In [1]:
```
# Assignment: DSC680 - Project 1 – Employee Attrition Prediction
# Name: Bezawada, Sashidhar
# Date: 2024-03-30
# Milestone 2 : Draft of White Paper
```

In [2]:
```
# Import libraries
import pandas as pd
import numpy as np
import os
from __future__ import print_function, division
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import statistics
import plotly.express as px

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (accuracy_score, log_loss, classification_report)
```

1. Import the data frame and ensure that the data is loaded properly

In [3]:
```
#Load the dataset as a Pandas data frame
#Read education.csv
df_attrition= pd.read_csv("datasets/WA_Fn-UseC_-HR-Employee-Attrition.csv")
#Display the first ten rows of data
df_attrition.head(10)
```

Out[3]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumber	...	Relatio
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences	1	1	...	
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences	1	2	...	
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other	1	4	...	
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sciences	1	5	...	
4	27	No	Travel_Rarely	591	Research & Development	2	1	Medical	1	7	...	
5	32	No	Travel_Frequently	1005	Research & Development	2	2	Life Sciences	1	8	...	
6	59	No	Travel_Rarely	1324	Research & Development	3	3	Medical	1	10	...	
7	30	No	Travel_Rarely	1358	Research & Development	24	1	Life Sciences	1	11	...	
8	38	No	Travel_Frequently	216	Research & Development	23	3	Life Sciences	1	12	...	
9	36	No	Travel_Rarely	1299	Research & Development	27	3	Medical	1	13	...	

10 rows × 35 columns

Attributes and Descriptions Sl Attribute Name Attribute Description 1 Age The age of the employee. 2 Attrition Whether the employee has left the company or is still employed. 3 BusinessTravel The frequency of the employee's business travel (e.g., "Travel Rarely," "Travel Frequently," "Non-Travel"). 4 DailyRate The daily salary rate of the employee. 5 Department The department in which the employee works (e.g., "Sales," "Human Resources," "Research & Development"). 6 DistanceFromHome The distance between the employee's home and workplace. 7 Education The level of education the employee has achieved. 8 EducationField The field of study for the employee's education. 9 EmployeeCount The number of employees with the same count (possibly constant for all records). 10 EmployeeNumber A unique identifier for each employee. 11 EnvironmentSatisfaction The employee's satisfaction with their working environment. 12 Gender The gender of the employee. 13 HourlyRate The hourly salary rate of the employee. 14 JobInvolvement How engaged the employee is in their current job. 15 JobLevel The level of the employee's job within the organization. 16 JobRole The specific role or position the employee holds. 17 JobSatisfaction The employee's satisfaction with their job. 18 MaritalStatus The marital status of the employee. 19 MonthlyIncome The monthly salary of the employee. 20 MonthlyRate The monthly rate at which the employee is paid. 21 NumCompaniesWorked The number of companies the employee has worked for previously. 22 Over18 Whether the employee is over 18 years old. 23 OverTime Whether the employee works overtime. 24 PercentSalaryHike The percentage increase in the employee's salary. 25 PerformanceRating The employee's performance rating. 26 RelationshipSatisfaction The employee's satisfaction with their relationships at work. 27 StandardHours The standard number of working hours for the employee. 28 StockOptionLevel The level of stock options the employee has. 29 TotalWorkingYears The total number of years the employee has worked. 30 TrainingTimesLastYear How many times the employee received training last year. 31 WorkLifeBalance How well the employee perceives their work-life balance. 32 YearsAtCompany The number of years the employee has been with the company. 33 YearsInCurrentRole The number of years the employee has been in

their current role. 34 YearsSinceLastPromotion The number of years since the employee's last promotion. 35 YearsWithCurrManager The number of years the employee has been managed by their current manager.

In [4]:
```
#Find the information in the data frame.
df_attrition.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
```

Exploratory Data Analysis

In [5]:
```
#Check for Duplicates
have_duplicate_rows = df_attrition.duplicated().any()
have_duplicate_rows
```

Out[5]:
```
False
```

In [6]:
```
#Check for Missing Values
missing_df = df_attrition.isnull().sum().to_frame().rename(columns={0:"Total No. of Missing Values"})
missing_df["% of Missing Values"] = round((missing_df["Total No. of Missing Values"]/len(df_attrition))*100,2)
missing_df
```

Out[6]:

	Total No. of Missing Values	% of Missing Values
Age	0	0.0
Attrition	0	0.0
BusinessTravel	0	0.0
DailyRate	0	0.0
Department	0	0.0
DistanceFromHome	0	0.0
Education	0	0.0
EducationField	0	0.0
EmployeeCount	0	0.0
EmployeeNumber	0	0.0
EnvironmentSatisfaction	0	0.0
Gender	0	0.0
HourlyRate	0	0.0
JobInvolvement	0	0.0
JobLevel	0	0.0
JobRole	0	0.0
JobSatisfaction	0	0.0
MaritalStatus	0	0.0
MonthlyIncome	0	0.0
MonthlyRate	0	0.0
NumCompaniesWorked	0	0.0
Over18	0	0.0
OverTime	0	0.0
PercentSalaryHike	0	0.0
PerformanceRating	0	0.0
RelationshipSatisfaction	0	0.0
StandardHours	0	0.0
StockOptionLevel	0	0.0

	Total No. of Missing Values	% of Missing Values
TotalWorkingYears	0	0.0
TrainingTimesLastYear	0	0.0
WorkLifeBalance	0	0.0
YearsAtCompany	0	0.0
YearsInCurrentRole	0	0.0
YearsSinceLastPromotion	0	0.0
YearsWithCurrManager	0	0.0

In [7]:
```
#Describing each field in the dataset *( Transpose the table)
df_attrition.describe(include="O").T
```

Out[7]:

	count	unique	top	freq
Attrition	1470	2	No	1233
BusinessTravel	1470	3	Travel_Rarely	1043
Department	1470	3	Research & Development	961
EducationField	1470	6	Life Sciences	606
Gender	1470	2	Male	882
JobRole	1470	9	Sales Executive	326
MaritalStatus	1470	3	Married	673
Over18	1470	1	Y	1470
OverTime	1470	2	No	1054

In [8]:
```
#Describing each field in the dataset *( Transpose the table)
df_attrition.describe(exclude="O").T
```

Out[8]:

	count	mean	std	min	25%	50%	75%	max
Age	1470.0	36.923810	9.135373	18.0	30.00	36.0	43.00	60.0
DailyRate	1470.0	802.485714	403.509100	102.0	465.00	802.0	1157.00	1499.0
DistanceFromHome	1470.0	9.192517	8.106864	1.0	2.00	7.0	14.00	29.0
Education	1470.0	2.912925	1.024165	1.0	2.00	3.0	4.00	5.0
EmployeeCount	1470.0	1.000000	0.000000	1.0	1.00	1.0	1.00	1.0
EmployeeNumber	1470.0	1024.865306	602.024335	1.0	491.25	1020.5	1555.75	2068.0
EnvironmentSatisfaction	1470.0	2.721769	1.093082	1.0	2.00	3.0	4.00	4.0
HourlyRate	1470.0	65.891156	20.329428	30.0	48.00	66.0	83.75	100.0
JobInvolvement	1470.0	2.729932	0.711561	1.0	2.00	3.0	3.00	4.0
JobLevel	1470.0	2.063946	1.106940	1.0	1.00	2.0	3.00	5.0
JobSatisfaction	1470.0	2.728571	1.102846	1.0	2.00	3.0	4.00	4.0
MonthlyIncome	1470.0	6502.931293	4707.956783	1009.0	2911.00	4919.0	8379.00	19999.0
MonthlyRate	1470.0	14313.103401	7117.786044	2094.0	8047.00	14235.5	20461.50	26999.0
NumCompaniesWorked	1470.0	2.693197	2.498009	0.0	1.00	2.0	4.00	9.0
PercentSalaryHike	1470.0	15.209524	3.659938	11.0	12.00	14.0	18.00	25.0
PerformanceRating	1470.0	3.153741	0.360824	3.0	3.00	3.0	3.00	4.0
RelationshipSatisfaction	1470.0	2.712245	1.081209	1.0	2.00	3.0	4.00	4.0
StandardHours	1470.0	80.000000	0.000000	80.0	80.00	80.0	80.00	80.0
StockOptionLevel	1470.0	0.793878	0.852077	0.0	0.00	1.0	1.00	3.0
TotalWorkingYears	1470.0	11.279592	7.780782	0.0	6.00	10.0	15.00	40.0
TrainingTimesLastYear	1470.0	2.799320	1.289271	0.0	2.00	3.0	3.00	6.0
WorkLifeBalance	1470.0	2.761224	0.706476	1.0	2.00	3.0	3.00	4.0
YearsAtCompany	1470.0	7.008163	6.126525	0.0	3.00	5.0	9.00	40.0
YearsInCurrentRole	1470.0	4.229252	3.623137	0.0	2.00	3.0	7.00	18.0
YearsSinceLastPromotion	1470.0	2.187755	3.222430	0.0	0.00	1.0	3.00	15.0
YearsWithCurrManager	1470.0	4.123129	3.568136	0.0	2.00	3.0	7.00	17.0

In [9]:
```
category_cols = df_attrition.select_dtypes(include="O").columns

for column in category_cols:
    print('Unique values of ', column, set(df_attrition[column]))
    print("-"*127)
```

```
Unique values of  Attrition {'No', 'Yes'}
-------------------------------------------------------------------------------------------------------------------------------
Unique values of  BusinessTravel {'Travel_Frequently', 'Non-Travel', 'Travel_Rarely'}
-------------------------------------------------------------------------------------------------------------------------------
Unique values of  Department {'Sales', 'Research & Development', 'Human Resources'}
-------------------------------------------------------------------------------------------------------------------------------
Unique values of  EducationField {'Technical Degree', 'Human Resources', 'Other', 'Medical', 'Life Sciences', 'Marketing'}
-------------------------------------------------------------------------------------------------------------------------------
Unique values of  Gender {'Male', 'Female'}
-------------------------------------------------------------------------------------------------------------------------------
Unique values of  JobRole {'Healthcare Representative', 'Human Resources', 'Manufacturing Director', 'Sales Representative', 'Man
ager', 'Research Scientist', 'Sales Executive', 'Laboratory Technician', 'Research Director'}
-------------------------------------------------------------------------------------------------------------------------------
Unique values of  MaritalStatus {'Divorced', 'Single', 'Married'}
-------------------------------------------------------------------------------------------------------------------------------
Unique values of  Over18 {'Y'}
-------------------------------------------------------------------------------------------------------------------------------
Unique values of  OverTime {'No', 'Yes'}
-------------------------------------------------------------------------------------------------------------------------------
```

Data Transformations

In [10]:
```
#Deleting the following Columns as their values is same across all observations
# Over18 ='Y' , EmployeeCount = 1 , StandardHours = 80

df_attrition.drop(columns=["Over18","EmployeeCount", "StandardHours"], inplace=True)
```

In [11]:
```
#Replacing "Yes" and "No" in traget variable with 1 and 0

df_attrition.Attrition.replace({"Yes":1,"No":0}, inplace=True)
df_attrition.OverTime.replace({"Yes":1,"No":0}, inplace=True)
```

In [12]:
```
#Labelling of Categories in Numerical Features

df_attrition["BusinessTravel"] = df_attrition["BusinessTravel"].replace({1:"Non-Travel", 2:"Travel_Rarely", 3:"Travel_Frequently"}

df_attrition["Education"]      = df_attrition["Education"].replace({1:"Below College",2:"College",3:"Bachelor",4:"Master",5:"Doctc

df_attrition["JobInvolvement"] = df_attrition["JobInvolvement"].replace({1:"Low",2:"Medium",3:"High",4:"Very High"})

df_attrition["JobLevel"]       = df_attrition["JobLevel"].replace({1:"Entry Level",2:"Junior Level",3:"Mid Level",4:"Senior Level"
```

```
df_attrition["JobSatisfaction"] = df_attrition["JobSatisfaction"].replace({1:"Low",2:"Medium",3:"High",4:"Very High"})

df_attrition["PerformanceRating"] = df_attrition["PerformanceRating"].replace({1:"Low",2:"Good",3:"Excellent",4:"Outstanding"})

df_attrition["RelationshipSatisfaction"] = df_attrition["RelationshipSatisfaction"].replace({1:"Low",2:"Medium",3:"High",4:"Very H

df_attrition["WorkLifeBalance"] = df_attrition["WorkLifeBalance"].replace({1:"Bad",2:"Good",3:"Better",4:"Best"})
```

In [13]:
```
df_attrition.head(5)
```

Out[13]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeNumber	EnvironmentSatisfaction	...
0	41	1	Travel_Rarely	1102	Sales	1	College	Life Sciences	1	2	...
1	49	0	Travel_Frequently	279	Research & Development	8	Below College	Life Sciences	2	3	...
2	37	1	Travel_Rarely	1373	Research & Development	2	College	Other	4	4	...
3	33	0	Travel_Frequently	1392	Research & Development	3	Master	Life Sciences	5	4	...
4	27	0	Travel_Rarely	591	Research & Development	2	Below College	Medical	7	1	...

5 rows × 32 columns

As we can observe from the dataset, our target column with which we can point our model to train on would be the "Attrition" column.

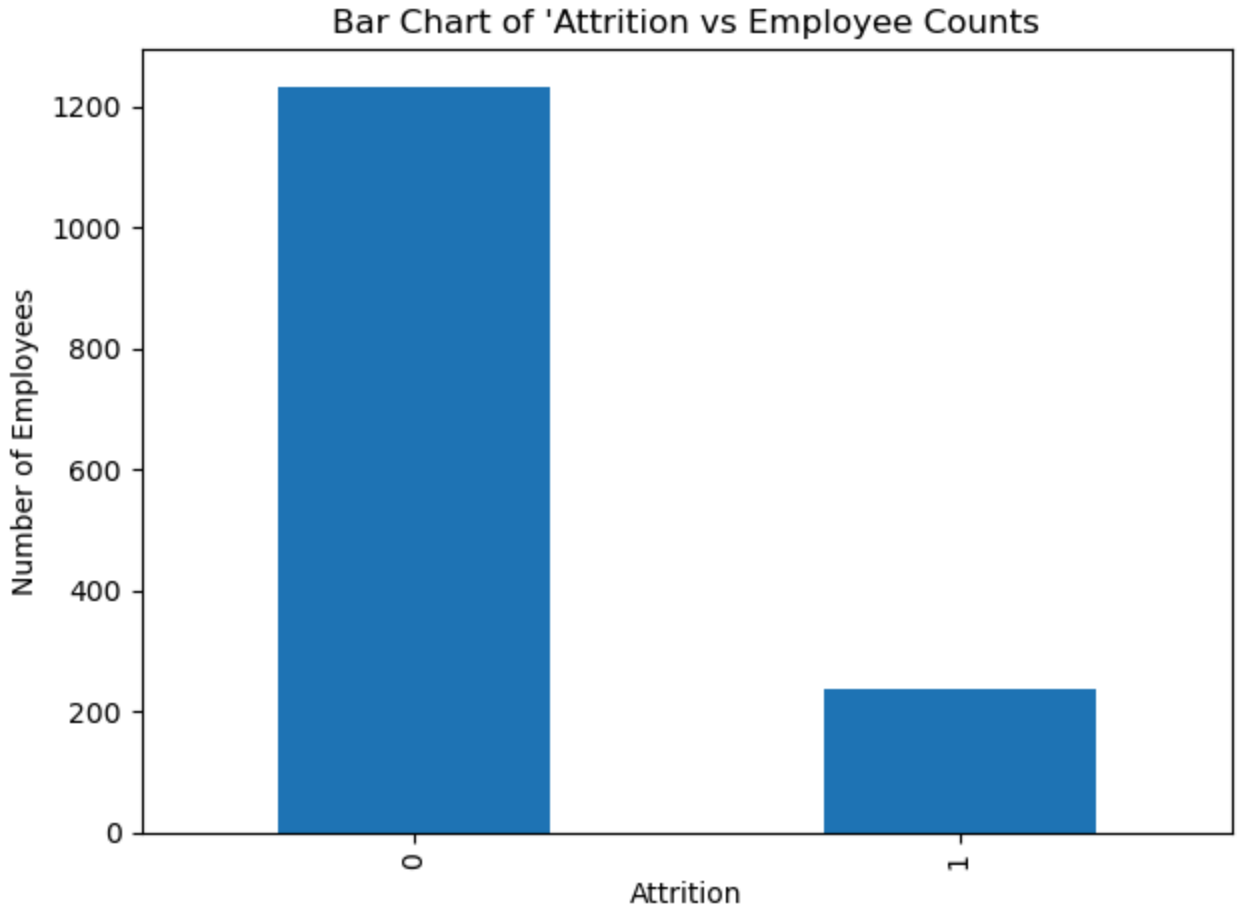
In [14]:
```
df_attrition.Attrition
```

Out[14]:
```
0       1
1       0
2       1
3       0
4       0
       ..
1465    0
1466    0
1467    0
1468    0
1469    0
Name: Attrition, Length: 1470, dtype: int64
```

Charts

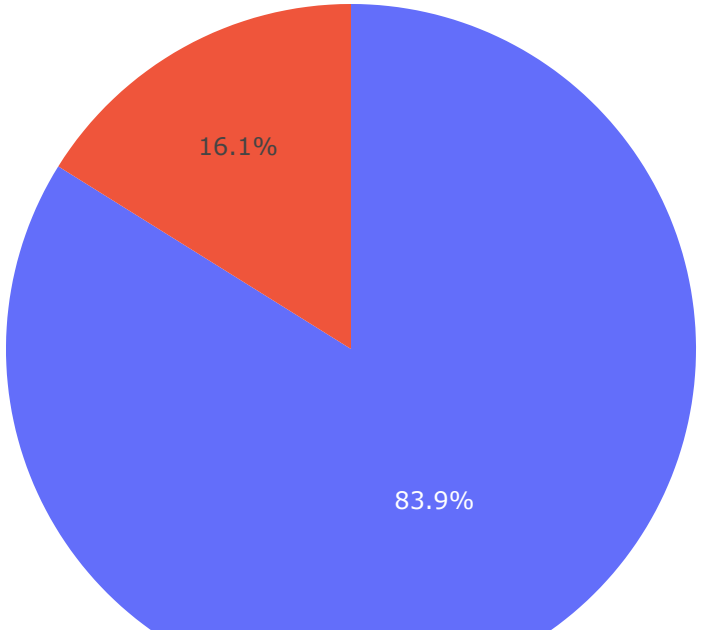
In [15]:
```
# Bar Chart of Attrition
print(df_attrition['Attrition'].value_counts())
df_attrition['Attrition'].value_counts().plot(kind='bar')
plt.title("Bar Chart of 'Attrition vs Employee Counts")
plt.xlabel('Attrition')
plt.ylabel('Number of Employees')
plt.tight_layout()
plt.show()
```

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

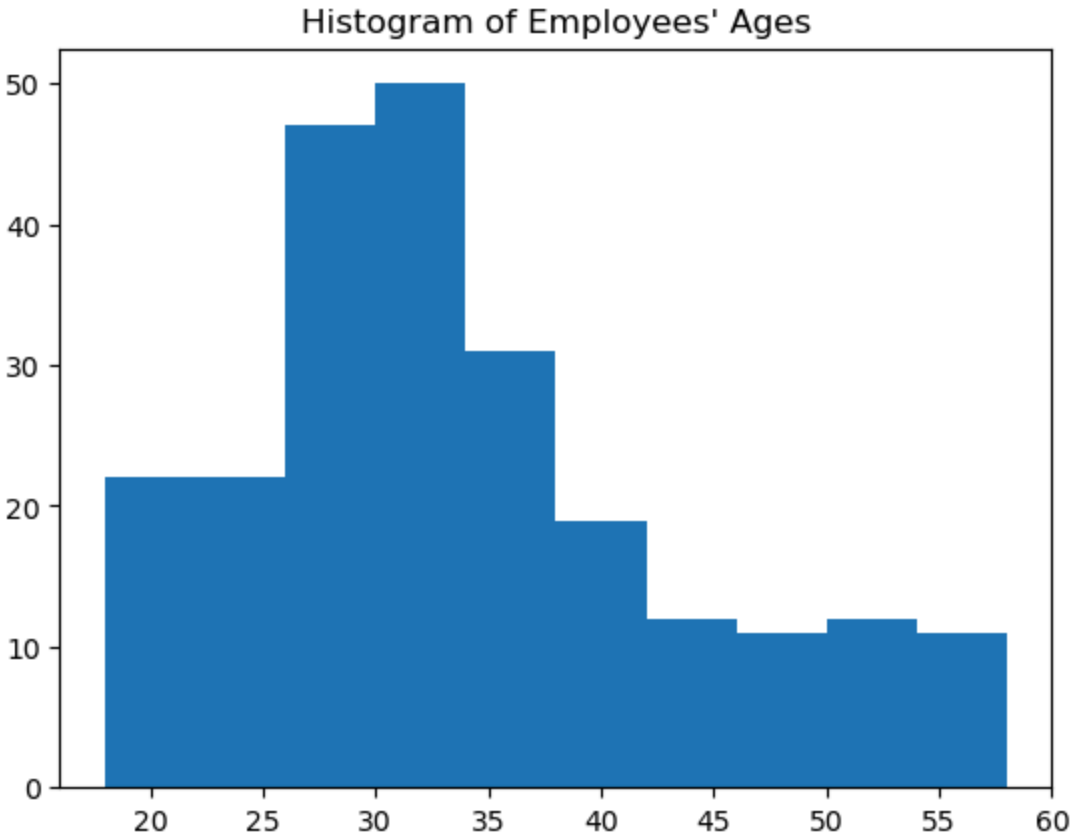


In [16]:
```
fig = px.pie(df_attrition['Attrition'],  names='Attrition',title='Employee Attrition Rate')
fig.show()
```

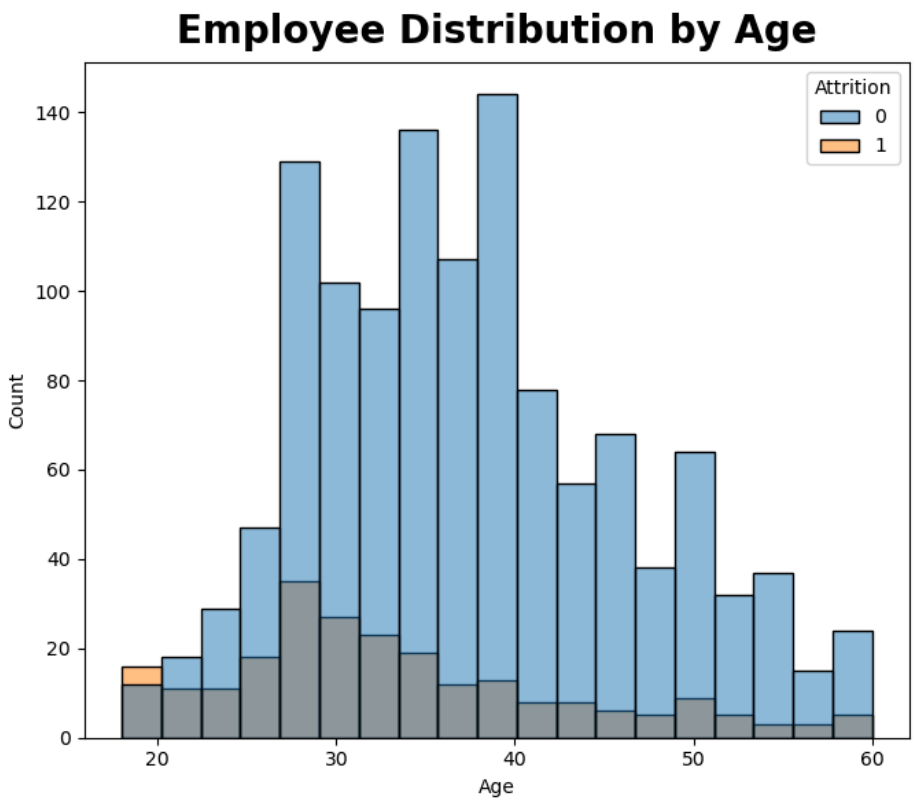
Employee Attrition Rate

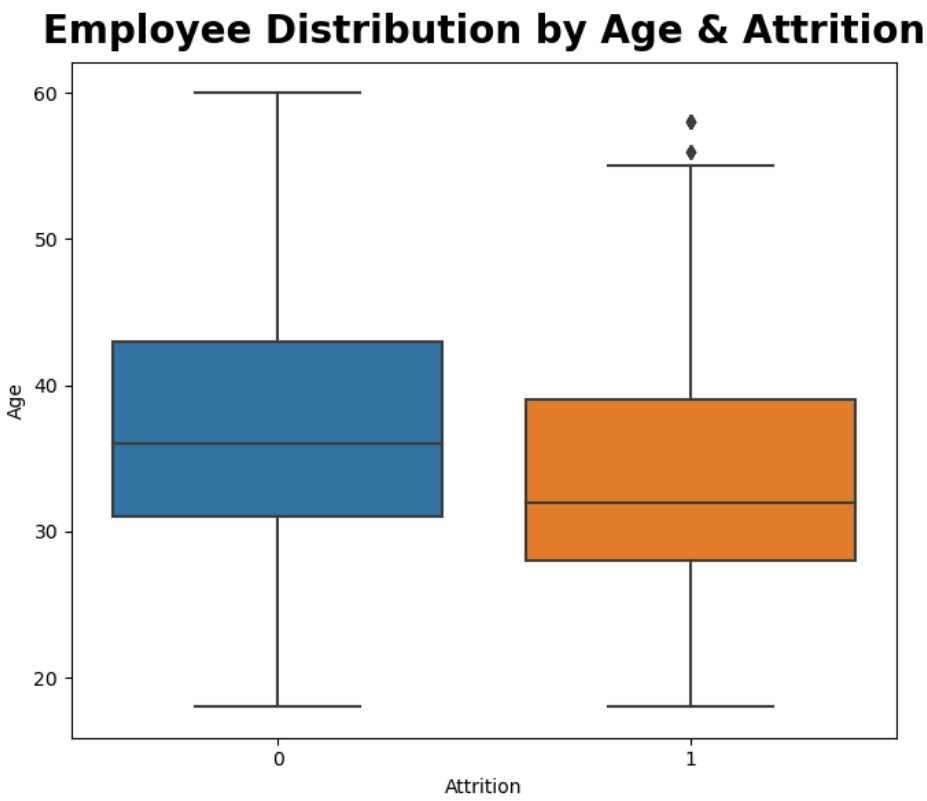


In [17]:
```
# Histogram of Age
new_df_attrition=df_attrition[df_attrition['Attrition']==1]
plt.hist(new_df_attrition['Age'],bins=10)
plt.title("Histogram of Employees' Ages")
plt.show()
```



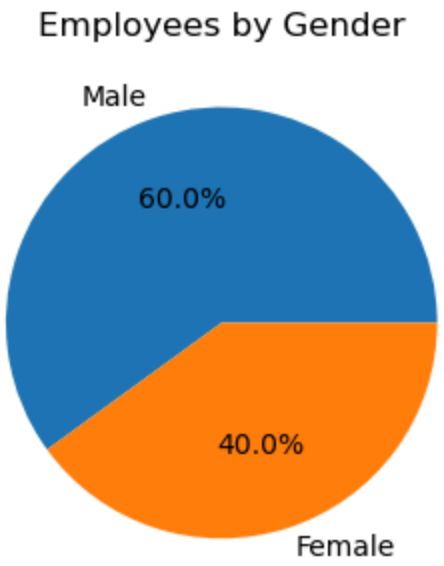
In [18]:
```
#Visualization to show Employee Distribution by Age.
plt.figure(figsize=(13.5,6))
plt.subplot(1,2,1)
sns.histplot(x="Age",hue="Attrition",data=df_attrition)
plt.title("Employee Distribution by Age",fontweight="black",size=20,pad=10)


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

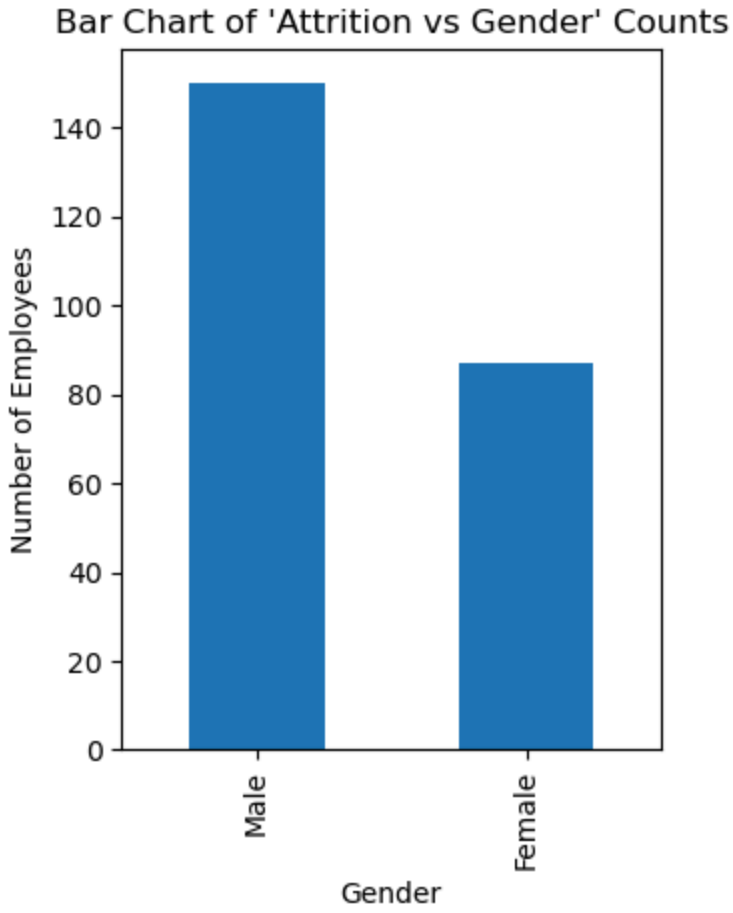




In [19]:

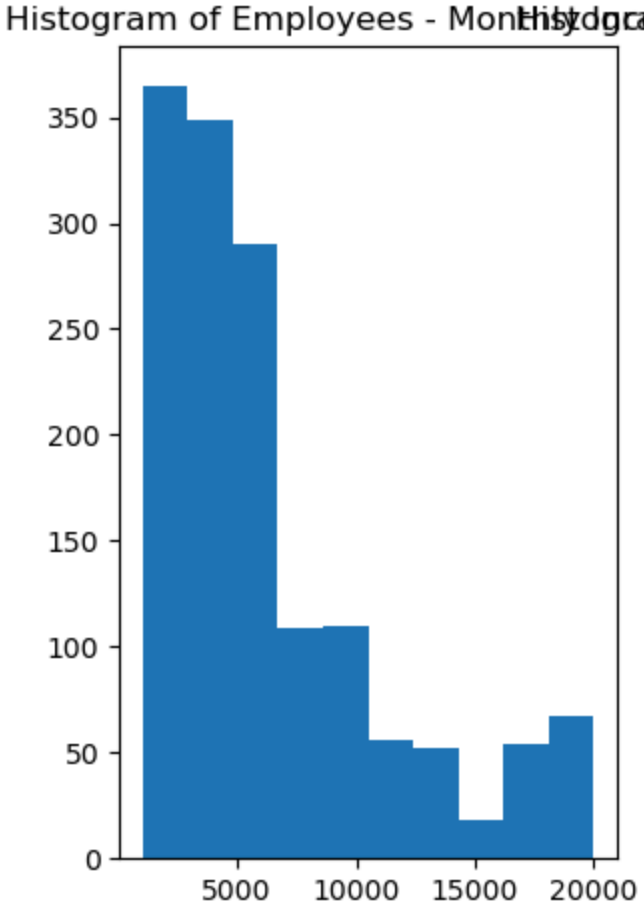
```
#Visualization to show Total Employees by Businees Travel.
plt.subplot(1,2,1)
value_1 = df_attrition['Gender'].value_counts()
plt.title("Employees by Gender")
plt.pie(value_1.values, labels=value_1.index, autopct="%.1f%%")

#Visualization to show Employee Attrition by Gender.
plt.subplot(1,2,2)
new_df_attrition=df_attrition[df_attrition['Attrition']==1]
new_df_attrition['Gender'].value_counts().plot(kind='bar')
plt.title("Bar Chart of 'Attrition vs Gender' Counts")
plt.xlabel('Gender')
plt.ylabel('Number of Employees')
plt.tight_layout()
plt.show()
```

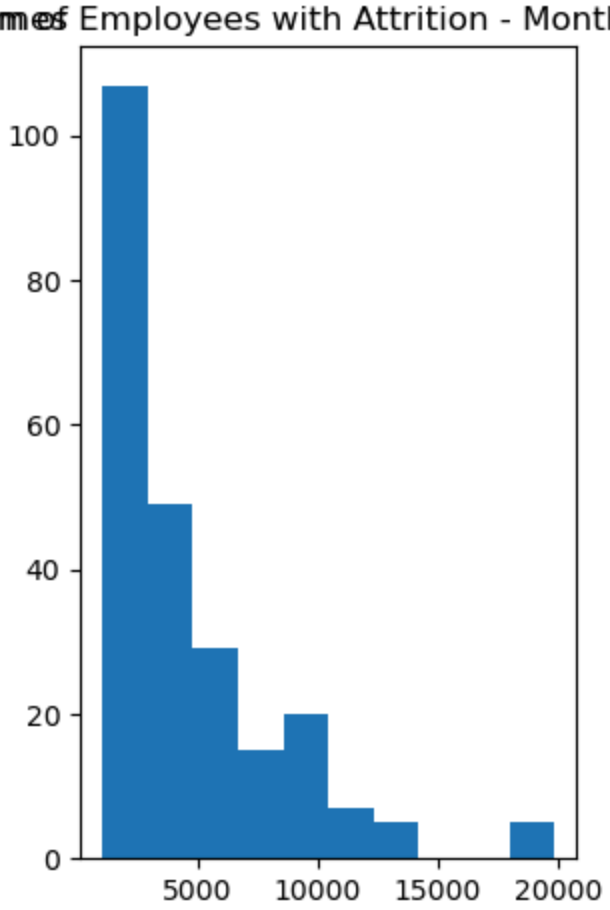




Age group of the dataset is between 18 to 60

In [20]:
```
#Histogram of Monthly Income with Bins
plt.subplot(1,2,1)
plt.hist(df_attrition['MonthlyIncome'],bins=10)
plt.title("Histogram of Employees - Monthly Incomes")

#Visualization to show Attrition Rate by Businees Travel.
plt.subplot(1,2,2)
new_df = df_attrition[df_attrition["Attrition"]==1]
plt.hist(new_df['MonthlyIncome'],bins=10)
plt.title("Histogram of Employees with Attrition - Monthly Incomes")
plt.tight_layout()
plt.show()
```





In [21]:
```
#calculating the median income among the employees
income_med = statistics.median(df_attrition['MonthlyIncome'])
print("Median Employee Monthly Income: $",income_med)
```

```
Median Employee Monthly Income: $ 4919.0
```

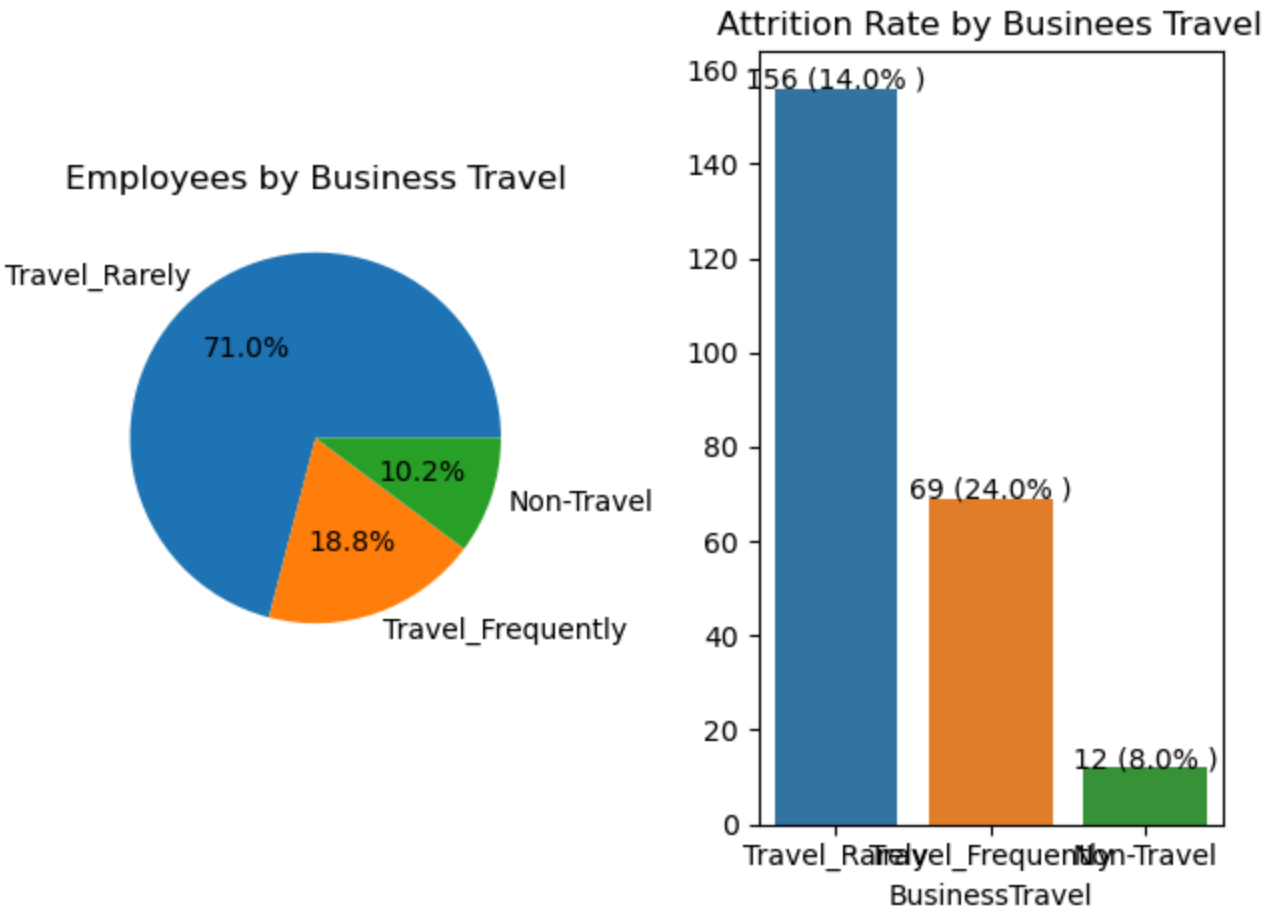
In [22]:
```
#Analyzing outliers from Monthly income analysis
#less than $10,000
reg_income = df_attrition[df_attrition.MonthlyIncome < 10000]

reg_income.shape
```

Out[22]:
```
(1189, 32)
```

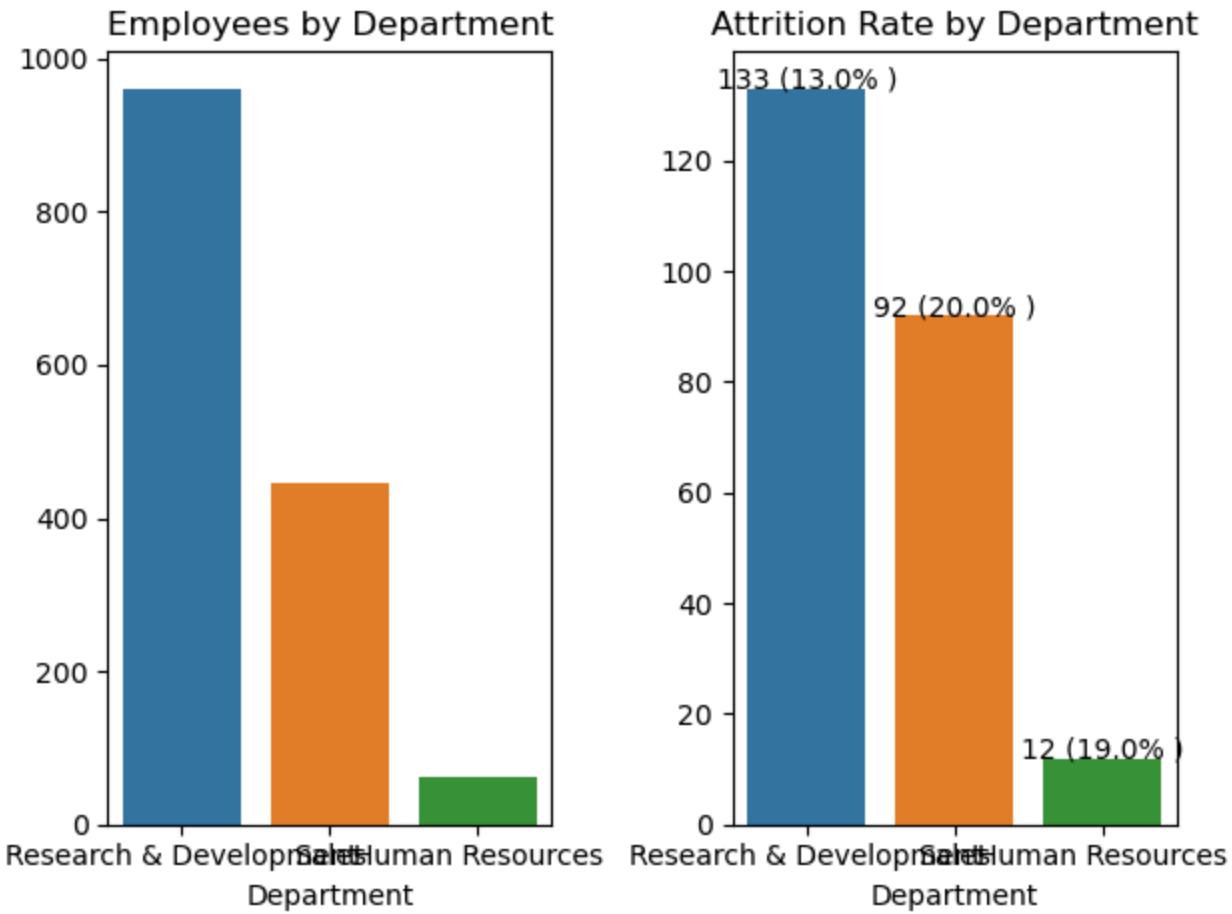
In [23]:
```
#high income = greater than $10,000
high_income = df_attrition[df_attrition.MonthlyIncome > 10000]
high_income.shape
```

Out[23]:
```
(281, 32)
```

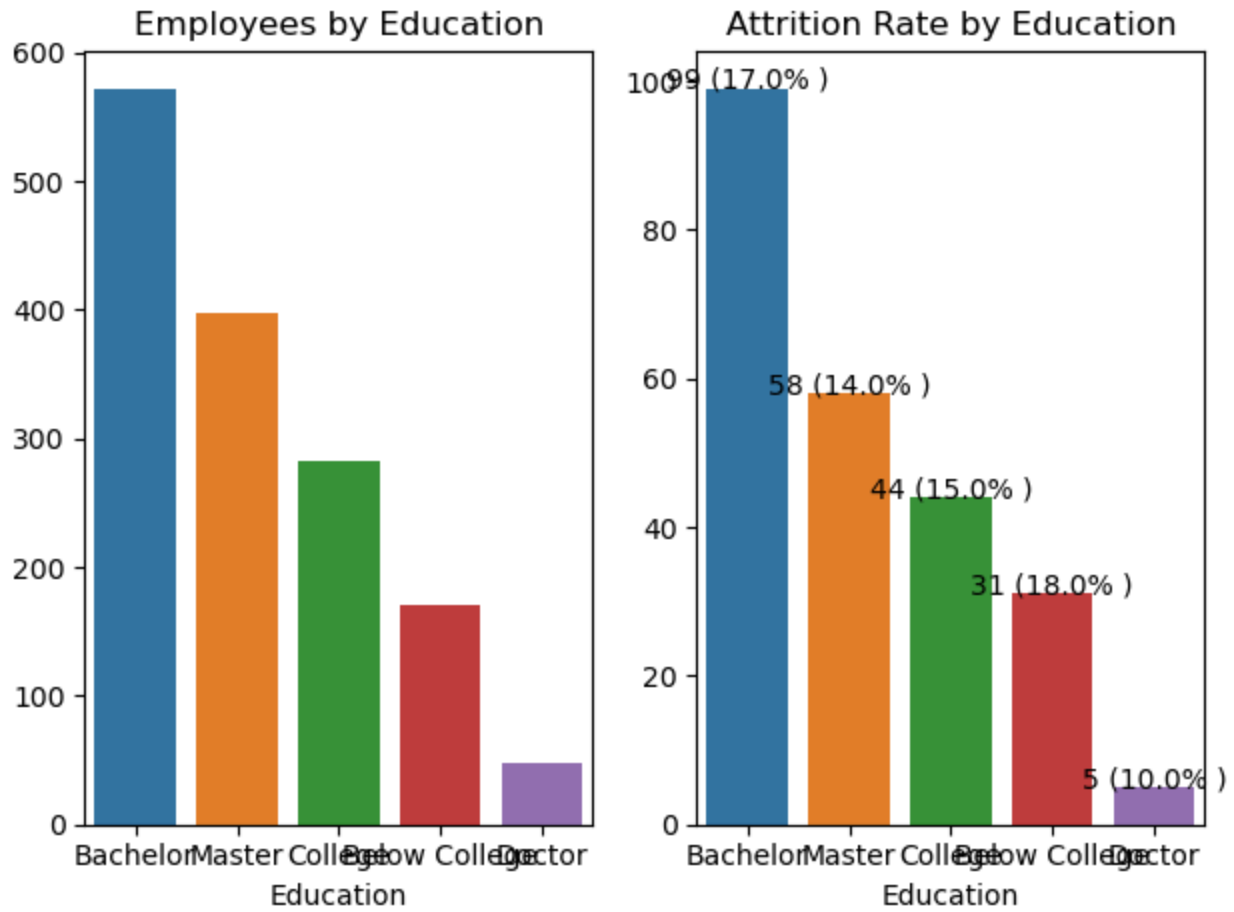
In [24]:
```
#Visualization to show Total Employees by Businees Travel.
plt.subplot(1,2,1)
value_1 = df_attrition["BusinessTravel"].value_counts()
plt.title("Employees by Business Travel")
plt.pie(value_1.values, labels=value_1.index, autopct="%.1f%%")

#Visualization to show Attrition Rate by Businees Travel.
plt.subplot(1,2,2)
new_df = df_attrition[df_attrition["Attrition"]==1]
value_2 = new_df["BusinessTravel"].value_counts()
attrition_rate = np.floor((value_2/value_1)*100).values
sns.barplot(x=value_2.index,y=value_2.values)
plt.title("Attrition Rate by Businees Travel")
for index,value in enumerate(value_2):
    plt.text(index,value,str(value)+" ("+str(attrition_rate[index])+"% )",ha="center")
plt.tight_layout()
plt.show()
```



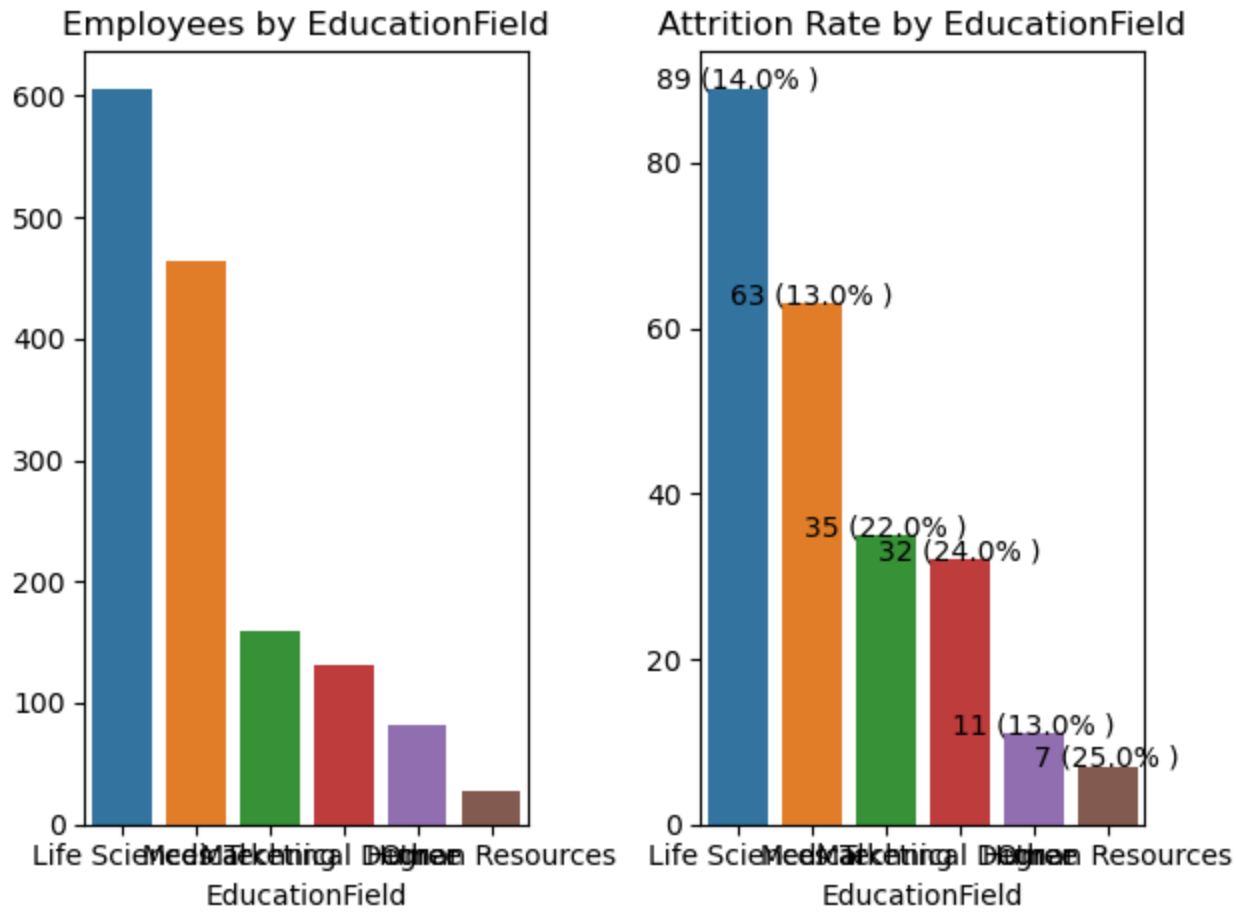
In [25]:
```
#Visualization to show Employee by Department.
plt.subplot(1,2,1)
value_1 = df_attrition["Department"].value_counts()
sns.barplot(x=value_1.index, y=value_1.values)
plt.title("Employees by Department")

#Visualization to show Employee Attrition Rate by Department.
plt.subplot(1,2,2)
new_df = df_attrition[df_attrition["Attrition"]==1]
value_2 = new_df["Department"].value_counts()
attrition_rate = np.floor((value_2/value_1)*100).values
sns.barplot(x=value_2.index, y=value_2.values)
plt.title("Attrition Rate by Department")
for index,value in enumerate(value_2):
    plt.text(index,value,str(value)+" ("+str(attrition_rate[index])+"% )",ha="center")
plt.tight_layout()
plt.show()
```



In [26]:
```
#Visualization to show Employee by Education.
plt.subplot(1,2,1)
value_1 = df_attrition["Education"].value_counts()
sns.barplot(x=value_1.index, y=value_1.values)
plt.title("Employees by Education")

#Visualization to show Employee Attrition Rate by Education.
plt.subplot(1,2,2)
new_df = df_attrition[df_attrition["Attrition"]==1]
value_2 = new_df["Education"].value_counts()
attrition_rate = np.floor((value_2/value_1)*100).values
sns.barplot(x=value_2.index, y=value_2.values)
plt.title("Attrition Rate by Education")
for index,value in enumerate(value_2):
    plt.text(index,value,str(value)+" ("+str(attrition_rate[index])+"% )",ha="center")
plt.tight_layout()
plt.show()
```



In [27]:
```
#Visualization to show Employee by EducationField.
plt.subplot(1,2,1)
value_1 = df_attrition["EducationField"].value_counts()
sns.barplot(x=value_1.index, y=value_1.values)
plt.title("Employees by EducationField")

#Visualization to show Employee Attrition Rate by EducationField.
plt.subplot(1,2,2)
new_df = df_attrition[df_attrition["Attrition"]==1]
value_2 = new_df["EducationField"].value_counts()
attrition_rate = np.floor((value_2/value_1)*100).values
sns.barplot(x=value_2.index, y=value_2.values)
plt.title("Attrition Rate by EducationField")
for index,value in enumerate(value_2):
    plt.text(index,value,str(value)+" ("+str(attrition_rate[index])+"% )",ha="center")
plt.tight_layout()
plt.show()
```



In [28]:
```
df_attrition["DistanceFromHome"].describe().to_frame().T
```

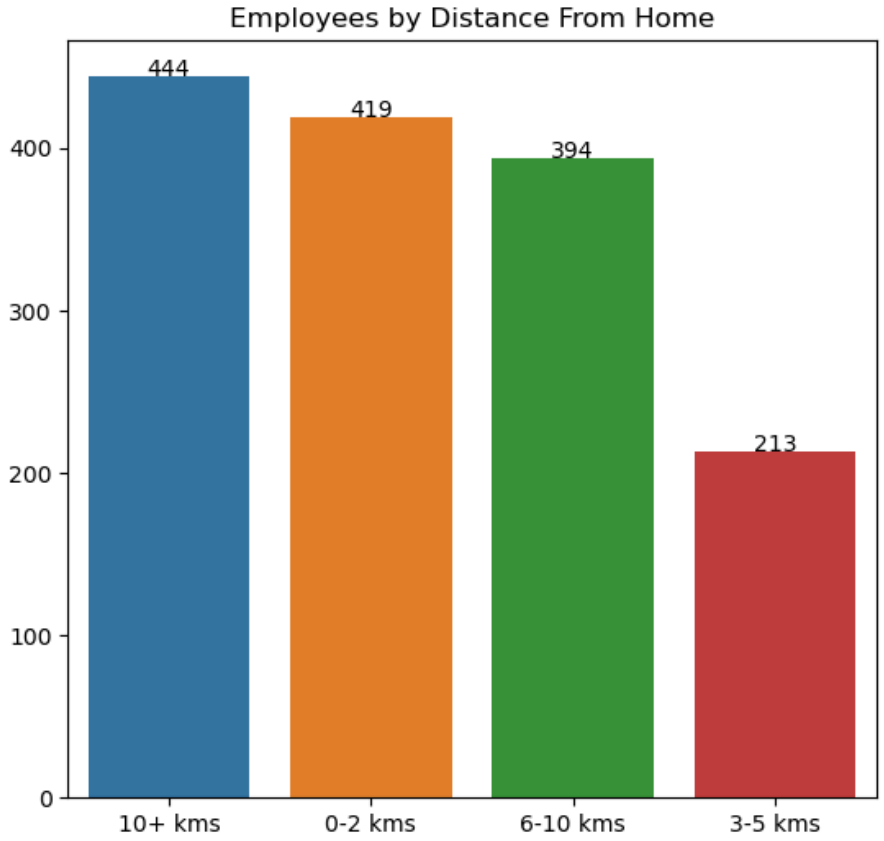
Out[28]:

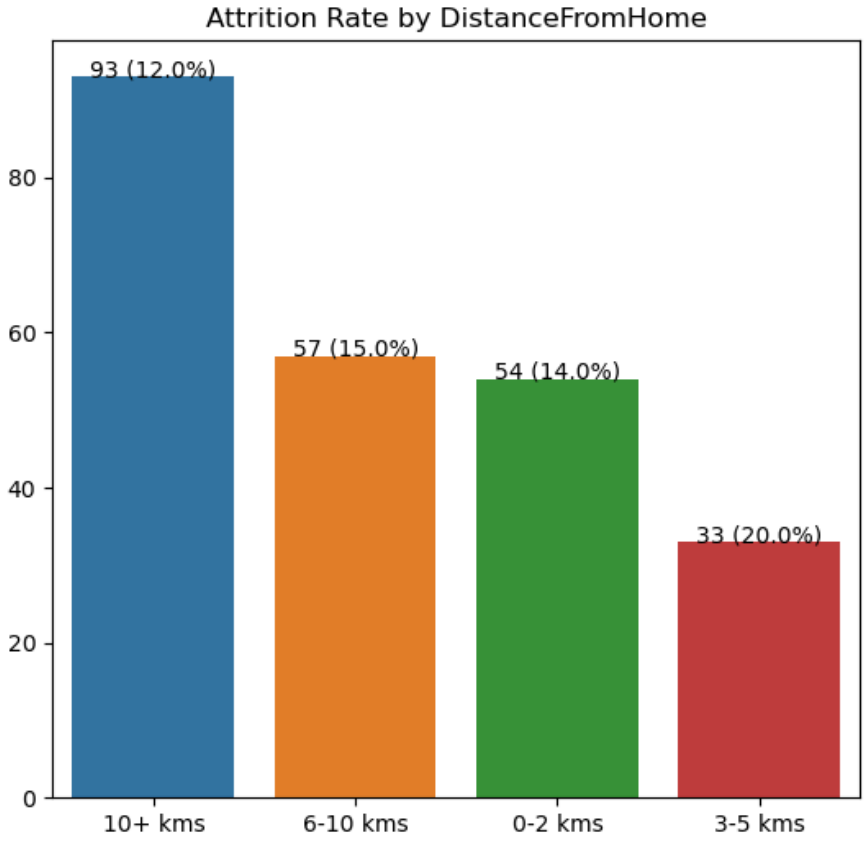
	count	mean	std	min	25%	50%	75%	max
DistanceFromHome	1470.0	9.192517	8.106864	1.0	2.0	7.0	14.0	29.0

In [29]:
```
# Define the bin edges for the groups
bin_edges = [0,2,5,10,50]

# Define the labels for the groups
bin_labels = ['0-2 kms', '3-5 kms', '6-10 kms',"10+ kms"]

# Cutting the DistanceFromHome column into groups
df_attrition['DistanceGroup'] = pd.cut(df_attrition['DistanceFromHome'], bins=bin_edges, labels=bin_labels)

df_attrition['DistanceGroup'].value_counts()
```

Out[29]:
```
DistanceGroup
10+ kms     444
0-2 kms     419
6-10 kms    394
3-5 kms     213
Name: count, dtype: int64
```

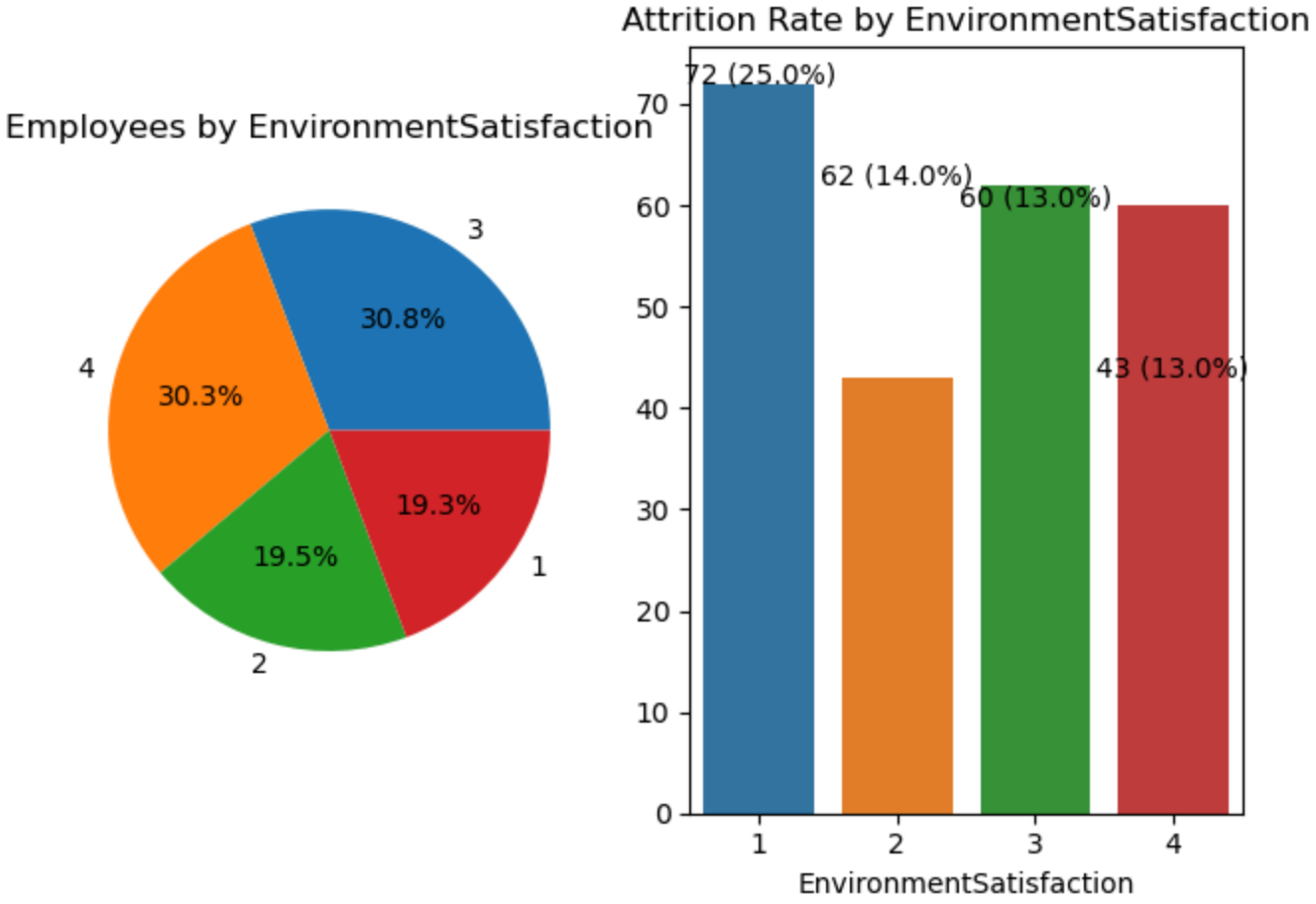
In [30]:
```
##Visualization to show Total Employees by DistnaceFromHome.
plt.figure(figsize=(14,6))
plt.subplot(1,2,1)
value_1 = df_attrition["DistanceGroup"].value_counts()
sns.barplot(x=value_1.index.tolist(), y=value_1.values)
plt.title("Employees by Distance From Home")
for index, value in enumerate(value_1.values):
    plt.text(index,value,value,ha="center")

 #Visualization to show Attrition Rate by DistanceFromHome.
plt.subplot(1,2,2)
new_df = df_attrition[df_attrition["Attrition"]==1]
value_2 = new_df["DistanceGroup"].value_counts()
attrition_rate = np.floor((value_2/value_1)*100).values
sns.barplot(x=value_2.index.tolist(),y= value_2.values)
plt.title("Attrition Rate by DistanceFromHome")
for index,value in enumerate(value_2.values):
    plt.text(index,value, str(value)+" ("+str(attrition_rate[index])+"%)",ha="center")

plt.show()
```





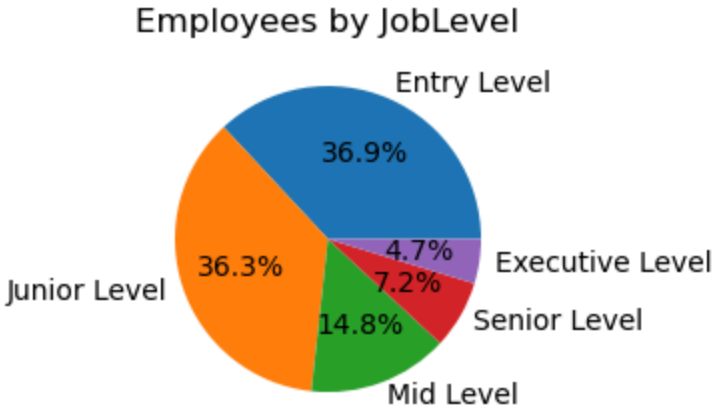
In [31]:

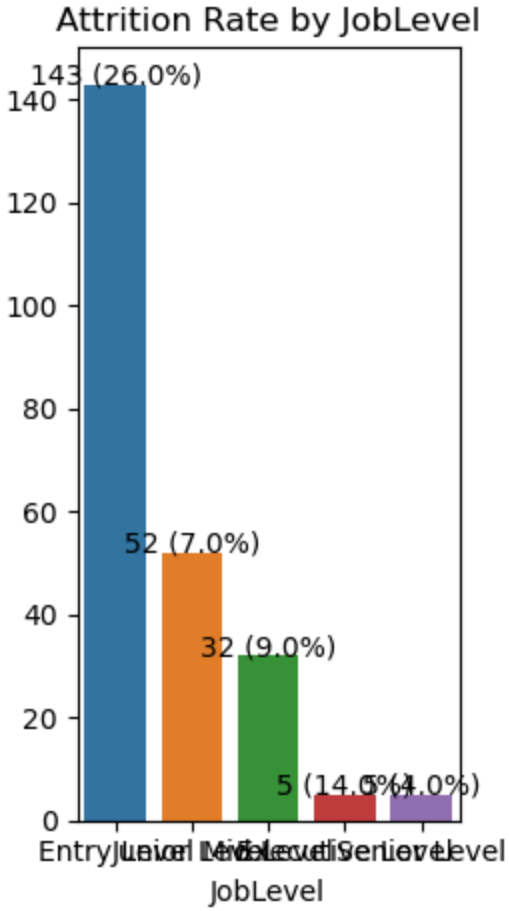
```
##Visualization to show Total Employees by EnvironmentSatisfaction.
plt.subplot(1,2,1)
value_1 = df_attrition["EnvironmentSatisfaction"].value_counts()
plt.pie(value_1.values, labels=value_1.index, autopct="%.1f%%")
plt.title("Employees by EnvironmentSatisfaction")

#Visualization to show Attrition Rate by EnvironmentSatisfaction.
plt.subplot(1,2,2)
new_df = df_attrition[df_attrition["Attrition"]==1]
value_2 = new_df["EnvironmentSatisfaction"].value_counts()
attrition_rate = np.floor((value_2/value_1)*100).values
sns.barplot(x=value_2.index,y= value_2.values)
plt.title("Attrition Rate by EnvironmentSatisfaction")
for index,value in enumerate(value_2.values):
    plt.text(index,value, str(value)+" ("+str(attrition_rate[index])+"%)",ha="center")
plt.tight_layout()
plt.show()
```



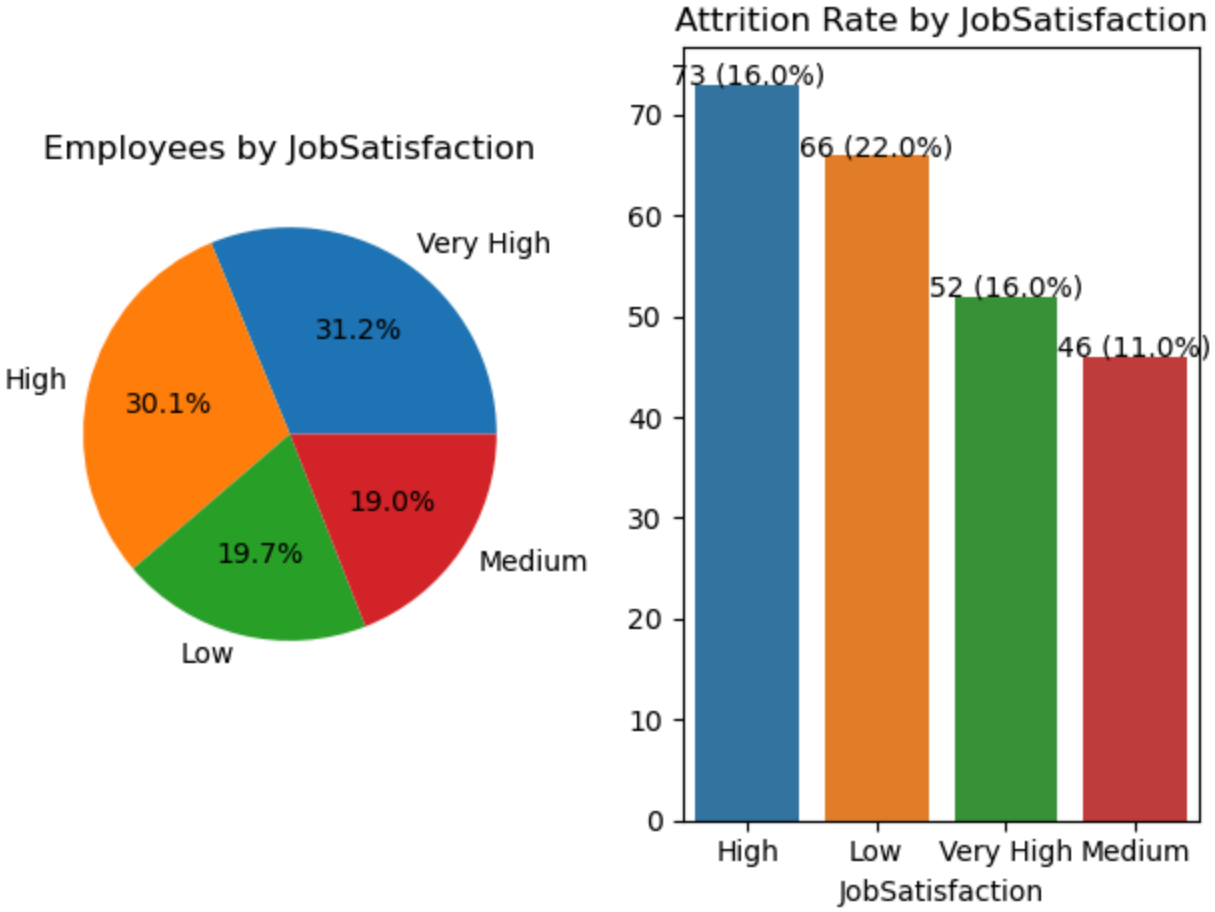
In [32]:

```
#Visualization to show Total Employees by JobLevel.
plt.subplot(1,2,1)
value_1 = df_attrition["JobLevel"].value_counts()
plt.pie(value_1.values, labels=value_1.index, autopct="%.1f%%")
plt.title("Employees by JobLevel")

#Visualization to show Attrition Rate by JobLevel.
plt.subplot(1,2,2)
new_df = df_attrition[df_attrition["Attrition"]==1]
value_2 = new_df["JobLevel"].value_counts()
attrition_rate = np.floor((value_2/value_1)*100).values
sns.barplot(x=value_2.index,y= value_2.values)
plt.title("Attrition Rate by JobLevel")
for index,value in enumerate(value_2.values):
    plt.text(index,value, str(value)+" ("+str(attrition_rate[index])+"%)",ha="center")
plt.tight_layout()
plt.show()
```



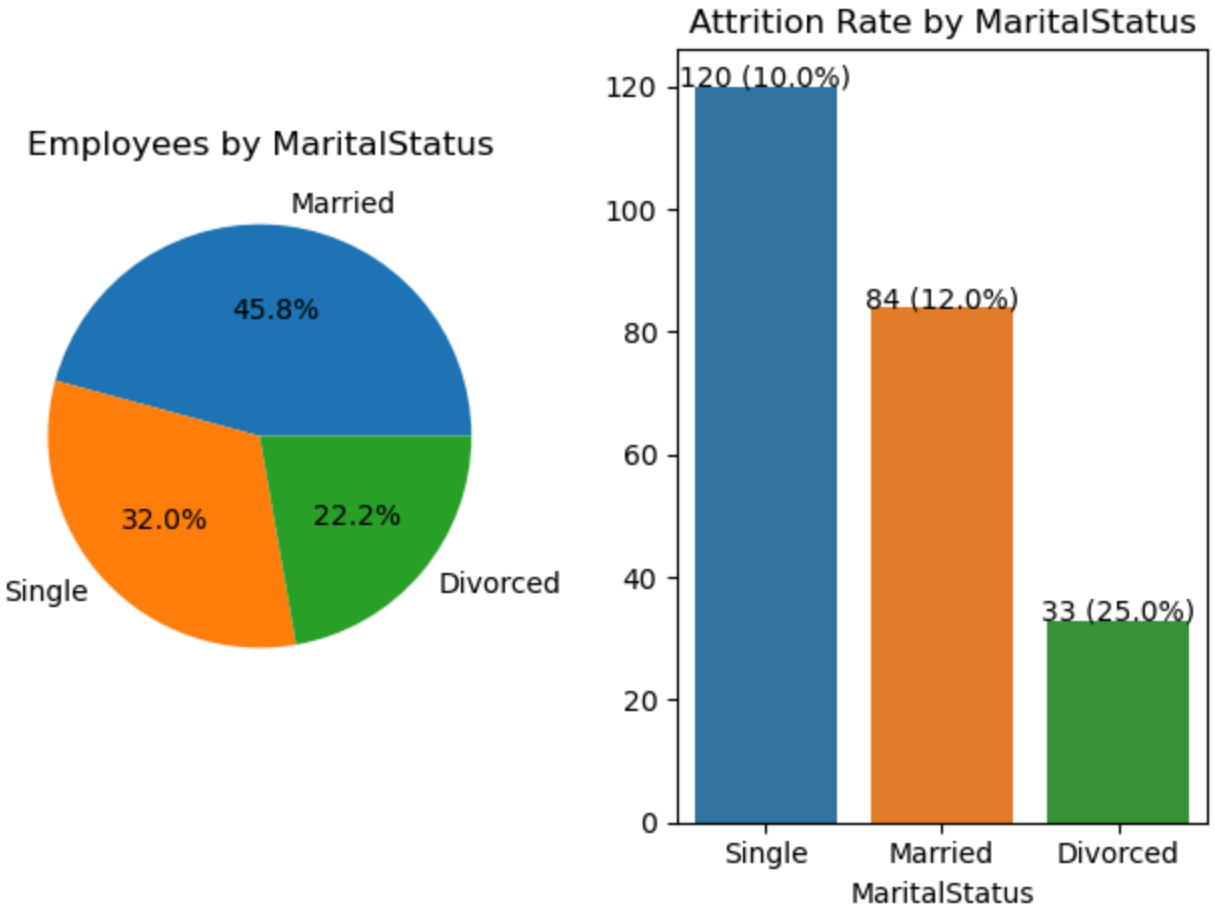


In [33]:
```
#Visualization to show Total Employees by JobSatisfaction.
plt.subplot(1,2,1)
value_1 = df_attrition["JobSatisfaction"].value_counts()
plt.pie(value_1.values, labels=value_1.index, autopct="%.1f%%")
plt.title("Employees by JobSatisfaction")

#Visualization to show Attrition Rate by JobSatisfaction.
plt.subplot(1,2,2)
new_df = df_attrition[df_attrition["Attrition"]==1]
value_2 = new_df["JobSatisfaction"].value_counts()
attrition_rate = np.floor((value_2/value_1)*100).values
sns.barplot(x=value_2.index,y= value_2.values)
plt.title("Attrition Rate by JobSatisfaction")
for index,value in enumerate(value_2.values):
    plt.text(index,value, str(value)+" ("+str(attrition_rate[index])+"%)",ha="center")
plt.tight_layout()
plt.show()
```



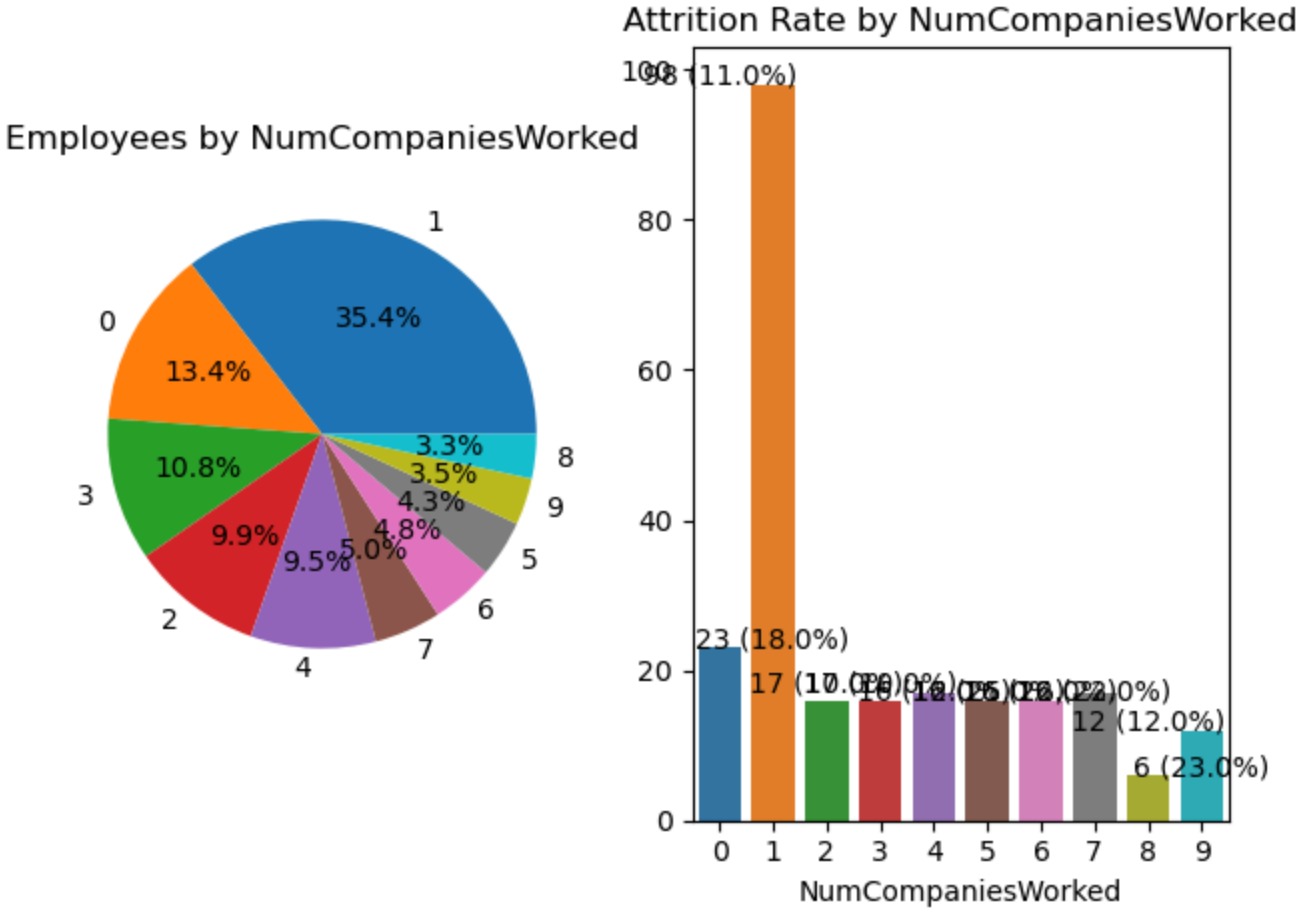
In [34]:

```
#Visualization to show Total Employees by MaritalStatus.
plt.subplot(1,2,1)
value_1 = df_attrition["MaritalStatus"].value_counts()
plt.pie(value_1.values, labels=value_1.index, autopct="%.1f%%")
plt.title("Employees by MaritalStatus")

#Visualization to show Attrition Rate by MaritalStatus.
plt.subplot(1,2,2)
new_df = df_attrition[df_attrition["Attrition"]==1]
value_2 = new_df["MaritalStatus"].value_counts()
attrition_rate = np.floor((value_2/value_1)*100).values
sns.barplot(x=value_2.index,y= value_2.values)
plt.title("Attrition Rate by MaritalStatus")
for index,value in enumerate(value_2.values):
    plt.text(index,value, str(value)+" ("+str(attrition_rate[index])+"%)",ha="center")
plt.tight_layout()
plt.show()
```

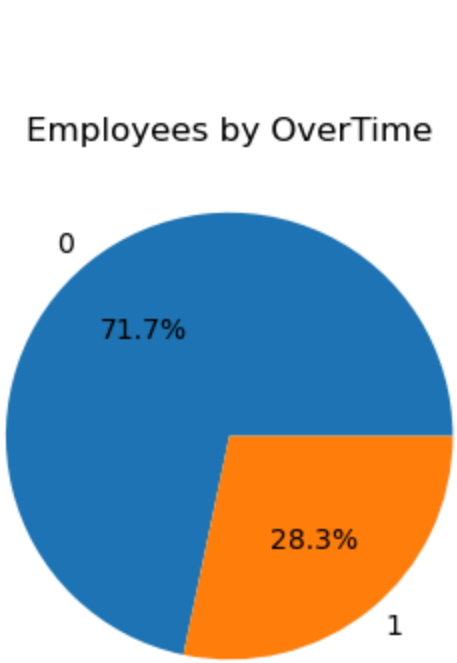


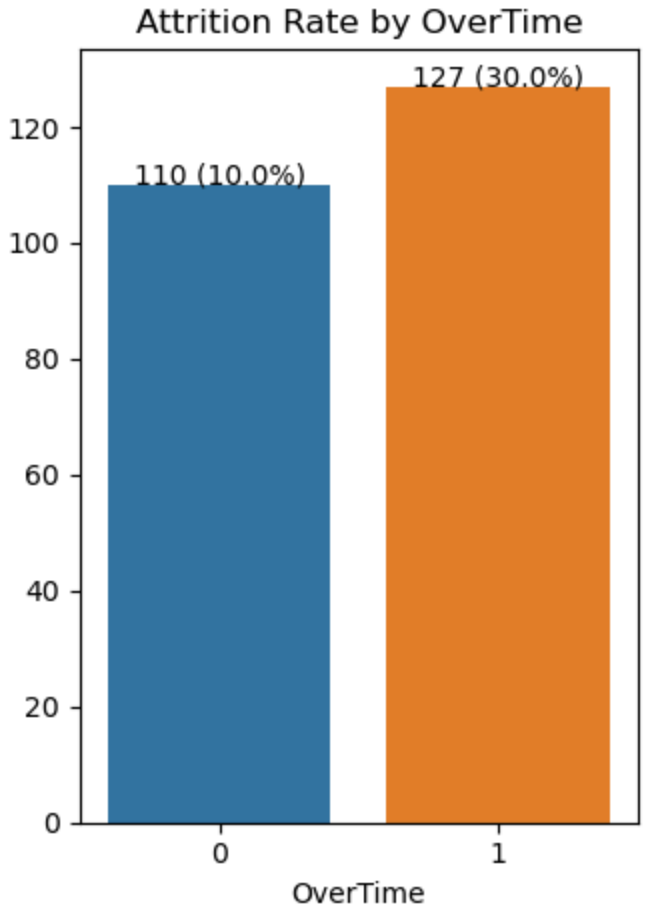
In [35]:

```
#Visualization to show Total Employees by NumCompaniesWorked.
plt.subplot(1,2,1)
value_1 = df_attrition["NumCompaniesWorked"].value_counts()
plt.pie(value_1.values, labels=value_1.index, autopct="%.1f%%")
plt.title("Employees by NumCompaniesWorked")

#Visualization to show Attrition Rate by NumCompaniesWorked.
plt.subplot(1,2,2)
new_df = df_attrition[df_attrition["Attrition"]==1]
value_2 = new_df["NumCompaniesWorked"].value_counts()
attrition_rate = np.floor((value_2/value_1)*100).values
sns.barplot(x=value_2.index,y= value_2.values)
plt.title("Attrition Rate by NumCompaniesWorked")
for index,value in enumerate(value_2.values):
    plt.text(index,value, str(value)+" ("+str(attrition_rate[index])+"%)",ha="center")
plt.tight_layout()
plt.show()
```

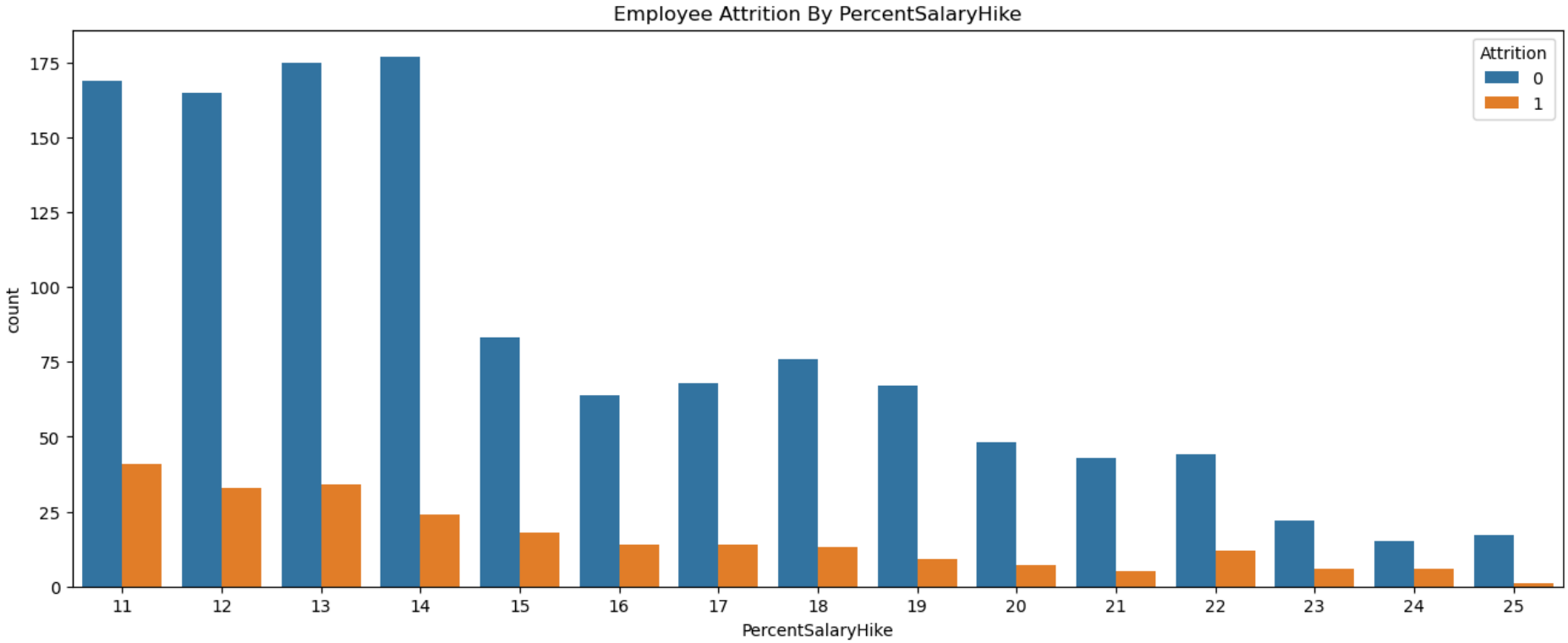


In [36]:
```
#Visualization to show Total Employees by OverTime.
plt.subplot(1,2,1)
value_1 = df_attrition["OverTime"].value_counts()
plt.pie(value_1.values, labels=value_1.index, autopct="%.1f%%")
plt.title("Employees by OverTime")

#Visualization to show Attrition Rate by OverTime.
plt.subplot(1,2,2)
new_df = df_attrition[df_attrition["Attrition"]==1]
value_2 = new_df["OverTime"].value_counts()
attrition_rate = np.floor((value_2/value_1)*100).values
sns.barplot(x=value_2.index,y= value_2.values)
cnt = len(pd.unique(df_attrition['OverTime']))
plt.title("Attrition Rate by OverTime")
for index,value in enumerate(value_2.values):
    plt.text(cnt-index-1,value, str(value)+" ("+str(attrition_rate[cnt-index-1])+"%)",ha="center")
plt.tight_layout()
plt.show()
```



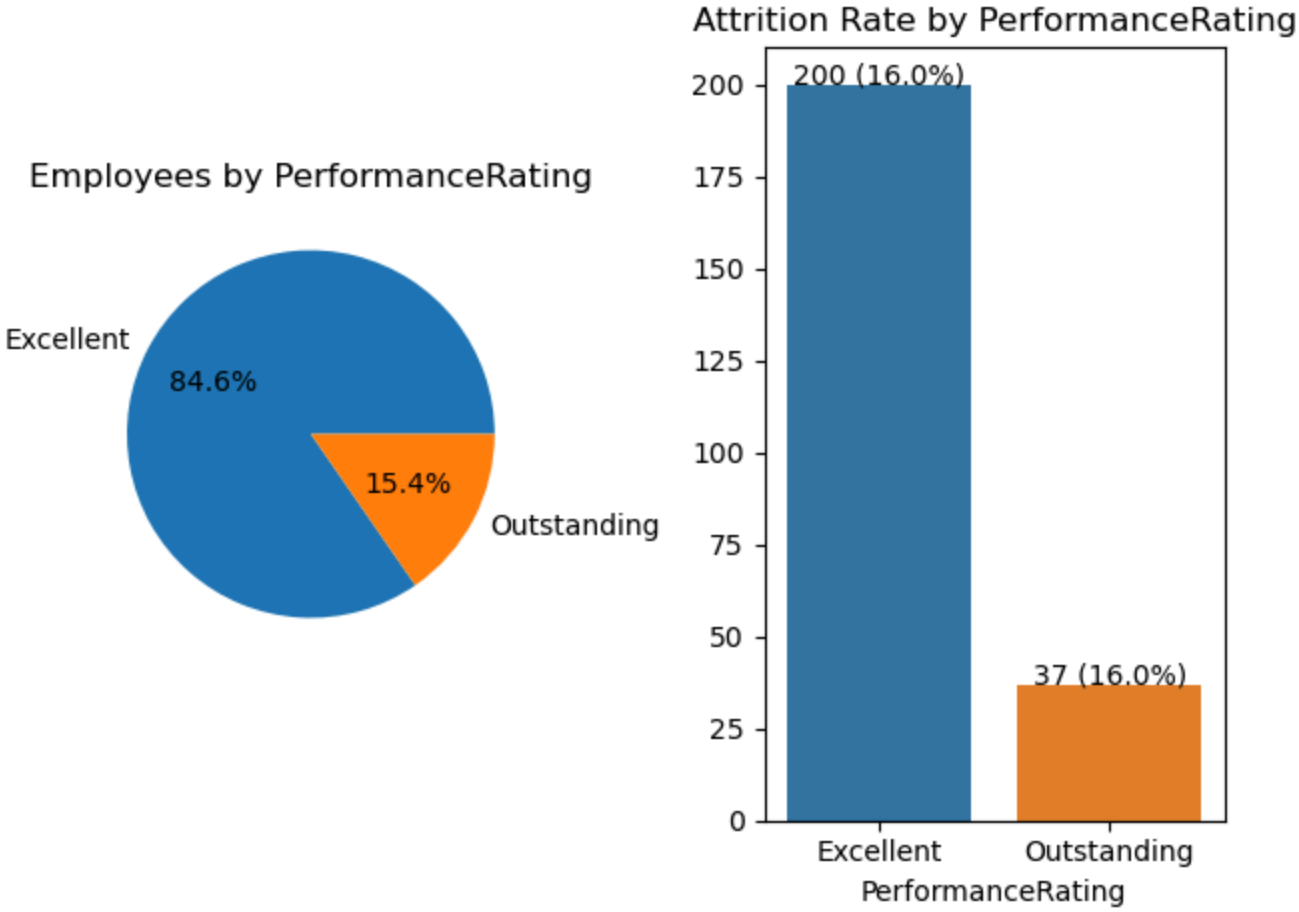


In [37]:
```
#Visualization to show Employee Distribution by Percentage Salary Hike.
plt.figure(figsize=(16,6))
sns.countplot(x="PercentSalaryHike", hue="Attrition", data=df_attrition)
plt.title("Employee Attrition By PercentSalaryHike")
plt.show()
```



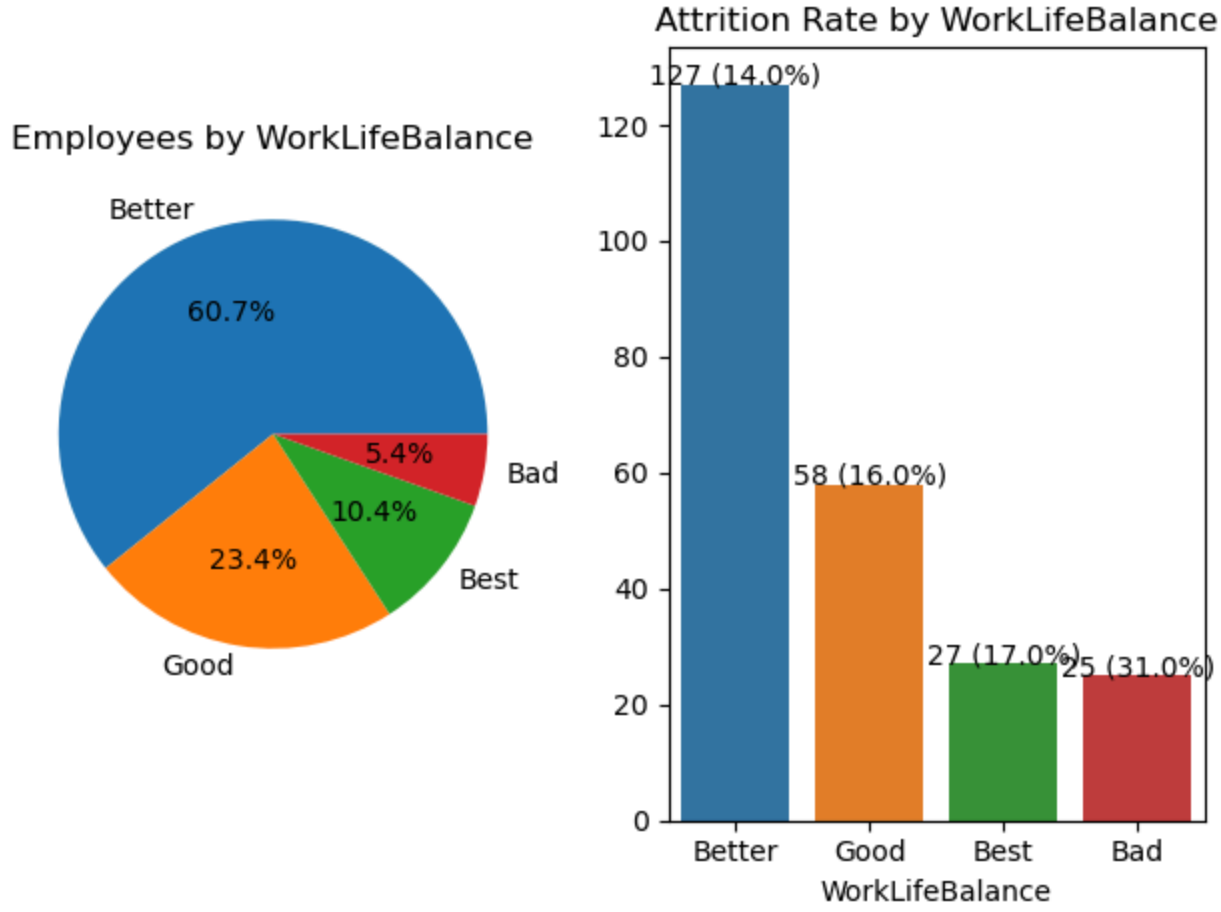
In [38]:

```
#Visualization to show Total Employees by PerformanceRating.
plt.subplot(1,2,1)
value_1 = df_attrition["PerformanceRating"].value_counts()
plt.pie(value_1.values, labels=value_1.index, autopct="%.1f%%")
plt.title("Employees by PerformanceRating")

#Visualization to show Attrition Rate by PerformanceRating.
plt.subplot(1,2,2)
new_df = df_attrition[df_attrition["Attrition"]==1]
value_2 = new_df["PerformanceRating"].value_counts()
attrition_rate = np.floor((value_2/value_1)*100).values
sns.barplot(x=value_2.index,y= value_2.values)
plt.title("Attrition Rate by PerformanceRating")
for index,value in enumerate(value_2.values):
    plt.text(index,value, str(value)+" ("+str(attrition_rate[index])+"%)",ha="center")
plt.tight_layout()
plt.show()
```



In [39]:

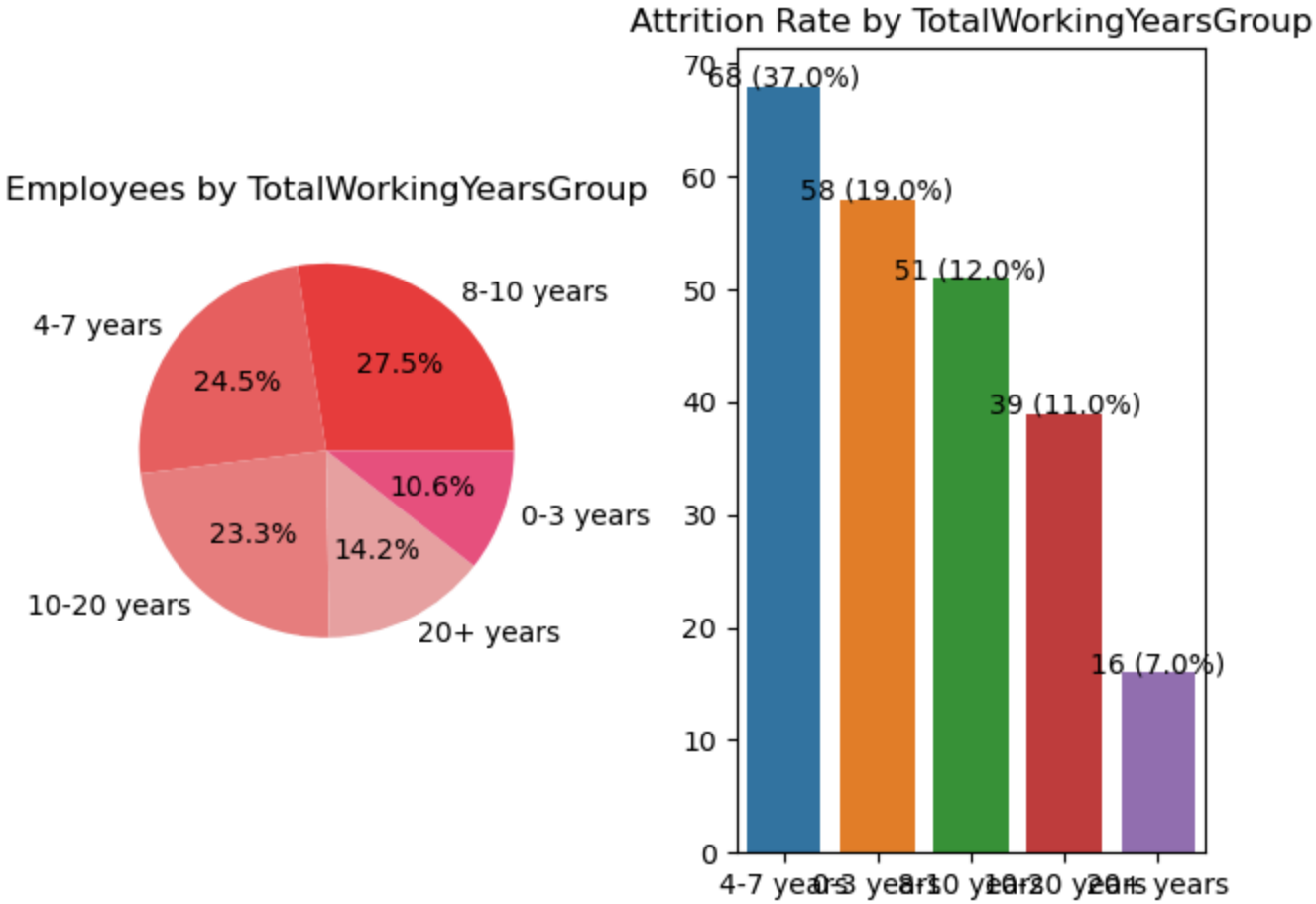
```
#Visualization to show Total Employees by WorkLifeBalance.
plt.subplot(1,2,1)
value_1 = df_attrition["WorkLifeBalance"].value_counts()
plt.pie(value_1.values, labels=value_1.index, autopct="%.1f%%")
plt.title("Employees by WorkLifeBalance")

#Visualization to show Attrition Rate by WorkLifeBalance.
plt.subplot(1,2,2)
new_df = df_attrition[df_attrition["Attrition"]==1]
value_2 = new_df["WorkLifeBalance"].value_counts()
attrition_rate = np.floor((value_2/value_1)*100).values
sns.barplot(x=value_2.index,y= value_2.values)
plt.title("Attrition Rate by WorkLifeBalance")
for index,value in enumerate(value_2.values):
    plt.text(index,value, str(value)+" ("+str(attrition_rate[index])+"%)",ha="center")
plt.tight_layout()
plt.show()
```



In [40]:
```
# Define the bin edges for the groups
bin_group = [0, 3, 7, 10, 20, 50]

# Define the labels for the groups
bin_labels = ['0-3 years','4-7 years', '8-10 years', '10-20 years', "20+ years"]

# Cut the DailyRate column into groups
df_attrition["TotalWorkingYearsGroup"] = pd.cut(df_attrition['TotalWorkingYears'], bins=bin_group, labels=bin_labels)
```

In [41]:
```
#Visualization to show Total Employees by TotalWorkingYears.
plt.subplot(1,2,1)
value_1 = df_attrition["TotalWorkingYearsGroup"].value_counts()
plt.pie(value_1.values, labels=value_1.index, autopct="%.1f%%",colors=['#E84040', '#E96060', '#E88181', '#E7A1A1','#E85181'])
plt.title("Employees by TotalWorkingYearsGroup")

#Visualization to show Attrition Rate by TotalWorkingYears.
plt.subplot(1,2,2)
new_df = df_attrition[df_attrition["Attrition"]==1]
value_2 = new_df["TotalWorkingYearsGroup"].value_counts()
```

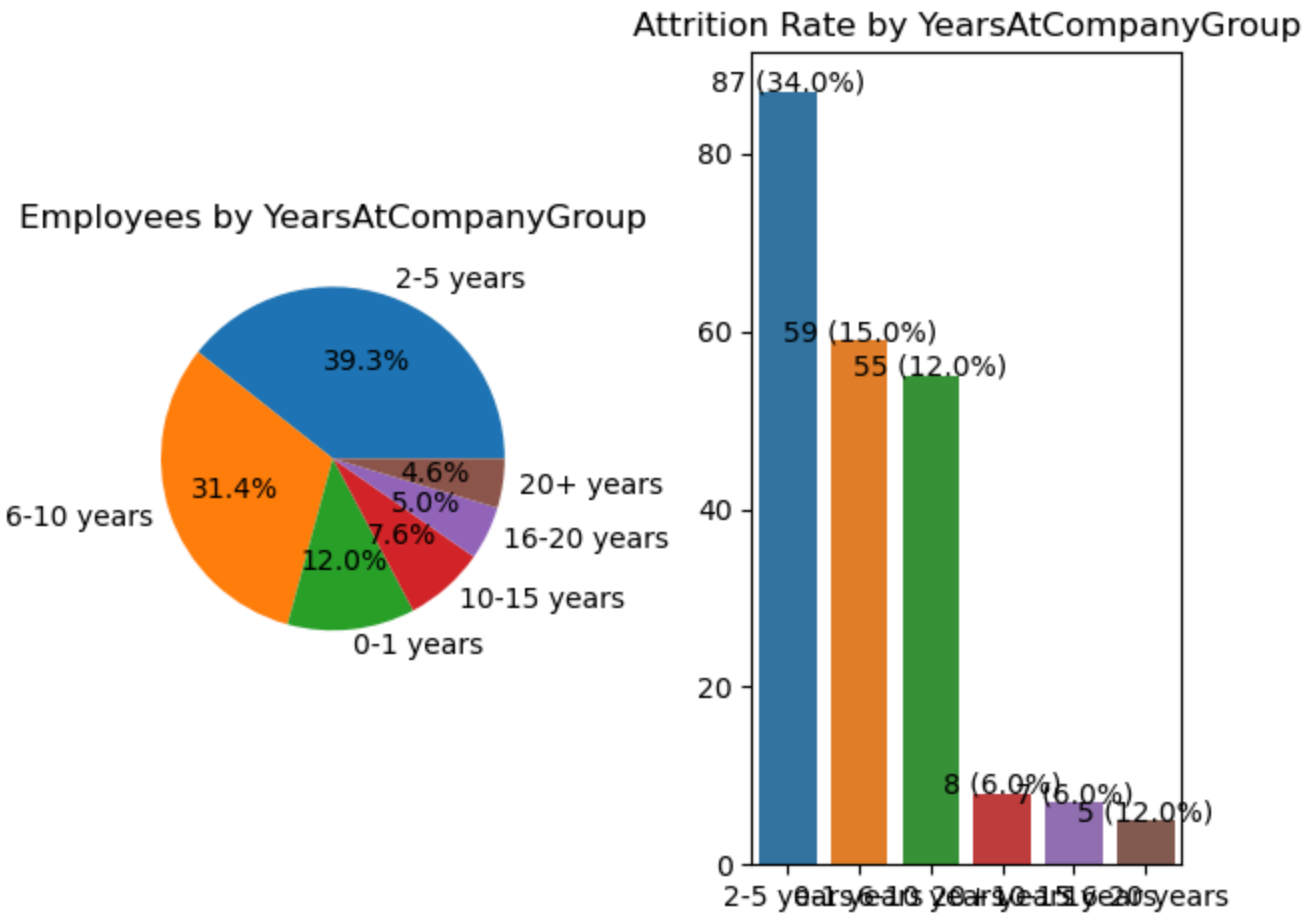
```
attrition_rate = np.floor((value_2/value_1)*100).values
sns.barplot(x=value_2.index.tolist(),y= value_2.values)
plt.title("Attrition Rate by TotalWorkingYearsGroup")
for index,value in enumerate(value_2.values):
    plt.text(index,value, str(value)+" ("+str(attrition_rate[index])+"%)",ha="center")
plt.tight_layout()
plt.show()
```



In [42]:
```
# Define the bin edges for the groups
bin_group = [0, 1, 5, 10, 15, 20, 50]

# Define the labels for the groups
bin_labels = ['0-1 years', '2-5 years', '6-10 years', '10-15 years', '16-20 years', "20+ years"]

# Cut the DailyRate column into groups
df_attrition["YearsAtCompanyGroup"] = pd.cut(df_attrition['YearsAtCompany'], bins=bin_group, labels=bin_labels)
```

In [43]:
```
#Visualization to show Total Employees by YearsAtCompanyGroup.
plt.subplot(1,2,1)
```

```
value_1 = df_attrition["YearsAtCompanyGroup"].value_counts()
plt.pie(value_1.values, labels=value_1.index, autopct="%.1f%%")
plt.title("Employees by YearsAtCompanyGroup")

#Visualization to show Attrition Rate by TotalWorkingYears.
plt.subplot(1,2,2)
new_df = df_attrition[df_attrition["Attrition"]==1]
value_2 = new_df["YearsAtCompanyGroup"].value_counts()
attrition_rate = np.floor((value_2/value_1)*100).values
sns.barplot(x=value_2.index.tolist(),y= value_2.values)
plt.title("Attrition Rate by YearsAtCompanyGroup")
for index,value in enumerate(value_2.values):
    plt.text(index,value, str(value)+" ("+str(attrition_rate[index])+"%)",ha="center")
plt.tight_layout()
plt.show()
```

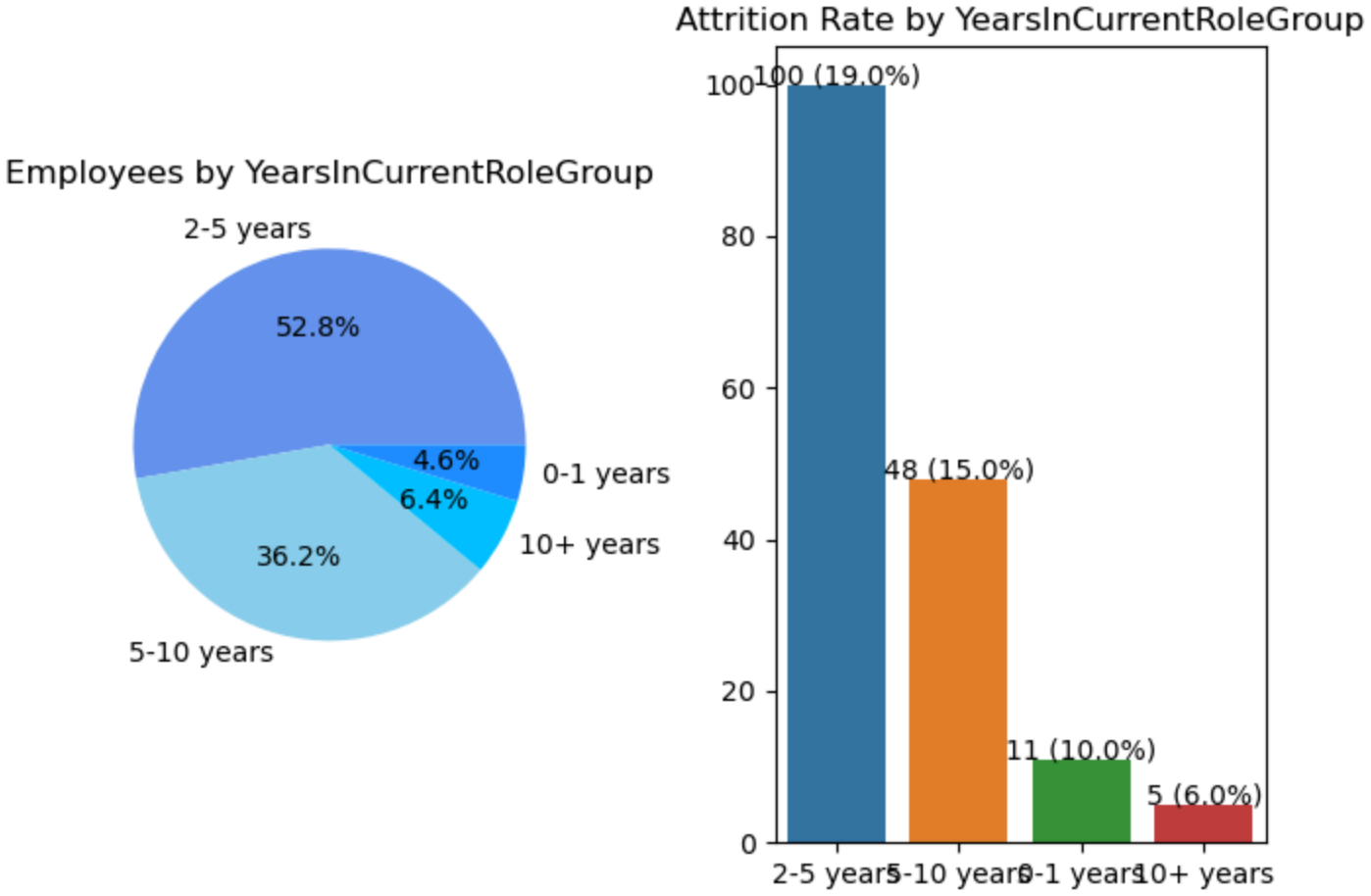


In [44]:
```
# Define the bin edges for the groups
bin_edges = [0, 1, 5, 10, 20]

# Define the labels for the groups
```

```
bin_labels = ['0-1 years', '2-5 years', '5-10 years', "10+ years"]

# Cut the DailyRate column into groups
df_attrition["YearsInCurrentRoleGroup"] = pd.cut(df_attrition['YearsInCurrentRole'], bins=bin_edges, labels=bin_labels)
```

In [45]:

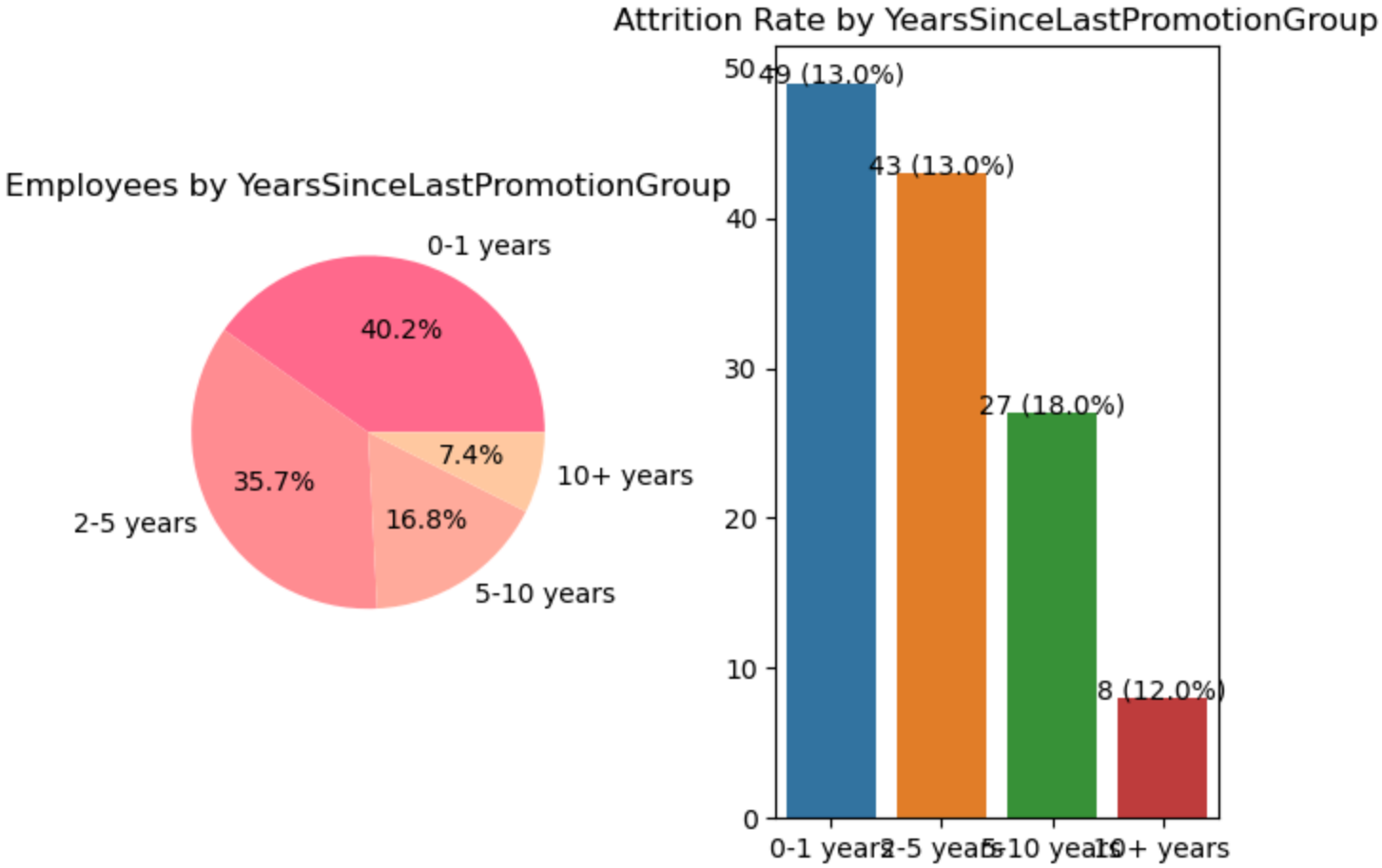
```
#Visualization to show Total Employees by YearsInCurrentRoleGroup.
plt.subplot(1,2,1)
value_1 = df_attrition["YearsInCurrentRoleGroup"].value_counts()
plt.pie(value_1.values, labels=value_1.index, autopct="%.1f%%",colors=['#6495ED', '#87CEEB', '#00BFFF', '#1E90FF'])
plt.title("Employees by YearsInCurrentRoleGroup")

#Visualization to show Attrition Rate by YearsInCurrentRoleGroup.
plt.subplot(1,2,2)
new_df = df_attrition[df_attrition["Attrition"]==1]
value_2 = new_df["YearsInCurrentRoleGroup"].value_counts()
attrition_rate = np.floor((value_2/value_1)*100).values
sns.barplot(x=value_2.index.tolist(),y= value_2.values)
plt.title("Attrition Rate by YearsInCurrentRoleGroup")
for index,value in enumerate(value_2.values):
    plt.text(index,value, str(value)+" ("+str(attrition_rate[index])+"%)",ha="center")
plt.tight_layout()
plt.show()
```



In [46]:
```
# Define the bin edges for the groups
bin_edges = [0, 1, 5, 10, 20]

# Define the labels for the groups
bin_labels = ['0-1 years', '2-5 years', '5-10 years', "10+ years"]

# Cut the DailyRate column into groups
df_attrition["YearsSinceLastPromotionGroup"] = pd.cut(df_attrition['YearsSinceLastPromotion'], bins=bin_edges, labels=bin_labels)
```

In [47]:
```
#Visualization to show Total Employees by YearsSinceLastPromotionGroup.
plt.subplot(1,2,1)
value_1 = df_attrition["YearsSinceLastPromotionGroup"].value_counts()
plt.pie(value_1.values, labels=value_1.index, autopct="%.1f%%",colors=['#FF6D8C', '#FF8C94', '#FFAC9B', '#FFCBA4'])
plt.title("Employees by YearsSinceLastPromotionGroup")

#Visualization to show Attrition Rate by YearsSinceLastPromotionGroup.
plt.subplot(1,2,2)
new_df = df_attrition[df_attrition["Attrition"]==1]
value_2 = new_df["YearsSinceLastPromotionGroup"].value_counts()
```

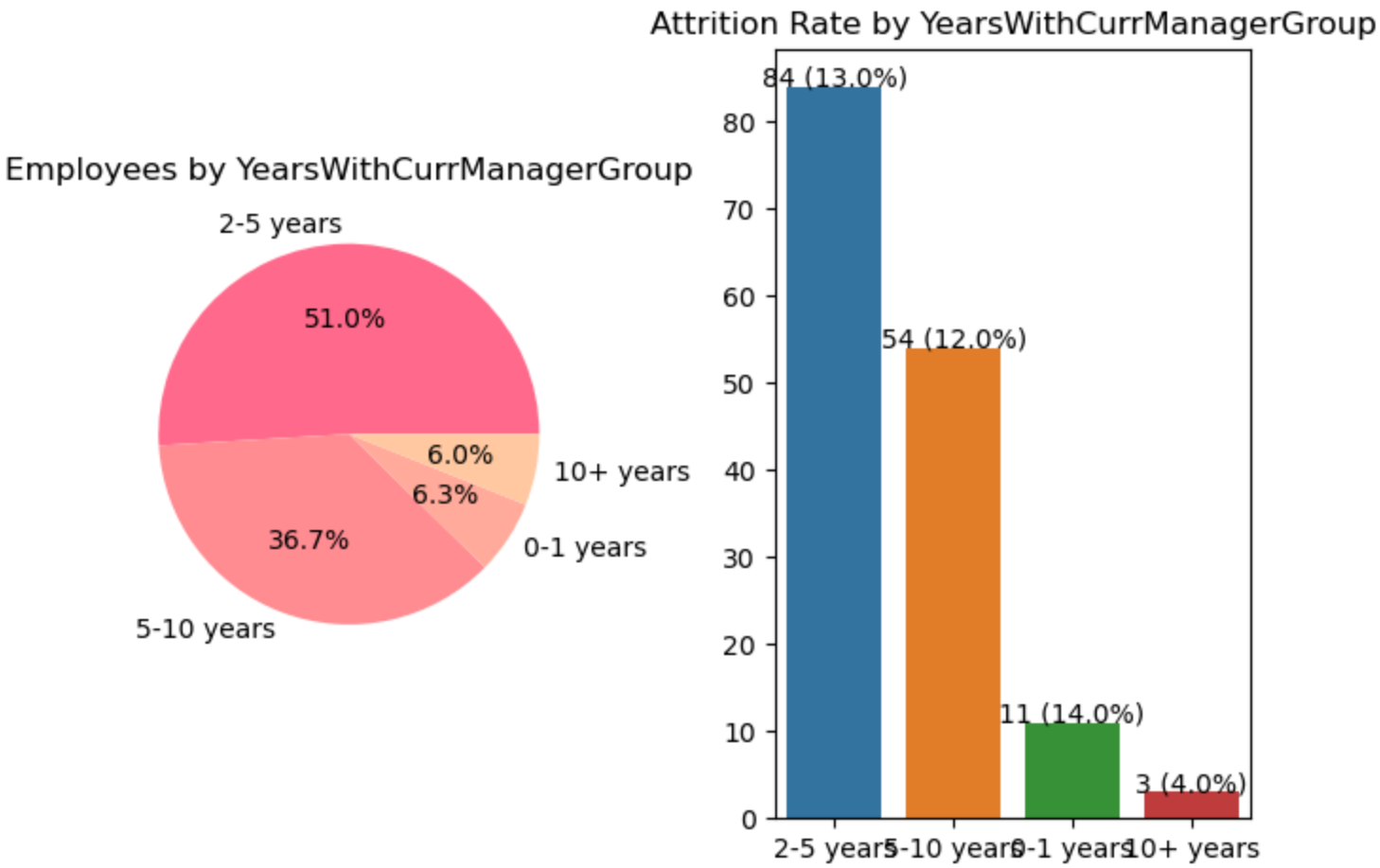
```
attrition_rate = np.floor((value_2/value_1)*100).values
sns.barplot(x=value_2.index.tolist(),y= value_2.values)
plt.title("Attrition Rate by YearsSinceLastPromotionGroup")
for index,value in enumerate(value_2.values):
    plt.text(index,value, str(value)+" ("+str(attrition_rate[index])+"%)",ha="center")
plt.tight_layout()
plt.show()
```



In [48]:
```
# Define the bin edges for the groups
bin_edges = [0, 1, 5, 10, 20]

# Define the labels for the groups
bin_labels = ['0-1 years', '2-5 years', '5-10 years', "10+ years"]

# Cut the DailyRate column into groups
df_attrition["YearsWithCurrManagerGroup"] = pd.cut(df_attrition['YearsWithCurrManager'], bins=bin_edges, labels=bin_labels)
```

In [49]:
```
#Visualization to show Total Employees by YearsWithCurrManagerGroup.
plt.subplot(1,2,1)
```

```
value_1 = df_attrition["YearsWithCurrManagerGroup"].value_counts()
plt.pie(value_1.values, labels=value_1.index, autopct="%.1f%%",colors=['#FF6D8C', '#FF8C94', '#FFAC9B', '#FFCBA4'])
plt.title("Employees by YearsWithCurrManagerGroup")

#Visualization to show Attrition Rate by YearsWithCurrManagerGroup.
plt.subplot(1,2,2)
new_df = df_attrition[df_attrition["Attrition"]==1]
value_2 = new_df["YearsWithCurrManagerGroup"].value_counts()
attrition_rate = np.floor((value_2/value_1)*100).values
sns.barplot(x=value_2.index.tolist(),y= value_2.values)
plt.title("Attrition Rate by YearsWithCurrManagerGroup")
for index,value in enumerate(value_2.values):
    plt.text(index,value, str(value)+" ("+str(attrition_rate[index])+"%)",ha="center")
plt.tight_layout()
plt.show()
```



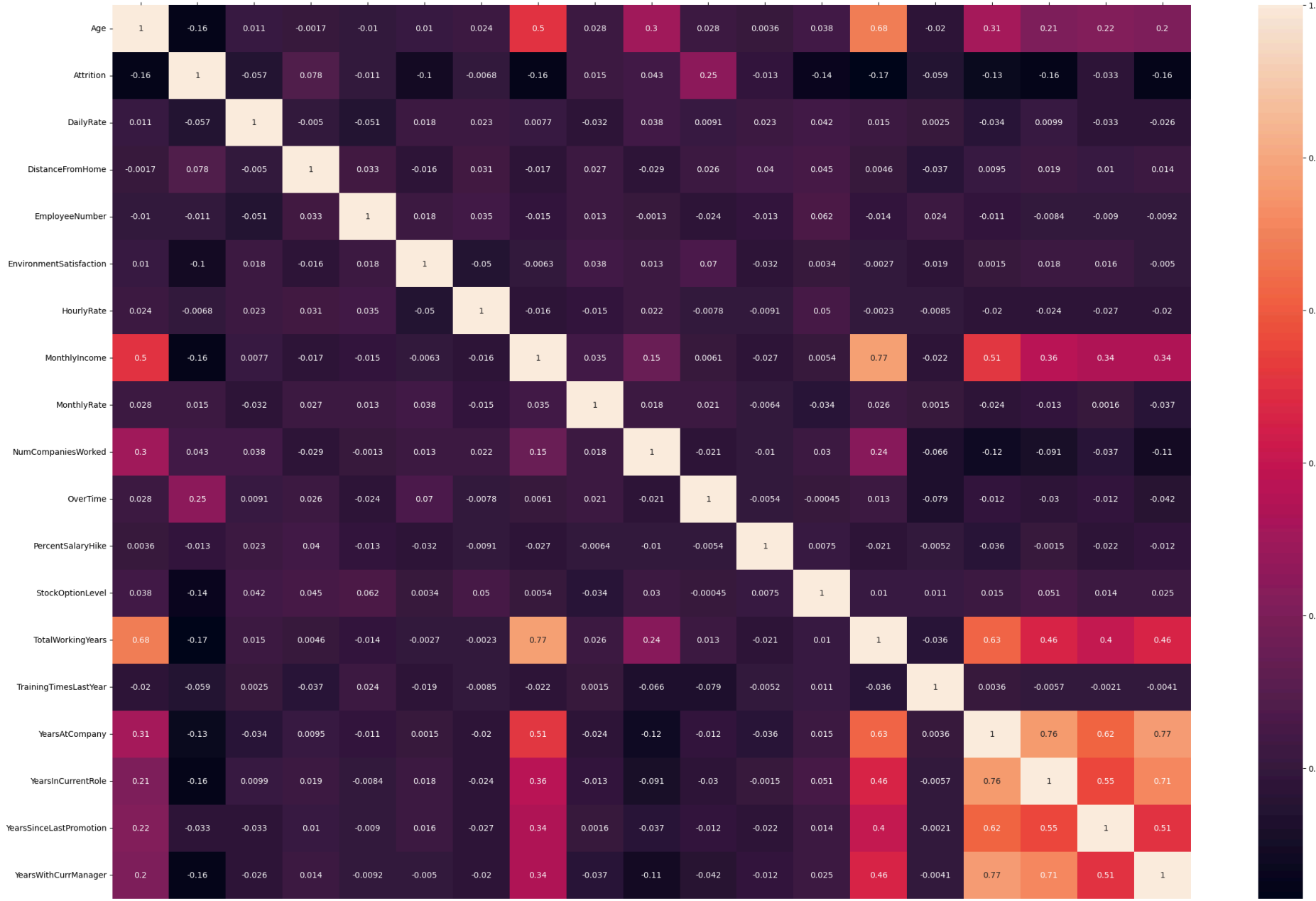
Correlation of Features

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 [50]:
```
#creating a list of only numerical values
df_attrition_numeric = df_attrition.select_dtypes(include=['int64'])
df_attrition_numeric.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 19 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Age                       1470 non-null   int64
 1   Attrition                 1470 non-null   int64
 2   DailyRate                 1470 non-null   int64
 3   DistanceFromHome          1470 non-null   int64
 4   EmployeeNumber            1470 non-null   int64
 5   EnvironmentSatisfaction   1470 non-null   int64
 6   HourlyRate                1470 non-null   int64
 7   MonthlyIncome             1470 non-null   int64
 8   MonthlyRate               1470 non-null   int64
 9   NumCompaniesWorked        1470 non-null   int64
 10  OverTime                  1470 non-null   int64
 11  PercentSalaryHike         1470 non-null   int64
 12  StockOptionLevel          1470 non-null   int64
 13  TotalWorkingYears         1470 non-null   int64
 14  TrainingTimesLastYear     1470 non-null   int64
 15  YearsAtCompany            1470 non-null   int64
 16  YearsInCurrentRole        1470 non-null   int64
 17  YearsSinceLastPromotion   1470 non-null   int64
 18  YearsWithCurrManager      1470 non-null   int64
dtypes: int64(19)
memory usage: 218.3 KB
```

In [51]:
```
# Calculate the correlation matrix
correlation_matrix = df_attrition_numeric.astype(float).corr()
plt.figure(figsize=(30, 20))
fig = sns.heatmap(correlation_matrix,annot=True)
fig.set(xlabel='',ylabel='')
fig.xaxis.tick_top()
```


Age
Attrition
DailyRate
DistanceFromHome
EmployeeNumber
EnvironmentSatisfaction
HourlyRate
MonthlyIncome
MonthlyRate
NumCompaniesWorked
OverTime
PercentSalaryHike
StockOptionLevel
TotalWorkingYears
TrainingTimesLastYear
YearsAtCompany
YearsInCurrentRole
YearsSinceLastPromotion
YearsWithCurrManager
1.0
0.8
0.6
0.4
0.2
0.0

Analysis : From the correlation plots, Most of the columns seem to be poorly correlated with one another. Generally when making a predictive model, it would be preferable to train a model with features that are not too correlated with one another so that we do not need to deal with redundant features. In the case that we have quite a lot of correlated features one could perhaps apply a technique such as Principal Component Analysis (PCA) to reduce the feature space.

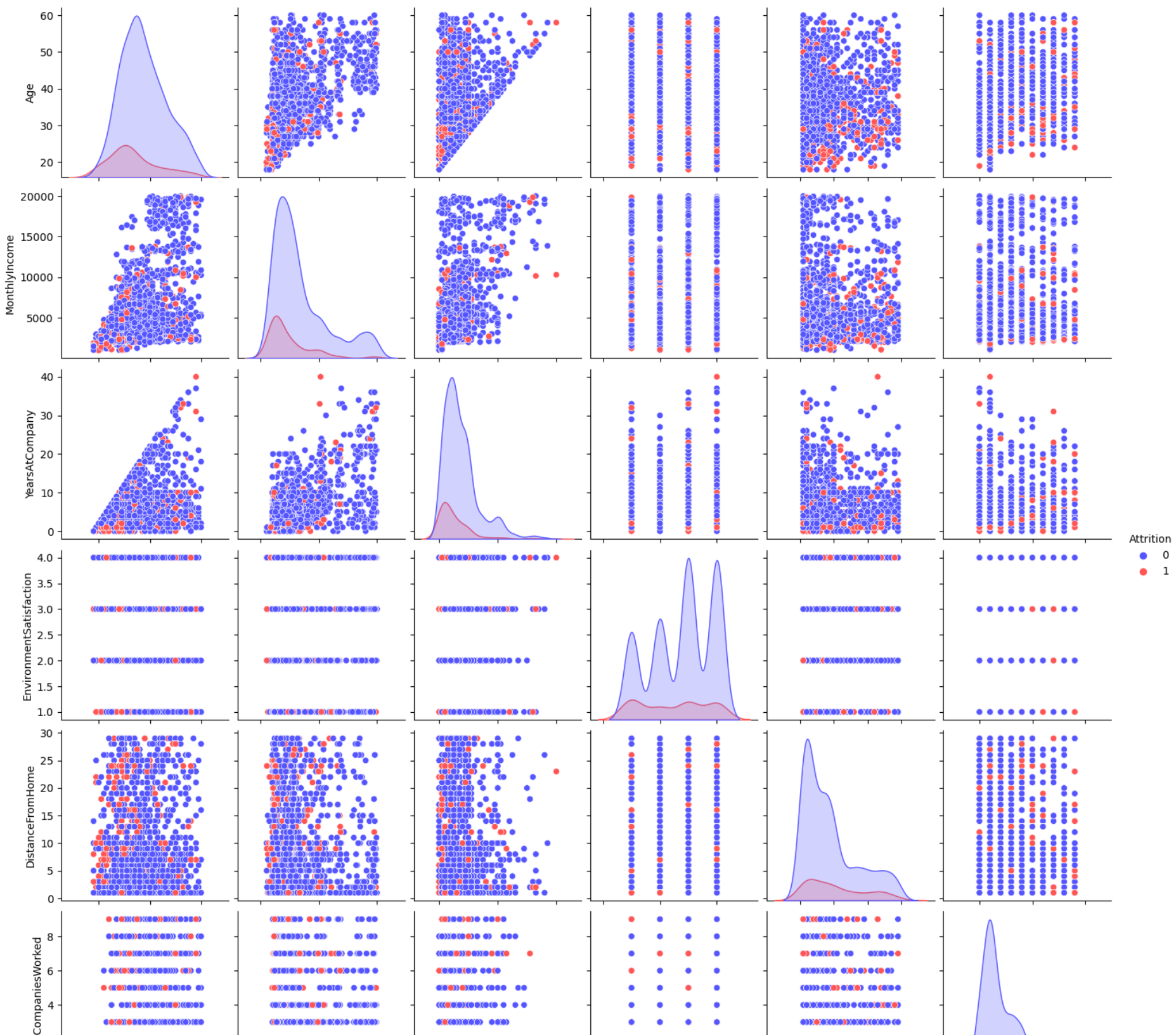
Pairplot Visualisations Now let us create some Seaborn pairplots and set it against the target variable which is our Attrition column to get a feel for how the various features are distributed vis-a-vis employee attrition

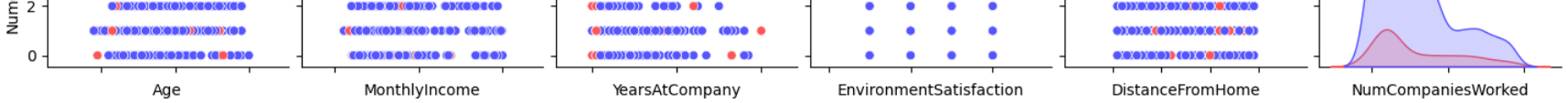
In [52]:
```
# Refining our list of numerical variables
numerical_list = ['Age','Attrition','MonthlyIncome','YearsAtCompany','EnvironmentSatisfaction','DistanceFromHome','NumCompaniesWor

graph = sns.pairplot(df_attrition[numerical_list], hue='Attrition', palette='seismic', diag_kind = 'kde',diag_kws=dict(shade=True)
graph.set(xticklabels=[])
```

```
C:\Users\bsash\anaconda3\Lib\site-packages\seaborn\axisgrid.py:1507: FutureWarning:



`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.


C:\Users\bsash\anaconda3\Lib\site-packages\seaborn\axisgrid.py:1507: FutureWarning:



`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.


C:\Users\bsash\anaconda3\Lib\site-packages\seaborn\axisgrid.py:1507: FutureWarning:



`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.


C:\Users\bsash\anaconda3\Lib\site-packages\seaborn\axisgrid.py:1507: FutureWarning:



`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.


C:\Users\bsash\anaconda3\Lib\site-packages\seaborn\axisgrid.py:1507: FutureWarning:



`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.


C:\Users\bsash\anaconda3\Lib\site-packages\seaborn\axisgrid.py:1507: FutureWarning:



`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.


C:\Users\bsash\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning:
```

```
The figure layout has changed to tight
```

Out[52]: <seaborn.axisgrid.PairGrid at 0x2217bd8c590>


Age
MonthlyIncome
YearsAtCompany
EnvironmentSatisfaction
DistanceFromHome
CompaniesWorked
Attrition
0
1



One-Hot Encoding for Non-Numerical Variables

In [53]:
```
#Describing each field in the dataset *( Transpose the table)
df_attrition.describe(exclude="int64").T
```

Out[53]:

	count	unique	top	freq
BusinessTravel	1470	3	Travel_Rarely	1043
Department	1470	3	Research & Development	961
Education	1470	5	Bachelor	572
EducationField	1470	6	Life Sciences	606
Gender	1470	2	Male	882
JobInvolvement	1470	4	High	868
JobLevel	1470	5	Entry Level	543
JobRole	1470	9	Sales Executive	326
JobSatisfaction	1470	4	Very High	459
MaritalStatus	1470	3	Married	673
PerformanceRating	1470	2	Excellent	1244
RelationshipSatisfaction	1470	4	High	459
WorkLifeBalance	1470	4	Better	893
DistanceGroup	1470	4	10+ kms	444
TotalWorkingYearsGroup	1459	5	8-10 years	401
YearsAtCompanyGroup	1426	6	2-5 years	561
YearsInCurrentRoleGroup	1226	4	2-5 years	647
YearsSinceLastPromotionGroup	889	4	0-1 years	357
YearsWithCurrManagerGroup	1207	4	2-5 years	615

The above 13 variables need to be encoded for use in modeling

In [54]:
```
from sklearn.preprocessing import LabelEncoder
```

In [55]:
```
#diagnosis
cat_cols = ['BusinessTravel','Department','Education','EducationField','Gender','JobInvolvement','JobLevel','JobSatisfaction','Mar

#turning categories into their numerical counterparts using LabelEncoder
for var in cat_cols:
    number = LabelEncoder()
    df_attrition[var+"cat"] = number.fit_transform(df_attrition[var].astype('str'))
```

In [56]:
```
#Describing each field in the dataset *( Transpose the table)
df_attrition.describe(exclude="O").T
```

Out[56]:

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
Age	1470.0	NaN	NaN	NaN	36.92381	9.135373	18.0	30.0	36.0	43.0	60.0
Attrition	1470.0	NaN	NaN	NaN	0.161224	0.367863	0.0	0.0	0.0	0.0	1.0
DailyRate	1470.0	NaN	NaN	NaN	802.485714	403.5091	102.0	465.0	802.0	1157.0	1499.0
DistanceFromHome	1470.0	NaN	NaN	NaN	9.192517	8.106864	1.0	2.0	7.0	14.0	29.0
EmployeeNumber	1470.0	NaN	NaN	NaN	1024.865306	602.024335	1.0	491.25	1020.5	1555.75	2068.0
EnvironmentSatisfaction	1470.0	NaN	NaN	NaN	2.721769	1.093082	1.0	2.0	3.0	4.0	4.0
HourlyRate	1470.0	NaN	NaN	NaN	65.891156	20.329428	30.0	48.0	66.0	83.75	100.0
MonthlyIncome	1470.0	NaN	NaN	NaN	6502.931293	4707.956783	1009.0	2911.0	4919.0	8379.0	19999.0
MonthlyRate	1470.0	NaN	NaN	NaN	14313.103401	7117.786044	2094.0	8047.0	14235.5	20461.5	26999.0
NumCompaniesWorked	1470.0	NaN	NaN	NaN	2.693197	2.498009	0.0	1.0	2.0	4.0	9.0
OverTime	1470.0	NaN	NaN	NaN	0.282993	0.450606	0.0	0.0	0.0	1.0	1.0
PercentSalaryHike	1470.0	NaN	NaN	NaN	15.209524	3.659938	11.0	12.0	14.0	18.0	25.0
StockOptionLevel	1470.0	NaN	NaN	NaN	0.793878	0.852077	0.0	0.0	1.0	1.0	3.0
TotalWorkingYears	1470.0	NaN	NaN	NaN	11.279592	7.780782	0.0	6.0	10.0	15.0	40.0
TrainingTimesLastYear	1470.0	NaN	NaN	NaN	2.79932	1.289271	0.0	2.0	3.0	3.0	6.0
YearsAtCompany	1470.0	NaN	NaN	NaN	7.008163	6.126525	0.0	3.0	5.0	9.0	40.0
YearsInCurrentRole	1470.0	NaN	NaN	NaN	4.229252	3.623137	0.0	2.0	3.0	7.0	18.0
YearsSinceLastPromotion	1470.0	NaN	NaN	NaN	2.187755	3.22243	0.0	0.0	1.0	3.0	15.0
YearsWithCurrManager	1470.0	NaN	NaN	NaN	4.123129	3.568136	0.0	2.0	3.0	7.0	17.0
DistanceGroup	1470	4	10+ kms	444	NaN	NaN	NaN	NaN	NaN	NaN	NaN
TotalWorkingYearsGroup	1459	5	8-10 years	401	NaN	NaN	NaN	NaN	NaN	NaN	NaN
YearsAtCompanyGroup	1426	6	2-5 years	561	NaN	NaN	NaN	NaN	NaN	NaN	NaN
YearsInCurrentRoleGroup	1226	4	2-5 years	647	NaN	NaN	NaN	NaN	NaN	NaN	NaN
YearsSinceLastPromotionGroup	889	4	0-1 years	357	NaN	NaN	NaN	NaN	NaN	NaN	NaN
YearsWithCurrManagerGroup	1207	4	2-5 years	615	NaN	NaN	NaN	NaN	NaN	NaN	NaN
BusinessTravelcat	1470.0	NaN	NaN	NaN	1.607483	0.665455	0.0	1.0	2.0	2.0	2.0
Departmentcat	1470.0	NaN	NaN	NaN	1.260544	0.527792	0.0	1.0	1.0	2.0	2.0
Educationcat	1470.0	NaN	NaN	NaN	1.680272	1.639316	0.0	0.0	1.0	4.0	4.0

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
EducationFieldcat	1470.0	NaN	NaN	NaN	2.247619	1.331369	0.0	1.0	2.0	3.0	5.0
Gendercat	1470.0	NaN	NaN	NaN	0.6	0.490065	0.0	0.0	1.0	1.0	1.0
JobInvolvementcat	1470.0	NaN	NaN	NaN	0.860544	1.103991	0.0	0.0	0.0	2.0	3.0
JobLevelcat	1470.0	NaN	NaN	NaN	1.506803	1.311164	0.0	0.0	2.0	2.0	4.0
JobSatisfactioncat	1470.0	NaN	NaN	NaN	1.514286	1.215175	0.0	0.0	2.0	3.0	3.0
MaritalStatuscat	1470.0	NaN	NaN	NaN	1.097279	0.730121	0.0	1.0	1.0	2.0	2.0
PerformanceRatingcat	1470.0	NaN	NaN	NaN	0.153741	0.360824	0.0	0.0	0.0	0.0	1.0
RelationshipSatisfactioncat	1470.0	NaN	NaN	NaN	1.481633	1.209505	0.0	0.0	1.5	3.0	3.0
WorkLifeBalancecat	1470.0	NaN	NaN	NaN	2.021088	0.745463	0.0	2.0	2.0	2.0	3.0

In [57]:
```
#creating a list of only numerical values
df_attrition_numeric = df_attrition.select_dtypes(exclude="O").select_dtypes(exclude=["category"])
df_attrition_numeric.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   DailyRate                     1470 non-null   int64
 3   DistanceFromHome              1470 non-null   int64
 4   EmployeeNumber                1470 non-null   int64
 5   EnvironmentSatisfaction       1470 non-null   int64
 6   HourlyRate                    1470 non-null   int64
 7   MonthlyIncome                 1470 non-null   int64
 8   MonthlyRate                   1470 non-null   int64
 9   NumCompaniesWorked            1470 non-null   int64
 10  OverTime                      1470 non-null   int64
 11  PercentSalaryHike             1470 non-null   int64
 12  StockOptionLevel              1470 non-null   int64
 13  TotalWorkingYears             1470 non-null   int64
 14  TrainingTimesLastYear         1470 non-null   int64
 15  YearsAtCompany                1470 non-null   int64
 16  YearsInCurrentRole            1470 non-null   int64
 17  YearsSinceLastPromotion       1470 non-null   int64
 18  YearsWithCurrManager          1470 non-null   int64
 19  BusinessTravelcat             1470 non-null   int32
 20  Departmentcat                 1470 non-null   int32
 21  Educationcat                  1470 non-null   int32
 22  EducationFieldcat             1470 non-null   int32
 23  Gendercat                     1470 non-null   int32
 24  JobInvolvementcat             1470 non-null   int32
 25  JobLevelcat                   1470 non-null   int32
 26  JobSatisfactioncat            1470 non-null   int32
 27  MaritalStatuscat              1470 non-null   int32
 28  PerformanceRatingcat          1470 non-null   int32
 29  RelationshipSatisfactioncat   1470 non-null   int32
 30  WorkLifeBalancecat            1470 non-null   int32
dtypes: int32(12), int64(19)
memory usage: 287.2 KB
```

As we can observe from the dataset, our target column with which we can point our model to train on would be the "Attrition" column.

In [58]:
```
df_attrition_numeric['Attrition'].value_counts()
```

Out[58]:
```
Attrition
0    1233
1     237
Name: count, dtype: int64
```

Building a model and evaluating

Having performed some exploratory data analysis and simple feature engineering as well as having ensured that all categorical values are encoded, we are now ready to proceed onto building our models.

Will evaluate using different learning models.

Splitting Data into Train and Test sets

But before we even start training a model, we will split our dataset into a training set and a test sets.

In [59]:
```
#matrices of features
x = df_attrition_numeric.drop(labels=['Attrition'],axis=1)
y = df_attrition_numeric['Attrition']
col=x.columns
```

In [60]:
```
x.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 30 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Age                           1470 non-null   int64
 1   DailyRate                     1470 non-null   int64
 2   DistanceFromHome              1470 non-null   int64
 3   EmployeeNumber                1470 non-null   int64
 4   EnvironmentSatisfaction       1470 non-null   int64
 5   HourlyRate                    1470 non-null   int64
 6   MonthlyIncome                 1470 non-null   int64
 7   MonthlyRate                   1470 non-null   int64
 8   NumCompaniesWorked            1470 non-null   int64
 9   OverTime                      1470 non-null   int64
 10  PercentSalaryHike             1470 non-null   int64
 11  StockOptionLevel              1470 non-null   int64
 12  TotalWorkingYears             1470 non-null   int64
 13  TrainingTimesLastYear         1470 non-null   int64
 14  YearsAtCompany                1470 non-null   int64
 15  YearsInCurrentRole            1470 non-null   int64
 16  YearsSinceLastPromotion       1470 non-null   int64
 17  YearsWithCurrManager          1470 non-null   int64
 18  BusinessTravelcat             1470 non-null   int32
 19  Departmentcat                 1470 non-null   int32
 20  Educationcat                  1470 non-null   int32
 21  EducationFieldcat             1470 non-null   int32
 22  Gendercat                     1470 non-null   int32
 23  JobInvolvementcat             1470 non-null   int32
 24  JobLevelcat                   1470 non-null   int32
 25  JobSatisfactioncat            1470 non-null   int32
 26  MaritalStatuscat              1470 non-null   int32
 27  PerformanceRatingcat          1470 non-null   int32
 28  RelationshipSatisfactioncat   1470 non-null   int32
 29  WorkLifeBalancecat            1470 non-null   int32
dtypes: int32(12), int64(18)
memory usage: 275.8 KB
```

In [61]:
```
# Import the train_test_split method
#train, test, split
from sklearn.model_selection import train_test_split

# Split data into train and test sets as well as for validation and testing
x_train, x_test, y_train, y_test= train_test_split(x, y, train_size= 0.40, random_state=10);
```

I will be using oversampling technique known as SMOTE to treat the imbalance.

In [62]:
```
#pip install -U imbalanced-learn
```

In [63]:
```
from imblearn.over_sampling import SMOTE
sm = SMOTE()
x_train_smote, y_train_smote = sm.fit_resample(x_train,y_train)
```

In [64]:
```
# summarize the new class distribution
from collections import Counter
counter = Counter(y_train)
print(counter)
counter = Counter(y_test)
print(counter)
counter = Counter(y_train_smote)
print(counter)
```

```
Counter({0: 503, 1: 85})
Counter({0: 730, 1: 152})
Counter({0: 503, 1: 503})
```

In [65]:
```
# Sklearn regression algorithms
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder

# Sklearn regression model evaluation function
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_report, roc_curve, precision_recall_cu
from sklearn.metrics import f1_score
from sklearn.metrics import balanced_accuracy_score
import warnings

from sklearn.datasets import make_classification
from numpy import mean


from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
from sklearn import metrics
```

In [66]:
```
# Confusion Matrix
def plot_confusion_matrix() :
    x_axis_labels = ['Did not leave','Left']# labels for x-axis
```

```
    y_axis_labels = ['Did not leave','Left'] # labels for y-axis
    sns.heatmap(cm,annot=True,fmt='g',cmap=plt.cm.Blues,xticklabels=x_axis_labels, yticklabels=y_axis_labels)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show
```

In [67]:
```
# Precision, Recall, F1 Score
def show_metrics():
    print("Accuracy of model       :",accuracy_score(y_test, y_predict))
    print("Precision Score         :",precision_score(y_test, y_predict))
    print("F1 Score                :",f1_score(y_test, y_predict))
    print("Recall Score            :",recall_score(y_test, y_predict))
    print("AUC Score               :",metrics.roc_auc_score(y_test,y_predict))
    print("Balanced Accuracy Score :",balanced_accuracy_score(y_test, y_predict))
```

In [68]:
```
# ROC curve
def plot_roc(y_test, logpred):
    roc_auc = metrics.roc_auc_score(y_test, logpred)
    fpr, tpr, thresholds = metrics.roc_curve(y_test, logpred)
    plt.figure()

    plt.plot(fpr, tpr, label = 'ROC curve', color ='orange', linewidth = 2)
    plt.plot([0,1],[0,1], 'k--', linewidth = 2)
    plt.xlim([0.0,1.0])
    plt.ylim([0.0,1.0])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.show();
```

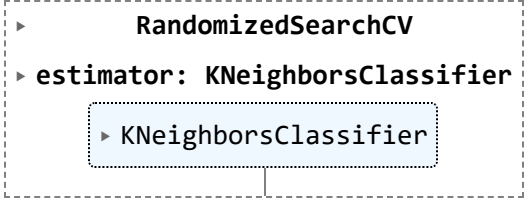
K-Nearest Neighbor

Straight forward pattern recognition model which allows the testing of several k values and leaf sizes to determine the best performance

In [69]:
```
# Use the KNN classifier to fit data:
classifier = KNeighborsClassifier()
k_range    = list(range(1, 50))
leaf_size = list(range(1, 50))
weight_options = ['uniform', 'distance']
algorithm = ['auto']
param_grid = {'n_neighbors': k_range, 'leaf_size': leaf_size, 'weights': weight_options, 'algorithm': algorithm}
```

In [70]:
```
rand_knn = RandomizedSearchCV(classifier, param_grid, cv=10, scoring="accuracy", n_iter=100, random_state=42)
rand_knn.fit(x_train_smote,y_train_smote.values.ravel())
```

Out[70]:



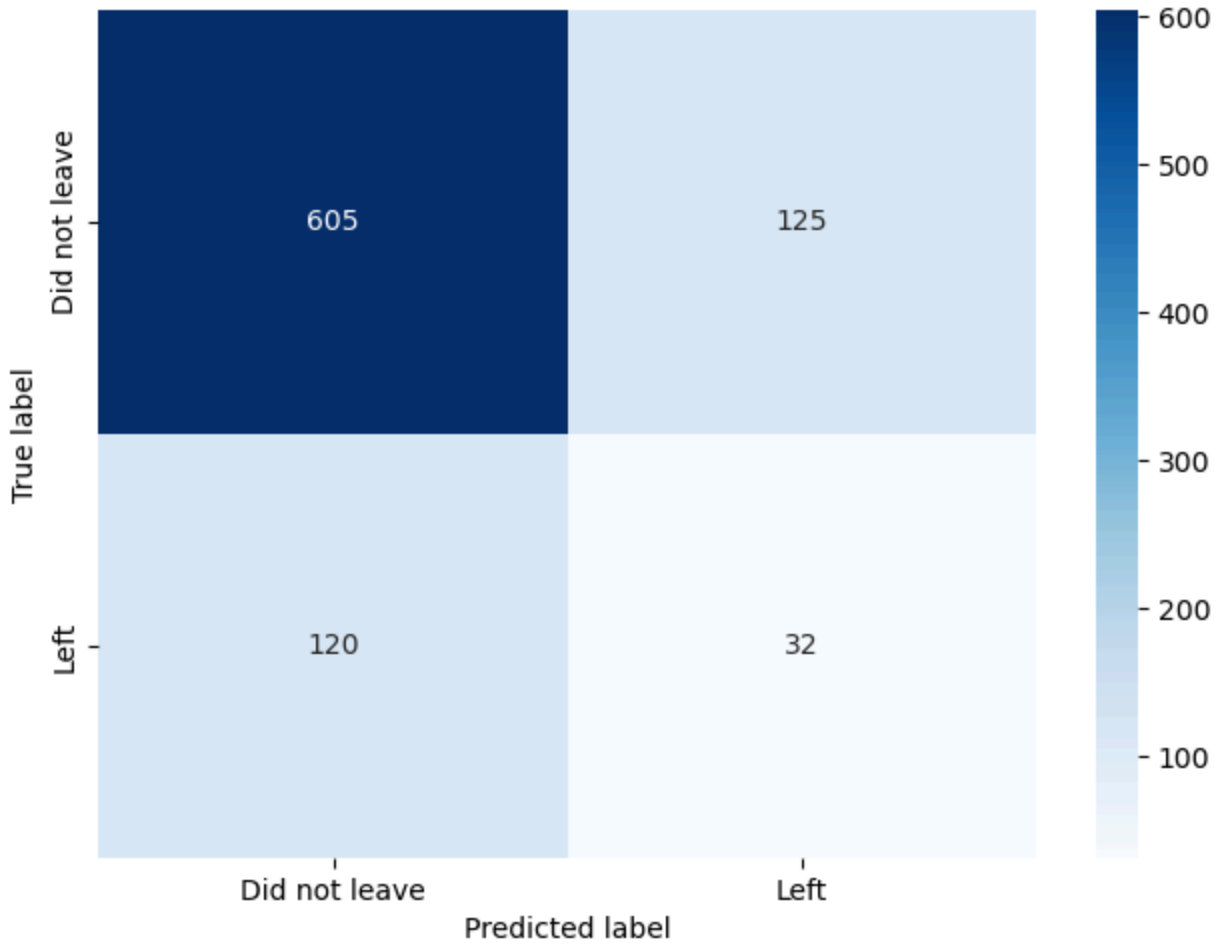
In [71]:
```
#Summary of performance results
print("-"*100)
print("KNN Results")
print("Best Accuracy :",rand_knn.best_score_)
print("Best Parameters :",rand_knn.best_params_)
print("Best Estimator :",rand_knn.best_estimator_)
```

```
----------------------------------------------------------------------------------------------------
KNN Results
Best Accuracy : 0.844920792079208
Best Parameters : {'weights': 'distance', 'n_neighbors': 2, 'leaf_size': 15, 'algorithm': 'auto'}
Best Estimator : KNeighborsClassifier(leaf_size=15, n_neighbors=2, weights='distance')
```

In [72]:
```
# Use the KNN classifier to fit data:
K_value=2
classifier = KNeighborsClassifier(n_neighbors = K_value, leaf_size = 4 , algorithm = 'auto')
classifier.fit(x_train_smote, y_train_smote)
# Predict y data with classifier:
y_predict = classifier.predict(x_test)
show_metrics()
```

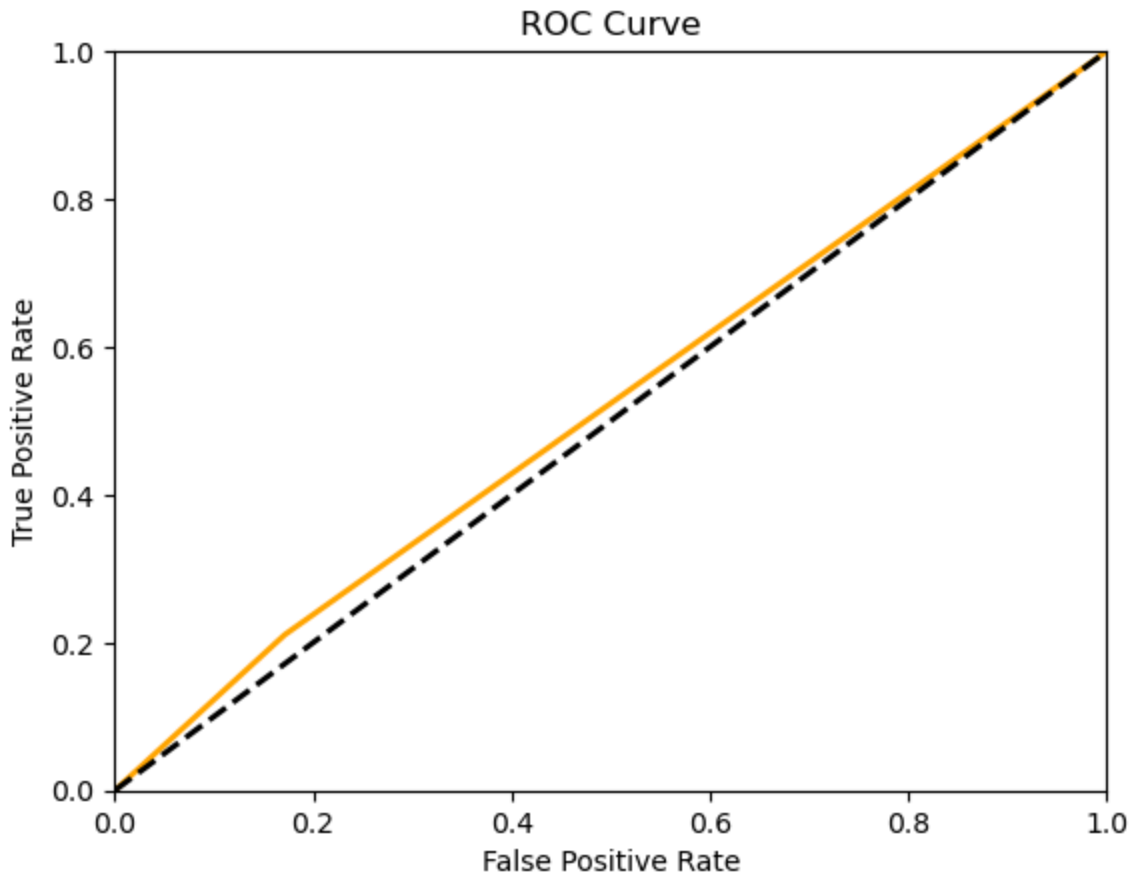
```
Accuracy of model       : 0.7222222222222222
Precision Score         : 0.20382165605095542
F1 Score                : 0.20711974110032363
Recall Score            : 0.21052631578947367
AUC Score               : 0.5196467195385723
Balanced Accuracy Score : 0.5196467195385724
```

In [73]:
```
# Print results:
cm=metrics.confusion_matrix(y_test, y_predict)
plot_confusion_matrix()
```



In [74]:
```
print(classification_report(y_test, y_predict))
plot_roc(y_test, y_predict)
```

```
              precision    recall  f1-score   support

           0       0.83      0.83      0.83       730
           1       0.20      0.21      0.21       152

    accuracy                           0.72       882
   macro avg       0.52      0.52      0.52       882
weighted avg       0.73      0.72      0.72       882
```



Random Forest Classifier

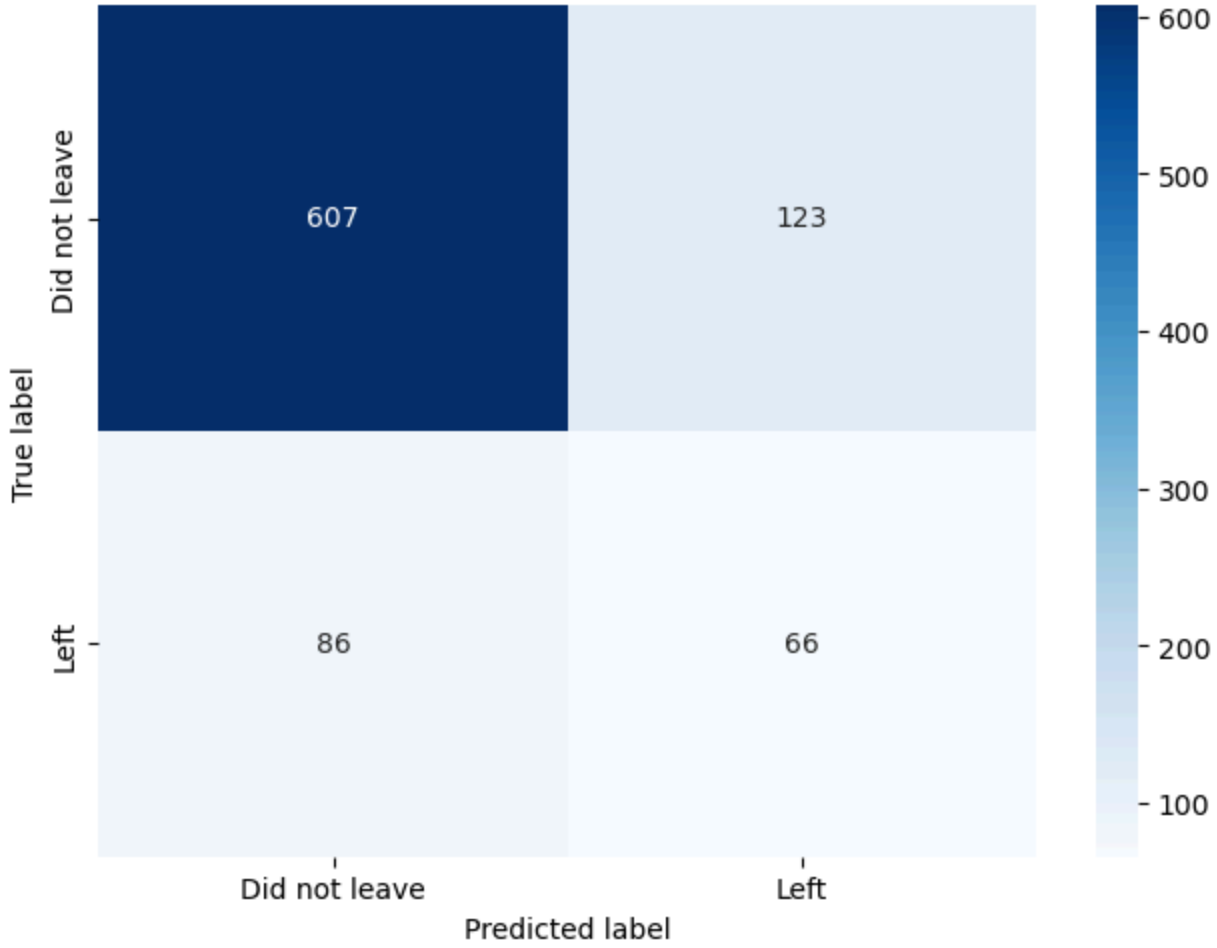
In [75]:
```
seed = 0   # We set our random seed to zero for reproducibility
# Random Forest parameters
rf_params = {
    'n_jobs': -1,
    'n_estimators': 1000,
#     'warm_start': True,
    'max_features': 0.3,
    'max_depth': 4,
    'min_samples_leaf': 2,
    'max_features' : 'sqrt',
    'random_state' : seed,
    'verbose': 0
}
```

In [76]:
```
classifier = RandomForestClassifier(**rf_params)
classifier.fit(x_train_smote, y_train_smote)
# Predict y data with classifier:
```

```
y_predict = classifier.predict(x_test)
show_metrics()
```

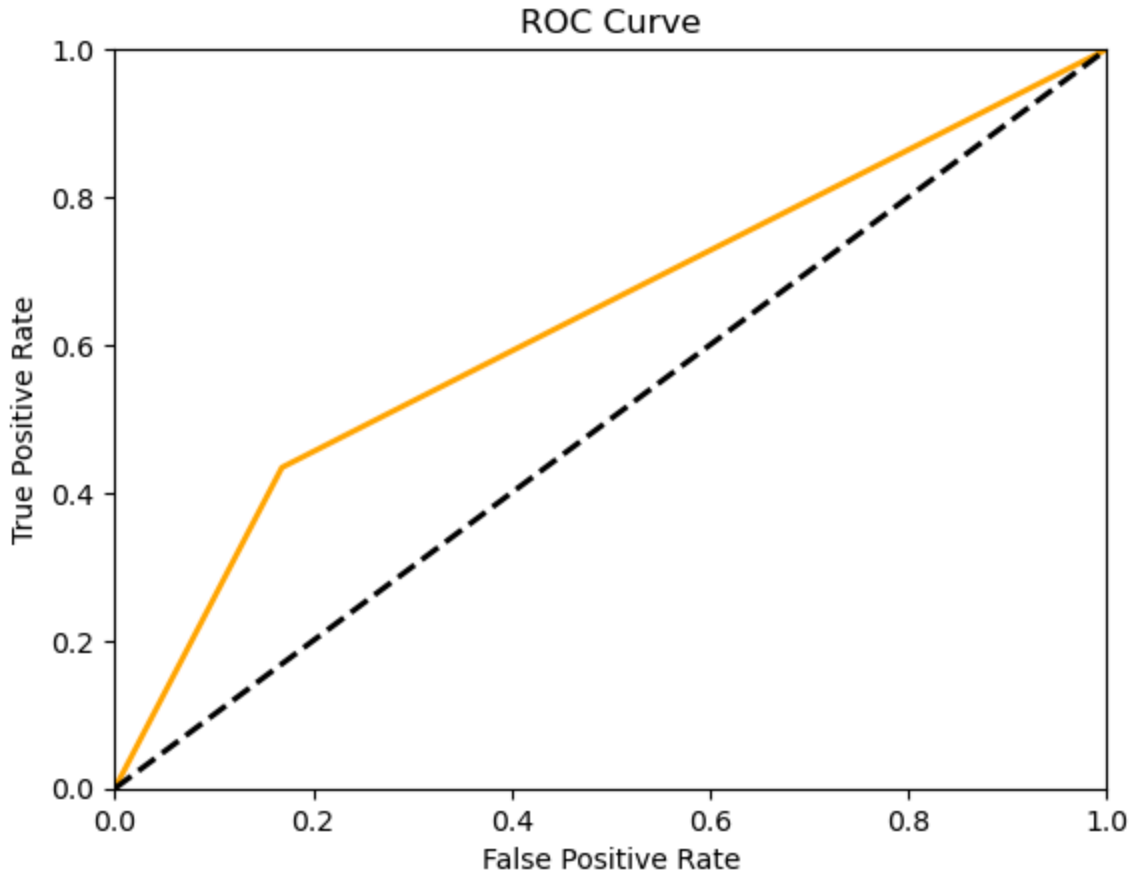
```
Accuracy of model       : 0.7630385487528345
Precision Score         : 0.3492063492063492
F1 Score                : 0.3870967741935484
Recall Score            : 0.4342105263157895
AUC Score               : 0.632858687815429
Balanced Accuracy Score : 0.632858687815429
```

In [77]:
```
# Print results:
cm=metrics.confusion_matrix(y_test, y_predict)
plot_confusion_matrix()
```



In [78]:
```
print(classification_report(y_test, y_predict))
plot_roc(y_test, y_predict)
```

```
              precision    recall  f1-score   support

           0       0.88      0.83      0.85       730
           1       0.35      0.43      0.39       152

    accuracy                           0.76       882
   macro avg       0.61      0.63      0.62       882
weighted avg       0.79      0.76      0.77       882
```



Accuracy of the model

As observed, our Random Forest returns an accuracy of approx 73.58% for its predictions and on first glance this might seem to be a pretty good performing model.

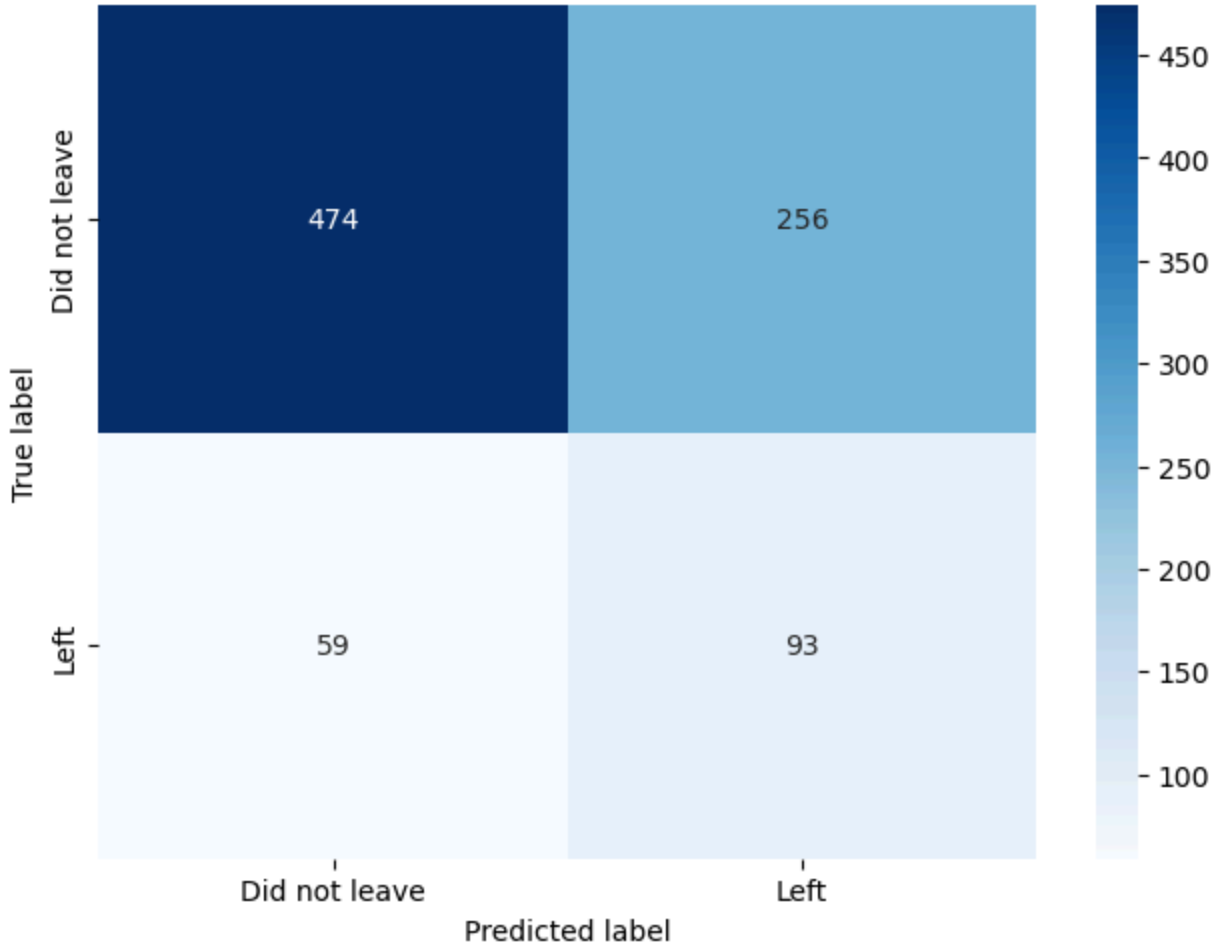
Naive Bayes

In [79]:
```
from sklearn.naive_bayes import GaussianNB
```

```
classifier = GaussianNB()
classifier.fit(x_train_smote, y_train_smote)
# Predict y data with classifier:
y_predict = classifier.predict(x_test)
show_metrics()
```

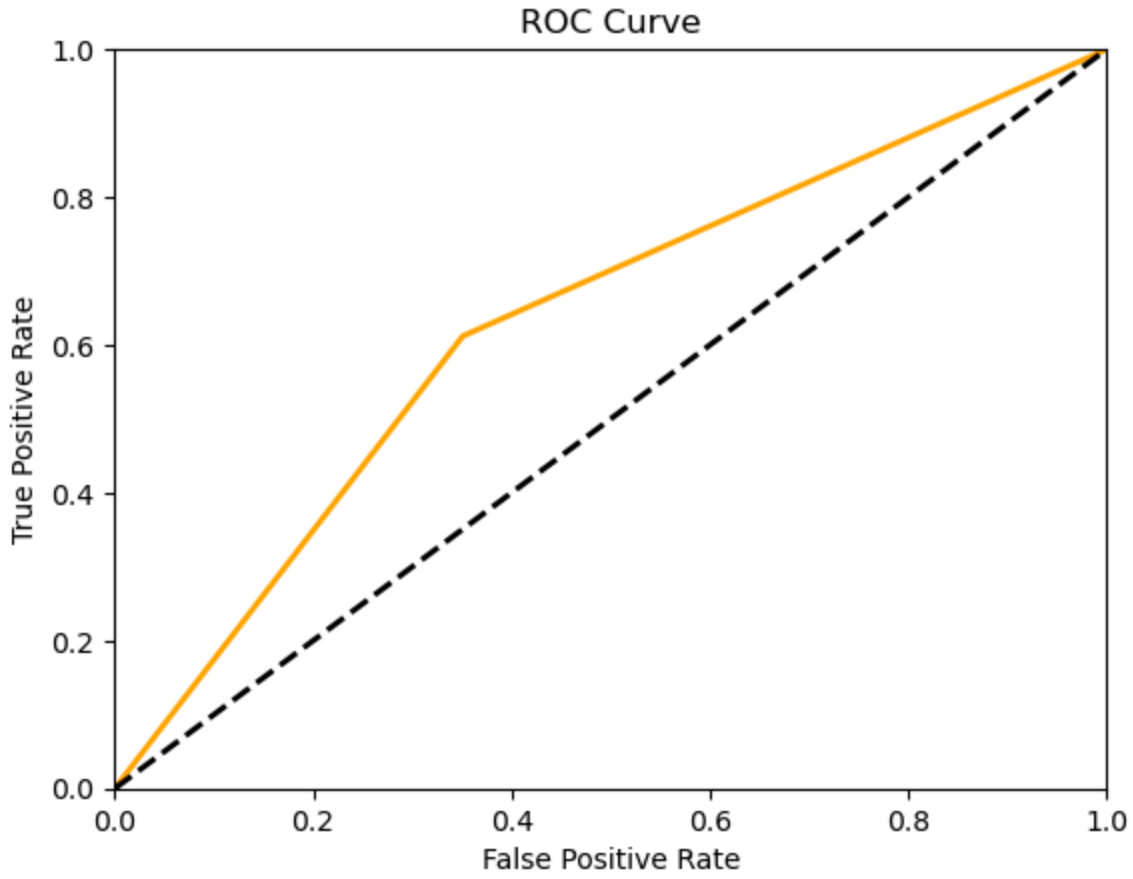
```
Accuracy of model       : 0.6428571428571429
Precision Score         : 0.2664756446991404
F1 Score                : 0.37125748502994016
Recall Score            : 0.6118421052631579
AUC Score               : 0.6305785868781544
Balanced Accuracy Score : 0.6305785868781543
```

In [80]:
```
# Print results:
cm=metrics.confusion_matrix(y_test, y_predict)
plot_confusion_matrix()
```



In [81]:
```
print(classification_report(y_test, y_predict))
plot_roc(y_test, y_predict)
```

```
              precision    recall  f1-score   support

           0       0.89      0.65      0.75       730
           1       0.27      0.61      0.37       152

    accuracy                           0.64       882
   macro avg       0.58      0.63      0.56       882
weighted avg       0.78      0.64      0.69       882
```



Naive Bayes Model accuracy is very poor

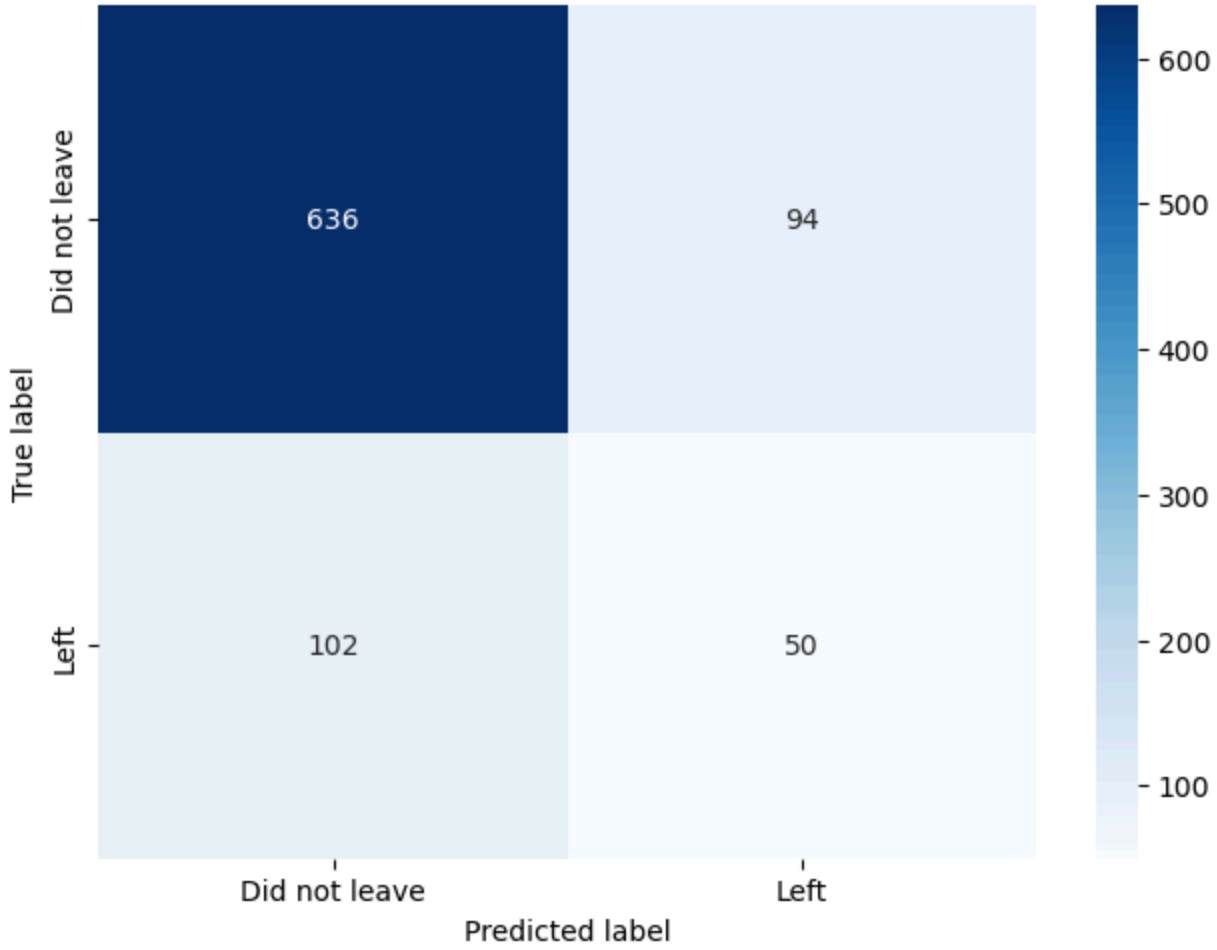
Decision Tree Classifier

In [82]:
```
from sklearn.tree import DecisionTreeClassifier

# train model
classifier = DecisionTreeClassifier(max_depth = 10, random_state= 101, max_features =None , min_samples_leaf = 30)

classifier.fit(x_train_smote, y_train_smote)
```

```
# Predict y data with classifier:
y_predict = classifier.predict(x_test)
show_metrics()
```

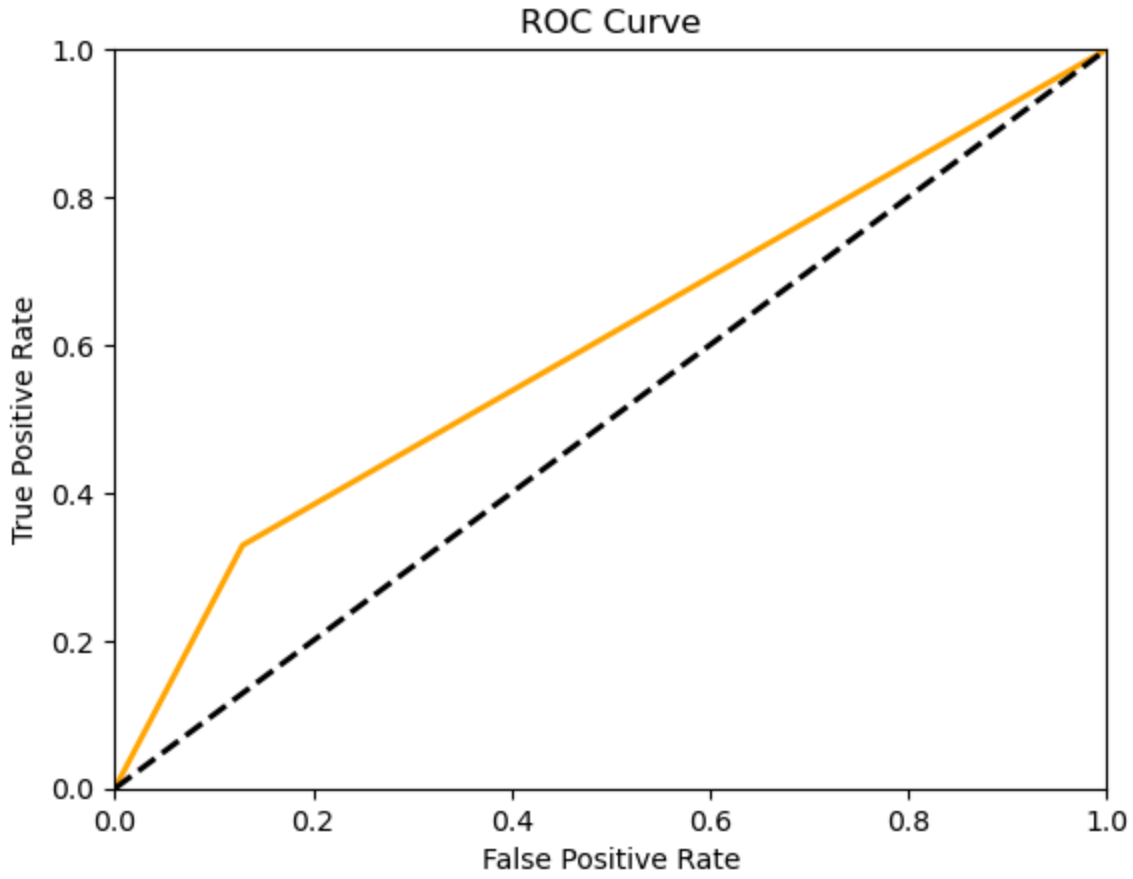
```
Accuracy of model       : 0.7777777777777778
Precision Score         : 0.3472222222222222
F1 Score                : 0.33783783783783783
Recall Score            : 0.32894736842105265
AUC Score               : 0.6000901225666907
Balanced Accuracy Score : 0.6000901225666907
```

In [83]:
```
# Print results:
cm=metrics.confusion_matrix(y_test, y_predict)
plot_confusion_matrix()
```



In [84]:
```
print(classification_report(y_test, y_predict))
plot_roc(y_test, y_predict)
```

```
              precision    recall  f1-score   support

           0       0.86      0.87      0.87       730
           1       0.35      0.33      0.34       152

    accuracy                           0.78       882
   macro avg       0.60      0.60      0.60       882
weighted avg       0.77      0.78      0.78       882
```



Accuracy score is good, however the model is not predicting the Defaults well

XGBoost Model

In [85]:
```
#pip install xgboost
```

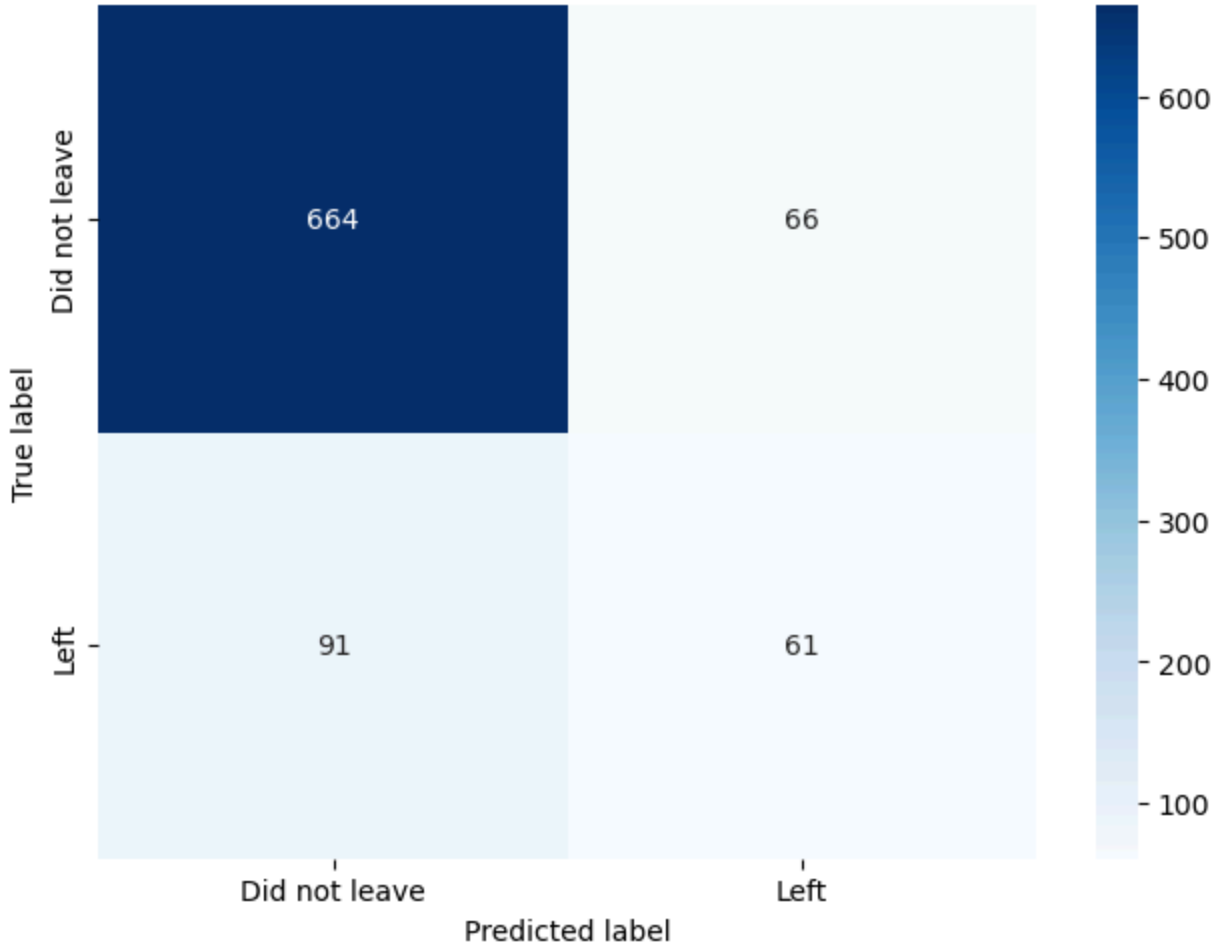
In [86]:
```
from xgboost import XGBClassifier

# train model
```

```
classifier = XGBClassifier(objective = 'binary:logistic',eval_metric = 'logloss',seed = 42, use_label_encoder = False)

classifier.fit(x_train_smote, y_train_smote)
# Predict y data with classifier:
y_predict = classifier.predict(x_test)
show_metrics()
```

```
Accuracy of model       : 0.8219954648526077
Precision Score         : 0.48031496062992124
F1 Score                : 0.43727598566308246
Recall Score            : 0.40131578947368424
AUC Score               : 0.6554524152847874
Balanced Accuracy Score : 0.6554524152847874
```

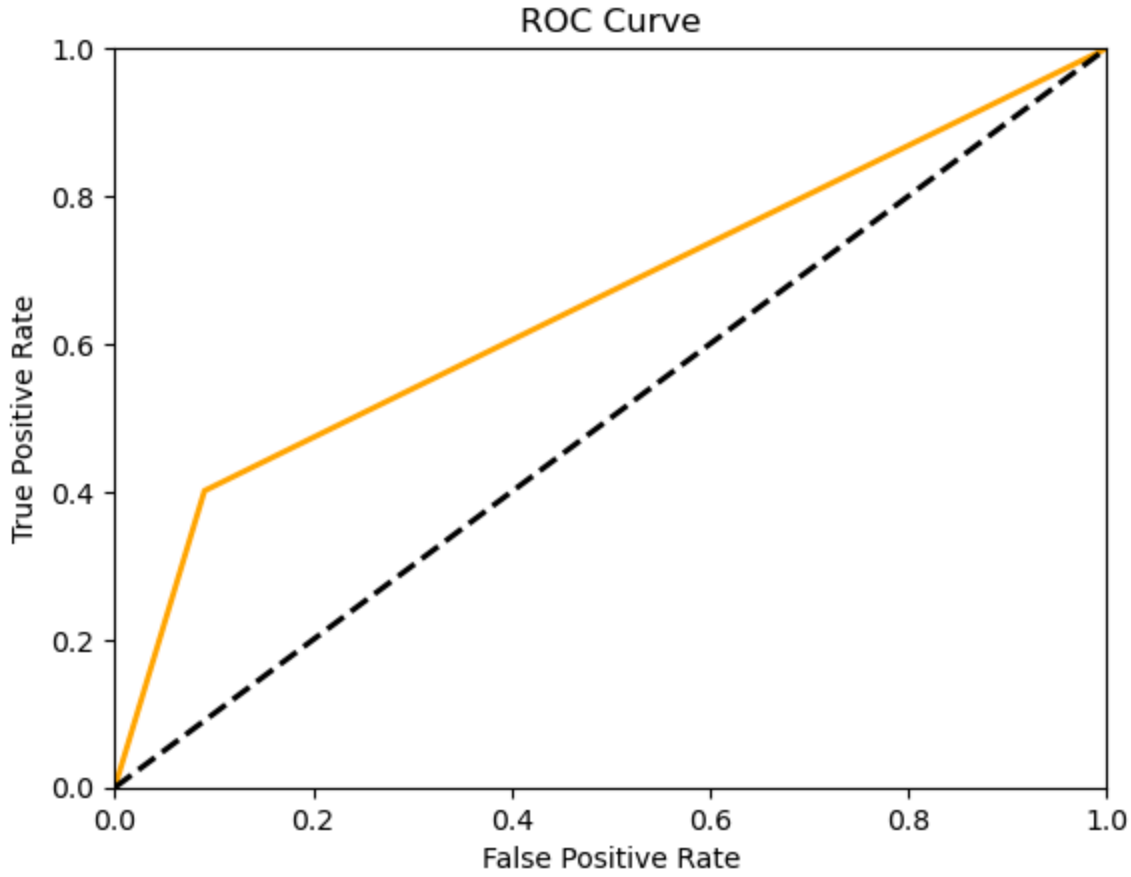
In [87]:

```
# Print results:
cm=metrics.confusion_matrix(y_test, y_predict)
plot_confusion_matrix()
```



In [88]:
```
print(classification_report(y_test, y_predict))
plot_roc(y_test, y_predict)
```

```
              precision    recall  f1-score   support

           0       0.88      0.91      0.89       730
           1       0.48      0.40      0.44       152

    accuracy                           0.82       882
   macro avg       0.68      0.66      0.67       882
weighted avg       0.81      0.82      0.82       882
```



Gradient Boosted Classifier

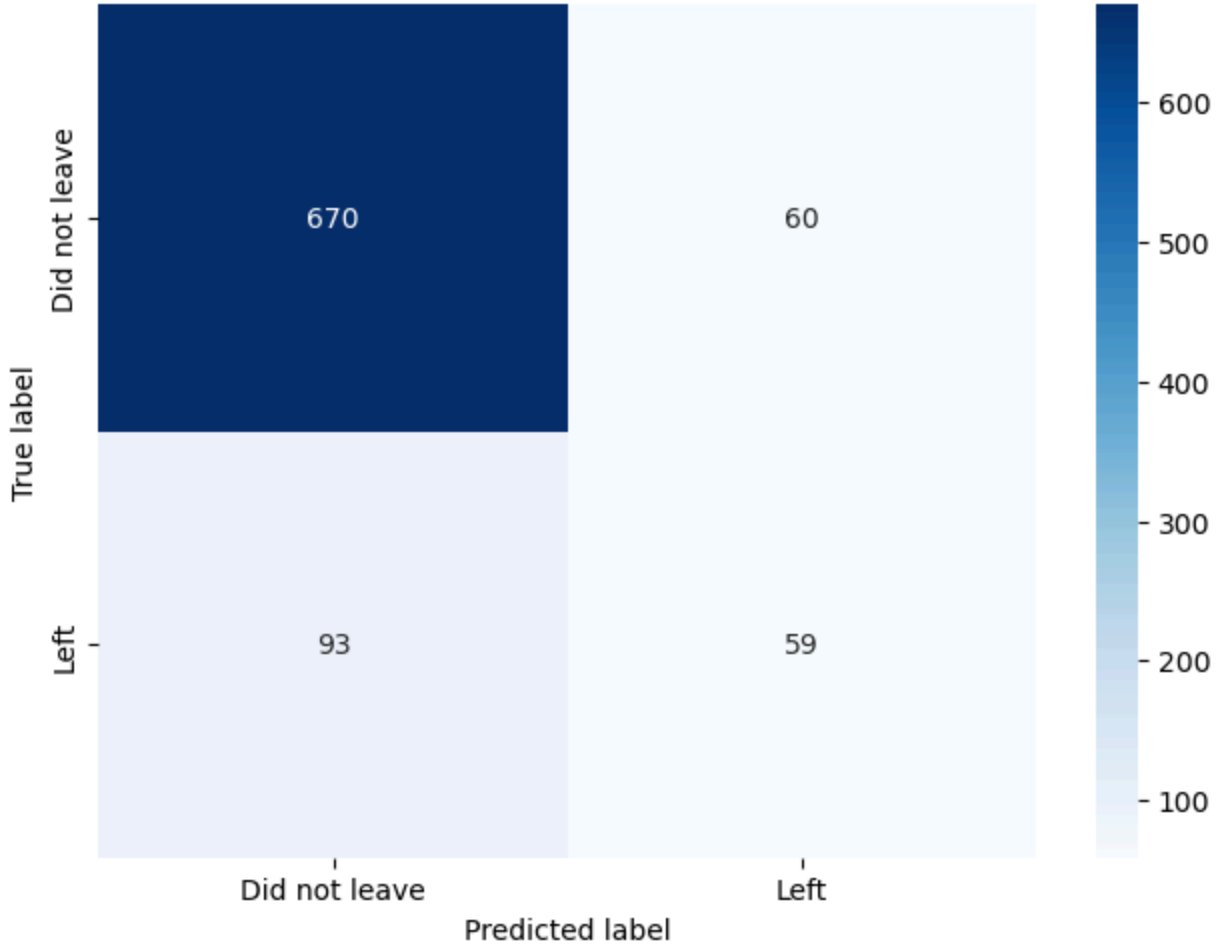
In [89]:
```
# Gradient Boosting Parameters
gb_params ={
    'n_estimators': 1500,
    'max_features': 0.9,
    'learning_rate' : 0.25,
```

```
    'max_depth': 4,
    'min_samples_leaf': 2,
    'subsample': 1,
    'max_features' : 'sqrt',
    'random_state' : seed,
    'verbose': 0
}
```

In [90]:
```
# train model
classifier =  GradientBoostingClassifier(**gb_params)

classifier.fit(x_train_smote, y_train_smote)
# Predict y data with classifier:
y_predict = classifier.predict(x_test)
show_metrics()
```

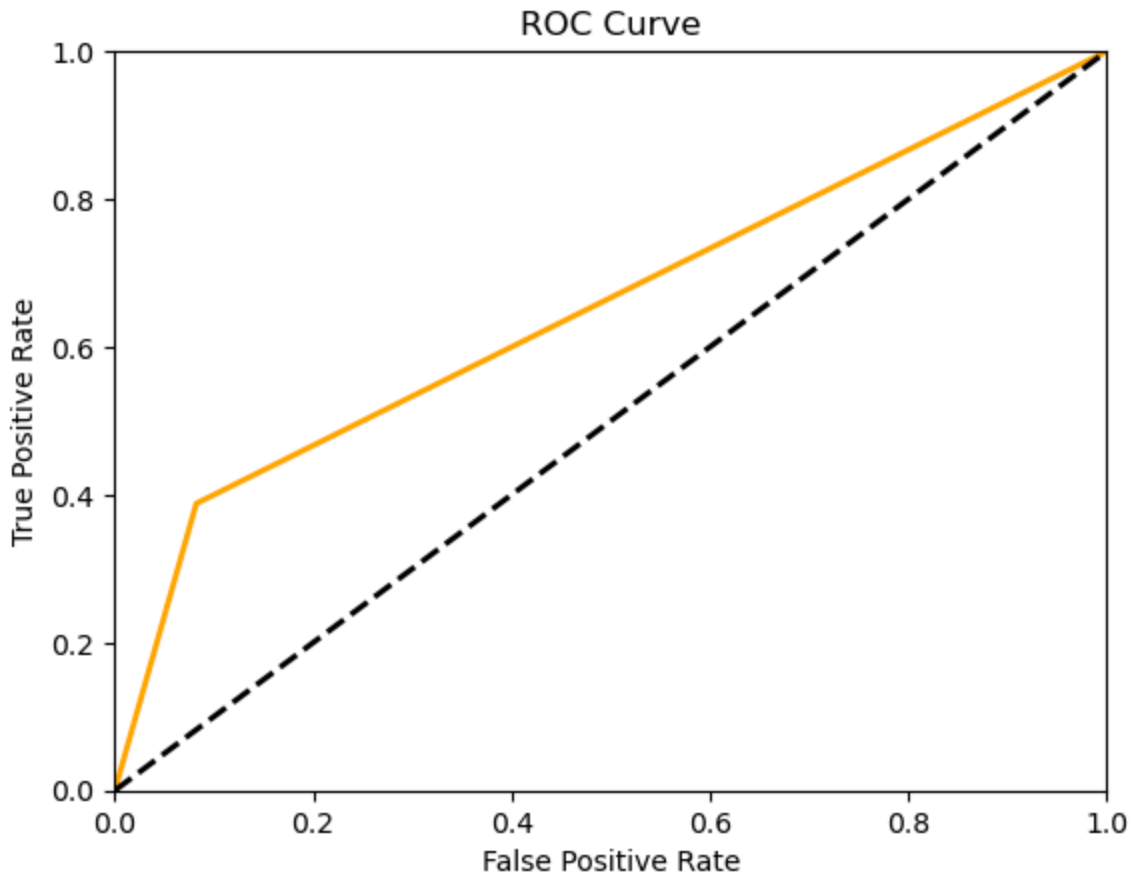
```
Accuracy of model       : 0.826530612244898
Precision Score         : 0.4957983193277311
F1 Score                : 0.4354243542435424
Recall Score            : 0.3881578947368421
AUC Score               : 0.6529830569574622
Balanced Accuracy Score : 0.6529830569574622
```

In [91]:
```
# Print results:
cm=metrics.confusion_matrix(y_test, y_predict)
plot_confusion_matrix()
```



In [92]:
```
print(classification_report(y_test, y_predict))
plot_roc(y_test, y_predict)
```

```
              precision    recall  f1-score   support

           0       0.88      0.92      0.90       730
           1       0.50      0.39      0.44       152

    accuracy                           0.83       882
   macro avg       0.69      0.65      0.67       882
weighted avg       0.81      0.83      0.82       882
```



Comparing all the models based on Model Performance

In [93]:
```
comparison_frame = pd.DataFrame({'Model':['Test_accuracy                :',
                                          'Test_Precision_Score         :',
                                          'Test_F1_Score                :',
                                          'Test_Recall_Score            :',
                                          'Test_AUC_Score               :',
                                          'Test_Balanced_Accuracy_Score:'],
                                 'K-Nearest Neighbor(SMOTE)':[0.73469,0.23026,0.23026,0.23026,0.53499,0.53499],
                                 'Random Forest (SMOTE)':[0.73583,0.32159,0.38522,0.48026,0.63465,0.63465],
                                 'Naive Bayes(SMOTE)':[0.62925,0.26027,0.36750,0.62500,0.62757,0.62757],
                                 'Decision Tree (SMOTE)':[0.71088,0.25359,0.29363,0.34868,0.56749,0.56749],
                                 'XGBoost (SMOTE)':[ 0.81859,0.46491,0.39850,0.34868,0.63256,0.63256],
                                 'Gradient Boosted Classifier (SMOTE)':[0.81859,0.46551,0.40298,0.35526,0.63516,0.63516]}

comparison_frame
```

Out[93]:

	Model	K-Nearest Neighbor(SMOTE)	Random Forest (SMOTE)	Naive Bayes(SMOTE)	Decision Tree (SMOTE)	XGBoost (SMOTE)	Gradient Boosted Classifier (SMOTE)
0	Test_accuracy :	0.73469	0.73583	0.62925	0.71088	0.81859	0.81859
1	Test_Precision_Score :	0.23026	0.32159	0.26027	0.25359	0.46491	0.46551
2	Test_F1_Score :	0.23026	0.38522	0.36750	0.29363	0.39850	0.40298
3	Test_Recall_Score :	0.23026	0.48026	0.62500	0.34868	0.34868	0.35526
4	Test_AUC_Score :	0.53499	0.63465	0.62757	0.56749	0.63256	0.63516
5	Test_Balanced_Accuracy_Score:	0.53499	0.63465	0.62757	0.56749	0.63256	0.63516

Results interpretation :

SMOTE uses a nearest neighbors algorithm to generate new and synthetic data we used for training our model.

Naive Bayes - Model accuracy is the lowest among the 6 Models.

K-Nearest Neighbor - Has good Accuracy Score, but has the lowest F1 & Recall scores. The False positive & False negative cases are higher than all models.

Random Forest - Accuracy score is good, however the model is not predicting the Attrition correctly (precision is low) and it identifies False negatives better than all (Recall Score is high).

Decision Tree - Accuracy score is low, however the model is not predicting the Attrition correctly (precision is low)

XGBoost - Accuracy is high as well as it is identifying the Attrition better. Also, it has high Recall & F1 Scores.

Gradient Boosted Classifier - Accuracy is highest of all . And it is identifying the Attrition with the highest Precision Score. It also has the highest Recall, F1 & AUC Scores.

So, the best result is obtained by Gradient Boosted Classifier after deploying SMOTE.