Importing Libraries

In [230]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import model_selection
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import warnings
```

Loading the dataset

In [231]:

In [232]:
```
#Loading the top 5 rows
```

Out[232]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumber	...	R
0	37	No	Travel_Rarely	1372	Research & Development	1	3	Life Sciences	1	391	...	
1	37	No	Travel_Rarely	1439	Research & Development	4	1	Life Sciences	1	1394	...	
2	42	No	Travel_Rarely	635	Sales	1	1	Life Sciences	1	387	...	
3	35	No	Travel_Rarely	1402	Sales	28	4	Life Sciences	1	1554	...	
4	24	No	Travel_Rarely	1371	Sales	10	4	Marketing	1	507	...	

5 rows × 35 columns

Data cleaning and Preprocessing

In [233]:

Out[233]:
```
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
```

In [234]:

Out[234]:
```
0
```

In [235]:

Out[235]:
```
Age                          43
Attrition                     2
BusinessTravel                3
DailyRate                   703
Department                    3
DistanceFromHome             29
Education                     5
EducationField                6
EmployeeCount                 1
EmployeeNumber             1029
EnvironmentSatisfaction       4
Gender                        2
HourlyRate                   71
JobInvolvement                4
JobLevel                      5
JobRole                       9
JobSatisfaction               4
MaritalStatus                 3
MonthlyIncome               961
MonthlyRate                1006
NumCompaniesWorked           10
Over18                        1
OverTime                      2
PercentSalaryHike            15
PerformanceRating             2
RelationshipSatisfaction      4
StandardHours                 1
StockOptionLevel              4
TotalWorkingYears            40
TrainingTimesLastYear         7
WorkLifeBalance               4
YearsAtCompany               36
YearsInCurrentRole           19
YearsSinceLastPromotion      16
YearsWithCurrManager         17
dtype: int64
```

The data has no missing values nor duplicates. The data is ready for analysis

In [236]:
```
# dropping these two columns because they only have one value
```

Exploratory Data Analysis

In [285]:
```
#Checking the shape of the dataset
```

Out[285]:
```
(1029, 33)
```

There are 33 columns and 1029 rows in this dataset.

In [286]:
```
# Checking more information of the dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1029 entries, 0 to 1028
Data columns (total 33 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   Age                       1029 non-null   int64 
 1   Attrition                 1029 non-null   int64 
 2   BusinessTravel            1029 non-null   object
 3   DailyRate                 1029 non-null   int64 
 4   Department                1029 non-null   object
 5   DistanceFromHome          1029 non-null   int64 
 6   Education                 1029 non-null   int64 
 7   EducationField            1029 non-null   object
 8   EmployeeNumber            1029 non-null   int64 
 9   EnvironmentSatisfaction   1029 non-null   int64 
 10  Gender                    1029 non-null   object
 11  HourlyRate                1029 non-null   int64 
 12  JobInvolvement            1029 non-null   int64 
 13  JobLevel                  1029 non-null   int64 
 14  JobRole                   1029 non-null   object
 15  JobSatisfaction           1029 non-null   int64 
 16  MaritalStatus             1029 non-null   object
 17  MonthlyIncome             1029 non-null   int64 
 18  MonthlyRate               1029 non-null   int64 
 19  NumCompaniesWorked        1029 non-null   int64 
 20  Over18                    1029 non-null   object
 21  OverTime                  1029 non-null   object
 22  PercentSalaryHike         1029 non-null   int64 
 23  PerformanceRating         1029 non-null   int64 
 24  RelationshipSatisfaction  1029 non-null   int64 
 25  StockOptionLevel          1029 non-null   int64 
 26  TotalWorkingYears         1029 non-null   int64 
 27  TrainingTimesLastYear     1029 non-null   int64 
 28  WorkLifeBalance           1029 non-null   int64 
 29  YearsAtCompany            1029 non-null   int64 
 30  YearsInCurrentRole        1029 non-null   int64 
 31  YearsSinceLastPromotion   1029 non-null   int64 
 32  YearsWithCurrManager      1029 non-null   int64 
dtypes: int64(25), object(8)
memory usage: 265.4+ KB
```

In [287]:
```
#Extracting the columns that are of datatype object
```

Out[287]:
```
BusinessTravel    object
Department        object
EducationField    object
Gender            object
JobRole           object
MaritalStatus     object
Over18            object
OverTime          object
dtype: object
```

These are the categorical columns in the dataset.

Univariate analysis of the Target column

In [240]:

Out[240]:
```
No     868
Yes    161
Name: Attrition, dtype: int64
```

In [241]:
```
Attrition = data.query("Attrition == 'Yes'")
```

```
161 employees in the dataset left the company.
```

In [242]:
```
# Let's encode the attrition column so we can use it for EDA
data['Attrition'] = data['Attrition'].factorize(['No','Yes'])[0]
```

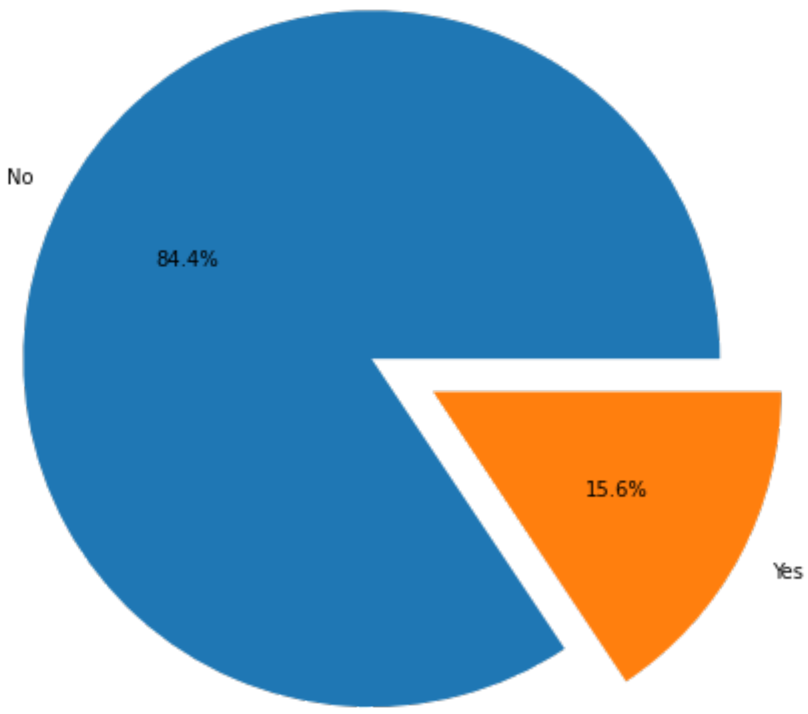
Out[242]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeNumber	EnvironmentSatisfactio
0	37	0	Travel_Rarely	1372	Research & Development	1	3	Life Sciences	391	
1	37	0	Travel_Rarely	1439	Research & Development	4	1	Life Sciences	1394	
2	42	0	Travel_Rarely	635	Sales	1	1	Life Sciences	387	
3	35	0	Travel_Rarely	1402	Sales	28	4	Life Sciences	1554	
4	24	0	Travel_Rarely	1371	Sales	10	4	Marketing	507	

5 rows × 33 columns

Attrition: No = 0 Yes = 1

In [243]:
```
plt.figure(figsize=(8,8))
pie = data.groupby('Attrition')['Attrition'].count()
```



84% of the employees in the dataset have not left the company.

In [288]:
```
#Extracting columns of the dataset that are of datatype int
```

Out[288]:
```
Age                         int64
Attrition                   int64
DailyRate                   int64
DistanceFromHome            int64
Education                   int64
EmployeeNumber              int64
EnvironmentSatisfaction     int64
HourlyRate                  int64
JobInvolvement              int64
JobLevel                    int64
JobSatisfaction             int64
MonthlyIncome               int64
MonthlyRate                 int64
NumCompaniesWorked          int64
PercentSalaryHike           int64
PerformanceRating           int64
RelationshipSatisfaction    int64
StockOptionLevel            int64
TotalWorkingYears           int64
TrainingTimesLastYear       int64
WorkLifeBalance             int64
YearsAtCompany              int64
YearsInCurrentRole          int64
YearsSinceLastPromotion     int64
YearsWithCurrManager        int64
dtype: object
```

These columns are numeric.

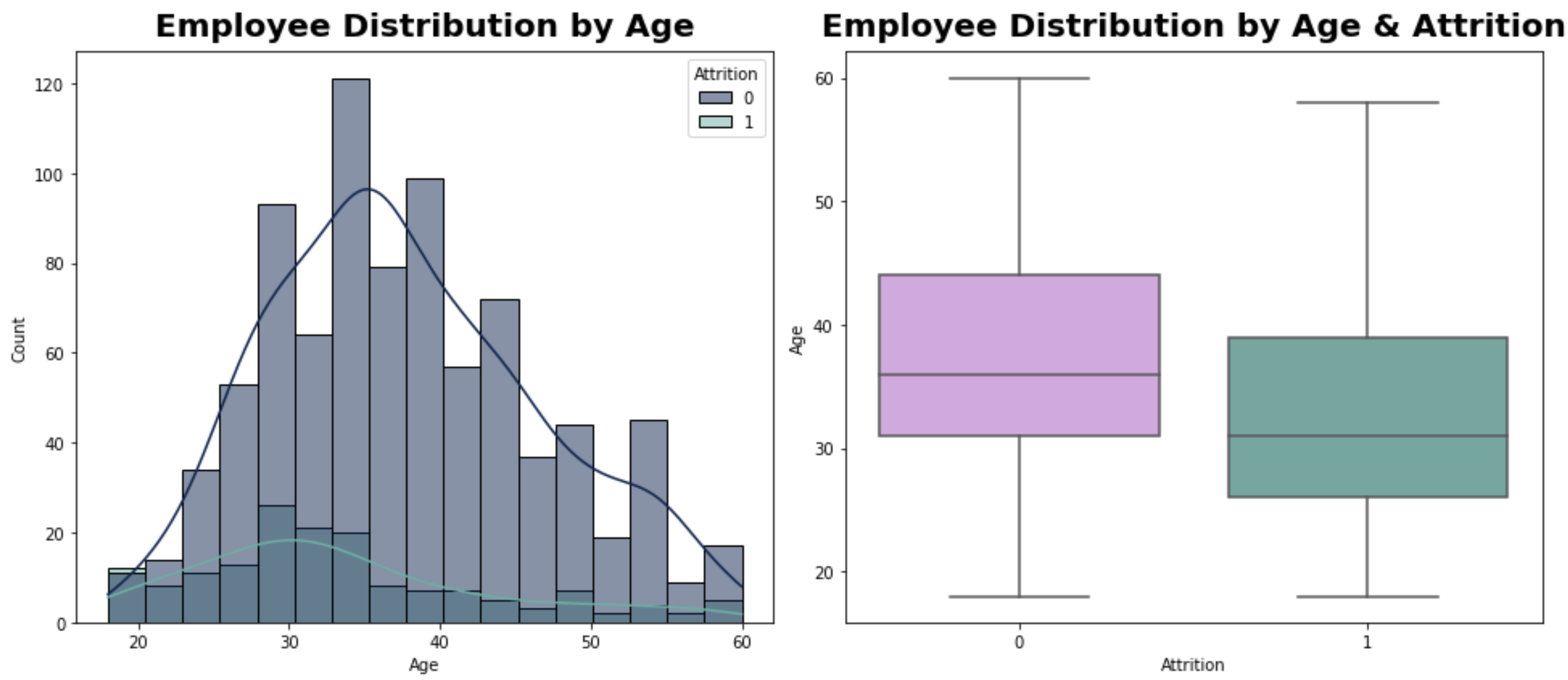
Age vs Attrition

In [289]:
```
#Checking the distribution of the age column
```

Out[289]:
```
Age
35      57
36      52
34      46
29      44
32      43
31      42
38      40
30      39
33      38
28      36
dtype: int64
```

In [246]:
```
#Visualization to show Employee Distribution by Age.
plt.figure(figsize=(13.5,6))
plt.subplot(1,2,1)
sns.histplot(x="Age",hue="Attrition",data=data,kde=True,palette=["#11264e","#6faea4"])
plt.title("Employee Distribution by Age",fontweight="black",size=20,pad=10)


#Visualization to show Employee Distribution by Age & Attrition.
plt.subplot(1,2,2)
sns.boxplot(x="Attrition",y="Age",data=data,palette=["#D4A1E7","#6faea4"])
plt.title("Employee Distribution by Age & Attrition",fontweight="black",size=20,pad=10)
plt.tight_layout()
```



1.Most of the emloyees are between age 30 to 40.

2.We can clearly observe a trend that as the age is increasing the attrition is decreasing.

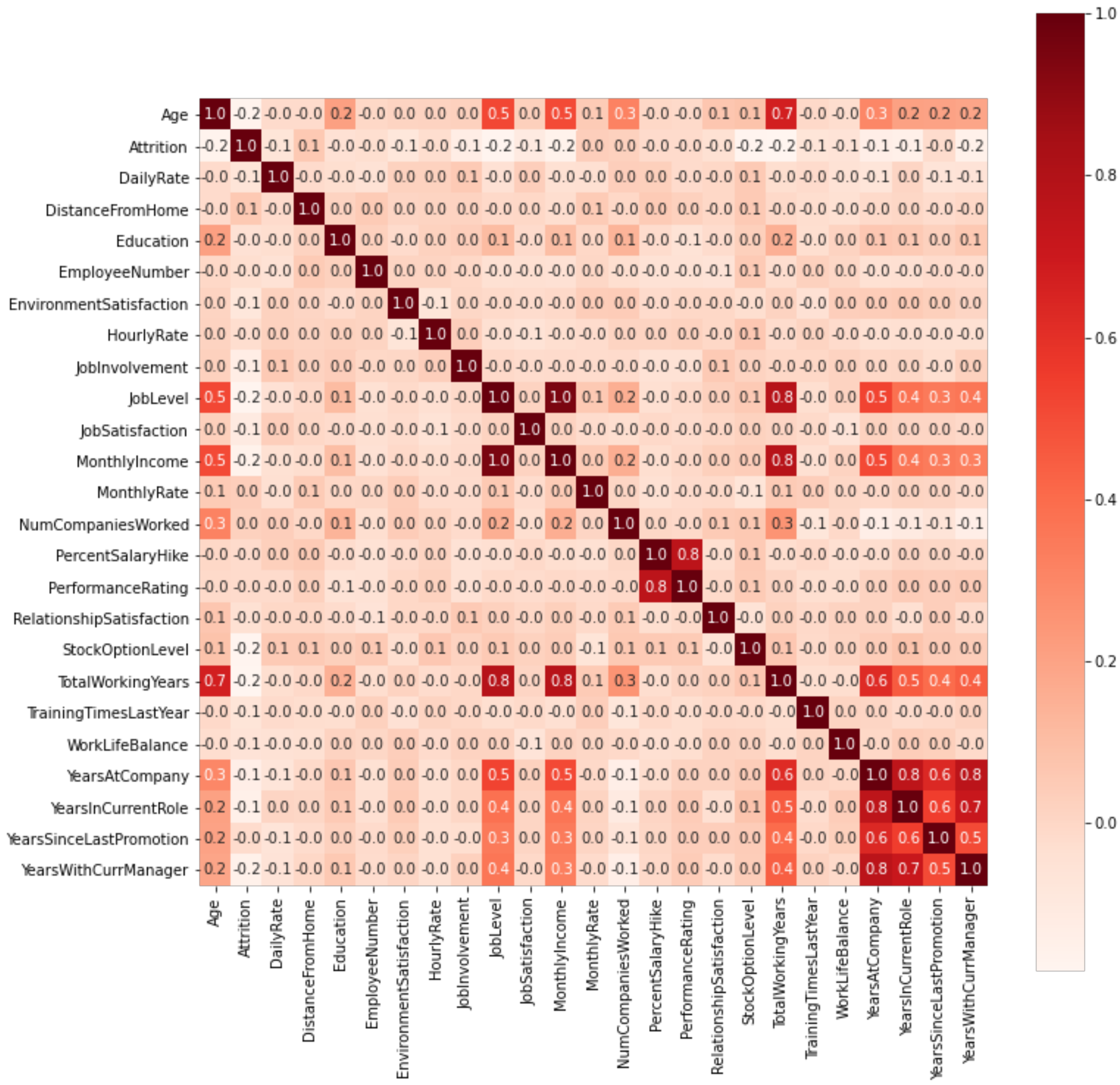
3.From the boxplot we can also observe that the medain age of employee who left the organization is less than the employees who are working in the organization.

4.Employees with young age leaves the company more compared to elder employees.

Correlation Analysis

In [247]:
```
# We can use a heatmap to check correlation between the variables.
corr = data.corr()
plt.figure(figsize=(12,12))
```

Out[247]: <AxesSubplot:>



As we can see, there isn't a very strong correlation of the target column with any of the numerical columns. But we can see other correlations such as;

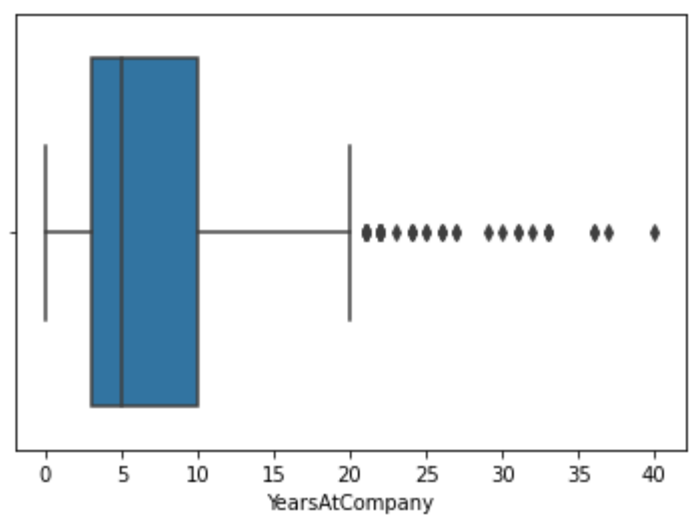
```
More senior employees have higher total working years (very obvious)
Higher performance ratings lead to salary hike percentage to increase
The more years an employee puts in, the more their monthly income increases
A lot of employees remain in their current role and also under the same manager as years pass by meaning they
don't get promotion and this could be a major factor contributing to attrition
```

From here, we can deduct that the lack of promotions may be a crucial factor to attritions.

Years At Company

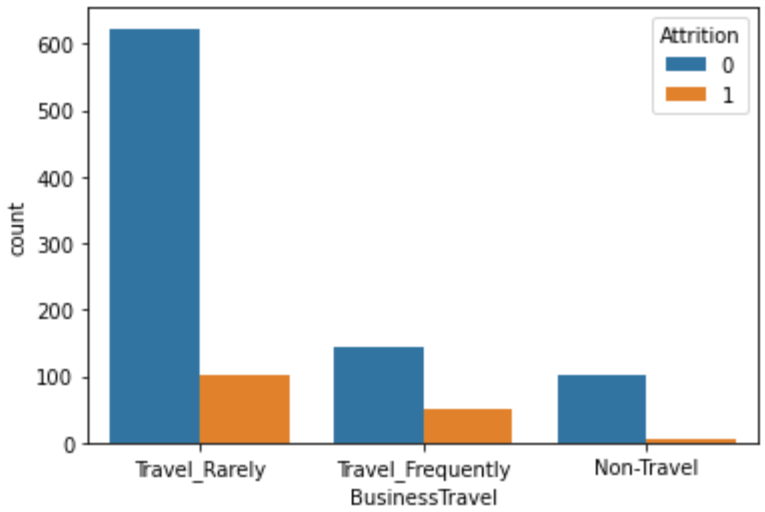
In [248]:

Out[248]: <AxesSubplot:xlabel='YearsAtCompany'>



Most employees remain in the company for 3-9 years with median being 5 years.

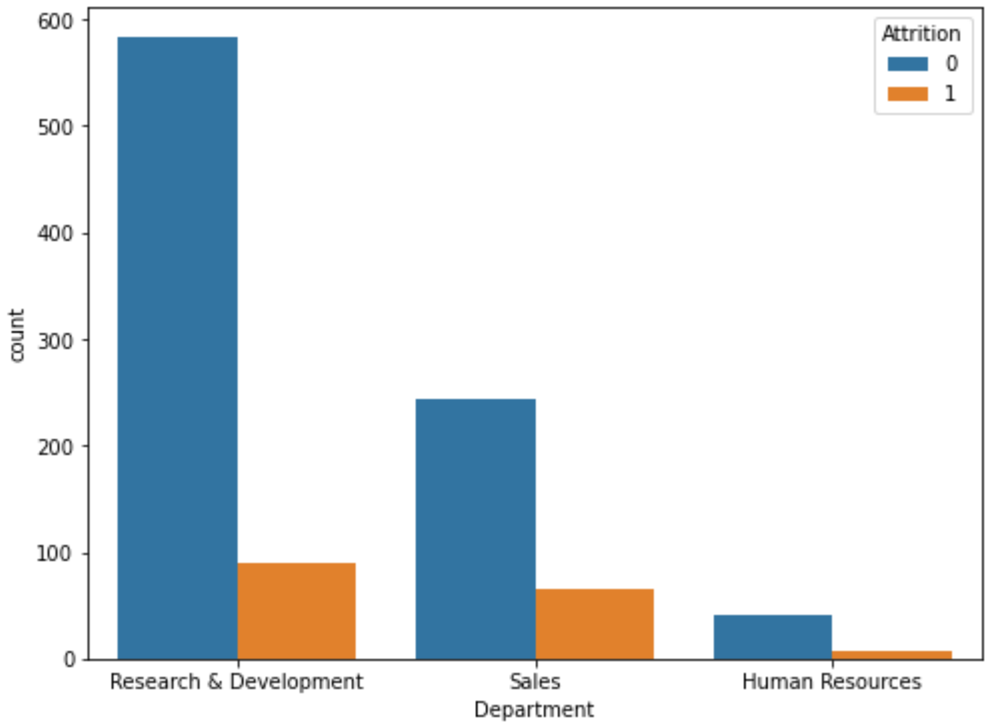
In [249]:



Most employees who travel rarely don't leave the company. From the plot we can tell, sending employees on business travels or not doesn't really make much of a difference and doesn't have a significant effect on attrition.

Department

In [250]:
```
plt.figure(figsize=(8,6))
```



In [251]:

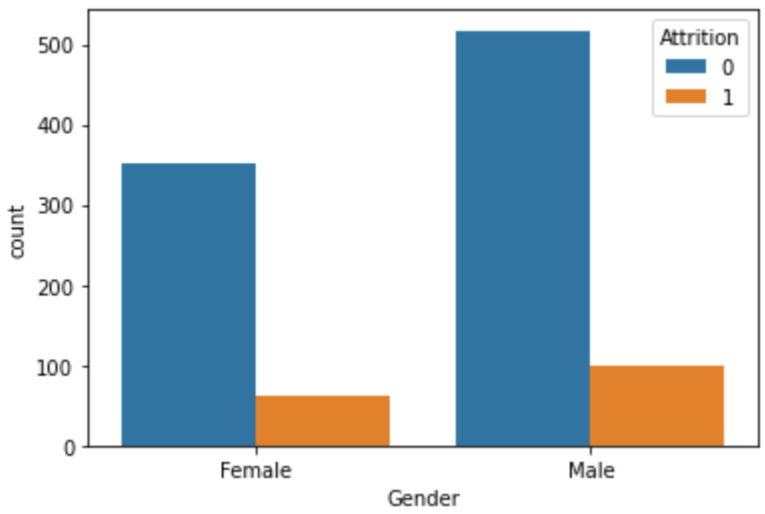
Out[251]:
```
Research & Development    672
Sales                     309
Human Resources            48
Name: Department, dtype: int64
```

Most attritions are from the research & development department only for sales department to come second by a small margin. Human resources has the least number of attritions. But we need to keep in mind that R&D has a lot more employees than sales and HR.

If we considered percentage of attritions per department, we would see that the HR department has most attritions.

Gender

In [252]:



Clearly there are more males in the organisation than females, so attritions are higher but slightly. I don't think gender is too significant a factor behind attritions.

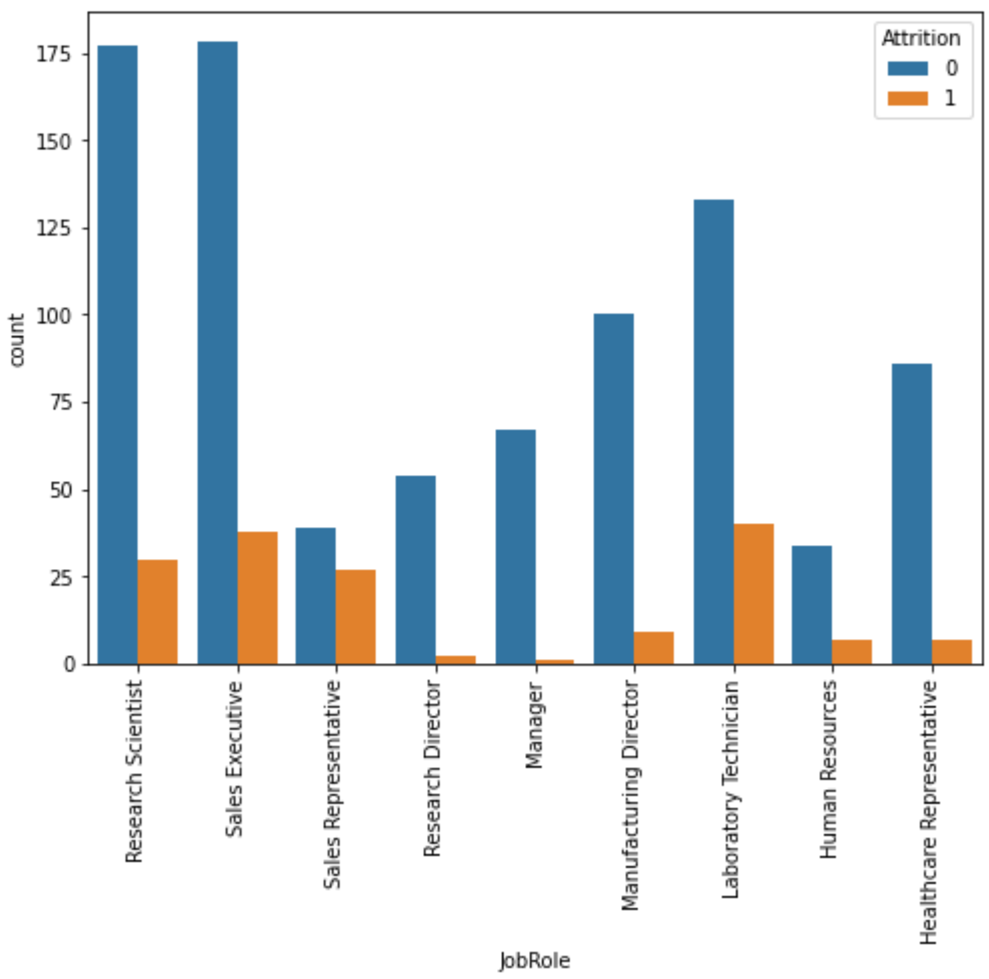
JobRole

In [253]:

```
plt.figure(figsize=(8,6))
sns.countplot(x='JobRole', hue='Attrition', data=data);
```

Out[253]:

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8]),
 [Text(0, 0, 'Research Scientist'),
  Text(1, 0, 'Sales Executive'),
  Text(2, 0, 'Sales Representative'),
  Text(3, 0, 'Research Director'),
  Text(4, 0, 'Manager'),
  Text(5, 0, 'Manufacturing Director'),
  Text(6, 0, 'Laboratory Technician'),
  Text(7, 0, 'Human Resources'),
  Text(8, 0, 'Healthcare Representative')])
```

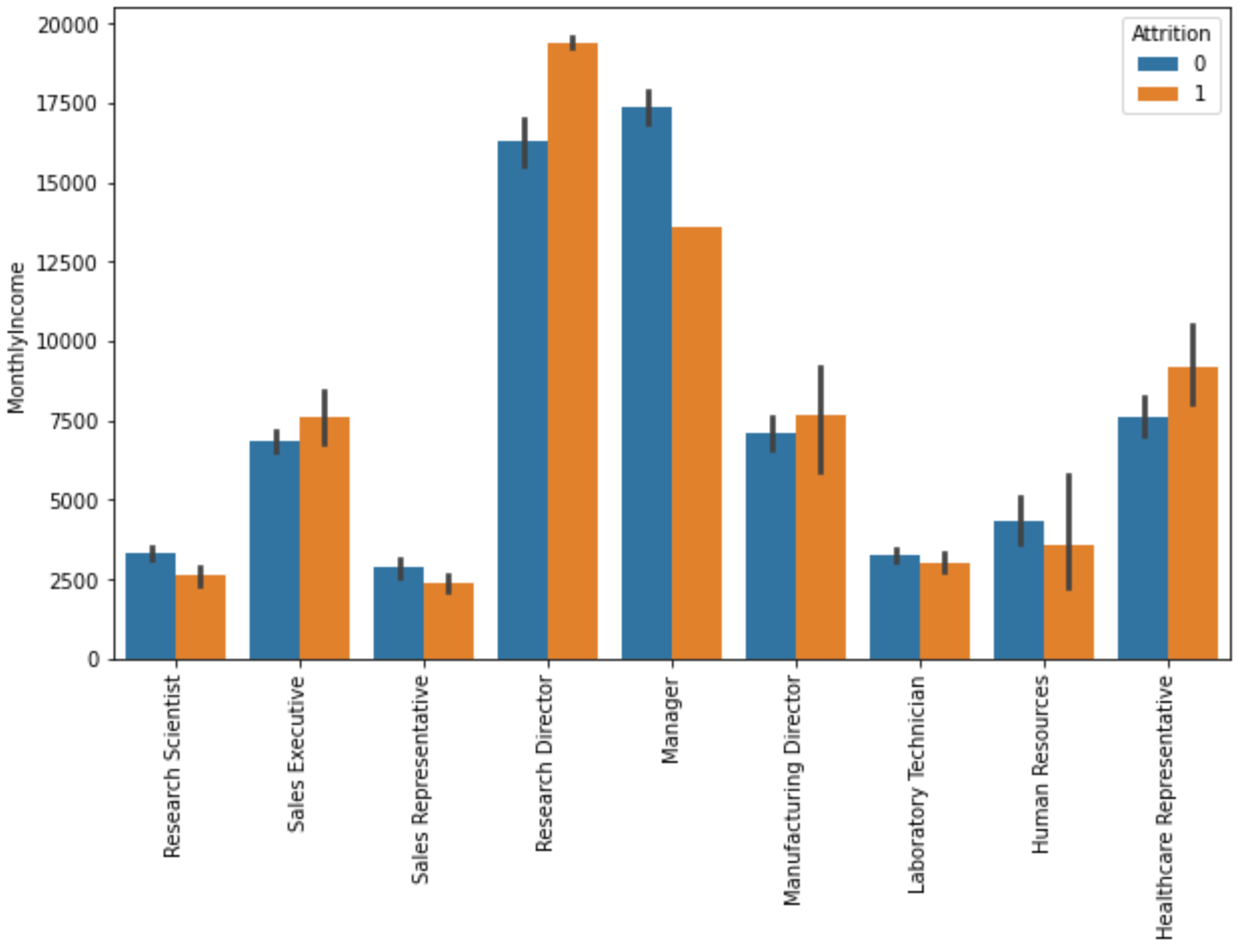


Among job roles, most laboratory technicians have departed from their jobs, only for research scientists, sales executives and sales representatives (% wise) to trail behind. We could look into salaries of each job roles and see if that may be the reason.

To check if attrition in jobrole is affected by monthly income

In [254]:
```
plt.figure(figsize=(10,6))
sns.barplot(x='JobRole', y='MonthlyIncome', hue='Attrition', data=data)
```

Out[254]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8]),
 [Text(0, 0, 'Research Scientist'),
  Text(1, 0, 'Sales Executive'),
  Text(2, 0, 'Sales Representative'),
  Text(3, 0, 'Research Director'),
  Text(4, 0, 'Manager'),
  Text(5, 0, 'Manufacturing Director'),
  Text(6, 0, 'Laboratory Technician'),
  Text(7, 0, 'Human Resources'),
  Text(8, 0, 'Healthcare Representative')])
```



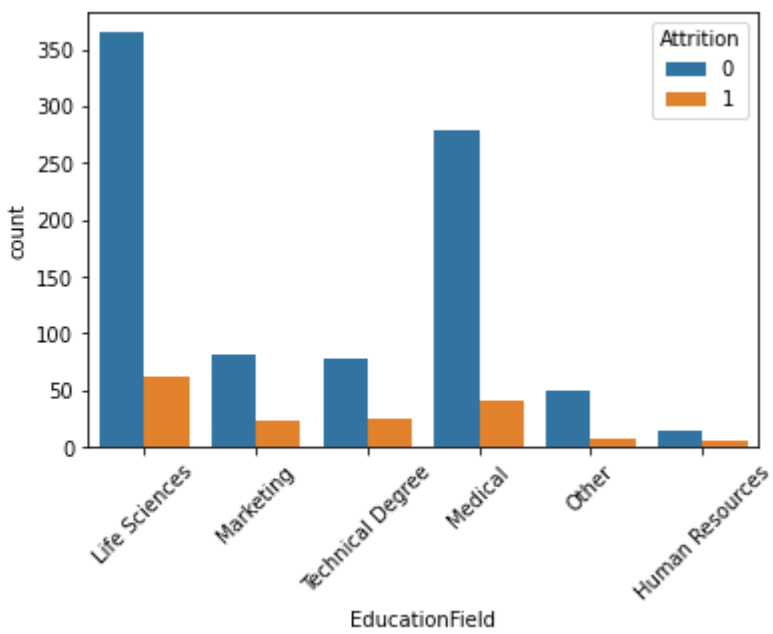
As doubted, laboratory technicians, research scientists and sales representatives and executives have very low salary and this could be a major factor behind attritions.

Also, as we had seen earlier, the HR department had the most attritions and we can see they have very low salaries as well so once again, this is something to think about.

Checking if attrition is affected by EducationField

In [255]:
```
sns.countplot(x='EducationField', hue='Attrition', data=data);
```

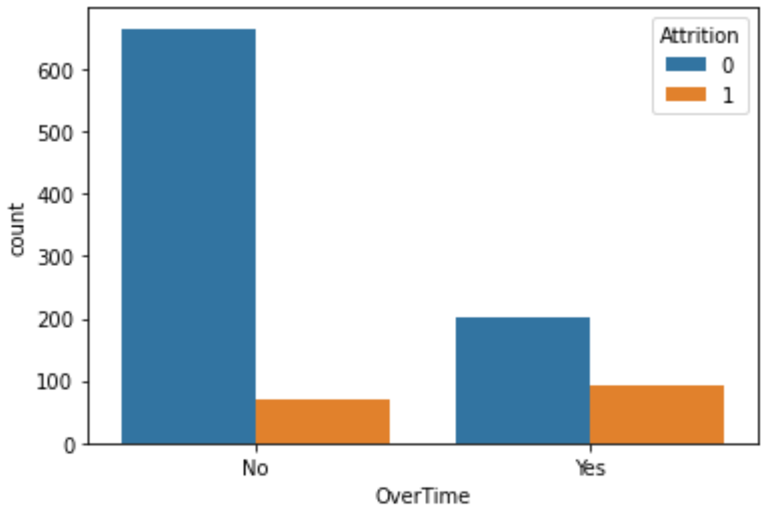
Out[255]:
```
(array([0, 1, 2, 3, 4, 5]),
 [Text(0, 0, 'Life Sciences'),
  Text(1, 0, 'Marketing'),
  Text(2, 0, 'Technical Degree'),
  Text(3, 0, 'Medical'),
  Text(4, 0, 'Other'),
  Text(5, 0, 'Human Resources')])
```



I don't think the degrees of employees really matter here as most of the number of attritions are similar

Checking if attrition is affected by Overtime

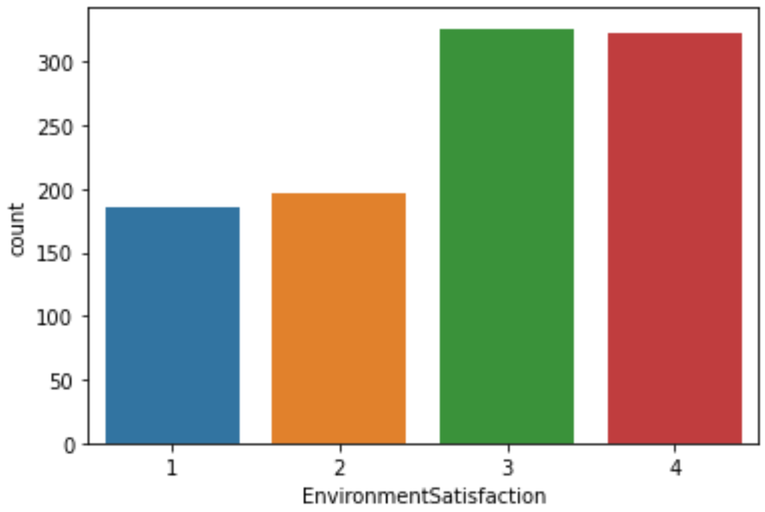
In [256]:



Overtime hours aren't a very crucial factor either.

Checking if attrition is caused by Environment Dissatisfaction

In [257]:



Most employees seem to be satisfied with the working environment

Splitting the Data

In [265]:

```
# Separating the features from the target (In the process, we will drop features that we don't think are key factors.)
X = data.drop(['Attrition'],axis=1) # Features
y = data['Attrition'] # Target
```

In [266]:

```
# Splitting data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.3)
print('X train size: ', len(X_train))
print('X test size: ', len(X_test))
print('y train size: ', len(y_train))
```

```
X train size:  720
X test size:  309
y train size:  720
y test size:  309
```

Encoding all categorical variables

In [267]:

In [268]:
```
# Enoding categorical variables with Ordinal Encoder
OE = OrdinalEncoder()
columns_OE = ['BusinessTravel', 'Education', 'EnvironmentSatisfaction', 'JobInvolvement',
              'JobSatisfaction','WorkLifeBalance','PerformanceRating','RelationshipSatisfaction']
X_train[columns_OE] = OE.fit_transform(X_train[columns_OE])
X_test[columns_OE] = OE.transform(X_test[columns_OE])
```

Out[268]:

	Age	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeNumber	EnvironmentSatisfaction	Gen
994	42	2.0	1396	Research & Development	6	2.0	Medical	1911	2.0	M
410	38	2.0	433	Human Resources	1	2.0	Human Resources	1152	2.0	M
625	35	2.0	1224	Sales	7	3.0	Life Sciences	1962	2.0	Fem
297	25	2.0	685	Research & Development	1	2.0	Life Sciences	350	0.0	Fem
399	53	2.0	1070	Research & Development	3	3.0	Medical	386	2.0	M
...	...	...	...	...	...	...	...	...	...	
481	29	1.0	410	Research & Development	2	0.0	Life Sciences	1513	3.0	Fem
276	42	2.0	1128	Research & Development	13	2.0	Medical	1803	1.0	M
10	36	2.0	363	Research & Development	1	2.0	Technical Degree	1237	2.0	Fem
829	36	2.0	311	Research & Development	7	2.0	Life Sciences	1659	0.0	M
385	27	1.0	591	Research & Development	2	2.0	Medical	1648	3.0	M

720 rows × 32 columns

In [269]:
```
# Transforming bicategoric variables into binary values
X_train['OverTime'].replace({'Yes': 1,
                                       'No':0}, inplace=True)
X_test['OverTime'].replace({'Yes': 1,
                                       'No':0}, inplace=True)
X_train['Gender'].replace({'Male': 1,
                                       'Female':0}, inplace=True)
X_test['Gender'].replace({'Male': 1,
```

In [270]:

Out[270]:
```
0    506
1    214
Name: OverTime, dtype: int64
```

In [271]:

Out[271]:
```
1    181
0    128
Name: Gender, dtype: int64
```

In [272]:

Out[272]:

	Age	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeNumber	EnvironmentSatisfaction	Gen
994	42	2.0	1396	Research & Development	6	2.0	Medical	1911	2.0	
410	38	2.0	433	Human Resources	1	2.0	Human Resources	1152	2.0	
625	35	2.0	1224	Sales	7	3.0	Life Sciences	1962	2.0	
297	25	2.0	685	Research & Development	1	2.0	Life Sciences	350	0.0	
399	53	2.0	1070	Research & Development	3	3.0	Medical	386	2.0	
...	...	...	...	...	...	...	...	...	...	
481	29	1.0	410	Research & Development	2	0.0	Life Sciences	1513	3.0	
276	42	2.0	1128	Research & Development	13	2.0	Medical	1803	1.0	
10	36	2.0	363	Research & Development	1	2.0	Technical Degree	1237	2.0	
829	36	2.0	311	Research & Development	7	2.0	Life Sciences	1659	0.0	
385	27	1.0	591	Research & Development	2	2.0	Medical	1648	3.0	

720 rows × 32 columns

In [273]:

Out[273]:
```
Index(['Age', 'BusinessTravel', 'DailyRate', 'Department', 'DistanceFromHome',
       'Education', 'EducationField', 'EmployeeNumber',
       'EnvironmentSatisfaction', 'Gender', 'HourlyRate', 'JobInvolvement',
       'JobLevel', 'JobRole', 'JobSatisfaction', 'MaritalStatus',
       'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'Over18',
       'OverTime', 'PercentSalaryHike', 'PerformanceRating',
       'RelationshipSatisfaction', 'StockOptionLevel', 'TotalWorkingYears',
       'TrainingTimesLastYear', 'WorkLifeBalance', 'YearsAtCompany',
       'YearsInCurrentRole', 'YearsSinceLastPromotion',
       'YearsWithCurrManager'],
      dtype='object')
```

In [274]:
```
# Enoding categorical variables with One Hot Encoder
OHE = OneHotEncoder(handle_unknown = 'ignore', sparse=False)
columns_OHE = ['Department', 'EducationField', 'JobRole', 'MaritalStatus']
X_train_cols = pd.DataFrame(OHE.fit_transform(X_train[columns_OHE]))
X_test_cols = pd.DataFrame(OHE.transform(X_test[columns_OHE]))
# Putting index back
X_train_cols.index = X_train.index
X_test_cols.index = X_test.index
# Removing categorical columns
num_X_train = X_train.drop([col for col in X_train.columns if X_train[col].dtype == "object"], axis = 1)
num_X_test = X_test.drop([col for col in X_test.columns if X_test[col].dtype == "object"], axis = 1)
# Adding one-hot encoded columns to numerical features
X_train = pd.concat([num_X_train,X_train_cols ],axis = 1)
X_test = pd.concat([num_X_test, X_test_cols], axis = 1)
```

In [275]:

Out[275]:

	Age	BusinessTravel	DailyRate	DistanceFromHome	Education	EmployeeNumber	EnvironmentSatisfaction	Gender	HourlyRate	JobInvolvem
994	42	2.0	1396	6	2.0	1911	2.0	1	83	
410	38	2.0	433	1	2.0	1152	2.0	1	37	
625	35	2.0	1224	7	3.0	1962	2.0	0	55	
297	25	2.0	685	1	2.0	350	0.0	0	62	
399	53	2.0	1070	3	3.0	386	2.0	1	45	
...	...	...	...	...	...	...	...	...	...	
481	29	1.0	410	2	0.0	1513	3.0	0	97	
276	42	2.0	1128	13	2.0	1803	1.0	1	95	
10	36	2.0	363	1	2.0	1237	2.0	0	77	
829	36	2.0	311	7	2.0	1659	0.0	1	77	
385	27	1.0	591	2	2.0	1648	3.0	1	87	

720 rows × 48 columns

In [276]:
```
# Rescaling Data
Scaler = MinMaxScaler()
Scaling_Cols = ['TrainingTimesLastYear','YearsAtCompany','TotalWorkingYears',
               'YearsInCurrentRole','YearsSinceLastPromotion','YearsWithCurrManager',
               'PercentSalaryHike','Age','DailyRate','DistanceFromHome','HourlyRate',
               'MonthlyIncome','MonthlyRate','NumCompaniesWorked']
X_train[Scaling_Cols] = Scaler.fit_transform(X_train[Scaling_Cols])
X_test[Scaling_Cols] = Scaler.transform(X_test[Scaling_Cols])
```

Out[276]:

	Age	BusinessTravel	DailyRate	DistanceFromHome	Education	EmployeeNumber	EnvironmentSatisfaction	Gender	HourlyRate	JobInvol
994	0.571429	2.0	0.926218	0.178571	2.0	1911	2.0	1	0.757143	
410	0.476190	2.0	0.236390	0.000000	2.0	1152	2.0	1	0.100000	
625	0.404762	2.0	0.803009	0.214286	3.0	1962	2.0	0	0.357143	
297	0.166667	2.0	0.416905	0.000000	2.0	350	0.0	0	0.457143	
399	0.833333	2.0	0.692693	0.071429	3.0	386	2.0	1	0.214286	
...	...	...	...	...	...	...	...	...	...	
481	0.261905	1.0	0.219914	0.035714	0.0	1513	3.0	0	0.957143	
276	0.571429	2.0	0.734241	0.428571	2.0	1803	1.0	1	0.928571	
10	0.428571	2.0	0.186246	0.000000	2.0	1237	2.0	0	0.671429	
829	0.428571	2.0	0.148997	0.214286	2.0	1659	0.0	1	0.671429	
385	0.214286	1.0	0.349570	0.035714	2.0	1648	3.0	1	0.814286	

720 rows × 48 columns

In [277]:
```
print(y_train.value_counts())
```

```
0    617
1    103
Name: Attrition, dtype: int64
0    251
1     58
Name: Attrition, dtype: int64
```

Our target variable is unbalanced, with much more 'No' values than 'Yes'.I will SMOTE to synthetically create more 'Yes' values and have a 50/50 distribution for both classes during training. I prefer to oversample our minor class than undersampling the major class because undersampling may cause a loss of relevant data.

In [278]:
```
# Dealing with Class Imbalance using SMOTE
from imblearn.over_sampling import SMOTE
```

In [279]:

Out[279]:
```
1    617
0    617
Name: Attrition, dtype: int64
```

In [280]:
```
# from sklearn.linear_model import LogisticRegression
# from sklearn.metrics import accuracy_score, classification_report
# # Fit a Logistic Regression model
# model = LogisticRegression(random_state=42)

# # Fit the model on the scaled training data
# model.fit(X_train, y_train)

# # Evaluate the model
# # Predictions on the testing set
# y_pred = model.predict(X_test)

# # Model accuracy
# accuracy = accuracy_score(y_test, y_predict)
# print(f"Model Accuracy: {accuracy}")

# # Classification report
# report = classification_report(y_test, y_predict)
```

In [281]:

In [282]:
```python
# Fit the model on the scaled training data
model = LogisticRegression(max_iter=1000, random_state=42)
model.fit(X_train, y_train)

# Evaluate the model
# Predictions on the testing set
y_pred = model.predict(X_test)

# Model accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy}")

# Classification report
report = classification_report(y_test, y_pred)
```

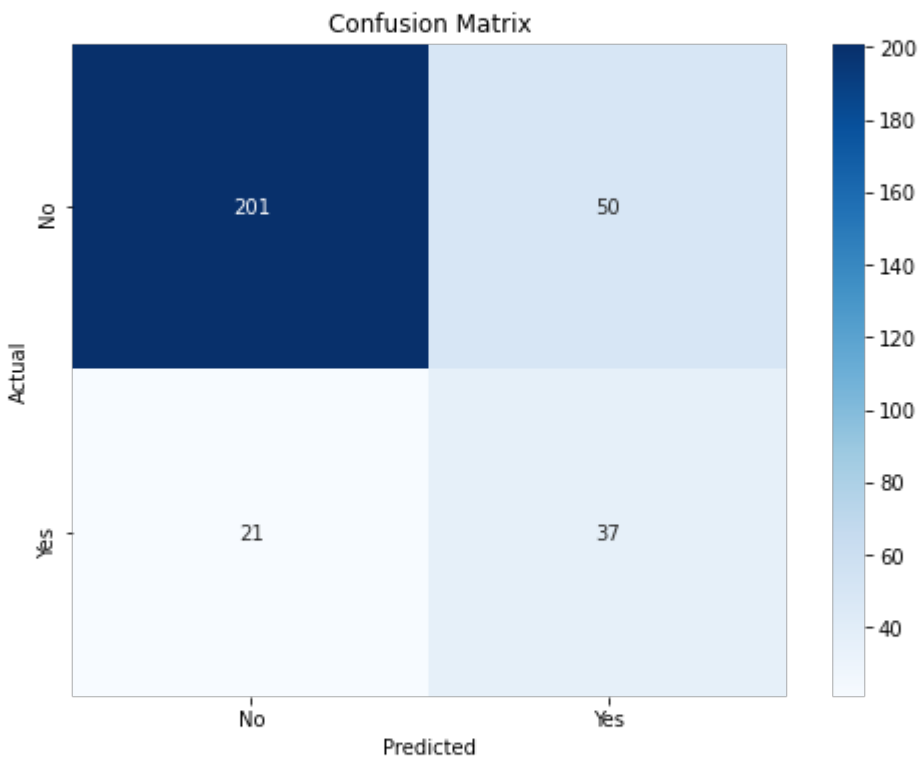
```
Model Accuracy: 0.7702265372168284
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.80      0.85       251
           1       0.43      0.64      0.51        58

    accuracy                           0.77       309
   macro avg       0.67      0.72      0.68       309
weighted avg       0.82      0.77      0.79       309
```

In [283]:
```python
# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)
# Visualize confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Confusion Matrix:
 [[201  50]
 [ 21  37]]
```



Interpretation of the confusion matrix

True Positives (TP): 30 These are the cases where the model correctly predicted "Yes" for attrition.

True Negatives (TN): 202 These are the cases where the model correctly predicted "No" for attrition.

False Positives (FP): 55 These are the cases where the model incorrectly predicted "Yes" for attrition when it was actually "No". This is also known as a Type I error.

False Negatives (FN): 22 These are the cases where the model incorrectly predicted "No" for attrition when it was actually "Yes". This is also known as a Type II error.

The model has a high number of True Negatives (202) and a relatively lower number of True Positives (30), indicating that it performs better at identifying "No" for attrition than "Yes". The precision for predicting "Yes" is quite low (0.35), meaning that when the model predicts attrition, it is only correct about 35% of the time. The recall for predicting "Yes" is moderate (0.58), meaning the model correctly identifies 58% of the actual "Yes" cases. The model's overall accuracy is 0.75, indicating that 75% of the predictions are correct. However, this is influenced by the class imbalance, as there are more "No" cases than "Yes".