HR Analysis Project Code

# TASK #1: IMPORT LIBRARIES AND DATASETS

import pandas as pd  
import numpy as np  
import seaborn as sns  
import matplotlib.pyplot as plt

pd.set\_option('max\_rows', 99999)  
pd.set\_option('max\_colwidth', 100)

# Mount your drive using the following commands:  
# For more information regarding mounting, please check this out: https://stackoverflow.com/questions/46986398/import-data-into-google-colaboratory  
  
from google.colab import drive  
drive.mount('/content/drive/')

Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mount("/content/drive/", force\_remount=True).

## Reading the dataset

# Have to include the full link to the csv file containing your dataset  
  
employee\_df = pd.read\_csv('drive/MyDrive/Human\_Resources.csv')  
employee\_df.head(5)

##Data Cleaning

###Ckecking duplicate records

# No duplicate records found  
duplicate = employee\_df[employee\_df.duplicated()]  
duplicate.count()

Age 0  
Attrition 0  
BusinessTravel 0  
DailyRate 0  
Department 0  
DistanceFromHome 0  
Education 0  
EducationField 0  
EmployeeCount 0  
EmployeeNumber 0  
EnvironmentSatisfaction 0  
Gender 0  
HourlyRate 0  
JobInvolvement 0  
JobLevel 0  
JobRole 0  
JobSatisfaction 0  
MaritalStatus 0  
MonthlyIncome 0  
MonthlyRate 0  
NumCompaniesWorked 0  
Over18 0  
OverTime 0  
PercentSalaryHike 0  
PerformanceRating 0  
RelationshipSatisfaction 0  
StandardHours 0  
StockOptionLevel 0  
TotalWorkingYears 0  
TrainingTimesLastYear 0  
WorkLifeBalance 0  
YearsAtCompany 0  
YearsInCurrentRole 0  
YearsSinceLastPromotion 0  
YearsWithCurrManager 0  
dtype: int64

###Checking null values

# No null values found  
employee\_df.isnull().sum()

Age 0  
Attrition 0  
BusinessTravel 0  
DailyRate 0  
Department 0  
DistanceFromHome 0  
Education 0  
EducationField 0  
EmployeeCount 0  
EmployeeNumber 0  
EnvironmentSatisfaction 0  
Gender 0  
HourlyRate 0  
JobInvolvement 0  
JobLevel 0  
JobRole 0  
JobSatisfaction 0  
MaritalStatus 0  
MonthlyIncome 0  
MonthlyRate 0  
NumCompaniesWorked 0  
Over18 0  
OverTime 0  
PercentSalaryHike 0  
PerformanceRating 0  
RelationshipSatisfaction 0  
StandardHours 0  
StockOptionLevel 0  
TotalWorkingYears 0  
TrainingTimesLastYear 0  
WorkLifeBalance 0  
YearsAtCompany 0  
YearsInCurrentRole 0  
YearsSinceLastPromotion 0  
YearsWithCurrManager 0  
dtype: int64

## Data exploration

###Data shape

print("Number of rows: ",len(employee\_df),"\t","Number of columns: ",len(employee\_df.columns))

Number of rows: 1470 Number of columns: 35

###Dataframe information

employee\_df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1470 entries, 0 to 1469  
Data columns (total 35 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Age 1470 non-null int64   
 1 Attrition 1470 non-null object  
 2 BusinessTravel 1470 non-null object  
 3 DailyRate 1470 non-null int64   
 4 Department 1470 non-null object  
 5 DistanceFromHome 1470 non-null int64   
 6 Education 1470 non-null int64   
 7 EducationField 1470 non-null object  
 8 EmployeeCount 1470 non-null int64   
 9 EmployeeNumber 1470 non-null int64   
 10 EnvironmentSatisfaction 1470 non-null int64   
 11 Gender 1470 non-null object  
 12 HourlyRate 1470 non-null int64   
 13 JobInvolvement 1470 non-null int64   
 14 JobLevel 1470 non-null int64   
 15 JobRole 1470 non-null object  
 16 JobSatisfaction 1470 non-null int64   
 17 MaritalStatus 1470 non-null object  
 18 MonthlyIncome 1470 non-null int64   
 19 MonthlyRate 1470 non-null int64   
 20 NumCompaniesWorked 1470 non-null int64   
 21 Over18 1470 non-null object  
 22 OverTime 1470 non-null object  
 23 PercentSalaryHike 1470 non-null int64   
 24 PerformanceRating 1470 non-null int64   
 25 RelationshipSatisfaction 1470 non-null int64   
 26 StandardHours 1470 non-null int64   
 27 StockOptionLevel 1470 non-null int64   
 28 TotalWorkingYears 1470 non-null int64   
 29 TrainingTimesLastYear 1470 non-null int64   
 30 WorkLifeBalance 1470 non-null int64   
 31 YearsAtCompany 1470 non-null int64   
 32 YearsInCurrentRole 1470 non-null int64   
 33 YearsSinceLastPromotion 1470 non-null int64   
 34 YearsWithCurrManager 1470 non-null int64   
dtypes: int64(26), object(9)  
memory usage: 402.1+ KB

###Descriptive statistics of dataframe

#Overall stats  
overall\_stat = employee\_df.describe()  
overall\_stat

# TASK #2: VISUALIZE DATASET

## 3 Categorical columns to Numeric conversion

# Replacing 'Attritition' , 'overtime' , 'Over18' column with integers before performing any visualizations   
employee\_df['Attrition'] = employee\_df['Attrition'].apply(lambda x:1 if x == 'Yes' else 0) #1: left/Yes & 0: stayed/No  
employee\_df['OverTime'] = employee\_df['OverTime'].apply(lambda x:1 if x == 'Yes' else 0)  
employee\_df['Over18'] = employee\_df['Over18'].apply(lambda x:1 if x == 'Y' else 0)

##Plot 1: Visualizing missing data

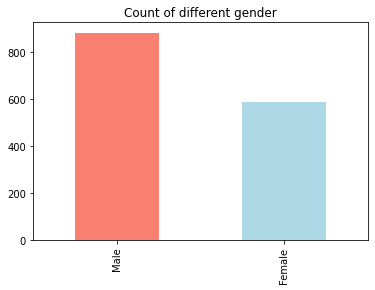
# Visualizing missing data with seaborn heatmap()  
  
plt.figure(figsize=(12,8))  
sns.heatmap(employee\_df.isnull(), yticklabels = False, cbar = False, cmap="Blues")  
  
plt.tight\_layout()  
plt.show()



##Plot 2: Bar chart of 'Gender' column

employee\_df['Gender'].value\_counts().plot(kind='bar',color=['salmon','lightblue'],title="Count of different gender")

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fd6e1964990>

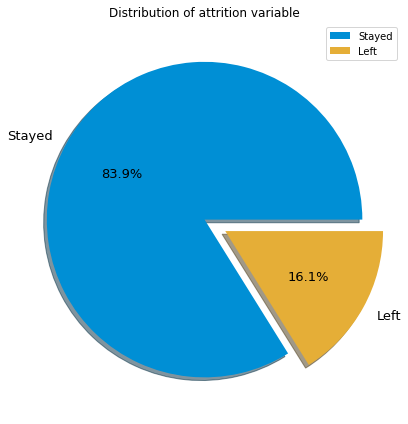


employee\_df['Gender'].value\_counts()

Male 882  
Female 588  
Name: Gender, dtype: int64

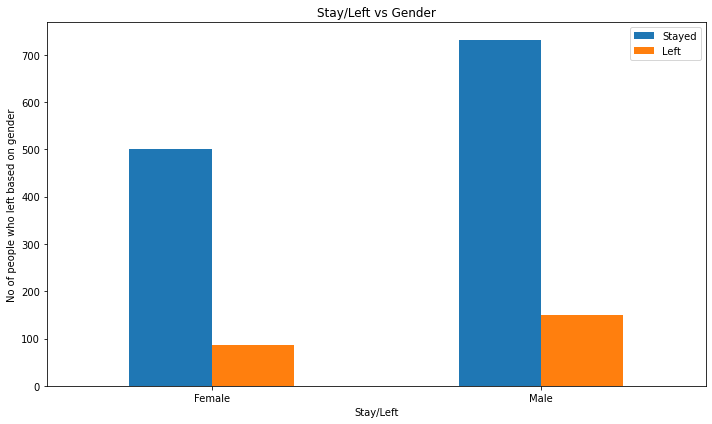
##Plot 3: Pie chart of 'Attrition' column

#Pie chart: Left/Stayed  
from matplotlib import legend  
  
plt.figure(figsize=(10,6))   
slices = employee\_df['Attrition'].value\_counts()  
  
plt.pie(slices,  
 autopct='%1.1f%%',  
 explode=[0,0.15],  
 labels=['Stayed', 'Left'],  
 colors=['#008fd5','#e5ae37'],  
 shadow=True,  
 textprops = {"fontsize":13}   
 )  
  
plt.title('Distribution of attrition variable')  
plt.legend(loc="upper right")  
plt.tight\_layout()  
plt.show()



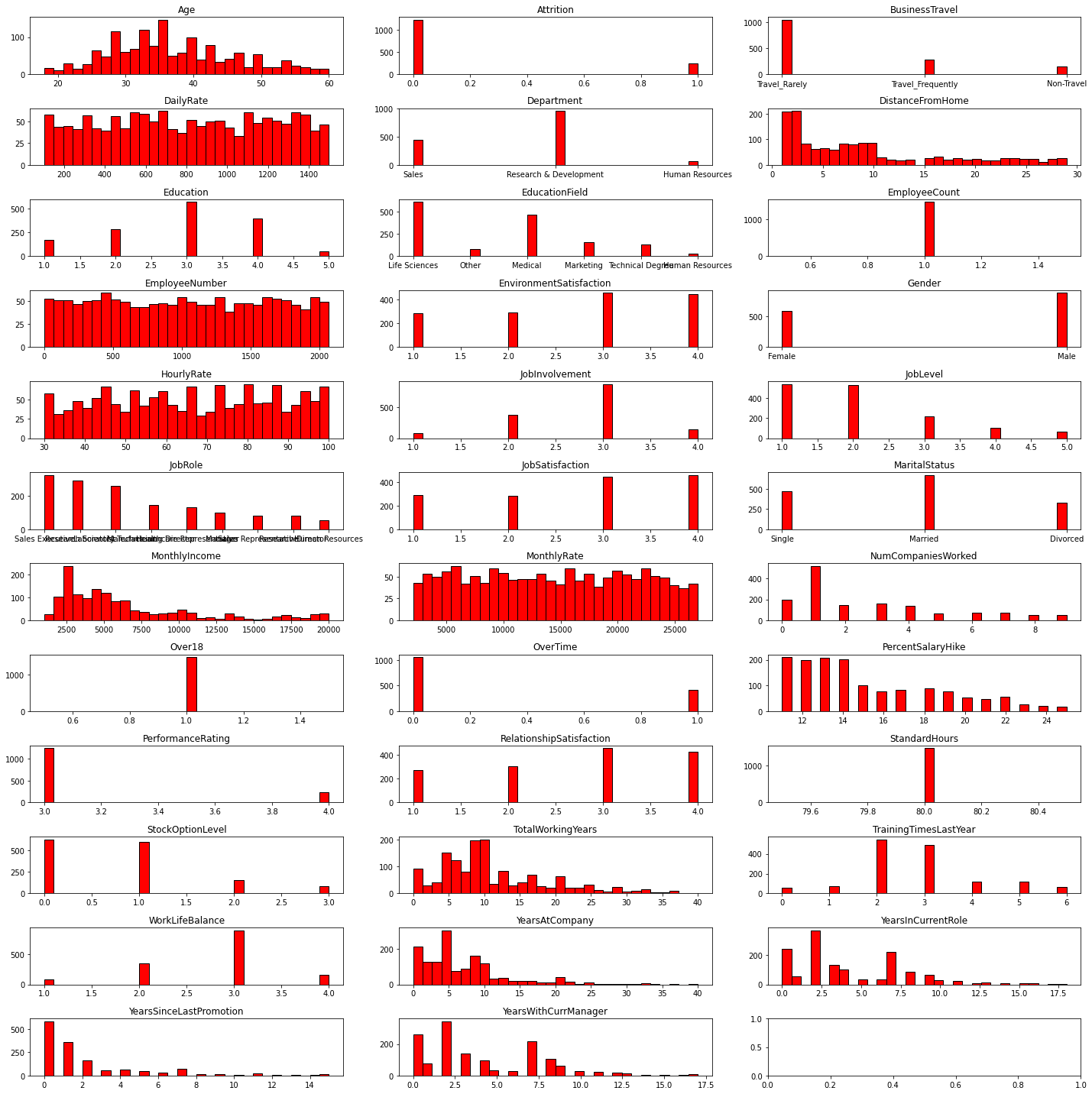
##Plot 4: Bar chart Gender vs. Attrition

#Bar chart Gender vs. Attrition  
pd.crosstab(employee\_df['Gender'],employee\_df['Attrition'] ).plot(kind="bar",figsize=(10,6))  
plt.title("Stay/Left vs Gender")  
plt.xlabel("Stay/Left")  
plt.ylabel("No of people who left based on gender")  
plt.legend(['Stayed', 'Left'])  
plt.xticks(rotation=0)  
plt.tight\_layout()



**bold text**##Plot 5: Histogram of all columns

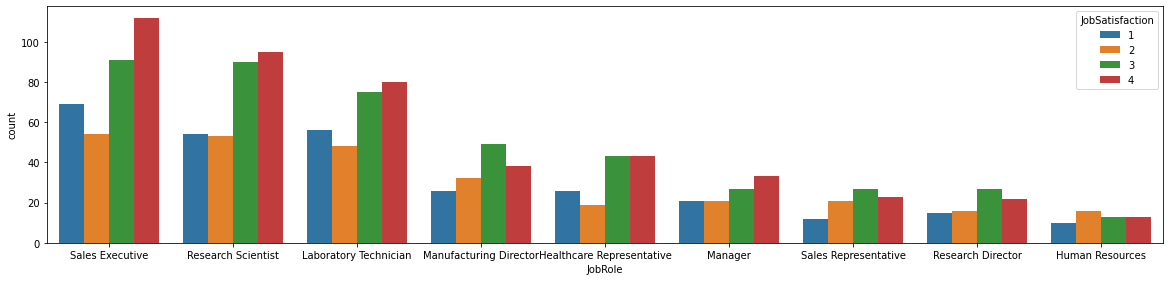
# Several features such as 'MonthlyIncome' and 'TotalWorkingYears' are tail heavy  
# It makes sense to drop 'EmployeeCount' and 'Standardhours' since they do not change from one employee to the other  
plt.subplots(nrows=12,ncols=3, figsize=(20,20))  
count = 0  
for x in employee\_df.columns:  
 count = count+1  
 plt.subplot(12,3,count)  
 plt.hist(employee\_df[x],edgecolor='black', bins=30, color = 'r')   
 plt.title(x)  
plt.tight\_layout()  
plt.show()



##Plot 5 Jobsatisfaction vs. JobRole

plt.figure(figsize=[20,20])  
plt.subplot(411)  
sns.countplot(x = 'JobRole', hue = 'JobSatisfaction', data = employee\_df)

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fd6e13c69d0>



##Dropping unnecessary columns

# 4 colums dropperd: It makes sense to drop 'EmployeeCount' , 'Standardhours' and 'Over18' since they do not change from one employee to the other  
# Let's drop 'EmployeeNumber' as well  
employee\_df.drop(['EmployeeCount', 'StandardHours', 'Over18', 'EmployeeNumber'], axis=1, inplace=True)

# Let's see how many employees left the company!   
left\_df = employee\_df[employee\_df['Attrition'] == 1]  
stayed\_df = employee\_df[employee\_df['Attrition'] == 0]

# Count the number of employees who stayed and left  
# It seems that we are dealing with an imbalanced dataset   
print("Total = ", len(employee\_df))  
  
print("Number of employees who left the company =", len(left\_df))  
print("Percentage of employees who left the company =", 1.\*len(left\_df)/len(employee\_df)\*100.0, "%")  
   
print("Number of employees who did not leave the company (stayed) =", len(stayed\_df))  
print("Percentage of employees who did not leave the company (stayed) =", 1.\*len(stayed\_df)/len(employee\_df)\*100.0, "%")

Total = 1470  
Number of employees who left the company = 237  
Percentage of employees who left the company = 16.122448979591837 %  
Number of employees who did not leave the company (stayed) = 1233  
Percentage of employees who did not leave the company (stayed) = 83.87755102040816 %

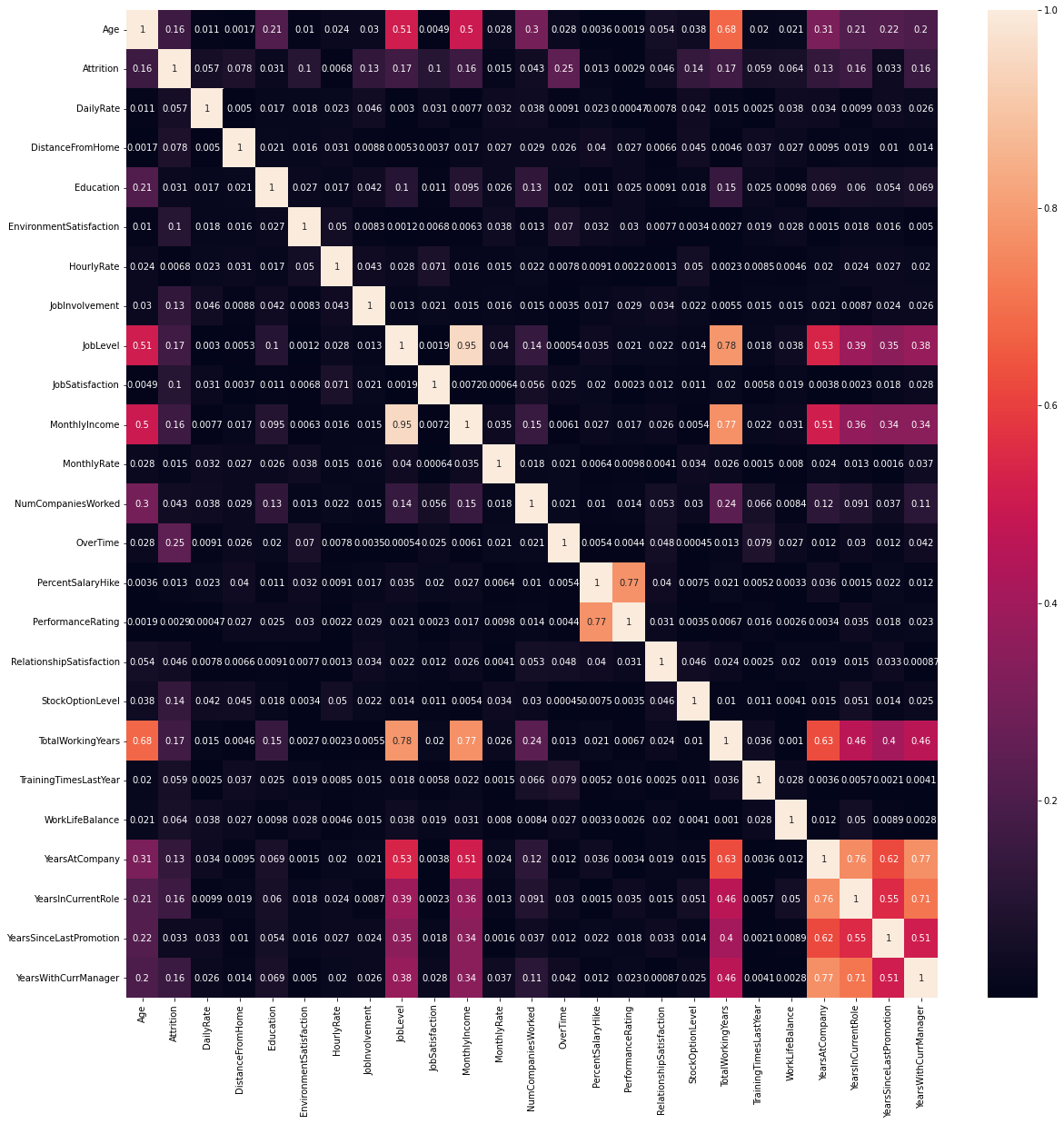
left\_df.describe()  
  
# Let's compare the mean and std of the employees who stayed and left   
# 'age': mean age of the employees who stayed is higher compared to who left  
# 'DailyRate': Rate of employees who stayed is higher  
# 'DistanceFromHome': Employees who stayed live closer to home   
# 'EnvironmentSatisfaction' & 'JobSatisfaction': Employees who stayed are generally more satisifed with their jobs  
# 'StockOptionLevel': Employees who stayed tend to have higher stock option level

stayed\_df.describe()

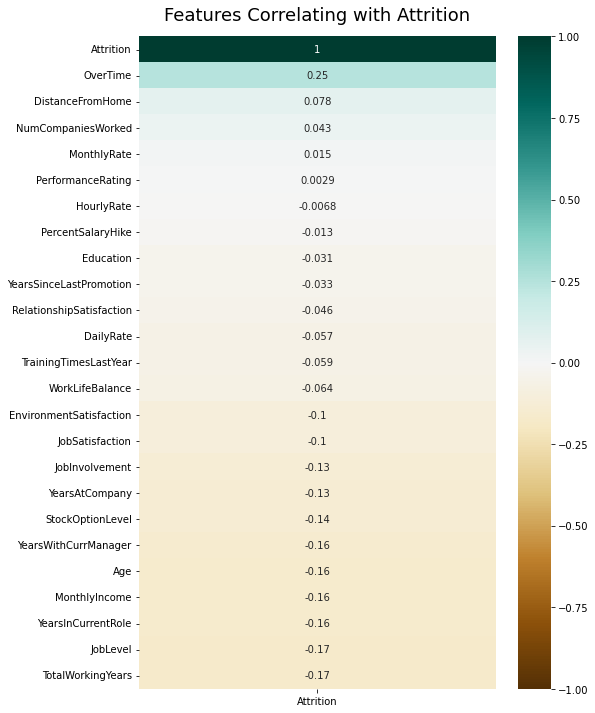
##Plot 6: Correlation matrix

correlations = employee\_df.corr(method='pearson').abs()  
f, ax = plt.subplots(figsize = (20, 20))  
sns.heatmap(correlations, annot = True)  
  
# JobLevel is strongly correlated with TotalWorkingYears  
# MonthlyIncome is strongly correlated with JobLevel  
# MonthlyIncome is strongly correlated with TotalWorkingYears  
# Age is stongly correlated with monthly income

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fd6de949090>



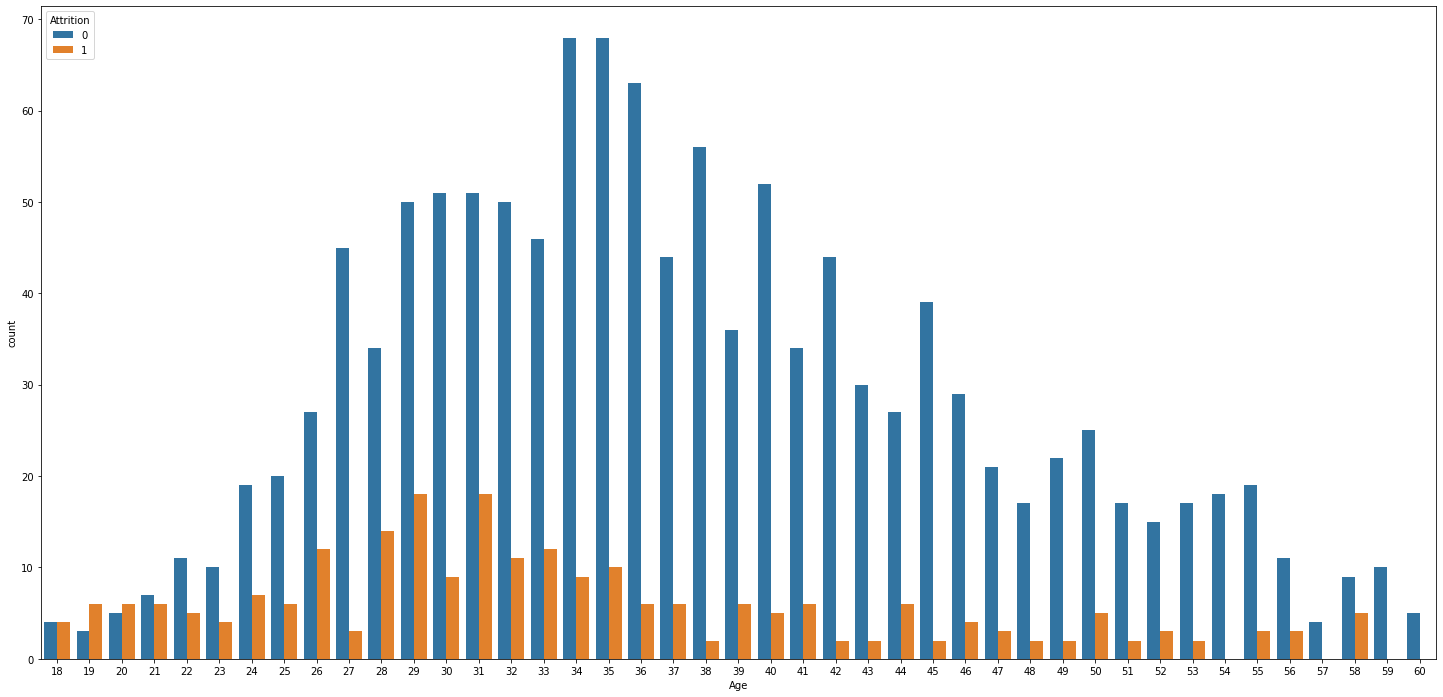
#Correlation of Independent Variables with the Dependent Variable(Attrition)  
plt.figure(figsize=(8, 12))  
heatmap = sns.heatmap(employee\_df.corr()[['Attrition']].sort\_values(by='Attrition', ascending=False), vmin=-1, vmax=1, annot=True, cmap='BrBG')  
heatmap.set\_title('Features Correlating with Attrition', fontdict={'fontsize':18}, pad=16);

­

## Plot 7: Histogram Age vs. Attrition

plt.figure(figsize=[25, 12])  
sns.countplot(x = 'Age', hue = 'Attrition', data = employee\_df)

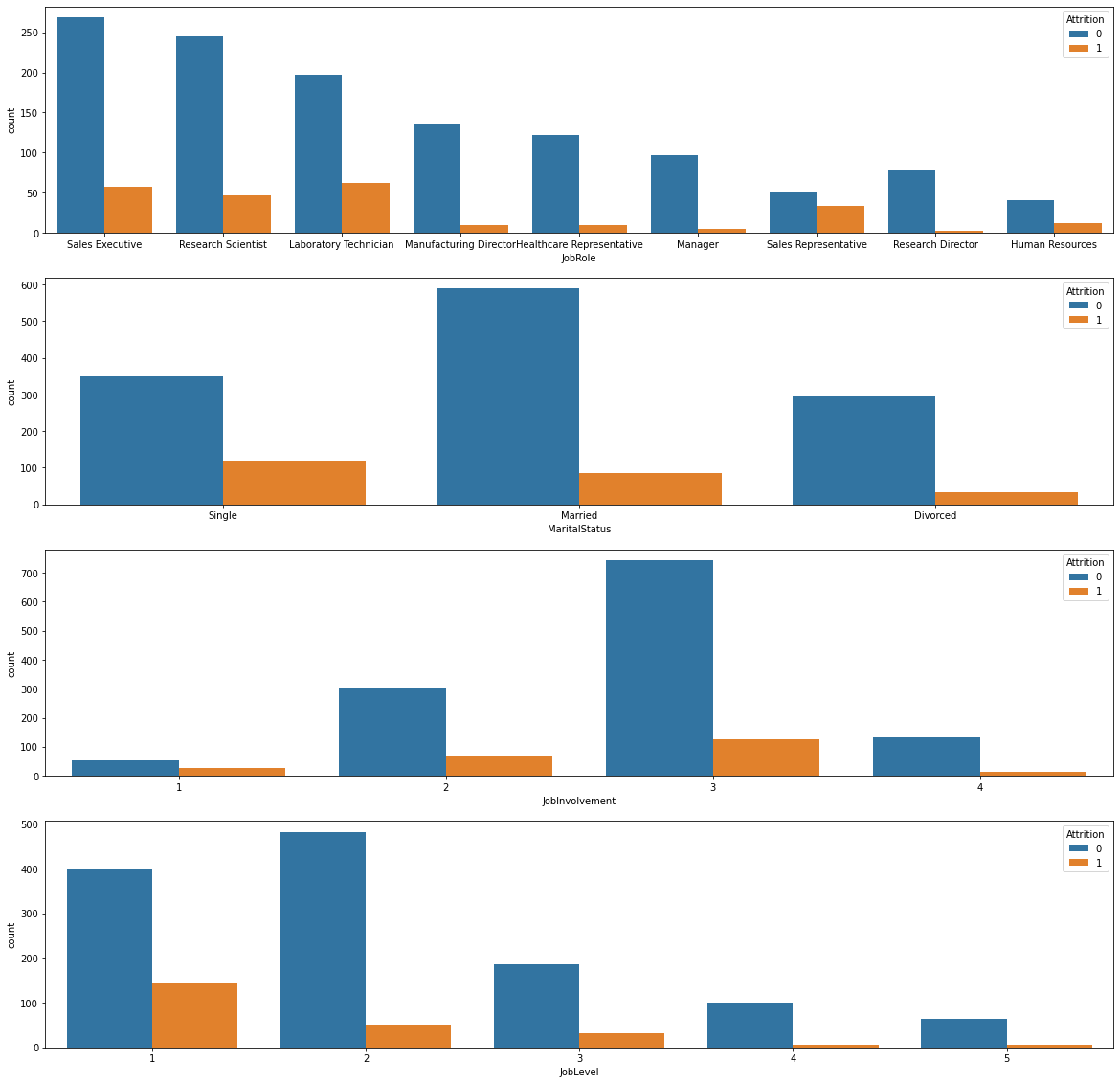
<matplotlib.axes.\_subplots.AxesSubplot at 0x7fd6e2ac3110>



##Plot 8: Histogram Attrition vs. JobRole, MaritalStatus, JobInvolvement, and JobLevel

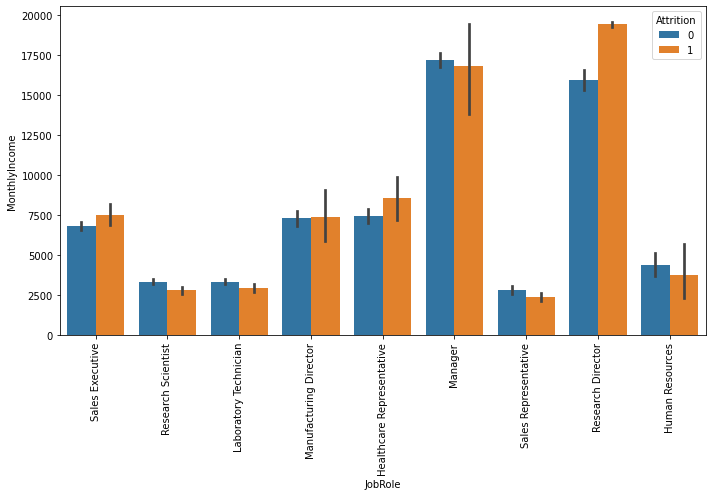
plt.figure(figsize=[20,20])  
plt.subplot(411)  
sns.countplot(x = 'JobRole', hue = 'Attrition', data = employee\_df)  
plt.subplot(412)  
sns.countplot(x = 'MaritalStatus', hue = 'Attrition', data = employee\_df)  
plt.subplot(413)  
sns.countplot(x = 'JobInvolvement', hue = 'Attrition', data = employee\_df)  
plt.subplot(414)  
sns.countplot(x = 'JobLevel', hue = 'Attrition', data = employee\_df)  
  
# Single employees tend to leave compared to married and divorced  
# Sales Representitives tend to leave compared to any other job   
# Less involved employees tend to leave the company   
# Less experienced (low job level) tend to leave the company

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fd6e28f1690>



###Plot 8.1: JobRole vs. MonthlyIncome

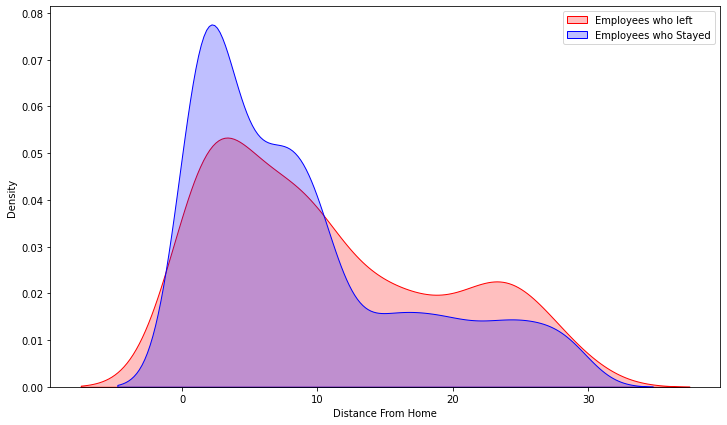
plt.figure(figsize=(10,7))  
sns.barplot(x='JobRole', y='MonthlyIncome', hue='Attrition', data=employee\_df)  
plt.xticks(rotation=90)  
plt.tight\_layout()  
plt.show()



##Plot 9: KDE of DistanceFromHome column

# KDE (Kernel Density Estimate) is used for visualizing the Probability Density of a continuous variable.   
# KDE describes the probability density at different values in a continuous variable.   
  
plt.figure(figsize=(12,7))  
  
sns.kdeplot(left\_df['DistanceFromHome'], label = 'Employees who left', shade = True, color = 'r')  
sns.kdeplot(stayed\_df['DistanceFromHome'], label = 'Employees who Stayed', shade = True, color = 'b')  
  
plt.xlabel('Distance From Home')  
plt.legend()

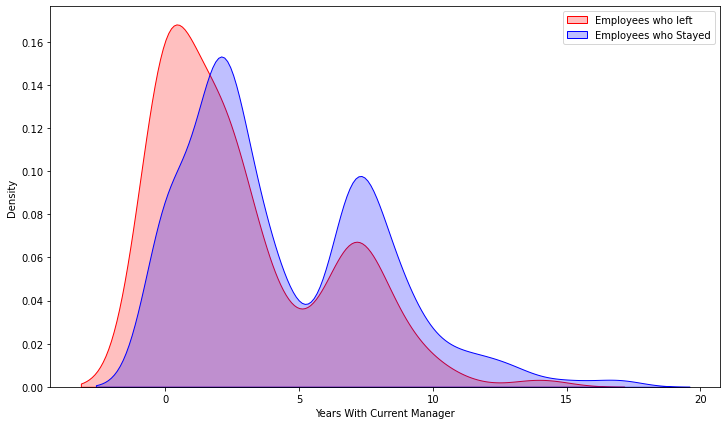
<matplotlib.legend.Legend at 0x7fd6de1b35d0>



##Plot 10: KDE for YearsWithCurrManager column

plt.figure(figsize=(12,7))  
  
sns.kdeplot(left\_df['YearsWithCurrManager'], label = 'Employees who left', shade = True, color = 'r')  
sns.kdeplot(stayed\_df['YearsWithCurrManager'], label = 'Employees who Stayed', shade = True, color = 'b')  
  
plt.xlabel('Years With Current Manager')  
plt.legend()

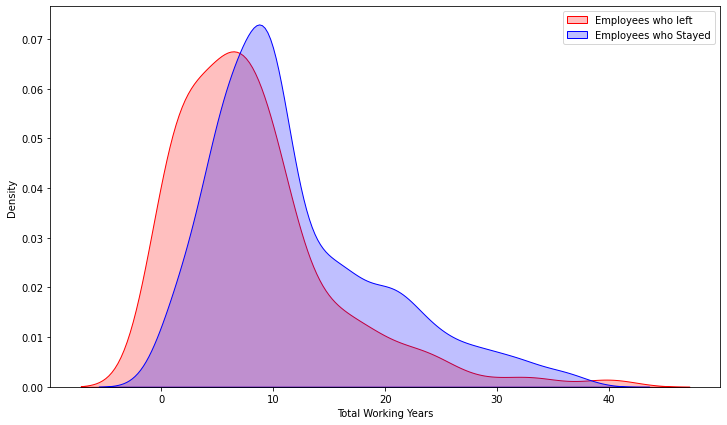
<matplotlib.legend.Legend at 0x7fd6da008910>



##Plot 11: KDE of TotalWorkingYears column

plt.figure(figsize=(12,7))  
  
sns.kdeplot(left\_df['TotalWorkingYears'], shade = True, label = 'Employees who left', color = 'r')  
sns.kdeplot(stayed\_df['TotalWorkingYears'], shade = True, label = 'Employees who Stayed', color = 'b')  
  
plt.xlabel('Total Working Years')  
plt.legend()

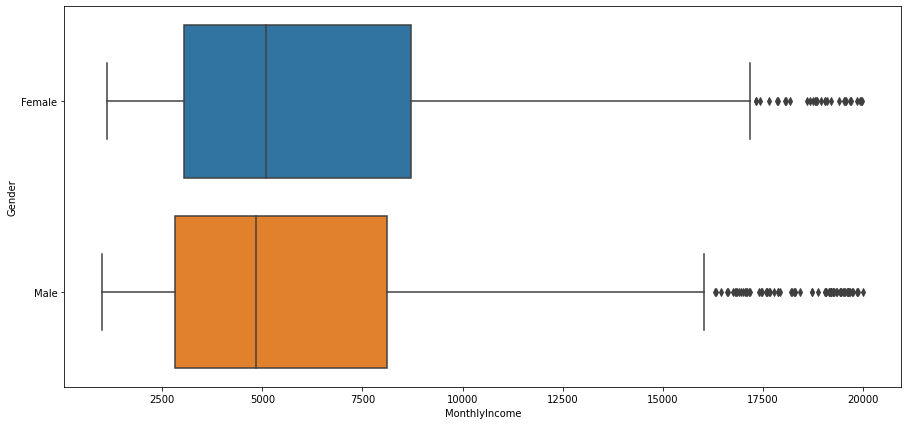
<matplotlib.legend.Legend at 0x7fd6d9464990>



##Plot 12: Box plot MonthlyIncome vs. Gender

# Let's see the Gender vs. Monthly Income  
plt.figure(figsize=(15, 7))  
sns.boxplot(x = 'MonthlyIncome', y = 'Gender', data = employee\_df)

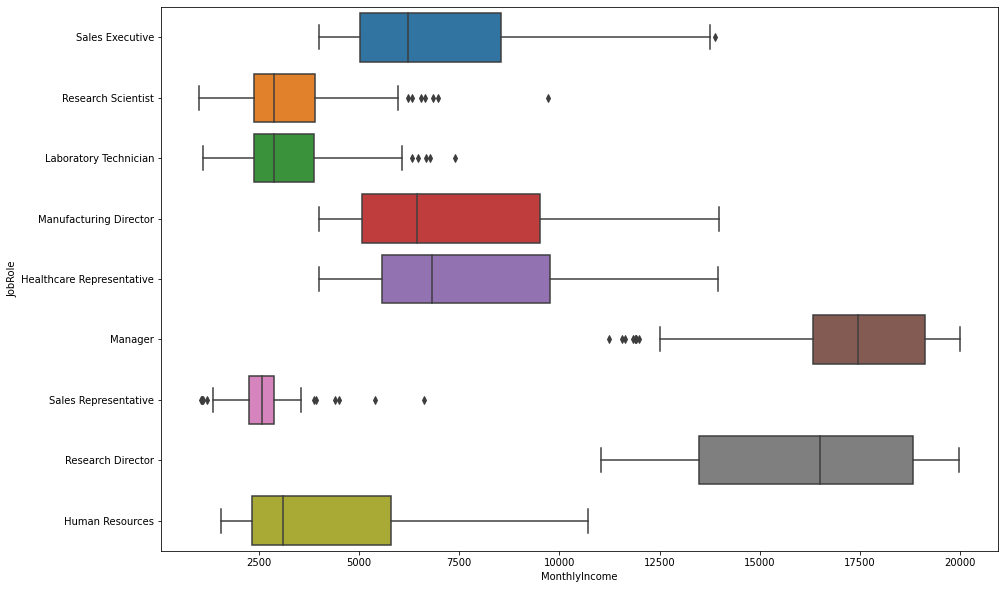
<matplotlib.axes.\_subplots.AxesSubplot at 0x7fd6da16be90>



##Plot 13: Box plot of MonthlyIncome vs. JobRole

# Let's see the monthly income vs. job role  
plt.figure(figsize=(15, 10))  
sns.boxplot(x = 'MonthlyIncome', y = 'JobRole', data = employee\_df)

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fd6d9443d90>



# TASK #3: CREATE TESTING AND TRAINING DATASET

# Data type conversions   
employee\_df['BusinessTravel'] = employee\_df['BusinessTravel'].astype('category')  
employee\_df['Department'] = employee\_df['Department'].astype('category')  
employee\_df['EducationField'] = employee\_df['EducationField'].astype('category')  
employee\_df['Gender'] = employee\_df['Gender'].astype('category')  
employee\_df['JobRole'] = employee\_df['JobRole'].astype('category')  
employee\_df['MaritalStatus'] = employee\_df['MaritalStatus'].astype('category')

employee\_df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1470 entries, 0 to 1469  
Data columns (total 31 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Age 1470 non-null int64   
 1 Attrition 1470 non-null int64   
 2 BusinessTravel 1470 non-null category  
 3 DailyRate 1470 non-null int64   
 4 Department 1470 non-null category  
 5 DistanceFromHome 1470 non-null int64   
 6 Education 1470 non-null int64   
 7 EducationField 1470 non-null category  
 8 EnvironmentSatisfaction 1470 non-null int64   
 9 Gender 1470 non-null category  
 10 HourlyRate 1470 non-null int64   
 11 JobInvolvement 1470 non-null int64   
 12 JobLevel 1470 non-null int64   
 13 JobRole 1470 non-null category  
 14 JobSatisfaction 1470 non-null int64   
 15 MaritalStatus 1470 non-null category  
 16 MonthlyIncome 1470 non-null int64   
 17 MonthlyRate 1470 non-null int64   
 18 NumCompaniesWorked 1470 non-null int64   
 19 OverTime 1470 non-null int64   
 20 PercentSalaryHike 1470 non-null int64   
 21 PerformanceRating 1470 non-null int64   
 22 RelationshipSatisfaction 1470 non-null int64   
 23 StockOptionLevel 1470 non-null int64   
 24 TotalWorkingYears 1470 non-null int64   
 25 TrainingTimesLastYear 1470 non-null int64   
 26 WorkLifeBalance 1470 non-null int64   
 27 YearsAtCompany 1470 non-null int64   
 28 YearsInCurrentRole 1470 non-null int64   
 29 YearsSinceLastPromotion 1470 non-null int64   
 30 YearsWithCurrManager 1470 non-null int64   
dtypes: category(6), int64(25)  
memory usage: 296.9 KB

#List of categories of each categorical columns   
  
print("BusinessTravel: ",employee\_df['BusinessTravel'].cat.categories)  
print("Department: ",employee\_df['Department'].cat.categories)  
print("EducationField: ",employee\_df['EducationField'].cat.categories)  
print("Gender: ",employee\_df['Gender'].cat.categories)  
print("JobRole: ",employee\_df['JobRole'].cat.categories)  
print("MaritalStatus: ",employee\_df['MaritalStatus'].cat.categories)

BusinessTravel: Index(['Non-Travel', 'Travel\_Frequently', 'Travel\_Rarely'], dtype='object')  
Department: Index(['Human Resources', 'Research & Development', 'Sales'], dtype='object')  
EducationField: Index(['Human Resources', 'Life Sciences', 'Marketing', 'Medical', 'Other',  
 'Technical Degree'],  
 dtype='object')  
Gender: Index(['Female', 'Male'], dtype='object')  
JobRole: Index(['Healthcare Representative', 'Human Resources', 'Laboratory Technician',  
 'Manager', 'Manufacturing Director', 'Research Director',  
 'Research Scientist', 'Sales Executive', 'Sales Representative'],  
 dtype='object')  
MaritalStatus: Index(['Divorced', 'Married', 'Single'], dtype='object')

#Approach to encoding categorical values is to use a technique called label encoding. Label encoding is simply converting each value in a column to a number.  
# cat code: The categorical type is a process of factorization. Meaning that each unique value or category is given a incremented integer value starting from zero.  
employee\_df["BusinessTravel"] = employee\_df["BusinessTravel"].cat.codes  
employee\_df['Department']= employee\_df['Department'].cat.codes   
employee\_df['EducationField']= employee\_df['EducationField'].cat.codes   
employee\_df['Gender']= employee\_df['Gender'].cat.codes   
employee\_df['JobRole']= employee\_df['JobRole'].cat.codes   
employee\_df['MaritalStatus']= employee\_df['MaritalStatus'].cat.codes

employee\_df.head(5)

from sklearn.preprocessing import OneHotEncoder  
  
onehotencoder = OneHotEncoder()  
  
X\_cat = employee\_df[['BusinessTravel', 'Department', 'EducationField', 'Gender', 'JobRole', 'MaritalStatus']]  
X\_cat = onehotencoder.fit\_transform(X\_cat).toarray()   
X\_cat = pd.DataFrame(X\_cat)  
X\_numerical = employee\_df[['Age', 'DailyRate', 'DistanceFromHome', 'Education', 'EnvironmentSatisfaction', 'HourlyRate', 'JobInvolvement', 'JobLevel', 'JobSatisfaction', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'OverTime', 'PercentSalaryHike', 'PerformanceRating', 'RelationshipSatisfaction', 'StockOptionLevel', 'TotalWorkingYears' ,'TrainingTimesLastYear' , 'WorkLifeBalance', 'YearsAtCompany' ,'YearsInCurrentRole', 'YearsSinceLastPromotion', 'YearsWithCurrManager']]  
X\_all = pd.concat([X\_cat, X\_numerical], axis = 1)

X\_all.head(5)

# scalling pandas dataframe using Min-Max Normalization  
from sklearn.preprocessing import MinMaxScaler  
  
scaler = MinMaxScaler()  
  
x = scaler.fit\_transform(X\_all)  
x

/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.  
 FutureWarning,  
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.  
 FutureWarning,

array([[0. , 0. , 1. , ..., 0.22222222, 0. ,  
 0.29411765],  
 [0. , 1. , 0. , ..., 0.38888889, 0.06666667,  
 0.41176471],  
 [0. , 0. , 1. , ..., 0. , 0. ,  
 0. ],  
 ...,  
 [0. , 0. , 1. , ..., 0.11111111, 0. ,  
 0.17647059],  
 [0. , 1. , 0. , ..., 0.33333333, 0. ,  
 0.47058824],  
 [0. , 0. , 1. , ..., 0.16666667, 0.06666667,  
 0.11764706]])

x.shape

(1470, 50)

y = employee\_df['Attrition']

from sklearn.model\_selection import train\_test\_split  
  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.25, random\_state=42)

x\_train.shape # Features

(1102, 50)

x\_test.shape # Features

(368, 50)

y\_train.shape # Target variable

(1102,)

y\_test.shape # Target variable

(368,)

# TASK #4: TRAIN AND EVALUATE A LOGISTIC REGRESSION CLASSIFIER

from sklearn.linear\_model import LogisticRegression  
from sklearn.metrics import accuracy\_score  
  
  
model\_logisticRegr = LogisticRegression()  
model\_logisticRegr.fit(x\_train, y\_train)

LogisticRegression()

y\_pred = model\_logisticRegr.predict(x\_test) # Make predictions on entire test data

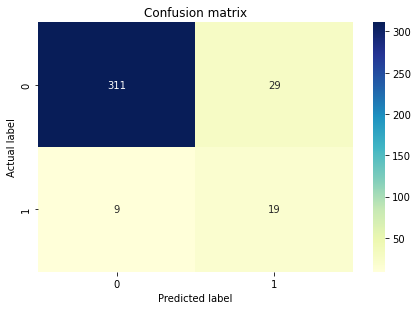
y\_pred

array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,  
 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,  
 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,  
 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,  
 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0])

##Plot 14: Confusion matrix of logistic regression model

from sklearn import metrics  
# Testing Set Performance  
  
#-----confusion matrix------------------  
  
cm = metrics.confusion\_matrix(y\_pred, y\_test, labels=[0,1]) # 1: left and 0: stayed  
  
  
# create heatmap  
sns.heatmap(cm, annot=True, cmap='YlGnBu', fmt='g')  
  
ax.xaxis.set\_label\_position('top')  
  
plt.tight\_layout()  
plt.title('Confusion matrix')  
plt.ylabel('Actual label')  
plt.xlabel('Predicted label')

Text(0.5, 15.0, 'Predicted label')



print(metrics.classification\_report(y\_test, y\_pred))  
# 1: left and 0: stayed

precision recall f1-score support  
  
 0 0.91 0.97 0.94 320  
 1 0.68 0.40 0.50 48  
  
 accuracy 0.90 368  
 macro avg 0.80 0.68 0.72 368  
weighted avg 0.88 0.90 0.88 368

# accuracy table for training and test data  
from prettytable import PrettyTable  
   
# Specifying the Column Names while initializing the Table  
myTable = PrettyTable(["Accuracy on Training Set ", "Accuracy on Testing Set"])  
   
# Adding rows  
myTable.add\_row([model\_logisticRegr.score(x\_train, y\_train) \* 100, 100 \* accuracy\_score(y\_test, y\_pred)])  
  
print(myTable)

+---------------------------+-------------------------+  
| Accuracy on Training Set | Accuracy on Testing Set |  
+---------------------------+-------------------------+  
| 88.83847549909257 | 89.67391304347827 |  
+---------------------------+-------------------------+

#Accuracy  
acc\_logReg= (100 \* (accuracy\_score(y\_test, y\_pred)))  
acc\_logReg

89.67391304347827

# TASK #7: TRAIN AND EVALUATE A RANDOM FOREST CLASSIFIER

from sklearn.ensemble import RandomForestClassifier  
  
model\_randomForest = RandomForestClassifier()  
model\_randomForest.fit(x\_train, y\_train)

RandomForestClassifier()

y\_pred = model\_randomForest.predict(x\_test)

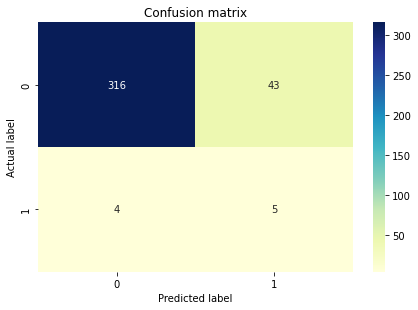
y\_pred

array([0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,  
 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

##Plot 15: Confusion matrix of random forest model

from sklearn import metrics  
# Testing Set Performance  
  
#-----confusion matrix------------------  
  
cm = metrics.confusion\_matrix(y\_pred, y\_test, labels=[0,1]) # 1: left and 0: stayed  
  
  
# create heatmap  
sns.heatmap(cm, annot=True, cmap='YlGnBu', fmt='g')  
  
ax.xaxis.set\_label\_position('top')  
  
plt.tight\_layout()  
plt.title('Confusion matrix')  
plt.ylabel('Actual label')  
plt.xlabel('Predicted label')

Text(0.5, 15.0, 'Predicted label')



print(metrics.classification\_report(y\_test, y\_pred))  
# 1: left and 0: stayed

precision recall f1-score support  
  
 0 0.88 0.99 0.93 320  
 1 0.56 0.10 0.18 48  
  
 accuracy 0.87 368  
 macro avg 0.72 0.55 0.55 368  
weighted avg 0.84 0.87 0.83 368

# accuracy table for training and test data  
from prettytable import PrettyTable  
   
# Specifying the Column Names while initializing the Table  
myTable = PrettyTable(["Accuracy on Training Set ", "Accuracy on Testing Set"])  
   
# Adding rows  
myTable.add\_row([model\_randomForest.score(x\_train, y\_train)\* 100, 100 \* accuracy\_score(y\_test, y\_pred)])  
  
print(myTable)

+---------------------------+-------------------------+  
| Accuracy on Training Set | Accuracy on Testing Set |  
+---------------------------+-------------------------+  
| 99.90925589836661 | 87.22826086956522 |  
+---------------------------+-------------------------+

#Accuracy  
acc\_rForest= (100 \* (accuracy\_score(y\_test, y\_pred)))  
acc\_rForest

87.22826086956522

# TASK #:8 TRAIN AND EVALUATE A DEEP LEARNING MODEL

import tensorflow as tf  
tf.random.set\_seed(42)

nn\_model = tf.keras.models.Sequential()  
nn\_model.add(tf.keras.layers.Dense(units=500, activation='relu', input\_shape=(50, )))  
nn\_model.add(tf.keras.layers.Dense(units=500, activation='relu'))  
nn\_model.add(tf.keras.layers.Dense(units=500, activation='relu'))  
nn\_model.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

nn\_model.summary()

Model: "sequential\_5"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 Layer (type) Output Shape Param #   
=================================================================  
 dense\_20 (Dense) (None, 500) 25500   
   
 dense\_21 (Dense) (None, 500) 250500   
   
 dense\_22 (Dense) (None, 500) 250500   
   
 dense\_23 (Dense) (None, 1) 501   
   
=================================================================  
Total params: 527,001  
Trainable params: 527,001  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

nn\_model.compile(optimizer='Adam', loss='binary\_crossentropy', metrics = ['accuracy'])

epochs\_hist = nn\_model.fit(x\_train, y\_train, epochs = 100, batch\_size = 50)

Epoch 1/100  
23/23 [==============================] - 1s 10ms/step - loss: 0.4420 - accuracy: 0.8167  
Epoch 2/100  
23/23 [==============================] - 0s 10ms/step - loss: 0.3395 - accuracy: 0.8630  
Epoch 3/100  
23/23 [==============================] - 0s 10ms/step - loss: 0.3045 - accuracy: 0.8711  
Epoch 4/100  
23/23 [==============================] - 0s 10ms/step - loss: 0.2750 - accuracy: 0.8956  
Epoch 5/100  
23/23 [==============================] - 0s 9ms/step - loss: 0.2466 - accuracy: 0.9011  
Epoch 6/100  
23/23 [==============================] - 0s 9ms/step - loss: 0.2317 - accuracy: 0.9002  
Epoch 7/100  
23/23 [==============================] - 0s 11ms/step - loss: 0.1919 - accuracy: 0.9229  
Epoch 8/100  
23/23 [==============================] - 0s 9ms/step - loss: 0.1761 - accuracy: 0.9310  
Epoch 9/100  
23/23 [==============================] - 0s 10ms/step - loss: 0.2085 - accuracy: 0.9201  
Epoch 10/100  
23/23 [==============================] - 0s 14ms/step - loss: 0.1362 - accuracy: 0.9537  
Epoch 11/100  
23/23 [==============================] - 0s 13ms/step - loss: 0.0901 - accuracy: 0.9746  
Epoch 12/100  
23/23 [==============================] - 0s 12ms/step - loss: 0.0851 - accuracy: 0.9637  
Epoch 13/100  
23/23 [==============================] - 0s 10ms/step - loss: 0.0646 - accuracy: 0.9773  
Epoch 14/100  
23/23 [==============================] - 0s 10ms/step - loss: 0.0373 - accuracy: 0.9891  
Epoch 15/100  
23/23 [==============================] - 0s 10ms/step - loss: 0.0510 - accuracy: 0.9855  
Epoch 16/100  
23/23 [==============================] - 0s 9ms/step - loss: 0.0315 - accuracy: 0.9909  
Epoch 17/100  
23/23 [==============================] - 0s 11ms/step - loss: 0.0165 - accuracy: 0.9964  
Epoch 18/100  
23/23 [==============================] - 0s 12ms/step - loss: 0.0091 - accuracy: 0.9973  
Epoch 19/100  
23/23 [==============================] - 0s 14ms/step - loss: 0.0071 - accuracy: 0.9991  
Epoch 20/100  
23/23 [==============================] - 0s 12ms/step - loss: 0.0042 - accuracy: 0.9991  
Epoch 21/100  
23/23 [==============================] - 0s 12ms/step - loss: 0.0066 - accuracy: 0.9982  
Epoch 22/100  
23/23 [==============================] - 0s 12ms/step - loss: 0.0101 - accuracy: 0.9973  
Epoch 23/100  
23/23 [==============================] - 0s 12ms/step - loss: 0.0213 - accuracy: 0.9918  
Epoch 24/100  
23/23 [==============================] - 0s 11ms/step - loss: 0.0439 - accuracy: 0.9864  
Epoch 25/100  
23/23 [==============================] - 0s 12ms/step - loss: 0.0425 - accuracy: 0.9846  
Epoch 26/100  
23/23 [==============================] - 0s 13ms/step - loss: 0.0219 - accuracy: 0.9955  
Epoch 27/100  
23/23 [==============================] - 0s 11ms/step - loss: 0.0233 - accuracy: 0.9918  
Epoch 28/100  
23/23 [==============================] - 0s 11ms/step - loss: 0.0411 - accuracy: 0.9864  
Epoch 29/100  
23/23 [==============================] - 0s 12ms/step - loss: 0.0175 - accuracy: 0.9946  
Epoch 30/100  
23/23 [==============================] - 0s 12ms/step - loss: 0.0140 - accuracy: 0.9964  
Epoch 31/100  
23/23 [==============================] - 0s 11ms/step - loss: 0.0042 - accuracy: 1.0000  
Epoch 32/100  
23/23 [==============================] - 0s 10ms/step - loss: 0.0023 - accuracy: 1.0000  
Epoch 33/100  
23/23 [==============================] - 0s 10ms/step - loss: 0.0211 - accuracy: 0.9973  
Epoch 34/100  
23/23 [==============================] - 0s 10ms/step - loss: 0.0366 - accuracy: 0.9846  
Epoch 35/100  
23/23 [==============================] - 0s 9ms/step - loss: 0.0241 - accuracy: 0.9946  
Epoch 36/100  
23/23 [==============================] - 0s 9ms/step - loss: 0.0278 - accuracy: 0.9909  
Epoch 37/100  
23/23 [==============================] - 0s 10ms/step - loss: 0.0080 - accuracy: 0.9982  
Epoch 38/100  
23/23 [==============================] - 0s 10ms/step - loss: 0.0028 - accuracy: 0.9991  
Epoch 39/100  
23/23 [==============================] - 0s 10ms/step - loss: 0.0016 - accuracy: 1.0000  
Epoch 40/100  
23/23 [==============================] - 0s 11ms/step - loss: 4.1813e-04 - accuracy: 1.0000  
Epoch 41/100  
23/23 [==============================] - 0s 11ms/step - loss: 3.1066e-04 - accuracy: 1.0000  
Epoch 42/100  
23/23 [==============================] - 0s 12ms/step - loss: 2.4129e-04 - accuracy: 1.0000  
Epoch 43/100  
23/23 [==============================] - 0s 12ms/step - loss: 1.9717e-04 - accuracy: 1.0000  
Epoch 44/100  
23/23 [==============================] - 0s 10ms/step - loss: 1.6520e-04 - accuracy: 1.0000  
Epoch 45/100  
23/23 [==============================] - 0s 10ms/step - loss: 1.4110e-04 - accuracy: 1.0000  
Epoch 46/100  
23/23 [==============================] - 0s 10ms/step - loss: 1.2210e-04 - accuracy: 1.0000  
Epoch 47/100  
23/23 [==============================] - 0s 10ms/step - loss: 1.0553e-04 - accuracy: 1.0000  
Epoch 48/100  
23/23 [==============================] - 0s 11ms/step - loss: 9.3435e-05 - accuracy: 1.0000  
Epoch 49/100  
23/23 [==============================] - 0s 11ms/step - loss: 8.2508e-05 - accuracy: 1.0000  
Epoch 50/100  
23/23 [==============================] - 0s 12ms/step - loss: 7.3152e-05 - accuracy: 1.0000  
Epoch 51/100  
23/23 [==============================] - 0s 11ms/step - loss: 6.5872e-05 - accuracy: 1.0000  
Epoch 52/100  
23/23 [==============================] - 0s 11ms/step - loss: 5.9653e-05 - accuracy: 1.0000  
Epoch 53/100  
23/23 [==============================] - 0s 12ms/step - loss: 5.3687e-05 - accuracy: 1.0000  
Epoch 54/100  
23/23 [==============================] - 0s 12ms/step - loss: 4.8767e-05 - accuracy: 1.0000  
Epoch 55/100  
23/23 [==============================] - 0s 11ms/step - loss: 4.4522e-05 - accuracy: 1.0000  
Epoch 56/100  
23/23 [==============================] - 0s 10ms/step - loss: 3.9903e-05 - accuracy: 1.0000  
Epoch 57/100  
23/23 [==============================] - 0s 10ms/step - loss: 3.6358e-05 - accuracy: 1.0000  
Epoch 58/100  
23/23 [==============================] - 0s 10ms/step - loss: 3.3220e-05 - accuracy: 1.0000  
Epoch 59/100  
23/23 [==============================] - 0s 9ms/step - loss: 3.0487e-05 - accuracy: 1.0000  
Epoch 60/100  
23/23 [==============================] - 0s 10ms/step - loss: 2.8108e-05 - accuracy: 1.0000  
Epoch 61/100  
23/23 [==============================] - 0s 10ms/step - loss: 2.5951e-05 - accuracy: 1.0000  
Epoch 62/100  
23/23 [==============================] - 0s 11ms/step - loss: 2.4047e-05 - accuracy: 1.0000  
Epoch 63/100  
23/23 [==============================] - 0s 11ms/step - loss: 2.2197e-05 - accuracy: 1.0000  
Epoch 64/100  
23/23 [==============================] - 0s 9ms/step - loss: 2.0551e-05 - accuracy: 1.0000  
Epoch 65/100  
23/23 [==============================] - 0s 10ms/step - loss: 1.9197e-05 - accuracy: 1.0000  
Epoch 66/100  
23/23 [==============================] - 0s 9ms/step - loss: 1.7935e-05 - accuracy: 1.0000  
Epoch 67/100  
23/23 [==============================] - 0s 10ms/step - loss: 1.6864e-05 - accuracy: 1.0000  
Epoch 68/100  
23/23 [==============================] - 0s 10ms/step - loss: 1.5813e-05 - accuracy: 1.0000  
Epoch 69/100  
23/23 [==============================] - 0s 10ms/step - loss: 1.4833e-05 - accuracy: 1.0000  
Epoch 70/100  
23/23 [==============================] - 0s 10ms/step - loss: 1.3997e-05 - accuracy: 1.0000  
Epoch 71/100  
23/23 [==============================] - 0s 10ms/step - loss: 1.3198e-05 - accuracy: 1.0000  
Epoch 72/100  
23/23 [==============================] - 0s 10ms/step - loss: 1.2291e-05 - accuracy: 1.0000  
Epoch 73/100  
23/23 [==============================] - 0s 17ms/step - loss: 1.1484e-05 - accuracy: 1.0000  
Epoch 74/100  
23/23 [==============================] - 0s 20ms/step - loss: 1.0873e-05 - accuracy: 1.0000  
Epoch 75/100  
23/23 [==============================] - 1s 35ms/step - loss: 1.0292e-05 - accuracy: 1.0000  
Epoch 76/100  
23/23 [==============================] - 1s 46ms/step - loss: 9.7401e-06 - accuracy: 1.0000  
Epoch 77/100  
23/23 [==============================] - 1s 40ms/step - loss: 9.2918e-06 - accuracy: 1.0000  
Epoch 78/100  
23/23 [==============================] - 1s 44ms/step - loss: 8.8392e-06 - accuracy: 1.0000  
Epoch 79/100  
23/23 [==============================] - 1s 49ms/step - loss: 8.4137e-06 - accuracy: 1.0000  
Epoch 80/100  
23/23 [==============================] - 1s 36ms/step - loss: 8.0339e-06 - accuracy: 1.0000  
Epoch 81/100  
23/23 [==============================] - 1s 47ms/step - loss: 7.6776e-06 - accuracy: 1.0000  
Epoch 82/100  
23/23 [==============================] - 1s 44ms/step - loss: 7.3038e-06 - accuracy: 1.0000  
Epoch 83/100  
23/23 [==============================] - 1s 36ms/step - loss: 6.9799e-06 - accuracy: 1.0000  
Epoch 84/100  
23/23 [==============================] - 1s 23ms/step - loss: 6.6955e-06 - accuracy: 1.0000  
Epoch 85/100  
23/23 [==============================] - 0s 14ms/step - loss: 6.4289e-06 - accuracy: 1.0000  
Epoch 86/100  
23/23 [==============================] - 0s 9ms/step - loss: 6.1395e-06 - accuracy: 1.0000  
Epoch 87/100  
23/23 [==============================] - 0s 11ms/step - loss: 5.8982e-06 - accuracy: 1.0000  
Epoch 88/100  
23/23 [==============================] - 0s 10ms/step - loss: 5.6769e-06 - accuracy: 1.0000  
Epoch 89/100  
23/23 [==============================] - 0s 12ms/step - loss: 5.4786e-06 - accuracy: 1.0000  
Epoch 90/100  
23/23 [==============================] - 0s 12ms/step - loss: 5.2489e-06 - accuracy: 1.0000  
Epoch 91/100  
23/23 [==============================] - 0s 13ms/step - loss: 5.0682e-06 - accuracy: 1.0000  
Epoch 92/100  
23/23 [==============================] - 0s 12ms/step - loss: 4.8731e-06 - accuracy: 1.0000  
Epoch 93/100  
23/23 [==============================] - 0s 14ms/step - loss: 4.6952e-06 - accuracy: 1.0000  
Epoch 94/100  
23/23 [==============================] - 0s 12ms/step - loss: 4.5166e-06 - accuracy: 1.0000  
Epoch 95/100  
23/23 [==============================] - 0s 13ms/step - loss: 4.3537e-06 - accuracy: 1.0000  
Epoch 96/100  
23/23 [==============================] - 0s 12ms/step - loss: 4.1965e-06 - accuracy: 1.0000  
Epoch 97/100  
23/23 [==============================] - 0s 11ms/step - loss: 4.0796e-06 - accuracy: 1.0000  
Epoch 98/100  
23/23 [==============================] - 0s 11ms/step - loss: 3.9197e-06 - accuracy: 1.0000  
Epoch 99/100  
23/23 [==============================] - 0s 12ms/step - loss: 3.7891e-06 - accuracy: 1.0000  
Epoch 100/100  
23/23 [==============================] - 0s 13ms/step - loss: 3.6588e-06 - accuracy: 1.0000

y\_pred = nn\_model.predict(x\_test)  
y\_pred = (y\_pred > 0.5)

12/12 [==============================] - 0s 4ms/step

y\_pred

array([[False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [False],  
 [ True],  
 [ True],  
 [False],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [False],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [False],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [False],  
 [False],  
 [False],  
 [False],  
 [ True],  
 [ True],  
 [False],  
 [False],  
 [ True],  
 [False]])

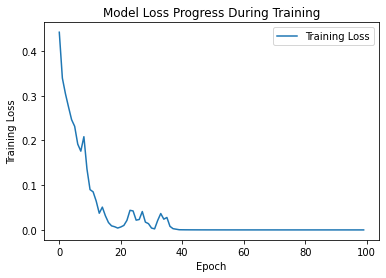
epochs\_hist.history.keys()

dict\_keys(['loss', 'accuracy'])

##Plot 16: Line chart of model loss of epochs\_hist variable

plt.plot(epochs\_hist.history['loss'])  
plt.title('Model Loss Progress During Training')  
plt.xlabel('Epoch')  
plt.ylabel('Training Loss')  
plt.legend(['Training Loss'])

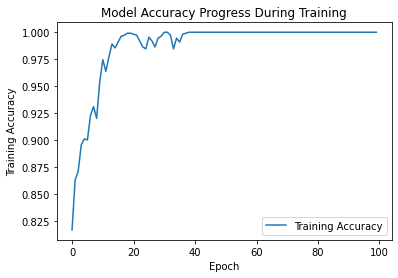
<matplotlib.legend.Legend at 0x7fd6daa56610>



##Plot 17: Line chart of model accuracy of epochs\_hist variable

plt.plot(epochs\_hist.history['accuracy'])  
plt.title('Model Accuracy Progress During Training')  
plt.xlabel('Epoch')  
plt.ylabel('Training Accuracy')  
plt.legend(['Training Accuracy'])

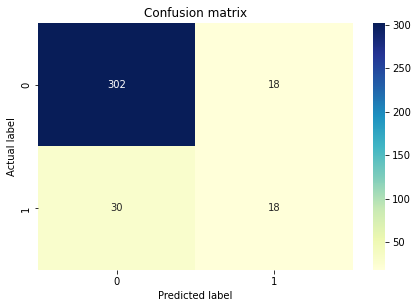
<matplotlib.legend.Legend at 0x7fd6dd1cb1d0>



##Plot 18: Confusion matrix of NN model

from sklearn import metrics  
# Testing Set Performance  
  
#-----confusion matrix------------------  
  
cm = metrics.confusion\_matrix( y\_test,y\_pred, labels=[0,1]) # 1: left and 0: stayed  
  
  
# create heatmap  
sns.heatmap(cm, annot=True, cmap='YlGnBu', fmt='g')  
  
ax.xaxis.set\_label\_position('top')  
  
plt.tight\_layout()  
plt.title('Confusion matrix')  
plt.ylabel('Actual label')  
plt.xlabel('Predicted label')

Text(0.5, 15.0, 'Predicted label')



print(metrics.classification\_report(y\_test, y\_pred))

precision recall f1-score support  
  
 0 0.91 0.94 0.93 320  
 1 0.50 0.38 0.43 48  
  
 accuracy 0.87 368  
 macro avg 0.70 0.66 0.68 368  
weighted avg 0.86 0.87 0.86 368

#Accuracy  
acc\_nn= (100 \* (accuracy\_score(y\_test, y\_pred)))  
acc\_nn

86.95652173913044

#Model comparision

# accuracy table for each model on testing dataset  
from prettytable import PrettyTable  
   
# Specifying the Column Names while initializing the Table  
myTable = PrettyTable(["Models ", "Accuracy on Testing Set"])  
   
# Adding rows  
myTable.add\_row(["Logistic Regression", acc\_logReg])  
myTable.add\_row(["Random Forest", acc\_rForest])  
myTable.add\_row(["Artificial Neural Net", acc\_nn])  
  
  
print(myTable)

+-----------------------+-------------------------+  
| Models | Accuracy on Testing Set |  
+-----------------------+-------------------------+  
| Logistic Regression | 89.67391304347827 |  
| Random Forest | 87.22826086956522 |  
| Artificial Neural Net | 86.95652173913044 |  
+-----------------------+-------------------------+

#Plot 19: ROC curve of models

# Receiver Operating Characteristic(ROC) curve is a plot of the true positive rate against the false positive rate.   
#It shows the tradeoff between sensitivity and specificity.  
  
# ROC curve: Logistic Regression  
plt.figure(figsize=(10,6))  
y\_pred\_proba = model\_logisticRegr.predict\_proba(x\_test)[::,1]  
fpr, tpr, \_ = metrics.roc\_curve(y\_test, y\_pred\_proba)  
auc = metrics.roc\_auc\_score(y\_test, y\_pred\_proba)  
plt.plot(fpr, tpr, label='Logistic Regression (area = %.2f)' %auc)  
  
  
# ROC curve: Random Forest  
y\_pred\_proba = model\_randomForest.predict\_proba(x\_test)[::,1]  
fpr, tpr, \_ = metrics.roc\_curve(y\_test, y\_pred\_proba)  
auc = metrics.roc\_auc\_score(y\_test, y\_pred\_proba)  
  
plt.plot(fpr, tpr, label='Random Forest (area = %.2f)' %auc)  
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Reference line')  
plt.xlabel('False positive rate(Specificity)')  
plt.ylabel('True positive rate(Sensitivity)')  
plt.legend(loc="lower right")  
plt.title("ROC curve for model evaluation")  
plt.show()  
  
  
  
  
#AUC score for the case is 0.81. AUC score 1 represents a perfect classifier, and 0.5 represents a worthless classifier.

