HR Analytics Project- Understanding the Attrition in HR

Problem Statement: Every year a lot of companies hire a number of employees. The companies invest time and money in training those employees, not just this but there are training programs within the companies for their existing employees as well. The aim of these programs is to increase the effectiveness of their employees. But where HR Analytics fit in this? and is it just about improving the performance of employees?

HR Analytics

Human resource analytics (HR analytics) is an area in the field of analytics that refers to applying analytic processes to the human resource department of an organization in the hope of improving employee performance and therefore getting a better return on investment. HR analytics does not just deal with gathering data on employee efficiency. Instead, it aims to provide insight into each process by gathering data and then using it to make relevant decisions about how to improve these processes.

Attrition in HR

Attrition in human resources refers to the gradual loss of employees overtime. In general, relatively high attrition is problematic for companies. HR professionals often assume a leadership role in designing company compensation programs, work culture, and motivation systems that help the organization retain top employees.

How does Attrition affect companies? and how does HR Analytics help in analyzing attrition? We will discuss the first question here and for the second question, we will write the code and try to understand the process step by step.

Attrition affecting Companies

A major problem in high employee attrition is its cost to an organization. Job postings, hiring processes, paperwork, and new hire training are some of the common expenses of losing employees and replacing them. Additionally, regular employee turnover prohibits your organization from increasing its collective knowledge base and experience over time. This is especially concerning if your business is customer-facing, as customers often prefer to interact with familiar people. Errors and issues are more likely if you constantly have new workers.

In [1]:

```
import warnings
warnings.simplefilter("ignore")
warnings.filterwarnings("ignore")
import joblib

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from scipy.stats import zscore

from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.neighbors import KNeighborsClassifier
import xgboost as xgb

from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
```

In [2]:
```
df=pd.read_csv('D:\\Project\\IBM_HR_Attrition_Rate_Analytics-master\\WA_Fn-UseC
df
```

Out[2]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	Ed
0	41	Yes	Travel_Rarely	1102	Sales	1	2	
1	49	No	Travel_Frequently	279	Research & Development	8	1	
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	
3	33	No	Travel_Frequently	1392	Research & Development	3	4	
4	27	No	Travel_Rarely	591	Research & Development	2	1	
...	...	...	...	...	...	...	...	
1465	36	No	Travel_Frequently	884	Research & Development	23	2	
1466	39	No	Travel_Rarely	613	Research & Development	6	1	
1467	27	No	Travel_Rarely	155	Research & Development	4	3	
1468	49	No	Travel_Frequently	1023	Sales	2	3	
1469	34	No	Travel_Rarely	628	Research & Development	8	3	

1470 rows × 35 columns

As we have dataset having 1470 rows and 35 columns below .in this 35 columns one is our target means Attrition and rest are features.

EDA

In [3]:
```
df.columns
```

Out[3]:
```
Index(['Age', 'Attrition', 'BusinessTravel', 'DailyRate', 'Department',
       'DistanceFromHome', 'Education', 'EducationField', 'EmployeeCount',
       'EmployeeNumber', 'EnvironmentSatisfaction', 'Gender', 'HourlyRate',
       'JobInvolvement', 'JobLevel', 'JobRole', 'JobSatisfaction',
       'MaritalStatus', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
       'Over18', 'OverTime', 'PercentSalaryHike', 'PerformanceRating',
       'RelationshipSatisfaction', 'StandardHours', 'StockOptionLevel',
       'TotalWorkingYears', 'TrainingTimesLastYear', 'WorkLifeBalance',
       'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion',
       'YearsWithCurrManager'],
      dtype='object')
```

In [24]:
```
df.isnull().sum()
```

Out[24]:
```
Age                         0
Attrition                   0
BusinessTravel              0
DailyRate                   0
Department                  0
DistanceFromHome            0
Education                   0
EducationField              0
EnvironmentSatisfaction     0
Gender                      0
HourlyRate                  0
JobInvolvement              0
JobLevel                    0
JobRole                     0
JobSatisfaction             0
MaritalStatus               0
MonthlyIncome               0
MonthlyRate                 0
NumCompaniesWorked          0
OverTime                    0
PercentSalaryHike           0
PerformanceRating           0
RelationshipSatisfaction    0
StockOptionLevel            0
TotalWorkingYears           0
TrainingTimesLastYear       0
WorkLifeBalance             0
YearsAtCompany              0
YearsInCurrentRole          0
YearsSinceLastPromotion     0
YearsWithCurrManager        0
dtype: int64
```

fortunately we dont have any null value present in dataset

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
```

As using to this method to know the type of data and we can say that most of our data are in int form but our target is in object form

In [5]:
```
df.describe().T
```

Out[5]:

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

from the above describe transpose method it is displaying that there is no missing data as every columns have same count (1470) And maybe columns like Monthlyincome, Totalworkingyears,YearsAtCompany,YearsinCurrentRole,YearSinceLastPromotion,YearWithCurrM has huge differece in their 75 % and 100% so they must have some amount of outlier that is need to be treated.

In [6]:
```
df.drop(["EmployeeCount", "EmployeeNumber", "Over18", "StandardHours"], axis=1,
```

EmployeeCount=All the data use to fill is same that is 1. so there would be no mean to have them in dataset EmployerNumber=It is just a unique number given to employer that doesnot effect our target data . Over18= It's just a Labour law in india that below 18 you just cant get employed StandardHours=Its also giving only single value for all our columns

In [7]:
```
df.shape
```

Out[7]:
```
(1470, 31)
```

In [8]:
```
object_datatypes=[]
for x in df.dtypes.index:
    if df.dtypes[x]=='O':
        object_datatypes.append(x)
object_datatypes
```

Out[8]:
```
['Attrition',
 'BusinessTravel',
 'Department',
 'EducationField',
 'Gender',
 'JobRole',
 'MaritalStatus',
 'OverTime']
```

here is the all columns having object type datatype and these are 8 in numbers.

In [9]:
```
integer_datatypes=[]
for x in df.dtypes.index:
    if df.dtypes[x]=='int64':
      integer_datatypes.append(x)
integer_datatypes
```

Out[9]:
```
['Age',
 'DailyRate',
 'DistanceFromHome',
 'Education',
 'EnvironmentSatisfaction',
 'HourlyRate',
 'JobInvolvement',
 'JobLevel',
 'JobSatisfaction',
 'MonthlyIncome',
 'MonthlyRate',
 'NumCompaniesWorked',
 'PercentSalaryHike',
 'PerformanceRating',
 'RelationshipSatisfaction',
 'StockOptionLevel',
 'TotalWorkingYears',
 'TrainingTimesLastYear',
 'WorkLifeBalance',
 'YearsAtCompany',
 'YearsInCurrentRole',
 'YearsSinceLastPromotion',
 'YearsWithCurrManager']
```

Here we have 23 columns holding integer datatype.

In [10]:
```
df.nunique().to_frame("Unique values")
```

Out[10]:

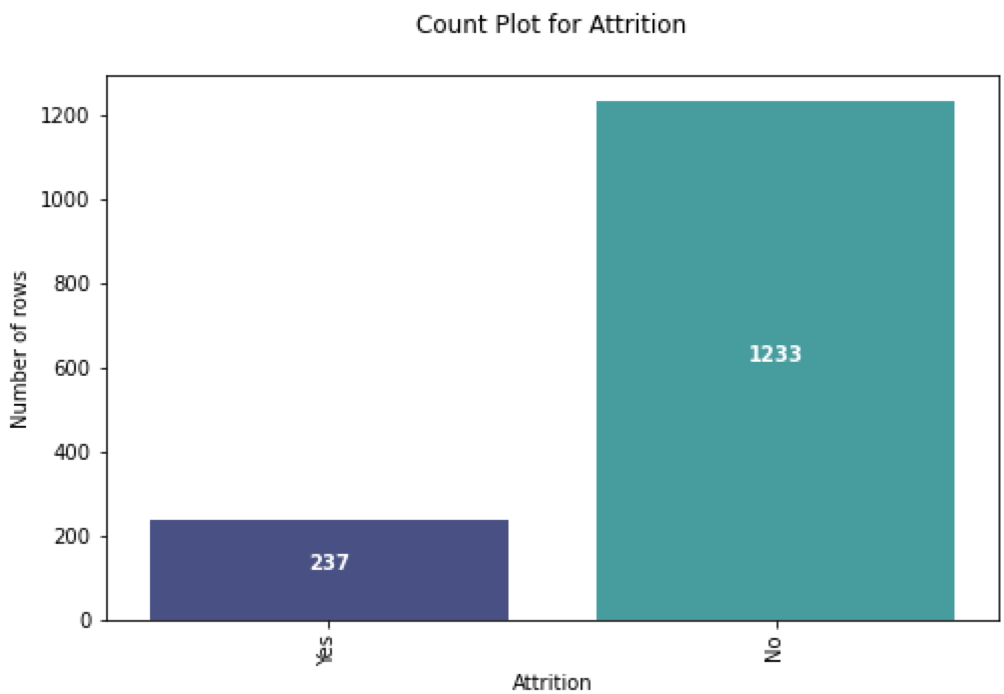
	Unique values
Age	43
Attrition	2
BusinessTravel	3
DailyRate	886
Department	3
DistanceFromHome	29
Education	5
EducationField	6
EnvironmentSatisfaction	4
Gender	2
HourlyRate	71
JobInvolvement	4
JobLevel	5
JobRole	9
JobSatisfaction	4
MaritalStatus	3
MonthlyIncome	1349
MonthlyRate	1427
NumCompaniesWorked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

In [11]:
```
for col in object_datatypes:
    print(col)
    print(df[col].value_counts())
    print("="*80)
```

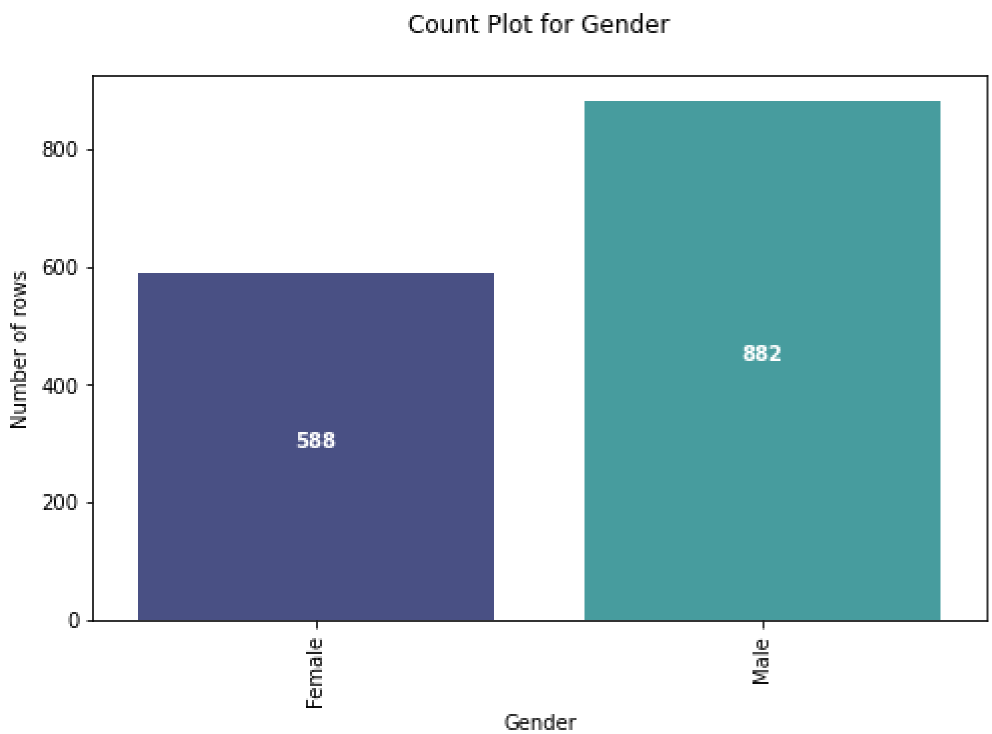
```
Attrition
No     1233
Yes     237
Name: Attrition, dtype: int64
===========================================================================
=====
BusinessTravel
Travel_Rarely        1043
Travel_Frequently     277
Non-Travel            150
Name: BusinessTravel, dtype: int64
===========================================================================
=====
Department
Research & Development    961
Sales                     446
Human Resources            63
Name: Department, dtype: int64
===========================================================================
```

Visualization

In [12]:
```
plt.figure(figsize=(8,5))
col_name = 'Attrition'
values = df[col_name].value_counts()
index = 0
ax = sns.countplot(df[col_name], palette="mako")

for i in ax.get_xticklabels():
    ax.text(index, values[i.get_text()]/2, values[i.get_text()],
            horizontalalignment="center", fontweight='bold', color='w')
    index += 1

plt.title(f"Count Plot for {col_name}\n")
plt.ylabel(f"Number of rows")
plt.xticks(rotation=90)
plt.show()
```

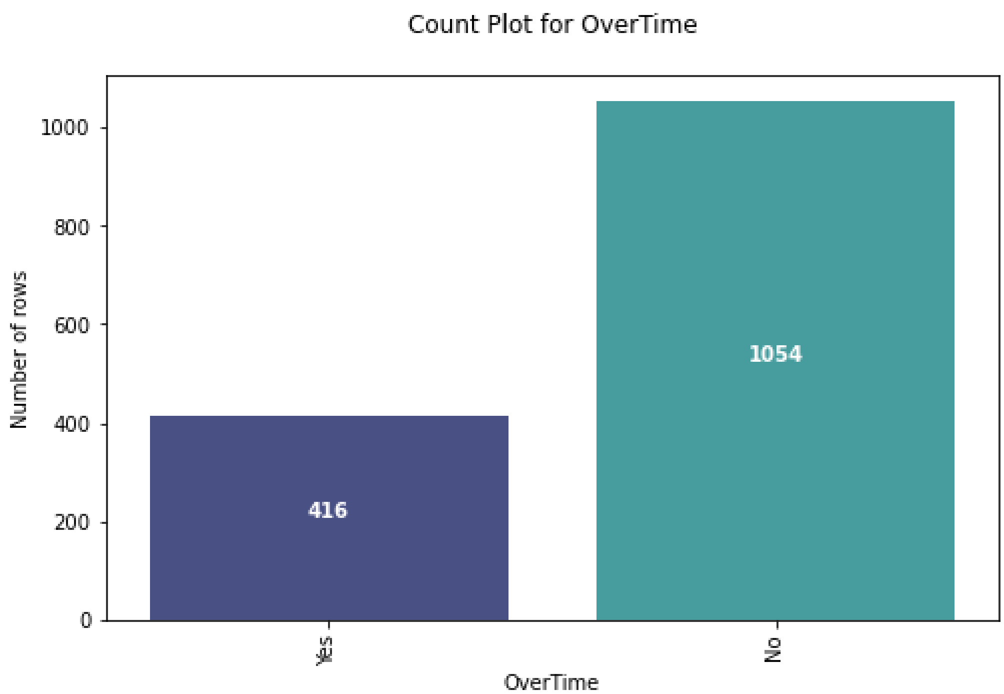


In [13]:
```
plt.figure(figsize=(8,5))
col_name = 'Gender'
values = df[col_name].value_counts()
index = 0
ax = sns.countplot(df[col_name], palette="mako")

for i in ax.get_xticklabels():
    ax.text(index, values[i.get_text()]/2, values[i.get_text()],
            horizontalalignment="center", fontweight='bold', color='w')
    index += 1

plt.title(f"Count Plot for {col_name}\n")
plt.ylabel(f"Number of rows")
plt.xticks(rotation=90)
plt.show()
```

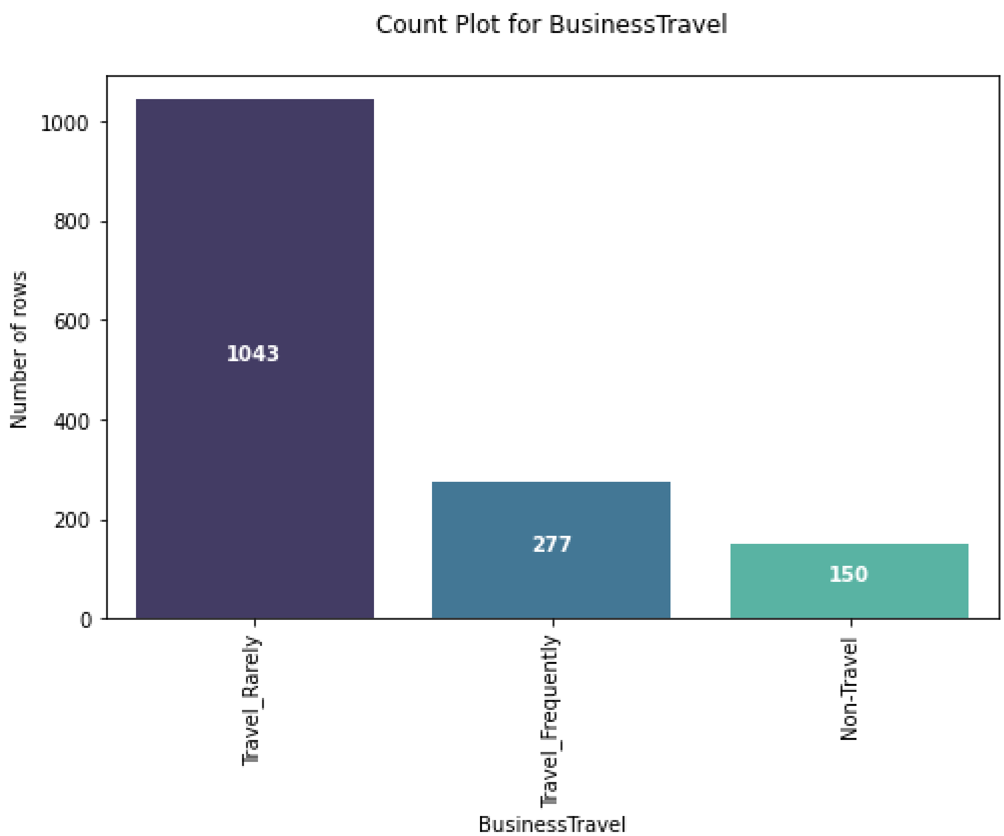


here we can see the count of Male is more than that of female.

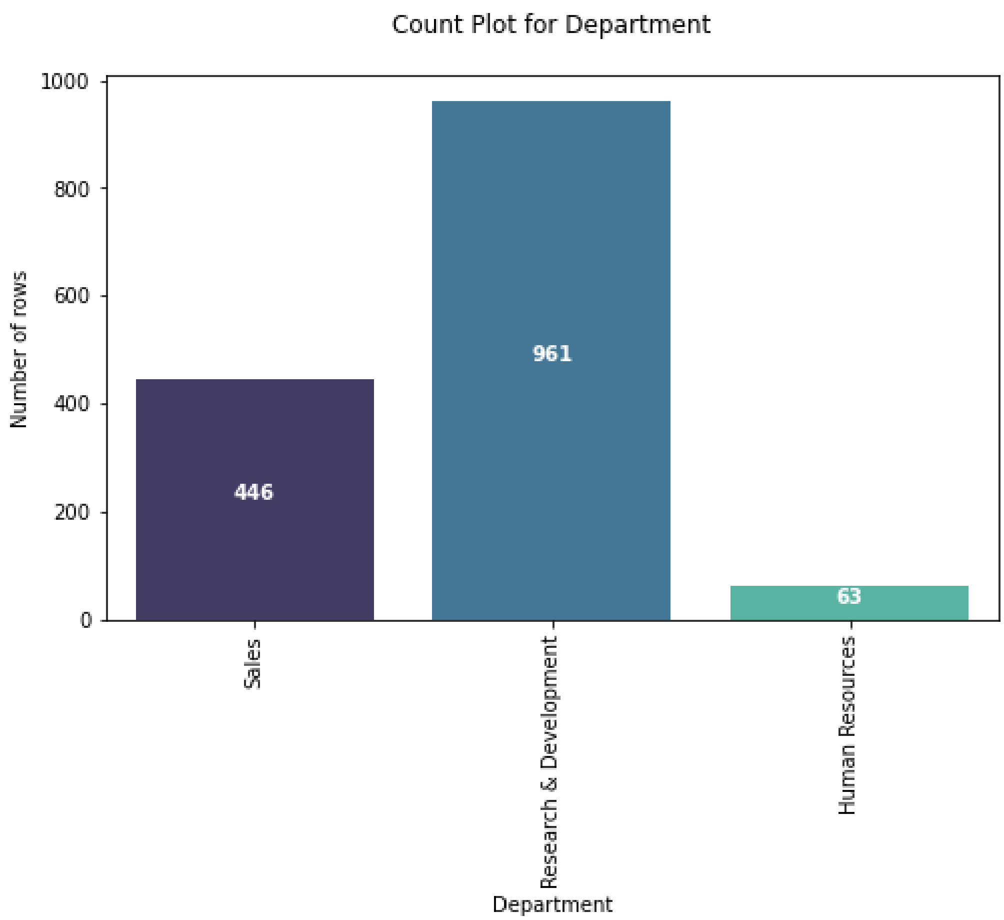
In [14]:
```
plt.figure(figsize=(8,5))
col_name = 'OverTime'
values = df[col_name].value_counts()
index = 0
ax = sns.countplot(df[col_name], palette="mako")

for i in ax.get_xticklabels():
    ax.text(index, values[i.get_text()]/2, values[i.get_text()],
            horizontalalignment="center", fontweight='bold', color='w')
    index += 1

plt.title(f"Count Plot for {col_name}\n")
plt.ylabel(f"Number of rows")
plt.xticks(rotation=90)
plt.show()
```



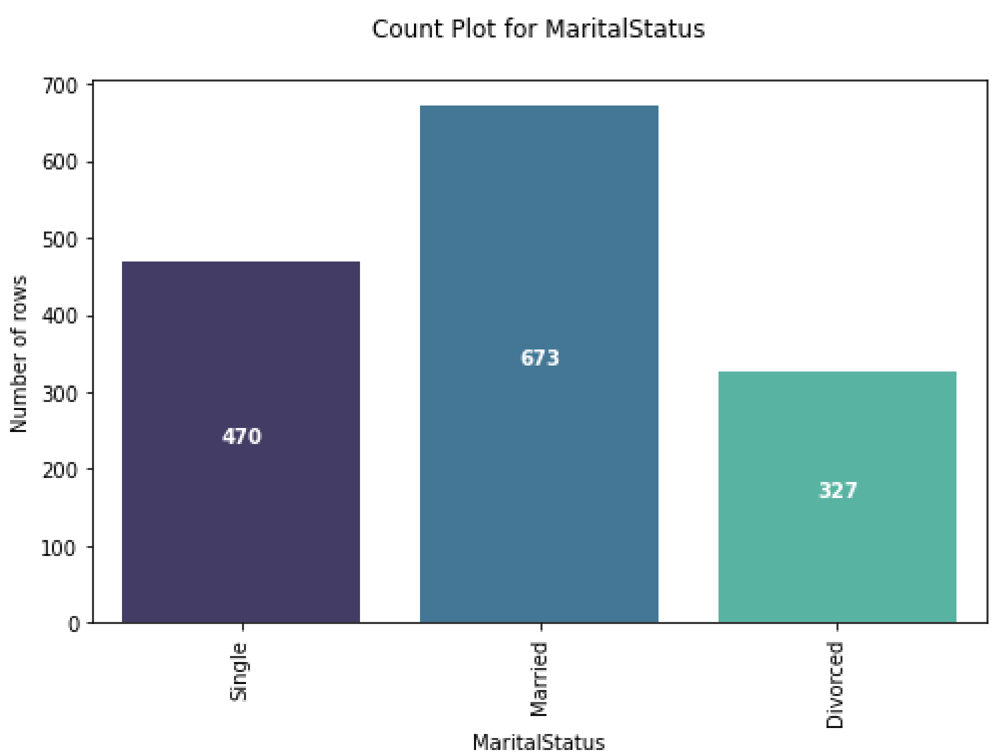
In [15]:
```
plt.figure(figsize=(8,5))
col_name = 'BusinessTravel'
values = df[col_name].value_counts()
index = 0
ax = sns.countplot(df[col_name], palette="mako")

for i in ax.get_xticklabels():
    ax.text(index, values[i.get_text()]/2, values[i.get_text()],
            horizontalalignment="center", fontweight='bold', color='w')
    index += 1

plt.title(f"Count Plot for {col_name}\n")
plt.ylabel(f"Number of rows")
plt.xticks(rotation=90)
plt.show()
```



In [16]:
```
plt.figure(figsize=(8,5))
col_name = 'Department'
values = df[col_name].value_counts()
index = 0
ax = sns.countplot(df[col_name], palette="mako")

for i in ax.get_xticklabels():
    ax.text(index, values[i.get_text()]/2, values[i.get_text()],
            horizontalalignment="center", fontweight='bold', color='w')
    index += 1

plt.title(f"Count Plot for {col_name}\n")
plt.ylabel(f"Number of rows")
plt.xticks(rotation=90)
plt.show()
```

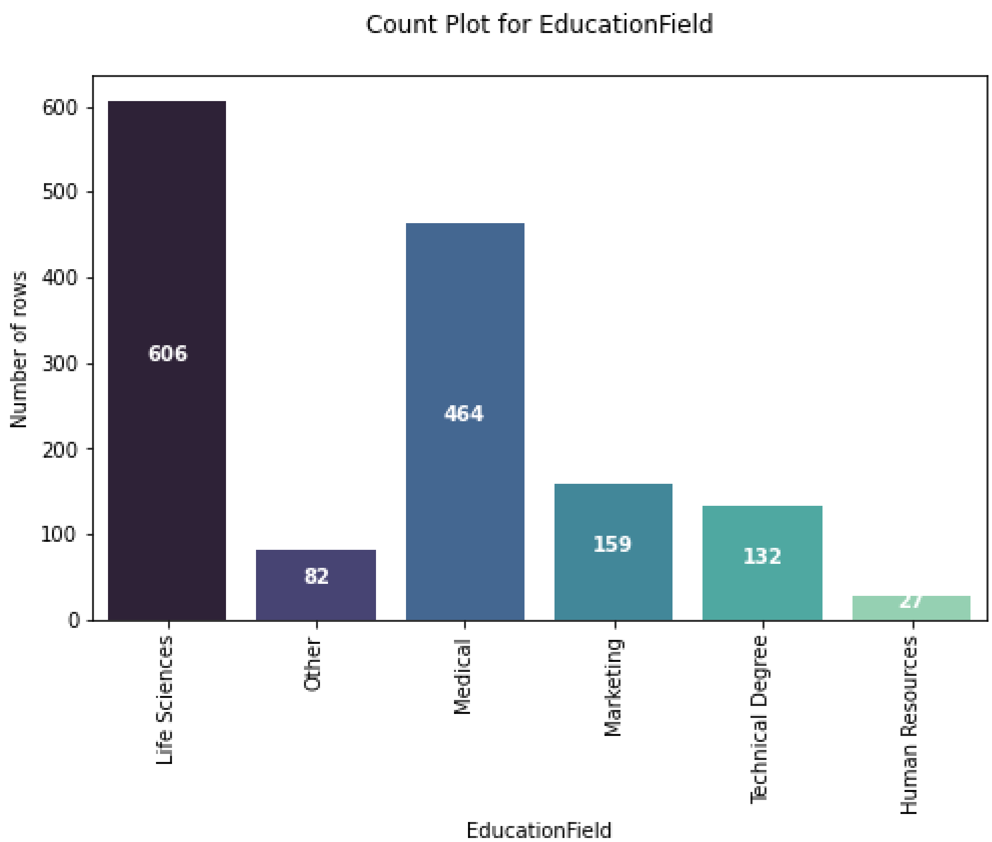


In [17]:
```
plt.figure(figsize=(8,5))
col_name = 'MaritalStatus'
values = df[col_name].value_counts()
index = 0
ax = sns.countplot(df[col_name], palette="mako")

for i in ax.get_xticklabels():
    ax.text(index, values[i.get_text()]/2, values[i.get_text()],
            horizontalalignment="center", fontweight='bold', color='w')
    index += 1

plt.title(f"Count Plot for {col_name}\n")
plt.ylabel(f"Number of rows")
plt.xticks(rotation=90)
plt.show()
```

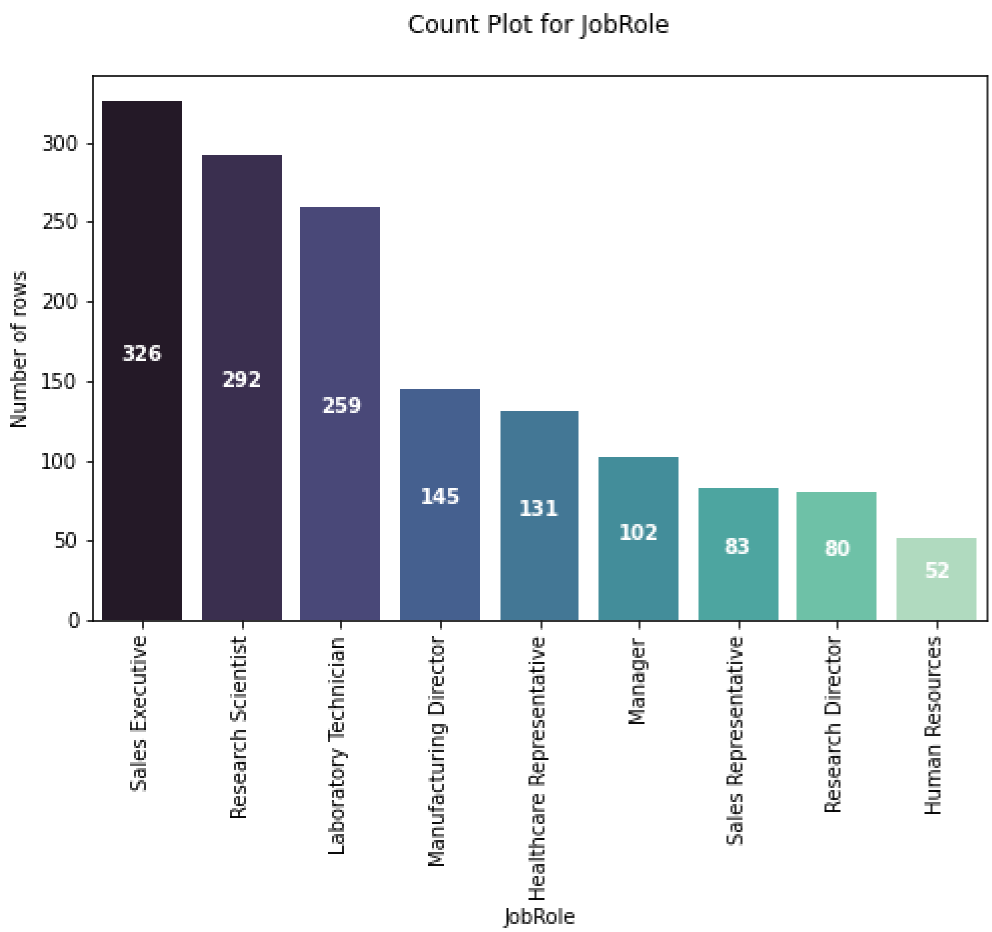


In [18]:

```
plt.figure(figsize=(8,5))
col_name = 'EducationField'
values = df[col_name].value_counts()
index = 0
ax = sns.countplot(df[col_name], palette="mako")

for i in ax.get_xticklabels():
    ax.text(index, values[i.get_text()]/2, values[i.get_text()],
            horizontalalignment="center", fontweight='bold', color='w')
    index += 1

plt.title(f"Count Plot for {col_name}\n")
plt.ylabel(f"Number of rows")
plt.xticks(rotation=90)
plt.show()
```

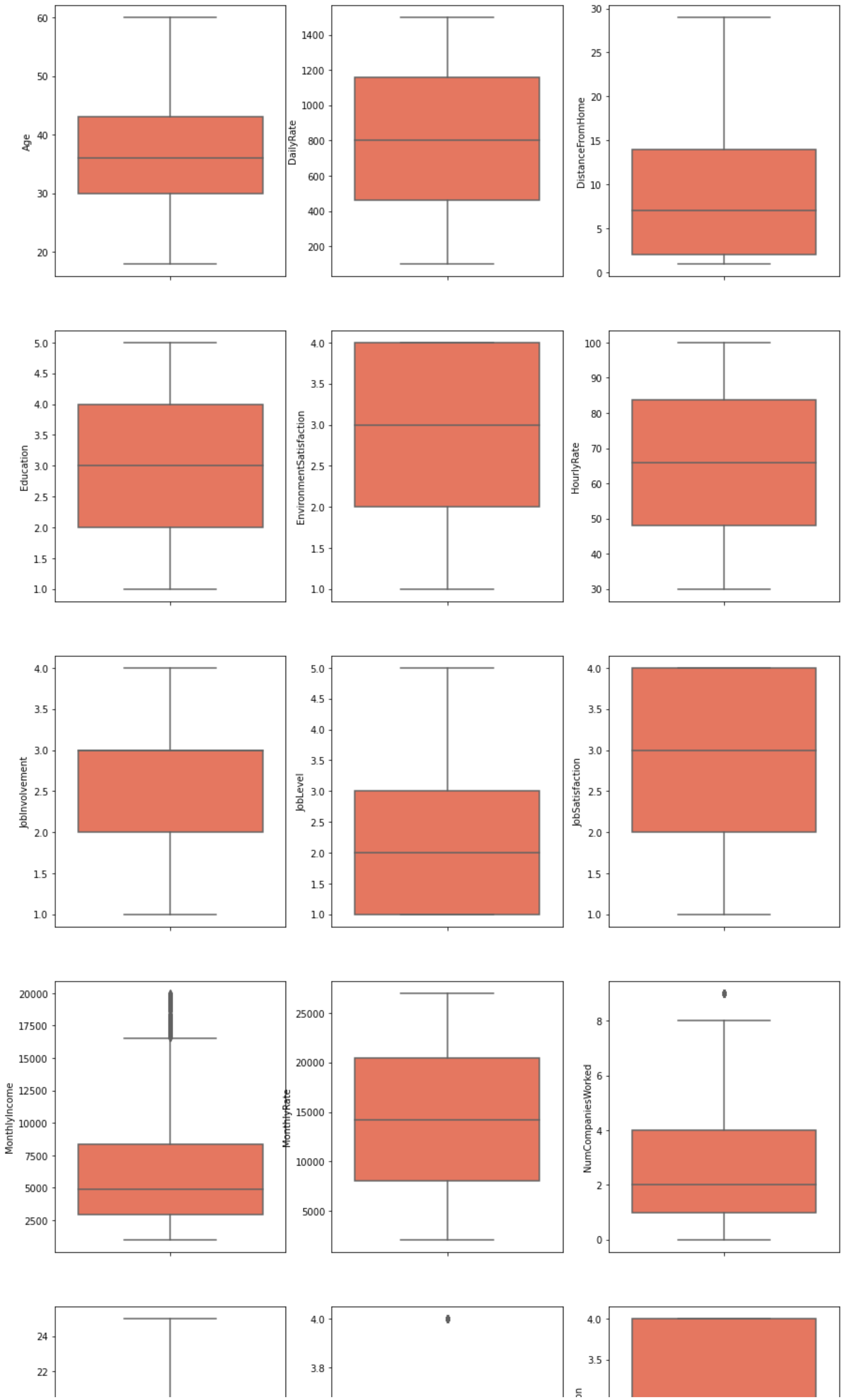


In [19]:
```
plt.figure(figsize=(8,5))
col_name = 'JobRole'
values = df[col_name].value_counts()
index = 0
ax = sns.countplot(df[col_name], palette="mako")

for i in ax.get_xticklabels():
    ax.text(index, values[i.get_text()]/2, values[i.get_text()],
            horizontalalignment="center", fontweight='bold', color='w')
    index += 1

plt.title(f"Count Plot for {col_name}\n")
plt.ylabel(f"Number of rows")
plt.xticks(rotation=90)
plt.show()
```

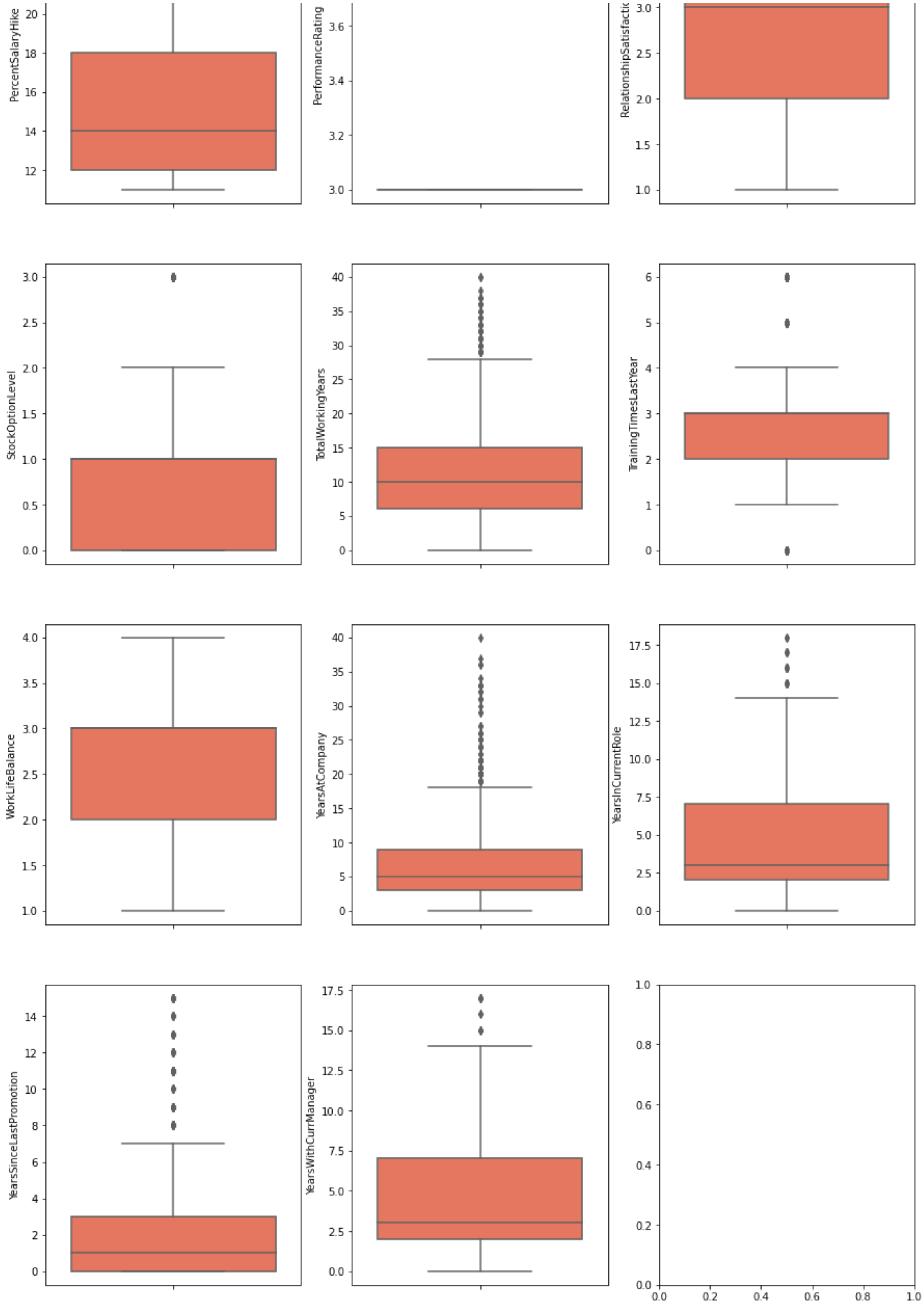


In [20]:

```
fig, ax = plt.subplots(ncols=3, nrows=8, figsize=(15,50))
index = 0
ax = ax.flatten()
for col, value in df[integer_datatypes].items():
    sns.boxplot(y=col, data=df, ax=ax[index], palette="Reds")
    index += 1
plt.show()
```

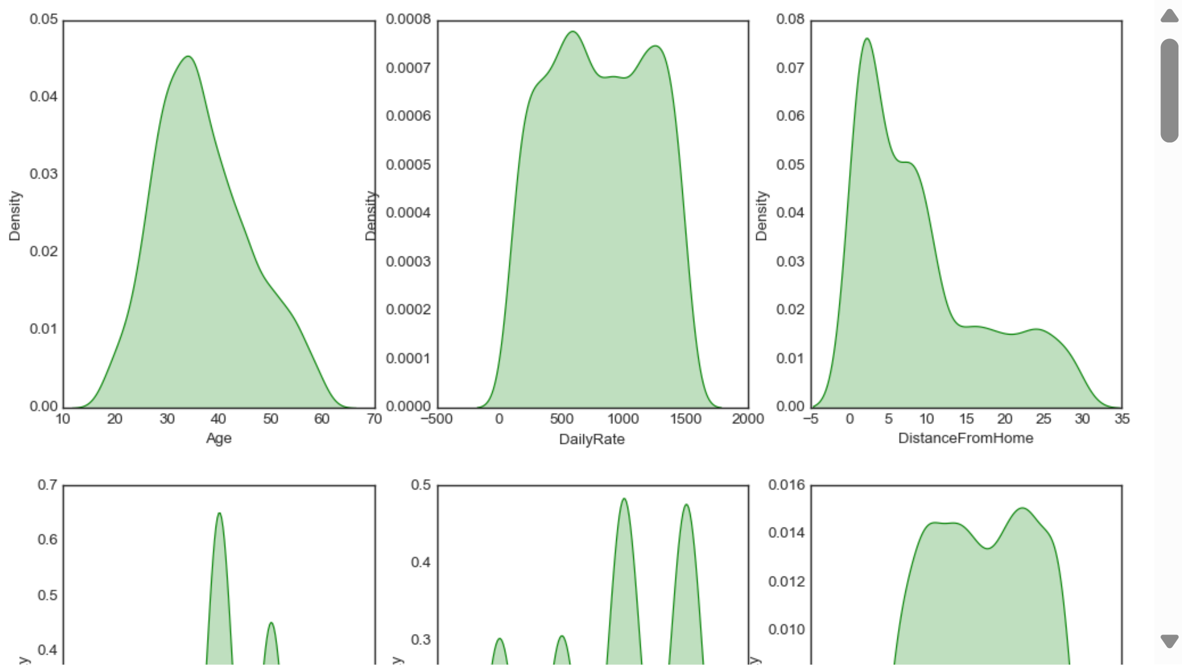


Age
DailyRate
DistanceFromHome
Education
EnvironmentSatisfaction
HourlyRate
JobInvolvement
JobLevel
JobSatisfaction
MonthlyIncome
MonthlyRate
NumCompaniesWorked



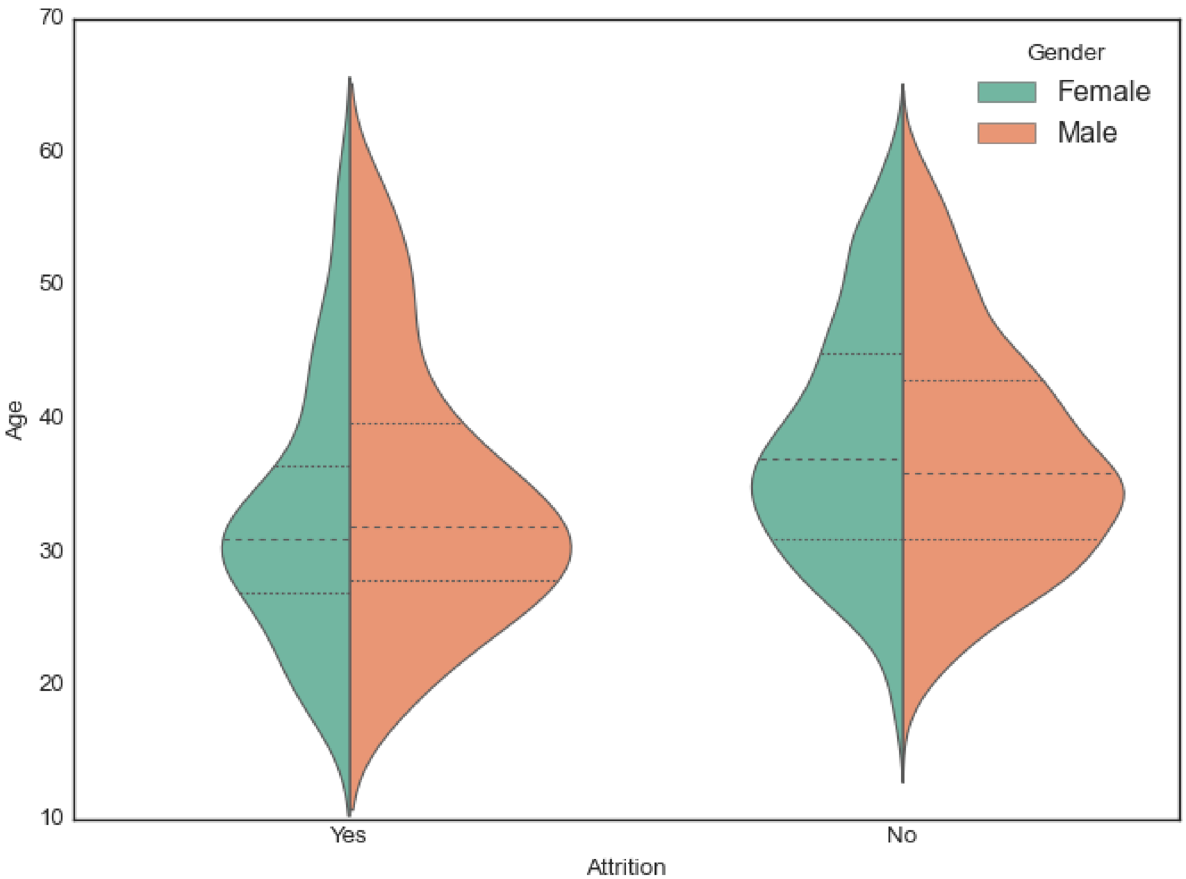
From the above boxplot we have many columns having outliers they need to be treated well

In [26]:
```
fig, ax = plt.subplots(ncols=3, nrows=8, figsize=(15,50))
index = 0
ax = ax.flatten()
for col, value in df[integer_datatypes].items():
    sns.distplot(value, ax=ax[index], hist=False, color="g", kde_kws={"shade":
    index += 1
plt.show()
```



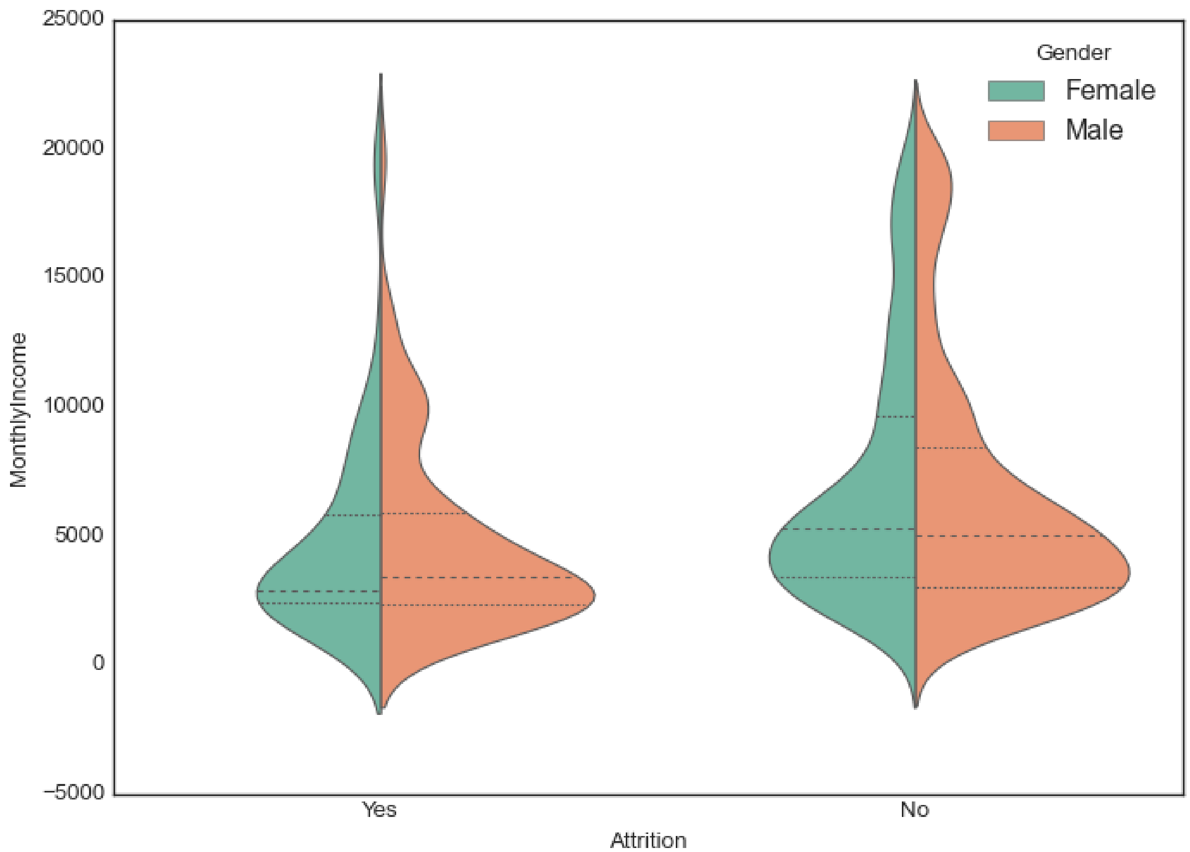
from the above distplot method we found that there is some skewness also present and they have to be treated before model selection.

In [27]:
```
plt.figure(figsize=(10,7))
sns.violinplot(x="Attrition", y="Age", hue="Gender", data=df,
               palette="Set2", split=True, scale="count", inner="quartile")
plt.show()
```



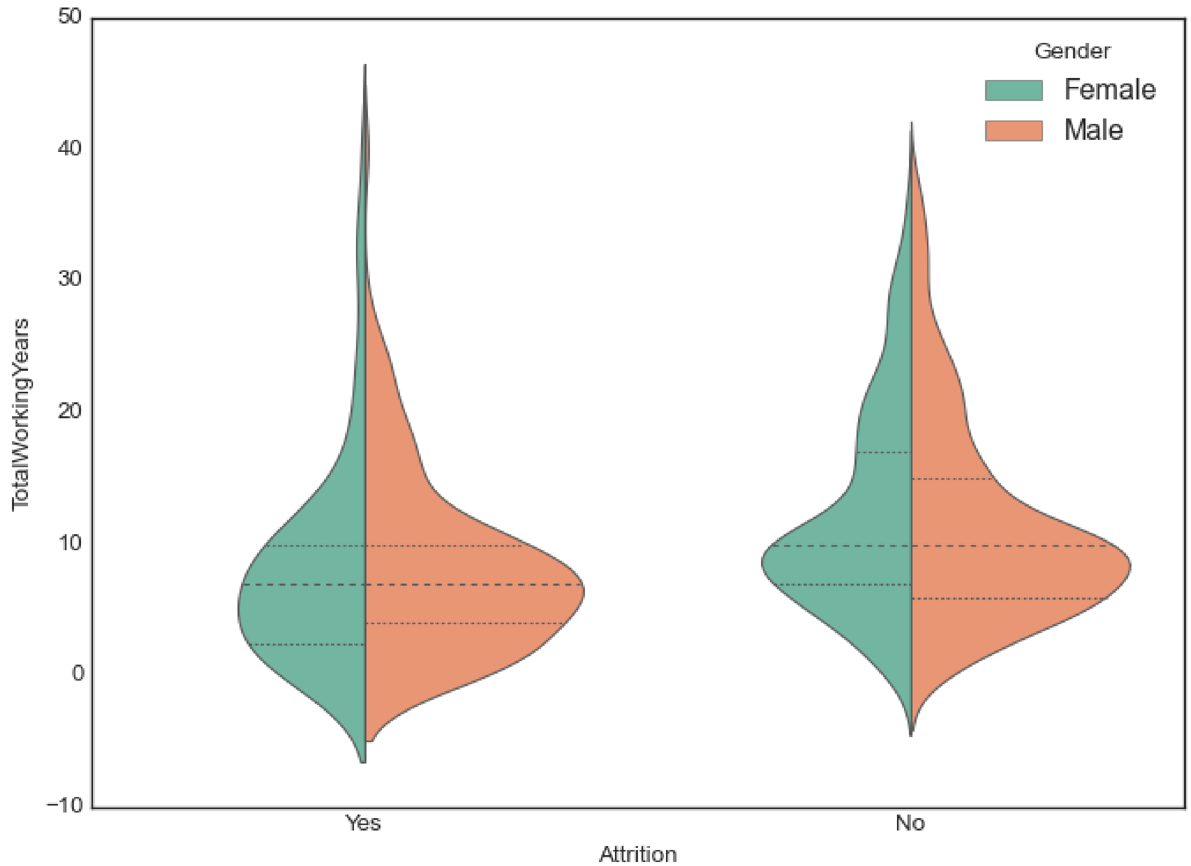
from above we can say that both male and female has wide range in age between 20 and 30

In [28]:
```
plt.figure(figsize=(10,7))
sns.violinplot(x="Attrition", y="MonthlyIncome", hue="Gender", data=df,
               palette="Set2", split=True, scale="count", inner="quartile")
plt.show()
```



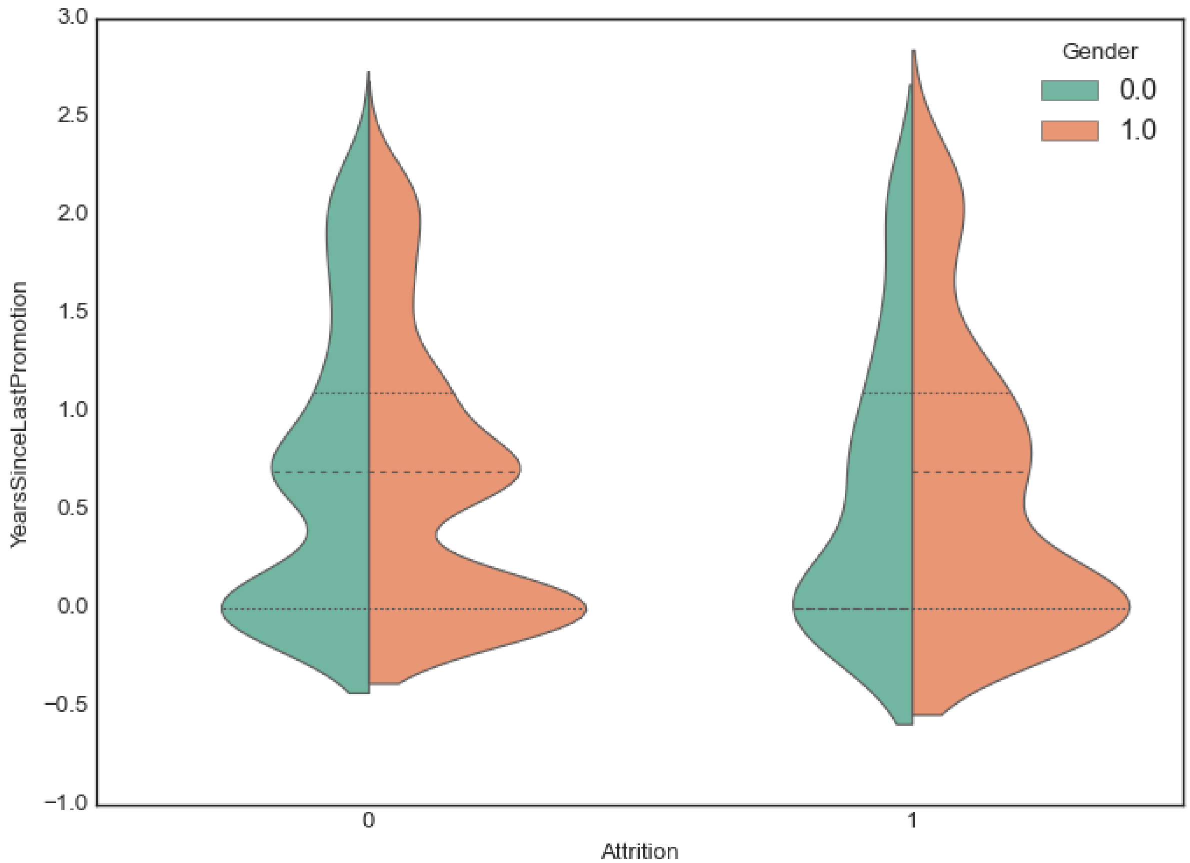
from above we have observation that most of male and female are having Monthlyincome less than 5000

In [29]:
```
plt.figure(figsize=(10,7))
sns.violinplot(x="Attrition", y="TotalWorkingYears", hue="Gender", data=df,
               palette="Set2", split=True, scale="count", inner="quartile")
plt.show()
```



from the above observation male and female mostly have Totlaworkexperienve in between range 0 to 10 years

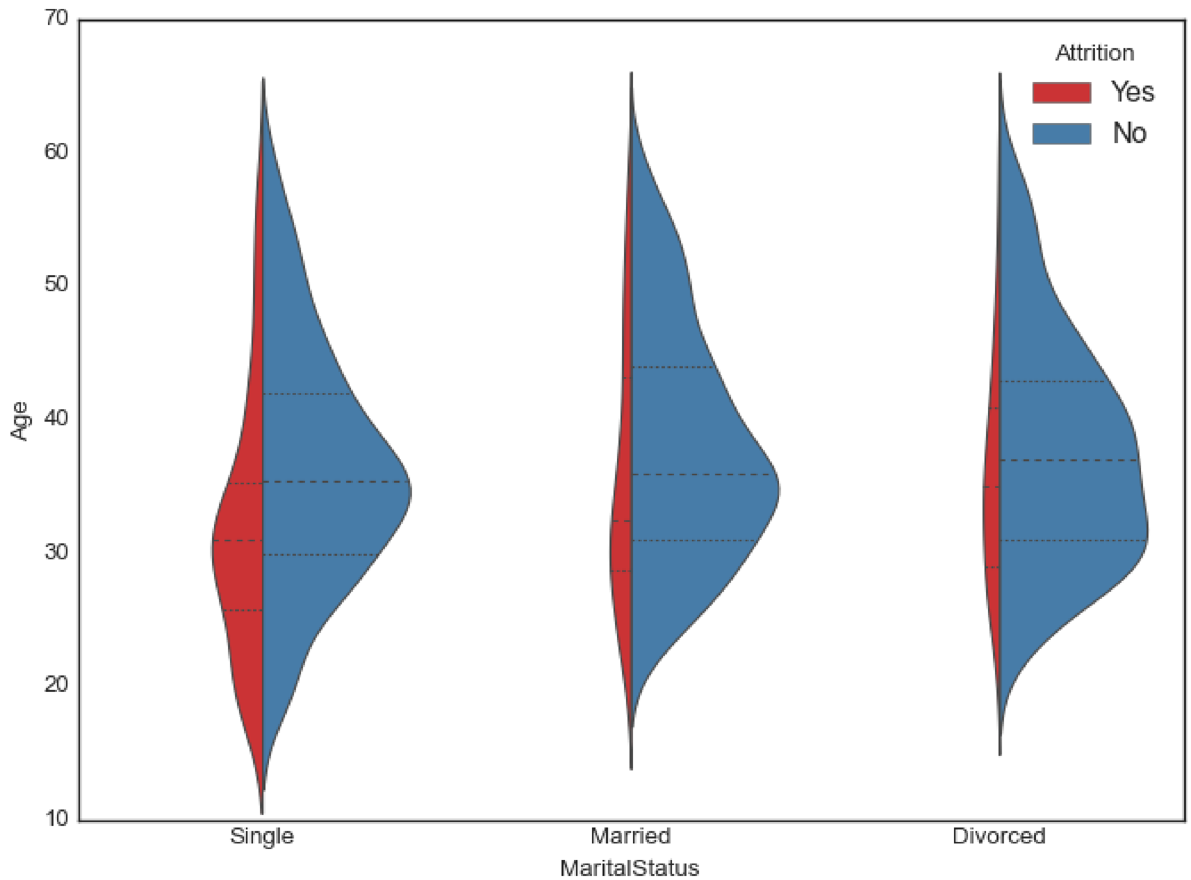
In [93]:
```
plt.figure(figsize=(10,7))
sns.violinplot(x="Attrition", y="YearsSinceLastPromotion", hue="Gender", data=d
               palette="Set2", split=True, scale="count", inner="quartile")
plt.show()
```



from above plot we have that for both male and female mostly are less than 5 years since they have got promoted

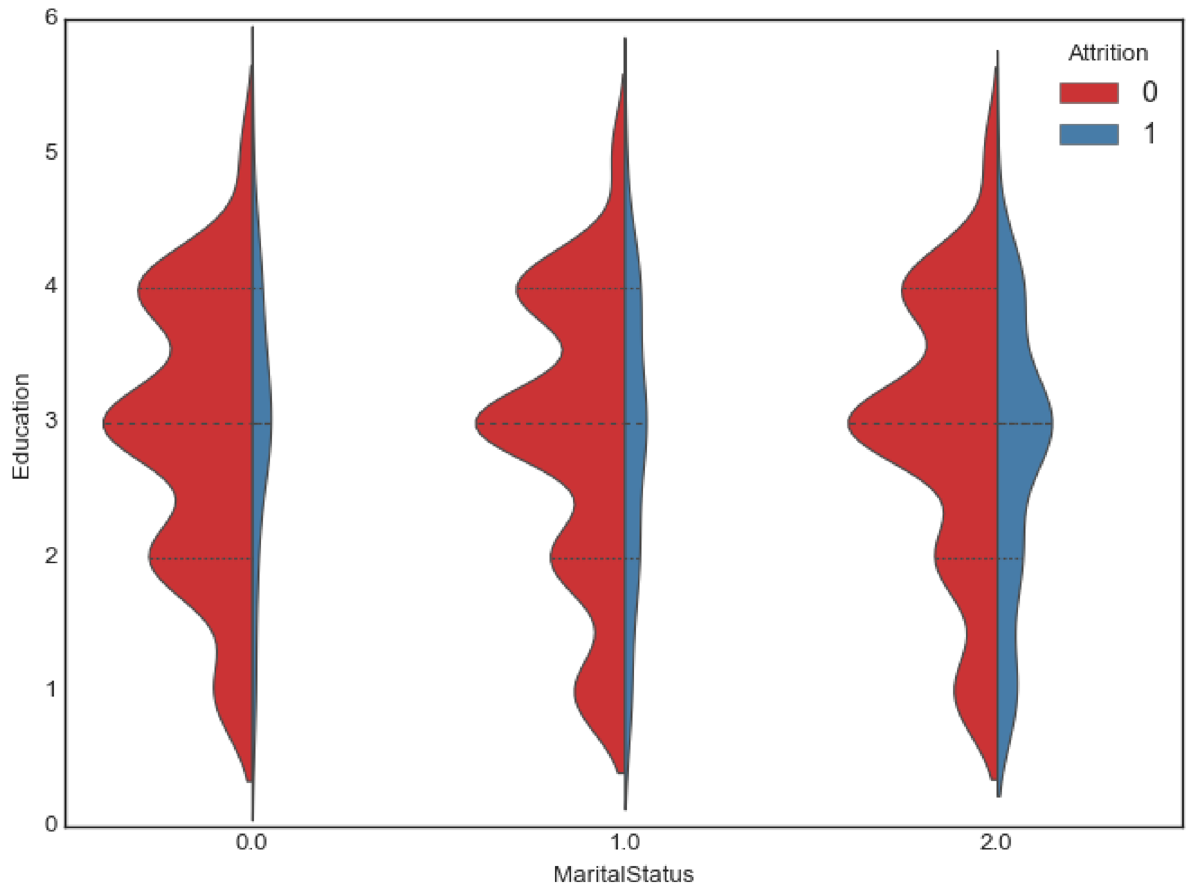
In [31]:

```
plt.figure(figsize=(10,7))
sns.violinplot(x="MaritalStatus", y="Age", hue="Attrition", data=df,
               palette="Set1", split=True, scale="count", inner="quartile")
plt.show()
```



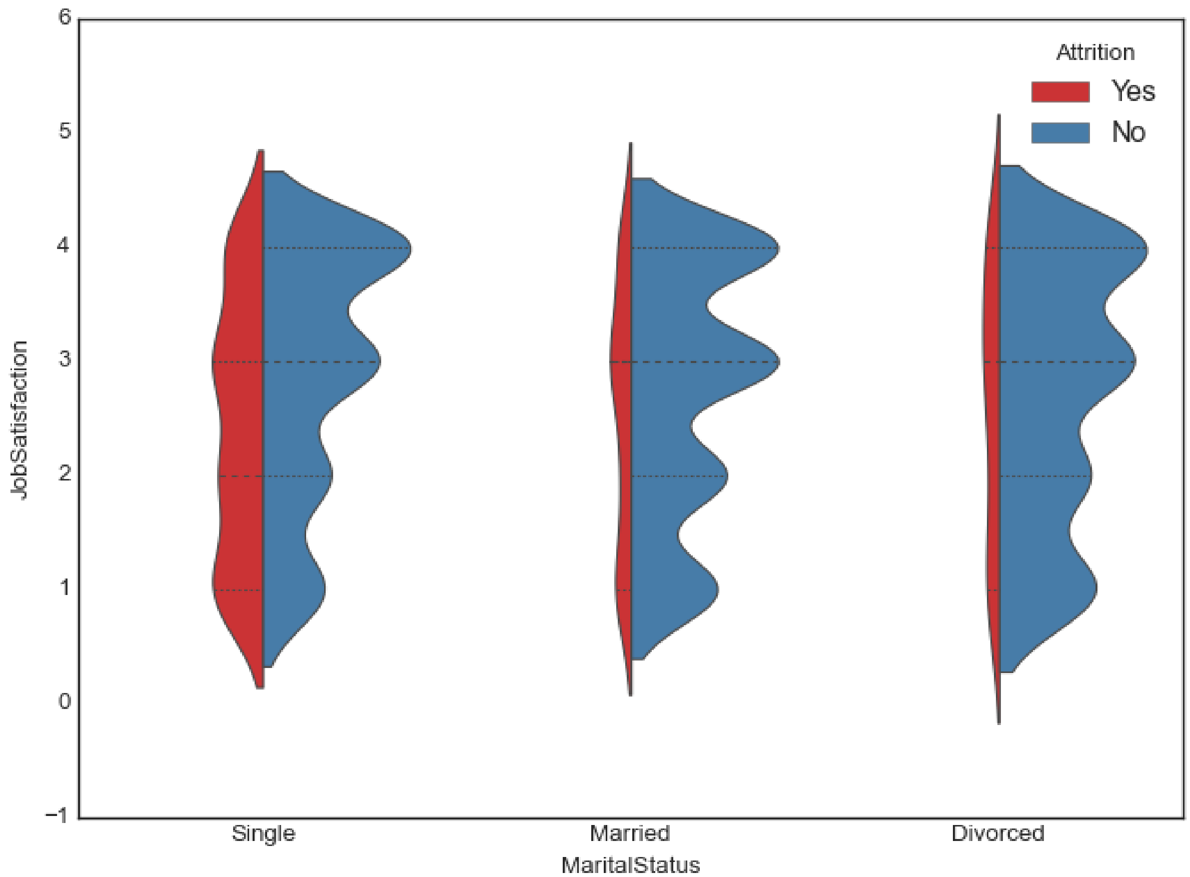
in this plot we can see that there is some Attrition when you are single and less when you are married but little when you are divorced.

In [94]:
```
plt.figure(figsize=(10,7))
sns.violinplot(x="MaritalStatus", y="Education", hue="Attrition", data=df,
               palette="Set1", split=True, scale="count", inner="quartile")
plt.show()
```



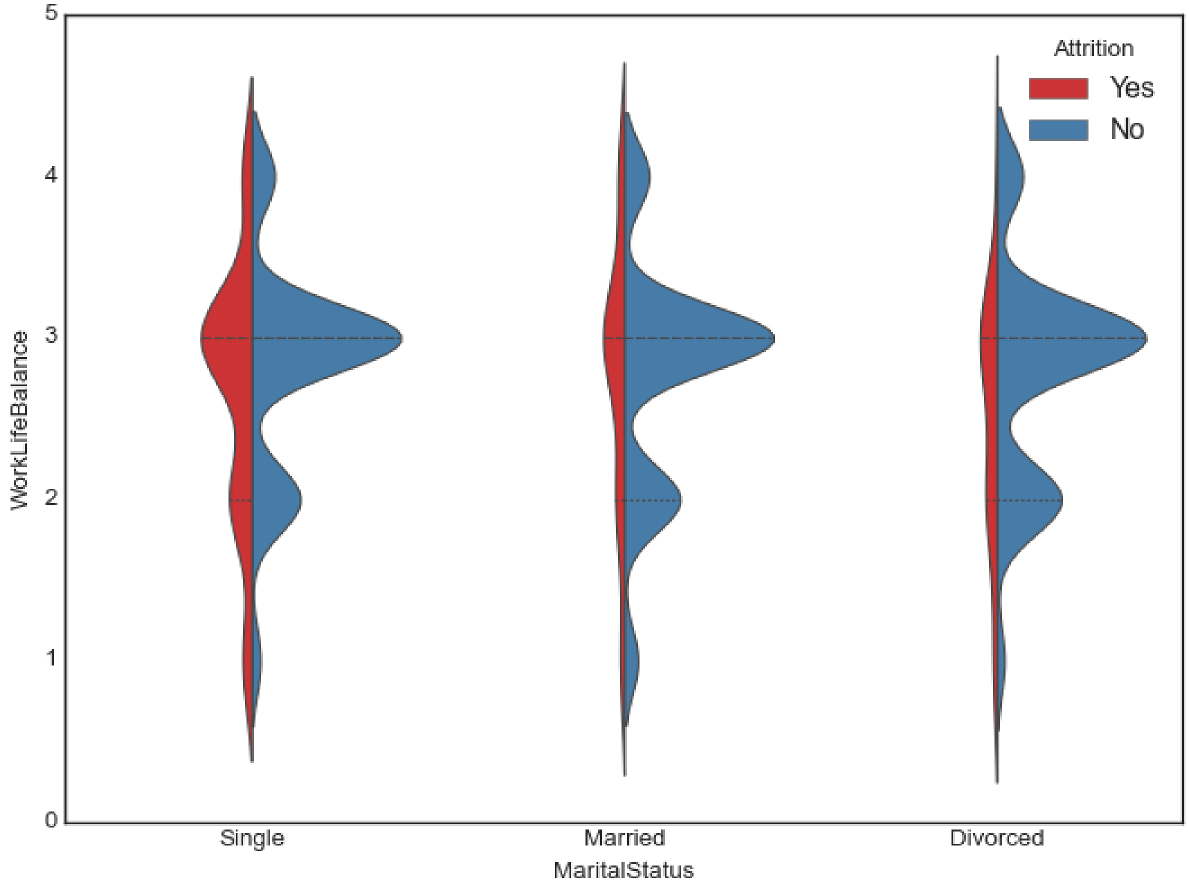
Type *Markdown* and LaTeX: α^2

In [33]:
```
plt.figure(figsize=(10,7))
sns.violinplot(x="MaritalStatus", y="JobSatisfaction", hue="Attrition", data=df
               palette="Set1", split=True, scale="count", inner="quartile")
plt.show()
```



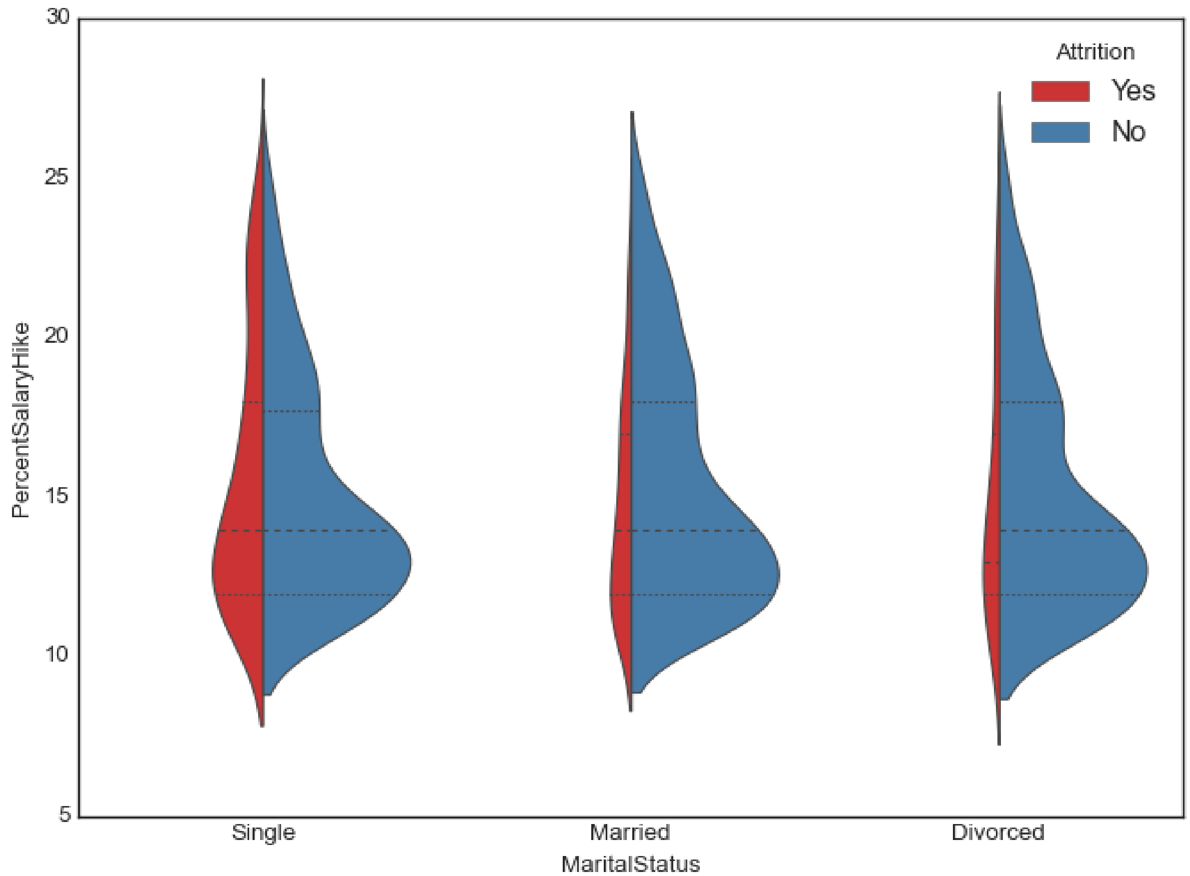
above plot showing single are more satisfied with their job other than they are married or divorced.

In [34]:
```
plt.figure(figsize=(10,7))
sns.violinplot(x="MaritalStatus", y="WorkLifeBalance", hue="Attrition", data=df
               palette="Set1", split=True, scale="count", inner="quartile")
plt.show()
```



from the above graph we can see that worklife of single is much balanced than married or divorced.

In [35]:
```
plt.figure(figsize=(10,7))
sns.violinplot(x="MaritalStatus", y="PercentSalaryHike", hue="Attrition", data=
               palette="Set1", split=True, scale="count", inner="quartile")
plt.show()
```



from the above figure we can say that hike in salary of single is higher than married or some one who got divorced. maybe,perhaps a single have less burden so there work is more effectiove so they get more hike in their salary.

Encoding the categorical object datatype columns

In [36]:
```
le=LabelEncoder()
df['Attrition']=le.fit_transform(df['Attrition'])
```

In [37]:
```
df['Attrition']
```

Out[37]:
```
0       1
1       0
2       1
3       0
4       0
       ..
1465    0
1466    0
1467    0
1468    0
1469    0
Name: Attrition, Length: 1470, dtype: int32
```

i have used Label Encoder to Attrition that is our target column as it was object datatype we converted into integer64 type.

In [38]:
```
# Ordinal Encoder

oe = OrdinalEncoder()
df['BusinessTravel'] = oe.fit_transform(df['BusinessTravel'].values.reshape(-1,
df['Department'] = oe.fit_transform(df['Department'].values.reshape(-1,1))
df['EducationField'] = oe.fit_transform(df['EducationField'].values.reshape(-1,
df['Gender'] = oe.fit_transform(df['Gender'].values.reshape(-1,1))
df['JobRole'] = oe.fit_transform(df['JobRole'].values.reshape(-1,1))
df['MaritalStatus'] = oe.fit_transform(df['MaritalStatus'].values.reshape(-1,1)
df['OverTime'] = oe.fit_transform(df['OverTime'].values.reshape(-1,1))
```

i have used Ordinal Encoder in order to convert our some columns from object datatype to integer datatype.

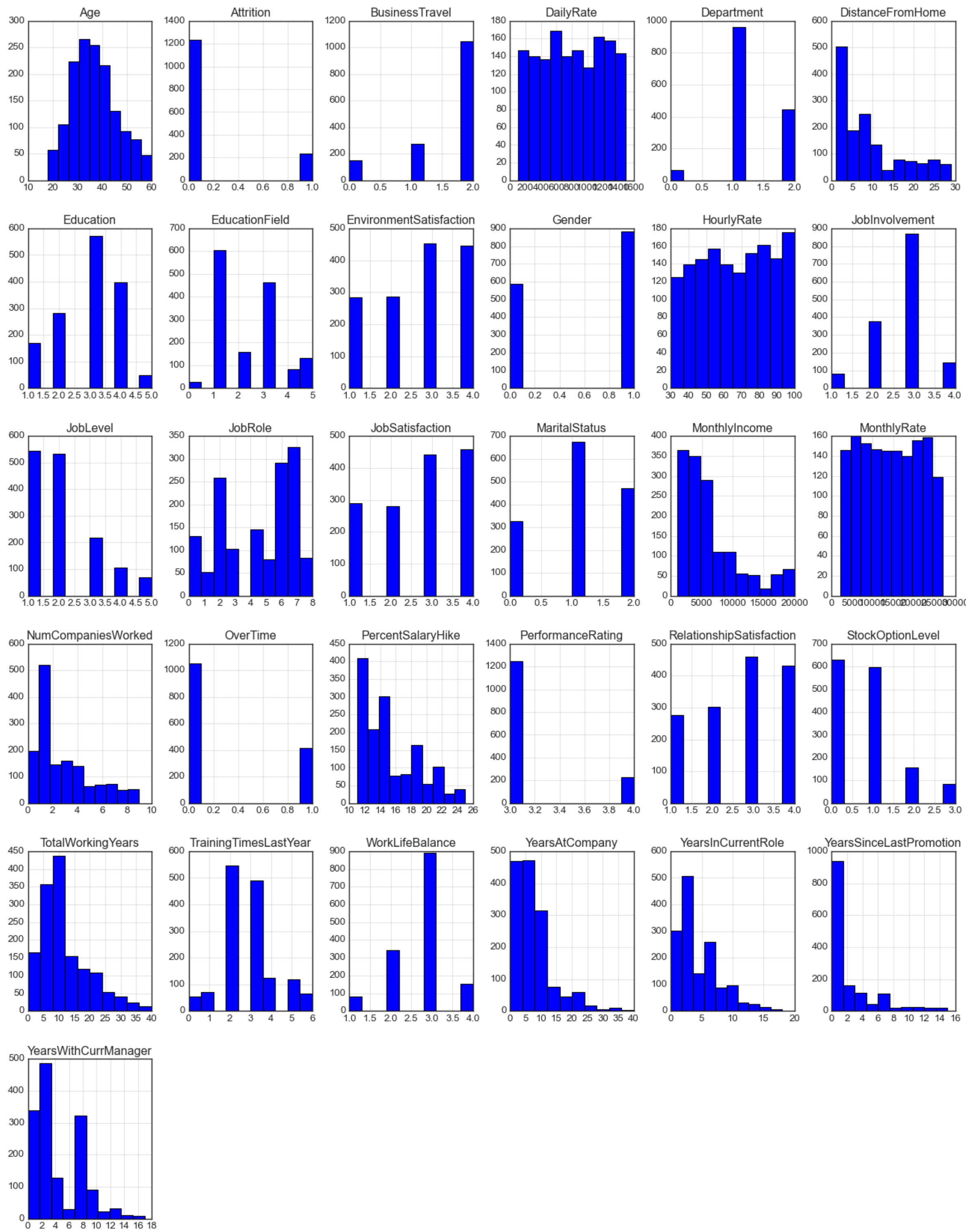
In [39]:
```
df.head()
```

Out[39]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	Educatic
0	41	1	2.0	1102	2.0	1	2	
1	49	0	1.0	279	1.0	8	1	
2	37	1	2.0	1373	1.0	2	2	
3	33	0	1.0	1392	1.0	3	4	
4	27	0	2.0	591	1.0	2	1	

5 rows × 31 columns

```
In [41]: df.hist(figsize=(20,25))
         plt.show()
```

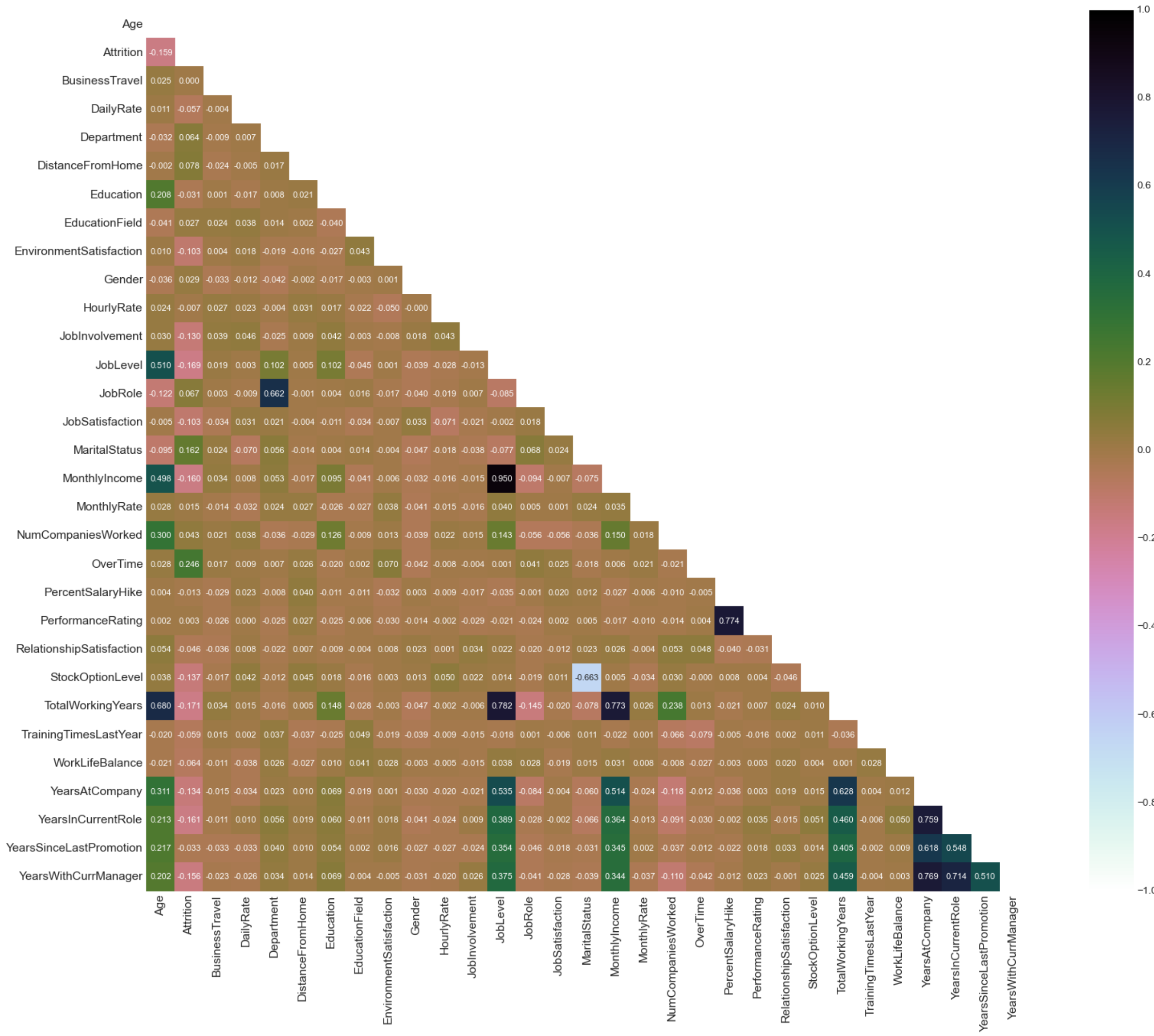


Using the above histogram we are able to plot all the columns of are dataset post application of encoding technique where we do not have any object datatype columns anymore.

Correlation using a Heatmap

Positive correlation - A correlation of +1 indicates a perfect positive correlation, meaning that both variables move in the same direction together. Negative correlation - A correlation of –1 indicates a perfect negative correlation, meaning that as one variable goes up, the other goes down

In [42]:
```python
upper_triangle = np.triu(df.corr())
plt.figure(figsize=(26,18))
sns.heatmap(df.corr(), vmin=-1, vmax=1, annot=True, square=True, fmt='0.3f',
            annot_kws={'size':10}, cmap="cubehelix_r", mask=upper_triangle)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```

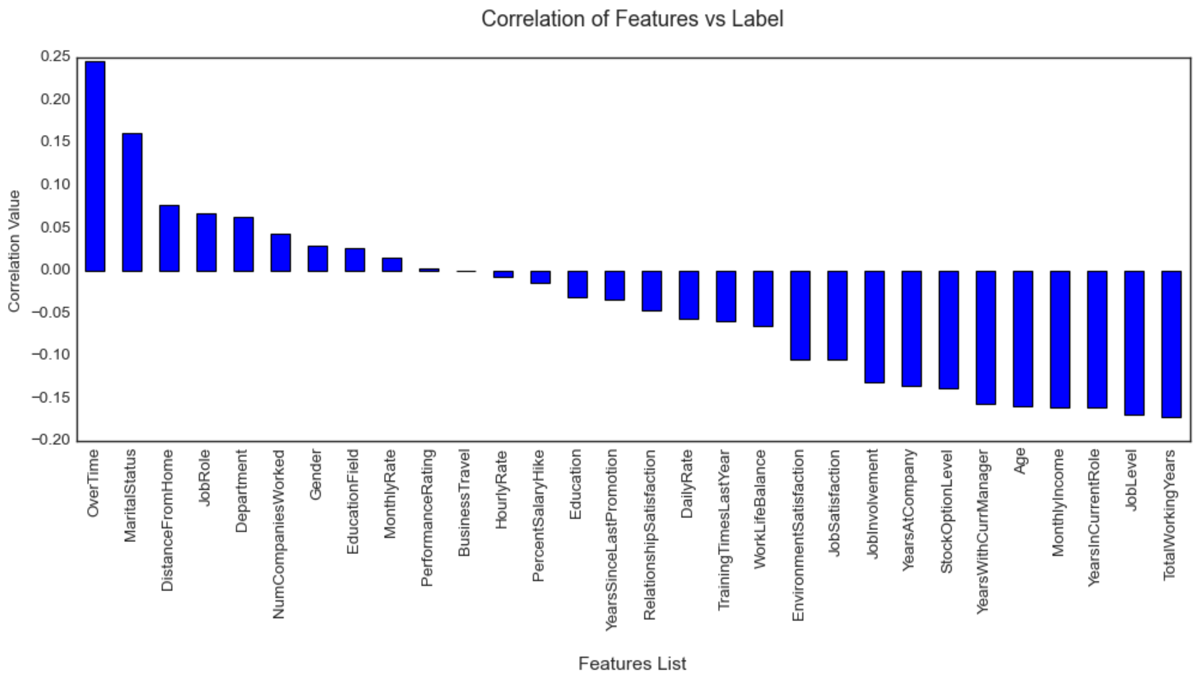


In the above heatmap we can see that our target label "Attrition" has both positive and negative correlations with the feature columns. Also we see very less or negligible amount of multi colinearity so we will not have to worry about it. Since the one's which are reflecting the value are inter dependent on those feature columns and I intend to retain and keep them.

Correlation Bar Plot comparing features with our label

In [43]:

```
df_corr = df.corr()
plt.figure(figsize=(15,5))
df_corr['Attrition'].sort_values(ascending=False).drop('Attrition').plot.bar()
plt.title("Correlation of Features vs Label\n", fontsize=16)
plt.xlabel("\nFeatures List", fontsize=14)
plt.ylabel("Correlation Value", fontsize=12)
plt.show()
```



In the above Bar Plot we are able to clearly define the feature columns that are positively correlated with our label and the feature columns that are negatively correlated with our label.

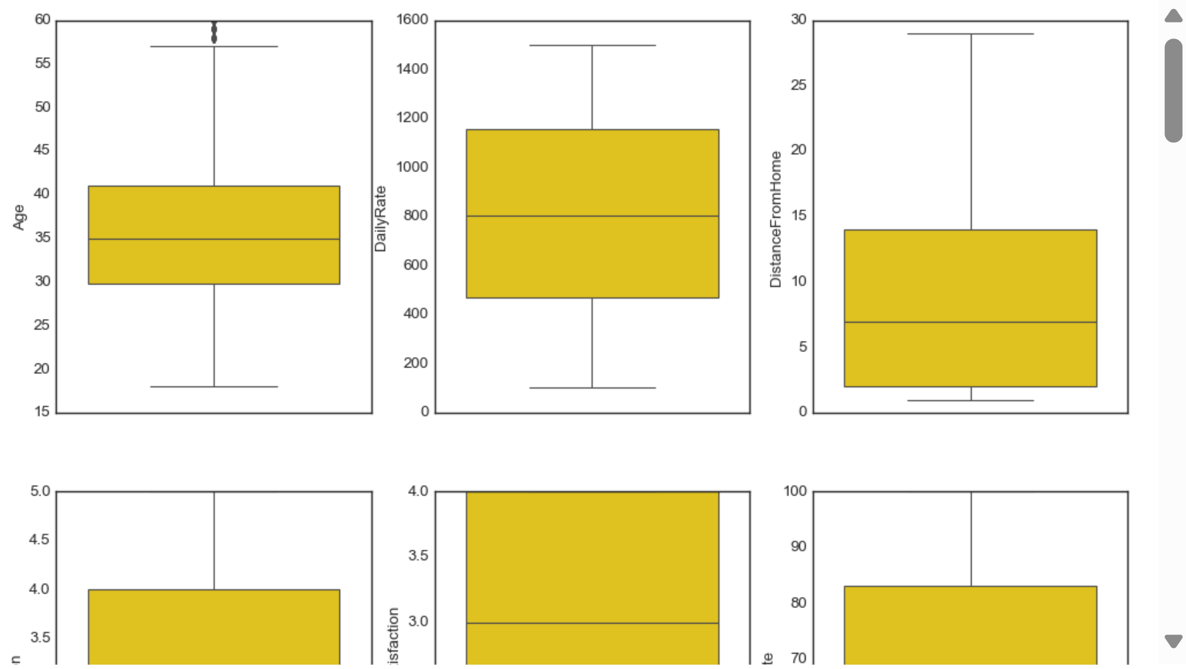
Using Z Score to remove outliers¶

In [45]:

```
z = np.abs(zscore(df))
threshold = 3
df1 = df[(z<3).all(axis = 1)]

print ("Shape of the dataframe before removing outliers: ", df.shape)
print ("Shape of the dataframe after removing outliers: ", df1.shape)
print ("Percentage of data loss post outlier removal: ", (df.shape[0]-df1.shape

df=df1.copy() # reassigning the changed dataframe name to our original datafram
```

```
Shape of the dataframe before removing outliers:  (1387, 31)
Shape of the dataframe after removing outliers:  (1304, 31)
Percentage of data loss post outlier removal:  5.984138428262437
```

In [47]:
```
fig, ax = plt.subplots(ncols=3, nrows=8, figsize=(15,50))
index = 0
ax = ax.flatten()
for col, value in df[integer_datatypes].items():
    sns.boxplot(y=col, data=df, ax=ax[index], palette="prism")
    index += 1
plt.show()
```



In the above box plot we can see that whatever outliers we could have afforded to lose from our numerical columns we have gotten rid of it. There are still presence of outliers but since they are in continous format we shall ignore it

In [48]:
```
df.skew()
```

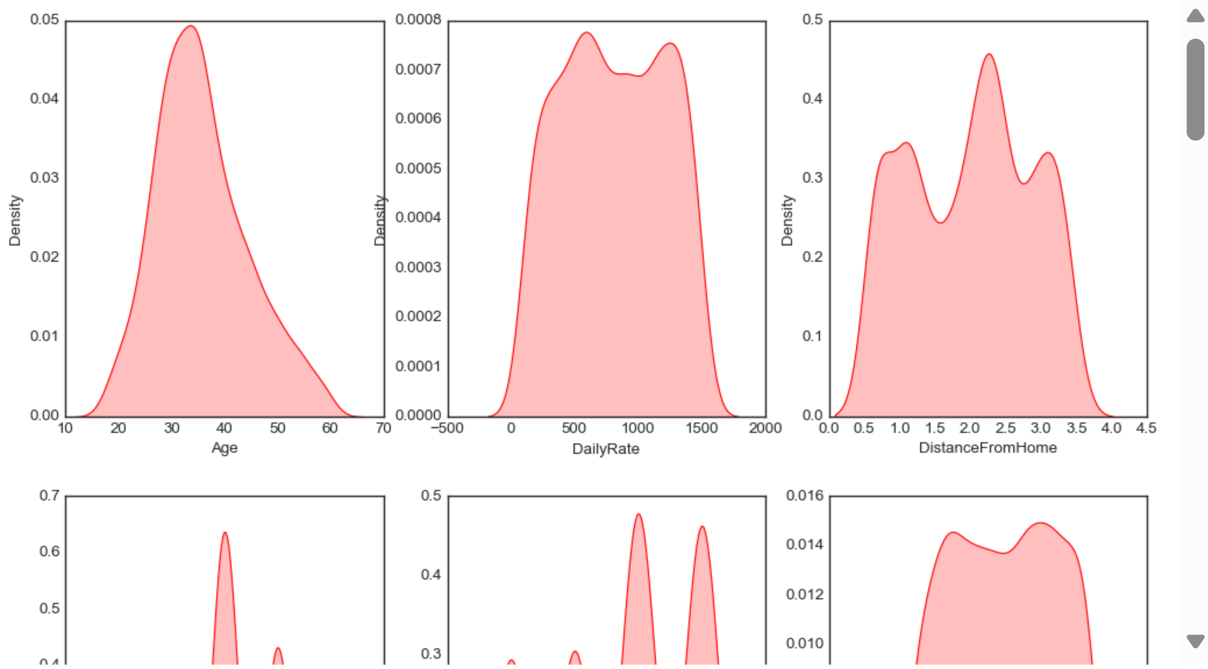
Out[48]:
```
Age                         0.526943
Attrition                   1.749533
BusinessTravel             -1.414592
DailyRate                  -0.014199
Department                  0.177341
DistanceFromHome            0.935391
Education                  -0.279545
EducationField              0.551717
EnvironmentSatisfaction    -0.319471
Gender                     -0.427052
HourlyRate                 -0.032833
JobInvolvement             -0.496090
JobLevel                    0.940555
JobRole                    -0.415162
JobSatisfaction            -0.335845
MaritalStatus              -0.173388
MonthlyIncome               1.531140
MonthlyRate                 0.039330
NumCompaniesWorked          1.046568
OverTime                    0.960516
PercentSalaryHike           0.796258
PerformanceRating           1.942566
RelationshipSatisfaction   -0.275672
StockOptionLevel            0.957507
TotalWorkingYears           0.980416
TrainingTimesLastYear       0.584609
WorkLifeBalance            -0.545492
YearsAtCompany              0.949117
YearsInCurrentRole          0.668397
YearsSinceLastPromotion     1.659455
YearsWithCurrManager        0.717479
dtype: float64
```

Using Log Transform to fix skewness

In [50]:
```
for col in integer_datatypes:
    if df.skew().loc[col]>0.55:
        df[col]=np.log1p(df[col])
```

I have applied Log Transformation on our numerical integer datatype columns to ensure that we do not have skewness in our dataset.

```
In [52]: fig, ax = plt.subplots(ncols=3, nrows=8, figsize=(15,50))
index = 0
ax = ax.flatten()
for col, value in df[integer_datatypes].items():
    sns.distplot(value, ax=ax[index], hist=False, color="r", kde_kws={"shade":
    index += 1
plt.show()
```



Splitting the dataset into 2 variables namely 'X' and 'Y' for feature and label

```
In [53]: X = df.drop('Attrition', axis=1)
Y = df['Attrition']
```

I have bifurcated the dataset into features and labels where X represents all the feature columns and Y represents the target label column

Resolving the class imbalance issue in label column¶

```
In [55]: Y.value_counts()

Out[55]: 0    1081
1     223
Name: Attrition, dtype: int64
```

Listing the values of our label column to count the number of rows occupied by each category. This indicates class imbalance that we will need to fix by using the oversampling method.

In [56]:
```
# adding samples to make all the categorical quality values same

oversample = SMOTE()
X, Y = oversample.fit_resample(X, Y)
```

SMOTE is the over sampling mechanism that we are using to ensure that all the categories present in our target label have the same value.

In [57]:
```
Y.value_counts()
```

Out[57]:
```
0    1081
1    1081
Name: Attrition, dtype: int64
```

After applying over sampling we are once again listing the values of our label column to cross verify the updated information. Here we see that we have successfully resolved the class imbalance problem and now all the categories have same data ensuring that the machine learning model does not get biased towards one category.

Feature Scaling

In [58]:
```
scaler = StandardScaler()
X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
X.head()
```

Out[58]:

	Age	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationFie
0	0.799511	0.674572	0.836820	1.412395	-1.724950	-0.749294	-0.9991:
1	1.746398	-0.975164	-1.285089	-0.566828	0.182231	-1.777848	-0.9991:
2	0.326067	0.674572	1.535529	-0.566828	-1.210817	-0.749294	1.3555!
3	-0.147376	-0.975164	1.584516	-0.566828	-0.846034	1.307815	-0.9991:
4	-0.857542	0.674572	-0.480672	-0.566828	-1.210817	-1.777848	0.5706!

5 rows × 30 columns

I am scaling my feature data to ensure that there is no issue with the data biasness over a particular column instead a standardization will occur helping us in having a uniform dataset value.

Finding best random state for building Regression Models

In [59]:
```
maxAccu=0
maxRS=0

for i in range(1, 1000):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, r
    lr=LogisticRegression()
    lr.fit(X_train, Y_train)
    pred = lr.predict(X_test)
    acc_score = (accuracy_score(Y_test, pred))*100

    if acc_score>maxAccu:
        maxAccu=acc_score
        maxRS=i

print("Best accuracy score is", maxAccu,"on Random State", maxRS)
```

```
Best accuracy score is 89.09426987060998 on Random State 526
```

Creating the training and testing data sets¶

In [67]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, rando
```

I am taking 25 percent of the complete dataset for training purpose and the remaing 75 percent with be used to train the machine learning models using the random state as 759

Machine Learning Model for Classification with Evaluation Metrics

In [68]:
```python
# Classification Model Function

def classify(model, X, Y):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, r

    # Training the model
    model.fit(X_train, Y_train)

    # Predicting Y_test
    pred = model.predict(X_test)

    # Accuracy Score
    acc_score = (accuracy_score(Y_test, pred))*100
    print("Accuracy Score:", acc_score)

    # Classification Report
    class_report = classification_report(Y_test, pred)
    print("\nClassification Report:\n", class_report)

    # Cross Validation Score
    cv_score = (cross_val_score(model, X, Y, cv=5).mean())*100
    print("Cross Validation Score:", cv_score)

    # Result of accuracy minus cv scores
    result = acc_score - cv_score
    print("\nAccuracy Score - Cross Validation Score is", result)
```

I have defined a class that will perform the train-test split, training of machine learning model, predicting the label value, getting the accuracy score, generating the classification report, getting the cross validation score and the result of difference between the accuracy score and cross validation score for any machine learning model that calls for this function.

In [69]:
```python
# Logistic Regression

model=LogisticRegression()
classify(model, X, Y)
```

```
Accuracy Score: 89.09426987060998

Classification Report:
               precision    recall  f1-score   support

           0       0.87      0.93      0.90       277
           1       0.92      0.85      0.88       264

    accuracy                           0.89       541
   macro avg       0.89      0.89      0.89       541
weighted avg       0.89      0.89      0.89       541

Cross Validation Score: 84.50742023778976

Accuracy Score - Cross Validation Score is 4.586849632820218
```

In [70]:
```
# Support Vector Classifier

model=SVC(C=1.0, kernel='rbf', gamma='auto', random_state=42)
classify(model, X, Y)
```

```
Accuracy Score: 95.19408502772643

Classification Report:
               precision    recall  f1-score   support

           0       0.93      0.99      0.95       277
           1       0.98      0.92      0.95       264

    accuracy                           0.95       541
   macro avg       0.95      0.95      0.95       541
weighted avg       0.95      0.95      0.95       541

Cross Validation Score: 90.7547472414678

Accuracy Score - Cross Validation Score is 4.439337786258633
```

In [71]:
```
# Decision Tree Classifier

model=DecisionTreeClassifier(random_state=21, max_depth=15)
classify(model, X, Y)
```

```
Accuracy Score: 84.65804066543437

Classification Report:
               precision    recall  f1-score   support

           0       0.87      0.82      0.85       277
           1       0.82      0.87      0.85       264

    accuracy                           0.85       541
   macro avg       0.85      0.85      0.85       541
weighted avg       0.85      0.85      0.85       541

Cross Validation Score: 83.12451886066205

Accuracy Score - Cross Validation Score is 1.5335218047723203
```

In [72]:
```
# Random Forest Classifier

model=RandomForestClassifier(max_depth=15, random_state=111)
classify(model, X, Y)
```

```
Accuracy Score: 92.79112754158965

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.98      0.93       277
           1       0.97      0.88      0.92       264

    accuracy                           0.93       541
   macro avg       0.93      0.93      0.93       541
weighted avg       0.93      0.93      0.93       541

Cross Validation Score: 90.57373193054487

Accuracy Score - Cross Validation Score is 2.217395611044779
```

In [73]:
```
# K Neighbors Classifier

model=KNeighborsClassifier(n_neighbors=15)
classify(model, X, Y)
```

```
Accuracy Score: 80.22181146025879

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.65      0.77       277
           1       0.72      0.96      0.83       264

    accuracy                           0.80       541
   macro avg       0.83      0.81      0.80       541
weighted avg       0.84      0.80      0.80       541

Cross Validation Score: 78.58534342656745

Accuracy Score - Cross Validation Score is 1.6364680336913437
```

In [74]:
```
# Extra Trees Classifier

model=ExtraTreesClassifier()
classify(model, X, Y)
```

```
Accuracy Score: 95.19408502772643

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.97      0.95       277
           1       0.96      0.94      0.95       264

    accuracy                           0.95       541
   macro avg       0.95      0.95      0.95       541
weighted avg       0.95      0.95      0.95       541

Cross Validation Score: 93.15958857240612

Accuracy Score - Cross Validation Score is 2.0344964553203084
```

In [75]:
```
# XGB Classifier

model=xgb.XGBClassifier(verbosity=0)
classify(model, X, Y)
```

```
Accuracy Score: 92.79112754158965

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.96      0.93       277
           1       0.95      0.90      0.92       264

    accuracy                           0.93       541
   macro avg       0.93      0.93      0.93       541
weighted avg       0.93      0.93      0.93       541

Cross Validation Score: 88.77202548969294

Accuracy Score - Cross Validation Score is 4.019102051896709
```

Hyper parameter tuning on the best classification ML model

In [77]:
```
#choosing Extra Tree Classifier
f_mod={'n_estimators':[20,40,60,80,100],'criterion':['gini','entropy'],'min_sam
```

In [79]:
```
GSCV=GridSearchCV(ExtraTreesClassifier(),f_mod,cv=5)
```

In [80]:
```
GSCV.fit(X_train,Y_train)
```

Out[80]:
```
GridSearchCV(cv=5, estimator=ExtraTreesClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'min_samples_split': [1, 2, 3, 4, 5],
                         'n_estimators': [20, 40, 60, 80, 100]})
```

In [81]:
```
print(GSCV.best_score_)
print(GSCV.best_params_)
```

```
0.9364501424501425
{'criterion': 'gini', 'max_features': 'log2', 'min_samples_split': 3, 'n_esti
mators': 100}
```

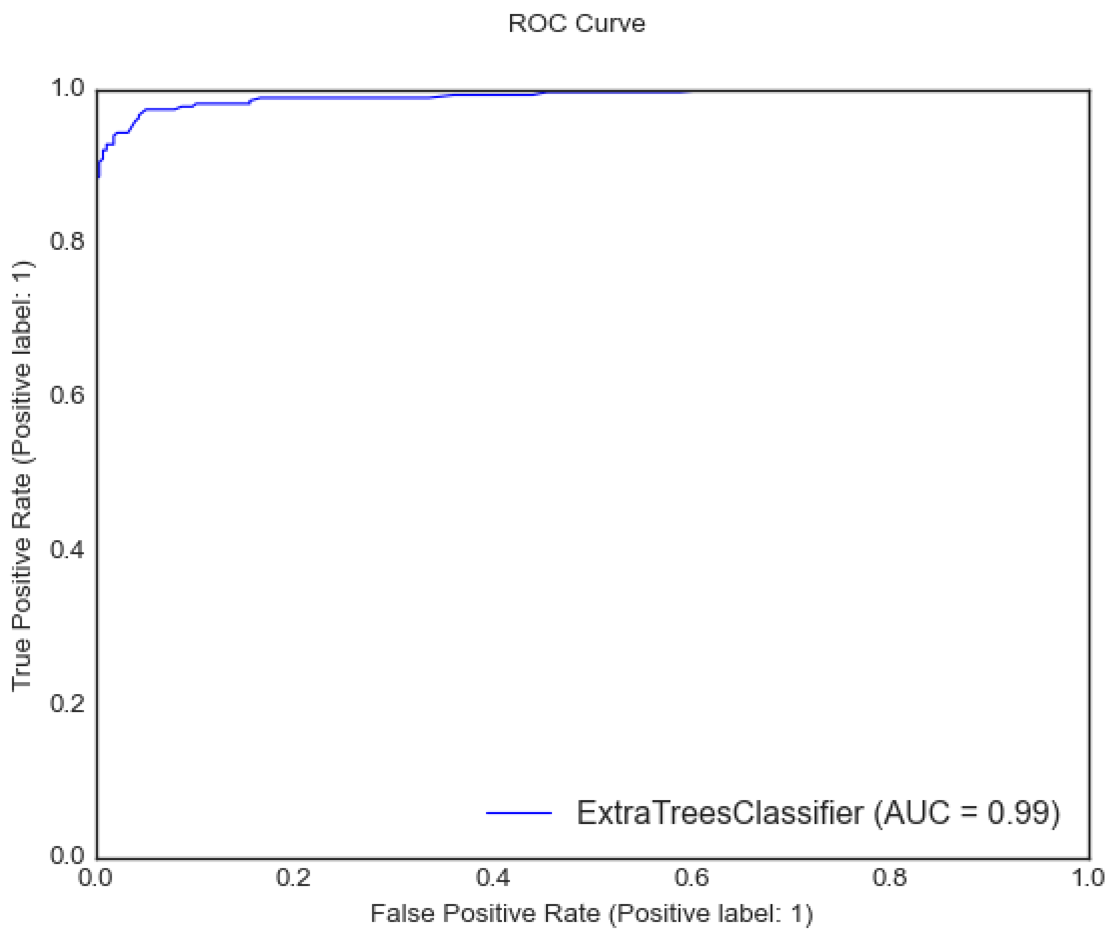
In [89]:
```
Final_Model= ExtraTreesClassifier(criterion='gini',max_features='log2',min_samp
classifier=Final_Model.fit(X_train,Y_train)
fmod_pred=Final_Model.predict(X_test)
fmod_accuracy=(accuracy_score(Y_test,fmod_pred))*100
accuracy_score=print('accuracy_score=',fmod_accuracy)
```

```
accuracy_score= 96.11829944547135
```

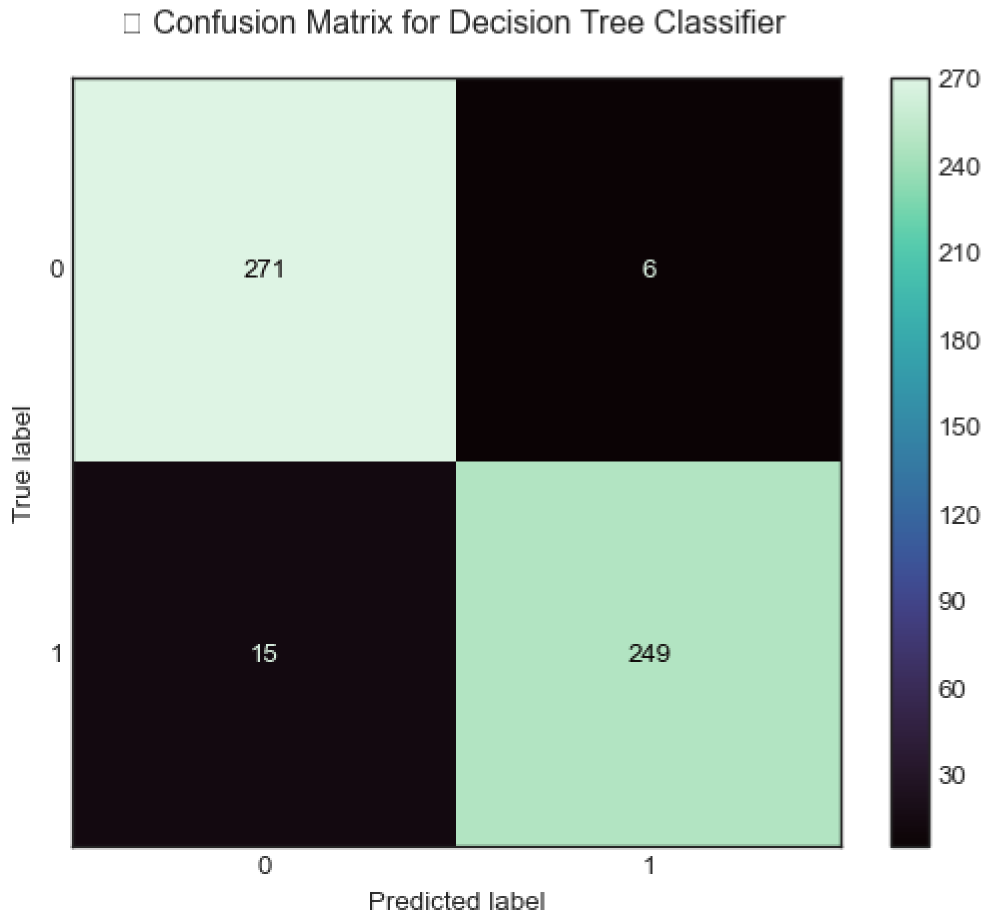
I have successfully incorporated the Hyper Parameter Tuning on my Final Model and received the accuracy score for it.

AUC ROC Curve

In [90]:
```
disp = metrics.plot_roc_curve(Final_Model, X_test, Y_test)
disp.figure_.suptitle("ROC Curve")
plt.show()
```



In [92]:
```
class_names = df.columns
metrics.plot_confusion_matrix(classifier, X_test, Y_test, cmap='mako')
plt.title('\t Confusion Matrix for Decision Tree Classifier \n')
plt.show()
```



In [101]:
```
filname='Final_Model03'
```

In [103]:
```
filename = "FinalModel_03"
joblib.dump(Final_Model, filename)
```

Out[103]: ['FinalModel_03']

In [ ]: