In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as pairplot
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
df=pd.read_csv('ibmhrdata.csv')  #Reading the datafiles and loading the datasets in jupyter notebook.
df
```

Out[2]:

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

In [3]:

```
df.head()
```

Out[3]:

| | Age | Attrition | BusinessTravel | DailyRate | Department | DistanceFromHome | Education | E |
|---|---|---|---|---|---|---|---|---|
| **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 | |

5 rows × 35 columns

In [4]:

```
df.shape #finding out the shape of the datasets
```

Out[4]:

```
(1470, 35)
```

In [5]:

```
df.dtypes #finding out the datatypes for each columns.
```

Out[5]:

```
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 [6]:

```
df.isna().sum() #finding out if their any null values in any columns in the datasets.
```

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 [8]:

```
df.isnull().values
```

Out[8]:

```
array([[False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       ...,
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False]])
```

In [9]:

```
df.describe()
```

Out[9]:

| | Age | DailyRate | DistanceFromHome | Education | EmployeeCount | Employe |
|---|---|---|---|---|---|---|
| **count** | 1470.000000 | 1470.000000 | 1470.000000 | 1470.000000 | 1470.0 | 147 |
| **mean** | 36.923810 | 802.485714 | 9.192517 | 2.912925 | 1.0 | 102 |
| **std** | 9.135373 | 403.509100 | 8.106864 | 1.024165 | 0.0 | 60 |
| **min** | 18.000000 | 102.000000 | 1.000000 | 1.000000 | 1.0 | |
| **25%** | 30.000000 | 465.000000 | 2.000000 | 2.000000 | 1.0 | 49 |
| **50%** | 36.000000 | 802.000000 | 7.000000 | 3.000000 | 1.0 | 102 |
| **75%** | 43.000000 | 1157.000000 | 14.000000 | 4.000000 | 1.0 | 155 |
| **max** | 60.000000 | 1499.000000 | 29.000000 | 5.000000 | 1.0 | 206 |

8 rows × 26 columns

In [11]:

```
df['Attrition'].value_counts()  #finding out the value of attrition.
```

Out[11]:

```
No     1233
Yes     237
Name: Attrition, dtype: int64
```

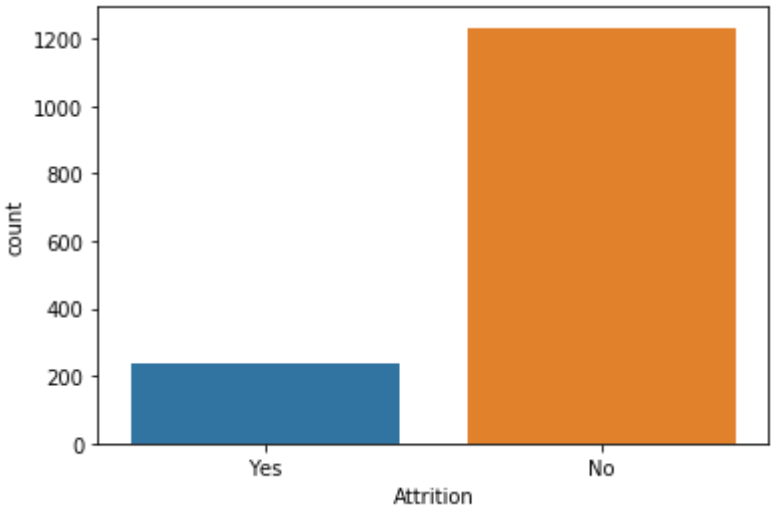
Data Visulization process

In [12]:

```
sns.countplot(df['Attrition']) #using countplot we are getting no of obeseration for at
trition columns.
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24f01d1b548>
```

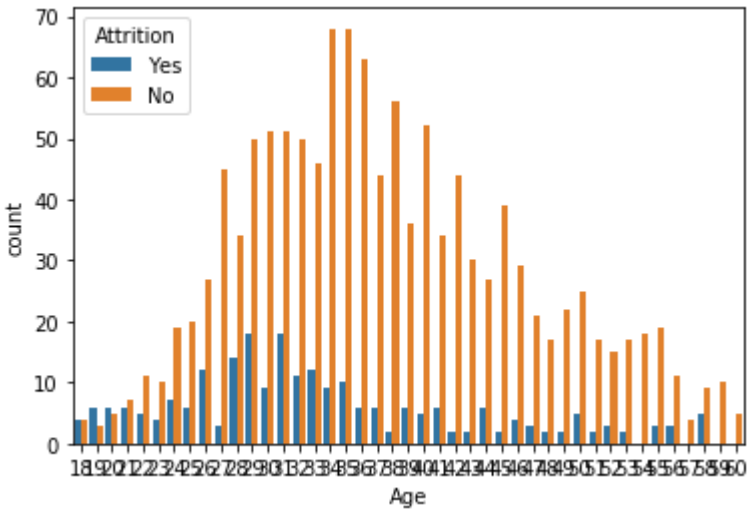



In [13]:

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

Out[13]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24f0249cb88>
```



In [14]:

```
for i in df.columns:
    if df[i].dtypes== object:
        print(str(i) + ': '+ str(df[i].unique()) )
        print(df[i].value_counts())

        print(' ')
```

```
Attrition: ['Yes' 'No']
No     1233
Yes     237
Name: Attrition, dtype: int64

BusinessTravel: ['Travel_Rarely' 'Travel_Frequently' 'Non-Travel']
Travel_Rarely        1043
Travel_Frequently     277
Non-Travel            150
Name: BusinessTravel, dtype: int64

Department: ['Sales' 'Research & Development' 'Human Resources']
Research & Development    961
Sales                     446
Human Resources            63
Name: Department, dtype: int64

EducationField: ['Life Sciences' 'Other' 'Medical' 'Marketing' 'Technical
Degree'
 'Human Resources']
Life Sciences       606
Medical             464
Marketing           159
Technical Degree    132
Other                82
Human Resources      27
Name: EducationField, dtype: int64

Gender: ['Female' 'Male']
Male      882
Female    588
Name: Gender, dtype: int64

JobRole: ['Sales Executive' 'Research Scientist' 'Laboratory Technician'
 'Manufacturing Director' 'Healthcare Representative' 'Manager'
 'Sales Representative' 'Research Director' 'Human Resources']
Sales Executive              326
Research Scientist           292
Laboratory Technician        259
Manufacturing Director       145
Healthcare Representative    131
Manager                      102
Sales Representative          83
Research Director             80
Human Resources               52
Name: JobRole, dtype: int64

MaritalStatus: ['Single' 'Married' 'Divorced']
Married     673
Single      470
Divorced    327
Name: MaritalStatus, dtype: int64

Over18: ['Y']
Y    1470
Name: Over18, dtype: int64

OverTime: ['Yes' 'No']
No     1054
Yes     416
```

```
Name: OverTime, dtype: int64
```

In [15]:

```
df['StandardHours'].unique()
```

Out[15]:

```
array([80], dtype=int64)
```

In [16]:

```
df['EmployeeCount'].unique()
```

Out[16]:

```
array([1], dtype=int64)
```

In [18]:

```
df=df.drop('Over18',axis=1)  #removing the unnecessary columns.
df=df.drop('EmployeeCount',axis=1)
df=df.drop('StandardHours',axis=1)
df=df.drop('EmployeeNumber',axis=1)
```

In [19]:

```
df.corr() #finding the correlation between the variables inside the dataset.
```

Out[19]:

| | Age | DailyRate | DistanceFromHome | Education | EnvironmentSa |
|---|---|---|---|---|---|
| **Age** | 1.000000 | 0.010661 | -0.001686 | 0.208034 | |
| **DailyRate** | 0.010661 | 1.000000 | -0.004985 | -0.016806 | |
| **DistanceFromHome** | -0.001686 | -0.004985 | 1.000000 | 0.021042 | |
| **Education** | 0.208034 | -0.016806 | 0.021042 | 1.000000 | |
| **EnvironmentSatisfaction** | 0.010146 | 0.018355 | -0.016075 | -0.027128 | |
| **HourlyRate** | 0.024287 | 0.023381 | 0.031131 | 0.016775 | |
| **JobInvolvement** | 0.029820 | 0.046135 | 0.008783 | 0.042438 | |
| **JobLevel** | 0.509604 | 0.002966 | 0.005303 | 0.101589 | |
| **JobSatisfaction** | -0.004892 | 0.030571 | -0.003669 | -0.011296 | |
| **MonthlyIncome** | 0.497855 | 0.007707 | -0.017014 | 0.094961 | |
| **MonthlyRate** | 0.028051 | -0.032182 | 0.027473 | -0.026084 | |
| **NumCompaniesWorked** | 0.299635 | 0.038153 | -0.029251 | 0.126317 | |
| **PercentSalaryHike** | 0.003634 | 0.022704 | 0.040235 | -0.011111 | |
| **PerformanceRating** | 0.001904 | 0.000473 | 0.027110 | -0.024539 | |
| **RelationshipSatisfaction** | 0.053535 | 0.007846 | 0.006557 | -0.009118 | |
| **StockOptionLevel** | 0.037510 | 0.042143 | 0.044872 | 0.018422 | |
| **TotalWorkingYears** | 0.680381 | 0.014515 | 0.004628 | 0.148280 | |
| **TrainingTimesLastYear** | -0.019621 | 0.002453 | -0.036942 | -0.025100 | |
| **WorkLifeBalance** | -0.021490 | -0.037848 | -0.026556 | 0.009819 | |
| **YearsAtCompany** | 0.311309 | -0.034055 | 0.009508 | 0.069114 | |
| **YearsInCurrentRole** | 0.212901 | 0.009932 | 0.018845 | 0.060236 | |
| **YearsSinceLastPromotion** | 0.216513 | -0.033229 | 0.010029 | 0.054254 | |
| **YearsWithCurrManager** | 0.202089 | -0.026363 | 0.014406 | 0.069065 | |

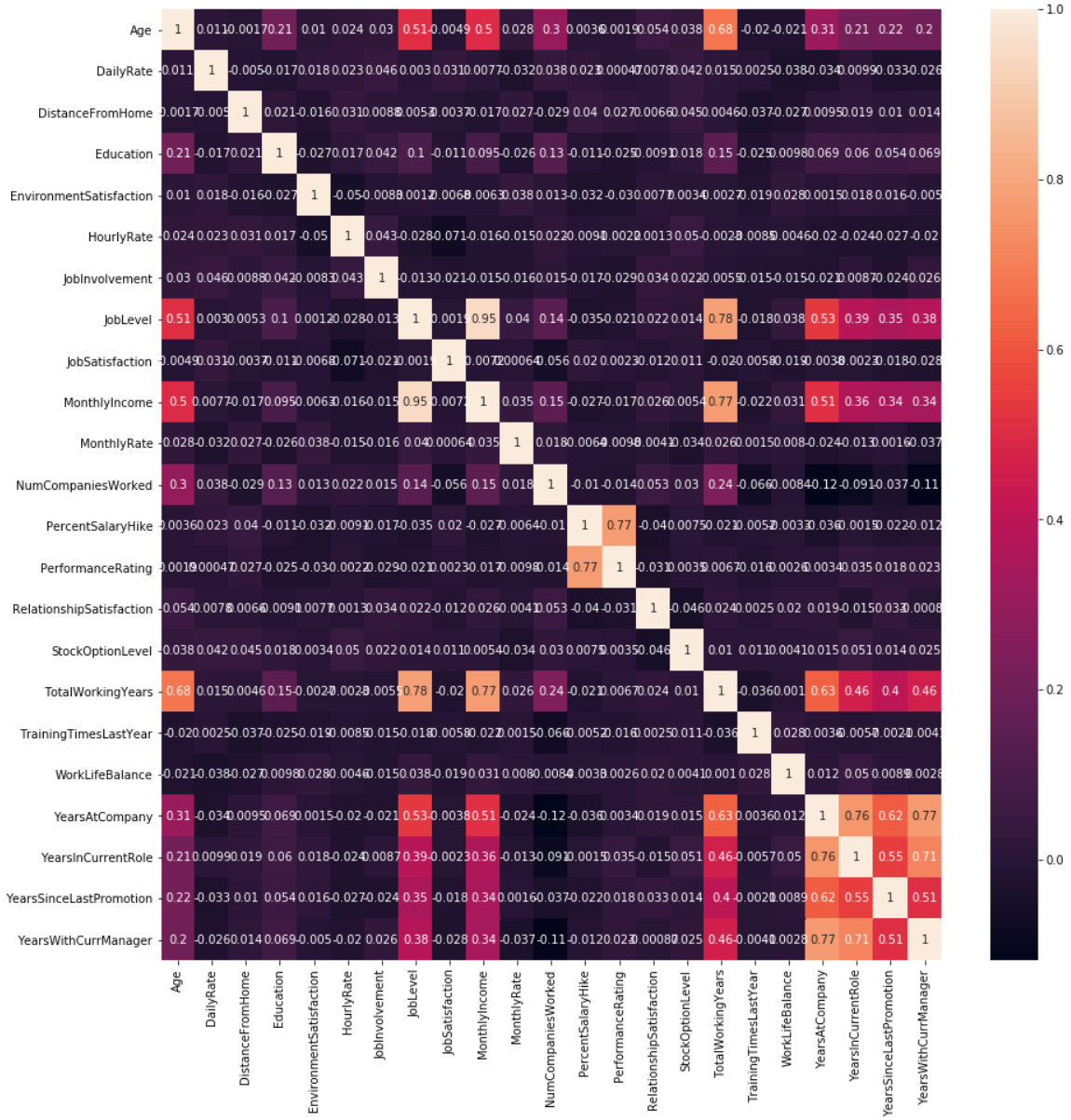
23 rows × 23 columns

In [21]:

```
plt.figure(figsize=(15,15))
sns.heatmap(df.corr(),annot=True)
```

Out[21]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24f005b3388>
```




In [22]:

```
from sklearn.preprocessing import LabelEncoder
```

In [26]:

```
for column in df.columns:
    if df[column].dtypes == np.number:
        continue
    df[column]=LabelEncoder().fit_transform(df[column]) #transforming the lables & normalizing it.
```

In [27]:

```
df
```

Out[27]:

| | Age | Attrition | BusinessTravel | DailyRate | Department | DistanceFromHome | Education |
|---|---|---|---|---|---|---|---|
| **0** | 23 | 1 | 2 | 624 | 2 | 0 | 1 |
| **1** | 31 | 0 | 1 | 113 | 1 | 7 | 0 |
| **2** | 19 | 1 | 2 | 805 | 1 | 1 | 1 |
| **3** | 15 | 0 | 1 | 820 | 1 | 2 | 3 |
| **4** | 9 | 0 | 2 | 312 | 1 | 1 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **1465** | 18 | 0 | 1 | 494 | 1 | 22 | 1 |
| **1466** | 21 | 0 | 2 | 327 | 1 | 5 | 0 |
| **1467** | 9 | 0 | 2 | 39 | 1 | 3 | 2 |
| **1468** | 31 | 0 | 1 | 579 | 2 | 1 | 2 |
| **1469** | 16 | 0 | 2 | 336 | 1 | 7 | 2 |

1470 rows × 31 columns

In [28]:

```
df['Age_years']=df['Age']
```

In [29]:

```
df=df.drop('Age',axis=1)
```

In [30]:

```
df.head()
```

Out[30]:

| | Attrition | BusinessTravel | DailyRate | Department | DistanceFromHome | Education | Educatio |
|---|---|---|---|---|---|---|---|
| **0** | 1 | 2 | 624 | 2 | 0 | 1 | |
| **1** | 0 | 1 | 113 | 1 | 7 | 0 | |
| **2** | 1 | 2 | 805 | 1 | 1 | 1 | |
| **3** | 0 | 1 | 820 | 1 | 2 | 3 | |
| **4** | 0 | 2 | 312 | 1 | 1 | 0 | |

5 rows × 31 columns

Finding out the best model for the datasets

In [55]:

```
y=df.iloc[:,-1]
```

In [56]:

```
y.shape
```

Out[56]:

```
(1470,)
```

In [57]:

```
x=df.iloc[:,0:-1]
```

In [58]:

```
x.shape
```

Out[58]:

```
(1470, 30)
```

In [59]:

```
from sklearn.model_selection import train_test_split
```

In [90]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.15,random_state=0)
```

In [95]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [96]:

```
forest=RandomForestClassifier(n_estimators=10,criterion='entropy',random_state=0)
```

In [97]:

```
forest.fit(x_train,y_train)
```

Out[97]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='entropy', max_depth=None, max_features
='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=0, verbo
se=0,
                       warm_start=False)
```

In [98]:

```
forest.score(x_train,y_train)
```

Out[98]:

```
0.9911929543634908
```

In [103]:

```
from sklearn.metrics import confusion_matrix
```

In [105]:

```
cm=confusion_matrix(y_test,forest.predict(x_test))
```

In [106]:

```
print(cm)
```

```
[[0 0 1 ... 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]]
```

Conclusion: We have used random forest classifier and find out how this model can be helpful as a standard technique for this dataset and find out model accuracy,also used to evaluate the performance of a classification model though confusion matrix.