In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
filepath = 'C:/Users/HP/Documents/python_one_campus/Assignment/'
filename = 'WA_Fn-UseC_-HR-Employee-Attrition.csv'

df = pd.read_csv(filepath + filename)
```

Exploratory Analysis

In [3]:
```
df.head() #checking the data
```

Out[3]:

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

5 rows × 35 columns

In [4]:
```
df.describe()
```

Out[4]:

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumb
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.0	1470.0000
mean	36.923810	802.485714	9.192517	2.912925	1.0	1024.8653
std	9.135373	403.509100	8.106864	1.024165	0.0	602.0243
min	18.000000	102.000000	1.000000	1.000000	1.0	1.0000
25%	30.000000	465.000000	2.000000	2.000000	1.0	491.2500
50%	36.000000	802.000000	7.000000	3.000000	1.0	1020.5000
75%	43.000000	1157.000000	14.000000	4.000000	1.0	1555.7500
max	60.000000	1499.000000	29.000000	5.000000	1.0	2068.0000

8 rows × 26 columns

In [5]:
```
1 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 [6]:
```
1 print(df.nunique())
```

```
Age                         43
Attrition                    2
BusinessTravel               3
DailyRate                  886
Department                   3
DistanceFromHome            29
Education                    5
EducationField               6
EmployeeCount                1
EmployeeNumber            1470
EnvironmentSatisfaction      4
Gender                       2
HourlyRate                  71
JobInvolvement               4
JobLevel                     5
JobRole                      9
JobSatisfaction              4
MaritalStatus                3
MonthlyIncome             1349
MonthlyRate               1427
NumCompaniesWorked          10
Over18                       1
OverTime                     2
PercentSalaryHike           15
PerformanceRating            2
RelationshipSatisfaction     4
StandardHours                1
StockOptionLevel             4
TotalWorkingYears           40
TrainingTimesLastYear        7
WorkLifeBalance              4
YearsAtCompany              37
YearsInCurrentRole          19
YearsSinceLastPromotion     16
YearsWithCurrManager        18
dtype: int64
```

In [8]:
```
1 df.columns
```

Out[8]:
```
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 [9]:
```
#removing the columns with one unique value because they are zero-variance p
#they do not contain any information for modeling
#EmployeeCount, Over18, StandardHours
#ea is employee attrition
ea = df.loc[:, ['Age', 'BusinessTravel', 'DailyRate', 'Department',
       'DistanceFromHome', 'Education', 'EducationField',
       'EmployeeNumber', 'EnvironmentSatisfaction', 'Gender', 'HourlyRate',
       'JobInvolvement', 'JobLevel', 'JobRole', 'JobSatisfaction',
       'MaritalStatus', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked'
       'OverTime', 'PercentSalaryHike', 'PerformanceRating',
       'RelationshipSatisfaction', 'StockOptionLevel',
       'TotalWorkingYears', 'TrainingTimesLastYear', 'WorkLifeBalance',
       'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion',
       'YearsWithCurrManager', 'Attrition']]
```

In [10]:
```
ea.head() #new dataframe
```

Out[10]:

	Age	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	E
0	41	Travel_Rarely	1102	Sales	1	2	Life Sciences	
1	49	Travel_Frequently	279	Research & Development	8	1	Life Sciences	
2	37	Travel_Rarely	1373	Research & Development	2	2	Other	
3	33	Travel_Frequently	1392	Research & Development	3	4	Life Sciences	
4	27	Travel_Rarely	591	Research & Development	2	1	Medical	

5 rows × 32 columns

In [11]:
```
#To show data distribution
ea.hist()
plt.gcf().set_size_inches(20,20)
plt.show()
```

In [12]:
```
1 ea.isna().sum() #to check for missing values
```

Out[12]:
```
Age                         0
BusinessTravel              0
DailyRate                   0
Department                  0
DistanceFromHome            0
Education                   0
EducationField              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
OverTime                    0
PercentSalaryHike           0
PerformanceRating           0
RelationshipSatisfaction    0
StockOptionLevel            0
TotalWorkingYears           0
TrainingTimesLastYear       0
WorkLifeBalance             0
YearsAtCompany              0
YearsInCurrentRole          0
YearsSinceLastPromotion     0
YearsWithCurrManager        0
Attrition                   0
dtype: int64
```

In [13]:
```
ea.info()

```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 32 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   Age                       1470 non-null   int64 
 1   BusinessTravel            1470 non-null   object
 2   DailyRate                 1470 non-null   int64 
 3   Department                1470 non-null   object
 4   DistanceFromHome          1470 non-null   int64 
 5   Education                 1470 non-null   int64 
 6   EducationField            1470 non-null   object
 7   EmployeeNumber            1470 non-null   int64 
 8   EnvironmentSatisfaction   1470 non-null   int64 
 9   Gender                    1470 non-null   object
 10  HourlyRate                1470 non-null   int64 
 11  JobInvolvement            1470 non-null   int64 
 12  JobLevel                  1470 non-null   int64 
 13  JobRole                   1470 non-null   object
 14  JobSatisfaction           1470 non-null   int64 
 15  MaritalStatus             1470 non-null   object
 16  MonthlyIncome             1470 non-null   int64 
 17  MonthlyRate               1470 non-null   int64 
 18  NumCompaniesWorked        1470 non-null   int64 
 19  OverTime                  1470 non-null   object
 20  PercentSalaryHike         1470 non-null   int64 
 21  PerformanceRating         1470 non-null   int64 
 22  RelationshipSatisfaction  1470 non-null   int64 
 23  StockOptionLevel          1470 non-null   int64 
 24  TotalWorkingYears         1470 non-null   int64 
 25  TrainingTimesLastYear     1470 non-null   int64 
 26  WorkLifeBalance           1470 non-null   int64 
 27  YearsAtCompany            1470 non-null   int64 
 28  YearsInCurrentRole        1470 non-null   int64 
 29  YearsSinceLastPromotion   1470 non-null   int64 
 30  YearsWithCurrManager      1470 non-null   int64 
 31  Attrition                 1470 non-null   object
dtypes: int64(24), object(8)
memory usage: 367.6+ KB
```

In [14]:
```
#some columns have object as their datatype, they need to be changed to cate
ea['BusinessTravel'] = ea['BusinessTravel'].astype('category')
ea['Department'] = ea['Department'].astype('category')
ea['EducationField'] = ea['EducationField'].astype('category')
ea['Gender'] = ea['Gender'].astype('category')
ea['JobRole'] = ea['JobRole'].astype('category')
ea['MaritalStatus'] = ea['MaritalStatus'].astype('category')
ea['OverTime'] = ea['OverTime'].astype('category')
ea['Attrition'] = ea['Attrition'].astype('category')


```

In [15]:
```
#categorical data encoding
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
# Assigning numerical values and storing in another column


ea['BusinessTravel'] = le.fit_transform(ea['BusinessTravel'])
ea['Department'] =le.fit_transform(ea['Department'])
ea['EducationField'] =le.fit_transform(ea['EducationField'])
ea['Gender'] =le.fit_transform(ea['Gender'])
ea['JobRole'] =le.fit_transform(ea['JobRole'])
ea['MaritalStatus'] =le.fit_transform(ea['MaritalStatus'])
ea['OverTime'] =le.fit_transform(ea['OverTime'])
ea['Attrition'] =le.fit_transform(ea['Attrition'])
```

In [16]:
```
dfEncoded = ea
```

In [17]:
```
#putting Age column as the last column to make it easy for identification
#as the output variable for analysis
#eAge is employee age
eAge = dfEncoded.loc[:, ['BusinessTravel', 'DailyRate', 'Department', 'Dista
        'Education',
       'EducationField', 'EmployeeNumber', 'EnvironmentSatisfaction', 'Gende
       'JobInvolvement', 'JobLevel', 'JobRole', 'JobSatisfaction', 'MaritalS
       'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'OverTime', 'Pe
       'PerformanceRating', 'RelationshipSatisfaction', 'StockOptionLevel',
       'TrainingTimesLastYear', 'WorkLifeBalance', 'YearsAtCompany', 'YearsI
       'YearsSinceLastPromotion', 'YearsWithCurrManager', 'Attrition','Age']
```

In [18]:
```
eAge.head()
```

Out[18]:

	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	Employe
0	2	1102	2	1	2	1	
1	1	279	1	8	1	1	
2	2	1373	1	2	2	4	
3	1	1392	1	3	4	1	
4	2	591	1	2	1	3	

5 rows × 32 columns

In [19]:
```
1 eAge.columns
```

Out[19]:
```
Index(['BusinessTravel', 'DailyRate', 'Department', 'DistanceFromHome',
       'Education', 'EducationField', 'EmployeeNumber',
       'EnvironmentSatisfaction', 'Gender', 'HourlyRate', 'JobInvolvement',
       'JobLevel', 'JobRole', 'JobSatisfaction', 'MaritalStatus',
       'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'OverTime',
       'PercentSalaryHike', 'PerformanceRating', 'RelationshipSatisfaction',
       'StockOptionLevel', 'TotalWorkingYears', 'TrainingTimesLastYear',
       'WorkLifeBalance', 'YearsAtCompany', 'YearsInCurrentRole',
       'YearsSinceLastPromotion', 'YearsWithCurrManager', 'Attrition', 'Age'],
      dtype='object')
```

In [20]:

```
# Rescale data (between 0 and 1)
from pandas import read_csv
from numpy import set_printoptions
from sklearn.preprocessing import MinMaxScaler


array = eAge.values
# separate array into input and output components
#X is the predictor variable
#Y is the outcome variable
X = array[:,0:31]
Y = array[:,31]
demo = MinMaxScaler(feature_range=(0, 1))
rescaledX = demo.fit_transform(X)
# summarize transformed data
set_printoptions(precision=3)

print(rescaledX[0:5,:])

```

```
[[1.000e+00 7.158e-01 1.000e+00 0.000e+00 2.500e-01 2.000e-01 0.000e+00
  3.333e-01 0.000e+00 9.143e-01 6.667e-01 2.500e-01 8.750e-01 1.000e+00
  1.000e+00 2.625e-01 6.981e-01 8.889e-01 1.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 2.000e-01 0.000e+00 0.000e+00 1.500e-01 2.222e-01
  0.000e+00 2.941e-01 1.000e+00]
 [5.000e-01 1.267e-01 5.000e-01 2.500e-01 0.000e+00 2.000e-01 4.838e-04
  6.667e-01 1.000e+00 4.429e-01 3.333e-01 2.500e-01 7.500e-01 3.333e-01
  5.000e-01 2.170e-01 9.160e-01 1.111e-01 0.000e+00 8.571e-01 1.000e+00
  1.000e+00 3.333e-01 2.500e-01 5.000e-01 6.667e-01 2.500e-01 3.889e-01
  6.667e-02 4.118e-01 0.000e+00]
 [1.000e+00 9.098e-01 5.000e-01 3.571e-02 2.500e-01 8.000e-01 1.451e-03
  1.000e+00 1.000e+00 8.857e-01 3.333e-01 0.000e+00 2.500e-01 6.667e-01
  1.000e+00 5.692e-02 1.213e-02 6.667e-01 1.000e+00 2.857e-01 0.000e+00
  3.333e-01 0.000e+00 1.750e-01 5.000e-01 6.667e-01 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 1.000e+00]
 [5.000e-01 9.234e-01 5.000e-01 7.143e-02 7.500e-01 2.000e-01 1.935e-03
  1.000e+00 0.000e+00 3.714e-01 6.667e-01 0.000e+00 7.500e-01 6.667e-01
  5.000e-01 1.001e-01 8.458e-01 1.111e-01 1.000e+00 0.000e+00 0.000e+00
  6.667e-01 0.000e+00 2.000e-01 5.000e-01 6.667e-01 2.000e-01 3.889e-01
  2.000e-01 0.000e+00 0.000e+00]
 [1.000e+00 3.500e-01 5.000e-01 3.571e-02 0.000e+00 6.000e-01 2.903e-03
  0.000e+00 1.000e+00 1.429e-01 6.667e-01 0.000e+00 2.500e-01 3.333e-01
  5.000e-01 1.295e-01 5.837e-01 1.000e+00 0.000e+00 7.143e-02 0.000e+00
  1.000e+00 3.333e-01 1.500e-01 5.000e-01 6.667e-01 5.000e-02 1.111e-01
  1.333e-01 1.176e-01 0.000e+00]]
```

In [21]:
```
eAgeRescaled = pd.DataFrame(rescaledX)
eAgeRescaled.columns = ['BusinessTravel', 'DailyRate', 'Department', 'Distan
       'EducationField', 'EmployeeNumber', 'EnvironmentSatisfaction', 'Gende
       'JobInvolvement', 'JobLevel', 'JobRole', 'JobSatisfaction', 'MaritalS
       'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'OverTime', 'Pe
       'PerformanceRating', 'RelationshipSatisfaction', 'StockOptionLevel',
       'TrainingTimesLastYear', 'WorkLifeBalance', 'YearsAtCompany', 'YearsI
       'YearsSinceLastPromotion', 'YearsWithCurrManager', 'Attrition']
eAgeRescaled['Age'] = eAge['Age']
eAgeRescaled
```

Out[21]:

	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	Empl
0	1.0	0.715820	1.0	0.000000	0.25	0.2	
1	0.5	0.126700	0.5	0.250000	0.00	0.2	
2	1.0	0.909807	0.5	0.035714	0.25	0.8	
3	0.5	0.923407	0.5	0.071429	0.75	0.2	
4	1.0	0.350036	0.5	0.035714	0.00	0.6	
...	...	...	...	...	...	...	
1465	0.5	0.559771	0.5	0.785714	0.25	0.6	
1466	1.0	0.365784	0.5	0.178571	0.00	0.6	
1467	1.0	0.037938	0.5	0.107143	0.50	0.2	
1468	0.5	0.659270	1.0	0.035714	0.50	0.6	
1469	1.0	0.376521	0.5	0.250000	0.50	0.6	

1470 rows × 32 columns

In [22]:
```
eAgeRescaled.describe()
```

Out[22]:

	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.000000	1470.000000
mean	0.803741	0.501421	0.630272	0.292590	0.478231	0.449524
std	0.332727	0.288840	0.263896	0.289531	0.256041	0.266274
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.500000	0.259843	0.500000	0.035714	0.250000	0.200000
50%	1.000000	0.501074	0.500000	0.214286	0.500000	0.400000
75%	1.000000	0.755190	1.000000	0.464286	0.750000	0.600000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 32 columns

In [23]:
```
eAge = eAgeRescaled
```

In [24]:
```
# Normalize data (length of 1)
from sklearn.preprocessing import Normalizer
from pandas import read_csv
from numpy import set_printoptions

# separate array into input and output components
array = eAge.values
X = array[:,0:31]
Y = array[:,31]
scaler = Normalizer().fit(X)
normalizedX = scaler.transform(X)
# summarize transformed data
set_printoptions(precision=5)
print(normalizedX[0:5,:])
```

```
[[3.10398e-01 2.22189e-01 3.10398e-01 0.00000e+00 7.75995e-02 6.20796e-02
  0.00000e+00 1.03466e-01 0.00000e+00 2.83792e-01 2.06932e-01 7.75995e-02
  2.71598e-01 3.10398e-01 3.10398e-01 8.14651e-02 2.16674e-01 2.75909e-01
  3.10398e-01 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 6.20796e-02
  0.00000e+00 0.00000e+00 4.65597e-02 6.89773e-02 0.00000e+00 9.12935e-02
  3.10398e-01]
 [1.74127e-01 4.41238e-02 1.74127e-01 8.70635e-02 0.00000e+00 6.96508e-02
  1.68483e-04 2.32169e-01 3.48254e-01 1.54227e-01 1.16085e-01 8.70635e-02
  2.61191e-01 1.16085e-01 1.74127e-01 7.55743e-02 3.19001e-01 3.86949e-02
  0.00000e+00 2.98504e-01 3.48254e-01 3.48254e-01 1.16085e-01 8.70635e-02
  1.74127e-01 2.32169e-01 8.70635e-02 1.35432e-01 2.32169e-02 1.43399e-01
  0.00000e+00]
 [3.07879e-01 2.80110e-01 1.53939e-01 1.09957e-02 7.69697e-02 2.46303e-01
  4.46849e-04 3.07879e-01 3.07879e-01 2.72693e-01 1.02626e-01 0.00000e+00
  7.69697e-02 2.05252e-01 3.07879e-01 1.75259e-02 3.73336e-03 2.05252e-01
  3.07879e-01 8.79653e-02 0.00000e+00 1.02626e-01 0.00000e+00 5.38788e-02
  1.53939e-01 2.05252e-01 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00
  3.07879e-01]
 [1.77359e-01 3.27550e-01 1.77359e-01 2.53371e-02 2.66039e-01 7.09438e-02
  6.86442e-04 3.54719e-01 0.00000e+00 1.31753e-01 2.36479e-01 0.00000e+00
  2.66039e-01 2.36479e-01 1.77359e-01 3.54906e-02 3.00026e-01 3.94132e-02
  3.54719e-01 0.00000e+00 0.00000e+00 2.36479e-01 0.00000e+00 7.09438e-02
  1.77359e-01 2.36479e-01 7.09438e-02 1.37946e-01 7.09438e-02 0.00000e+00
  0.00000e+00]
 [3.81818e-01 1.33650e-01 1.90909e-01 1.36364e-02 0.00000e+00 2.29091e-01
  1.10833e-03 0.00000e+00 3.81818e-01 5.45454e-02 2.54545e-01 0.00000e+00
  9.54545e-02 1.27273e-01 1.90909e-01 4.94413e-02 2.22882e-01 3.81818e-01
  0.00000e+00 2.72727e-02 0.00000e+00 3.81818e-01 1.27273e-01 5.72727e-02
  1.90909e-01 2.54545e-01 1.90909e-02 4.24242e-02 5.09091e-02 4.49198e-02
  0.00000e+00]]
```

In [25]:
```
eAgeNormalized = pd.DataFrame(rescaledX)
eAgeNormalized.columns = ['BusinessTravel', 'DailyRate', 'Department', 'Dist
       'EducationField', 'EmployeeNumber', 'EnvironmentSatisfaction', 'Gende
       'JobInvolvement', 'JobLevel', 'JobRole', 'JobSatisfaction', 'MaritalS
       'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'OverTime', 'Pe
       'PerformanceRating', 'RelationshipSatisfaction', 'StockOptionLevel',
       'TrainingTimesLastYear', 'WorkLifeBalance', 'YearsAtCompany', 'YearsI
       'YearsSinceLastPromotion', 'YearsWithCurrManager', 'Attrition']
eAgeNormalized['Age'] = eAge['Age']
eAgeNormalized
```

Out[25]:

	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	Empl
0	1.0	0.715820	1.0	0.000000	0.25	0.2	
1	0.5	0.126700	0.5	0.250000	0.00	0.2	
2	1.0	0.909807	0.5	0.035714	0.25	0.8	
3	0.5	0.923407	0.5	0.071429	0.75	0.2	
4	1.0	0.350036	0.5	0.035714	0.00	0.6	
...	...	...	...	...	...	...	
1465	0.5	0.559771	0.5	0.785714	0.25	0.6	
1466	1.0	0.365784	0.5	0.178571	0.00	0.6	
1467	1.0	0.037938	0.5	0.107143	0.50	0.2	
1468	0.5	0.659270	1.0	0.035714	0.50	0.6	
1469	1.0	0.376521	0.5	0.250000	0.50	0.6	

1470 rows × 32 columns

In [26]:
```
eAge = eAgeNormalized
```

In [27]:
```
1 eAge.describe()
```

Out[27]:

	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.000000	1470.000000
mean	0.803741	0.501421	0.630272	0.292590	0.478231	0.449524
std	0.332727	0.288840	0.263896	0.289531	0.256041	0.266274
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.500000	0.259843	0.500000	0.035714	0.250000	0.200000
50%	1.000000	0.501074	0.500000	0.214286	0.500000	0.400000
75%	1.000000	0.755190	1.000000	0.464286	0.750000	0.600000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 32 columns

In [28]:
```
1 eAge.head()
```

Out[28]:

	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	Employe
0	1.0	0.715820	1.0	0.000000	0.25	0.2	
1	0.5	0.126700	0.5	0.250000	0.00	0.2	
2	1.0	0.909807	0.5	0.035714	0.25	0.8	
3	0.5	0.923407	0.5	0.071429	0.75	0.2	
4	1.0	0.350036	0.5	0.035714	0.00	0.6	

5 rows × 32 columns

In [29]:
```
#to identify which attributes (and combination of attributes)contribute the
#Feature Extraction with RFE

from pandas import read_csv
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
# load data

array = eAge.values
X = array[:,0:31]
Y = array[:,31]
# feature extraction
model = LogisticRegression(max_iter = 500)
rfe = RFE(model, 18)
fit = rfe.fit(X, Y)
print("Num Features: {}".format(fit.n_features_))
print("Selected Features: {}".format(fit.support_))
print("Feature Ranking: {}".format(fit.ranking_))
```

```
C:\Users\HP\anaconda3\lib\site-packages\sklearn\utils\validation.py:70: FutureW
arning: Pass n_features_to_select=18 as keyword args. From version 1.0 (renamin
g of 0.25) passing these as positional arguments will result in an error
  warnings.warn(f"Pass {args_msg} as keyword args. From version "

Num Features: 18
Selected Features: [ True False  True False  True False  True False False False
True  True
  True False  True  True  True  True False False False  True False  True
 False  True False  True  True  True  True]
Feature Ranking: [ 1  4  1  8  1  6  1 13  5  9  1  1  1 11  1  1  1  1 12 10 1
4  1  2  1
  7  1  3  1  1  1  1]
```

In [30]:
```
viableCols = []
rankList = list(fit.ranking_)
columnList = ['BusinessTravel', 'DailyRate', 'Department', 'DistanceFromHome
       'EducationField', 'EmployeeNumber', 'EnvironmentSatisfaction', 'Gende
       'JobInvolvement', 'JobLevel', 'JobRole', 'JobSatisfaction', 'MaritalS
       'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'OverTime', 'Pe
       'PerformanceRating', 'RelationshipSatisfaction', 'StockOptionLevel',
       'TrainingTimesLastYear', 'WorkLifeBalance', 'YearsAtCompany', 'YearsI
       'YearsSinceLastPromotion', 'YearsWithCurrManager', 'Attrition','Age']
for rank, col in zip(rankList, columnList):
    idx = rank
    colname = col
    if idx == 1:
        viableCols.append(colname)
print(viableCols)
```

```
['BusinessTravel', 'Department', 'Education', 'EmployeeNumber', 'JobInvolvemen
t', 'JobLevel', 'JobRole', 'MaritalStatus', 'MonthlyIncome', 'MonthlyRate', 'Nu
mCompaniesWorked', 'RelationshipSatisfaction', 'TotalWorkingYears', 'WorkLifeBa
lance', 'YearsInCurrentRole', 'YearsSinceLastPromotion', 'YearsWithCurrManage
r', 'Attrition']
```

In [31]:
```
eAgeModel = eAge.loc[:, viableCols] #eAgeModel is the employee age data that
eAgeModel
```

Out[31]:

	BusinessTravel	Department	Education	EmployeeNumber	JobInvolvement	JobLevel	JobRol
0	1.0	1.0	0.25	0.000000	0.666667	0.25	0.87
1	0.5	0.5	0.00	0.000484	0.333333	0.25	0.75
2	1.0	0.5	0.25	0.001451	0.333333	0.00	0.25
3	0.5	0.5	0.75	0.001935	0.666667	0.00	0.75
4	1.0	0.5	0.00	0.002903	0.666667	0.00	0.25
...	...	...	...	...	...	...	
1465	0.5	0.5	0.25	0.996613	1.000000	0.25	0.25
1466	1.0	0.5	0.00	0.997097	0.333333	0.50	0.00
1467	1.0	0.5	0.50	0.998065	1.000000	0.25	0.50
1468	0.5	1.0	0.50	0.998549	0.333333	0.25	0.87
1469	1.0	0.5	0.50	1.000000	1.000000	0.25	0.25

1470 rows × 18 columns

In [32]:
```
eAgeModel['Age'] = eAge['Age']
eAgeModel
```

Out[32]:

	BusinessTravel	Department	Education	EmployeeNumber	JobInvolvement	JobLevel	JobRol
0	1.0	1.0	0.25	0.000000	0.666667	0.25	0.87
1	0.5	0.5	0.00	0.000484	0.333333	0.25	0.75
2	1.0	0.5	0.25	0.001451	0.333333	0.00	0.25
3	0.5	0.5	0.75	0.001935	0.666667	0.00	0.75
4	1.0	0.5	0.00	0.002903	0.666667	0.00	0.25
...	...	...	...	...	...	...	
1465	0.5	0.5	0.25	0.996613	1.000000	0.25	0.25
1466	1.0	0.5	0.00	0.997097	0.333333	0.50	0.00
1467	1.0	0.5	0.50	0.998065	1.000000	0.25	0.50
1468	0.5	1.0	0.50	0.998549	0.333333	0.25	0.87
1469	1.0	0.5	0.50	1.000000	1.000000	0.25	0.25

1470 rows × 19 columns

Regression algorithms to predict the age of

employees leaving the firm

In [34]:
```
# Linear Regression
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression

array = eAgeModel.values

X = array[:,0:18]
Y = array[:,18]
kfold = KFold(n_splits=10, random_state=7, shuffle = True)
model = LinearRegression()
scoring = 'neg_mean_squared_error'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print(results.mean())
```

```
-41.646798794070435
```

In [35]:
```
# Lasso Regression
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import Lasso

array = eAgeModel.values
X = array[:,0:18]
Y = array[:,18]
kfold = KFold(n_splits=10, random_state=7, shuffle = True)
model = Lasso()
scoring = 'neg_mean_squared_error'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print(results.mean())
```

```
-71.94709308788553
```

In [36]:
```
# ElasticNet Regression
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import ElasticNet

array = eAgeModel.values
X = array[:,0:18]
Y = array[:,18]
kfold = KFold(n_splits=10, random_state=7, shuffle = True)
model = ElasticNet()
scoring = 'neg_mean_squared_error'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print(results.mean())
```

```
-75.68337458204027
```

In [37]:
```
# KNN Regression
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsRegressor

array = eAgeModel.values
X = array[:,0:18]
Y = array[:,18]
kfold = KFold(n_splits=10, random_state=7, shuffle = True)
model = KNeighborsRegressor()
scoring = 'neg_mean_squared_error'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print(results.mean())
```

```
-61.06560544217687
```

Classification algorithms to predict the attrition of employees

In [38]:
```
dfEncoded = ea #ea is employee attriton data
```

In [39]:
```
ea.describe()
```

Out[39]:

Balance	YearsAtCompany	YearsInCurrentRole	YearsSinceLastPromotion	YearsWithCurrManager	
).000000	1470.000000	1470.000000	1470.000000	1470.000000	147(
2.761224	7.008163	4.229252	2.187755	4.123129	(
).706476	6.126525	3.623137	3.222430	3.568136	(
1.000000	0.000000	0.000000	0.000000	0.000000	(
2.000000	3.000000	2.000000	0.000000	2.000000	(
3.000000	5.000000	3.000000	1.000000	3.000000	(
3.000000	9.000000	7.000000	3.000000	7.000000	(
4.000000	40.000000	18.000000	15.000000	17.000000	

In [40]:
```
1 ea.head()
```

Out[40]:

	Age	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	Em
0	41	2	1102	2	1	2	1	
1	49	1	279	1	8	1	1	
2	37	2	1373	1	2	2	4	
3	33	1	1392	1	3	4	1	
4	27	2	591	1	2	1	3	

5 rows × 32 columns

In [41]:
```
1 ea.columns
```

Out[41]:
```
Index(['Age', 'BusinessTravel', 'DailyRate', 'Department', 'DistanceFromHome',
       'Education', 'EducationField', 'EmployeeNumber',
       'EnvironmentSatisfaction', 'Gender', 'HourlyRate', 'JobInvolvement',
       'JobLevel', 'JobRole', 'JobSatisfaction', 'MaritalStatus',
       'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'OverTime',
       'PercentSalaryHike', 'PerformanceRating', 'RelationshipSatisfaction',
       'StockOptionLevel', 'TotalWorkingYears', 'TrainingTimesLastYear',
       'WorkLifeBalance', 'YearsAtCompany', 'YearsInCurrentRole',
       'YearsSinceLastPromotion', 'YearsWithCurrManager', 'Attrition'],
      dtype='object')
```

In [42]:

```
# Rescale data (between 0 and 1) with attrition as Y
from pandas import read_csv
from numpy import set_printoptions
from sklearn.preprocessing import MinMaxScaler

array = ea.values
# separate array into input and output components
#X is the predictor variable
#Y is the outcome variable
X = array[:,0:31]
Y = array[:,31]
demo = MinMaxScaler(feature_range=(0, 1))
rescaledX = demo.fit_transform(X)
# summarize transformed data
set_printoptions(precision=3)

print(rescaledX[0:5,:])

```

```
[[5.476e-01 1.000e+00 7.158e-01 1.000e+00 0.000e+00 2.500e-01 2.000e-01
  0.000e+00 3.333e-01 0.000e+00 9.143e-01 6.667e-01 2.500e-01 8.750e-01
  1.000e+00 1.000e+00 2.625e-01 6.981e-01 8.889e-01 1.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 2.000e-01 0.000e+00 0.000e+00 1.500e-01
  2.222e-01 0.000e+00 2.941e-01]
 [7.381e-01 5.000e-01 1.267e-01 5.000e-01 2.500e-01 0.000e+00 2.000e-01
  4.838e-04 6.667e-01 1.000e+00 4.429e-01 3.333e-01 2.500e-01 7.500e-01
  3.333e-01 5.000e-01 2.170e-01 9.160e-01 1.111e-01 0.000e+00 8.571e-01
  1.000e+00 1.000e+00 3.333e-01 2.500e-01 5.000e-01 6.667e-01 2.500e-01
  3.889e-01 6.667e-02 4.118e-01]
 [4.524e-01 1.000e+00 9.098e-01 5.000e-01 3.571e-02 2.500e-01 8.000e-01
  1.451e-03 1.000e+00 1.000e+00 8.857e-01 3.333e-01 0.000e+00 2.500e-01
  6.667e-01 1.000e+00 5.692e-02 1.213e-02 6.667e-01 1.000e+00 2.857e-01
  0.000e+00 3.333e-01 0.000e+00 1.750e-01 5.000e-01 6.667e-01 0.000e+00
  0.000e+00 0.000e+00 0.000e+00]
 [3.571e-01 5.000e-01 9.234e-01 5.000e-01 7.143e-02 7.500e-01 2.000e-01
  1.935e-03 1.000e+00 0.000e+00 3.714e-01 6.667e-01 0.000e+00 7.500e-01
  6.667e-01 5.000e-01 1.001e-01 8.458e-01 1.111e-01 1.000e+00 0.000e+00
  0.000e+00 6.667e-01 0.000e+00 2.000e-01 5.000e-01 6.667e-01 2.000e-01
  3.889e-01 2.000e-01 0.000e+00]
 [2.143e-01 1.000e+00 3.500e-01 5.000e-01 3.571e-02 0.000e+00 6.000e-01
  2.903e-03 0.000e+00 1.000e+00 1.429e-01 6.667e-01 0.000e+00 2.500e-01
  3.333e-01 5.000e-01 1.295e-01 5.837e-01 1.000e+00 0.000e+00 7.143e-02
  0.000e+00 1.000e+00 3.333e-01 1.500e-01 5.000e-01 6.667e-01 5.000e-02
  1.111e-01 1.333e-01 1.176e-01]]
```

In [43]:
```
eaRescaled = pd.DataFrame(rescaledX)
eaRescaled.columns = ['Age', 'BusinessTravel', 'DailyRate', 'Department', 'D
       'EducationField', 'EmployeeNumber', 'EnvironmentSatisfaction', 'Gende
       'JobInvolvement', 'JobLevel', 'JobRole', 'JobSatisfaction', 'MaritalS
       'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'OverTime', 'Pe
       'PerformanceRating', 'RelationshipSatisfaction', 'StockOptionLevel',
       'TrainingTimesLastYear', 'WorkLifeBalance', 'YearsAtCompany', 'YearsI
       'YearsSinceLastPromotion', 'YearsWithCurrManager']
eaRescaled['Attrition'] = ea['Attrition']
eaRescaled
```

Out[43]:

	Age	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationFi
0	0.547619	1.0	0.715820	1.0	0.000000	0.25	
1	0.738095	0.5	0.126700	0.5	0.250000	0.00	
2	0.452381	1.0	0.909807	0.5	0.035714	0.25	
3	0.357143	0.5	0.923407	0.5	0.071429	0.75	
4	0.214286	1.0	0.350036	0.5	0.035714	0.00	
...	...	...	...	...	...	...	
1465	0.428571	0.5	0.559771	0.5	0.785714	0.25	
1466	0.500000	1.0	0.365784	0.5	0.178571	0.00	
1467	0.214286	1.0	0.037938	0.5	0.107143	0.50	
1468	0.738095	0.5	0.659270	1.0	0.035714	0.50	
1469	0.380952	1.0	0.376521	0.5	0.250000	0.50	

1470 rows × 32 columns

In [44]:
```
eaRescaled.describe()
```

Out[44]:

	Age	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	E
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.000000	1470.000000	
mean	0.450567	0.803741	0.501421	0.630272	0.292590	0.478231	
std	0.217509	0.332727	0.288840	0.263896	0.289531	0.256041	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.285714	0.500000	0.259843	0.500000	0.035714	0.250000	
50%	0.428571	1.000000	0.501074	0.500000	0.214286	0.500000	
75%	0.595238	1.000000	0.755190	1.000000	0.464286	0.750000	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

8 rows × 32 columns

In [45]:
```
ea = eaRescaled
```

In [46]:
```
# Normalize data (length of 1)
from sklearn.preprocessing import Normalizer
from pandas import read_csv
from numpy import set_printoptions

# separate array into input and output components
array = ea.values
X = array[:,0:31]
Y = array[:,31]
scaler = Normalizer().fit(X)
normalizedX = scaler.transform(X)
# summarize transformed data
set_printoptions(precision=5)
print(normalizedX[0:5,:])
```

```
[[1.76020e-01 3.21428e-01 2.30084e-01 3.21428e-01 0.00000e+00 8.03570e-02
  6.42856e-02 0.00000e+00 1.07143e-01 0.00000e+00 2.93877e-01 2.14285e-01
  8.03570e-02 2.81249e-01 3.21428e-01 3.21428e-01 8.43600e-02 2.24374e-01
  2.85714e-01 3.21428e-01 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00
  6.42856e-02 0.00000e+00 0.00000e+00 4.82142e-02 7.14284e-02 0.00000e+00
  9.45376e-02]
 [2.48952e-01 1.68645e-01 4.27346e-02 1.68645e-01 8.43224e-02 0.00000e+00
  6.74579e-02 1.63178e-04 2.24860e-01 3.37290e-01 1.49371e-01 1.12430e-01
  8.43224e-02 2.52967e-01 1.12430e-01 1.68645e-01 7.31949e-02 3.08958e-01
  3.74766e-02 0.00000e+00 2.89105e-01 3.37290e-01 3.37290e-01 1.12430e-01
  8.43224e-02 1.68645e-01 2.24860e-01 8.43224e-02 1.31168e-01 2.24860e-02
  1.38884e-01]
 [1.44845e-01 3.20185e-01 2.91306e-01 1.60092e-01 1.14352e-02 8.00462e-02
  2.56148e-01 4.64709e-04 3.20185e-01 3.20185e-01 2.83592e-01 1.06728e-01
  0.00000e+00 8.00462e-02 2.13456e-01 3.20185e-01 1.82264e-02 3.88258e-03
  2.13456e-01 3.20185e-01 9.14813e-02 0.00000e+00 1.06728e-01 0.00000e+00
  5.60323e-02 1.60092e-01 2.13456e-01 0.00000e+00 0.00000e+00 0.00000e+00
  0.00000e+00]
 [1.25681e-01 1.75953e-01 3.24953e-01 1.75953e-01 2.51362e-02 2.63930e-01
  7.03812e-02 6.80999e-04 3.51906e-01 0.00000e+00 1.30708e-01 2.34604e-01
  0.00000e+00 2.63930e-01 2.34604e-01 1.75953e-01 3.52092e-02 2.97647e-01
  3.91007e-02 3.51906e-01 0.00000e+00 0.00000e+00 2.34604e-01 0.00000e+00
  7.03812e-02 1.75953e-01 2.34604e-01 7.03812e-02 1.36852e-01 7.03812e-02
  0.00000e+00]
 [8.15457e-02 3.80547e-01 1.33205e-01 1.90273e-01 1.35909e-02 0.00000e+00
  2.28328e-01 1.10463e-03 0.00000e+00 3.80547e-01 5.43638e-02 2.53698e-01
  0.00000e+00 9.51366e-02 1.26849e-01 1.90273e-01 4.92767e-02 2.22140e-01
  3.80547e-01 0.00000e+00 2.71819e-02 0.00000e+00 3.80547e-01 1.26849e-01
  5.70820e-02 1.90273e-01 2.53698e-01 1.90273e-02 4.22829e-02 5.07395e-02
  4.47702e-02]]
```

In [47]:
```
eaNormalized = pd.DataFrame(normalizedX)
eaNormalized.columns = ['Age', 'BusinessTravel', 'DailyRate', 'Department',
       'EducationField', 'EmployeeNumber', 'EnvironmentSatisfaction', 'Gende
       'JobInvolvement', 'JobLevel', 'JobRole', 'JobSatisfaction', 'MaritalS
       'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'OverTime', 'Pe
       'PerformanceRating', 'RelationshipSatisfaction', 'StockOptionLevel',
       'TrainingTimesLastYear', 'WorkLifeBalance', 'YearsAtCompany', 'YearsI
       'YearsSinceLastPromotion', 'YearsWithCurrManager']
eaNormalized['Attrition'] = ea['Attrition']
eaNormalized
```

Out[47]:

	Age	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationFi
0	0.176020	0.321428	0.230084	0.321428	0.000000	0.080357	0.064
1	0.248952	0.168645	0.042735	0.168645	0.084322	0.000000	0.067
2	0.144845	0.320185	0.291306	0.160092	0.011435	0.080046	0.256
3	0.125681	0.175953	0.324953	0.175953	0.025136	0.263930	0.070
4	0.081546	0.380547	0.133205	0.190273	0.013591	0.000000	0.228
...	...	...	...	...	...	...	
1465	0.143500	0.167417	0.187430	0.167417	0.263083	0.083708	0.200
1466	0.172268	0.344537	0.126026	0.172268	0.061524	0.000000	0.206
1467	0.070222	0.327704	0.012433	0.163852	0.035111	0.163852	0.065
1468	0.237796	0.161087	0.212400	0.322175	0.011506	0.161087	0.193
1469	0.132925	0.348929	0.131379	0.174464	0.087232	0.174464	0.209

1470 rows × 32 columns

In [48]:
```
ea = eaNormalized
```

In [49]:
```
1 ea.describe()
```

Out[49]:

	Age	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	E
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.000000	1470.000000	
mean	0.149404	0.269174	0.167707	0.210014	0.097389	0.159918	
std	0.067989	0.113565	0.096300	0.084025	0.095689	0.085229	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.099589	0.188136	0.089458	0.162288	0.012796	0.089827	
50%	0.143253	0.312180	0.165647	0.179955	0.072306	0.167842	
75%	0.195194	0.343330	0.251598	0.296860	0.154730	0.225166	
max	0.346334	0.479337	0.416890	0.408828	0.412158	0.447192	

8 rows × 32 columns

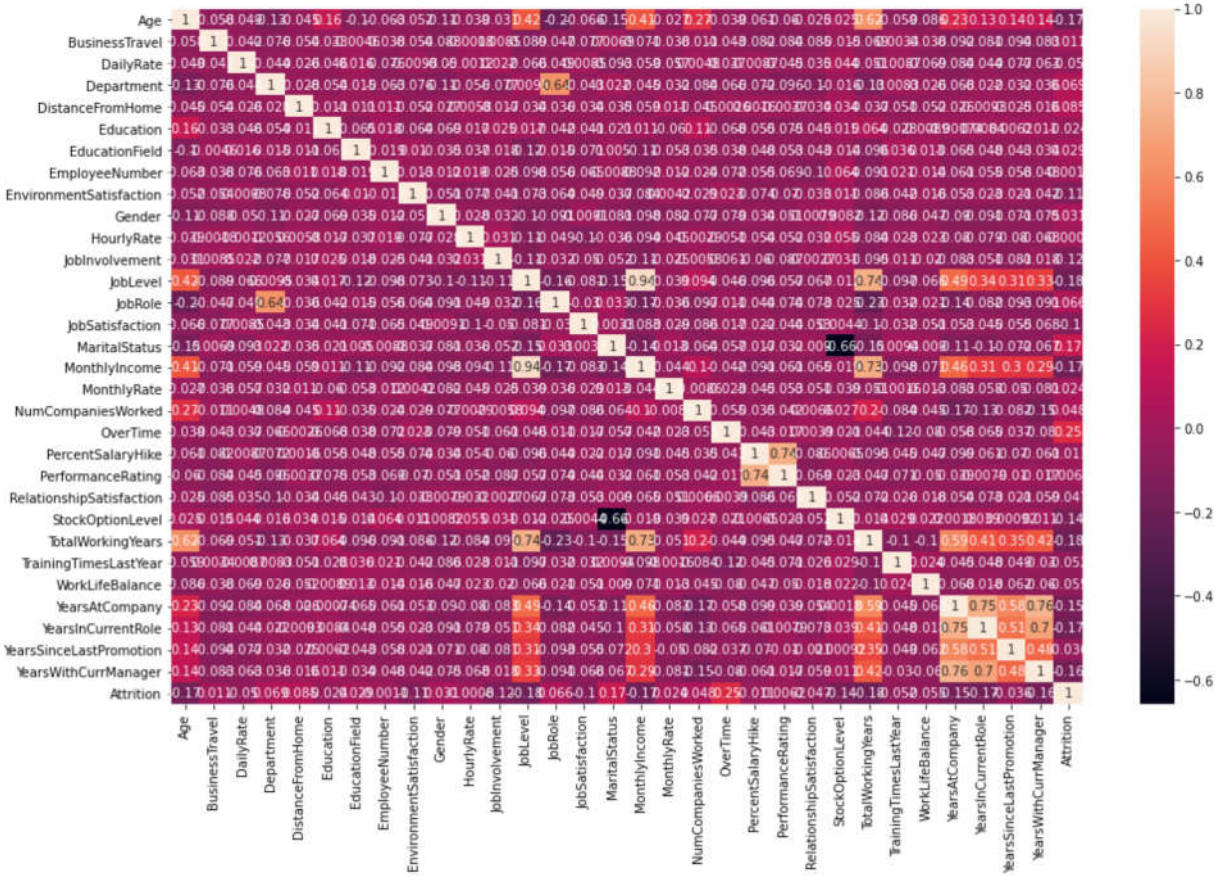
In [50]:

```
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pandas import set_option
data = ea
set_option('display.width', 100)

plt.figure(figsize=(16,10))

sns.heatmap(data.corr(), annot = True)
```

Out[50]: <AxesSubplot:>



In [51]:
```
#to identify which attributes (and combination of attributes)contribute the
# Feature Extraction with RFE
from pandas import read_csv
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
# load data

array = ea.values
X = array[:,0:31]
Y = array[:,31]
# feature extraction
model = LogisticRegression()
rfe = RFE(model, 18)
fit = rfe.fit(X, Y)
print("Num Features: {}".format(fit.n_features_))
print("Selected Features: {}".format(fit.support_))
print("Feature Ranking: {}".format(fit.ranking_))
```

```
Num Features: 18
Selected Features: [ True False False  True  True False False False  True False
False  True
  True False  True  True  True False  True  True False False  True  True
  True False  True False  True  True  True]
Feature Ranking: [ 1 11  3  1  1 12  8 10  1  6  7  1  1 14  1  1  1  9  1  1
4 13  1  1
  1  2  1  5  1  1  1]

C:\Users\HP\anaconda3\lib\site-packages\sklearn\utils\validation.py:70: FutureW
arning: Pass n_features_to_select=18 as keyword args. From version 1.0 (renamin
g of 0.25) passing these as positional arguments will result in an error
  warnings.warn(f"Pass {args_msg} as keyword args. From version "
```

In [52]:
```
ea.columns
```

Out[52]:
```
Index(['Age', 'BusinessTravel', 'DailyRate', 'Department', 'DistanceFromHome',
'Education',
       'EducationField', 'EmployeeNumber', 'EnvironmentSatisfaction', 'Gender',
'HourlyRate',
       'JobInvolvement', 'JobLevel', 'JobRole', 'JobSatisfaction', 'MaritalStat
us',
       'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'OverTime', 'Perce
ntSalaryHike',
       'PerformanceRating', 'RelationshipSatisfaction', 'StockOptionLevel', 'To
talWorkingYears',
       'TrainingTimesLastYear', 'WorkLifeBalance', 'YearsAtCompany', 'YearsInCu
rrentRole',
       'YearsSinceLastPromotion', 'YearsWithCurrManager', 'Attrition'],
      dtype='object')
```

In [53]:
```
viableCols = []
rankList = list(fit.ranking_)
columnList = ['Age', 'BusinessTravel', 'DailyRate', 'Department', 'DistanceF
       'EducationField', 'EmployeeNumber', 'EnvironmentSatisfaction', 'Gende
       'JobInvolvement', 'JobLevel', 'JobRole', 'JobSatisfaction', 'MaritalS
       'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'OverTime', 'Pe
       'PerformanceRating', 'RelationshipSatisfaction', 'StockOptionLevel',
       'TrainingTimesLastYear', 'WorkLifeBalance', 'YearsAtCompany', 'YearsI
       'YearsSinceLastPromotion', 'YearsWithCurrManager', 'Attrition']
for rank, col in zip(rankList, columnList):
    idx = rank
    colname = col
    if idx == 1:
        viableCols.append(colname)
print(viableCols)
```

```
['Age', 'Department', 'DistanceFromHome', 'EnvironmentSatisfaction', 'JobInvolv
ement', 'JobLevel', 'JobSatisfaction', 'MaritalStatus', 'MonthlyIncome', 'NumCo
mpaniesWorked', 'OverTime', 'RelationshipSatisfaction', 'StockOptionLevel', 'To
talWorkingYears', 'WorkLifeBalance', 'YearsInCurrentRole', 'YearsSinceLastPromo
tion', 'YearsWithCurrManager']
```

In [54]:
```
eaModel = ea.loc[:, viableCols] #eaModel is the employee attiriton data that
eaModel
```

Out[54]:

	Age	Department	DistanceFromHome	EnvironmentSatisfaction	JobInvolvement	JobLeve
0	0.176020	0.321428	0.000000	0.107143	0.214285	0.08035
1	0.248952	0.168645	0.084322	0.224860	0.112430	0.08432
2	0.144845	0.160092	0.011435	0.320185	0.106728	0.00000
3	0.125681	0.175953	0.025136	0.351906	0.234604	0.00000
4	0.081546	0.190273	0.013591	0.000000	0.253698	0.00000
...	...	...	...	...	...	...
1465	0.143500	0.167417	0.263083	0.223222	0.334833	0.08370
1466	0.172268	0.172268	0.061524	0.344537	0.114846	0.17226
1467	0.070222	0.163852	0.035111	0.109235	0.327704	0.08192
1468	0.237796	0.322175	0.011506	0.322175	0.107392	0.08054
1469	0.132925	0.174464	0.087232	0.116310	0.348929	0.08723

1470 rows × 18 columns

In [55]:
```
1 eaModel['Attrition'] = ea['Attrition']
2 eaModel
```

Out[55]:

	Age	Department	DistanceFromHome	EnvironmentSatisfaction	JobInvolvement	JobLeve
0	0.176020	0.321428	0.000000	0.107143	0.214285	0.08035
1	0.248952	0.168645	0.084322	0.224860	0.112430	0.08432
2	0.144845	0.160092	0.011435	0.320185	0.106728	0.00000
3	0.125681	0.175953	0.025136	0.351906	0.234604	0.00000
4	0.081546	0.190273	0.013591	0.000000	0.253698	0.00000
...	...	...	...	...	...	..
1465	0.143500	0.167417	0.263083	0.223222	0.334833	0.08370
1466	0.172268	0.172268	0.061524	0.344537	0.114846	0.17226
1467	0.070222	0.163852	0.035111	0.109235	0.327704	0.08192
1468	0.237796	0.322175	0.011506	0.322175	0.107392	0.08054
1469	0.132925	0.174464	0.087232	0.116310	0.348929	0.08723

1470 rows × 19 columns

Classification Models for Attrition

In [56]:
```
# Logistic Regression Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
#from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression



array = eaModel.values # Converting input data to arrays
X = array[:,0:18] # Algorithms ONLY accept Arrays
Y = array[:,18]


num_folds = 10
kfold = KFold(n_splits=num_folds)#, shuffle = True, random_state=7)
model = LogisticRegression(max_iter = 500)

results = cross_val_score(model, X, Y, cv=kfold) # epochs

score = np.mean(results)
print(results)
print(results.mean())


```

```
[0.84354 0.87075 0.87075 0.88435 0.83673 0.82313 0.85714 0.87075 0.84354
 0.89796]
0.8598639455782312
```

In [57]:
```
# LDA Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

array = eaModel.values
X = array[:,0:18]
Y = array[:,18]
num_folds = 10
kfold = KFold(n_splits=10,shuffle = True, random_state=7)
model = LinearDiscriminantAnalysis()
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.8727891156462585
```

In [58]:
```
#KNN Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier

array = eaModel.values
X = array[:,0:18]
Y = array[:,18]
num_folds = 10
kfold = KFold(n_splits=10, random_state=7, shuffle = True)
model = KNeighborsClassifier()
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.8435374149659864
```

In [59]:
```
# Gaussian Naive Bayes Classification
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import GaussianNB

array = eaModel.values
X = array[:,0:18]
Y = array[:,18]
kfold = KFold(n_splits=10, random_state=7, shuffle=True)
model = GaussianNB()
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.7993197278911565
```

In [60]:
```
'''LDA was the best in predictibg the attrition rate because
 the data had a Gaussian distribution for the numerical input variables
 and there was no multicolinearity between the data
 Logistic regression also performed well is prediction
 because the value of the target variable is categorical in nature.
and has binary output which is either a 0 or 1.
KNN was also used because data is clearly labeled (0/1)
and there was no missing data or outliers
Naive Bayes didn't perform as the others because of its
assumption of independence among predictors
i.e the presence of a particular feature in a class
is unrelated to the presence of any other feature'''
```

Out[60]:
```
"LDA was the best in predictibg the attrition rate because \n the data had a Ga
ussian distribution for the numerical input variables\n and there was no multic
olinearity between the data\n Logistic regression also performed well is predic
tion \n because the value of the target variable is categorical in nature. \nan
d has binary output which is either a 0 or 1.\nKNN was also used because data i
s clearly labeled (0/1) \nand there was no missing data or outliers\nNaive Baye
s didn't perform as the others because of its\nassumption of independence among
predictors \ni.e the presence of a particular feature in a class \nis unrelated
to the presence of any other feature"
```

In [ ]:
```
1
```

In [ ]:
```
1
```