IBM HR Analytics Employee Attrition & Performance

- sample 1, Sample 2, Sample 3, Sample 4
- Sample 5*

In [1]:
```
import sys
sys.path.append('../')
```

In [52]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import warnings

warnings.filterwarnings('ignore',  category=FutureWarning)

from fxEDA import *
from fxMachineLearning import *
```

In [3]:
```
df = pd.read_csv('./datasets/IBM HR Employee Attrition.csv')
df.sample(10)
```

Out[3]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeI
1294	41	No	Travel_Rarely	447	Research & Development	5	3	Life Sciences	1	
141	45	No	Travel_Rarely	1316	Research & Development	29	3	Medical	1	
1019	36	No	Travel_Rarely	329	Sales	16	4	Marketing	1	
504	45	Yes	Travel_Frequently	306	Sales	26	4	Life Sciences	1	
170	27	No	Travel_Rarely	1157	Research & Development	17	3	Technical Degree	1	
863	33	No	Travel_Rarely	147	Human Resources	2	3	Human Resources	1	
449	39	No	Travel_Frequently	443	Research & Development	8	1	Life Sciences	1	
445	55	No	Travel_Rarely	1117	Sales	18	5	Life Sciences	1	
1448	41	No	Travel_Rarely	930	Sales	3	3	Life Sciences	1	
159	34	No	Travel_Frequently	303	Sales	2	4	Marketing	1	

10 rows × 35 columns

Education

1. 'Below College'
2. 'College'
3. 'Bachelor'
4. 'Master'
5. 'Doctor'

EnvironmentSatisfaction / JobInvolvement / JobSatisfaction / RelationshipSatisfaction

1. 'Low'
2. 'Medium'
3. 'High'
4. 'Very High'

PerformanceRating

1. 'Low'
2. 'Good'
3. 'Excellent'
4. 'Outstanding'

WorkLifeBalance

1. 'Bad'
2. 'Good'
3. 'Better'
4. 'Best'

In [4]:
```
df.shape
```

Out[4]: (1470, 35)

In [5]:
```
df.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
```

In [6]:
```
#fxMissingValues(df)
```

In [7]:
```
df.groupby('Attrition').agg(['count', 'nunique']).T
```

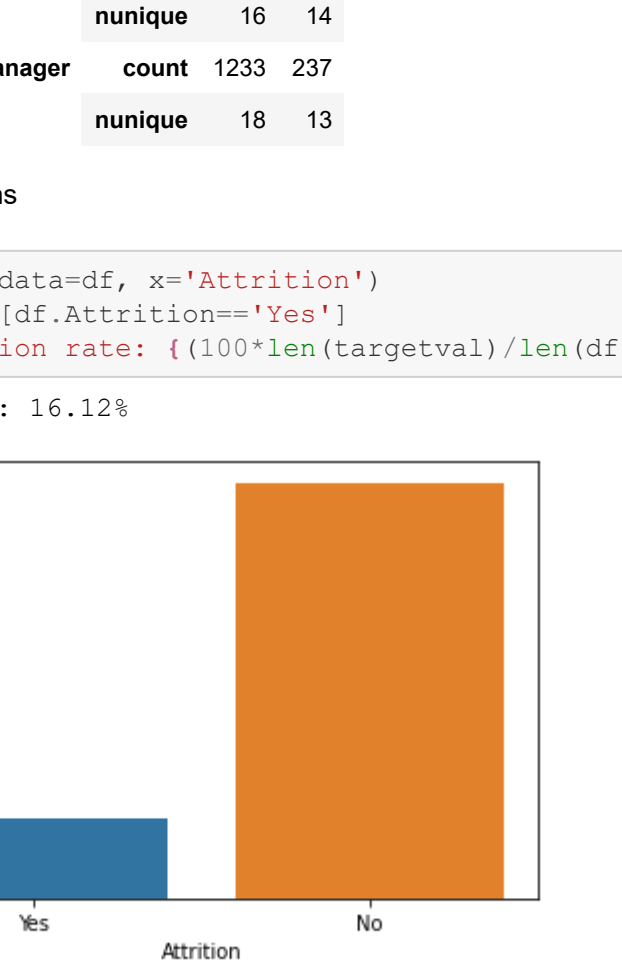
Out[7]:

	Attrition	No	Yes
Age	**count**	1233	237
	nunique	43	39
BusinessTravel	**count**	1233	237
	nunique	3	3
DailyRate	**count**	1233	237
...	**...**	...	...
YearsInCurrentRole	**nunique**	19	15
YearsSinceLastPromotion	**count**	1233	237
	nunique	16	14
YearsWithCurrManager	**count**	1233	237
	nunique	18	13

68 rows × 2 columns

In [8]:
```
sns.countplot(data=df, x='Attrition')
targetval = df[df.Attrition=='Yes']
print(f'Attrition rate: {(100*len(targetval)/len(df)):.2f}%')
```

Attrition rate: 16.12%



In [9]:
```
dfx = df.copy()
dfx.drop(['EmployeeNumber', 'EmployeeCount', 'StandardHours', 'Over18'], axis=1, inplace=True)
```

In [10]:
```
# mapping some numerical value to categorical
dfx.Education = dfx.Education.map({1: 'Below College', 2: 'College', 3: 'Bachelor', 4: 'Master', 5: 'Doctor'})

satisfactiondict={1: 'Low', 2: 'Medium', 3: 'High', 4: 'Very High'}
dfx.EnvironmentSatisfaction = dfx.EnvironmentSatisfaction.map(satisfactiondict)
dfx.JobInvolvement = dfx.JobInvolvement.map(satisfactiondict)
dfx.JobSatisfaction = dfx.JobSatisfaction.map(satisfactiondict)
dfx.RelationshipSatisfaction= dfx.RelationshipSatisfaction.map(satisfactiondict)

dfx.PerformanceRating = dfx.PerformanceRating.map({1: 'Low', 2: 'Good', 3: 'Excellent', 4: 'Outstanding'})

dfx.WorkLifeBalance = dfx.WorkLifeBalance.map({1: 'Bad', 2: 'Good', 3: 'Better', 4: 'Best'})
```

In [11]:
```
dfx.JobLevel = dfx.JobLevel.astype('category')
dfx.StockOptionLevel = dfx.StockOptionLevel.astype('category')
```

In [12]:
```
dfn = dfx.select_dtypes(include=np.number)
dfn.iloc[:, :10]
```

Out[12]:

	Age	DailyRate	DistanceFromHome	HourlyRate	MonthlyIncome	MonthlyRate	NumCompaniesWorked	PercentSalaryHike	TotalWork
0	41	1102	1	94	5993	19479	8	11	
1	49	279	8	61	5130	24907	1	23	
2	37	1373	2	92	2090	2396	6	15	
3	33	1392	3	56	2909	23159	1	11	
4	27	591	2	40	3468	16632	9	12	
...	...	...	...	...	...	...	...	...	
1465	36	884	23	41	2571	12290	4	17	
1466	39	613	6	42	9991	21457	4	15	
1467	27	155	4	87	6142	5174	1	20	
1468	49	1023	2	63	5390	13243	2	14	
1469	34	628	8	82	4404	10228	2	12	

1470 rows × 10 columns

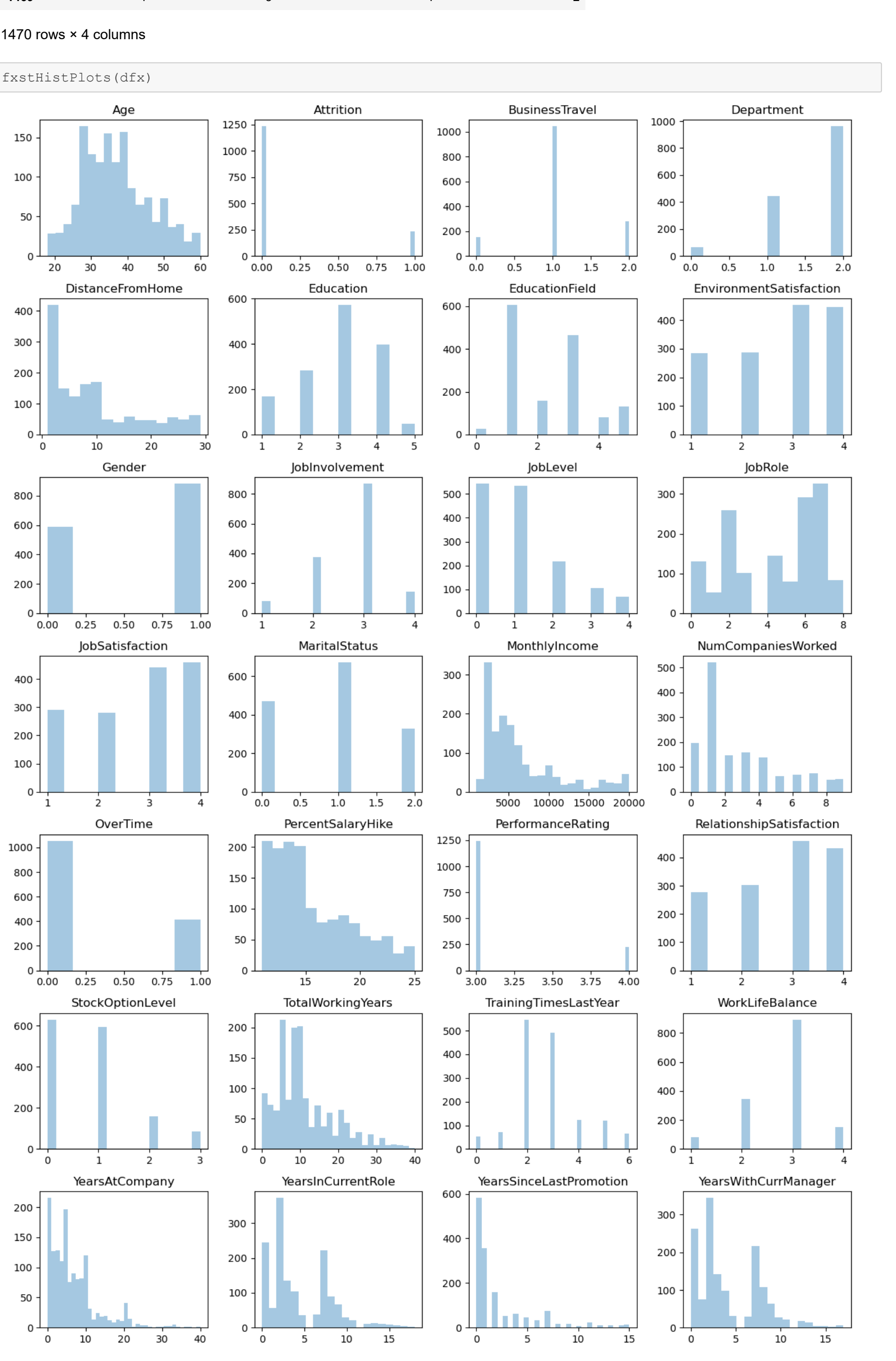
In [13]:
```
dfn.iloc[:, 10:]
```

Out[13]:

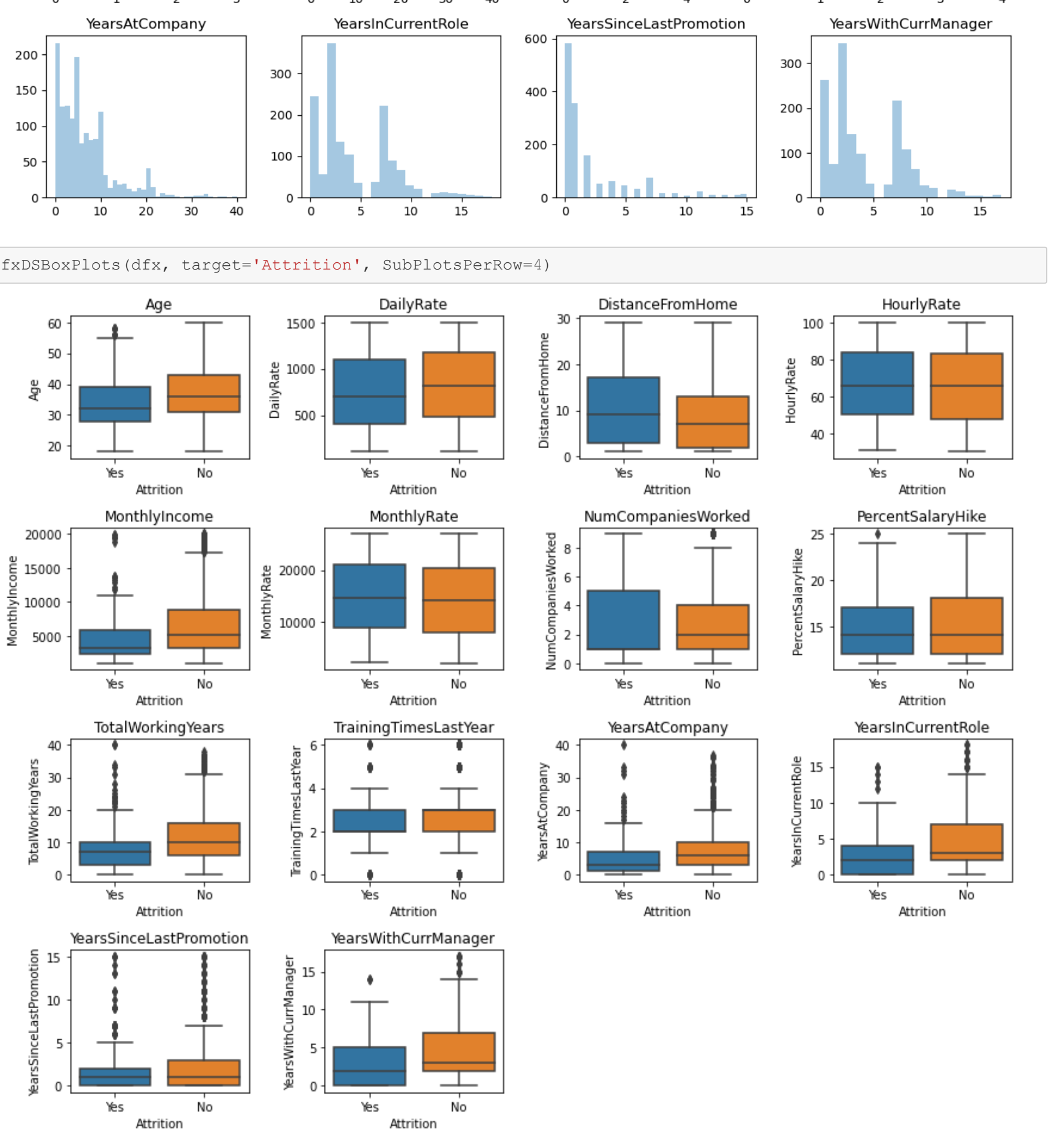
	YearsAtCompany	YearsInCurrentRole	YearsSinceLastPromotion	YearsWithCurrManager
0	6	4	0	5
1	10	7	1	7
2	0	0	0	0
3	8	7	3	0
4	2	2	2	2
...	...	...	...	...
1465	5	2	0	3
1466	7	7	1	7
1467	6	2	0	3
1468	9	6	0	8
1469	4	3	1	2

1470 rows × 4 columns

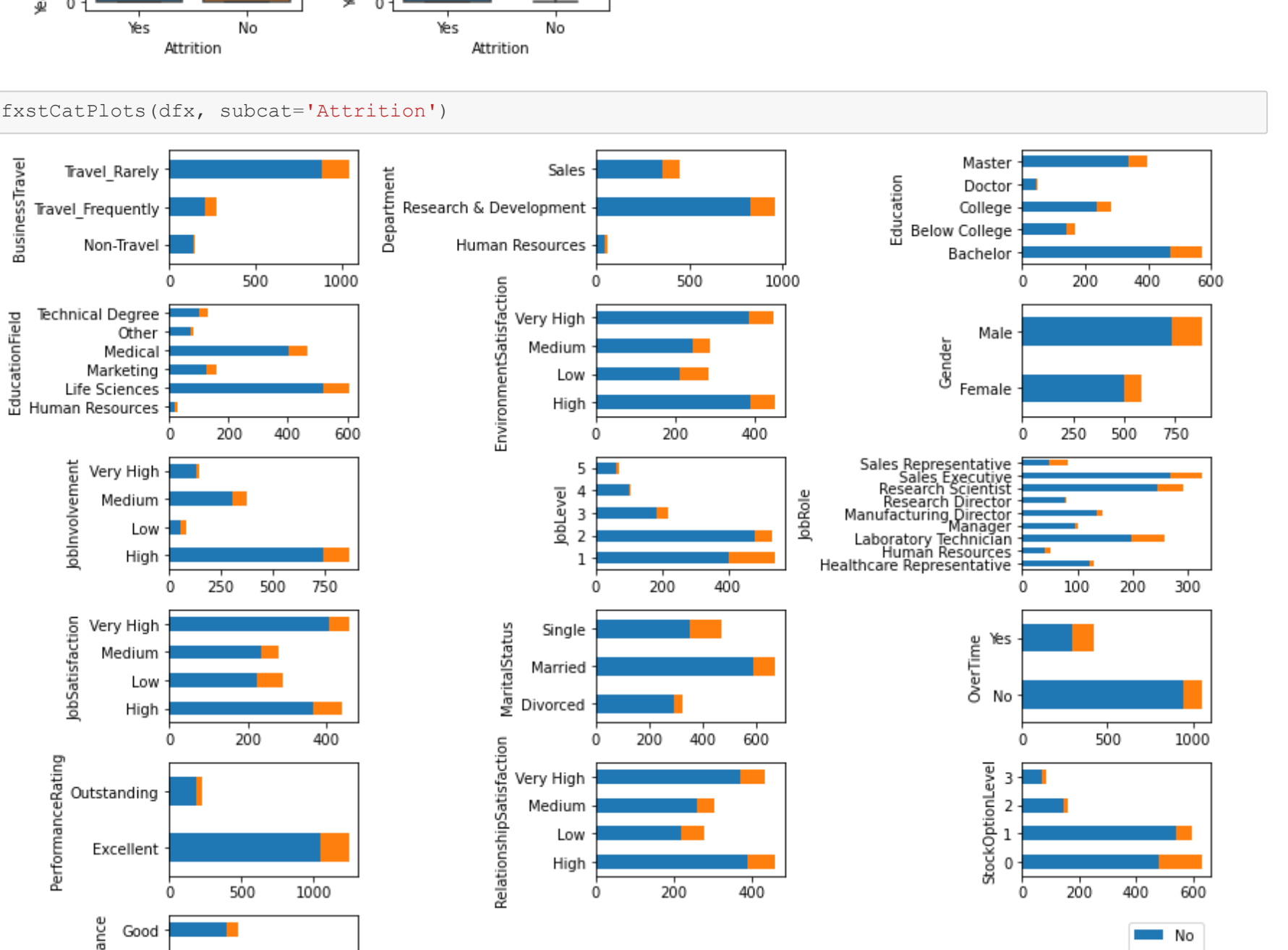
In [53]:
```
fxstHistPlots(dfx)
```



In [15]:
```
fxDSBoxPlots(dfx, target='Attrition', SubPlotsPerRow=4)
```

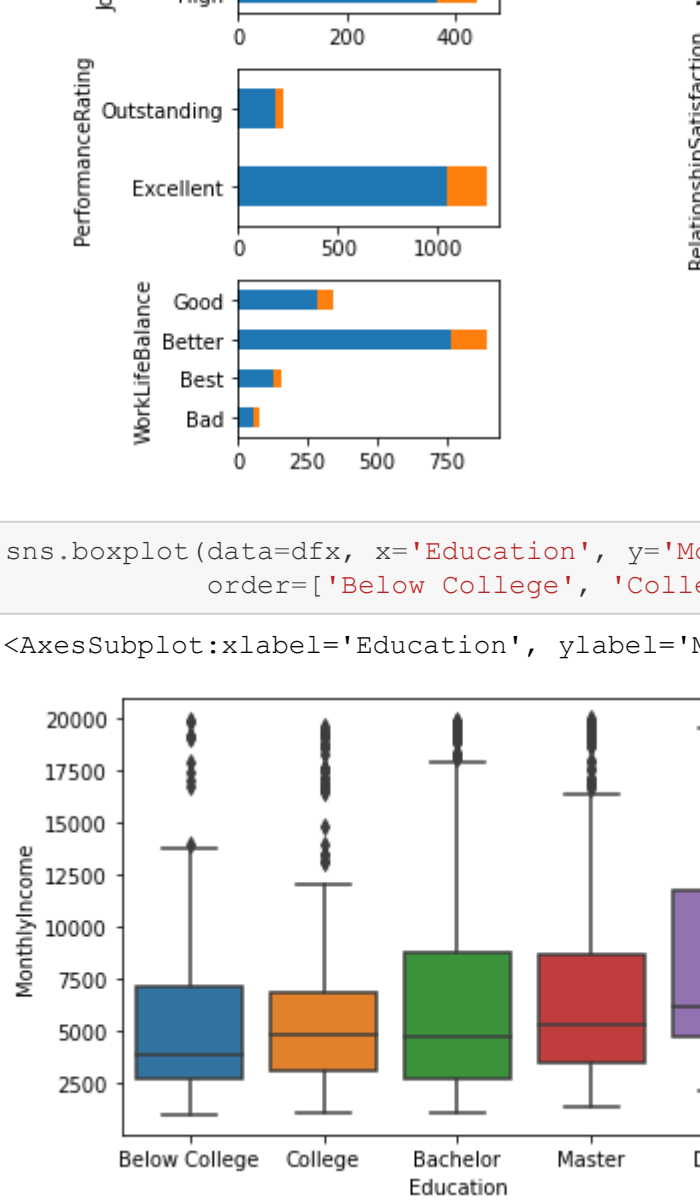


In [16]:
```
fxstCatPlots(dfx, subcat='Attrition')
```

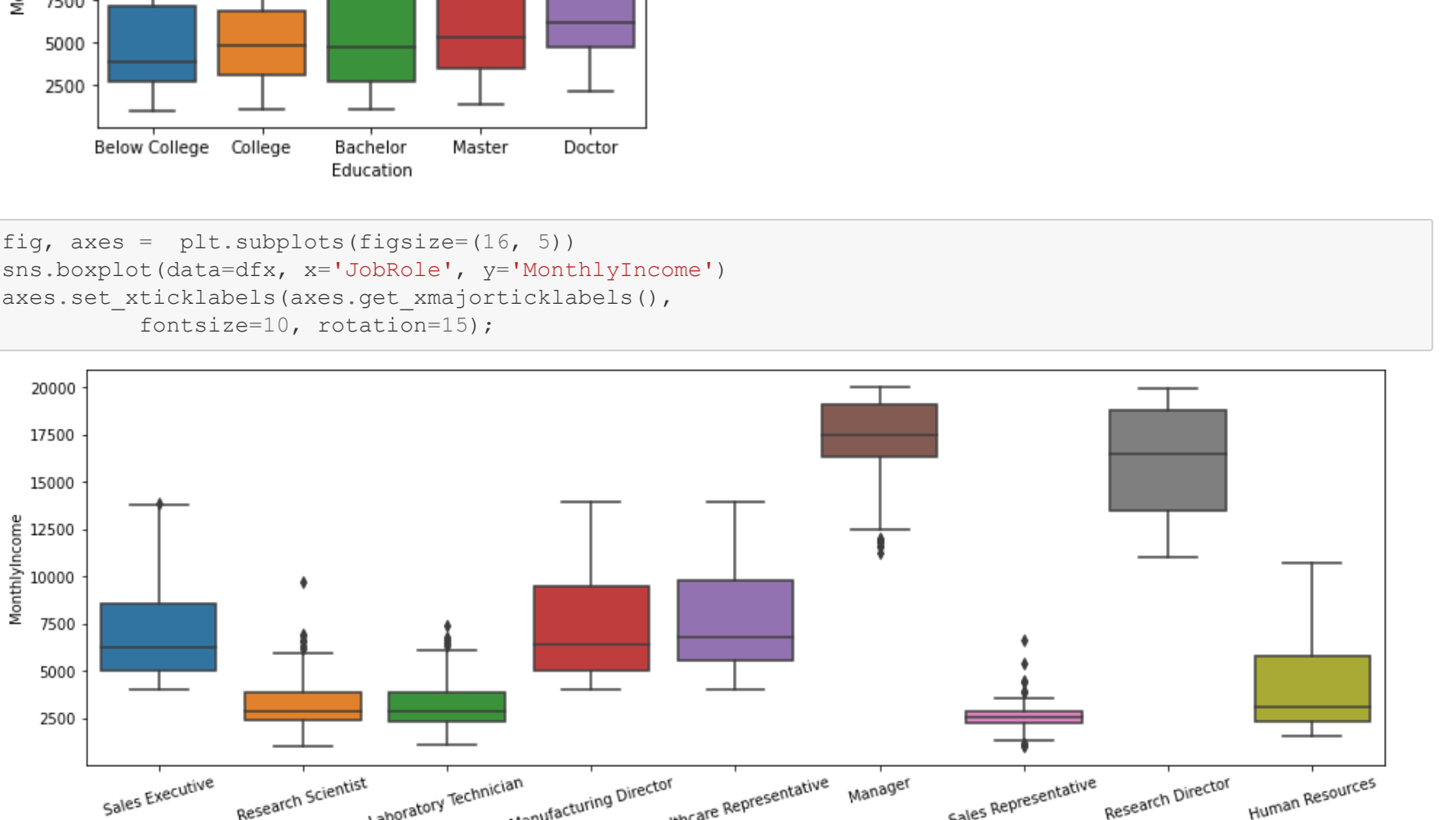


In [17]:
```
sns.boxplot(data=dfx, x='Education', y='MonthlyIncome',
            order=['Below College', 'College', 'Bachelor', 'Master', 'Doctor'])
```

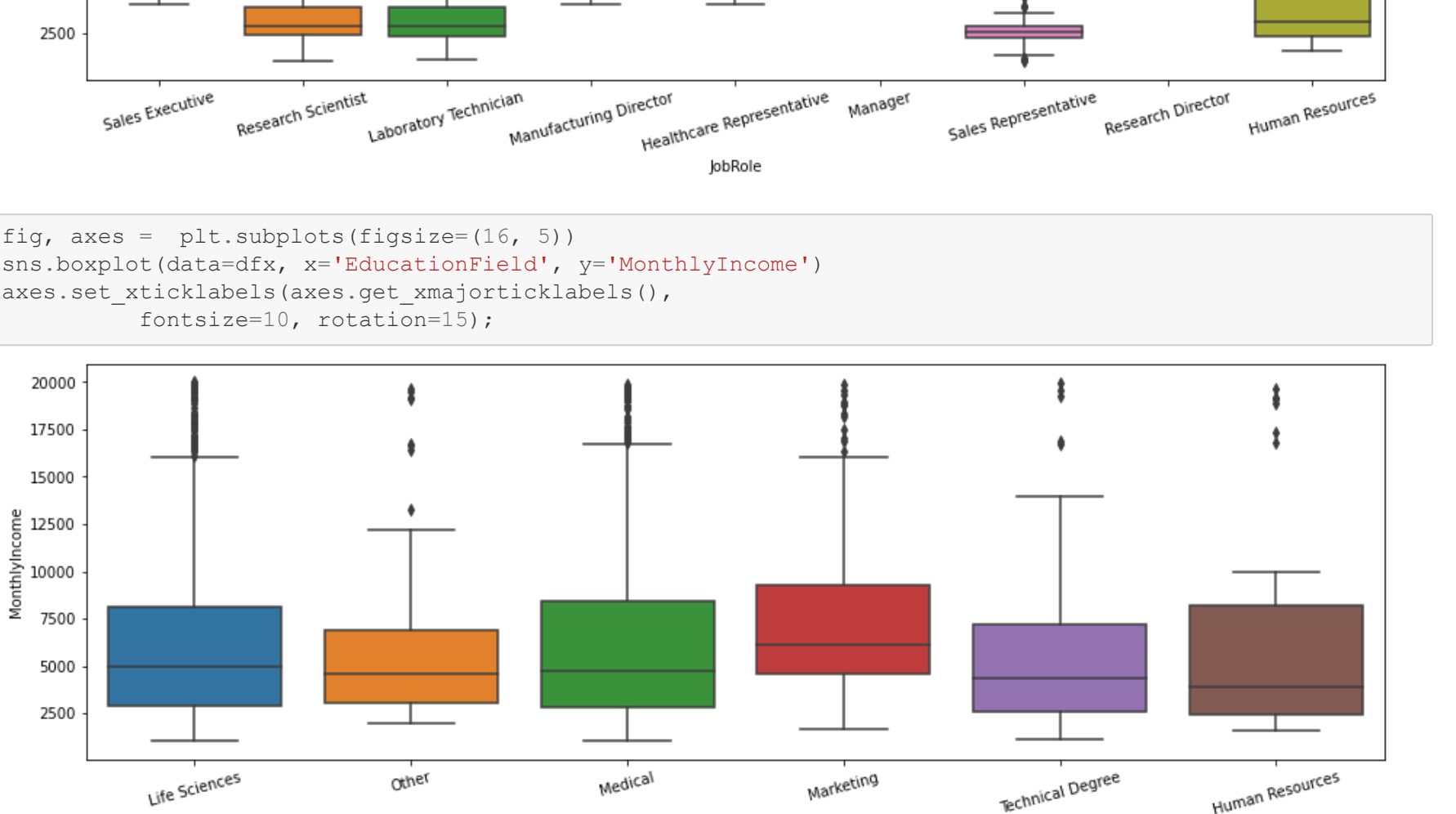
Out[17]: <AxesSubplot:xlabel='Education', ylabel='MonthlyIncome'>



In [18]:
```
fig, axes =  plt.subplots(figsize=(16, 5))
sns.boxplot(data=dfx, x='JobRole', y='MonthlyIncome')
axes.set_xticklabels(axes.get_xmajorticklabels(),
         fontsize=10, rotation=15);
```

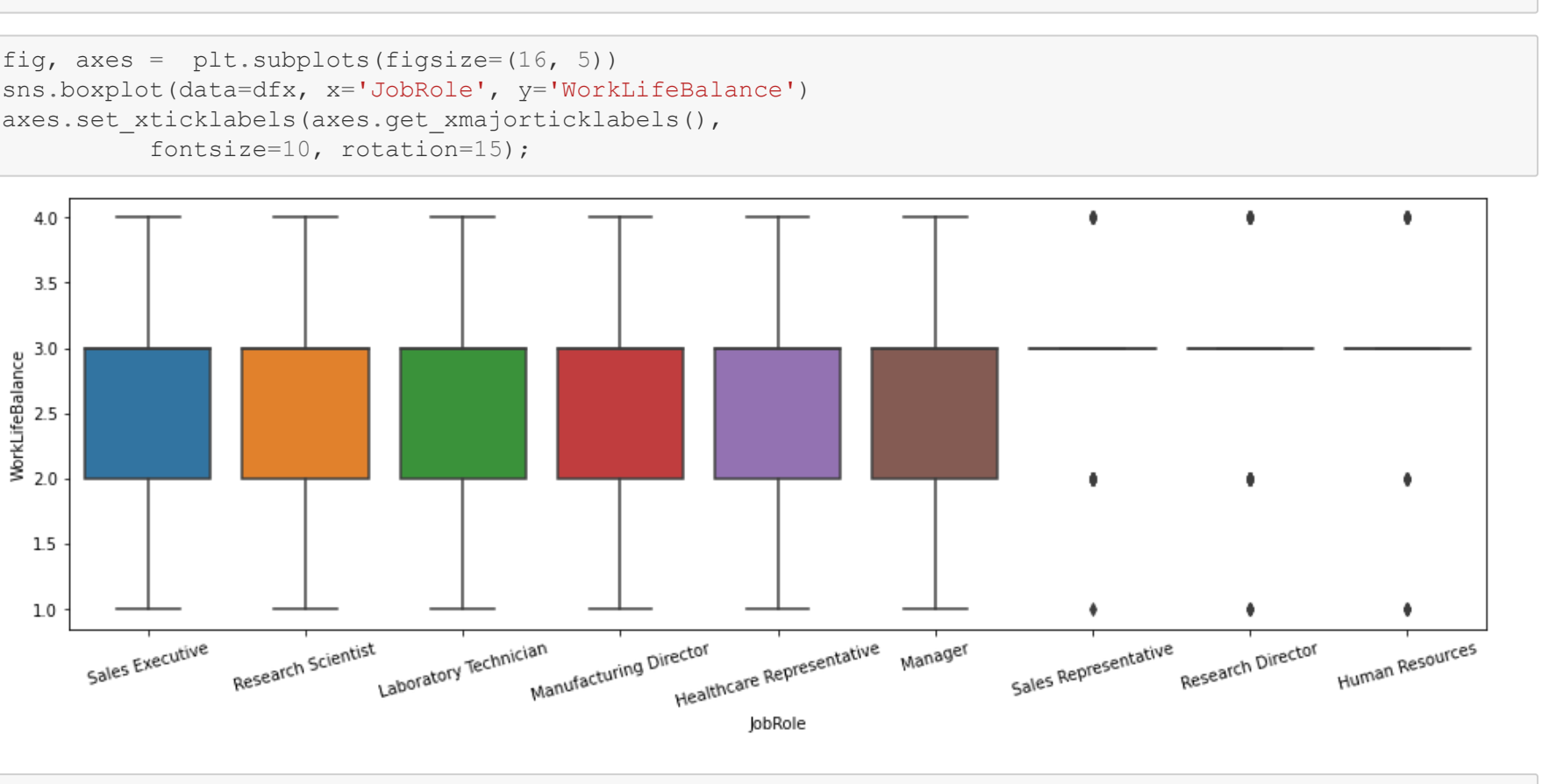


In [19]:
```
fig, axes =  plt.subplots(figsize=(16, 5))
sns.boxplot(data=dfx, x='EducationField', y='MonthlyIncome')
axes.set_xticklabels(axes.get_xmajorticklabels(),
         fontsize=10, rotation=15);
```



In [20]:
```
dfx.WorkLifeBalance = dfx.WorkLifeBalance.map({'Bad': 1, 'Good': 2, 'Better': 3, 'Best': 4})
```

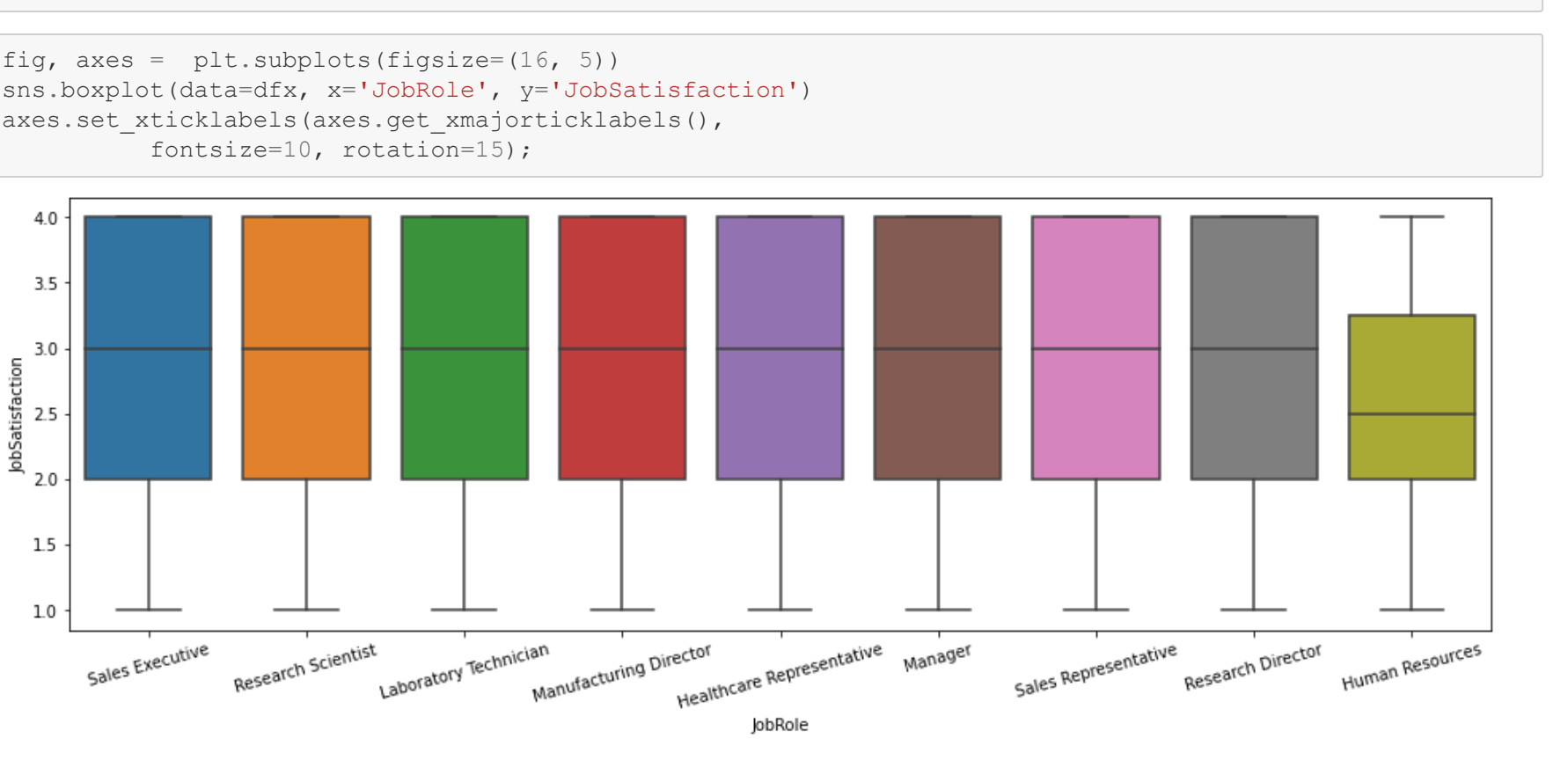
In [21]:
```
fig, axes =  plt.subplots(figsize=(16, 5))
sns.boxplot(data=dfx, x='JobRole', y='WorkLifeBalance')
axes.set_xticklabels(axes.get_xmajorticklabels(),
         fontsize=10, rotation=15);
```



In [22]:
```
satisfactiondict_r = {v: k for k, v in satisfactiondict.items()}
```

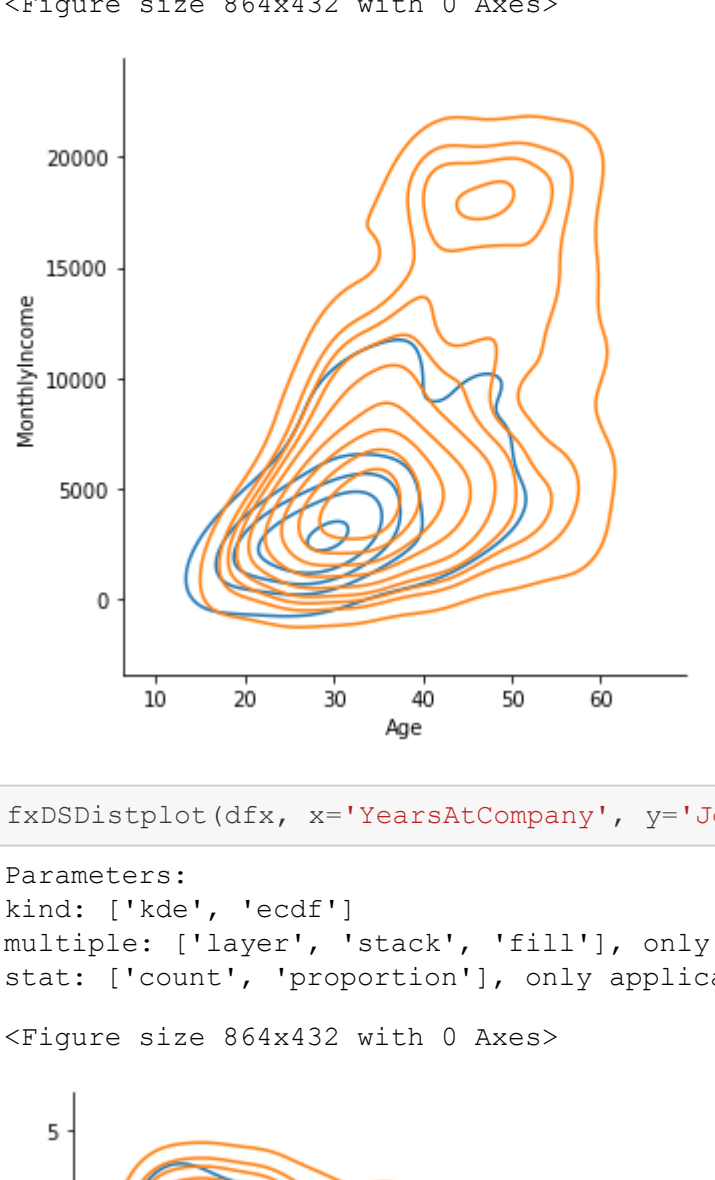
In [23]:
```
dfx.JobSatisfaction = dfx.JobSatisfaction.map(satisfactiondict_r)
dfx.EnvironmentSatisfaction = dfx.EnvironmentSatisfaction.map(satisfactiondict_r)
dfx.JobInvolvement = dfx.JobInvolvement.map(satisfactiondict_r)
dfx.RelationshipSatisfaction = dfx.RelationshipSatisfaction.map(satisfactiondict_r)

dfx.PerformanceRating = dfx.PerformanceRating.map({'Low': 1, 'Good': 2, 'Excellent': 3, 'Outstanding': 4})
```

In [24]:
```
fig, axes =  plt.subplots(figsize=(16, 5))
sns.boxplot(data=dfx, x='JobRole', y='JobSatisfaction')
axes.set_xticklabels(axes.get_xmajorticklabels(),
         fontsize=10, rotation=15);
```



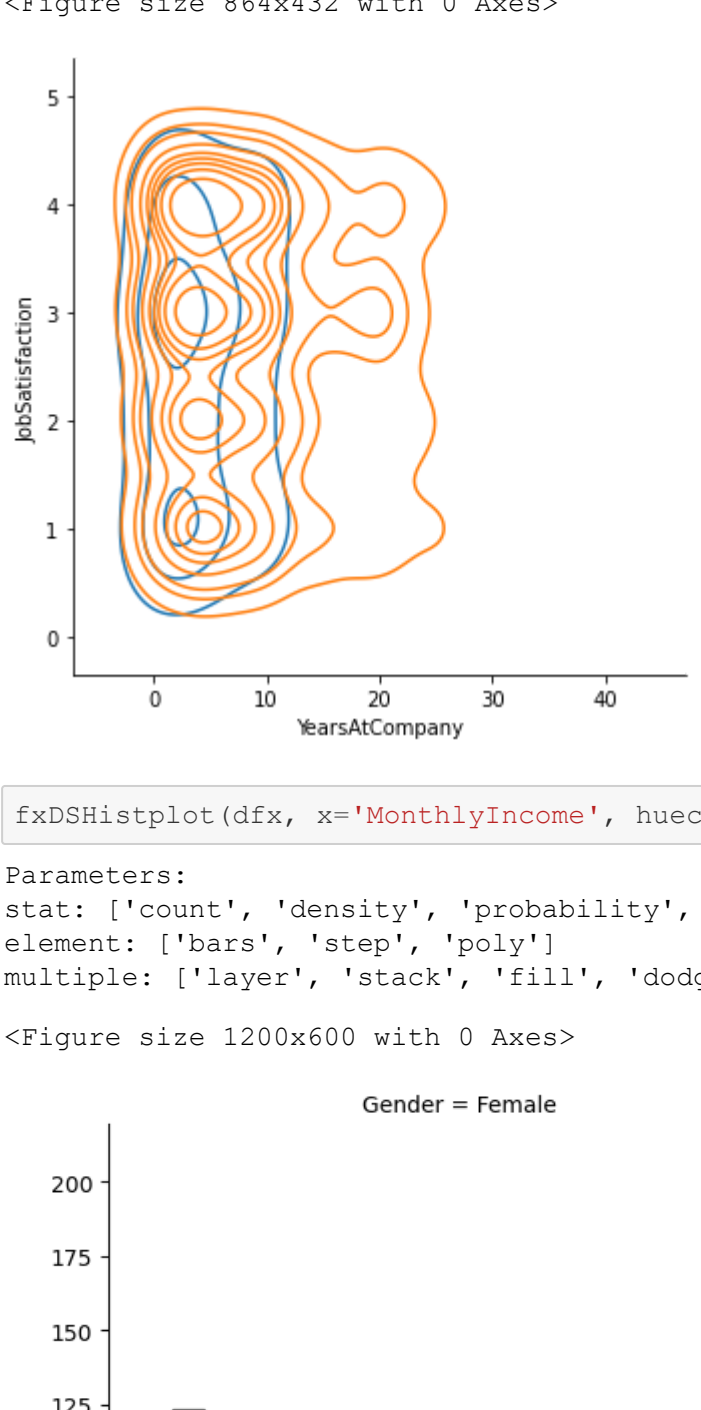
In [26]:
```
fxDSDistplot(dfx, x='Age', y='MonthlyIncome', huecol='Attrition', fill=False);
```

```
Parameters:
kind: ['kde', 'ecdf']
multiple: ['layer', 'stack', 'fill'], only applicable for 'kde'
stat: ['count', 'proportion'], only applicable for 'ecdf'

<Figure size 864x432 with 0 Axes>
```



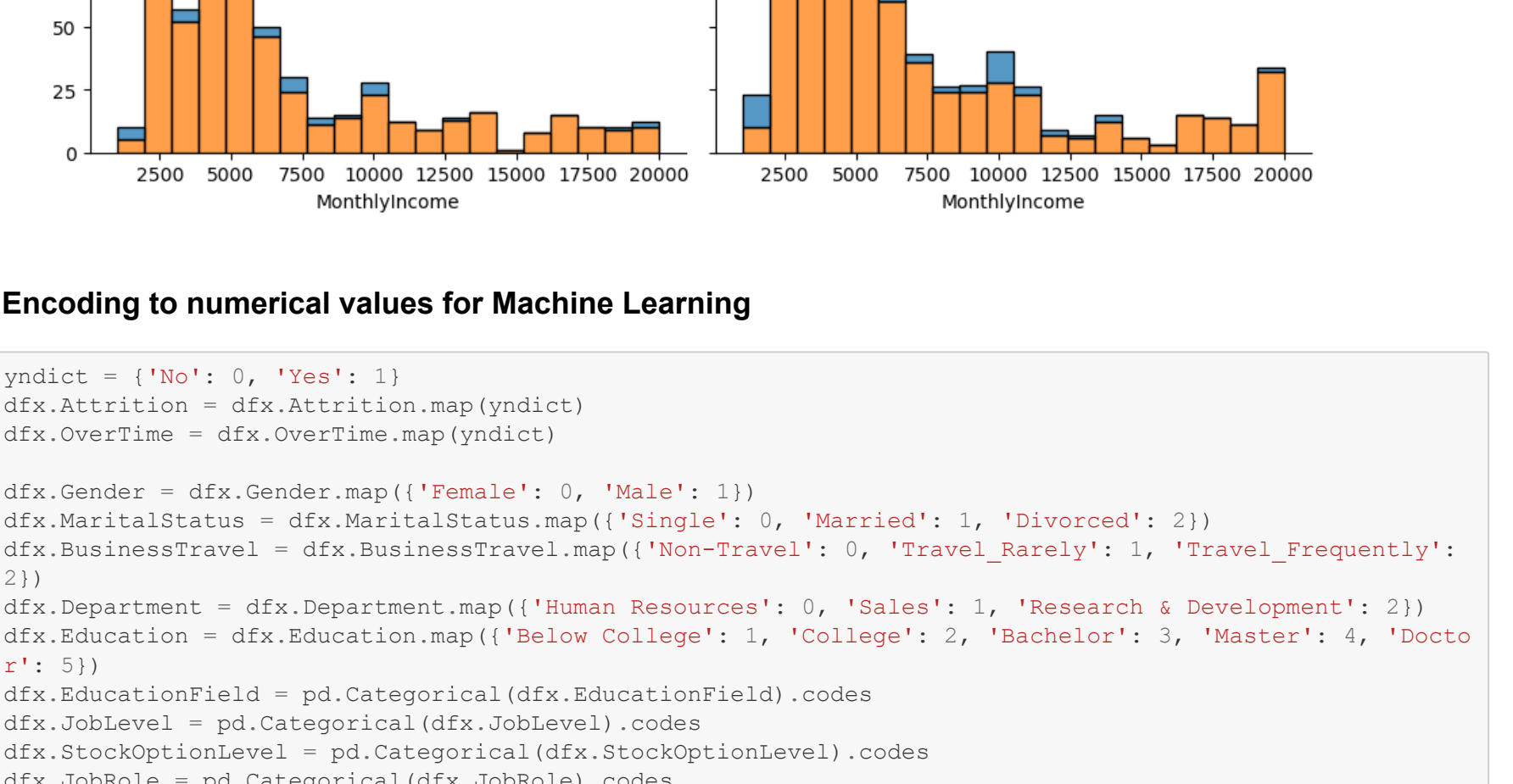
In [27]:
```
fxDSDistplot(dfx, x='YearsAtCompany', y='JobSatisfaction', huecol='Attrition', fill=False);
```

```
Parameters:
kind: ['kde', 'ecdf']
multiple: ['layer', 'stack', 'fill'], only applicable for 'kde'
stat: ['count', 'proportion'], only applicable for 'ecdf'

<Figure size 864x432 with 0 Axes>
```



In [134]:
```
fxDSHistplot(dfx, x='MonthlyIncome', huecol='Attrition', multiple='stack', mcol='Gender');
```

```
Parameters:
stat: ['count', 'density', 'probability', 'frequency']
element: ['bars', 'step', 'poly']
multiple: ['layer', 'stack', 'fill', 'dodge']

<Figure size 1200x600 with 0 Axes>
```



Encoding to numerical values for Machine Learning

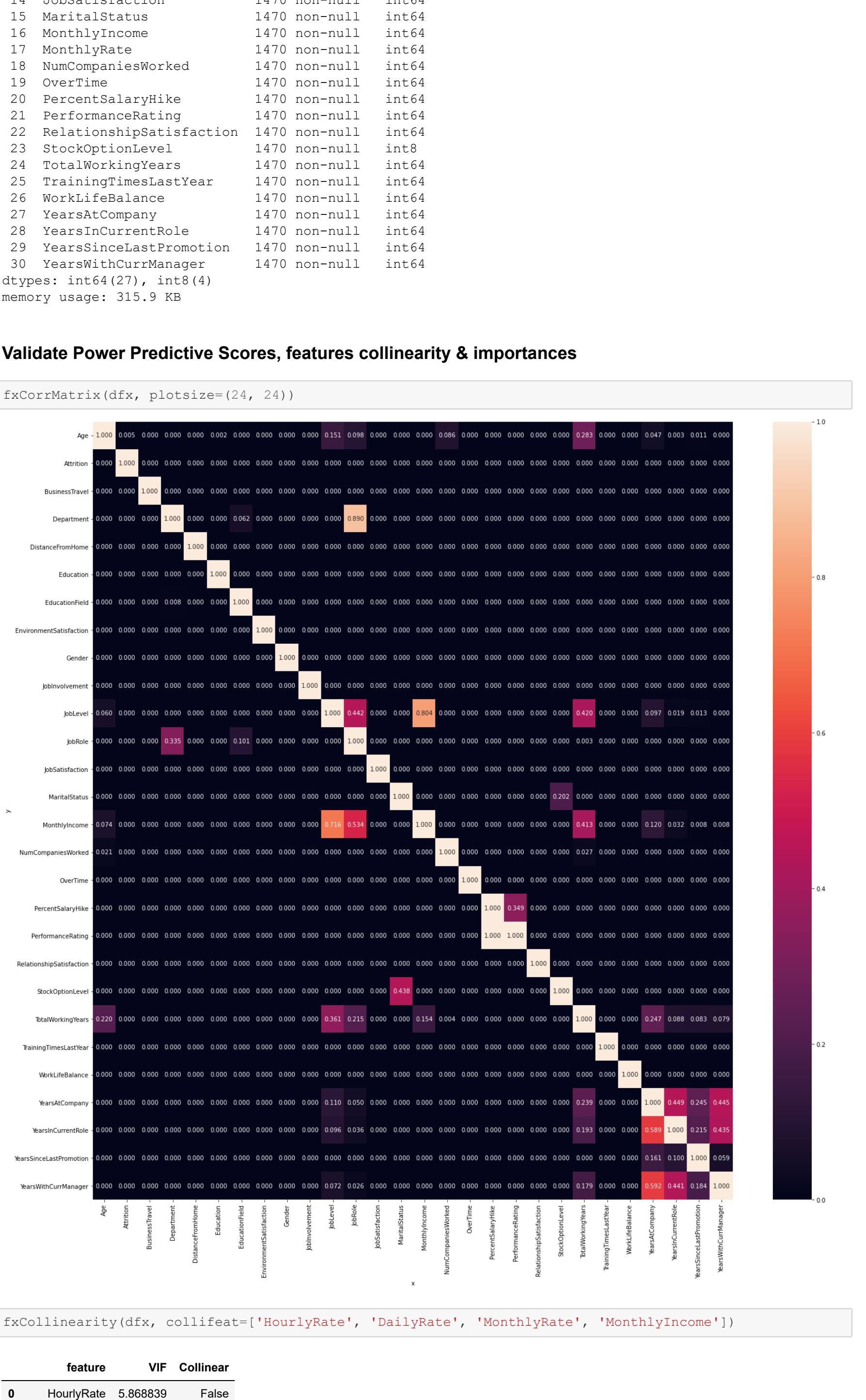
In [29]:
```
yndict = {'No': 0, 'Yes': 1}
dfx.Attrition = dfx.Attrition.map(yndict)
dfx.OverTime = dfx.OverTime.map(yndict)

dfx.Gender = dfx.Gender.map({'Female': 0, 'Male': 1})
dfx.MaritalStatus = dfx.MaritalStatus.map({'Single': 0, 'Married': 1, 'Divorced': 2})
dfx.BusinessTravel = dfx.BusinessTravel.map({'Non-Travel': 0, 'Travel_Rarely': 1, 'Travel_Frequently': 2})
dfx.Department = dfx.Department.map({'Human Resources': 0, 'Sales': 1, 'Research & Development': 2})
dfx.Education = dfx.Education.map({'Below College': 1, 'College': 2, 'Bachelor': 3, 'Master': 4, 'Doctor': 5})
dfx.EducationField = pd.Categorical(dfx.EducationField).codes
dfx.JobLevel = pd.Categorical(dfx.JobLevel).codes
dfx.StockOptionLevel = pd.Categorical(dfx.StockOptionLevel).codes
dfx.JobRole = pd.Categorical(dfx.JobRole).codes
```

In [30]:
```
dfx.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   int64
 2   BusinessTravel            1470 non-null   int64
 3   DailyRate                 1470 non-null   int64
 4   Department                1470 non-null   int64
 5   DistanceFromHome          1470 non-null   int64
 6   Education                 1470 non-null   int64
 7   EducationField            1470 non-null   int8
 8   EnvironmentSatisfaction   1470 non-null   int64
 9   Gender                    1470 non-null   int64
 10  HourlyRate                1470 non-null   int64
 11  JobInvolvement            1470 non-null   int64
 12  JobLevel                  1470 non-null   int8
 13  JobRole                   1470 non-null   int8
 14  JobSatisfaction           1470 non-null   int64
 15  MaritalStatus             1470 non-null   int64
 16  MonthlyIncome             1470 non-null   int64
 17  MonthlyRate               1470 non-null   int64
 18  NumCompaniesWorked        1470 non-null   int64
 19  OverTime                  1470 non-null   int64
 20  PercentSalaryHike         1470 non-null   int64
 21  PerformanceRating         1470 non-null   int64
 22  RelationshipSatisfaction  1470 non-null   int64
 23  StockOptionLevel          1470 non-null   int8
 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(27), int8(4)
memory usage: 315.9 KB
```

Validate Power Predictive Scores, features collinearity & importances

In [41]:
```
fxCorrMatrix(dfx, plotsize=(24, 24))
```



In [33]:
```
fxCollinearity(dfx, collifeat=['HourlyRate', 'DailyRate', 'MonthlyRate', 'MonthlyIncome'])
```

Out[33]:

	feature	VIF	Collinear
0	HourlyRate	5.868839	False
1	DailyRate	4.070855	False
2	MonthlyRate	4.045189	False
3	MonthlyIncome	2.668589	False

In [ ]:

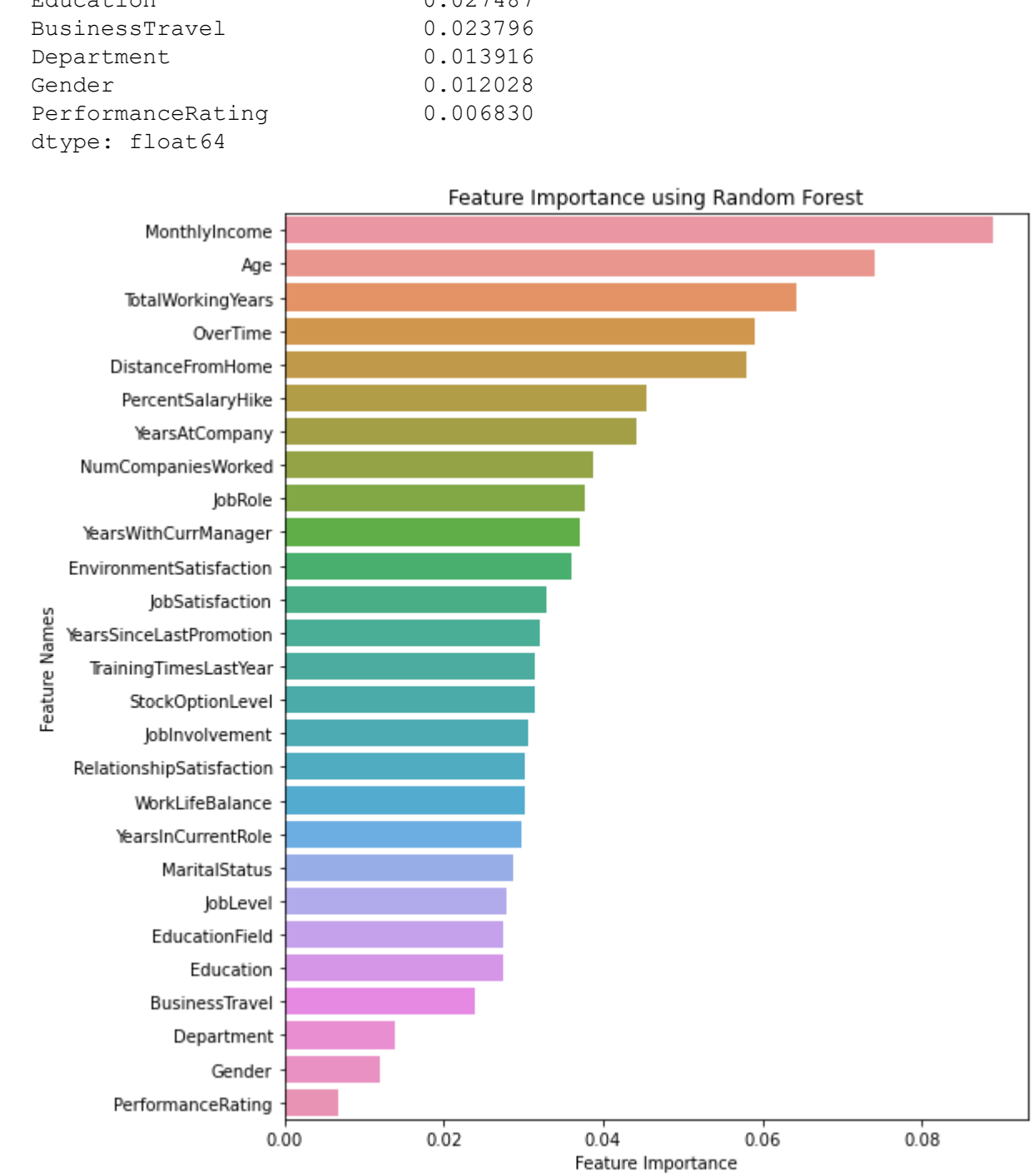
In [34]:
```
dfx.drop(['HourlyRate', 'DailyRate', 'MonthlyRate'], axis=1, inplace=True)
```

In [36]: `fxFeatImportances(dfx, target='Attrition', plotsize=(8, 10))`

```
Set the 'classifier='Gradient Boosting, Ada Boosting, Decision Tree, Light GBM for feature importance
s
--- 0.48 seconds ---
```

Out[36]:

```
MonthlyIncome               0.088927
Age                         0.074031
TotalWorkingYears           0.064352
OverTime                    0.059086
DistanceFromHome            0.058000
PercentSalaryHike           0.045389
YearsAtCompany              0.044266
NumCompaniesWorked          0.038727
JobRole                     0.037719
YearsWithCurrManager        0.036987
EnvironmentSatisfaction     0.036066
JobSatisfaction             0.032855
YearsSinceLastPromotion     0.032045
TrainingTimesLastYear       0.031504
StockOptionLevel            0.031373
JobInvolvement              0.030531
RelationshipSatisfaction    0.030179
WorkLifeBalance             0.030163
YearsInCurrentRole          0.029727
MaritalStatus               0.028641
JobLevel                    0.027845
EducationField              0.027529
Education                   0.027487
BusinessTravel              0.023796
Department                  0.013916
Gender                      0.012028
PerformanceRating           0.006830
dtype: float64
```

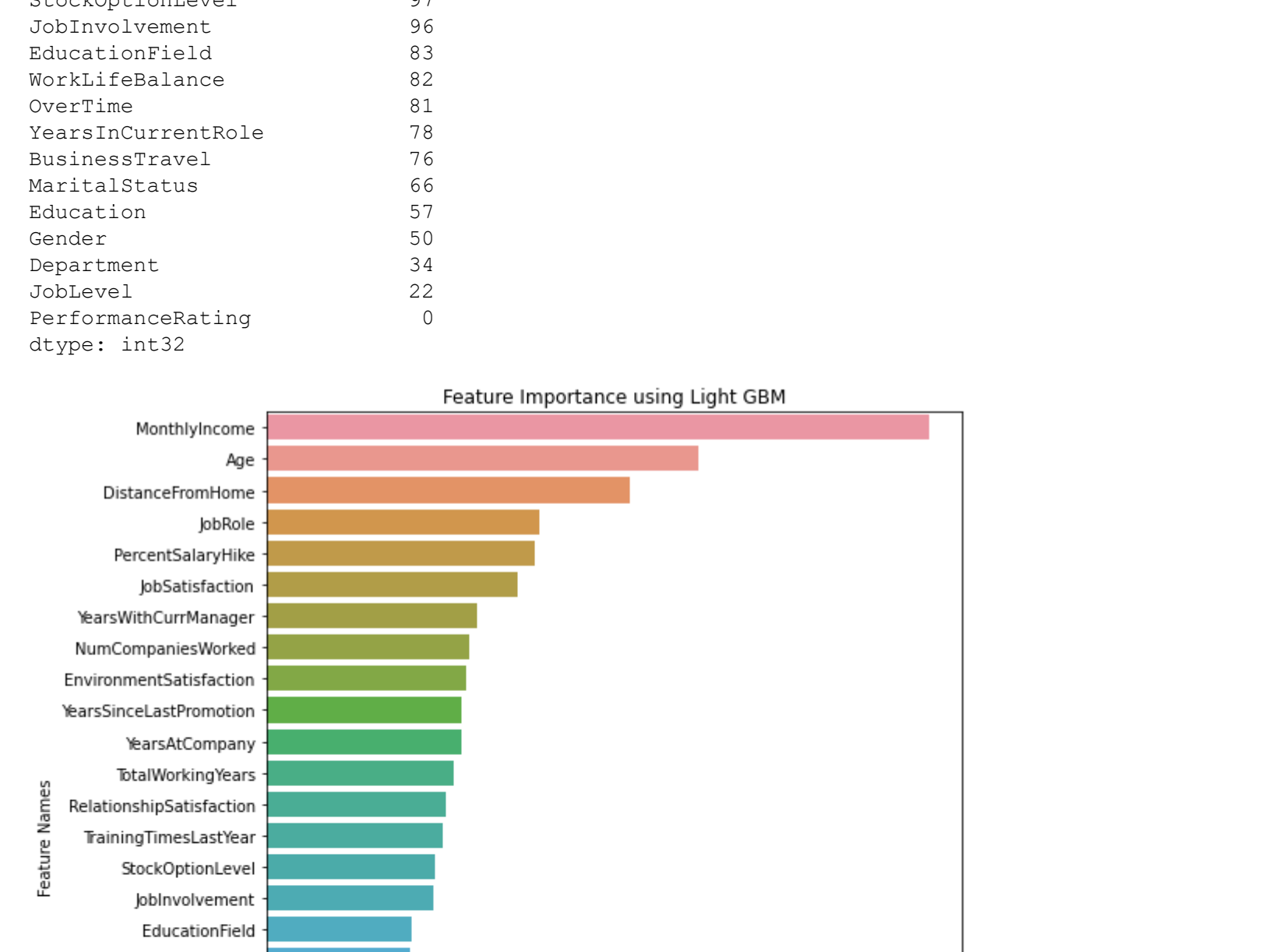


In [37]: `fxFeatImportances(dfx, target='Attrition', classifier='Light GBM', plotsize=(8, 10))`

```
Set the 'classifier='Random Forest, Gradient Boosting, Ada Boosting, Decision Tree for feature import
ances
--- 0.18 seconds ---
```

Out[37]:

```
MonthlyIncome               380
Age                         248
DistanceFromHome            208
JobRole                     157
PercentSalaryHike           154
JobSatisfaction             144
YearsWithCurrManager        121
NumCompaniesWorked          116
EnvironmentSatisfaction     115
YearsSinceLastPromotion     112
YearsAtCompany              112
TotalWorkingYears           107
RelationshipSatisfaction    103
TrainingTimesLastYear       101
StockOptionLevel             97
JobInvolvement               96
EducationField               83
WorkLifeBalance              82
OverTime                     81
YearsInCurrentRole           78
BusinessTravel               76
MaritalStatus                66
Education                    57
Gender                       50
Department                   34
JobLevel                     22
PerformanceRating             0
dtype: int32
```

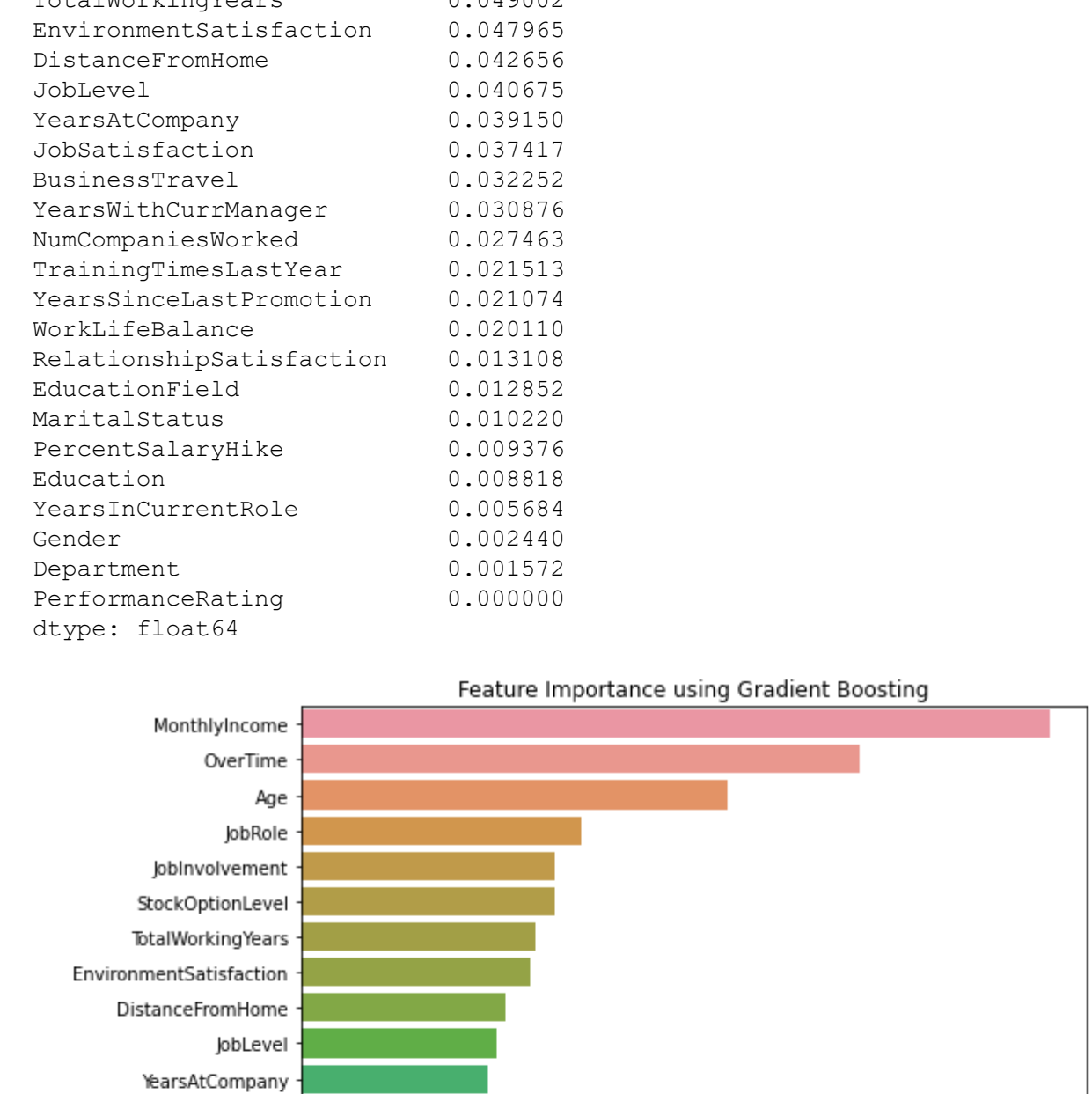


In [38]: `fxFeatImportances(dfx, target='Attrition', classifier='Gradient Boosting', plotsize=(8, 10))`

```
Set the 'classifier='Random Forest, Ada Boosting, Decision Tree, Light GBM for feature importances
--- 0.39 seconds ---
```

Out[38]:

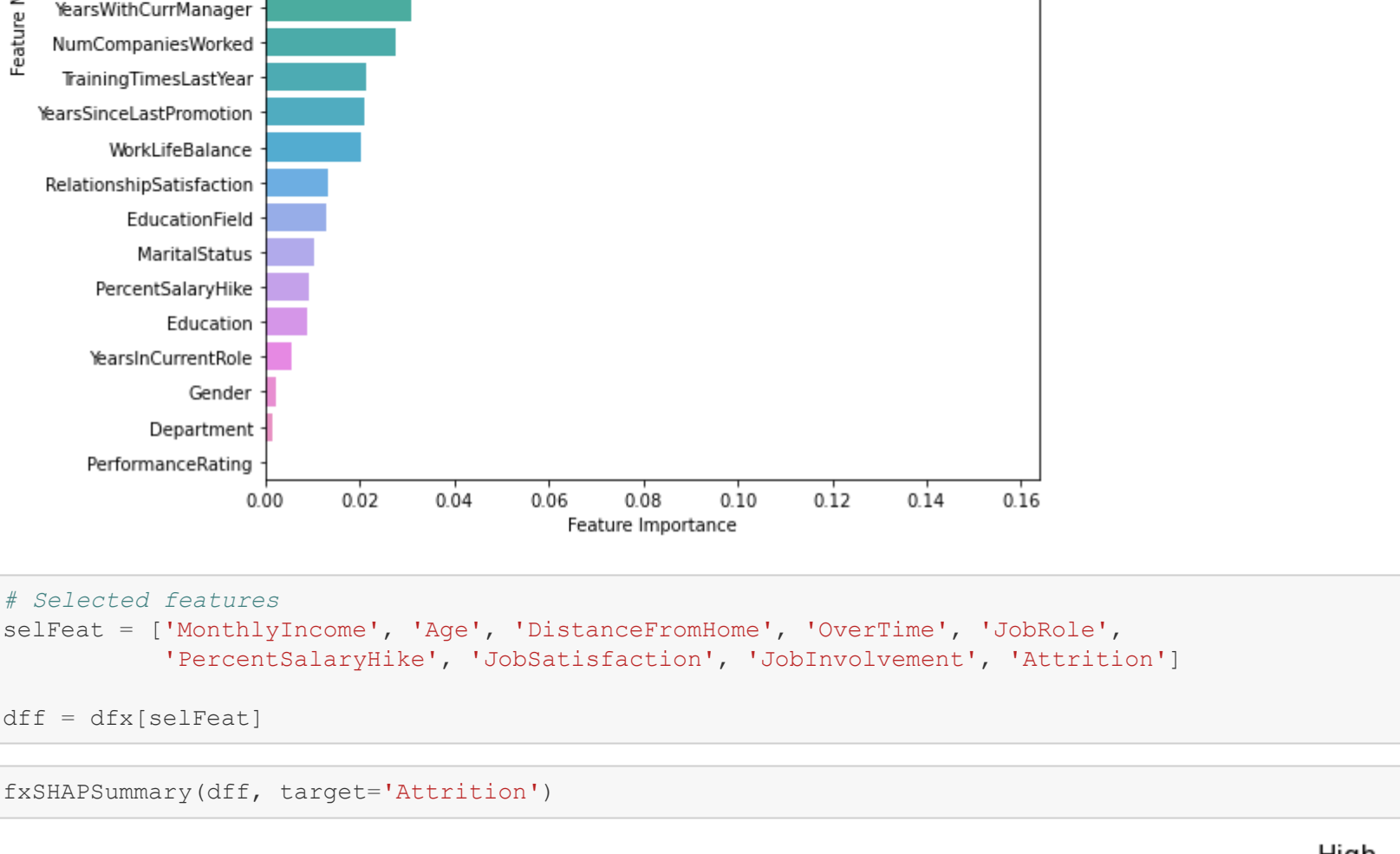
```
MonthlyIncome               0.156134
OverTime                    0.116416
Age                         0.089048
JobRole                     0.058440
JobInvolvement              0.052961
StockOptionLevel            0.052779
TotalWorkingYears           0.049002
EnvironmentSatisfaction     0.047965
DistanceFromHome            0.042656
JobLevel                    0.040675
YearsAtCompany              0.039150
JobSatisfaction             0.037417
BusinessTravel              0.032252
YearsWithCurrManager        0.030876
NumCompaniesWorked          0.027463
TrainingTimesLastYear       0.021513
YearsSinceLastPromotion     0.021074
WorkLifeBalance             0.020110
RelationshipSatisfaction    0.013108
EducationField              0.012852
MaritalStatus               0.010220
PercentSalaryHike           0.009376
Education                   0.008818
YearsInCurrentRole          0.005684
Gender                      0.002440
Department                  0.001572
PerformanceRating           0.000000
dtype: float64
```



In [60]:

```python
# Selected features
selFeat = ['MonthlyIncome', 'Age', 'DistanceFromHome', 'OverTime', 'JobRole',
           'PercentSalaryHike', 'JobSatisfaction', 'JobInvolvement', 'Attrition']

dff = dfx[selFeat]
```

In [61]: `fxSHAPSummary(dff, target='Attrition')`




```
--- 0.55 seconds ---
```

Split Train & Test sets

In [77]:

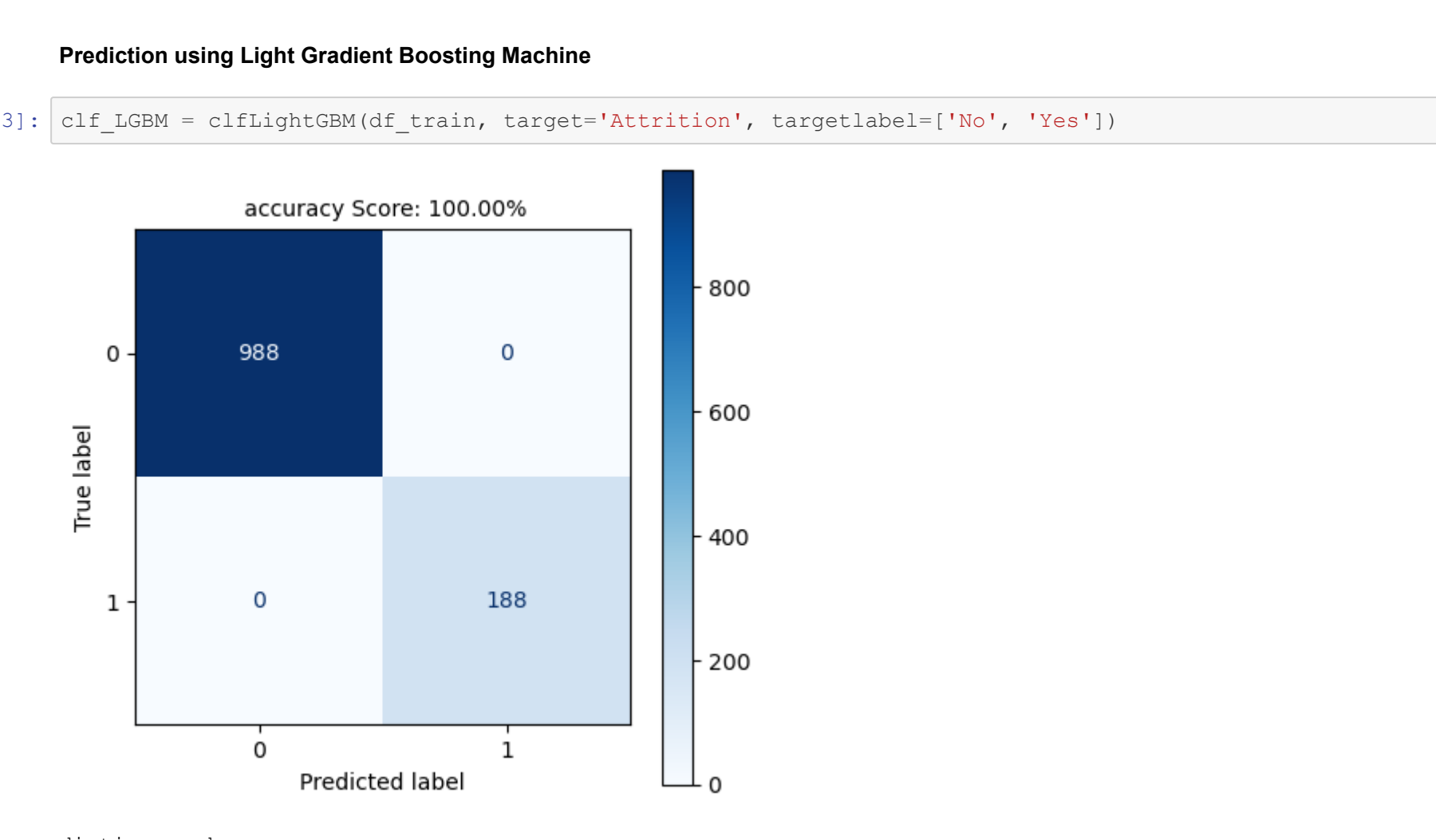
```python
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(dff.drop('Attrition', axis=1), dff.Attrition, test_
size=0.2, random_state=0)

df_train = x_train.copy()
df_train['Attrition'] = y_train

df_test = x_test.copy()
df_test['Attrition'] = y_test
```

Prediction using Light Gradient Boosting Machine

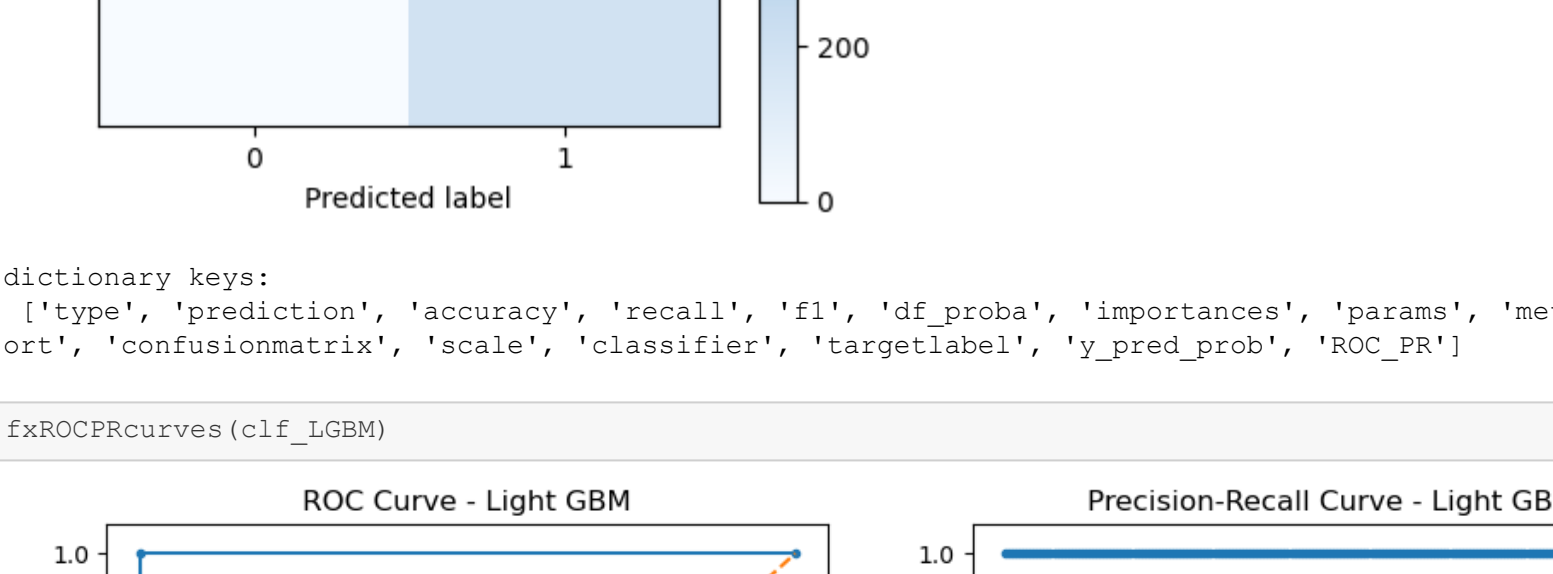
In [63]: `clf_LGBM = clfLightGBM(df_train, target='Attrition', targetlabel=['No', 'Yes'])`


```
dictionary keys:
 ['type', 'prediction', 'accuracy', 'recall', 'f1', 'df_proba', 'importances', 'params', 'metrics_rep
ort', 'confusionmatrix', 'scale', 'classifier', 'targetlabel', 'y_pred_prob', 'ROC_PR']
```

In [64]: `fxROCPRcurves(clf_LGBM)`

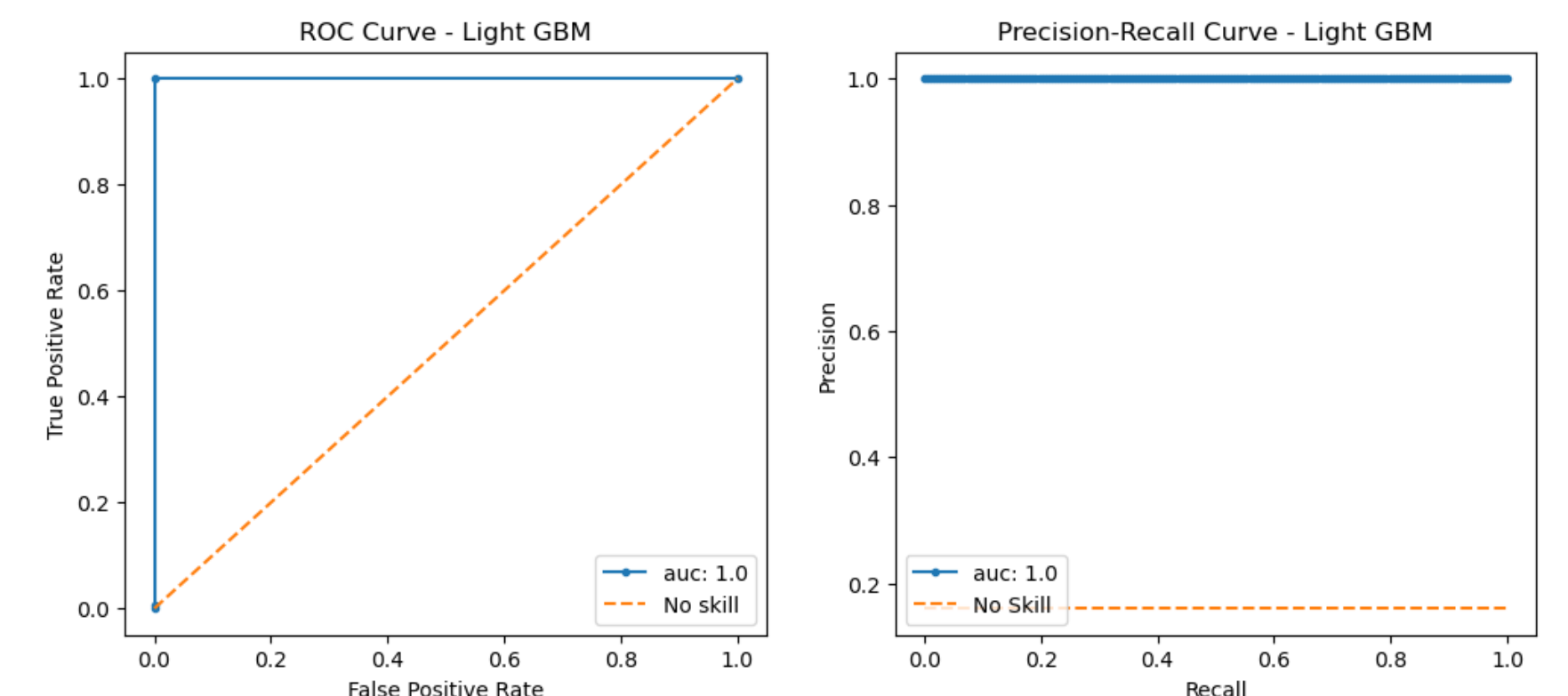


In [69]: `KfoldCV(df_train, 'Attrition', clf_LGBM['classifier'])`



In [71]: `fxLearningCurve(df_train, 'Attrition', clf_LGBM['classifier'])`

Out[71]: `<module 'matplotlib.pyplot' from 'c:\\program files\\python38\\lib\\site-packages\\matplotlib\\pyplo
t.py'>`

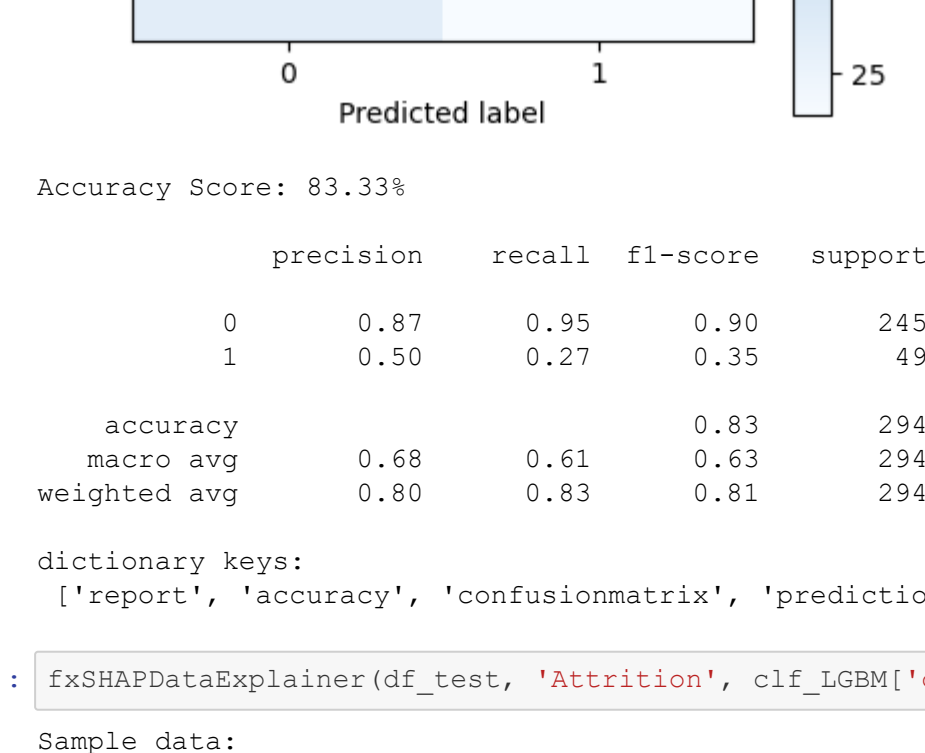


In [51]: `#fxSHAPcv(dfx, 'Attrition', clf_LGBM['classifier'])`

Evalution using test data

In [111]:

```python
perf_LGBM = clfEvaluator(df_test, 'Attrition', clf_LGBM['classifier'], targetlabel=['No', 'Yes'],
                         supstr='LGBM', explainer=True)
```




```
Accuracy Score: 83.33%

              precision    recall  f1-score   support

           0       0.87      0.95      0.90       245
           1       0.50      0.27      0.35        49

    accuracy                           0.83       294
   macro avg       0.68      0.61      0.63       294
weighted avg       0.80      0.83      0.81       294

dictionary keys:
 ['report', 'accuracy', 'confusionmatrix', 'prediction']
```

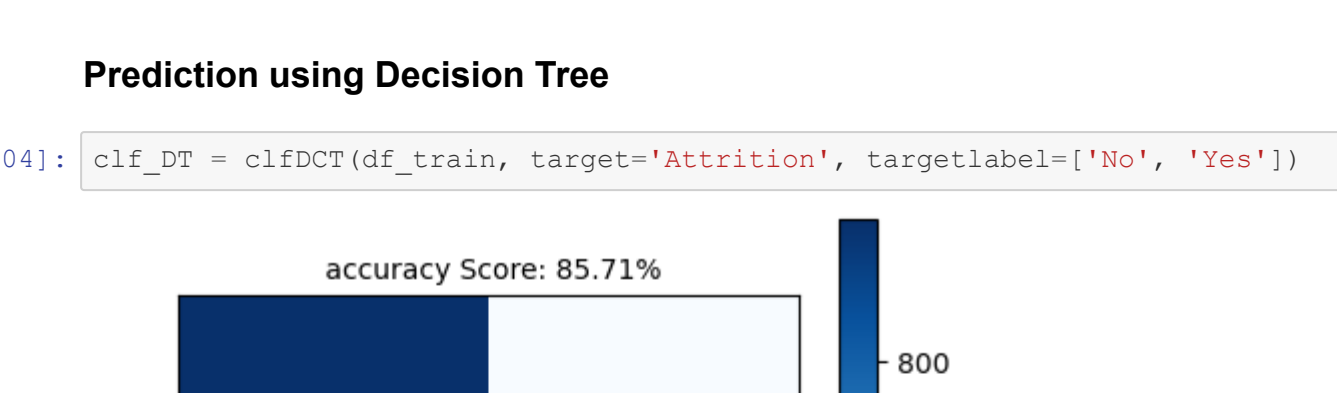
In [103]: `fxSHAPDataExplainer(df_test, 'Attrition', clf_LGBM['classifier'], dataidx=3, select='random5')`

Sample data:

	MonthlyIncome	Age	DistanceFromHome	OverTime	JobRole
875	4541	44	29	0	6
722	2684	38	10	0	6
254	6931	29	20	0	7
572	4335	29	27	0	0
1412	3748	30	1	0	2

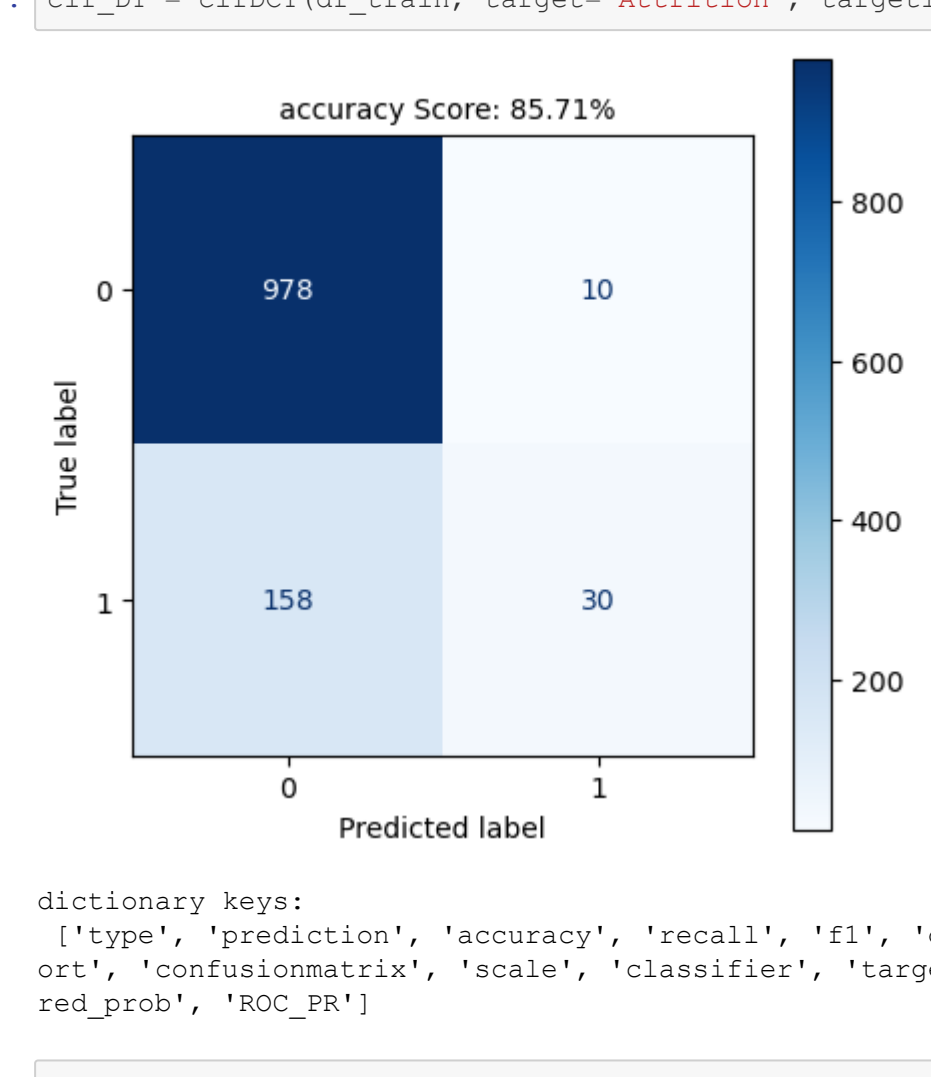
	PercentSalaryHike	JobSatisfaction	JobInvolvement	Attrition	predict
875	25	4	3	0	0
722	17	3	3	0	0
254	14	4	3	0	0
572	12	3	3	0	0
1412	13	2	3	0	0

Out[103]:



Prediction using Decision Tree

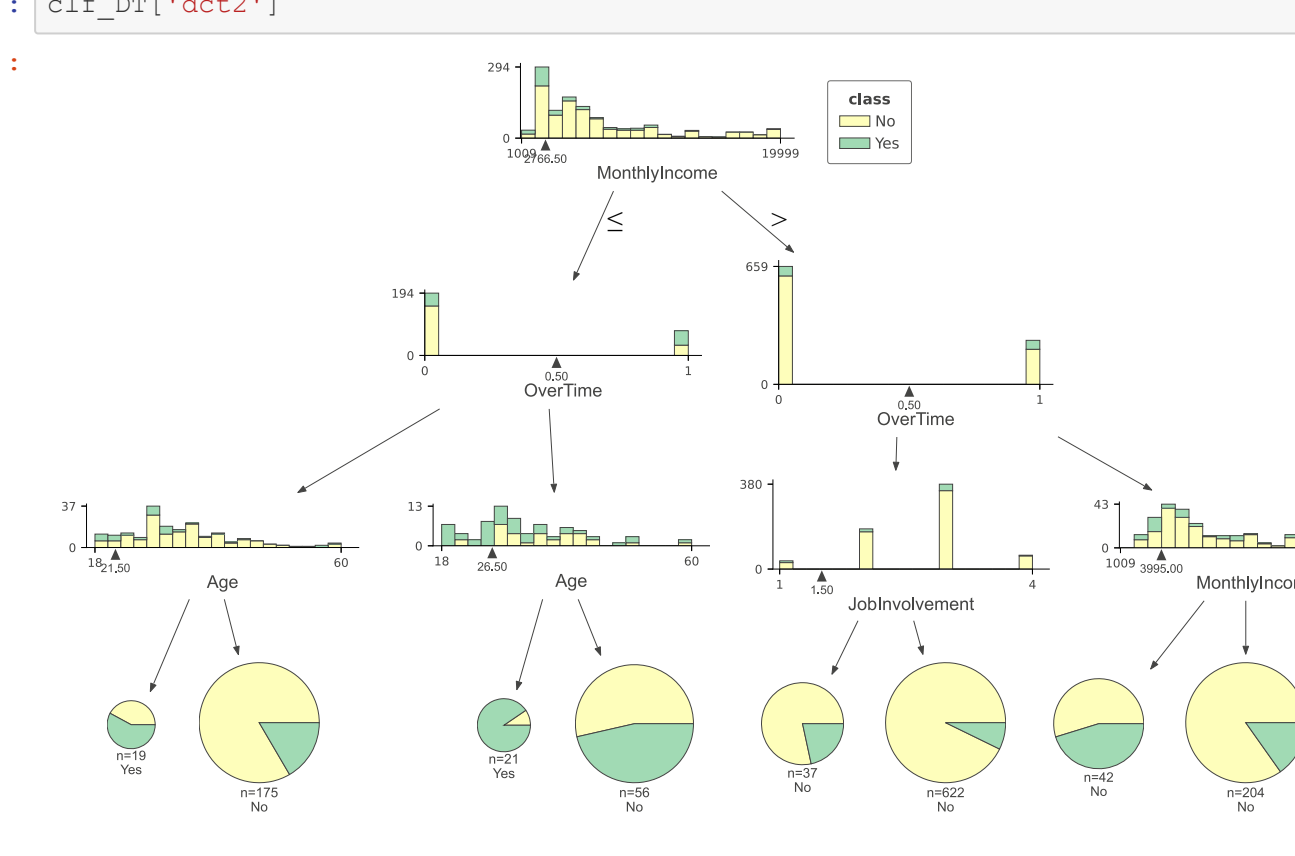
In [104]: `clf_DT = clfDCT(df_train, target='Attrition', targetlabel=['No', 'Yes'])`



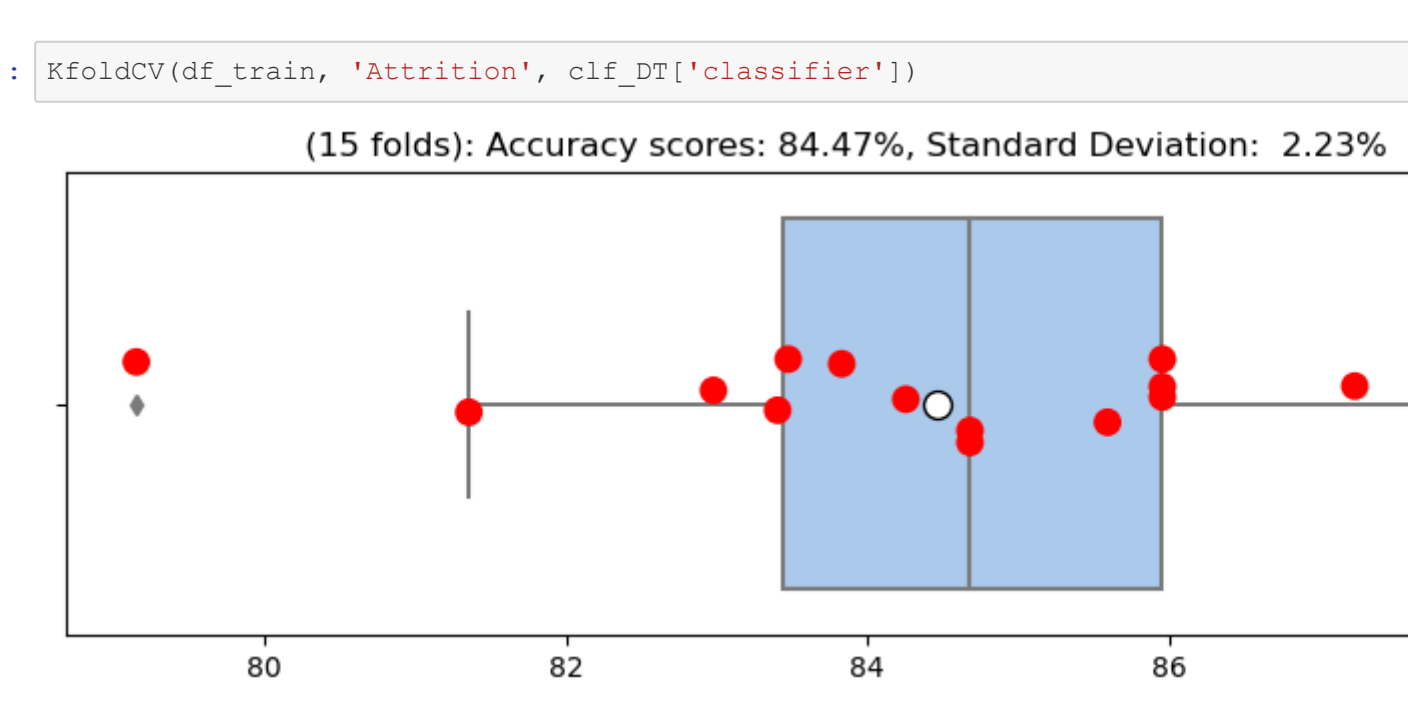

```
dictionary keys:
 ['type', 'prediction', 'accuracy', 'recall', 'f1', 'df_proba', 'params', 'importances', 'metrics_rep
ort', 'confusionmatrix', 'scale', 'classifier', 'targetlabel', 'leaves', 'dct0', 'dct1', 'dct2', 'y_p
red_prob', 'ROC_PR']
```

In [112]: `clf_DT['dct2']`

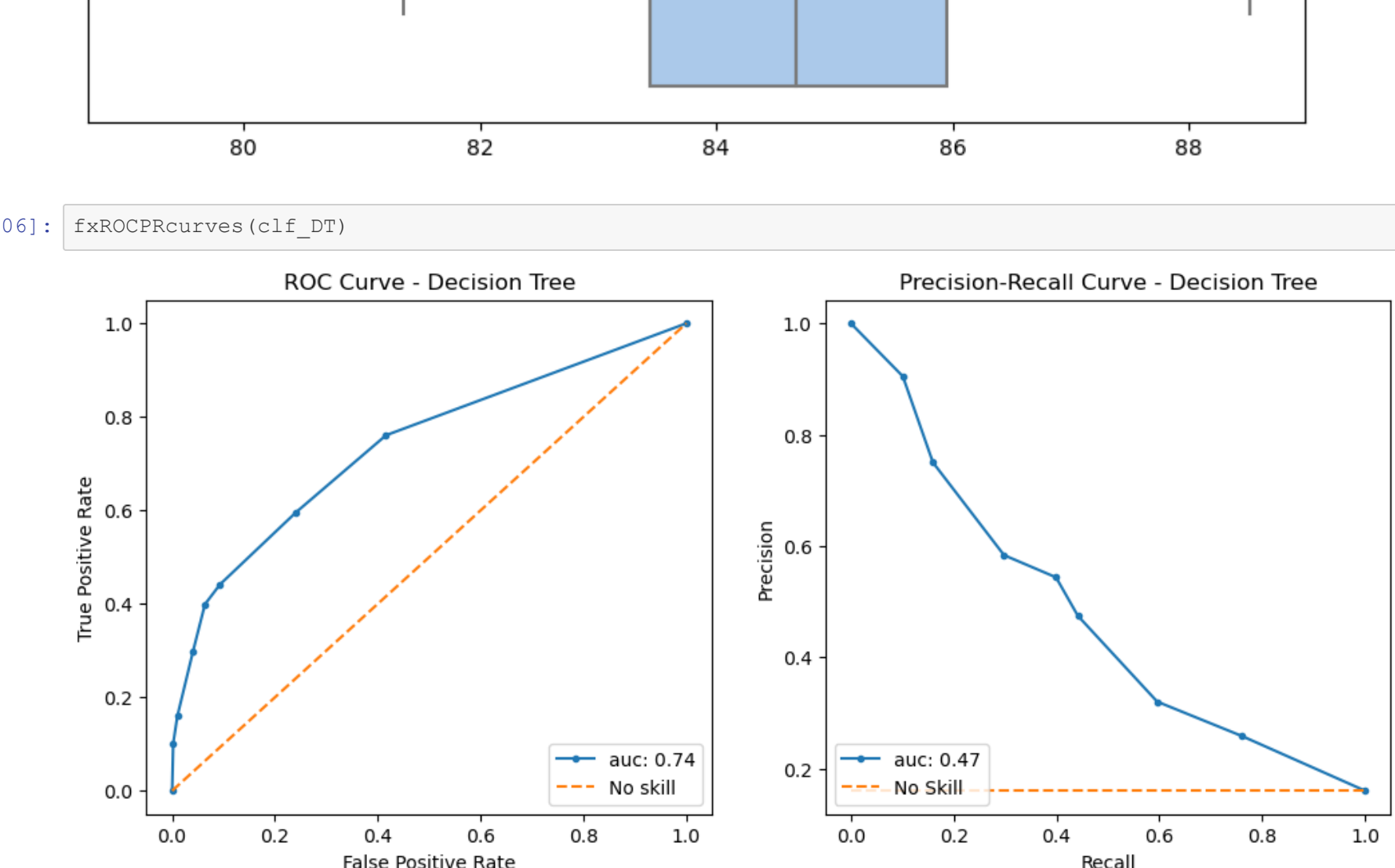
Out[112]:



In [105]: `KfoldCV(df_train, 'Attrition', clf_DT['classifier'])`



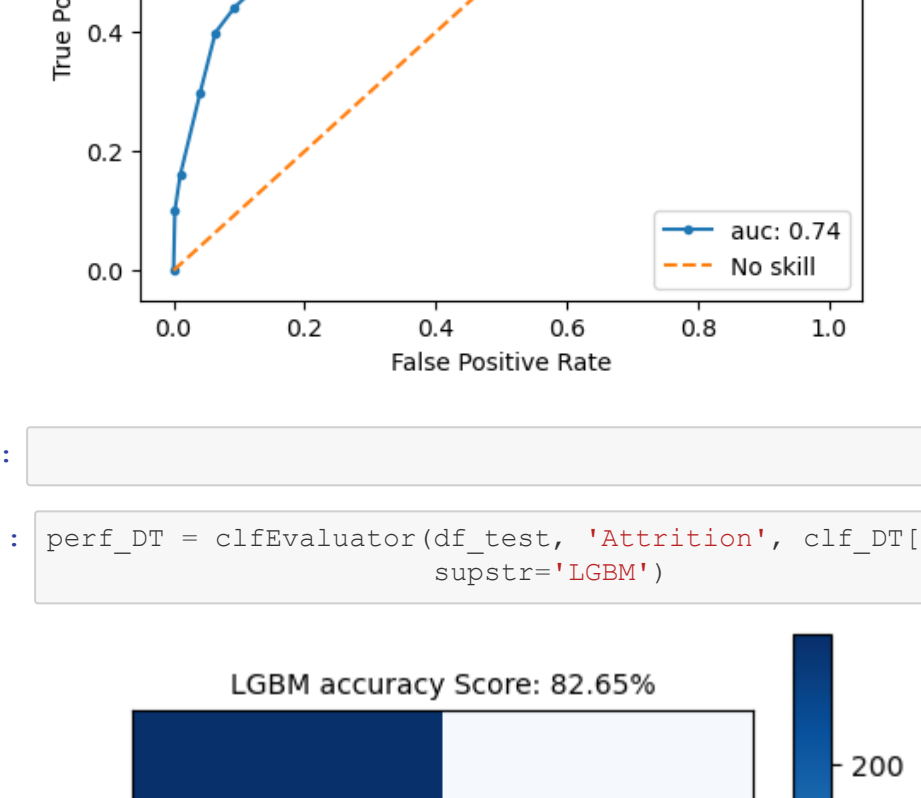
In [106]: `fxROCPRcurves(clf_DT)`



In [ ]:

In [108]:

```python
perf_DT = clfEvaluator(df_test, 'Attrition', clf_DT['classifier'], targetlabel=['No', 'Yes'],
                       supstr='LGBM')
```




```
Accuracy Score: 82.65%

              precision    recall  f1-score   support

           0       0.84      0.98      0.90       245
           1       0.38      0.06      0.11        49

    accuracy                           0.83       294
   macro avg       0.61      0.52      0.50       294
weighted avg       0.76      0.83      0.77       294

dictionary keys:
 ['report', 'accuracy', 'confusionmatrix', 'prediction']
```

In [110]: `perf_DT['prediction'].head()`

Out[110]:

	MonthlyIncome	Age	DistanceFromHome	OverTime	JobRole	PercentSalaryHike	JobSatisfaction	JobInvolvement	truth	prediction
442	9980	36	10	0	7	14	4	3	No	No
1091	4320	33	25	0	4	13	2	2	No	No
981	4614	35	18	1	7	18	3	3	Yes	No
785	10322	40	20	0	0	20	4	3	No	No
1332	2439	29	24	1	6	24	4	2	Yes	No

In [ ]: