Prediction of Employee Atrrition Using Python- Akshaya

Import Libraries

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot
import matplotlib.pyplot as plt
import matplotlib.pylab as pylab
import matplotlib
%matplotlib inline
```

Import Dataset

In [2]:

```
df = pd.read_csv('D:Attrition.csv')
```

In [3]:

```
df.shape
```

Out[3]:

```
(1470, 35)
```

In [4]:

```
df.info()
```

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

```
 28  TotalWorkingYears         1470 non-null   int64
 29  TrainingTimesLastYear     1470 non-null   int64
 30  WorkLifeBalance           1470 non-null   int64
 31  YearsAtCompany            1470 non-null   int64
 32  YearsInCurrentRole        1470 non-null   int64
 33  YearsSinceLastPromotion   1470 non-null   int64
 34  YearsWithCurrManager      1470 non-null   int64
dtypes: int64(26), object(9)
memory usage: 402.1+ KB
```

In [5]:

```
df.describe()
```

Out[5]:

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumber	EnvironmentSatisfaction	Hourly
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.0	1470.000000	1470.000000	1470.00
mean	36.923810	802.485714	9.192517	2.912925	1.0	1024.865306	2.721769	65.89
std	9.135373	403.509100	8.106864	1.024165	0.0	602.024335	1.093082	20.32
min	18.000000	102.000000	1.000000	1.000000	1.0	1.000000	1.000000	30.00
25%	30.000000	465.000000	2.000000	2.000000	1.0	491.250000	2.000000	48.00
50%	36.000000	802.000000	7.000000	3.000000	1.0	1020.500000	3.000000	66.00
75%	43.000000	1157.000000	14.000000	4.000000	1.0	1555.750000	4.000000	83.75
max	60.000000	1499.000000	29.000000	5.000000	1.0	2068.000000	4.000000	100.00

8 rows × 26 columns

In [6]:

```
df['Attrition'] = df['Attrition'].apply(
    lambda x: 0 if x == 'No' else 1)
df.shape
```

Out[6]:

```
(1470, 35)
```

In [7]:

```
df.head()
```

Out[7]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	Employeel
0	41	1	Travel_Rarely	1102	Sales	1	2	Life Sciences	1	
1	49	0	Travel_Frequently	279	Research & Development	8	1	Life Sciences	1	
2	37	1	Travel_Rarely	1373	Research & Development	2	2	Other	1	
3	33	0	Travel_Frequently	1392	Research & Development	3	4	Life Sciences	1	
4	27	0	Travel_Rarely	591	Research & Development	2	1	Medical	1	

5 rows × 35 columns

Exploratory Data Analysis

In [8]:

```
for column in df.columns:
```

```
    print(f"{column}: Number of unique values {df[column].nunique()}")
```

```
Age: Number of unique values 43
Attrition: Number of unique values 2
BusinessTravel: Number of unique values 3
DailyRate: Number of unique values 886
Department: Number of unique values 3
DistanceFromHome: Number of unique values 29
Education: Number of unique values 5
EducationField: Number of unique values 6
EmployeeCount: Number of unique values 1
EmployeeNumber: Number of unique values 1470
EnvironmentSatisfaction: Number of unique values 4
Gender: Number of unique values 2
HourlyRate: Number of unique values 71
JobInvolvement: Number of unique values 4
JobLevel: Number of unique values 5
JobRole: Number of unique values 9
JobSatisfaction: Number of unique values 4
MaritalStatus: Number of unique values 3
MonthlyIncome: Number of unique values 1349
MonthlyRate: Number of unique values 1427
NumCompaniesWorked: Number of unique values 10
Over18: Number of unique values 1
OverTime: Number of unique values 2
PercentSalaryHike: Number of unique values 15
PerformanceRating: Number of unique values 2
RelationshipSatisfaction: Number of unique values 4
StandardHours: Number of unique values 1
StockOptionLevel: Number of unique values 4
TotalWorkingYears: Number of unique values 40
TrainingTimesLastYear: Number of unique values 7
WorkLifeBalance: Number of unique values 4
YearsAtCompany: Number of unique values 37
YearsInCurrentRole: Number of unique values 19
YearsSinceLastPromotion: Number of unique values 16
YearsWithCurrManager: Number of unique values 18
```

In [9]:

```
cat_col = []
for column in df.columns:
    if df[column].dtype == object and len(df[column].unique()) <= 30:
        cat_col.append(column)
        print(f"{column} : {df[column].unique()}")
        print(df[column].value_counts())
        print("--------------------------------")


print(len(cat_col))
```

```
BusinessTravel : ['Travel_Rarely' 'Travel_Frequently' 'Non-Travel']
Travel_Rarely        1043
Travel_Frequently     277
Non-Travel            150
Name: BusinessTravel, dtype: int64
--------------------------------
Department : ['Sales' 'Research & Development' 'Human Resources']
Research & Development    961
Sales                     446
Human Resources            63
Name: Department, dtype: int64
--------------------------------
EducationField : ['Life Sciences' 'Other' 'Medical' 'Marketing' 'Technical Degree'
 'Human Resources']
Life Sciences       606
Medical             464
Marketing           159
Technical Degree    132
Other                82
Human Resources      27
Name: EducationField, dtype: int64
--------------------------------
Gender : ['Female' 'Male']
Male      882
```

```
Female    588
Name: Gender, dtype: int64
--------------------------------
JobRole : ['Sales Executive' 'Research Scientist' 'Laboratory Technician'
 'Manufacturing Director' 'Healthcare Representative' 'Manager'
 'Sales Representative' 'Research Director' 'Human Resources']
Sales Executive              326
Research Scientist           292
Laboratory Technician        259
Manufacturing Director       145
Healthcare Representative    131
Manager                      102
Sales Representative          83
Research Director             80
Human Resources               52
Name: JobRole, dtype: int64
--------------------------------
MaritalStatus : ['Single' 'Married' 'Divorced']
Married     673
Single      470
Divorced    327
Name: MaritalStatus, dtype: int64
--------------------------------
Over18 : ['Y']
Y    1470
Name: Over18, dtype: int64
--------------------------------
OverTime : ['Yes' 'No']
No     1054
Yes     416
Name: OverTime, dtype: int64
--------------------------------
8
```

In [10]:

```
dis_col = []
for column in df.columns:
    if df[column].dtypes != object and df[column].nunique() < 30:
        print(f"{column} : {df[column].unique()}")
        dis_col.append(column)
        print("------------------------------------")


print(len(dis_col))
```

```
Attrition : [1 0]
------------------------------------
DistanceFromHome : [ 1  8  2  3 24 23 27 16 15 26 19 21  5 11  9  7  6 10  4 25 12 18 29 22
 14 20 28 17 13]
------------------------------------
Education : [2 1 4 3 5]
------------------------------------
EmployeeCount : [1]
------------------------------------
EnvironmentSatisfaction : [2 3 4 1]
------------------------------------
JobInvolvement : [3 2 4 1]
------------------------------------
JobLevel : [2 1 3 4 5]
------------------------------------
JobSatisfaction : [4 2 3 1]
------------------------------------
NumCompaniesWorked : [8 1 6 9 0 4 5 2 7 3]
------------------------------------
PercentSalaryHike : [11 23 15 12 13 20 22 21 17 14 16 18 19 24 25]
------------------------------------
PerformanceRating : [3 4]
------------------------------------
RelationshipSatisfaction : [1 4 2 3]
------------------------------------
StandardHours : [80]
------------------------------------
StockOptionLevel : [0 1 3 2]
------------------------------------
```

```
TrainingTimesLastYear : [0 3 2 5 1 4 6]
------------------------------------
WorkLifeBalance : [1 3 2 4]
------------------------------------
YearsInCurrentRole : [ 4  7  0  2  5  9  8  3  6 13  1 15 14 16 11 10 12 18 17]
------------------------------------
YearsSinceLastPromotion : [ 0  1  3  2  7  4  8  6  5 15  9 13 12 10 11 14]
------------------------------------
YearsWithCurrManager : [ 5  7  0  2  6  8  3 11 17  1  4 12  9 10 15 13 16 14]
------------------------------------
19
```

In [11]:

```python
cont_col = []
for column in df.columns:
    if df[column].dtypes != object and df[column].nunique() > 30:
        print(f"{column} : Minimum: {df[column].min()}, Maximum: {df[column].max()}")
        cont_col.append(column)
        print("-------------------------------------")

print(len(cont_col))
```

```
Age : Minimum: 18, Maximum: 60
-------------------------------------
DailyRate : Minimum: 102, Maximum: 1499
-------------------------------------
EmployeeNumber : Minimum: 1, Maximum: 2068
-------------------------------------
HourlyRate : Minimum: 30, Maximum: 100
-------------------------------------
MonthlyIncome : Minimum: 1009, Maximum: 19999
-------------------------------------
MonthlyRate : Minimum: 2094, Maximum: 26999
-------------------------------------
TotalWorkingYears : Minimum: 0, Maximum: 40
-------------------------------------
YearsAtCompany : Minimum: 0, Maximum: 40
-------------------------------------
8
```

In [12]:

```python
from sklearn.preprocessing import LabelEncoder

label = LabelEncoder()
df["Attrition"] = label.fit_transform(df.Attrition)
```

In [13]:

```python
df.head()
```

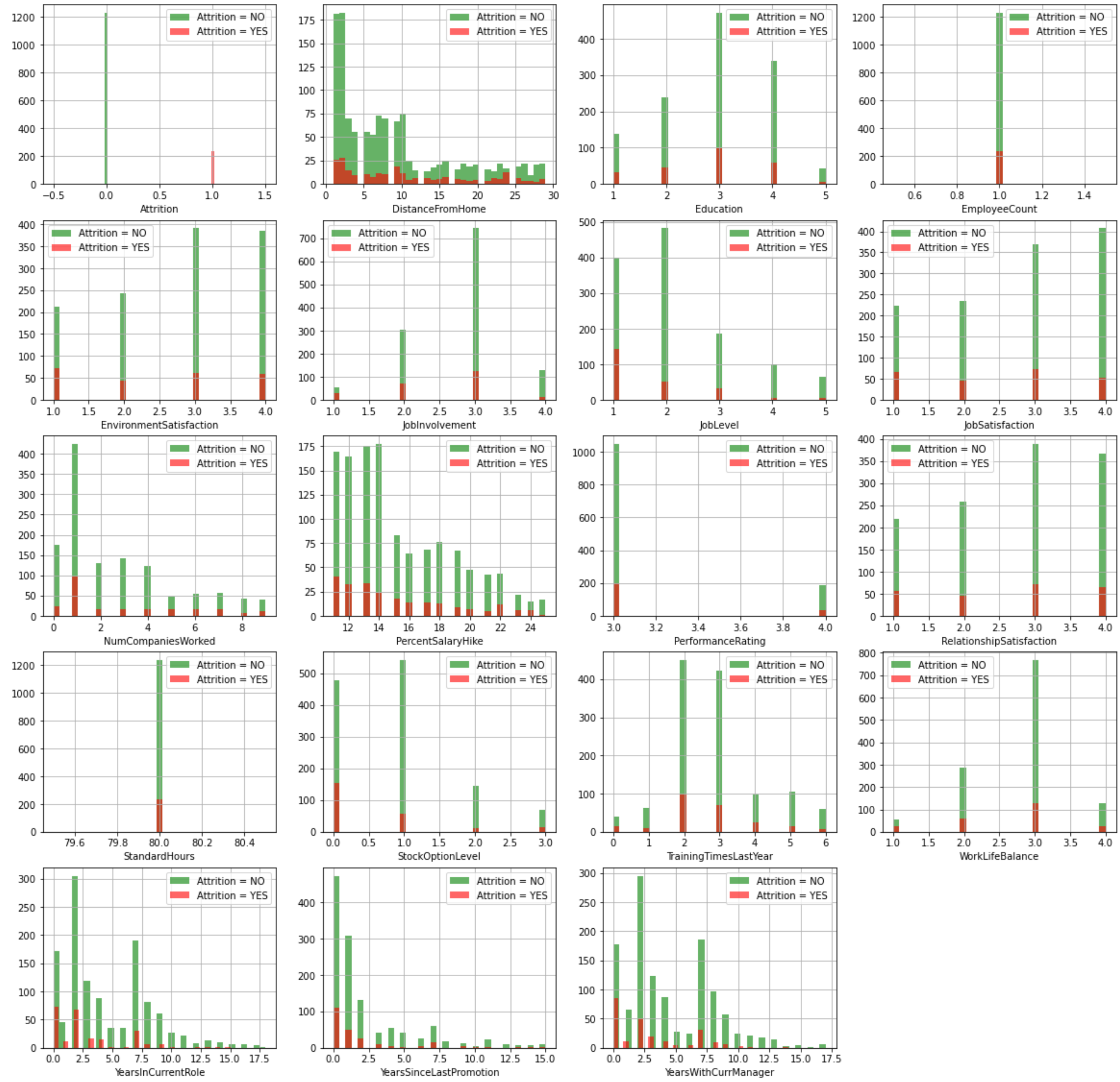
Out[13]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	Employeel
0	41	1	Travel_Rarely	1102	Sales	1	2	Life Sciences	1	
1	49	0	Travel_Frequently	279	Research & Development	8	1	Life Sciences	1	
2	37	1	Travel_Rarely	1373	Research & Development	2	2	Other	1	
3	33	0	Travel_Frequently	1392	Research & Development	3	4	Life Sciences	1	
4	27	0	Travel_Rarely	591	Research & Development	2	1	Medical	1	

5 rows × 35 columns

In [14]:

```
plt.figure(figsize=(20, 20))

for i, column in enumerate(dis_col, 1):
    plt.subplot(5, 4, i)
    df[df["Attrition"] == 0][column].hist(bins=35, color='green', label='Attrition = NO', alpha=0.6
)
    df[df["Attrition"] == 1][column].hist(bins=35, color='red', label='Attrition = YES', alpha=0.6)
    plt.legend()
    plt.xlabel(column)
```



Inferences

We can infer that Education,Environment Satisfaction, Job Satisfaction, Performance Rating, and Relationship Satisfaction features don't have big impact on the detrmination of Attrition of employees.

Num of companies worked : People worked in more num of companies are likely to quit.

Yrs since promotion: people who dont get promotion for a longer period are likely to quit.

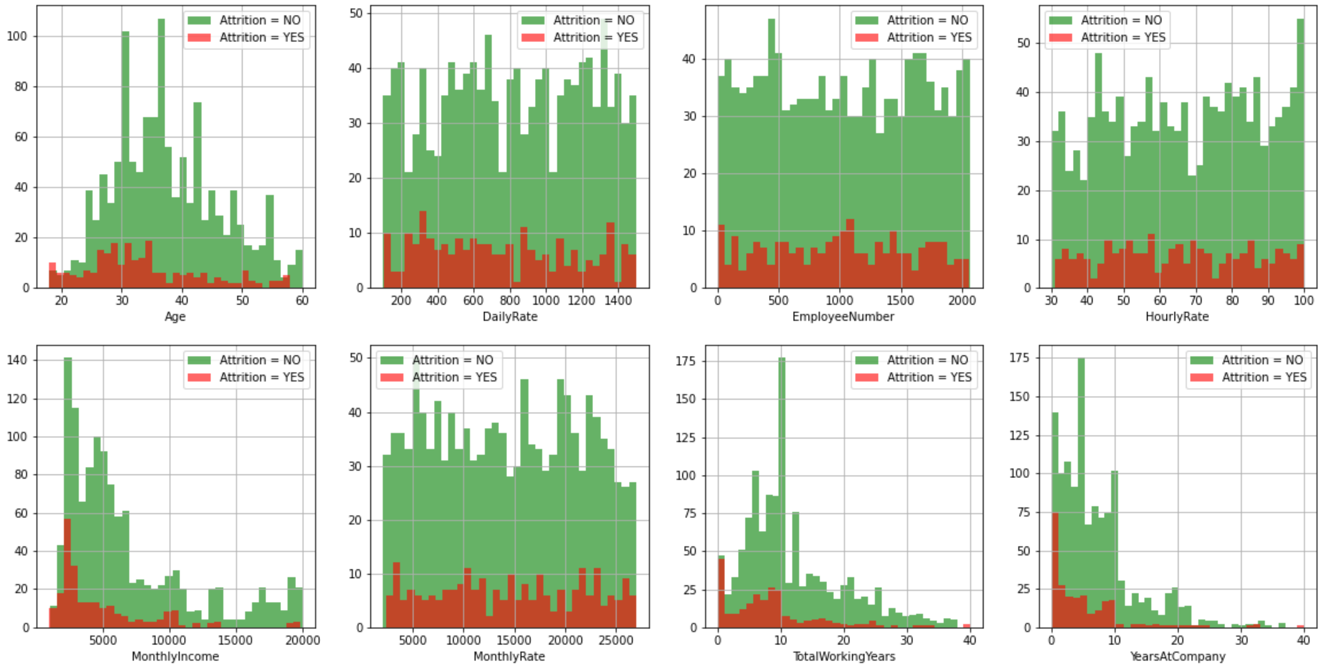
Yrs with current manager: People who worked more yrs with the current manager are likely to quit.

Yrs in current role: people staying in current role for longer period are likely to quit.

In [15]:

```
plt.figure(figsize=(20, 10))
```

```
plt.figure(figsize=(20, 10))

for i, column in enumerate(cont_col, 1):
    plt.subplot(2, 4, i)
    df[df["Attrition"] == 0][column].hist(bins=35, color='green', label='Attrition = NO', alpha=0.6
)
    df[df["Attrition"] == 1][column].hist(bins=35, color='red', label='Attrition = YES', alpha=0.6)
    plt.legend()
    plt.xlabel(column)
```



Inferences

Monthly income: people who has less than 5000 salary are more likely to quit.

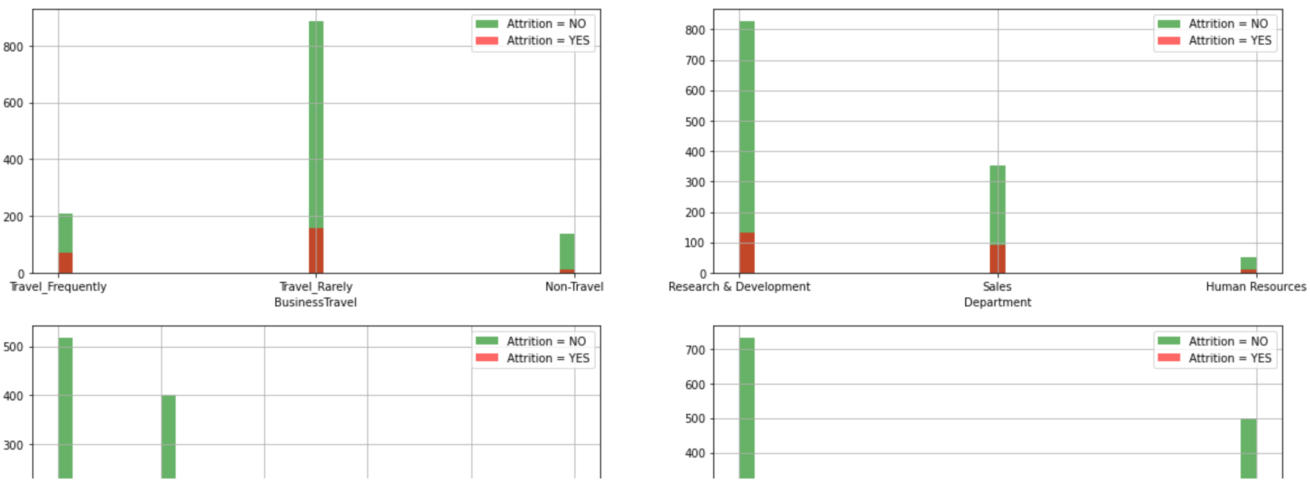
Total working years: As the total working years increases people are less likely to leave the company.

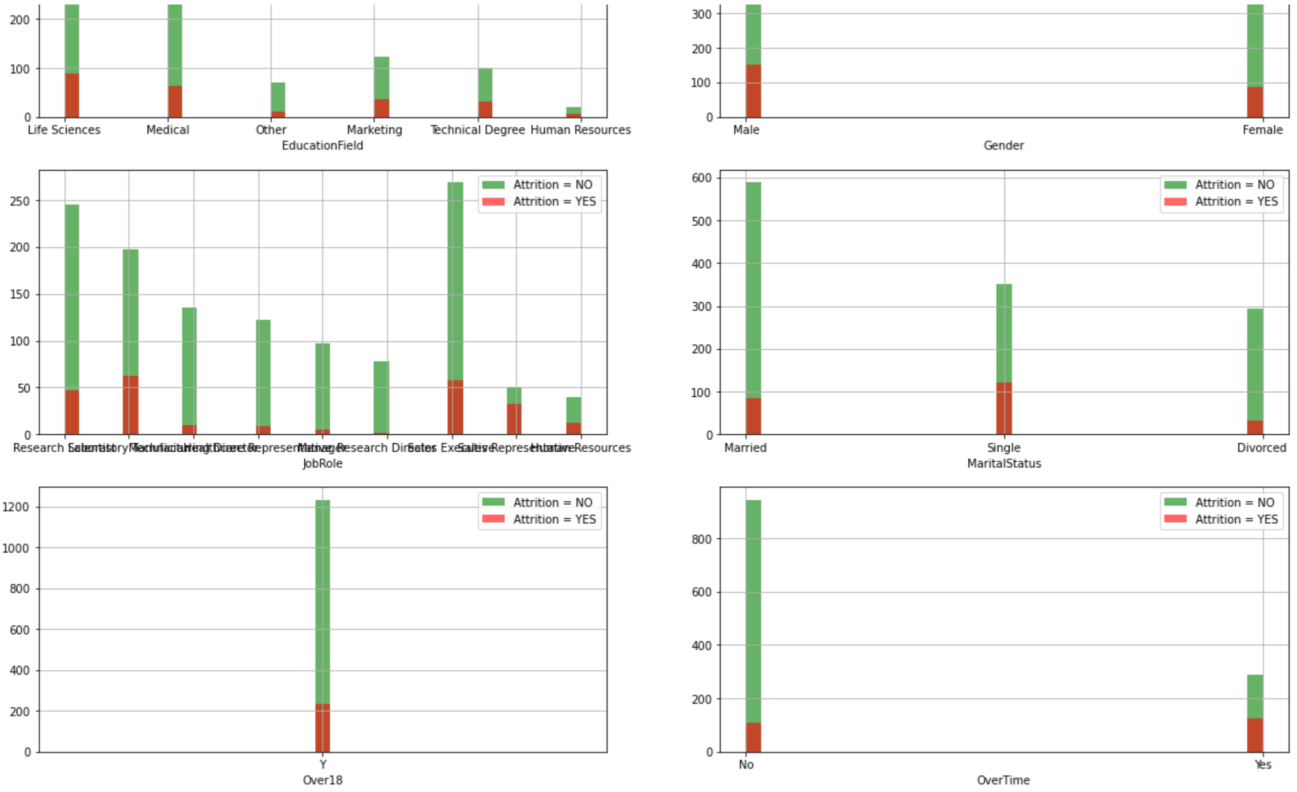
Years at company : Loyal Employees are less likely to leave the company.

Daily rate, monthly rate does not have any impact on attrition.

In [16]:

```
plt.figure(figsize=(20, 20))
for i, column in enumerate(cat_col, 1):
    plt.subplot(4, 2, i)
    df[df["Attrition"] == 0][column].hist(bins=35, color='green', label='Attrition = NO', alpha=0.6
)
    df[df["Attrition"] == 1][column].hist(bins=35, color='red', label='Attrition = YES', alpha=0.6)
    plt.legend()
    plt.xlabel(column)
```





Inferences

The workers with low JobLevel, MonthlyIncome, YearAtCompany, and TotalWorkingYears are more likely to quit their jobs. BusinessTravel : The workers who travel alot are more likely to quit then other employees.

Department : The worker in Research & Development are more likely to stay then the workers on other departement.

EducationField : The workers with Human Resources and Technical Degree are more likely to quit then employees from other fields of educations.

Gender : The Male are more likely to quit.

JobRole : The workers in Laboratory Technician, Sales Representative, and Human Resources are more likely to quit the workers in other positions.

MaritalStatus : The workers who have Single marital status are more likely to quit the Married, and Divorced.

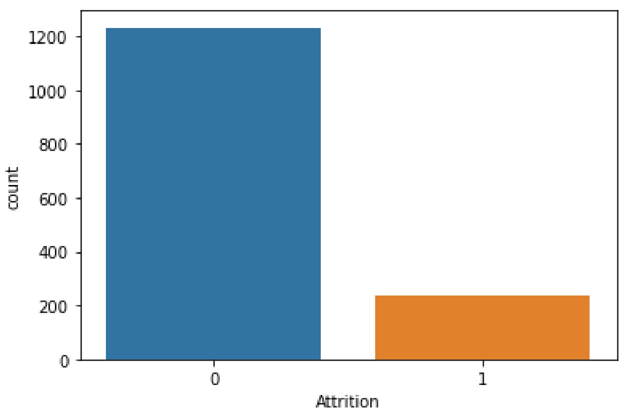
OverTime : The workers who work more hours are likely to leave then others.

In [17]:

```
sns.countplot(x='Attrition',data=df)
```

Out[17]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2b7b01b2700>
```



In [18]:

```
df['Attrition'].value_counts()
```

Out[18]:

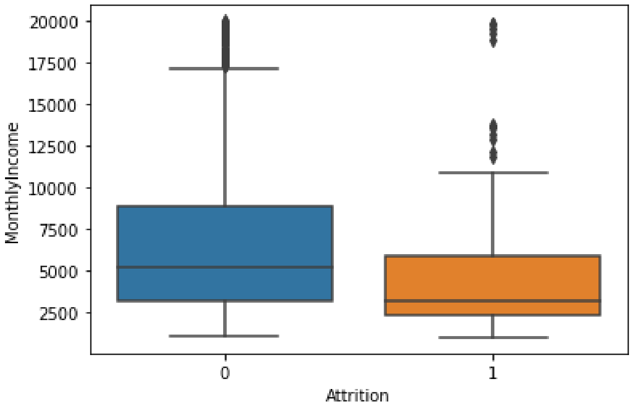
```
0    1233
1     237
Name: Attrition, dtype: int64
```

In [19]:

```
sns.boxplot(y ='MonthlyIncome',
            x ='Attrition',
            data = df)
```

Out[19]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2b7b0190130>
```

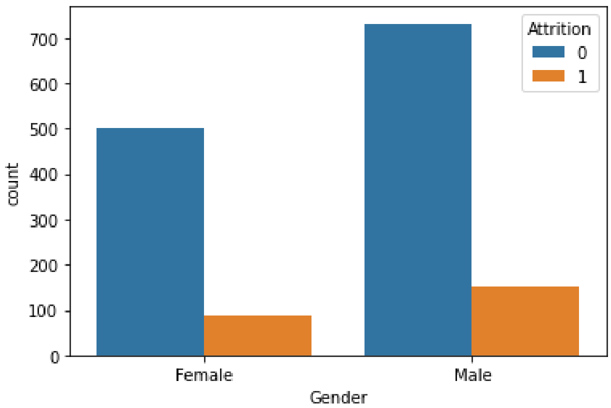


In [20]:

```
sns.countplot(data=df, x='Gender', hue='Attrition')
```

Out[20]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2b7ae5d8be0>
```

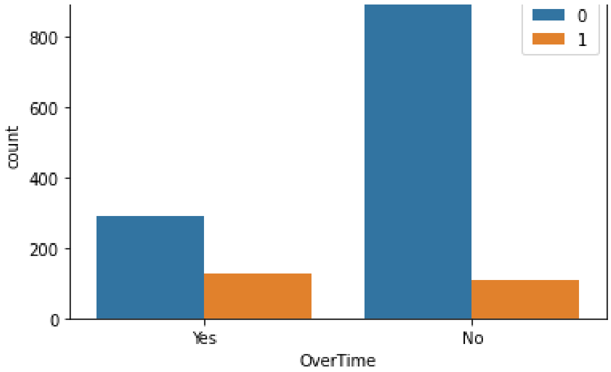


In [21]:

```
sns.countplot(data=df, x='OverTime', hue='Attrition')
```

Out[21]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2b7b00aba60>
```

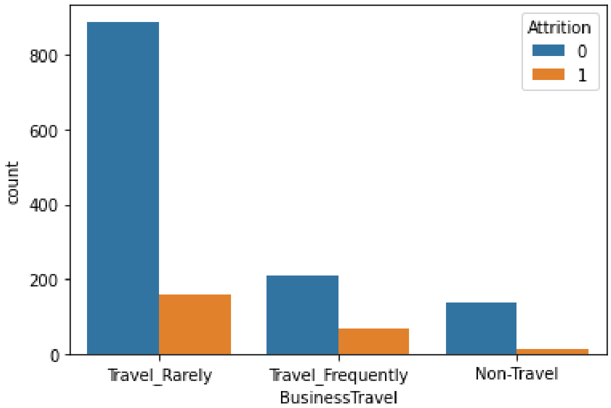


In [22]:

```
sns.countplot(data=df,x='BusinessTravel', hue='Attrition')
```

Out[22]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2b7ae5d8cd0>
```

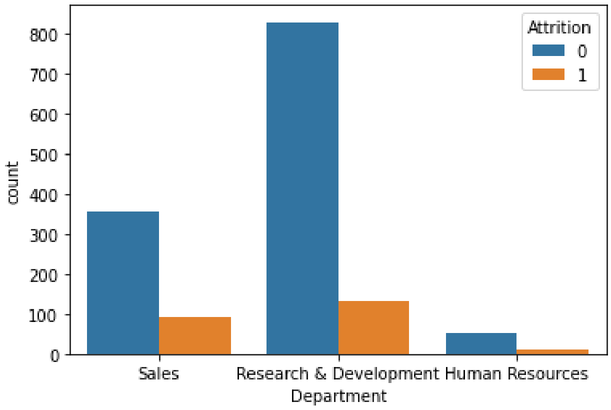


In [23]:

```
sns.countplot(data=df,x='Department', hue='Attrition')
```

Out[23]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2b7af4ec310>
```



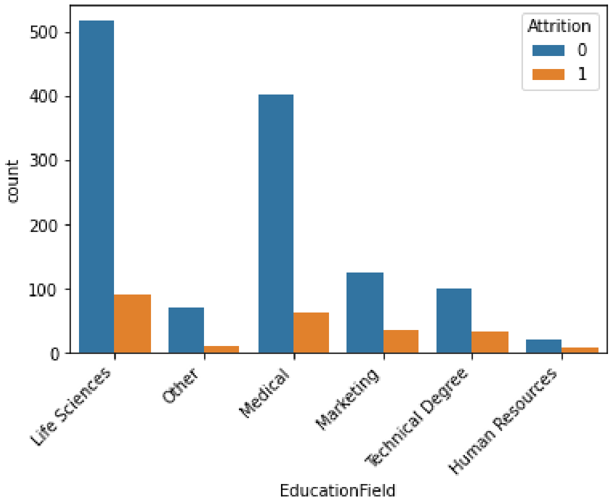
In [24]:

```
chart=sns.countplot(data=df,x='EducationField', hue='Attrition')
chart.set_xticklabels(chart.get_xticklabels(), rotation=45, horizontalalignment='right')
```

Out[24]:

```
[Text(0, 0, 'Life Sciences'),
 Text(0, 0, 'Other'),
```

```
 Text(0, 0, 'Other'),
 Text(0, 0, 'Medical'),
 Text(0, 0, 'Marketing'),
 Text(0, 0, 'Technical Degree'),
 Text(0, 0, 'Human Resources')]
```

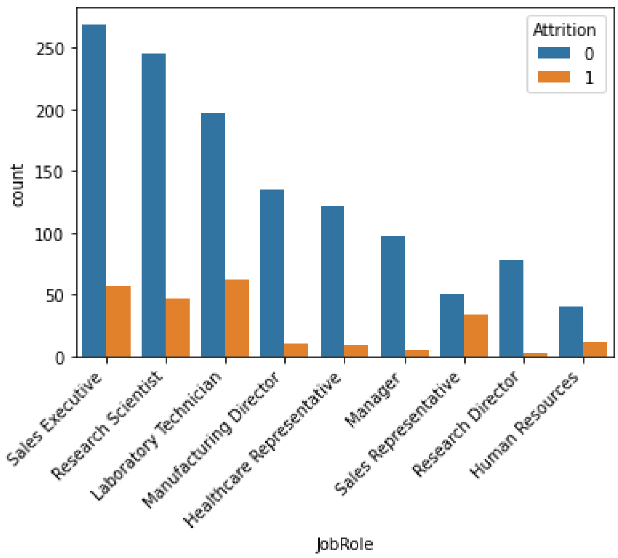


In [25]:

```
chart=sns.countplot(data=df,x='JobRole', hue='Attrition')
chart.set_xticklabels(chart.get_xticklabels(), rotation=45, horizontalalignment='right')
```

Out[25]:

```
[Text(0, 0, 'Sales Executive'),
 Text(0, 0, 'Research Scientist'),
 Text(0, 0, 'Laboratory Technician'),
 Text(0, 0, 'Manufacturing Director'),
 Text(0, 0, 'Healthcare Representative'),
 Text(0, 0, 'Manager'),
 Text(0, 0, 'Sales Representative'),
 Text(0, 0, 'Research Director'),
 Text(0, 0, 'Human Resources')]
```

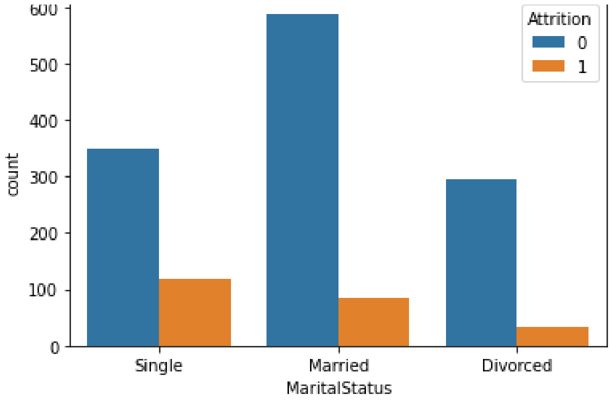


In [26]:

```
sns.countplot(data=df,x='MaritalStatus', hue='Attrition')
```

Out[26]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2b7b00eeca0>
```



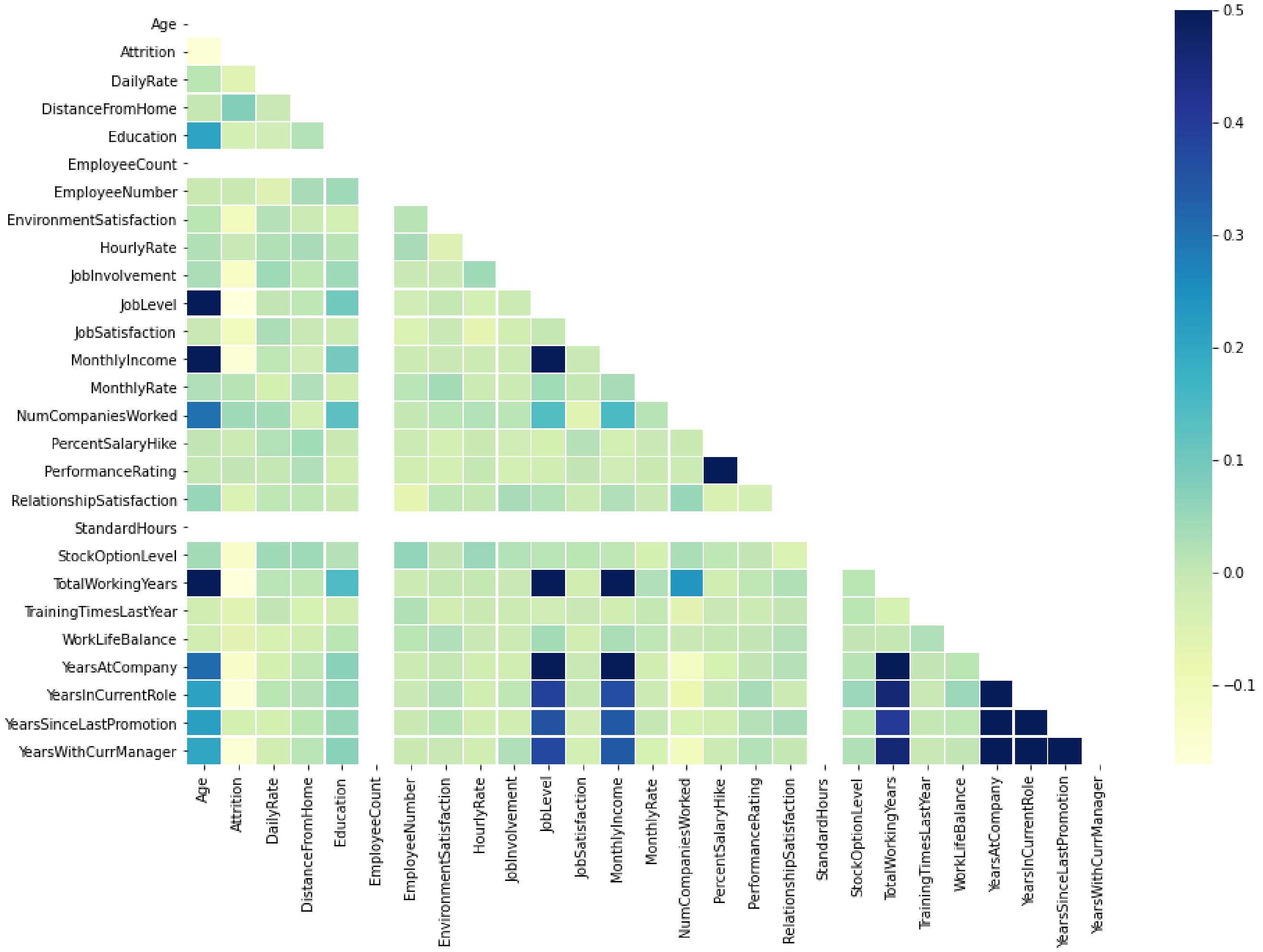
Correlation

In [27]:

```
corr = df.corr()
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True
plt.figure(figsize=(15, 10))
sns.heatmap(corr,
            vmax=.5,
            mask=mask,
            # annot=True, fmt='.2f',
            linewidths=.2, cmap="YlGnBu")
```

Out[27]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2b7aed0fa30>
```



Inferences

Monthly income is highly correlated with Job level.

Job level is highly correlated with total working years.

Monthly income is highly correlated with total working hours.

Age is also positively correlated with the Total working Years.

Feature Engineering

In [28]:

```
dummy_col = [column for column in df.drop('Attrition', axis=1).columns if df[column].nunique() < 20
]
data = pd.get_dummies(df, columns=dummy_col, drop_first=True, dtype='uint8')
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Columns: 137 entries, Age to YearsWithCurrManager_17
dtypes: int64(10), uint8(127)
memory usage: 297.3 KB
```

In [29]:

```
# Remove duplicate Features
data = data.T.drop_duplicates()
data = data.T

# Remove Duplicate Rows
data.drop_duplicates(inplace=True)

print(data.shape)
```

```
(1470, 137)
```

In [30]:

```
data.head()
```

Out[30]:

	Age	Attrition	DailyRate	DistanceFromHome	EmployeeNumber	HourlyRate	MonthlyIncome	MonthlyRate	TotalWorkingYears	Year
0	41	1	1102	1	1	94	5993	19479	8	
1	49	0	279	8	2	61	5130	24907	10	
2	37	1	1373	2	4	92	2090	2396	7	
3	33	0	1392	3	5	56	2909	23159	8	
4	27	0	591	2	7	40	3468	16632	6	

5 rows × 137 columns

Feature Scaling & Data Spliting

In [31]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X = data.drop('Attrition', axis=1)
y = data.Attrition

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42,
                                                    stratify=y)
```

```
scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.transform(X_test)
X_std = scaler.transform(X)
```

Defining Evaluation functions

In [32]:

```
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, f1_sco
re

def print_score(clf, X_train, y_train, X_test, y_test, train=True):
    if train:
        pred = clf.predict(X_train)
        print("Train Result:\n================================================")
        print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
        print("_______________________________________________")
        print("Classification Report:", end='')
        print(f"\tPrecision Score: {precision_score(y_train, pred) * 100:.2f}%")
        print(f"\t\t\tRecall Score: {recall_score(y_train, pred) * 100:.2f}%")
        print(f"\t\t\tF1 score: {f1_score(y_train, pred) * 100:.2f}%")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_train, pred)}\n")

    elif train==False:
        pred = clf.predict(X_test)
        print("Test Result:\n================================================")
        print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
        print("_______________________________________________")
        print("Classification Report:", end='')
        print(f"\tPrecision Score: {precision_score(y_test, pred) * 100:.2f}%")
        print(f"\t\t\tRecall Score: {recall_score(y_test, pred) * 100:.2f}%")
        print(f"\t\t\tF1 score: {f1_score(y_test, pred) * 100:.2f}%")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n")
```

Model Building

In [33]:

```
from sklearn import svm, tree, linear_model, neighbors
from sklearn import naive_bayes, ensemble, discriminant_analysis, gaussian_process
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [34]:

```
from sklearn.model_selection import GridSearchCV
from sklearn import feature_selection
from sklearn import model_selection
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report, precision_recall_curve
from sklearn.metrics import auc, roc_auc_score, roc_curve, recall_score, log_loss
from sklearn.metrics import f1_score, accuracy_score, roc_auc_score, make_scorer
from sklearn.metrics import average_precision_score
```

In [35]:

```
models = []
models.append(('Logistic Regression', LogisticRegression(solver='liblinear', random_state=42,
                                                         class_weight='balanced')))
```

```python
models.append(('Random Forest', RandomForestClassifier(
    n_estimators=100, random_state=42)))
models.append(('SVM', SVC(gamma='auto', random_state=42)))
models.append(('KNN', KNeighborsClassifier()))
models.append(('Decision Tree Classifier',
               DecisionTreeClassifier(random_state=42)))
models.append(('Gaussian NB', GaussianNB()))
```

In [36]:

```python
acc_results = []
auc_results = []
names = []
# set table to table to populate with performance results
col = ['Algorithm', 'ROC AUC Mean', 'ROC AUC STD',
       'Accuracy Mean', 'Accuracy STD']
df_results = pd.DataFrame(columns=col)
i = 0
# evaluate each model using cross-validation
for name, model in models:
    kfold = model_selection.KFold(
        n_splits=10, random_state=7)  # 10-fold cross-validation

    cv_acc_results = model_selection.cross_val_score(  # accuracy scoring
        model, X_train, y_train, cv=kfold, scoring='accuracy')

    cv_auc_results = model_selection.cross_val_score(  # roc_auc scoring
        model, X_train, y_train, cv=kfold, scoring='roc_auc')

    acc_results.append(cv_acc_results)
    auc_results.append(cv_auc_results)
    names.append(name)
    df_results.loc[i] = [name,
                         round(cv_auc_results.mean()*100, 2),
                         round(cv_auc_results.std()*100, 2),
                         round(cv_acc_results.mean()*100, 2),
                         round(cv_acc_results.std()*100, 2)
                         ]
    i += 1
df_results.sort_values(by=['ROC AUC Mean'], ascending=False)
```

```
C:\Users\hp\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:293: FutureWarning:
Setting a random_state has no effect since shuffle is False. This will raise an error in 0.24. You
should leave random_state to its default (None), or set shuffle=True.
  warnings.warn(
C:\Users\hp\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:293: FutureWarning:
Setting a random_state has no effect since shuffle is False. This will raise an error in 0.24. You
should leave random_state to its default (None), or set shuffle=True.
  warnings.warn(
C:\Users\hp\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:293: FutureWarning:
Setting a random_state has no effect since shuffle is False. This will raise an error in 0.24. You
should leave random_state to its default (None), or set shuffle=True.
  warnings.warn(
C:\Users\hp\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:293: FutureWarning:
Setting a random_state has no effect since shuffle is False. This will raise an error in 0.24. You
should leave random_state to its default (None), or set shuffle=True.
  warnings.warn(
C:\Users\hp\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:293: FutureWarning:
Setting a random_state has no effect since shuffle is False. This will raise an error in 0.24. You
should leave random_state to its default (None), or set shuffle=True.
  warnings.warn(
C:\Users\hp\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:293: FutureWarning:
Setting a random_state has no effect since shuffle is False. This will raise an error in 0.24. You
should leave random_state to its default (None), or set shuffle=True.
  warnings.warn(
```

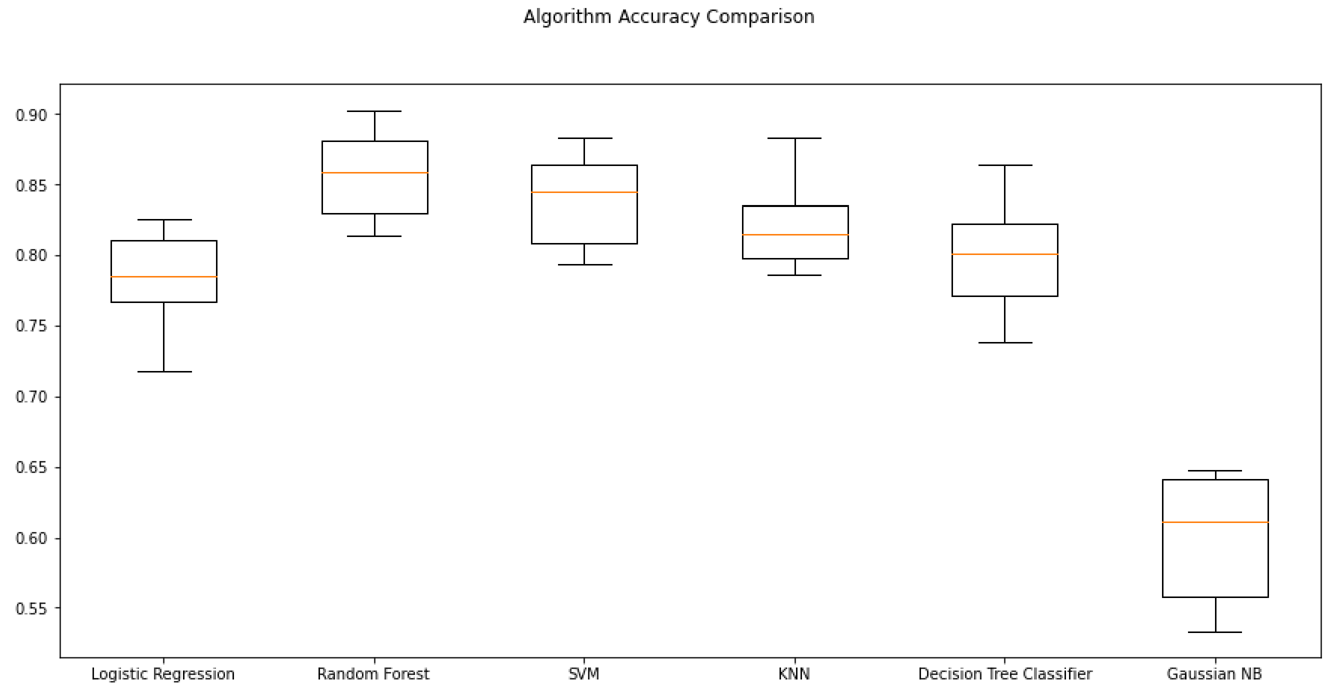
Out[36]:

	Algorithm	ROC AUC Mean	ROC AUC STD	Accuracy Mean	Accuracy STD
0	Logistic Regression	83.05	6.67	78.33	3.11
1	Random Forest	76.37	5.83	85.61	2.93
5	Gaussian NB	75.13	7.38	60.06	4.42

	Algorithm	ROC AUC Mean	ROC AUC STD	Accuracy Mean	Accuracy STD
4	Decision Tree Classifier	61.87	4.62	79.98	3.70
3	KNN	61.81	10.69	82.12	2.80
2	SVM	50.00	0.00	83.86	3.28

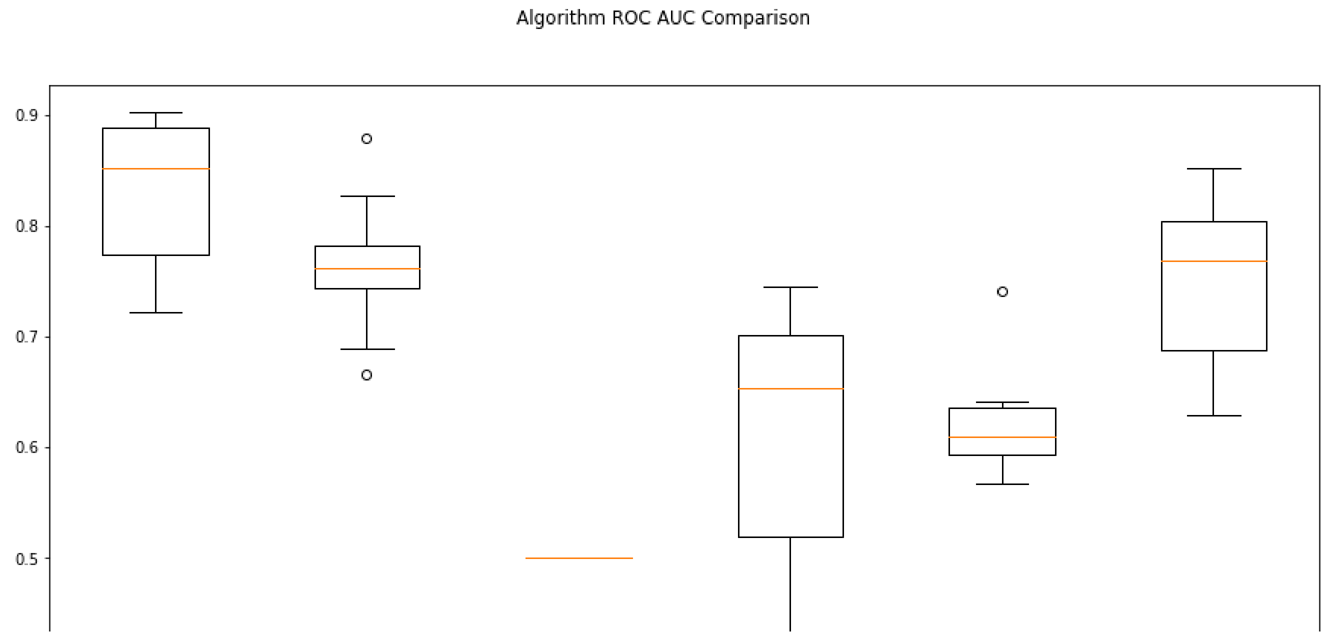
In [37]:

```
fig = plt.figure(figsize=(15, 7))
fig.suptitle('Algorithm Accuracy Comparison')
ax = fig.add_subplot(111)
plt.boxplot(acc_results)
ax.set_xticklabels(names)
plt.show()
```



In [38]:

```
fig = plt.figure(figsize=(15, 7))
fig.suptitle('Algorithm ROC AUC Comparison')
ax = fig.add_subplot(111)
plt.boxplot(auc_results)
ax.set_xticklabels(names)
plt.show()
```



Inferences

1.Logistic Regression and Random forest gives better results than other other models so we can furthur fine tune those 2 models and see if we can get better results.

Fine Tuning RandomForest and Evaluation(2 Fold GridSearch CV)

In [39]:

```
rf = RandomForestClassifier(class_weight = "balanced",
                                        random_state=7)
param_grid = {'n_estimators': [50, 75, 100],
              'min_samples_split':[4,6,8],
              'min_samples_leaf': [ 2, 3, 4],
              'max_depth': [5, 10, 15]}

grid_obj = GridSearchCV(rf,
                        iid=True,
                        return_train_score=True,
                        param_grid=param_grid,
                        scoring='roc_auc',
                        cv=2)

grid_fit = grid_obj.fit(X_train, y_train)
rf_opt = grid_fit.best_estimator_

print('='*20)
print("best params: " + str(grid_obj.best_estimator_))
print("best params: " + str(grid_obj.best_params_))
print('best score:', grid_obj.best_score_)
print('='*20)
```

```
C:\Users\hp\anaconda3\lib\site-packages\sklearn\model_selection\_search.py:847: FutureWarning: The parameter 'iid' is deprecated in 0.22 and will be removed in 0.24.
  warnings.warn(
```

```
====================
best params: RandomForestClassifier(class_weight='balanced', max_depth=10,
                       min_samples_leaf=4, min_samples_split=4, random_state=7)
best params: {'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 4, 'n_estimators': 100}
best score: 0.7757187688604962
====================
```

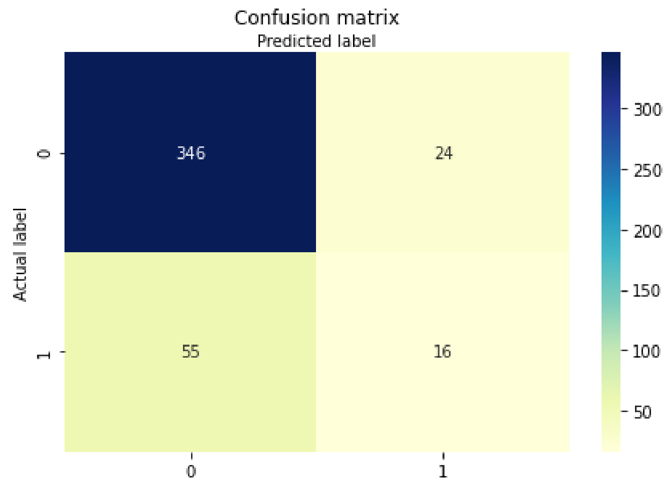
Evaluation

In [40]:

```
cnf_matrix = metrics.confusion_matrix(y_test, rf_opt.predict(X_test))
class_names=[0,1] #classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[40]:

```
Text(0.5, 257.44, 'Predicted label')
```



In [41]:

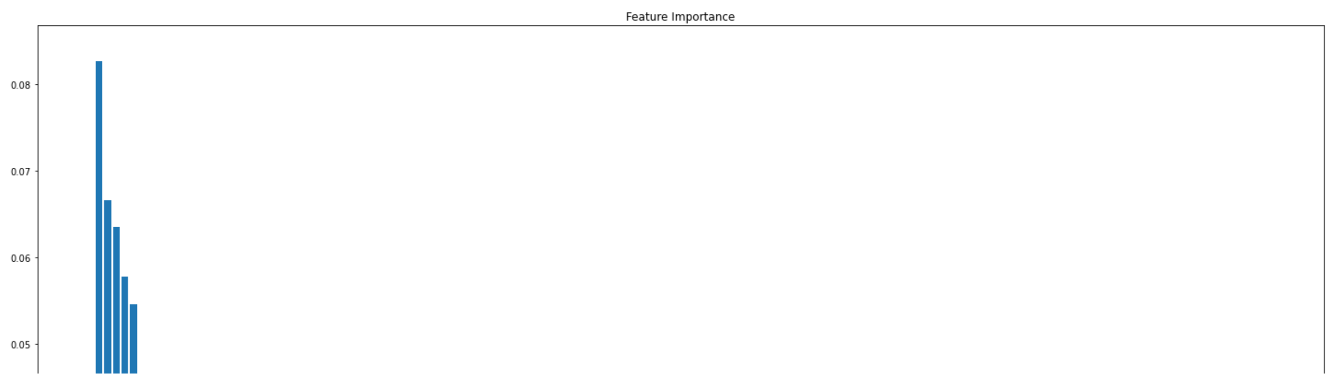
```
print('Accuracy of RandomForest on test set: {:.2f}'.format(rf_opt.score(X_test, y_test)*100))
rf_opt.fit(X_train, y_train)

print(classification_report(y_test, rf_opt.predict(X_test)))

rf_opt.fit(X_train, y_train) # fit optimised model to the training data
probs = rf_opt.predict_proba(X_test) # predict probabilities
probs = probs[:, 1] # we will only keep probabilities associated with the employee leaving
rf_opt_roc_auc = roc_auc_score(y_test, probs) # calculate AUC score using test dataset
print('AUC score: %.3f' % rf_opt_roc_auc)
```

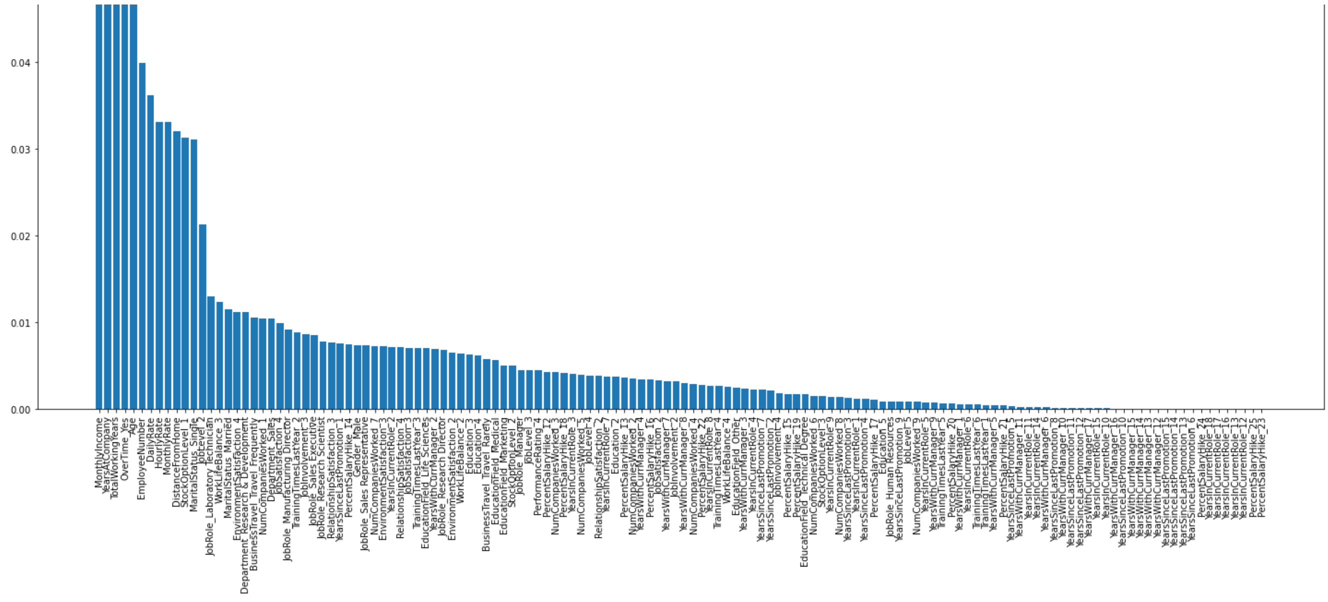
```
Accuracy of RandomForest on test set: 82.09
              precision    recall  f1-score   support

           0       0.86      0.94      0.90       370
           1       0.40      0.23      0.29        71

    accuracy                           0.82       441
   macro avg       0.63      0.58      0.59       441
weighted avg       0.79      0.82      0.80       441

AUC score: 0.763
```

Feature Importances

In [42]:

```
importances = rf_opt.feature_importances_
indices = np.argsort(importances)[::-1] # Sort feature importances in descending order
names = [X_train.columns[i] for i in indices] # Rearrange feature names so they match the sorted fe
ature importances
plt.figure(figsize=(25, 15)) # Create plot
plt.title("Feature Importance") # Create plot title
plt.bar(range(X_train.shape[1]), importances[indices]) # Add bars
plt.xticks(range(X_train.shape[1]), names, rotation=90) # Add feature names as x-axis labels
plt.show() # Show plot
```





Fine Tuning Logit Model and Evaluation by Grid Search CV

In [43]:

```
kfold = model_selection.KFold(n_splits=10, random_state=7)
modelCV = LogisticRegression(solver='liblinear',
                             class_weight="balanced",
                             random_state=7)
scoring = 'roc_auc'
results = model_selection.cross_val_score(
    modelCV, X_train, y_train, cv=kfold, scoring=scoring)
print("AUC score (STD): %.2f (%.2f)" % (results.mean(), results.std()))
```

```
C:\Users\hp\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:293: FutureWarning:
Setting a random_state has no effect since shuffle is False. This will raise an error in 0.24. You
should leave random_state to its default (None), or set shuffle=True.
  warnings.warn(
```

```
AUC score (STD): 0.83 (0.07)
```

In [44]:

```
param_grid = {'C': np.arange(1e-03, 2, 0.01)} # hyper-parameter list to fine-tune
log_gs = GridSearchCV(LogisticRegression(solver='liblinear', # setting GridSearchCV
                                         class_weight="balanced",
                                         random_state=7),
                      iid=True,
                      return_train_score=True,
                      param_grid=param_grid,
                      scoring='roc_auc',
                      cv=2)

log_grid = log_gs.fit(X_train, y_train)
log_opt = log_grid.best_estimator_
results = log_gs.cv_results_

print('='*20)
print("best params: " + str(log_gs.best_estimator_))
print("best params: " + str(log_gs.best_params_))
print('best score:', log_gs.best_score_)
print('='*20)
```

```
====================
best params: LogisticRegression(C=0.23099999999999996, class_weight='balanced',
                   random_state=7, solver='liblinear')
best params: {'C': 0.23099999999999996}
best score: 0.7994106739917911
====================
```

```
C:\Users\hp\anaconda3\lib\site-packages\sklearn\model_selection\_search.py:847: FutureWarning: The
parameter 'iid' is deprecated in 0.22 and will be removed in 0.24.
  warnings.warn(
```

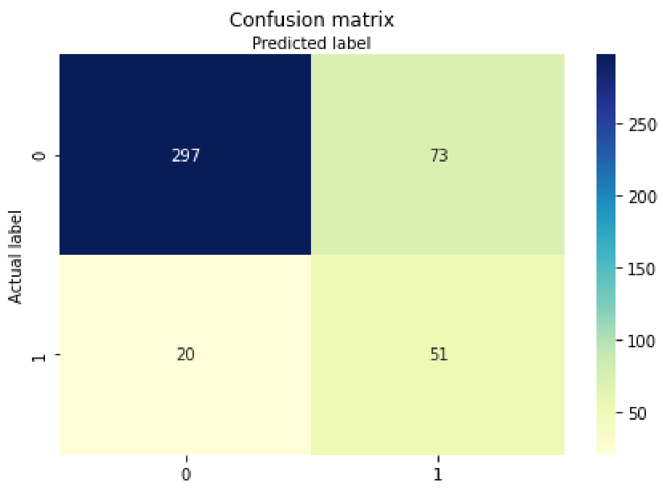
Evaluation

In [45]:

```
## Confusion Matrix
cnf_matrix = metrics.confusion_matrix(y_test, log_opt.predict(X_test))
class_names=[0,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[45]:

```
Text(0.5, 257.44, 'Predicted label')
```



In [46]:

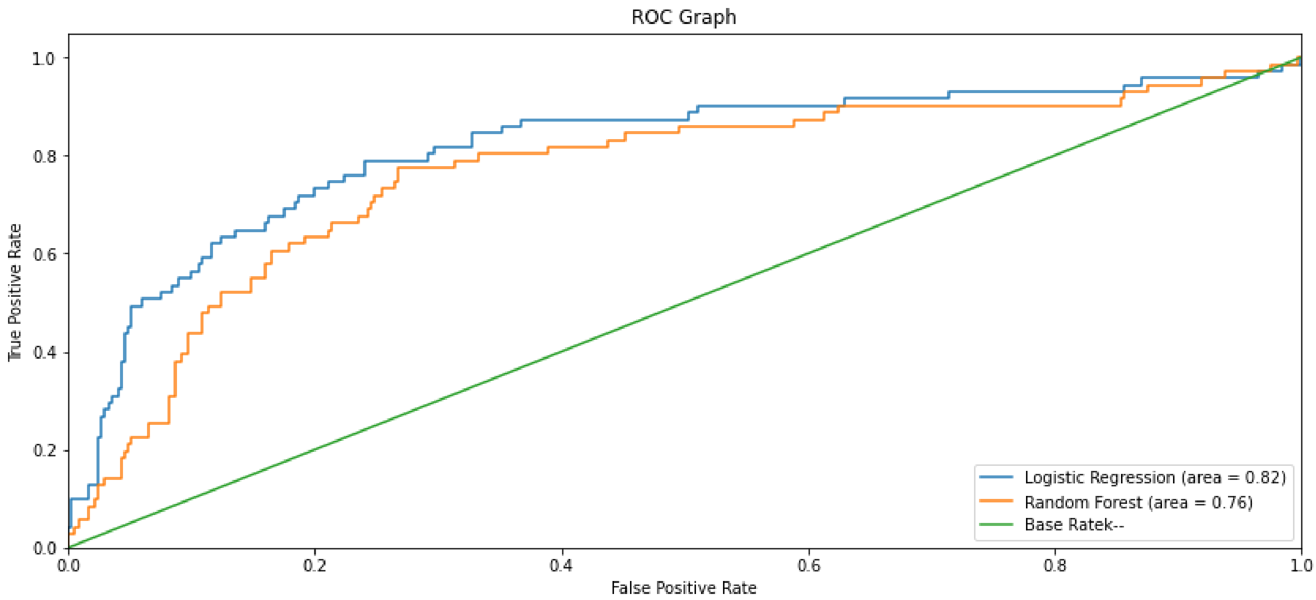
```
print('Accuracy of Logistic Regression Classifier on test set: {:.2f}'.format(log_opt.score(X_test
, y_test)*100))

log_opt.fit(X_train, y_train)
print(classification_report(y_test, log_opt.predict(X_test)))

log_opt.fit(X_train, y_train) # fit optimised model to the training data
probs = log_opt.predict_proba(X_test) # predict probabilities
probs = probs[:, 1] # we will only keep probabilities associated with the employee leaving
logit_roc_auc = roc_auc_score(y_test, probs) # calculate AUC score using test dataset
print('AUC score: %.3f' % logit_roc_auc)
```

```
Accuracy of Logistic Regression Classifier on test set: 78.91
              precision    recall  f1-score   support

           0       0.94      0.80      0.86       370
           1       0.41      0.72      0.52        71

    accuracy                           0.79       441
   macro avg       0.67      0.76      0.69       441
weighted avg       0.85      0.79      0.81       441

AUC score: 0.816
```

Comparision of Final Models by AUC scores

In [47]:

```
# Create ROC Graph
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, log_opt.predict_proba(X_test)[:,1])
rf_fpr, rf_tpr, rf_thresholds = roc_curve(y_test, rf_opt.predict_proba(X_test)[:,1])
plt.figure(figsize=(14, 6))

# Plot Logistic Regression ROC
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
# Plot Random Forest ROC
plt.plot(rf_fpr, rf_tpr, label='Random Forest (area = %0.2f)' % rf_opt_roc_auc)
# Plot Base Rate ROC
plt.plot([0,1], [0,1],label='Base Rate' 'k--')

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Graph')
plt.legend(loc="lower right")
plt.show()
```



Inferences

1. Logistic regression performs better classification than Random forest.
2. So the final model for prediction would be Logistic regression.