IBM Attrition data analysis

This project is for the IBM dataset which contains the attrition related data of employees of 170+ countries; the objective is to build a machine learning model which will predict if an employee is likely to attrite or not

In [74]:
```
#importing the necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

In [75]:
```
#importing the dataset
ibm_data = pd.read_csv("IBM_Attrition_Data.csv")
```

Examining the dataset

In [76]:
```
#examining the columns in the dataset
ibm_data.columns
```

Out[76]:
```
Index(['Age', 'Attrition', 'Department', 'DistanceFromHome', 'Education',
       'EducationField', 'EnvironmentSatisfaction', 'JobSatisfaction',
       'MaritalStatus', 'MonthlyIncome', 'NumCompaniesWorked',
       'WorkLifeBalance', 'YearsAtCompany'],
      dtype='object')
```

In [77]:
```
#viewing the first 5 records of the dataset
ibm_data.head(5)
```

Out[77]:

	Age	Attrition	Department	DistanceFromHome	Education	EducationField	Environment
0	41	Yes	Sales	1	2	Life Sciences	
1	49	No	Research & Development	8	1	Life Sciences	
2	37	Yes	Research & Development	2	2	Other	
3	33	No	Research & Development	3	4	Life Sciences	
4	27	No	Research & Development	2	1	Medical	

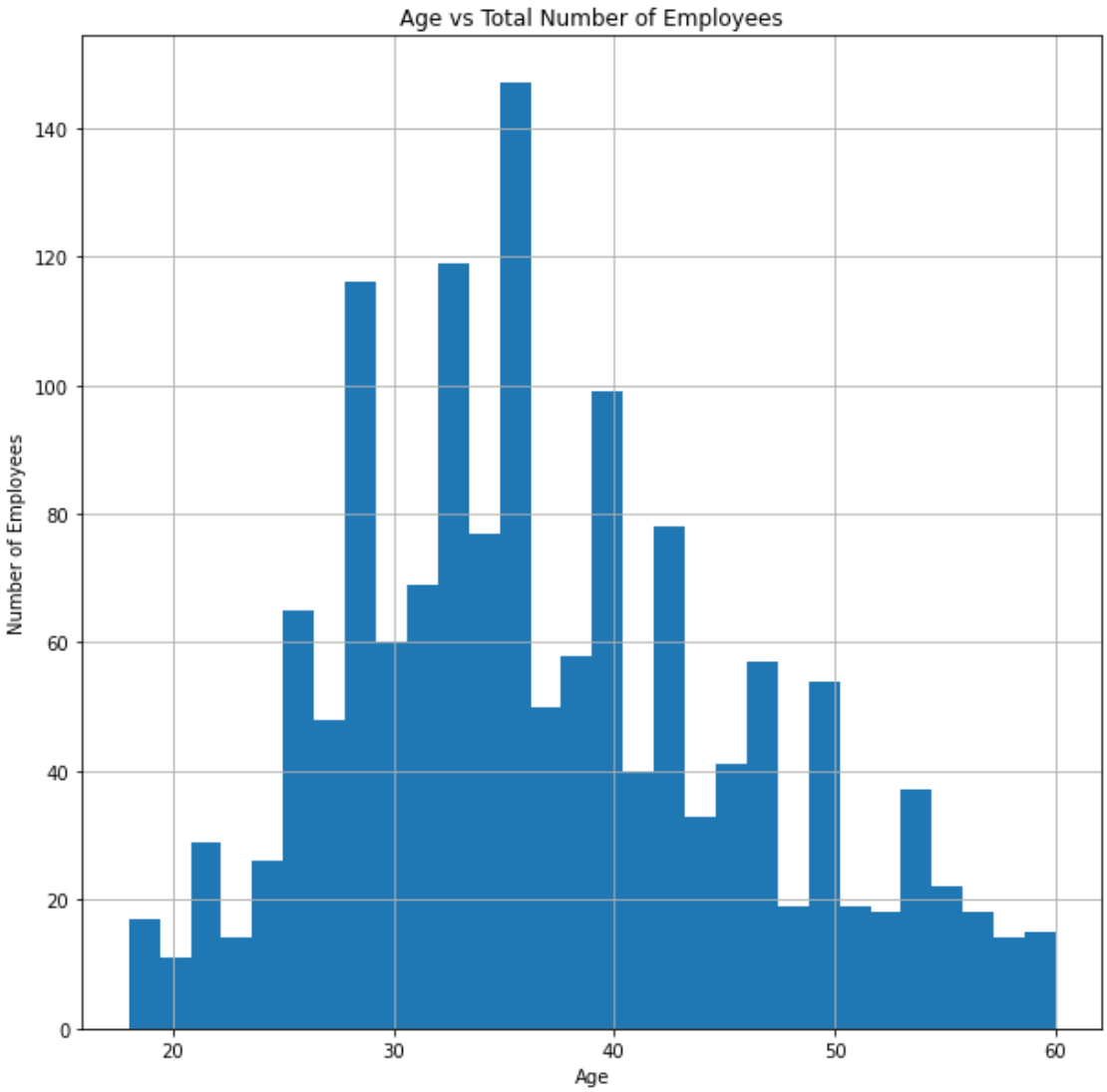
In [78]:
```
#viewing the total number of records of the dataset
ibm_data.shape
```

Out[78]:
```
(1470, 13)
```

Visualising the dataset

In [79]:
```
#age distribution via histogram
plt.figure(figsize=(10,10))
```

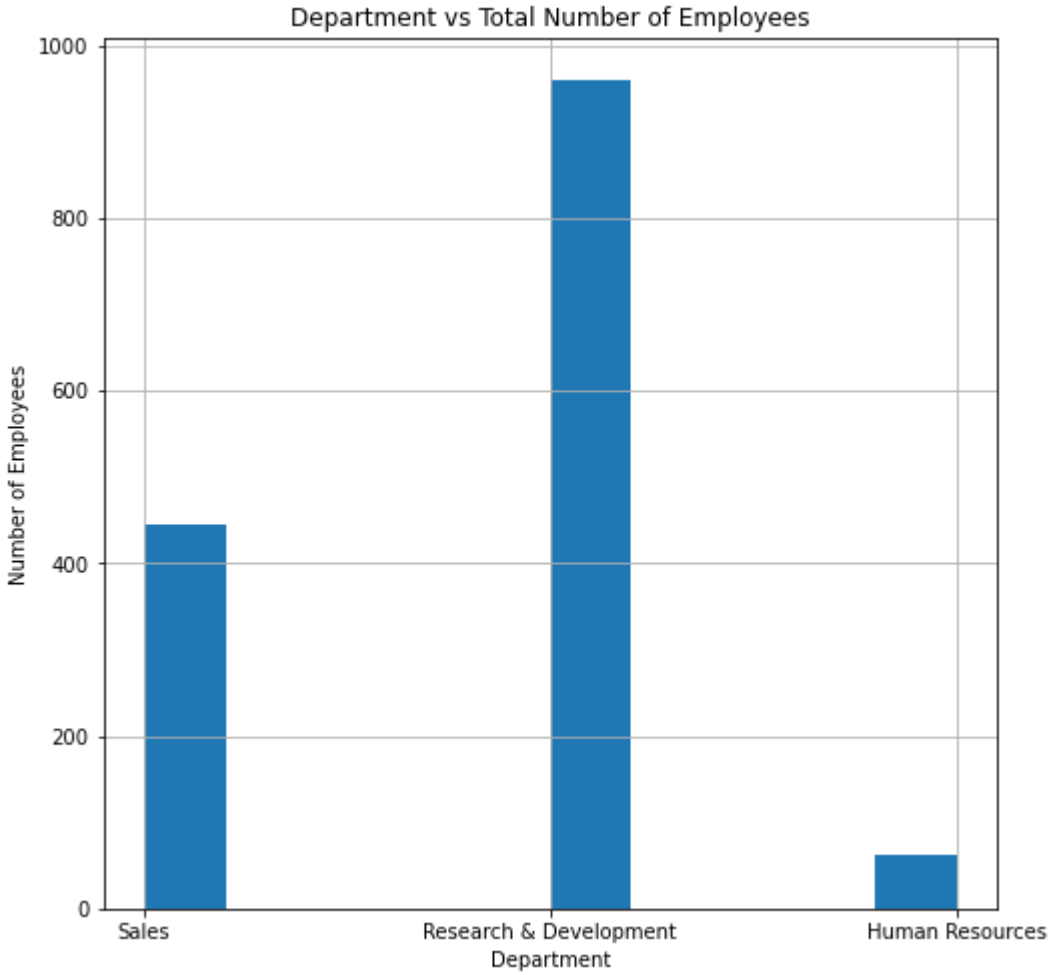
```
ibm_data['Age'].hist(bins=30)
plt.title("Age vs Total Number of Employees")
plt.xlabel("Age")
plt.ylabel("Number of Employees")
plt.show()
```



In [80]:
```
ibm_data["Department"].value_counts()
```

Out[80]:
```
Research & Development    961
Sales                     446
Human Resources            63
Name: Department, dtype: int64
```

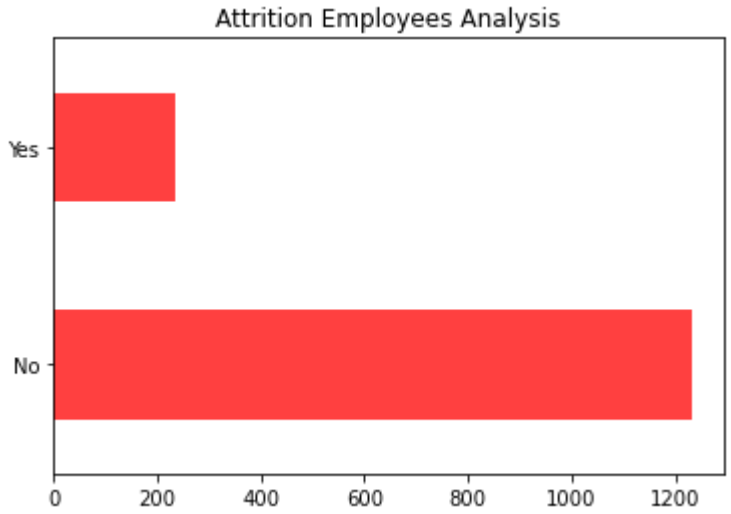
In [81]:
```
#department distribution via barchart
plt.figure(figsize=(8,8))
ibm_data['Department'].hist()
plt.title("Department vs Total Number of Employees")
plt.xlabel("Department")
plt.ylabel("Number of Employees")
plt.show()
```



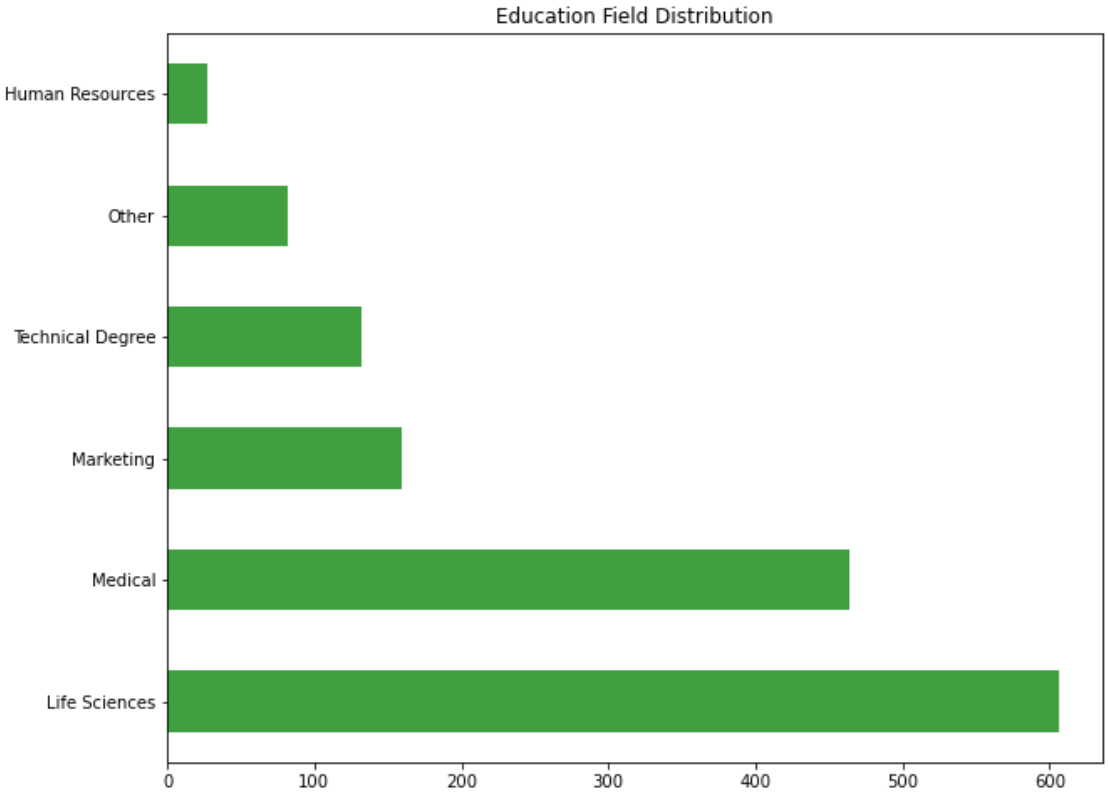
In [82]:
```
#attrition employees analysis
ibm_data["Attrition"].value_counts()
```

Out[82]:
```
No     1233
Yes     237
Name: Attrition, dtype: int64
```

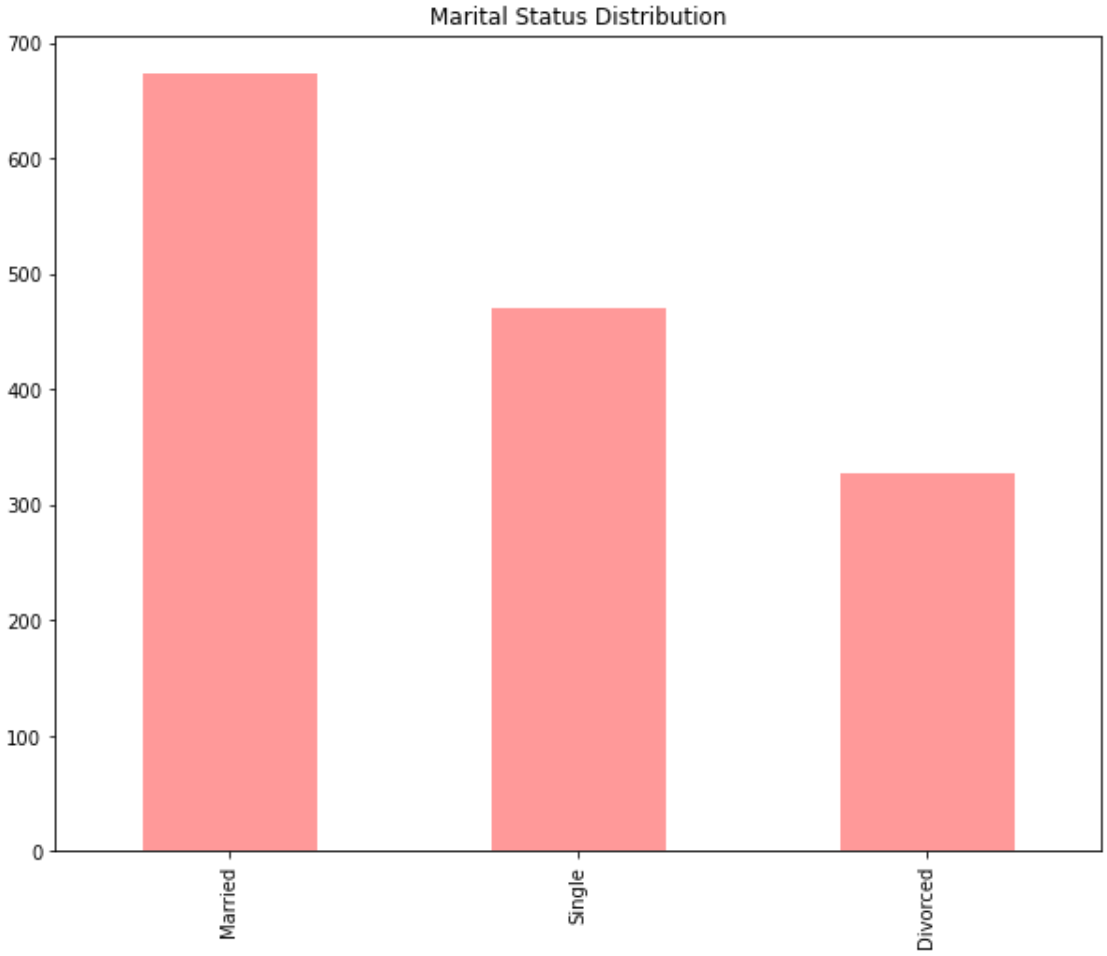
In [83]:
```
ibm_data.Attrition.value_counts().plot(kind='barh',color='r',alpha=
plt.title("Attrition Employees Analysis")
plt.show()
```



In [84]:
```
plt.figure(figsize=(10,8))
ibm_data.EducationField.value_counts().plot(kind='barh',color='g',a
plt.title("Education Field Distribution")
plt.show()
```



In [85]:
```
plt.figure(figsize=(10,8))
ibm_data.MaritalStatus.value_counts().plot(kind='bar',color='r',alp
plt.title("Marital Status Distribution")
plt.show()
```



Understanding the dataset

In [86]:
```
#Describing the dataset
ibm_data.describe()
```

Out[86]:

	Age	DistanceFromHome	Education	EnvironmentSatisfaction	JobSatisfacti
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.000(
mean	36.923810	9.192517	2.912925	2.721769	2.728(
std	9.135373	8.106864	1.024165	1.093082	1.102(
min	18.000000	1.000000	1.000000	1.000000	1.000(
25%	30.000000	2.000000	2.000000	2.000000	2.000(
50%	36.000000	7.000000	3.000000	3.000000	3.000(
75%	43.000000	14.000000	4.000000	4.000000	4.000(
max	60.000000	29.000000	5.000000	4.000000	4.000(

Preparing the data

There is a need to replace the categorical data to numerical data

In [87]:
```
#Converting categorical data to numerical data
#1 replacing attrition data
ibm_data['Attrition'].replace('Yes',1, inplace=True)
ibm_data['Attrition'].replace('No',0, inplace=True)
```

In [88]:
```
#2 replacing Marital status
ibm_data['MaritalStatus'].replace('Married',1, inplace=True)
ibm_data['MaritalStatus'].replace('Single',2, inplace=True)
ibm_data['MaritalStatus'].replace('Divorced',3, inplace=True)
```

In [89]:
```
#3 replacing Education field data
ibm_data['EducationField'].replace('Life Sciences',1, inplace=True)
ibm_data['EducationField'].replace('Medical',2, inplace=True)
ibm_data['EducationField'].replace('Marketing', 3, inplace=True)
ibm_data['EducationField'].replace('Other',4, inplace=True)
ibm_data['EducationField'].replace('Technical Degree',5, inplace=Tr
ibm_data['EducationField'].replace('Human Resources',6, inplace=Tr
```

In [90]:
```
#4 replacing department data
ibm_data['Department'].replace('Research & Development',1, inplace=
ibm_data['Department'].replace('Sales',2, inplace=True)
ibm_data['Department'].replace('Human Resources',3, inplace=True)
```

In [91]:
```
#verifying all data has been converted to numerical datapoints
ibm_data.head(5)
```

Out[91]:

	Age	Attrition	Department	DistanceFromHome	Education	EducationField	EnvironmentS
0	41	1	2	1	2	1	
1	49	0	1	8	1	1	
2	37	1	1	2	2	4	
3	33	0	1	3	4	1	
4	27	0	1	2	1	2	

In [92]:
```
#creating X and Y variables
X = ibm_data.drop(['Attrition'], axis = 1)
Y = ibm_data["Attrition"]
```

In [93]:
```
X.head(5)
```

Out[93]:

	Age	Department	DistanceFromHome	Education	EducationField	EnvironmentSatisfactio
0	41	2	1	2	1	
1	49	1	8	1	1	
2	37	1	2	2	4	
3	33	1	3	4	1	
4	27	1	2	1	2	

In [94]:
```
type(X)
```

Out[94]:
```
pandas.core.frame.DataFrame
```

In [95]:
```
X.describe()
```

Out[95]:

	Age	Department	DistanceFromHome	Education	EducationField	Environ
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.000000	
mean	36.923810	1.389116	9.192517	2.912925	2.150340	
std	9.135373	0.568893	8.106864	1.024165	1.350636	
min	18.000000	1.000000	1.000000	1.000000	1.000000	
25%	30.000000	1.000000	2.000000	2.000000	1.000000	
50%	36.000000	1.000000	7.000000	3.000000	2.000000	
75%	43.000000	2.000000	14.000000	4.000000	3.000000	
max	60.000000	3.000000	29.000000	5.000000	6.000000	

In [96]:
```
Y.head(5)
```

Out[96]:
```
0    1
1    0
2    1
3    0
4    0
Name: Attrition, dtype: int64
```

In [97]:
```
type(Y)
```

Out[97]:
```
pandas.core.series.Series
```

In [98]:
```
#importing sklearn for train & test split, model training and valid
from sklearn.model_selection import train_test_split
```

In [99]:
```
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=
```

In [100]:
```
X_train.shape
```

Out[100]:
```
(1102, 12)
```

In [101]:
```
X_test.shape
```

Out[101]:
```
(368, 12)
```

In [102]:
```
y_train.shape
```

Out[102]:
```
(1102,)
```

In [103]:
```
y_test.shape
```

Out[103]:
```
(368,)
```

Training the model

In [104]:
```
#importing the logistic regression model
from sklearn.linear_model import LogisticRegression
```

In [105]:
```
logRes = LogisticRegression()
```

In [106]:
```
attrition_model = logRes.fit(X_train, y_train)
```

```
/home/satheesh/anaconda3/lib/python3.8/site-packages/sklearn/linea
r_model/_logistic.py:762: ConvergenceWarning: lbfgs failed to conv
erge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (ht
tps://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver opti
ons:
    https://scikit-learn.org/stable/modules/linear_model.html#logi
stic-regression (https://scikit-learn.org/stable/modules/linear_mo
del.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

In [107]:
```
attrition_model.predict_proba(X_test)
```

Out[107]:
```
array([[0.91701785, 0.08298215],
       [0.84932799, 0.15067201],
       [0.82822504, 0.17177496],
       [0.78839905, 0.21160095],
       [0.84256786, 0.15743214],
       [0.84060397, 0.15939603],
       [0.79040601, 0.20959399],
       [0.76358794, 0.23641206],
       [0.97555055, 0.02444945],
       [0.82374186, 0.17625814],
       [0.97378812, 0.02621188],
       [0.79637961, 0.20362039],
       [0.93234146, 0.06765854],
       [0.71589723, 0.28410277],
       [0.86521635, 0.13478365],
       [0.90663809, 0.09336191],
       [0.92783777, 0.07216223],
       [0.88616961, 0.11383039],
       [0.8281925 , 0.1718075 ],
       [0.59417517, 0.40582483]
```

In [108]:
```
#check accuracy on training data
attrition_model.score(X_train, y_train)
```

Out[108]:
```
0.8439201451905626
```

In [109]:
```
#check accuracy on testing data
attrition_model.score(X_test, y_test)
```

Out[109]:
```
0.8396739130434783
```

Model validation using metrics

In [110]:
```
#predicting for X_test as input
y_predicted = attrition_model.predict(X_test)
```

In [111]:
```
y_predicted.shape
```

Out[111]:
```
(368,)
```

Since both the accuracies are almost equal we can conclude that the model is performing in the same manner as the training data

In [112]:
```
#checking for model metrics
from sklearn import metrics
```

In [113]:
```
print(metrics.accuracy_score(y_test, y_predicted))
```

```
0.8396739130434783
```

In [114]:
```
print(metrics.classification_report(y_test, y_predicted))
```

```
              precision    recall  f1-score   support

           0       0.85      0.99      0.91       310
           1       0.40      0.03      0.06        58

    accuracy                           0.84       368
   macro avg       0.62      0.51      0.49       368
weighted avg       0.78      0.84      0.78       368
```

In [115]:
```
print(metrics.confusion_matrix(y_test, y_predicted))
```

```
[[307   3]
 [ 56   2]]
```

The model does not perform well and might not serve the business purpose; the business would like to identify correctly the chances of an individual leaving and since the precision, recall and subsequent f1 score are lesser for the "attrites", we might not have solved the business problem. This will require further hyper parameter tuning