Logistic Regression, D-Tree and Random Forest Model - Case Study

September 10, 2023

This case study deals with applying logistic regression, Decision Tree Model and Random Forest Model on the Employee Attrition Dataset.

[6]:
```
# Import libraries
import numpy as np
import pandas as pd
import seaborn as sns
import sklearn
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import␣
↪accuracy_score,f1_score,recall_score,precision_score, confusion_matrix,␣
↪classification_report
```

[7]:
```
import warnings
warnings.filterwarnings('ignore')
```

[8]:
```
#import HRAnalytics dataset and look at the first five rows.
Dataset = pd.read_csv("HR_Analytics.csv.csv")
Dataset.head()
```

[8]:
```
   Age Attrition     BusinessTravel  DailyRate              Department  \
0   41       Yes      Travel_Rarely       1102                   Sales
1   49        No  Travel_Frequently        279  Research & Development
2   37       Yes      Travel_Rarely       1373  Research & Development
3   33        No  Travel_Frequently       1392  Research & Development
4   27        No      Travel_Rarely        591  Research & Development

   DistanceFromHome  Education EducationField  EmployeeCount  EmployeeNumber  \
0                 1          2  Life Sciences              1               1
1                 8          1  Life Sciences              1               2
2                 2          2          Other              1               4
3                 3          4  Life Sciences              1               5
```

```
4                  2           1        Medical             1               7

   …  RelationshipSatisfaction StandardHours  StockOptionLevel  \
0  …                         1            80                 0
1  …                         4            80                 1
2  …                         2            80                 0
3  …                         3            80                 0
4  …                         4            80                 1

   TotalWorkingYears  TrainingTimesLastYear WorkLifeBalance  YearsAtCompany  \
0                  8                      0               1               6
1                 10                      3               3              10
2                  7                      3               3               0
3                  8                      3               3               8
4                  6                      3               3               2

  YearsInCurrentRole  YearsSinceLastPromotion  YearsWithCurrManager
0                  4                        0                     5
1                  7                        1                     7
2                  0                        0                     0
3                  7                        3                     0
4                  2                        2                     2

[5 rows x 35 columns]
```

[9]:

```
Dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Age                       1470 non-null   int64
 1   Attrition                 1470 non-null   object
 2   BusinessTravel            1470 non-null   object
 3   DailyRate                 1470 non-null   int64
 4   Department                1470 non-null   object
 5   DistanceFromHome          1470 non-null   int64
 6   Education                 1470 non-null   int64
 7   EducationField            1470 non-null   object
 8   EmployeeCount             1470 non-null   int64
 9   EmployeeNumber            1470 non-null   int64
 10  EnvironmentSatisfaction   1470 non-null   int64
 11  Gender                    1470 non-null   object
 12  HourlyRate                1470 non-null   int64
 13  JobInvolvement            1470 non-null   int64
 14  JobLevel                  1470 non-null   int64
 15  JobRole                   1470 non-null   object
```

```
 16  JobSatisfaction           1470 non-null   int64
 17  MaritalStatus             1470 non-null   object
 18  MonthlyIncome             1470 non-null   int64
 19  MonthlyRate               1470 non-null   int64
 20  NumCompaniesWorked        1470 non-null   int64
 21  Over18                    1470 non-null   object
 22  OverTime                  1470 non-null   object
 23  PercentSalaryHike         1470 non-null   int64
 24  PerformanceRating         1470 non-null   int64
 25  RelationshipSatisfaction  1470 non-null   int64
 26  StandardHours             1470 non-null   int64
 27  StockOptionLevel          1470 non-null   int64
 28  TotalWorkingYears         1470 non-null   int64
 29  TrainingTimesLastYear     1470 non-null   int64
 30  WorkLifeBalance           1470 non-null   int64
 31  YearsAtCompany            1470 non-null   int64
 32  YearsInCurrentRole        1470 non-null   int64
 33  YearsSinceLastPromotion   1470 non-null   int64
 34  YearsWithCurrManager      1470 non-null   int64
dtypes: int64(26), object(9)
memory usage: 402.1+ KB
```

[10]:
```
#Attrition BusinessTravel Department EducationField Gender JobRole Over18␣
 ↪OverTime
Dataset['Attrition'].unique()
```

[10]:
```
array(['Yes', 'No'], dtype=object)
```

[11]:
```
Dataset['BusinessTravel'].unique()
```

[11]:
```
array(['Travel_Rarely', 'Travel_Frequently', 'Non-Travel'], dtype=object)
```

[12]:
```
Dataset['Department'].unique()
```

[12]:
```
array(['Sales', 'Research & Development', 'Human Resources'], dtype=object)
```

[13]:
```
Dataset['EducationField'].unique()
```

[13]:
```
array(['Life Sciences', 'Other', 'Medical', 'Marketing',
       'Technical Degree', 'Human Resources'], dtype=object)
```

[14]:
```
Dataset['Gender'].unique()
```

[14]:
```
array(['Female', 'Male'], dtype=object)
```

[15]:
```
Dataset['JobRole'].unique()
```

[15]:
```
array(['Sales Executive', 'Research Scientist', 'Laboratory Technician',
       'Manufacturing Director', 'Healthcare Representative', 'Manager',
       'Sales Representative', 'Research Director', 'Human Resources'],
      dtype=object)
```

[16]:
```python
Dataset['Over18'].unique() # OverTime
```

[16]:
```
array(['Y'], dtype=object)
```

[17]:
```python
Dataset['OverTime'].unique()
```

[17]:
```
array(['Yes', 'No'], dtype=object)
```

[18]:
```python
Dataset['EmployeeNumber'].unique()
```

[18]:
```
array([   1,    2,    4, …, 2064, 2065, 2068], dtype=int64)
```

[19]:
```python
Dataset['EmployeeCount'].unique()
```

[19]:
```
array([1], dtype=int64)
```

[20]:
```python
Dataset['StandardHours'].unique()
```

[20]:
```
array([80], dtype=int64)
```

When we take a look at the unique values in the categorical columns, "Over18" column will not contribute to our prediction, since it has only one value "Y". So it would be better to remove that column alone.

Also in the numerical columns, 1. "EmployeeNumber" column looks like employee id data, so that column can also be removed. 2. "EmployeeCount" column have only one value (1), so that can also be removed. 3. "StandardHours column too have only one value 80 hours. So, the above mentioned numerical columns can be removed, since it is in no way going make any difference in the data interpretation and visualization.

[21]:
```python
Dataset.shape
```

[21]:
```
(1470, 35)
```

[22]:
```python
cols_to_remove = ['Over18','EmployeeCount','StandardHours','EmployeeNumber']
Dataset = Dataset.drop(cols_to_remove,axis = 1)
```

[23]:
```python
Dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 31 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Age                       1470 non-null   int64
```

```
 1   Attrition                 1470 non-null   object
 2   BusinessTravel            1470 non-null   object
 3   DailyRate                 1470 non-null   int64
 4   Department                1470 non-null   object
 5   DistanceFromHome          1470 non-null   int64
 6   Education                 1470 non-null   int64
 7   EducationField            1470 non-null   object
 8   EnvironmentSatisfaction   1470 non-null   int64
 9   Gender                    1470 non-null   object
 10  HourlyRate                1470 non-null   int64
 11  JobInvolvement            1470 non-null   int64
 12  JobLevel                  1470 non-null   int64
 13  JobRole                   1470 non-null   object
 14  JobSatisfaction           1470 non-null   int64
 15  MaritalStatus             1470 non-null   object
 16  MonthlyIncome             1470 non-null   int64
 17  MonthlyRate               1470 non-null   int64
 18  NumCompaniesWorked        1470 non-null   int64
 19  OverTime                  1470 non-null   object
 20  PercentSalaryHike         1470 non-null   int64
 21  PerformanceRating         1470 non-null   int64
 22  RelationshipSatisfaction  1470 non-null   int64
 23  StockOptionLevel          1470 non-null   int64
 24  TotalWorkingYears         1470 non-null   int64
 25  TrainingTimesLastYear     1470 non-null   int64
 26  WorkLifeBalance           1470 non-null   int64
 27  YearsAtCompany            1470 non-null   int64
 28  YearsInCurrentRole        1470 non-null   int64
 29  YearsSinceLastPromotion   1470 non-null   int64
 30  YearsWithCurrManager      1470 non-null   int64
dtypes: int64(23), object(8)
memory usage: 356.1+ KB
```

[24]:
```
#Check for NULL values.
Dataset.isna().sum()
```

[24]:
```
Age                        0
Attrition                  0
BusinessTravel             0
DailyRate                  0
Department                 0
DistanceFromHome           0
Education                  0
EducationField             0
EnvironmentSatisfaction    0
Gender                     0
HourlyRate                 0
```

```
JobInvolvement              0
JobLevel                    0
JobRole                     0
JobSatisfaction             0
MaritalStatus               0
MonthlyIncome               0
MonthlyRate                 0
NumCompaniesWorked          0
OverTime                    0
PercentSalaryHike           0
PerformanceRating           0
RelationshipSatisfaction    0
StockOptionLevel            0
TotalWorkingYears           0
TrainingTimesLastYear       0
WorkLifeBalance             0
YearsAtCompany              0
YearsInCurrentRole          0
YearsSinceLastPromotion     0
YearsWithCurrManager        0
dtype: int64
```

Data Visualisation

Now let us plot the values in the columns to check the frequency of the data distribution.

With respect to this Dataset, since we are going to analyse the possible factors for Employee Attrition, let us filter the data with Attrition = Yes for Data Visualization purpose.

[25]:
```
empAttr=Dataset.loc[Dataset.Attrition == 'Yes']
```

[26]:
```
empAttr.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 237 entries, 0 to 1461
Data columns (total 31 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Age                       237 non-null    int64
 1   Attrition                 237 non-null    object
 2   BusinessTravel            237 non-null    object
 3   DailyRate                 237 non-null    int64
 4   Department                237 non-null    object
 5   DistanceFromHome          237 non-null    int64
 6   Education                 237 non-null    int64
 7   EducationField            237 non-null    object
 8   EnvironmentSatisfaction   237 non-null    int64
 9   Gender                    237 non-null    object
 10  HourlyRate                237 non-null    int64
```

```
 11  JobInvolvement            237 non-null    int64
 12  JobLevel                  237 non-null    int64
 13  JobRole                   237 non-null    object
 14  JobSatisfaction           237 non-null    int64
 15  MaritalStatus             237 non-null    object
 16  MonthlyIncome             237 non-null    int64
 17  MonthlyRate               237 non-null    int64
 18  NumCompaniesWorked        237 non-null    int64
 19  OverTime                  237 non-null    object
 20  PercentSalaryHike         237 non-null    int64
 21  PerformanceRating         237 non-null    int64
 22  RelationshipSatisfaction  237 non-null    int64
 23  StockOptionLevel          237 non-null    int64
 24  TotalWorkingYears         237 non-null    int64
 25  TrainingTimesLastYear     237 non-null    int64
 26  WorkLifeBalance           237 non-null    int64
 27  YearsAtCompany            237 non-null    int64
 28  YearsInCurrentRole        237 non-null    int64
 29  YearsSinceLastPromotion   237 non-null    int64
 30  YearsWithCurrManager      237 non-null    int64
dtypes: int64(23), object(8)
memory usage: 59.2+ KB
```

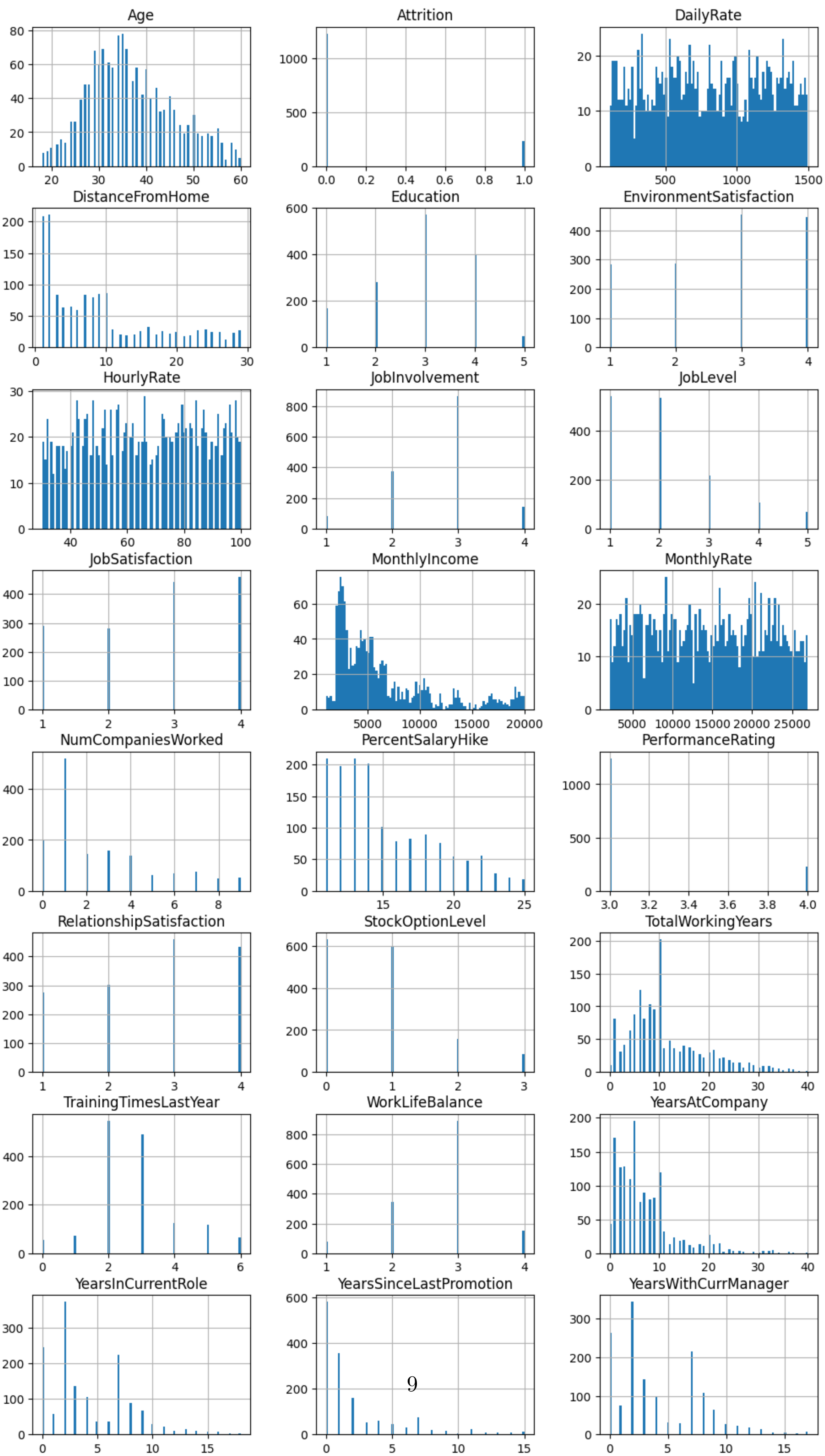
[27]:
```
Dataset['Attrition'] = Dataset['Attrition'].replace({'Yes': 1, 'No': 0})
Dataset.head(10)
```

[27]:
```
   Age  Attrition     BusinessTravel  DailyRate              Department  \
0   41          1      Travel_Rarely       1102                   Sales
1   49          0  Travel_Frequently        279  Research & Development
2   37          1      Travel_Rarely       1373  Research & Development
3   33          0  Travel_Frequently       1392  Research & Development
4   27          0      Travel_Rarely        591  Research & Development
5   32          0  Travel_Frequently       1005  Research & Development
6   59          0      Travel_Rarely       1324  Research & Development
7   30          0      Travel_Rarely       1358  Research & Development
8   38          0  Travel_Frequently        216  Research & Development
9   36          0      Travel_Rarely       1299  Research & Development

   DistanceFromHome  Education EducationField  EnvironmentSatisfaction  \
0                 1          2  Life Sciences                        2
1                 8          1  Life Sciences                        3
2                 2          2          Other                        4
3                 3          4  Life Sciences                        4
4                 2          1        Medical                        1
5                 2          2  Life Sciences                        4
6                 3          3        Medical                        3
7                24          1  Life Sciences                        4
```

```
8               23          3  Life Sciences                        4
9               27          3        Medical                        3

   Gender  …  PerformanceRating  RelationshipSatisfaction  StockOptionLevel  \
0  Female  …                  3                         1                 0
1    Male  …                  4                         4                 1
2    Male  …                  3                         2                 0
3  Female  …                  3                         3                 0
4    Male  …                  3                         4                 1
5    Male  …                  3                         3                 0
6  Female  …                  4                         1                 3
7    Male  …                  4                         2                 1
8    Male  …                  4                         2                 0
9    Male  …                  3                         2                 2

  TotalWorkingYears  TrainingTimesLastYear WorkLifeBalance  YearsAtCompany  \
0                 8                      0               1               6
1                10                      3               3              10
2                 7                      3               3               0
3                 8                      3               3               8
4                 6                      3               3               2
5                 8                      2               2               7
6                12                      3               2               1
7                 1                      2               3               1
8                10                      2               3               9
9                17                      3               2               7

   YearsInCurrentRole  YearsSinceLastPromotion YearsWithCurrManager
0                   4                        0                    5
1                   7                        1                    7
2                   0                        0                    0
3                   7                        3                    0
4                   2                        2                    2
5                   7                        3                    6
6                   0                        0                    0
7                   0                        0                    0
8                   7                        1                    8
9                   7                        7                    7

[10 rows x 31 columns]
```

[28]:

```
Dataset.hist(stacked = False,bins = 100, figsize=(12,30),layout=(11,3))
plt.show()
```


Age
Attrition
DailyRate
DistanceFromHome
Education
EnvironmentSatisfaction
HourlyRate
JobInvolvement
JobLevel
JobSatisfaction
MonthlyIncome
MonthlyRate
NumCompaniesWorked
PercentSalaryHike
PerformanceRating
RelationshipSatisfaction
StockOptionLevel
TotalWorkingYears
TrainingTimesLastYear
WorkLifeBalance
YearsAtCompany
YearsInCurrentRole
YearsSinceLastPromotion
YearsWithCurrManager

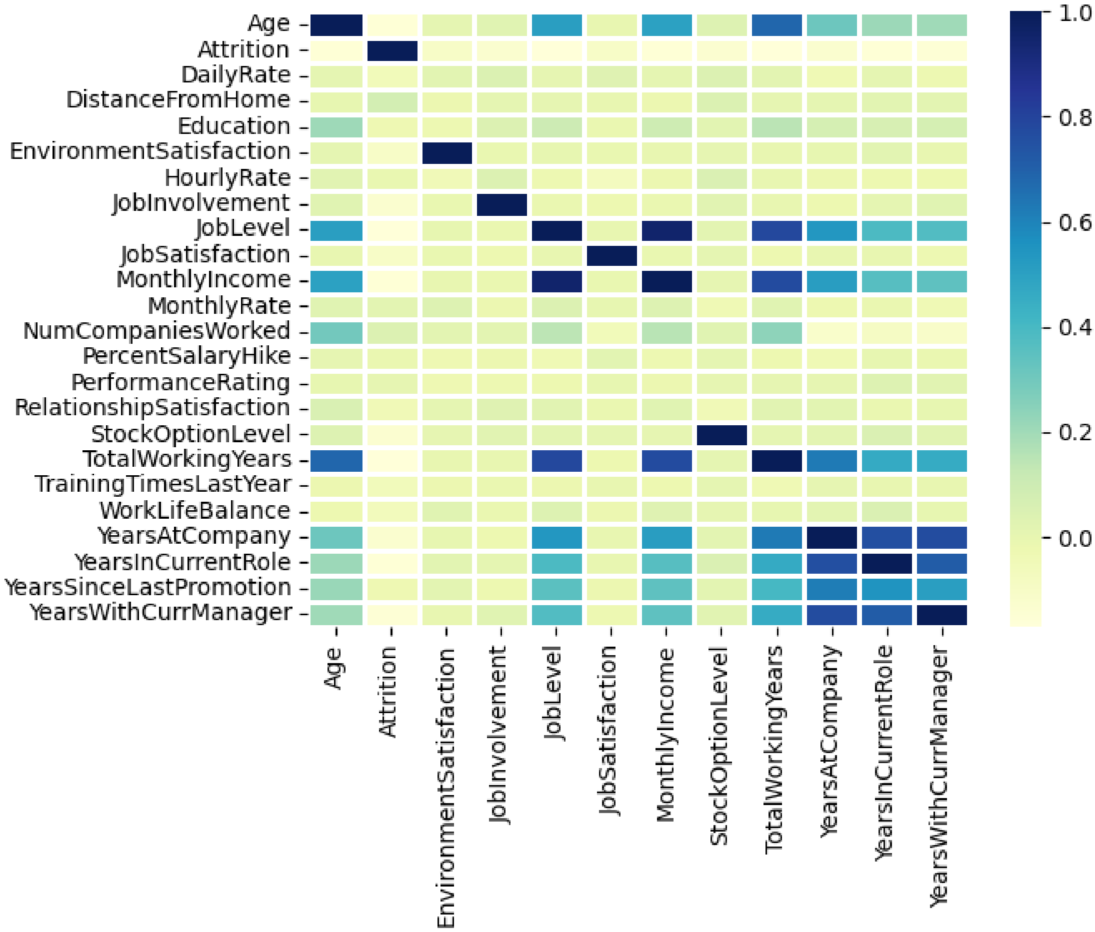
[30]:
```
#Dataset['Attrition'].dtypefor colname in WaterQuality.columns:
Dataset['Attrition'] = Dataset['Attrition'].astype('int64')


#Dataset.corr(numeric_only=True)
Dataset.corr(numeric_only=True)['Attrition']
```

[30]:
```
Age                        -0.159205
Attrition                   1.000000
DailyRate                  -0.056652
DistanceFromHome            0.077924
Education                  -0.031373
EnvironmentSatisfaction    -0.103369
HourlyRate                 -0.006846
JobInvolvement             -0.130016
JobLevel                   -0.169105
JobSatisfaction            -0.103481
MonthlyIncome              -0.159840
MonthlyRate                 0.015170
NumCompaniesWorked          0.043494
PercentSalaryHike          -0.013478
PerformanceRating           0.002889
RelationshipSatisfaction   -0.045872
StockOptionLevel           -0.137145
TotalWorkingYears          -0.171063
TrainingTimesLastYear      -0.059478
WorkLifeBalance            -0.063939
YearsAtCompany             -0.134392
YearsInCurrentRole         -0.160545
YearsSinceLastPromotion    -0.033019
YearsWithCurrManager       -0.156199
Name: Attrition, dtype: float64
```

Let us see the correlation for all the numerical columns.

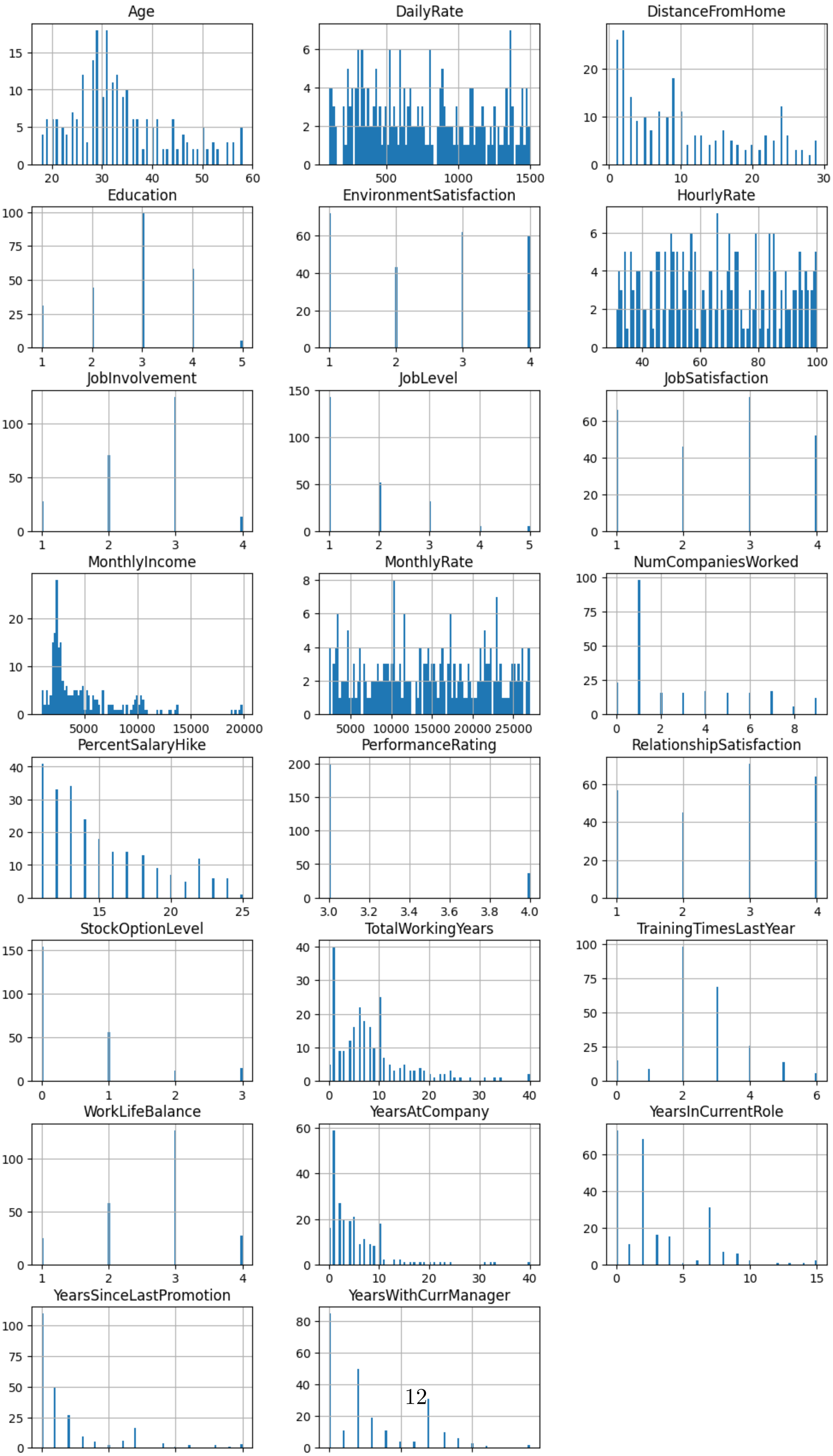
[33]:
```
#Numerical  Columns
#Checking correlation between numerical columns.
numCols =␣
↪['Age','Attrition','EnvironmentSatisfaction','JobInvolvement','JobLevel','JobSatisfaction',
    ␣
↪'StockOptionLevel','TotalWorkingYears','YearsAtCompany','YearsInCurrentRole','YearsWithCurr

sns.heatmap(Dataset.corr(numeric_only=True)[numCols][numCols], annot=None,␣
↪cmap='YlGnBu',linewidths=1)
```

[33]: <Axes: >



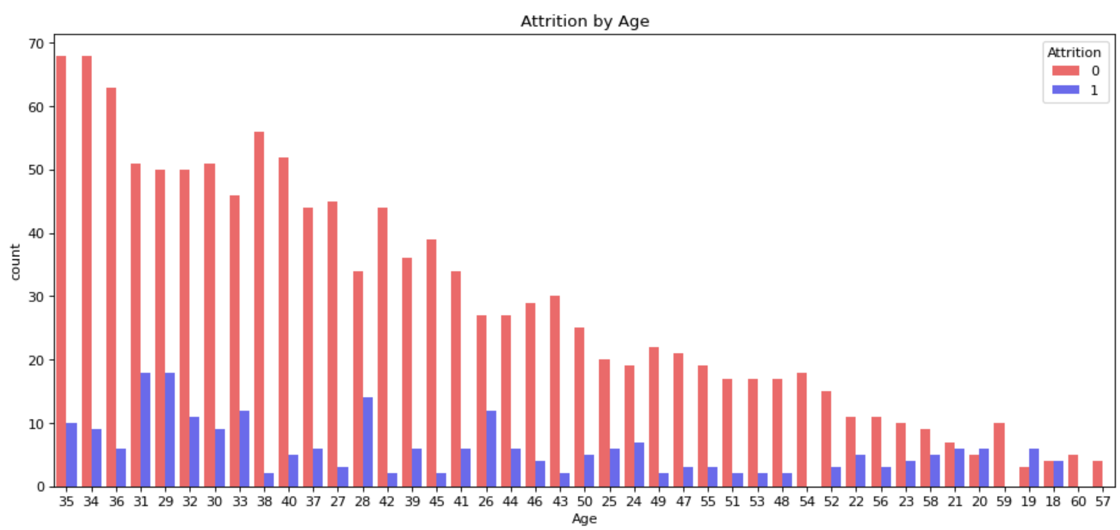
Looking at the correlation matrix, we could come to a conclusion that very few variables have correlation and that too a positive weak correlation.

[34]:
```
empAttr.hist(stacked = False,bins = 100, figsize=(12,30),layout=(11,3))
plt.show()
```


Age
DailyRate
DistanceFromHome
Education
EnvironmentSatisfaction
HourlyRate
JobInvolvement
JobLevel
JobSatisfaction
MonthlyIncome
MonthlyRate
NumCompaniesWorked
PercentSalaryHike
PerformanceRating
RelationshipSatisfaction
StockOptionLevel
TotalWorkingYears
TrainingTimesLastYear
WorkLifeBalance
YearsAtCompany
YearsInCurrentRole
YearsSinceLastPromotion
YearsWithCurrManager

Based on the data distribution in the Attrition Dataset, following are the key observations. 1. Attrition rate is higher in the age group of late twenties and early thirties 2. Ironically employess those who reside closer to the work location left job the most in this case. 3. Monthly income, Salary Hike percentage and Performance Rating also played major role in employee attrition. Lesser the income / Salary hike percentage / Performance Rating higher the attrition count. 4. Employees with no stock options leaves find less binding with the organisation and tends to leave. 5. Employee those who have more than average Job Satisfaction, Relationship satisfaction and Work Life Balance quits the most in this case. So Sticking to an organisation doesnt have relationship with these parameters 6. Attrition rate is high with the Employees those who have one year of experience in the current organisation. 7. Based on the "YearsSinceLastPromotion" column data, employees basically wait till a promotion to leave from the company in order to switch to a better role with better pay. 8. On an average , employees decide to leave after getting couple of years of experience in the current role.

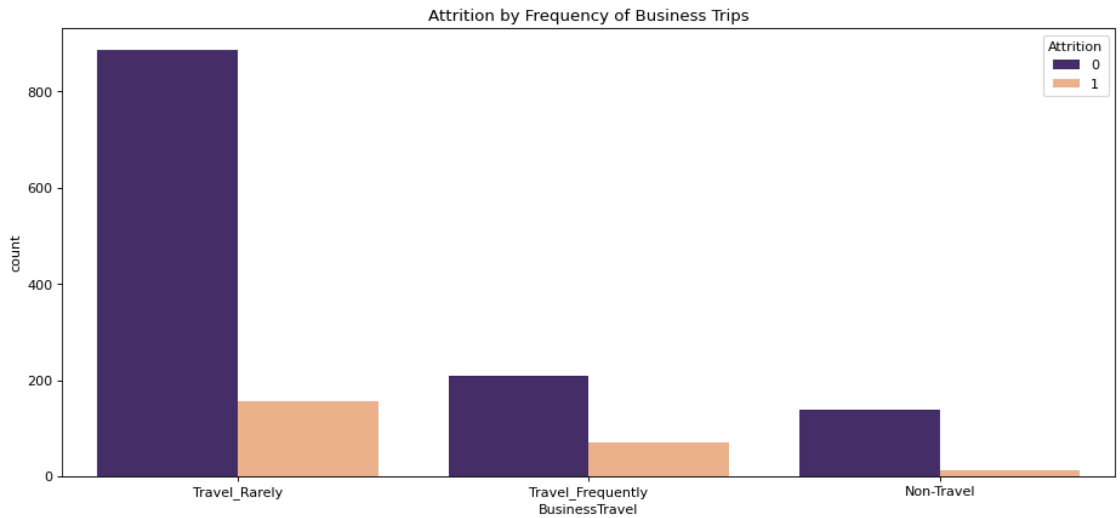
[35]:
```python
#Attrition by Age
plt.figure(figsize=(14,6), dpi=80)
sns.countplot(data=Dataset,
              x='Age',
              hue='Attrition',
              order = Dataset['Age'].value_counts().index,
              palette='seismic_r').set_title('Attrition by Age');
```



Employee between the age of 28-35 mostly preferred to leave the organisation than in the other age group.

[36]:
```python
#Attrition by Frequency of Business Travel
plt.figure(figsize=(14,6), dpi=80)
```

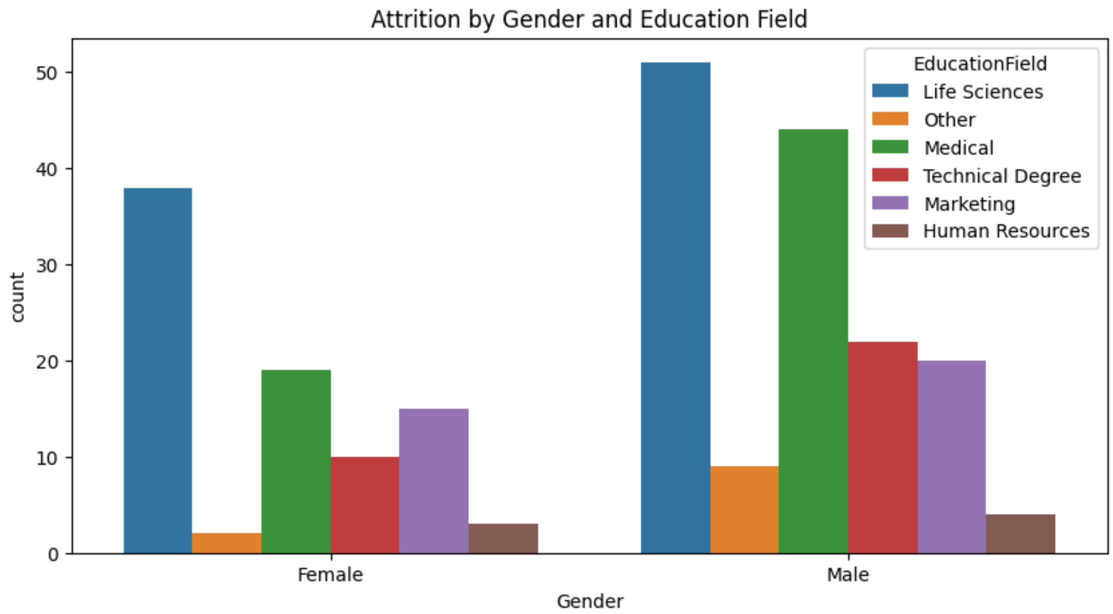
```
sns.countplot(data=Dataset,
              x='BusinessTravel',
              hue='Attrition',
              order = Dataset['BusinessTravel'].value_counts().index,
              palette=['#432371',"#FAAE7B"]).set_title('Attrition by Frequency␣
↪of Business Trips');
```



Compared to those who didn't travel at all or those who traveled pretty frequently, those who rarely travels tend to leave the most

[37]:
```
plt.figure(figsize = (10,5))

plt.title("Attrition by Gender and Education Field")
sns.countplot(x = 'Gender', hue = 'EducationField', data = empAttr)
```

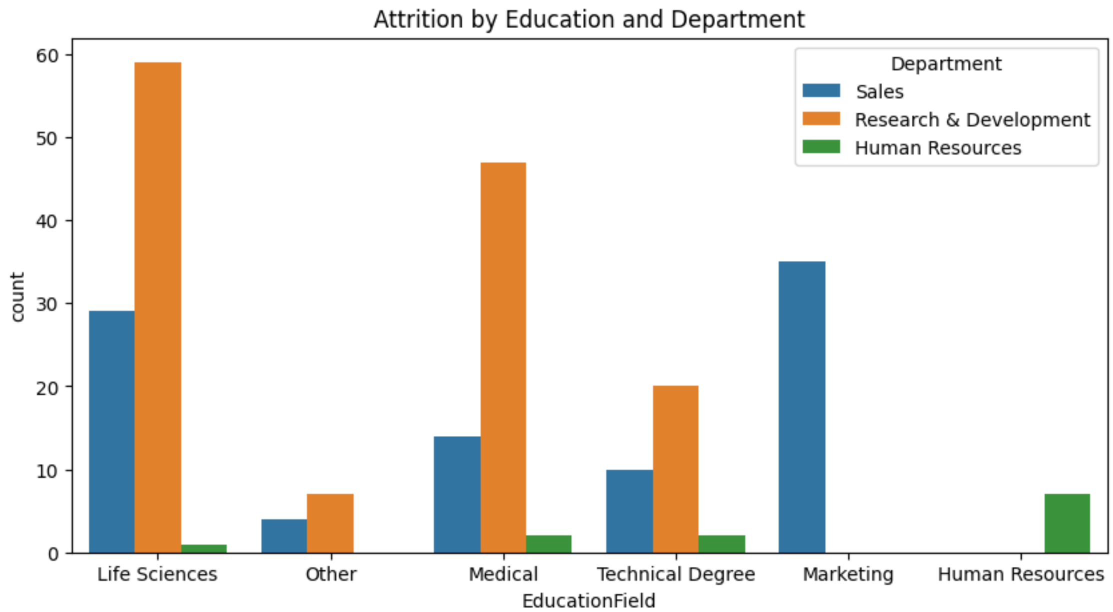
[37]:
```
<Axes: title={'center': 'Attrition by Gender and Education Field'},
xlabel='Gender', ylabel='count'>
```



- Attrition of Male employees is comparatively higher than female employees.
- With respect to Education fields,Attrition from Life Science field is the highest and Medical field being the second highest.

[38]:
```
plt.figure(figsize = (10,5))

plt.title("Attrition by Education and Department")
sns.countplot(x = 'EducationField', hue = 'Department', data = empAttr)
```

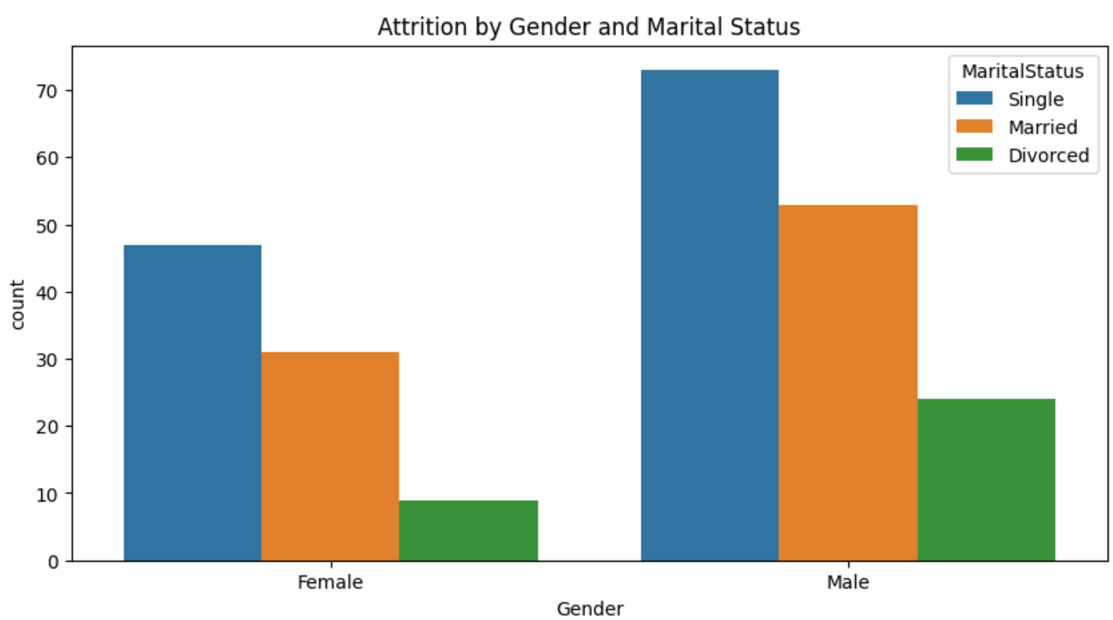
[38]:
```
<Axes: title={'center': 'Attrition by Education and Department'},
xlabel='EducationField', ylabel='count'>
```



on seeing attrition rate in departments, employees from Research department leaves the most.

[39]:
```
plt.figure(figsize = (10,5))

plt.title("Attrition by Gender and Marital Status")
sns.countplot(x = 'Gender', hue = 'MaritalStatus', data = empAttr)
```

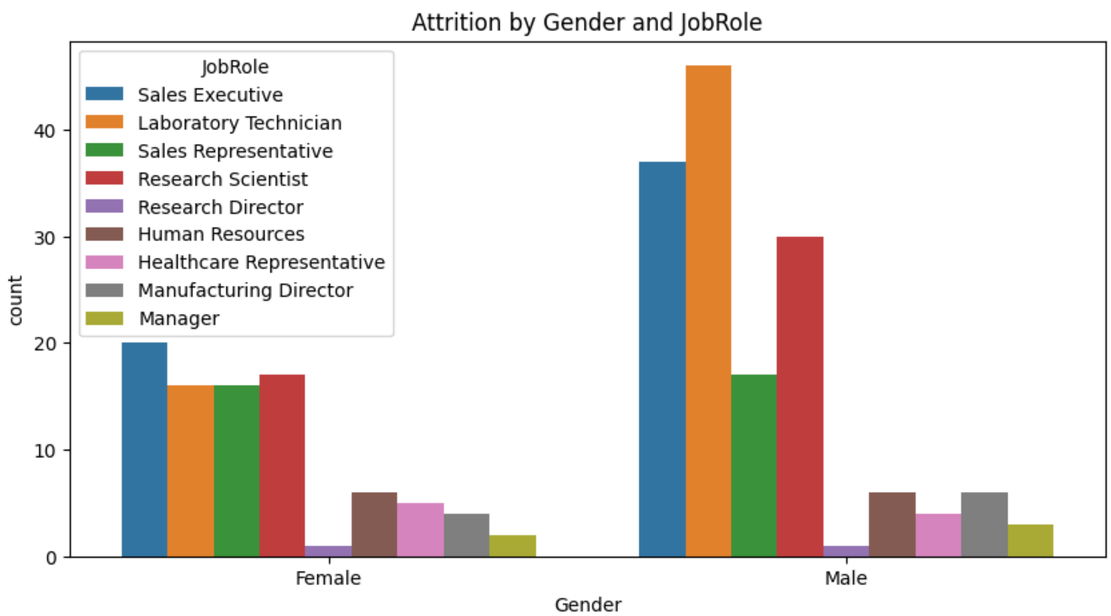
[39]:
```
<Axes: title={'center': 'Attrition by Gender and Marital Status'},
xlabel='Gender', ylabel='count'>
```



Single Men tends to leave from organisation the most than married or divorced Me. Same is the case with Female employees as well

[40]:
```
plt.figure(figsize = (10,5))

plt.title("Attrition by Gender and JobRole")
sns.countplot(x = 'Gender', hue = 'JobRole', data = empAttr)
```

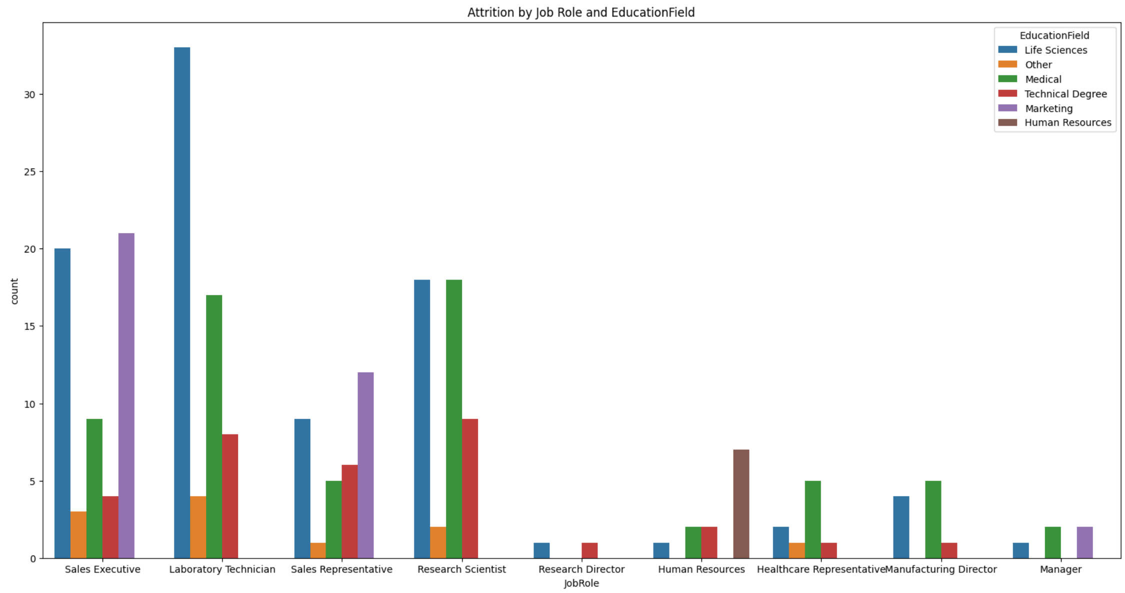
[40]:
```
<Axes: title={'center': 'Attrition by Gender and JobRole'}, xlabel='Gender',
ylabel='count'>
```



- Laboratory Technicians' attrition rate is higher compared to other Job roles among male employees.
- Sales Representatives' attrition rate is higher compared to other Job roles among female employees.

[41]:
```
plt.figure(figsize = (20,10))

plt.title("Attrition by Job Role and EducationField")
sns.countplot(x = 'JobRole', hue = 'EducationField', data = empAttr)
```

[41]:
```
<Axes: title={'center': 'Attrition by Job Role and EducationField'},
xlabel='JobRole', ylabel='count'>
```



Attrition rate of Life science graduates working in Laboratory Technician role is higher than others

Calculate the Attrition Ratio

[42]:
```
nYes = empAttr.shape[0]
nNo = Dataset.shape[0] - nYes
print(nYes,nNo)
AttritionRate = (nYes/(nYes+nNo))*100
AttritionRate
EmployeeWhoChoseToStay = (nNo/(nYes+nNo))*100
AttritionRate, EmployeeWhoChoseToStay
```

```
237 1233
```

[42]:
```
(16.122448979591837, 83.87755102040816)
```

According to this Dataset, 83.88% of employees chose to stay in the company, whereas 16.12% of employees chose to quit.

[43]:
```
#Categorical  Columns
catCols =␣
↪['BusinessTravel','Department','EducationField','Gender','JobRole','OverTime','MaritalStatu
Dataset = pd.get_dummies(Dataset, columns=catCols)
Dataset.head()
```

[43]:
```
   Age  Attrition  DailyRate  DistanceFromHome  Education  \
0   41          1       1102                 1          2
1   49          0        279                 8          1
2   37          1       1373                 2          2
```

```
3   33          0       1392                3          4
4   27          0        591                2          1

   EnvironmentSatisfaction  HourlyRate  JobInvolvement  JobLevel  \
0                        2          94               3         2
1                        3          61               2         2
2                        4          92               2         1
3                        4          56               3         1
4                        1          40               3         1

   JobSatisfaction  …  JobRole_Manufacturing Director  \
0                4  …                           False
1                2  …                           False
2                3  …                           False
3                3  …                           False
4                2  …                           False

   JobRole_Research Director  JobRole_Research Scientist  \
0                      False                       False
1                      False                        True
2                      False                       False
3                      False                        True
4                      False                       False

   JobRole_Sales Executive  JobRole_Sales Representative  OverTime_No  \
0                     True                         False        False
1                    False                         False         True
2                    False                         False        False
3                    False                         False        False
4                    False                         False         True

   OverTime_Yes  MaritalStatus_Divorced  MaritalStatus_Married  \
0          True                   False                  False
1         False                   False                   True
2          True                   False                  False
3          True                   False                   True
4         False                   False                   True

   MaritalStatus_Single
0                  True
1                 False
2                  True
3                 False
4                 False

[5 rows x 52 columns]
```

Train and Test Data split and Logistic Regression

[44]:
```python
X = Dataset.drop('Attrition',axis=1) #Independent Variables
#X = Dataset[numCols]
Y = Dataset['Attrition']   # Target or Dependent Variable

X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.
 ↪3,random_state=20)
X_train.head()
```

[44]:
```
      Age  DailyRate  DistanceFromHome  Education  EnvironmentSatisfaction  \
1420   41        642                 1          3                        4
743    59        715                 2          3                        3
267    25        675                 5          2                        2
1322   46        706                 2          2                        4
1281   35        303                27          3                        3

      HourlyRate  JobInvolvement  JobLevel  JobSatisfaction  MonthlyIncome  \
1420          76               3         1                4           2782
743           69               2         4                4          13726
267           85               4         2                1           4000
1322          82               3         3                4           8578
1281          84               3         2                4           5813

      …  JobRole_Manufacturing Director  JobRole_Research Director  \
1420  …                           False                      False
743   …                            True                      False
267   …                           False                      False
1322  …                            True                      False
1281  …                           False                      False

      JobRole_Research Scientist  JobRole_Sales Executive  \
1420                        True                    False
743                        False                    False
267                        False                    False
1322                       False                    False
1281                       False                     True

      JobRole_Sales Representative  OverTime_No  OverTime_Yes  \
1420                         False         True         False
743                          False        False          True
267                          False         True         False
1322                         False         True         False
1281                         False        False          True

      MaritalStatus_Divorced  MaritalStatus_Married  MaritalStatus_Single
1420                   False                   True                 False
743                    False                  False                  True
267                     True                  False                 False
```

```
1322                    True               False               False
1281                   False               False                True

[5 rows x 51 columns]
```

[45]:
```
print("{0:0.2f}% data is in training set".format((len(X_train)/len(Dataset.
 ↪index)) * 100))
print("{0:0.2f}% data is in test set".format((len(X_test)/len(Dataset.index)) *␣
 ↪100))
```

```
70.00% data is in training set
30.00% data is in test set
```

Logistic Regression

[46]:
```
# Fit the model on train
model = LogisticRegression(solver="liblinear")
model.fit(X_train, Y_train)
#predict on test
#
y_predict = model.predict(X_test)
```

Finding the score for Train Dataset.

[47]:
```
model_score = model.score(X_train, Y_train)
print(model_score)
```

```
0.8892128279883382
```

Finding the score for Train Dataset.

[48]:
```
model_score = model.score(X_test, Y_test)
print(model_score)
```

```
0.8684807256235828
```

Looking at the R^2 values of both Train and Test Dataset, The Logistic Regression model fits pretty well for this Employee Attrition Dataset. Also, Bias and Variance values seems to be low. So we can conclude that this model fits well.

Decision Tree Model

Gini Impurity

[49]:
```
model_gini=DecisionTreeClassifier(criterion='gini')
```

[50]:
```
model_gini.fit(X_train, Y_train)
```

[50]:
```
DecisionTreeClassifier()
```

[51]:
```
model_gini.score(X_train, Y_train)
```

[51]: 1.0

[52]:
```
model_gini.score(X_test,Y_test)
```

[52]: 0.8140589569160998

Entropy

[53]:
```
model_ent=DecisionTreeClassifier(criterion='entropy')
```

[54]:
```
model_ent.fit(X_train, Y_train)
```

[54]: DecisionTreeClassifier(criterion='entropy')

[55]:
```
model_ent.score(X_train, Y_train)
```

[55]: 1.0

[56]:
```
model_ent.score(X_test,Y_test)
```

[56]: 0.7664399092970522

The train and test scores for both Gini and Entropy models interprets that there is Overfitting. In order to avoid this overfitting issue, So we will need to handle the outliers.

For now, let us conclude that Logistic Regression is the best fit method for this dataset.

However we would need to see the handle the overfitting issue faced in Decision Tree model. Also would need to work on Random forest model for this dataset.

[57]:
```
#Below mentioned is a generic function to calculate accuracy score, confusion␣
 ↪matrix and  classification report
#for decision tree model. This function would print all the required values in␣
 ↪a matrix format.

def print_score(clf, X_train, y_train, X_test, y_test, train=True):
    if train: # for training data
        pred = clf.predict(X_train)
        clf_report = pd.DataFrame(classification_report(y_train, pred,␣
 ↪output_dict=True))
        print("Train Result:\n================================================")
        print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
        print("_______________________________________________")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_train, pred)}\n")

    elif train==False: # for testing data
        pred = clf.predict(X_test)
```

```
        clf_report = pd.DataFrame(classification_report(y_test, pred,␣
↪output_dict=True))
        print("Test Result:\n================================================")
        print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
        print("_______________________________________________")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n")
```

First we'll see the detailed score for decision tree training and test data.

```
[58]: dtree = DecisionTreeClassifier(random_state=42)
dtree.fit(X_train, Y_train)

print_score(dtree, X_train, Y_train, X_test, Y_test, train=True)
print_score(dtree, X_train, Y_train, X_test, Y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
CLASSIFICATION REPORT:
               0      1  accuracy  macro avg  weighted avg
precision    1.0    1.0       1.0        1.0           1.0
recall       1.0    1.0       1.0        1.0           1.0
f1-score     1.0    1.0       1.0        1.0           1.0
support    862.0  167.0       1.0     1029.0        1029.0
_______________________________________________
Confusion Matrix:
 [[862   0]
 [  0 167]]

Test Result:
================================================
Accuracy Score: 78.68%
_______________________________________________
CLASSIFICATION REPORT:
                    0          1  accuracy   macro avg  weighted avg
precision    0.879452   0.342105  0.786848    0.610779      0.794159
recall       0.865229   0.371429  0.786848    0.618329      0.786848
f1-score     0.872283   0.356164  0.786848    0.614223      0.790359
support    371.000000  70.000000  0.786848  441.000000    441.000000
_______________________________________________
Confusion Matrix:
 [[321  50]
 [ 44  26]]
```

With the previous results, we could see that with normal decision tree model, without any tuning, there is overfitting issue.

Now we will try to experiment with a combination of the parameters in the decision tree classifier method with the help of the GridSearchCV method. This method would try to find out results for all the probable method parameter combinations and find out the best score among them. We would basically refer this step as HyperParameter Tuning

[59]:
```
from sklearn.model_selection import GridSearchCV

params = {
    "criterion":("gini", "entropy"),
    "splitter":("best", "random"),
    "max_depth":(list(range(1, 20))),
    "min_samples_split":[2, 3, 4],
    "min_samples_leaf":list(range(1, 20)),
}


dtree = DecisionTreeClassifier(random_state=42)
dtreeXv = GridSearchCV(
    dtree,
    params,
    scoring="f1",
    n_jobs=-1,
    verbose=1,
    cv=5
)


dtreeXv.fit(X_train, Y_train)
best_params = dtreeXv.best_params_
print(f"Best paramters: {best_params})")

dtree = DecisionTreeClassifier(**best_params)
dtree.fit(X_train, Y_train)
print_score(dtree, X_train, Y_train, X_test, Y_test, train=True)
print_score(dtree, X_train, Y_train, X_test, Y_test, train=False)
```

```
Fitting 5 folds for each of 4332 candidates, totalling 21660 fits
Best paramters: {'criterion': 'gini', 'max_depth': 7, 'min_samples_leaf': 10,
'min_samples_split': 2, 'splitter': 'best'})
Train Result:
================================================
Accuracy Score: 89.21%
_______________________________________________
CLASSIFICATION REPORT:
                  0          1  accuracy    macro avg  weighted avg
precision  0.906826   0.764151  0.892128     0.835488      0.883670
recall     0.970998   0.485030  0.892128     0.728014      0.892128
```

```
f1-score     0.937815    0.593407  0.892128     0.765611      0.881920
support    862.000000  167.000000  0.892128  1029.000000   1029.000000
_______________________________________________
Confusion Matrix:
 [[837  25]
 [ 86  81]]

Test Result:
================================================
Accuracy Score: 85.26%
_______________________________________________
CLASSIFICATION REPORT:
                    0          1  accuracy   macro avg  weighted avg
precision    0.890306   0.551020  0.852608    0.720663      0.836451
recall       0.940701   0.385714  0.852608    0.663208      0.852608
f1-score     0.914810   0.453782  0.852608    0.684296      0.841631
support    371.000000  70.000000  0.852608  441.000000    441.000000
_______________________________________________
Confusion Matrix:
 [[349  22]
 [ 43  27]]
```

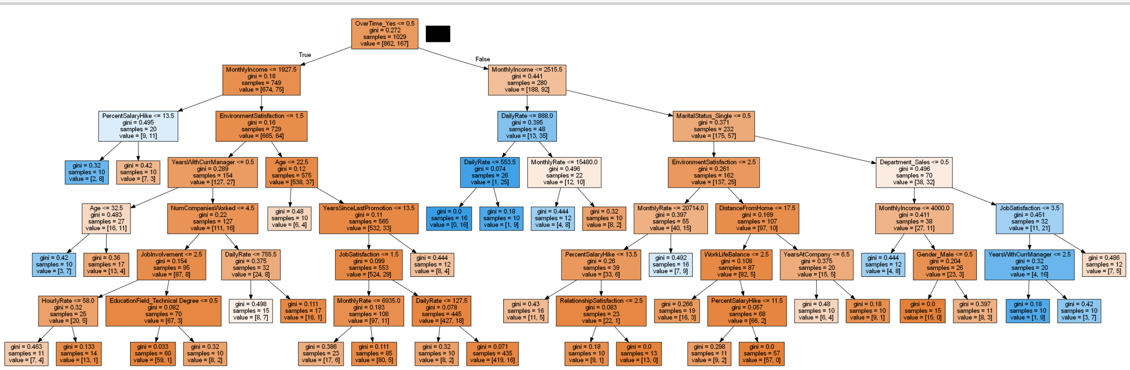
[63]:

```python
from IPython.display import Image
from six import StringIO
from sklearn.tree import export_graphviz
import pydot

features = list(Dataset.columns)
features.remove("Attrition")
```

[62]:

```python
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'
dot_data = StringIO()
export_graphviz(dtree, out_file=dot_data, feature_names=features, filled=True)
graph = pydot.graph_from_dot_data(dot_data.getvalue())
Image(graph[0].create_png())
```

[62]:



Random Forest Model

[64]:
```python
from sklearn.ensemble import RandomForestClassifier

ranFC = RandomForestClassifier(n_estimators=100)
ranFC.fit(X_train, Y_train)

print_score(ranFC, X_train, Y_train, X_test, Y_test, train=True)
print_score(ranFC, X_train, Y_train, X_test, Y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
CLASSIFICATION REPORT:
               0      1  accuracy  macro avg  weighted avg
precision    1.0    1.0       1.0        1.0           1.0
recall       1.0    1.0       1.0        1.0           1.0
f1-score     1.0    1.0       1.0        1.0           1.0
support    862.0  167.0       1.0     1029.0        1029.0
_______________________________________________
Confusion Matrix:
 [[862   0]
 [  0 167]]

Test Result:
================================================
Accuracy Score: 85.26%
_______________________________________________
CLASSIFICATION REPORT:
                    0          1  accuracy   macro avg  weighted avg
precision    0.862559   0.631579  0.852608    0.747069      0.825896
recall       0.981132   0.171429  0.852608    0.576280      0.852608
f1-score     0.918033   0.269663  0.852608    0.593848      0.815117
support    371.000000  70.000000  0.852608  441.000000    441.000000
_______________________________________________
Confusion Matrix:
 [[364   7]
 [ 58  12]]
```

Again with Random Forest model as well we are facing Overfitting issue. So we will need to sort out this with Hyperparameter Tuning.For Random Forest Model, we will need to do Randomized Search Cross Validation method.

[65]:

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV

n_estimators = [int(x) for x in np.linspace(start=200, stop=2000, num=10)]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(10, 110, num=11)]
max_depth.append(None)

min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
bootstrap = [True, False]

random_grid = {
    'n_estimators': n_estimators,
    'max_features': max_features,
    'max_depth': max_depth,
    'min_samples_split': min_samples_split,
    'min_samples_leaf': min_samples_leaf,
    'bootstrap': bootstrap
}

ranFC = RandomForestClassifier(random_state=42)

randomizedSrchCV = RandomizedSearchCV(
    estimator=ranFC,
    scoring='f1',
    param_distributions=random_grid,
    n_iter=200,
    cv=5,
    verbose=1,
    random_state=42,
    n_jobs=-1
)

randomizedSrchCV.fit(X_train, Y_train)
rf_best_params = randomizedSrchCV.best_params_
print(f"Best paramters: {rf_best_params})")

ranFC = RandomForestClassifier(**rf_best_params)
ranFC.fit(X_train, Y_train)

print_score(ranFC, X_train, Y_train, X_test, Y_test, train=True)
print_score(ranFC, X_train, Y_train, X_test, Y_test, train=False)
```

```
Fitting 5 folds for each of 200 candidates, totalling 1000 fits
Best paramters: {'n_estimators': 800, 'min_samples_split': 2,
'min_samples_leaf': 1, 'max_features': 'sqrt', 'max_depth': 40, 'bootstrap':
False})
```

```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
CLASSIFICATION REPORT:
               0      1  accuracy  macro avg  weighted avg
precision    1.0    1.0       1.0        1.0           1.0
recall       1.0    1.0       1.0        1.0           1.0
f1-score     1.0    1.0       1.0        1.0           1.0
support    862.0  167.0       1.0     1029.0        1029.0
_______________________________________________
Confusion Matrix:
 [[862   0]
 [  0 167]]

Test Result:
================================================
Accuracy Score: 86.39%
_______________________________________________
CLASSIFICATION REPORT:
                    0          1  accuracy   macro avg  weighted avg
precision    0.872902   0.708333  0.863946    0.790618      0.846780
recall       0.981132   0.242857  0.863946    0.611995      0.863946
f1-score     0.923858   0.361702  0.863946    0.642780      0.834627
support    371.000000  70.000000  0.863946  441.000000    441.000000
_______________________________________________
Confusion Matrix:
 [[364   7]
 [ 53  17]]
```

[66]:
```python
n_estimators = [100, 500, 1000, 1500]
max_features = ['auto', 'sqrt']
max_depth = [2, 3, 5]
max_depth.append(None)
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4, 10]
bootstrap = [True, False]

params_grid = {
    'n_estimators': n_estimators,
    'max_features': max_features,
    'max_depth': max_depth,
    'min_samples_split': min_samples_split,
    'min_samples_leaf': min_samples_leaf,
    'bootstrap': bootstrap
}
```

```python
ranGS = RandomForestClassifier(random_state=42)

ranGSCV = GridSearchCV(
    ranGS,
    params_grid,
    scoring="f1",
    cv=5,
    verbose=1,
    n_jobs=-1
)

ranGSCV.fit(X_train, Y_train)
best_params = ranGSCV.best_params_
print(f"Best parameters: {best_params}")

ranGS = RandomForestClassifier(**best_params)
ranGS.fit(X_train, Y_train)

print_score(ranGS, X_train, Y_train, X_test, Y_test, train=True)
print_score(ranGS, X_train, Y_train, X_test, Y_test, train=False)
```

```
Fitting 5 folds for each of 768 candidates, totalling 3840 fits
Best parameters: {'bootstrap': False, 'max_depth': None, 'max_features': 'sqrt',
'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 1500}
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
CLASSIFICATION REPORT:
               0      1  accuracy  macro avg  weighted avg
precision    1.0    1.0       1.0        1.0           1.0
recall       1.0    1.0       1.0        1.0           1.0
f1-score     1.0    1.0       1.0        1.0           1.0
support    862.0  167.0       1.0     1029.0        1029.0
_______________________________________________
Confusion Matrix:
 [[862   0]
 [  0 167]]

Test Result:
================================================
Accuracy Score: 86.62%
_______________________________________________
CLASSIFICATION REPORT:
                   0          1  accuracy   macro avg  weighted avg
precision   0.873206   0.739130  0.866213    0.806168      0.851924
recall      0.983827   0.242857  0.866213    0.613342      0.866213
```

```
f1-score      0.925222   0.365591  0.866213    0.645407      0.836392
support     371.000000  70.000000  0.866213  441.000000    441.000000
-----------------------------------------------
Confusion Matrix:
 [[365   6]
 [ 53  17]]
```

Hyperparameter tuning for Random forest did not do much of help to find out the best fit, still there is overfitting issue.

So to conclude 1. Logistic regression fits well with its default parameters. 2. D-tree model fits well after performing Hyperparameter Tuning 3. Neither default parameters, nor Hyperparamter tuning works for Random Forest model.

[ ]: