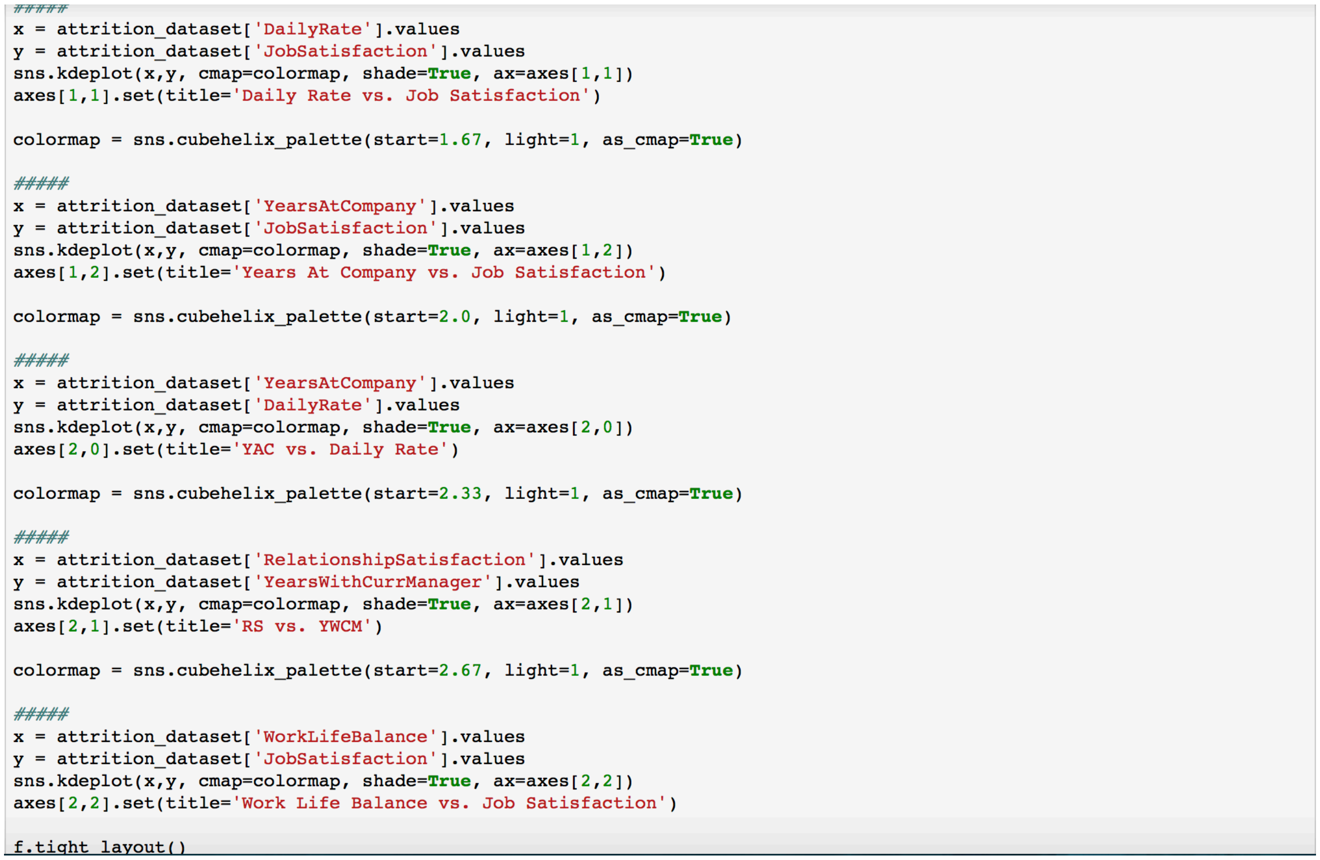
After the data in the dataset was confirmed to be continuous, the distributions between the variables were observed to identify any abnormalities. This was observed using a KDE plot.

First a template of subplots were made, and then each plot was specified.



```
# creating the graph
f, axes = plt.subplots(3,3, figsize=(10,10), sharex=False, sharey = False)
s = np.linspace(0,3,5)
colormap = sns.cubehelix_palette(start=0.0, light=1, as_cmap=True)

# seperate plots generated

x = attrition_dataset['Age'].values
y = attrition_dataset['TotalWorkingYears'].values
sns.kdeplot(x,y, cmap=colormap, shade=True, cut=5, ax=axes[0,0])
axes[0,0].set(title='Age vs. TWY')

colormap = sns.cubehelix_palette(start=0.33, light=1, as_cmap=True)

#####
x = attrition_dataset['Age'].values
y = attrition_dataset['DailyRate'].values
sns.kdeplot(x,y, cmap=colormap, shade=True, ax=axes[0,1])
axes[0,1].set(title='Age vs. Daily Rate')

colormap = sns.cubehelix_palette(start=0.67, light=1, as_cmap=True)

#####
x = attrition_dataset['YearsInCurrentRole'].values
y = attrition_dataset['Age'].values
sns.kdeplot(x,y, cmap=colormap, shade=True, ax=axes[0,2])
axes[0,2].set(title='Years in Current Role vs. Age')

colormap = sns.cubehelix_palette(start=0.1, light=1, as_cmap=True)

#####
x = attrition_dataset['DailyRate'].values
y = attrition_dataset['DistanceFromHome'].values
sns.kdeplot(x,y, cmap=colormap, shade=True, ax=axes[1,0])
axes[1,0].set(title='Daily Rate vs. DFH')

colormap = sns.cubehelix_palette(start=1.33, light=1, as_cmap=True)
```

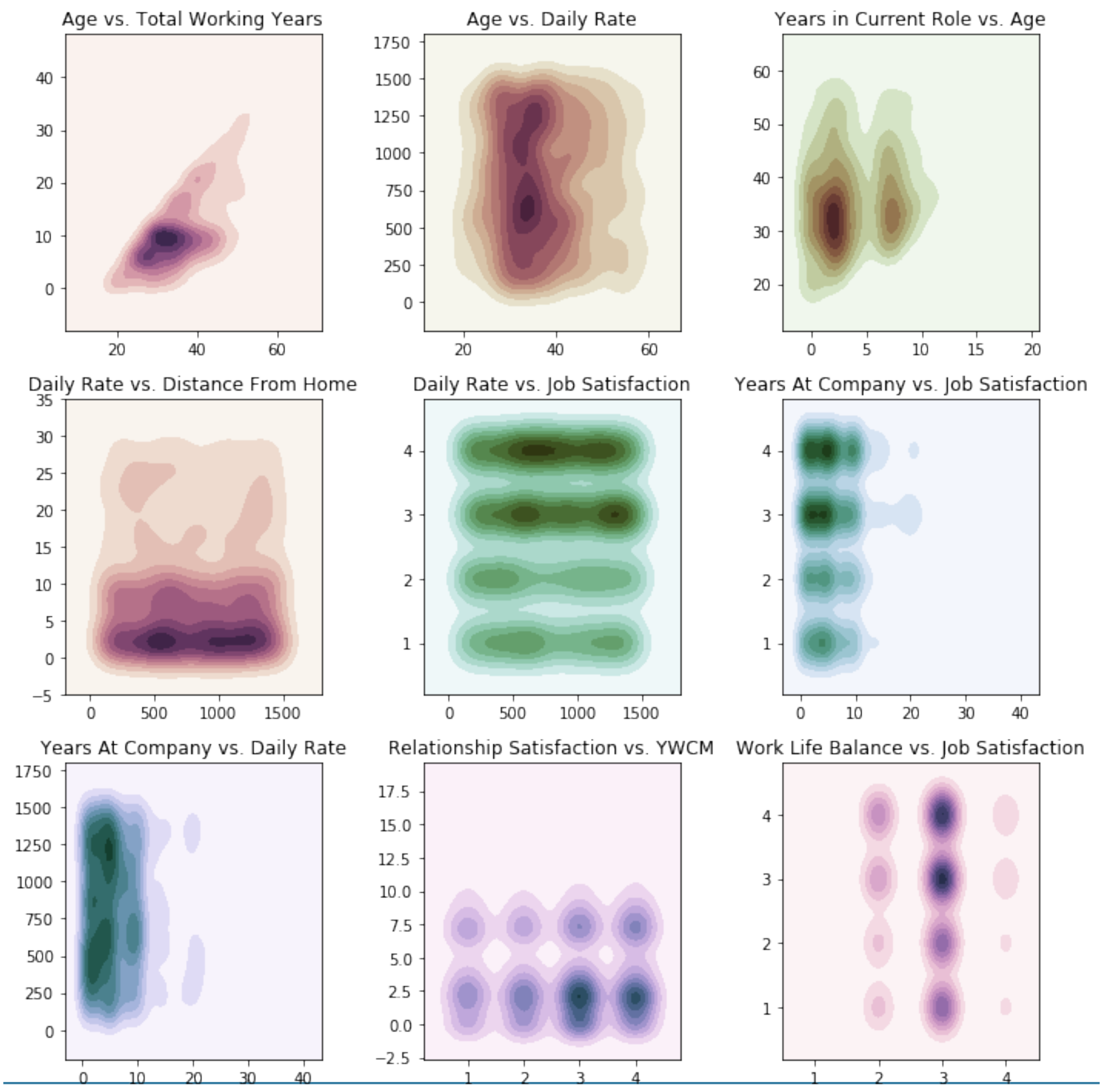
Continuation of plots being created.



```
#####
x = attrition_dataset['DailyRate'].values
y = attrition_dataset['JobSatisfaction'].values
sns.kdeplot(x,y, cmap=colormap, shade=True, ax=axes[1,1])
axes[1,1].set(title='Daily Rate vs. Job Satisfaction')

colormap = sns.cubehelix_palette(start=1.67, light=1, as_cmap=True)

#####
x = attrition_dataset['YearsAtCompany'].values
y = attrition_dataset['JobSatisfaction'].values
sns.kdeplot(x,y, cmap=colormap, shade=True, ax=axes[1,2])
axes[1,2].set(title='Years At Company vs. Job Satisfaction')

colormap = sns.cubehelix_palette(start=2.0, light=1, as_cmap=True)

#####
x = attrition_dataset['YearsAtCompany'].values
y = attrition_dataset['DailyRate'].values
sns.kdeplot(x,y, cmap=colormap, shade=True, ax=axes[2,0])
axes[2,0].set(title='YAC vs. Daily Rate')

colormap = sns.cubehelix_palette(start=2.33, light=1, as_cmap=True)

#####
x = attrition_dataset['RelationshipSatisfaction'].values
y = attrition_dataset['YearsWithCurrManager'].values
sns.kdeplot(x,y, cmap=colormap, shade=True, ax=axes[2,1])
axes[2,1].set(title='RS vs. YWCM')

colormap = sns.cubehelix_palette(start=2.67, light=1, as_cmap=True)

#####
x = attrition_dataset['WorkLifeBalance'].values
y = attrition_dataset['JobSatisfaction'].values
sns.kdeplot(x,y, cmap=colormap, shade=True, ax=axes[2,2])
axes[2,2].set(title='Work Life Balance vs. Job Satisfaction')

f.tight_layout()
```

Results

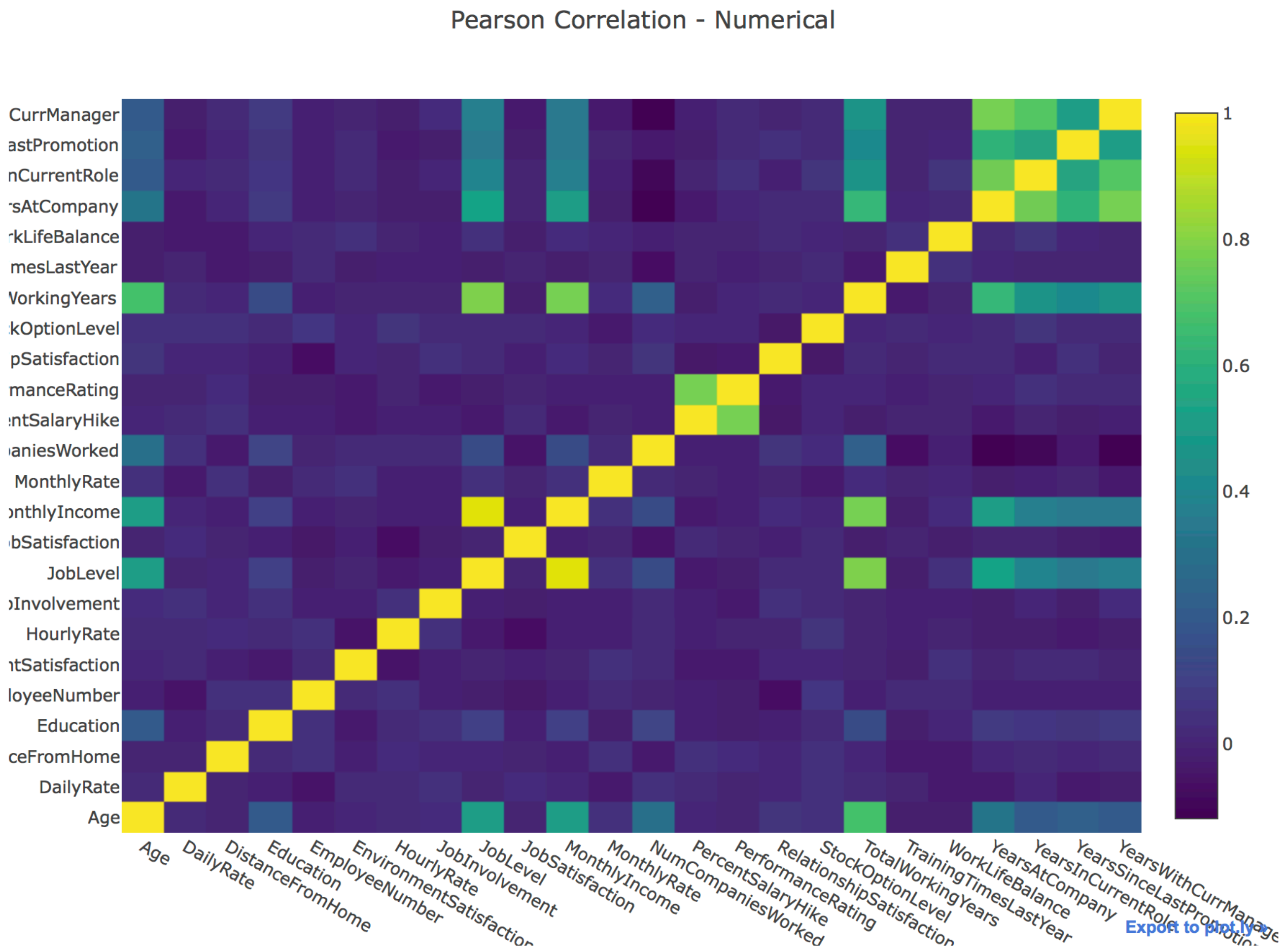
Legend of Plot:
YWCM = Years With Current Manager



Data Correlations

The data was then analyzed to find correlations between its specific features. This was accomplished by visualizing a correlation matrix obtained from the heat map function.

```python
# creating list of only numerical categories
numerical = [u'Age', u'DailyRate', u'DistanceFromHome', u'Education',
                    u'EmployeeNumber', u'EnvironmentSatisfaction',
                    u'HourlyRate', u'JobInvolvement', u'JobLevel',
                    u'JobSatisfaction', u'MonthlyIncome', u'MonthlyRate',
                    u'NumCompaniesWorked', u'PercentSalaryHike', u'PerformanceRating',
                    u'RelationshipSatisfaction', u'StockOptionLevel', u'TotalWorkingYears',
                    u'TrainingTimesLastYear', u'WorkLifeBalance', u'YearsAtCompany',
                    u'YearsInCurrentRole', u'YearsSinceLastPromotion', u'YearsWithCurrManager']

data = [
        go.Heatmap(
                z=attrition_dataset[numerical].astype(float).corr().values,
                x=attrition_dataset[numerical].columns.values,
                y=attrition_dataset[numerical].columns.values,
                colorscale='Viridis',
                reversescale=False,
                text=True,
                opacity=1.0
            )
        ]
layout = go.Layout(
        title='Pearson Correlation - Numerical',
        xaxis = dict(ticks='', nticks=36),
        yaxis = dict(ticks=''),
        width = 900, height =700,
        )

fig = go.Figure(data=data,layout=layout)
py.iplot(fig, filename='labelled-heatmap')
```



From the two visualizations created, we can observe that there is very little correlation. This is ideal, because too much correlation can lead to redundant features when the predictive model is being trained.

MODIFICATION OF THE DATASET

The dataset now must be modified before predictive models can be run on it:

First, the categorical variables must be turned into dummy variables so that the data can be analyzed.

```python
# creates empty list for catergorical data

categorical = []
for col, value in attrition_dataset.iteritems():
    if value.dtype == 'object':
        categorical.append(col)
        # adds column if type is object

# stores numerical columns in a list using .difference
numerical = attrition_dataset.columns.difference(categorical)

# and storing categorical data in dataframe
dataset_cat = attrition_dataset[categorical]
# need to drop target column, attrition
dataset_cat = dataset_cat.drop(['Attrition'], axis=1)
# get_dummies method creates dummy variables from categorical variables in one line of code
dataset_cat = pd.get_dummies(dataset_cat)
dataset_cat.head()
```

	BusinessTravel_Non-Travel	BusinessTravel_Travel_Frequently	BusinessTravel_Travel_Rarely	Department_Human Resources	Department_Research & Development	Department_Sales	EducationF
0	0	0	1	0	0	1	
1	0	1	0	0	1	0	
2	0	0	1	0	1	0	
3	0	1	0	0	1	0	
4	0	0	1	0	1	0	

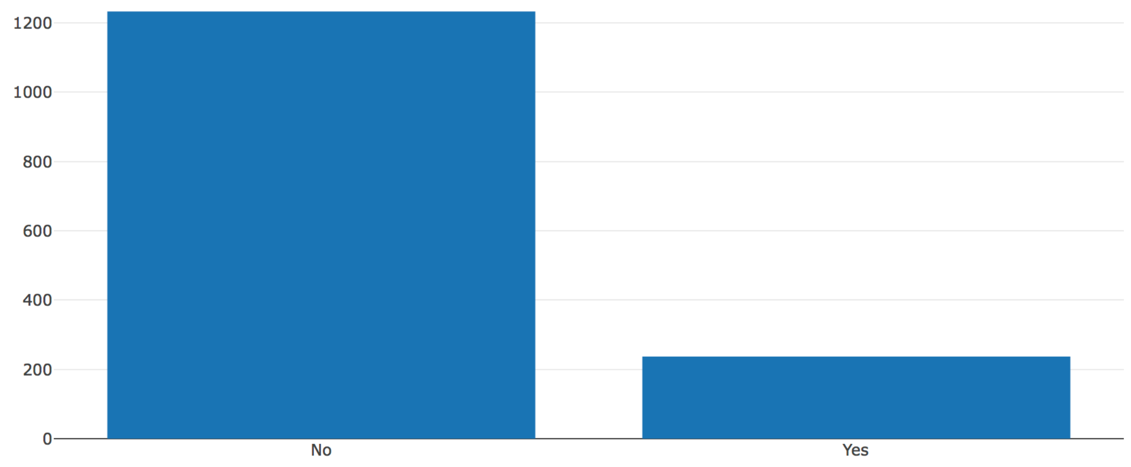
5 rows × 29 columns

Once the categorical variables are changed into dummy variables and are contained in a dataframe, the numerical values are converted into a dataframe. This allows the concatenation of both types of variables.

```
# store numerical data in a datafram
dataset_num = dataset[numerical]
# can process numerical data here if needed

# concat both modified dataframes
dataset_final = pd.concat([dataset_num, dataset_cat], axis=1)
```

Next, the categorical variable of Attrition, originally displayed “Yes” or “No”, was converted into a binary result of “Yes” = 1 and “No” = 0. The total number of each category were observed via a simple bar graph.

```
#######################
# make dictionary for our target (attrition)
target_variable = {'Yes':1, 'No':0}
# add in new column
target = attrition_dataset["Attrition"].apply(lambda x:target_variable[x])


# see the total number of Attrition
data = [go.Bar(
        x=attrition_dataset['Attrition'].value_counts().index.values,
        y=attrition_dataset['Attrition'].value_counts().values
        )]
py.iplot(data, filename='basic-bar')
```



From these results, we see a large discrepancy between the two categories. This could be a problem later on when implementing the machine learning models. Typically if the rate of event is less than 5%, the data is considered to be part of an imbalanced dataset. Attrition here occurs at a rate of 16%, so we will just use normal predictive methods. However, we will keep this large discrepancy in mind and create separate test data that is modified to account of the discrepancy. In this experiment, SMOTE was used on the test data. SMOTE is a technique that uses a subset of the minority class to produce to produce new, synthetic instances.

```
# split data
from sklearn.cross_validation import train_test_split

train_x, test_x, train_y, test_y = train_test_split(dataset_final, target, train_size =0.75, random_state=0)
```

IMPLEMENTATION OF PREDICTIVE MODELS

Now that the dataset is correctly modified, we are now ready to use our machine learning algorithms on the data:

Predictive Model 1: Random Forest Classifier

First the parameters of the random forest classifier are defined and the model is created.

```
# initializing random forest parameters
rf_params = {
        'n_jobs': -1,
        'n_estimators': 800,
        'warm_start': True,
        'max_features': 0.3,
        'max_depth': 9,
        'min_samples_leaf': 2,
        'max_features': 'sqrt',
        'random_state': 0,
        'verbose': 0
        }

# create model
rf = RandomForestClassifier(**rf_params)
```

The model was then trained using both the original data, as well as the data modified by SMOTE.

Original Data:

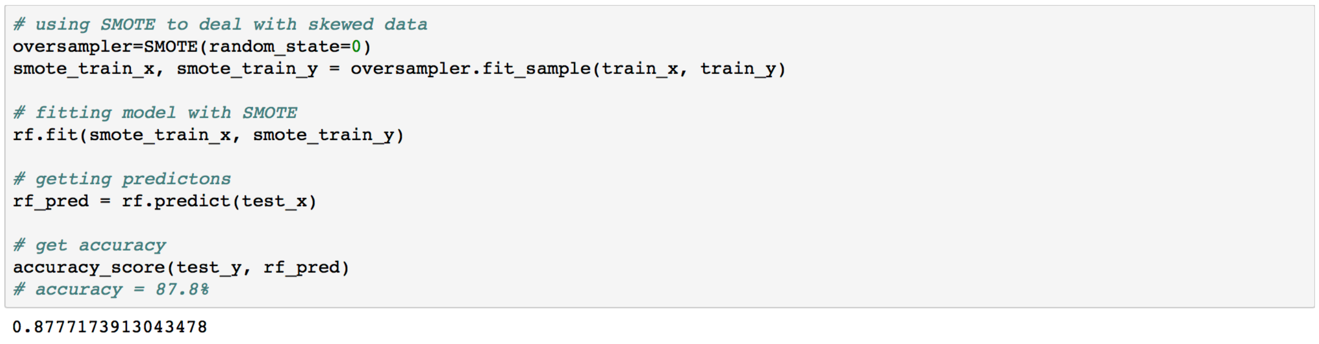
Accuracy of Model Prediction: 86.7%

```
# fitting model
rf.fit(train_x, train_y)

# getting predictons
rf_pred = rf.predict(test_x)

# get accuracy
accuracy_score(test_y, rf_pred)
# accuracy = 86.7%
```

```
0.8668478260869565
```

Data Modified By SMOTE:

Accuracy of Model Prediction: 87.8%



```
# using SMOTE to deal with skewed data
oversampler=SMOTE(random_state=0)
smote_train_x, smote_train_y = oversampler.fit_sample(train_x, train_y)

# fitting model with SMOTE
rf.fit(smote_train_x, smote_train_y)

# getting predictons
rf_pred = rf.predict(test_x)

# get accuracy
accuracy_score(test_y, rf_pred)
# accuracy = 87.8%
```

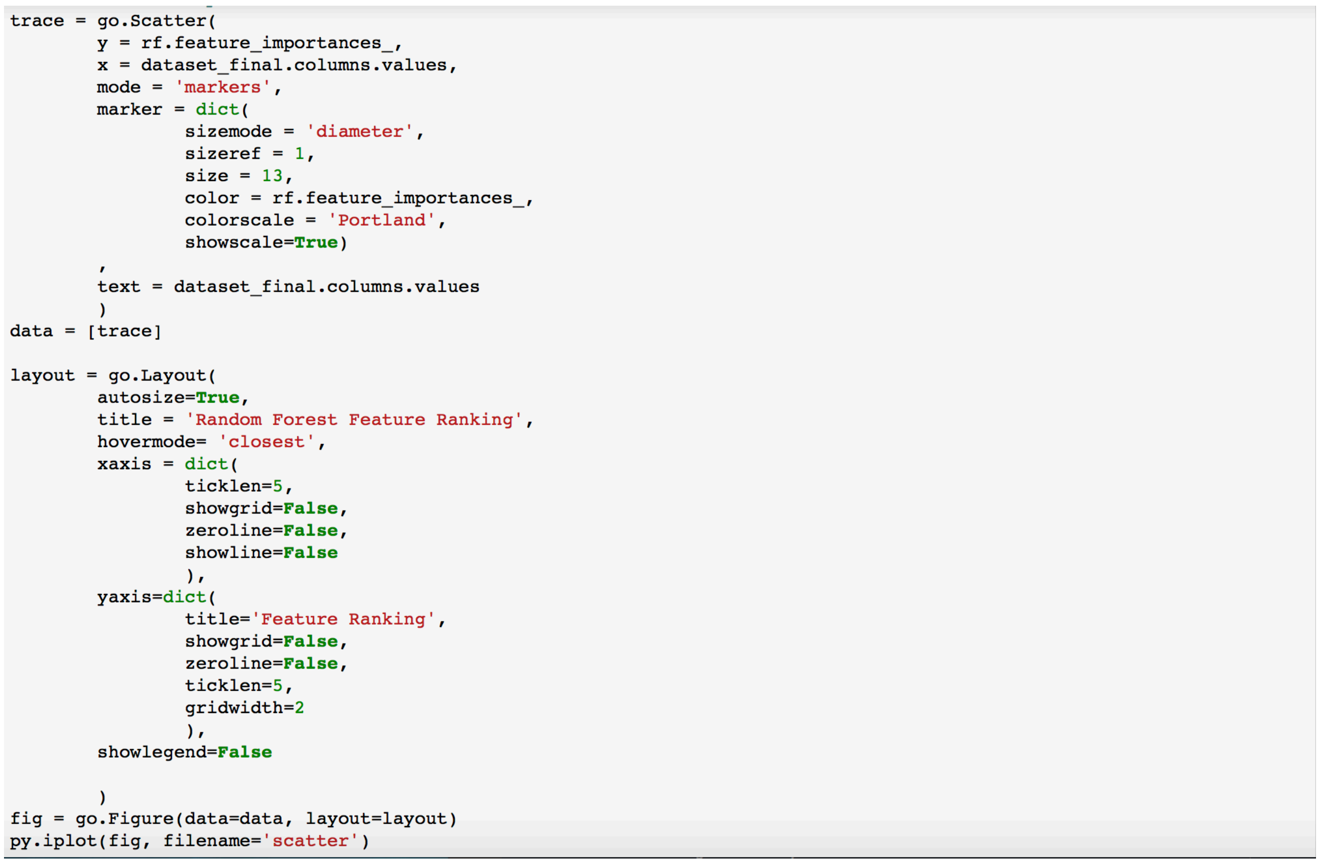
```
0.8777173913043478
```

Conclusions:

Both methods increase the accuracy of prediction, however not by much. The random forest classifier that was run on the SMOTE modified data had a slightly higher accuracy rate at 87.8%. Another method must be trialed to see if better results can be obtained.

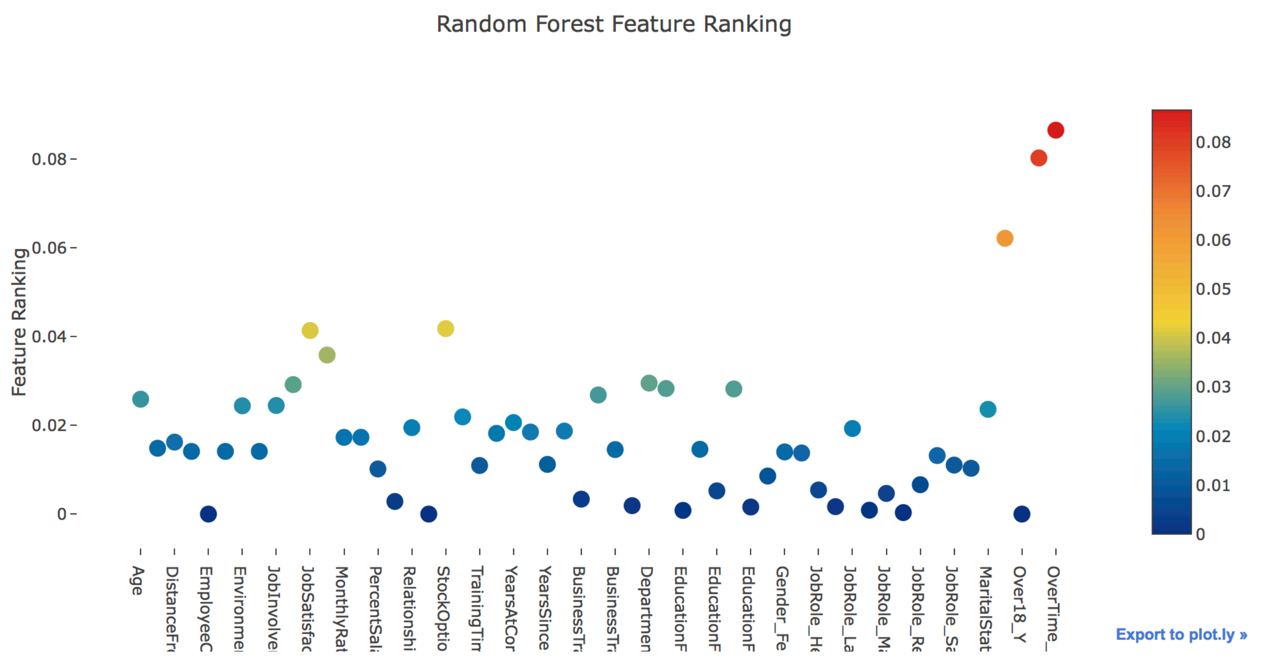
Feature Ranking With The Random Forest Classifier:

With feature ranking, we are able to see which features contributed the most to the predictive analysis.



```
trace = go.Scatter(
        y = rf.feature_importances_,
        x = dataset_final.columns.values,
        mode = 'markers',
        marker = dict(
                sizemode = 'diameter',
                sizeref = 1,
                size = 13,
                color = rf.feature_importances_,
                colorscale = 'Portland',
                showscale=True)
        ,
        text = dataset_final.columns.values
        )
data = [trace]

layout = go.Layout(
        autosize=True,
        title = 'Random Forest Feature Ranking',
        hovermode= 'closest',
        xaxis = dict(
                ticklen=5,
                showgrid=False,
                zeroline=False,
                showline=False
                ),
        yaxis=dict(
                title='Feature Ranking',
                showgrid=False,
                zeroline=False,
                ticklen=5,
                gridwidth=2
                ),
        showlegend=False

        )
fig = go.Figure(data=data, layout=layout)
py.iplot(fig, filename='scatter')
```

Results of Feature Ranking:



From the graph we see that the most importance factors contributing to the attrition rate prediction are overtime. Marital status also shows significance.

Predictive Model 2:

First the parameters of the Gradient Boosted Classifier are defined and the model is created. This model is an ensemble method similar to the Random Forest Classifier.

```
# initializing gradient boosting parameters
gb_params = {'n_estimators': 500,
             'max_features' : 0.9,
             'learning_rate': 0.2,
             'max_depth': 11,
             'min_samples_leaf': 2,
             'subsample': 1,
             'max_features': 'sqrt',
             'random_state': 0,
             'verbose': 0
             }
# creating model
gb = GradientBoostingClassifier(**gb_params)
```

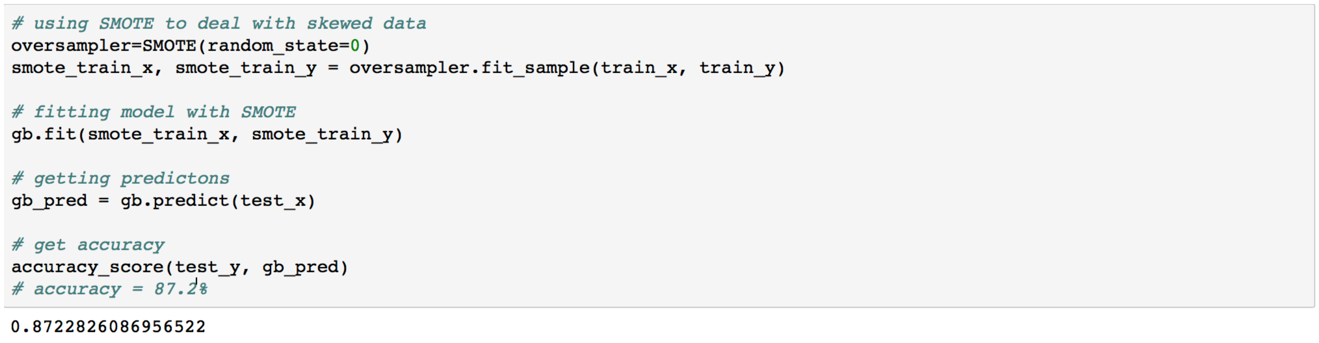
The model was then trained using both the original data, as well as the data modified by SMOTE.

Original Data:

```
# fitting model
gb.fit(train_x, train_y)

# getting predictons
gb_pred = rf.predict(test_x)

# get accuracy
accuracy_score(test_y, rf_pred)
# accuracy = 86.7%
```

Accuracy of Model Prediction: 86.7%

Data Modified By SMOTE:



```
# using SMOTE to deal with skewed data
oversampler=SMOTE(random_state=0)
smote_train_x, smote_train_y = oversampler.fit_sample(train_x, train_y)

# fitting model with SMOTE
gb.fit(smote_train_x, smote_train_y)

# getting predictons
gb_pred = gb.predict(test_x)

# get accuracy
accuracy_score(test_y, gb_pred)
# accuracy = 87.2%
```

```
0.8722826086956522
```

Accuracy of Model Prediction: 87.2%

Conclusions:

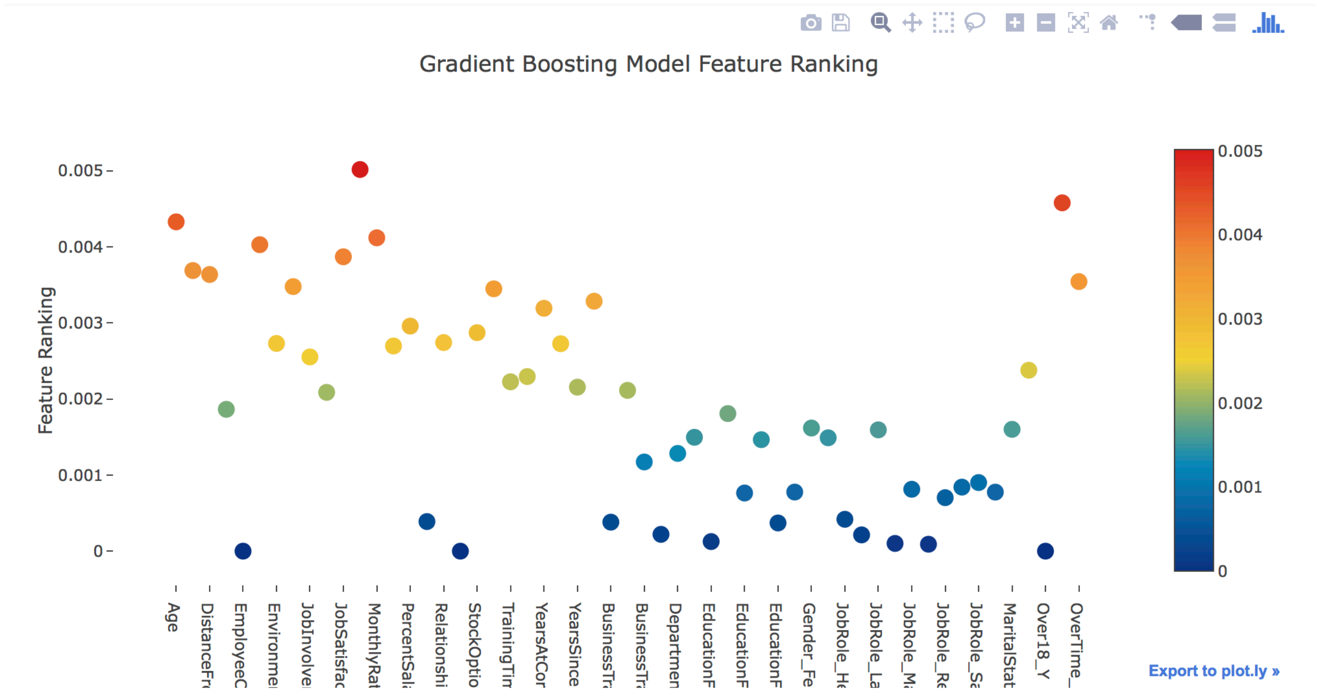
Almost identical to the Random Forest Classifier, both methods increase the accuracy of prediction, however not by much. The gradient boosted classifier that was run on the SMOTE modified data had a slightly higher accuracy rate at 87.2%. This percentage of accuracy however, was lower than the accuracy of the random forest classifier.

Feature Ranking With The Gradient Boosted Classifier:

With feature ranking, we are able to see which features contributed the most to the predictive analysis.

```python
trace = go.Scatter(
        y = gb.feature_importances_,
        x = dataset_final.columns.values,
        mode = 'markers',
        marker = dict(
                sizemode = 'diameter',
                sizeref = 1,
                size = 13,
                color = gb.feature_importances_,
                colorscale = 'Portland',
                showscale=True)
        ,
        text = dataset_final.columns.values
        )
data = [trace]

layout = go.Layout(
        autosize=True,
        title = 'Gradient Boosting Model Feature Ranking',
        hovermode= 'closest',
        xaxis = dict(
                ticklen=5,
                showgrid=False,
                zeroline=False,
                showline=False
                ),
        yaxis=dict(
                title='Feature Ranking',
                showgrid=False,
                zeroline=False,
                ticklen=5,
                gridwidth=2
                ),
        showlegend=False

        )
fig = go.Figure(data=data, layout=layout)
py.iplot(fig, filename='scatter')
```

Results of Feature Ranking:



From the graph we see that the most importance factors contributing to the attrition rate prediction are overtime, monthly income, and age. Other strong contributors include distance from home, job satisfaction, and monthly/ hourly rate.

Final Conclusions:

Based on the proposed predictive models, the most accurate method was the Random Forest Classifier with the SMOTE modified data. This combination produced an 87.8% accuracy rate in predicting whether an employee will quit or not. Though this number seems high, the original distribution of employee attrition rate was 84% to 16%. This means that a random guess that an employee would stay gives an 84% chance of being accurate. This model has increased these odds by more than 3%. The main contributing factors for employee attrition in this model were overtime.

For further exploration, different models can be tested on the data such as Naive Bayes, Kernel SVM, etc. As well as different models being tested, different parameters could also be tested on the models tested in this experiment. This could potentially lead to a higher accuracy prediction rate.