- Education : 1 = 'Below College' , 2 = 'College' , 3 = 'Bachelor', 4 = 'Master' , 5 = 'Doctor'
- Environment Satisfaction : 1 = 'Low', 2 = 'Medium', 3 = 'High', 4 = 'Very High'
- Job Involvement : 1 = 'Low' , 2 = 'Medium', 3 = 'High' , 4 = 'Very High'
- Job Satisfaction : 1 = 'Low' , 2 = 'Medium' , 3 = 'High' , 4 = 'Very High'
- Performance Rating : 1 = 'Low' , 2 = 'Good' , 3 = 'Excellent' , 4 = 'Outstanding'
- Relationship Satisfaction : 1 = 'Low' , 2 = 'Medium' , 3 = 'High' , 4 = 'Very High'
- WorkLife Balance : 1 = 'Bad' , 2 = 'Good' , 3 = 'Better' , 4 = 'Best'

Start coding or generate with AI.

DATA PREPROCESSING

Importing LIbraries and Understanding Data

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

import warnings
warnings.filterwarnings("ignore")
```

Reading the Data

```
Data = pd.read_csv("/content/employee.csv")
```

```
Data.shape
```

```
(1470, 35)
```

```
Data.head()
```

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

```
pd.set_option('display.max_columns', None)
Data.head()
```

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

Checking the summary of data

```
Data.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
```

```
Data.describe()
```

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNu
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.0	1470.0(
mean	36.923810	802.485714	9.192517	2.912925	1.0	1024.8(
std	9.135373	403.509100	8.106864	1.024165	0.0	602.0:
min	18.000000	102.000000	1.000000	1.000000	1.0	1.0(
25%	30.000000	465.000000	2.000000	2.000000	1.0	491.2:
50%	36.000000	802.000000	7.000000	3.000000	1.0	1020.5(
75%	43.000000	1157.000000	14.000000	4.000000	1.0	1555.7:
max	60.000000	1499.000000	29.000000	5.000000	1.0	2068.0(

Total Unique Value

```
Data.nunique()
```

```
Age                         43
Attrition                    2
BusinessTravel               3
DailyRate                  886
Department                   3
DistanceFromHome            29
Education                    5
EducationField               6
EmployeeCount                1
EmployeeNumber            1470
```

```
EnvironmentSatisfaction        4
Gender                         2
HourlyRate                    71
JobInvolvement                 4
JobLevel                       5
JobRole                        9
JobSatisfaction                4
MaritalStatus                  3
MonthlyIncome               1349
MonthlyRate                 1427
NumCompaniesWorked            10
Over18                         1
OverTime                       2
PercentSalaryHike             15
PerformanceRating              2
RelationshipSatisfaction       4
StandardHours                  1
StockOptionLevel               4
TotalWorkingYears             40
TrainingTimesLastYear          7
WorkLifeBalance                4
YearsAtCompany                37
YearsInCurrentRole            19
YearsSinceLastPromotion       16
YearsWithCurrManager          18
dtype: int64
```

Missing Value

```
Data.isnull().sum()
```

```
Age                         0
Attrition                   0
BusinessTravel              0
DailyRate                   0
Department                  0
DistanceFromHome            0
Education                   0
EducationField              0
EmployeeCount               0
EmployeeNumber              0
EnvironmentSatisfaction     0
Gender                      0
HourlyRate                  0
JobInvolvement              0
JobLevel                    0
JobRole                     0
JobSatisfaction             0
MaritalStatus               0
MonthlyIncome               0
MonthlyRate                 0
NumCompaniesWorked          0
Over18                      0
OverTime                    0
PercentSalaryHike           0
PerformanceRating           0
RelationshipSatisfaction    0
StandardHours               0
StockOptionLevel            0
TotalWorkingYears           0
TrainingTimesLastYear       0
WorkLifeBalance             0
```

```
YearsAtCompany              0
YearsInCurrentRole          0
YearsSinceLastPromotion     0
YearsWithCurrManager        0
dtype: int64
```

CHECKING FOR DUPLICATES

```
print(Data.duplicated().value_counts())
Data.drop_duplicates(inplace=True)
print(len(Data))
```

```
False    1470
Name: count, dtype: int64
1470
```

```
num_cols = Data.select_dtypes(include=['float64', 'int64']).columns

Q1 = Data[num_cols].quantile(0.25)
Q3 = Data[num_cols].quantile(0.75)

IQR = Q3 - Q1

outlier_step = 1.5 * IQR
outliers_iqr = ((Data[num_cols] < (Q1 - outlier_step)) | (Data[num_cols] > (Q3 + outlier_step))).any(axis=1)

num_outliers_iqr = outliers_iqr.sum()
print(f'Number of outliers identified using IQR method: {num_outliers_iqr}')
```

```
Number of outliers identified using IQR method: 691
```

Dropping the unwanted columns

```
Employees_Data = Data.drop(['EmployeeCount','EmployeeNumber','Over18','StandardHours'], axis = 1)

Employees_Data.head()
```

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

Visualising Data

```
Data.hist(figsize=(20,20))
plt.show()
```

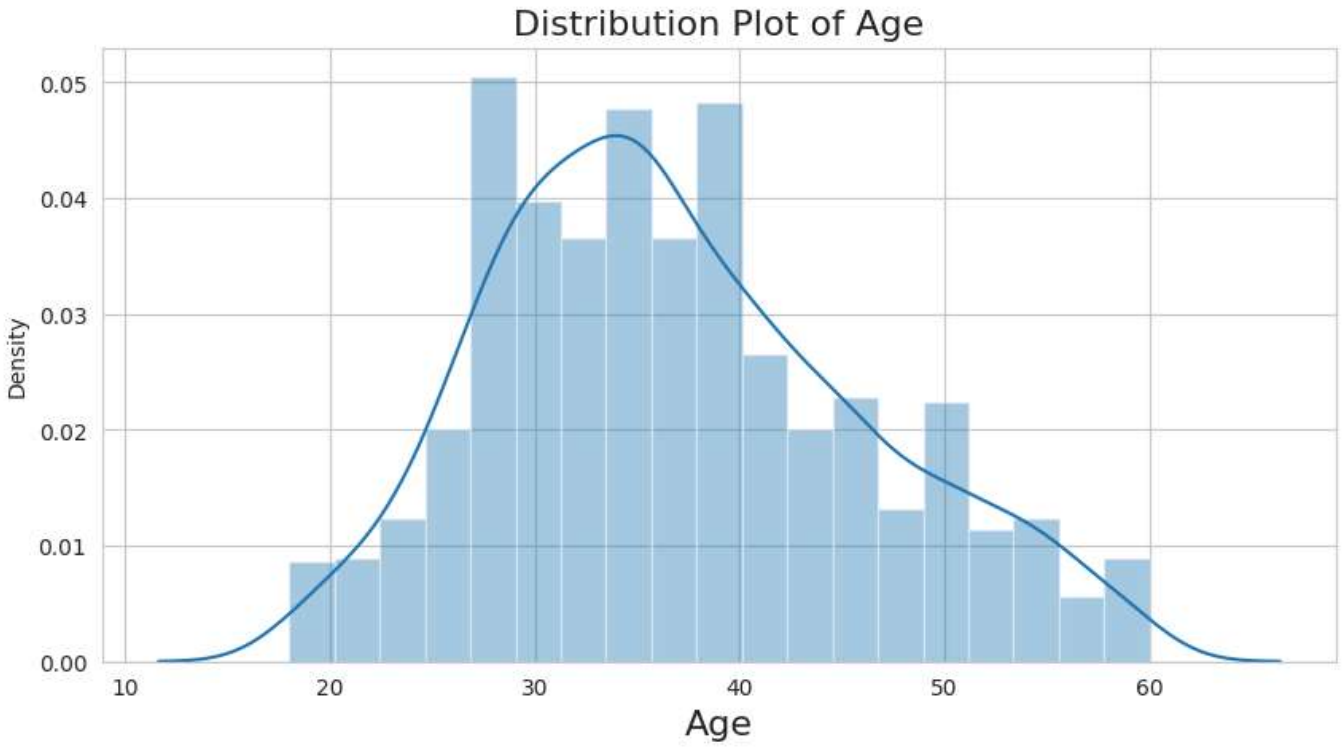
- DistanceFromHome: Most of employees live close to company which most distance are below 10km.
- MonthlyIncome: The majority of monthly income of employees are centred at around 5000. Only a few of people get high income over 10000.
- NumCompaniesWorked: Most employees only worked for one company.

- TotalWorkingYears, YearsAtCompany, YearsInCurrentRole, YearsSinceLastPromotion, YearsWithCurrManager: These time type data are right skewed which most of people stay in company only for a few years.
- DailyRate, HourlyRate, MonthlyRate distributed uniformly which the figure is similar in different intervals.

Age

```
sns.set_style("whitegrid")
plt.figure(figsize=(10,5))
sns.distplot(Data['Age'])
plt.xlabel('Age', fontsize=16)
plt.title('Distribution Plot of Age', fontsize=16)
```

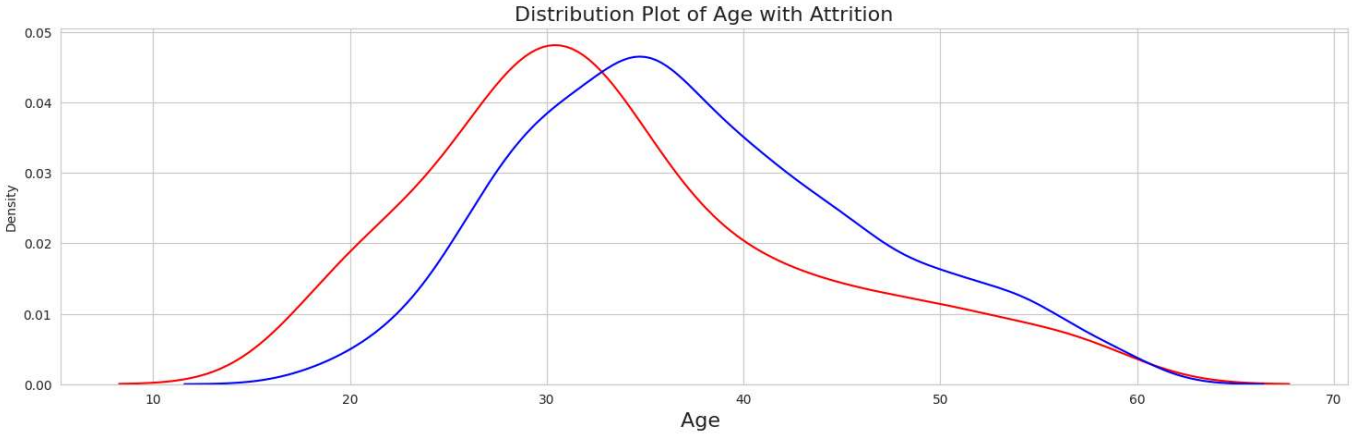
```
Text(0.5, 1.0, 'Distribution Plot of Age')
```



- The age distribution of this data set distributed normally which cover from 20 to 60. Most employees are 30 to 40.

```
plt.figure(figsize=(18,5))

sns.distplot(Data['Age'][Data['Attrition'] == 'Yes'], hist = False, color = 'red')

sns.distplot(Data['Age'][Data['Attrition'] == 'No'], hist = False, color = 'blue')

plt.xlabel('Age ', fontsize = 16)
plt.title('Distribution Plot of Age with Attrition', fontsize = 16)
```

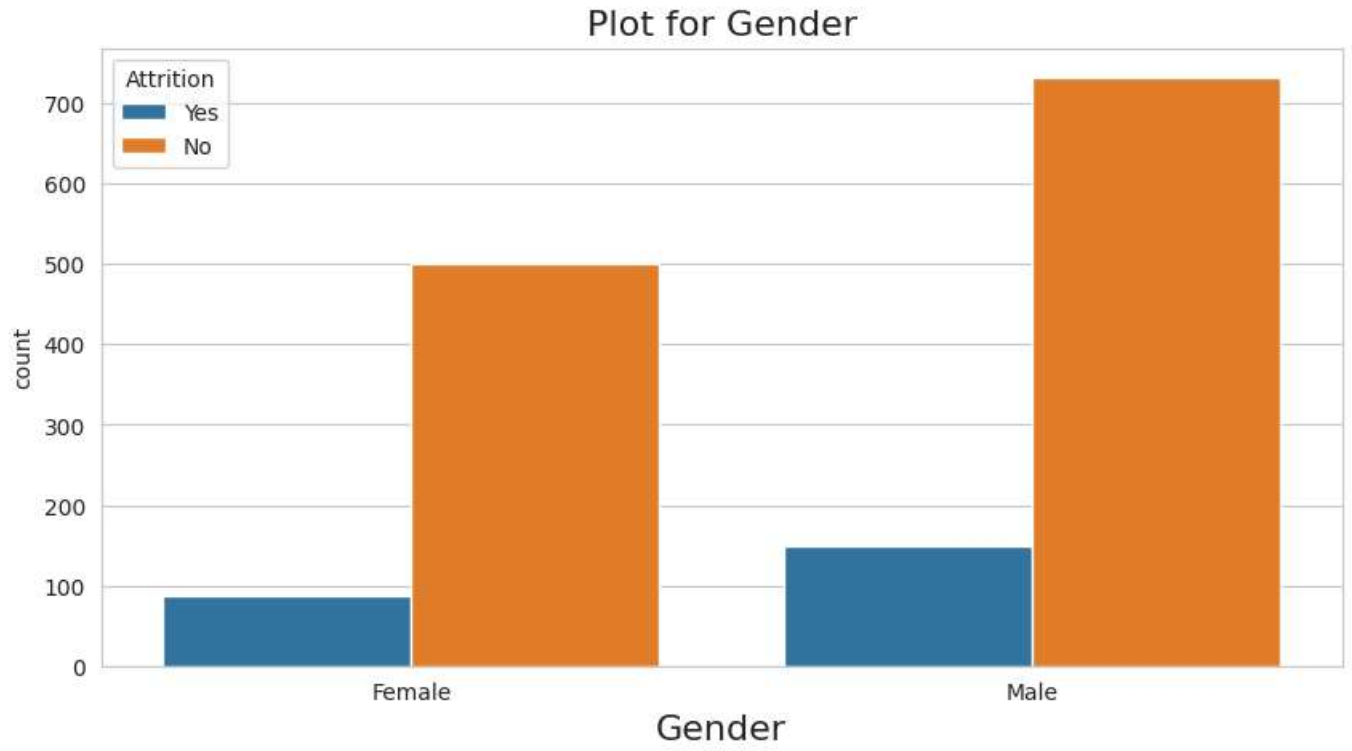
```
Text(0.5, 1.0, 'Distribution Plot of Age with Attrition')
```



Gender

```
plt.figure(figsize=(10,5))
sns.countplot(x = 'Gender', hue="Attrition", data = Data)
plt.xlabel('Gender', fontsize=16)
plt.title('Plot for Gender', fontsize=16)
```

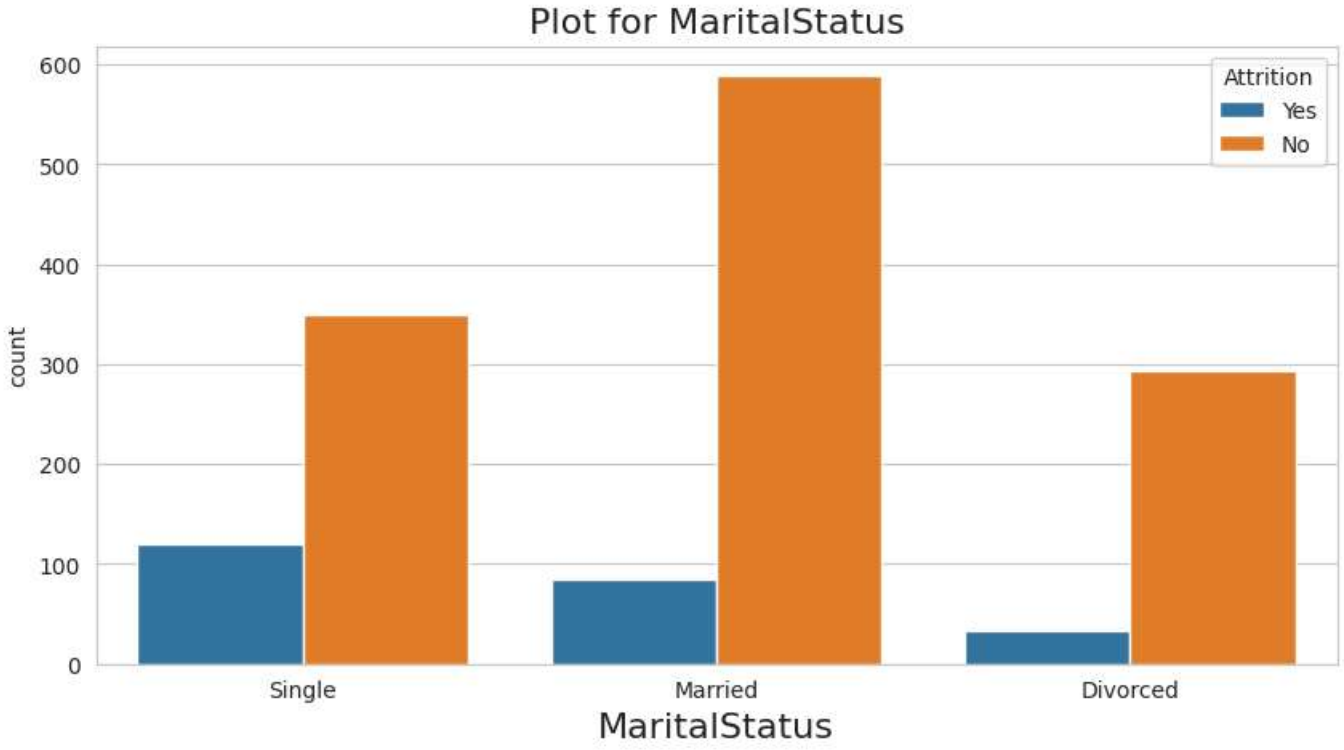
```
Text(0.5, 1.0, 'Plot for Gender')
```



Marital Status

```
plt.figure(figsize=(10,5))
sns.countplot(x = 'MaritalStatus', hue="Attrition", data = Data)
plt.xlabel('MaritalStatus', fontsize=16)
plt.title('Plot for MaritalStatus', fontsize=16)
```

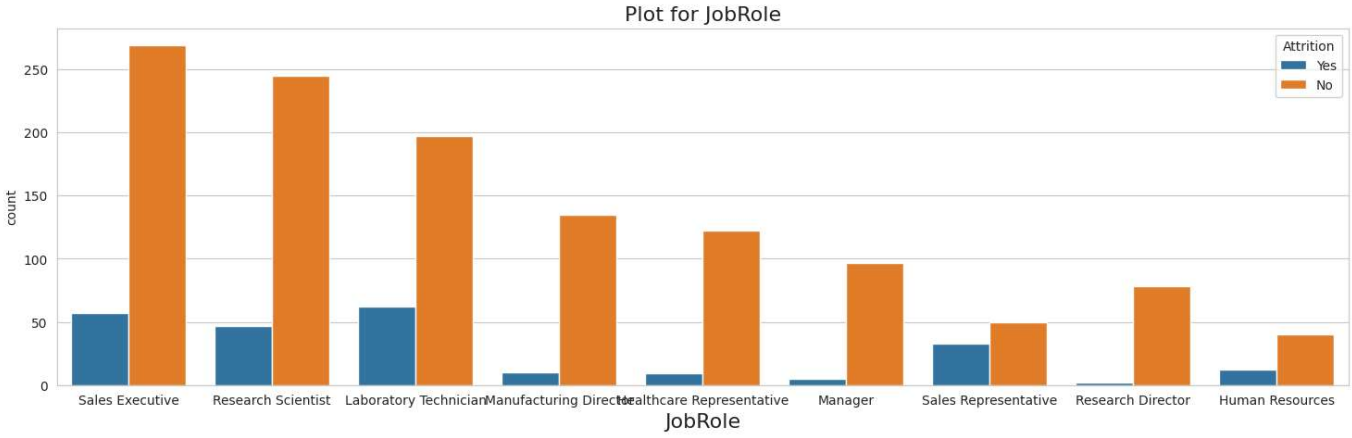
```
Text(0.5, 1.0, 'Plot for MaritalStatus')
```



Job Role

```
plt.figure(figsize=(18,5))
sns.countplot(x = 'JobRole', hue="Attrition",data = Data)
plt.xlabel('JobRole', fontsize=16)
plt.title('Plot for JobRole', fontsize=16)
```

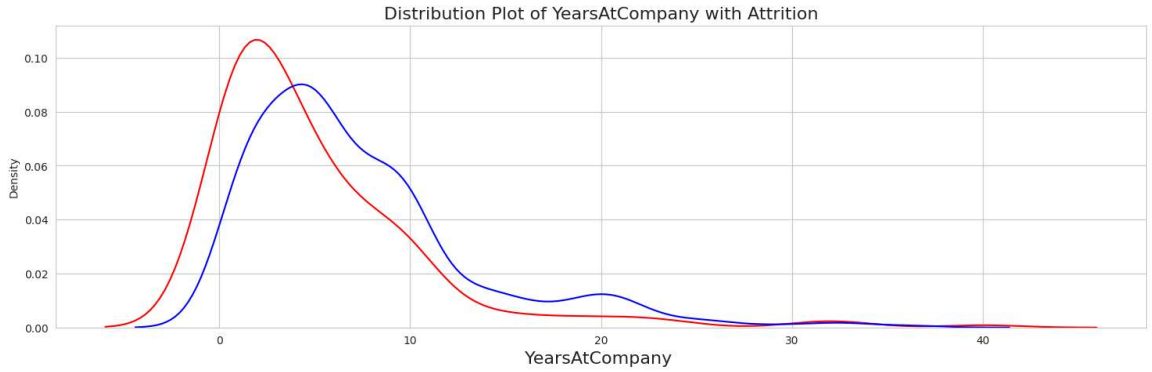
```
Text(0.5, 1.0, 'Plot for JobRole')
```



Years At Company

```
plt.figure(figsize=(18,5))

sns.distplot(Data['YearsAtCompany'][Data['Attrition'] == 'Yes'], hist = False, color = 'red')

sns.distplot(Data['YearsAtCompany'][Data['Attrition'] == 'No'], hist = False, color = 'blue')

plt.xlabel('YearsAtCompany ', fontsize = 16)
plt.title('Distribution Plot of YearsAtCompany with Attrition', fontsize = 16)
```

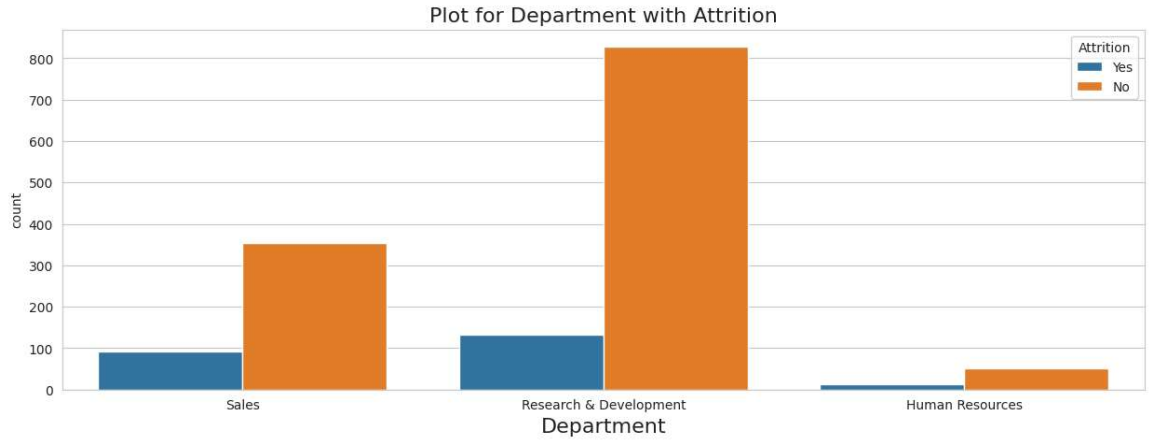
```
Text(0.5, 1.0, 'Distribution Plot of YearsAtCompany with Attrition')
```



Department

```
plt.figure(figsize=(15,5))
sns.countplot(x = 'Department', hue = 'Attrition', data = Data)
plt.xlabel('Department', fontsize=16)
plt.title('Plot for Department with Attrition', fontsize=16)
```

```
Text(0.5, 1.0, 'Plot for Department with Attrition')
```



```
Attrition_Yes = Data.groupby(Data['Department'][Data['Attrition'] == 'Yes']).size().reset_index(name = "Attrition_Y
Attrition_No = Data.groupby(Data['Department'][Data['Attrition'] == 'No']).size().reset_index(name = "Attrition_No"

Total = Data.groupby(Data['Department']).size().reset_index(name = 'Total')

Department_Table = Attrition_Yes.join(Attrition_No['Attrition_No']).join(Total['Total'])

Department_Table
```

	Department	Attrition_Yes	Attrition_No	Total
0	Human Resources	12	51	63
1	Research & Development	133	828	961
2	Sales	92	354	446

Next steps: Generate code with Department_Table View recommended plots

```
Department_Table['Attrition_Yes'] = round((Attrition_Yes['Attrition_Yes']/Total['Total'])*100,2)
Department_Table['Attrition_No'] = round((Attrition_No['Attrition_No']/Total['Total'])*100,2)

Department_perc = Department_Table.iloc[:,0:3]
Department_perc = Department_perc.rename(columns={'Attrition_Yes': 'Attrition_Yes(%)', 'Attrition_No': 'Attrition_N

Department_perc
```

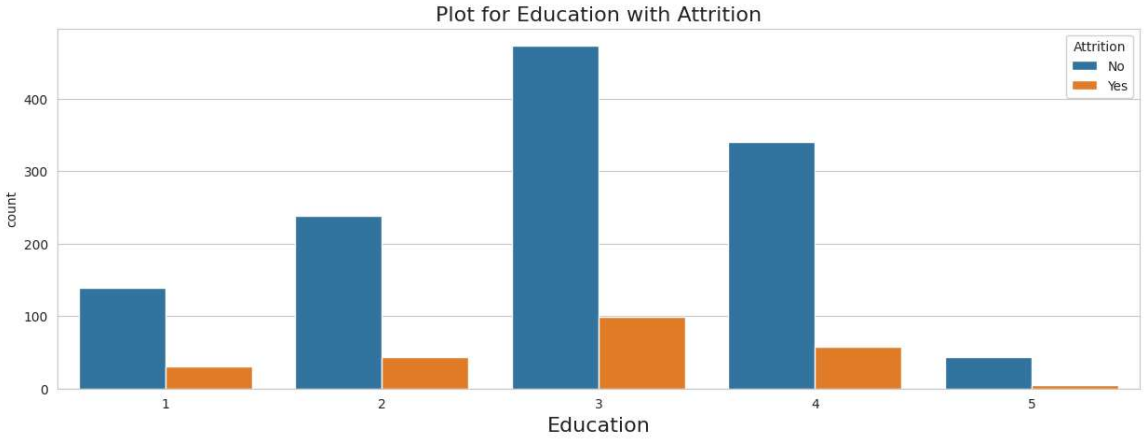
	Department	Attrition_Yes(%)	Attrition_No(%)
0	Human Resources	19.05	80.95
1	Research & Development	13.84	86.16
2	Sales	20.63	79.37

Next steps: Generate code with Department_perc View recommended plots

Education

```
plt.figure(figsize=(15,5))
sns.countplot(x = 'Education', hue = 'Attrition',data = Data)
plt.xlabel('Education', fontsize=16)
plt.title('Plot for Education with Attrition', fontsize=16)


# 1 = 'Below College' , 2 = 'College' , 3 = 'Bachelor', 4 = 'Master' , 5 = 'Doctor'
```

```
Text(0.5, 1.0, 'Plot for Education with Attrition')
```




```
Attrition_Yes = Data.groupby(Data['Education'][Data['Attrition'] == 'Yes']).size().reset_index(name = "Attrition_Ye
Attrition_No = Data.groupby(Data['Education'][Data['Attrition'] == 'No']).size().reset_index(name = "Attrition_No")

Total = Data.groupby(Data['Education']).size().reset_index(name = 'Total')

Education_Table = Attrition_Yes.join(Attrition_No['Attrition_No']).join(Total['Total'])

Education_Table['Education'] = ['Below College' , 'College' , 'Bachelor', 'Master' , 'Doctor']

# 1 = 'Below College' , 2 = 'College' , 3 = 'Bachelor', 4 = 'Master' , 5 = 'Doctor'

Education_Table
```

	Education	Attrition_Yes	Attrition_No	Total
0	Below College	31	139	170
1	College	44	238	282
2	Bachelor	99	473	572
3	Master	58	340	398
4	Doctor	5	43	48

Next steps: Generate code with Education_Table View recommended plots

```
Education_Table['Attrition_Yes'] = round((Attrition_Yes['Attrition_Yes']/Total['Total'])*100,2)
Education_Table['Attrition_No'] = round((Attrition_No['Attrition_No']/Total['Total'])*100,2)

Education_perc = Education_Table.iloc[:,0:3]
Education_perc = Education_perc.rename(columns={'Attrition_Yes': 'Attrition_Yes(%)', 'Attrition_No': 'Attrition_No(

Education_perc
```

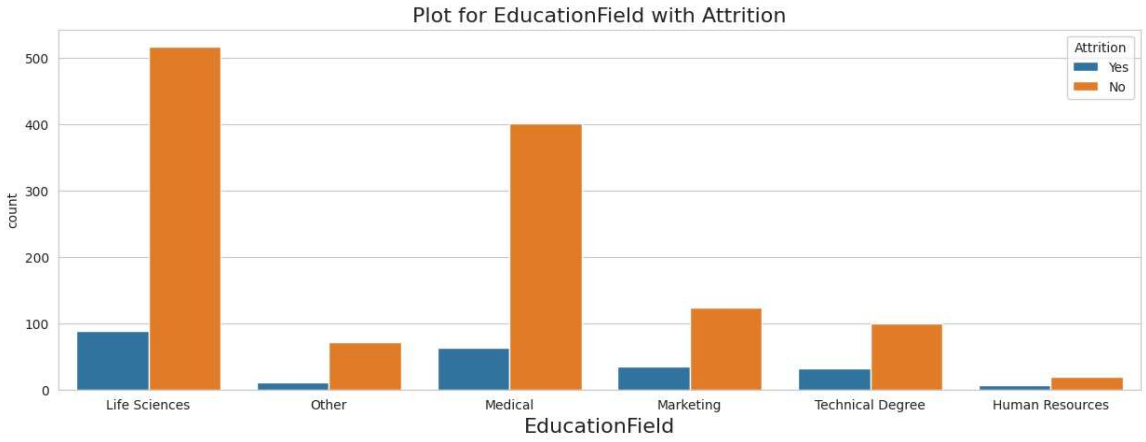
	Education	Attrition_Yes(%)	Attrition_No(%)
0	Below College	18.24	81.76
1	College	15.60	84.40
2	Bachelor	17.31	82.69
3	Master	14.57	85.43
4	Doctor	10.42	89.58

Next steps: Generate code with Education_perc View recommended plots

Education Field

```
plt.figure(figsize=(15,5))
sns.countplot(x = 'EducationField', hue = 'Attrition', data = Data)
plt.xlabel('EducationField', fontsize=16)
plt.title('Plot for EducationField with Attrition', fontsize=16)
```

```
Text(0.5, 1.0, 'Plot for EducationField with Attrition')
```




```
Attrition_Yes = Data.groupby(Data['EducationField'][Data['Attrition'] == 'Yes']).size().reset_index(name = "Attriti
Attrition_No = Data.groupby(Data['EducationField'][Data['Attrition'] == 'No']).size().reset_index(name = "Attrition

Total = Data.groupby(Data['EducationField']).size().reset_index(name = 'Total')

EducationField_Table = Attrition_Yes.join(Attrition_No['Attrition_No']).join(Total['Total'])

EducationField_Table
```

	EducationField	Attrition_Yes	Attrition_No	Total
0	Human Resources	7	20	27
1	Life Sciences	89	517	606
2	Marketing	35	124	159
3	Medical	63	401	464
4	Other	11	71	82
5	Technical Degree	32	100	132

Next steps: Generate code with EducationField_Table View recommended plots

```
EducationField_Table['Attrition_Yes'] = round((Attrition_Yes['Attrition_Yes']/Total['Total'])*100,2)
EducationField_Table['Attrition_No'] = round((Attrition_No['Attrition_No']/Total['Total'])*100,2)

EducationField_perc = EducationField_Table.iloc[:,0:3]
EducationField_perc = EducationField_perc.rename(columns={'Attrition_Yes': 'Attrition_Yes(%)', 'Attrition_No': 'Att

EducationField_perc
```

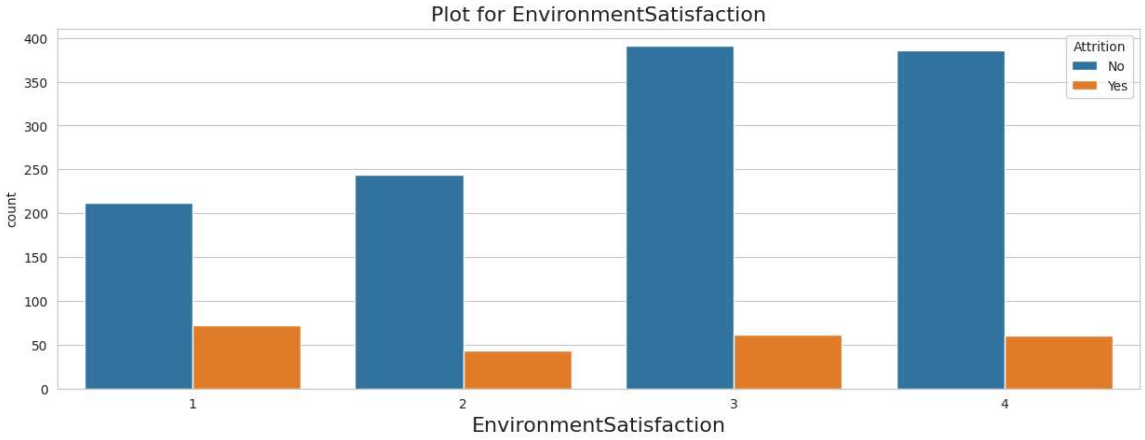
	EducationField	Attrition_Yes(%)	Attrition_No(%)
0	Human Resources	25.93	74.07
1	Life Sciences	14.69	85.31
2	Marketing	22.01	77.99
3	Medical	13.58	86.42
4	Other	13.41	86.59
5	Technical Degree	24.24	75.76

Next steps: Generate code with EducationField_perc View recommended plots

Environment Satisfaction

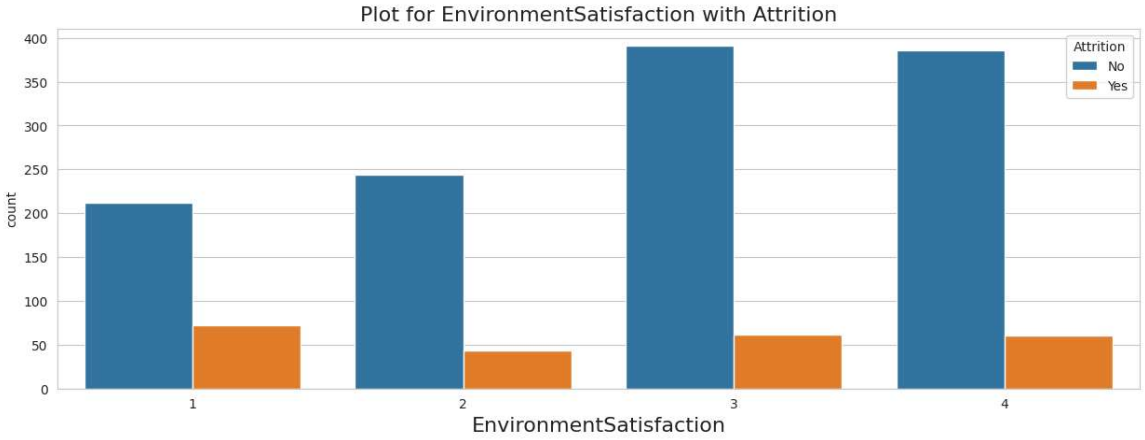
```
plt.figure(figsize=(15,5))
sns.countplot(x = 'EnvironmentSatisfaction',hue="Attrition" ,data = Data)
plt.xlabel('EnvironmentSatisfaction', fontsize=16)
plt.title('Plot for EnvironmentSatisfaction', fontsize=16)
```

```
Text(0.5, 1.0, 'Plot for EnvironmentSatisfaction')
```



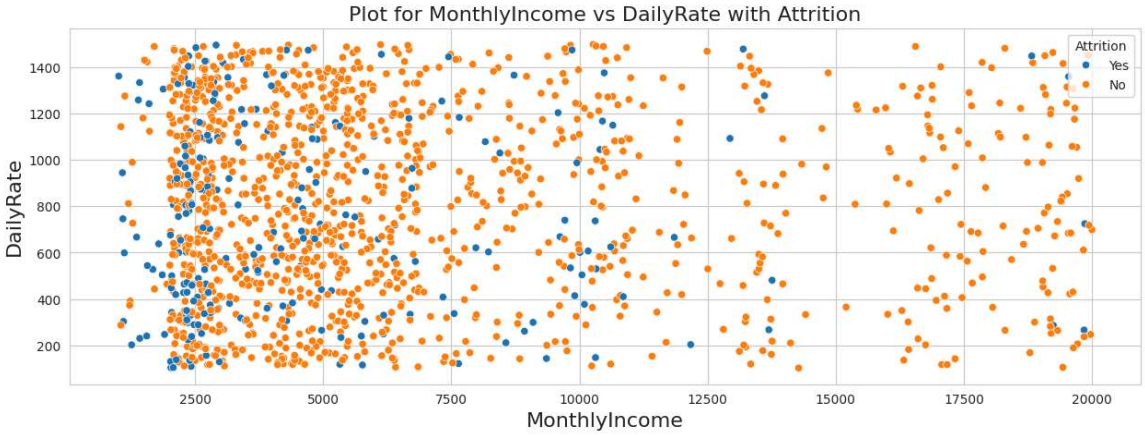

```
plt.figure(figsize=(15,5))
sns.countplot(x = 'EnvironmentSatisfaction', hue = 'Attrition', data = Data)
plt.xlabel('EnvironmentSatisfaction', fontsize=16)
plt.title('Plot for EnvironmentSatisfaction with Attrition', fontsize=16)
```

```
Text(0.5, 1.0, 'Plot for EnvironmentSatisfaction with Attrition')
```



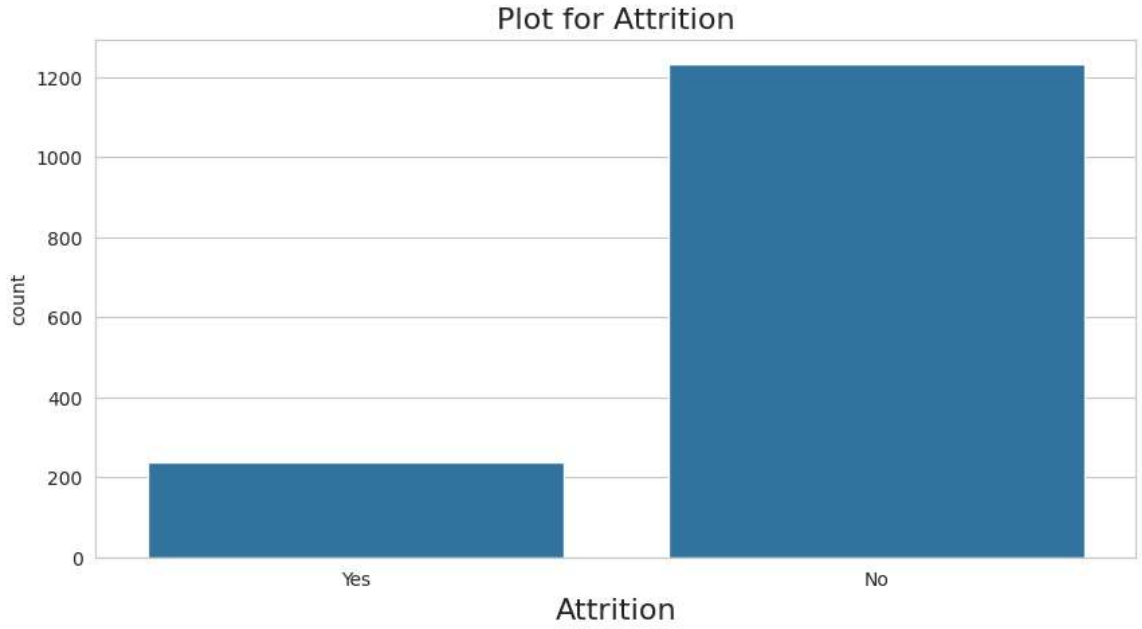

```
plt.figure(figsize=(15,5))
sns.scatterplot(x = 'MonthlyIncome', y = 'DailyRate', hue = 'Attrition', data = Data)
plt.xlabel('MonthlyIncome', fontsize=16)
plt.ylabel('DailyRate', fontsize=16)
plt.title('Plot for MonthlyIncome vs DailyRate with Attrition', fontsize=16)
```

```
Text(0.5, 1.0, 'Plot for MonthlyIncome vs DailyRate with Attrition')
```




```
plt.figure(figsize=(10,5))
sns.countplot(x = 'Attrition', data = Data)
plt.xlabel('Attrition', fontsize=16)
plt.title('Plot for Attrition', fontsize=16)
```

```
Text(0.5, 1.0, 'Plot for Attrition')
```



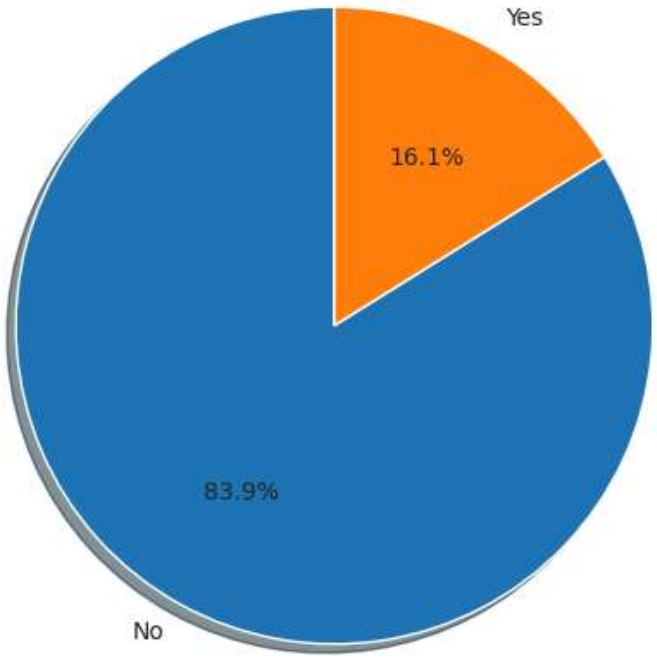
There is a high imbalance between yes and no

LABEL ENCODING

```
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()

#Changing Yes to 1 and No to 0


Data['Attrition'] = labelencoder.fit_transform(Data['Attrition'])


Data['Attrition'].value_counts()
```

```
Attrition
0    1233
1     237
Name: count, dtype: int64
```

```
Attrition = ['No', 'Yes']

data = [1233,237]

fig = plt.figure(figsize =(15,6))
plt.pie(data, labels = Attrition, autopct='%1.1f%%', shadow=True, startangle=90)
plt.show()
```



Separate the numerical & catagorical columns

```
Data_num = Data.select_dtypes(include=[np.number]).copy()
Data_num.head()
```

	Age	Attrition	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumbe
0	41	1	1102	1	2	1	
1	49	0	279	8	1	1	
2	37	1	1373	2	2	1	
3	33	0	1392	3	4	1	
4	27	0	591	2	1	1	

```
Data_char = Data.select_dtypes(include='object').copy()
Data_char.head()
```

	BusinessTravel	Department	EducationField	Gender	JobRole	MaritalStatus	Over18	OverTime
0	Travel_Rarely	Sales	Life Sciences	Female	Sales Executive	Single	Y	Yes
1	Travel_Frequently	Research & Development	Life Sciences	Male	Research Scientist	Married	Y	No
2	Travel_Rarely	Research & Development	Other	Male	Laboratory Technician	Single	Y	Yes
3	Travel_Frequently	Research & Development	Life Sciences	Female	Research Scientist	Married	Y	Yes
4	Travel_Rarely	Research & Development	Medical	Male	Laboratory Technician	Married	Y	No

Next steps: Generate code with `Data_char` View recommended plots

Creating Dummy variables

```
Data_dummy = pd.get_dummies(Data_char)

Data_dummy.head()
```

	BusinessTravel_Non-Travel	BusinessTravel_Travel_Frequently	BusinessTravel_Travel_Rarely
0	False	False	True
1	False	True	False
2	False	False	True
3	False	True	False
4	False	False	True

Combining the Data

```
Data_combined = pd.concat([Data_num,Data_dummy], axis=1)

Data_combined.head()
```

	Age	Attrition	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumbe
0	41	1	1102	1	2	1	
1	49	0	279	8	1	1	
2	37	1	1373	2	2	1	
3	33	0	1392	3	4	1	
4	27	0	591	2	1	1	

PreProcessing Data

```
x = Data_combined.drop(columns=['Attrition'])
y = Data_combined[['Attrition']]
```

Modeling

- Split data into Train & Test

```
from sklearn.model_selection import train_test_split
x_train, x_test,y_train, y_test = train_test_split(x,y, test_size = 0.25, random_state = 100)
```

```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(1102, 55)
(1102, 1)
(368, 55)
(368, 1)
```

Decision Tree

```
from sklearn import tree
model = tree.DecisionTreeClassifier()
```

```
model.fit(x_train,y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

```
y_pred = model.predict(x_test)
```

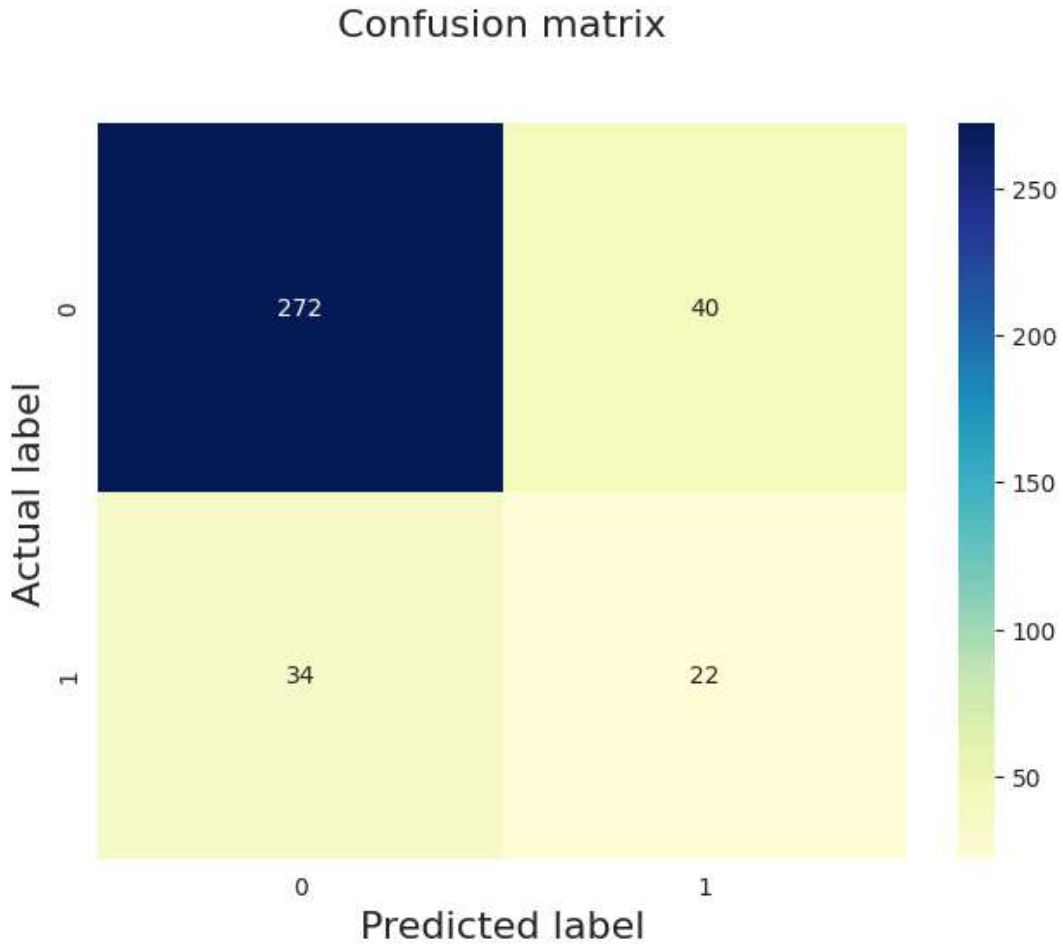
Confusion Matrix

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
print(cm)
```

```
[[272  40]
 [ 34  22]]
```

```
sns.heatmap(pd.DataFrame(cm), annot=True, cmap="YlGnBu" ,fmt='g')
plt.tight_layout()
plt.title('Confusion matrix', y=1.1, fontsize = 16)
plt.ylabel('Actual label', fontsize = 16)
plt.xlabel('Predicted label', fontsize = 16)
```

```
Text(0.5, 23.52222222222222, 'Predicted label')
```




```
from sklearn import metrics
metrics.accuracy_score(y_test,y_pred)
```

```
0.7989130434782609
```

```
Y_pred_test = model.predict(x_test)
Y_pred_train = model.predict(x_train)

print('Accuracy Train:',metrics.accuracy_score(y_train,Y_pred_train))
print('Accuracy Test:',metrics.accuracy_score(y_test,Y_pred_test))
```

```
Accuracy Train: 1.0
Accuracy Test: 0.7989130434782609
```

```
from sklearn import metrics

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: 0.7989130434782609
Precision: 0.3548387096774194
Recall: 0.39285714285714285
```

Area under ROC curve

```
from sklearn.metrics import roc_auc_score
roc_auc = roc_auc_score(y_test,model.predict_proba(x_test)[:,1])
print('Aera under ROC cure:',roc_auc)
```

```
Aera under ROC cure: 0.6323260073260073
```

Changing Criterion

Entropy with Max depth = 3

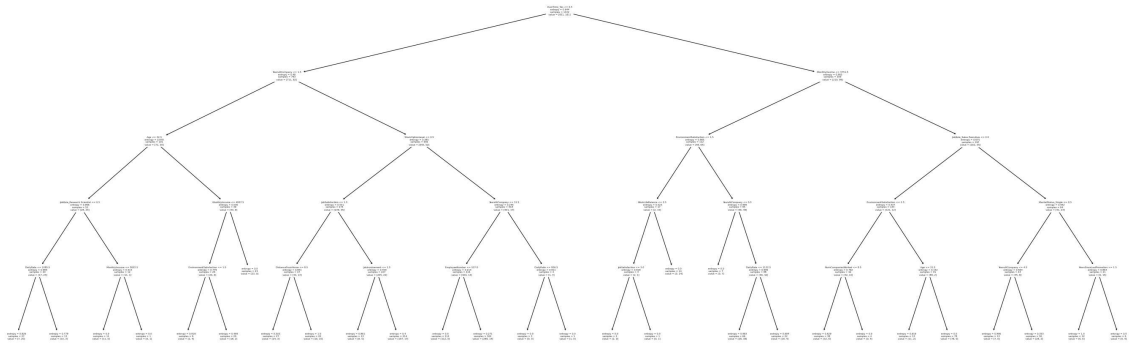
```
from sklearn import tree
model = tree.DecisionTreeClassifier(criterion='entropy',max_depth=5)
```

```
model.fit(x_train,y_train)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=5)
```

```
y_pred = model.predict(x_test)
```

```
from sklearn import tree
plt.figure(figsize=(35,12))
tree.plot_tree(model,feature_names=x_train.columns.values);
```



```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
print(cm)
```

```
[[293  19]
 [ 40  16]]
```

```
Y_pred_test = model.predict(x_test)
Y_pred_train = model.predict(x_train)

print('Accuracy Train:',metrics.accuracy_score(y_train,Y_pred_train))
print('Accuracy Test:',metrics.accuracy_score(y_test,Y_pred_test))
```

```
Accuracy Train: 0.8938294010889292
Accuracy Test: 0.8396739130434783
```

```
from sklearn import metrics
metrics.accuracy_score(y_test,y_pred)
```

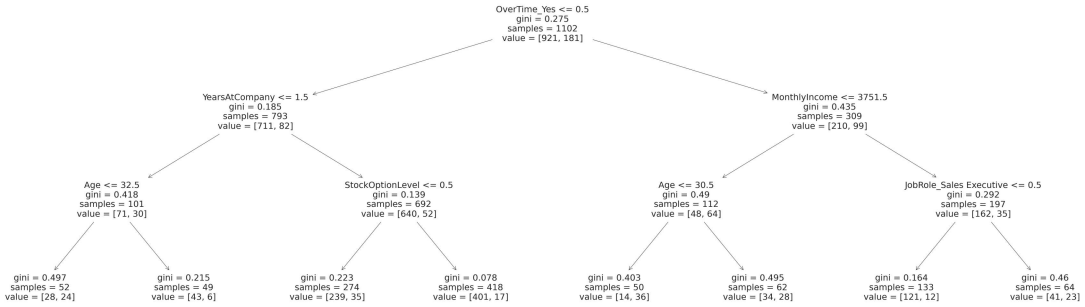
```
0.8396739130434783
```

Gini with Max depth = 3

```
from sklearn import tree
model = tree.DecisionTreeClassifier(criterion='gini', max_depth = 3)


model.fit(x_train,y_train)

y_pred = model.predict(x_test)

from sklearn import metrics
metrics.accuracy_score(y_test,y_pred)
```

```
0.8614130434782609
```

```
from sklearn import tree
plt.figure(figsize=(65,20))
tree.plot_tree(model,feature_names=x_train.columns.values);
```



Decision Tree Implementation

```
from sklearn import tree
dt_model = tree.DecisionTreeClassifier(criterion='gini',class_weight='balanced', max_depth=15,random_state=15)

dt_model.fit(x_train, y_train)

Y_pred_test = dt_model.predict(x_test)
Y_pred_train = dt_model.predict(x_train)

print('Accuracy Train:',metrics.accuracy_score(y_train,Y_pred_train))
print('Accuracy Test:',metrics.accuracy_score(y_test,Y_pred_test))
```

```
Accuracy Train: 0.9954627949183303
Accuracy Test: 0.7853260869565217
```

```
metrics.accuracy_score(y_test, Y_pred_test)
tree1_auc = roc_auc_score(y_test, Y_pred_test)
print("Decision Tree AUC:", tree1_auc)
```

```
Decision Tree AUC: 0.5950091575091575
```

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import classification_report
DT = dt_model.fit(x_train, y_train)

print ("\n\n ---Decision Tree Model---")
roc_auc = roc_auc_score(y_test, dt_model.predict(x_test))
print ("Random Forest AUC = %2.2f" % roc_auc)
print(classification_report(y_test, dt_model.predict(x_test)))
```

```
 ---Decision Tree Model---
Random Forest AUC = 0.60
              precision    recall  f1-score   support

           0       0.88      0.87      0.87       312
           1       0.31      0.32      0.31        56

    accuracy                           0.79       368
   macro avg       0.59      0.60      0.59       368
weighted avg       0.79      0.79      0.79       368
```

RandomForest Implementation

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, roc_auc_score, classification_report

# Initial RandomForestClassifier
rf_model = RandomForestClassifier(n_estimators=150, criterion='entropy', class_weight='balanced', max_leaf_nodes=20
rf_model.fit(x_train, y_train)

# Predictions
Y_pred_train = rf_model.predict(x_train)
Y_pred_test = rf_model.predict(x_test)

# Evaluation
print('Accuracy Train:', accuracy_score(y_train, Y_pred_train))
print('Accuracy Test:', accuracy_score(y_test, Y_pred_test))
```

```
Accuracy Train: 0.8820326678765881
Accuracy Test: 0.7934782608695652
```

```
rf_roc_auc = roc_auc_score(y_test, rf_model.predict_proba(x_test)[:,1])
print('Area under ROC curve:', rf_roc_auc)
```

```
Area under ROC curve: 0.7468234890109889
```

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import classification_report
rf = rf_model.fit(x_train, y_train)

print ("\n\n ---Random Forest Model---")
rf_roc_auc = roc_auc_score(y_test, rf_model.predict(x_test))
print ("Random Forest AUC = %2.2f" % rf_roc_auc)
print(classification_report(y_test, rf_model.predict(x_test)))
```

```
 ---Random Forest Model---
Random Forest AUC = 0.56
              precision    recall  f1-score   support

           0       0.86      0.98      0.92       312
           1       0.57      0.14      0.23        56

    accuracy                           0.85       368
   macro avg       0.72      0.56      0.57       368
weighted avg       0.82      0.85      0.81       368
```

```
from sklearn.metrics import roc_curve
rf_FPR,rf_TPR,rf_Thresholds = roc_curve(y_test,rf_model.predict_proba(x_test)[:,1])

fpr_series = pd.Series(rf_FPR)
tpr_series = pd.Series(rf_TPR)
thresholds_series = pd.Series(rf_Thresholds)

df = pd.concat([fpr_series,tpr_series,thresholds_series],axis=1,keys=['FPR','TPR','THRESHOLD'])
df.sort_values(by='TPR',ascending=False)
```

	FPR	TPR	THRESHOLD
51	1.000000	1.000000	0.00
50	0.993590	1.000000	0.01
49	0.935897	1.000000	0.03
48	0.907051	0.982143	0.04
47	0.849359	0.946429	0.05
46	0.810897	0.910714	0.06
45	0.766026	0.892857	0.07
44	0.695513	0.857143	0.08
43	0.657051	0.839286	0.09
42	0.612179	0.839286	0.10
41	0.551282	0.821429	0.11
40	0.509615	0.821429	0.12
39	0.483974	0.785714	0.13
38	0.467949	0.767857	0.14
37	0.410256	0.750000	0.15
36	0.362179	0.732143	0.16
35	0.342949	0.714286	0.17
34	0.320513	0.714286	0.18
33	0.288462	0.696429	0.19
32	0.262821	0.660714	0.20
31	0.240385	0.660714	0.21
30	0.214744	0.625000	0.22
29	0.208333	0.607143	0.23
27	0.176282	0.589286	0.25
28	0.182692	0.589286	0.24
26	0.160256	0.589286	0.26
25	0.150641	0.571429	0.27
24	0.141026	0.535714	0.28
23	0.121795	0.500000	0.29
22	0.108974	0.482143	0.30
21	0.102564	0.464286	0.31
20	0.092949	0.428571	0.32
19	0.086538	0.410714	0.35
18	0.080128	0.375000	0.36
17	0.070513	0.357143	0.38

16	0.057692	0.357143	0.39
15	0.054487	0.285714	0.40
13	0.044872	0.267857	0.42
14	0.048077	0.267857	0.41
12	0.032051	0.232143	0.44
11	0.022436	0.232143	0.45
10	0.022436	0.214286	0.46
9	0.022436	0.178571	0.47
8	0.022436	0.142857	0.50
7	0.016026	0.142857	0.52
6	0.012821	0.125000	0.53
5	0.006410	0.107143	0.60
4	0.006410	0.071429	0.61
2	0.000000	0.053571	0.73
3	0.006410	0.053571	0.62
1	0.000000	0.017857	0.94
0	0.000000	0.000000	1.94

Hyper parameter tuning

```
model_params = {'n_estimators' :[140,145,150,155,160], 'max_leaf_nodes':range(10,20),'criterion':['gini','entropy']


rf_model = RandomForestClassifier(random_state = 15)
```

Random Search CV

```
from sklearn.model_selection import RandomizedSearchCV
random_search_object = RandomizedSearchCV(rf_model, model_params,n_iter = 10, cv = 3,random_state = 15)

random_search_best_model = random_search_object.fit(x_train,y_train)


Y_pred_test = random_search_best_model.predict(x_test)
Y_pred_train = random_search_best_model.predict(x_train)

print('Accuracy Train:',metrics.accuracy_score(y_train,Y_pred_train))
print('Accuracy Test:',metrics.accuracy_score(y_test,Y_pred_test))
```

```
Accuracy Train: 0.8720508166969148
Accuracy Test: 0.8586956521739131
```

```
from sklearn.metrics import roc_auc_score
rscv_roc_auc = roc_auc_score(y_test,random_search_object.predict_proba(x_test)[:,1])
print('Aa under ROC cure:',rscv_roc_auc)
```

```
Aera under ROC cure: 0.724988553113553
```

```
random_search_best_model.best_params_
```

```
{'n_estimators': 145, 'max_leaf_nodes': 17, 'criterion': 'gini'}
```

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import classification_report
Rs = random_search_best_model.fit(x_train, y_train)

print ("\n\n ---RandomSearchCV Model---")
Rs_roc_auc = roc_auc_score(y_test, random_search_best_model.predict(x_test))
print ("Random search AUC = %2.2f" % Rs_roc_auc)
print(classification_report(y_test, random_search_best_model.predict(x_test)))
```

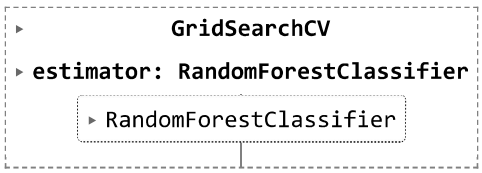
```
 ---RandomSearchCV Model---
Random search AUC = 0.55
              precision    recall  f1-score   support

           0       0.86      0.99      0.92       312
           1       0.75      0.11      0.19        56

    accuracy                           0.86       368
   macro avg       0.81      0.55      0.56       368
weighted avg       0.84      0.86      0.81       368
```

Grid Search

```
from sklearn.model_selection import GridSearchCV
grid_search_object = GridSearchCV(rf_model, model_params, cv = 3)
```

```
grid_search_best_model = grid_search_object.fit(x_train,y_train)
```

```
grid_search_best_model
```




```
Y_pred_test = grid_search_best_model.predict(x_test)
Y_pred_train = grid_search_best_model.predict(x_train)

print('Accuracy Train:',metrics.accuracy_score(y_train,Y_pred_train))
print('Accuracy Test:',metrics.accuracy_score(y_test,Y_pred_test))
```

```
Accuracy Train: 0.8711433756805808
Accuracy Test: 0.8586956521739131
```

```
from sklearn.metrics import roc_auc_score
gscv_roc_auc = roc_auc_score(y_test,grid_search_best_model.predict_proba(x_test)[:,1])
print('Area under ROC curve:',gscv_roc_auc)
```

```
Area under ROC curve: 0.725103021978022
```

```
from sklearn.metrics import classification_report

gscv =grid_search_best_model.fit(x_train, y_train)

print ("\n\n ---Grid Search Model---")
gscv_roc_aucc = roc_auc_score(y_test, grid_search_best_model.predict(x_test))
print ("Grid Search AUC = %2.2f" % gscv_roc_auc)
print(classification_report(y_test, grid_search_best_model.predict(x_test)))
```

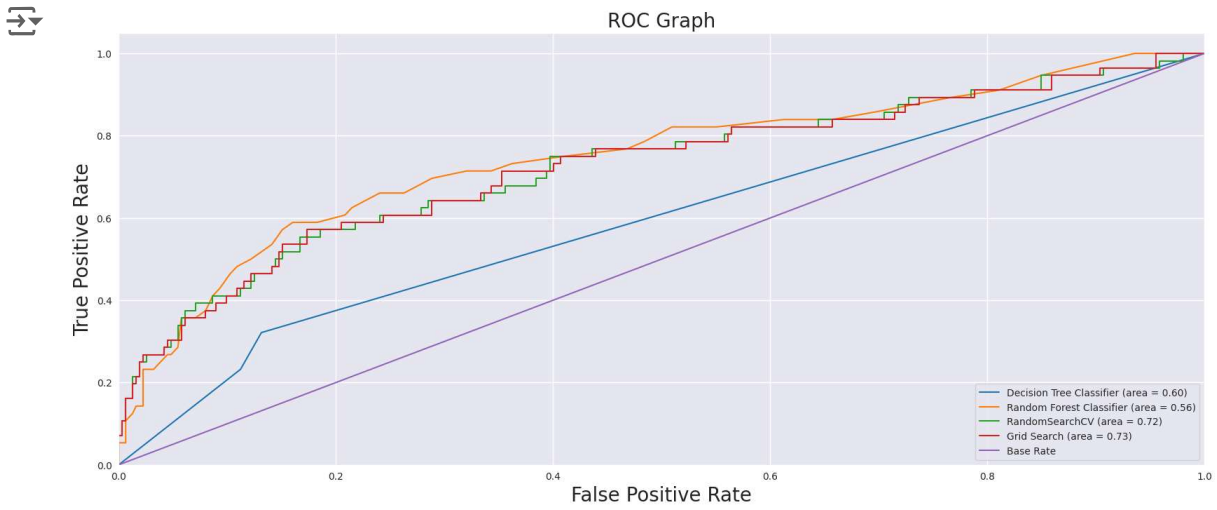
```
 ---Grid Search Model---
Grid Search AUC = 0.73
              precision    recall  f1-score   support

           0       0.86      0.99      0.92       312
           1       0.75      0.11      0.19        56

    accuracy                           0.86       368
   macro avg       0.81      0.55      0.56       368
weighted avg       0.84      0.86      0.81       368
```

Start coding or generate with AI.

```
rf_model.fit(x_train, y_train)
```

```
RandomForestClassifier(random_state=15)
```

```
# Create ROC Graph
from sklearn.metrics import roc_curve

df = pd.concat([fpr_series,tpr_series,thresholds_series],axis=1,keys=['FPR','TPR','THRESHOLD'])
dt_FPR,dt_TPR,dt_Thresholds = roc_curve(y_test, dt_model.predict_proba(x_test)[:,1])
rf_FPR,rf_TPR,rf_Thresholds = roc_curve(y_test, rf_model.predict_proba(x_test)[:,1])
Rs_FPR,Rs_TPR,Rs_Thresholds = roc_curve(y_test, random_search_best_model.predict_proba(x_test)[:,1])
Gs_FPR,Gs_TPR,Gs_Thresholds = roc_curve(y_test, grid_search_best_model.predict_proba(x_test)[:,1])

sns.set_style("darkgrid")
plt.figure(figsize=(20,8))

# Plot Decision Tree ROC
plt.plot(dt_FPR,dt_TPR, label='Decision Tree Classifier (area = %0.2f)' % roc_auc)

# Plot Random Forest ROC
plt.plot(rf_FPR,rf_TPR, label='Random Forest Classifier (area = %0.2f)' % rf_roc_auc)

# Plot Decision Tree ROC
plt.plot(Rs_FPR,Rs_TPR, label='RandomSearchCV (area = %0.2f)' % rscv_roc_auc)

# Plot Grid Search ROC
plt.plot(Gs_FPR,Gs_TPR, label='Grid Search (area = %0.2f)' % gscv_roc_auc)


# Plot Base Rate ROC
plt.plot([0,1], [0,1],label='Base Rate')

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize = 20)
plt.ylabel('True Positive Rate', fontsize = 20)
plt.title('ROC Graph', fontsize = 20)
plt.legend(loc="lower right")
plt.show()
```




```
confusion_matrix(y_test, dt_model.predict(x_test))
```

```
array([[271,  41],
       [ 38,  18]])
```

```
confusion_matrix(y_test, rf_model.predict(x_test))
```

```
array([[306,   6],
       [ 48,   8]])
```

```
confusion_matrix(y_test, random_search_best_model.predict(x_test))
```

```
array([[310,   2],
       [ 50,   6]])
```

```
confusion_matrix(y_test, grid_search_best_model.predict(x_test))
```

```
array([[310,   2],
       [ 50,   6]])
```

```
# Get Feature Importances
feature_importances = pd.DataFrame(rf_model.feature_importances_, index = x_train.columns,
                                   columns=['importance']).sort_values('importance', ascending=False)
feature_importances = feature_importances.reset_index()
feature_importances
```

	index	importance
0	MonthlyIncome	0.070123
1	Age	0.053042
2	TotalWorkingYears	0.047649
3	MonthlyRate	0.044219
4	YearsAtCompany	0.043391
5	DailyRate	0.043120
6	EmployeeNumber	0.041581
7	HourlyRate	0.040879
8	OverTime_Yes	0.038840
9	DistanceFromHome	0.037098
10	YearsWithCurrManager	0.030840
11	OverTime_No	0.030465
12	NumCompaniesWorked	0.030139
13	YearsInCurrentRole	0.028750