In the Wake of the Great Resignation: Predicting Employee Churn

Halle Davis

Ivan Chavez

Uyen Pham

Shiley-Marcos School of Engineering, University of San Diego

Abstract

The nation is in the midst of the Great Resignation: a historic wave of voluntary job resignations following the COVID-19 pandemic. In light of these historic labor market circumstances, it is more important than ever to understand your internal labor conditions. The paper's objective is to create a model that predicts whether or not an employee will voluntarily resign based on HR's information about the employee. If the model is successful, a business will be able to leverage it to know what existing employees need additional effort to be retained or, in more dire circumstances, what the business's labor needs will be based on predicted churn. Using dummy data created for a hackathon, the team completed the objective by creating 5 models and determining which model type is the best fit for the paper's use case: Logistic Regression, Naive Bayes, CART, C5, and Random Forest. In order to compare the final models, the group decided that the most important evaluation metric was the F1 score. The reason why the team selected F1 is because the F1 score weighs both true positive rate (precision) and true negative rate (recall), meaning that the model will be strong at prediction both positively and negatively. In this paper's use case both the positive cases and the negative cases are important to identify. As a result, Random Forest is the best model because it has the highest F1-score out of all the models.

Keywords: great resignation, labor shortage, churn, voluntary resignation, machine learning, data mining, logistic regression, naive bayes, CART, C5, random forest

Table of Contents

In the Wake of the Great Resignation: Predicting Employee Churn

The nation is in the midst of the Great Resignation: a historic wave of voluntary job resignations following the COVID-19 pandemic. In 2022, the Bureau of Labor Statistics reported that the number of job openings has surpassed the number of unemployed people (Bureau of Labor Statistics, 2022). In light of these historic labor market circumstances, it is more important than ever to understand your internal labor conditions. Without data that can drive either proper retention efforts or labor forecasting, a business can personally succumb to the global labor shortage which can lead to lowered customer satisfaction, an inability to fulfill orders and obligations, and, in dire cases, complete shutdown.

With the stakes being set, the group aims to utilize data to navigate these extreme conditions. The objective is to create a model that predicts whether or not an employee will voluntarily resign based on HR's information about the employee. If the model is successful, a business will be able to leverage it to know what existing employees need additional effort to be retained or, in extreme circumstances, what the business's labor needs will be based on predicted churn. In order to find the best model for this use case, the group will run 5 different models and compare them on a selected evaluation metric and make an ultimate determination on what model will best complete the group's objective.

Methodology

To provide a proof of concept, this report utilizes dummy data specifically generated for this purpose from a private hackathon. There are 9 variables in the dataset and 4653 records. The data columns vary from categorical to numeric data: 'Education', 'JoiningYear', 'City', 'PaymentTier', 'Age', 'Gender', 'EverBenched', 'ExperienceInCurrentDomain', 'LeaveOrNot'.

Education refers to the education level of the employee: whether their highest level of completed education is a Bachelor's Degree, Master's Degree, or PHD. Joining year is the year the employee joined the company. City refers to what office the employee is based in. Payment tier is an ordinal variable categorizing the worker into three payment levels: 1 being the highest and 3 being the lowest. Age and gender are basic descriptors of the employee's demographics. EverBenched refers to whether the employee has ever been left out of a project for 1 month or more. Experience refers to the number of years of experience the worker has in the domain they are employed in. Finally, the dependent variable, LeaveOrNot refers to whether the employee will leave the company in the next two years: 0 for no, they will not leave, and 1 for yes, they will leave.

The data has no outliers in the numerical data and no missing data across the dataset. The group does not anticipate any imbalancing issues: 34.3% of the dataset has a value of 1 and 65.7% of the dataset has a value of 0 for LeaveOrNot. However, 1,889 duplicates were found in the data. The group assumes that the high number of duplicate values are due to the fact that the data is dummy data. In addition, the data has small categorical variables ($k < 10$) and no continuous variables, so there are only so many combinations. There are not any uniquely identifying columns such as employee ID to confirm whether or not these are true duplicates. Ultimately, to be safe, the team dropped the duplicates because the removal still results in a high sample size: 2764.

Table 1 shows a descriptive breakdown of all of the variables, including measures of center and range. Some quick insights that stood out during the exploration phase were that most people began working at the company in 2017, very few of the employees in the talent pool had

6-7 years of experience, and most workers had not recently been benched on a project. As a part of feature construction, the team created a duration variable. The duration variable refers to how long someone has been on assignment with the company. It is currently calculated by subtracting their join year from the year the data was collected (2020). The reason why the team created the duration variable is because in the long term usage of the final model, the group agrees that the constructive information gained from joining year is not the year itself that people joined the company, but rather how long they've been with the company. If the model is to be dispersed, the team does not want different joining years and different data collection years to throw off the predictions.

Upon exploring the independent variables' relationship with the dependent variable, the team discovered that there wasn't a strong correlation between any of the numerical variables and LeaveOrNot. Figure 1 shows the correlation matrix, which will illustrate that the strongest correlation is between joining year and leave or not, with a r-squared equaling 0.15. The group ran various overlaid bar plots to further investigate whether any other variables had an impact on leave or not. Some insights that stood out to the group were that nearly everyone who joined in 2018 (the most recent join year) was categorized as a voluntary resigner, more people were categorized as voluntary resigners in payment tier 2 than any other tier, and age exhibited absolutely no relationship with leave or not. Figure 2 shows the overlaid bar plots that support these cursory insights.

To reiterate, the paper's objective is to create a model that predicts whether or not an employee will voluntarily resign. The models that the report will utilize are Logistic Regression, Naive Bayes, CART Decision Tree, C.5 Decision Tree, and Random Forest. All models will utilize the same 4 variables to ensure accurate comparison. The group decided that the 4 best variables to utilize in the final versions of each model are gender, duration, city, and payment tier based on the outcomes of the EDA, various model iterations, and the decision tree feature importance scores, displayed in Figure 3,

Logistic Regression

Logistic Regression can be categorized as a probabilistic discriminative model, indicating it directly estimates the odds of a data instance x using its attribute values (Tan et al., 2019). Thus the model can be described as shown in Equation 1.

$$p(y) = \frac{exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2 ... + \beta_p x_p)}{1+exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2 ... + \beta_p x_p)} + \varepsilon \tag{1}$$

Since the scope of the project was to predict the odds of an employee resigning, the following 'x' attributes consisting of *City*, *Gender*, *Duration*, and *PaymentTier* were selected to predict the odds of *LeaveOrNot*. The following Equations 2 & 3 establish the initial parametric and descriptive forms of the model.

$$p(LeaveOrNot) = \frac{exp(\beta_0 + \beta_1(City) + \beta_2(Gender) + \beta_3(Duration) + \beta_4(PaymentTier))}{1+exp(\beta_0 + \beta_1(City) + \beta_2(Gender) + \beta_3(Duration) + \beta_4(PaymentTier))} + \varepsilon \tag{2}$$

$$\hat{p}(LeaveOrNot) = \frac{exp(\beta_0 + \beta_1(City) + \beta_2(Gender) + \beta_3(Duration) + \beta_4(PaymentTier))}{1+exp(\beta_0 + \beta_1(City) + \beta_2(Gender) + \beta_3(Duration) + \beta_4(PaymentTier))} \tag{3}$$

It is worth noting that all predictor variables were dummy encoded prior to training the model, consideration for the ordinal attributes *Duration* and *PaymentTier* were taken into account. It was found that encoding all values yielded a stronger model that significantly outperformed the Baseline model shown in Table 2 and Figure 6, which the report will explore more in the Results section.

Naïve Bayes

Unlike the Logistic Regression, Naïve Bayes is classified as a probabilistic classification model, essentially probability is utilized to represent relationships between attributes and class labels. The relationships can be described via Bayes Theorem shown in Equation 4.

$$p(Y = y * | X *) = \frac{p(X^*|Y=y^*)p(Y=y^*)}{p(X^*)} \tag{4}$$

Naïve Bayes was utilized to evaluate the given attributes *City*, *Gender*, *Duration*, and *PaymentTier*.

CART

Figure 4 depicts the CART decision tree, which was set with a max of 5 leaf nodes, but only resulted in 3 beyond the root node. The root node begins the tree with the duration variable, separating people based on whether they have been with the company for less than or equal to 2.5 years or those who have been with the company longer. Those who have been with the company for that short of a time are classified as leaving; the decision is supported with a 0.037 ginin score. Thereafter, the tree asks whether the person is paid at pay tier 3 or less. If they are paid at pay tier 3, they are classified as won't leave, and if they are paid less, the tree further asks if they are a man. If the person is a man, they are categorized as not leaving. If the person is a woman, the tree further asks if they work at the Pune city office. Regardless of whether they work at the Pune city office or not, they are classified as leaving, but the decision on whether they work at the Pune city office or not further reduces the gini score.

C5.0

Similar to the CART model, C5.0 is also a decision tree method. However, it uses entropy for measuring the impurity to decide on splitting points. The model decision tree can be visualized in Figure 5 with the root node starting from Duration and splitting at 2.5 years. Any employee who has worked less than or equal 2.5 years is classified as yes-leave (with entropy = 0.152). With Duration greater than 2.5 years, employees with PaymentTier greater than 2.5 classified as no-leave (with entropy = 0.792). Employees with PaymentTier less than or equal to 2.5 and male gender are no-leave (with entropy = 0.905). On the other hand, with PaymentTier is greater than 2.5 and gender is female who live in Pune city (with entropy = 0.692) would be most certain to leave the job compared to other cities. Figure 5 depicts the complete tree.

Random Forest

Random forest is an ensemble method which builds a series of decision trees (Larose & Larose, 2019). Each tree is built on a random sample with replacement of the original employee training data and chooses the best variable for splitting at each node based on gini criterion. Each instance is given a classification (yes or no to leaving the job) as a vote by every tree. The model is set to build 100 trees and a maximum depth of 5.

Results

All of the models were cross-validated using a 60/40 train/test split. Table 2 shows the evaluation metrics for the 5 classification models. For additional comparison, the team ran a baseline model which classified 50% of the test values as negative and 50% as positive regardless of input values.

Random Forest had the highest accuracy and C5 had the worst – Random Forest was able to correctly predict the classes for 77% of the test data, whereas C5 could only do it for 71.2%. Of course, error rates will follow the same pattern, and Random Forest has the lowest error rate and C5 has the highest. Random Forest also had the highest sensitivity, and CART had the lowest sensitivity. Random Forest was able to capture 54.8% of the true positive records in its predictions, whereas CART could only capture 45.1%. In fact, the CART decision tree's sensitivity was so low that it did not beat the baseline, which captured 50.9% of positive records in its predictions. Continuing to succeed, Random Forest also had the highest specificity, and CART had the lowest. Random Forest was able to capture 92.6% of the true negative records in its predictions, whereas CART could only capture 89%. In terms of precision, Random Forest is still the most successful model, as of the positives it predicted 83.9% were true positives. On the other hand, CART had the lowest precision at 73.5%. Across all F measures, Random Forest was the most successful, which makes sense because F measures combine specificity and precision, and Random Forest had the highest measures of each.

With all of that taken into account, the group decided that the most important evaluation metric for choosing a model in this scenario is the F1 score. The paper's use case requires that positive cases are correctly identified – those that are leaving – so that a business can increase retention efforts to make them stay. However, the use case also requires that the model identifies the negative cases – those that are staying – because knowing what subset of the business's workforce is loyal will help us guide its recruitment efforts, e.g. it is known that the business doesn't need to find a replacement for its accounting team. Therefore, the reason why the team selected F1 is because the F1 score weighs both true positive rate (precision) and true negative rate (recall), meaning that the model will be strong at prediction both positively and negatively. As a result, Random Forest is the best model because it has the highest F1 score out of all the models.

Conclusion

As a final note, this report reiterates our recommendation to use the Random Forest model when attempting to classify employees into voluntary resigners and loyalists. In future studies on this topic, the team recommends that a layer of industry/specialization and job level is added. It would be interesting to see if certain industries and specializations were more loyal or if certain job levels, such as entry level employees, were more likely to leave and career jump. In addition, it is our hope that we are able to complete this project again using real world data.

.

References

Larose, C., & Larose, D. (2019). *Data Science Using Python and R*. Wiley.

Tan, P.-N., Steinbach, M., Karpatne, A., & Kumar, V. (2020). *Introduction to data mining* (Second Edition). Pearson.

Tejashvi. (2021). *Employee Future Prediction (Version 1)*. Retrieved from https://www.kaggle.com/datasets/tejashvi14/employee-future-prediction

U.S. Bureau of Labor Statistics, Unemployment Level [UNEMPLOY], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/UNEMPLOY, April 12, 2022.

Tables

Table 1

Descriptive Statistics of Independent Variables

	Education	City	Payment Tier	Age	Gender	Ever Benched	Experience In Current Domain	Leave Or Not	Duration
Count	2764	2764	2764	2764	2764	2764	2764	2764	2764
Unique	3	3	-	-	2	2	-	2	-
Top	Bachelors	Bangalore	-	-	Male	No	-	No	-
Freq	1971	1171	-	-	1529	2403	-	1676	-
Mean	-	-	2.636	30.953	-	-	2.644	-	4.91
Std	-	-	0.624	5.109	-	-	1.611	-	1.886
Min	-	-	1	22	-	-	0	-	2
25%	-	-	2	27	-	-	1	-	3
50%	-	-	3	30	-	-	2	-	5
75%	-	-	3	35	-	-	4	-	7
Max	-	-	3	41	-	-	4	-	7

Table 2

Evaluation Metrics for the Model

	Logistic Regression	Naive Bayes	CART	C5.0	Random Forest	Baseline
Accuracy	0.740	0.738	0.713	0.712	**0.770**	0.515
Error Rate	0.260	0.262	0.287	0.288	**0.230**	0.485
Sensitivity	0.502	0.520	0.451	0.456	**0.548**	0.509
Specificity	0.906	0.891	0.890	0.891	**0.926**	0.520
Precision	0.790	0.770	0.735	0.746	**0.839**	0.427
AUC	**0.757**	0.745	0.744	0.748	0.715	0.500
F1	0.614	0.620	0.559	0.566	**0.663**	0.464
F2	0.542	0.556	0.489	0.495	**0.589**	0.490
F0.5	0.701	0.702	0.652	0.662	**0.758**	0.441

Note. The highest value for each evaluation metric is bolded.

Figures

Figure 1

Correlation Heatmap

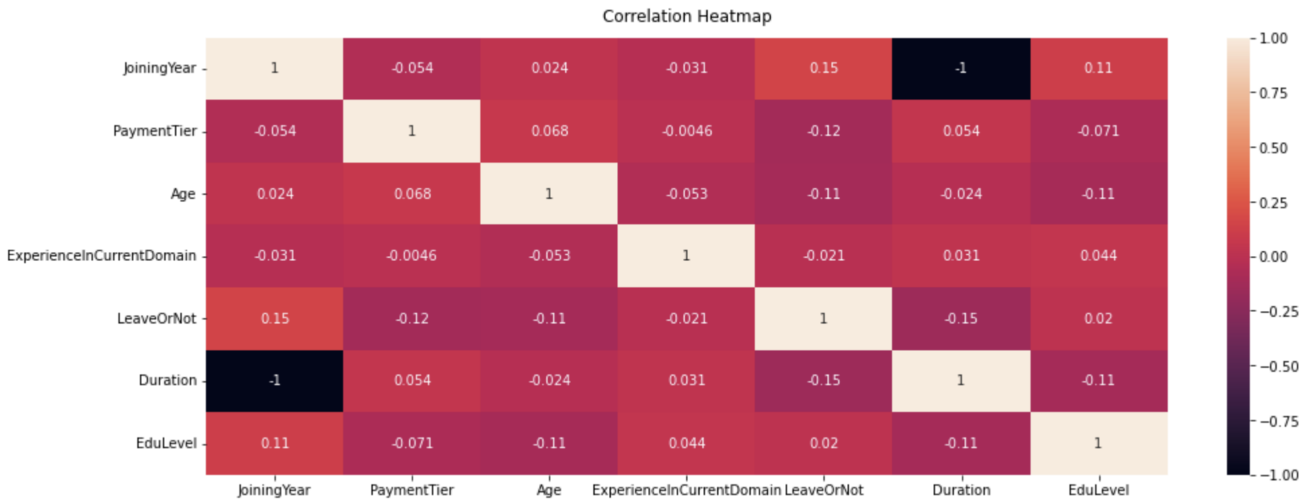


Figure 2

Various Overlaid Barplots

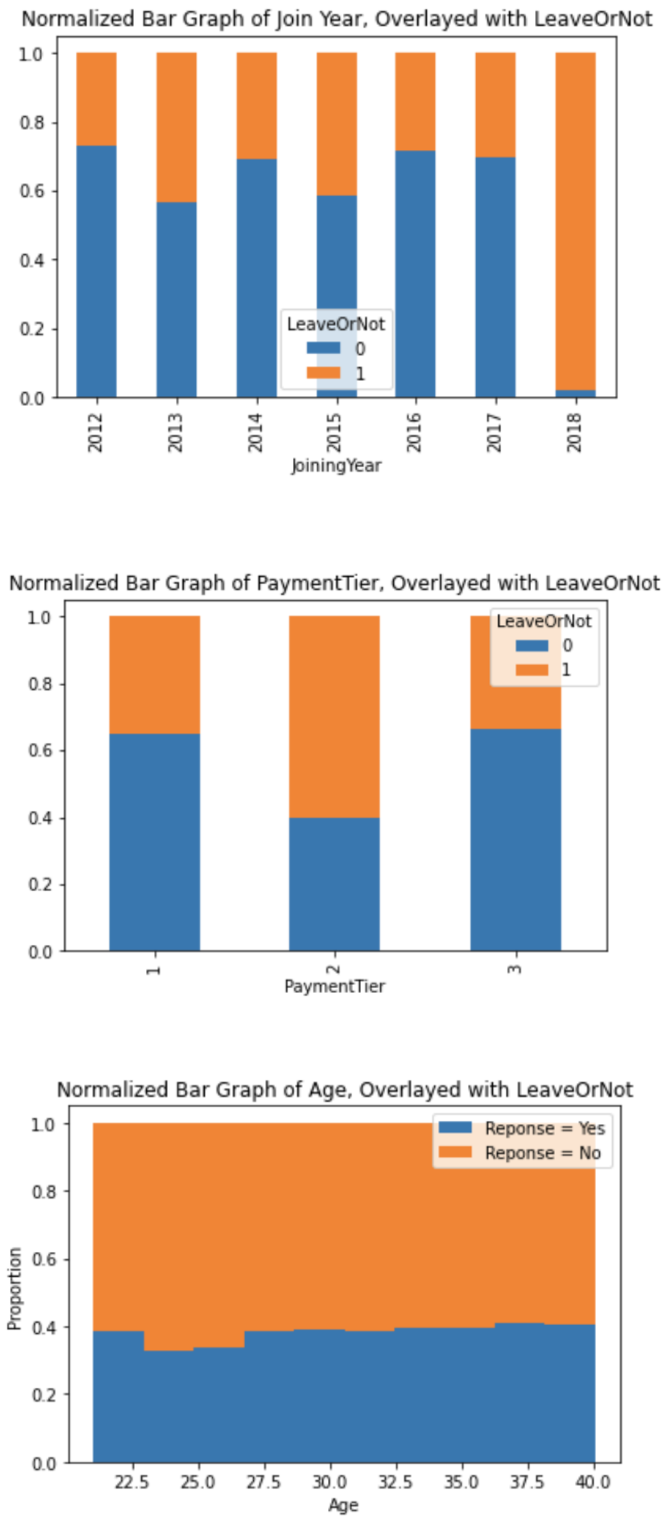


Figure 3

Feature Importance Scores for Independent Variables

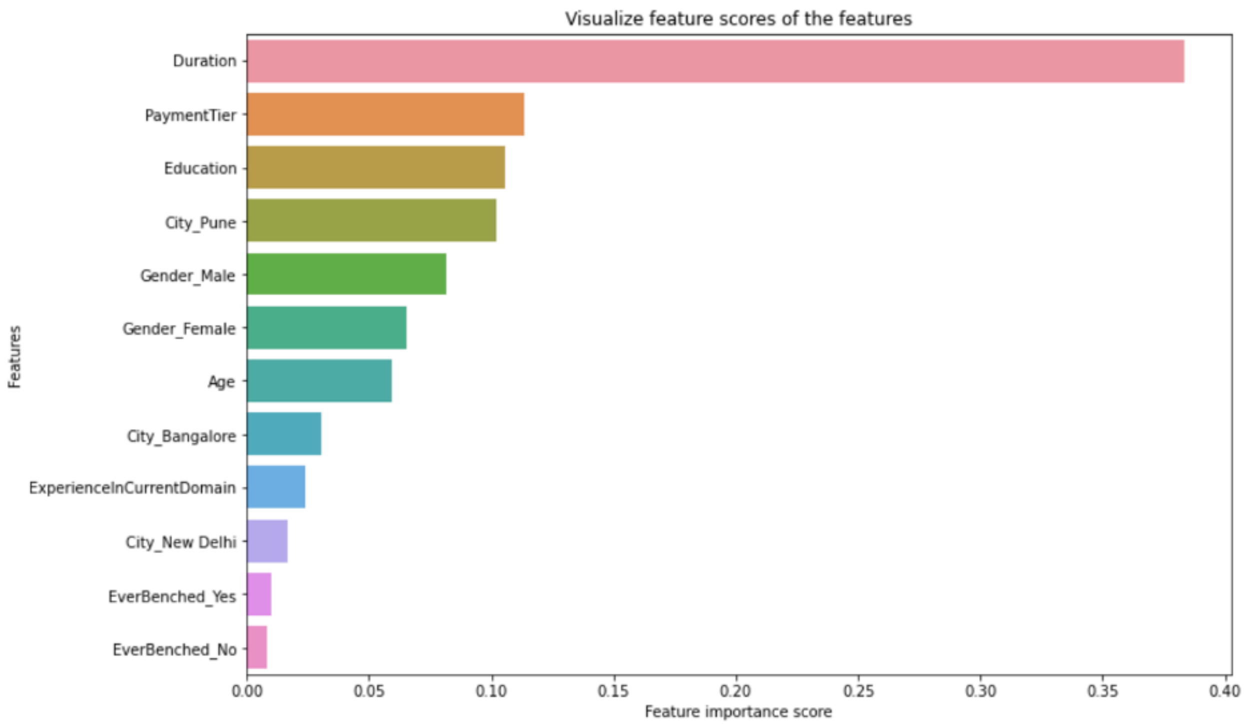


Figure 4

CART Decision Tree

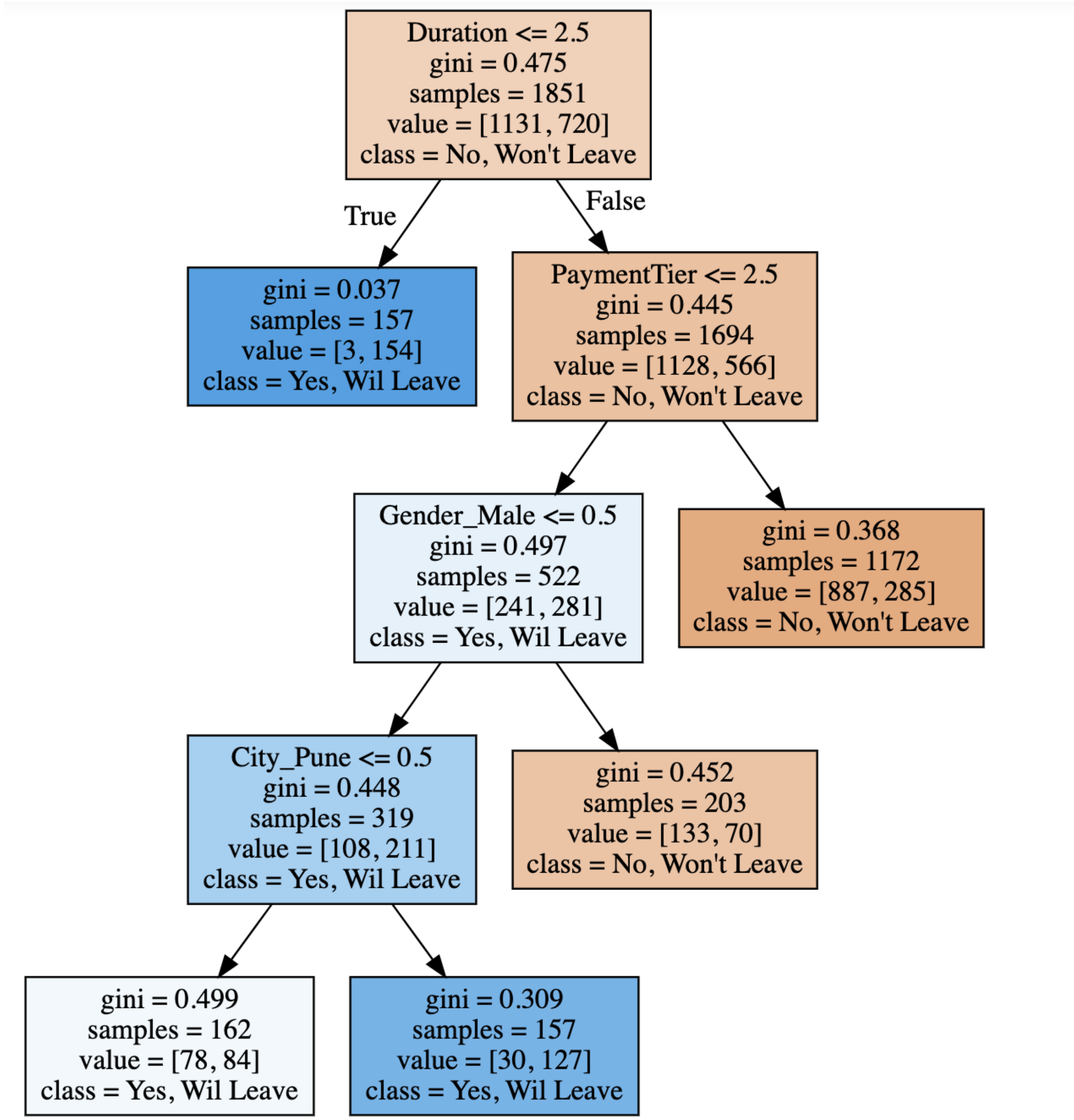


Figure 5

C5.0 Decision Tree

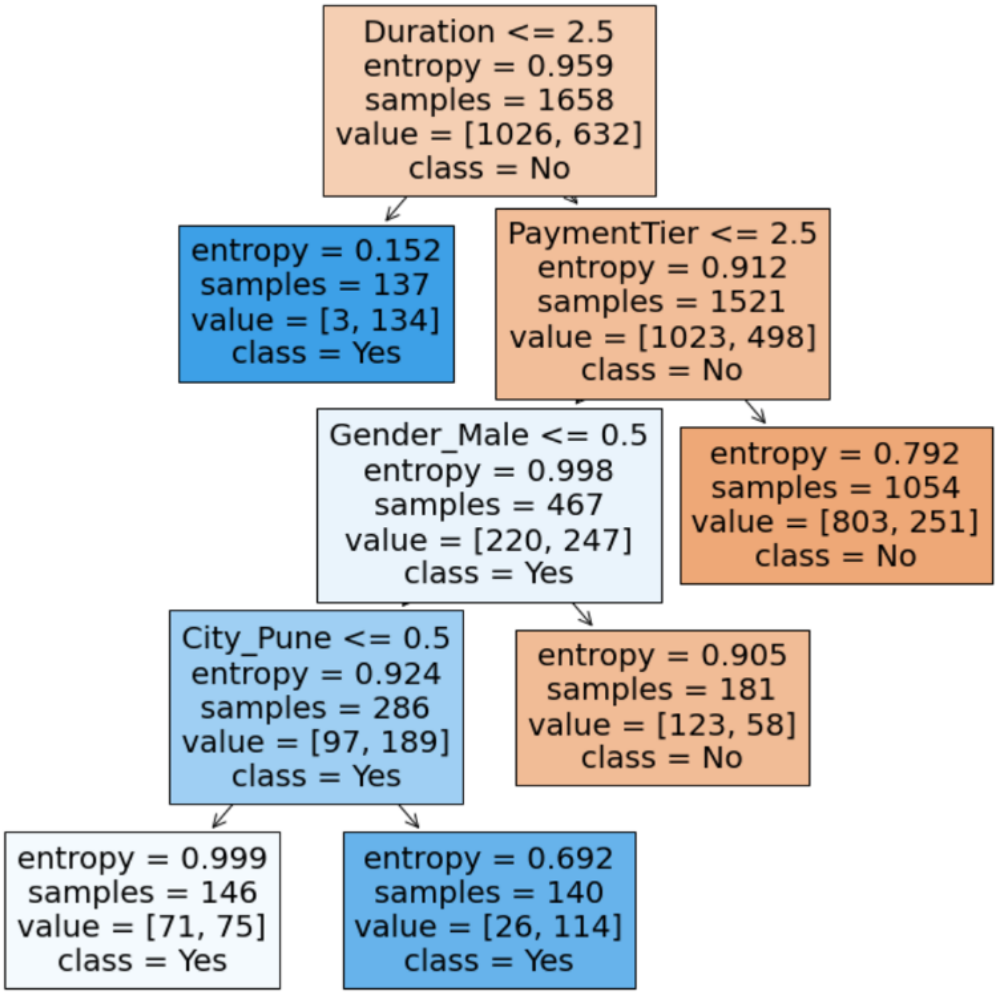
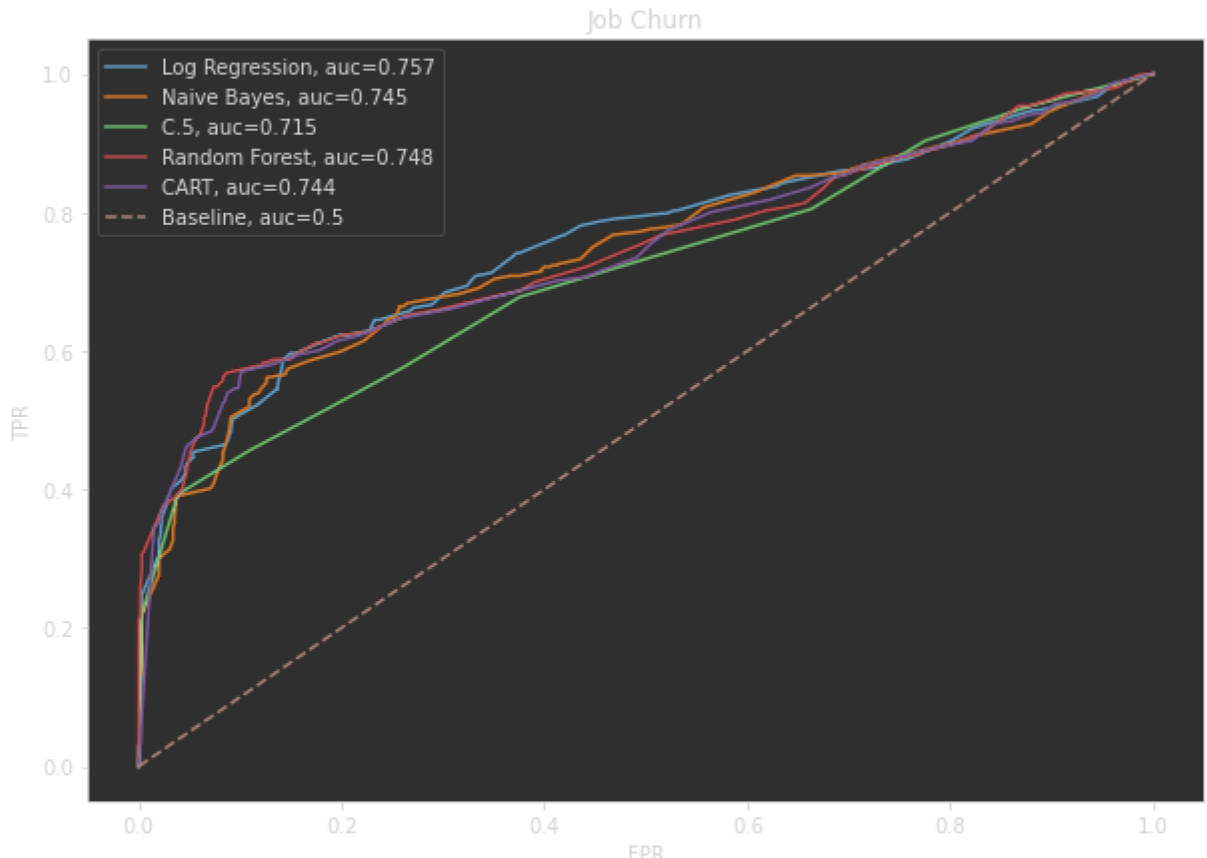


Figure 6

ROC Model Performance



halle_EDA

April 17, 2022

1 Appendix

1.1 Team 3 Final Project: EDA

1.1.1 Importing Libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from scipy.stats import mode
import plotly.graph_objects as go
import random
from scipy import stats
import statsmodels.tools.tools as stattools
from sklearn.tree import DecisionTreeClassifier, export_graphviz
import graphviz
from sklearn.tree import plot_tree
from sklearn import tree
from sklearn.metrics import confusion_matrix
from sklearn.datasets import make_classification
from sklearn.metrics import plot_confusion_matrix
```

1.1.2 Importing CSV Dataset from E-Commerce

[2]:
```
d = pd.read_csv("Employee.csv")
```

1.1.3 Describing Data Shape

[3]:
```
d.shape
```

[3]:
```
(4653, 9)
```

1.1.4 Taking a Peek at the Data

[4]:
```
d.head()
```

[4]:
```
   Education  JoiningYear       City  PaymentTier  Age  Gender EverBenched  \
0  Bachelors         2017  Bangalore            3   34    Male          No
1  Bachelors         2013       Pune            1   28  Female          No
2  Bachelors         2014  New Delhi            3   38  Female          No
3    Masters         2016  Bangalore            3   27    Male          No
4    Masters         2017       Pune            3   24    Male         Yes

   ExperienceInCurrentDomain  LeaveOrNot
0                          0           0
1                          3           1
2                          2           0
3                          5           1
4                          2           1
```

[5]:
```
d.describe(include='all')
```

[5]:
```
        Education  JoiningYear       City  PaymentTier          Age Gender  \
count        4653  4653.000000       4653  4653.000000  4653.000000   4653
unique          3          NaN          3          NaN          NaN      2
top     Bachelors          NaN  Bangalore          NaN          NaN   Male
freq         3601          NaN       2228          NaN          NaN   2778
mean          NaN  2015.062970        NaN     2.698259    29.393295    NaN
std           NaN     1.863377        NaN     0.561435     4.826087    NaN
min           NaN  2012.000000        NaN     1.000000    22.000000    NaN
25%           NaN  2013.000000        NaN     3.000000    26.000000    NaN
50%           NaN  2015.000000        NaN     3.000000    28.000000    NaN
75%           NaN  2017.000000        NaN     3.000000    32.000000    NaN
max           NaN  2018.000000        NaN     3.000000    41.000000    NaN

       EverBenched  ExperienceInCurrentDomain   LeaveOrNot
count         4653                4653.000000  4653.000000
unique           2                        NaN          NaN
top             No                        NaN          NaN
freq          4175                        NaN          NaN
mean           NaN                   2.905652     0.343864
std            NaN                   1.558240     0.475047
min            NaN                   0.000000     0.000000
25%            NaN                   2.000000     0.000000
50%            NaN                   3.000000     0.000000
75%            NaN                   4.000000     1.000000
max            NaN                   7.000000     1.000000
```

[6]:
```
d.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4653 entries, 0 to 4652
Data columns (total 9 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Education                  4653 non-null   object
 1   JoiningYear                4653 non-null   int64
 2   City                       4653 non-null   object
 3   PaymentTier                4653 non-null   int64
 4   Age                        4653 non-null   int64
 5   Gender                     4653 non-null   object
 6   EverBenched                4653 non-null   object
 7   ExperienceInCurrentDomain  4653 non-null   int64
 8   LeaveOrNot                 4653 non-null   int64
dtypes: int64(5), object(4)
memory usage: 327.3+ KB
```

1.1.5 Remove Duplicates

[7]:
```
d.duplicated().sum()
```

[7]:
```
1889
```

There are 1889 duplicates.

[8]:
```
d = d.drop_duplicates()
```

1.1.6 Remove Nulls

[9]:
```
d.isnull().sum() #Check number of nulls
```

[9]:
```
Education                    0
JoiningYear                  0
City                         0
PaymentTier                  0
Age                          0
Gender                       0
EverBenched                  0
ExperienceInCurrentDomain    0
LeaveOrNot                   0
dtype: int64
```

No nulls.

1.1.7 Reduce Redundant Data

[10]:
```
d.columns
```

[10]:
```
Index(['Education', 'JoiningYear', 'City', 'PaymentTier', 'Age', 'Gender',
       'EverBenched', 'ExperienceInCurrentDomain', 'LeaveOrNot'],
      dtype='object')
```

I can't see any columns that we should drop before exploring their relationships.

1.1.8 Update Data Types

[11]:
```
d.dtypes
```

[11]:
```
Education                    object
JoiningYear                   int64
City                         object
PaymentTier                   int64
Age                           int64
Gender                       object
EverBenched                  object
ExperienceInCurrentDomain     int64
LeaveOrNot                    int64
dtype: object
```

No data types need to be updated.

1.1.9 Feature Construction

[12]:
```
d['Duration'] = 2020 - d['JoiningYear']
```

Creating a new variable "duration"

[13]:
```
edlevel = {'Bachelors': 1, 'Masters': 2, 'PHD': 3}

d['EduLevel'] = d['Education'].map(edlevel)
```

Making education level numeric

1.2 2. Data Analysis and Visualization

[14]:
```
d.columns
```

[14]:
```
Index(['Education', 'JoiningYear', 'City', 'PaymentTier', 'Age', 'Gender',
       'EverBenched', 'ExperienceInCurrentDomain', 'LeaveOrNot', 'Duration',
       'EduLevel'],
      dtype='object')
```

1.2.1 Provide Measure of Centrality and Distribution with Visualizations

Joining Year

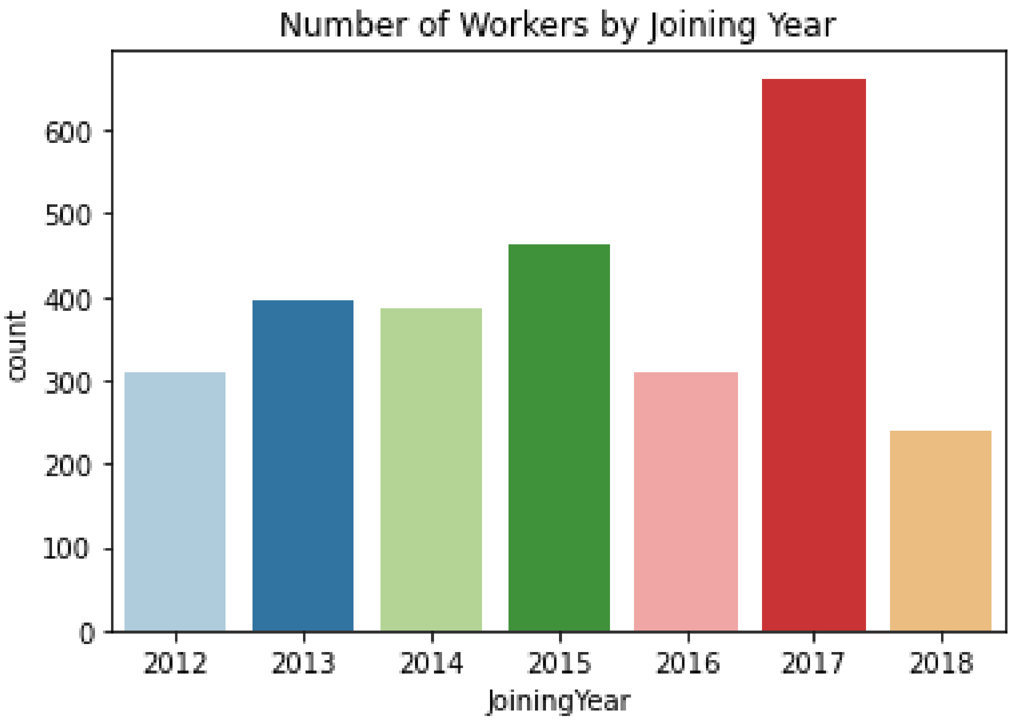
[15]:
```
print("Value Counts")
print(d['JoiningYear'].value_counts())
print("\nMean")
```

```
print(d['JoiningYear'].mean())
print("\nMedian")
print(d['JoiningYear'].median())
print("\nMode")
print(mode(d['JoiningYear']).mode[0])
print("\nStandard Deviation")
print(d['JoiningYear'].std())
```

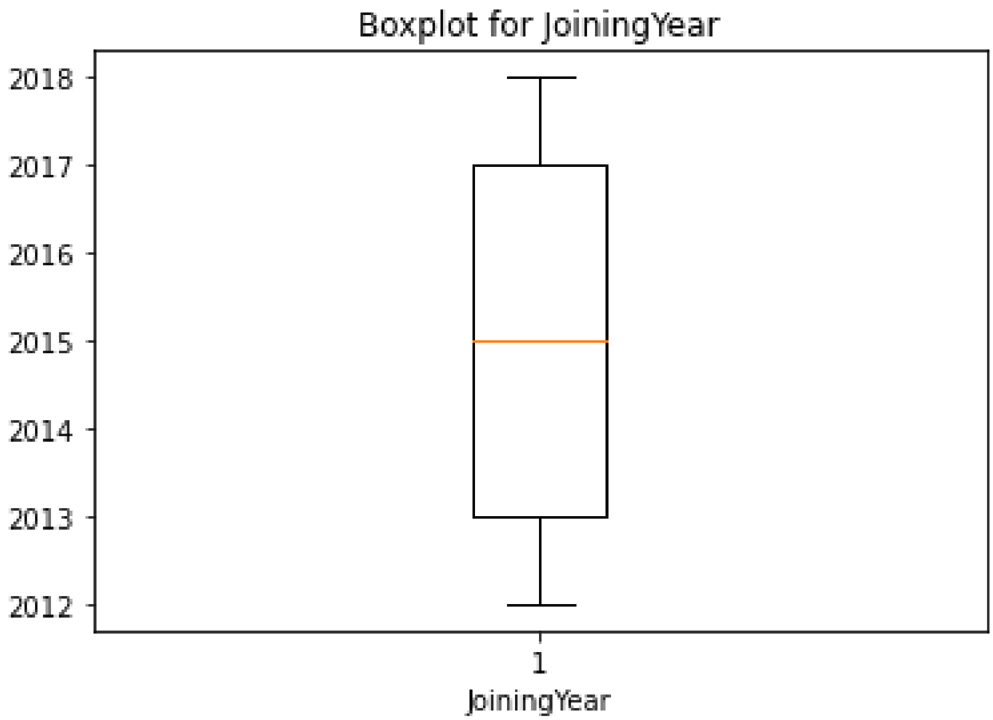
```
Value Counts
2017    662
2015    464
2013    396
2014    385
2016    310
2012    308
2018    239
Name: JoiningYear, dtype: int64

Mean
2015.090448625181

Median
2015.0

Mode
2017

Standard Deviation
1.8859431864163927
```

[16]:
```
sns.countplot(x = d['JoiningYear'], palette = "Paired")
plt.title("Number of Workers by Joining Year")
```

[16]:
```
Text(0.5, 1.0, 'Number of Workers by Joining Year')
```



[17]:
```
fig = plt.figureSize = ((5,8))
plt.boxplot(d['JoiningYear'])
plt.xlabel('JoiningYear')
plt.title('Boxplot for JoiningYear')
plt.show()
```



Duration

[18]:
```
print("Value Counts")
print(d['Duration'].value_counts())
print("\nMean")
print(d['Duration'].mean())
print("\nMedian")
print(d['Duration'].median())
print("\nMode")
print(mode(d['Duration']).mode[0])
print("\nStandard Deviation")
print(d['Duration'].std())
```

```
Value Counts
3    662
5    464
7    396
6    385
4    310
8    308
2    239
Name: Duration, dtype: int64

Mean
4.909551374819102
```

```
Median
5.0

Mode
3

Standard Deviation
1.8859431864163927
```

Visualizations are not provided because they would be the same as joiningyear.

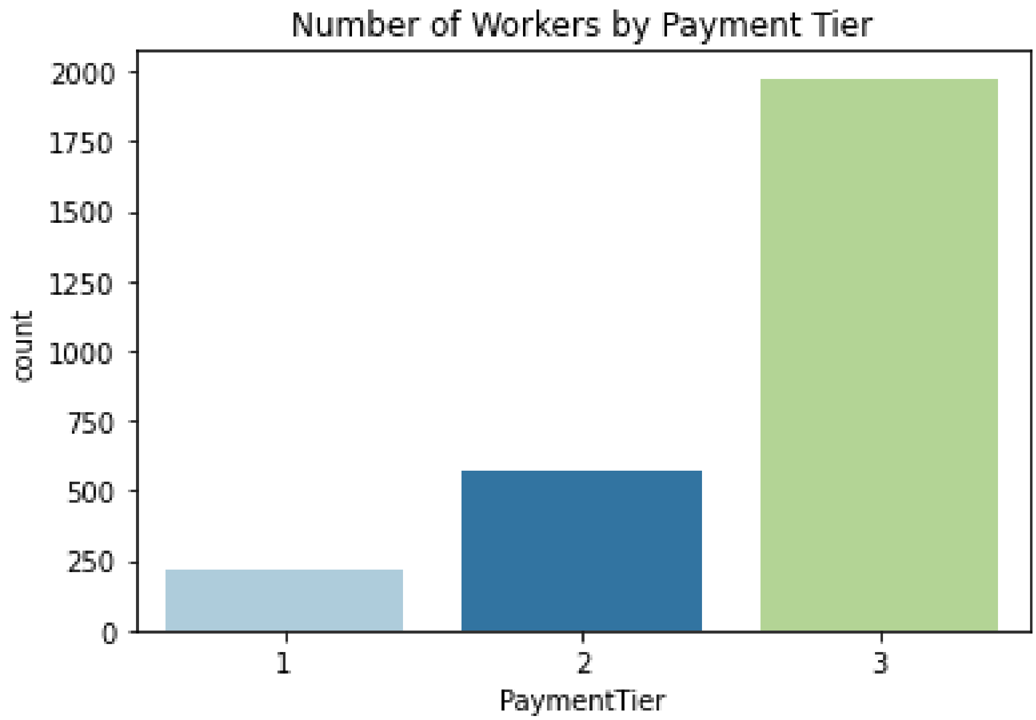
PaymentTier

[19]:
```python
print("Value Counts")
print(d['PaymentTier'].value_counts())
print("\nMean")
print(d['PaymentTier'].mean())
print("\nMedian")
print(d['PaymentTier'].median())
print("\nMode")
print(mode(d['PaymentTier']).mode[0])
print("\nStandard Deviation")
print(d['PaymentTier'].std())
```

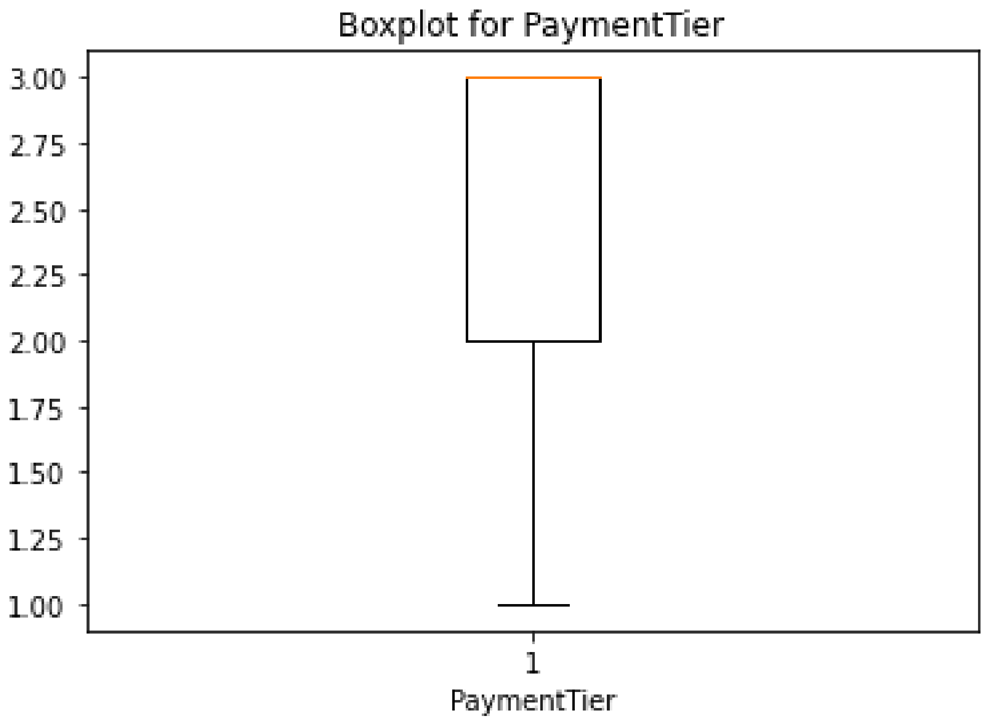
```
Value Counts
3    1976
2     570
1     218
Name: PaymentTier, dtype: int64

Mean
2.6360347322720696

Median
3.0

Mode
3

Standard Deviation
0.6240014652933755
```

[20]:
```python
sns.countplot(x = d['PaymentTier'], palette = "Paired")
plt.title("Number of Workers by Payment Tier")

#Is 3 the highest or lowest payment tier?
```

[20]:
```
Text(0.5, 1.0, 'Number of Workers by Payment Tier')
```



[21]:
```
fig = plt.figureSize = ((5,8))
plt.boxplot(d['PaymentTier'])
plt.xlabel('PaymentTier')
plt.title('Boxplot for PaymentTier')
plt.show()
```



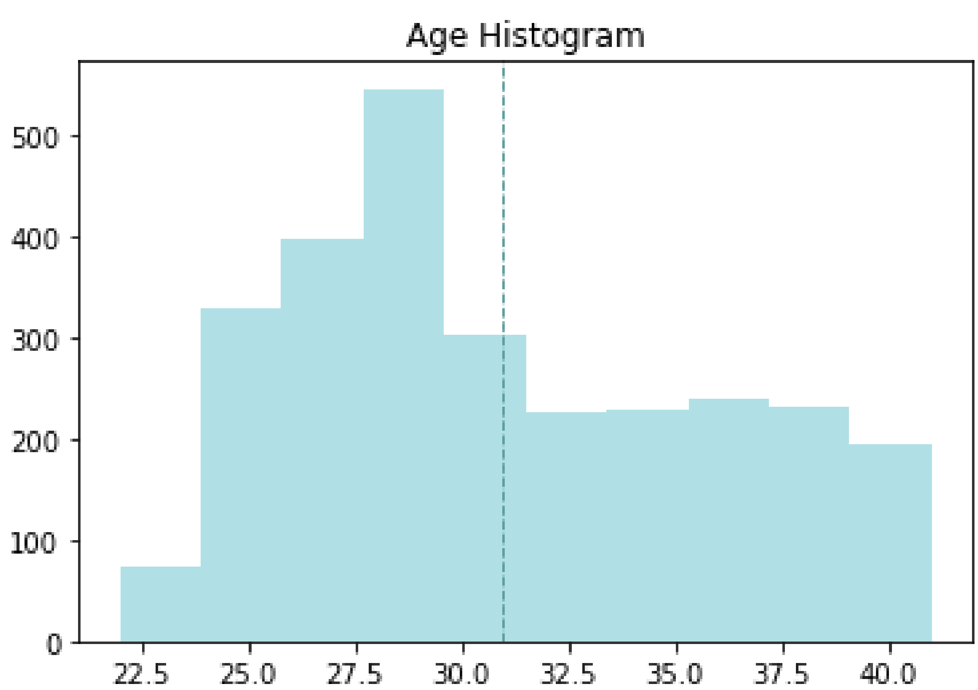
Age

[22]:
```
print("Value Counts")
print(d['Age'].value_counts())
print("\nMean")
print(d['Age'].mean())
print("\nMedian")
print(d['Age'].median())
print("\nMode")
print(mode(d['Age']).mode[0])
print("\nStandard Deviation")
print(d['Age'].std())
```

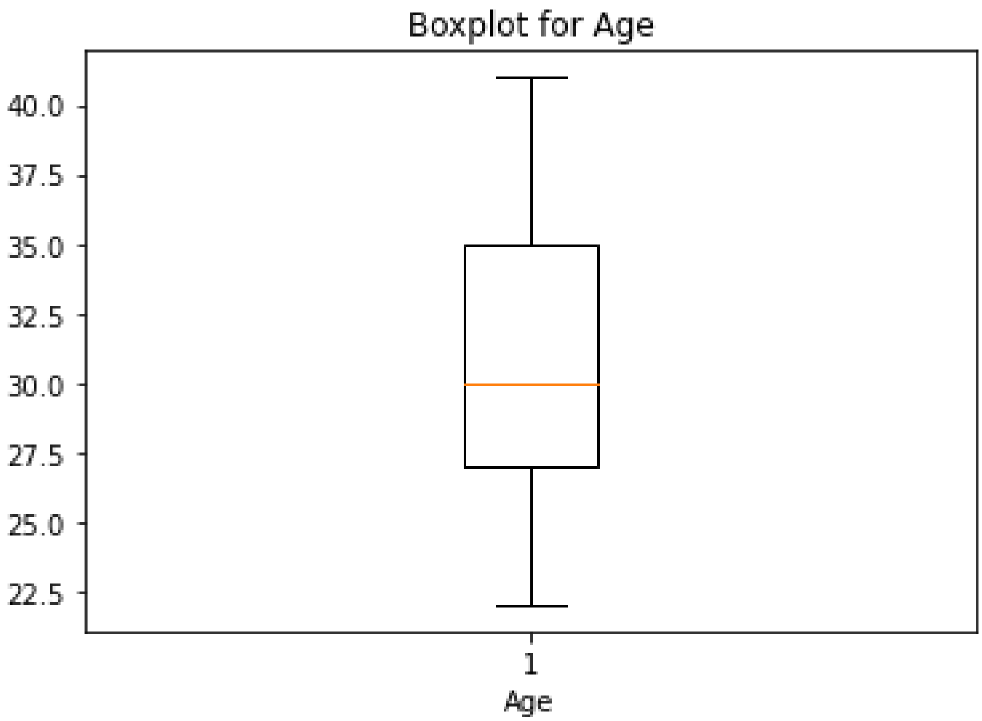
```
Value Counts
28    365
27    218
30    186
29    180
26    179
25    167
24    161
36    121
40    119
37    119
34    118
```

```
38    117
39    115
31    115
33    114
32    113
35    110
41     75
23     41
22     31
Name: Age, dtype: int64

Mean
30.952966714905934

Median
30.0

Mode
28

Standard Deviation
5.108872076631115
```

[23]:
```
plt.hist(d['Age'], color = 'powderblue')
plt.axvline(d['Age'].mean(), color='cadetblue', linestyle='dashed', linewidth=1)
plt.title("Age Histogram")
plt.show()
```



[24]:
```
fig = plt.figureSize = ((5,8))
plt.boxplot(d['Age'])
plt.xlabel('Age')
plt.title('Boxplot for Age')
plt.show()
```



ExperienceInCurrentDomain

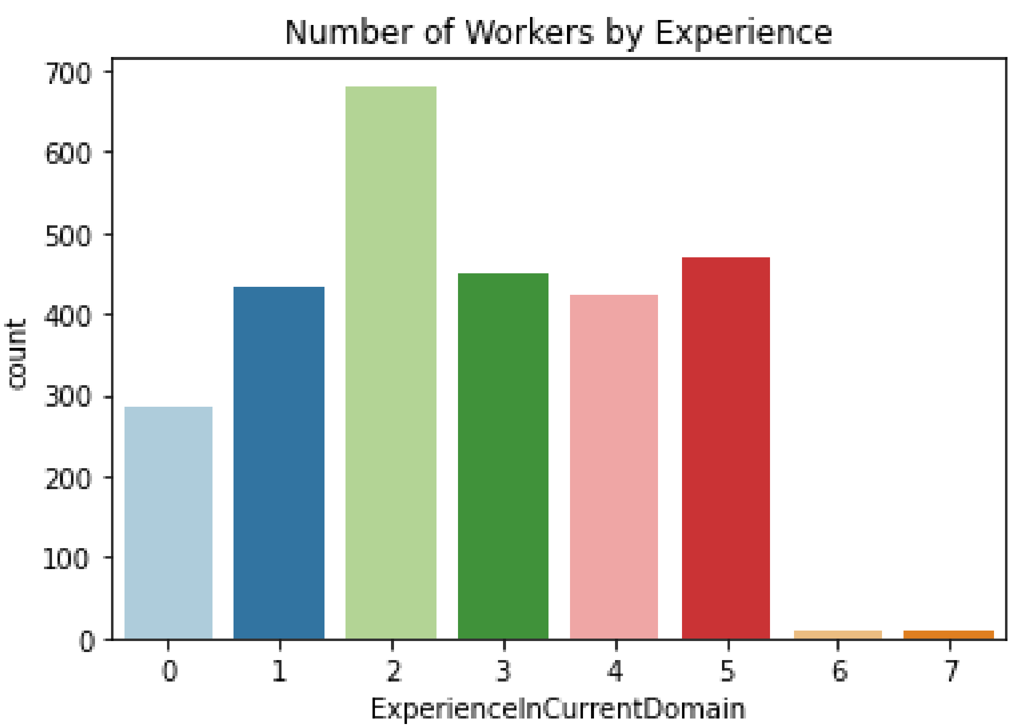
[25]:
```
print("Value Counts")
print(d['ExperienceInCurrentDomain'].value_counts())
print("\nMean")
print(d['ExperienceInCurrentDomain'].mean())
print("\nMedian")
print(d['ExperienceInCurrentDomain'].median())
print("\nMode")
print(mode(d['ExperienceInCurrentDomain']).mode[0])
print("\nStandard Deviation")
print(d['ExperienceInCurrentDomain'].std())
```

```
Value Counts
2    681
5    470
```

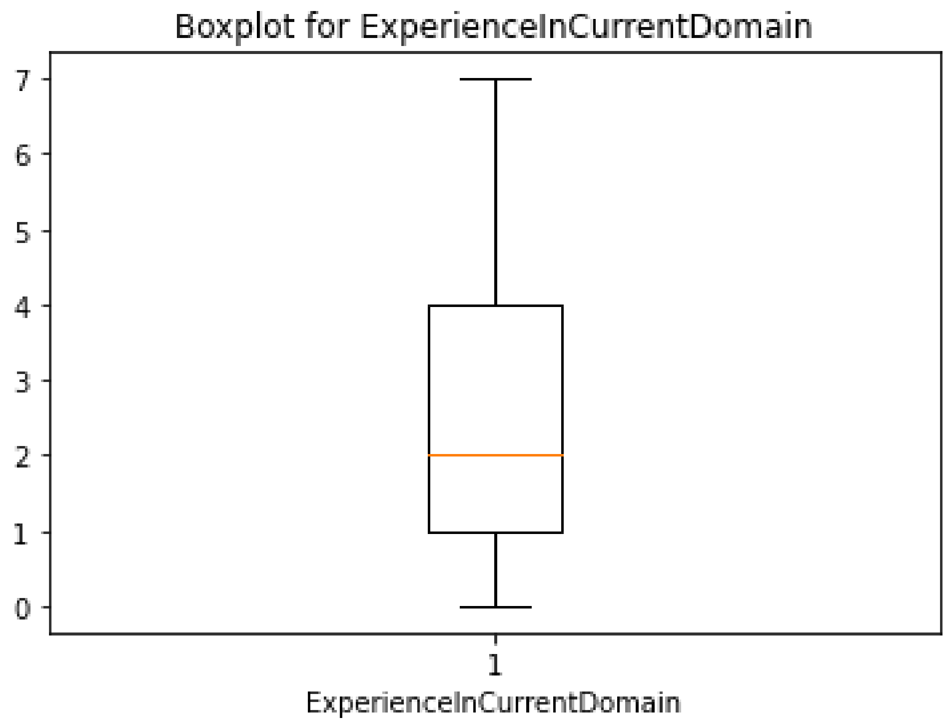
```
3    451
1    433
4    425
0    287
7      9
6      8
Name: ExperienceInCurrentDomain, dtype: int64

Mean
2.644356005788712

Median
2.0

Mode
2

Standard Deviation
1.6106101731390896
```

[26]:
```
sns.countplot(x = d['ExperienceInCurrentDomain'], palette = "Paired")
plt.title("Number of Workers by Experience")
```

[26]:
```
Text(0.5, 1.0, 'Number of Workers by Experience')
```



[27]:
```
fig = plt.figureSize = ((5,8))
plt.boxplot(d['ExperienceInCurrentDomain'])
plt.xlabel('ExperienceInCurrentDomain')
plt.title('Boxplot for ExperienceInCurrentDomain')
plt.show()
```



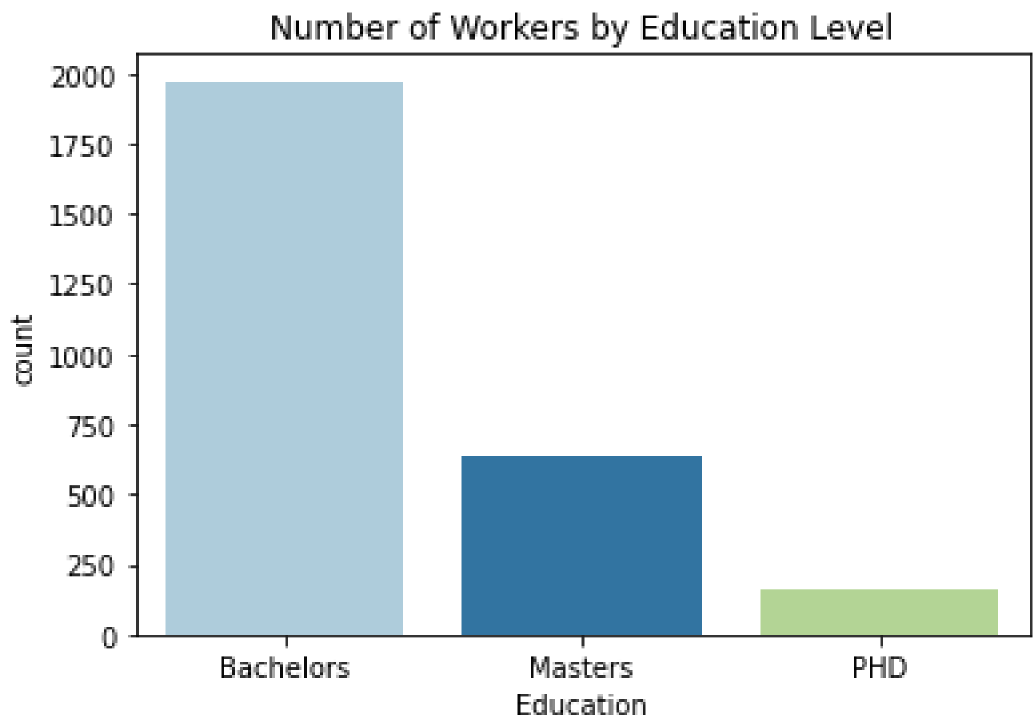
Education

[28]:
```
print("Value Counts")
print(d['Education'].value_counts())
print("\nMode")
print(mode(d['Education']).mode[0])
```

```
Value Counts
Bachelors    1971
Masters       637
PHD           156
Name: Education, dtype: int64

Mode
Bachelors
```

[29]:
```
sns.countplot(x = d['Education'], palette = "Paired")
plt.title("Number of Workers by Education Level")
```

[29]: Text(0.5, 1.0, 'Number of Workers by Education Level')



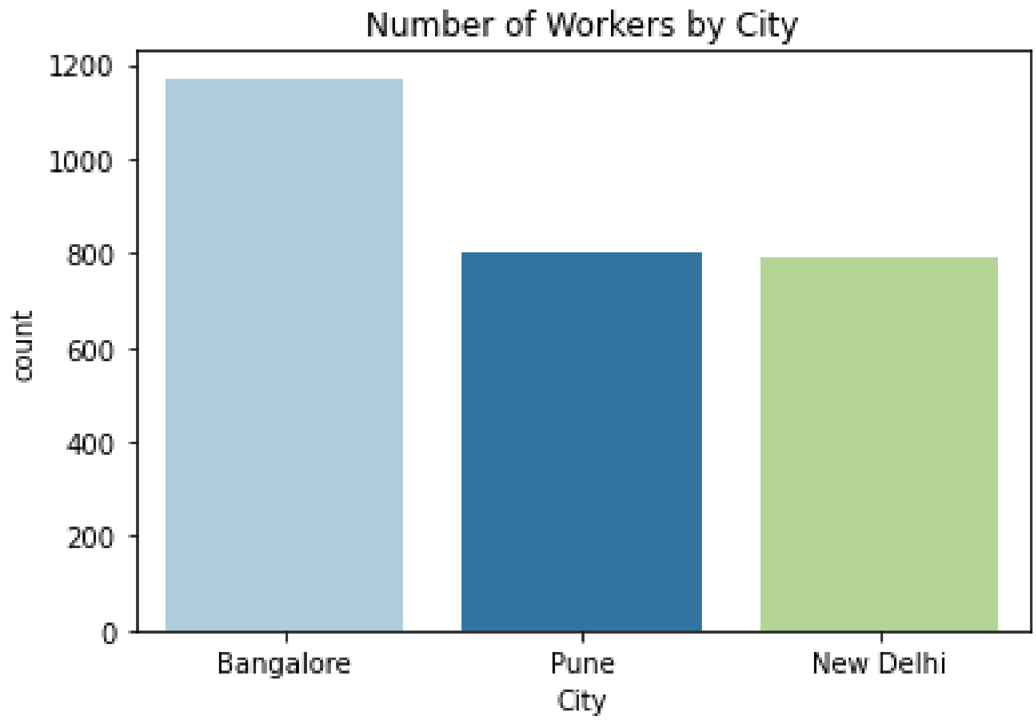
City

[30]:
```python
print("Value Counts")
print(d['City'].value_counts())
print("\nMode")
print(mode(d['City']).mode[0])
```

```
Value Counts
Bangalore    1171
Pune          801
New Delhi     792
Name: City, dtype: int64

Mode
Bangalore
```

[31]:
```python
sns.countplot(x = d['City'], palette = "Paired")
plt.title("Number of Workers by City")
```

[31]: Text(0.5, 1.0, 'Number of Workers by City')



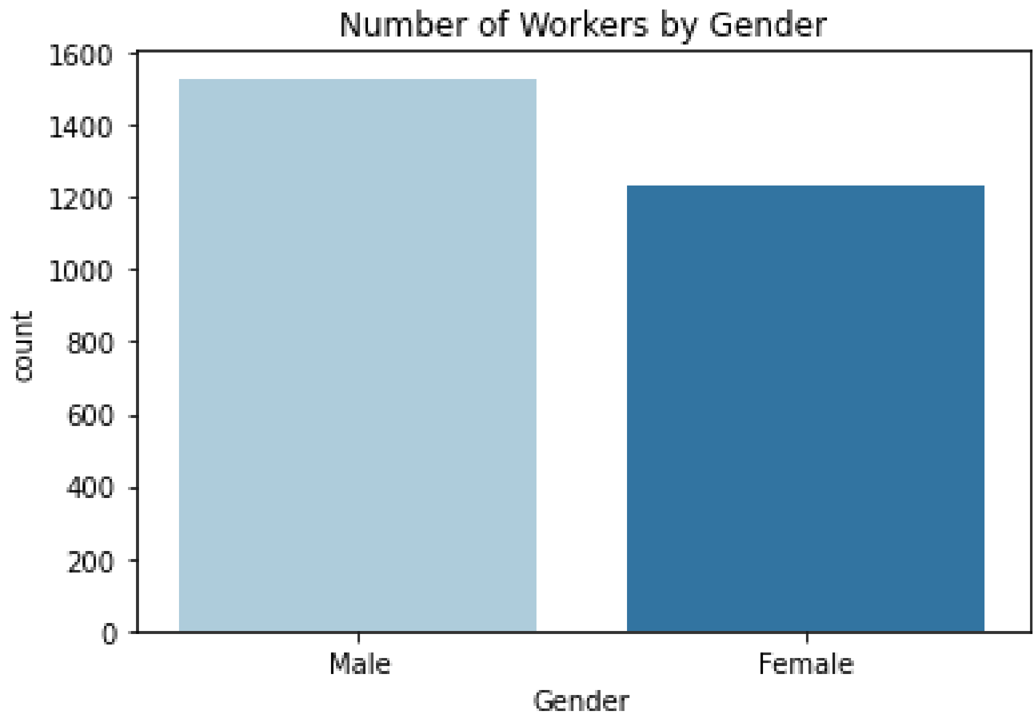
Gender

[32]:
```
print("Value Counts")
print(d['Gender'].value_counts())
print("\nMode")
print(mode(d['Gender']).mode[0])
```

```
Value Counts
Male      1529
Female    1235
Name: Gender, dtype: int64

Mode
Male
```

[33]:
```
sns.countplot(x = d['Gender'], palette = "Paired")
plt.title("Number of Workers by Gender")
```

[33]: `Text(0.5, 1.0, 'Number of Workers by Gender')`



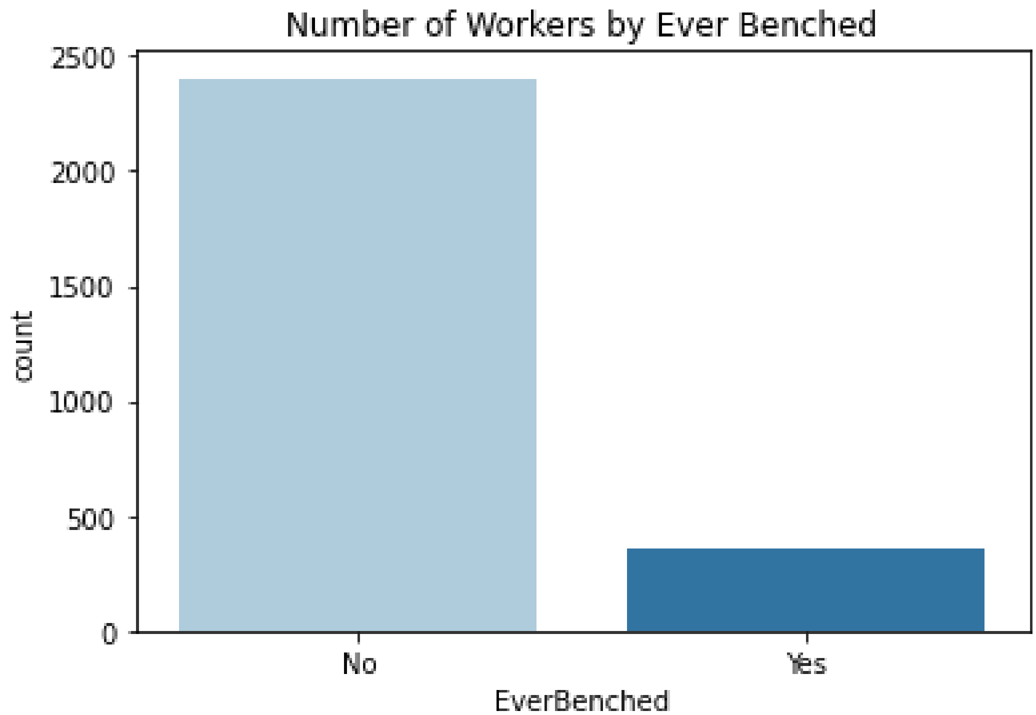
EverBenched

[34]:
```
print("Value Counts")
print(d['EverBenched'].value_counts())
print("\nMode")
print(mode(d['EverBenched']).mode[0])
```

```
Value Counts
No     2403
Yes     361
Name: EverBenched, dtype: int64

Mode
No
```

[35]:
```
sns.countplot(x = d['EverBenched'], palette = "Paired")
plt.title("Number of Workers by Ever Benched")
```

[35]: `Text(0.5, 1.0, 'Number of Workers by Ever Benched')`



LeaveorNot

[36]:
```
print("Value Counts")
print(d['LeaveOrNot'].value_counts())
print("\nMode")
print(mode(d['LeaveOrNot']).mode[0])
```

```
Value Counts
0    1676
1    1088
Name: LeaveOrNot, dtype: int64

Mode
0
```

[37]:
```
sns.countplot(x = d['LeaveOrNot'], palette = "Paired")
plt.title("Number of Workers by LeaveOrNot")
```

[37]: Text(0.5, 1.0, 'Number of Workers by LeaveOrNot')

Overall: Quantitative

[38]:
```
#d.hist(bins=30,figsize=(15,10))
#plt.show()
```

1.2.2 Diagonse Correlation

Dependent variable is LeaveOrNot

[39]:
```
d.groupby(by = "LeaveOrNot").mean()
```

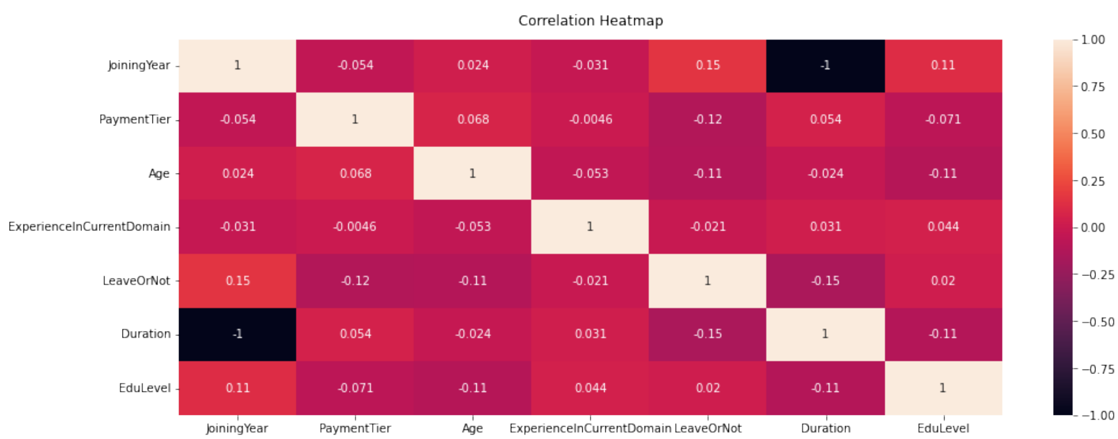
[39]:
```
            JoiningYear  PaymentTier        Age  ExperienceInCurrentDomain  \
LeaveOrNot
0           2014.861575     2.696301  31.426014                   2.671838
1           2015.443015     2.543199  30.224265                   2.602022

            Duration  EduLevel
LeaveOrNot
0           5.138425  1.334129
1           4.556985  1.357537
```

[40]:
```
d.corr()
```

[40]:
```
                           JoiningYear  PaymentTier       Age  \
JoiningYear                   1.000000    -0.053823  0.024445
```

```
PaymentTier                -0.053823     1.000000  0.067514
Age                         0.024445     0.067514  1.000000
ExperienceInCurrentDomain  -0.031228    -0.004602 -0.053276
LeaveOrNot                  0.150650    -0.119891 -0.114943
Duration                   -1.000000     0.053823 -0.024445
EduLevel                    0.113858    -0.071380 -0.107324

                           ExperienceInCurrentDomain  LeaveOrNot  Duration  \
JoiningYear                                -0.031228    0.150650 -1.000000
PaymentTier                                -0.004602   -0.119891  0.053823
Age                                        -0.053276   -0.114943 -0.024445
ExperienceInCurrentDomain                   1.000000   -0.021181  0.031228
LeaveOrNot                                 -0.021181    1.000000 -0.150650
Duration                                    0.031228   -0.150650  1.000000
EduLevel                                    0.043842    0.019661 -0.113858

                           EduLevel
JoiningYear                0.113858
PaymentTier               -0.071380
Age                       -0.107324
ExperienceInCurrentDomain  0.043842
LeaveOrNot                 0.019661
Duration                  -0.113858
EduLevel                   1.000000
```

[41]:
```
plt.figure(figsize=(16, 6))
heatmap = sns.heatmap(d.corr(), vmin=-1, vmax=1, annot=True)
heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':12}, pad=12);

#Nothing with a correlation of more than .5 generally or with our dependent␣
↪variable
#Strongest relationship is between JoiningYear/Duration and LeaveOrNot
```



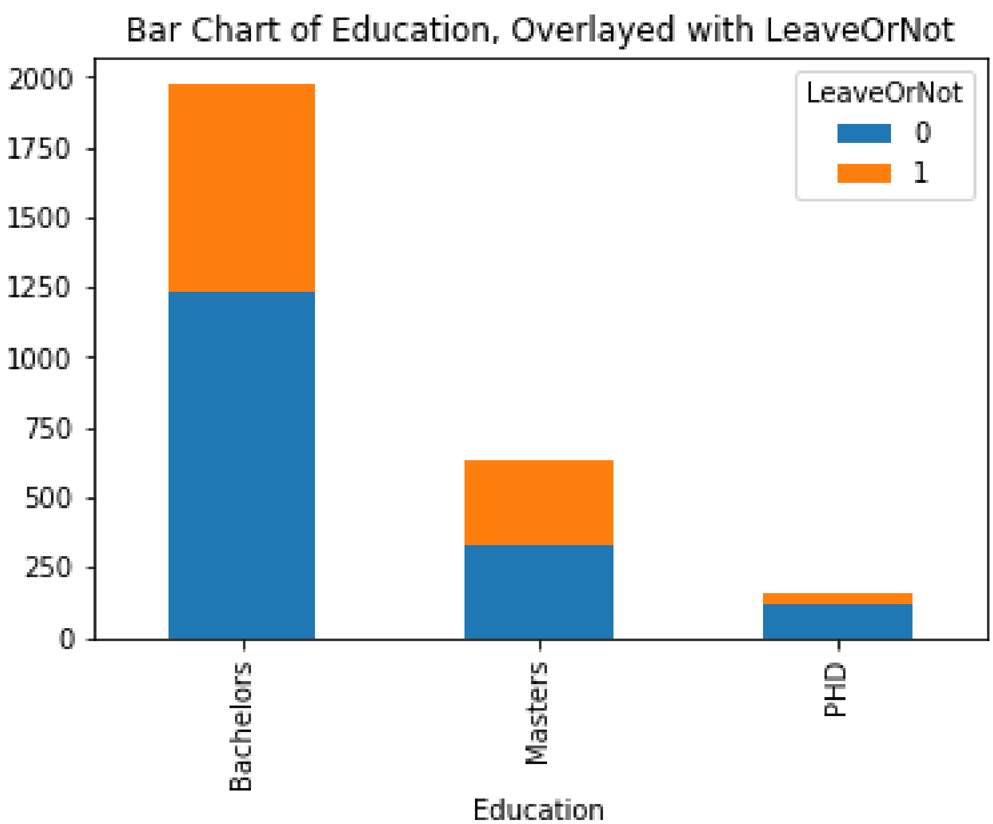
Education and LeaveOrNot

[42]:
```
eduleave = pd.crosstab(d['Education'], d['LeaveOrNot'])
eduleave
```

[42]:
```
LeaveOrNot     0    1
Education
Bachelors   1232  739
Masters      328  309
PHD          116   40
```

[43]:
```
eduleave.plot(kind='bar', stacked = True, title = 'Bar Chart of Education,␣
↪Overlayed with LeaveOrNot')
```

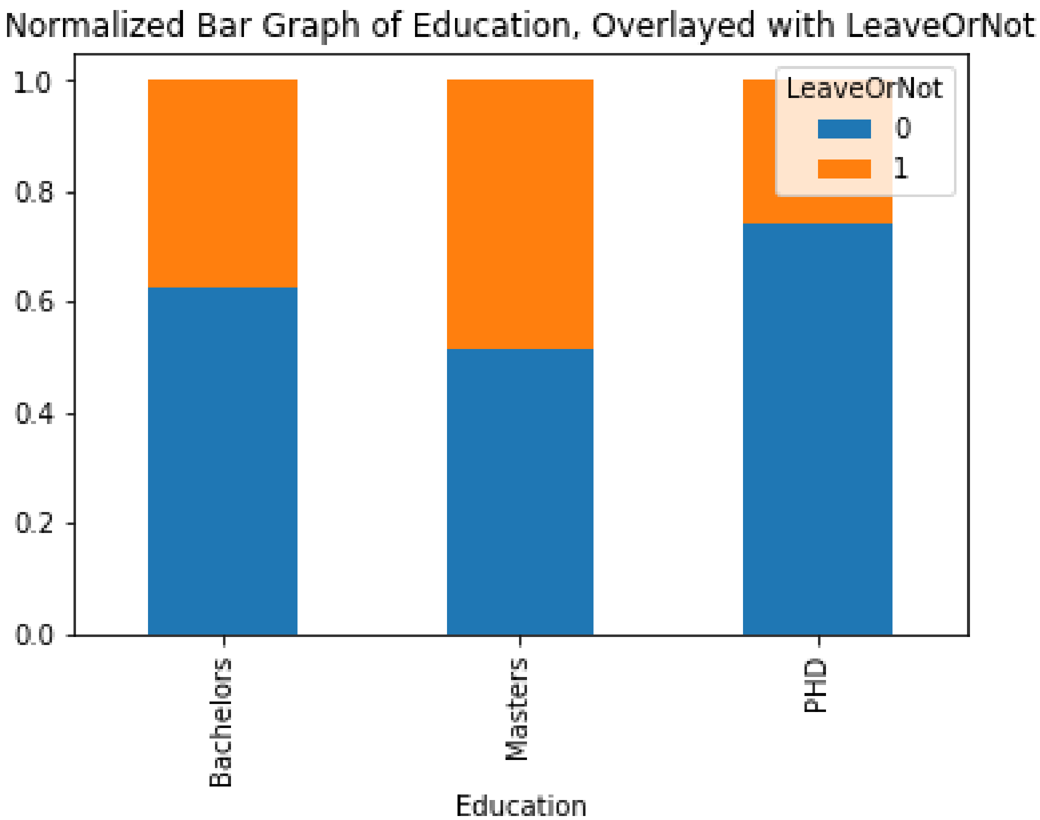
[43]:
```
<AxesSubplot:title={'center':'Bar Chart of Education, Overlayed with
LeaveOrNot'}, xlabel='Education'>
```



[44]:
```
eduleave_n = eduleave.div(eduleave.sum(1), axis = 0)
```

[45]:
```
eduleave_n.plot(kind='bar', stacked=True, title = 'Normalized Bar Graph of␣
↪Education, Overlayed with LeaveOrNot')
```

[45]: <AxesSubplot:title={'center':'Normalized Bar Graph of Education, Overlayed with LeaveOrNot'}, xlabel='Education'>



[46]:
```
round(eduleave.div(eduleave.sum(0), axis = 1)*100, 1)
```

[46]:
```
LeaveOrNot     0     1
Education
Bachelors   73.5  67.9
Masters     19.6  28.4
PHD          6.9   3.7
```

INSIGHT

Most of our folks have Bachelor's. People with Master's are more liekly to leave.

We can include in model, seems that this has an effect.

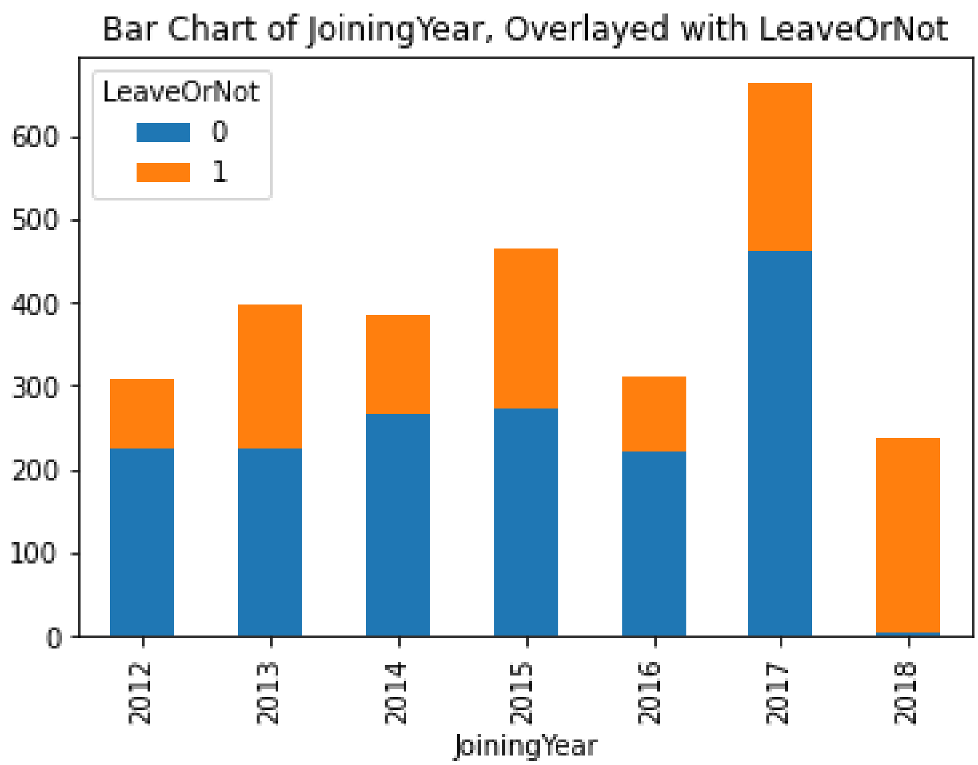
JoiningYear/Duration and LeaveOrNot

[47]:
```
yearleave = pd.crosstab(d['JoiningYear'], d['LeaveOrNot'])
yearleave
```

[47]:
```
LeaveOrNot      0    1
JoiningYear
2012          225   83
2013          225  171
2014          266  119
2015          272  192
2016          222   88
2017          461  201
2018            5  234
```

[48]:
```
yearleave.plot(kind='bar', stacked = True, title = 'Bar Chart of JoiningYear,␣
↪Overlayed with LeaveOrNot')
```

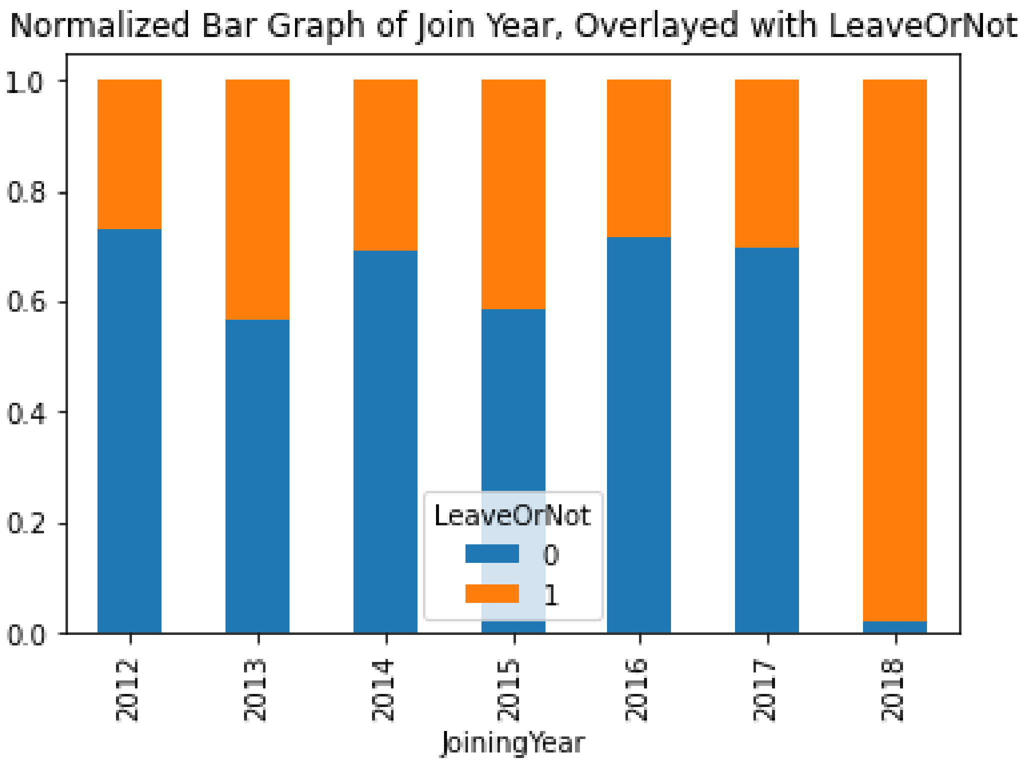
[48]:
```
<AxesSubplot:title={'center':'Bar Chart of JoiningYear, Overlayed with
LeaveOrNot'}, xlabel='JoiningYear'>
```



[49]:
```
yearleave_n = yearleave.div(yearleave.sum(1), axis = 0)
```

[135]:
```
yearleave_n.plot(kind='bar', stacked=True, title = 'Normalized Bar Graph of␣
↪Join Year, Overlayed with LeaveOrNot')
```

[135]: <AxesSubplot:title={'center':'Normalized Bar Graph of Join Year, Overlayed with LeaveOrNot'}, xlabel='JoiningYear'>




```
[51]: round(yearleave.div(yearleave.sum(0), axis = 1)*100, 1)
```

```
[51]: LeaveOrNot      0     1
      JoiningYear
      2012          13.4   7.6
      2013          13.4  15.7
      2014          15.9  10.9
      2015          16.2  17.6
      2016          13.2   8.1
      2017          27.5  18.5
      2018           0.3  21.5
```

INSIGHT This has a correlation of .2 and it seems to have an erratic relationship, I think it's being pulled by the 2018 outliers. I would actually recommend we drop.

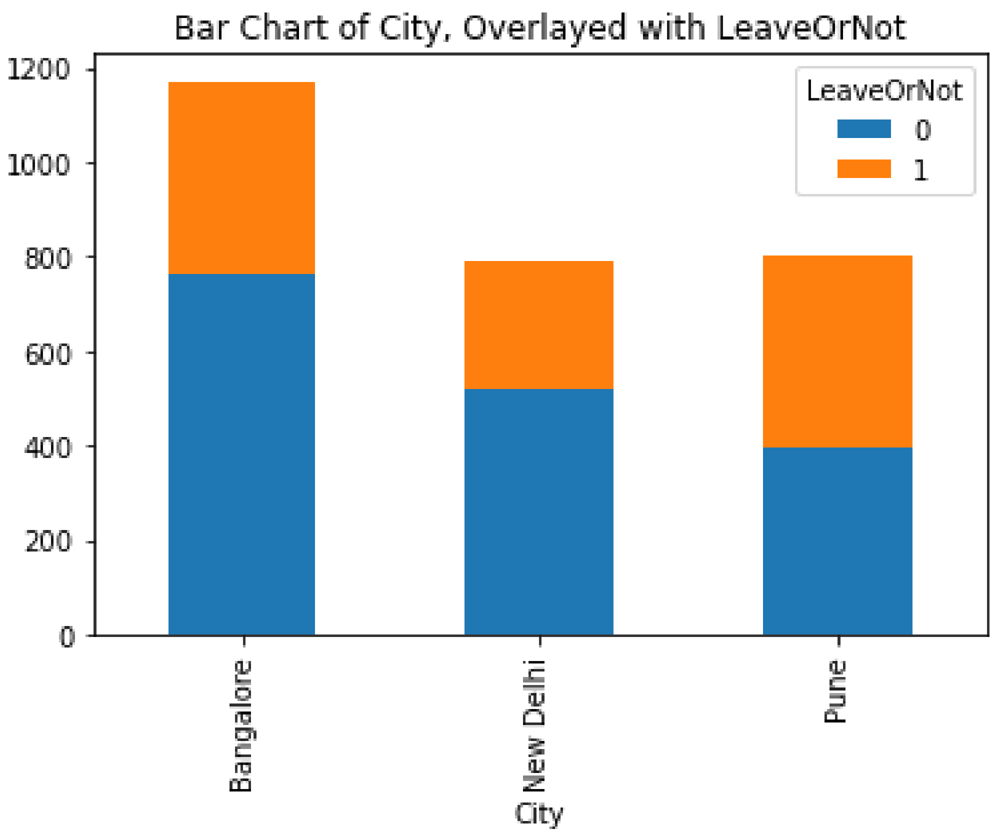
City and LeaveOrNot

```
[52]: cityleave = pd.crosstab(d['City'], d['LeaveOrNot'])
      cityleave
```

[52]:
```
LeaveOrNot    0    1
City
Bangalore   761  410
New Delhi   522  270
Pune        393  408
```

[53]:
```
cityleave.plot(kind='bar', stacked = True, title = 'Bar Chart of City,␣
↪Overlayed with LeaveOrNot')
```

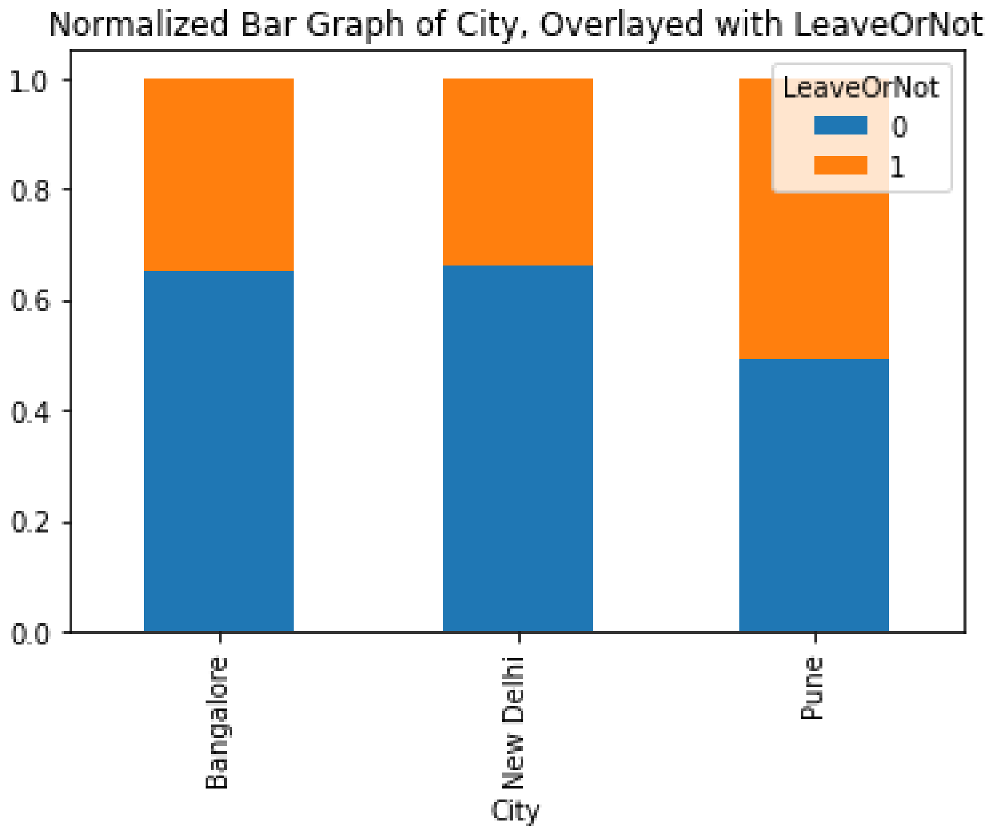
[53]:
```
<AxesSubplot:title={'center':'Bar Chart of City, Overlayed with LeaveOrNot'},
xlabel='City'>
```



[54]:
```
cityleave_n = cityleave.div(cityleave.sum(1), axis = 0)
```

[55]:
```
cityleave_n.plot(kind='bar', stacked=True, title = 'Normalized Bar Graph of␣
↪City, Overlayed with LeaveOrNot')
```

[55]:
```
<AxesSubplot:title={'center':'Normalized Bar Graph of City, Overlayed with
LeaveOrNot'}, xlabel='City'>
```



[56]:
```
round(cityleave.div(cityleave.sum(0), axis = 1)*100, 1)
```

[56]:
```
LeaveOrNot     0     1
City
Bangalore   45.4  37.7
New Delhi   31.1  24.8
Pune        23.4  37.5
```

INSIGHT This does seem to have an effect, let's include.

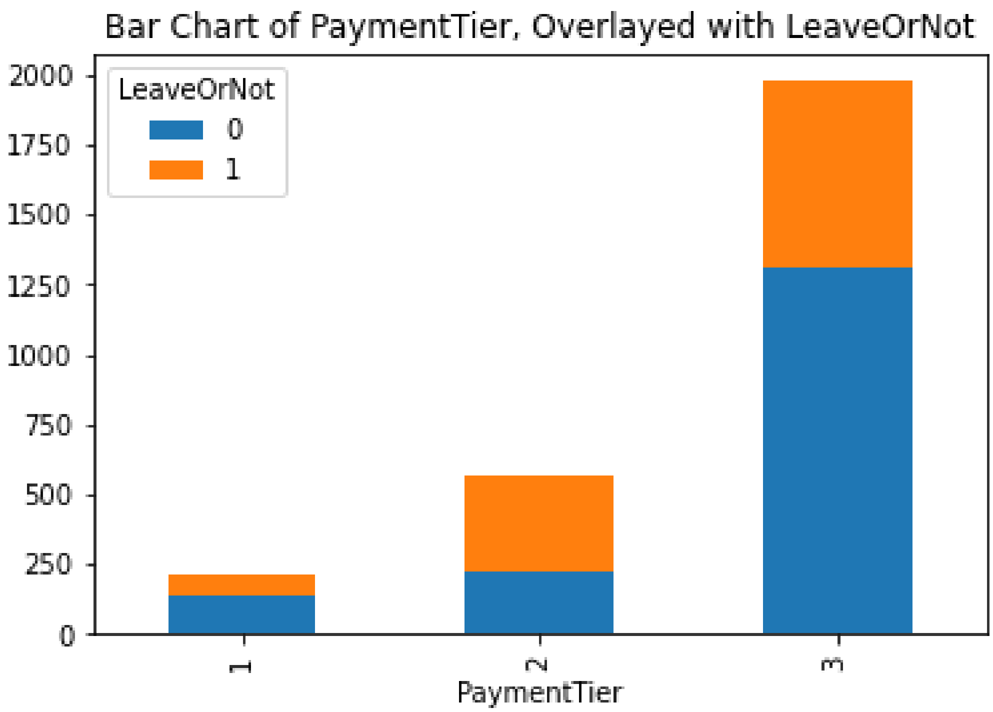
PaymentTier and LeaveOrNot

[57]:
```
payleave = pd.crosstab(d['PaymentTier'], d['LeaveOrNot'])
payleave
```

[57]:
```
LeaveOrNot      0    1
PaymentTier
1             141   77
2             227  343
3            1308  668
```

[58]:
```
payleave.plot(kind='bar', stacked = True, title = 'Bar Chart of PaymentTier,␣
↪Overlayed with LeaveOrNot')
```

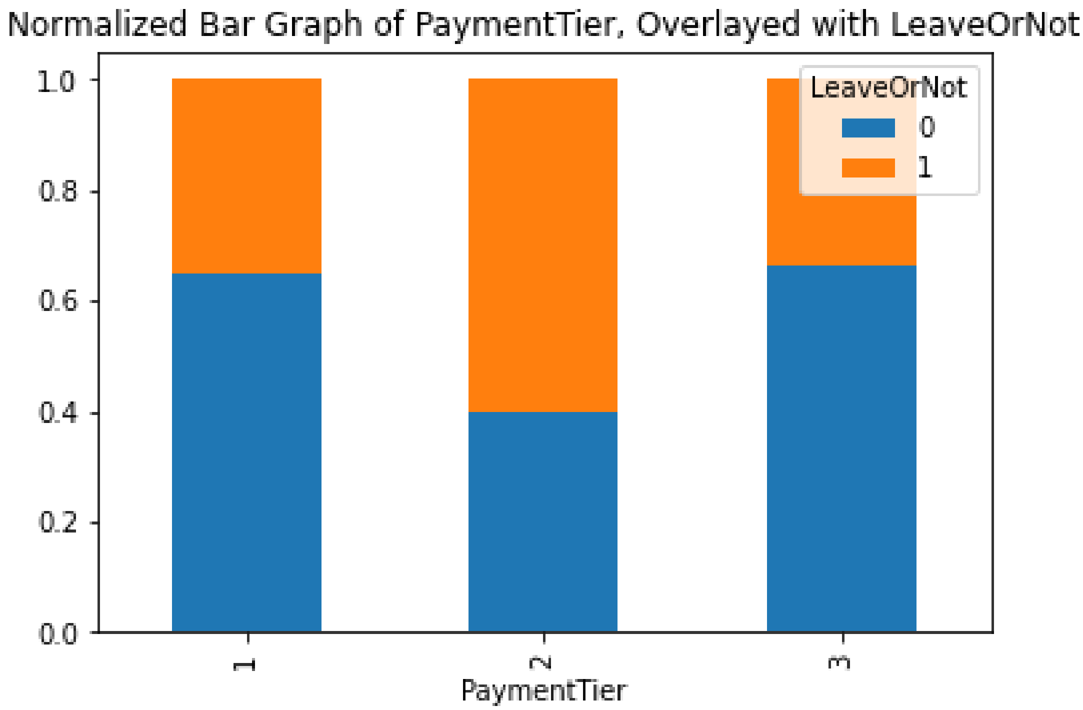
[58]:
```
<AxesSubplot:title={'center':'Bar Chart of PaymentTier, Overlayed with
LeaveOrNot'}, xlabel='PaymentTier'>
```



[59]:
```
payleave_n = payleave.div(payleave.sum(1), axis = 0)
```

[60]:
```
payleave_n.plot(kind='bar', stacked=True, title = 'Normalized Bar Graph of␣
↪PaymentTier, Overlayed with LeaveOrNot')
```

[60]:
```
<AxesSubplot:title={'center':'Normalized Bar Graph of PaymentTier, Overlayed
with LeaveOrNot'}, xlabel='PaymentTier'>
```



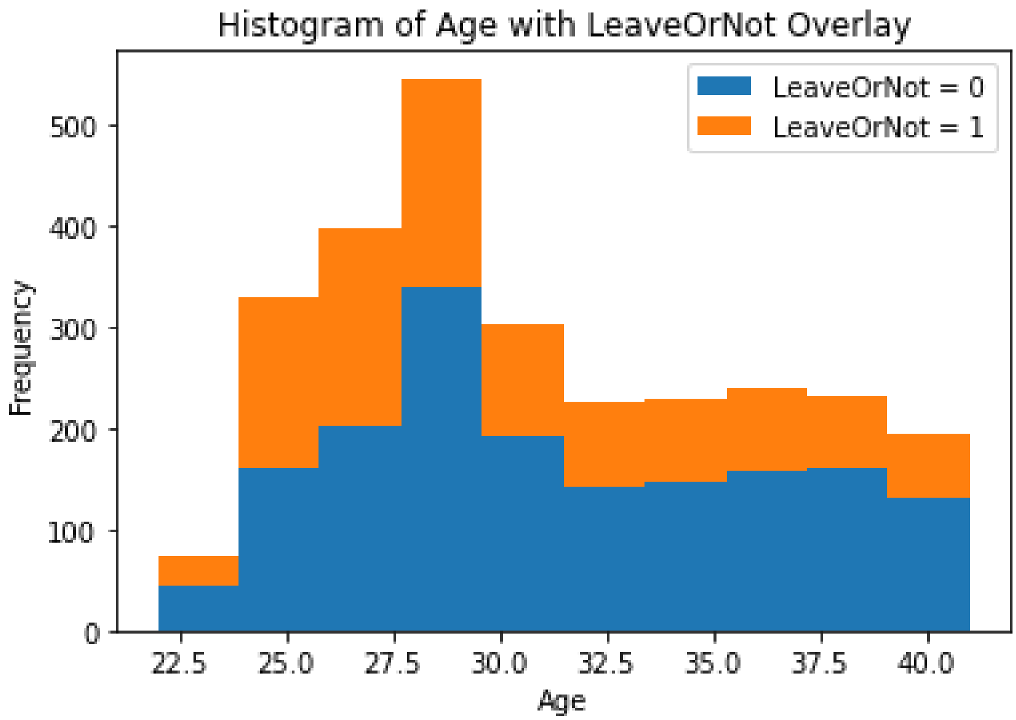
[61]:
```
round(payleave.div(payleave.sum(0), axis = 1)*100, 1)
```

[61]:
```
LeaveOrNot      0     1
PaymentTier
1             8.4   7.1
2            13.5  31.5
3            78.0  61.4
```

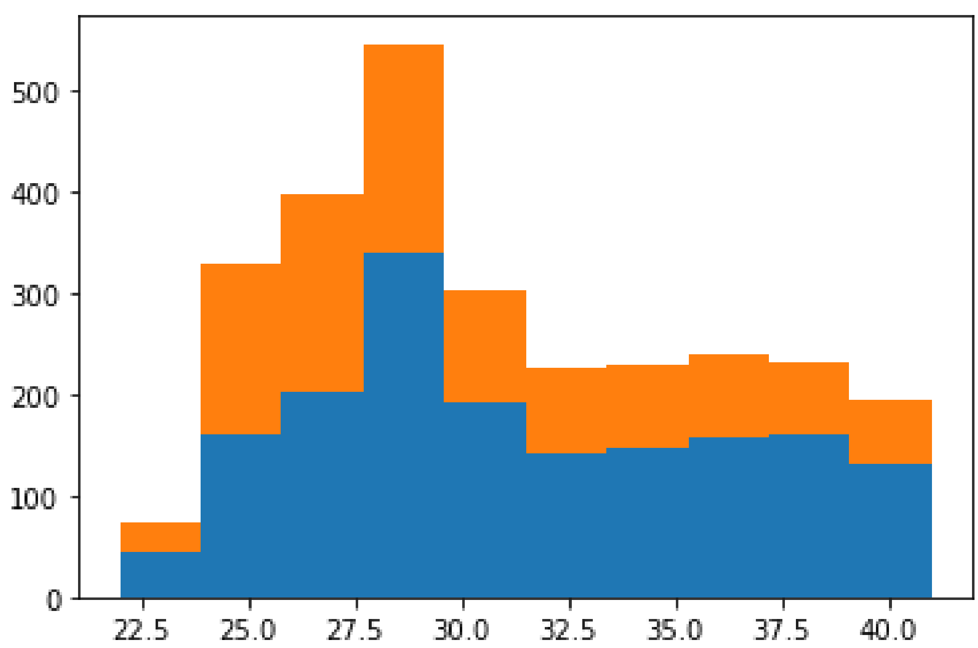
INSIGHT This does seem to have an effect, let's include.

Age and LeaveOrNot

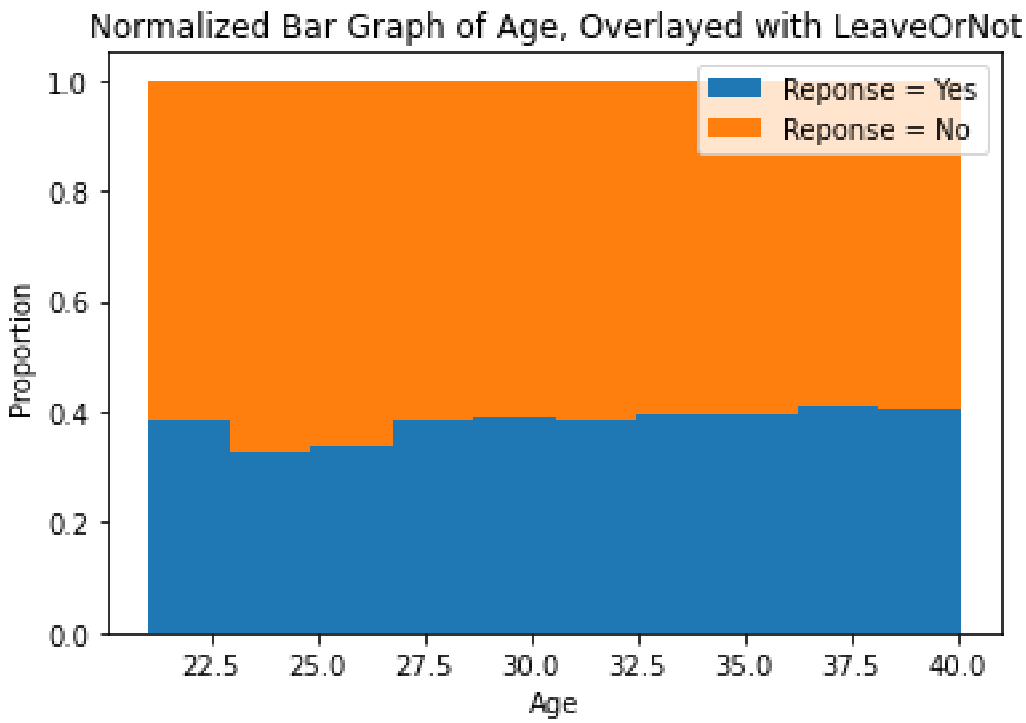
[62]:
```
d_age_0 = d[d.LeaveOrNot == 0]['Age']
d_age_1 = d[d.LeaveOrNot == 1]['Age']
plt.hist([d_age_0, d_age_1], bins = 10, stacked = True)
plt.legend(['LeaveOrNot = 0','LeaveOrNot = 1'])
plt.title('Histogram of Age with LeaveOrNot Overlay')
plt.xlabel('Age'); plt.ylabel('Frequency'); plt.show()
```



[63]:
```
(n, bins, patches) = plt.hist([d_age_0, d_age_1], bins = 10, stacked = True)
```



[134]:
```
n_table = np.column_stack((n[0], n[1]))
n_norm = n_table / n_table.sum(axis=1) [:, None]
ourbins = np.column_stack((bins[0:10], bins[1:11]))
p1 = plt.bar(x=ourbins[:,0], height = n_norm[:,0], width = ourbins[:,␣
↪1]-ourbins[:,0])
p2 = plt.bar(x=ourbins[:,0], height = n_norm[:,1], width = ourbins[:,␣
↪1]-ourbins[:,0], bottom = n_norm[:,0])
plt.legend(['Reponse = Yes','Reponse = No'])
plt.title('Normalized Bar Graph of Age, Overlayed with LeaveOrNot')
plt.xlabel('Age'); plt.ylabel('Proportion'); plt.show()
```



INSIGHT No affect. Do not include.

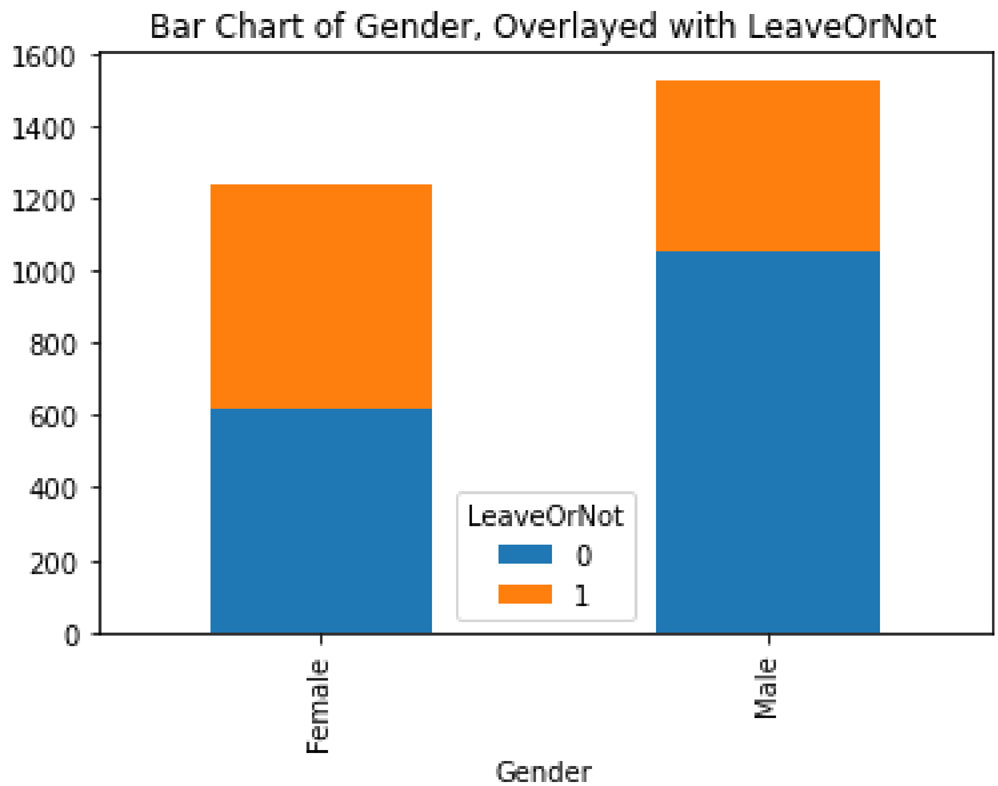
Gender and LeaveOrNot

[65]:
```
genderleave = pd.crosstab(d['Gender'], d['LeaveOrNot'])
genderleave
```

[65]:
```
LeaveOrNot     0    1
Gender
Female       621  614
Male        1055  474
```

[66]:
```
genderleave.plot(kind='bar', stacked = True, title = 'Bar Chart of Gender,␣
↪Overlayed with LeaveOrNot')
```

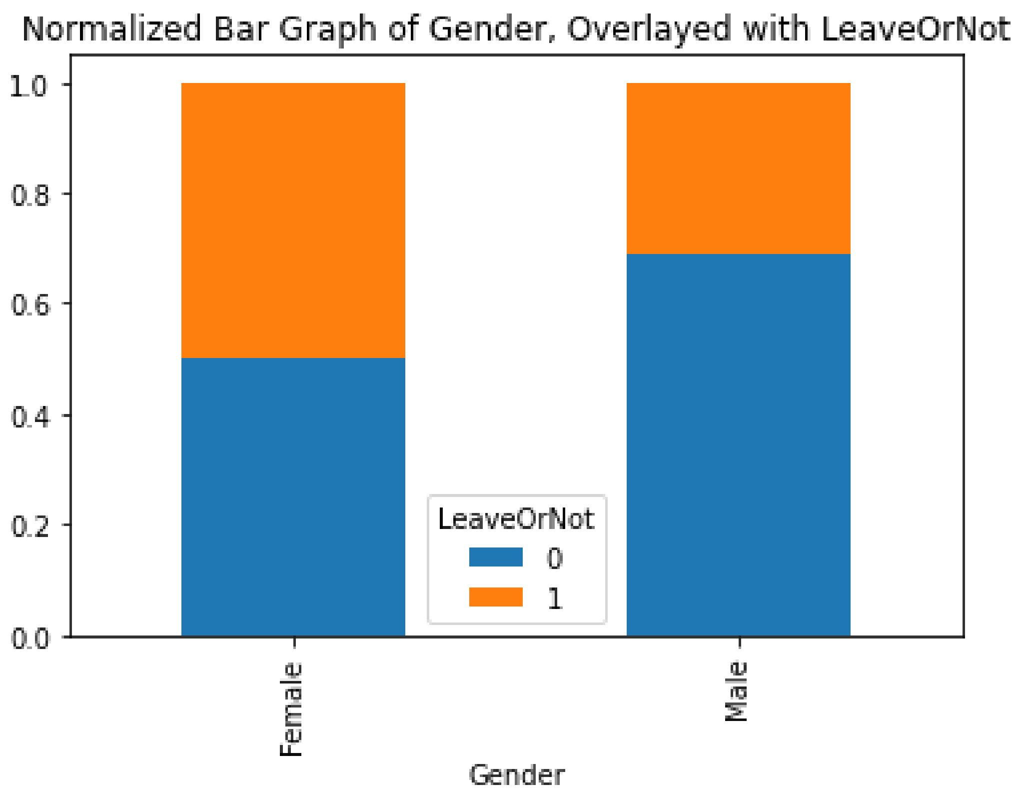
[66]: <AxesSubplot:title={'center':'Bar Chart of Gender, Overlayed with LeaveOrNot'}, xlabel='Gender'>



[67]:
```
genderleave_n = genderleave.div(genderleave.sum(1), axis = 0)
```

[68]:
```
genderleave_n.plot(kind='bar', stacked=True, title = 'Normalized Bar Graph of␣
↪Gender, Overlayed with LeaveOrNot')
```

[68]: <AxesSubplot:title={'center':'Normalized Bar Graph of Gender, Overlayed with LeaveOrNot'}, xlabel='Gender'>



[69]:
```
round(genderleave.div(genderleave.sum(0), axis = 1)*100, 1)
```

[69]:
```
LeaveOrNot      0     1
Gender
Female       37.1  56.4
Male         62.9  43.6
```

INSIGHT This does seem to have an effect, let's include.

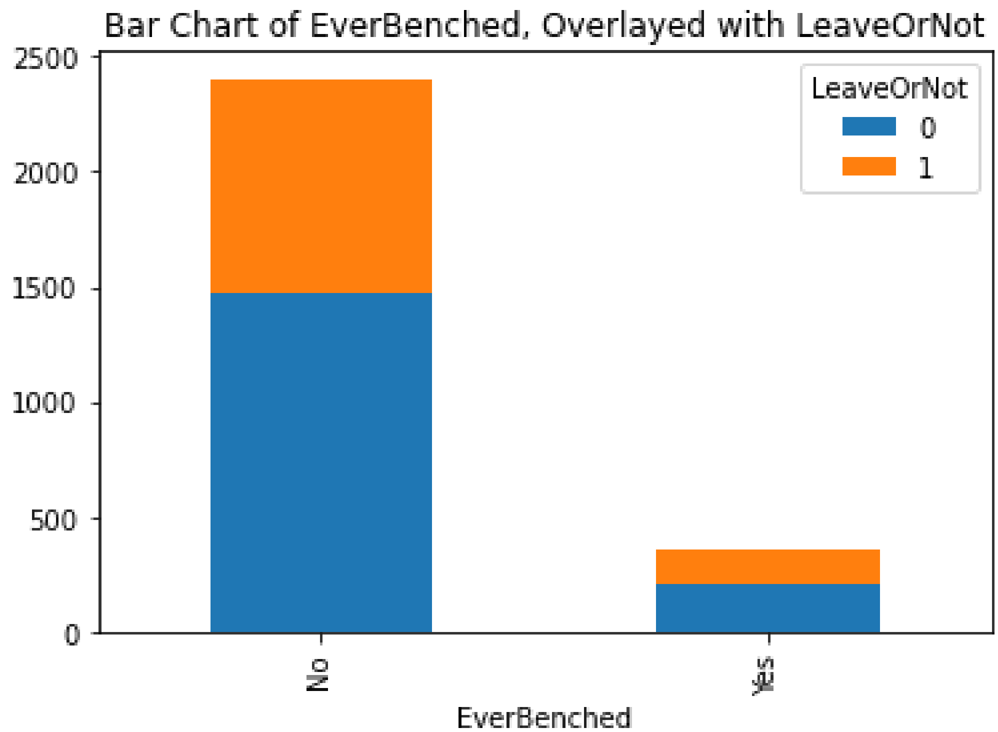
EverBenched and LeaveOrNot

[70]:
```
benchleave = pd.crosstab(d['EverBenched'], d['LeaveOrNot'])
benchleave
```

[70]:
```
LeaveOrNot      0    1
EverBenched
No           1474  929
Yes           202  159
```

[71]:
```
benchleave.plot(kind='bar', stacked = True, title = 'Bar Chart of EverBenched,␣
 ↪Overlayed with LeaveOrNot')
```

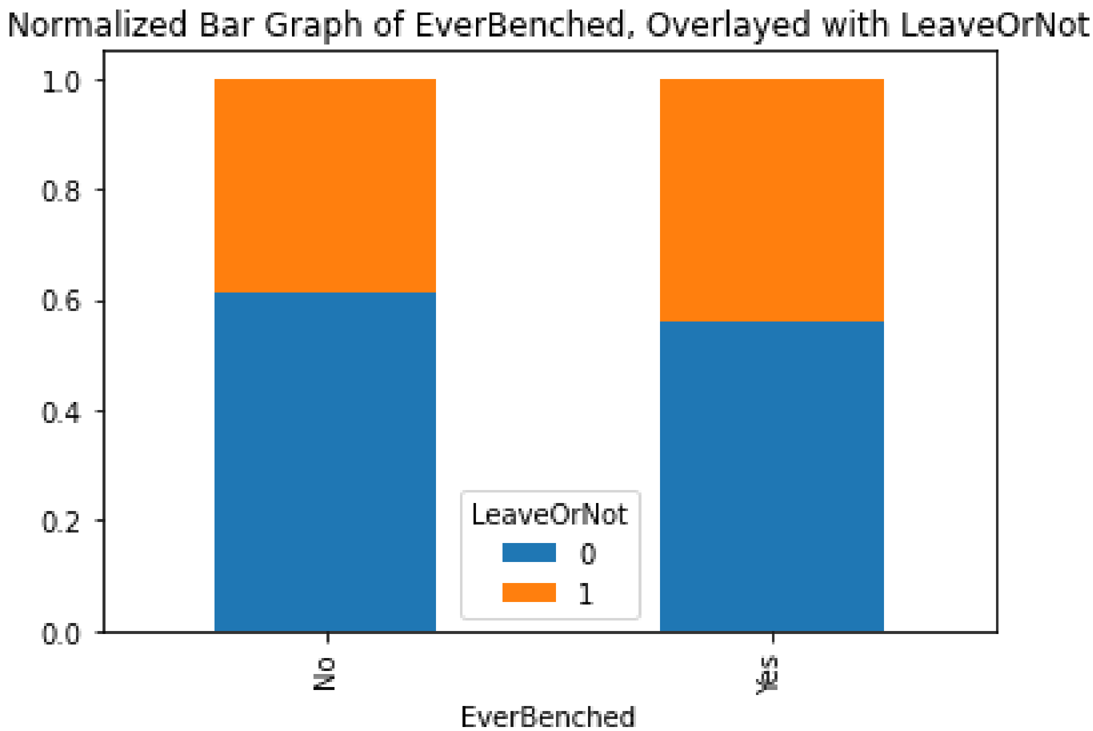
[71]: <AxesSubplot:title={'center':'Bar Chart of EverBenched, Overlayed with LeaveOrNot'}, xlabel='EverBenched'>



[72]:
```
benchleave_n = benchleave.div(benchleave.sum(1), axis = 0)
```

[73]:
```
benchleave_n.plot(kind='bar', stacked=True, title = 'Normalized Bar Graph of␣
↪EverBenched, Overlayed with LeaveOrNot')
```

[73]: <AxesSubplot:title={'center':'Normalized Bar Graph of EverBenched, Overlayed with LeaveOrNot'}, xlabel='EverBenched'>



[74]:
```
round(benchleave.div(benchleave.sum(0), axis = 1)*100, 1)
```

[74]:
```
LeaveOrNot      0     1
EverBenched
No           87.9  85.4
Yes          12.1  14.6
```

INSIGHT This does seem to have an effect, let's include.

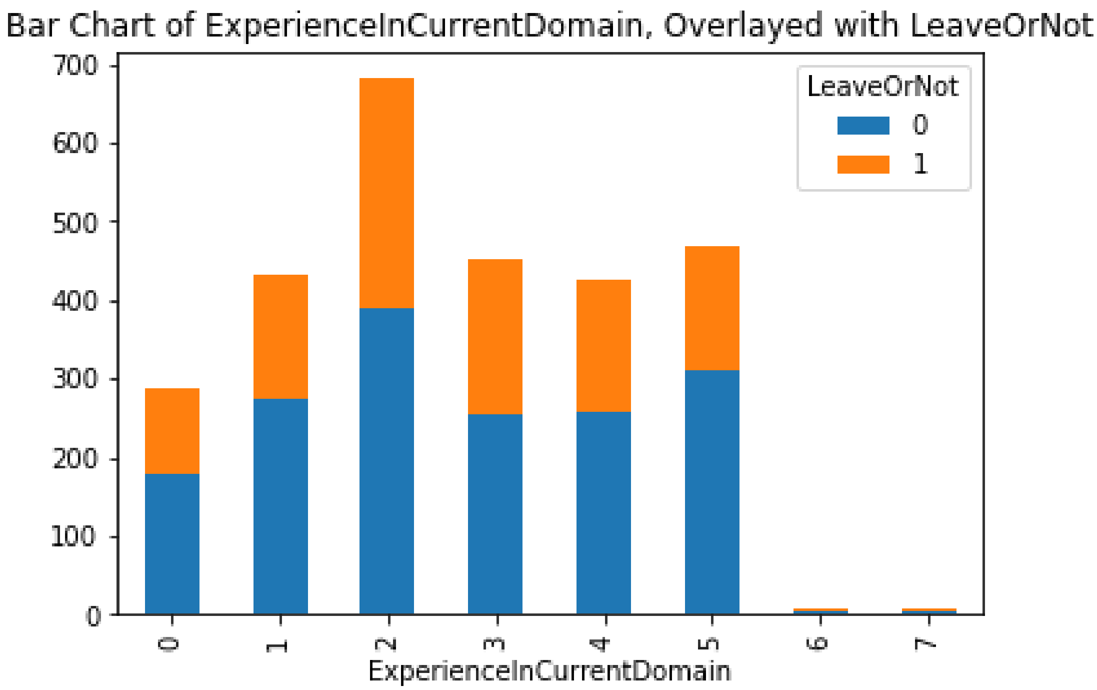
ExperienceinCurrentDomain and LeaveOrNot

[75]:
```
expleave = pd.crosstab(d['ExperienceInCurrentDomain'], d['LeaveOrNot'])
expleave
```

[75]:
```
LeaveOrNot                   0    1
ExperienceInCurrentDomain
0                          178  109
1                          273  160
2                          390  291
3                          255  196
4                          258  167
5                          310  160
6                            6    2
7                            6    3
```

[76]:
```
expleave.plot(kind='bar', stacked = True, title = 'Bar Chart of␣
↪ExperienceInCurrentDomain, Overlayed with LeaveOrNot')
```

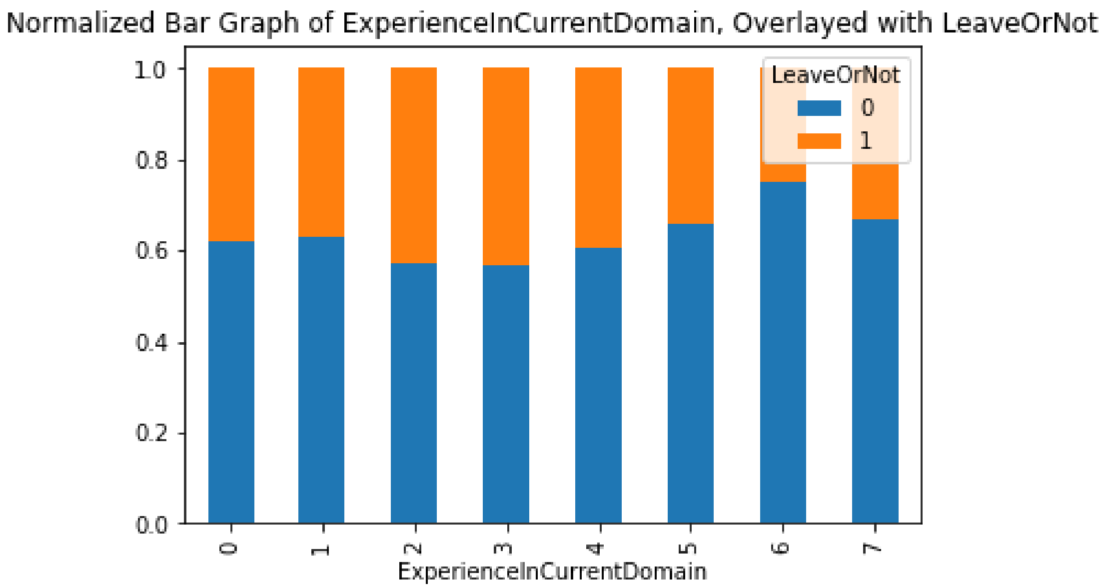
[76]:
```
<AxesSubplot:title={'center':'Bar Chart of ExperienceInCurrentDomain, Overlayed
with LeaveOrNot'}, xlabel='ExperienceInCurrentDomain'>
```



[77]:
```
expleave_n = expleave.div(expleave.sum(1), axis = 0)
```

[78]:
```
expleave_n.plot(kind='bar', stacked=True, title = 'Normalized Bar Graph of␣
↪ExperienceInCurrentDomain, Overlayed with LeaveOrNot')
```

[78]:
```
<AxesSubplot:title={'center':'Normalized Bar Graph of ExperienceInCurrentDomain,
Overlayed with LeaveOrNot'}, xlabel='ExperienceInCurrentDomain'>
```



[79]:
```
round(expleave.div(expleave.sum(0), axis = 1)*100, 1)
```

[79]:
```
LeaveOrNot                    0     1
ExperienceInCurrentDomain
0                          10.6  10.0
1                          16.3  14.7
2                          23.3  26.7
3                          15.2  18.0
4                          15.4  15.3
5                          18.5  14.7
6                           0.4   0.2
7                           0.4   0.3
```

INSIGHT Let's combine into ranges and see if there's more of an effect.

[80]:
```
exprange = {0: '0-1', 1: '0-1', 2: '2-3', 3: '2-3', 4: '4-5', 5: '4-5', 6:␣
↪'6-7', 7: '6-7'}

d['ExpYearRange'] = d['ExperienceInCurrentDomain'].map(exprange)
```

[81]:
```
exprleave = pd.crosstab(d['ExpYearRange'], d['LeaveOrNot'])
exprleave
```

[81]:
```
LeaveOrNot      0    1
ExpYearRange
0-1           451  269
2-3           645  487
```

```
4-5              568  327
6-7               12    5
```

[82]:
```
round(exprleave.div(exprleave.sum(0), axis = 1)*100, 1)
```

[82]:
```
LeaveOrNot         0     1
ExpYearRange
0-1             26.9  24.7
2-3             38.5  44.8
4-5             33.9  30.1
6-7              0.7   0.5
```

INSIGHT To include or not include?

[ ]:
```

```

2.0-iac-baseline

April 17, 2022

1 Baseline Notebook

[26]:
```
import pandas as pd
from sklearn.dummy import DummyClassifier
from sklearn.model_selection import train_test_split
```

1.1 Testing for baseline values

[27]:
```
employee = pd.read_csv("../data/employee_cleaned.csv")
```

1.2 Partition Data prior to testing baseline model

[28]:
```
X = employee.loc[:, employee.columns != 'LeaveOrNot']
y = employee.loc[:, employee.columns == 'LeaveOrNot']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.40,␣
 ↪random_state=7)
```

1.3 Initializing Dummy Classifier

Strategy implemented for this project is the uniform approach: “uniform”: generates predictions uniformly at random from the list of unique classes observed in y_train, i.e. each class has equal probability. Random State set to 7 for reproducibility

[29]:
```
dummy_clf = DummyClassifier(strategy='uniform', random_state = 7)
dummy_clf.fit(X_train,y_train.values.ravel())
```

[29]:
```
DummyClassifier(random_state=7, strategy='uniform')
```

1.4 Storing predicted results for baseline metrics

[30]:
```
# Check for Model Accuracy
y_pred = dummy_clf.predict(X_test)
```

1.5 Evaluating metrics

[31]:
```
ypred = pd.crosstab(y_test['LeaveOrNot'], y_pred, rownames=['Actual'],␣
↪colnames=['Predicted'])
ypred['Total'] = ypred.sum(axis=1);
ypred.loc['Total'] = ypred.sum()
print(ypred)
```

```
Predicted    0    1  Total
Actual
0          338  312    650
1          224  232    456
Total      562  544   1106
```

[32]:
```
TP = ypred[1][1]
TN = ypred[0][0]
FP = ypred[1][0]
FN = ypred[0][1]
TAN = TN + FP
TAP = FN + TP
TPN = TN + FN
TPP = FP + TP
GT = ypred['Total']['Total']
```

[33]:
```
from tabulate import tabulate
```

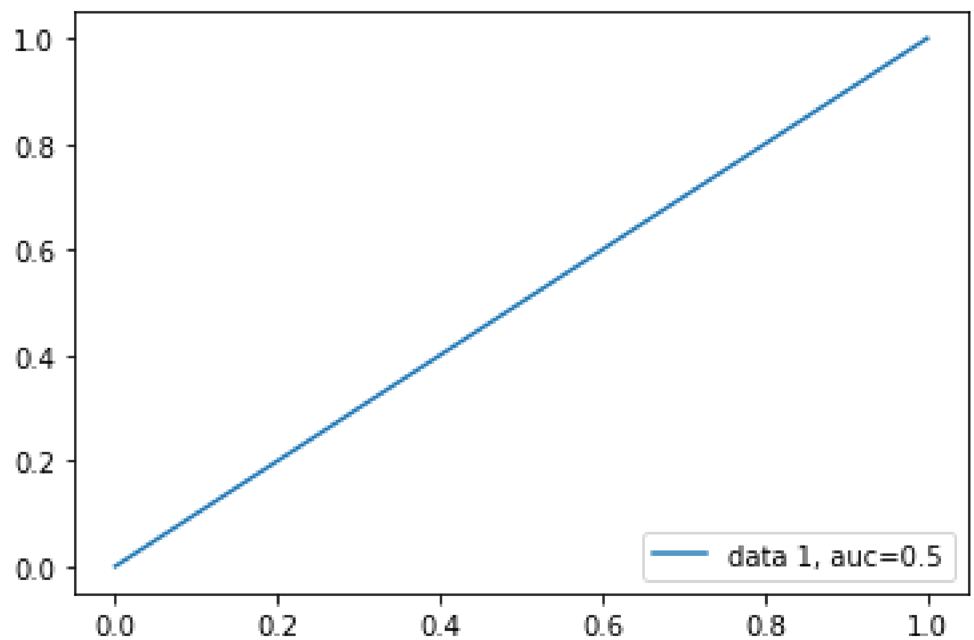
[34]:
```
accuracy = round((TN + TP) / GT, 4)
sensitivity = round(TP / TAP, 4)
specificity = round(TN / TAN, 4)
precision = round(TP / TPP, 4)
recall = round(TP / (TP + FN), 4)
pxr = precision * recall
ppr = precision + recall
F1 = round((pxr / ppr) * 2, 4)
F2 = round((pxr / ((4 * precision) + recall)) * 5, 4)
F05 = round((pxr / ((0.25 * precision) + recall)) * 1.25, 4)
```

[35]:
```
data = [["Accuracy", "(TN+TP)/GT", accuracy], ["Error rate", "1-Accuracy", 1 -␣
↪accuracy],
        ["Sensitivity = Recall", "TP/TAP", sensitivity], ["Specificity", "TN/
↪TAN", specificity],
        ["Precision", "TP/TPP", precision], ["F1", "2*(precision*recall)/
↪(precision+recall)", F1],
        ["F2", "5*(precision*recall)/((4*precision)+recall)", F2],
        ["F0.5", "1.25*(precision*recall)/((0.25*precision)+recall)", F05]]
col_names = ["Evaluation Measure", "Formula", "Value"]
print(tabulate(data, headers=col_names, tablefmt="fancy_grid"))
```

```
 Evaluation Measure    Formula
Value

 Accuracy              (TN+TP)/GT
0.5154

 Error rate            1-Accuracy
0.4846

 Sensitivity = Recall  TP/TAP
0.5088

 Specificity           TN/TAN
0.52

 Precision             TP/TPP
0.4265

 F1                    2*(precision*recall)/(precision+recall)
0.464

 F2                    5*(precision*recall)/((4*precision)+recall)
0.4899

 F0.5                  1.25*(precision*recall)/((0.25*precision)+recall)
0.4408
```

[36]:

```

```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
data 1, auc=0.5

2.0-iac-logistic-regression

April 17, 2022

1 Logistic Regression Notebook

[309]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
from sklearn import metrics
```

1.1 Testing for initial hypothesis LeaveOrNot ~ Gender + EverBenched + PaymentTier

[310]:
```
employee = pd.read_csv("../data/employee_cleaned.csv")
```

1.2 Encoding Categories via panda dummy variables

[311]:
```
employee = pd.get_dummies(employee, columns=["Gender",␣
 ↪"EverBenched","PaymentTier"])
employee = employee.drop(columns=['Age', 'Education',␣
 ↪'ExperienceInCurrentDomain','City','JoiningYear','Duration'])
```

1.3 Partition Data prior to training model

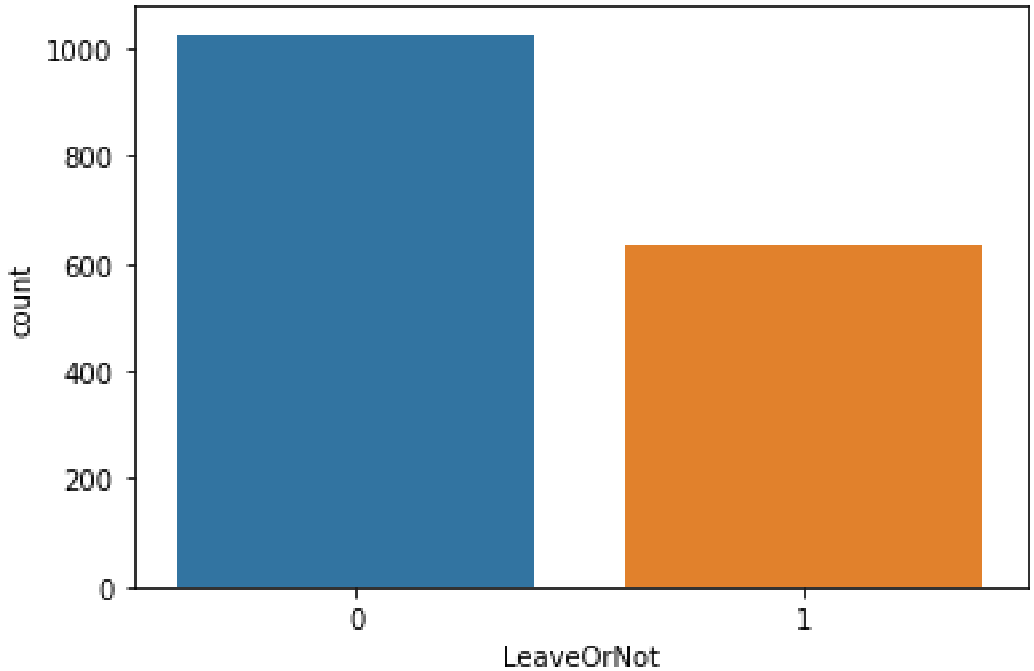
[312]:
```
X = employee.loc[:, employee.columns != 'LeaveOrNot']
X = sm.add_constant(X)
y = employee.loc[:, employee.columns == 'LeaveOrNot']
```

```
/Users/ivan/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
```

[313]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.40,␣
 ↪random_state=7)
```

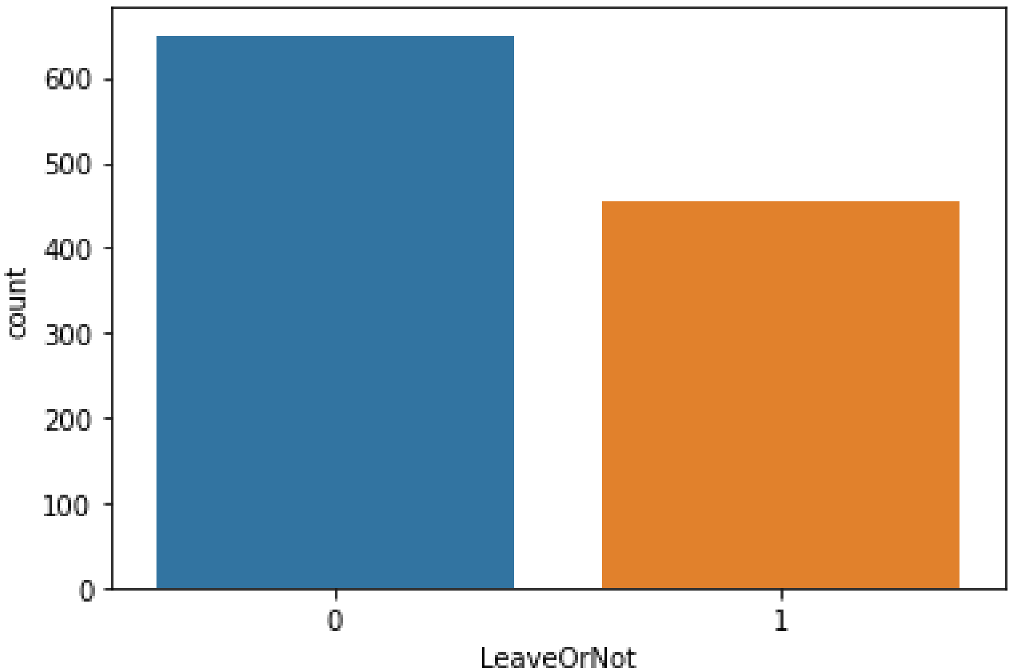
[314]:
```
sns.countplot(x="LeaveOrNot", data = y_train)
```

[314]: <AxesSubplot:xlabel='LeaveOrNot', ylabel='count'>



[315]:
```
sns.countplot(x="LeaveOrNot", data = y_test)
```

[315]: <AxesSubplot:xlabel='LeaveOrNot', ylabel='count'>



1.4 Executing Logistic Regression

[316]:
```
from sklearn.linear_model import LogisticRegression

logReg = LogisticRegression().fit(X_train, y_train.values.ravel())
```

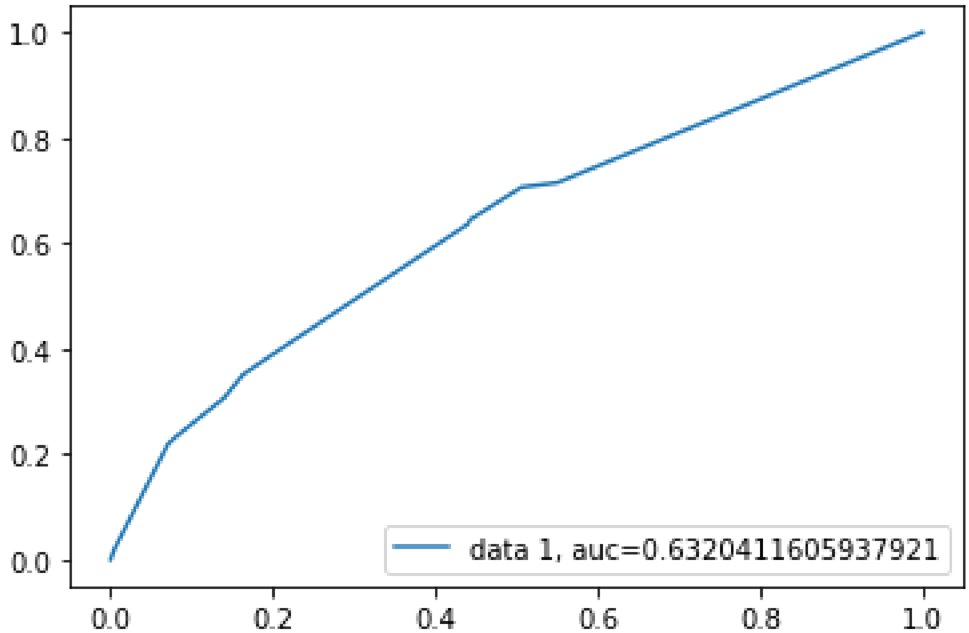
1.5 Predicting Values and Evaluating Metrics

[317]:
```
y_pred = logReg.predict(X_test)
ypred = pd.crosstab(y_test['LeaveOrNot'], y_pred, rownames=['Actual'],␣
  ↪colnames=['Predicted'])
ypred['Total'] = ypred.sum(axis=1);
ypred.loc['Total'] = ypred.sum()
print(ypred)
```

```
Predicted    0    1  Total
Actual
0          599   51    650
1          351  105    456
Total      950  156   1106
```

[318]:
```
y_pred_proba = logReg.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
```

```
plt.legend(loc=4)
plt.show()
```



[319]:
```
TP = ypred[1][1]
TN = ypred[0][0]
FP = ypred[1][0]
FN = ypred[0][1]
TAN = TN + FP
TAP = FN + TP
TPN = TN + FN
TPP = FP + TP
GT = ypred['Total']['Total']
```

[320]:
```
from tabulate import tabulate
```

[321]:
```
accuracy = round((TN + TP) / GT, 4)
sensitivity = round(TP / TAP, 4)
specificity = round(TN / TAN, 4)
precision = round(TP / TPP, 4)
recall = round(TP / (TP + FN), 4)
pxr = precision * recall
ppr = precision + recall
F1 = round((pxr / ppr) * 2, 4)
F2 = round((pxr / ((4 * precision) + recall)) * 5, 4)
F05 = round((pxr / ((0.25 * precision) + recall)) * 1.25, 4)
```

[322]:
```
data = [["Accuracy", "(TN+TP)/GT", accuracy], ["Error rate", "1-Accuracy", 1 -␣
↪accuracy],
        ["Sensitivity = Recall", "TP/TAP", sensitivity], ["Specificity", "TN/
↪TAN", specificity],
        ["Precision", "TP/TPP", precision], ["F1", "2*(precision*recall)/
↪(precision+recall)", F1],
        ["F2", "5*(precision*recall)/((4*precision)+recall)", F2],
        ["F0.5", "1.25*(precision*recall)/((0.25*precision)+recall)", F05]]
col_names = ["Evaluation Measure", "Formula", "Value"]
print(tabulate(data, headers=col_names, tablefmt="fancy_grid"))
```

```
 Evaluation Measure    Formula
Value

 Accuracy              (TN+TP)/GT
0.6365

 Error rate            1-Accuracy
0.3635

 Sensitivity = Recall  TP/TAP
0.2303

 Specificity           TN/TAN
0.9215

 Precision             TP/TPP
0.6731

 F1                    2*(precision*recall)/(precision+recall)
0.3432

 F2                    5*(precision*recall)/((4*precision)+recall)
0.2652

 F0.5                  1.25*(precision*recall)/((0.25*precision)+recall)
0.4862
```

1.6 Testing for alt hypothesis LeaveOrNot ~ City + Gender + Duration + PaymentTier

[323]:
```
employee = pd.read_csv("../data/employee_cleaned.csv")
```

1.7 Encoding Categories via panda dummy variables

[324]:
```
employee = pd.get_dummies(employee, columns=["City","Gender","Duration",␣
↪"PaymentTier"])
employee = employee.drop(columns =␣
↪['Age','Education','EverBenched','ExperienceInCurrentDomain', 'JoiningYear'])
```

1.8 Partition Data prior to training model

[325]:
```
X = employee.loc[:, employee.columns != 'LeaveOrNot']
X = sm.add_constant(X)
y = employee.loc[:, employee.columns == 'LeaveOrNot']
```

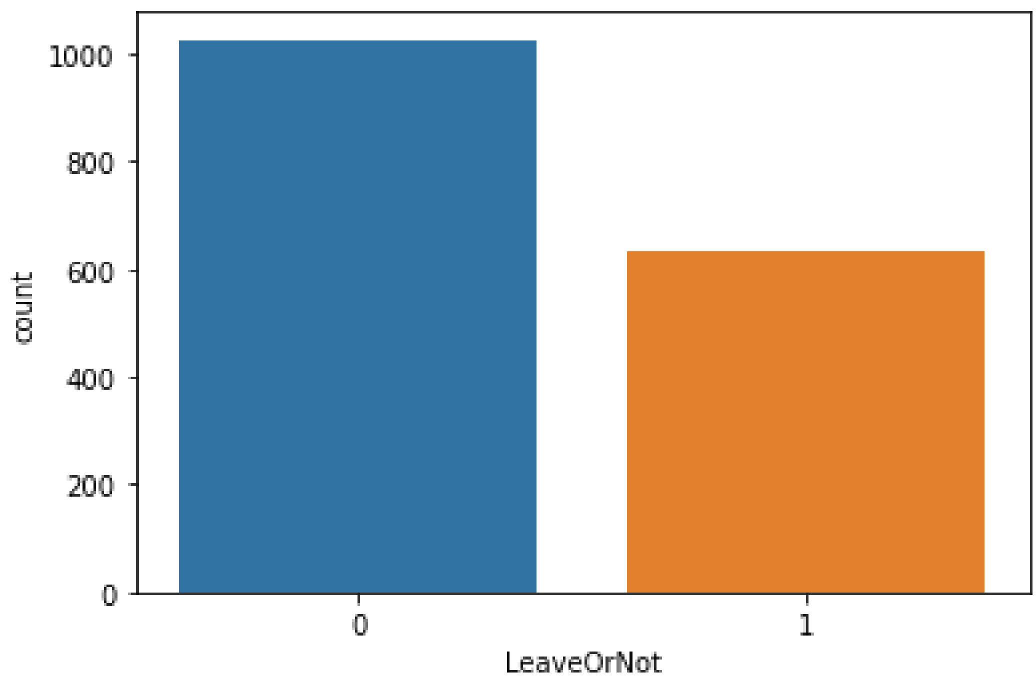
```
/Users/ivan/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of
pandas all arguments of concat except for the argument 'objs' will be keyword-
only
  x = pd.concat(x[::order], 1)
```

[326]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.40,␣
↪random_state=7)
```

1.9 Validating Partition

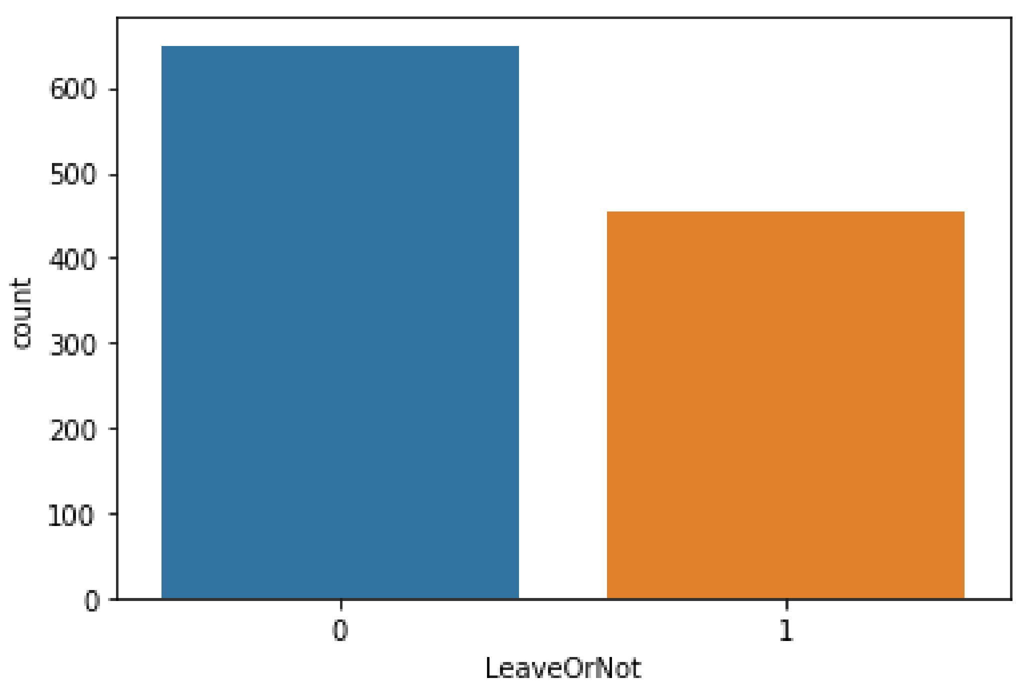
[327]:
```
sns.countplot(x="LeaveOrNot", data = y_train)
```

[327]: `<AxesSubplot:xlabel='LeaveOrNot', ylabel='count'>`



[328]:
```
sns.countplot(x="LeaveOrNot", data = y_test)
```

[328]: <AxesSubplot:xlabel='LeaveOrNot', ylabel='count'>



1.10 Executing Logistic Regression

[329]:
```
from sklearn.linear_model import LogisticRegression
```

[330]:
```
logReg = LogisticRegression().fit(X_train, y_train.values.ravel())
```

1.11 Predicting Values and Evaluating Metrics

[331]:
```
y_pred = logReg.predict(X_test)
```

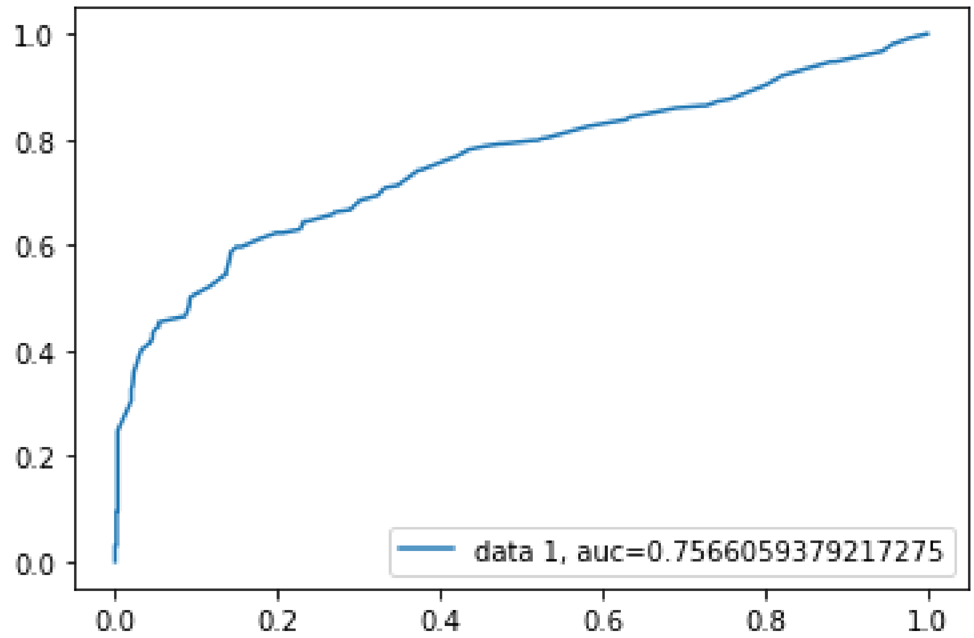
[332]:
```
ypred = pd.crosstab(y_test['LeaveOrNot'], y_pred, rownames = ['Actual'],␣
 ↪colnames = ['Predicted'])
ypred['Total'] = ypred.sum(axis=1); ypred.loc['Total'] = ypred.sum()
print(ypred)
```

```
Predicted    0    1  Total
Actual
0          589   61    650
1          227  229    456
Total      816  290   1106
```

[333]:
```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Accuracy: 0.7396021699819169
Precision: 0.7896551724137931
Recall: 0.5021929824561403
```

[334]:
```
y_pred_proba = logReg.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



[335]:
```
np.savetxt("../data/log-regression/y_proba.csv", y_pred_proba, delimiter=",")
y_test.to_csv('../data/log-regression/y_test.csv', index=False)
```

[336]:
```
TP = ypred[1][1]
TN = ypred[0][0]
FP = ypred[1][0]
FN = ypred[0][1]
TAN = TN + FP
TAP = FN + TP
TPN = TN + FN
TPP = FP + TP
GT = ypred['Total']['Total']
```

[337]:
```
from tabulate import tabulate
```

[338]:
```
accuracy = round((TN + TP) / GT, 4)
sensitivity = round(TP / TAP, 4)
specificity = round(TN / TAN, 4)
precision = round(TP / TPP, 4)
recall = round(TP / (TP + FN), 4)
pxr = precision * recall
ppr = precision + recall
F1 = round((pxr / ppr) * 2, 4)
F2 = round((pxr / ((4 * precision) + recall)) * 5, 4)
F05 = round((pxr / ((0.25 * precision) + recall)) * 1.25, 4)
```

[339]:
```python
data = [["Accuracy", "(TN+TP)/GT", accuracy], ["Error rate", "1-Accuracy", 1 -
 ↪accuracy],
        ["Sensitivity = Recall", "TP/TAP", sensitivity], ["Specificity", "TN/
 ↪TAN", specificity],
        ["Precision", "TP/TPP", precision], ["F1", "2*(precision*recall)/
 ↪(precision+recall)", F1],
        ["F2", "5*(precision*recall)/((4*precision)+recall)", F2],
        ["F0.5", "1.25*(precision*recall)/((0.25*precision)+recall)", F05]]
col_names = ["Evaluation Measure", "Formula", "Value"]
print(tabulate(data, headers=col_names, tablefmt="fancy_grid"))
```

```
 Evaluation Measure     Formula
Value

 Accuracy               (TN+TP)/GT
0.7396

 Error rate             1-Accuracy
0.2604

 Sensitivity = Recall   TP/TAP
0.5022

 Specificity            TN/TAN
0.9062

 Precision              TP/TPP
0.7897

 F1                     2*(precision*recall)/(precision+recall)
0.614

 F2                     5*(precision*recall)/((4*precision)+recall)
0.5416

 F0.5                   1.25*(precision*recall)/((0.25*precision)+recall)
0.7086
```

2.0-iac-naive-bayes

April 17, 2022

1 Naive Bayes Notebook

[26]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
import numpy as np
```

1.1 Testing for initial hypothesis LeaveOrNot ~ Gender + EverBenched + PaymentTier

[27]:
```
employee = pd.read_csv("../data/employee_cleaned.csv")
```

1.2 Encoding Categories via panda dummy variables

[28]:
```
employee = pd.get_dummies(employee, columns=["Gender",␣
 ↪"EverBenched","PaymentTier"])
employee = employee.drop(columns=['Age', 'Education',␣
 ↪'ExperienceInCurrentDomain','City','JoiningYear','Duration'])
```

1.3 Partition Data prior to training model

[29]:
```
X = employee.loc[:, employee.columns != 'LeaveOrNot']
y = employee.loc[:, employee.columns == 'LeaveOrNot']
```

[30]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.40,␣
 ↪random_state=7)
```

1.4 Running Naive Bayes Algorithm

[31]:
```
nb = MultinomialNB().fit(X_train, y_train.values.ravel())
```

[32]:
```
y_pred = nb.predict(X_test)
```

1.5 Evaluating Naive Bayes Predictions

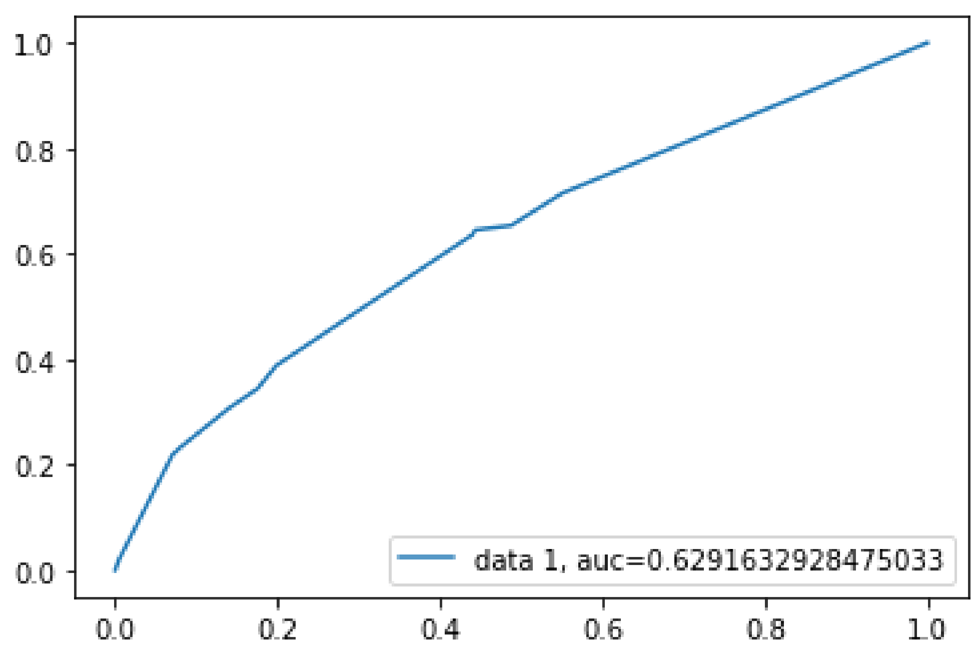
[33]:
```
ypred = pd.crosstab(y_test['LeaveOrNot'], y_pred, rownames = ['Actual'],␣
 ↪colnames = ['Predicted'])
ypred['Total'] = ypred.sum(axis=1); ypred.loc['Total'] = ypred.sum()
print(ypred)
```

```
Predicted    0    1  Total
Actual
0          598   52    650
1          350  106    456
Total      948  158   1106
```

[34]:
```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Accuracy: 0.6365280289330922
Precision: 0.6708860759493671
Recall: 0.2324561403508772
```

[35]:
```
y_pred_proba = nb.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



[36]:
```
TP = ypred[1][1]
TN = ypred[0][0]
FP = ypred[1][0]
FN = ypred[0][1]
TAN = TN + FP
TAP = FN + TP
TPN = TN + FN
TPP = FP + TP
GT = ypred['Total']['Total']
```

[37]:
```
from tabulate import tabulate
```

[38]:
```
accuracy = round((TN + TP) / GT, 4)
sensitivity = round(TP / TAP, 4)
specificity = round(TN / TAN, 4)
precision = round(TP / TPP, 4)
recall = round(TP / (TP + FN), 4)
pxr = precision * recall
ppr = precision + recall
F1 = round((pxr / ppr) * 2, 4)
F2 = round((pxr / ((4 * precision) + recall)) * 5, 4)
F05 = round((pxr / ((0.25 * precision) + recall)) * 1.25, 4)
```

[39]:
```
data = [["Accuracy", "(TN+TP)/GT", accuracy], ["Error rate", "1-Accuracy", 1 -␣
↪accuracy],
        ["Sensitivity = Recall", "TP/TAP", sensitivity], ["Specificity", "TN/
↪TAN", specificity],
        ["Precision", "TP/TPP", precision], ["F1", "2*(precision*recall)/
↪(precision+recall)", F1],
        ["F2", "5*(precision*recall)/((4*precision)+recall)", F2],
        ["F0.5", "1.25*(precision*recall)/((0.25*precision)+recall)", F05]]
col_names = ["Evaluation Measure", "Formula", "Value"]
print(tabulate(data, headers=col_names, tablefmt="fancy_grid"))
```

```
 Evaluation Measure     Formula
Value


 Accuracy               (TN+TP)/GT
0.6365


 Error rate             1-Accuracy
0.3635
```

```
 Sensitivity = Recall   TP/TAP
0.2325


 Specificity            TN/TAN
0.92


 Precision              TP/TPP
0.6709


 F1                     2*(precision*recall)/(precision+recall)
0.3453


 F2                     5*(precision*recall)/((4*precision)+recall)
0.2675


 F0.5                   1.25*(precision*recall)/((0.25*precision)+recall)
0.4872
```

1.6 Testing for alt hypothesis LeaveOrNot ~ City + Gender + Duration + PaymentTier

[40]:
```
#Resetting Dataframe
employee = pd.read_csv("../data/employee_cleaned.csv")
```

1.7 Encoding Categories via panda dummy variables

[41]:
```
employee = pd.get_dummies(employee, columns=["City","Gender","Duration",␣
 ↪"PaymentTier"])
employee = employee.drop(columns =␣
 ↪['Age','Education','EverBenched','ExperienceInCurrentDomain', 'JoiningYear'])
```

1.8 Partition Data prior to training model

[42]:
```
X = employee.loc[:, employee.columns != 'LeaveOrNot']
y = employee.loc[:, employee.columns == 'LeaveOrNot']
```

[43]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.40,␣
 ↪random_state=7)
```

1.9 Running Naive Bayes Algorithm

[44]:
```
nb = MultinomialNB().fit(X_train, y_train.values.ravel())
```

[45]:
```
y_pred = nb.predict(X_test)
```

1.10 Evaluating Naive Bayes Predictions

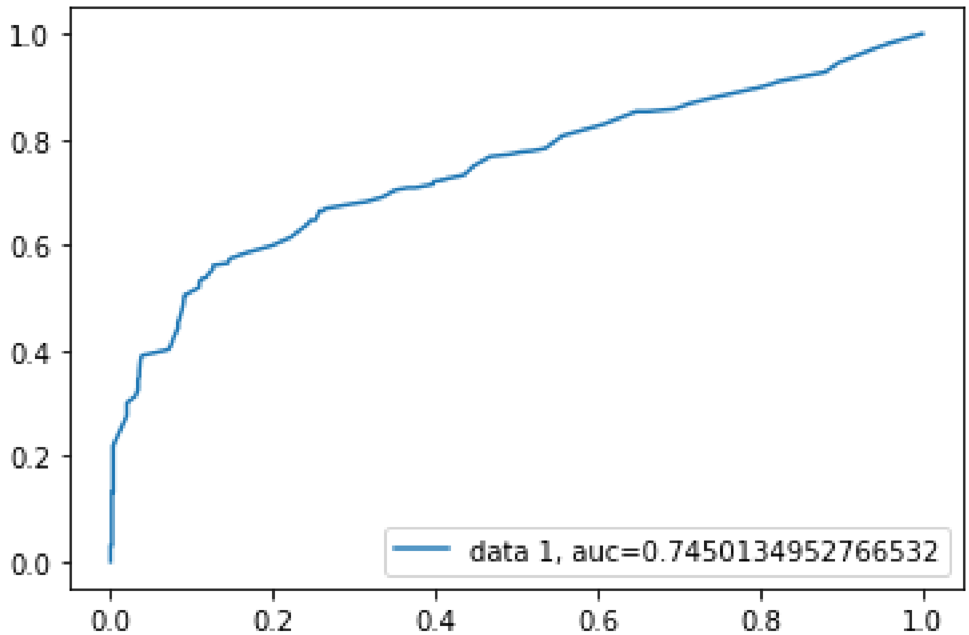
[46]:
```
ypred = pd.crosstab(y_test['LeaveOrNot'], y_pred, rownames = ['Actual'],␣
 ↪colnames = ['Predicted'])
ypred['Total'] = ypred.sum(axis=1); ypred.loc['Total'] = ypred.sum()
print(ypred)
```

```
Predicted    0    1  Total
Actual
0          579   71    650
1          219  237    456
Total      798  308   1106
```

[47]:
```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Accuracy: 0.7377938517179023
Precision: 0.7694805194805194
Recall: 0.5197368421052632
```

[48]:
```
y_pred_proba = nb.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



[49]:
```
np.savetxt("../data/naive-bayes/y_proba.csv", y_pred_proba, delimiter=",")
y_test.to_csv('../data/naive-bayes/y_test.csv', index=False)
```

[50]:
```
TP = ypred[1][1]
TN = ypred[0][0]
FP = ypred[1][0]
FN = ypred[0][1]
TAN = TN + FP
TAP = FN + TP
TPN = TN + FN
TPP = FP + TP
GT = ypred['Total']['Total']
```

[51]:
```
from tabulate import tabulate
```

[52]:
```
accuracy = round((TN + TP) / GT, 4)
sensitivity = round(TP / TAP, 4)
specificity = round(TN / TAN, 4)
precision = round(TP / TPP, 4)
recall = round(TP / (TP + FN), 4)
pxr = precision * recall
ppr = precision + recall
F1 = round((pxr / ppr) * 2, 4)
F2 = round((pxr / ((4 * precision) + recall)) * 5, 4)
F05 = round((pxr / ((0.25 * precision) + recall)) * 1.25, 4)
```

[53]:
```
data = [["Accuracy", "(TN+TP)/GT", accuracy], ["Error rate", "1-Accuracy", 1 -␣
↪accuracy],
        ["Sensitivity = Recall", "TP/TAP", sensitivity], ["Specificity", "TN/
↪TAN", specificity],
        ["Precision", "TP/TPP", precision], ["F1", "2*(precision*recall)/
↪(precision+recall)", F1],
        ["F2", "5*(precision*recall)/((4*precision)+recall)", F2],
        ["F0.5", "1.25*(precision*recall)/((0.25*precision)+recall)", F05]]
col_names = ["Evaluation Measure", "Formula", "Value"]
print(tabulate(data, headers=col_names, tablefmt="fancy_grid"))
```

```
 Evaluation Measure     Formula
Value

 Accuracy               (TN+TP)/GT
0.7378

 Error rate             1-Accuracy
0.2622

 Sensitivity = Recall   TP/TAP
0.5197

 Specificity            TN/TAN
0.8908

 Precision              TP/TPP
0.7695

 F1                     2*(precision*recall)/(precision+recall)
0.6204

 F2                     5*(precision*recall)/((4*precision)+recall)
0.5558

 F0.5                   1.25*(precision*recall)/((0.25*precision)+recall)
0.702
```

halle_CART

April 17, 2022

1 Team 3 Final Project: CART

1.1 1. Data Importing and Pre-Processing

1.1.1 Importing Libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from scipy.stats import mode
import plotly.graph_objects as go
import random
from scipy import stats
import statsmodels.tools.tools as stattools
from sklearn.tree import DecisionTreeClassifier, export_graphviz
import graphviz
from sklearn.tree import plot_tree
from sklearn import tree
from sklearn.metrics import confusion_matrix
from sklearn.datasets import make_classification
from sklearn.metrics import plot_confusion_matrix
```

1.1.2 Data Import and Pre-Processing

[2]:
```
d = pd.read_csv("Employee.csv")
```

[3]:
```
d = d.drop_duplicates()
```

[4]:
```
d['Duration'] = 2020 - d['JoiningYear']
```

[5]:
```
edlevel = {'Bachelors': 1, 'Masters': 2, 'PHD': 3}

d['EduLevel'] = d['Education'].map(edlevel)
```

1.1.3 Split Data into Train and Test

[6]:
```
dtrain, dtest = train_test_split(d, test_size = .33, random_state = 7)
```

[7]:
```
x = ['Original Dataset', 'Training Data', 'Test Data']
y = [d.shape[0], dtrain.shape[0], dtest.shape[0]]

fig = go.Figure(data=[go.Bar(x=x, y=y)])
fig.update_layout(title_text='Confirming Split')
fig.show()
```

1.1.4 Check for/Fix Any Imbalance Issues

[8]:
```
dtrain['LeaveOrNot'].value_counts()
```

[8]:
```
0    1131
1     720
Name: LeaveOrNot, dtype: int64
```

[9]:
```
ratio = dtrain['LeaveOrNot'].value_counts()[1]/dtrain.shape[0] * 100
ratio
```

[9]:
```
38.897893030794165
```

We shouldn't have any imbalanced data set issues.

1.1.5 Prepare Data for CART

[10]:
```
ytrain = dtrain[['LeaveOrNot']]
```

[11]:
```
ytest = dtest[['LeaveOrNot']]
```

[12]:
```
dtrain = pd.get_dummies(dtrain, prefix=None, columns=["City", "Gender",␣
↪"EverBenched"], drop_first=False)
```

[13]:
```
dtest = pd.get_dummies(dtest, prefix=None, columns=["City", "Gender",␣
↪"EverBenched"], drop_first=False)
```

1.1.6 Create Xtrain and Xtest

[14]:
```
dtrain.columns
```

[14]:
```
Index(['Education', 'JoiningYear', 'PaymentTier', 'Age',
       'ExperienceInCurrentDomain', 'LeaveOrNot', 'Duration', 'EduLevel',
       'City_Bangalore', 'City_New Delhi', 'City_Pune', 'Gender_Female',
       'Gender_Male', 'EverBenched_No', 'EverBenched_Yes'],
      dtype='object')
```

[15]:
```
Xdpgc = dtrain[["Duration", "PaymentTier", "Gender_Female", "Gender_Male",␣
↪"City_Bangalore", "City_New Delhi", "City_Pune"]]
Xtdpgc = dtest[["Duration", "PaymentTier", "Gender_Female", "Gender_Male",␣
↪"City_Bangalore", "City_New Delhi", "City_Pune"]]
```

[16]:
```
X_names = ["Duration", "PaymentTier", "Gender_Female", "Gender_Male",␣
↪"City_Bangalore", "City_New Delhi", "City_Pune"]
y_names = ["No, Won't Leave", "Yes, Wil Leave"]
```

1.1.7 Create Model

[17]:
```
cart = DecisionTreeClassifier(criterion = "gini", max_leaf_nodes = 5).
↪fit(Xdpgc,ytrain)
```

1.1.8 Metrics for Model

[18]:
```
predict = cart.predict(Xtdpgc)
```

[19]:
```
cm = confusion_matrix(ytest, predict)
cm
```

[19]:
```
array([[485,  60],
       [202, 166]])
```

[20]:
```
TN = cm[0][0]
FP = cm[0][1]
FN = cm[1][0]
TP = cm[1][1]
```

[21]:
```
GT = TN + FP + FN + TP
Accuracy = (TN + TP)/GT
ErrorRate = 1-Accuracy
Sensitivity = TP/(FN + TP)
Recall = Sensitivity
Specificity = TN/(TN + FP)
Precision = TP/(FP + TP)
F1 = (2*Precision*Recall)/(Precision + Recall)
F2 = (5*Precision*Recall)/((4*Precision) + Recall)
F0_5 = (1.25*Precision*Recall)/((.25*Precision)+Recall)
```

[22]:
```
print(Accuracy)
print(ErrorRate)
print(Sensitivity)
print(Specificity)
print(Precision)
print(F1)
print(F2)
```

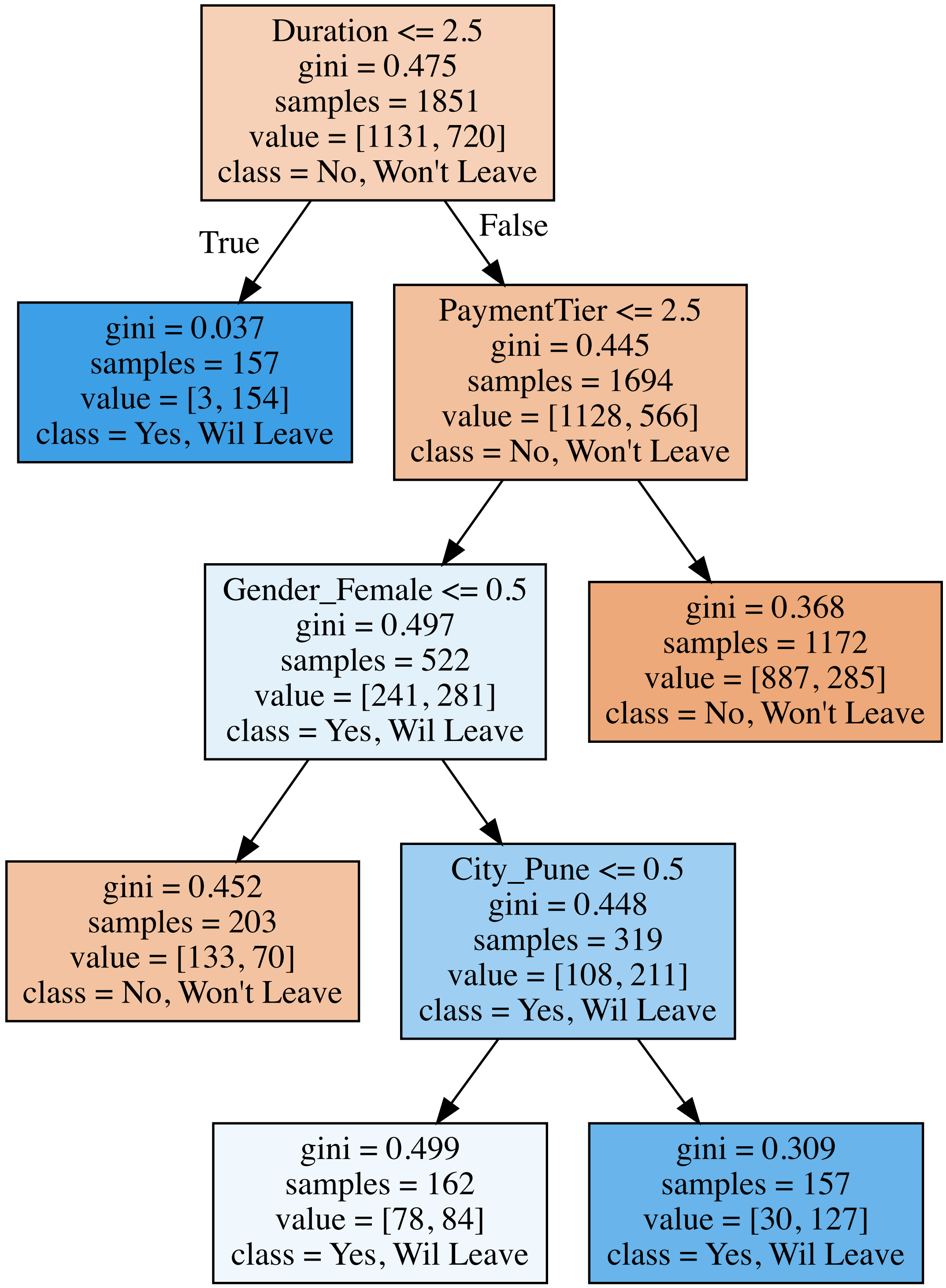
```python
print(F0_5)
```

```
0.7130339539978094
0.28696604600219056
0.45108695652173914
0.8899082568807339
0.7345132743362832
0.5589225589225588
0.48881036513545345
0.6525157232704403
```

1.1.9 Visualize Final Tree

[23]:
```python
tree_1 = tree.export_graphviz(cart, out_file=None, feature_names=X_names,
                              class_names=y_names, filled=True)
tree_1viz = graphviz.Source(tree_1, format="png")
tree_1viz
```

[23]:


Duration <= 2.5
gini = 0.475
samples = 1851
value = [1131, 720]
class = No, Won't Leave
True
False
gini = 0.037
samples = 157
value = [3, 154]
class = Yes, Wil Leave
PaymentTier <= 2.5
gini = 0.445
samples = 1694
value = [1128, 566]
class = No, Won't Leave
Gender_Female <= 0.5
gini = 0.497
samples = 522
value = [241, 281]
class = Yes, Wil Leave
gini = 0.368
samples = 1172
value = [887, 285]
class = No, Won't Leave
gini = 0.452
samples = 203
value = [133, 70]
class = No, Won't Leave
City_Pune <= 0.5
gini = 0.448
samples = 319
value = [108, 211]
class = Yes, Wil Leave
gini = 0.499
samples = 162
value = [78, 84]
class = Yes, Wil Leave
gini = 0.309
samples = 157
value = [30, 127]
class = Yes, Wil Leave

[ ]:

2.0-unp-C5

April 17, 2022

[23]:
```python
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, export_graphviz,  plot_tree
from sklearn import metrics
```

[24]:
```python
employee = pd.read_csv('../data/employee_cleaned.csv')
employee.head()
```

[24]:
```
   Education  JoiningYear       City  PaymentTier  Age  Gender EverBenched  \
0  Bachelors         2017  Bangalore            3   34    Male          No
1  Bachelors         2013       Pune            1   28  Female          No
2  Bachelors         2014  New Delhi            3   38  Female          No
3    Masters         2016  Bangalore            3   27    Male          No
4    Masters         2017       Pune            3   24    Male         Yes

   ExperienceInCurrentDomain  LeaveOrNot  Duration
0                          0           0         3
1                          3           1         7
2                          2           0         6
3                          5           1         4
4                          2           1         3
```

0.0.1 Replace Education categorical to numerical values

[25]:
```python
#employee['Education'].replace({'PHD': 'a', 'Masters': 'b', 'Bachelors': 'c'},␣
↪inplace=True)
employee['Education'].replace(to_replace = ['PHD', 'Masters', 'Bachelors'],␣
↪value = ['3', '2', '1'], inplace = True)
employee['Education'] = employee['Education'].astype('int64')
employee.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

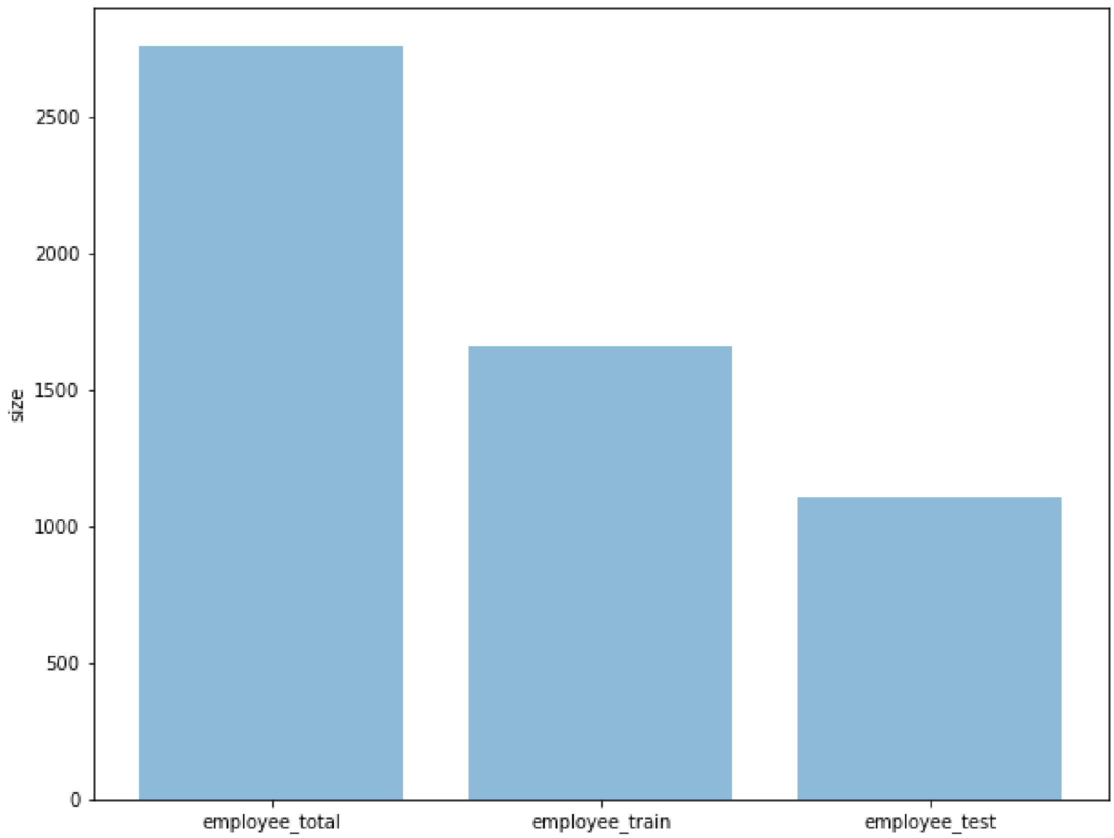
```
RangeIndex: 2764 entries, 0 to 2763
Data columns (total 10 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Education                  2764 non-null   int64
 1   JoiningYear                2764 non-null   int64
 2   City                       2764 non-null   object
 3   PaymentTier                2764 non-null   int64
 4   Age                        2764 non-null   int64
 5   Gender                     2764 non-null   object
 6   EverBenched                2764 non-null   object
 7   ExperienceInCurrentDomain  2764 non-null   int64
 8   LeaveOrNot                 2764 non-null   int64
 9   Duration                   2764 non-null   int64
dtypes: int64(7), object(3)
memory usage: 216.1+ KB
```

0.1 Split the data into training and test sets

[26]:
```python
employee_train, employee_test = train_test_split(employee, test_size = 0.40,
 ↪random_state = 7)
print(employee.shape)
print(employee_train.shape)
print(employee_test.shape)
```

```
(2764, 10)
(1658, 10)
(1106, 10)
```

[27]:
```python
portion = ('employee_total', 'employee_train', 'employee_test')
y_pos = np.arange(len(portion))
size = [2764,1658,1106]

plt.bar(y_pos, size, align='center', alpha=0.5)
plt.xticks(y_pos, portion)
plt.ylabel('size')

plt.show()
```



0.1.1 Separate data frames for interested predictor variables and response variable

0.1.2 *Data with all predictor variables

[28]:
```
#---Training set---

x_all = employee_train.drop(['LeaveOrNot'], axis= 1)
x_all = pd.get_dummies(x_all)
y_all = employee_train[['LeaveOrNot']]
y_names_all = ["No", "Yes"]

#---Test set---

x_test_all = employee_test.drop(['LeaveOrNot'], axis= 1)
x_test_all = pd.get_dummies(x_test_all)
y_test_all = employee_test[['LeaveOrNot']]

x_all.head(2)
```

[28]:
```
      Education  JoiningYear  PaymentTier  Age  ExperienceInCurrentDomain  \
2040          1         2017            3   33                          5
1872          1         2016            3   40                          0

      Duration  City_Bangalore  City_New Delhi  City_Pune  Gender_Female  \
2040         3               0               0          1              0
1872         4               1               0          0              0

      Gender_Male  EverBenched_No  EverBenched_Yes
2040            1               0                1
1872            1               1                0
```

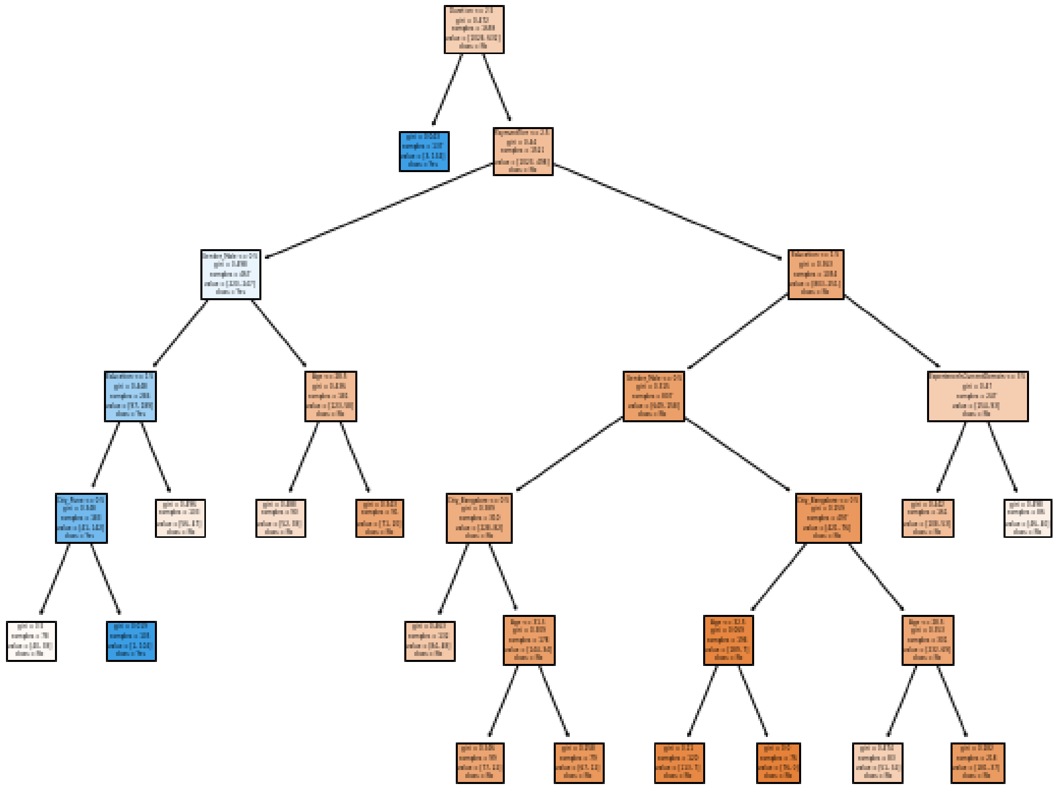
[29]:
```
#Run C5.0 using entropy criterion
C5_all = DecisionTreeClassifier(criterion = "gini", \
                                max_leaf_nodes = 15, \
                                min_samples_leaf= 75).fit(x_all,y_all)
export_graphviz (C5_all, out_file = 'C5_all.dot')

#predict income in training data set
y_train_pred_all = C5_all.predict(x_all)
y_train_pred_all
```

[29]:
```
array([0, 0, 0, …, 0, 0, 0], dtype=int64)
```

[30]:
```
#Visualize the tree
plt.rcParams['figure.figsize'] = (10, 8)
plot_tree(C5_all, feature_names=x_all.columns.values, filled=True,
         class_names=y_names_all)
```

[30]:
```
[Text(250.4659090909091, 403.81714285714287, 'Duration <= 2.5\ngini =
0.472\nsamples = 1658\nvalue = [1026, 632]\nclass = No'),
 Text(225.10227272727272, 341.69142857142856, 'gini = 0.043\nsamples =
137\nvalue = [3, 134]\nclass = Yes'),
 Text(275.82954545454544, 341.69142857142856, 'PaymentTier <= 2.5\ngini =
0.44\nsamples = 1521\nvalue = [1023, 498]\nclass = No'),
 Text(126.81818181818181, 279.5657142857143, 'Gender_Male <= 0.5\ngini =
0.498\nsamples = 467\nvalue = [220, 247]\nclass = Yes'),
 Text(76.0909090909091, 217.44, 'Education <= 1.5\ngini = 0.448\nsamples =
286\nvalue = [97, 189]\nclass = Yes'),
 Text(50.72727272727273, 155.3142857142857, 'City_Pune <= 0.5\ngini =
0.348\nsamples = 183\nvalue = [41, 142]\nclass = Yes'),
 Text(25.363636363636363, 93.18857142857144, 'gini = 0.5\nsamples = 78\nvalue =
[40, 38]\nclass = No'),
 Text(76.0909090909091, 93.18857142857144, 'gini = 0.019\nsamples = 105\nvalue =
[1, 104]\nclass = Yes'),
 Text(101.45454545454545, 155.3142857142857, 'gini = 0.496\nsamples = 103\nvalue
= [56, 47]\nclass = No'),
```

```
 Text(177.54545454545453, 217.44, 'Age <= 28.5\ngini = 0.436\nsamples =
181\nvalue = [123, 58]\nclass = No'),
 Text(152.1818181818182, 155.3142857142857, 'gini = 0.488\nsamples = 90\nvalue =
[52, 38]\nclass = No'),
 Text(202.9090909090909, 155.3142857142857, 'gini = 0.343\nsamples = 91\nvalue =
[71, 20]\nclass = No'),
 Text(424.84090909090907, 279.5657142857143, 'Education <= 1.5\ngini =
0.363\nsamples = 1054\nvalue = [803, 251]\nclass = No'),
 Text(342.4090909090909, 217.44, 'Gender_Male <= 0.5\ngini = 0.315\nsamples =
807\nvalue = [649, 158]\nclass = No'),
 Text(253.63636363636363, 155.3142857142857, 'City_Bangalore <= 0.5\ngini =
0.389\nsamples = 310\nvalue = [228, 82]\nclass = No'),
 Text(228.27272727272728, 93.18857142857144, 'gini = 0.463\nsamples = 132\nvalue
= [84, 48]\nclass = No'),
 Text(279.0, 93.18857142857144, 'Age <= 31.5\ngini = 0.309\nsamples = 178\nvalue
= [144, 34]\nclass = No'),
 Text(253.63636363636363, 31.062857142857126, 'gini = 0.346\nsamples = 99\nvalue
= [77, 22]\nclass = No'),
 Text(304.3636363636364, 31.062857142857126, 'gini = 0.258\nsamples = 79\nvalue
= [67, 12]\nclass = No'),
 Text(431.1818181818182, 155.3142857142857, 'City_Bangalore <= 0.5\ngini =
0.259\nsamples = 497\nvalue = [421, 76]\nclass = No'),
 Text(380.45454545454544, 93.18857142857144, 'Age <= 32.5\ngini = 0.069\nsamples
= 196\nvalue = [189, 7]\nclass = No'),
 Text(355.09090909090907, 31.062857142857126, 'gini = 0.11\nsamples = 120\nvalue
= [113, 7]\nclass = No'),
 Text(405.8181818181818, 31.062857142857126, 'gini = 0.0\nsamples = 76\nvalue =
[76, 0]\nclass = No'),
 Text(481.9090909090909, 93.18857142857144, 'Age <= 28.5\ngini = 0.353\nsamples
= 301\nvalue = [232, 69]\nclass = No'),
 Text(456.54545454545456, 31.062857142857126, 'gini = 0.474\nsamples = 83\nvalue
= [51, 32]\nclass = No'),
 Text(507.27272727272725, 31.062857142857126, 'gini = 0.282\nsamples =
218\nvalue = [181, 37]\nclass = No'),
 Text(507.27272727272725, 217.44, 'ExperienceInCurrentDomain <= 3.5\ngini =
0.47\nsamples = 247\nvalue = [154, 93]\nclass = No'),
 Text(481.9090909090909, 155.3142857142857, 'gini = 0.442\nsamples = 161\nvalue
= [108, 53]\nclass = No'),
 Text(532.6363636363636, 155.3142857142857, 'gini = 0.498\nsamples = 86\nvalue =
[46, 40]\nclass = No')]
```



[31]:
```python
#make prediction
y_pred_all = C5_all.predict(x_test_all)

y_actual_all = pd.Series(employee_test['LeaveOrNot'], name='Actual')
y_predicted_all = pd.Series(y_pred_all, name='Predicted')

#create confusion matrix
tab_all = pd.crosstab(y_actual_all, y_predicted_all)
tab_all['Total'] = tab_all.sum(axis =1)
tab_all.loc['Total'] = tab_all.sum()
#
tab_all
```

[31]:

Predicted	0	1	Total
Actual			
0	206	35	241
1	167	34	201
Total	373	69	442

[31]:

```

```

[32]:

```
print("Accuracy:",metrics.accuracy_score(y_test_all, y_pred_all))
print("Precision:",metrics.precision_score(y_test_all, y_pred_all))
print("Recall:",metrics.recall_score(y_test_all, y_pred_all))
```

```
Accuracy: 0.7414104882459313
Precision: 0.9829545454545454
Recall: 0.3793859649122807
```

Examine feature importance for C5_all model

[33]:

```
from sklearn.inspection import permutation_importance
from sklearn.datasets import load_boston
from matplotlib import pyplot as plt
```

[34]:

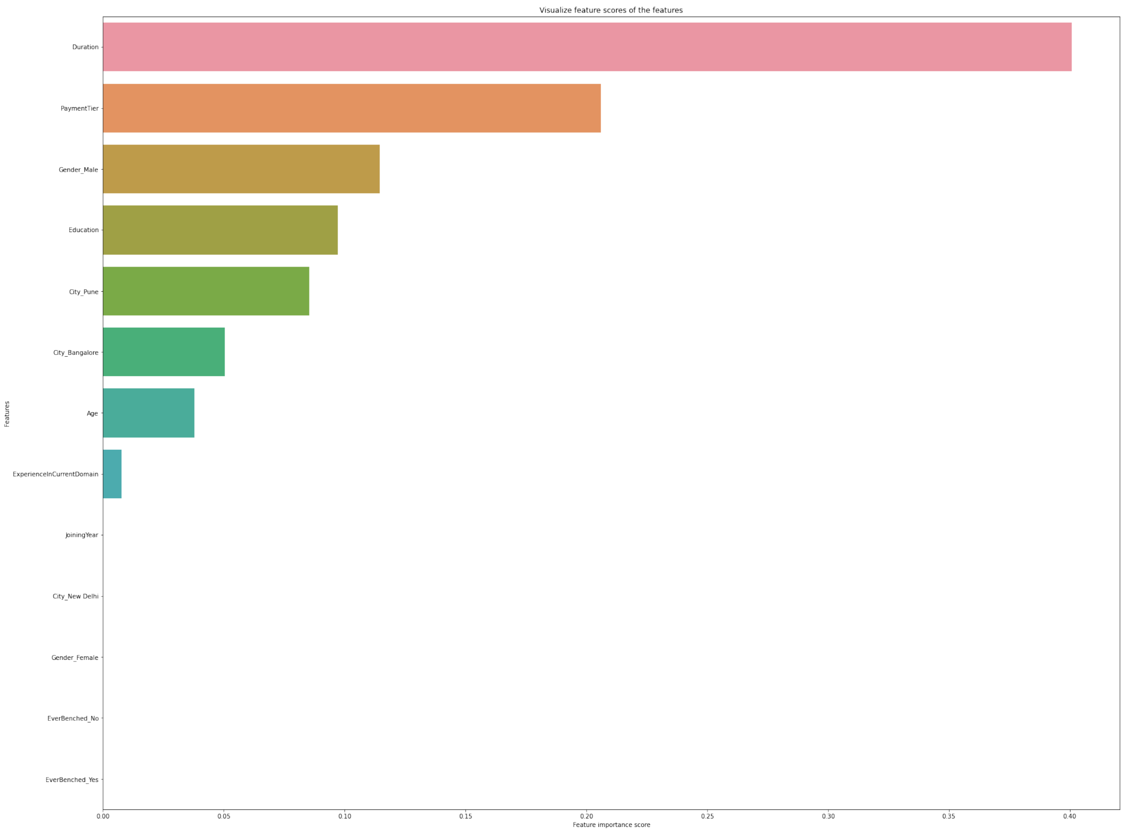
```
#Feature importance
feature_scores_all = pd.Series(C5_all.feature_importances_, index= x_all.
↪columns).sort_values(ascending=False)

feature_scores_all
```

[34]:

```
Duration                     0.400658
PaymentTier                  0.206009
Gender_Male                  0.114612
Education                    0.097214
City_Pune                    0.085358
City_Bangalore               0.050479
Age                          0.037873
ExperienceInCurrentDomain    0.007798
JoiningYear                  0.000000
City_New Delhi               0.000000
Gender_Female                0.000000
EverBenched_No               0.000000
EverBenched_Yes              0.000000
dtype: float64
```

[35]:

```
# Creating a seaborn bar plot
f, ax = plt.subplots(figsize=(30, 24))
ax = sns.barplot(x=feature_scores_all, y=feature_scores_all.index, data =␣
↪x_all[[]])
ax.set_title("Visualize feature scores of the features")
ax.set_yticklabels(feature_scores_all.index)
ax.set_xlabel("Feature importance score")
ax.set_ylabel("Features")
plt.show()
```



0.1.3 –>

0.1.4 –> It seems Duration, PaymentTier, Gender and City are a good candidate for a simplified model

0.1.5 *Data with Duration, PaymentTier, Gender and Education as predictor variables

[36]:
```
'Duration', 'PaymentTier', 'Gender', 'City'
```

[36]: ('Duration', 'PaymentTier', 'Gender', 'City')

[37]:
```
#---Training set---

x = employee_train[['Duration', 'PaymentTier', 'Gender', 'City']]
x = pd.get_dummies(x)
y = employee_train[['LeaveOrNot']]
x_names = ['Duration', 'PaymentTier', 'Gender', 'City']
x_names = x.columns.values
y_names = ["No", "Yes"]

#---Test set---
```

```python
x_test = employee_test[['Duration', 'PaymentTier', 'Gender', 'City']]
x_test = pd.get_dummies(x_test)
y_test = employee_test[['LeaveOrNot']]
x_test_names = x_test.columns
y_test_names = ["No", "Yes"]

x.head(2)
```

[37]:

```
      Duration  PaymentTier  Gender_Female  Gender_Male  City_Bangalore  \
2040         3            3              0            1               0
1872         4            3              0            1               1

      City_New Delhi  City_Pune
2040               0          1
1872               0          0
```

[38]:

```python
#Run C5.0 using entropy criterion
C5 = DecisionTreeClassifier(criterion = "gini", \
                                max_leaf_nodes = 10, \
                                min_samples_leaf= 75).fit(x,y)
export_graphviz (C5, out_file = 'C5.dot')

#predict income in training data set
y_train_pred = C5.predict(x)
y_train_pred
```

[38]:

```
array([0, 0, 1, …, 0, 0, 0], dtype=int64)
```

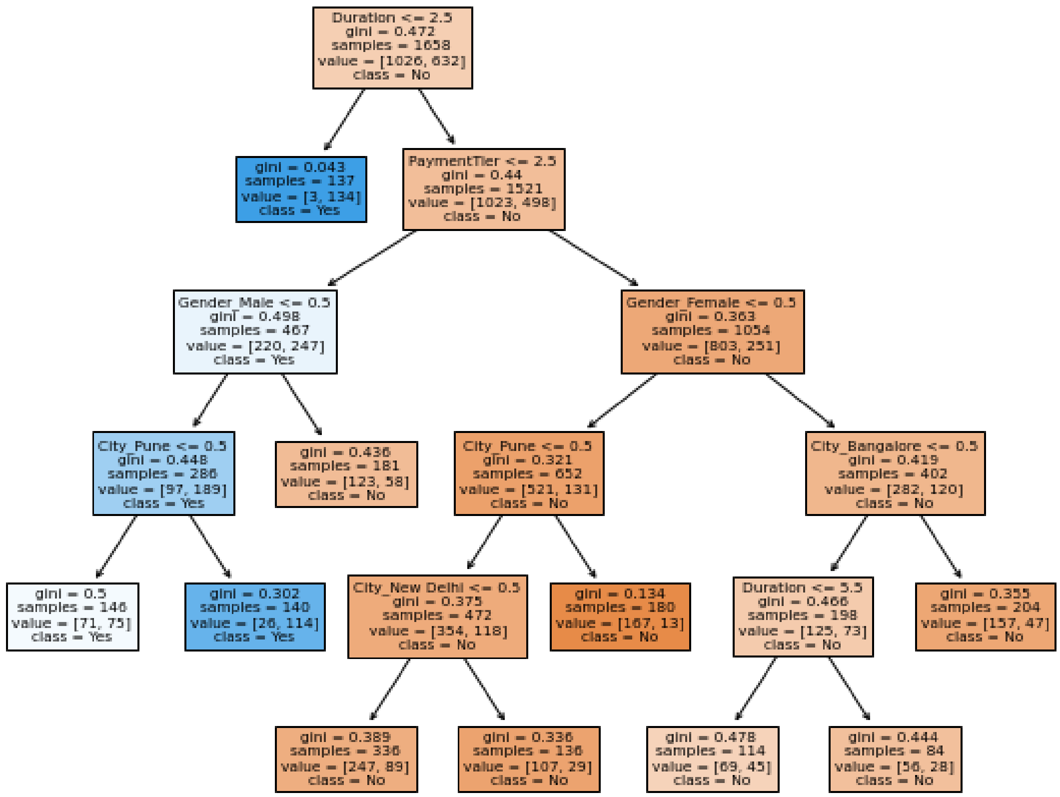
[39]:

```python
#Visualize the tree
plt.rcParams['figure.figsize'] = (10, 8)
plot_tree(C5, feature_names=x.columns.values, filled=True,
         class_names=y_names)
```

[39]:

```
[Text(209.25, 398.64, 'Duration <= 2.5\ngini = 0.472\nsamples = 1658\nvalue =
[1026, 632]\nclass = No'),
 Text(162.75, 326.15999999999997, 'gini = 0.043\nsamples = 137\nvalue = [3,
134]\nclass = Yes'),
 Text(255.75, 326.15999999999997, 'PaymentTier <= 2.5\ngini = 0.44\nsamples =
1521\nvalue = [1023, 498]\nclass = No'),
 Text(139.5, 253.67999999999998, 'Gender_Male <= 0.5\ngini = 0.498\nsamples =
467\nvalue = [220, 247]\nclass = Yes'),
 Text(93.0, 181.2, 'City_Pune <= 0.5\ngini = 0.448\nsamples = 286\nvalue = [97,
189]\nclass = Yes'),
 Text(46.5, 108.71999999999997, 'gini = 0.5\nsamples = 146\nvalue = [71,
75]\nclass = Yes'),
 Text(139.5, 108.71999999999997, 'gini = 0.302\nsamples = 140\nvalue = [26,
```

```
114]\nclass = Yes'),
 Text(186.0, 181.2, 'gini = 0.436\nsamples = 181\nvalue = [123, 58]\nclass =
No'),
 Text(372.0, 253.67999999999998, 'Gender_Female <= 0.5\ngini = 0.363\nsamples =
1054\nvalue = [803, 251]\nclass = No'),
 Text(279.0, 181.2, 'City_Pune <= 0.5\ngini = 0.321\nsamples = 652\nvalue =
[521, 131]\nclass = No'),
 Text(232.5, 108.71999999999997, 'City_New Delhi <= 0.5\ngini = 0.375\nsamples =
472\nvalue = [354, 118]\nclass = No'),
 Text(186.0, 36.23999999999995, 'gini = 0.389\nsamples = 336\nvalue = [247,
89]\nclass = No'),
 Text(279.0, 36.23999999999995, 'gini = 0.336\nsamples = 136\nvalue = [107,
29]\nclass = No'),
 Text(325.5, 108.71999999999997, 'gini = 0.134\nsamples = 180\nvalue = [167,
13]\nclass = No'),
 Text(465.0, 181.2, 'City_Bangalore <= 0.5\ngini = 0.419\nsamples = 402\nvalue =
[282, 120]\nclass = No'),
 Text(418.5, 108.71999999999997, 'Duration <= 5.5\ngini = 0.466\nsamples =
198\nvalue = [125, 73]\nclass = No'),
 Text(372.0, 36.23999999999995, 'gini = 0.478\nsamples = 114\nvalue = [69,
45]\nclass = No'),
 Text(465.0, 36.23999999999995, 'gini = 0.444\nsamples = 84\nvalue = [56,
28]\nclass = No'),
 Text(511.5, 108.71999999999997, 'gini = 0.355\nsamples = 204\nvalue = [157,
47]\nclass = No')]
```



[40]:
```python
#make prediction
y_pred = C5.predict(x_test)

y_actual = pd.Series(employee_test['LeaveOrNot'], name='Actual')
y_predicted = pd.Series(y_pred, name='Predicted')

#create confusion matrix
tab1 = pd.crosstab(y_actual, y_predicted)
tab1['Total'] = tab1.sum(axis =1)
tab1.loc['Total'] = tab1.sum()
tab1
```

[40]:

Predicted	0	1	Total
Actual			
0	184	57	241
1	150	51	201
Total	334	108	442

[41]:
```python
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

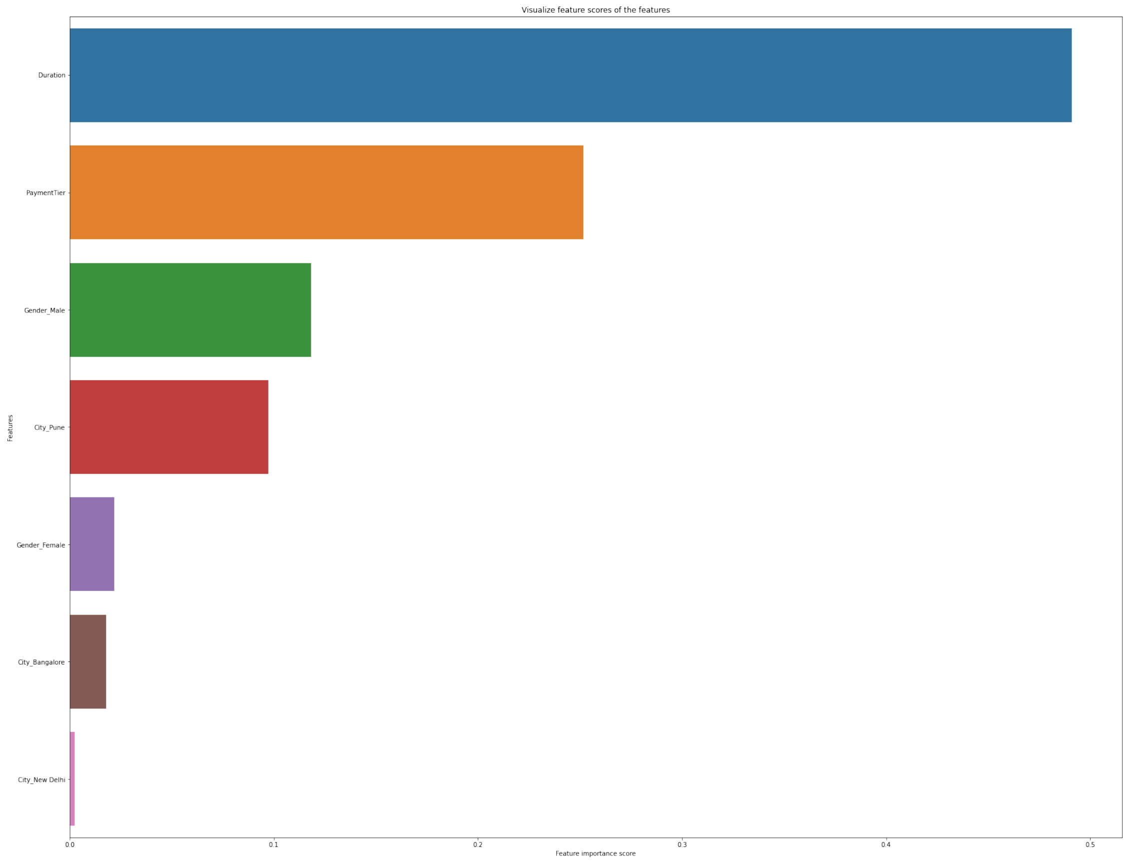
```
Accuracy: 0.7115732368896925
Precision: 0.7455197132616488
Recall: 0.45614035087719296
```

Examine feature importance for C5 model

[42]:
```python
#Feature importance
feature_scores = pd.Series(C5.feature_importances_, index= x.columns).
↪sort_values(ascending=False)

feature_scores
```

[42]:
```
Duration          0.491106
PaymentTier       0.251653
Gender_Male       0.118136
City_Pune         0.097273
Gender_Female     0.021782
City_Bangalore    0.017675
City_New Delhi    0.002375
dtype: float64
```

[43]:
```python
# Creating a seaborn bar plot
f, ax = plt.subplots(figsize=(30, 24))
ax = sns.barplot(x=feature_scores, y=feature_scores.index, data = x[[]])
ax.set_title("Visualize feature scores of the features")
ax.set_yticklabels(feature_scores.index)
ax.set_xlabel("Feature importance score")
ax.set_ylabel("Features")
plt.show()
```


Visualize feature scores of the features
Duration
PaymentTier
Gender_Male
City_Pune
Gender_Female
City_Bangalore
City_New Delhi
Features
0.0
0.1
0.2
0.3
0.4
0.5
Feature importance score

2.0-unp-random-forest

April 17, 2022

[20]:
```
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn import tree
```

[21]:
```
employee = pd.read_csv("../data/employee_cleaned.csv")
employee.head()
```

[21]:
```
   Education  JoiningYear       City  PaymentTier  Age  Gender EverBenched  \
0  Bachelors         2017  Bangalore            3   34    Male          No
1  Bachelors         2013       Pune            1   28  Female          No
2  Bachelors         2014  New Delhi            3   38  Female          No
3    Masters         2016  Bangalore            3   27    Male          No
4    Masters         2017       Pune            3   24    Male         Yes

   ExperienceInCurrentDomain  LeaveOrNot  Duration
0                          0           0         3
1                          3           1         7
2                          2           0         6
3                          5           1         4
4                          2           1         3
```

[22]:
```
#employee['Education'].replace({'PHD': 'a', 'Masters': 'b', 'Bachelors': 'c'},␣
↪inplace=True)
employee['Education'].replace(to_replace = ['PHD', 'Masters', 'Bachelors'],␣
↪value = ['3', '2', '1'], inplace = True)
employee['Education'] = employee['Education'].astype('int64')
employee.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

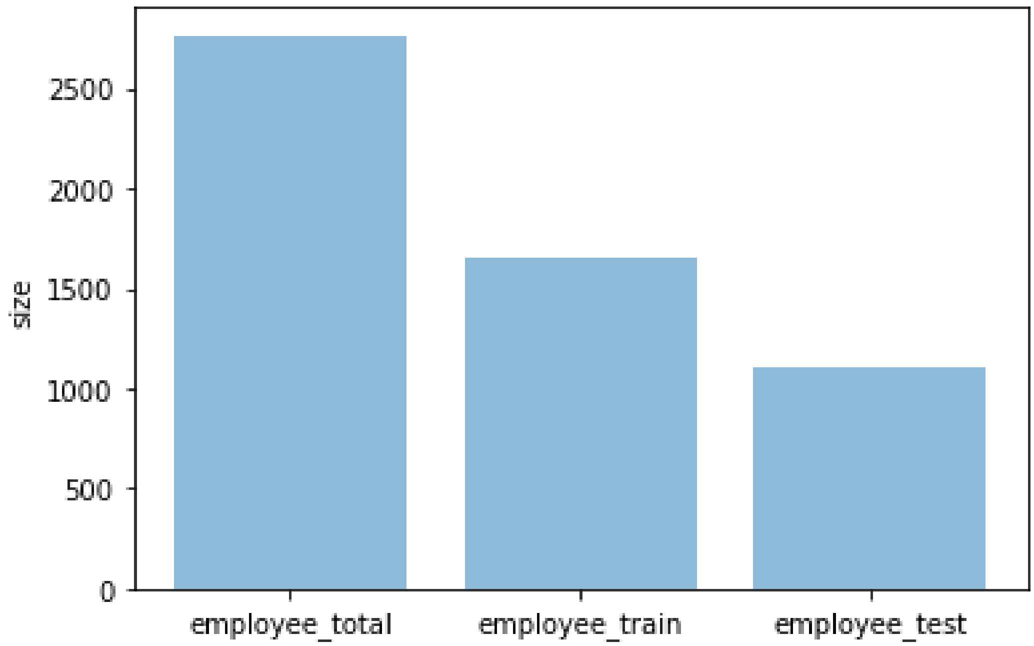
```
RangeIndex: 2764 entries, 0 to 2763
Data columns (total 10 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Education                  2764 non-null   int64
 1   JoiningYear                2764 non-null   int64
 2   City                       2764 non-null   object
 3   PaymentTier                2764 non-null   int64
 4   Age                        2764 non-null   int64
 5   Gender                     2764 non-null   object
 6   EverBenched                2764 non-null   object
 7   ExperienceInCurrentDomain  2764 non-null   int64
 8   LeaveOrNot                 2764 non-null   int64
 9   Duration                   2764 non-null   int64
dtypes: int64(7), object(3)
memory usage: 216.1+ KB
```

0.1 Split the data into training and test sets

[23]:
```
employee_train, employee_test = train_test_split(employee, test_size = 0.40,␣
 ↪random_state = 7)
print(employee.shape)
print(employee_train.shape)
print(employee_test.shape)
```

```
(2764, 10)
(1658, 10)
(1106, 10)
```

[24]:
```
portion = ('employee_total', 'employee_train', 'employee_test')
y_pos = np.arange(len(portion))
size = [2764,1658,1106]

plt.bar(y_pos, size, align='center', alpha=0.5)
plt.xticks(y_pos, portion)
plt.ylabel('size')

plt.show()
```



0.1.1 Separate data frames for interested predictor variables and response variable

0.1.2 *Data with all predictor variables

[25]:
```
#---Training set---

x_all = employee_train.drop(['LeaveOrNot'], axis= 1)
x_all = pd.get_dummies(x_all)
y_all = employee_train[['LeaveOrNot']]
y_names_all = ["No", "Yes"]

#---Test set---

x_test_all = employee_test.drop(['LeaveOrNot'], axis= 1)
x_test_all = pd.get_dummies(x_test_all)
y_test_all = employee_test[['LeaveOrNot']]

x_all.head(2)
```

[25]:
```
      Education  JoiningYear  PaymentTier  Age  ExperienceInCurrentDomain  \
2040          1         2017            3   33                          5
1872          1         2016            3   40                          0

      Duration  City_Bangalore  City_New Delhi  City_Pune  Gender_Female  \
2040         3               0               0          1              0
1872         4               1               0          0              0
```

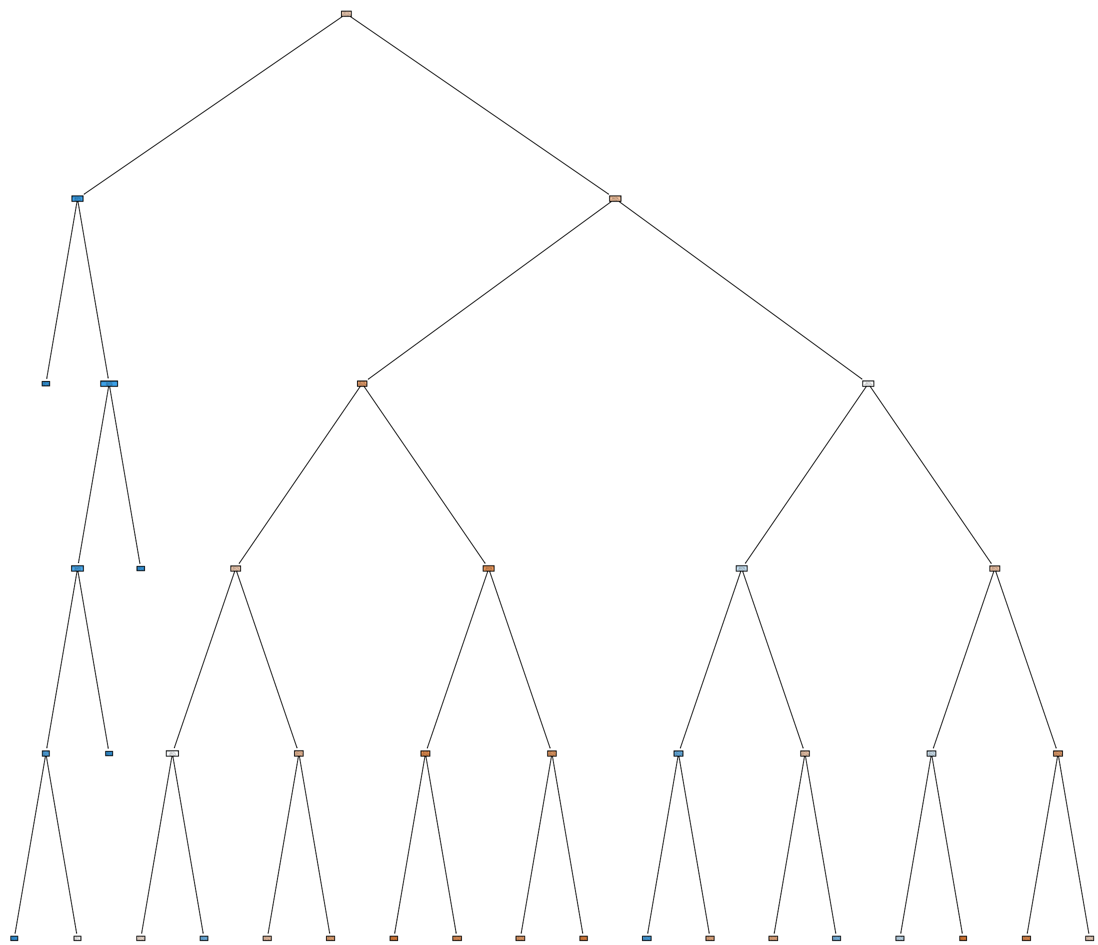
```
      Gender_Male  EverBenched_No  EverBenched_Yes
2040            1               0                1
1872            1               1                0
```

[26]:
```
#Change response variable to one-dimension array
rfy_all = np.ravel(y_all)
```

[27]:
```
#The n_estimators= 100, criterion  = "gini" id default (don't have to specufy)--
#--set max_depth to limit the depth of the tree, or limit branches
#--Set random_state for reproducible results
rf_all = RandomForestClassifier(n_estimators = 100, \
                              criterion = "gini", max_depth=5, random_state =␣
 ↪42).fit(x_all,rfy_all)

y_train_pred_all = rf_all.predict(x_all)
y_train_pred_all
```

[27]:
```
array([0, 0, 0, …, 0, 0, 0], dtype=int64)
```

[28]:
```
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=400)
#use .estimator[] to specify individual tree
tree.plot_tree(rf_all.estimators_[0],
               feature_names = x_all.columns.values,
               class_names=y_names_all,
               filled = True);
fig.savefig('rf_all_individualtree.png')
```



[29]:
```
#Make prediction
y_pred_all = rf_all.predict(x_test_all)

#count number of predict for each class
unique, counts = np.unique(y_pred_all, return_counts=True)
dict(zip(unique, counts))
```

[29]: {0: 818, 1: 288}

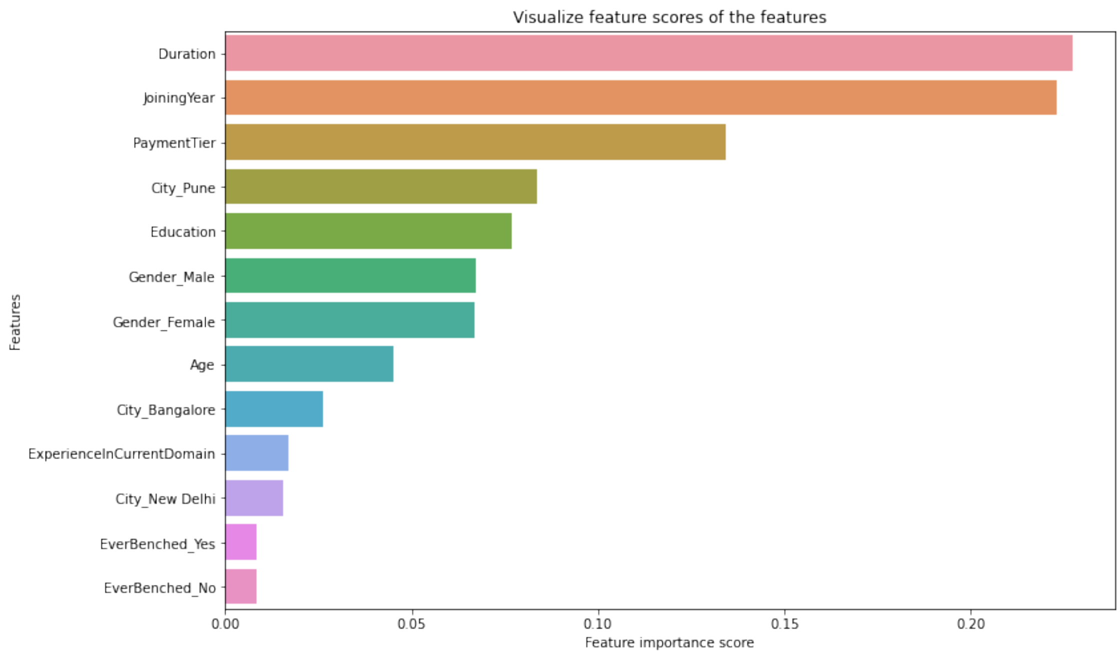
[30]:
```
print("Accuracy:",metrics.accuracy_score(y_test_all, y_pred_all))
print("Precision:",metrics.precision_score(y_test_all, y_pred_all))
print("Recall:",metrics.recall_score(y_test_all, y_pred_all))
```

```
Accuracy: 0.7631103074141049
Precision: 0.8368055555555556
Recall: 0.5285087719298246
```

[31]:
```
#Feature importance
feature_scores_all = pd.Series(rf_all.feature_importances_, index= x_all.
↪columns).sort_values(ascending=False)

feature_scores_all
```

[31]:
```
Duration                     0.227348
JoiningYear                  0.223179
PaymentTier                  0.134139
City_Pune                    0.083759
Education                    0.076733
Gender_Male                  0.067233
Gender_Female                0.066949
Age                          0.045255
City_Bangalore               0.026260
ExperienceInCurrentDomain    0.016981
City_New Delhi               0.015509
EverBenched_Yes              0.008379
EverBenched_No               0.008276
dtype: float64
```

[32]:
```
# Creating a seaborn bar plot
f, ax = plt.subplots(figsize=(12, 8))
ax = sns.barplot(x=feature_scores_all, y=feature_scores_all.index, data =␣
↪x_all[[]])
ax.set_title("Visualize feature scores of the features")
ax.set_yticklabels(feature_scores_all.index)
ax.set_xlabel("Feature importance score")
ax.set_ylabel("Features")
plt.show()
```



0.1.3 –> It seems Duration, PaymentTier, Education and City are a good candidate for a simplified model

0.1.4 *Data with Duration, PaymentTier, Education and City as predictor variables

[33]:
```
#---Training set---

x = employee_train[['Duration', 'PaymentTier', 'Gender', 'City']]
x = pd.get_dummies(x)
y = employee_train[['LeaveOrNot']]
x_names = ['Duration', 'PaymentTier', 'Gender']
x_names = x.columns.values
y_names = ["No", "Yes"]

#---Test set---

x_test = employee_test[['Duration', 'PaymentTier', 'Gender', 'City']]
x_test = pd.get_dummies(x_test)
y_test = employee_test[['LeaveOrNot']]
x_test_names = x_test.columns
y_test_names = ["No", "Yes"]

x.head(2)
```

[33]:
```
      Duration  PaymentTier  Gender_Female  Gender_Male  City_Bangalore  \
2040         3            3              0            1               0
```

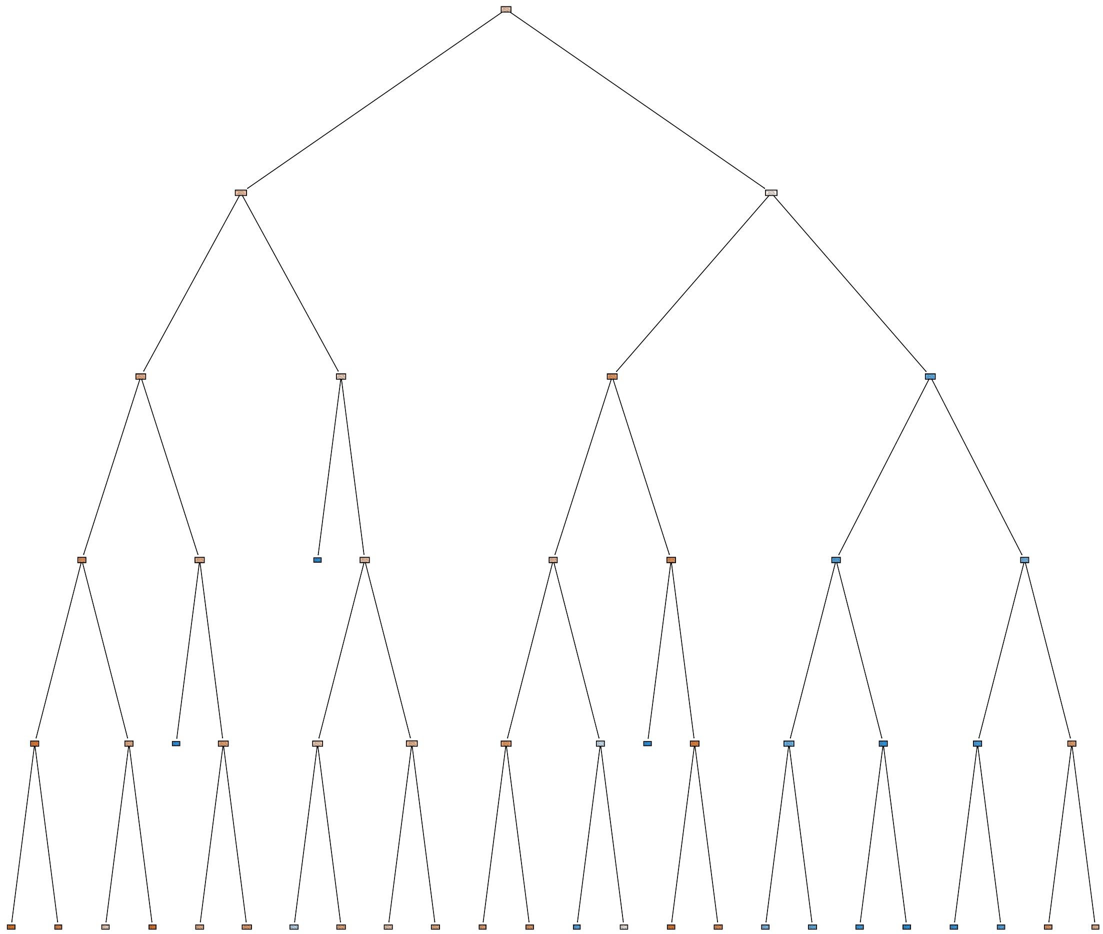
```
1872         4            3              0            1               1

      City_New Delhi  City_Pune
2040               0          1
1872               0          0
```

[34]:
```
#Change response variable to one-dimension array
rfy = np.ravel(y)
```

[35]:
```
#The n_estimators= 100, criterion  = "gini" id default (don't have to specufy)--
#--set max_depth to limit the depth of the tree, or limit branches
#--Set random_state for reproducible results
rf = RandomForestClassifier(n_estimators = 100, \
                              criterion = "gini", max_depth=5, random_state =␣
 ↪42).fit(x,rfy)

y_train_pred = rf.predict(x)
y_train_pred
```

[35]:
```
array([0, 0, 0, …, 0, 0, 0], dtype=int64)
```

[36]:
```
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=400)
#use .estimator[] to specify individual tree
tree.plot_tree(rf.estimators_[0],
               feature_names = x.columns.values,
               class_names=y_names,
               filled = True);
fig.savefig('rf_individualtree.png')
```



[37]:
```
#Make prediction
y_pred2 = rf.predict(x_test)

#count number of predict for each class
unique, counts = np.unique(y_pred2, return_counts=True)
dict(zip(unique, counts))
```

[37]: {0: 808, 1: 298}

[38]:
```
#calculate the Evaluation measure based on the contingency above
TN, FP, FN, TP = confusion_matrix(y_test, y_pred2).ravel()
Specificity2 = TN / (TN+FP)
Accuracy2 = metrics.accuracy_score(y_test, y_pred2)
Precision2 = metrics.precision_score(y_test, y_pred2)
Recall2 = metrics.recall_score(y_test, y_pred2)
```

```
F1_Score2 = metrics.f1_score(y_test, y_pred2)
ErrorRate2 = 1-Accuracy2
F2_Score2 = (5*Precision2*Recall2)/((4*Precision2)+Recall2)
F0point5_Score2 = (1.25*Precision2*Recall2)/((0.25*Precision2)+Recall2)

print("Accuracy:", Accuracy2)
print("Precision:", Precision2)
print("Recall:", Recall2 )
```

```
Accuracy: 0.7703435804701627
Precision: 0.8389261744966443
Recall: 0.5482456140350878
```