Data Science Project

KaggleX

HR Employee Attrition Case Study

Problem Statement

Background :

McCurr Consultancy is an MNC that has thousands of employees spread across the globe. The company believes in hiring the best talent available and retaining them for as long as possible. A huge amount of resources is spent on retaining existing employees through various initiatives. The Head of People Operations wants to bring down the cost of retaining employees. For this, he proposes limiting the incentives to only those employees who are at risk of attrition. As a recently hired Data Scientist in the People Operations Department, you

have been asked to identify patterns in characteristics of employees who leave the organization. Also, you have to use this information to predict if an employee is at risk of attrition. This information will be used to target them with incentives.

Objective :

You, as a Data Scientist at McCurr Consultancy, are tasked with analyzing the data provided to identify the different factors that drive attrition, and build a model that can help to predict attrition.

Dataset :

The data contains demographic details, work-related metrics and attrition flag.

- EmployeeNumber - Employee Identifier
- Attrition - Did the employee attrite?
- Age - Age of the employee
- BusinessTravel - Travel commitments for the job
- DailyRate - Data description not available**
- Department - Employee Department
- DistanceFromHome - Distance from work to home (in km)
- Education - 1-Below College, 2-College, 3-Bachelor, 4-Master,5-Doctor
- EducationField - Field of Education
- EmployeeCount - Employee Count in a row
- EnvironmentSatisfaction - 1-Low, 2-Medium, 3-High, 4-Very High
- Gender - Employee's gender
- HourlyRate - Data description not available**
- JobInvolvement - 1-Low, 2-Medium, 3-High, 4-Very High
- JobLevel - Level of job (1 to 5)
- JobRole - Job Roles
- JobSatisfaction - 1-Low, 2-Medium, 3-High, 4-Very High
- MaritalStatus - Marital Status
- MonthlyIncome - Monthly Salary
- MonthlyRate - Data description not available**
- NumCompaniesWorked - Number of companies worked at
- Over18 - Over 18 years of age?
- OverTime - Overtime?
- PercentSalaryHike - The percentage increase in salary last year
- PerformanceRating - 1-Low, 2-Good, 3-Excellent, 4-Outstanding
- RelationshipSatisfaction - 1-Low, 2-Medium, 3-High, 4-Very High
- StandardHours - Standard Hours
- StockOptionLevel - Stock Option Level
- TotalWorkingYears - Total years worked
- TrainingTimesLastYear - Number of training attended last year
- WorkLifeBalance - 1-Low, 2-Good, 3-Excellent, 4-Outstanding
- YearsAtCompany - Years at Company
- YearsInCurrentRole - Years in the current role
- YearsSinceLastPromotion - Years since the last promotion
- YearsWithCurrManager - Years with the current manager

** In the real world, you will not find definitions for some of your variables. It is a part of the analysis to figure out what they might mean.

Importing necessary libraries

In [ ]:

```
# Libraries to help with reading and manipulating data
import numpy as np
import pandas as pd

# libaries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Library to split data
from sklearn.model_selection import train_test_split

from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import accuracy_score,precision_score,recall_sco
import scipy.stats as stats

import warnings
warnings.filterwarnings('ignore')
```

Reading the dataset

In [ ]:

```
hr=pd.read_csv("HR_Employee_Attrition-1.csv")
```

In [ ]:

```
# copying data to another varaible to avoid any changes to original d
data=hr.copy()
```

Overview of the dataset

View the first and last 5 rows of the dataset.

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

Out[4]:

	EmployeeNumber	Attrition	Age	BusinessTravel	DailyRate	Department	DistanceFron
0	1	Yes	41	Travel_Rarely	1102	Sales	
1	2	No	49	Travel_Frequently	279	Research & Development	
2	3	Yes	37	Travel_Rarely	1373	Research & Development	
3	4	No	33	Travel_Frequently	1392	Research & Development	
4	5	No	27	Travel_Rarely	591	Research & Development	

5 rows × 35 columns

In [ ]:
```
1 data.tail()
```

Out[5]:

	EmployeeNumber	Attrition	Age	BusinessTravel	DailyRate	Department	DistanceF
2935	2936	No	36	Travel_Frequently	884	Research & Development	
2936	2937	No	39	Travel_Rarely	613	Research & Development	
2937	2938	No	27	Travel_Rarely	155	Research & Development	
2938	2939	No	49	Travel_Frequently	1023	Sales	
2939	2940	No	34	Travel_Rarely	628	Research & Development	

5 rows × 35 columns

Understand the shape of the dataset.

In [ ]:
```
1 data.shape
```

Out[6]: (2940, 35)

- The dataset has 2940 rows and 35 columns of data

Check the data types of the columns for the dataset.

In [ ]:
```
1 data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2940 entries, 0 to 2939
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   EmployeeNumber            2940 non-null   int64 
 1   Attrition                 2940 non-null   object
 2   Age                       2940 non-null   int64 
 3   BusinessTravel            2940 non-null   object
 4   DailyRate                 2940 non-null   int64 
 5   Department                2940 non-null   object
 6   DistanceFromHome          2940 non-null   int64 
 7   Education                 2940 non-null   int64 
 8   EducationField            2940 non-null   object
 9   EmployeeCount             2940 non-null   int64 
 10  EnvironmentSatisfaction   2940 non-null   int64 
 11  Gender                    2940 non-null   object
 12  HourlyRate                2940 non-null   int64 
 13  JobInvolvement            2940 non-null   int64 
 14  JobLevel                  2940 non-null   int64 
 15  JobRole                   2940 non-null   object
 16  JobSatisfaction           2940 non-null   int64 
 17  MaritalStatus             2940 non-null   object
 18  MonthlyIncome             2940 non-null   int64 
 19  MonthlyRate               2940 non-null   int64 
 20  NumCompaniesWorked        2940 non-null   int64 
 21  Over18                    2940 non-null   object
 22  OverTime                  2940 non-null   object
 23  PercentSalaryHike         2940 non-null   int64 
 24  PerformanceRating         2940 non-null   int64 
 25  RelationshipSatisfaction  2940 non-null   int64 
 26  StandardHours             2940 non-null   int64 
 27  StockOptionLevel          2940 non-null   int64 
 28  TotalWorkingYears         2940 non-null   int64 
 29  TrainingTimesLastYear     2940 non-null   int64 
 30  WorkLifeBalance           2940 non-null   int64 
 31  YearsAtCompany            2940 non-null   int64 
 32  YearsInCurrentRole        2940 non-null   int64 
 33  YearsSinceLastPromotion   2940 non-null   int64 
 34  YearsWithCurrManager      2940 non-null   int64 
dtypes: int64(26), object(9)
memory usage: 804.0+ KB
```

Observations -

- There are no null values in the dataset.
- We can convert the object type columns to categories.

converting "objects" to "category" reduces the data space required to store the dataframe

Fixing the data types

In [ ]:
```
cols = data.select_dtypes(['object'])
cols.columns
```

Out[8]:
```
Index(['Attrition', 'BusinessTravel', 'Department', 'EducationField', 'Gender',
       'JobRole', 'MaritalStatus', 'Over18', 'OverTime'],
      dtype='object')
```

In [ ]:
```
for i in cols.columns:
    data[i] = data[i].astype('category')
```

In [ ]:
```
1 data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2940 entries, 0 to 2939
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   EmployeeNumber            2940 non-null   int64
 1   Attrition                 2940 non-null   category
 2   Age                       2940 non-null   int64
 3   BusinessTravel            2940 non-null   category
 4   DailyRate                 2940 non-null   int64
 5   Department                2940 non-null   category
 6   DistanceFromHome          2940 non-null   int64
 7   Education                 2940 non-null   int64
 8   EducationField            2940 non-null   category
 9   EmployeeCount             2940 non-null   int64
 10  EnvironmentSatisfaction   2940 non-null   int64
 11  Gender                    2940 non-null   category
 12  HourlyRate                2940 non-null   int64
 13  JobInvolvement            2940 non-null   int64
 14  JobLevel                  2940 non-null   int64
 15  JobRole                   2940 non-null   category
 16  JobSatisfaction           2940 non-null   int64
 17  MaritalStatus             2940 non-null   category
 18  MonthlyIncome             2940 non-null   int64
 19  MonthlyRate               2940 non-null   int64
 20  NumCompaniesWorked        2940 non-null   int64
 21  Over18                    2940 non-null   category
 22  OverTime                  2940 non-null   category
 23  PercentSalaryHike         2940 non-null   int64
 24  PerformanceRating         2940 non-null   int64
 25  RelationshipSatisfaction  2940 non-null   int64
 26  StandardHours             2940 non-null   int64
 27  StockOptionLevel          2940 non-null   int64
 28  TotalWorkingYears         2940 non-null   int64
 29  TrainingTimesLastYear     2940 non-null   int64
 30  WorkLifeBalance           2940 non-null   int64
 31  YearsAtCompany            2940 non-null   int64
 32  YearsInCurrentRole        2940 non-null   int64
 33  YearsSinceLastPromotion   2940 non-null   int64
 34  YearsWithCurrManager      2940 non-null   int64
dtypes: category(9), int64(26)
memory usage: 624.6 KB
```

we can see that the memory usage has decreased from 804 KB to 624.4 KB , this technique is generally useful for bigger datasets.

Summary of the dataset

In [ ]:
```
1 data.describe().T
```

Out[11]:

	count	mean	std	min	25%	50%	
EmployeeNumber	2940.0	1470.500000	848.849221	1.0	735.75	1470.5	22
Age	2940.0	36.923810	9.133819	18.0	30.00	36.0	
DailyRate	2940.0	802.485714	403.440447	102.0	465.00	802.0	11
DistanceFromHome	2940.0	9.192517	8.105485	1.0	2.00	7.0	
Education	2940.0	2.912925	1.023991	1.0	2.00	3.0	
EmployeeCount	2940.0	1.000000	0.000000	1.0	1.00	1.0	
EnvironmentSatisfaction	2940.0	2.721769	1.092896	1.0	2.00	3.0	
HourlyRate	2940.0	65.891156	20.325969	30.0	48.00	66.0	
JobInvolvement	2940.0	2.729932	0.711440	1.0	2.00	3.0	
JobLevel	2940.0	2.063946	1.106752	1.0	1.00	2.0	
JobSatisfaction	2940.0	2.728571	1.102658	1.0	2.00	3.0	
MonthlyIncome	2940.0	6502.931293	4707.155770	1009.0	2911.00	4919.0	83
MonthlyRate	2940.0	14313.103401	7116.575021	2094.0	8045.00	14235.5	204
NumCompaniesWorked	2940.0	2.693197	2.497584	0.0	1.00	2.0	
PercentSalaryHike	2940.0	15.209524	3.659315	11.0	12.00	14.0	
PerformanceRating	2940.0	3.153741	0.360762	3.0	3.00	3.0	
RelationshipSatisfaction	2940.0	2.712245	1.081025	1.0	2.00	3.0	
StandardHours	2940.0	80.000000	0.000000	80.0	80.00	80.0	
StockOptionLevel	2940.0	0.793878	0.851932	0.0	0.00	1.0	
TotalWorkingYears	2940.0	11.279592	7.779458	0.0	6.00	10.0	
TrainingTimesLastYear	2940.0	2.799320	1.289051	0.0	2.00	3.0	
WorkLifeBalance	2940.0	2.761224	0.706356	1.0	2.00	3.0	
YearsAtCompany	2940.0	7.008163	6.125483	0.0	3.00	5.0	
YearsInCurrentRole	2940.0	4.229252	3.622521	0.0	2.00	3.0	
YearsSinceLastPromotion	2940.0	2.187755	3.221882	0.0	0.00	1.0	
YearsWithCurrManager	2940.0	4.123129	3.567529	0.0	2.00	3.0	

- EmployeeNumber is an ID variable and not useful for predictive modelling.
- Age of the employees range from 18 to 60 years and the average age is 36 years.
- EmployeeCount has only 1 as the value in all the rows and can be dropped as it will not be adding any information to our analysis.

- Standard Hours has only 80 as the value in all the rows and can be dropped as it will not be adding any information to our analysis.
- Hourly rate has a huge range, but we do not know what this variable stands for, yet. The same goes for daily and monthly rates.
- Monthly Income has a high range and the difference in mean and median indicate the presence of outliers.

In [ ]:
```
1 data.describe(include=['category']).T
```

Out[12]:

	count	unique	top	freq
Attrition	2940	2	No	2466
BusinessTravel	2940	3	Travel_Rarely	2086
Department	2940	3	Research & Development	1922
EducationField	2940	6	Life Sciences	1212
Gender	2940	2	Male	1764
JobRole	2940	9	Sales Executive	652
MaritalStatus	2940	3	Married	1346
Over18	2940	1	Y	2940
OverTime	2940	2	No	2108

- Attrition is our target variable with 84% records 'No' or employee will not attrite.
- Majority of the employees have low business travel requirements
- Majority of the employees are from the Research and Development department.
- All employees are over 18 years of age - we can drop this variable as it will not be adding any information to our analysis.
- There are more male employees than female employees.

Dropping columns which are not adding any information.

In [ ]:
```
1 data.drop(['EmployeeNumber','EmployeeCount','StandardHours','Over18']
```

Let's look at the unique values of all the categories

In [ ]:
```
1 cols_cat= data.select_dtypes(['category'])
```

In [ ]:

```
for i in cols_cat.columns:
    print('Unique values in',i, 'are :')
    print(cols_cat[i].value_counts())
    print('*'*50)
```

```
Unique values in Attrition are :
No     2466
Yes     474
Name: Attrition, dtype: int64
**************************************************
Unique values in BusinessTravel are :
Travel_Rarely        2086
Travel_Frequently     554
Non-Travel            300
Name: BusinessTravel, dtype: int64
**************************************************
Unique values in Department are :
Research & Development    1922
Sales                      892
Human Resources            126
Name: Department, dtype: int64
**************************************************
Unique values in EducationField are :
Life Sciences       1212
Medical              928
Marketing            318
Technical Degree     264
Other                164
Human Resources       54
Name: EducationField, dtype: int64
**************************************************
Unique values in Gender are :
Male      1764
Female    1176
Name: Gender, dtype: int64
**************************************************
Unique values in JobRole are :
Sales Executive              652
Research Scientist           584
Laboratory Technician        518
Manufacturing Director       290
Healthcare Representative    262
Manager                      204
Sales Representative         166
Research Director            160
Human Resources              104
Name: JobRole, dtype: int64
**************************************************
Unique values in MaritalStatus are :
Married     1346
Single       940
Divorced     654
Name: MaritalStatus, dtype: int64
**************************************************
Unique values in OverTime are :
No     2108
Yes     832
Name: OverTime, dtype: int64
**************************************************
```

In [ ]:
```
df = data.copy()
```

Exploratory Data Analysis (EDA) Summary

The below functions need to be defined to carry out the EDA.

In [ ]:
```
def histogram_boxplot(data, feature, figsize=(15, 10), kde=False, bin
    """
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (15,10))
    kde: whether to show the density curve (default False)
    bins: number of bins for histogram (default None)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=data, x=feature, ax=ax_box2, showmeans=True, color="viol
    )  # boxplot will be created and a triangle will indicate the mea
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins
    ) if bins else sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data[feature].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

In [ ]:

```
# function to create labeled barplots


def labeled_barplot(data, feature, perc=False, n=None):
    """
    Barplot with percentage at the top

    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of count (default is
    n: displays the top n category levels (default is None, i.e., dis
    """

    total = len(data[feature])  # Length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 2, 6))
    else:
        plt.figure(figsize=(n + 2, 6))

    plt.xticks(rotation=90, fontsize=15)
    ax = sns.countplot(
        data=data,
        x=feature,
        palette="Paired",
        order=data[feature].value_counts().index[:n],
    )

    for p in ax.patches:
        if perc == True:
            label = "{:.1f}%".format(
                100 * p.get_height() / total
            )  # percentage of each class of the category
        else:
            label = p.get_height()  # count of each level of the cate

        x = p.get_x() + p.get_width() / 2  # width of the plot
        y = p.get_height()  # height of the plot

        ax.annotate(
            label,
            (x, y),
            ha="center",
            va="center",
            size=12,
            xytext=(0, 5),
            textcoords="offset points",
        )  # annotate the percentage

    plt.show()  # show the plot
```

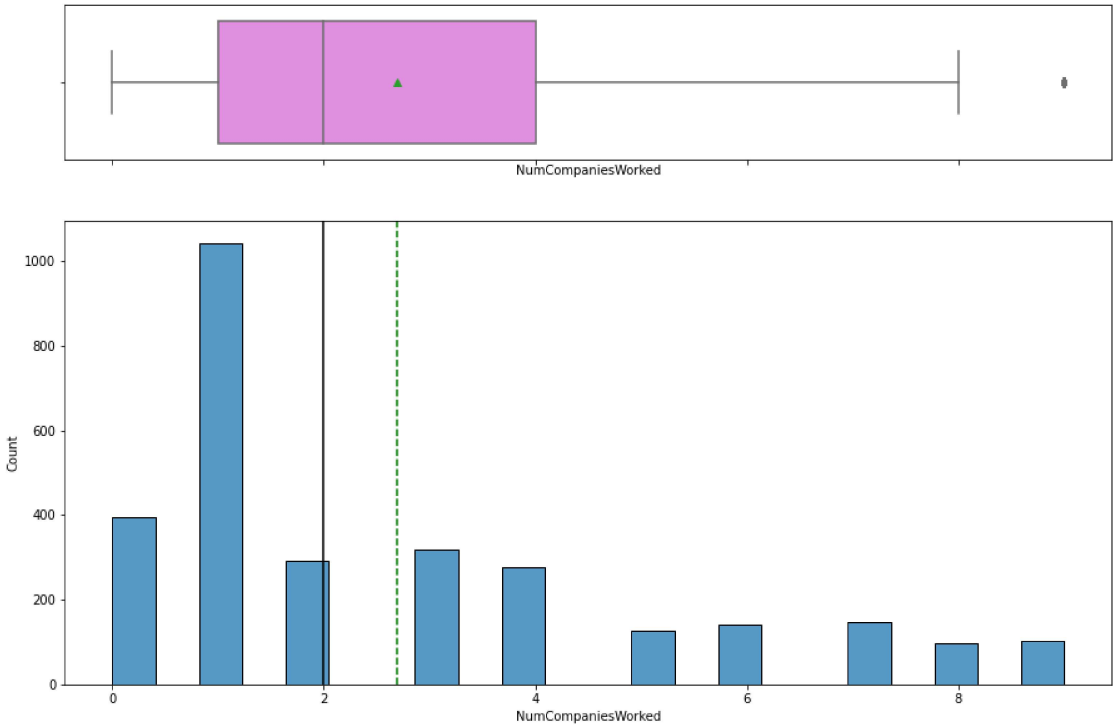
In [ ]:

```
# function to plot stacked bar chart


def stacked_barplot(data, predictor, target):
    """
    Print the category counts and plot a stacked bar chart

    data: dataframe
    predictor: independent variable
    target: target variable
    """
    count = data[predictor].nunique()
    sorter = data[target].value_counts().index[-1]
    tab1 = pd.crosstab(data[predictor], data[target], margins=True).s
        by=sorter, ascending=False
    )
    print(tab1)
    print("-" * 120)
    tab = pd.crosstab(data[predictor], data[target], normalize="index
        by=sorter, ascending=False
    )
    tab.plot(kind="bar", stacked=True, figsize=(count + 1, 5))
    plt.legend(
        loc="lower left",
        frameon=False,
    )
    plt.legend(loc="upper left", bbox_to_anchor=(1, 1))
    plt.show()
```

Univariate analysis

Observations on NumCompaniesWorked

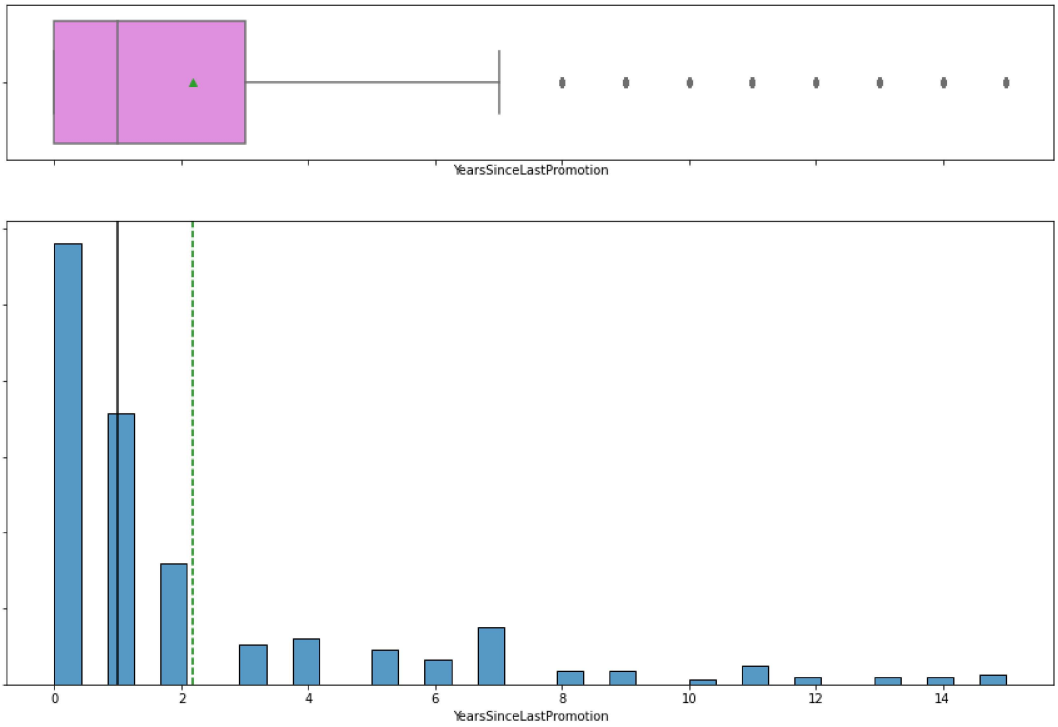
In [ ]:
```
1 histogram_boxplot(data,'NumCompaniesWorked')
```



- On average, people have worked at 2.5 companies. Median is 2.
- Most people have worked at only 1 company.
- Nearly 350 employees have worked at 0 companies, clearly this means this variable indicates the number of companies worked at before joining ours.
- There is an outlier employee who has changed 9 companies.

Observations on YearsInCurrentRole

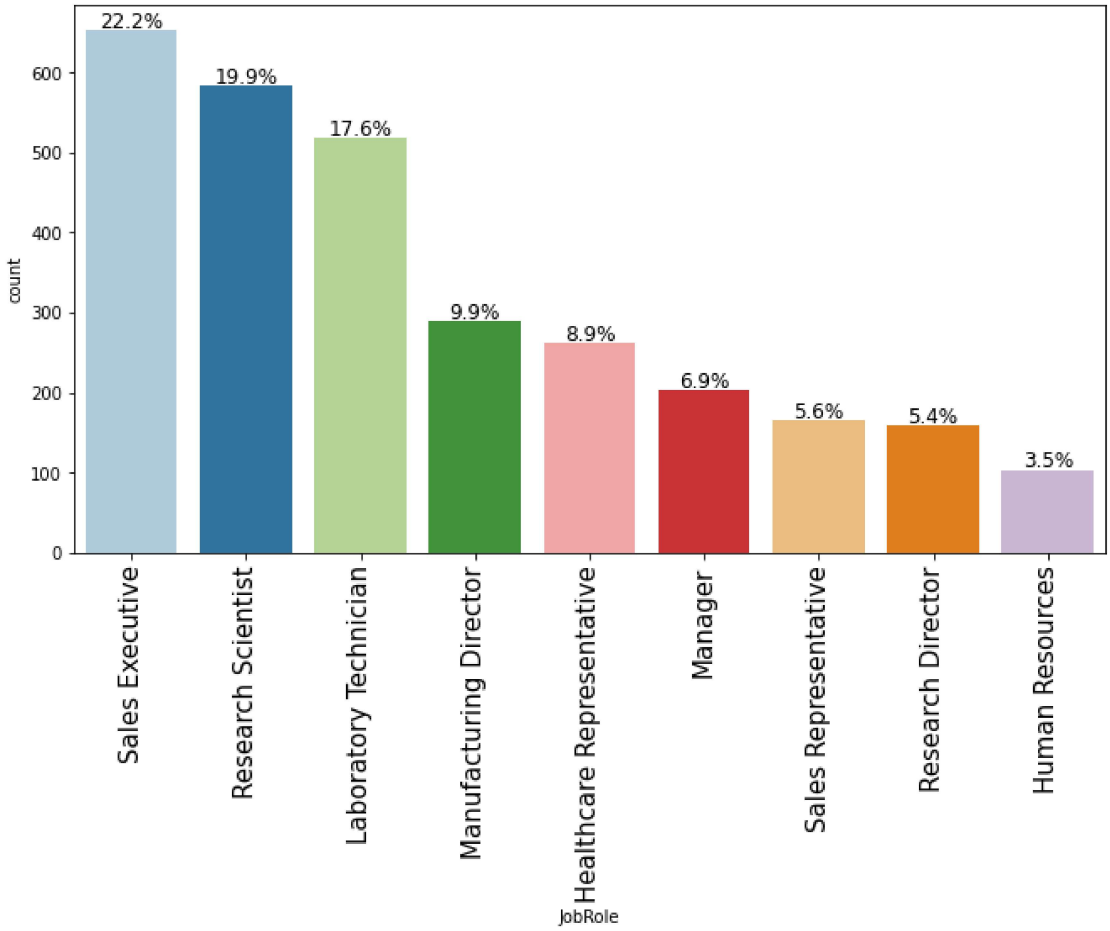
In [ ]:
```
1 histogram_boxplot(data,'YearsSinceLastPromotion')
```



- There are a few outliers in this right-skewed distribution, these are probably the people at the highest positions.
- Most employees have had a promotion in the last 2 years.
- 0 years since last promotion indicates many employees were recently promoted.

Observations on JobRole

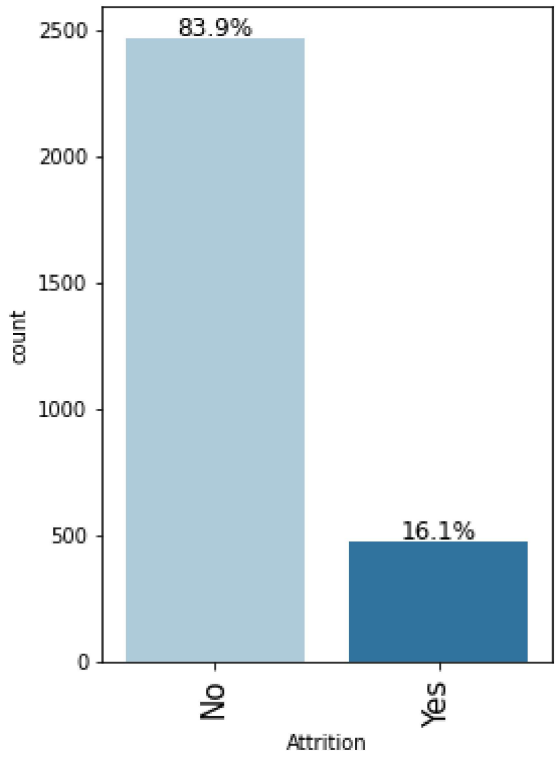
In [ ]:
```
1 labeled_barplot(data, "JobRole", perc=True)
```



- 22.2% of employees are Sales Executives followed by 20% of Research Scientists.

Observations on Attrition

In [ ]:
```
1 labeled_barplot(data, "Attrition", perc=True)
```



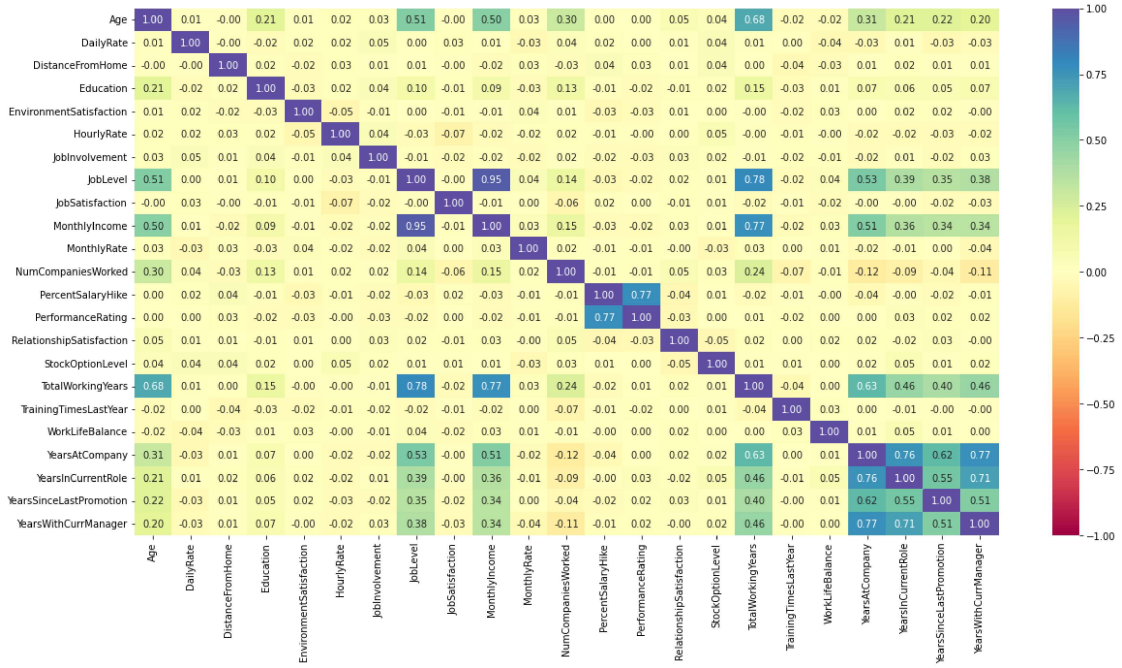
- 16% of the data points represent the employees who are going to attrite.

Bivariate Analysis

Correlation check

In [ ]:

```
1 plt.figure(figsize=(20,10))
2 sns.heatmap(data.corr(),annot=True,vmin=-1,vmax=1,fmt='.2f',cmap="Spe
3 plt.show()
```

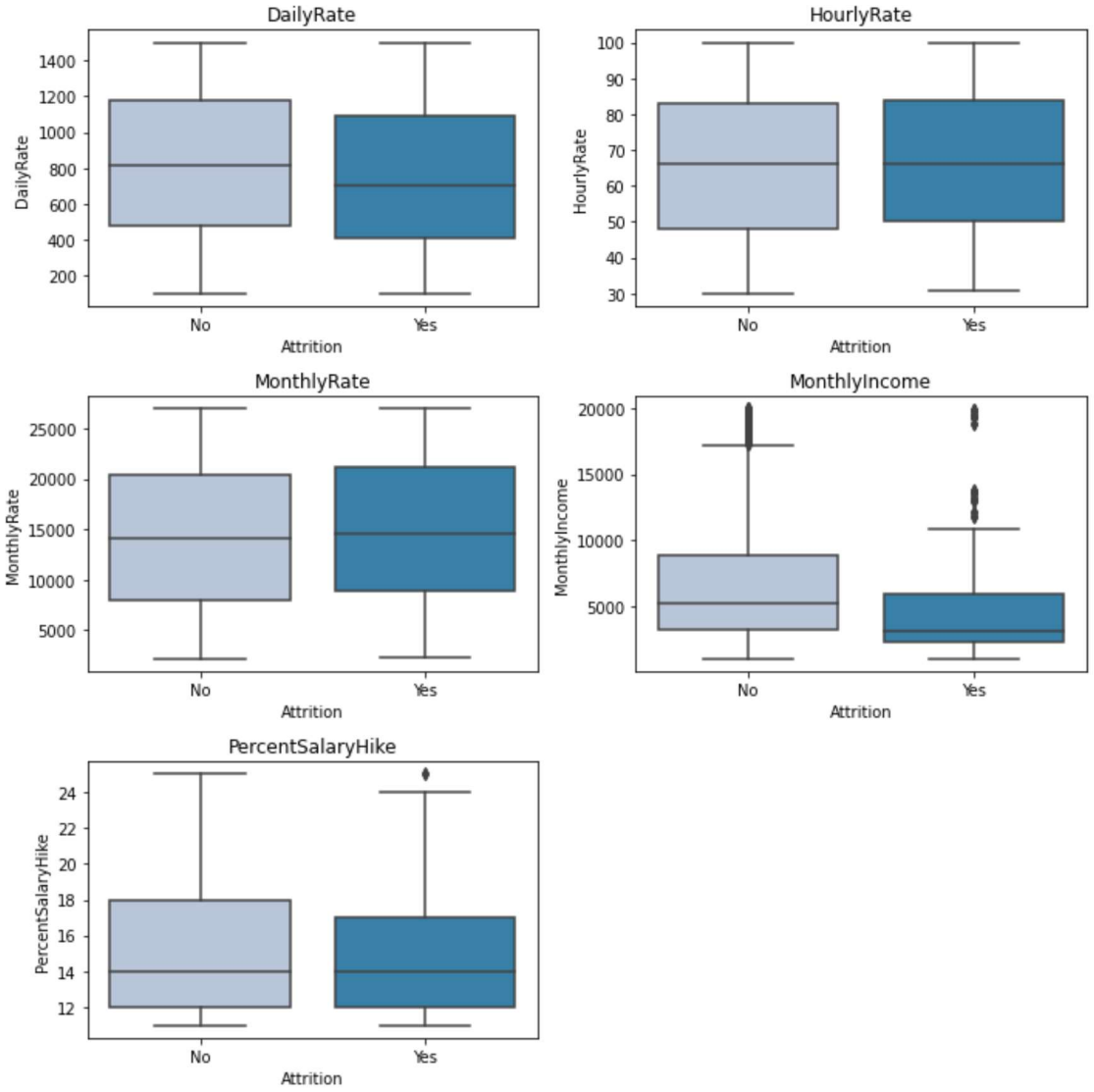


- There are a few variables that are correlated with each other but there are no surprises here.
- Unsurprisingly, TotalWorkingYears is highly correlated to Job Level (i.e., the longer you work the higher job level you achieve).
- HourlyRate, DailyRate, and MonthlyRate are completely uncorrelated with each other which makes it harder to understand what these variables might represent.
- MonthlyIncome is highly correlated to Job Level.
- Age is positively correlated JobLevel and Education (i.e., the older an employee is, the more educated and at a higher job level they are).
- Work-life Balance is correlated with none of the numeric values.

Attrition vs Earnings of employee

In [ ]:

```
cols = data[['DailyRate','HourlyRate','MonthlyRate','MonthlyIncome','
plt.figure(figsize=(10,10))

for i, variable in enumerate(cols):
                    plt.subplot(3,2,i+1)
                    sns.boxplot(data["Attrition"],data[variable],pal
                    plt.tight_layout()
                    plt.title(variable)
plt.show()
```

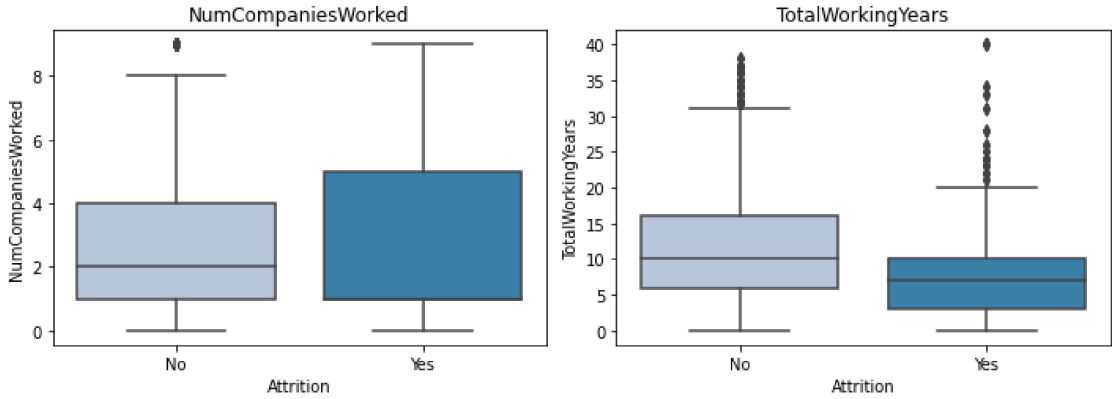


- Employees having lower Daily rate and less monthly wage are more likely to attrite.
- Monthly rate and the hourly rate doesn't seem to have any effect on attrition.
- Lesser salary hike also contributes to attrition.

Attrition vs Previous job roles

In [ ]:

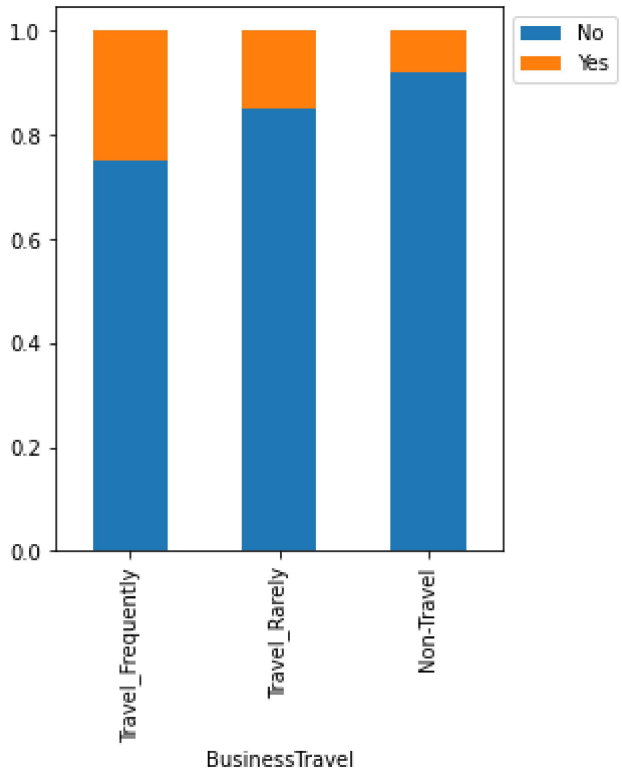
```
cols = data[['NumCompaniesWorked','TotalWorkingYears']].columns.tolis
plt.figure(figsize=(10,10))

for i, variable in enumerate(cols):
                    plt.subplot(3,2,i+1)
                    sns.boxplot(data["Attrition"],data[variable],pal
                    plt.tight_layout()
                    plt.title(variable)
plt.show()
```



- Employees who have worked in more companies generally tend to switch more jobs hence attriting.
- Employees who attrite generally have lesser years of experience.

In [ ]:
```
1 stacked_barplot(data, "BusinessTravel", "Attrition")
```

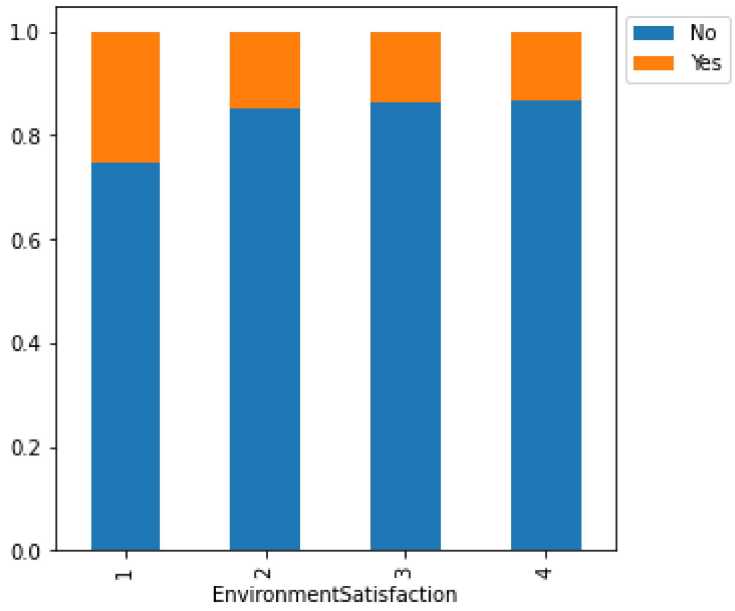
```
Attrition            No  Yes   All
BusinessTravel
All                2466  474  2940
Travel_Rarely      1774  312  2086
Travel_Frequently   416  138   554
Non-Travel          276   24   300
-------------------------------------------------------------------------------
-----------------------------------------------
```



- As the travel frequency increases, the Attrition rate increases.
- There's ~22% probability of employees attriting who travel frequently.

In [ ]:
```
1 stacked_barplot(data,"EnvironmentSatisfaction","Attrition")
```

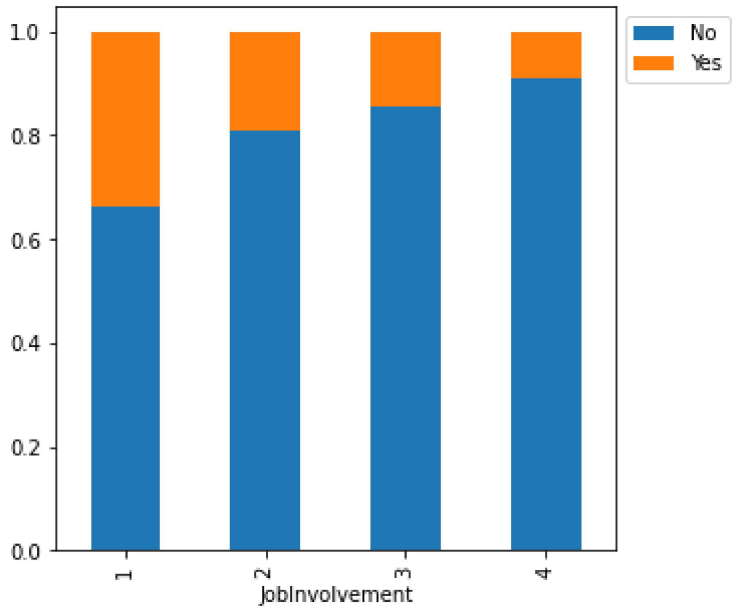
```
Attrition                  No  Yes   All
EnvironmentSatisfaction
All                      2466  474  2940
1                         424  144   568
3                         782  124   906
4                         772  120   892
2                         488   86   574
-------------------------------------------------------------------------------
------------------------------------------------
```



- Employees who say they have low satisfaction with their work environments are likely to attrite.
- There's a ~40% probability of attrition among employees with low ratings for environment satisfaction.

In [ ]:
```
1 stacked_barplot(data,"JobInvolvement","Attrition")
```

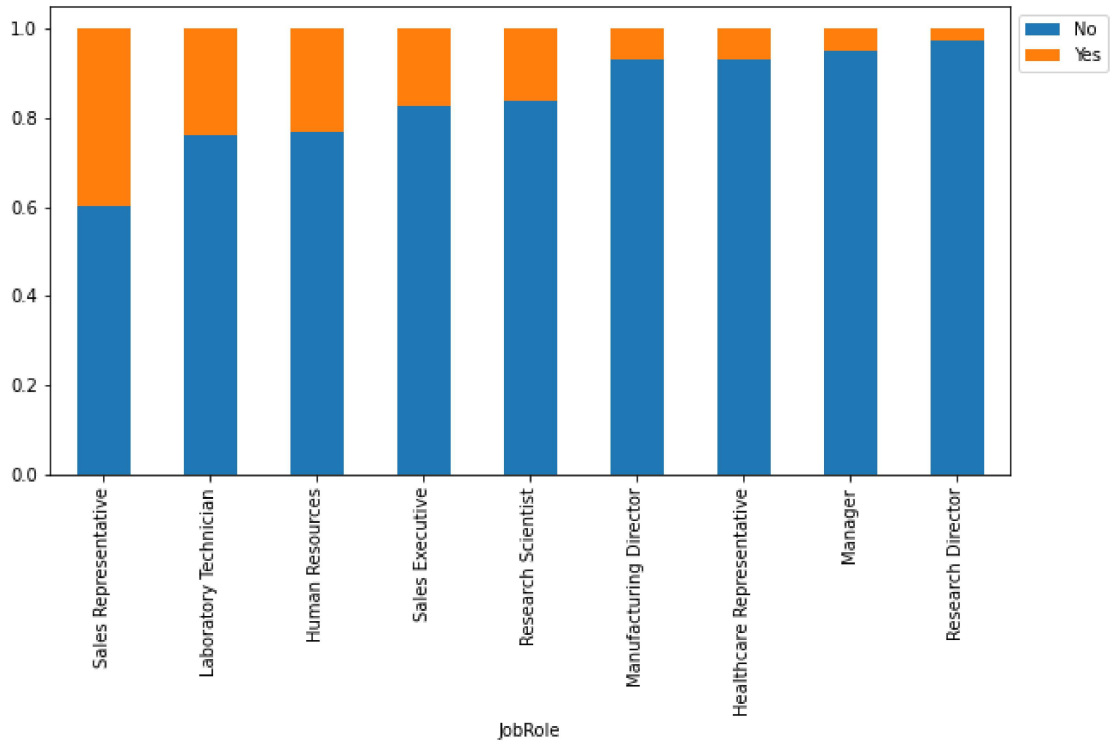
```
Attrition         No  Yes   All
JobInvolvement
All             2466  474  2940
3               1486  250  1736
2                608  142   750
1                110   56   166
4                262   26   288
-------------------------------------------------------------------------------
-----------------------------------------------
```



- Job Involvement looks like a very strong indicator of attrition.
- Higher the job involvement, greater is the chance that the employee will stay with us and not attrite.
- Employees unhappy with their job involment have ~55% probability of attriting (those who rated 0 and 1).
- Further investigation to understand how this variable was collected will give more insights.

In [ ]:
```
1 stacked_barplot(data,"JobRole","Attrition")
```

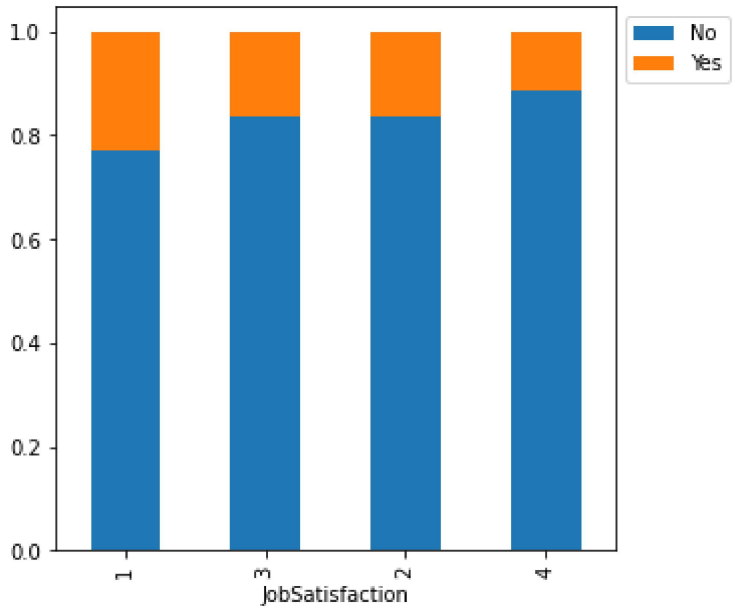
```
Attrition                    No  Yes   All
JobRole
All                        2466  474  2940
Laboratory Technician       394  124   518
Sales Executive             538  114   652
Research Scientist          490   94   584
Sales Representative        100   66   166
Human Resources              80   24   104
Manufacturing Director      270   20   290
Healthcare Representative   244   18   262
Manager                     194   10   204
Research Director           156    4   160
------------------------------------------------------------------------
-------------------------------------------------
```



- Sales Executives have an attrition probability of >40%.
- Laboratory Technicians and Human Resource personnel also have high probabilities of attrition.
- Attrition probability among Research Directors, Manufacturing directors Healthcare representatives, and Managers is much lower than the average attrition probability of 16%.

In [ ]:
```
1 stacked_barplot(data,"JobSatisfaction","Attrition")
```

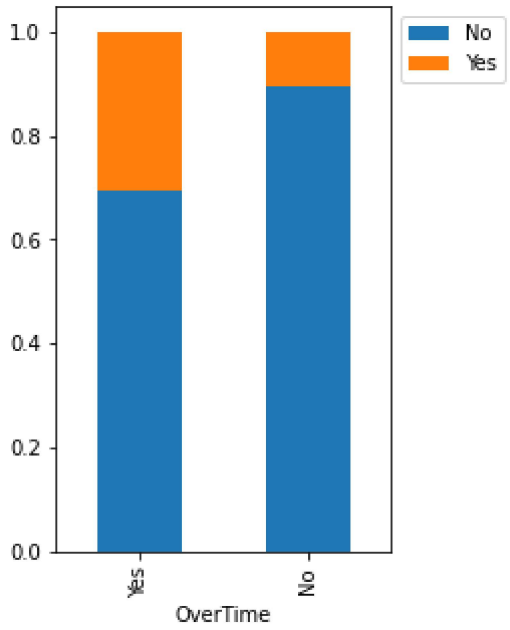
```
Attrition           No  Yes   All
JobSatisfaction
All               2466  474  2940
3                  738  146   884
1                  446  132   578
4                  814  104   918
2                  468   92   560
------------------------------------------------------------------------
------------------------------------------------
```



- As Job satisfaction increases, attrition probability decreases. This is intuitive but the attrition probability of people who rate 2 and 3 being almost the same is peculiar.

In [ ]:
```
1 stacked_barplot(data,"OverTime","Attrition")
```

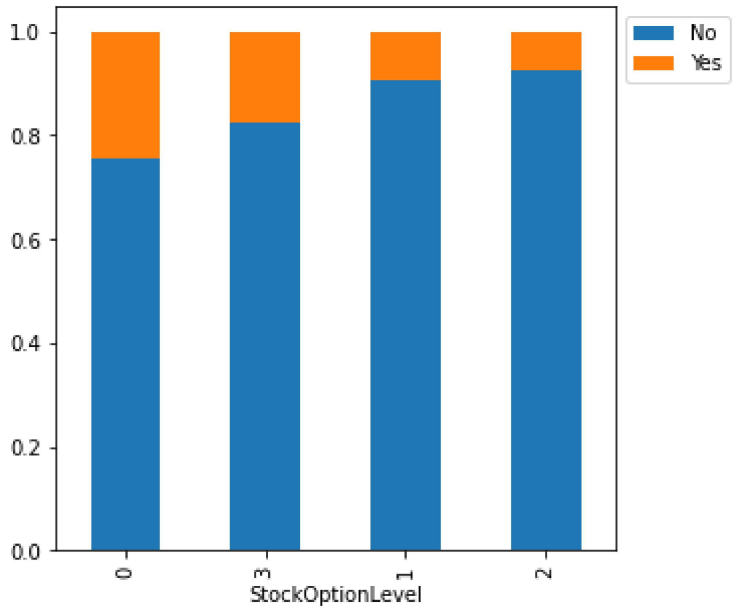
```
Attrition    No  Yes   All
OverTime
All        2466  474  2940
Yes         578  254   832
No         1888  220  2108
--------------------------------------------------------------------------------
------------------------------------------------
```



- Employees who work overtime tend to attrite more.
- There a ~35% probability of attrition among employees working overtime.

In [ ]:
```
1 stacked_barplot(data,"StockOptionLevel","Attrition")
```

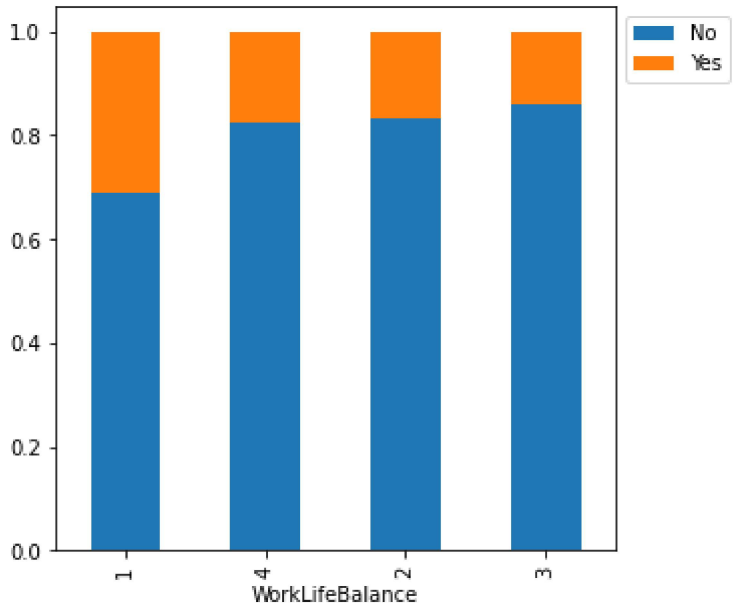
```
Attrition           No  Yes   All
StockOptionLevel
All               2466  474  2940
0                  954  308  1262
1                 1080  112  1192
3                  140   30   170
2                  292   24   316
-------------------------------------------------------------------------------
-----------------------------------------------
```



- ~22% Employees with highest and lowest stock options attrite the more than others.
- Company should investigate more on why employees with highest stock options are attriting and take this as an opportunity to re-consider their stocks policy.

In [ ]:
```
1 stacked_barplot(data,"WorkLifeBalance","Attrition")
```

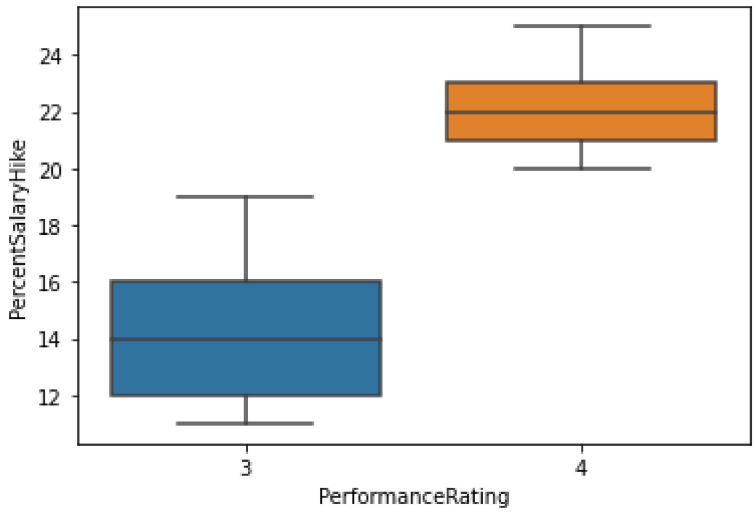
```
Attrition          No  Yes   All
WorkLifeBalance
All              2466  474  2940
3                1532  254  1786
2                 572  116   688
4                 252   54   306
1                 110   50   160
------------------------------------------------------------------------
-----------------------------------------------
```



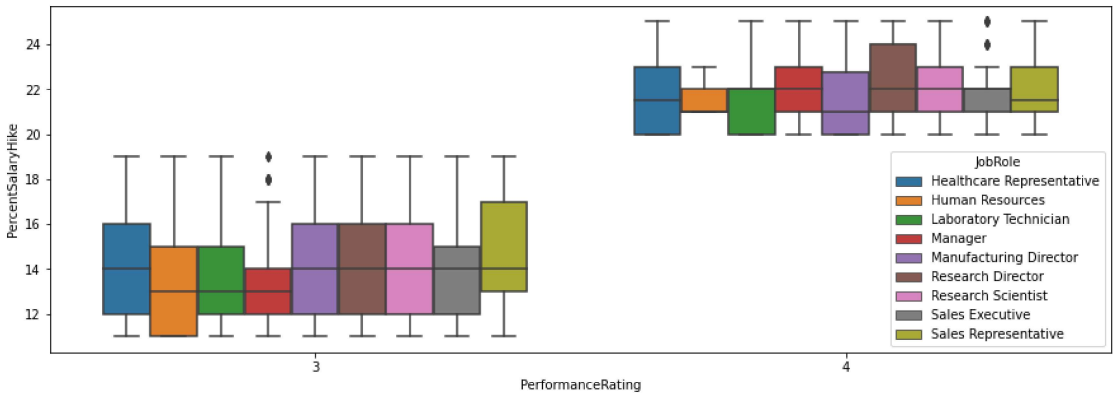
- Low work-life balance rating leads people to attrite, this is a good factor to preempt at attrition risk employees.

Checking if performace rating and salary hike are related-

In [ ]:
```
sns.boxplot(data['PerformanceRating'],data['PercentSalaryHike'])
plt.show()
```



In [ ]:
```
plt.figure(figsize=(15,5))
sns.boxplot(data['PerformanceRating'],data['PercentSalaryHike'],hue=d
plt.show()
```



Observations-

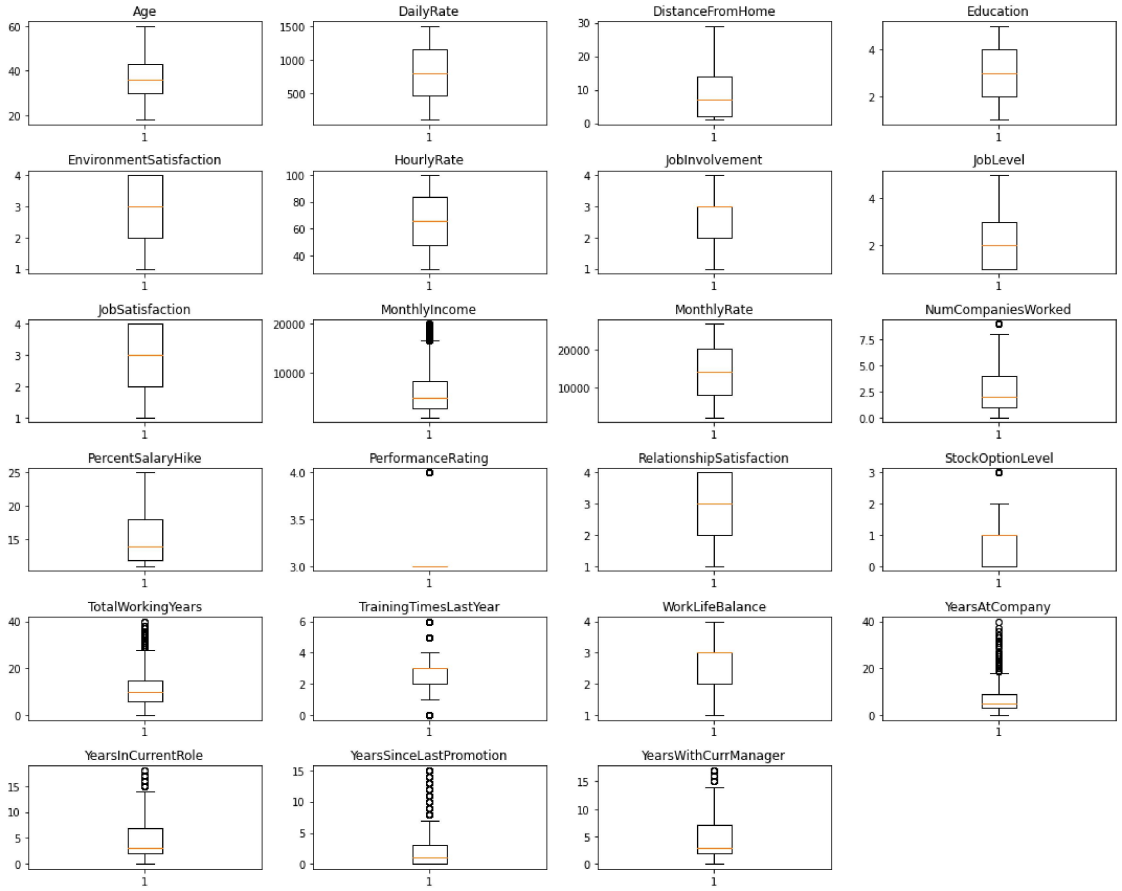
- Salary hikes are a function of Performance ratings.
- We have to investigate why the employees who get Excellent(3) and Outstanding(4) Performance rating attrite and how then can they be retained.

Data Preprocessing

Outlier Detection and Treatment

In [ ]:

```
# outlier detection using boxplot
numeric_columns = data.select_dtypes(include=np.number).columns.tolis


plt.figure(figsize=(15, 12))

for i, variable in enumerate(numeric_columns):
    plt.subplot(6, 4, i + 1)
    plt.boxplot(data[variable], whis=1.5)
    plt.tight_layout()
    plt.title(variable)

plt.show()
```



- There are quite a few outliers in the data.
- However, we will not treat them as they are proper values.

Data Preparataion for model building

- When classification problems exhibit a significant imbalance in the distribution of the target classes, it is good to use stratified sampling to ensure that relative class frequencies are approximately preserved in train and test sets.
- This is done using the `stratify` parameter in the train_test_split function.

In [ ]:
```
data['Attrition'] = data['Attrition'].apply(lambda x : 1 if x=='Yes'

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

X = pd.get_dummies(X,drop_first=True)
```

In [ ]:
```
# Splitting data into training and test set:
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.
print(X_train.shape, X_test.shape)
```

```
(2058, 44) (882, 44)
```

In [ ]:
```
y.value_counts(1)
```

Out[41]:
```
0    0.838776
1    0.161224
Name: Attrition, dtype: float64
```

In [ ]:
```
y_test.value_counts(1)
```

Out[42]:
```
0    0.839002
1    0.160998
Name: Attrition, dtype: float64
```

Let's define function to provide metric scores(accuracy,recall and precision) on train and test set and a function to show confusion matrix so that we do not have use the same code repetitively while evaluating models.

In [ ]:

```
# defining a function to compute different metrics to check performan
def model_performance_classification_sklearn(model, predictors, targe
    """
    Function to compute different metrics to check classification mod

    model: classifier
    predictors: independent variables
    target: dependent variable
    """

    # predicting using the independent variables
    pred = model.predict(predictors)

    acc = accuracy_score(target, pred)  # to compute Accuracy
    recall = recall_score(target, pred)  # to compute Recall
    precision = precision_score(target, pred)  # to compute Precision
    f1 = f1_score(target, pred)  # to compute F1-score

    # creating a dataframe of metrics
    df_perf = pd.DataFrame(
        {
            "Accuracy": acc,
            "Recall": recall,
            "Precision": precision,
            "F1": f1,
        },
        index=[0],
    )

    return df_perf
```

In [ ]:

```
def confusion_matrix_sklearn(model, predictors, target):
    """
    To plot the confusion_matrix with percentages

    model: classifier
    predictors: independent variables
    target: dependent variable
    """
    y_pred = model.predict(predictors)
    cm = confusion_matrix(target, y_pred)
    labels = np.asarray(
        [
            ["{0:0.0f}".format(item) + "\n{0:.2%}".format(item / cm.f
            for item in cm.flatten()
        ]
    ).reshape(2, 2)

    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=labels, fmt="")
    plt.ylabel("True label")
    plt.xlabel("Predicted label")
```

Model Building

Model evaluation criterion

Model can make wrong predictions as:

1. Predicting an employee will attrite and the employee doesn't attrite
2. Predicting an employee will not attrite and the employee attrites

Which case is more important?

- Predicting that employee will not attrite but he attrites i.e. losing on a valuable employee or asset.

How to reduce this loss i.e need to reduce False Negatives?

- Company wants Recall to be maximized, greater the Recall higher the chances of minimizing false negatives. Hence, the focus should be on increasing Recall or minimizing the false negatives or in other words identifying the true positives(i.e. Class 1) so that the company can provide incentives to control attrition rate especially for top-performers thereby optimizing the overall project cost in retaining the best talent.

Decision Tree Model

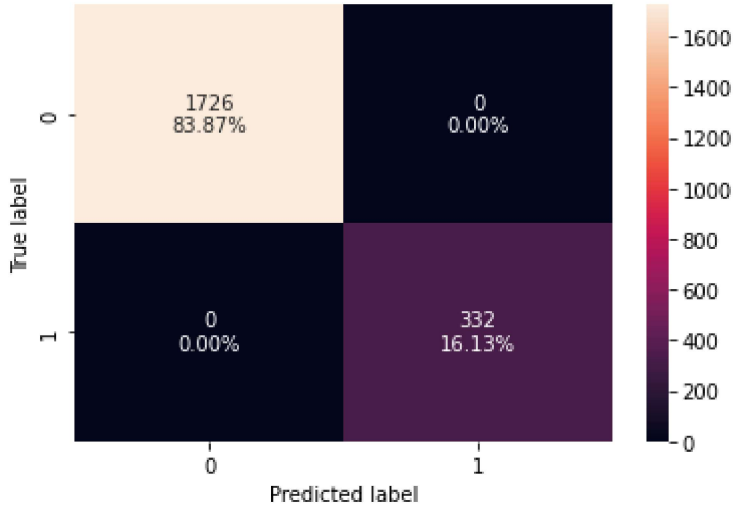
- We will build our model using the DecisionTreeClassifier function. Using default 'gini' criteria to split.
- If the frequency of class A is 10% and the frequency of class B is 90%, then class B will become the dominant class and the decision tree will become biased toward the dominant classes.
- In this case, we can pass a dictionary {0:0.17,1:0.83} to the model to specify the weight of each class and the decision tree will give more weightage to class 1.
- class_weight is a hyperparameter for the decision tree classifier.

In [ ]:
```
1 dtree = DecisionTreeClassifier(criterion='gini',class_weight={0:0.17,
```

In [ ]:
```
1 dtree.fit(X_train, y_train)
```

Out[46]:
```
DecisionTreeClassifier(class_weight={0: 0.17, 1: 0.83}, random_state=1)
```

In [ ]:
```
1 confusion_matrix_sklearn(dtree, X_train, y_train)
```



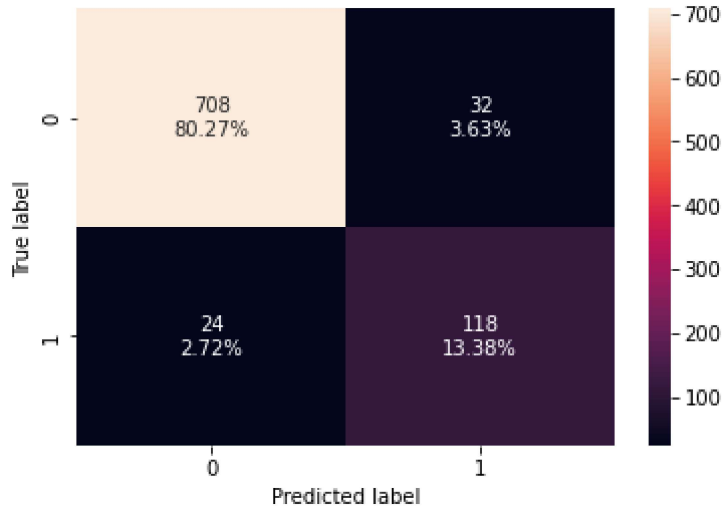
Confusion Matrix -

- Employee left and the model predicted it correctly that is employee will attrite : True Positive (observed=1,predicted=1)
- Employee didn't leave and the model predicted employee will attrite : False Positive (observed=0,predicted=1)
- Employee didn't leave and the model predicted employee will not attrite : True Negative (observed=0,predicted=0)
- Employee left and the model predicted that employee won't : False Negative (observed=1,predicted=0)

In [ ]:
```
1 dtree_model_train_perf=model_performance_classification_sklearn(dtree
2 print("Training performance \n",dtree_model_train_perf)
```

```
Training performance
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
```

In [ ]:
```
confusion_matrix_sklearn(dtree, X_test, y_test)
```



In [ ]:
```
dtree_model_test_perf=model_performance_classification_sklearn(dtree,
print("Testing performance \n",dtree_model_test_perf)
```

```
Testing performance
    Accuracy    Recall  Precision        F1
0  0.936508  0.830986   0.786667  0.808219
```

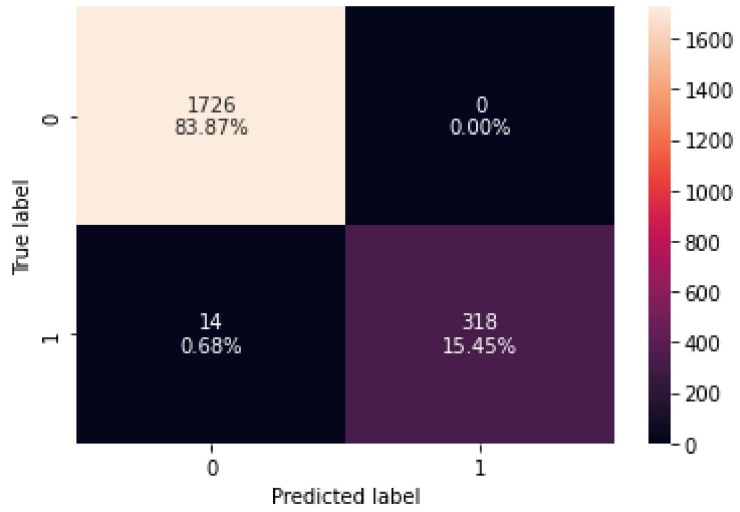
- Decision tree is working well on the training data but is not able to generalize well on the test data concerning the recall.

Bagging Classifier

In [ ]:
```
bagging = BaggingClassifier(random_state=1)
bagging.fit(X_train,y_train)
```

Out[51]:
```
BaggingClassifier(random_state=1)
```

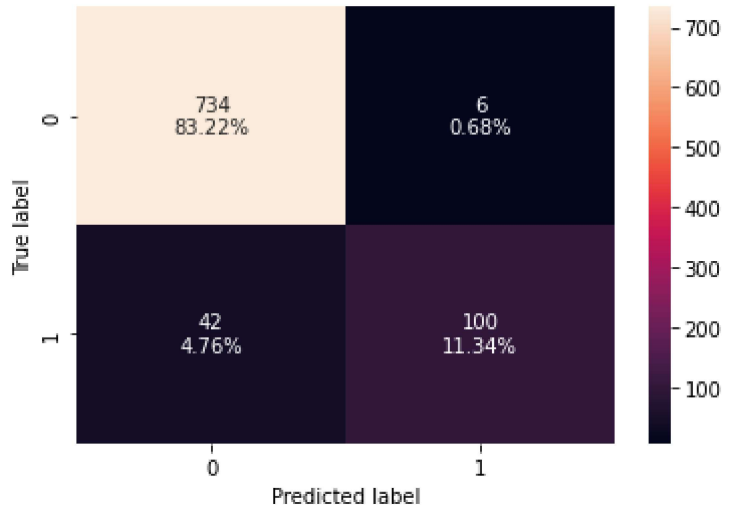
In [ ]:
```
1 confusion_matrix_sklearn(bagging, X_train, y_train)
```



In [ ]:
```
1 bagging_model_train_perf=model_performance_classification_sklearn(bag
2 print("Training performance \n",bagging_model_train_perf)
```

```
Training performance
    Accuracy    Recall  Precision        F1
0  0.993197  0.957831        1.0  0.978462
```

In [ ]:
```
1 confusion_matrix_sklearn(bagging, X_test, y_test)
```



In [ ]:
```
1 bagging_model_test_perf=model_performance_classification_sklearn(bagg
2 print("Testing performance \n",bagging_model_test_perf)
```

```
Testing performance
    Accuracy    Recall  Precision        F1
0  0.945578  0.704225   0.943396  0.806452
```

- Bagging classifier is overfitting on the training set and is performing poorly on the test set

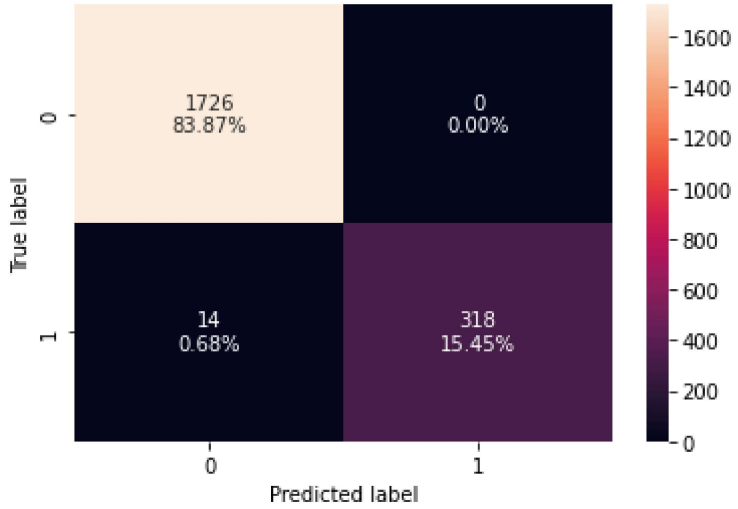
in terms of recall.

Bagging Classifier with weighted decision tree

In [ ]:
```
bagging_wt = BaggingClassifier(base_estimator=DecisionTreeClassifier(
bagging_wt.fit(X_train,y_train)
```

Out[56]:
```
BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight={0:
0.17,
                                                                      1:
0.83},
                                                        random_state=1),
                  random_state=1)
```

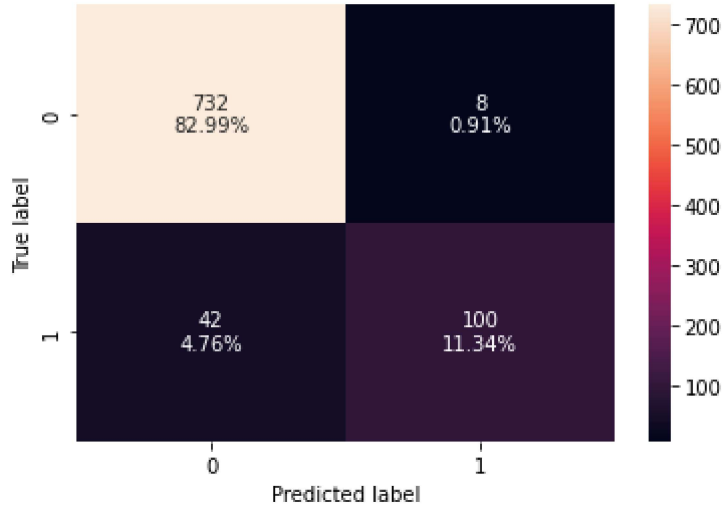
In [ ]:
```
confusion_matrix_sklearn(bagging_wt,X_train,y_train)
```



In [ ]:
```
bagging_wt_model_train_perf=model_performance_classification_sklearn(
print("Training performance \n",bagging_wt_model_train_perf)
```

```
Training performance
    Accuracy    Recall  Precision        F1
0  0.993197  0.957831        1.0  0.978462
```

In [ ]:
```
confusion_matrix_sklearn(bagging_wt,X_test,y_test)
```



In [ ]:
```
bagging_wt_model_test_perf=model_performance_classification_sklearn(b
print("Testing performance \n",bagging_wt_model_test_perf)
```

```
Testing performance
    Accuracy    Recall  Precision   F1
0  0.943311  0.704225   0.925926  0.8
```

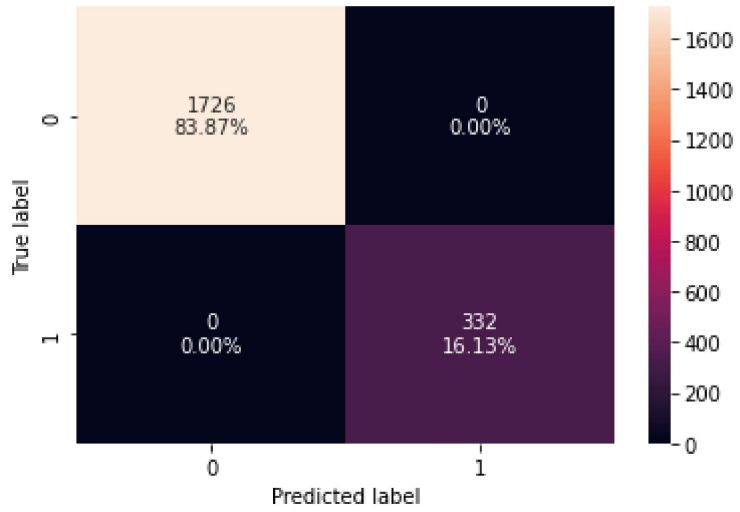
- Bagging classifier with a weighted decision tree is giving very good accuracy and prediction but is not able to generalize well on test data in terms of recall.

Random Forest

In [ ]:
```
rf = RandomForestClassifier(random_state=1)
rf.fit(X_train,y_train)
```

Out[61]:
```
RandomForestClassifier(random_state=1)
```

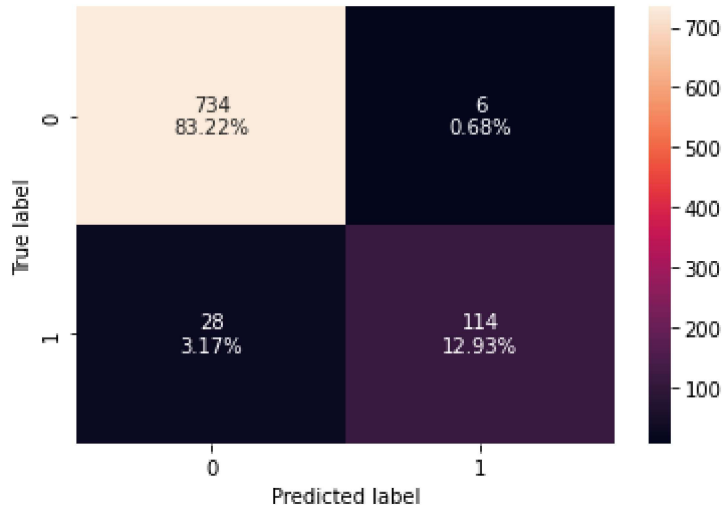
In [ ]:
```
confusion_matrix_sklearn(rf,X_train,y_train)
```



In [ ]:
```
rf_model_train_perf=model_performance_classification_sklearn(rf,X_tra
print("Training performance \n",rf_model_train_perf)
```

```
Training performance
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
```

In [ ]:
```
confusion_matrix_sklearn(rf,X_test,y_test)
```



In [ ]:
```
rf_model_test_perf=model_performance_classification_sklearn(rf,X_test
print("Testing performance \n",rf_model_test_perf)
```

```
Testing performance
    Accuracy    Recall  Precision        F1
0  0.961451  0.802817       0.95  0.870229
```

- Random Forest has performed well in terms of accuracy and precision, but it is not able to

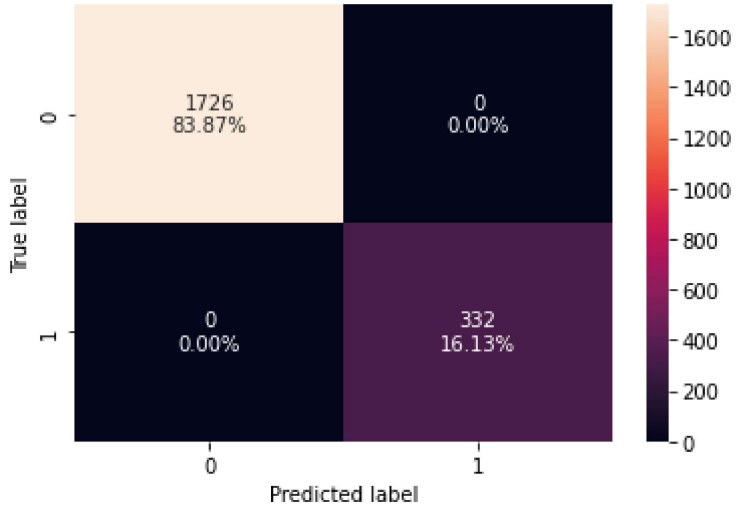
generalize well on the test data in terms of recall.

Random forest with class weights

In [ ]:
```
rf_wt = RandomForestClassifier(class_weight={0:0.17,1:0.83}, random_s
rf_wt.fit(X_train,y_train)
```

Out[66]:
```
RandomForestClassifier(class_weight={0: 0.17, 1: 0.83}, random_state=1)
```

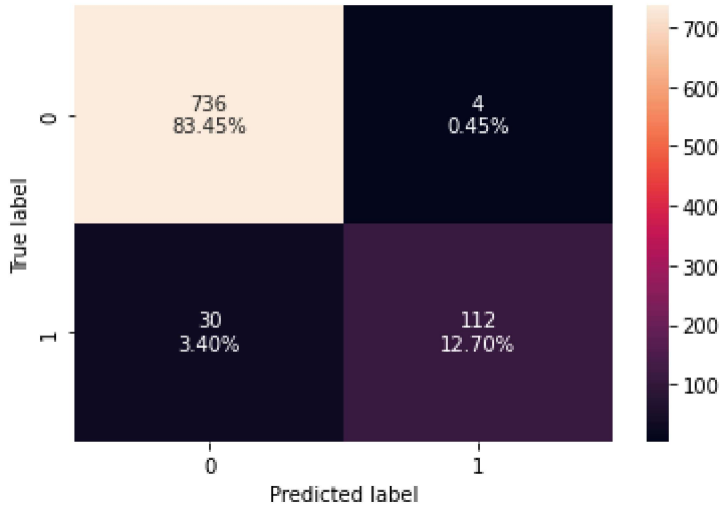
In [ ]:
```
confusion_matrix_sklearn(rf_wt, X_train,y_train)
```



In [ ]:
```
rf_wt_model_train_perf=model_performance_classification_sklearn(rf_wt
print("Training performance \n",rf_wt_model_train_perf)
```

```
Training performance
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
```

In [ ]:
```
confusion_matrix_sklearn(rf_wt, X_test,y_test)
```



In [ ]:

```
rf_wt_model_test_perf=model_performance_classification_sklearn(rf_wt,
print("Testing performance \n",rf_wt_model_test_perf)
```

```
Testing performance
    Accuracy    Recall  Precision        F1
0  0.961451  0.788732   0.965517  0.868217
```

- There is not much improvement in metrics of weighted random forest as compared to the unweighted random forest.

Tuning Models

Using GridSearch for Hyperparameter tuning model

- Hyperparameter tuning is also tricky in the sense that there is no direct way to calculate how a change in the hyperparameter value will reduce the loss of your model, so we usually resort to experimentation. i.e we'll use Grid search
- Grid search is a tuning technique that attempts to compute the optimum values of hyperparameters.
- It is an exhaustive search that is performed on a the specific parameter values of a model.
- The parameters of the estimator/model used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

Tuning Decision Tree

In [ ]:

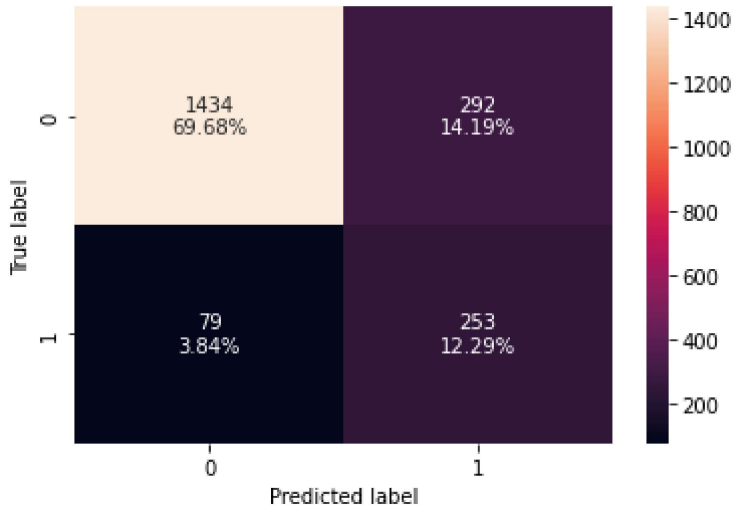
```
# Choose the type of classifier.
dtree_estimator = DecisionTreeClassifier(class_weight={0:0.17,1:0.83}

# Grid of parameters to choose from
parameters = {'max_depth': np.arange(2,30),
              'min_samples_leaf': [1, 2, 5, 7, 10],
              'max_leaf_nodes' : [2, 3, 5, 10,15],
              'min_impurity_decrease': [0.0001,0.001,0.01,0.1]
             }

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

# Run the grid search
grid_obj = GridSearchCV(dtree_estimator, parameters, scoring=scorer)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
dtree_estimator = grid_obj.best_estimator_

# Fit the best algorithm to the data.
dtree_estimator.fit(X_train, y_train)
```

Out[71]:

```
DecisionTreeClassifier(class_weight={0: 0.17, 1: 0.83}, max_depth=8,
                       max_leaf_nodes=15, min_impurity_decrease=0.0001,
                       min_samples_leaf=10, random_state=1)
```

In [ ]:

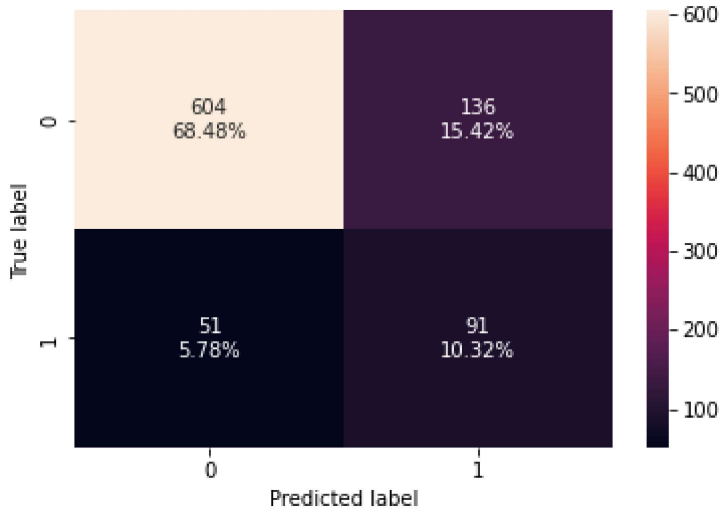
```
confusion_matrix_sklearn(dtree_estimator, X_train,y_train)
```



In [ ]:
```
dtree_estimator_model_train_perf=model_performance_classification_skl
print("Training performance \n",dtree_estimator_model_train_perf)
```

```
Training performance
    Accuracy    Recall  Precision        F1
0  0.819728  0.762048    0.46422  0.576967
```

In [ ]:
```
confusion_matrix_sklearn(dtree_estimator, X_test,y_test)
```



In [ ]:
```
dtree_estimator_model_test_perf=model_performance_classification_skle
print("Testing performance \n",dtree_estimator_model_test_perf)
```

```
Testing performance
    Accuracy    Recall  Precision        F1
0  0.787982  0.640845   0.400881  0.493225
```

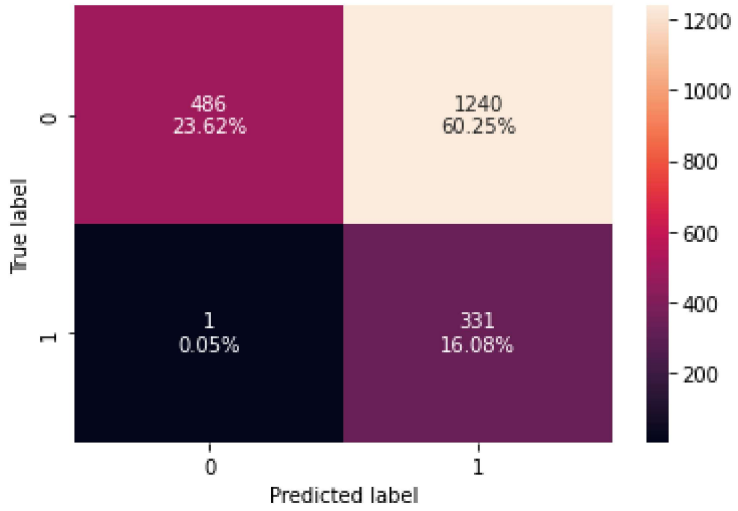
- Overfitting in decision tree has reduced but the recall has also reduced.

Tuning Bagging Classifier

In [ ]:
```
# grid search for bagging classifier
cl1 = DecisionTreeClassifier(class_weight={0:0.13,1:0.87},random_stat
param_grid = {'base_estimator':[cl1],
              'n_estimators':[5,7,15,51,101],
              'max_features': [0.7,0.8,0.9,1]
             }

grid = GridSearchCV(BaggingClassifier(random_state=1,bootstrap=True),
grid.fit(X_train, y_train)

## getting the best estimator
bagging_estimator  = grid.best_estimator_
bagging_estimator.fit(X_train,y_train)
```

Out[76]:
```
BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight={0:
0.13,
                                                        1:
0.87},
                                                        random_state=1),
                  max_features=1, n_estimators=51, random_state=1)
```

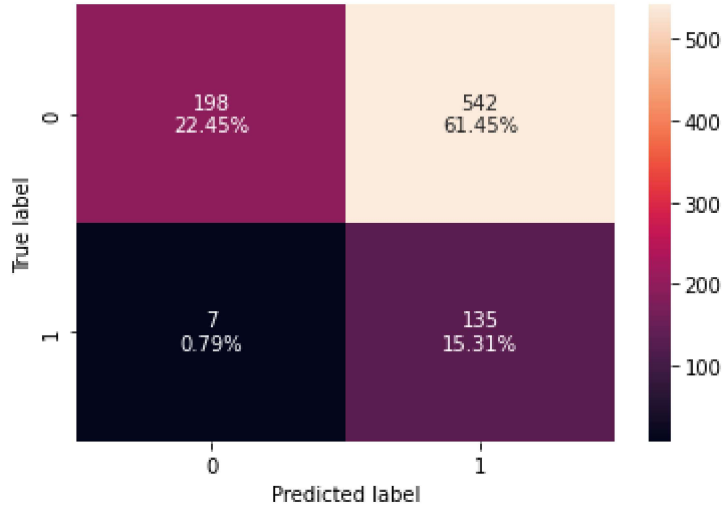
In [ ]:
```
confusion_matrix_sklearn(bagging_estimator, X_train,y_train)
```



In [ ]:
```
bagging_estimator_model_train_perf=model_performance_classification_s
print("Training performance \n",bagging_estimator_model_train_perf)
```

```
Training performance
    Accuracy    Recall  Precision        F1
0  0.396987  0.996988   0.210694  0.347872
```

In [ ]:
```
1 confusion_matrix_sklearn(bagging_estimator, X_test,y_test)
```



In [ ]:
```
1 bagging_estimator_model_test_perf=model_performance_classification_sk
2 print("Testing performance \n",bagging_estimator_model_test_perf)
```

```
Testing performance
    Accuracy    Recall  Precision       F1
0  0.377551  0.950704   0.199409  0.32967
```

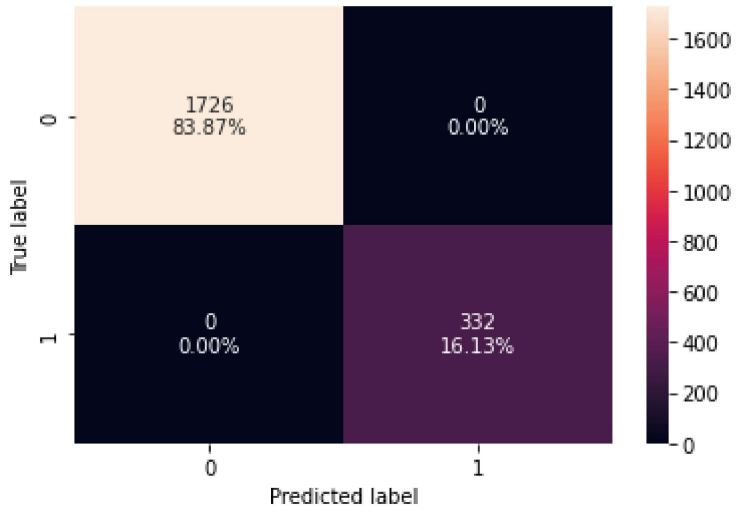
- Recall has improved but the accuracy and precision of the model has dropped drastically which is an indication that overall the model is making many mistakes.

Tuning Random Forest

In [ ]:
```
# Choose the type of classifier.
rf_estimator = RandomForestClassifier(random_state=1)

# Grid of parameters to choose from
parameters = {
        "n_estimators": [110,251,501],
        "min_samples_leaf": np.arange(1, 6,1),
        "max_features": [0.7,0.9,'log2','auto'],
        "max_samples": [0.7,0.9,None],
}

# Run the grid search
grid_obj = GridSearchCV(rf_estimator, parameters, scoring='recall',cv
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
rf_estimator = grid_obj.best_estimator_

# Fit the best algorithm to the data.
rf_estimator.fit(X_train, y_train)
```

Out[81]:
```
RandomForestClassifier(max_features=0.9, max_samples=0.9, n_estimators=1
10,
                       random_state=1)
```

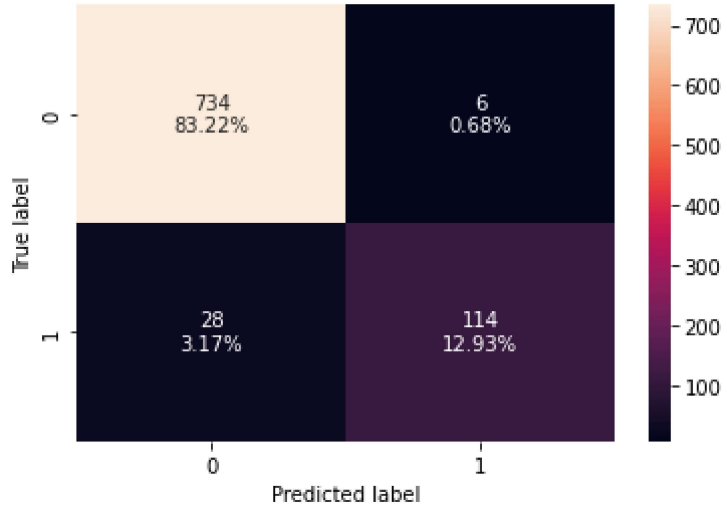
In [ ]:
```
confusion_matrix_sklearn(rf_estimator, X_train,y_train)
```



In [ ]:
```
rf_estimator_model_train_perf=model_performance_classification_sklear
print("Training performance \n",rf_estimator_model_train_perf)
```

```
Training performance
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
```

In [ ]:
```
1 confusion_matrix_sklearn(rf_estimator, X_test,y_test)
```



In [ ]:
```
1 rf_estimator_model_test_perf=model_performance_classification_sklearn
2 print("Testing performance \n",rf_estimator_model_test_perf)
```

```
Testing performance
    Accuracy    Recall  Precision        F1
0  0.961451  0.802817       0.95  0.870229
```

- Random forest after tuning has given same performance as un-tuned random forest.

Comparing all the models

In [ ]:

```
# training performance comparison

models_train_comp_df = pd.concat(
    [dtree_model_train_perf.T,bagging_model_train_perf.T, bagging_wt_
    rf_wt_model_train_perf.T,dtree_estimator_model_train_perf.T, bagg
     rf_estimator_model_train_perf.T],
    axis=1,
)
models_train_comp_df.columns = [
    "Decision Tree",
    "Bagging Classifier",
    "Weighted Bagging Classifier",
    "Random Forest Classifier",
    "Weighted Random Forest Classifier",
    "Decision Tree Estimator",
    "Bagging Estimator",
    "Random Forest Estimator"]
print("Training performance comparison:")
models_train_comp_df
```

```
Training performance comparison:
```

Out[86]:

	Decision Tree	Bagging Classifier	Weighted Bagging Classifier	Random Forest Classifier	Weighted Random Forest Classifier	Decision Tree Estimator	Bagging Estimator	Ran Fo Estim
Accuracy	1.0	0.993197	0.993197	1.0	1.0	0.819728	0.396987	
Recall	1.0	0.957831	0.957831	1.0	1.0	0.762048	0.996988	
Precision	1.0	1.000000	1.000000	1.0	1.0	0.464220	0.210694	
F1	1.0	0.978462	0.978462	1.0	1.0	0.576967	0.347872	

In [ ]:

```
# training performance comparison

models_test_comp_df = pd.concat(
    [dtree_model_test_perf.T,bagging_model_test_perf.T, bagging_wt_mo
    rf_wt_model_test_perf.T,dtree_estimator_model_test_perf.T, baggin
     rf_estimator_model_test_perf.T],
    axis=1,
)
models_test_comp_df.columns = [
    "Decision Tree",
    "Bagging Classifier",
    "Weighted Bagging Classifier",
    "Random Forest Classifier",
    "Weighted Random Forest Classifier",
    "Decision Tree Estimator",
    "Bagging Estimator",
    "Random Forest Estimator"]
print("Testing performance comparison:")
models_test_comp_df
```

```
Testing performance comparison:
```

Out[87]:

	Decision Tree	Bagging Classifier	Weighted Bagging Classifier	Random Forest Classifier	Weighted Random Forest Classifier	Decision Tree Estimator	Bagging Estimator	Ran Fo Estim
Accuracy	0.936508	0.945578	0.943311	0.961451	0.961451	0.787982	0.377551	0.961
Recall	0.830986	0.704225	0.704225	0.802817	0.788732	0.640845	0.950704	0.802
Precision	0.786667	0.943396	0.925926	0.950000	0.965517	0.400881	0.199409	0.950
F1	0.808219	0.806452	0.800000	0.870229	0.868217	0.493225	0.329670	0.870

- Decision tree performed well on training and test set.
- Bagging classifier overfitted the data before and after tuning.
- Random Forest with default parameters performed same as after tuning - As the final results depend on the parameters used/checked using GridSearchCV, There may be yet better parameters which may result in a better performance.

Feature importance of Random Forest

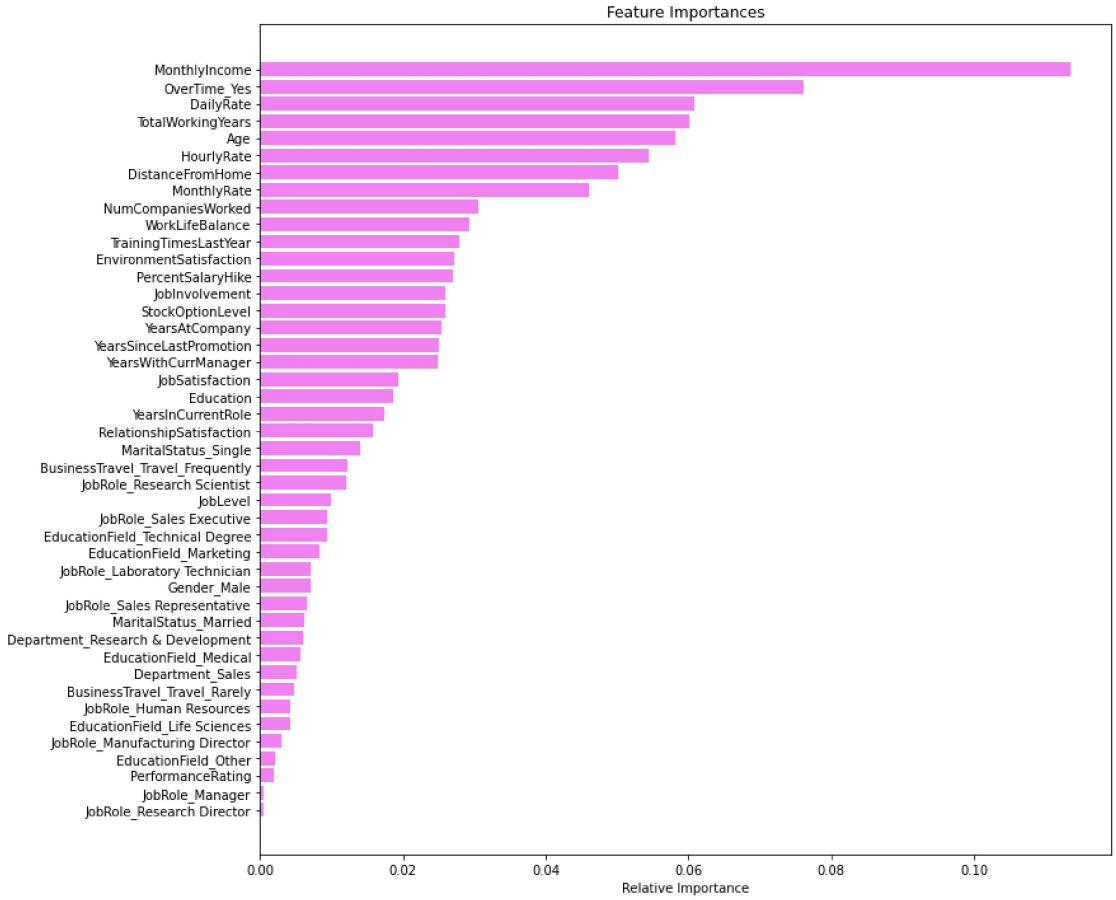
In [ ]:

```
# importance of features in the tree building ( The importance of a f
#(normalized) total reduction of the criterion brought by that featur

print (pd.DataFrame(rf.feature_importances_, columns = ["Imp"], index
```

	Imp
MonthlyIncome	0.080810
OverTime_Yes	0.059174
Age	0.055292
DailyRate	0.053302
TotalWorkingYears	0.052114
HourlyRate	0.050174
MonthlyRate	0.048463
DistanceFromHome	0.047056
YearsAtCompany	0.039540
PercentSalaryHike	0.031674
YearsWithCurrManager	0.031058
YearsInCurrentRole	0.029408
NumCompaniesWorked	0.029030
TrainingTimesLastYear	0.028230
EnvironmentSatisfaction	0.026136
StockOptionLevel	0.025781
JobInvolvement	0.025729
JobSatisfaction	0.025468
WorkLifeBalance	0.025087
JobLevel	0.023950
YearsSinceLastPromotion	0.023892
Education	0.021817
RelationshipSatisfaction	0.021528
MaritalStatus_Single	0.013513
BusinessTravel_Travel_Frequently	0.012322
Gender_Male	0.009384
MaritalStatus_Married	0.009269
JobRole_Research Scientist	0.008532
Department_Research & Development	0.008405
BusinessTravel_Travel_Rarely	0.008310
EducationField_Medical	0.007918
EducationField_Life Sciences	0.007847
EducationField_Technical Degree	0.007828
JobRole_Sales Representative	0.007542
Department_Sales	0.007284
EducationField_Marketing	0.007281
JobRole_Laboratory Technician	0.007070
JobRole_Sales Executive	0.005483
PerformanceRating	0.004814
JobRole_Human Resources	0.004303
JobRole_Manufacturing Director	0.003016
EducationField_Other	0.002766
JobRole_Manager	0.001687
JobRole_Research Director	0.000715

In [ ]:
```
feature_names = X_train.columns
```

In [ ]:
```
importances = rf_estimator.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(12,12))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='violet', a
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



- Monthly income is the most important feature for prediction followed by Overtime, Daily Rate and Age.

Business Insights and Recommendations

- We have been able to build a predictive model: a) that company can deploy this model to identify employees who are at the risk of attrition. b) that company can use to find the drivers of attrition. c) based on which company can take appropriate actions to build better retention policies.
- Factors that drive attrition - Monthly Income, Overtime, and Age.

- Monthly Income: Employees with lower income attrite more, which is also logical as they might get offers with higher pay in different organizations - the company should make sure that all employees are compensated based on industry standards.
- Overtime: Those employees who have to work overtime are the ones who attrite more - the company can provide some additional incentives to such employees to retain them.
- Age: Younger employees are the ones that attrite more- the company can make sure the new joiners have a friendly environment and better opportunities for excelling in their career.
- Distance From home is also an important factor for attrition - employees traveling more distance to reach the workplace are the ones attriting. For such employees, the company can provide cab facilities so that the commute of employees gets easier.
- As work-related travel frequency increases, Attrition rate also increases - the company should
- Training doesn't seem to have an impact on attrition- the company needs to investigate more here, if training does not impact employee retention then better cost planning can be done.
- Employee with more experience and the employees working for most years in the company is the loyal one's and generally do not attrite.
- Highest attrition is in the Sales department more research should go into this to check what is wrong in the sales department?
- Our data collection technique is working well as the ratings given by employees in - Environment Satisfaction, Job Satisfaction, Relationship Satisfaction, and Work-Life Balance shows a difference significant difference between attriting and non-attriting employees. These scales can act as a preliminary step to understand the dissatisfaction of employees - Lower the rating higher are the chances of attrition.