Exploratory Data Analysis and Prediction of IBM Employees.

DataSet Source : https://www.kaggle.com/datasets/rohitsahoo/employee

This dataset have 1058 Rows and 36 Columns.

About Dataset

Education

- 'Below College'
- 'College'
- 'Bachelor'
- 'Master'
- 'Doctor'

EnvironmentSatisfaction

- 'Low'
- 'Medium'
- 'High'
- 'Very High'

JobInvolvement

- 'Low'
- 'Medium'
- 'High'
- 'Very High'

JobSatisfaction

- 'Low'
- 'Medium'
- 'High'
- 'Very High'

PerformanceRating

- 'Low'
- 'Good'
- 'Excellent'

- 'Outstanding'

RelationshipSatisfaction

- 'Low'
- 'Medium'
- 'High'
- 'Very High'

WorkLifeBalance

- 'Bad'
- 'Good'
- 'Better'
- 'Best'

This notebook is structured as follows:

- Exploratory Data Analysis: In this section, we explore the dataset by taking a look at the feature distributions, how correlated one feature is to the other and create some Seaborn and Plotly visualisations
- Feature Engineering and Categorical Encoding: Conduct some feature engineering as well as encode all our categorical features into dummy variables
- Implementing Machine Learning models: We implement a Random Forest and a Gradient Boosted Model after which we look at feature importances from these respective models

Let's Go.

In [2]:
```
import numpy as np
import pandas as pd
import seaborn as sns
sns.set()
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

EDA and Prediction of train.csv Dataset.

1. Exploratory Data Analysis.

In [3]:
```
data=pd.read_csv('train.csv')
```

In [75]:
```
#Top 5 Records.

data.head()
```

Out[75]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	Education
0	41	1	Travel_Rarely	1102	Sales	1	2	Life Sci
1	49	0	Travel_Frequently	279	Research & Development	8	1	Life Sci
2	37	1	Travel_Rarely	1373	Research & Development	2	2	(
3	33	0	Travel_Frequently	1392	Research & Development	3	4	Life Sci
4	27	0	Travel_Rarely	591	Research & Development	2	1	Me

5 rows × 36 columns

Data quality checks

To look for any null values, we can just invoke the isnull call as follows

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

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

In [6]:
```
data.dtypes
```

Out[6]:
```
Age                          int64
Attrition                    int64
BusinessTravel              object
DailyRate                    int64
Department                  object
DistanceFromHome             int64
Education                    int64
EducationField              object
EmployeeCount                int64
EmployeeNumber               int64
EnvironmentSatisfaction      int64
Gender                      object
HourlyRate                   int64
JobInvolvement               int64
JobLevel                     int64
JobRole                     object
JobSatisfaction              int64
MaritalStatus               object
MonthlyIncome                int64
MonthlyRate                  int64
NumCompaniesWorked           int64
Over18                      object
OverTime                    object
PercentSalaryHike            int64
PerformanceRating            int64
RelationshipSatisfaction     int64
StandardHours                int64
StockOptionLevel             int64
TotalWorkingYears            int64
TrainingTimesLastYear        int64
WorkLifeBalance              int64
YearsAtCompany               int64
YearsInCurrentRole           int64
YearsSinceLastPromotion      int64
YearsWithCurrManager         int64
dtype: object
```

In [7]:
```
categorical=data.select_dtypes('object')
print(categorical.columns)
```

```
Index(['BusinessTravel', 'Department', 'EducationField', 'Gender', 'JobRole',
       'MaritalStatus', 'Over18', 'OverTime'],
      dtype='object')
```

In [9]:
```
numerical=data.select_dtypes('int64','float64')
print(numerical.columns)
```

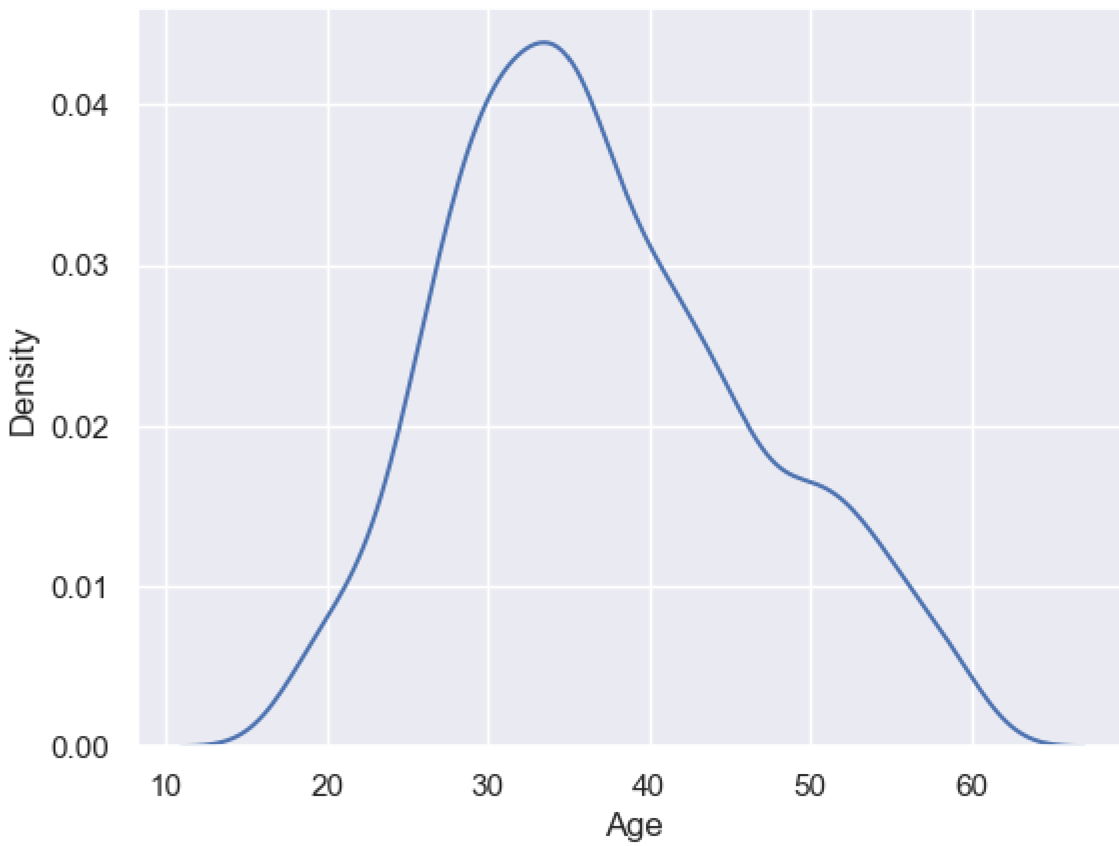
```
Index(['Age', 'Attrition', 'DailyRate', 'DistanceFromHome', 'Education',
       'EmployeeCount', 'EmployeeNumber', 'EnvironmentSatisfaction',
       'HourlyRate', 'JobInvolvement', 'JobLevel', 'JobSatisfaction',
       'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
       'PercentSalaryHike', 'PerformanceRating', 'RelationshipSatisfaction',
       'StandardHours', 'StockOptionLevel', 'TotalWorkingYears',
       'TrainingTimesLastYear', 'WorkLifeBalance', 'YearsAtCompany',
       'YearsInCurrentRole', 'YearsSinceLastPromotion',
       'YearsWithCurrManager'],
      dtype='object')
```

Distribution of the dataset :

Generally one of the first few steps in exploring the data would be to have a rough idea of how the features are distributed with one another. To do so, I shall invoke the familiar kdeplot function from the Seaborn plotting library and this generates bivariate plots as follows:

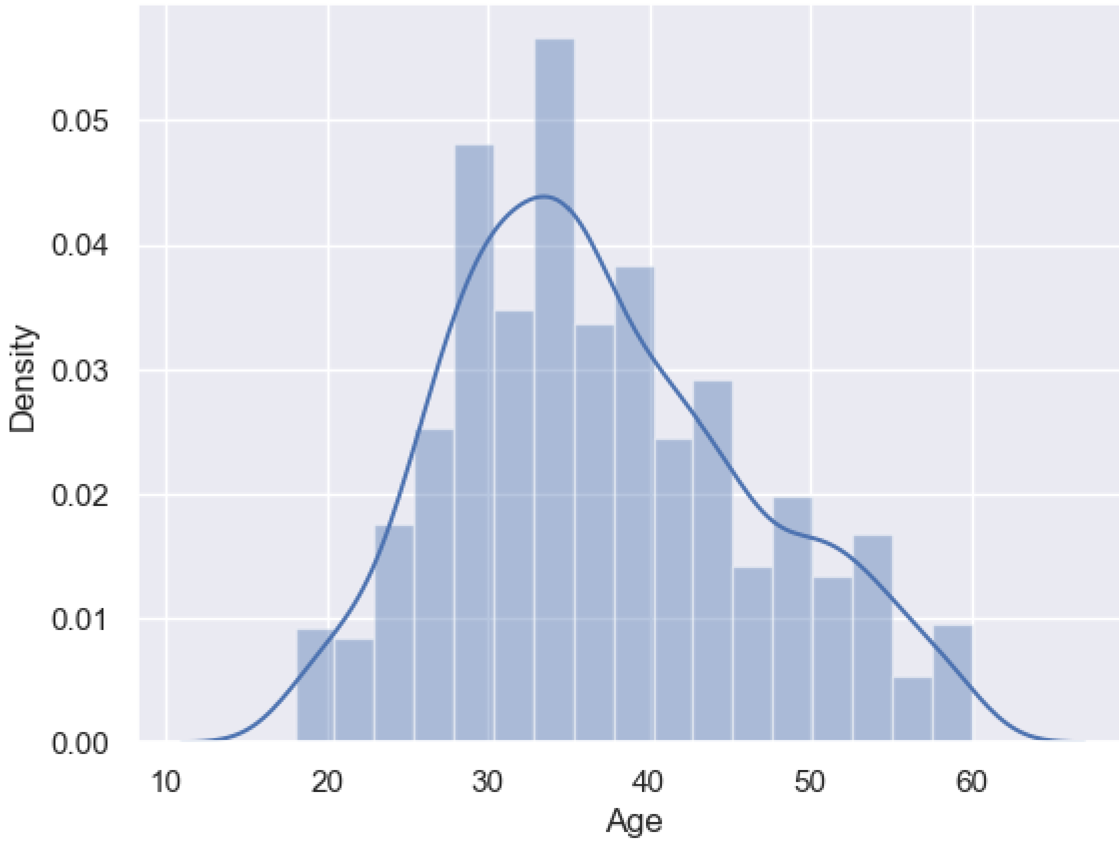
In [10]:
```
sns.kdeplot(data['Age'])
```

Out[10]:
```
<Axes: xlabel='Age', ylabel='Density'>
```

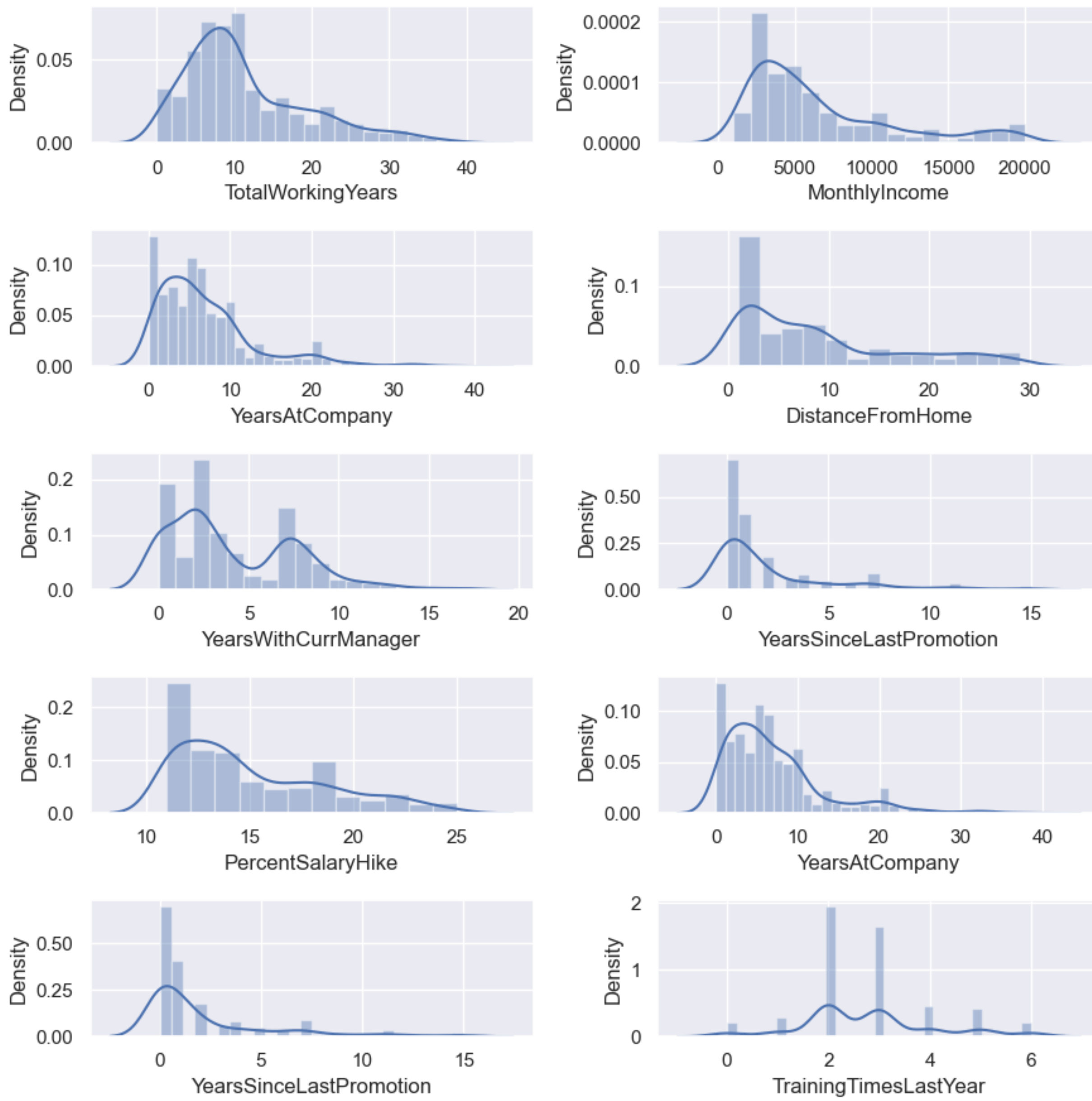


In [11]:
```
sns.distplot(data['Age'])
```

Out[11]:
```
<Axes: xlabel='Age', ylabel='Density'>
```

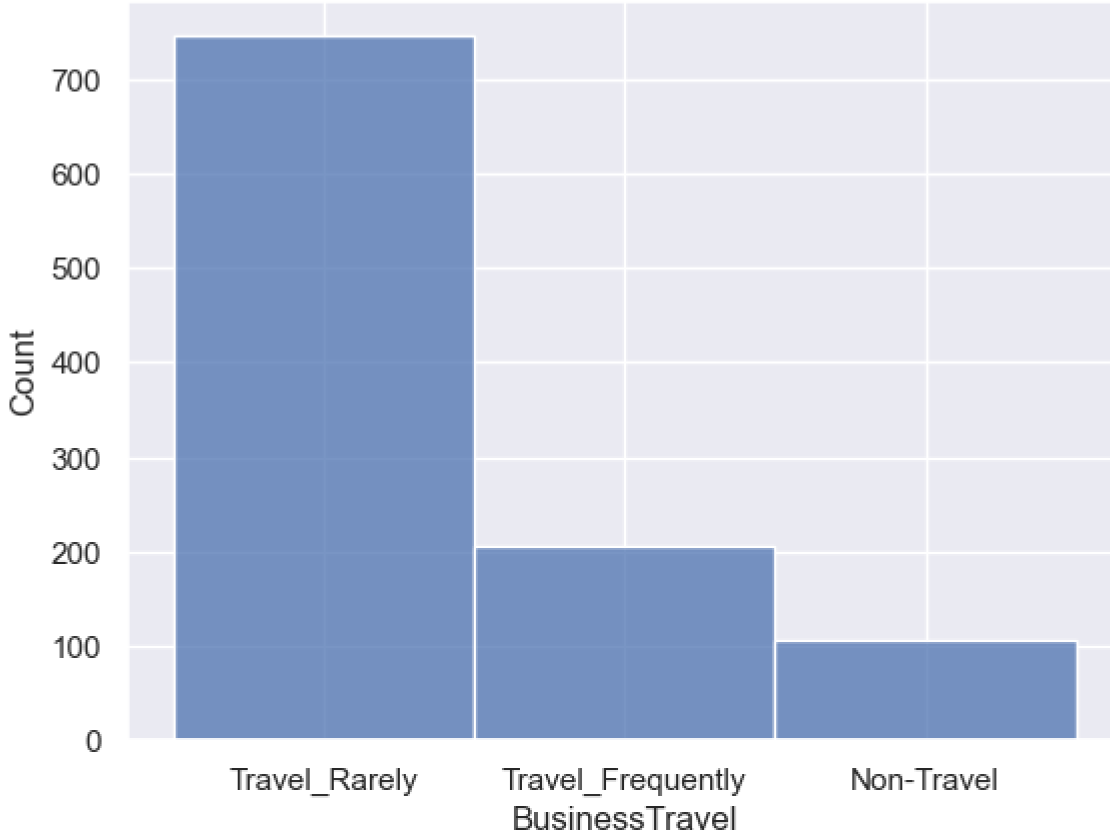


In [16]:
```
fig, ax=plt.subplots(5,2,figsize=(9,9))
sns.distplot(data['TotalWorkingYears'],ax=ax[0,0])
sns.distplot(data['MonthlyIncome'],ax=ax[0,1])
sns.distplot(data['YearsAtCompany'],ax=ax[1,0])
sns.distplot(data['DistanceFromHome'],ax=ax[1,1])
sns.distplot(data['YearsWithCurrManager'],ax=ax[2,0])
sns.distplot(data['YearsSinceLastPromotion'],ax=ax[2,1])
sns.distplot(data['PercentSalaryHike'],ax=ax[3,0])
sns.distplot(data['YearsAtCompany'],ax=ax[3,1])
sns.distplot(data['YearsSinceLastPromotion'],ax=ax[4,0])
sns.distplot(data['TrainingTimesLastYear'],ax=ax[4,1])
plt.tight_layout()
plt.show()
```



In [17]:
```
sns.histplot(data=data,x='BusinessTravel')
```

Out[17]:
```
<Axes: xlabel='BusinessTravel', ylabel='Count'>
```

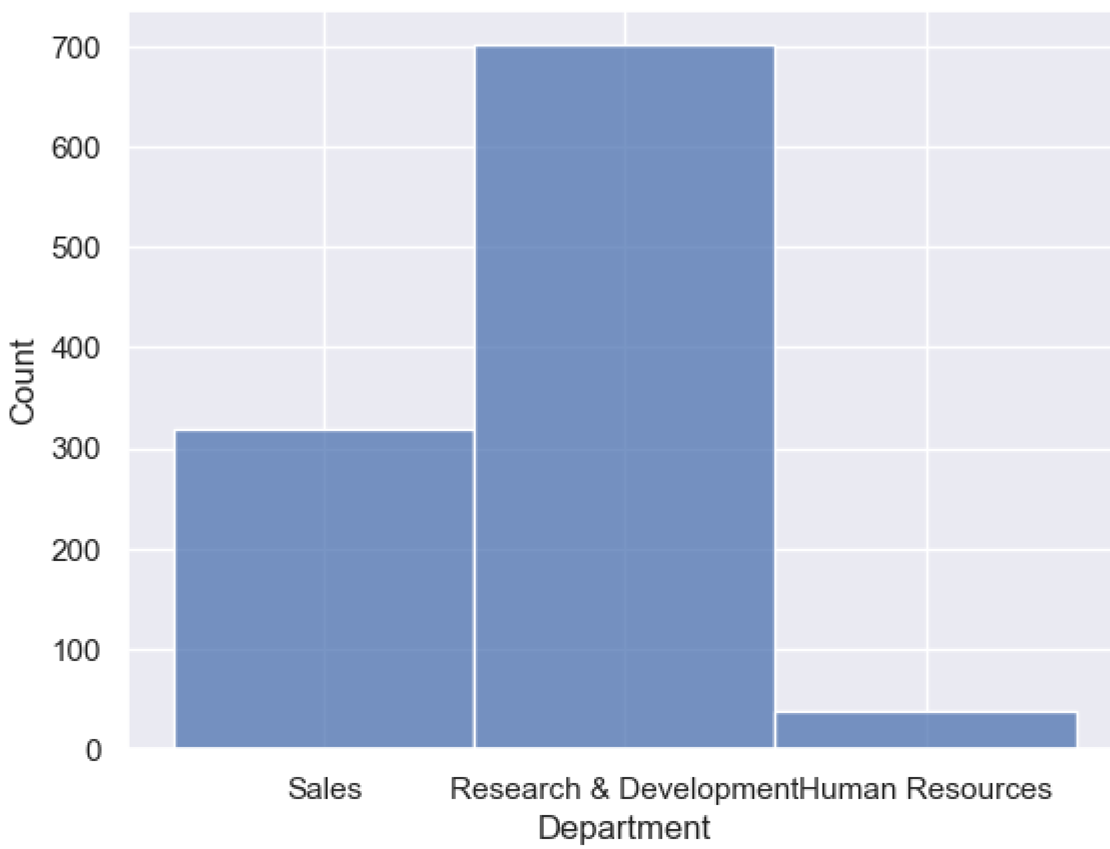


In [18]:

```
sns.histplot(data=data,x='Department')
```

Out[18]:

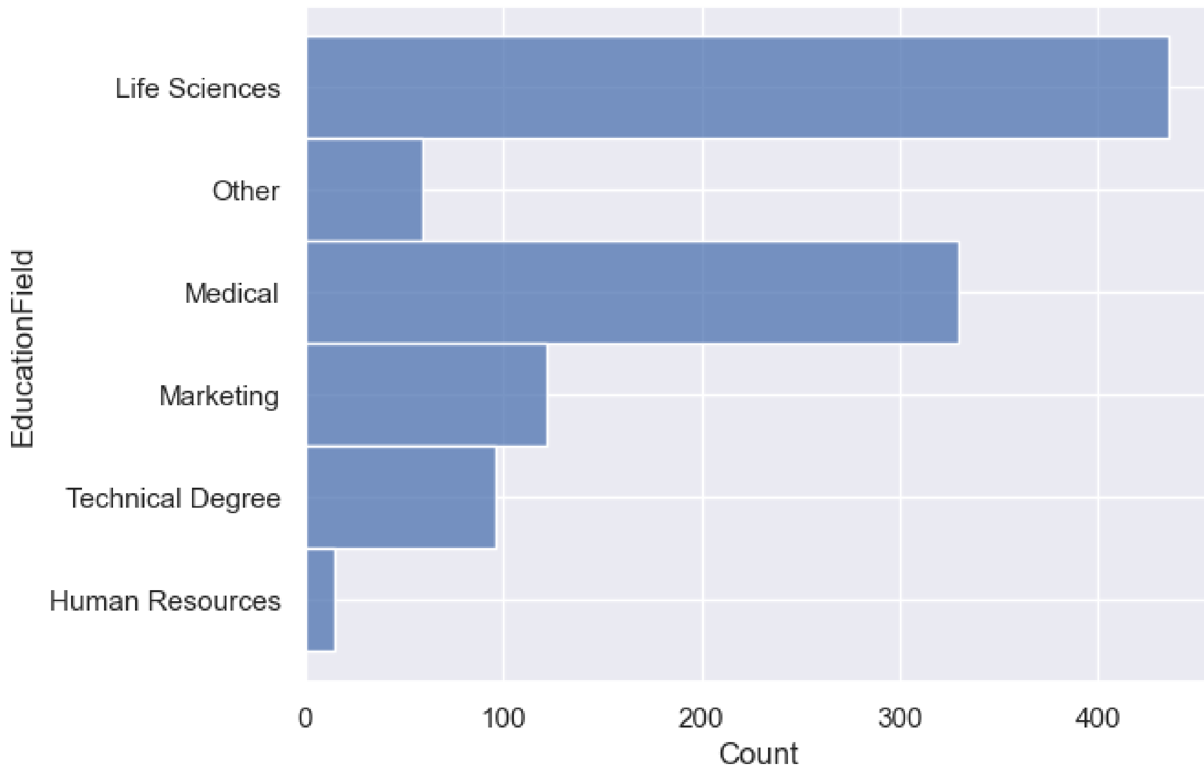
```
<Axes: xlabel='Department', ylabel='Count'>
```



In [19]:

```python
sns.histplot(data=data,y='EducationField')
```

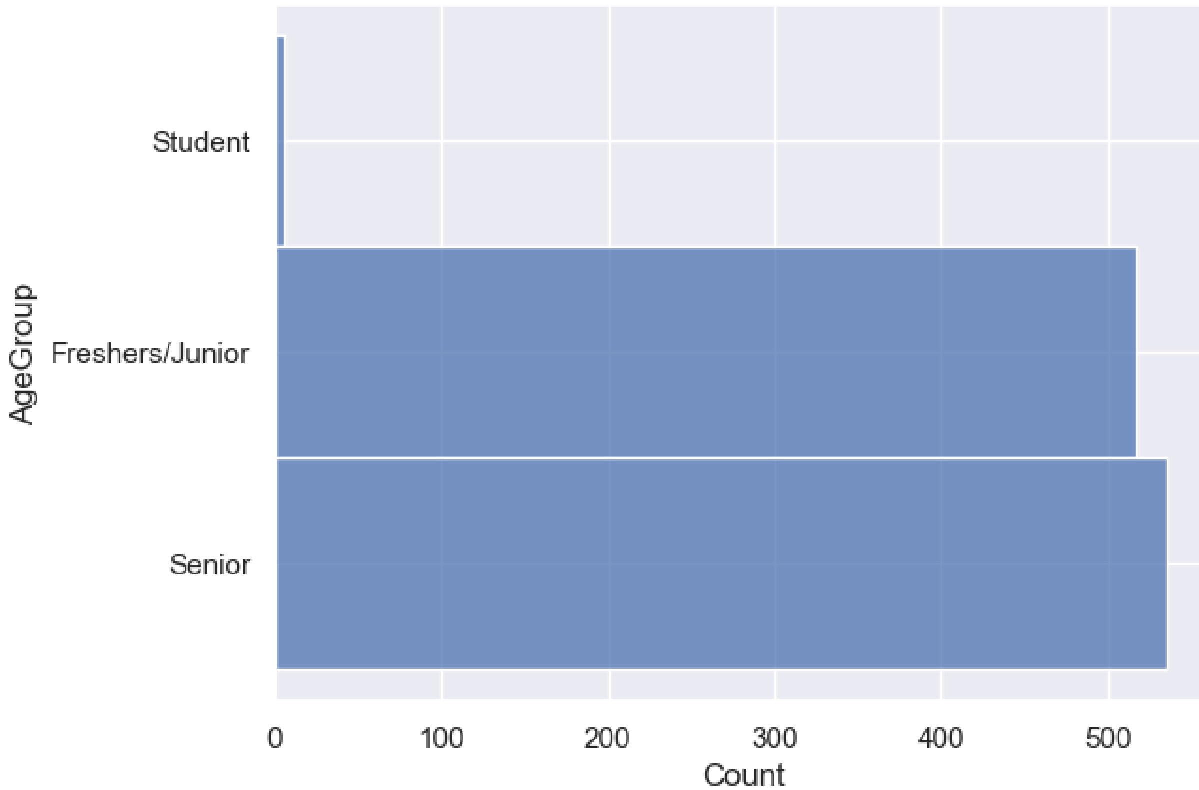
Out[19]: <Axes: xlabel='Count', ylabel='EducationField'>



In [22]:

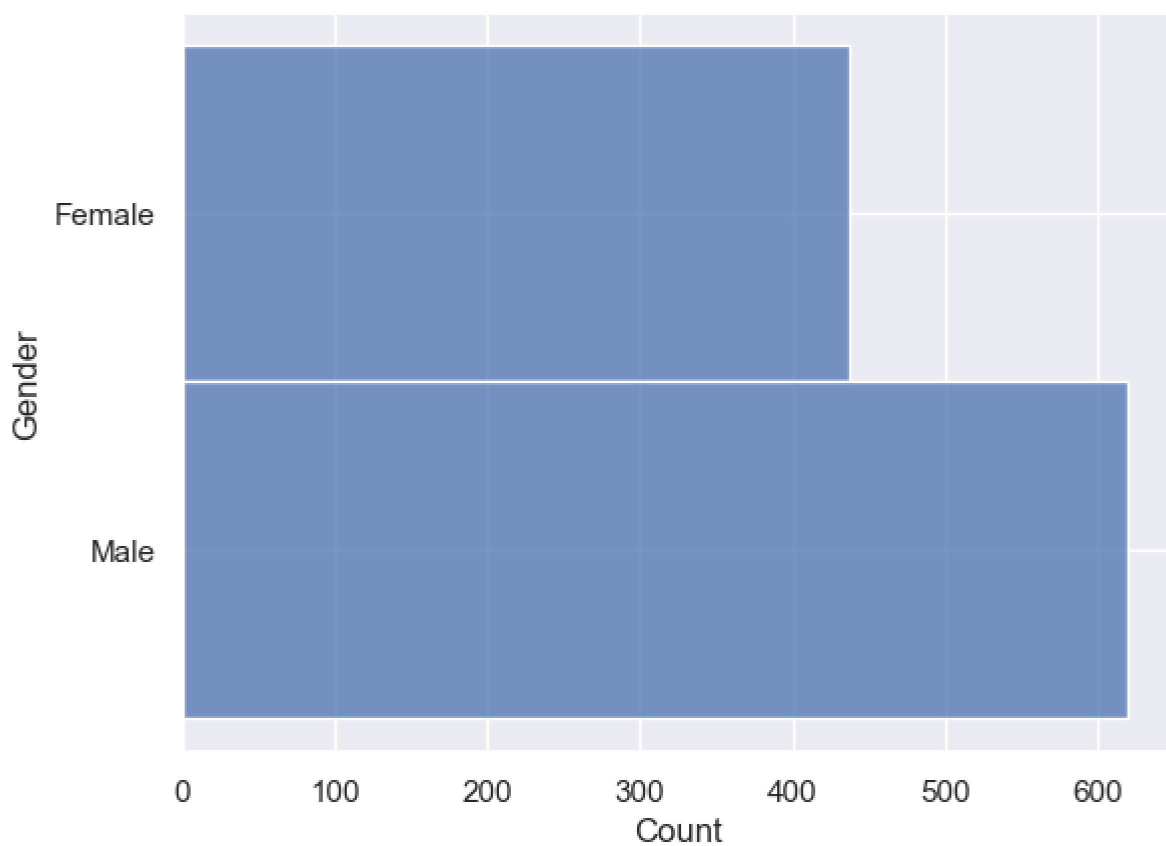
```python
bins=[0,18,35,np.inf]
labels=['Student','Freshers/Junior','Senior']
data['AgeGroup']=pd.cut(data['Age'],bins,labels=labels)
sns.histplot(data=data,y='AgeGroup')
```

Out[22]: <Axes: xlabel='Count', ylabel='AgeGroup'>



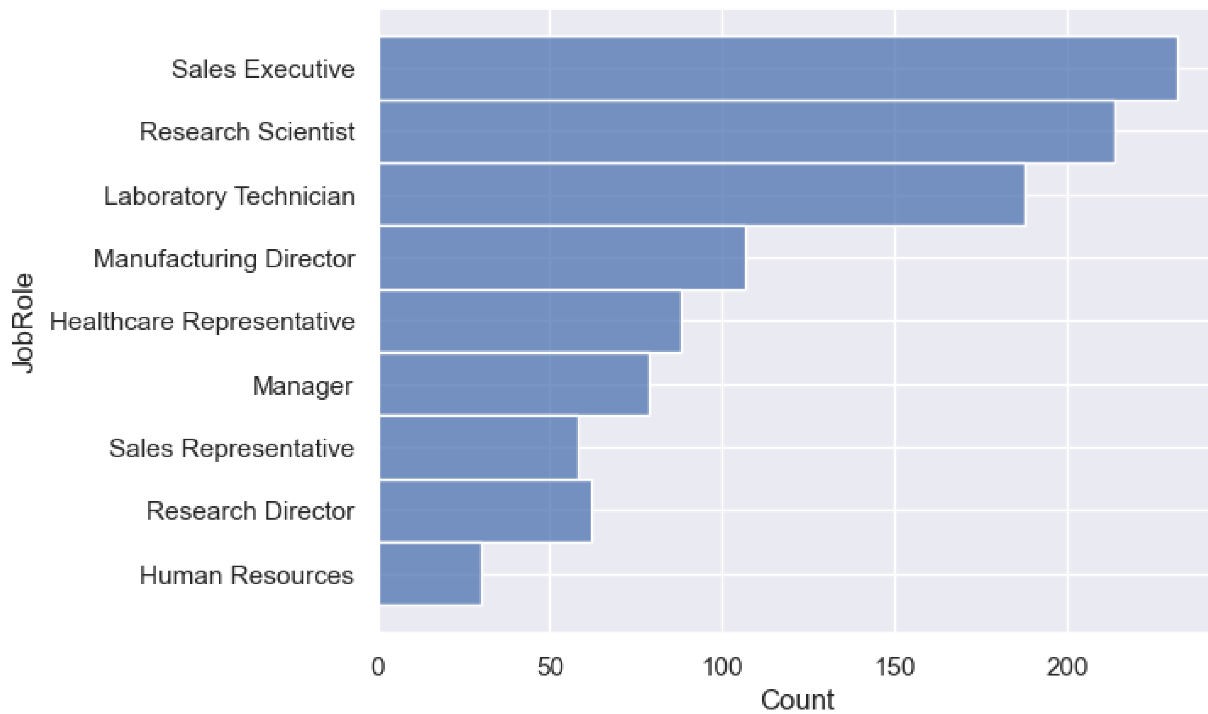
In [23]:
```
sns.histplot(data=data,y='Gender')
```

Out[23]: <Axes: xlabel='Count', ylabel='Gender'>



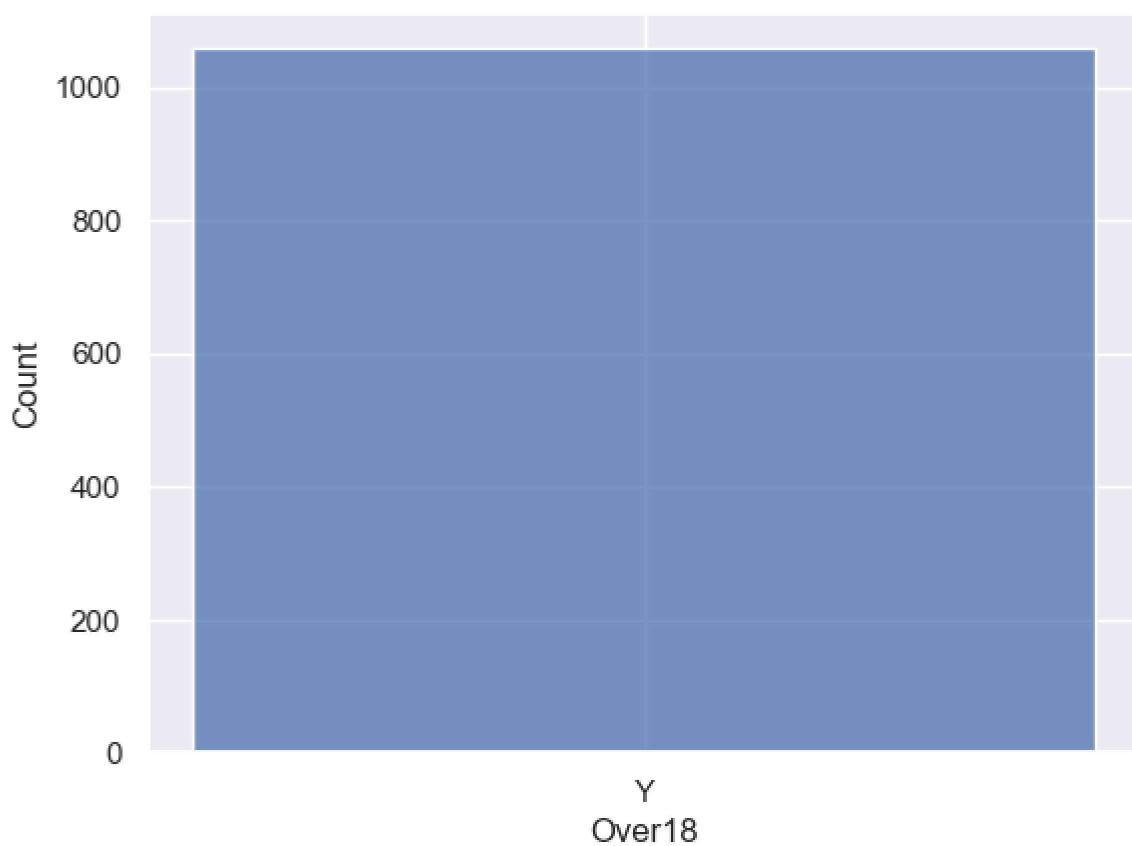
In [24]:
```
sns.histplot(data=data,y='JobRole')
```

Out[24]: <Axes: xlabel='Count', ylabel='JobRole'>



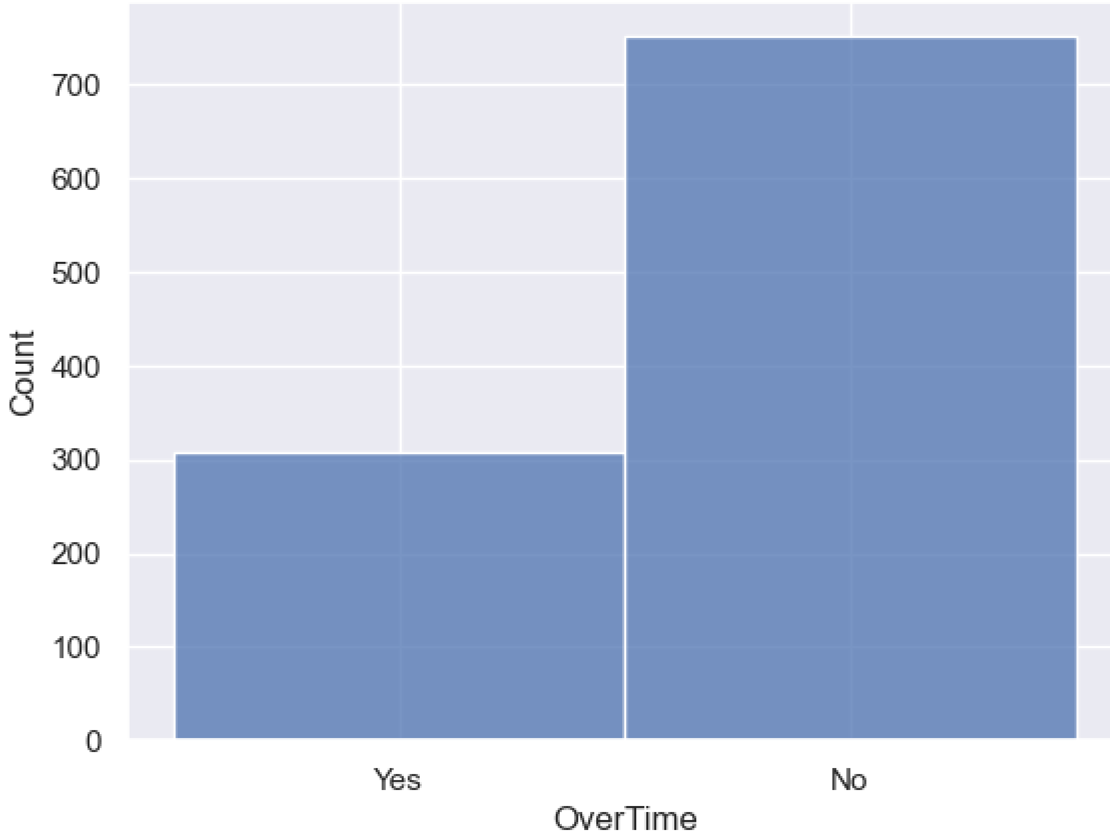
In [26]:
```
sns.histplot(data=data,x='Over18')
```

Out[26]: <Axes: xlabel='Over18', ylabel='Count'>



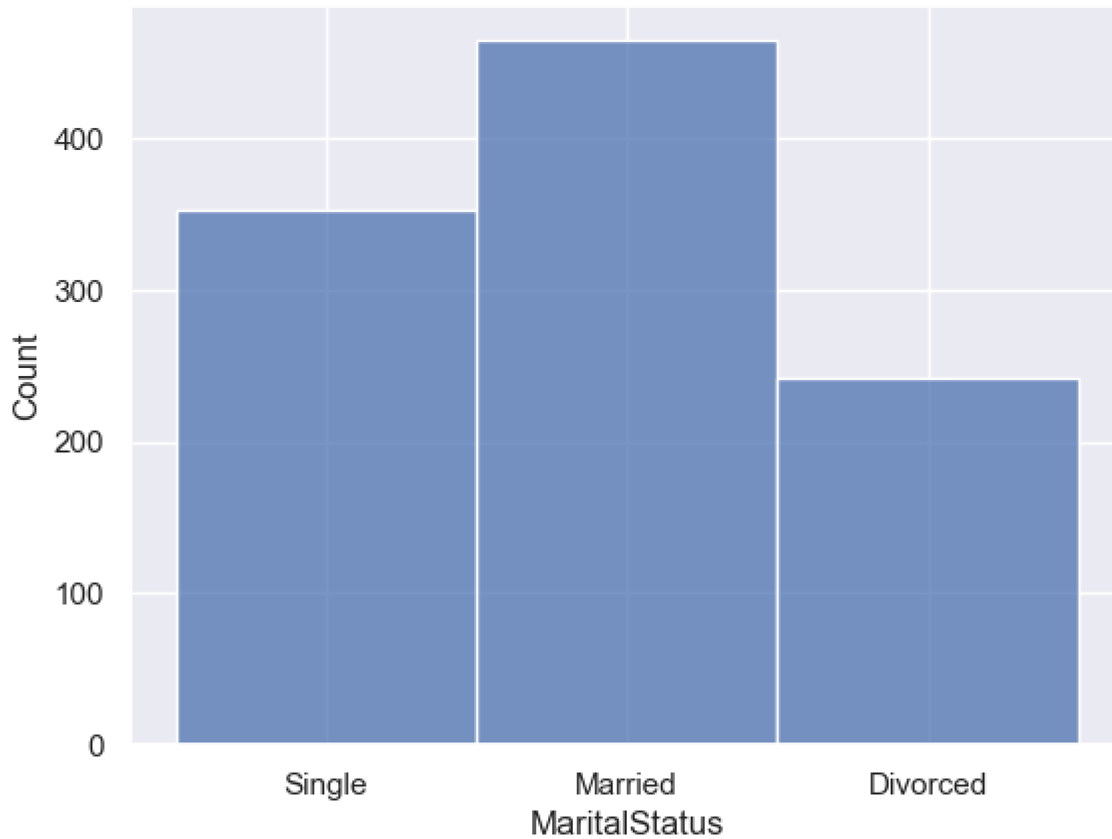
In [27]:
```
sns.histplot(data=data,x='OverTime')
```

Out[27]: <Axes: xlabel='OverTime', ylabel='Count'>



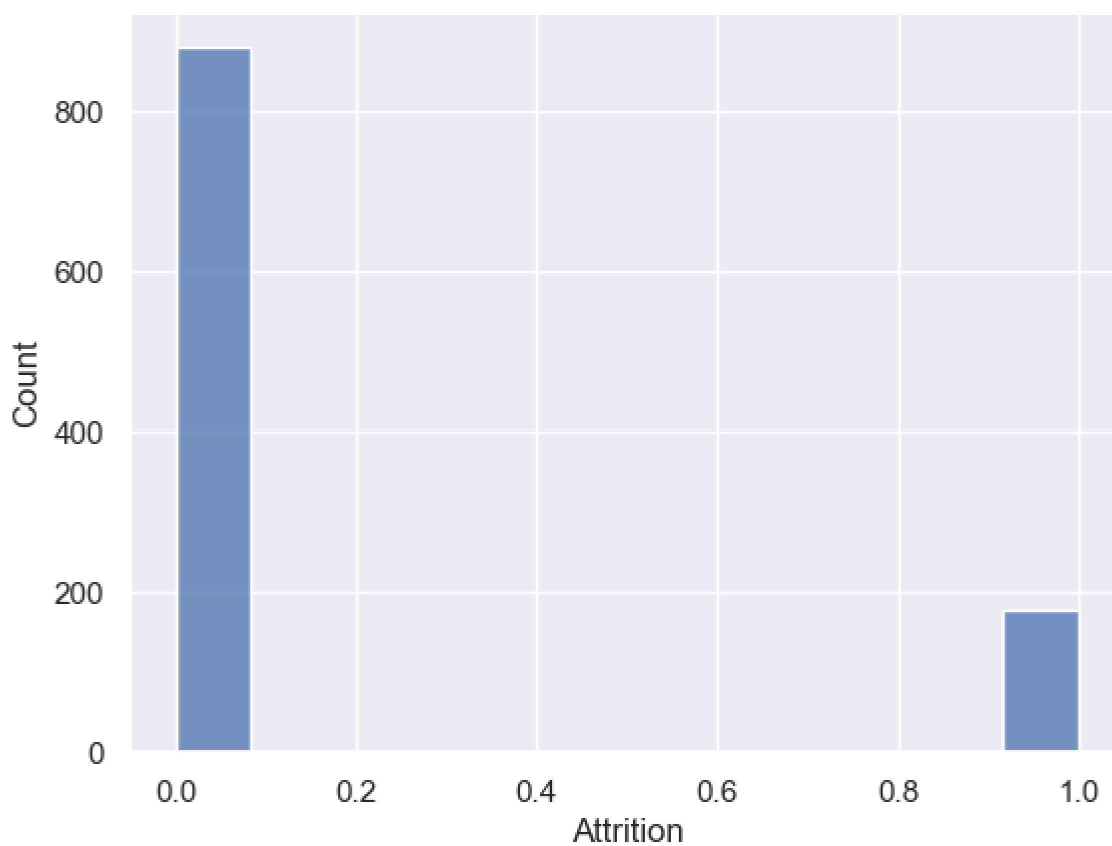
In [29]:
```
sns.histplot(data=data,x='MaritalStatus')
```

Out[29]: <Axes: xlabel='MaritalStatus', ylabel='Count'>



In [31]:
```
sns.histplot(data=data,x='Attrition')
```

Out[31]: <Axes: xlabel='Attrition', ylabel='Count'>



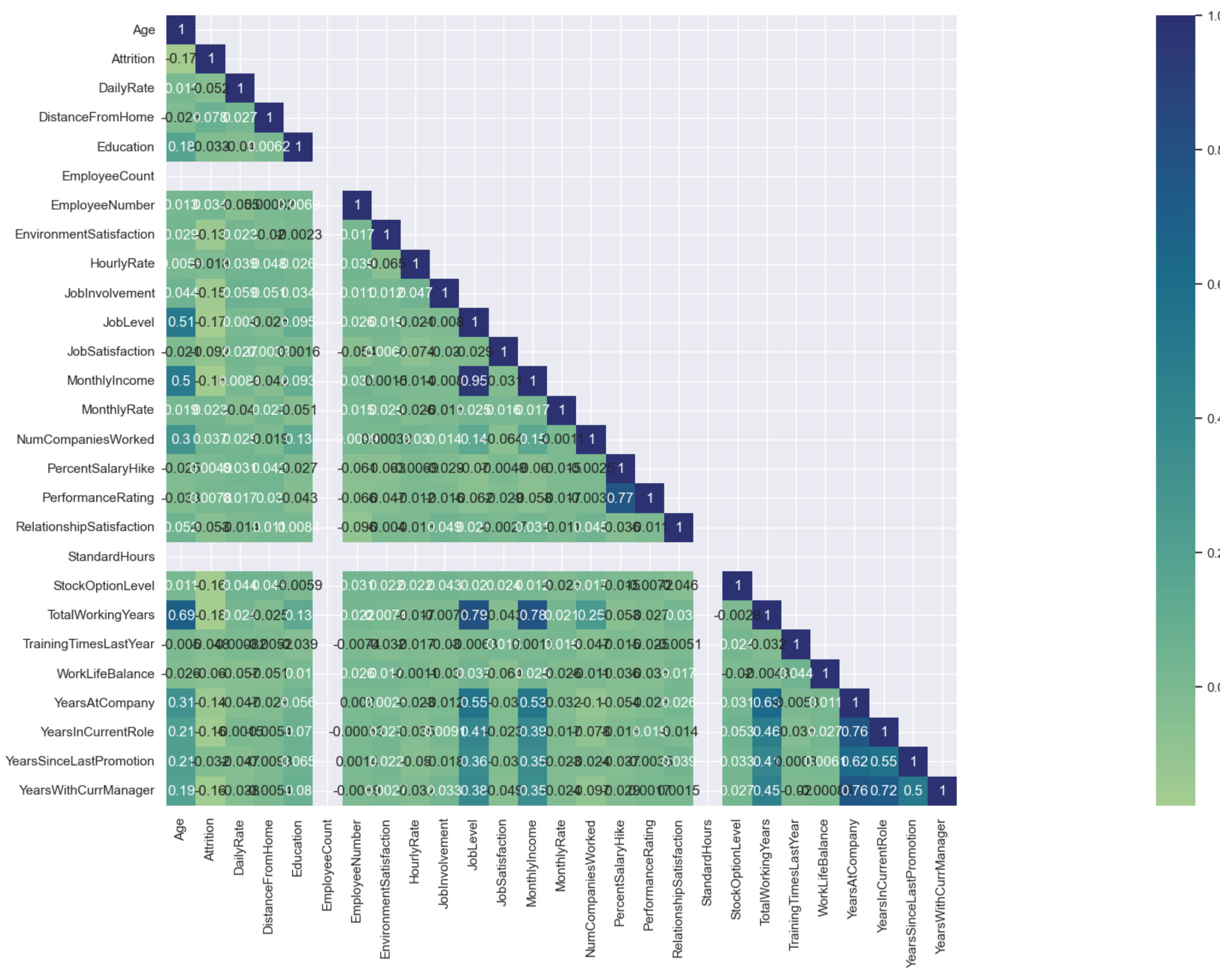
Correlation of Features

The next tool in a data explorer's arsenal is that of a correlation matrix. By plotting a correlation matrix, we have a very nice overview of how the features are related to one another. For a Pandas dataframe, we can conveniently use the call .corr which by default provides the Pearson Correlation values of the columns pairwise in that dataframe.

In this correlation plot, I will use the the Plotly library to produce a interactive Pearson correlation matrix via the Heatmap function as follows:

In [33]:
```
cor_mat=data.corr()
mask=np.array(cor_mat)
mask[np.tril_indices_from(mask)]=False
fig=plt.gcf()
fig.set_size_inches(60,12)
sns.heatmap(data=cor_mat,mask=mask,square=True,annot=True,cbar=True,cmap="crest")
```

Out[33]:
```
<Axes: >
```

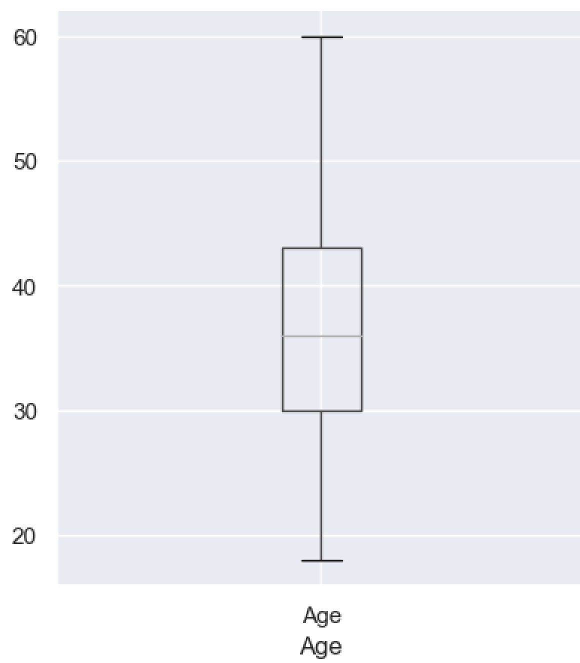


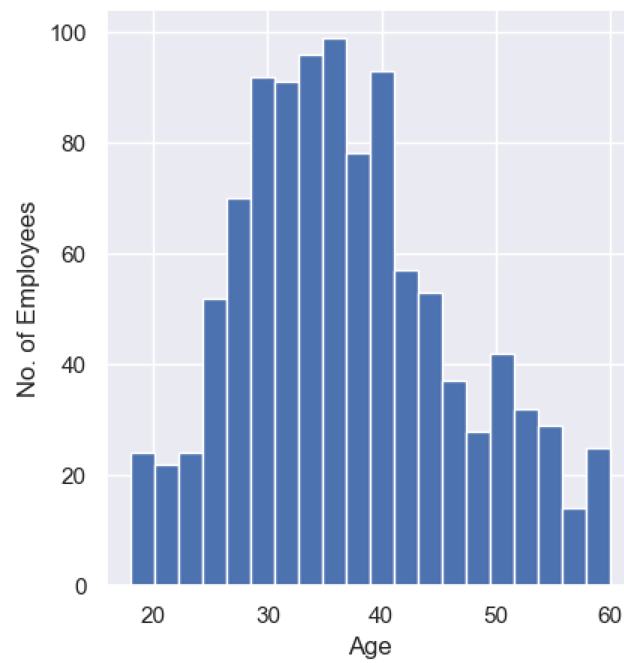
In [34]:
```
data.columns
```

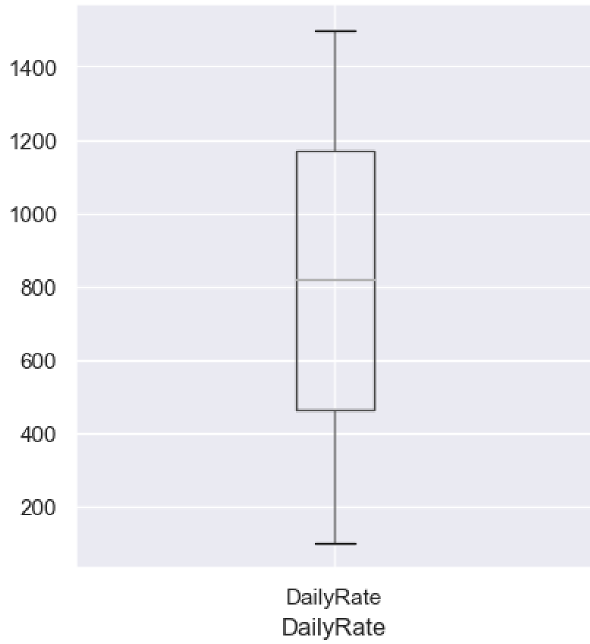
Out[34]:
```
Index(['Age', 'Attrition', 'BusinessTravel', 'DailyRate', 'Department',
       'DistanceFromHome', 'Education', 'EducationField', 'EmployeeCount',
       'EmployeeNumber', 'EnvironmentSatisfaction', 'Gender', 'HourlyRate',
       'JobInvolvement', 'JobLevel', 'JobRole', 'JobSatisfaction',
       'MaritalStatus', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
       'Over18', 'OverTime', 'PercentSalaryHike', 'PerformanceRating',
       'RelationshipSatisfaction', 'StandardHours', 'StockOptionLevel',
       'TotalWorkingYears', 'TrainingTimesLastYear', 'WorkLifeBalance',
       'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion',
       'YearsWithCurrManager', 'AgeGroup'],
      dtype='object')
```

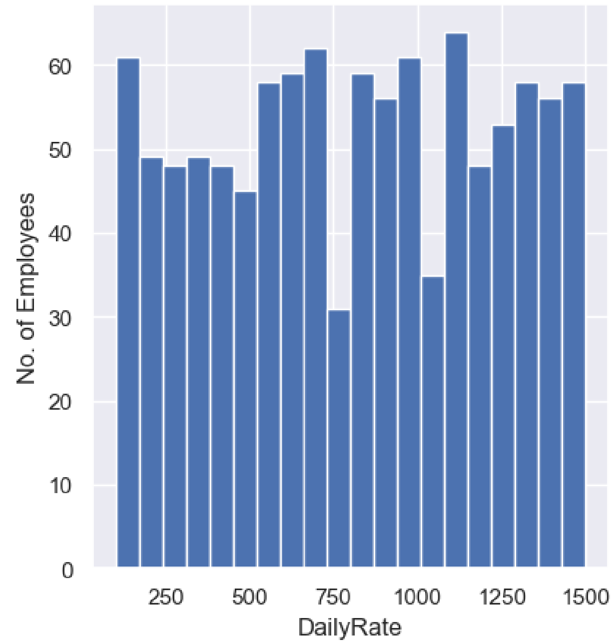
In [35]:
```
continious=['Age','DailyRate','HourlyRate','MonthlyIncome','MonthlyRate','TotalWorking
```

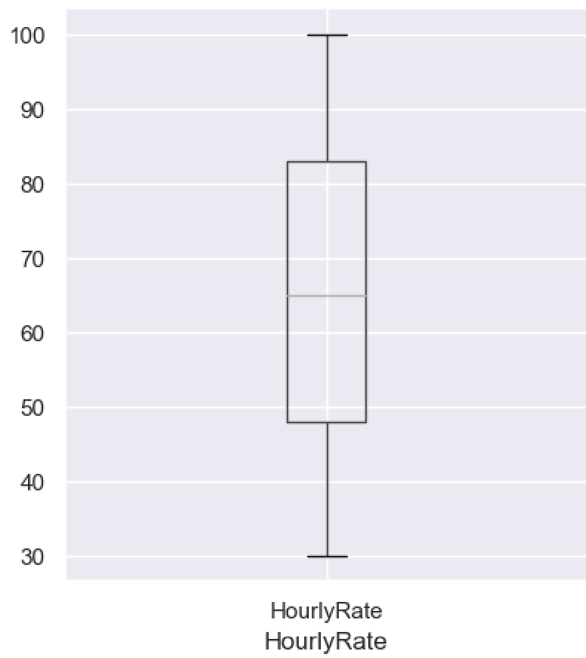
In [42]:
```
for var in continious:

    # BoxPlot...
    plt.figure(figsize=(10,5))
    plt.subplot(1,2,1)
    fig=data.boxplot(column=var)
    fig.set_xlabel(var)

    # HistPlot...
    plt.subplot(1,2,2)
    fig=data[var].hist(bins=20)
    fig.set_ylabel('No. of Employees')
    fig.set_xlabel(var)

    plt.show()
```

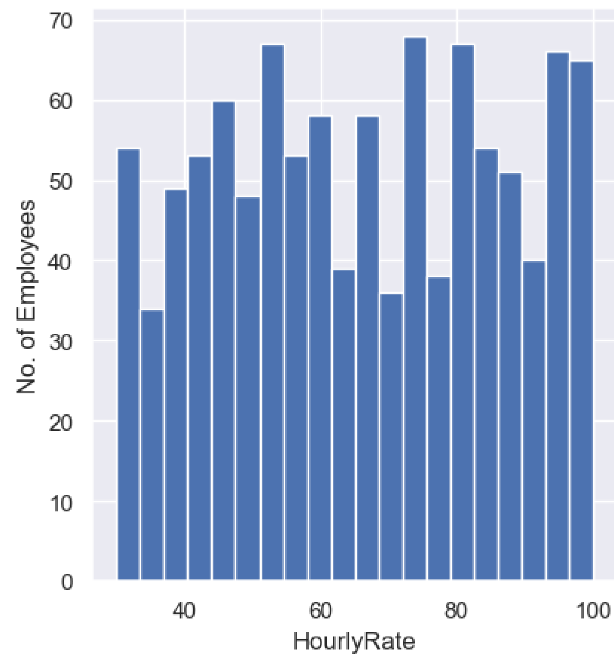


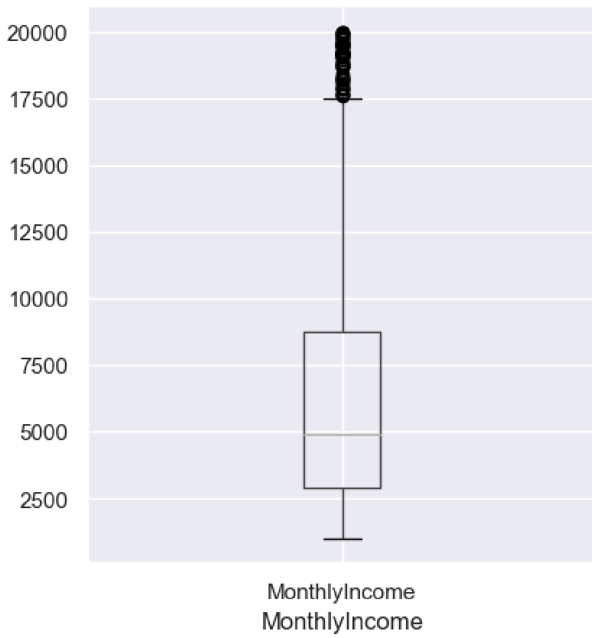


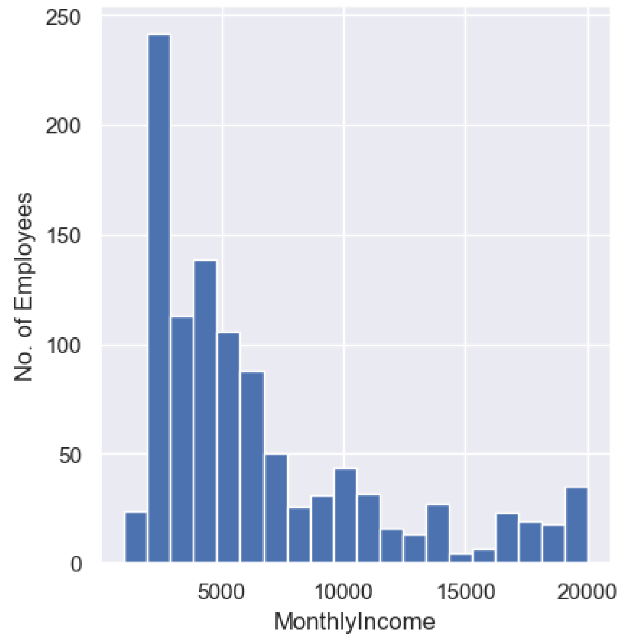

1400
1200
1000
800
600
400
200
DailyRate
DailyRate

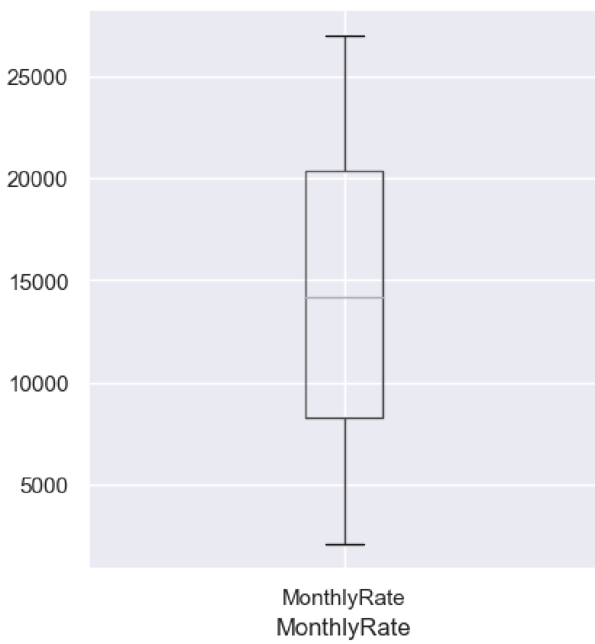

60
50
40
30
20
10
0
No. of Employees
250
500
750
1000
1250
1500
DailyRate

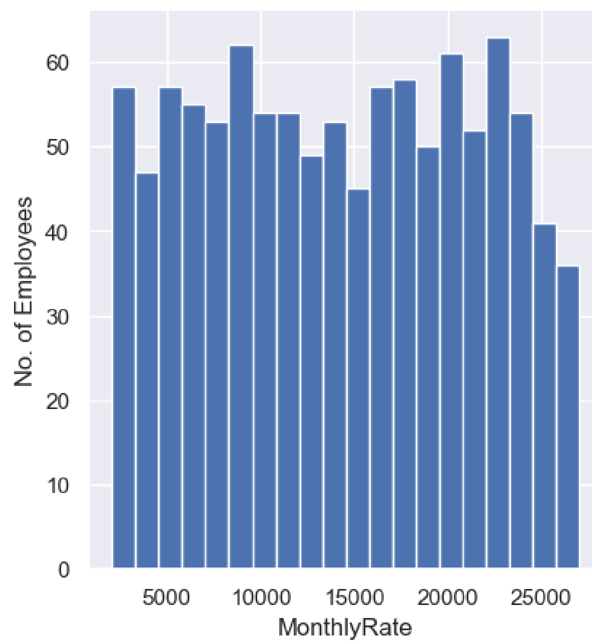

100
90
80
70
60
50
40
30
HourlyRate
HourlyRate

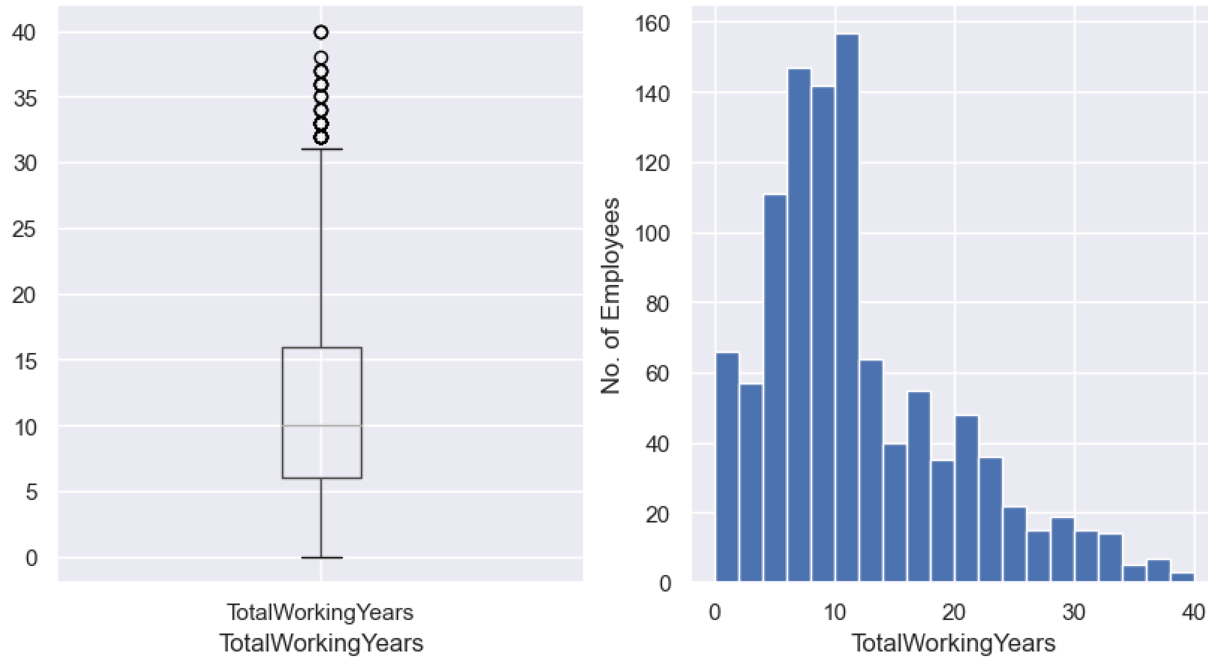

70
60
50
40
30
20
10
0
No. of Employees
40
60
80
100
HourlyRate

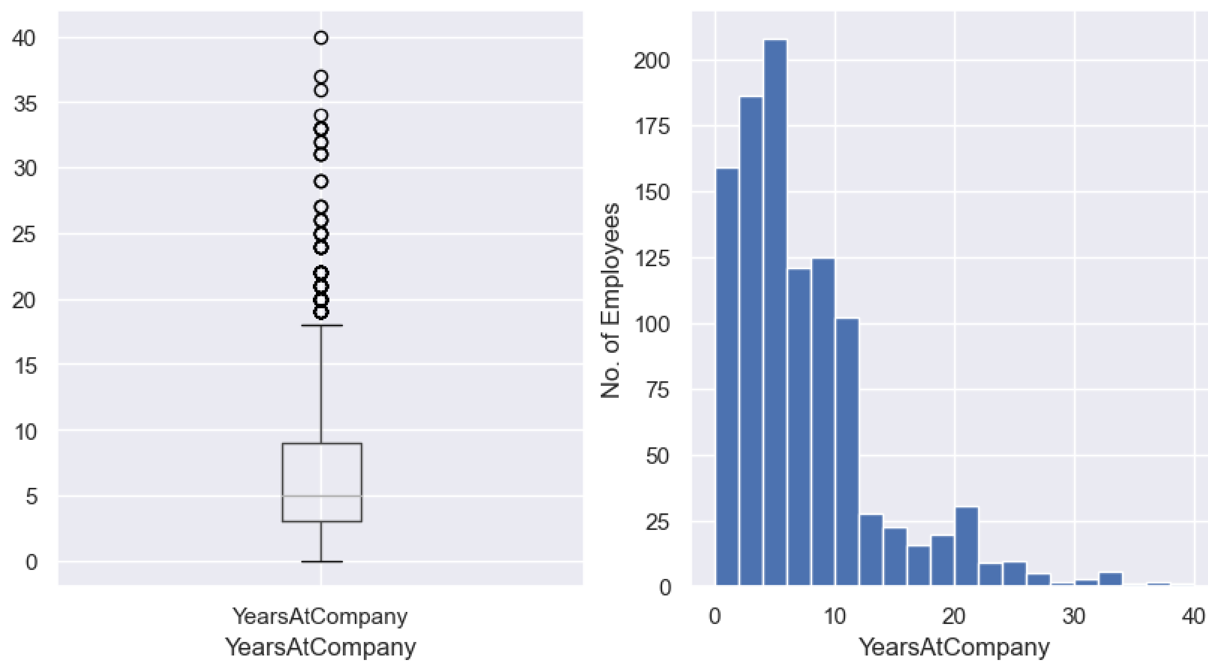

20000
17500
15000
12500
10000
7500
5000
2500
MonthlyIncome
MonthlyIncome


250
200
150
100
50
0
No. of Employees
5000
10000
15000
20000
MonthlyIncome


25000
20000
15000
10000
5000
MonthlyRate
MonthlyRate


60
50
40
30
20
10
0
No. of Employees
5000
10000
15000
20000
25000
MonthlyRate





In [43]:

```
data['TotalWorkingYears'].describe()
```

Out[43]:

```
count    1058.000000
mean       11.435728
std         8.016429
min         0.000000
25%         6.000000
50%        10.000000
75%        16.000000
max        40.000000
Name: TotalWorkingYears, dtype: float64
```

In [44]:

```
categorical.head()
```

Out[44]:

	BusinessTravel	Department	EducationField	Gender	JobRole	MaritalStatus	Over18	OverTim
0	Travel_Rarely	Sales	Life Sciences	Female	Sales Executive	Single	Y	Ye
1	Travel_Frequently	Research & Development	Life Sciences	Male	Research Scientist	Married	Y	N
2	Travel_Rarely	Research & Development	Other	Male	Laboratory Technician	Single	Y	Ye
3	Travel_Frequently	Research & Development	Life Sciences	Female	Research Scientist	Married	Y	Ye
4	Travel_Rarely	Research & Development	Medical	Male	Laboratory Technician	Married	Y	N

In [45]:
```
data_cat=pd.get_dummies(categorical)
```

In [46]:
```
data_cat.head()
```

Out[46]:

	BusinessTravel_Non-Travel	BusinessTravel_Travel_Frequently	BusinessTravel_Travel_Rarely	Department_Hun Resour
0	0	0	1	
1	0	1	0	
2	0	0	1	
3	0	1	0	
4	0	0	1	

5 rows × 29 columns

In [47]:
```
numerical.head()
```

Out[47]:

	Age	Attrition	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumber	Envi
0	41	1	1102	1	2	1	1	
1	49	0	279	8	1	1	2	
2	37	1	1373	2	2	1	4	
3	33	0	1392	3	4	1	5	
4	27	0	591	2	1	1	7	

5 rows × 27 columns

In [49]:
```
data_final=pd.concat([numerical,data_cat], axis=1)
```

In [50]:
```
data_final.head()
```

Out[50]:

	Age	Attrition	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumber	Envir
0	41	1	1102	1	2	1	1	
1	49	0	279	8	1	1	2	
2	37	1	1373	2	2	1	4	
3	33	0	1392	3	4	1	5	
4	27	0	591	2	1	1	7	

5 rows × 56 columns

In [52]:

```
data_final=data_final.drop('Attrition',axis=1)
data_final
```

Out[52]:

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumber	Environmen
0	41	1102	1	2	1	1	
1	49	279	8	1	1	2	
2	37	1373	2	2	1	4	
3	33	1392	3	4	1	5	
4	27	591	2	1	1	7	
...	...	...	...	...	...	...	
1053	57	405	1	2	1	1483	
1054	49	1490	7	4	1	1484	
1055	34	829	15	3	1	1485	
1056	28	1496	1	3	1	1486	
1057	29	115	13	3	1	1487	

1058 rows × 55 columns

In [56]:

```
target=data['Attrition']
```

Build Basline Models

In [58]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
```

In [60]:
```
X_train,X_test,y_train,y_test = train_test_split(data_final , target ,test_size=0.2, r
```

In [61]:
```
X_train.shape
```

Out[61]:
```
(846, 55)
```

In [62]:
```
X_test.shape
```

Out[62]:
```
(212, 55)
```

In [65]:
```
model=RandomForestClassifier()
model.fit(X_train,y_train)
y_pred=model.predict(X_test)
print('Accuracy : ', accuracy_score(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
Accuracy :  0.8915094339622641
              precision    recall  f1-score   support

           0       0.89      0.99      0.94       182
           1       0.89      0.27      0.41        30

    accuracy                           0.89       212
   macro avg       0.89      0.63      0.68       212
weighted avg       0.89      0.89      0.87       212
```

In [67]:
```
model=LogisticRegression()
model.fit(X_train,y_train)
y_predictions=model.predict(X_test)
print('Accuracy:',accuracy_score(y_test,y_predictions))
print(classification_report(y_test,y_predictions))
```

```
Accuracy: 0.8632075471698113
              precision    recall  f1-score   support

           0       0.86      1.00      0.93       182
           1       1.00      0.03      0.06        30

    accuracy                           0.86       212
   macro avg       0.93      0.52      0.50       212
weighted avg       0.88      0.86      0.80       212
```

In [69]:
```
model=DecisionTreeClassifier()
model.fit(X_train,y_train)
y_predictions = model.predict(X_test)
print('Accuracy : ',accuracy_score(y_test,y_predictions))
print(classification_report(y_test,y_predictions))
```

```
Accuracy :  0.7830188679245284
              precision    recall  f1-score   support

           0       0.89      0.86      0.87       182
           1       0.28      0.33      0.30        30

    accuracy                           0.78       212
   macro avg       0.58      0.60      0.59       212
weighted avg       0.80      0.78      0.79       212
```

In [71]:

```
model = KNeighborsClassifier()
model.fit(X_train,y_train)
y_predictions = model.predict(X_test)
print('Accuracy', accuracy_score(y_test,y_predictions))
print(classification_report(y_test,y_predictions))
```

```
Accuracy 0.8254716981132075
              precision    recall  f1-score   support

           0       0.86      0.95      0.90       182
           1       0.18      0.07      0.10        30

    accuracy                           0.83       212
   macro avg       0.52      0.51      0.50       212
weighted avg       0.76      0.83      0.79       212
```

In [73]:

```
model = SVC()
model.fit(X_train,y_train)
model_predictions = model.predict(X_test)
print('Accuracy : ', accuracy_score(y_test,y_predictions))
print(classification_report(y_test,y_predictions))
```

```
Accuracy :  0.8254716981132075
              precision    recall  f1-score   support

           0       0.86      0.95      0.90       182
           1       0.18      0.07      0.10        30

    accuracy                           0.83       212
   macro avg       0.52      0.51      0.50       212
weighted avg       0.76      0.83      0.79       212
```

Insights

In above I use some Machine Learning Algorithm to find the Accuracy Score of Test data and Predict Data:

1. First I use a Train-Test Split to evaluate a machine learning model's ability to predict a certain outcome accurately when exposed to real-world data it's never seen before. In a train-test split, I split an original dataset into two subsets—a training dataset and a testing dataset. Depending on the nature and complexity of the data. Then find a Predict data and move to next step.

1. Next I use a Five Machine Learning Algorithm i.e. :

- "Random Forest Classifier" where Accuracy is 0.8915094339622641
- "Logistic Regression" where Accuracy is 0.8632075471698113
- "Decision Tree Classifier" where Accuracy is 0.7830188679245284
- "K Neighbours Classifier" where Accuracy is 0.8254716981132075
- "SVC" where Accuracy is 0.8254716981132075

In [ ]: