About the Dataset

In [1]:
```
'''
Data source: https://www.kaggle.com/datasets/whenamancodes/hr-employee-attri
There are 1470 records in the data, each with 35 attributes, including 34 fe
one target attribute. Attrition is the target attribute and has the values "
other dependent attributes.
'''
```

Out[1]:
```
'\nData source: https://www.kaggle.com/datasets/whenamancodes/hr-employee-at
trition\nThere are 1470 records in the data, each with 35 attributes, includ
ing 34 feature attributes and \none target attribute. Attrition is the targe
t attribute and has the values "Yes" and "No," based on \nother dependent at
tributes. \n'
```

In [2]:
```
#Importing the necessary libraries and modules.
import pandas as pd                    #For importing and doing data-preprocess
import numpy as np                     #Allows us to create and manage the arra
import matplotlib.pyplot as plt        #Allows us to plot charts and graphs
import seaborn as sns                  #Allows us to plot graphs and charts
import tensorflow as tf                #For Artificial Neural Networks
import warnings
warnings.filterwarnings('ignore')
```

Data Extraction and Exploratory Data Analysis on the dataset

In [3]:
```
'''
Reading the dataset selected into a pandas dataframe using pandas library. S
we use the read_csv on from pandas object, from its library
'''

eda_df = pd.read_csv(r"C:\Users\cvvis\OneDrive\Desktop\Our DataMining Projec
```

In [4]:
```
#Checking the number of rows and columns in our dataset
eda_df.shape

#We understand that there are 1470 rows and 35 columns from the output below
#rows = 1470
#columns = 35
```

Out[4]:
```
(1470, 35)
```

In [5]:
```
#Checking some brief information about the schema

'''
We observe that a few columns here have missing values, since the count of t
equal to the total number of rows.
'''

'''
We can see that the columns are either Integer type or Object type. Object t
'''

eda_df.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Age                       1467 non-null   float64
 1   Attrition                 1470 non-null   object 
 2   BusinessTravel            1470 non-null   object 
 3   DailyRate                 1466 non-null   float64
 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             1465 non-null   float64
 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         1457 non-null   float64
 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: float64(4), int64(22), object(9)
memory usage: 402.1+ KB
```

In [6]:

```python
#Checking the first few rows to get a glimpse of how the data is
pd.set_option('display.max_columns', None)
eda_df.head()
```

Out[6]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education
0	NaN	Yes	Travel_Rarely	1102.0	Sales	1	2
1	49.0	No	Travel_Frequently	NaN	Research & Development	8	1
2	37.0	Yes	Travel_Rarely	1373.0	Research & Development	2	2
3	33.0	No	Travel_Frequently	1392.0	Research & Development	3	4
4	27.0	No	Travel_Rarely	591.0	Research & Development	2	1

In [7]:
```
#Doing a describe on the dataset to see how the statistical measures are.
'''
We can get the common statistical measures like count, mean, min, max, std e
this gives the results only to the numerical columns or the columns that are
'''

eda_df.describe()
```

Out[7]:

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	Employ
count	1467.000000	1466.000000	1470.000000	1470.000000	1470.0	14
mean	36.981595	803.036835	9.192517	2.912925	1.0	10
std	9.210236	403.611860	8.106864	1.024165	0.0	6
min	18.000000	102.000000	1.000000	1.000000	1.0	
25%	30.000000	466.250000	2.000000	2.000000	1.0	4
50%	36.000000	802.000000	7.000000	3.000000	1.0	10
75%	43.000000	1157.750000	14.000000	4.000000	1.0	15
max	70.000000	1499.000000	29.000000	5.000000	1.0	20

In [8]:
```
#To describe the statistical analysis for the columns of the dataframe that
'''
Here, we get the common statistics such as Unique, top etc... for columns in
the Attrition column has two unique values, and the same applies with the Ge
Business Travel have 3 unique values respectively.
'''

eda_df.describe(include = object)
```

Out[8]:

	Attrition	BusinessTravel	Department	EducationField	Gender	JobRole	MaritalSta
count	1470	1470	1470	1470	1470	1470	14
unique	2	3	3	6	2	9	
top	No	Travel_Rarely	Research & Development	Life Sciences	Male	Sales Executive	Marr
freq	1233	1043	961	606	882	326	6

In [9]:
```
#To get the column wise memory consumption and usage for the dataframe
eda_df.memory_usage()
```

Out[9]:
```
Index                          128
Age                          11760
Attrition                    11760
BusinessTravel               11760
DailyRate                    11760
Department                   11760
DistanceFromHome             11760
Education                    11760
EducationField               11760
EmployeeCount                11760
EmployeeNumber               11760
EnvironmentSatisfaction      11760
Gender                       11760
HourlyRate                   11760
JobInvolvement               11760
JobLevel                     11760
JobRole                      11760
JobSatisfaction              11760
MaritalStatus                11760
MonthlyIncome                11760
MonthlyRate                  11760
NumCompaniesWorked           11760
Over18                       11760
OverTime                     11760
PercentSalaryHike            11760
PerformanceRating            11760
RelationshipSatisfaction     11760
StandardHours                11760
StockOptionLevel             11760
TotalWorkingYears            11760
TrainingTimesLastYear        11760
WorkLifeBalance              11760
YearsAtCompany               11760
YearsInCurrentRole           11760
YearsSinceLastPromotion      11760
YearsWithCurrManager         11760
dtype: int64
```

In [ ]:

In [10]:
```
'''
Classification of columns as Quantitative - Discrete/Continuous, Qualitative

Qualitative:
Attrition
BusinessTravel
Department
EducationField
Gender
JobRole
MaritalStatus
Over18
OverTime


Quantitative:
Age
DailyRate
DistanceFromHome
Education
EmployeeCount
EmployeeNumber
EnvironmentSatisfaction
HourlyRate
```

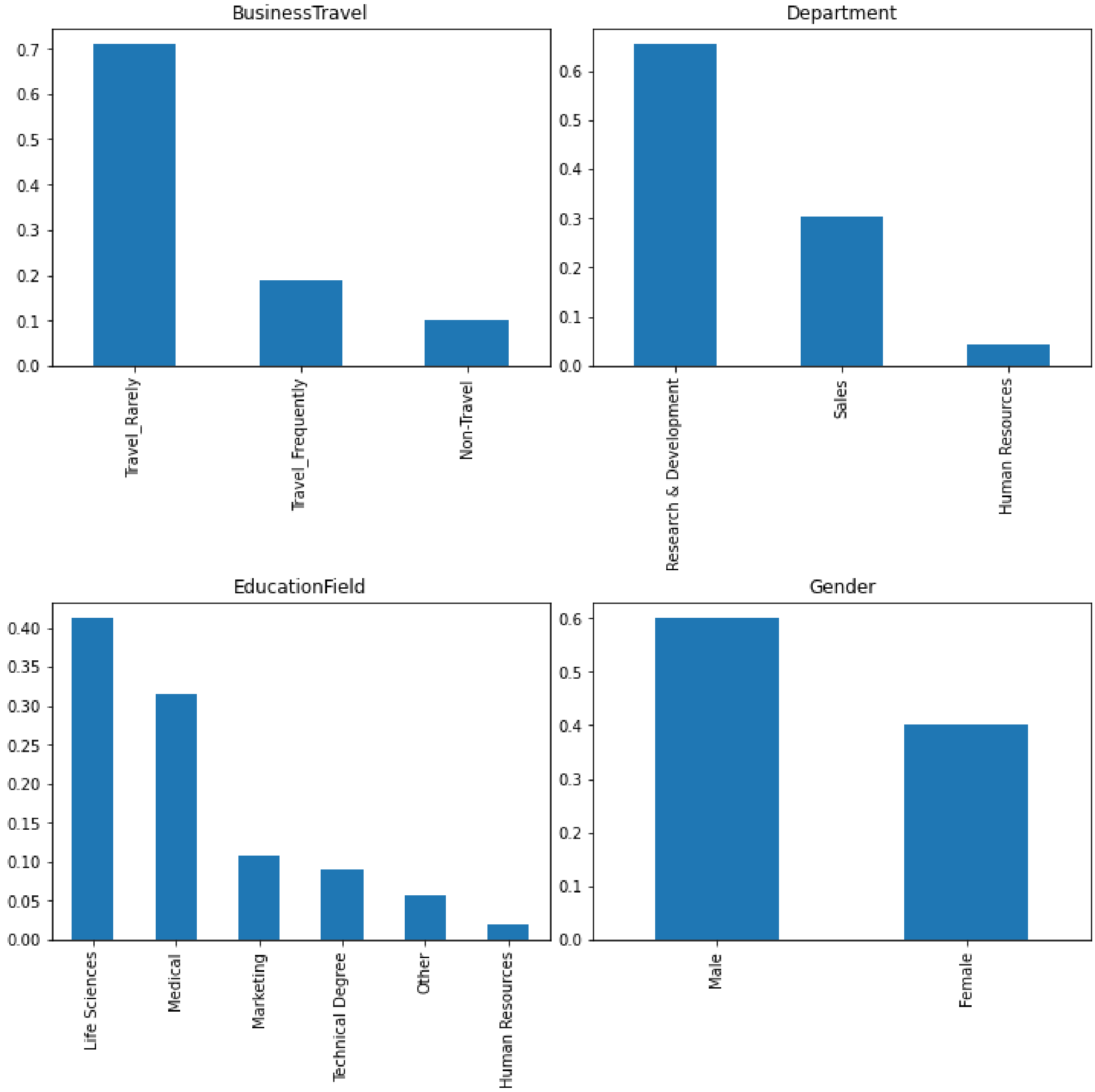
```
JobInvolvement
JobLevel
JobSatisfaction
MonthlyIncome
MonthlyRate
NumCompaniesWorked
PercentSalaryHike
PerformanceRating
RelationshipSatisfaction
StandardHours
StockOptionLevel
TotalWorkingYears
TrainingTimesLastYear
WorkLifeBalance
YearsAtCompany
YearsInCurrentRole
YearsSinceLastPromotion
YearsWithCurrManager
'''
```

Out[10]:

```
'\nClassification of columns as Quantitative - Discrete/Continuous, Qualitat
ive - Categorical and Ordinal.\n\nQualitative:\nAttrition\nBusinessTravel\nD
epartment\nEducationField\nGender\nJobRole\nMaritalStatus\nOver18\nOverTime
\n\n\nQuantitative:\nAge\nDailyRate\nDistanceFromHome\nEducation\nEmployeeCo
unt\nEmployeeNumber\nEnvironmentSatisfaction\nHourlyRate\nJobInvolvement\nJo
bLevel\nJobSatisfaction\nMonthlyIncome\nMonthlyRate\nNumCompaniesWorked\nPer
centSalaryHike\nPerformanceRating\nRelationshipSatisfaction\nStandardHours\n
StockOptionLevel\nTotalWorkingYears\nTrainingTimesLastYear\nWorkLifeBalance
\nYearsAtCompany\nYearsInCurrentRole\nYearsSinceLastPromotion\nYearsWithCurr
Manager\n'
```

In [11]:

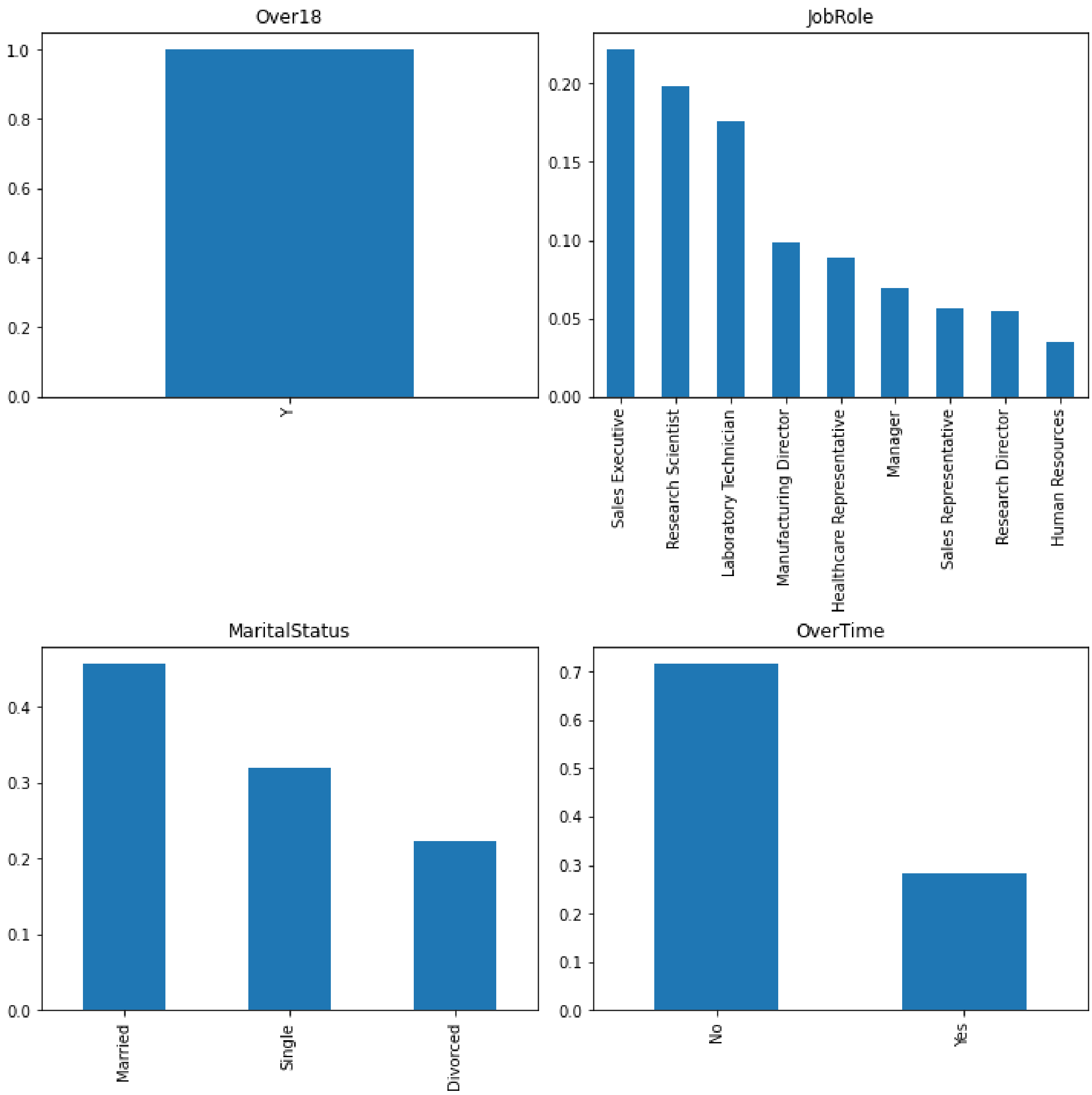
```
'''
Performing Univariate analysis on the Qualitative columns
'''



plt.subplot(221)
eda_df.BusinessTravel.value_counts(normalize = True).plot(kind = 'bar', titl
plt.tight_layout(pad = 0.5)

plt.subplot(222)
eda_df.Department.value_counts(normalize = True).plot(kind = 'bar', title =
plt.tight_layout(pad = 0.5)

plt.subplot(223)
eda_df.EducationField.value_counts(normalize = True).plot(kind = 'bar', titl
plt.tight_layout(pad = 0.5)


'''
We see the Male to Female ratio of employees is 60 to 40.
'''
plt.subplot(224)
eda_df.Gender.value_counts(normalize = True).plot(kind = 'bar', title = "Gen
plt.tight_layout(pad = 0.5)
```



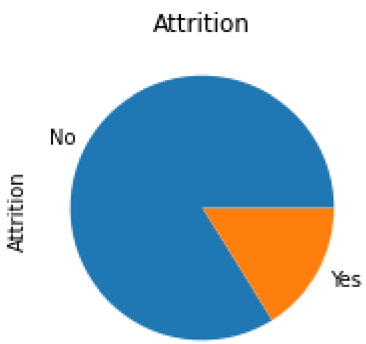
In [12]:

```
'''
We notice that every employee here is Over 18 years of age. Hence this colum
variable prediction.
'''



plt.subplot(221)
eda_df.Over18.value_counts(normalize = True).plot(kind = 'bar', title = "Ove
plt.tight_layout(pad = 0.7)

plt.subplot(222)
eda_df.JobRole.value_counts(normalize = True).plot(kind = 'bar', title = "Jo
plt.tight_layout(pad = 0.7)

'''
From the distribution, we see that around 50% are married, 30% are unmarried
'''
plt.subplot(223)
eda_df.MaritalStatus.value_counts(normalize = True).plot(kind = 'bar', title
plt.tight_layout(pad = 0.7)


'''
We notice that about 70% employees work overtime.
'''
plt.subplot(224)
```

```
eda_df.OverTime.value_counts(normalize = True).plot(kind = 'bar', title = "C
plt.tight_layout(pad = 0.7)
```

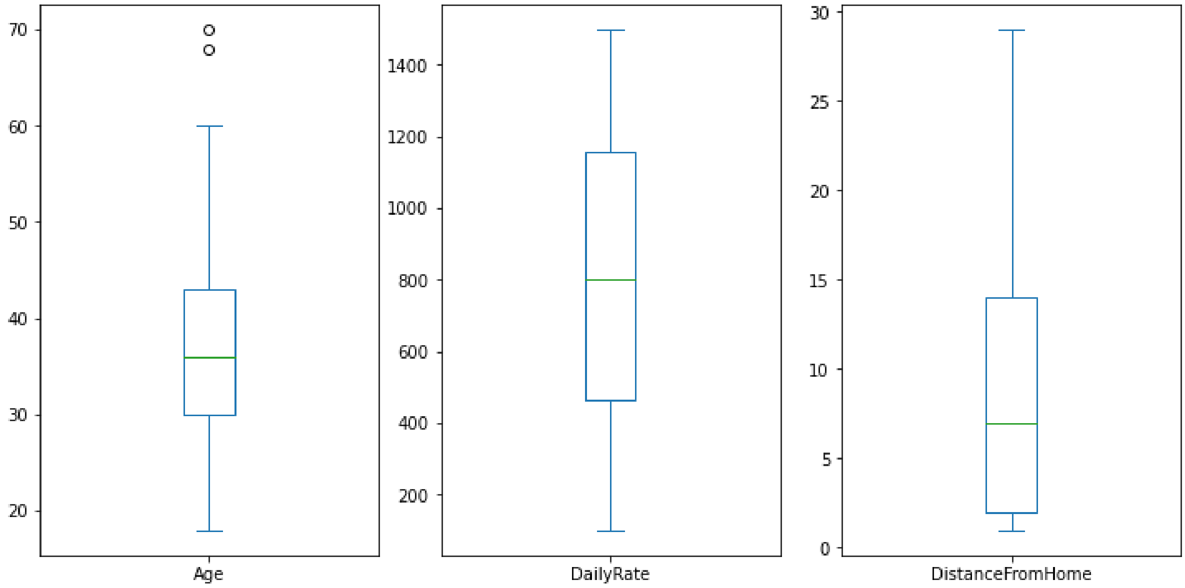


In [13]:
```
'''
From the target value distribution, we see that more than 3/4th of employees
'''
plt.subplot(221)
eda_df.Attrition.value_counts(normalize = True).plot(kind = 'pie', title = "
plt.tight_layout(pad = 0.5)
```

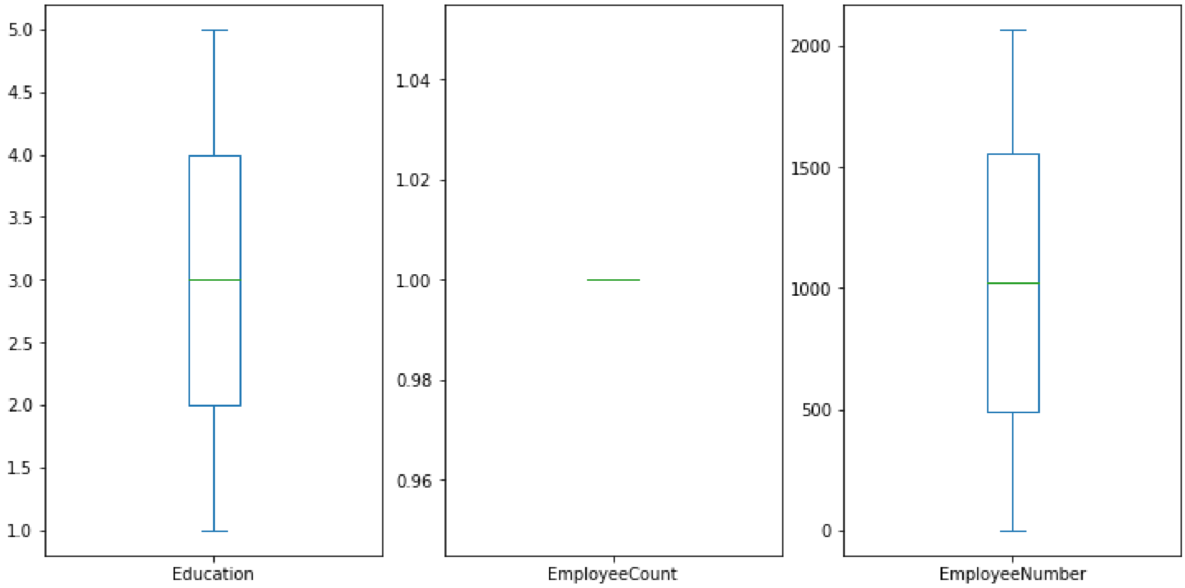


In [14]:
```
'''
Performing Univariate analysis on a few Quantitative columns. We will use Bo
information and help us find the outliers.
```

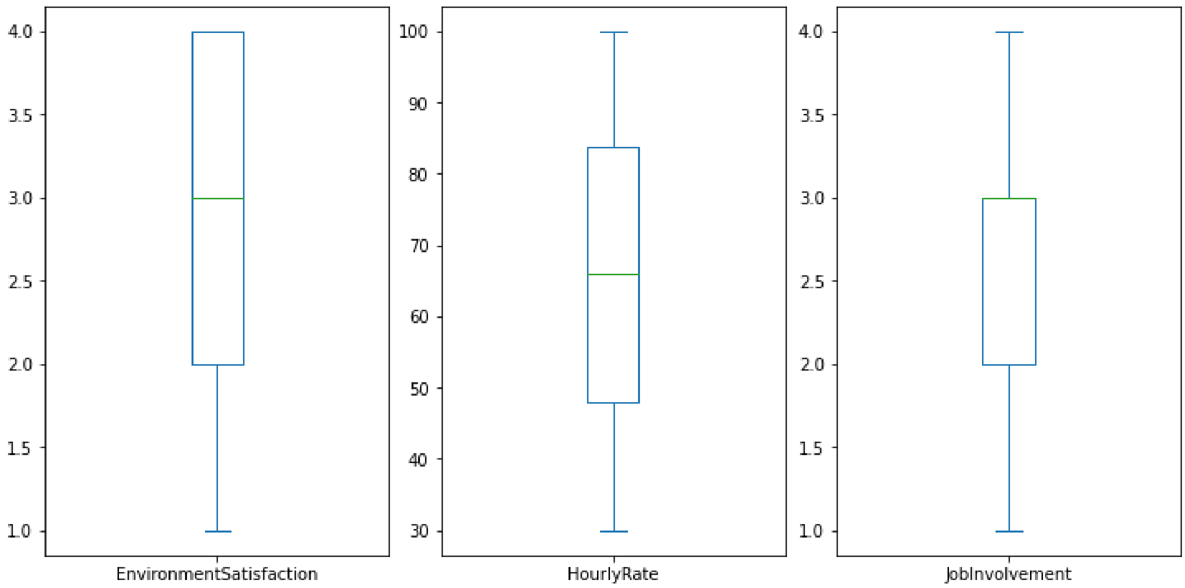
```
'''

'''
We see a couple of outliers in the age where the age of employees is close t
'''
plt.subplot(131)
eda_df.Age.plot.box(figsize=(10,5))
plt.tight_layout(pad=0.5)

plt.subplot(132)
eda_df.DailyRate.plot.box()
plt.tight_layout(pad=0.5)

plt.subplot(133)
eda_df.DistanceFromHome.plot.box()
plt.tight_layout(pad=0.5)
```



In [15]:
```
plt.subplot(131)
eda_df.Education.plot.box(figsize=(10,5))
plt.tight_layout(pad=0.5)

plt.subplot(132)
eda_df.EmployeeCount.plot.box()
plt.tight_layout(pad=0.5)

plt.subplot(133)
eda_df.EmployeeNumber.plot.box()
plt.tight_layout(pad=0.5)
```

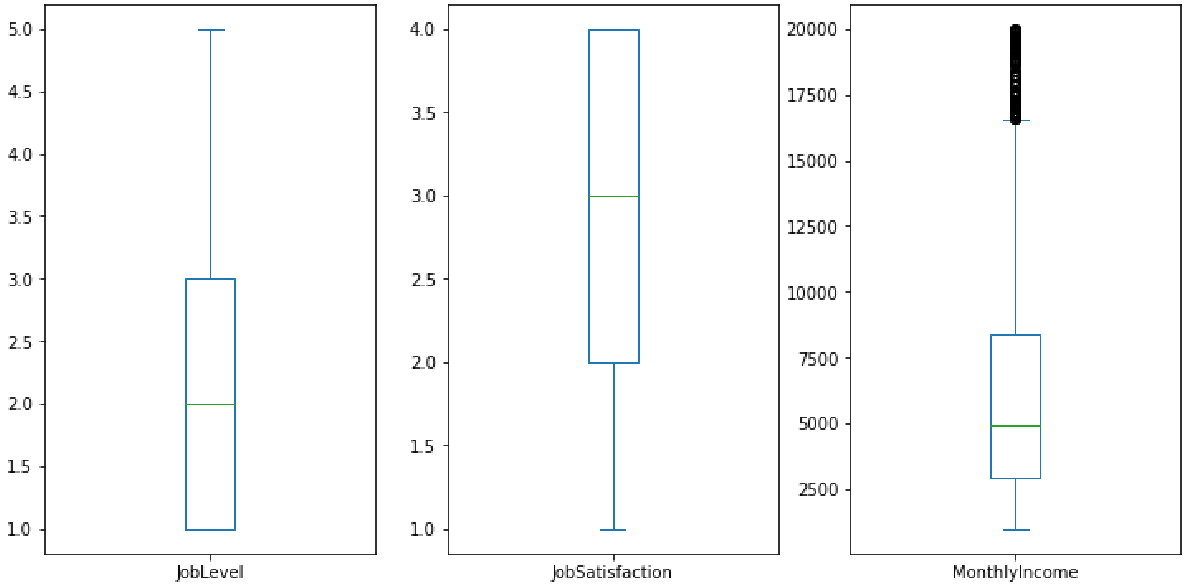


In [16]:
```
plt.subplot(131)
eda_df.EnvironmentSatisfaction.plot.box(figsize=(10,5))
plt.tight_layout(pad=0.5)

plt.subplot(132)
eda_df.HourlyRate.plot.box()
plt.tight_layout(pad=0.5)

plt.subplot(133)
eda_df.JobInvolvement.plot.box()
plt.tight_layout(pad=0.5)
```



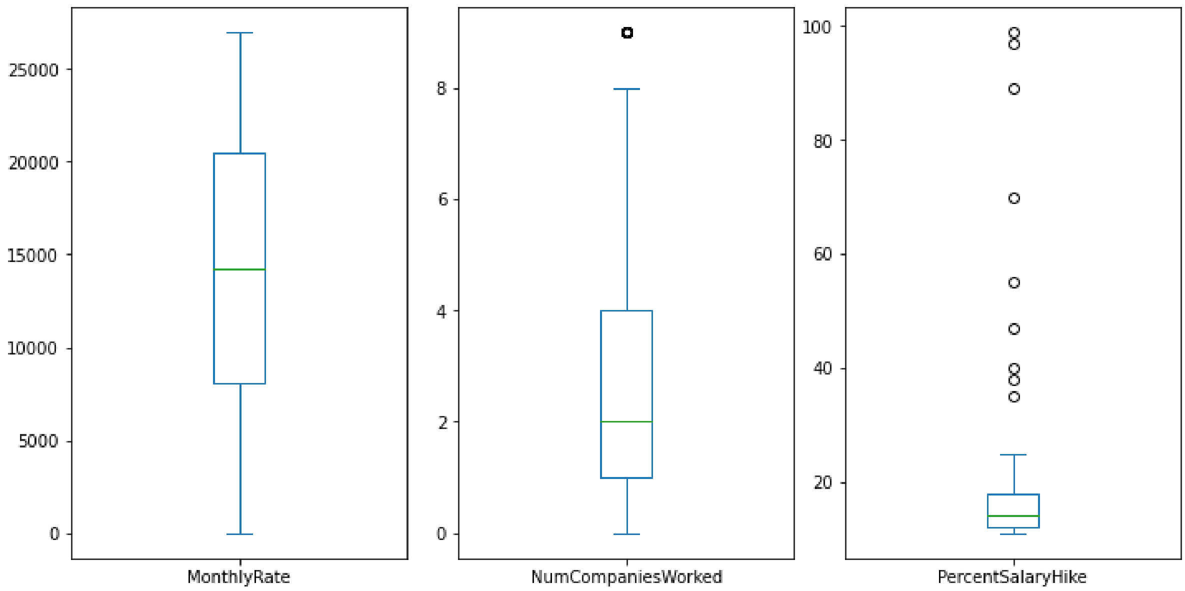
In [17]:
```
plt.subplot(131)
eda_df.JobLevel.plot.box(figsize=(10,5))
plt.tight_layout(pad=0.5)

plt.subplot(132)
eda_df.JobSatisfaction.plot.box()
plt.tight_layout(pad=0.5)

'''
There are a few employees whose monthly salary is extremely high and not in
'''
plt.subplot(133)
```

```
eda_df.MonthlyIncome.plot.box()
plt.tight_layout(pad=0.5)
```



In [18]:

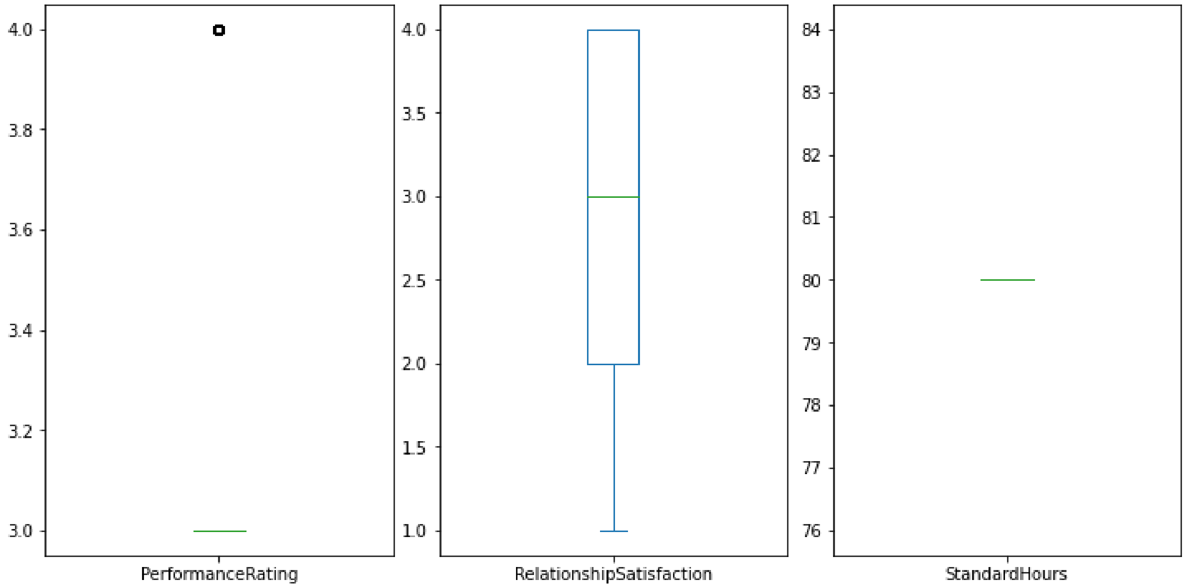
```
plt.subplot(131)
eda_df.MonthlyRate.plot.box(figsize=(10,5))
plt.tight_layout(pad=0.5)

plt.subplot(132)
eda_df.NumCompaniesWorked.plot.box()
plt.tight_layout(pad=0.5)

'''
A few employees received more than 50% percent hike. This shows the performa
'''
plt.subplot(133)
eda_df.PercentSalaryHike.plot.box()
plt.tight_layout(pad=0.5)
```



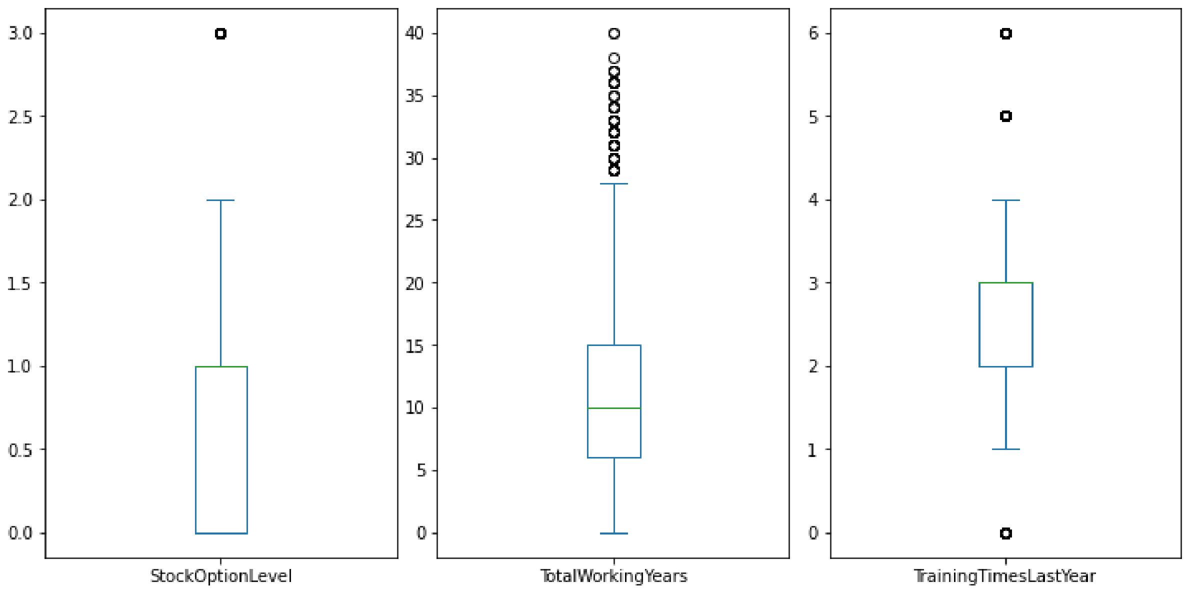
In [19]:

```
plt.subplot(131)
eda_df.PerformanceRating.plot.box(figsize=(10,5))
plt.tight_layout(pad=0.5)

plt.subplot(132)
eda_df.RelationshipSatisfaction.plot.box()
plt.tight_layout(pad=0.5)
```

```
plt.subplot(133)
eda_df.StandardHours.plot.box()
plt.tight_layout(pad=0.5)
```

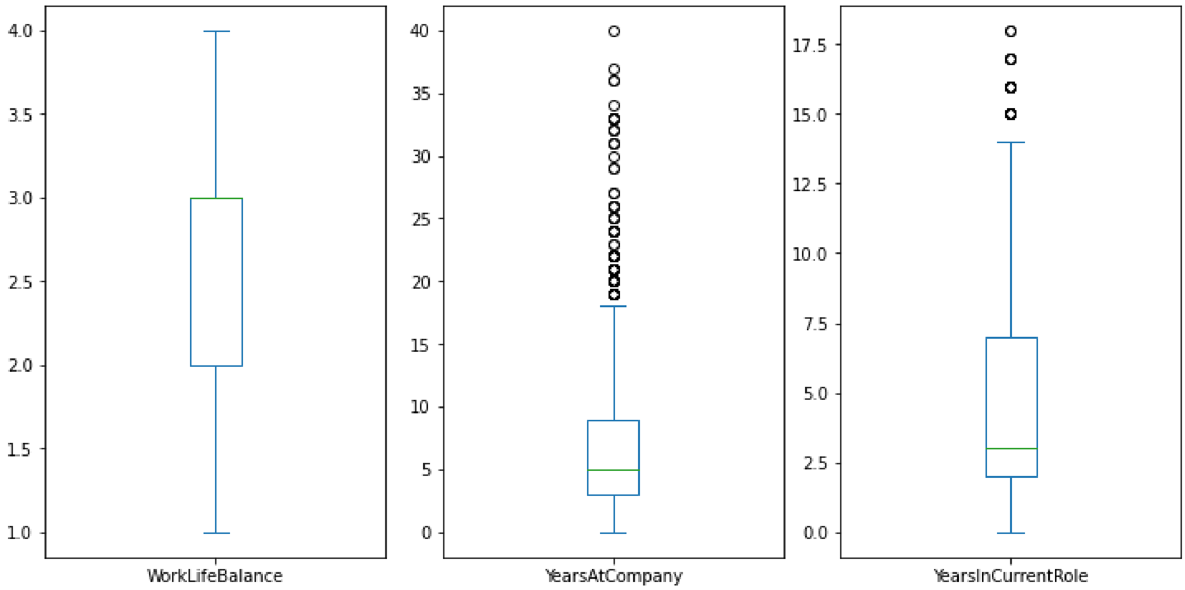


In [20]:
```
plt.subplot(131)
eda_df.StockOptionLevel.plot.box(figsize=(10,5))
plt.tight_layout(pad=0.5)

'''
A few employees have a total experience of >30 years of age.
'''
plt.subplot(132)
eda_df.TotalWorkingYears.plot.box()
plt.tight_layout(pad=0.5)

plt.subplot(133)
eda_df.TrainingTimesLastYear.plot.box()
plt.tight_layout(pad=0.5)
```

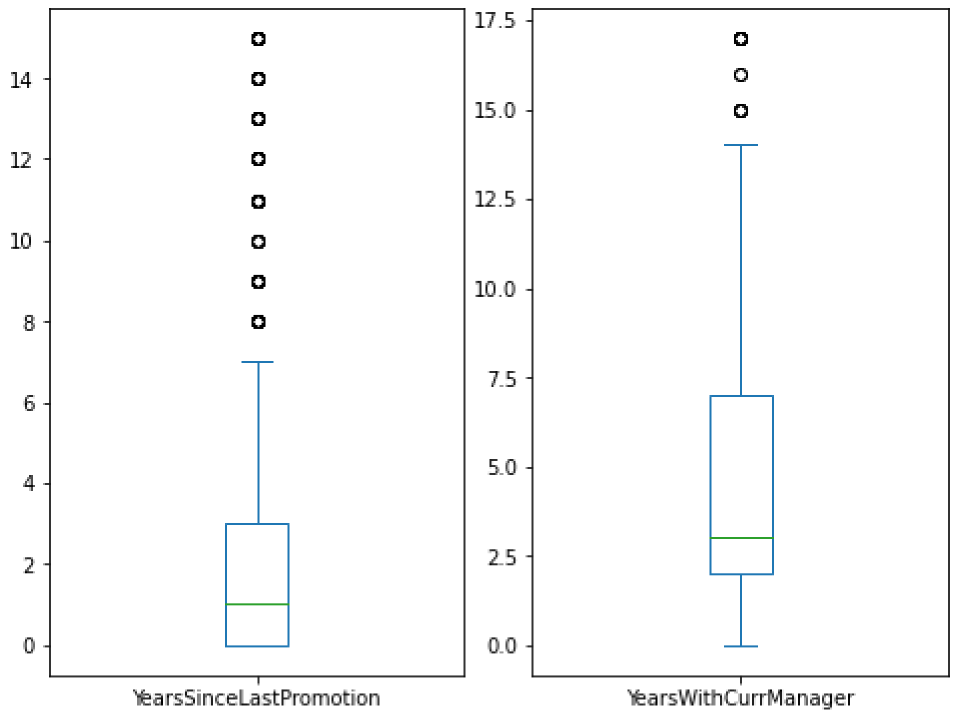


In [21]:
```
plt.subplot(131)
eda_df.WorkLifeBalance.plot.box(figsize=(10,5))
plt.tight_layout(pad=0.5)

plt.subplot(132)
```

```
eda_df.YearsAtCompany.plot.box()
plt.tight_layout(pad=0.5)

'''
There are around 4 employees who have been working for around  15 years in t
'''
plt.subplot(133)
eda_df.YearsInCurrentRole.plot.box()
plt.tight_layout(pad=0.5)
```



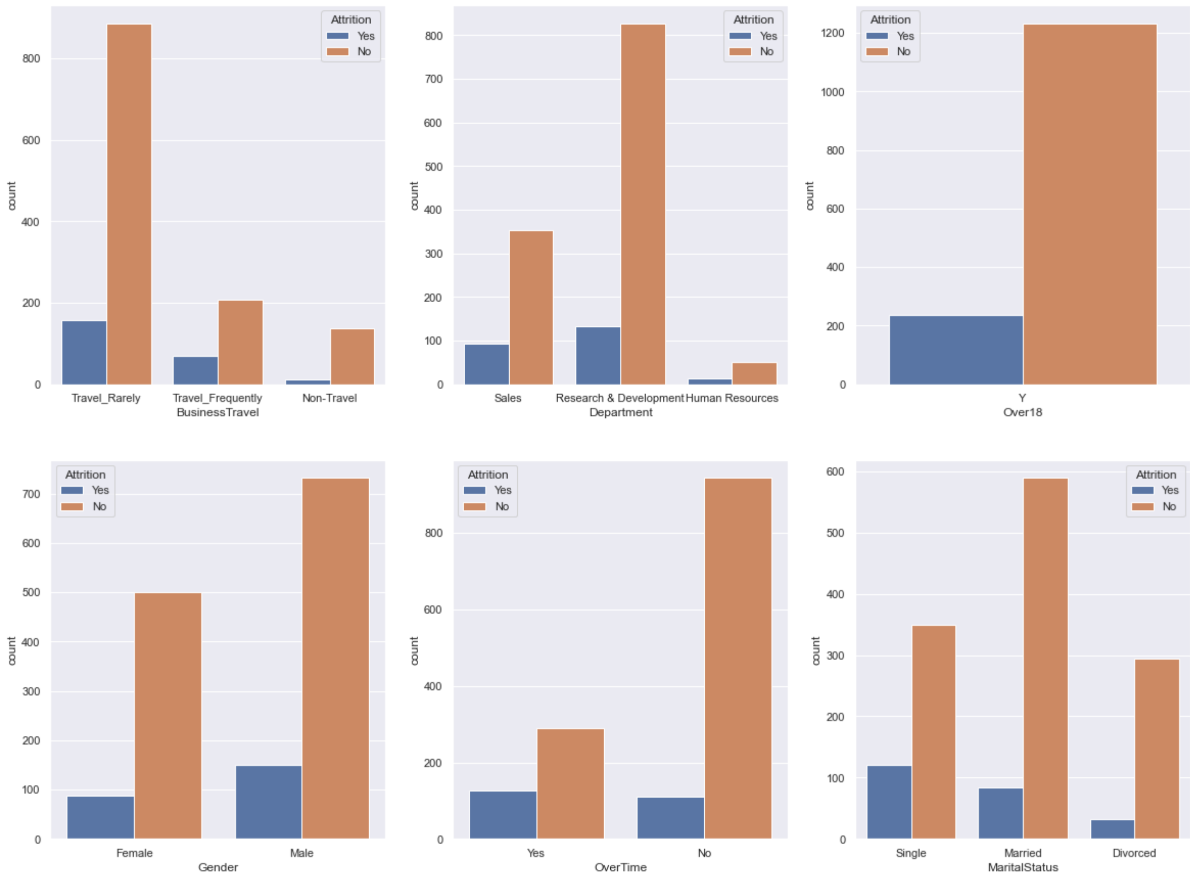
In [22]:
```
plt.subplot(131)
eda_df.YearsSinceLastPromotion.plot.box(figsize=(10,5))
plt.tight_layout(pad=0.5)

'''
Around 3 employees are under the same manager for more than 15 years.
'''
plt.subplot(132)
eda_df.YearsWithCurrManager.plot.box()
plt.tight_layout(pad=0.5)
```



In [ ]:

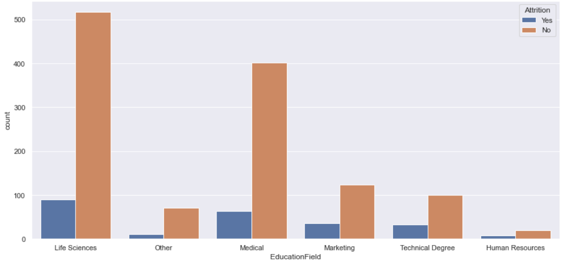
In [23]:
```
#Performing Bi-variate analysis on some selected features and the target col

sns.set(rc={'figure.figsize':(20,15)})
plt.subplot(231)
sns.countplot(x = "BusinessTravel", hue='Attrition', data = eda_df)

plt.subplot(232)
sns.countplot(x = "Department", hue='Attrition', data = eda_df)

plt.subplot(233)
sns.countplot(x = "Over18", hue='Attrition', data = eda_df)

plt.subplot(234)
sns.countplot(x = "Gender", hue='Attrition', data = eda_df)

plt.subplot(235)
sns.countplot(x = "OverTime", hue='Attrition', data = eda_df)

plt.subplot(236)
sns.countplot(x = "MaritalStatus", hue='Attrition', data = eda_df)
```

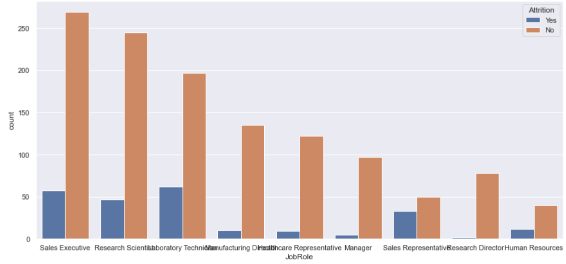
Out[23]: <AxesSubplot:xlabel='MaritalStatus', ylabel='count'>



In [24]:
```
sns.set(rc={'figure.figsize':(50,15)})

plt.subplot(231)
sns.countplot(x = "EducationField", hue='Attrition', data = eda_df)

plt.subplot(232)
sns.countplot(x = "JobRole", hue='Attrition', data = eda_df)
```

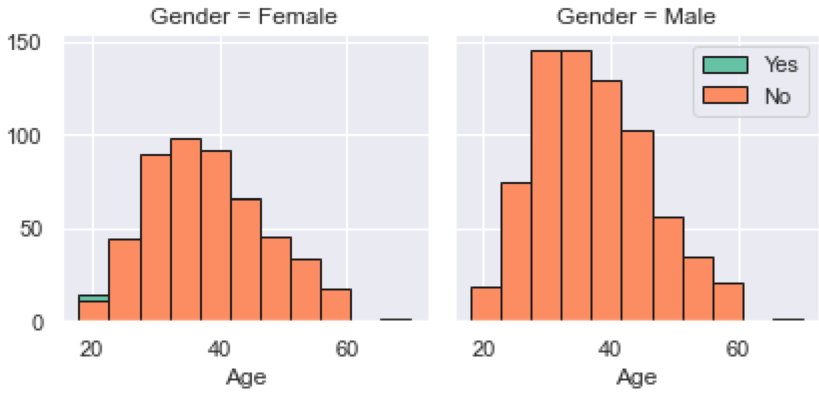
Out[24]: <AxesSubplot:xlabel='JobRole', ylabel='count'>





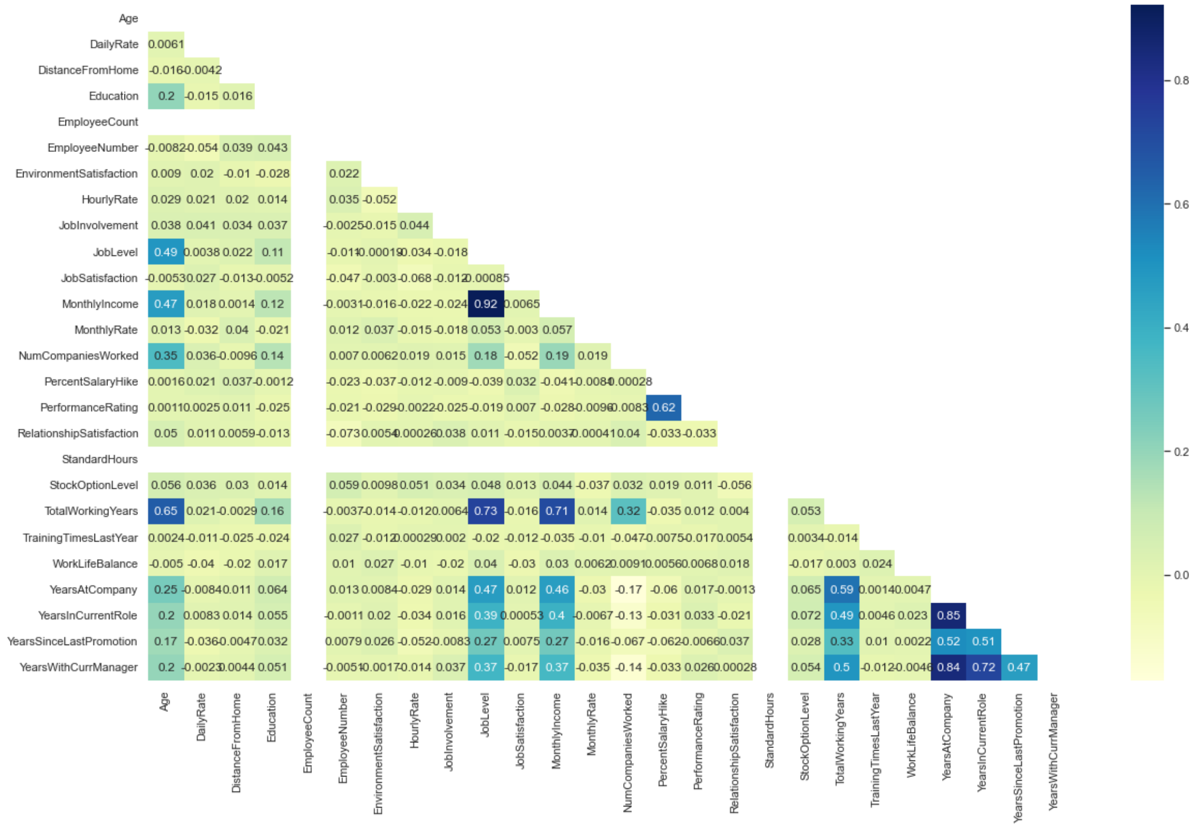
In [ ]:

In [25]:
```
#Performing Multi-variate analysis on some selected features and the target

bins = np.linspace(eda_df.Age.min(), eda_df.Age.max(),12)
graph = sns.FacetGrid(eda_df, col="Gender", hue="Attrition", palette="Set2",
graph.map(plt.hist, 'Age', bins=bins, ec="k")
graph.axes[-1].legend()
plt.show()
```



In [ ]:

In [26]:
```
#Correlation

'''
We find the correlation between the numerical values from the selected datas
Below is the representation of the correlation matrix after eliminating the

Here, we are considering the Spearman Rank correlation as it does not assume
Pearson's correlation.
'''


'''
For example, we see a positive correlation between Monthly income and Job le
'''
correlation_matrix = eda_df.corr(method="spearman")
mask = np.zeros_like(correlation_matrix)
mask[np.triu_indices_from(mask)] = True
with sns.axes_style("white"):
    f, ax = plt.subplots(figsize=(20,12))
    ax = sns.heatmap(correlation_matrix,
mask = mask,annot = True, cmap = "YlGnBu")
```



In [27]:
```
eda_df.corr(method="spearman")
```

Out[27]:

	Age	DailyRate	DistanceFromHome	Education	EmployeeCo
Age	1.000000	0.006100	-0.015531	0.203437	N
DailyRate	0.006100	1.000000	-0.004223	-0.014752	N
DistanceFromHome	-0.015531	-0.004223	1.000000	0.015708	N
Education	0.203437	-0.014752	0.015708	1.000000	N
EmployeeCount	NaN	NaN	NaN	NaN	N
EmployeeNumber	-0.008202	-0.054498	0.038906	0.042815	N
EnvironmentSatisfaction	0.008979	0.020453	-0.010401	-0.027625	N
HourlyRate	0.029170	0.020663	0.020446	0.014432	N
JobInvolvement	0.037507	0.041386	0.034430	0.037231	N
JobLevel	0.492290	0.003772	0.022148	0.107419	N
JobSatisfaction	-0.005349	0.027012	-0.013078	-0.005175	N
MonthlyIncome	0.472785	0.018142	0.001434	0.119624	N
MonthlyRate	0.013290	-0.031721	0.039806	-0.021201	N
NumCompaniesWorked	0.347198	0.035522	-0.009592	0.135103	N
PercentSalaryHike	0.001584	0.020960	0.037281	-0.001177	N
PerformanceRating	0.001140	0.002504	0.011320	-0.025081	N
RelationshipSatisfaction	0.050026	0.011456	0.005852	-0.013173	N
StandardHours	NaN	NaN	NaN	NaN	N
StockOptionLevel	0.055974	0.036329	0.030190	0.013794	N
TotalWorkingYears	0.654771	0.021222	-0.002912	0.162177	N
TrainingTimesLastYear	0.002378	-0.010602	-0.024848	-0.023749	N
WorkLifeBalance	-0.005005	-0.039908	-0.020402	0.017350	N
YearsAtCompany	0.252086	-0.008357	0.010513	0.064196	N
YearsInCurrentRole	0.198456	0.008342	0.013708	0.054567	N
YearsSinceLastPromotion	0.174579	-0.035929	-0.004685	0.032203	N
YearsWithCurrManager	0.196944	-0.002273	0.004448	0.051292	N

In [ ]:

```
```

Data Preprocessing

Positioning the target column to the end of the dataframe.

In [28]:

```
pre_processing_df = eda_df
pre_processing_df.head()
```

Out[28]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education
0	NaN	Yes	Travel_Rarely	1102.0	Sales	1	2
1	49.0	No	Travel_Frequently	NaN	Research & Development	8	1
2	37.0	Yes	Travel_Rarely	1373.0	Research & Development	2	2
3	33.0	No	Travel_Frequently	1392.0	Research & Development	3	4
4	27.0	No	Travel_Rarely	591.0	Research & Development	2	1

In [29]:

```
new_cols = ["Age", "BusinessTravel", "DailyRate", "Department", "DistanceFro
pre_processing_df = pre_processing_df[new_cols]
pre_processing_df.head()
```

Out[29]:

	Age	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	Education
0	NaN	Travel_Rarely	1102.0	Sales	1	2	Life Scie
1	49.0	Travel_Frequently	NaN	Research & Development	8	1	Life Scie
2	37.0	Travel_Rarely	1373.0	Research & Development	2	2	(
3	33.0	Travel_Frequently	1392.0	Research & Development	3	4	Life Scie
4	27.0	Travel_Rarely	591.0	Research & Development	2	1	Me

Feature Selection

In [30]:

```
'''
From the Univariate analysis, we understand that the columns 'Over18' and 'E
dependent variable. Hence we need to eliminate them to avoid the Curse Of Di
'''

pre_processing_df = pre_processing_df.drop(columns = ['Over18','EmployeeNumb
pre_processing_df.shape
```

Out[30]: (1470, 33)

Working on Missing Data

In [31]:

```
#To get the total null/missing values in the entire dataframe
pre_processing_df.isnull().sum().sum()
```

Out[31]: 25

In [32]:

```
#To see the sum of column wise null values in the dataframe
pre_processing_df.isnull().sum()
```

Out[32]:
```
Age                         3
BusinessTravel              0
DailyRate                   4
Department                  0
DistanceFromHome            0
Education                   0
EducationField              0
EmployeeCount               0
EnvironmentSatisfaction     0
Gender                      0
HourlyRate                  0
JobInvolvement              0
JobLevel                    0
JobRole                     0
JobSatisfaction             0
MaritalStatus               0
MonthlyIncome               5
MonthlyRate                 0
NumCompaniesWorked          0
OverTime                    0
PercentSalaryHike          13
PerformanceRating           0
RelationshipSatisfaction    0
StandardHours               0
StockOptionLevel            0
TotalWorkingYears           0
TrainingTimesLastYear       0
WorkLifeBalance             0
YearsAtCompany              0
YearsInCurrentRole          0
YearsSinceLastPromotion     0
YearsWithCurrManager        0
Attrition                   0
dtype: int64
```

In [33]:
```
'''
Here, we replace the missing values as follows:
1. For numerical columns, we replace the missing values with the Mean of the
2. For string columns, we replace the missing values with the most frequent
'''

'''
We see that the missing values are only for the numerical columns.
We use the Simple Imputer class from sklearn library to replace the missing
'''

from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values = np.nan, strategy = 'mean')
pre_processing_df.Age = imputer.fit_transform(pre_processing_df['Age'].value
pre_processing_df.DailyRate = imputer.fit_transform(pre_processing_df['Daily
pre_processing_df.MonthlyIncome = imputer.fit_transform(pre_processing_df['M
pre_processing_df.PercentSalaryHike = imputer.fit_transform(pre_processing_d
```

In [34]:
```
pre_processing_df.isnull().sum()
```

Out[34]:
```
Age                         0
BusinessTravel              0
DailyRate                   0
Department                  0
DistanceFromHome            0
Education                   0
EducationField              0
EmployeeCount               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
StandardHours               0
StockOptionLevel            0
TotalWorkingYears           0
TrainingTimesLastYear       0
WorkLifeBalance             0
YearsAtCompany              0
YearsInCurrentRole          0
YearsSinceLastPromotion     0
YearsWithCurrManager        0
Attrition                   0
dtype: int64
```

In [35]:
```
'''
The missing values are now handled.
'''
pre_processing_df.isnull().sum().sum()
```

Out[35]: 0

In [36]:
```
pre_processing_df.head()
```

Out[36]:

	Age	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	E
0	36.981595	Travel_Rarely	1102.000000	Sales	1	2	
1	49.000000	Travel_Frequently	803.036835	Research & Development	8	1	
2	37.000000	Travel_Rarely	1373.000000	Research & Development	2	2	
3	33.000000	Travel_Frequently	1392.000000	Research & Development	3	4	
4	27.000000	Travel_Rarely	591.000000	Research & Development	2	1	

Seperating the features and target columns into X and y respectively. X = {x1, x2, x3, ...} are feature vectors/columns

and y is the target column

In [37]:
```
'''
Separating the Independent and the Dependent variables.
'''
X = pre_processing_df.iloc[:, :-1].values
y = pre_processing_df.iloc[:, -1].values
```

In [38]:
```
print(X)
print(len(X))
```

```
[[36.98159509202454 'Travel_Rarely' 1102.0 ... 4 0 5]
 [49.0 'Travel_Frequently' 803.0368349249659 ... 7 1 7]
 [37.0 'Travel_Rarely' 1373.0 ... 0 0 0]
 ...
 [27.0 'Travel_Rarely' 155.0 ... 2 0 3]
 [49.0 'Travel_Frequently' 1023.0 ... 6 0 8]
 [34.0 'Travel_Rarely' 628.0 ... 3 1 2]]
1470
```

In [39]:
```
print(y, len(y))
```

```
['Yes' 'No' 'Yes' ... 'No' 'No' 'No'] 1470
```

Encoding the categorical data

In [40]:
```
print(X[0], len(X[0]))
```

```
[36.98159509202454 'Travel_Rarely' 1102.0 'Sales' 1 2 'Life Sciences' 1 2
 'Female' 94 3 2 'Sales Executive' 4 'Single' 5993.0 19479 8 'Yes' 11.0 3
 1 80 0 8 0 1 6 4 0 5] 32
```

In [41]:
```
#Encoding the Independent variables

'''
BusinessTravel - 1
Department - 3
EducationField - 6
Gender - 9
JobRole - 13
MaritalStatus - 15
OverTime - 19

Here, we use the OneHotEncoder to do the encoding.
'''

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder

ct = ColumnTransformer(transformers = [('encoder', OneHotEncoder(), [1, 3, 6
X = np.array(ct.fit_transform(X))
```

In [42]:
```
print(X[0], len(X[0]))
```

```
[0.0 0.0 1.0 0.0 0.0 1.0 0.0 1.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0
 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 0.0 1.0 36.98159509202454 1102.0 1 2 1 2
 94 3 2 4 5993.0 19479 8 11.0 3 1 80 0 8 0 1 6 4 0 5] 53
```

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

```
['Yes' 'No' 'Yes' ... 'No' 'No' 'No']
```

In [44]:
```
#Encoding the Dependent variable using Label Encoder
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
```

In [45]:
```
print(y)
```

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

Splitting the dataset into Training and Test sets

In [46]:
```
'''
Splitting the dataset into the Train and Test sets.
Training Dataset = 80%
Test Dataset = 20%
'''
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, r
```

In [47]:
```
print(X_train)
```

```
[[0.0 0.0 1.0 ... 3 1 4]
 [0.0 0.0 1.0 ... 0 0 0]
 [0.0 1.0 0.0 ... 2 7 7]
 ...
 [0.0 0.0 1.0 ... 7 7 7]
 [0.0 0.0 1.0 ... 13 1 9]
 [1.0 0.0 0.0 ... 0 0 0]]
```

In [48]:
```
print(X_test)
```

```
[[0.0 0.0 1.0 ... 3 0 8]
 [0.0 1.0 0.0 ... 0 0 0]
 [0.0 0.0 1.0 ... 2 0 1]
 ...
 [0.0 0.0 1.0 ... 13 15 2]
 [0.0 0.0 1.0 ... 2 0 3]
 [0.0 1.0 0.0 ... 1 0 0]]
```

In [49]:
```
print(y_train)
```

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

In [50]:
```
print(y_test)
```

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

Feature Scaling

In [51]:
```
'''
Scaling the features using Standardization technique so that the Machine lea
the columns with larger range values.
'''

from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [52]:
```
print(X_train)
```

```
[[-0.34021982 -0.479714    0.6403686  ... -0.33697039 -0.35447771
  -0.03090535]
 [-0.34021982 -0.479714    0.6403686  ... -1.16484749 -0.66697081
  -1.17562001]
 [-0.34021982  2.08457539 -1.56160062 ... -0.61292942  1.52048086
   0.82763065]
 ...
 [-0.34021982 -0.479714    0.6403686  ...  0.76686575  1.52048086
   0.82763065]
 [-0.34021982 -0.479714    0.6403686  ...  2.42261996 -0.35447771
   1.39998798]
 [ 2.93927615 -0.479714   -1.56160062 ... -1.16484749 -0.66697081
  -1.17562001]]
```

In [53]:
```
print(X_test)
```

```
[[-0.34021982 -0.479714    0.6403686  ... -0.33697039 -0.66697081
   1.11380932]
 [-0.34021982  2.08457539 -1.56160062 ... -1.16484749 -0.66697081
  -1.17562001]
 [-0.34021982 -0.479714    0.6403686  ... -0.61292942 -0.66697081
  -0.88944135]
 ...
 [-0.34021982 -0.479714    0.6403686  ...  2.42261996  4.02042563
  -0.60326268]
 [-0.34021982 -0.479714    0.6403686  ... -0.61292942 -0.66697081
  -0.31708402]
 [-0.34021982  2.08457539 -1.56160062 ... -0.88888845 -0.66697081
  -1.17562001]]
```

Dimensionality Reduction

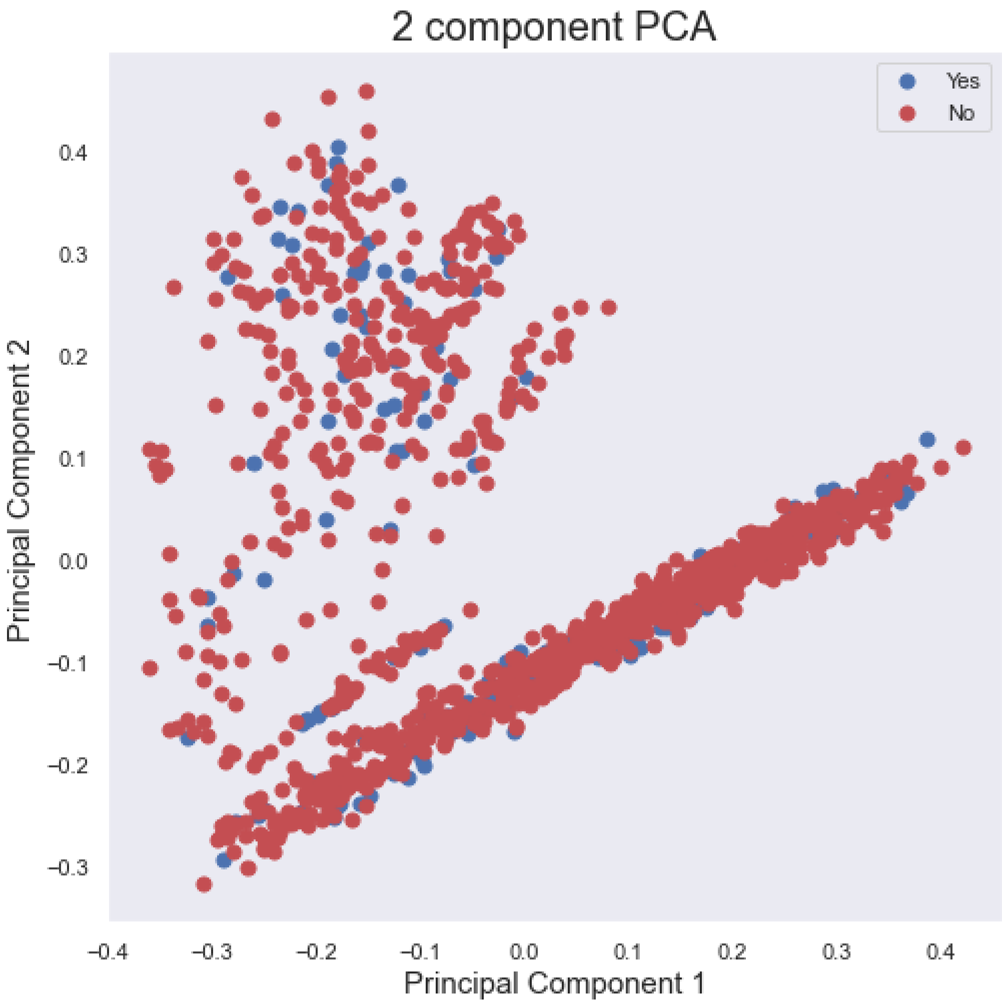
Principal Component Analysis (PCA)

In [54]:
```
'''
An unsupervised linear transformation method known as Principal Component An
(PCA) is frequently utilized in a variety of domains, most notably for featu
dimensionality reduction. Based on the connection between features, PCA aids
of patterns in data. Basically, PCA projects high-dimensional data onto a ne
same dimensions as the original subspace to identify the directions of maxim
We have reduced the dimensions to two components by using PCA as a dimension
approach. The most variance is captured by these two elements
'''

# Applying Kernel PCA
from sklearn.decomposition import KernelPCA
kpca = KernelPCA(n_components = 2, kernel = 'rbf')
X_train_kpca = kpca.fit_transform(X_train)
X_test_kpca = kpca.transform(X_test)

principalDf = pd.DataFrame(data = X_train_kpca, columns = ['principal compon

finalDf = pd.concat([principalDf, eda_df[['Attrition']]], axis = 1)
```

In [55]:
```
print(finalDf)
```

```
       principal component 1  principal component 2 Attrition
0                  -0.034527              -0.138296       Yes
1                   0.145143              -0.032412        No
2                   0.133250              -0.054014       Yes
3                  -0.305698              -0.093475        No
4                  -0.039198              -0.128377        No
...                      ...                    ...       ...
1465                     NaN                    NaN        No
1466                     NaN                    NaN        No
1467                     NaN                    NaN        No
1468                     NaN                    NaN        No
1469                     NaN                    NaN        No

[1470 rows x 3 columns]
```

In [56]:
```
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
targets = ['Yes', 'No']
colors = ['b', 'r']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['Attrition'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
```



In [57]:
```
from sklearn.linear_model import LogisticRegression
logistic_classifier = LogisticRegression()
logistic_classifier.fit(X_train_kpca, y_train)
y_pred = logistic_classifier.predict(X_test_kpca)
```

In [58]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

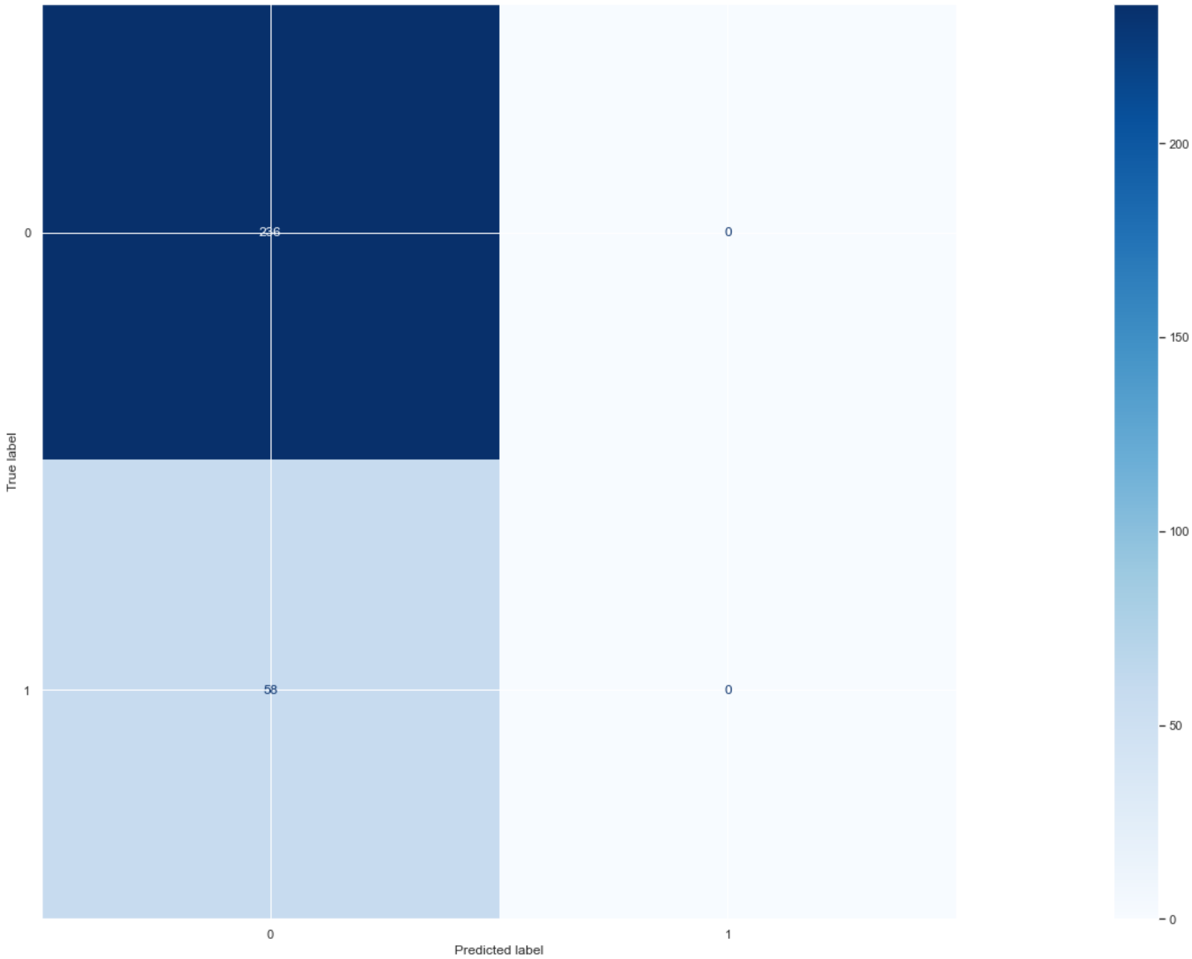
```
[[236   0]
 [ 58   0]]
```

Out[58]:
```
0.8027210884353742
```

In [ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay, classification_report

print(classification_report(y_test, y_pred))
ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap = 'Blues')
```

```
              precision    recall  f1-score   support

           0       0.80      1.00      0.89       236
           1       0.00      0.00      0.00        58

    accuracy                           0.80       294
   macro avg       0.40      0.50      0.45       294
weighted avg       0.64      0.80      0.71       294
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1c2b2a92
c10>
```



In [ ]:
```
from sklearn import metrics
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred)
auc = metrics.roc_auc_score(y_test, y_pred)

#create ROC curve and Calculate the AUC
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

In [ ]:
```
print(X_train_kpca)
```

Classification Models

Logistic Regression

In [ ]:
```
from sklearn.linear_model import LogisticRegression
logistic_classifier = LogisticRegression()
logistic_classifier.fit(X_train, y_train)
y_pred = logistic_classifier.predict(X_test)

#print(np.concatenate((y_pred.reshape(len(y_pred), 1)), (y_pred.reshape(len(
```

In [ ]:
```
print(y_pred)
```

In [ ]:
```
print(y_test)
```

In [ ]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

In [ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay, classification_report

print(classification_report(y_test, y_pred))
ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap = 'Blues')
```

In [ ]:
```
'''
Plotting the ROC and calulating the AUC.
The closer the Area Under Curve (AUC) to 1, the better the model performance
'''
from sklearn import metrics
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred)
auc = metrics.roc_auc_score(y_test, y_pred)

#create ROC curve and Calculate the AUC
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

In [ ]:
```
#Performing K-Fold cross-validation to evaluate the model better. Here, k =
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = logistic_classifier, X = X_train, y
print("Accuracies for the 10-folds is {}".format(accuracies))
print("Accuracy is {}".format(accuracies.mean()*100))
print("Standard Deviation is {}".format(accuracies.std()*100))
```

K-Nearest Neighbor

In [ ]:
```
'''
Number of neighbors considered is 5
Distance metric used is Euclidean
'''
from sklearn.neighbors import KNeighborsClassifier
knn_classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski',
knn_classifier.fit(X_train, y_train)
y_pred = knn_classifier.predict(X_test)
```

In [ ]:
```
print(y_pred)
```

In [ ]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

In [ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay, classification_report

print(classification_report(y_test, y_pred))
ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap = 'Blues')
```

In [ ]:
```
'''
Plotting the ROC and calulating the AUC.
The closer the Area Under Curve (AUC) to 1, the better the model performance
'''
```

```
from sklearn import metrics
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred)
auc = metrics.roc_auc_score(y_test, y_pred)

#create ROC curve and Calculate the AUC
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

In [ ]:
```
#Performing K-Fold cross-validation to evaluate the model better. Here, k =
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = knn_classifier, X = X_train, y = y_
print("Accuracies for the 10-folds is {}".format(accuracies))
print("Accuracy is {}".format(accuracies.mean()*100))
print("Standard Deviation is {}".format(accuracies.std()*100))
```

Support Vector Machine

In [ ]:
```
'''
Using the Linear Kernel for SVM since we figured out from the Grid Search te
gives better performance. Grid Search is implemented later in this section.
'''

from sklearn.svm import SVC
svm_classifier = SVC(kernel = 'linear', random_state = 0)
svm_classifier.fit(X_train, y_train)
y_pred = svm_classifier.predict(X_test)
```

In [ ]:
```
print(y_pred)
```

In [ ]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

In [ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay, classification_report

print(classification_report(y_test, y_pred))
ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap = 'Blues')
```

In [ ]:
```
'''
Plotting the ROC and calulating the AUC.
The closer the Area Under Curve (AUC) to 1, the better the model performance
'''

from sklearn import metrics
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred)
auc = metrics.roc_auc_score(y_test, y_pred)

#create ROC curve and Calculate the AUC
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

In [ ]:
```
#Performing K-Fold cross-validation to evaluate the model better. Here, k =
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = svm_classifier, X = X_train, y = y_
print("Accuracies for the 10-folds is {}".format(accuracies))
print("Accuracy is {}".format(accuracies.mean()*100))
print("Standard Deviation is {}".format(accuracies.std()*100))
```

Kernel SVM

In [ ]:
```
'''
Implementing the RBF kernel for SVM
'''

from sklearn.svm import SVC
kernel_svm_classifier = SVC(kernel = 'rbf', random_state = 0)
kernel_svm_classifier.fit(X_train, y_train)
y_pred = kernel_svm_classifier.predict(X_test)
```

In [ ]:
```
print(y_pred)
```

In [ ]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

In [ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay, classification_report

print(classification_report(y_test, y_pred))
ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap = 'Blues')
```

In [ ]:
```
'''
Plotting the ROC and calulating the AUC.
The closer the Area Under Curve (AUC) to 1, the better the model performance
'''

from sklearn import metrics
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred)
auc = metrics.roc_auc_score(y_test, y_pred)

#create ROC curve and Calculate the AUC
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

In [ ]:
```
#Performing K-Fold cross-validation to evaluate the model better. Here, k =
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = kernel_svm_classifier, X = X_train,
print("Accuracies for the 10-folds is {}".format(accuracies))
print("Accuracy is {}".format(accuracies.mean()*100))
print("Standard Deviation is {}".format(accuracies.std()*100))
```

In [ ]:
```
#Grid Search to find the best model and the best parameters
from sklearn.model_selection import GridSearchCV
parameters = [{'C': [0.25, 0.5, 0.75, 1], 'kernel': ['linear']}, {'C': [0.25
gridsearch =  GridSearchCV(estimator = svm_classifier, param_grid = paramete

gridsearch.fit(X_train, y_train)
best_accuracy = gridsearch.best_score_
```

```python
best_parameters = gridsearch.best_params_

print("Best Accuracy is {}".format(best_accuracy*100))
print("Best parameters are ", best_parameters)
```

Naive Bayes

In [ ]:
```python
'''
Probabilistic model for classification.
'''

from sklearn.naive_bayes import GaussianNB
naivebayes_classifier = GaussianNB()
naivebayes_classifier.fit(X_train, y_train)
y_pred = naivebayes_classifier.predict(X_test)
```

In [ ]:
```python
print(y_pred)
```

In [ ]:
```python
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

In [ ]:
```python
from sklearn.metrics import ConfusionMatrixDisplay, classification_report

print(classification_report(y_test, y_pred))
ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap = 'Blues')
```

In [ ]:
```python
'''
Plotting the ROC and calulating the AUC.
The closer the Area Under Curve (AUC) to 1, the better the model performance
'''

from sklearn import metrics
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred)
auc = metrics.roc_auc_score(y_test, y_pred)

#create ROC curve and Calculate the AUC
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

In [ ]:
```python
#Performing K-Fold cross-validation to evaluate the model better. Here, k =
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = naivebayes_classifier, X = X_train,
print("Accuracies for the 10-folds is {}".format(accuracies))
print("Accuracy is {}".format(accuracies.mean()*100))
print("Standard Deviation is {}".format(accuracies.std()*100))
```

Decision Tree

In [ ]:
```python
from sklearn.tree import DecisionTreeClassifier
decisiontree_classifier = DecisionTreeClassifier(criterion = 'entropy', rand
decisiontree_classifier.fit(X_train, y_train)
y_pred = decisiontree_classifier.predict(X_test)
```

In [ ]:
```
print(y_pred)
```

In [ ]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

In [ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay, classification_report

print(classification_report(y_test, y_pred))
ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap = 'Blues')
```

In [ ]:
```
'''
Plotting the ROC and calulating the AUC.
The closer the Area Under Curve (AUC) to 1, the better the model performance
'''

from sklearn import metrics
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred)
auc = metrics.roc_auc_score(y_test, y_pred)

#create ROC curve and Calculate the AUC
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

In [ ]:
```
#Performing K-Fold cross-validation to evaluate the model better. Here, k =
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = decisiontree_classifier, X = X_trai
print("Accuracies for the 10-folds is {}".format(accuracies))
print("Accuracy is {}".format(accuracies.mean()*100))
print("Standard Deviation is {}".format(accuracies.std()*100))
```

Random Forest

In [ ]:
```
#The maximum height upto which the trees can grow is 100. The criterion chos
from sklearn.ensemble import RandomForestClassifier
randomforest_classifier = RandomForestClassifier(n_estimators = 100, criteri
randomforest_classifier.fit(X_train, y_train)
y_pred = randomforest_classifier.predict(X_test)
```

In [ ]:
```
print(y_pred)
```

In [ ]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

In [ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay, classification_report

print(classification_report(y_test, y_pred))
ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap = 'Blues')
```

In [ ]:
```
'''
Plotting the ROC and calulating the AUC.
The closer the Area Under Curve (AUC) to 1, the better the model performance
'''
```

```
from sklearn import metrics
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred)
auc = metrics.roc_auc_score(y_test, y_pred)

#create ROC curve and Calculate the AUC
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

In [ ]:
```
#Performing K-Fold cross-validation to evaluate the model better. Here, k =
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = randomforest_classifier, X = X_trai
print("Accuracies for the 10-folds is {}".format(accuracies))
print("Accuracy is {}".format(accuracies.mean()*100))
print("Standard Deviation is {}".format(accuracies.std()*100))
```

Bagging Classifier

In [ ]:
```
'''
The base estimator chosen is DecisionTreeClassifier and the number of estima
'''
from sklearn import model_selection
from sklearn.ensemble import BaggingClassifier

kfold_cv = model_selection.KFold(n_splits = 5, shuffle = True)
bagging_classifier = BaggingClassifier(base_estimator = DecisionTreeClassifi
result = model_selection.cross_val_score(bagging_classifier, X_train, y_trai
print(result.mean())
```

In [ ]:
```
bagging_classifier.fit(X_train,y_train)
y_pred = bagging_classifier.predict(X_test)
```

In [ ]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

In [ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay, classification_report

print(classification_report(y_test, y_pred))
ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap = 'Blues')
```

In [ ]:
```
'''
Plotting the ROC and calulating the AUC.
The closer the Area Under Curve (AUC) to 1, the better the model performance
'''

from sklearn import metrics
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred)
auc = metrics.roc_auc_score(y_test, y_pred)

#create ROC curve and Calculate the AUC
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

In [ ]:
```
#Performing K-Fold cross-validation to evaluate the model better. Here, k =
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = bagging_classifier, X = X_train, y
print("Accuracies for the 10-folds is {}".format(accuracies))
print("Accuracy is {}".format(accuracies.mean()*100))
print("Standard Deviation is {}".format(accuracies.std()*100))
```

Adaboost Classifier

In [ ]:
```
'''
The base estimator chosen is DecisionTreeClassifier and the number of estima
The learning rate chosen is 2 in this case.
'''
from sklearn.ensemble import AdaBoostClassifier

adaboost_classifier =  AdaBoostClassifier(base_estimator = DecisionTreeClass
adaboost_classifier.fit(X_train, y_train)

result = model_selection.cross_val_score(adaboost_classifier, X_train, y_tra
print(result.mean())
```

In [ ]:
```
y_pred = adaboost_classifier.predict(X_test)
print(accuracy_score(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

In [ ]:
```
print(classification_report(y_test, y_pred))
```

In [ ]:
```
ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap = 'Blues')
```

In [ ]:
```
'''
Plotting the ROC and calulating the AUC.
The closer the Area Under Curve (AUC) to 1, the better the model performance
'''

from sklearn import metrics
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred)
auc = metrics.roc_auc_score(y_test, y_pred)

#create ROC curve and Calculate the AUC
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

In [ ]:
```
#Performing K-Fold cross-validation to evaluate the model better. Here, k =
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = adaboost_classifier, X = X_train, y
print("Accuracies for the 10-folds is {}".format(accuracies))
print("Accuracy is {}".format(accuracies.mean()*100))
print("Standard Deviation is {}".format(accuracies.std()*100))
```

XGBoost Classifier

In [ ]:
```
from xgboost import XGBClassifier
xgb_classifier = XGBClassifier()
```

```
xgb_classifier.fit(X_train, y_train)
y_pred = xgb_classifier.predict(X_test)
```

In [ ]:
```
print(y_pred)
```

In [ ]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

In [ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay, classification_report

print(classification_report(y_test, y_pred))
ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap = 'Blues')
```

In [ ]:
```
'''
Plotting the ROC and calulating the AUC.
The closer the Area Under Curve (AUC) to 1, the better the model performance
'''

from sklearn import metrics
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred)
auc = metrics.roc_auc_score(y_test, y_pred)

#create ROC curve and Calculate the AUC
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

In [ ]:
```
#Performing K-Fold cross-validation to evaluate the model better. Here, k =
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = randomforest_classifier, X = X_trai
print("Accuracies for the 10-folds is {}".format(accuracies))
print("Accuracy is {}".format(accuracies.mean()*100))
print("Standard Deviation is {}".format(accuracies.std()*100))
```

In [ ]:
```
from xgboost import plot_importance

ax=plot_importance(xgb_classifier)
plt.title('Feature importances')
plt.show()
```

Artificial Neural Networks

In [ ]:
```
tf.__version__
```

In [ ]:
```
#Initialize the ANN
ann_classifier = tf.keras.models.Sequential()

#Add 3 Hidden layers with the desired number of neurons and the activation f
ann_classifier.add(tf.keras.layers.Dense(units = 16, activation = 'relu'))
ann_classifier.add(tf.keras.layers.Dense(units = 12, activation = 'relu'))
ann_classifier.add(tf.keras.layers.Dense(units = 7, activation = 'relu'))

#Add the Output layer with the 1 neuron and the activation function
ann_classifier.add(tf.keras.layers.Dense(units = 1, activation = 'sigmoid'))
```

In [ ]:
```
#Compile the ANN
ann_classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', met

#Train the ANN on the training set
ann_classifier.fit(X_train, y_train, batch_size = 32, epochs = 160)
```

In [ ]:
```
y_pred = ann_classifier.predict(X_test)
y_pred = (y_pred > 0.5)
```

In [ ]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

In [ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay, classification_report

print(classification_report(y_test, y_pred))
ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap = 'Blues')
```

In [ ]:
```
'''
Plotting the ROC and calulating the AUC.
The closer the Area Under Curve (AUC) to 1, the better the model performance
'''

from sklearn import metrics
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred)
auc = metrics.roc_auc_score(y_test, y_pred)

#create ROC curve and Calculate the AUC
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

In [ ]:
```
'''
Conclusion:

When compared to "Yes," the dataset has more values for "No" than for "Yes"
though the data is not entirely accurate, we were nevertheless able to achie
above 85%. Dimensionality reduction has also shown that the data are only pa
separable. As we can see from the aforementioned classification models, Logi
SVM and Kernel SVM, Random Forest, and XGBoost Classifier offer greater comp
accuracies. Since the mean accuracy of 10-fold cross-validation is near to t
accuracy, we employed the K-Fold cross-validation procedures to make sure we
fortunate with the accuracy computation.
We are hopeful that the employee churn projection supplied by our developed
help HR in some way so that they can take the necessary actions to retain th
organization can use our methods to increase staff retention. It can be used
of fresh resources. When the time comes to lay off workers as part of organi
corporation can use churn modeling to make a rational decision rather than s
candidates at random
'''
```

In [ ]: