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 [4]:
```
df = pd.read_csv('C:\\Project\\WA_Fn-UseC_-HR-Employee-Attrition.csv')
df
```

Out[4]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	Edu
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Li
1	49	No	Travel_Frequently	279	Research & Development	8	1	Li
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Li
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	Li
1468	49	No	Travel_Frequently	1023	Sales	2	3	
1469	34	No	Travel_Rarely	628	Research & Development	8	3	

1470 rows × 35 columns

EDA

In [5]:
```
df.columns
```

Out[5]:
```
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 [6]:
```
df.isnull().sum()
```

Out[6]:
```
Age                         0
Attrition                   0
BusinessTravel              0
DailyRate                   0
Department                  0
DistanceFromHome            0
Education                   0
EducationField              0
EmployeeCount               0
EmployeeNumber              0
EnvironmentSatisfaction     0
Gender                      0
HourlyRate                  0
JobInvolvement              0
JobLevel                    0
JobRole                     0
JobSatisfaction             0
MaritalStatus               0
MonthlyIncome               0
MonthlyRate                 0
NumCompaniesWorked          0
Over18                      0
OverTime                    0
PercentSalaryHike           0
PerformanceRating           0
RelationshipSatisfaction    0
StandardHours               0
StockOptionLevel            0
TotalWorkingYears           0
TrainingTimesLastYear       0
WorkLifeBalance             0
YearsAtCompany              0
YearsInCurrentRole          0
YearsSinceLastPromotion     0
YearsWithCurrManager        0
dtype: int64
```

In [7]:
```
df.info()
```

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

In [8]:
```
df.describe().T
```

Out[8]:

	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	1
MonthlyRate	1470.0	14313.103401	7117.786044	2094.0	8047.00	14235.5	20461.50	2
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,YearWithCurrMan has huge differece in their 75 % and 100% so they must have some amount of outlier that is need to be treated.

In [9]:
```
df.drop(["EmployeeCount", "EmployeeNumber", "Over18", "StandardHours"], axis=1, i
```

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 [10]:
```
df.shape
```

Out[10]:
```
(1470, 31)
```

In [11]:
```
object_datatypes=[]
for x in df.dtypes.index:
    if df.dtypes[x]=='O':
        object_datatypes.append(x)
object_datatypes
```

Out[11]:
```
['Attrition',
 'BusinessTravel',
 'Department',
 'EducationField',
 'Gender',
 'JobRole',
 'MaritalStatus',
 'OverTime']
```

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

In [12]:
```
integer_datatypes=[]
for x in df.dtypes.index:
    if df.dtypes[x]=='int64':
      integer_datatypes.append(x)
integer_datatypes
```

Out[12]:
```
['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 [13]:
```
df.nunique().to_frame("Unique values")
```

Out[13]:

	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

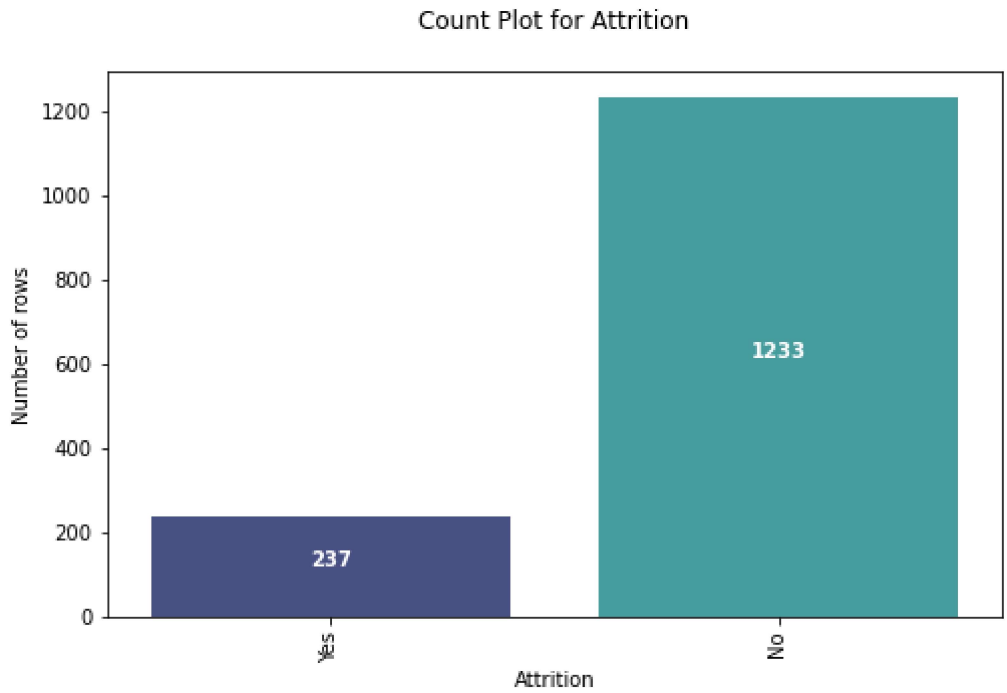
In [14]:
```
for col in object_datatypes:
    print(col)
    print(df[col].value_counts())
    print("="*80)
```

```
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
================================================================================
=
EducationField
Life Sciences       606
Medical             464
Marketing           159
Technical Degree    132
Other                82
Human Resources      27
Name: EducationField, dtype: int64
================================================================================
=
Gender
Male      882
Female    588
Name: Gender, dtype: int64
================================================================================
=
JobRole
Sales Executive              326
Research Scientist           292
Laboratory Technician        259
Manufacturing Director       145
Healthcare Representative    131
Manager                      102
Sales Representative          83
Research Director             80
Human Resources               52
Name: JobRole, dtype: int64
================================================================================
=
MaritalStatus
Married     673
Single      470
```

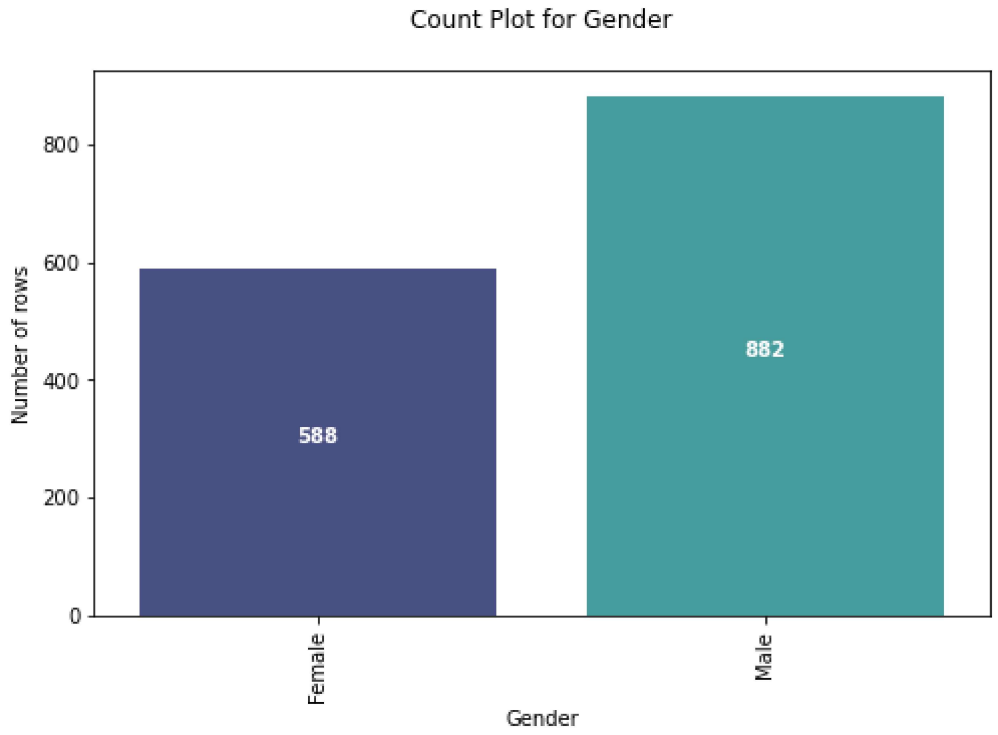
```
Divorced    327
Name: MaritalStatus, dtype: int64
================================================================================
=
OverTime
No     1054
Yes     416
Name: OverTime, dtype: int64
================================================================================
=
```

Visualization

In [15]:
```
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()
```

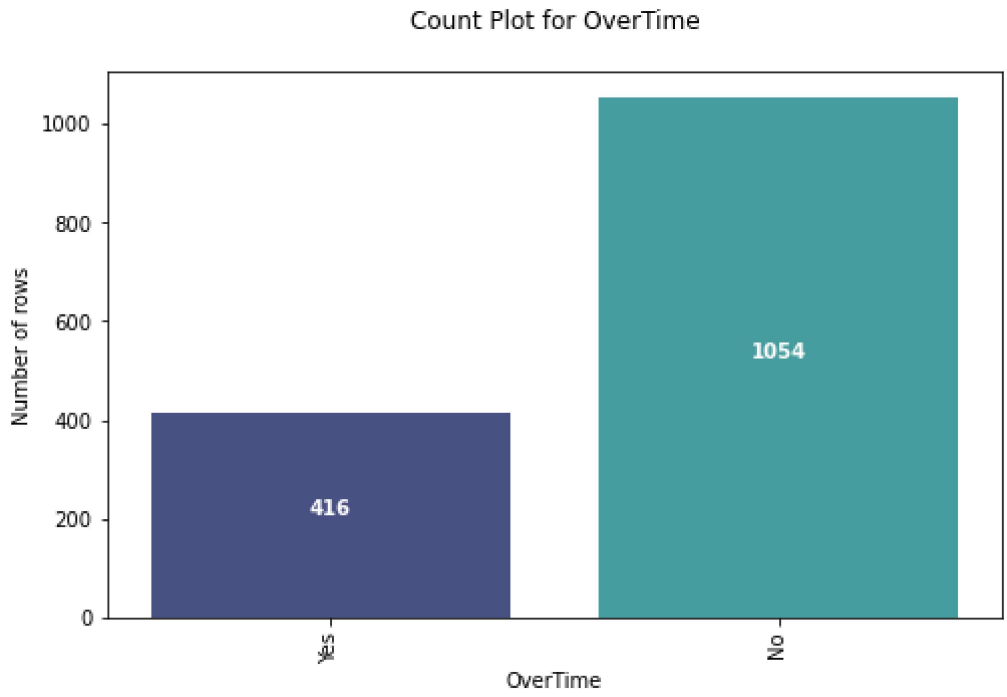


In [16]:
```
plt.figure(figsize=(8,5))
col_name = 'Gender'
values = df[col_name].value_counts()
index = 0
ax = sns.countplot(df[col_name], palette="mako")

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()
```

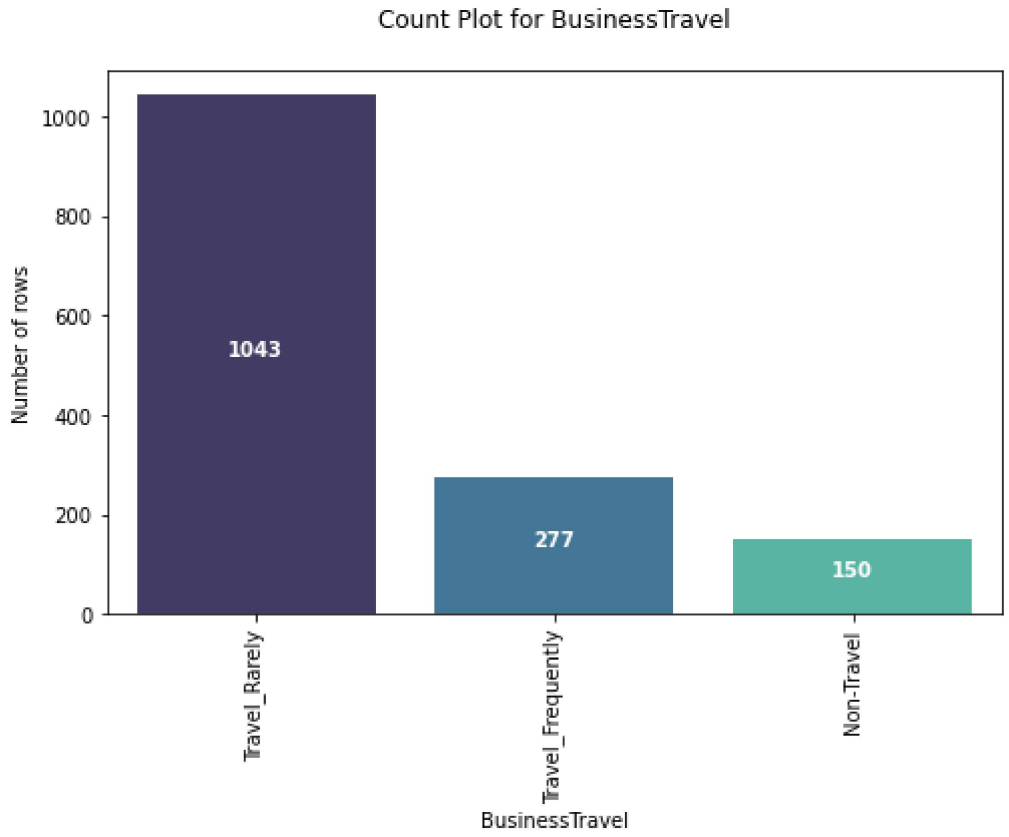


here we can see the count of Male is more than that of female.

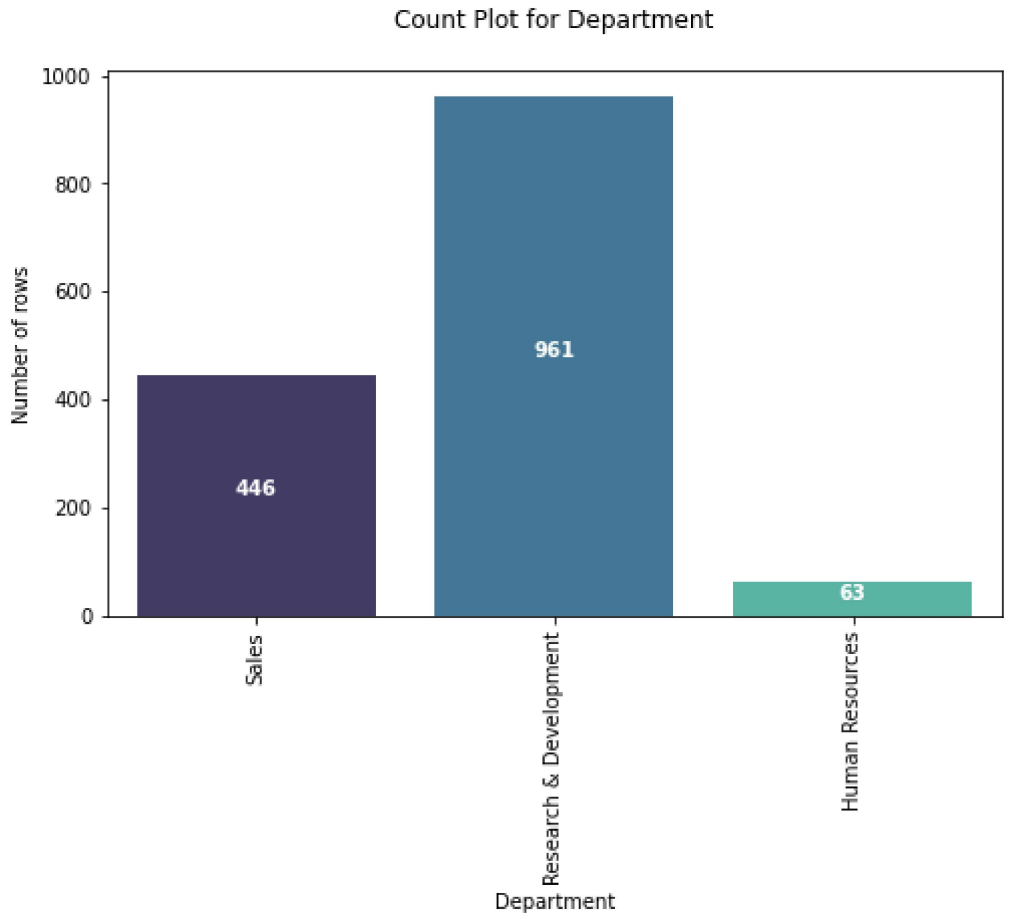
In [17]:
```
plt.figure(figsize=(8,5))
col_name = 'OverTime'
values = df[col_name].value_counts()
index = 0
ax = sns.countplot(df[col_name], palette="mako")

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()
```



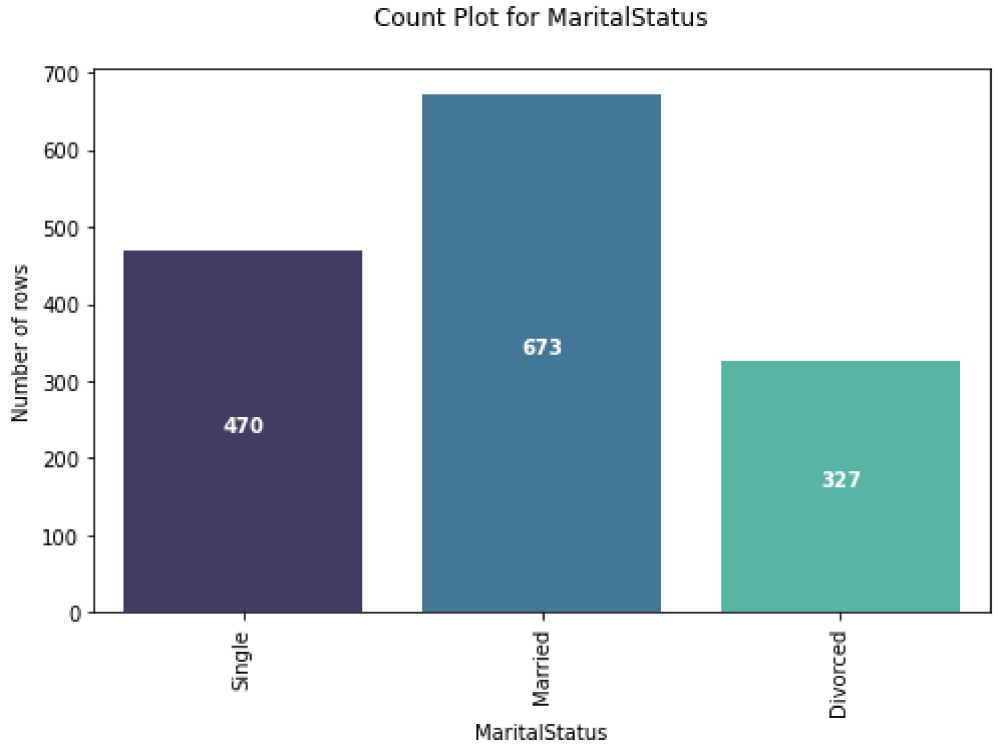
In [18]:
```
plt.figure(figsize=(8,5))
col_name = 'BusinessTravel'
values = df[col_name].value_counts()
index = 0
ax = sns.countplot(df[col_name], palette="mako")

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()
```



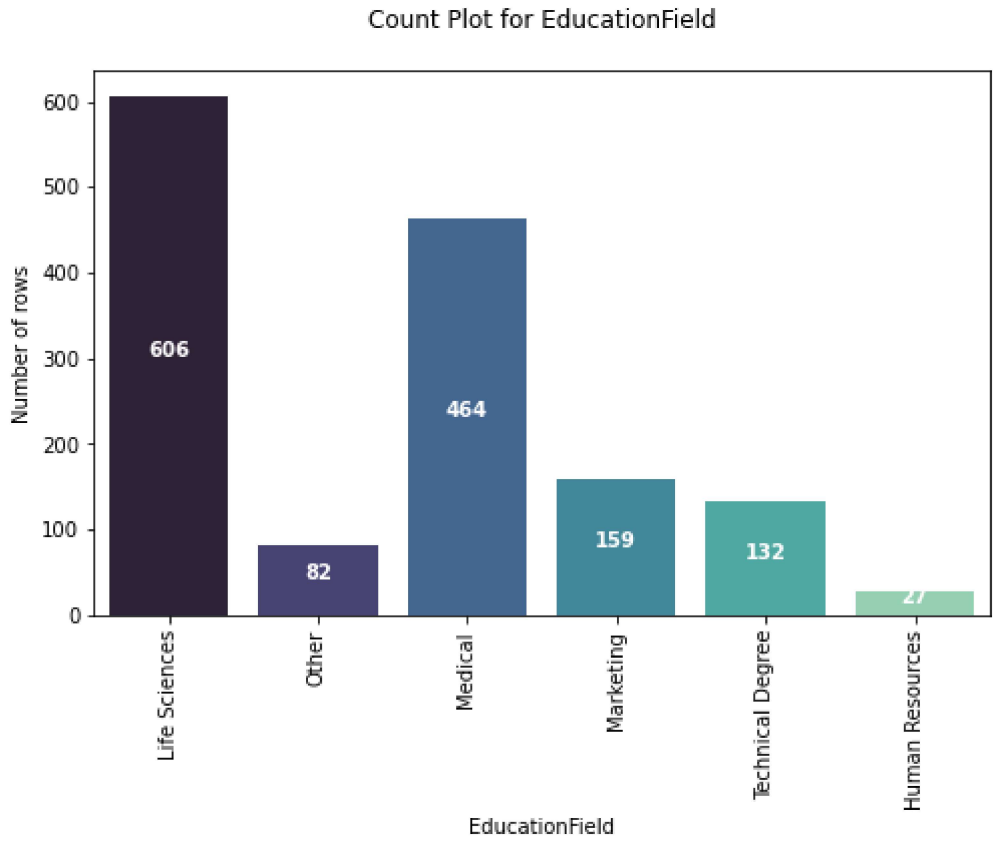
In [19]:
```
plt.figure(figsize=(8,5))
col_name = 'Department'
values = df[col_name].value_counts()
index = 0
ax = sns.countplot(df[col_name], palette="mako")

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()
```



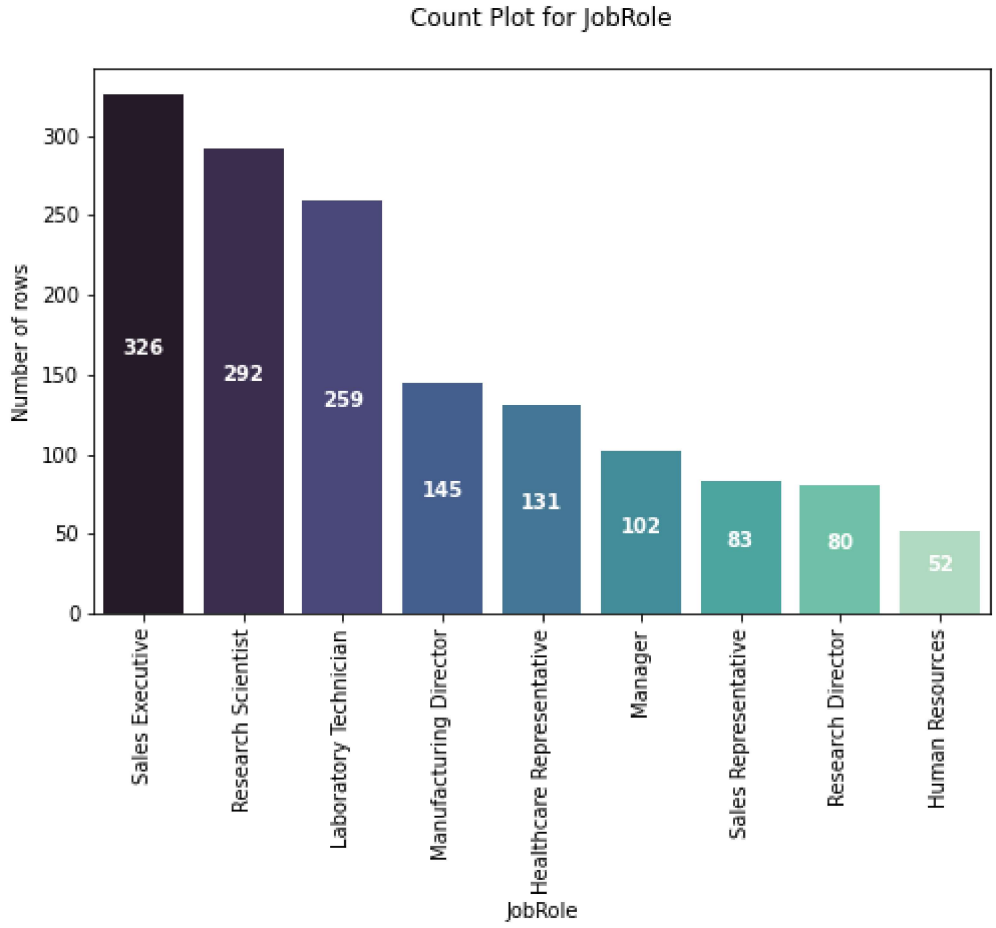
In [20]:
```
plt.figure(figsize=(8,5))
col_name = 'MaritalStatus'
values = df[col_name].value_counts()
index = 0
ax = sns.countplot(df[col_name], palette="mako")

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()
```



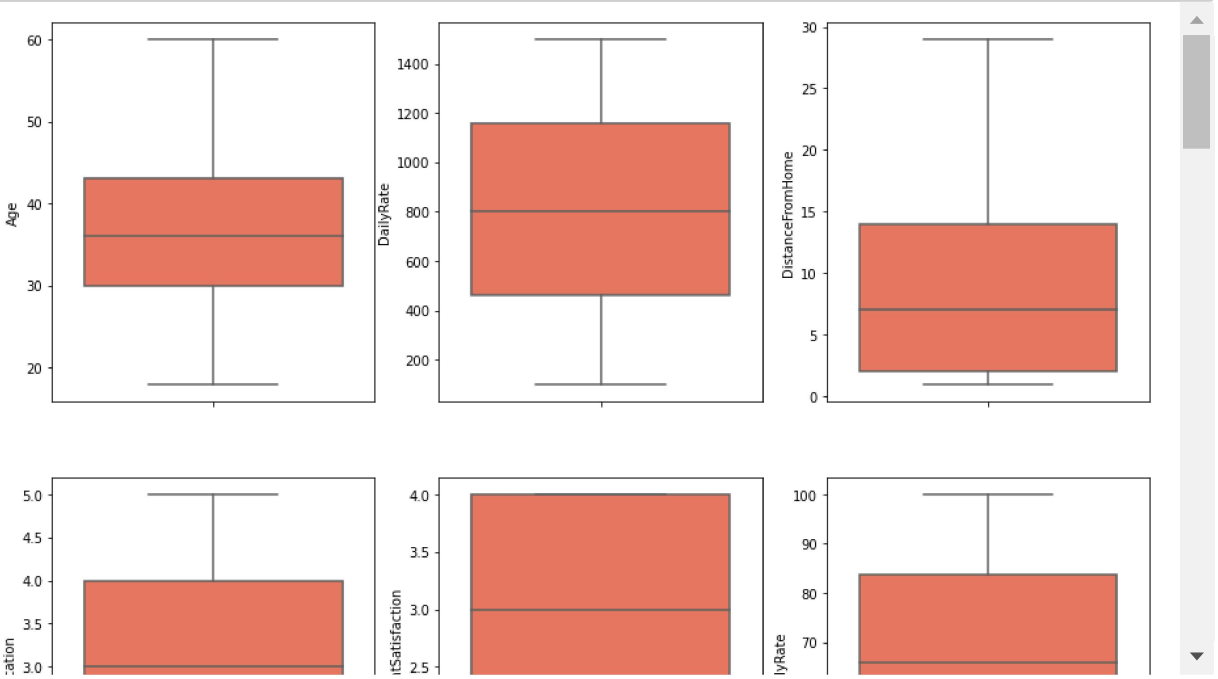
In [21]:
```
plt.figure(figsize=(8,5))
col_name = 'EducationField'
values = df[col_name].value_counts()
index = 0
ax = sns.countplot(df[col_name], palette="mako")

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 [22]:
```
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()
```

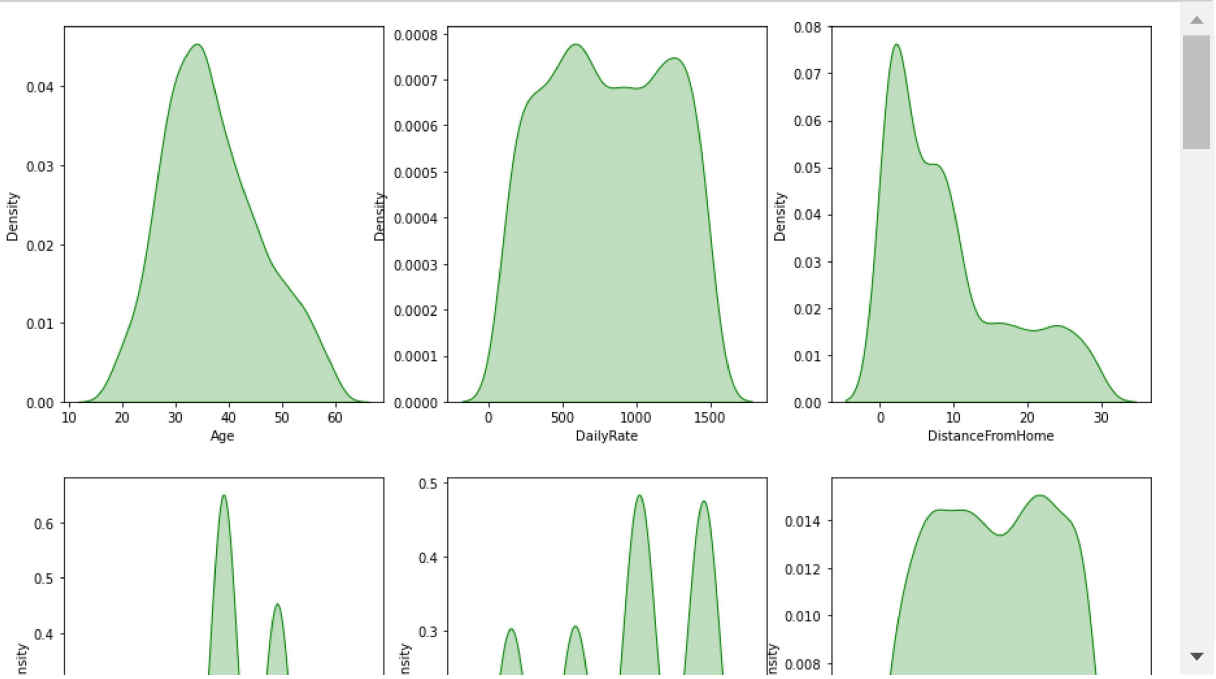


In [23]:
```
fig, ax = plt.subplots(ncols=3, nrows=8, figsize=(15,50))
index = 0
ax = ax.flatten()
for col, value in df[integer_datatypes].items():
    sns.boxplot(y=col, data=df, ax=ax[index], palette="Reds")
    index += 1
plt.show()
```



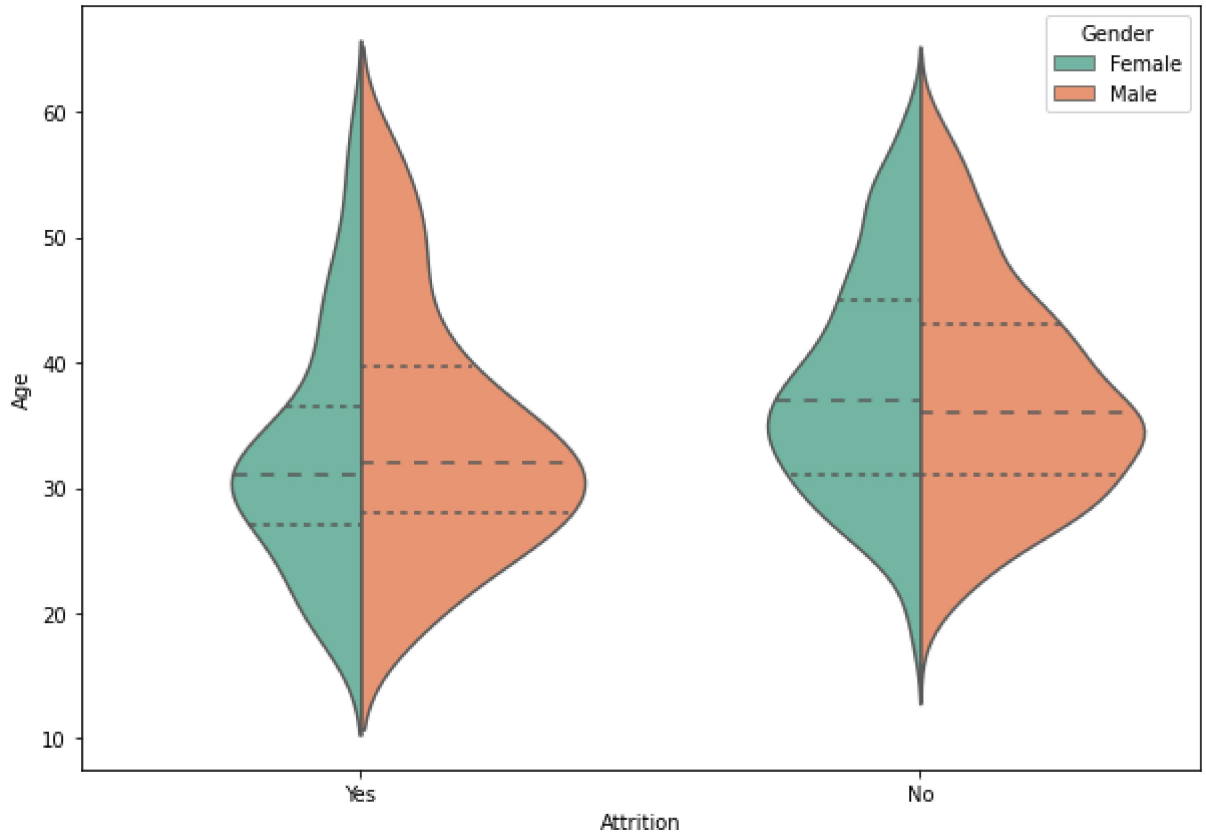
From the above boxplot we have many columns having outliers they need to be treated well

In [24]:
```
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": Tr
    index += 1
plt.show()
```



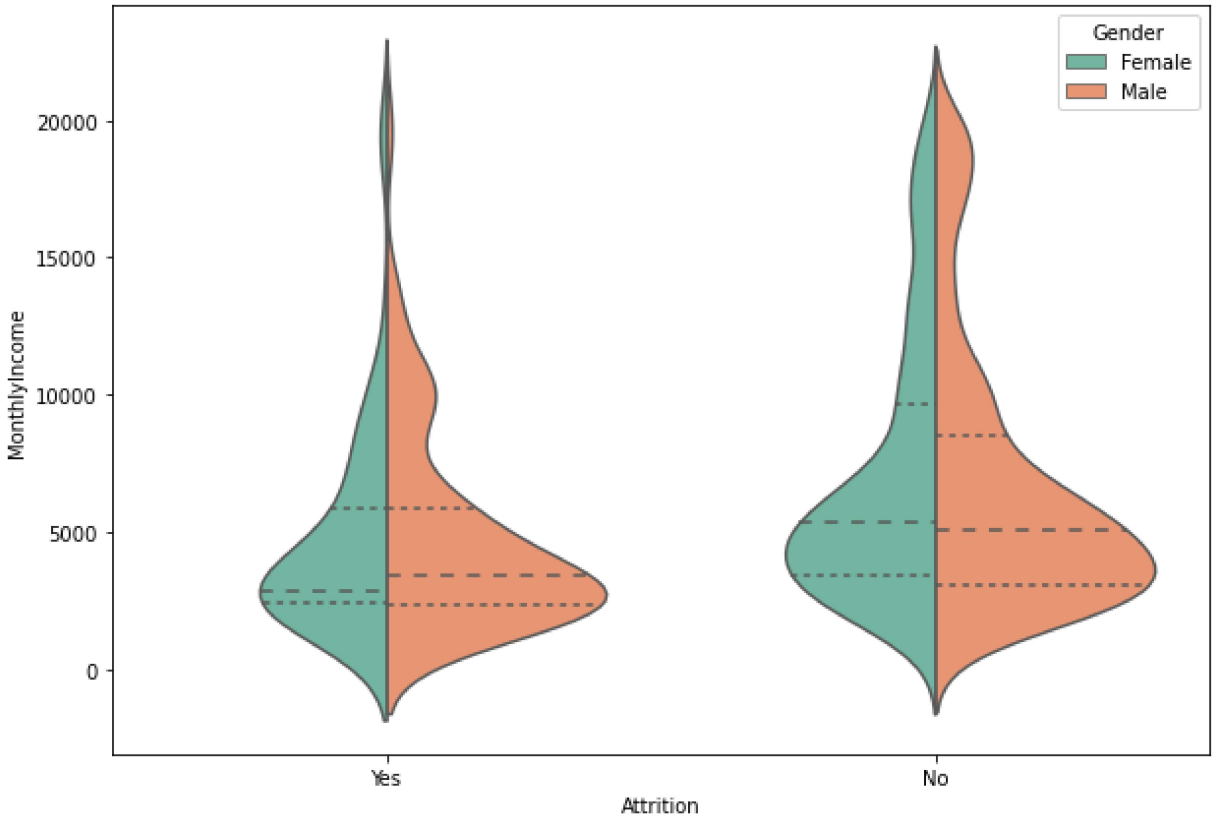
from the above distplot method we found that there is some skewness also present and they have to be treated before model selection.

In [25]:
```
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()
```



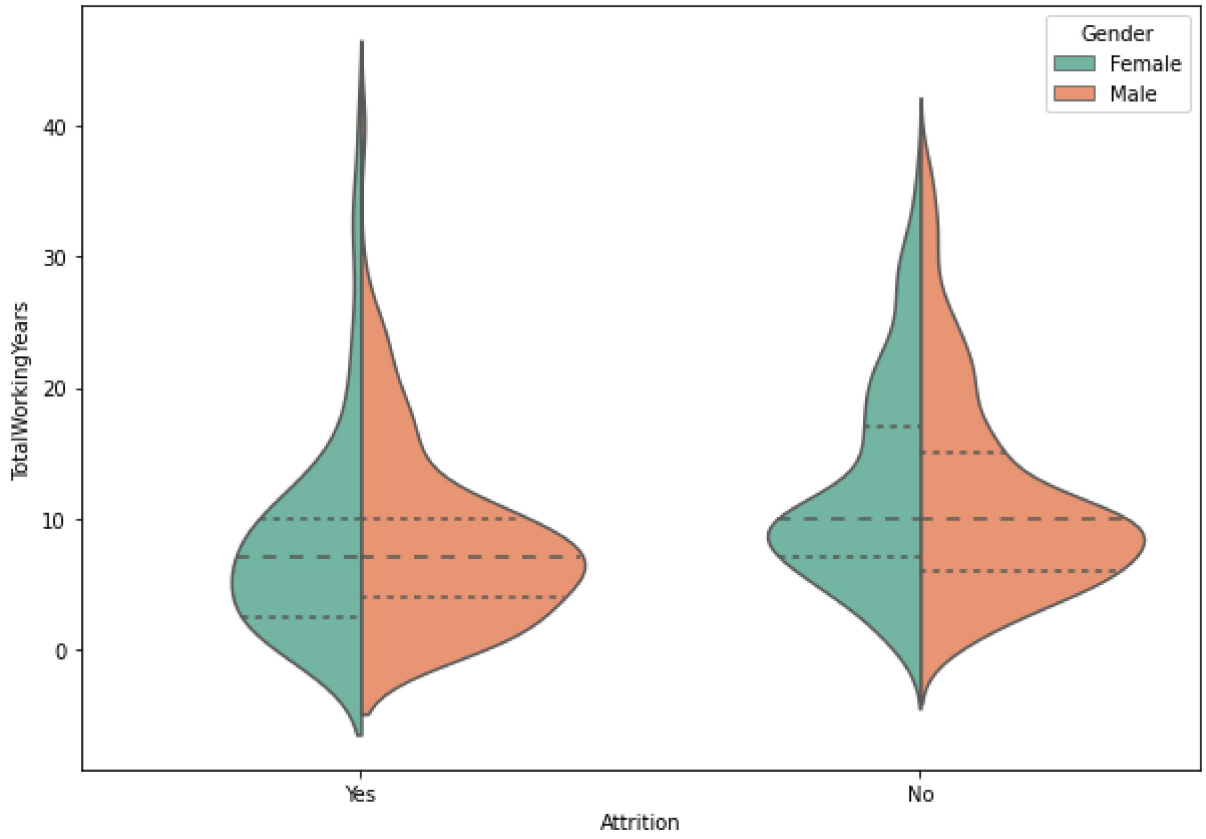
from above we can say that both male and female has wide range in age between 20 and 30

In [26]:
```
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()
```



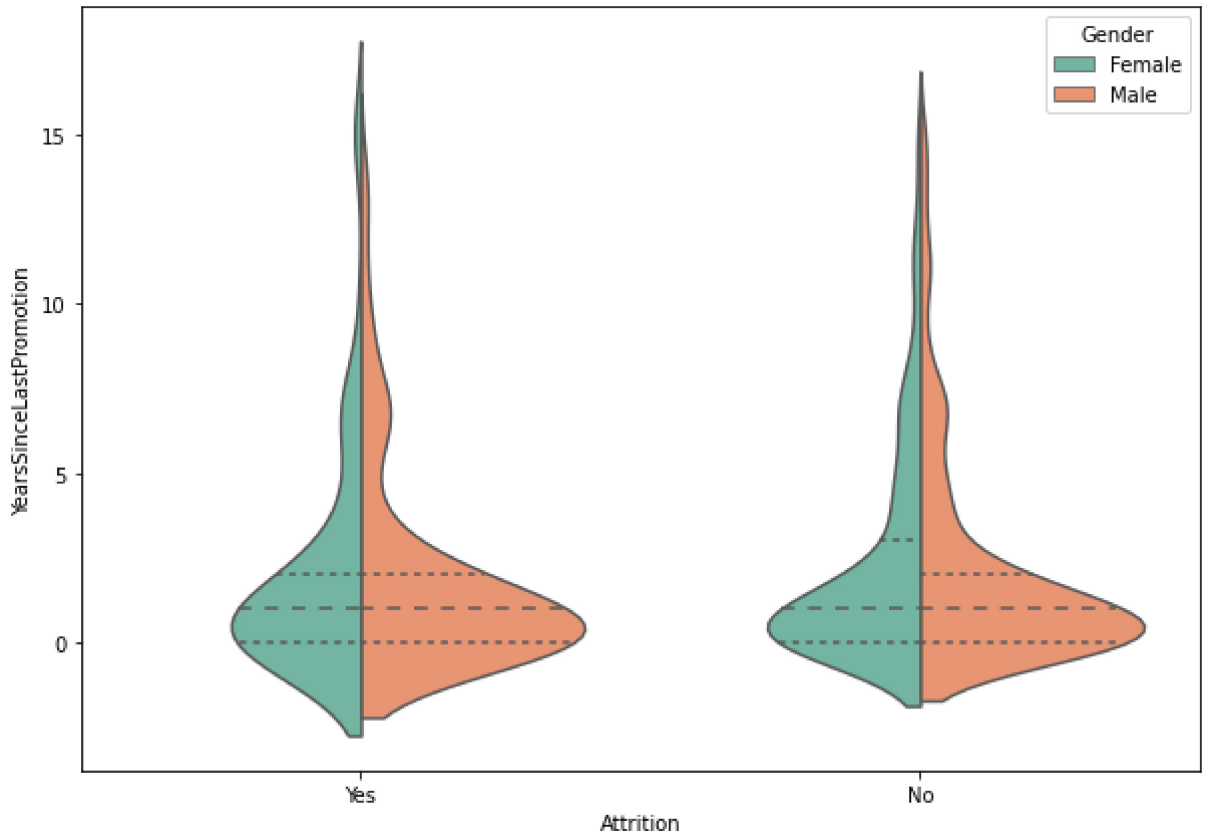
from above we have observation that most of male and female are having Monthlyincome less than 5000

In [27]:
```
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()
```



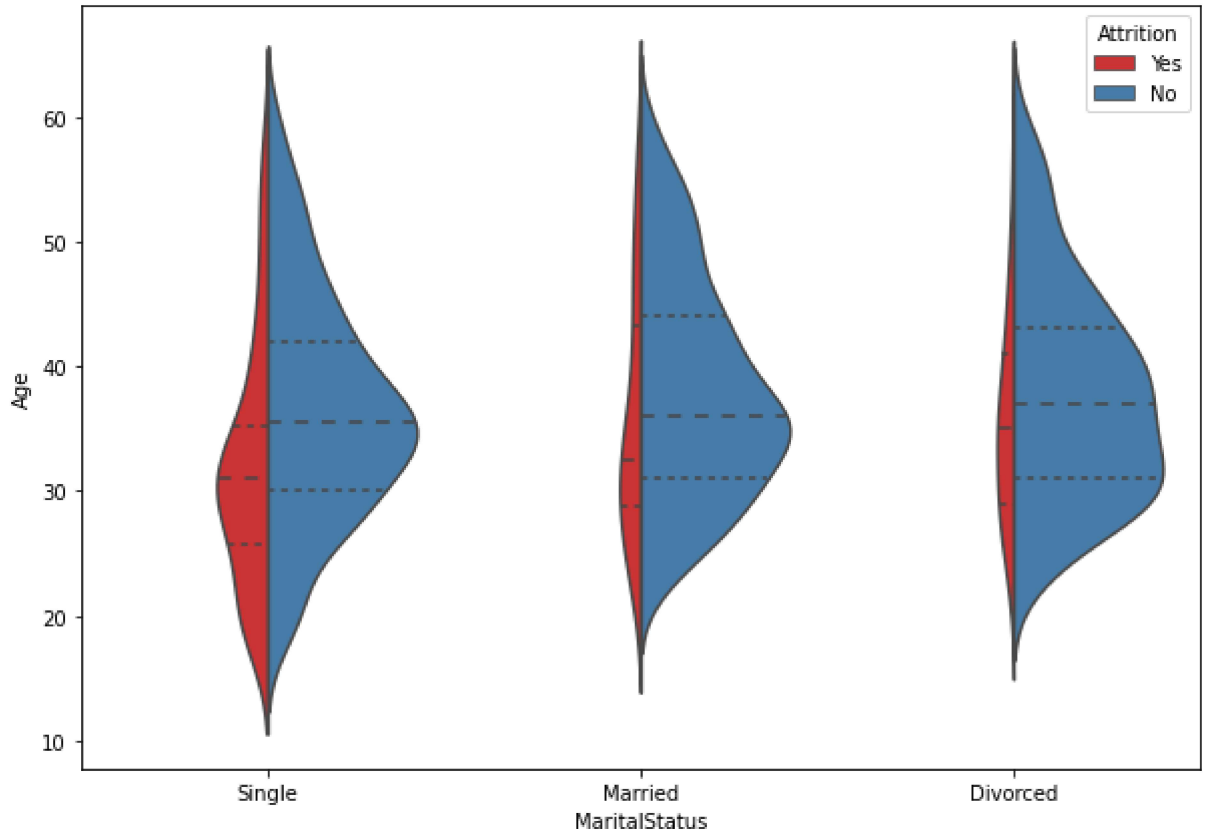
from the above observation male and female mostly have Totlaworkexperienve in between range 0 to 10 years

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



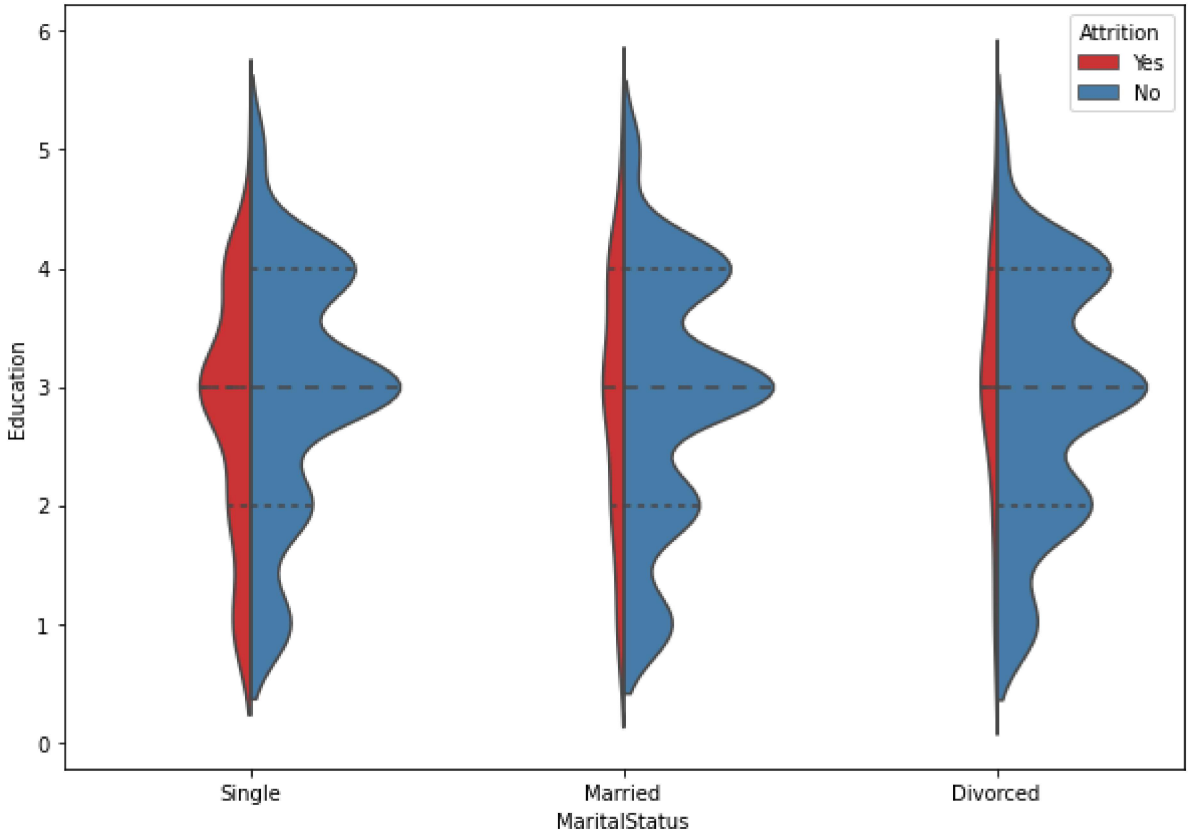
from above plot we have that for both male and female mostly are less than 5 years since they have got promoted

In [30]:
```
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()
```

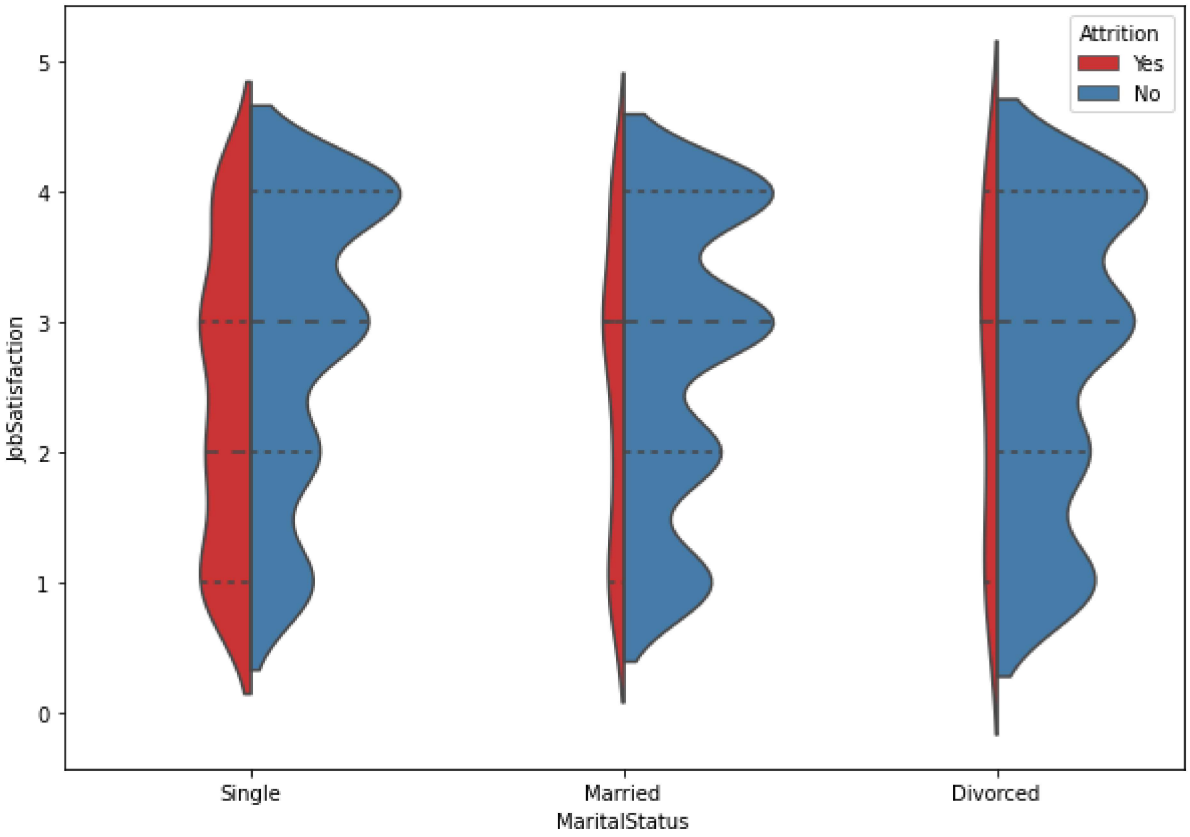


in this plot we can see that there is some Attrition when you are single and less when you are married but little when you are divorced.

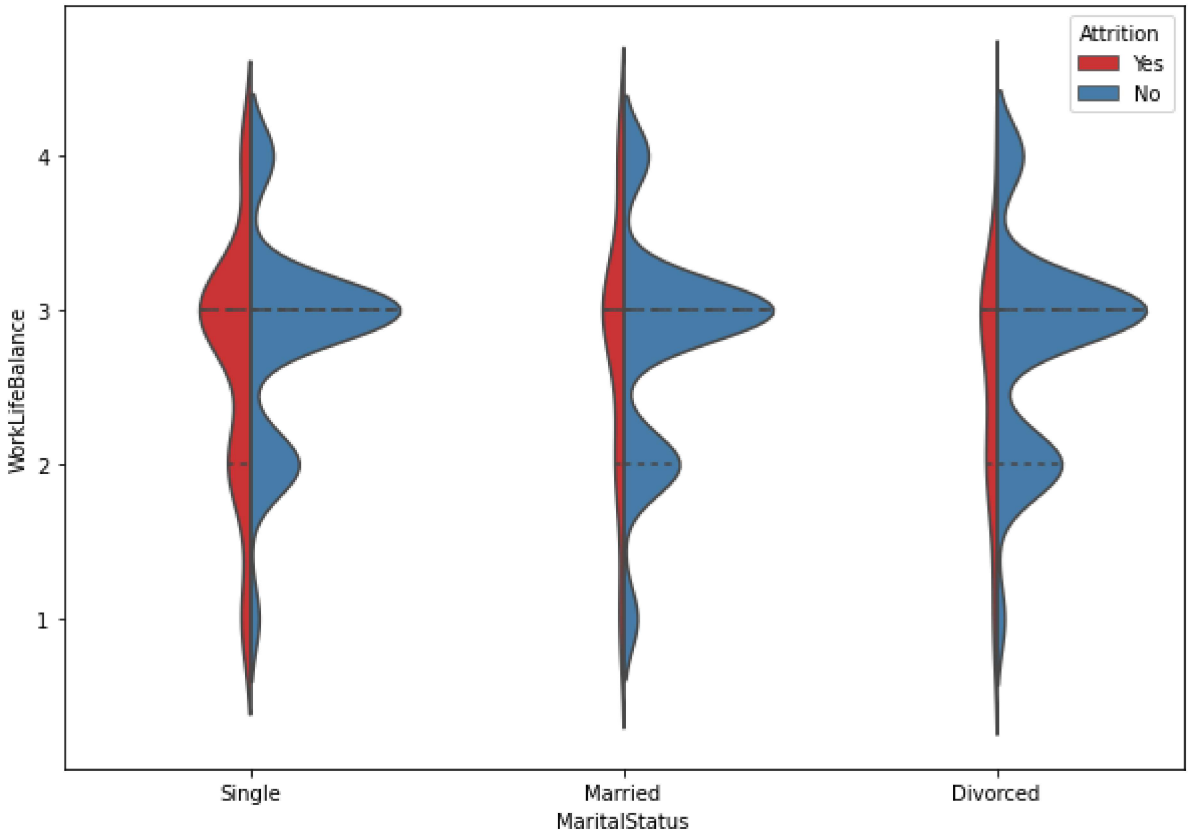
In [31]:
```
plt.figure(figsize=(10,7))
sns.violinplot(x="MaritalStatus", y="Education", hue="Attrition", data=df,
               palette="Set1", split=True, scale="count", inner="quartile")
plt.show()
```



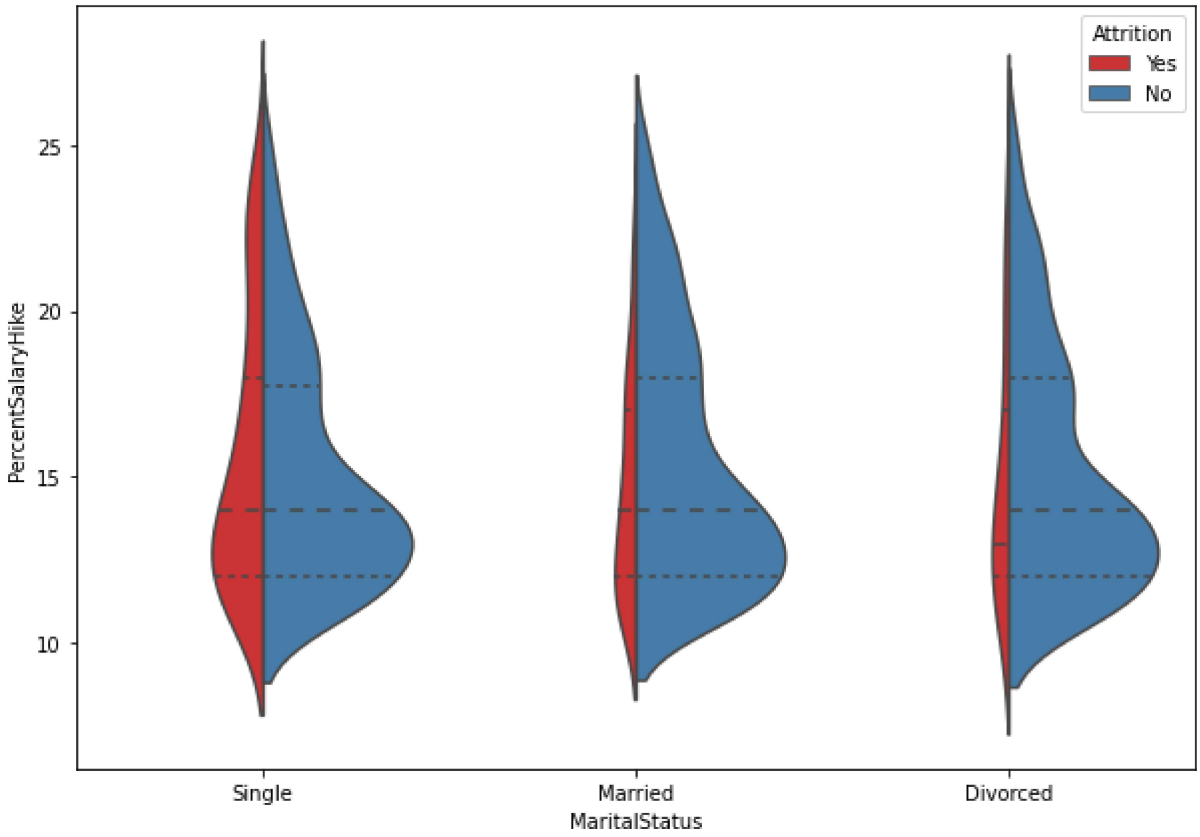
In [32]:
```
plt.figure(figsize=(10,7))
sns.violinplot(x="MaritalStatus", y="JobSatisfaction", hue="Attrition", data=df,
               palette="Set1", split=True, scale="count", inner="quartile")
plt.show()
```



In [33]:
```
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()
```



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



Encoding the categorical object datatype

columns

In [35]:
```
le=LabelEncoder()
df['Attrition']=le.fit_transform(df['Attrition'])
```

In [36]:
```
df['Attrition']
```

Out[36]:
```
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
```

In [37]:
```
# Ordinal Encoder

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

In [38]:
```
df.head()
```

Out[38]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	Education
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	

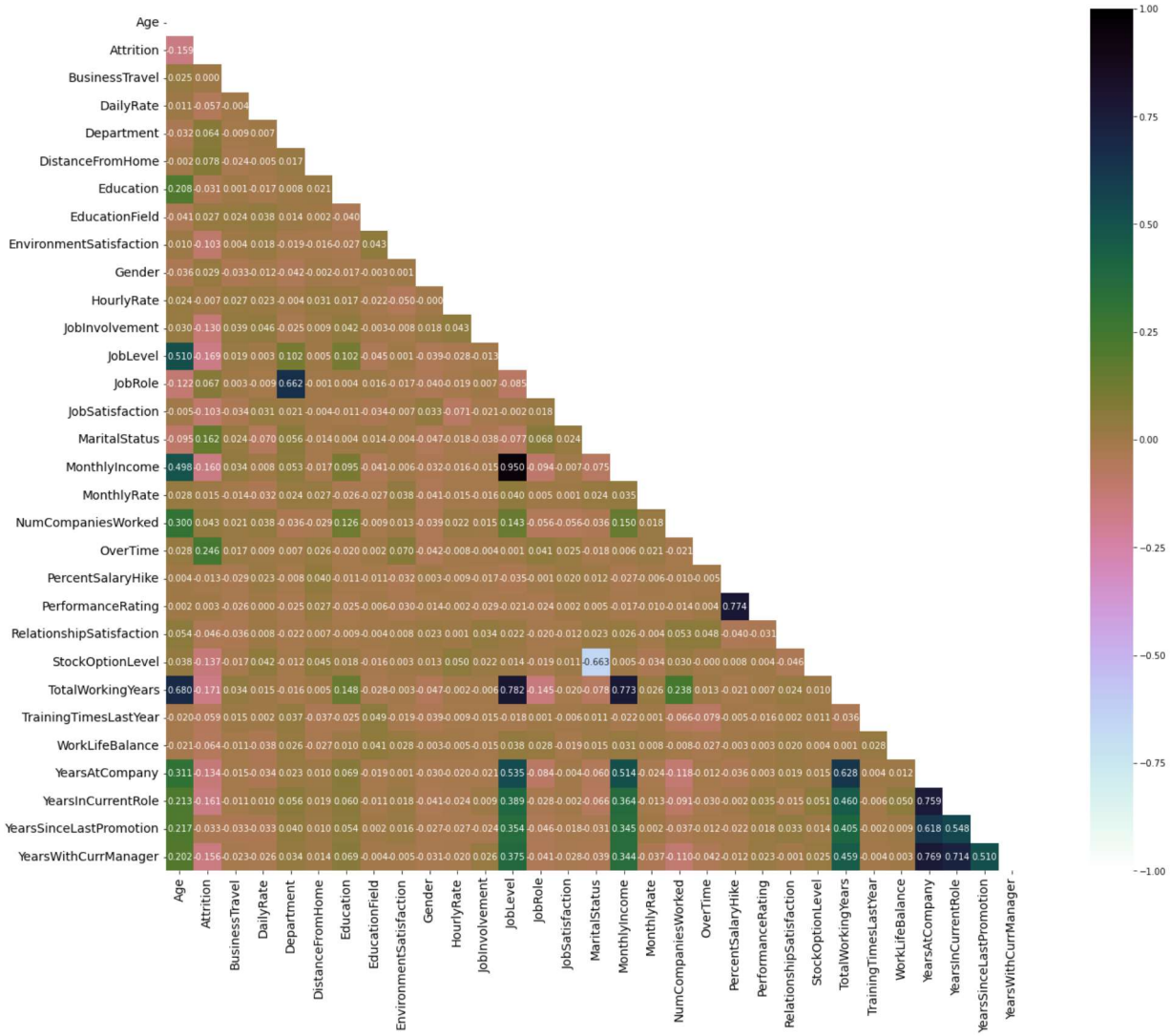
5 rows × 31 columns

In [39]:
```
df.hist(figsize=(20,25))
plt.show()
```

Correlation using a Heatmap

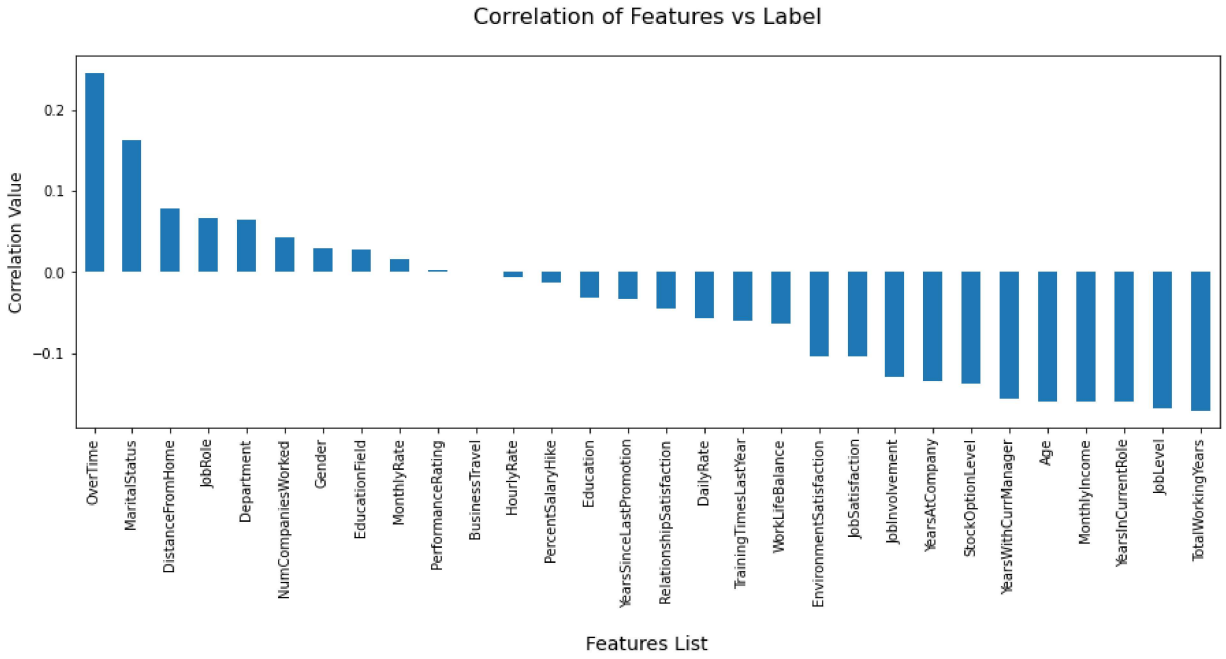
Positive correlation - A correlation of +1 indicates a perfect positive correlation, meaning that both variables move in the same direction together. Negative correlation - A correlation of –1 indicates a perfect negative correlation, meaning that as one variable goes up, the other goes down.

In [40]:
```
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()
```



Correlation Bar Plot comparing features with our label

In [41]:
```
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()
```

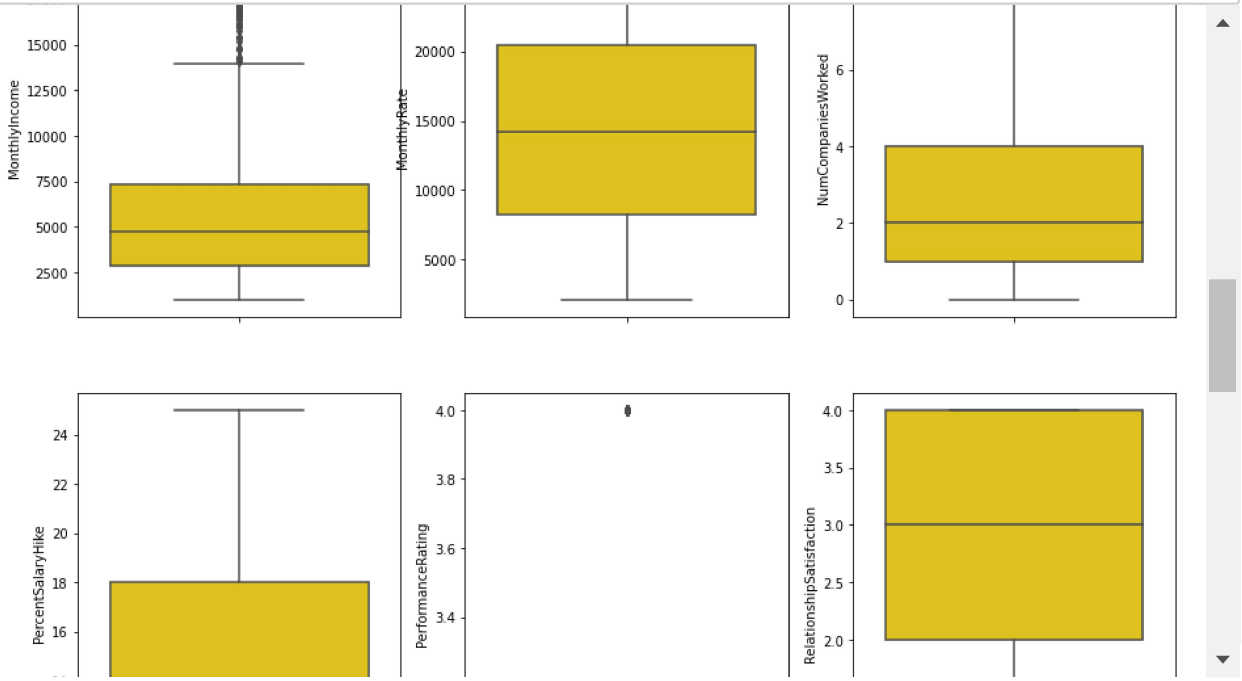


Using Z Score to remove outliers

In [42]:
```
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[0

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

```
Shape of the dataframe before removing outliers:  (1470, 31)
Shape of the dataframe after removing outliers:  (1387, 31)
Percentage of data loss post outlier removal:  5.646258503401361
```

In [43]:
```
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()
```



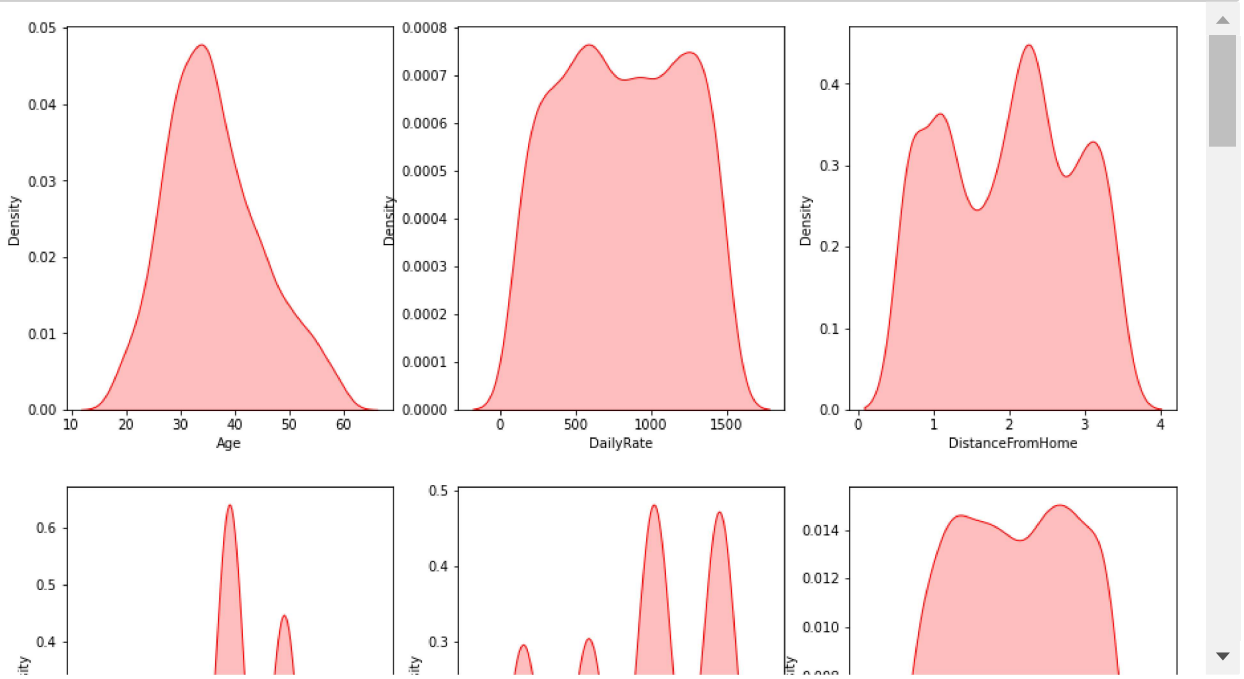
In [44]:
```
df.skew()
```

Out[44]:
```
Age                        0.472280
Attrition                  1.805983
BusinessTravel            -1.426774
DailyRate                 -0.017078
Department                 0.183919
DistanceFromHome           0.954752
Education                 -0.289024
EducationField             0.544868
EnvironmentSatisfaction   -0.325285
Gender                    -0.417296
HourlyRate                -0.030481
JobInvolvement            -0.501401
JobLevel                   1.126075
JobRole                   -0.386843
JobSatisfaction           -0.345612
MaritalStatus             -0.160952
MonthlyIncome              1.544770
MonthlyRate                0.030596
NumCompaniesWorked         1.037715
OverTime                   0.954751
PercentSalaryHike          0.800592
PerformanceRating          1.931566
RelationshipSatisfaction  -0.295686
StockOptionLevel           0.962332
TotalWorkingYears          1.034487
TrainingTimesLastYear      0.577614
WorkLifeBalance           -0.557100
YearsAtCompany             1.248623
YearsInCurrentRole         0.726675
YearsSinceLastPromotion    1.756335
YearsWithCurrManager       0.694506
dtype: float64
```

Using Log Transform to fix skewness

In [45]:
```
for col in integer_datatypes:
    if df.skew().loc[col]>0.55:
        df[col]=np.log1p(df[col])
```

In [46]:
```
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": Tr
    index += 1
plt.show()
```



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

In [47]:
```
X = df.drop('Attrition', axis=1)
Y = df['Attrition']
```

Resolving the class imbalance issue in label column

In [48]:
```
Y.value_counts()
```

Out[48]:
```
0    1158
1     229
Name: Attrition, dtype: int64
```

In [49]:
```
# adding samples to make all the categorical quality values same

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

In [50]:
```
Y.value_counts()
```

Out[50]:
```
1    1158
0    1158
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 [51]:
```
scaler = StandardScaler()
X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
X.head()
```

Out[51]:

	Age	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationFielc
0	0.695950	0.649086	0.816763	1.402424	-1.636311	-0.740261	-1.019801
1	1.626093	-1.033441	-1.316037	-0.569702	0.227612	-1.773682	-1.019801
2	0.230879	0.649086	1.519057	-0.569702	-1.133840	-0.740261	1.348176
3	-0.234192	-1.033441	1.568296	-0.569702	-0.777331	1.326580	-1.019801
4	-0.931799	0.649086	-0.507491	-0.569702	-1.133840	-1.773682	0.558850

5 rows × 30 columns

Finding best random state for building Regression Models

In [52]:
```
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, ran
    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 88.94645941278065 on Random State 827
```

Creating the training and testing data sets

In [53]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_
```

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 [54]:
```
# 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, ran

    # 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)
```

In [55]:
```
# Logistic Regression

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

```
Accuracy Score: 84.28324697754749

Classification Report:
               precision    recall  f1-score   support

           0       0.88      0.81      0.84       295
           1       0.81      0.88      0.85       284

    accuracy                           0.84       579
   macro avg       0.84      0.84      0.84       579
weighted avg       0.85      0.84      0.84       579

Cross Validation Score: 84.84834661502943

Accuracy Score - Cross Validation Score is -0.5650996374819357
```

In [56]:
```
# Support Vector Classifier

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

```
Accuracy Score: 89.63730569948186

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.87      0.90       295
           1       0.88      0.92      0.90       284

    accuracy                           0.90       579
   macro avg       0.90      0.90      0.90       579
weighted avg       0.90      0.90      0.90       579

Cross Validation Score: 90.6809041483578

Accuracy Score - Cross Validation Score is -1.0435984488759402
```

In [57]:
```
# Decision Tree Classifier

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

```
Accuracy Score: 84.80138169257341

Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.81      0.84       295
           1       0.82      0.89      0.85       284

    accuracy                           0.85       579
   macro avg       0.85      0.85      0.85       579
weighted avg       0.85      0.85      0.85       579

Cross Validation Score: 83.42984285395102

Accuracy Score - Cross Validation Score is 1.3715388386223992
```

In [58]:
```
# Random Forest Classifier

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

```
Accuracy Score: 92.91882556131262

Classification Report:
               precision    recall  f1-score   support

           0       0.92      0.94      0.93       295
           1       0.94      0.92      0.93       284

    accuracy                           0.93       579
   macro avg       0.93      0.93      0.93       579
weighted avg       0.93      0.93      0.93       579

Cross Validation Score: 90.90107618976688

Accuracy Score - Cross Validation Score is 2.017749371545733
```

In [59]:
```
# K Neighbors Classifier

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

```
Accuracy Score: 73.92055267702936

Classification Report:
               precision    recall  f1-score   support

           0       0.94      0.52      0.67       295
           1       0.66      0.97      0.78       284

    accuracy                           0.74       579
   macro avg       0.80      0.74      0.73       579
weighted avg       0.80      0.74      0.73       579

Cross Validation Score: 78.2827511730096

Accuracy Score - Cross Validation Score is -4.362198495980238
```

In [60]:
```
# Extra Trees Classifier

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

```
Accuracy Score: 94.47322970639033

Classification Report:
               precision    recall  f1-score   support

           0       0.96      0.93      0.95       295
           1       0.93      0.96      0.94       284

    accuracy                           0.94       579
   macro avg       0.94      0.94      0.94       579
weighted avg       0.95      0.94      0.94       579

Cross Validation Score: 94.26314515528415

Accuracy Score - Cross Validation Score is 0.21008455110617774
```

In [61]:
```
# XGB Classifier

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

```
Accuracy Score: 91.8825561312608

Classification Report:
               precision    recall  f1-score   support

           0       0.92      0.92      0.92       295
           1       0.91      0.92      0.92       284

    accuracy                           0.92       579
   macro avg       0.92      0.92      0.92       579
weighted avg       0.92      0.92      0.92       579

Cross Validation Score: 88.05513145155285

Accuracy Score - Cross Validation Score is 3.8274246797079456
```

Hyper parameter tuning on the best classification ML model

In [62]:
```
tra Tree Classifier
timators':[20,40,60,80,100],'criterion':['gini','entropy'],'min_samples_split':[1,
```

In [63]:
```
GSCV=GridSearchCV(ExtraTreesClassifier(),f_mod,cv=5)
GSCV.fit(X_train,Y_train)
```

Out[63]:

GridSearchCV
estimator: ExtraTreesClassifier
ExtraTreesClassifier

In [64]:
```
print(GSCV.best_score_)
print(GSCV.best_params_)
```

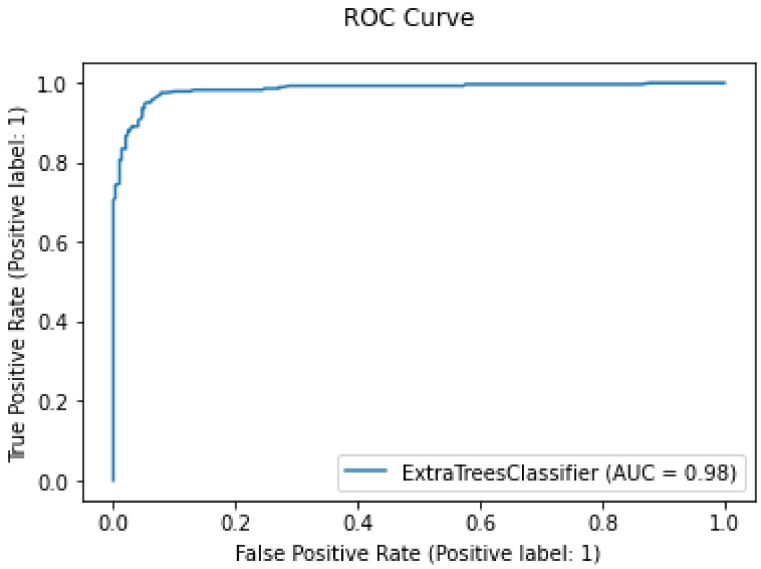
```
0.9378200669117891
{'criterion': 'entropy', 'max_features': 'log2', 'min_samples_split': 2, 'n_est
imators': 100}
```

In [65]:
```
Final_Model= ExtraTreesClassifier(criterion='gini',max_features='log2',min_sample
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)
```

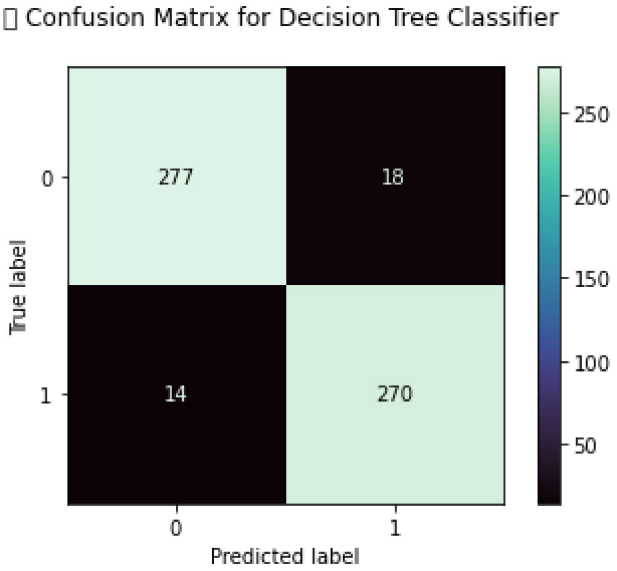
```
accuracy_score= 94.47322970639033
```

AUC ROC Curve

In [66]:
```
disp = metrics.plot_roc_curve(Final_Model, X_test, Y_test)
disp.figure_.suptitle("ROC Curve")
plt.show()
```



In [67]:
```
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 [68]:
```
filname='Final_Model02'
```

In [69]:
```
filename = "FinalModel_02"
joblib.dump(Final_Model, filename)
```

Out[69]:
```
['FinalModel_02']
```

In [ ]:
```

```