Micro Project Satish Kumar

In [1]:
```
# import 'Pandas'
import pandas as pd

# import 'Numpy'
import numpy as np

# import subpackage of Matplotlib
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

# import 'Seaborn'
import seaborn as sns

# to suppress warnings
from warnings import filterwarnings
filterwarnings('ignore')

# display all columns of the dataframe
pd.options.display.max_columns = None

# display all rows of the dataframe
pd.options.display.max_rows = None

# to display the float values upto 6 decimal places
pd.options.display.float_format = '{:.6f}'.format

# import train-test split
from sklearn.model_selection import train_test_split

# import various functions from statsmodels
import statsmodels
import statsmodels.api as sm

# import StandardScaler to perform scaling
from sklearn.preprocessing import StandardScaler

# import various functions from sklearn
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve
from sklearn.metrics import accuracy_score
```

In [142]:
```
df=pd.read_csv('HR_DATASET.csv')
df.head(5)
```

Out[142]:

| | Age | Attrition | BusinessTravel | DailyRate | Department | DistanceFromHome | Education | Educa |
|---|---|---|---|---|---|---|---|---|
| **0** | 41 | Yes | Travel_Rarely | 1102 | Sales | 1 | 2 | Life |
| **1** | 49 | No | Travel_Frequently | 279 | Research & Development | 8 | 1 | Life |
| **2** | 37 | Yes | Travel_Rarely | 1373 | Research & Development | 2 | 2 | |
| **3** | 33 | No | Travel_Frequently | 1392 | Research & Development | 3 | 4 | Life |
| **4** | 27 | No | Travel_Rarely | 591 | Research & Development | 2 | 1 | |

In [143]:
```
df.shape
```

Out[143]: (917, 35)

In [144]:
```
df.dtypes
```

Out[144]:
```
Age                        int64
Attrition                 object
BusinessTravel            object
DailyRate                  int64
Department                object
DistanceFromHome           int64
Education                  int64
EducationField            object
EmployeeCount              int64
EmployeeNumber             int64
EnvironmentSatisfaction    int64
Gender                    object
HourlyRate                 int64
JobInvolvement             int64
JobLevel                   int64
JobRole                   object
JobSatisfaction            int64
MaritalStatus             object
MonthlyIncome              int64
MonthlyRate                int64
NumCompaniesWorked         int64
Over18                    object
OverTime                  object
PercentSalaryHike          int64
PerformanceRating          int64
RelationshipSatisfaction   int64
StandardHours              int64
StockOptionLevel           int64
TotalWorkingYears          int64
TrainingTimesLastYear      int64
WorkLifeBalance            int64
YearsAtCompany             int64
YearsInCurrentRole         int64
YearsSinceLastPromotion    int64
YearsWithCurrManager       int64
dtype: object
```

In [145]:
```
df.describe()
```

Out[145]:

| | Age | DailyRate | DistanceFromHome | Education | EmployeeCount | EmployeeNumbe |
|---|---|---|---|---|---|---|
| **count** | 917.000000 | 917.000000 | 917.000000 | 917.000000 | 917.000000 | 917.00000 |
| **mean** | 36.938931 | 785.711014 | 9.294438 | 2.921483 | 1.000000 | 1413.46673 |
| **std** | 8.984981 | 401.902148 | 8.092566 | 1.020717 | 0.000000 | 388.60688 |
| **min** | 18.000000 | 102.000000 | 1.000000 | 1.000000 | 1.000000 | 1.00000 |
| **25%** | 30.000000 | 444.000000 | 2.000000 | 2.000000 | 1.000000 | 1084.00000 |
| **50%** | 36.000000 | 772.000000 | 7.000000 | 3.000000 | 1.000000 | 1425.00000 |
| **75%** | 43.000000 | 1141.000000 | 14.000000 | 4.000000 | 1.000000 | 1737.00000 |
| **max** | 60.000000 | 1498.000000 | 29.000000 | 5.000000 | 1.000000 | 2068.00000 |

In [146]:
```
# convert numerical variables to categorical (object)

df['EnvironmentSatisfaction'] = df['EnvironmentSatisfaction'].astype(object)
df['JobInvolvement'] = df['JobInvolvement'].astype(object)
df['JobLevel'] = df['JobLevel'].astype(object)
df['PerformanceRating'] = df['PerformanceRating'].astype(object)
df['RelationshipSatisfaction'] = df['RelationshipSatisfaction'].astype(object)
df['StockOptionLevel'] = df['StockOptionLevel'].astype(object)
df['WorkLifeBalance'] = df['WorkLifeBalance'].astype(object)
```

In [147]:
```
# dropping insignificant variables
column=['EmployeeCount','StandardHours','EmployeeNumber','Over18']
df=df.drop(columns=column,axis=1)
df.head()
```

Out[147]:

| | Age | Attrition | BusinessTravel | DailyRate | Department | DistanceFromHome | Education | Educa |
|---|---|---|---|---|---|---|---|---|
| **0** | 41 | Yes | Travel_Rarely | 1102 | Sales | 1 | 2 | Life |
| **1** | 49 | No | Travel_Frequently | 279 | Research & Development | 8 | 1 | Life |
| **2** | 37 | Yes | Travel_Rarely | 1373 | Research & Development | 2 | 2 | |
| **3** | 33 | No | Travel_Frequently | 1392 | Research & Development | 3 | 4 | Life |
| **4** | 27 | No | Travel_Rarely | 591 | Research & Development | 2 | 1 | |

In [148]:
```
# Check for duplicate data
dups = df.duplicated()
print('Number of duplicate rows = %d' % (dups.sum()))
df[dups]
```

```
Number of duplicate rows = 0
```

Out[148]:

| | Age | Attrition | BusinessTravel | DailyRate | Department | DistanceFromHome | Education | Educatio |
|---|---|---|---|---|---|---|---|---|

In [149]:
```
missing_data = df.isnull().sum()
# missing_data_per = missing_data*100/len(data)
# missing_data_per
missing_data
```

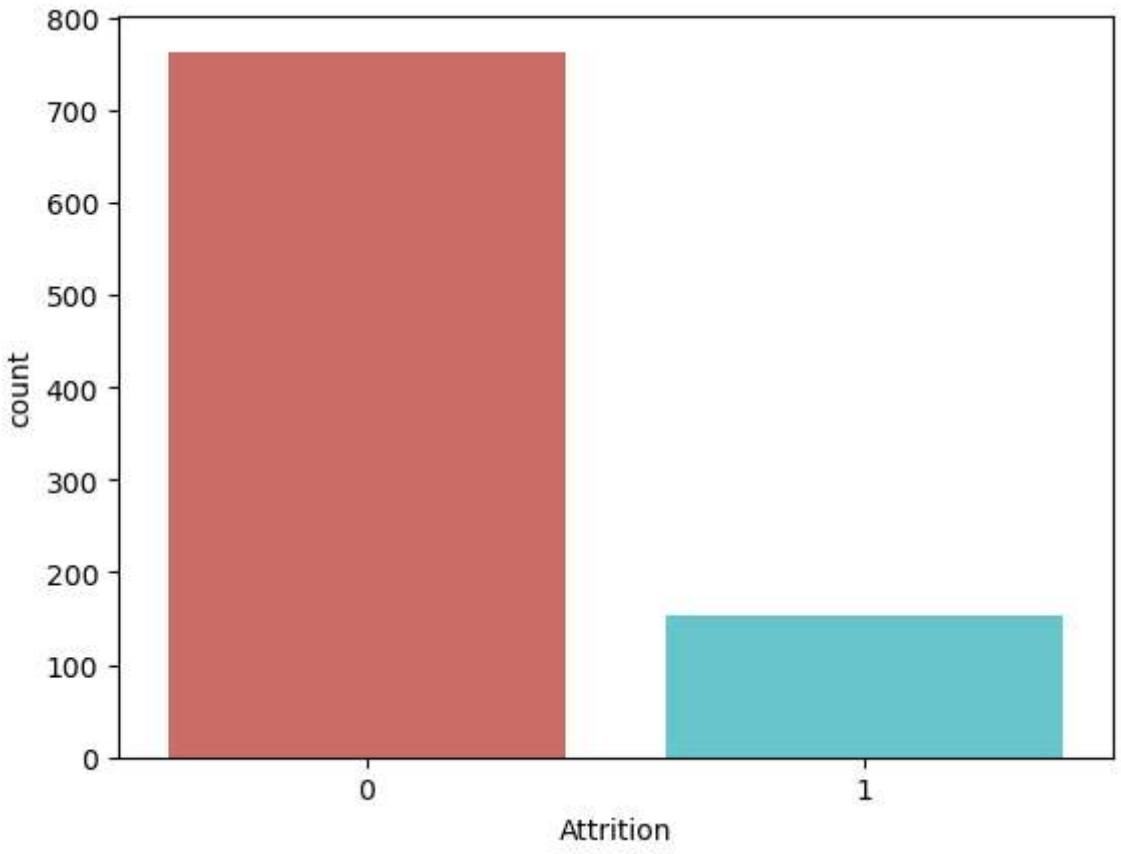
Out[149]:
```
Age                         0
Attrition                   0
BusinessTravel              0
DailyRate                   0
Department                  0
DistanceFromHome            0
Education                   0
EducationField              0
EnvironmentSatisfaction     0
Gender                      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
```

In [280]:
```
df['Attrition'].value_counts()
```

Out[280]:
```
0    763
1    154
Name: Attrition, dtype: int64
```

In [151]:
```
#A lambda function is a small anonymous function.
#A lambda function can take any number of arguments, but can only have one expr
df['Attrition']=df['Attrition'].apply(lambda x : 1 if x=='Yes' else 0)
```

In [152]:
```
sns.countplot(x= 'Attrition',data=df,palette ='hls')
plt.show()
```




In [153]:
```
num_col = df.select_dtypes(include=np.number).columns.tolist()
print(num_col)
print(type(num_col))
```

```
['Age', 'Attrition', 'DailyRate', 'DistanceFromHome', 'Education', 'HourlyRat
e', 'JobSatisfaction', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
'PercentSalaryHike', 'TotalWorkingYears', 'TrainingTimesLastYear', 'YearsAtCo
mpany', 'YearsInCurrentRole', 'YearsSinceLastPromotion', 'YearsWithCurrManage
r']
<class 'list'>
```

In [154]:
```
data_num = df[num_col]
data_num.head()
```

Out[154]:

| | Age | Attrition | DailyRate | DistanceFromHome | Education | HourlyRate | JobSatisfaction | MonthlyI |
|---|---|---|---|---|---|---|---|---|
| **0** | 41 | 1 | 1102 | 1 | 2 | 94 | 4 | |
| **1** | 49 | 0 | 279 | 8 | 1 | 61 | 2 | |
| **2** | 37 | 1 | 1373 | 2 | 2 | 92 | 3 | |
| **3** | 33 | 0 | 1392 | 3 | 4 | 56 | 3 | |
| **4** | 27 | 0 | 591 | 2 | 1 | 40 | 2 | |

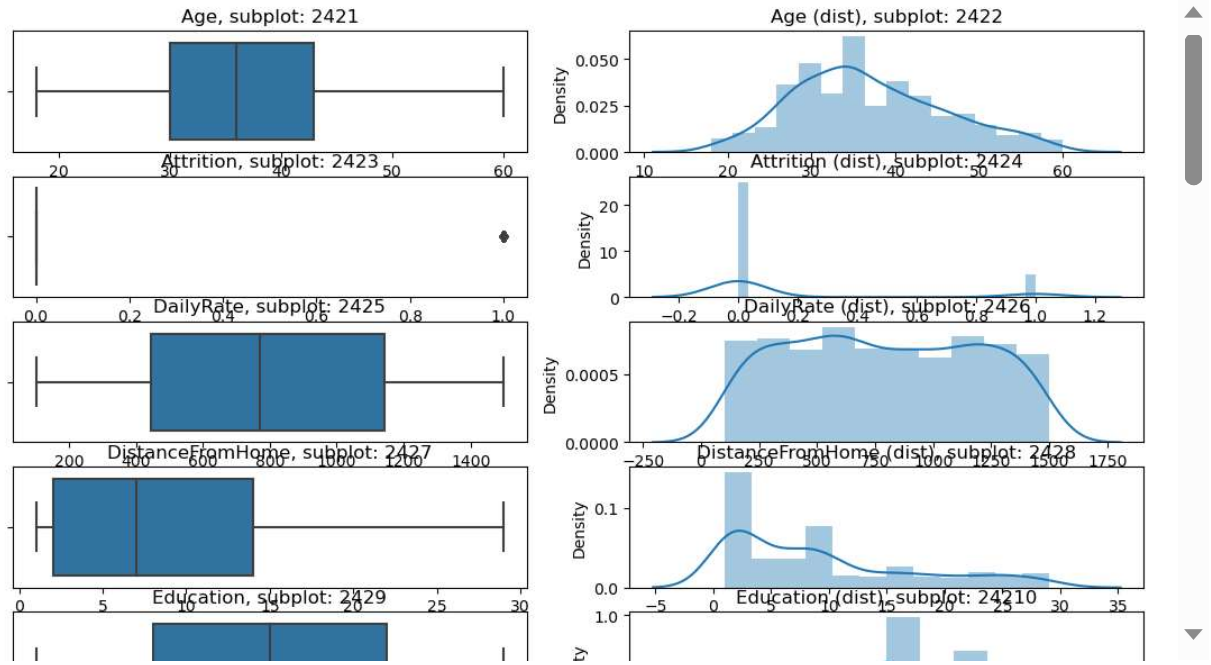
In [155]:
```
a = 24  # number of rows
b = 2  # number of columns
c = 1  # initialize plot counter

fig = plt.figure(figsize=(50,40))
fig.set_size_inches(13, 40)

for i in num_col:
    plt.subplot(a, b, c)
    plt.title('{}, subplot: {}{}{}'.format(i, a, b, c))
    plt.xlabel(i)
    sns.boxplot(data_num[i],orient="v",)
    c = c + 1

    plt.subplot(a, b, c)
    plt.title('{} (dist), subplot: {}{}{}'.format(i, a, b, c))
    plt.xlabel(i)
    sns.distplot(data_num[i])
    c = c + 1

plt.show();
```

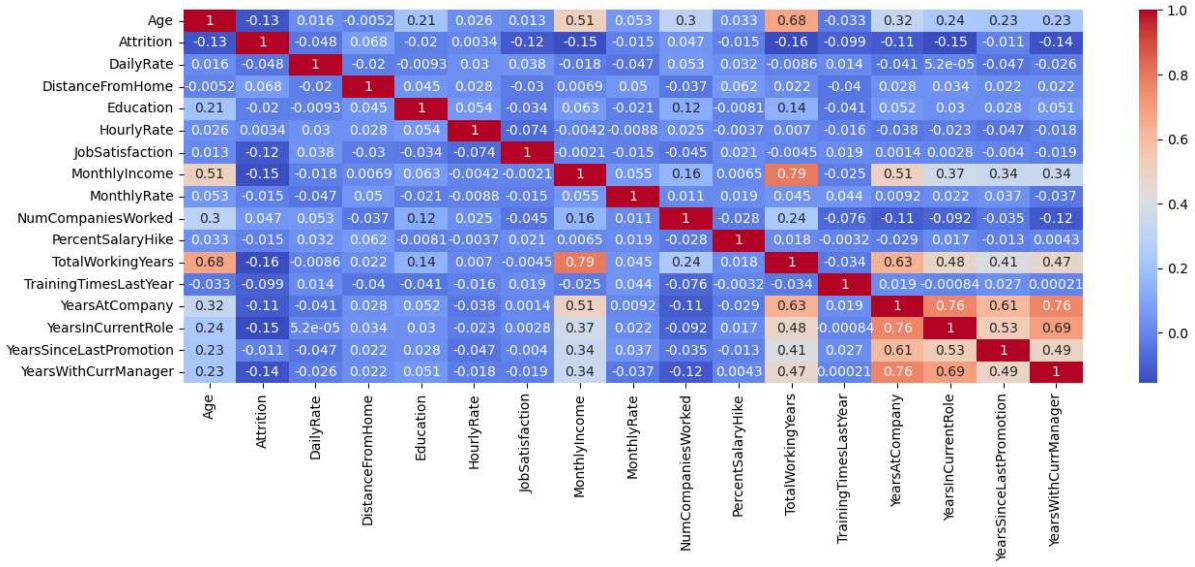



In [92]:
```
df.shape
```

Out[92]: (917, 32)

In [156]:
```
plt.figure(figsize=(15,5))
sns.heatmap(df.corr(),cmap='coolwarm',annot=True)
```

Out[156]: <AxesSubplot:>




In [157]:
```
cat_col = df.select_dtypes(include=np.object).columns.tolist()
print(cat_col)
print(type(cat_col))
```

```
['BusinessTravel', 'Department', 'EducationField', 'EnvironmentSatisfaction',
'Gender', 'JobInvolvement', 'JobLevel', 'JobRole', 'MaritalStatus', 'OverTim
e', 'PerformanceRating', 'RelationshipSatisfaction', 'StockOptionLevel', 'Wor
kLifeBalance']
<class 'list'>
```

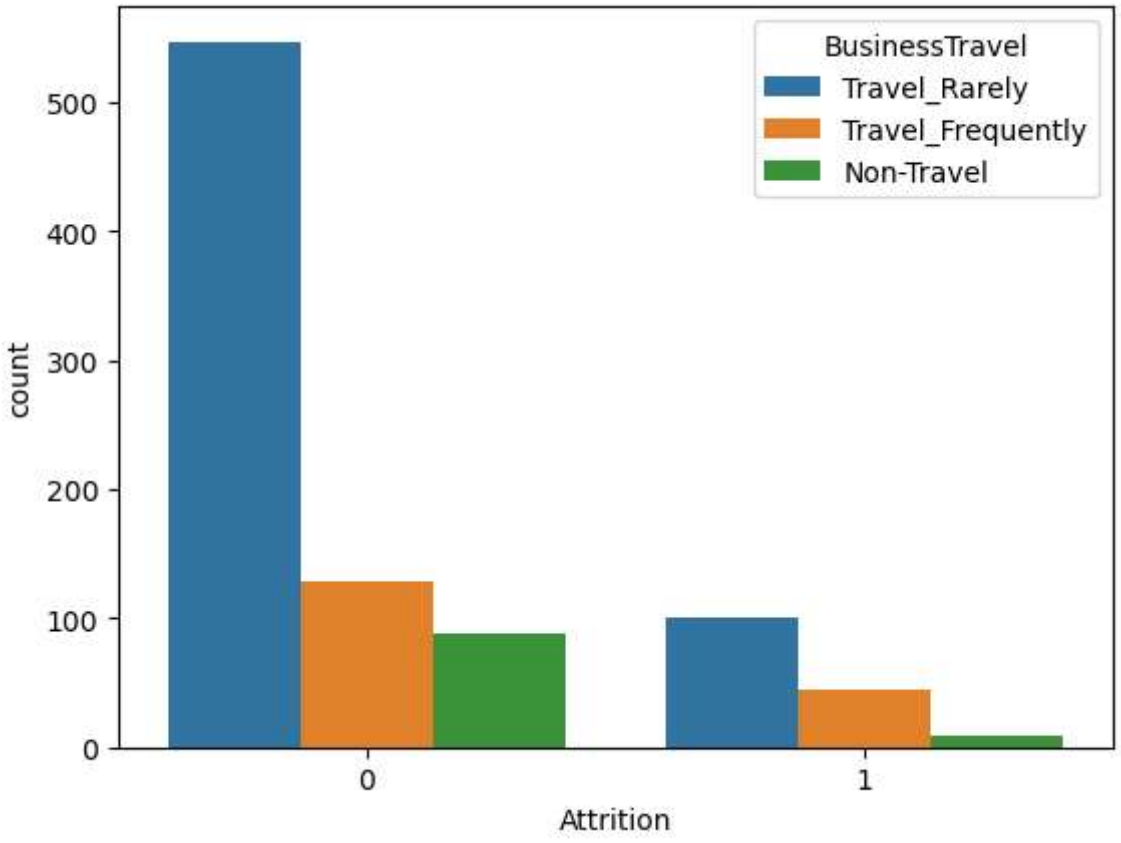
In [158]:
```
data_cat = df[cat_col]
data_cat.head()
```

Out[158]:

| | BusinessTravel | Department | EducationField | EnvironmentSatisfaction | Gender | JobInvolvemer |
|---|---|---|---|---|---|---|
| 0 | Travel_Rarely | Sales | Life Sciences | 2 | Female | |
| 1 | Travel_Frequently | Research & Development | Life Sciences | 3 | Male | |
| 2 | Travel_Rarely | Research & Development | Other | 4 | Male | |
| 3 | Travel_Frequently | Research & Development | Life Sciences | 4 | Female | |
| 4 | Travel_Rarely | Research & Development | Medical | 1 | Male | |

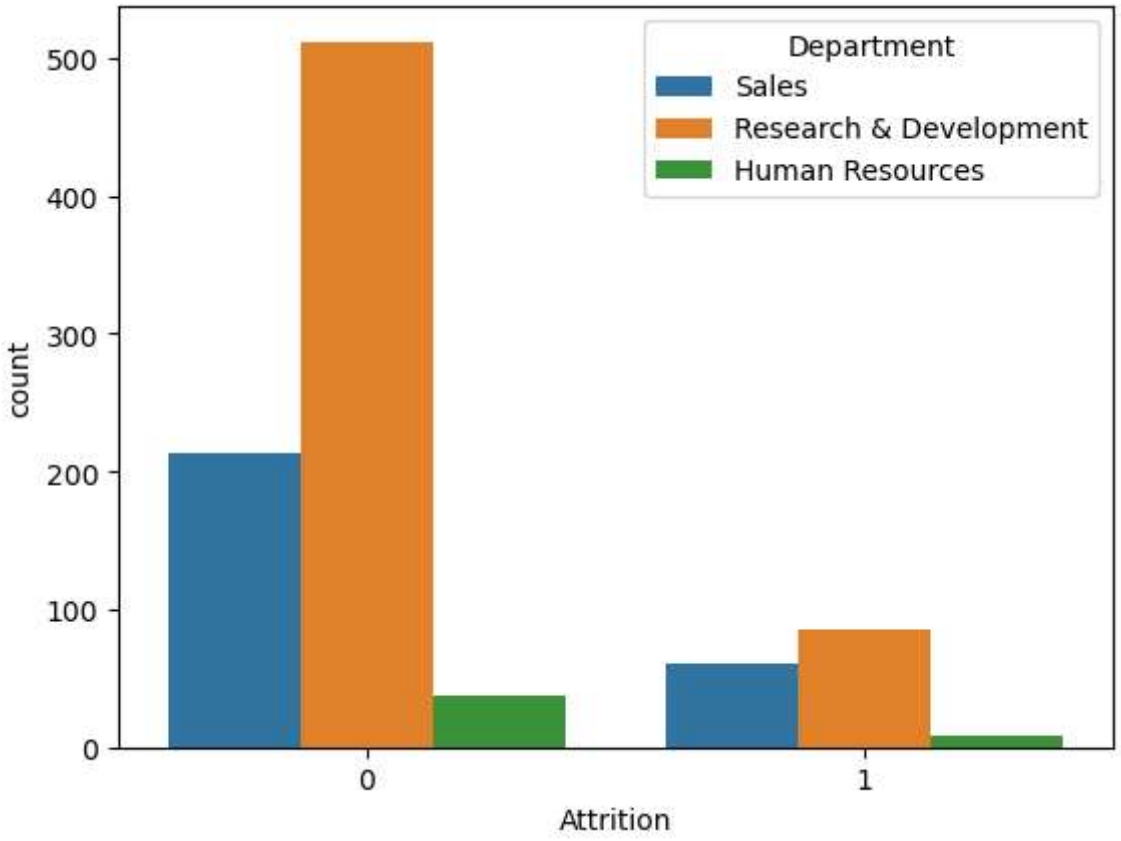
In [159]:
```
sns.countplot(data=df, x="Attrition", hue="BusinessTravel")
```

Out[159]:
```
<AxesSubplot:xlabel='Attrition', ylabel='count'>
```



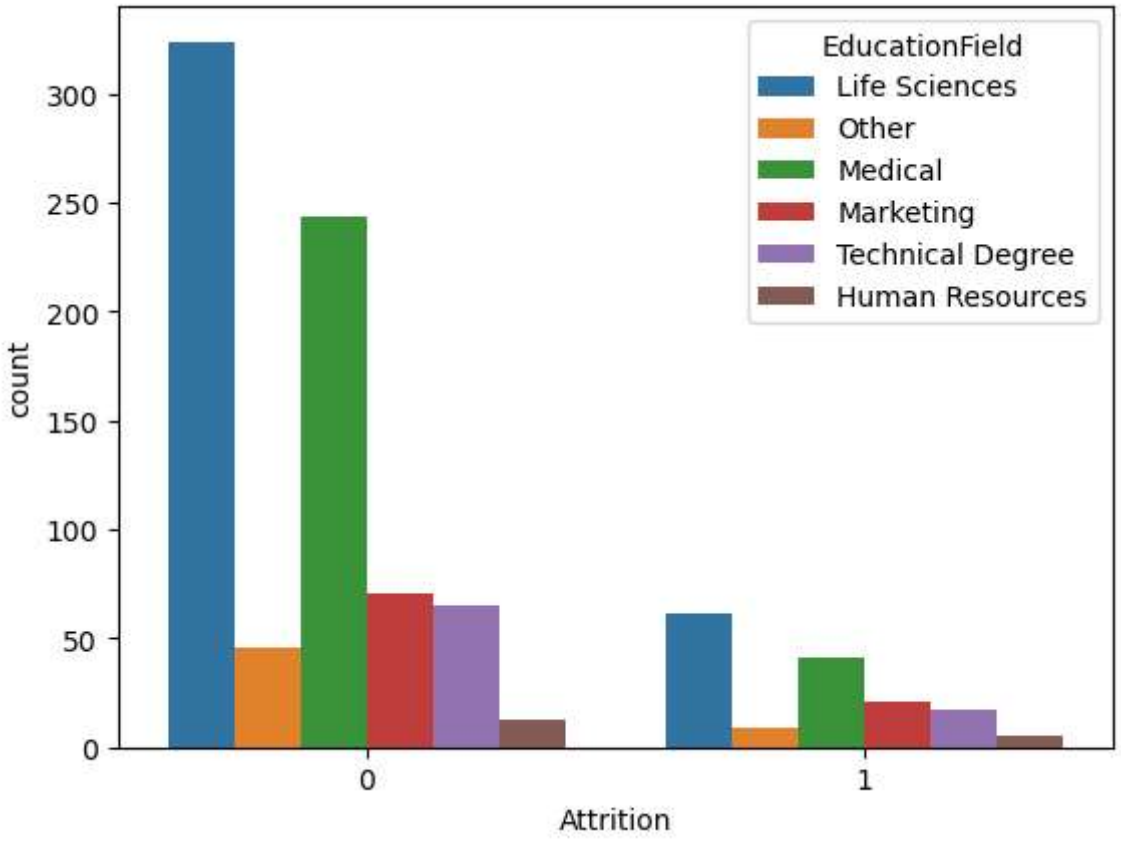
In [160]:
```
sns.countplot(data=df, x="Attrition", hue="Department")
```

Out[160]:
```
<AxesSubplot:xlabel='Attrition', ylabel='count'>
```



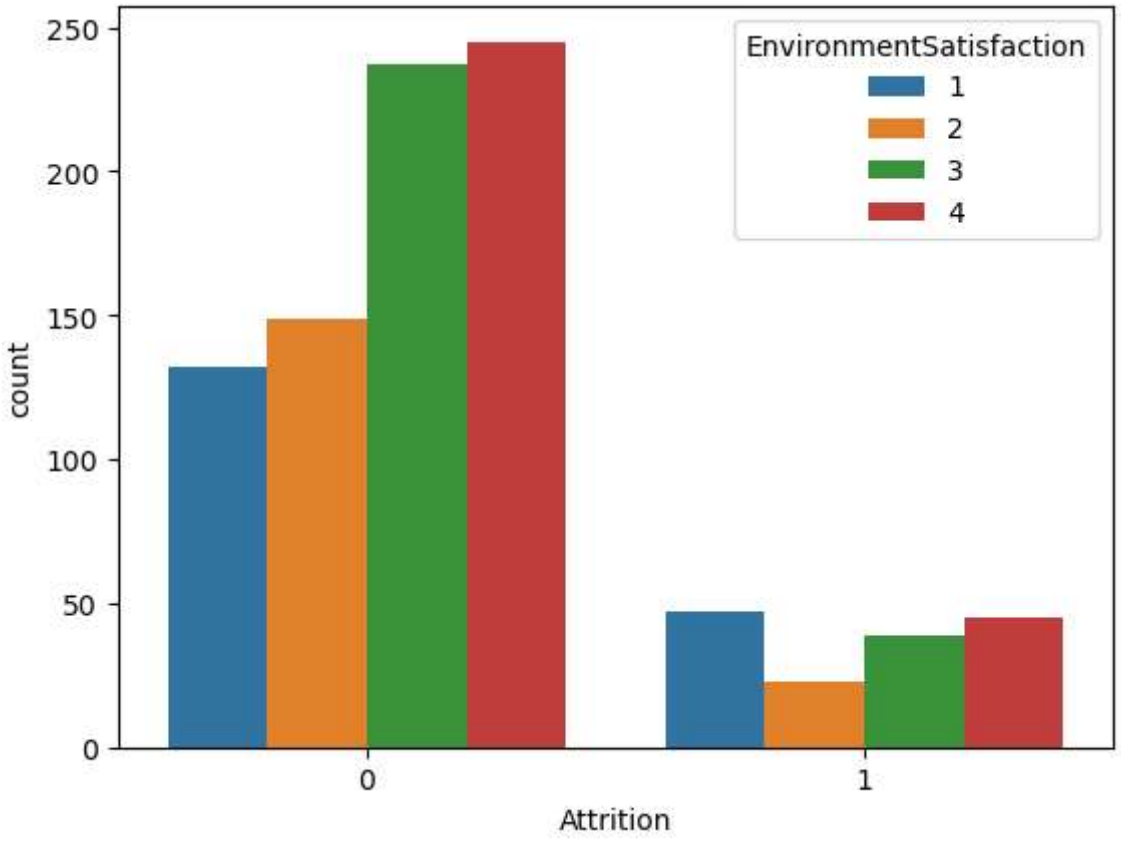
In [161]:
```
sns.countplot(data=df, x="Attrition", hue="EducationField")
```

Out[161]:
```
<AxesSubplot:xlabel='Attrition', ylabel='count'>
```



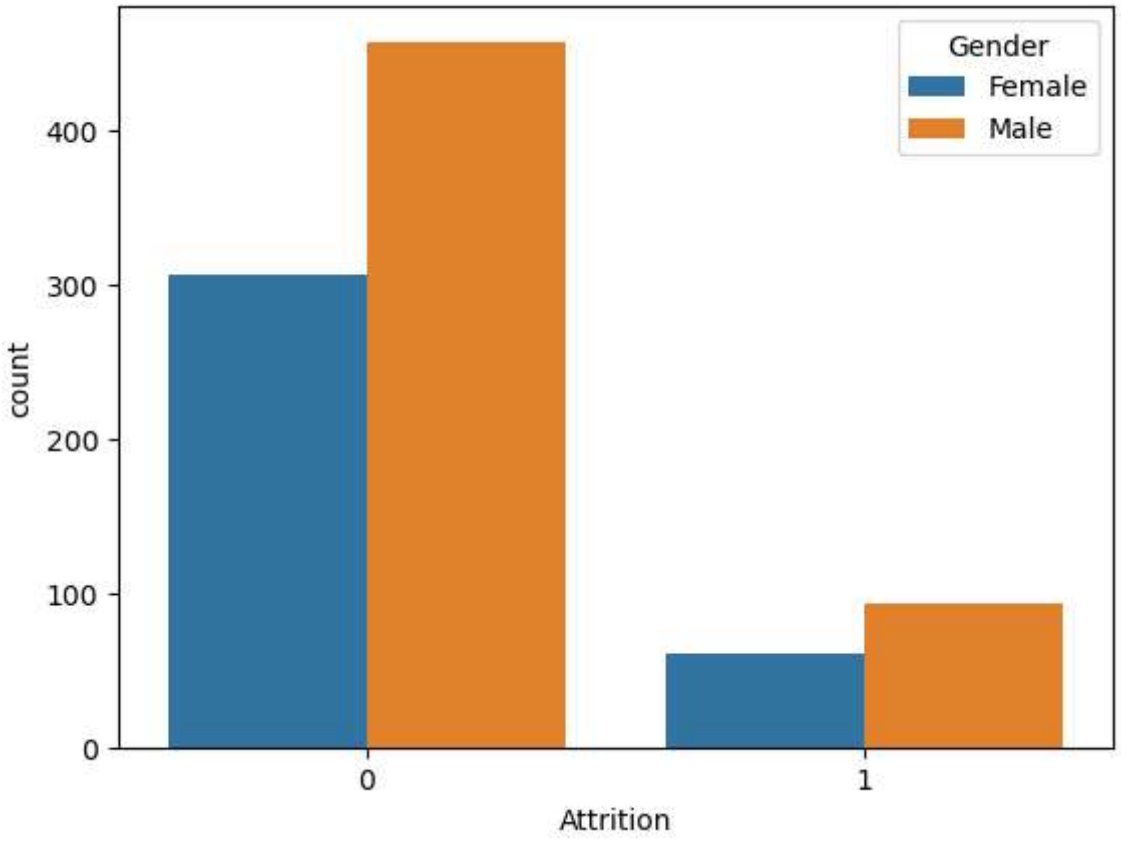
In [163]:
```
sns.countplot(data=df, x="Attrition", hue="EnvironmentSatisfaction")
```

Out[163]:
```
<AxesSubplot:xlabel='Attrition', ylabel='count'>
```



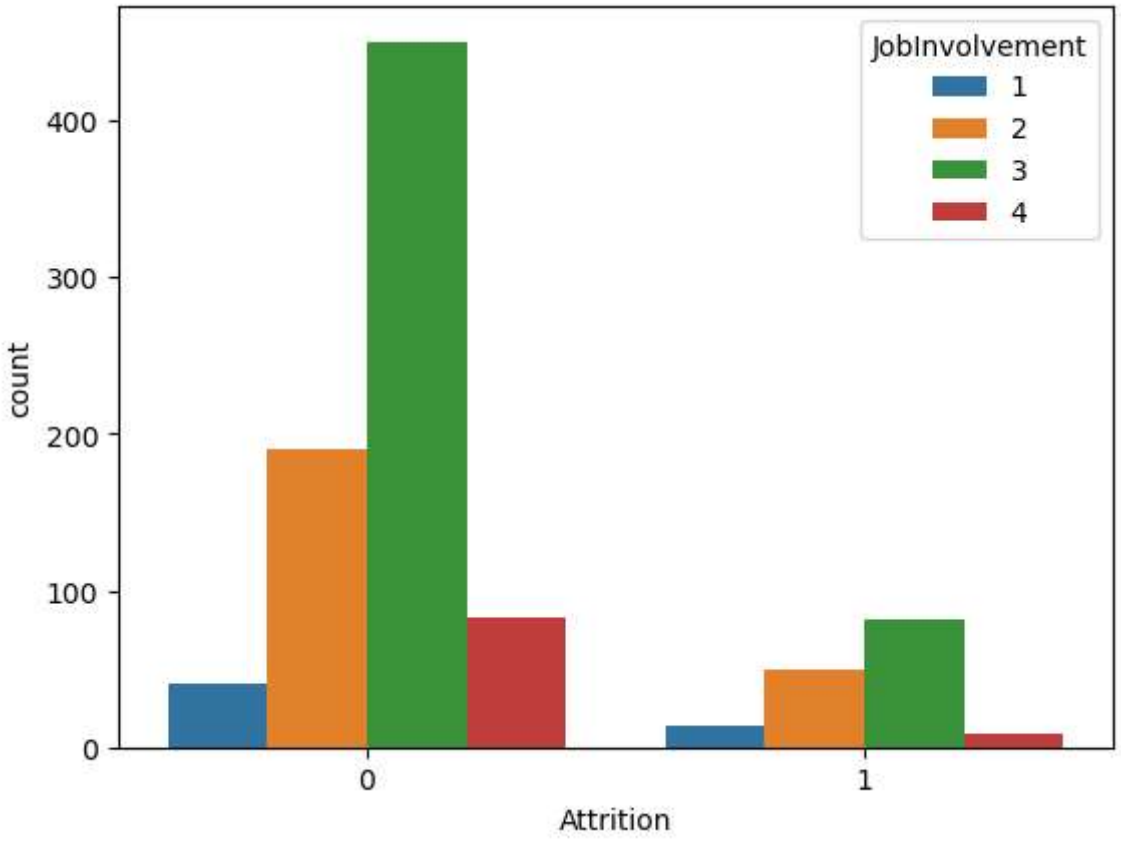
In [164]:
```
sns.countplot(data=df, x="Attrition", hue="Gender")
```

Out[164]:
```
<AxesSubplot:xlabel='Attrition', ylabel='count'>
```



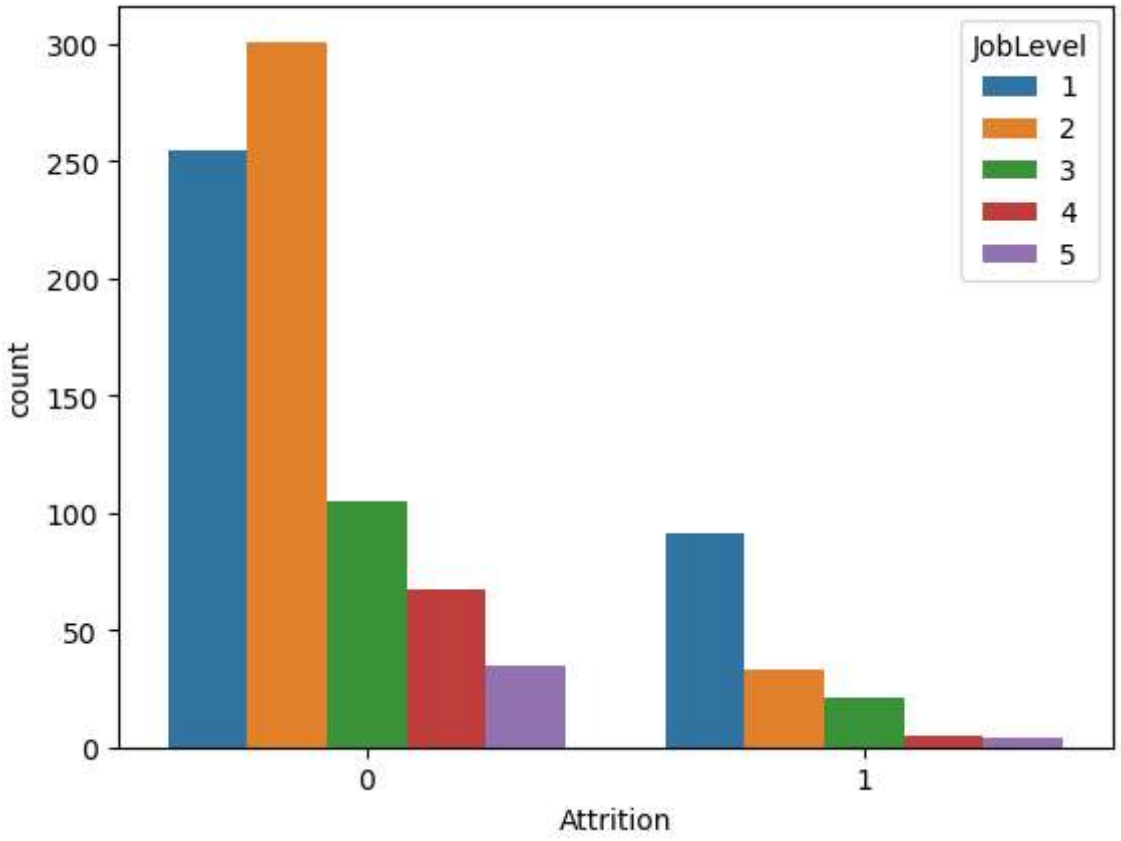
In [165]:
```
sns.countplot(data=df, x="Attrition", hue="JobInvolvement")
```

Out[165]:
```
<AxesSubplot:xlabel='Attrition', ylabel='count'>
```



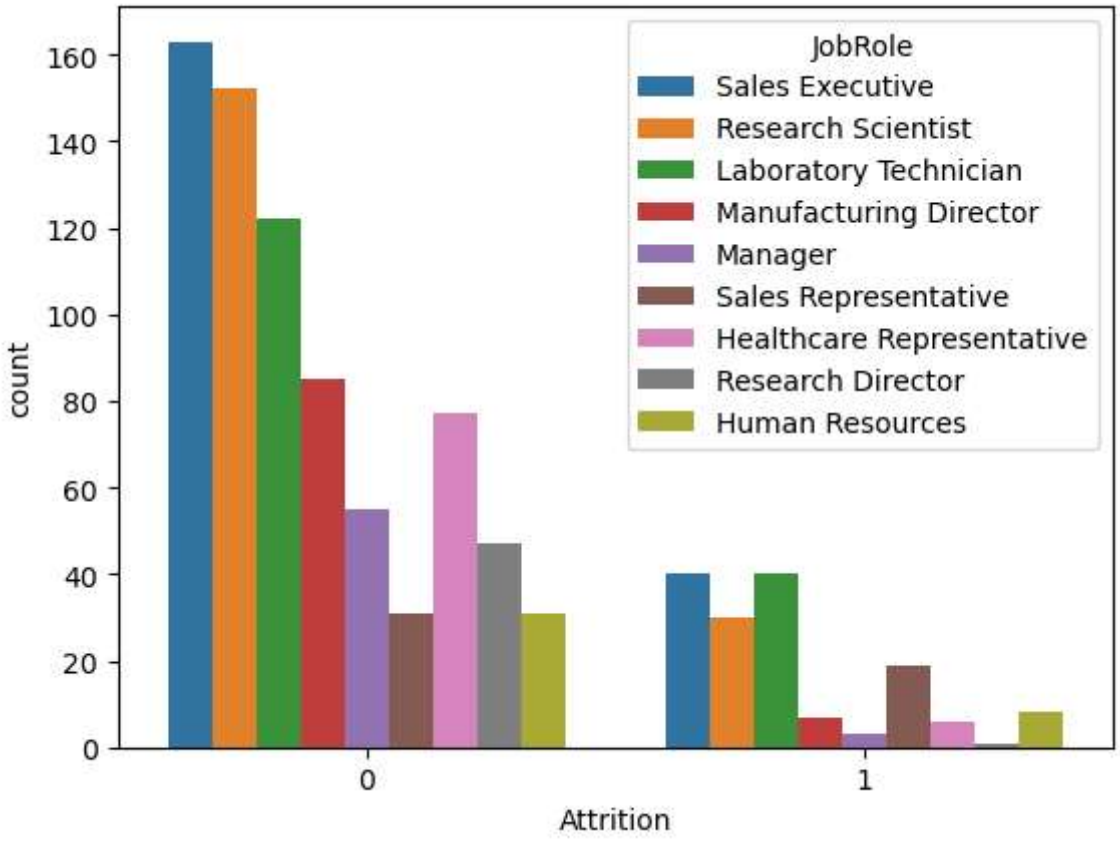
In [166]:
```
sns.countplot(data=df, x="Attrition", hue="JobLevel")
```

Out[166]:
```
<AxesSubplot:xlabel='Attrition', ylabel='count'>
```



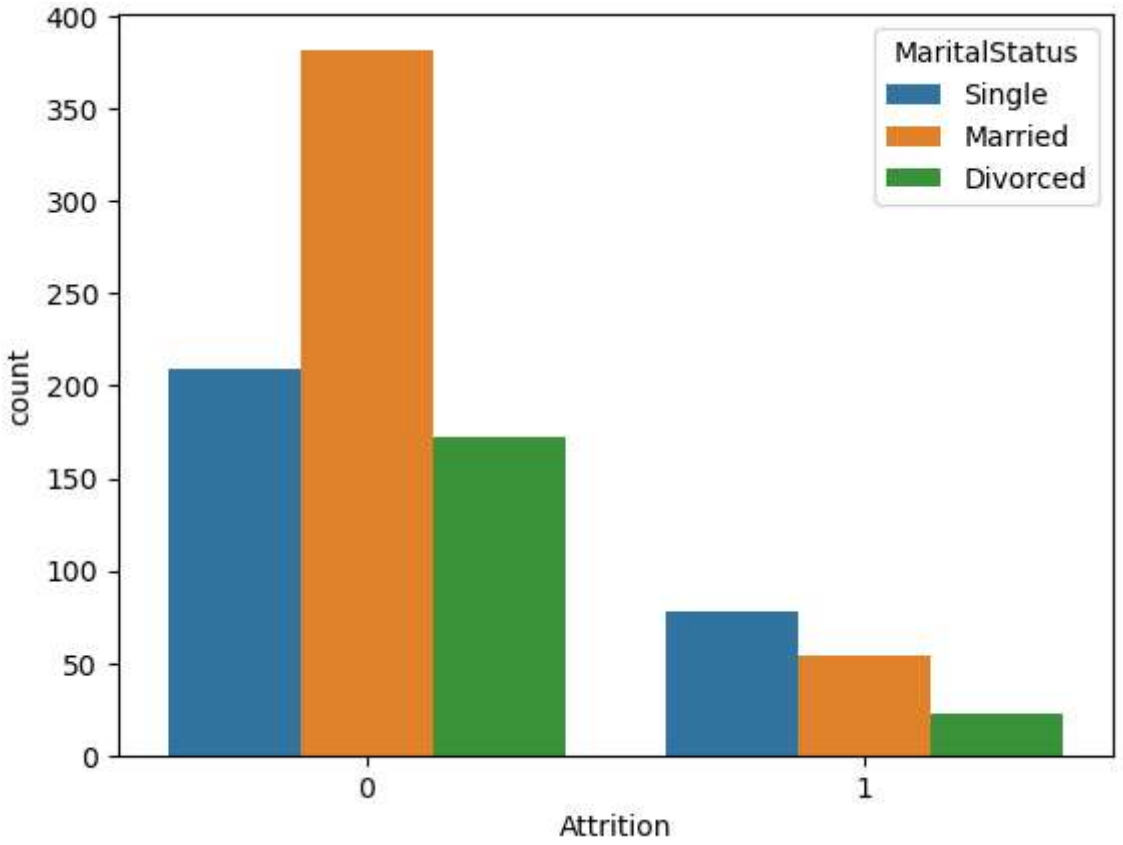
In [167]:
```
sns.countplot(data=df, x="Attrition", hue="JobRole")
```

Out[167]:
```
<AxesSubplot:xlabel='Attrition', ylabel='count'>
```



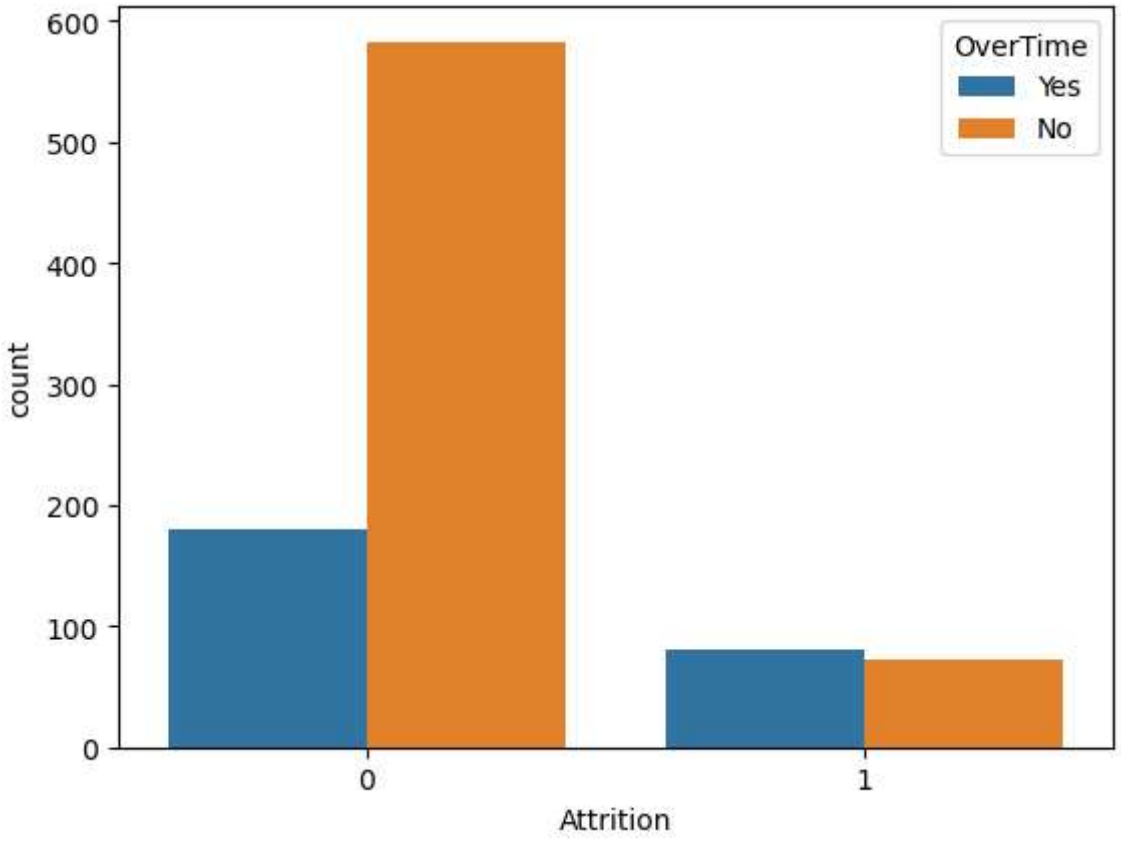
In [168]:
```
sns.countplot(data=df, x="Attrition", hue="MaritalStatus")
```

Out[168]:
```
<AxesSubplot:xlabel='Attrition', ylabel='count'>
```



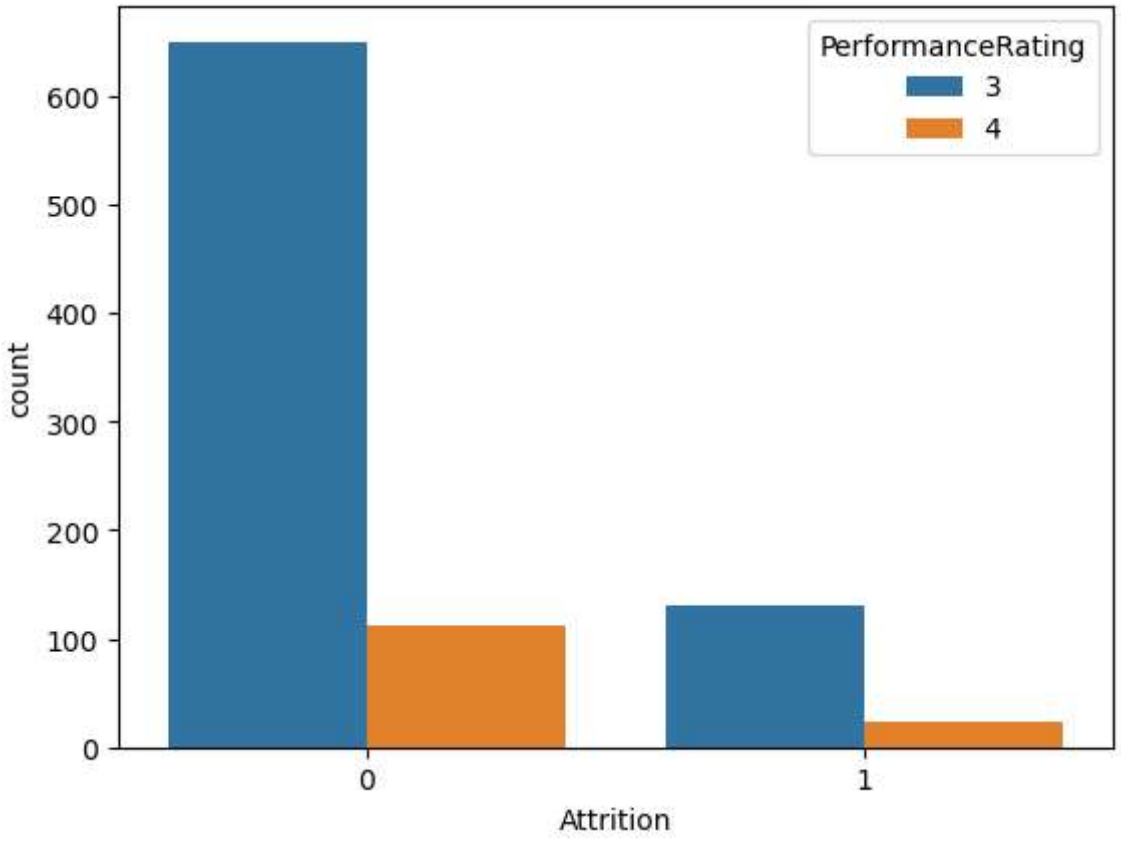
In [169]:
```
sns.countplot(data=df, x="Attrition", hue="OverTime")
```

Out[169]:
```
<AxesSubplot:xlabel='Attrition', ylabel='count'>
```



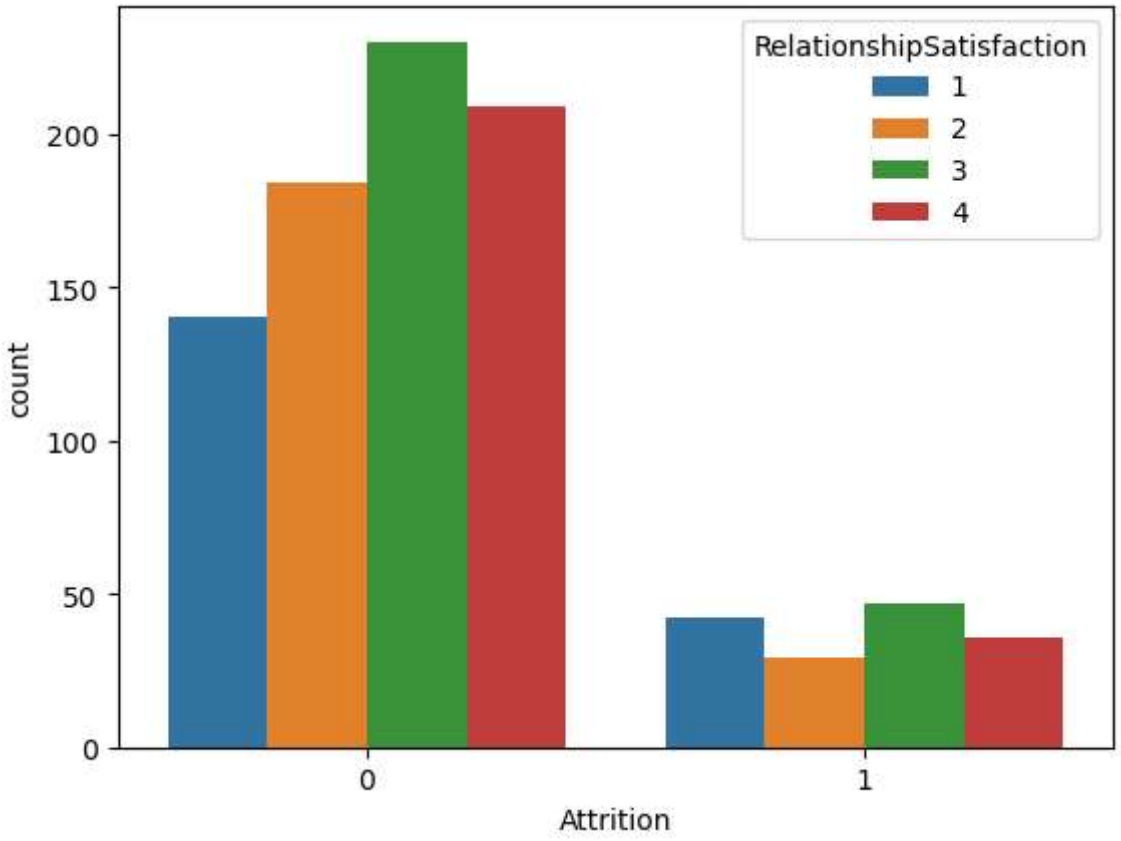
In [170]:
```
sns.countplot(data=df, x="Attrition", hue="PerformanceRating")
```

Out[170]:
```
<AxesSubplot:xlabel='Attrition', ylabel='count'>
```



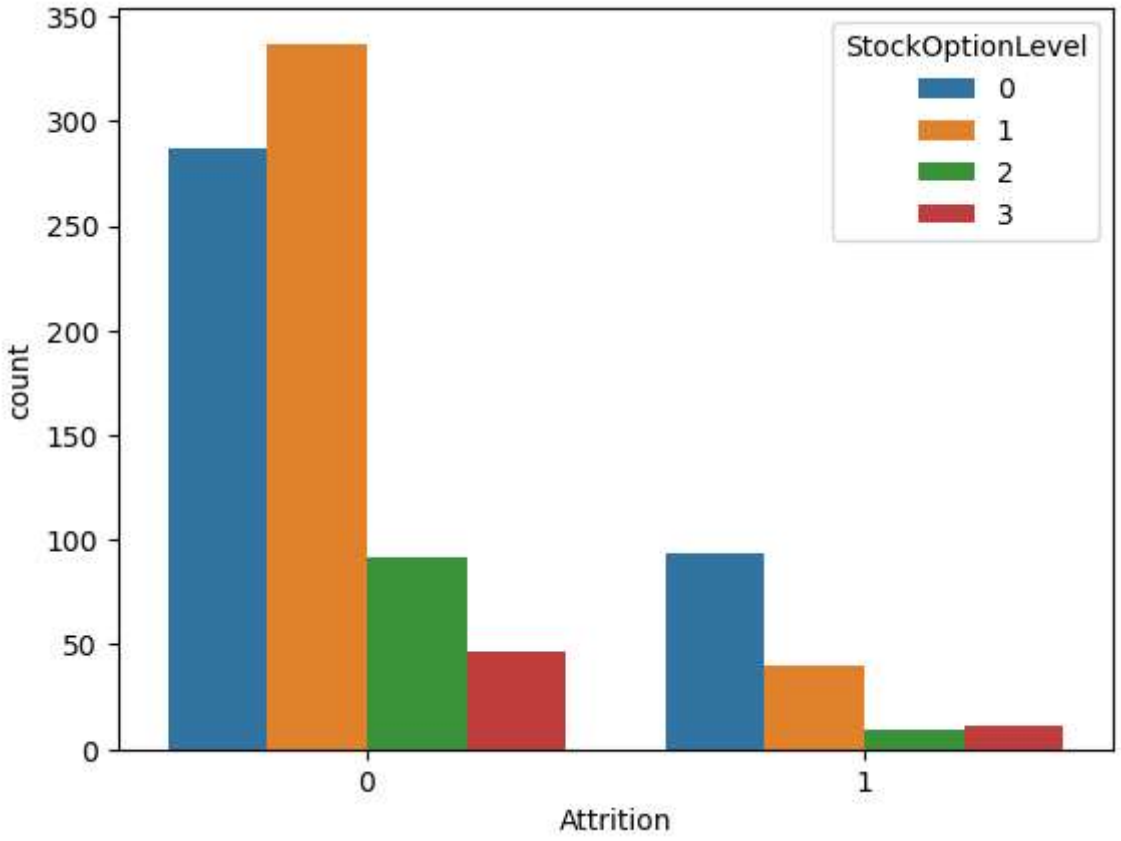
In [171]:
```
sns.countplot(data=df, x="Attrition", hue="RelationshipSatisfaction")
```

Out[171]:
```
<AxesSubplot:xlabel='Attrition', ylabel='count'>
```



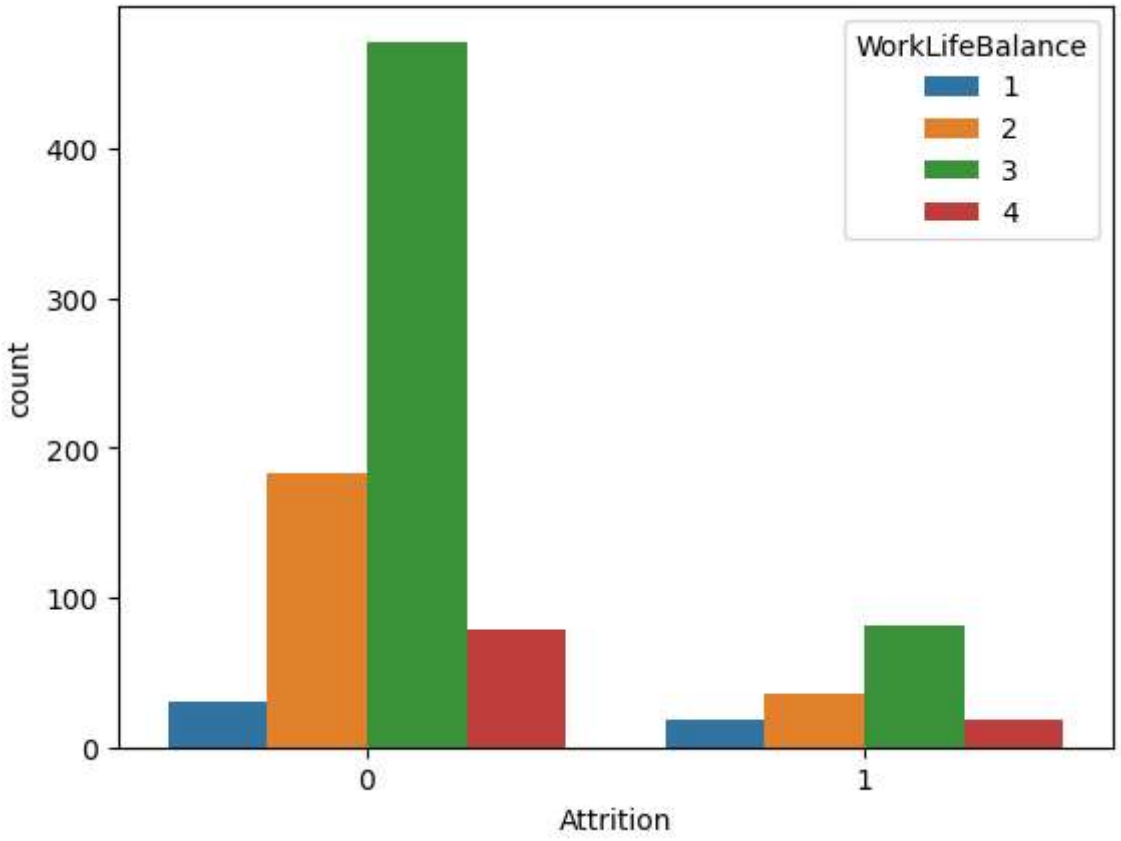
In [172]:
```
sns.countplot(data=df, x="Attrition", hue="StockOptionLevel")
```

Out[172]:
```
<AxesSubplot:xlabel='Attrition', ylabel='count'>
```



In [173]:
```
sns.countplot(data=df, x="Attrition", hue="WorkLifeBalance")
```

Out[173]: <AxesSubplot:xlabel='Attrition', ylabel='count'>




In [77]:
```
# Let us look at the target column which is 'Attrition' to understand how the d
df.groupby(["Attrition"]).mean()
```

Out[77]:

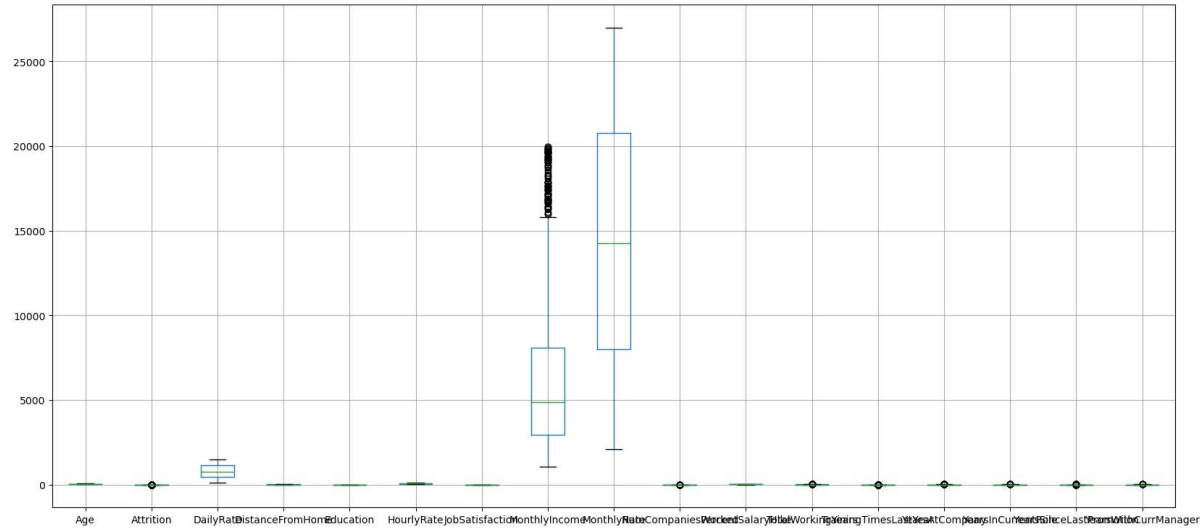
| Attrition | Age | DailyRate | DistanceFromHome | Education | HourlyRate | JobSatisfaction | Mon |
|---|---|---|---|---|---|---|---|
| No | 36.436681 | 799.710335 | 9.114993 | 2.925764 | 66.331878 | 2.761281 | 5 |
| Yes | 33.820000 | 744.086667 | 10.746667 | 2.880000 | 66.420000 | 2.400000 | 4 |

In [220]:
```
y = df['Attrition']

# store all the independent variables in a dataframe 'df_feature'
df_feature = df.drop('Attrition', axis = 1)
```

In [221]:
```
# use 'get_dummies' from pandas to create dummy variables
dummy_var = pd.get_dummies(data = data_cat, drop_first = True)
```

In [222]:
```
plt.figure(figsize=(20,9))
df.boxplot()
plt.show()
```




In [223]:
```
# initialize the standard scalar
X_scaler = StandardScaler()

# scale all the numerical columns
num_scaled = X_scaler.fit_transform(data_num)

# create a dataframe of scaled numerical variables
df_num_scaled = pd.DataFrame(num_scaled, columns = data_num.columns)
```

In [224]:
```
# concat the dummy variables with numeric features to create a dataframe of all
X = pd.concat([df_num_scaled, dummy_var], axis = 1)
X.head()
```

Out[224]:

| | Age | Attrition | DailyRate | DistanceFromHome | Education | HourlyRate | JobSatisfaction | M |
|---|---|---|---|---|---|---|---|---|
| **0** | 0.452231 | 2.225881 | 0.787410 | -1.025505 | -0.903273 | 1.372893 | 1.182422 | |
| **1** | 1.343091 | -0.449260 | -1.261470 | -0.160041 | -1.883511 | -0.258630 | -0.620206 | |
| **2** | 0.006800 | 2.225881 | 1.462071 | -0.901867 | -0.903273 | 1.274013 | 0.281108 | |
| **3** | -0.438630 | -0.449260 | 1.509372 | -0.778229 | 1.057203 | -0.505831 | 0.281108 | |
| **4** | -1.106775 | -0.449260 | -0.484738 | -0.901867 | -1.883511 | -1.296872 | -0.620206 | |

In [241]:
```
#Split the data into training and testing set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random
print('X_train', X_train.shape)
print('y_train', y_train.shape)
print('X_test', X_test.shape)
print('y_test', y_test.shape)
```

```
X_train (733, 59)
y_train (733,)
X_test (184, 59)
y_test (184,)
```

In [243]:
```
#Build the logistic regression model
logreg = LogisticRegression(max_iter=1000)
```

In [244]:
```
logreg.fit(X_train, y_train.values.ravel())
```

Out[244]:
```
LogisticRegression(max_iter=1000)
```

In [245]:
```
#Predict for test set
pred_test = logreg.predict(X_test)
```

In [246]:
```
pred_test
```

Out[246]:
```
array([0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 1, 1, 0, 0], dtype=int64)
```

In [256]:
```
#Predict for train set
pred_train = logreg.predict(X_train)
pred_train
```

Out[256]:
```
array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0], dtype=int64)
```

In [248]:
```
from sklearn.metrics import accuracy_score
test_accuracy=accuracy_score(y_test, pred_test)

print('Test Accuracy: ',np.round(test_accuracy,2))
```

```
Test Accuracy:  1.0
```

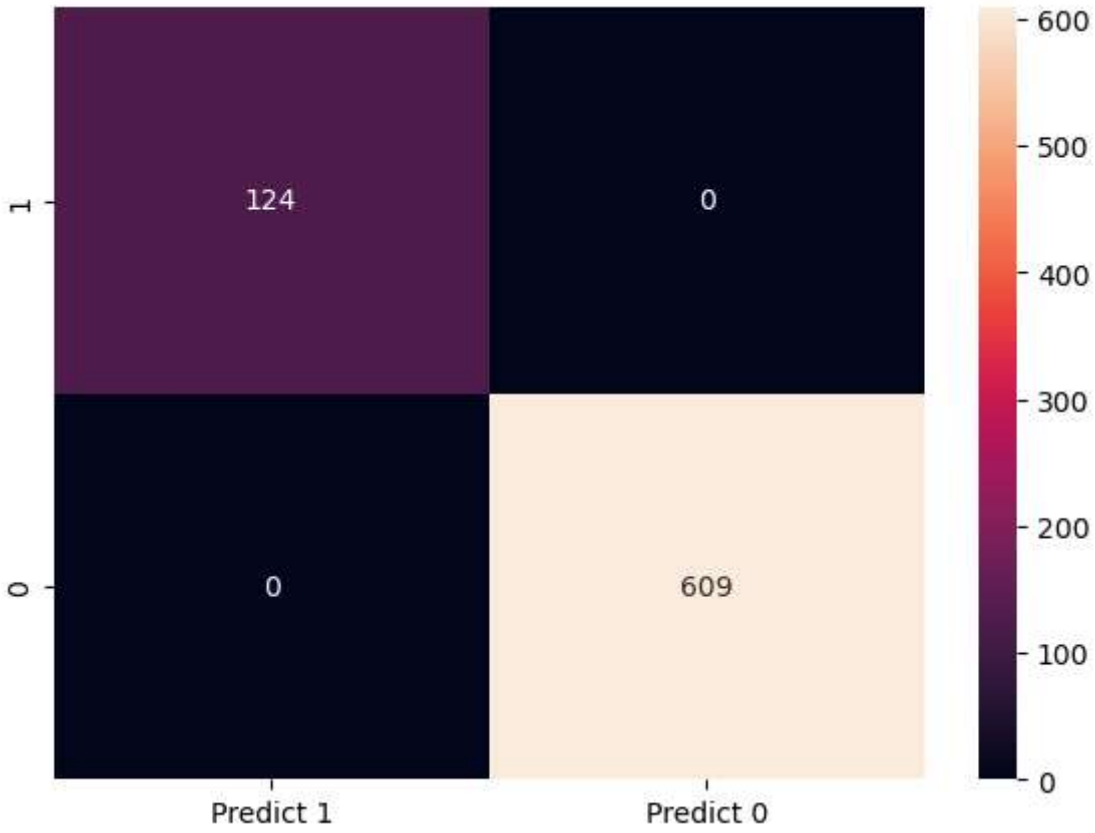
In [249]:
```
train_accuracy=accuracy_score(y_train, pred_train)

print('Train Accuracy: ',np.round(train_accuracy,2))
```

```
Train Accuracy:  1.0
```

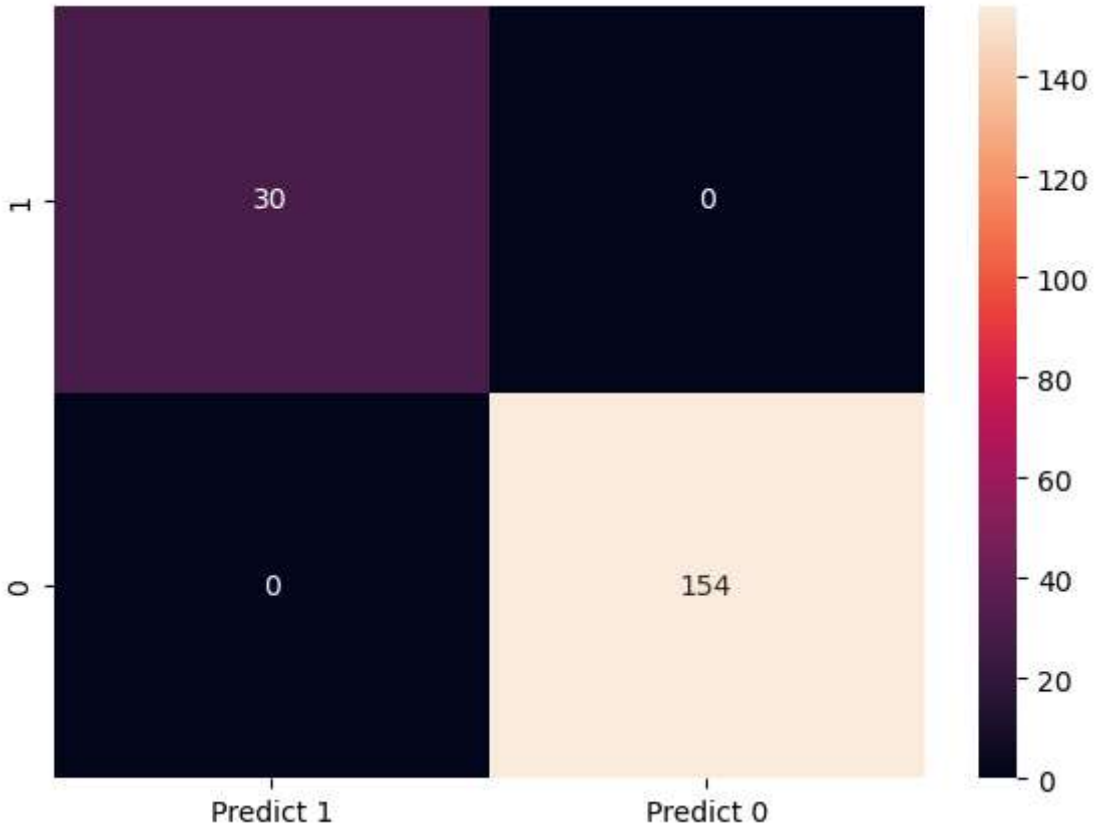
In [251]:
```
# cm=confusion_matrix(y_test, y_predict, labels=[1, 0])
cmat_train = confusion_matrix(y_train, pred_train, labels=[1, 0])
df_cmat_train = pd.DataFrame(cmat_train, index = [i for i in ["1", "0"]],
                             columns = [i for i in ["Predict 1","Predict 0"]] )

plt.figure(figsize = (7,5))
sns.heatmap(df_cmat_train, annot=True, fmt='g');
```



In [252]:
```
cmat_test = confusion_matrix(y_test, pred_test, labels=[1, 0])
df_cmat_test = pd.DataFrame(cmat_test, index = [i for i in ["1", "0"]],
                            columns = [i for i in ["Predict 1","Predict 0"]] )

plt.figure(figsize = (7,5))
sns.heatmap(df_cmat_test, annot=True, fmt='g');
```




In [253]:
```
print(classification_report(y_test, pred_test, digits=2))
```

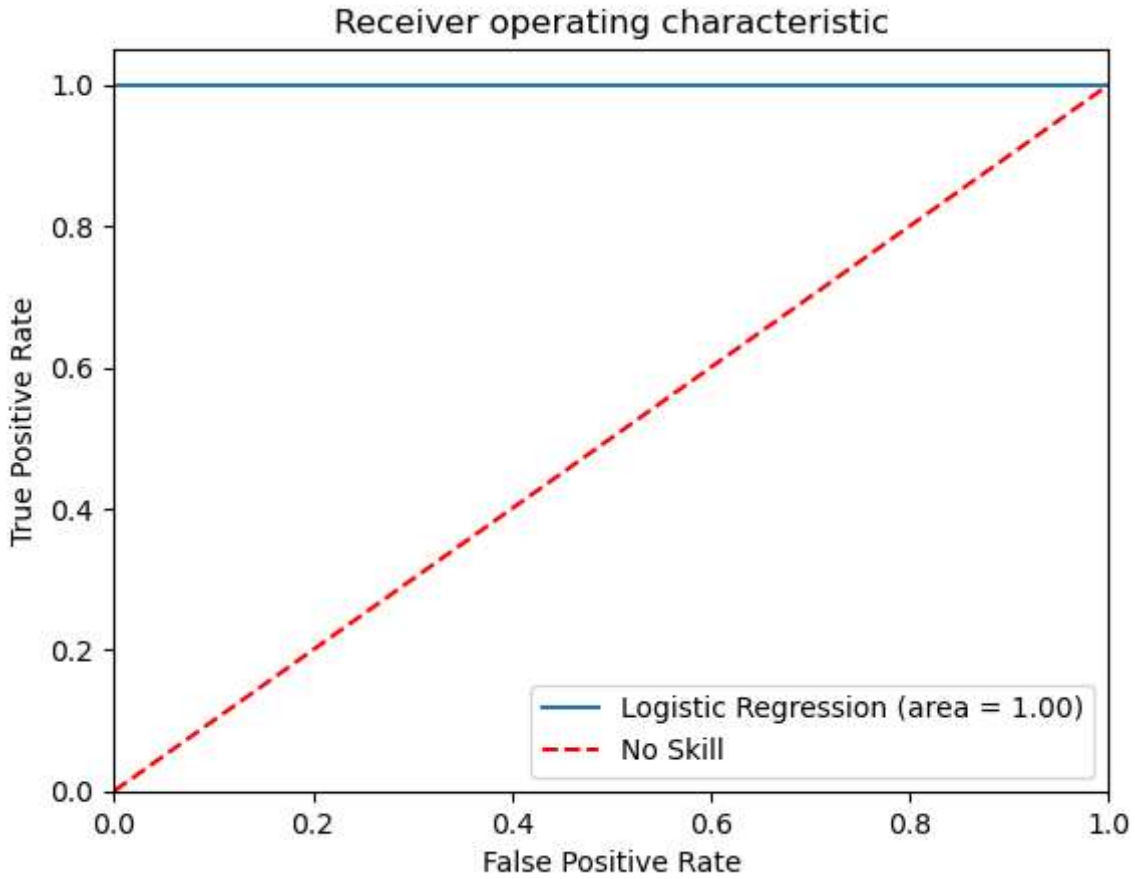
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       154
           1       1.00      1.00      1.00        30

    accuracy                           1.00       184
   macro avg       1.00      1.00      1.00       184
weighted avg       1.00      1.00      1.00       184
```

In [254]:
```
#AUC ROC curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve

logit_roc_auc = roc_auc_score(y_test, logreg.predict_proba(X_test)[:,1])
fpr, tpr, thresholds = roc_curve(y_test, logreg.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--',label='No Skill')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```



In [190]:
```
pip install xgboost
```

```
Collecting xgboostNote: you may need to restart the kernel to use updated pac
kages.

  Downloading xgboost-2.0.2-py3-none-win_amd64.whl (99.8 MB)
     ---------------------------------------- 99.8/99.8 MB 2.2 MB/s eta 0:00:
00
Requirement already satisfied: scipy in c:\users\satish kumar\anaconda3\lib\s
ite-packages (from xgboost) (1.9.1)
Requirement already satisfied: numpy in c:\users\satish kumar\anaconda3\lib\s
ite-packages (from xgboost) (1.21.5)
Installing collected packages: xgboost
Successfully installed xgboost-2.0.2
```

In [203]:
```
from xgboost import XGBClassifier
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
```

XGBoost classifier

In [258]:
```
xgb_clf=XGBClassifier(max_depth=10, gamma=1)
xgb_clf.fit(X_train,y_train)
```

Out[258]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=1, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=10, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

In [260]:
```
# make predictions on test data
y_pred = xgb_clf.predict(X_test)
y_pred
```

Out[260]:
```
array([0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 1, 1, 0, 0])
```

In [261]:
```
# check accuracy score
from sklearn.metrics import accuracy_score

print('XGBoost model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y
```

```
XGBoost model accuracy score: 1.0000
```

k-fold Cross Validation using XGBoost

In [271]:
```
# define data_dmatrix
data_dmatrix = xgb.DMatrix(data=X,label=y)
```

In [272]:
```
from xgboost import cv

params = {"objective":"binary:logistic",'colsample_bytree': 0.3,'learning_rate'
                'max_depth': 5, 'alpha': 10}

xgb_cv = cv(dtrain=data_dmatrix, params=params, nfold=3,
                    num_boost_round=50, early_stopping_rounds=10, metrics="auc"
```

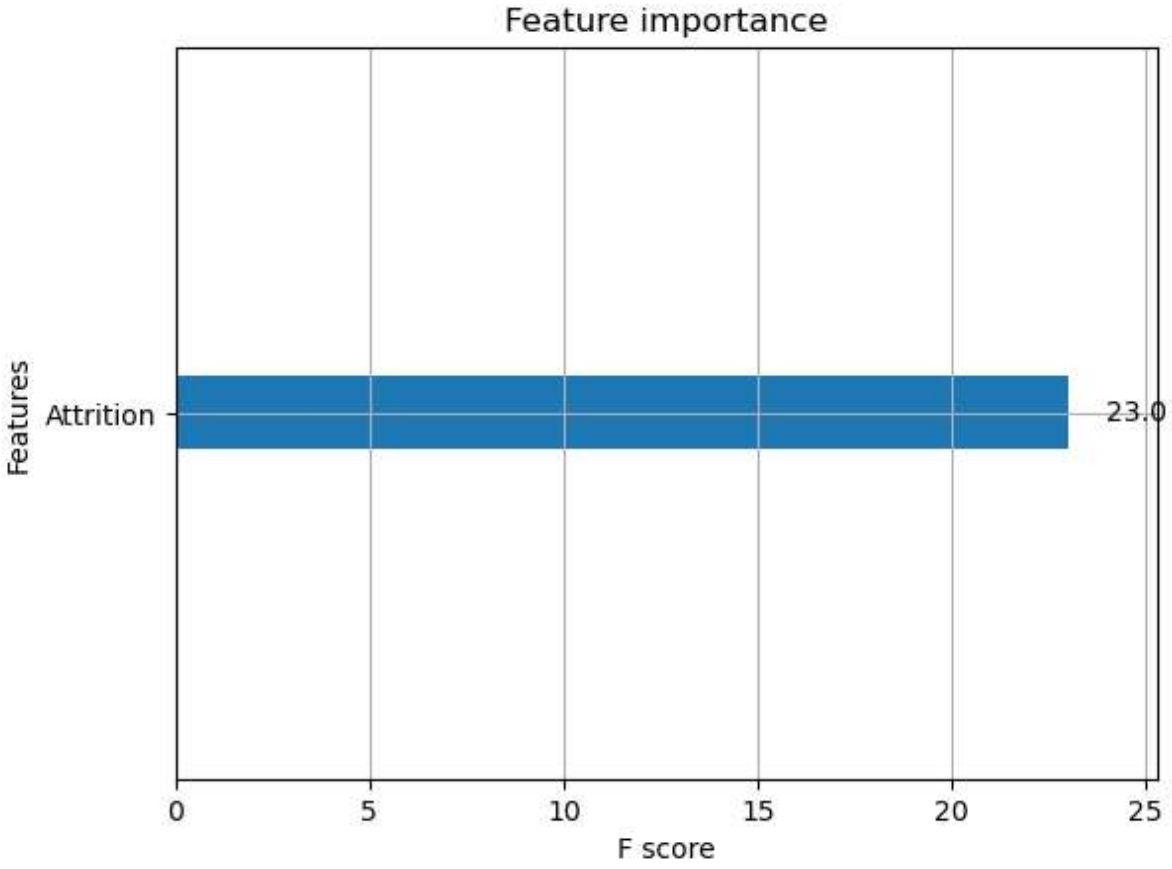
In [273]:
```
xgb_cv.head()
```

Out[273]:

| | train-auc-mean | train-auc-std | test-auc-mean | test-auc-std |
|---|---|---|---|---|
| **0** | 0.710926 | 0.002222 | 0.662896 | 0.027241 |
| **1** | 0.758518 | 0.019567 | 0.689866 | 0.024505 |
| **2** | 0.803171 | 0.005492 | 0.716769 | 0.026905 |
| **3** | 0.829372 | 0.003095 | 0.744656 | 0.028875 |
| **4** | 0.831312 | 0.008112 | 0.748912 | 0.033557 |

Feature importance with XGBoost

In [274]:
```
xgb.plot_importance(xgb_clf)
plt.rcParams['figure.figsize'] = [6, 4]
plt.show()
```




The data is unbalanced, so we have to use different sampling methods to balance the data.