In [16]:
```
import numpy as np
import pandas as pd
```

In [17]:
```
#Interactive plots using plotly
import plotly.graph_objects as go
import plotly.offline as plt
import plotly.figure_factory as ff
```

In [18]:
```
#For Model training
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import NearMiss
```

In [19]:
```
#Prepare data
source = pd.read_csv(r'C:\Users\zishe\.spyder-py3\repos\CIS 568\data.csv')
```

In [20]:
```
#Check for null values
source.isnull().any()
```

Out[20]:
```
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 [21]:
```
#What kind of dtypes are there
source.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
Age                         1470 non-null int64
Attrition                   1470 non-null object
BusinessTravel              1470 non-null object
DailyRate                   1470 non-null int64
Department                  1470 non-null object
DistanceFromHome            1470 non-null int64
Education                   1470 non-null int64
EducationField              1470 non-null object
EmployeeCount               1470 non-null int64
EmployeeNumber              1470 non-null int64
EnvironmentSatisfaction     1470 non-null int64
Gender                      1470 non-null object
HourlyRate                  1470 non-null int64
JobInvolvement              1470 non-null int64
JobLevel                    1470 non-null int64
JobRole                     1470 non-null object
JobSatisfaction             1470 non-null int64
MaritalStatus               1470 non-null object
MonthlyIncome               1470 non-null int64
MonthlyRate                 1470 non-null int64
NumCompaniesWorked          1470 non-null int64
Over18                      1470 non-null object
OverTime                    1470 non-null object
PercentSalaryHike           1470 non-null int64
PerformanceRating           1470 non-null int64
RelationshipSatisfaction    1470 non-null int64
StandardHours               1470 non-null int64
StockOptionLevel            1470 non-null int64
TotalWorkingYears           1470 non-null int64
TrainingTimesLastYear       1470 non-null int64
WorkLifeBalance             1470 non-null int64
YearsAtCompany              1470 non-null int64
YearsInCurrentRole          1470 non-null int64
YearsSinceLastPromotion     1470 non-null int64
YearsWithCurrManager        1470 non-null int64
dtypes: int64(26), object(9)
memory usage: 402.0+ KB
```

In [22]:
```
#Check the number of unique values in the data
source.nunique()
```

Out[22]:
```
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
```

In [23]:
```
#Dropping unnecessary data
features = source.drop(columns=['Attrition', 'EmployeeCount', 'Over18', 'StandardHours', 'EmployeeNumber'])
```

In [24]:
```
#Label OHE
label = source['Attrition']
label = label.apply(lambda x:{"Yes": 1, "No": 0}[x])
label = label.to_frame()
```

In [25]:
```
#Split numerical and categorical features
numFeatures = features.select_dtypes(include=['int64'])
catFeatures = features.select_dtypes(include=['object'])
#OHE categorical features
catFeatures = pd.get_dummies(catFeatures)
```

In [26]:
```
data = pd.concat([label, numFeatures, catFeatures], axis=1)
```

In [27]:
```
data.head()
```

Out[27]:

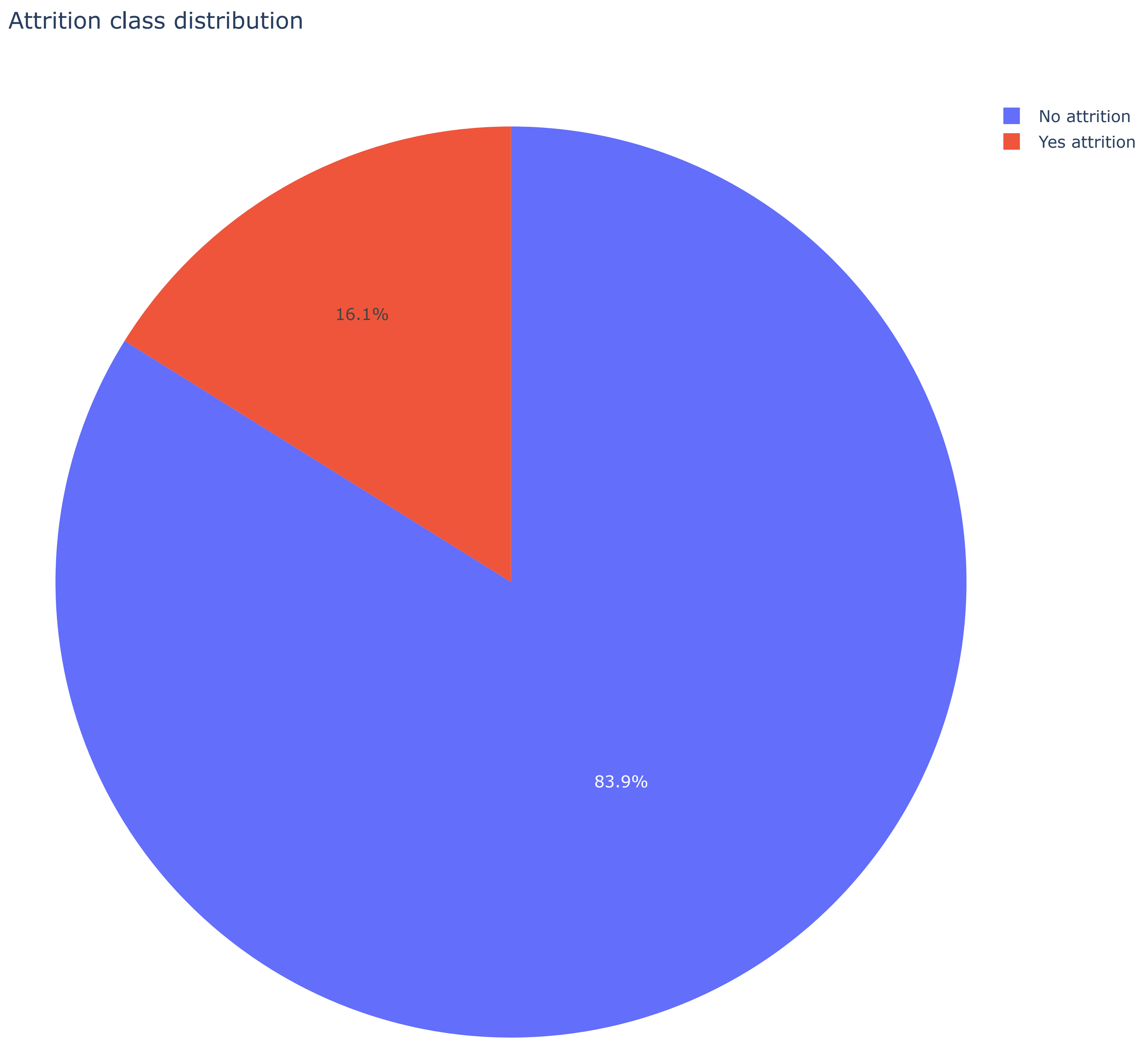
	Attrition	Age	DailyRate	DistanceFromHome	Education	EnvironmentSatisfaction	HourlyRate	JobInvolvement	JobLevel	JobSatisfaction	...	JobRole_Man
0	1	41	1102	1	2	2	94	3	2	4	...	
1	0	49	279	8	1	3	61	2	2	2	...	
2	1	37	1373	2	2	4	92	2	1	3	...	
3	0	33	1392	3	4	4	56	3	1	3	...	
4	0	27	591	2	1	1	40	3	1	2	...	

5 rows × 52 columns

In [28]:
```
#Splitting each attrition class for more details
nAttrition = data[(data["Attrition"] == 0)]
yAttrition = data[(data["Attrition"] == 1)]
```

In [29]:

```
#Look at the class distribution of attrition
trace = [go.Pie(
        values = [len(nAttrition), len(yAttrition)],
        labels=["No attrition", "Yes attrition"]
        )]
layout = go.Layout(
        width = 900,
        height = 900
        )
fig = go.Figure(data = trace, layout = layout)
fig.update_layout(title_text = "Attrition class distribution")
fig.show()
#plt.plot(fig)
```



There's a large class imbalance so if an accurate model is to be trained from this dataset, resampling will be required

In [30]:
```
z = data.describe()
display(z)
```

	Attrition	Age	DailyRate	DistanceFromHome	Education	EnvironmentSatisfaction	HourlyRate	JobInvolvement	JobLevel	JobSatisf
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.000000	1470.000000	1470.000000	1470.000000	1470.000000	1470.0
mean	0.161224	36.923810	802.485714	9.192517	2.912925	2.721769	65.891156	2.729932	2.063946	2.7
std	0.367863	9.135373	403.509100	8.106864	1.024165	1.093082	20.329428	0.711561	1.106940	1.1
min	0.000000	18.000000	102.000000	1.000000	1.000000	1.000000	30.000000	1.000000	1.000000	1.0
25%	0.000000	30.000000	465.000000	2.000000	2.000000	2.000000	48.000000	2.000000	1.000000	2.0
50%	0.000000	36.000000	802.000000	7.000000	3.000000	3.000000	66.000000	3.000000	2.000000	3.0
75%	0.000000	43.000000	1157.000000	14.000000	4.000000	4.000000	83.750000	3.000000	3.000000	4.0
max	1.000000	60.000000	1499.000000	29.000000	5.000000	4.000000	100.000000	4.000000	5.000000	4.0

8 rows × 52 columns

In [31]:
```
yInfo = yAttrition.describe()
nInfo = nAttrition.describe()
yInfo = yInfo.drop(yInfo.index[0])
nInfo = nInfo.drop(nInfo.index[0])
```

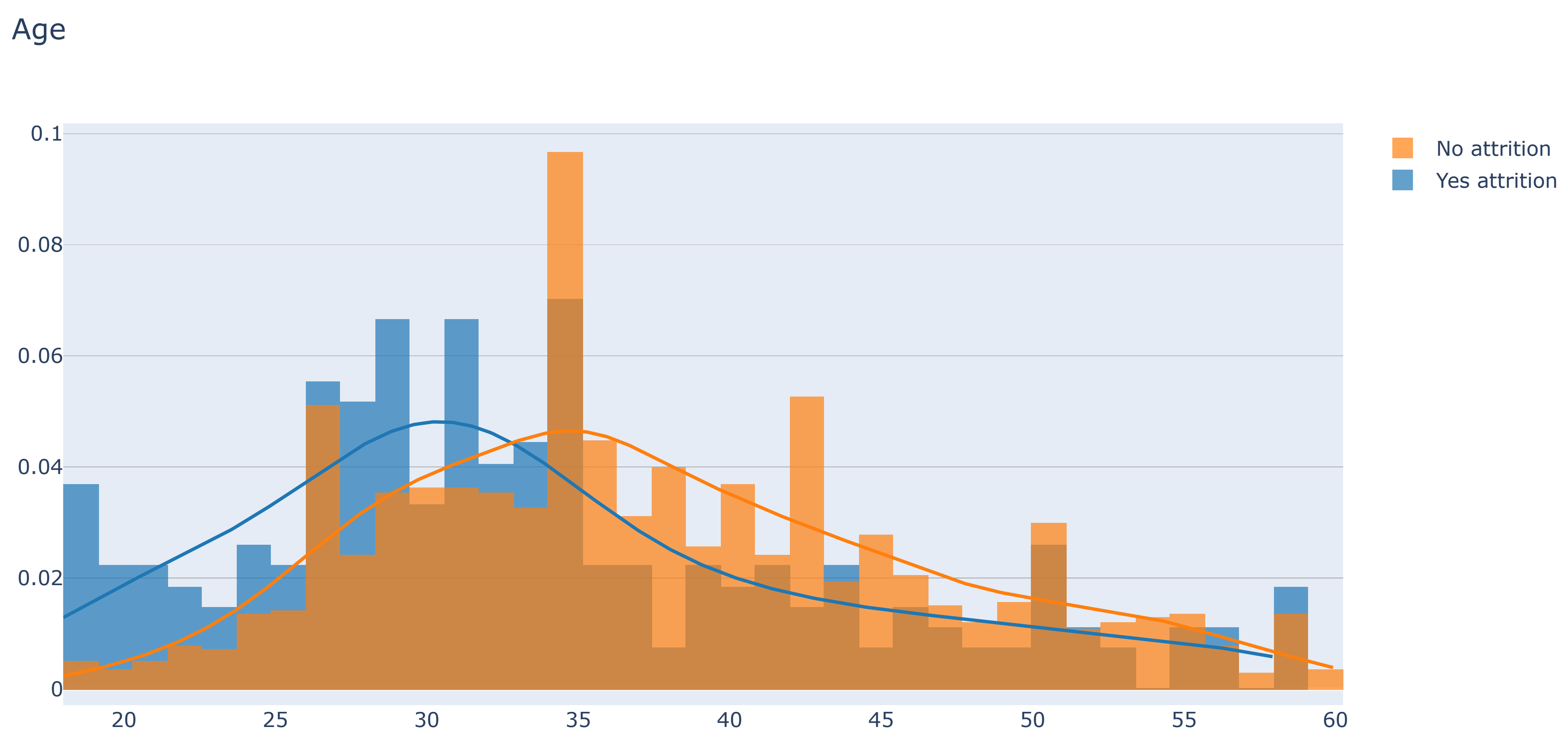
In [32]:
```
def distPlot(var):
#Distribution plot
    dat1 = yAttrition[var]
    dat2 = nAttrition[var]
    histData = [dat1,dat2]
    groupLabel = ["Yes attrition", "No attrition"]

    autoBin = (z.loc["std", var])/8

    fig = ff.create_distplot(
            hist_data = histData,
            group_labels = groupLabel,
            show_rug=False,
            bin_size = autoBin
            )
    fig.update_layout(title_text = var)
    #plt.plot(fig)
    fig.show()
```

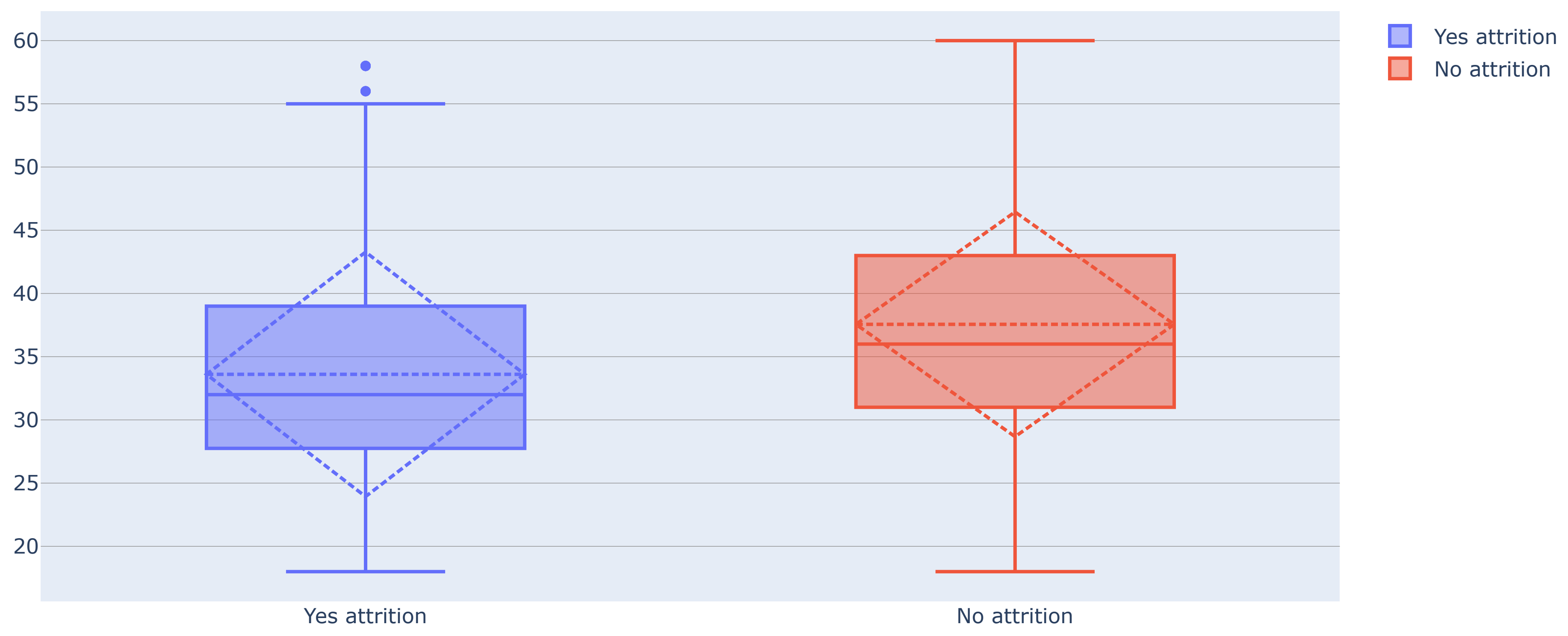
In [33]:
```
def box(var):
    fig = go.Figure()
    yBox = yAttrition[var]
    nBox = nAttrition[var]
    fig.add_trace(go.Box(y=yBox, boxmean="sd", name="Yes attrition"))
    fig.add_trace(go.Box(y=nBox, boxmean="sd", name="No attrition"))
    #plt.plot(fig)
    fig.show()
```

In [34]:
```
def comp(var):
    yMean = yInfo.loc["mean",var]
    nMean = nInfo.loc["mean",var]
    print("Yes-Attrition mean: \t" + str(yMean) + " ± " + str(yInfo.loc["std",var]))
    print("No-Attrition mean: \t" + str(nMean) + " ± " + str(nInfo.loc["std",var]))
    print("Absolute mean difference: \t"+ str(abs(yMean-nMean)))
```

In [35]:
```
def vis(var):
#Visualize distribution plot, box plot, and mean±std
    distPlot(var)
    box(var)
    comp(var)
```

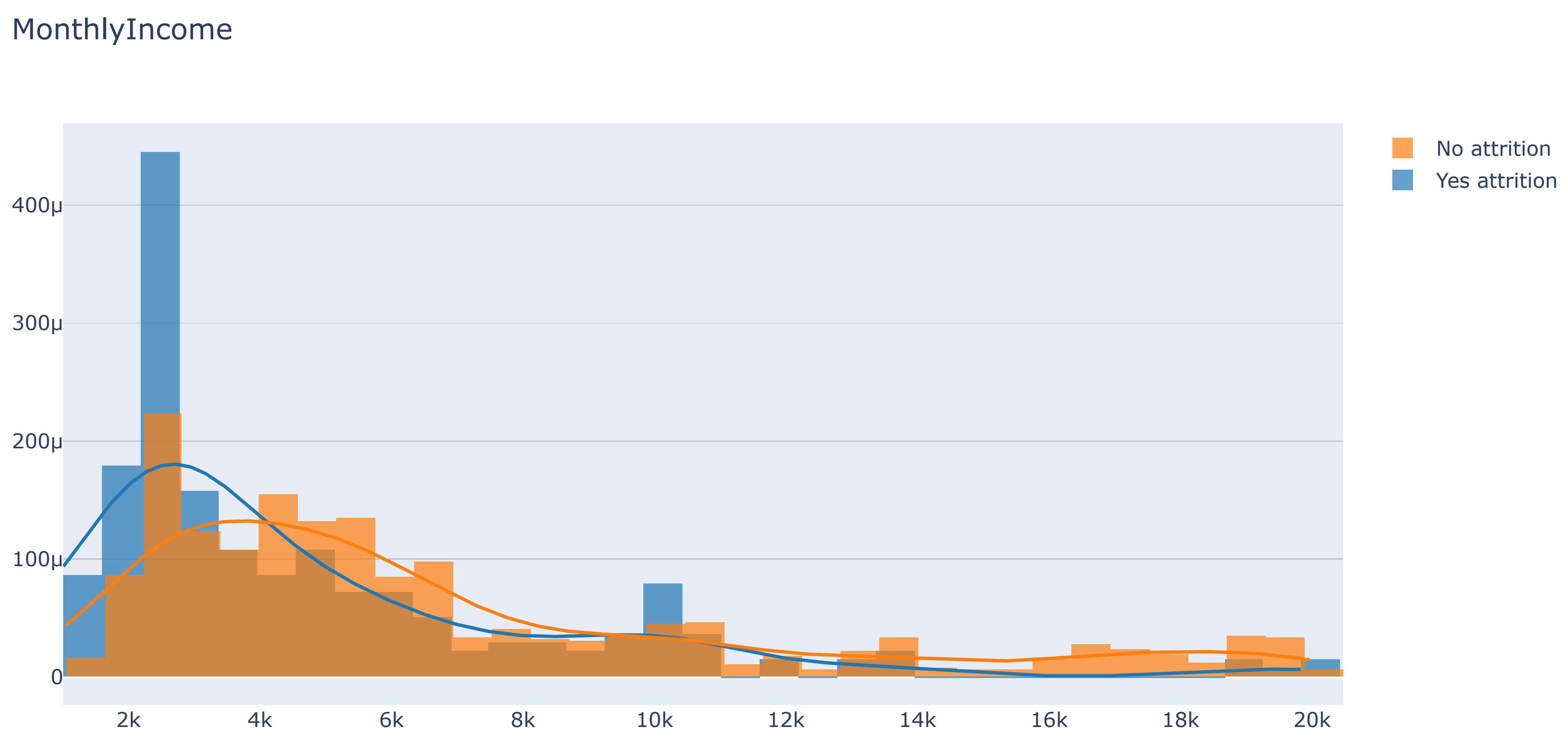
```
In [36]: vis("Age")
```

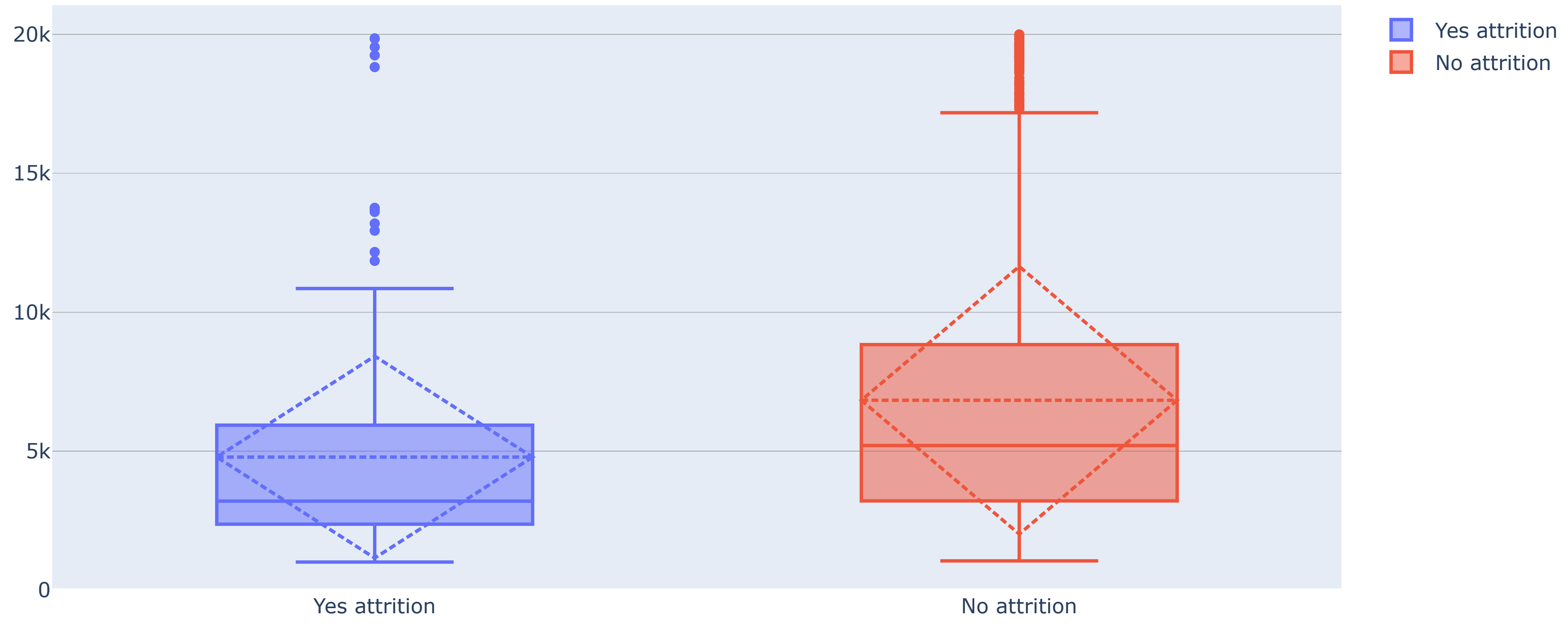





```
Yes-Attrition mean:     33.607594936708864 ± 9.689349895351622
No-Attrition mean:      37.561232765612324 ± 8.888360024976546
Absolute mean difference:       3.95363782890346
```

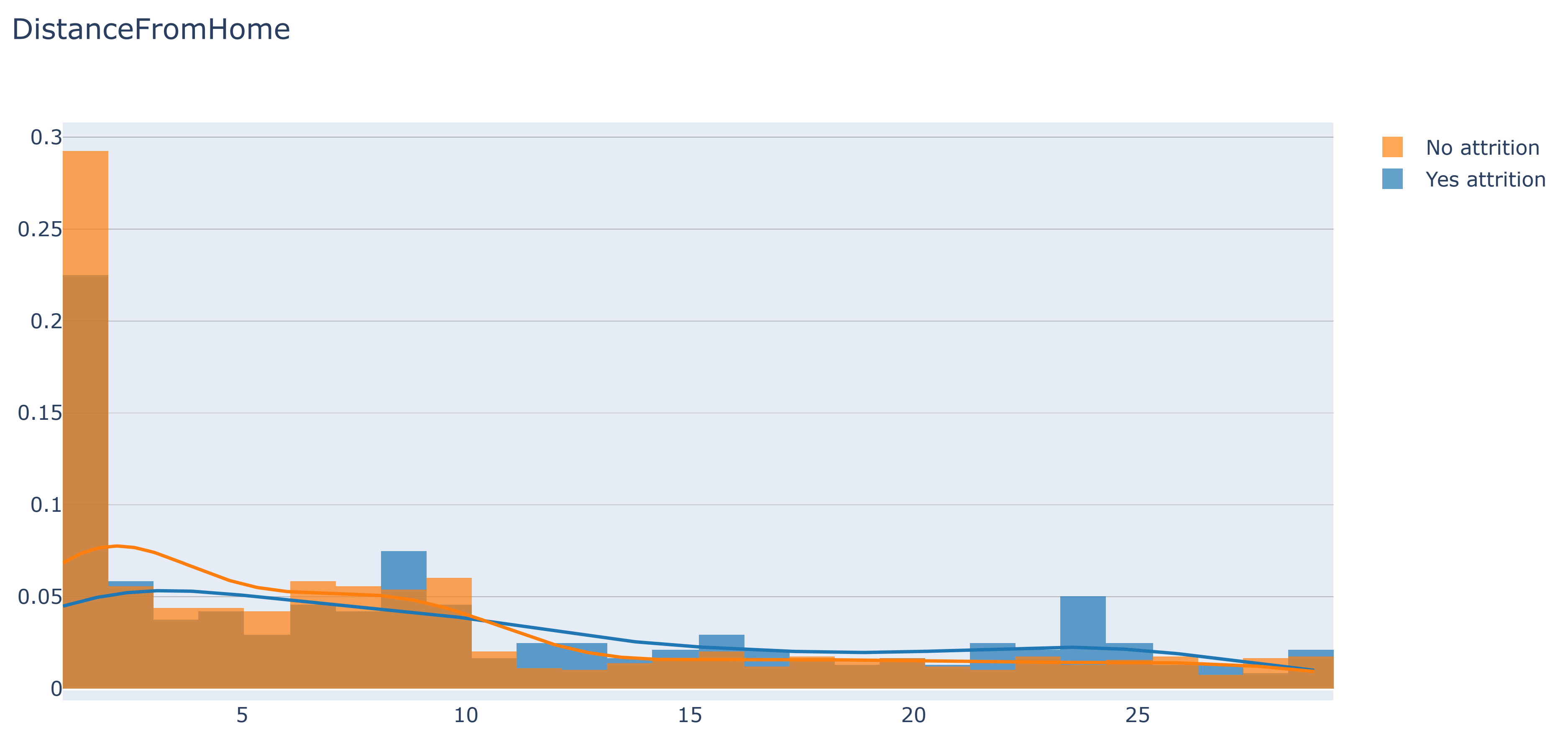
```
In [37]: vis("MonthlyIncome")
```

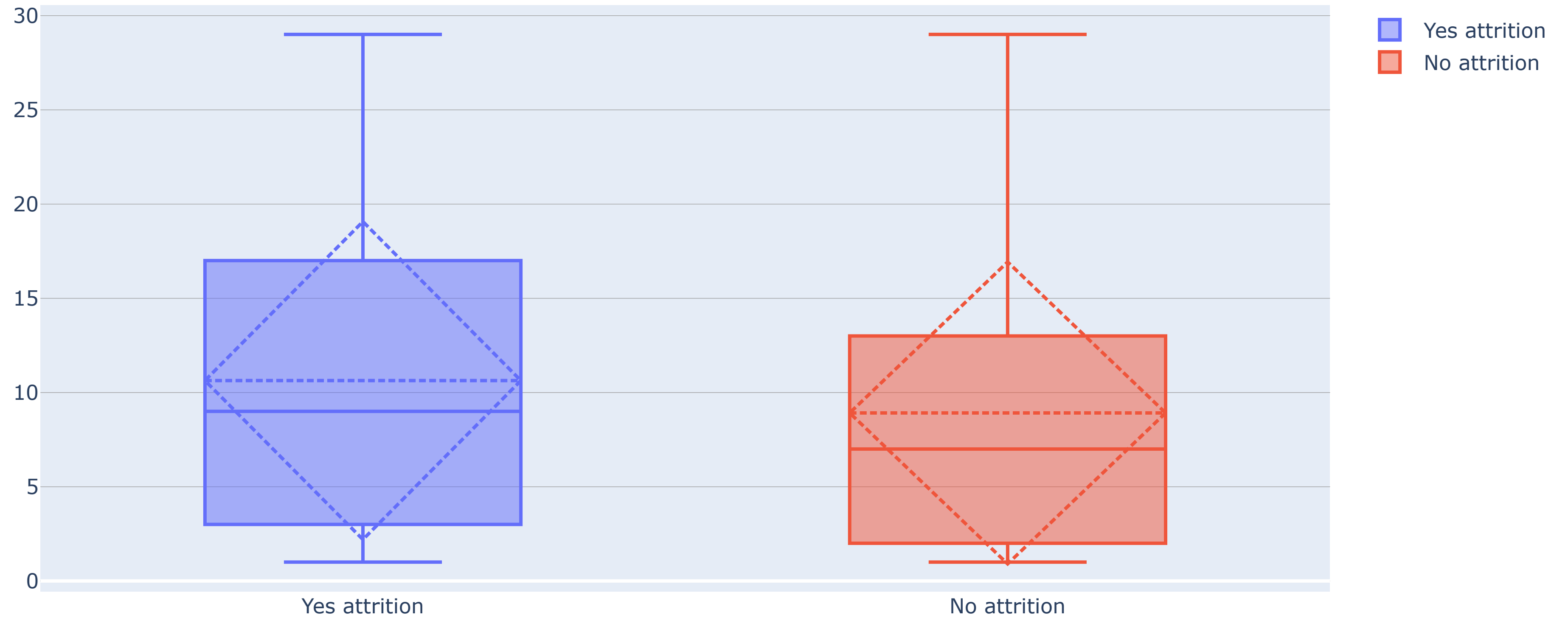





```
Yes-Attrition mean:     4787.0928270042195 ± 3640.2103671038512
No-Attrition mean:      6832.739659367397 ± 4818.208000784485
Absolute mean difference:       2045.646832363177
```

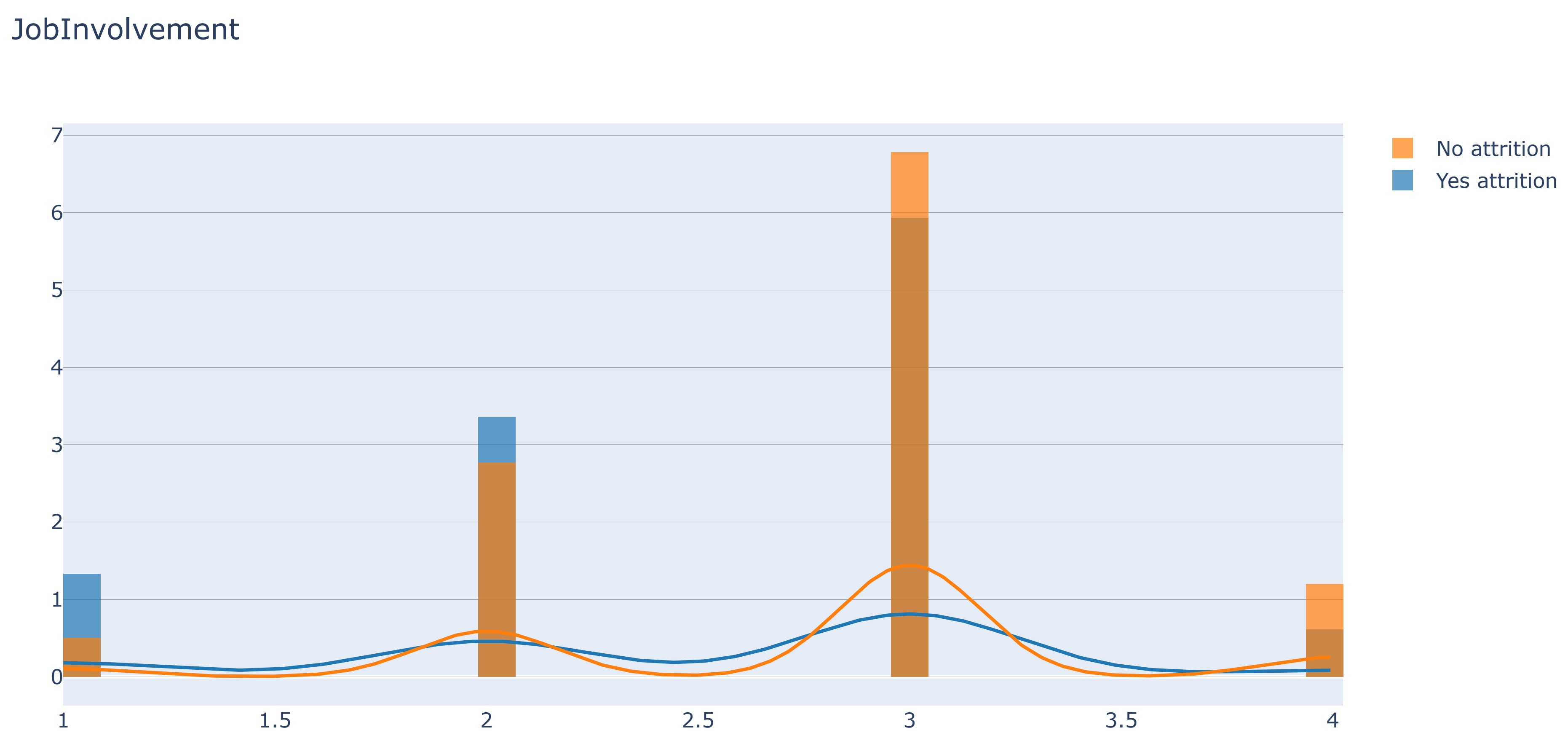
In [38]:
```
vis("DistanceFromHome")
```

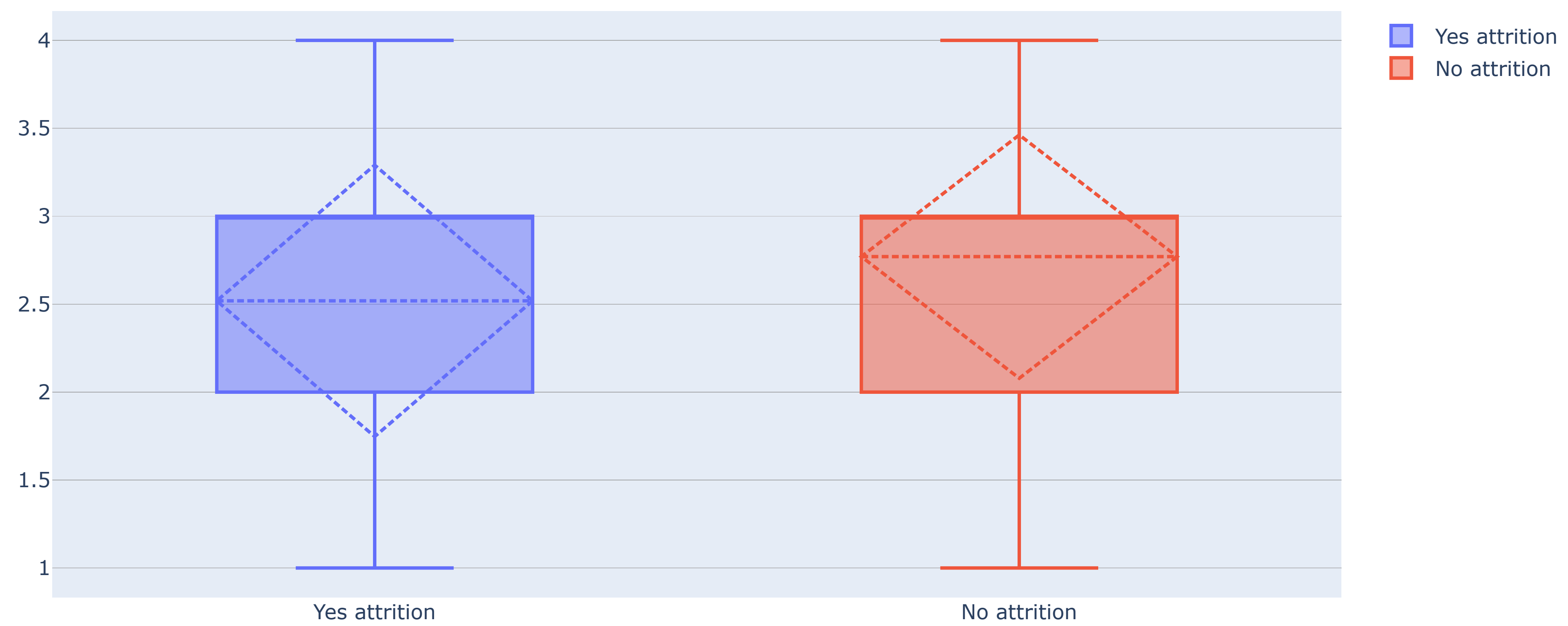





```
Yes-Attrition mean:     10.632911392405063 ± 8.452525269825024
No-Attrition mean:      8.915652879156529 ± 8.0126334854975
Absolute mean difference:       1.7172585132485345
```

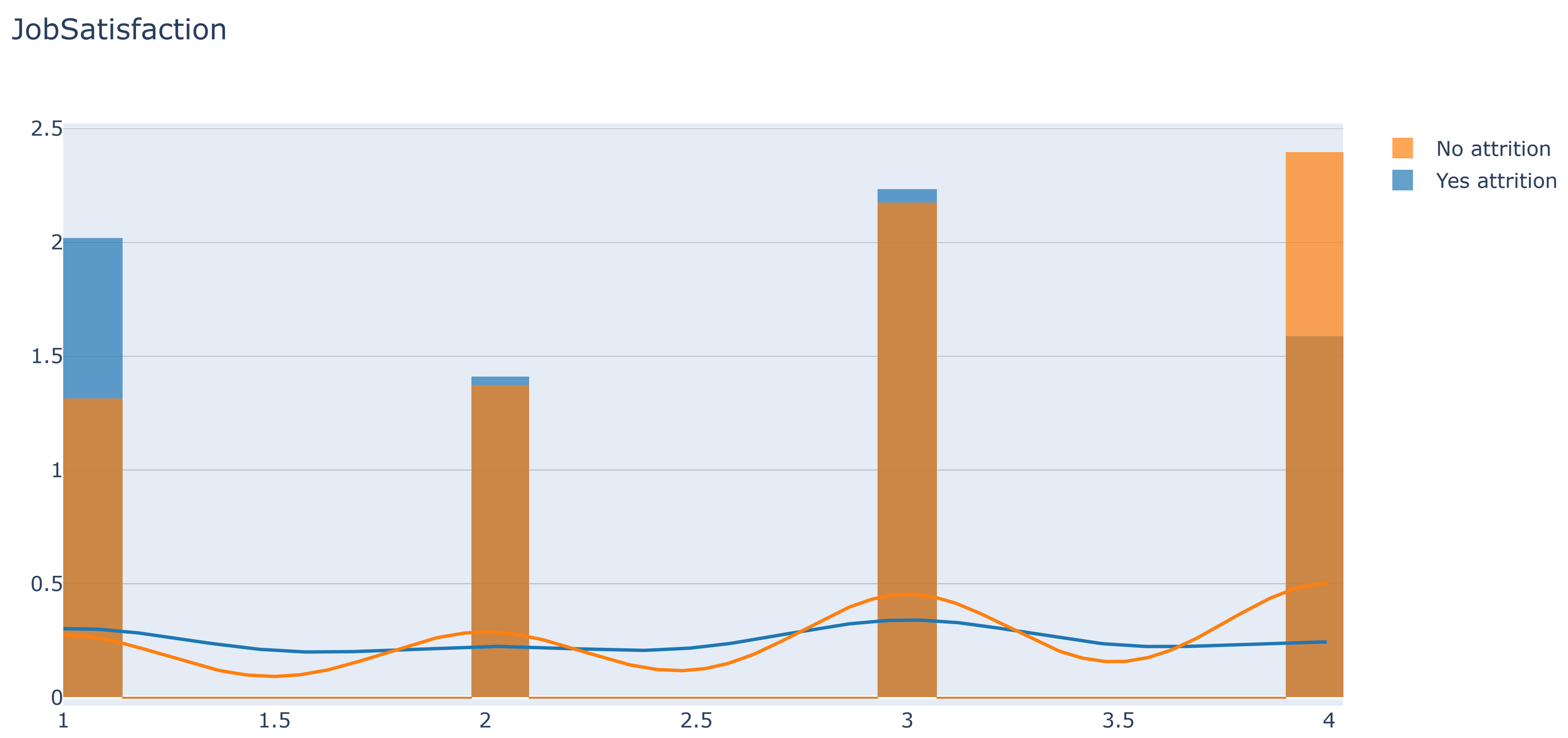
In [39]:
```
vis("JobInvolvement")
```

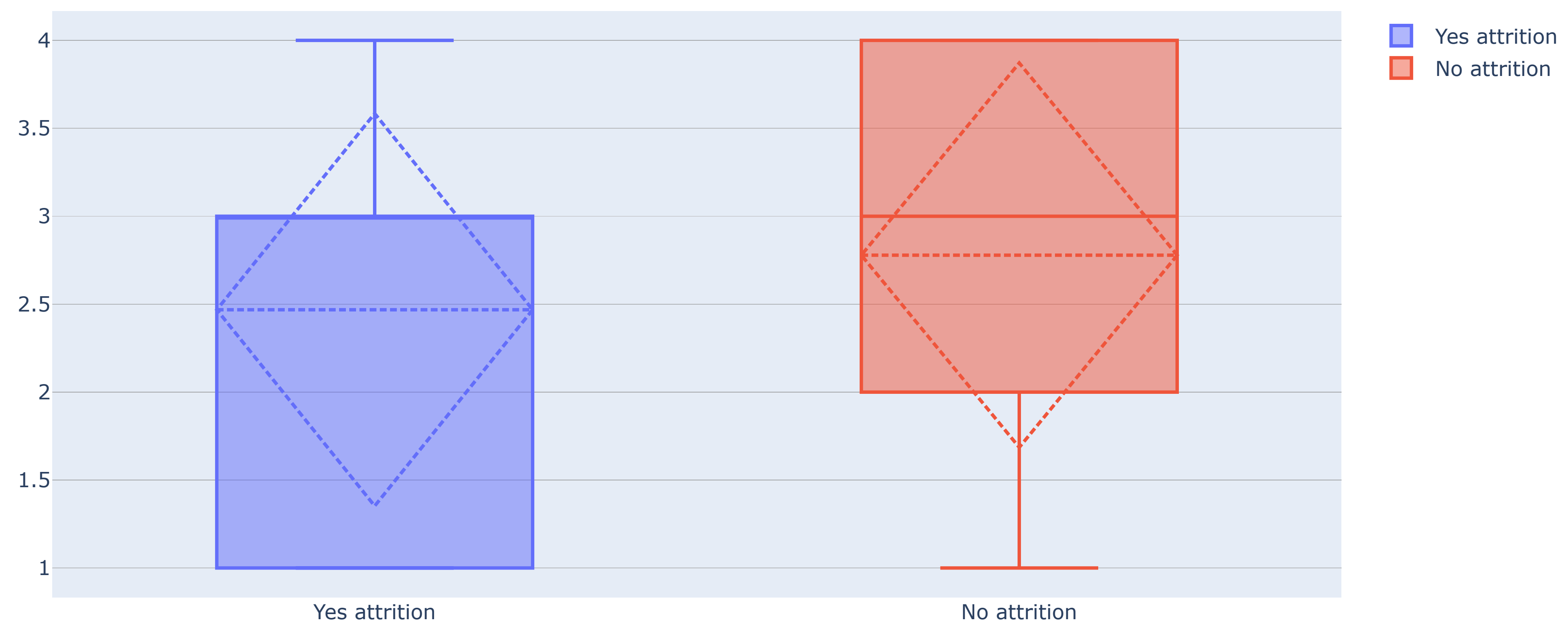





```
Yes-Attrition mean:     2.518987341772152 ± 0.7734047468056179
No-Attrition mean:      2.770478507704785 ± 0.6920498312234991
Absolute mean difference:       0.2514911659326331
```

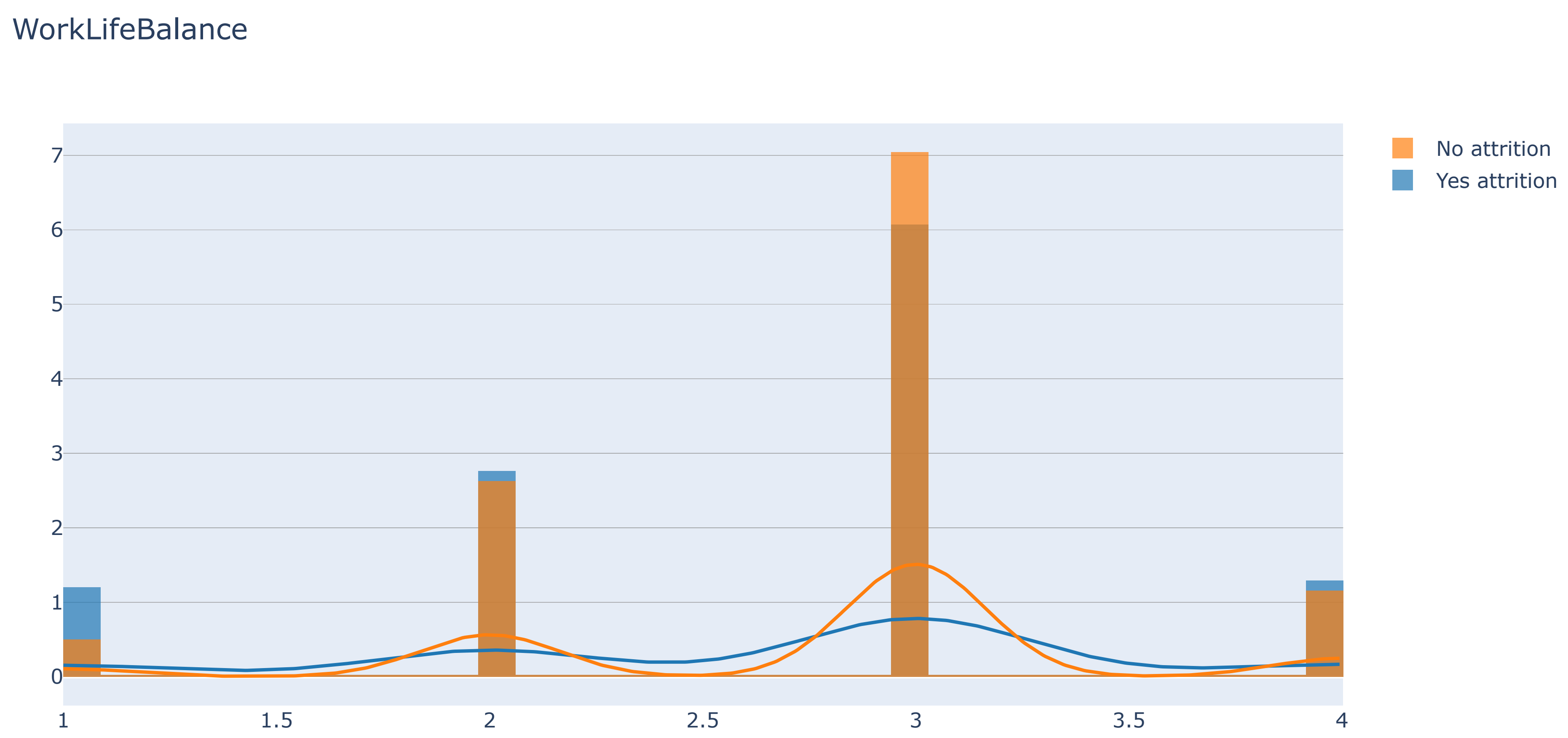
In [40]:
```
vis("JobSatisfaction")
```

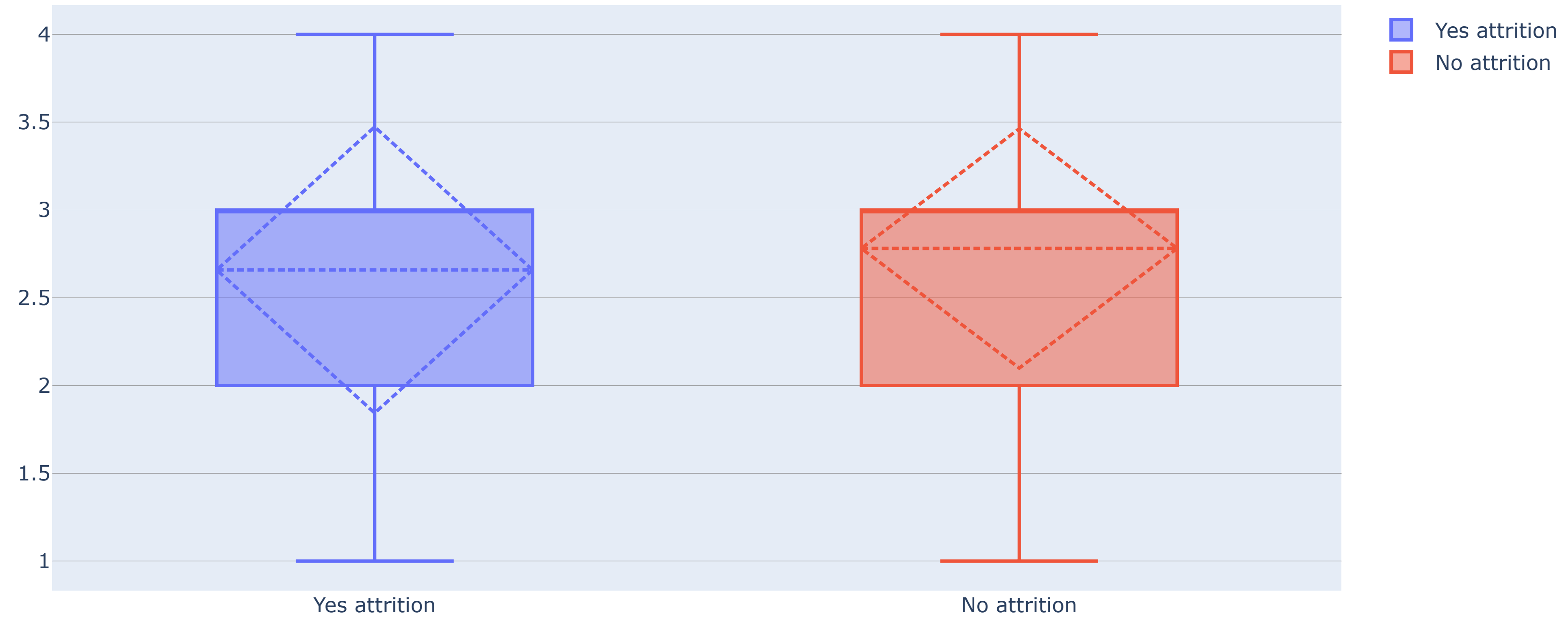





```
Yes-Attrition mean:     2.4683544303797467 ± 1.11805797549084
No-Attrition mean:      2.778588807785888 ± 1.093277400503898
Absolute mean difference:       0.31023437740614135
```

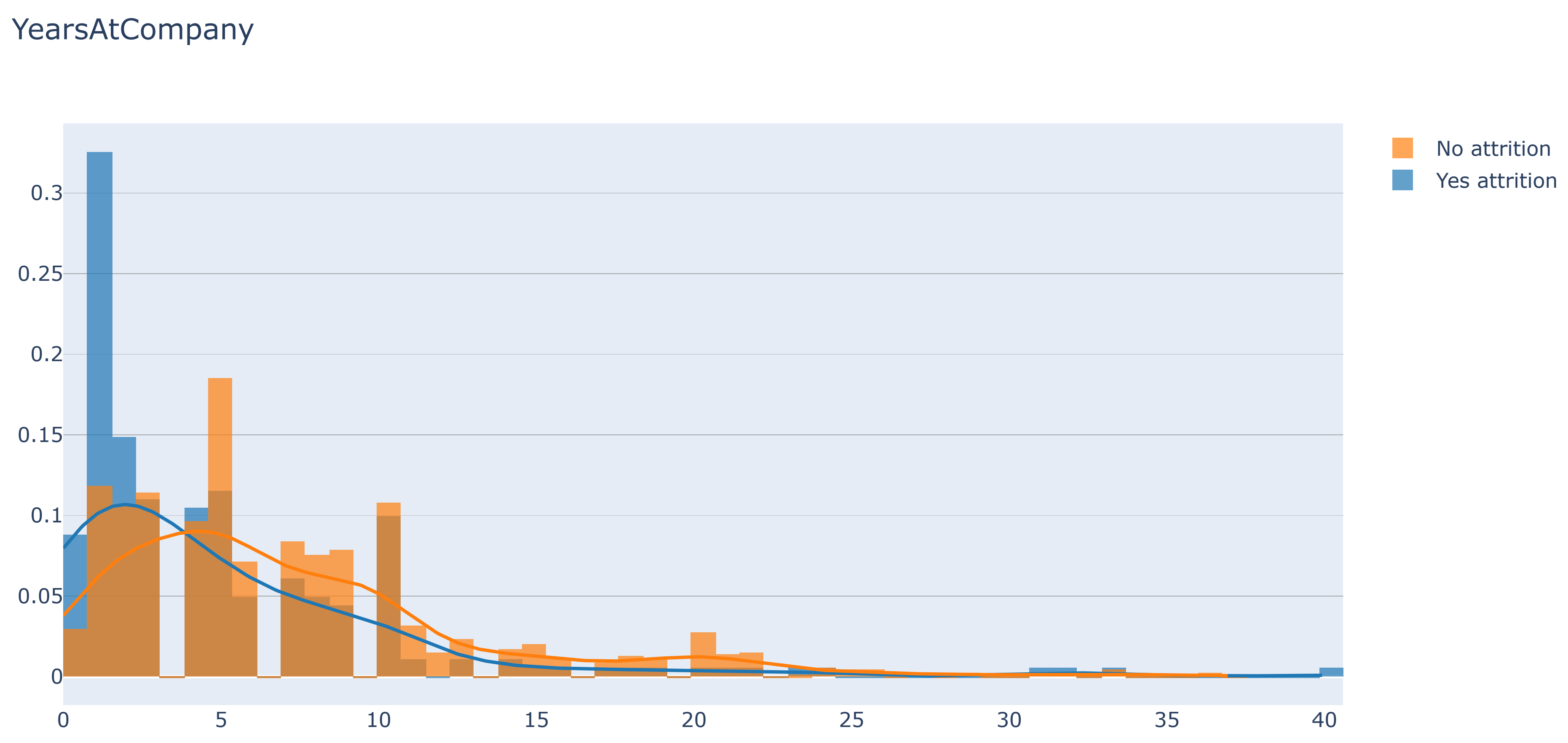
In [41]:
```
vis("WorkLifeBalance")
```

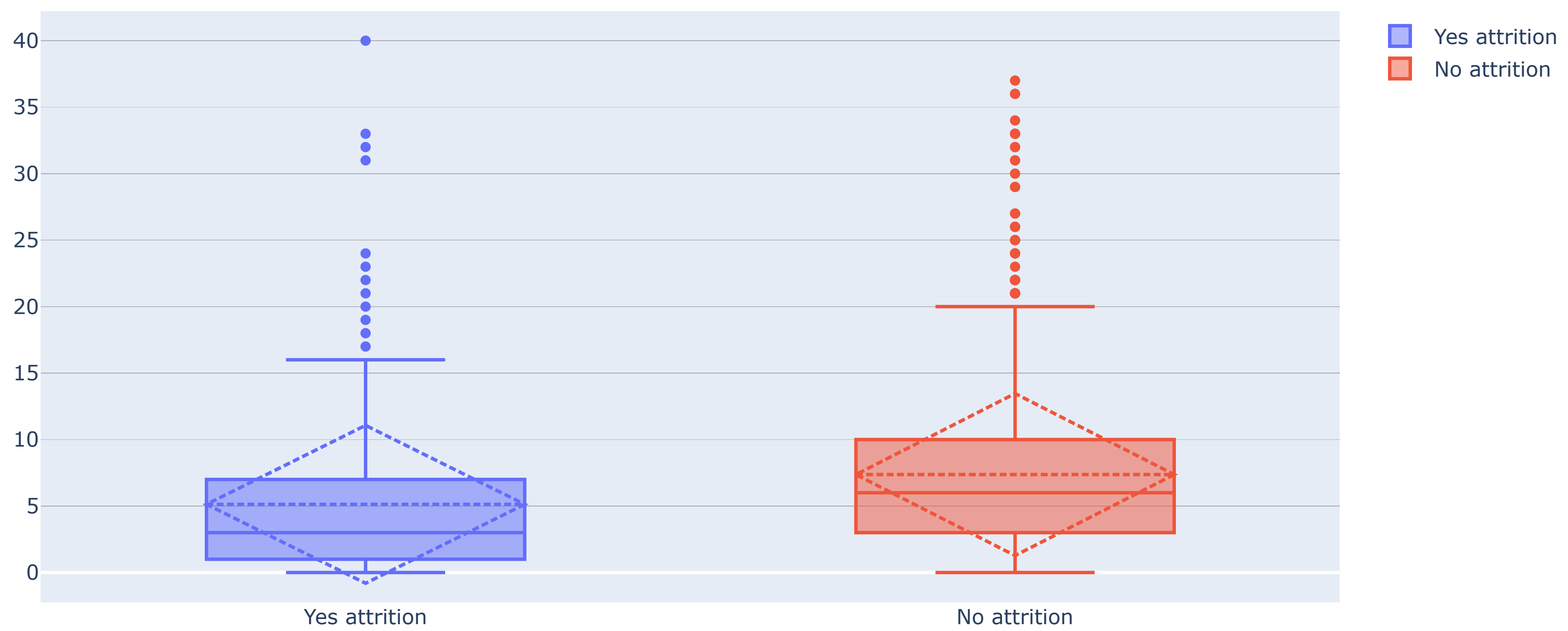





```
Yes-Attrition mean:     2.6582278481012658 ± 0.8164527856864083
No-Attrition mean:      2.781021897810219 ± 0.6819066520792771
Absolute mean difference:       0.12279404970895325
```

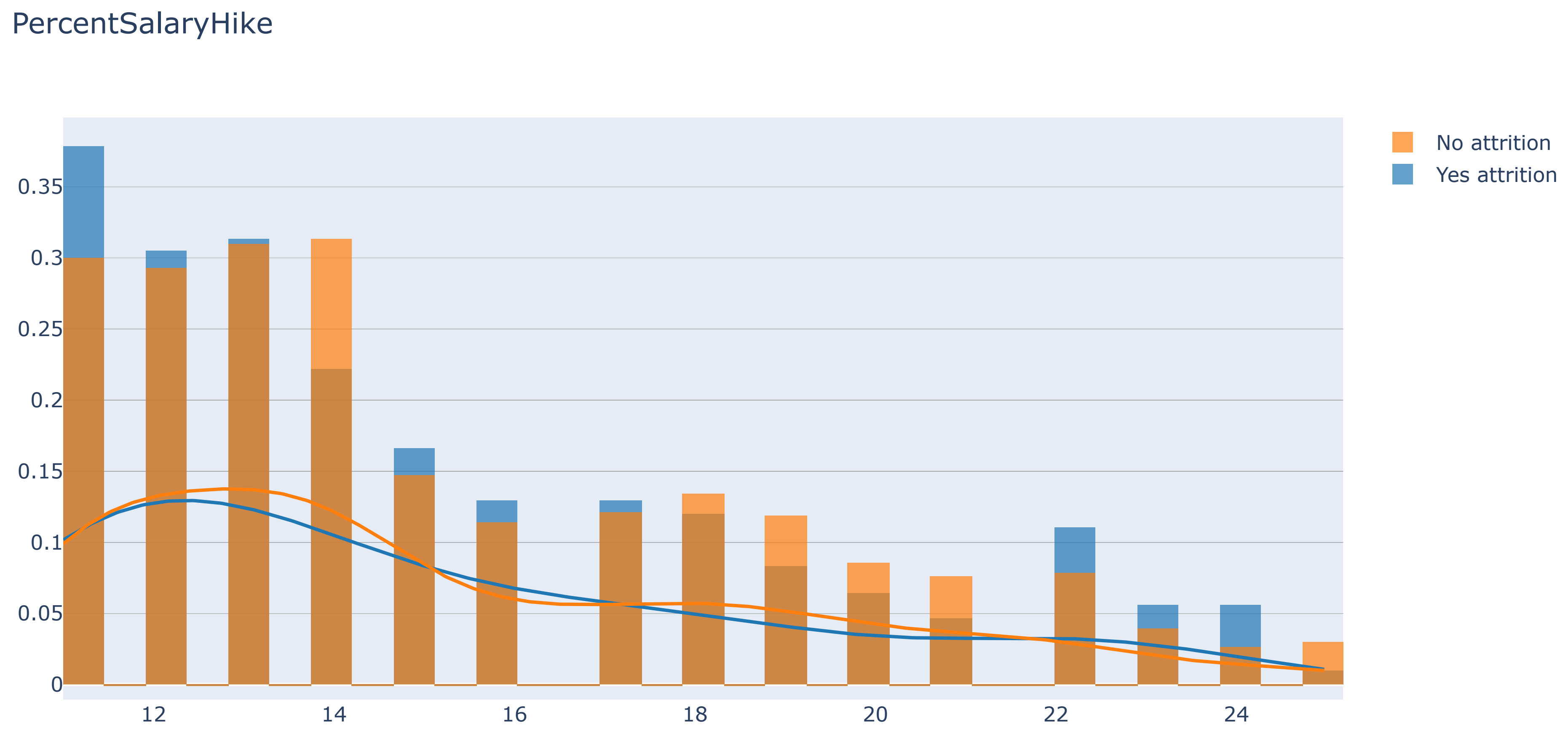
```
In [42]: vis("YearsAtCompany")
```

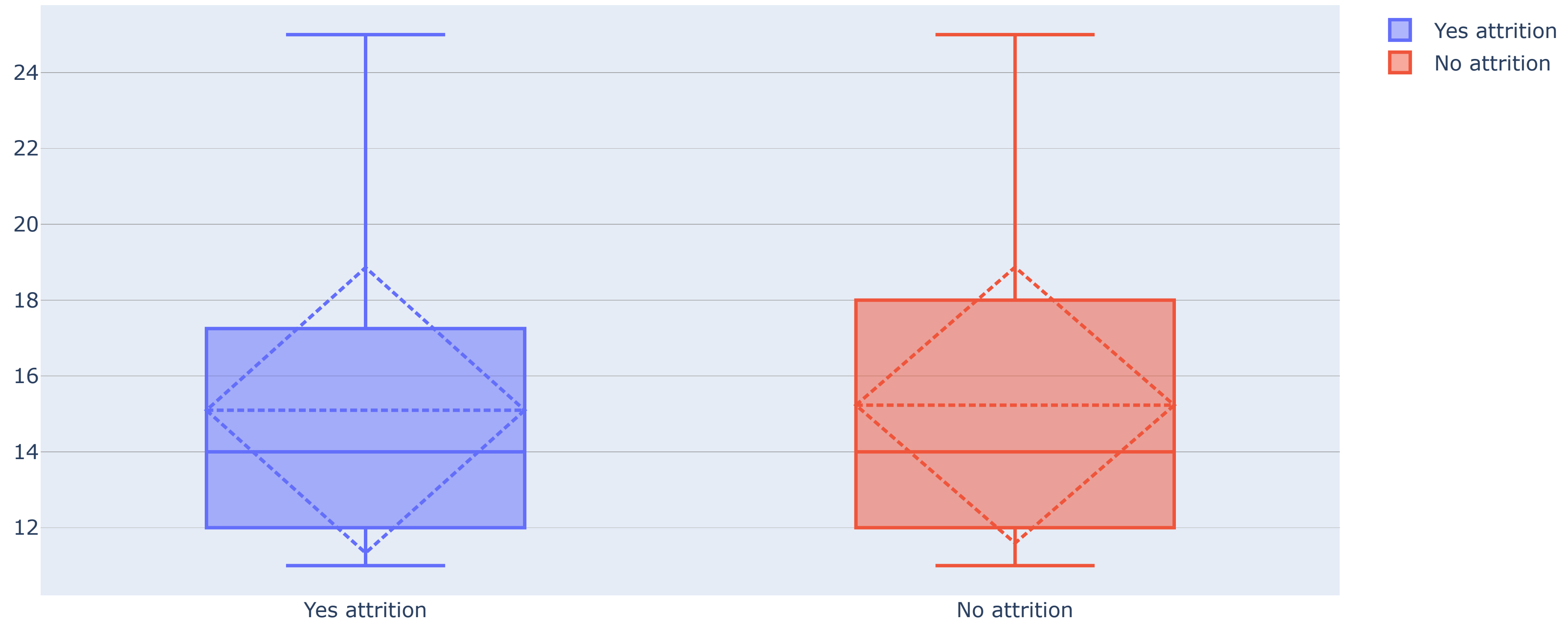





```
Yes-Attrition mean:     5.1308016877637135 ± 5.949984029204934
No-Attrition mean:      7.369018653690187 ± 6.096298144398664
Absolute mean difference:       2.2382169659264735
```

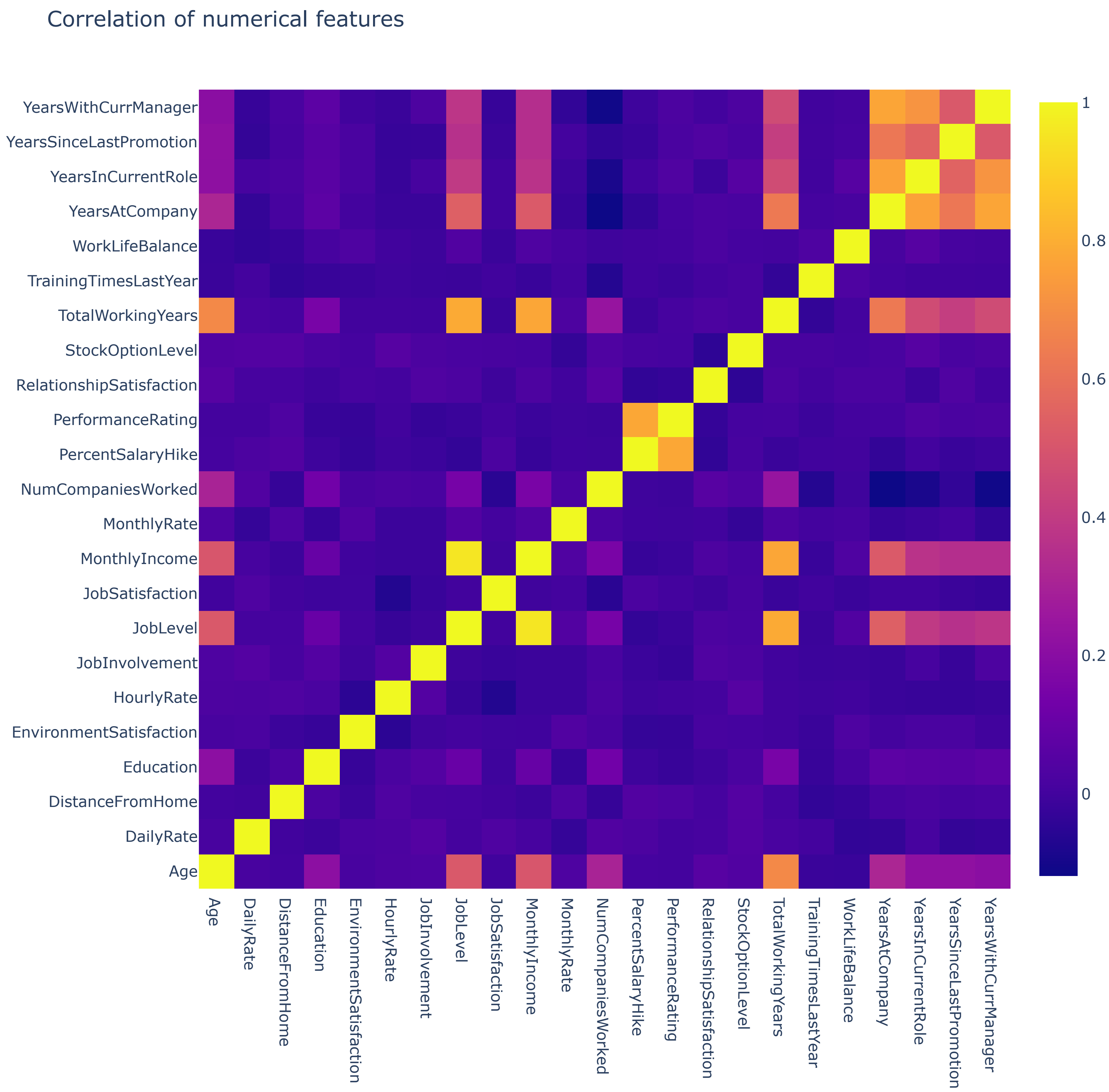
```
In [43]: vis("PercentSalaryHike")
```






```
Yes-Attrition mean:     15.09704641350211 ± 3.77029420045786
No-Attrition mean:      15.231143552311435 ± 3.639511269235044
Absolute mean difference:       0.1340971388093255
```

In [44]:
```
trace = go.Heatmap(
    z = numFeatures.corr(),
    y = numFeatures.columns,
    x = numFeatures.columns
)
layout = go.Layout(
    title = "Correlation of numerical features",
    width = 900,
    height = 900
    )
go.Figure(data = trace, layout = layout).show()
```



Some high correlation within features, possibly redundant

In [45]:
```
#Threshold for correlation to be dropped ≥0.75
data = data = data.drop(columns=["TotalWorkingYears","JobLevel","PerformanceRating","YearsInCurrentRole","YearsWithCurrManager"
])
```

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

Out[46]:

	Attrition	Age	DailyRate	DistanceFromHome	Education	EnvironmentSatisfaction	HourlyRate	JobInvolvement	JobSatisfaction	MonthlyIncome	...	JobRol
0	1	41	1102	1	2	2	94	3	4	5993	...	
1	0	49	279	8	1	3	61	2	2	5130	...	
2	1	37	1373	2	2	4	92	2	3	2090	...	
3	0	33	1392	3	4	4	56	3	3	2909	...	
4	0	27	591	2	1	1	40	3	2	3468	...	

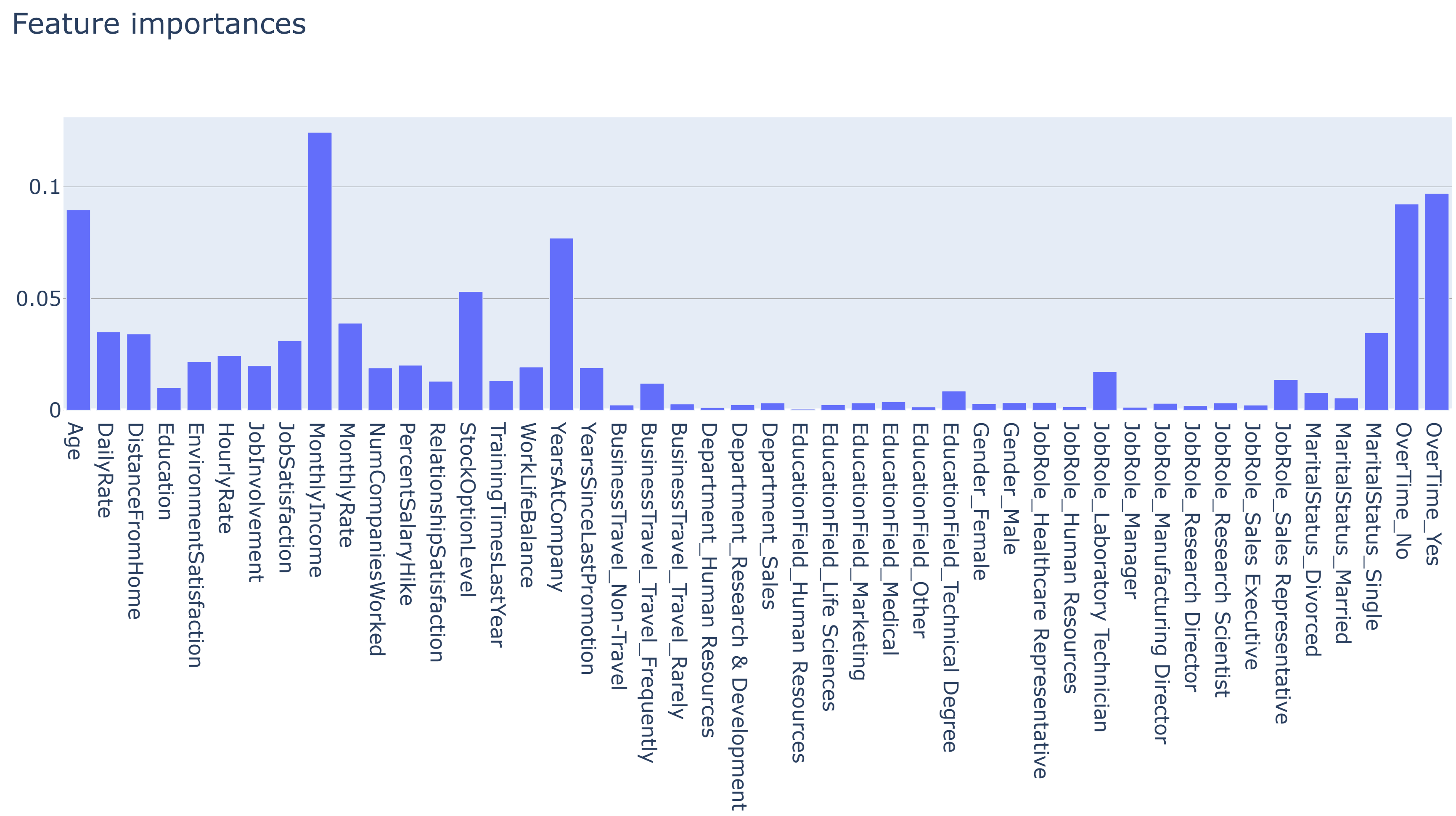
5 rows × 47 columns

In [47]:
```
#Splitting training and test set
data = data.drop(columns=["Attrition"])
xTrain, xTest, yTrain, yTest = train_test_split(data, label, test_size=0.30)
yTrain = np.ravel(yTrain)
yTest = np.ravel(yTest)
```

In [48]:
```
def modelSummary(xTrain, yTrain, xTest, yTest):
    #Create a model
    rf = RandomForestClassifier(n_estimators=1000, max_depth=4)
    #Fit model to training set
    rf.fit(xTrain, yTrain)
    #Predict against test set
    predictions = rf.predict(xTest)
    #Evaluate model results
    print(confusion_matrix(yTest, predictions))
    print(classification_report(yTest,predictions, target_names=["No","Yes"]))

    #Visualize the model feature importance ranks
    trace = go.Bar(
        y = rf.feature_importances_,
        x = data.columns.values
        )
    layout = go.Layout(
        title = "Feature importances"
        )
    fig = go.Figure(data = [trace], layout = layout)
    fig.show()
    #plt.plot(fig)
```

In [49]:
```
modelSummary(xTrain, yTrain, xTest, yTest)
```

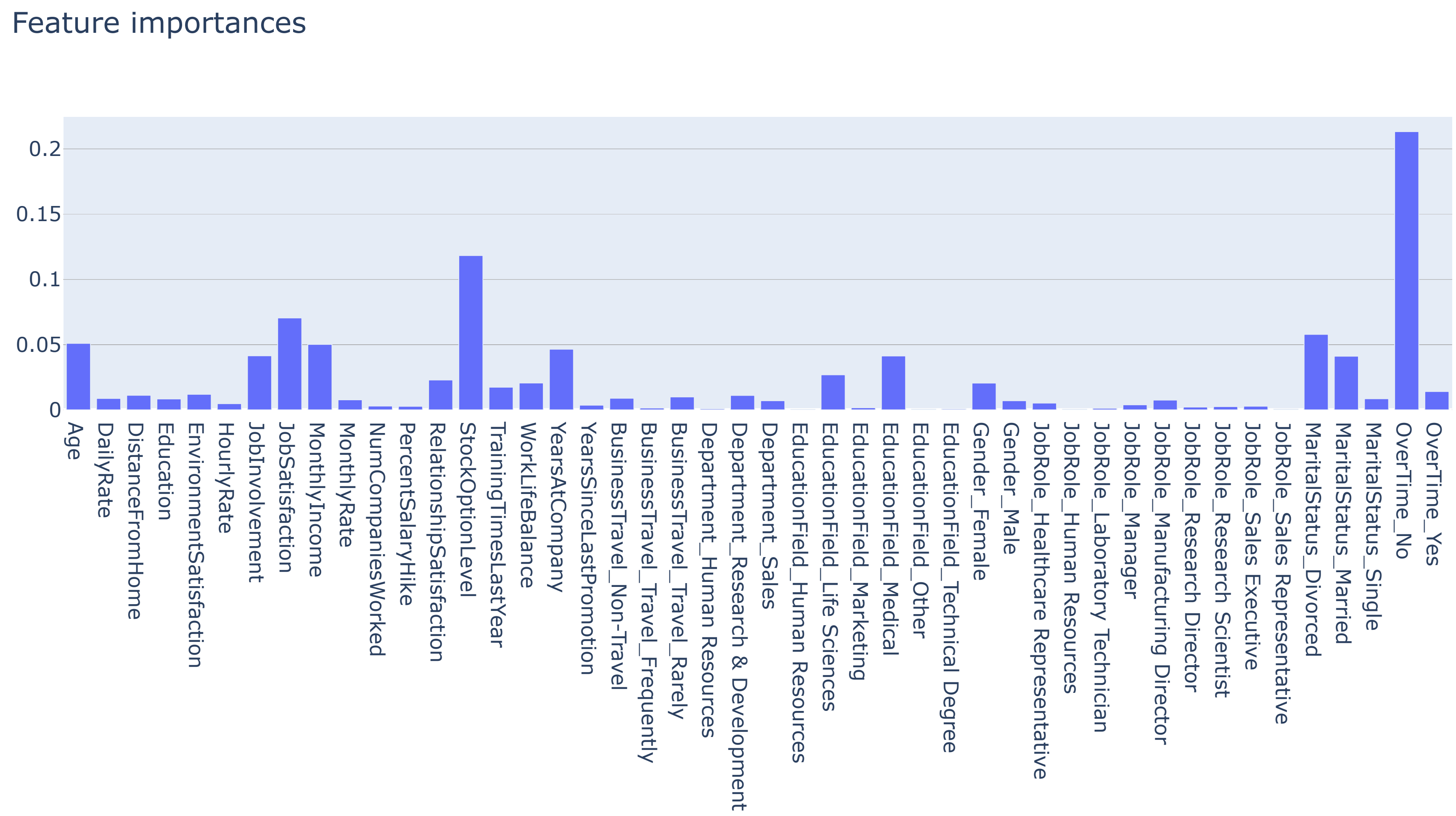
```
[[369   0]
 [ 70   2]]
              precision    recall  f1-score   support

          No       0.84      1.00      0.91       369
         Yes       1.00      0.03      0.05        72

   micro avg       0.84      0.84      0.84       441
   macro avg       0.92      0.51      0.48       441
weighted avg       0.87      0.84      0.77       441
```



While the No-attrition class is classified correctly, the Yes-attrition class is horrible. The imbalanced set makes it difficult to build an accurate model, so we'll try resampling the training set and train again to see how the model performs and how that changes the feature importances

In [50]:
```
#Fitting the model - Oversampling minority class
oversampler = SMOTE()
xOver, yOver = oversampler.fit_sample(xTrain, yTrain)
modelSummary(xOver, yOver, xTest, yTest)
```

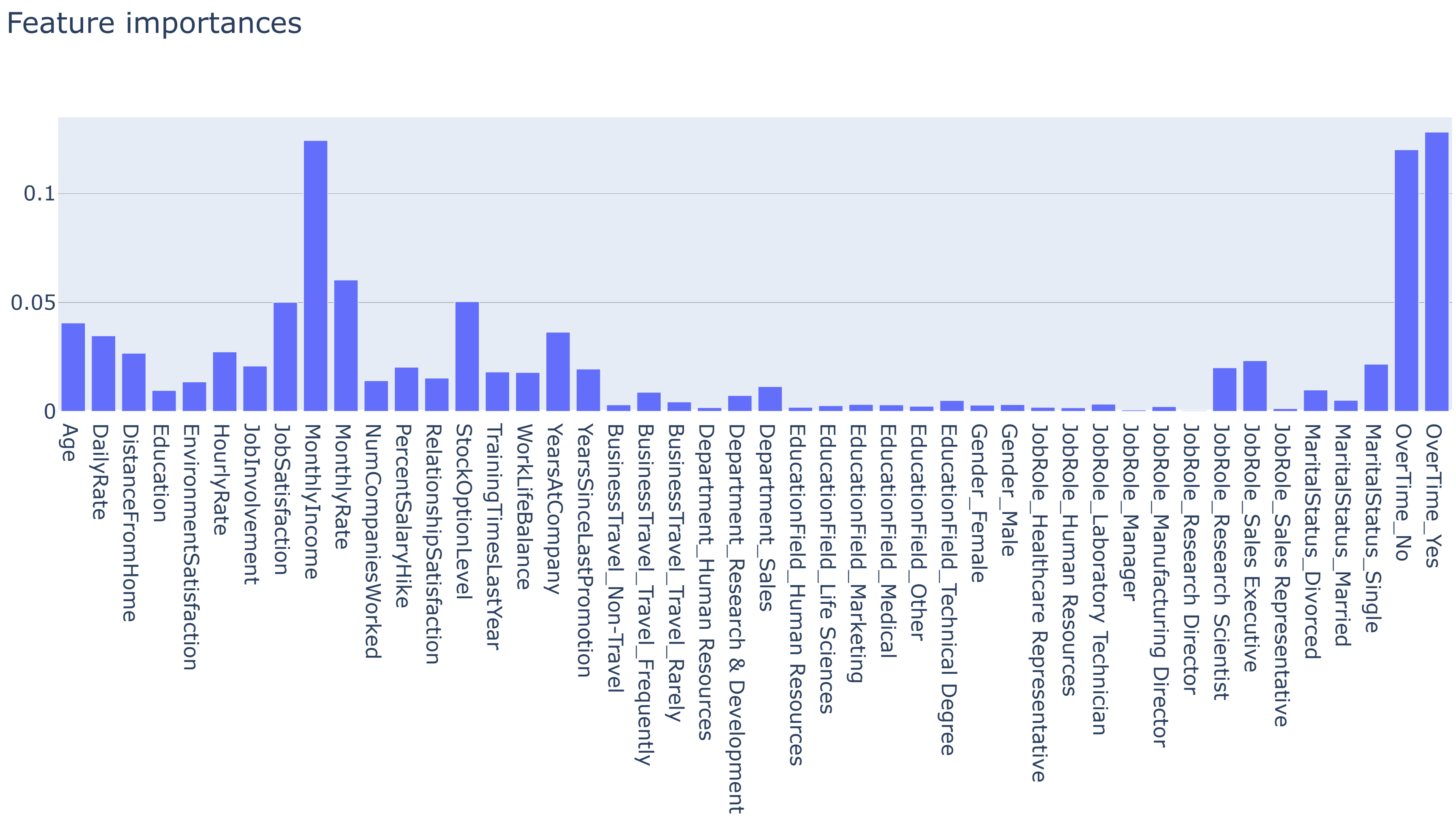
```
[[338  31]
 [ 39  33]]
              precision    recall  f1-score   support

          No       0.90      0.92      0.91       369
         Yes       0.52      0.46      0.49        72

   micro avg       0.84      0.84      0.84       441
   macro avg       0.71      0.69      0.70       441
weighted avg       0.83      0.84      0.84       441
```



The No-attrition accuracy has dropped slightly, but in return we have a much better Yes-attrition accuracy. It's still not fantastic, but it's a lot better than before. Since this was done by oversampling the minority class, we'll try undersampling the majority class next

In [51]:
```
#Fitting the model - Undersampling the majority class
undersampler = NearMiss()
xUnder, yUnder = undersampler.fit_sample(xTrain, yTrain)
modelSummary(xUnder, yUnder, xTest, yTest)
```

```
[[162 207]
 [ 18  54]]
              precision    recall  f1-score   support

          No       0.90      0.44      0.59       369
         Yes       0.21      0.75      0.32        72

   micro avg       0.49      0.49      0.49       441
   macro avg       0.55      0.59      0.46       441
weighted avg       0.79      0.49      0.55       441
```



Now the No-attrition class accuracy has dropped dramatically low, while the Yes-attrition is pretty decent. Overall though, it does not perform as well as the oversampled model from above

In [ ]: