Assignment 1

Please enter your **name, surname** and **student number** instead of `"NAME-HERE"`, `"SURNAME-HERE"`, `"NUMBER-HERE"` below.

In [112…
```
student = {
    'name' : "Defne" ,
    'surname' : "Odabaşı",
    'studentNumber' : "2443604"
}

print(student)
```

```
{'name': 'Defne', 'surname': 'Odabaşı', 'studentNumber': '2443604'}
```

Part I: Classification Problem

1. The "IBM HR Analytics Employee Attrition Dataset" should be downloaded from the Kaggle website: https://www.kaggle.com/datasets/pavansubhasht/ibm-hr-analytics-attrition-dataset
2. Use a seed value of 12345 for random number generators to ensure reproducibility in your experiments. [Mandatory, 0 points]
3. Conduct Exploratory Data Analysis (EDA) to gain insights into the dataset characteristics. Employ statistical summaries and visualizations to uncover patterns and anomalies. [10 points]
4. Execute data preprocessing to enhance model performance if deemed necessary. This may include handling missing values, encoding categorical variables, feature scaling, and any other technique that could improve the results. [5 points]
5. Implement 5-Fold Cross Validation to assess the robustness of your models. This approach ensures that the evaluation of your model is as accurate as possible. [5 points]
6. Develop and evaluate models using K-Nearest Neighbors (KNN), Naive Bayes, Perceptron, and Logistic Regression algorithms. Document the performance of each model. [30 points]
7. Investigate the outcomes using appropriate metrics such as accuracy, precision, recall, F1 score, and ROC-AUC curve where applicable. [5 points]
8. Discuss the results. Reflect on which model yielded the best performance and hypothesize why this might be the case. Consider the algorithm's suitability for the data distribution, complexity, and balance of the dataset. [15 points]

Your Discussion Here (You can double click and edit this block)

In [113…
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import Perceptron
```

In [114…
```
seed_value = 12345 #seed value
np.random.seed(seed_value)
```

Exploratory Data Analysis (EDA)

Gaining insides into dataset characteristics. Employing statistical summaries and visualizations to uncover patterns and anormalies:

In [115…
```
# Your code here (you can add more blocks as you need).

employee_attrition = pd.read_csv("C:\\Users\\defne\\Desktop\\2023-2024FallSemester\
employee_attrition.head()

# Please add comments where you think necessary.
```

Out[115]:

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

5 rows × 35 columns

In [116…
```
employee_attrition.shape
```

Out[116]:
```
(1470, 35)
```

In [117…
```
employee_attrition.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 [118…

```
employee_attrition.describe() #statistics from the data
```

Out[118]:

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNum
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.0	1470.000
mean	36.923810	802.485714	9.192517	2.912925	1.0	1024.865
std	9.135373	403.509100	8.106864	1.024165	0.0	602.024
min	18.000000	102.000000	1.000000	1.000000	1.0	1.000
25%	30.000000	465.000000	2.000000	2.000000	1.0	491.250
50%	36.000000	802.000000	7.000000	3.000000	1.0	1020.500
75%	43.000000	1157.000000	14.000000	4.000000	1.0	1555.750
max	60.000000	1499.000000	29.000000	5.000000	1.0	2068.000

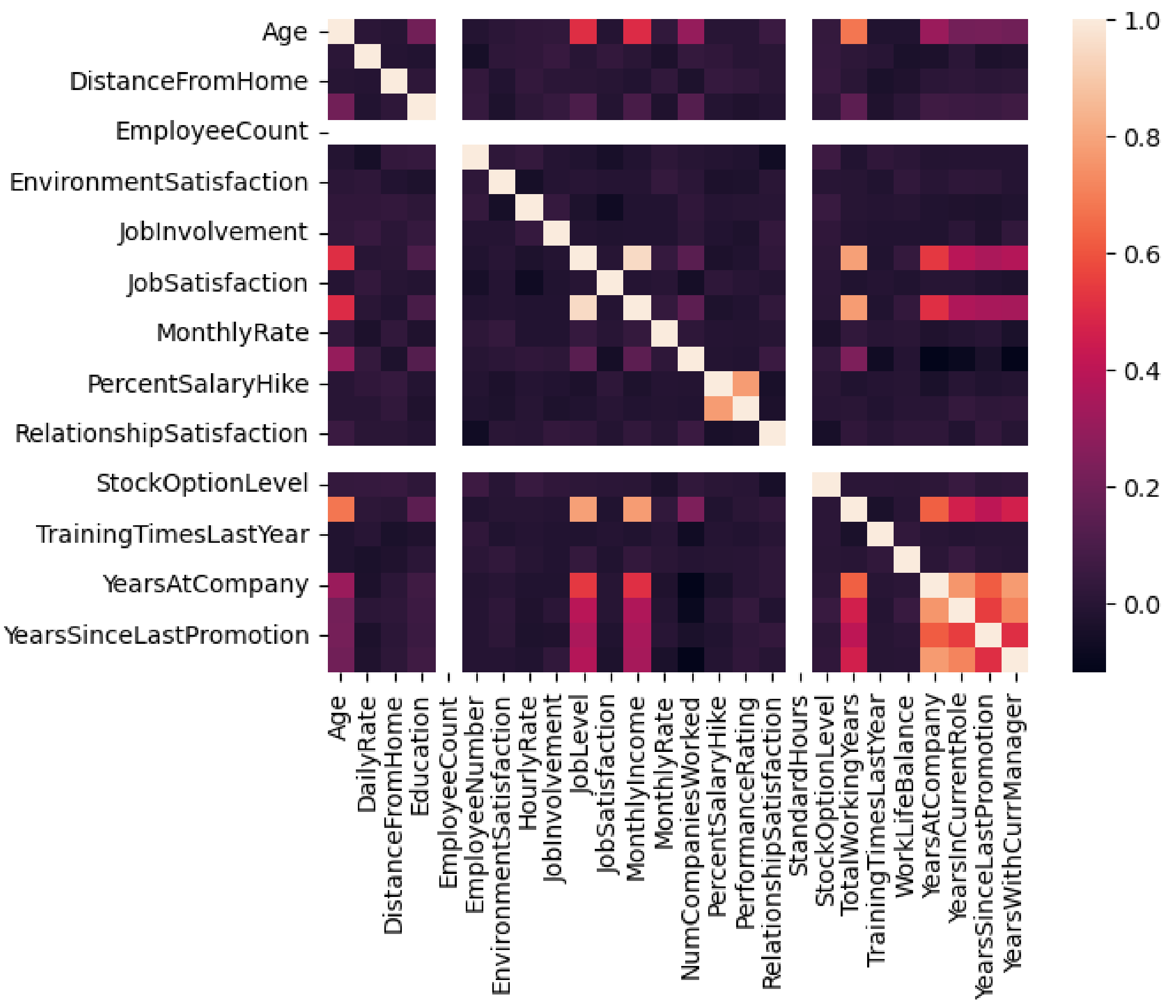
8 rows × 26 columns

In [119…

```
#Correlation plot
#after encoding it is provided again
sns.heatmap(employee_attrition.corr())
```

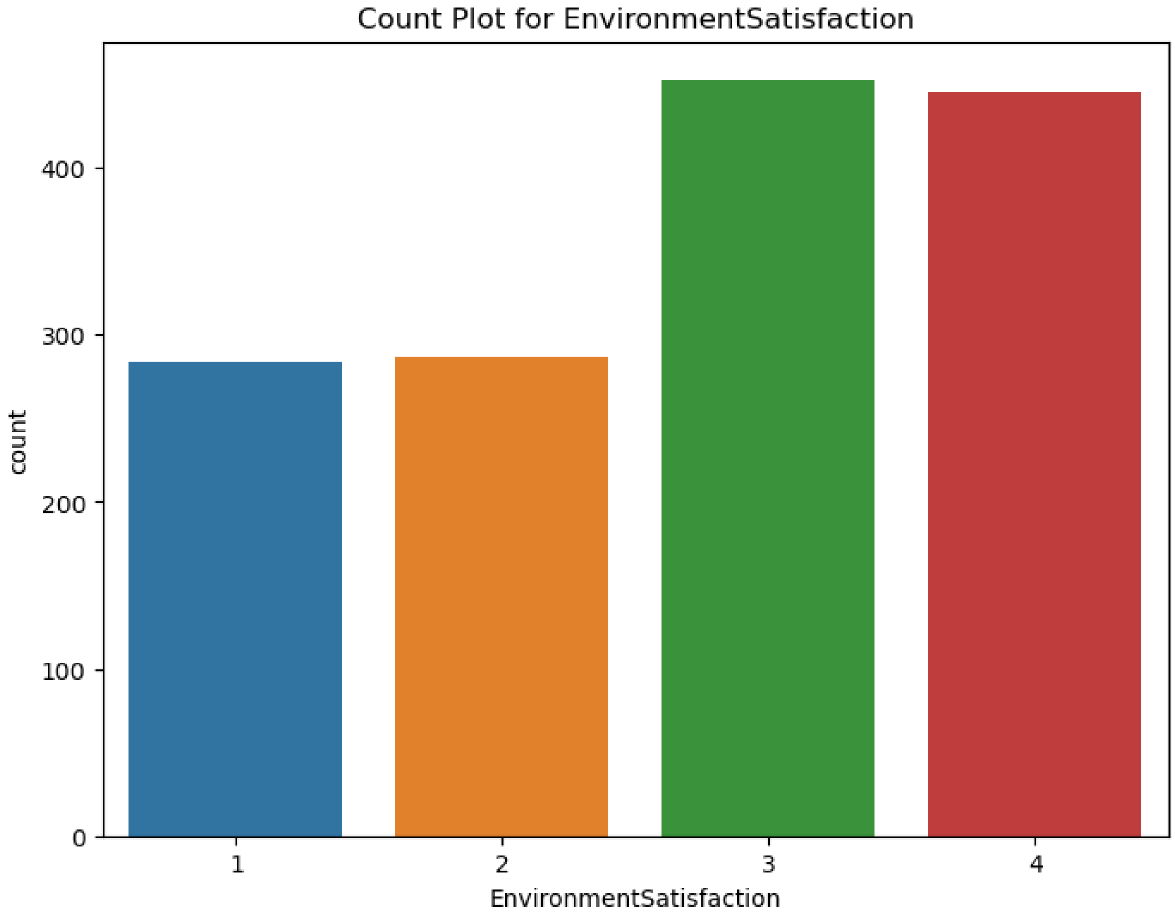
```
C:\Users\defne\AppData\Local\Temp\ipykernel_10008\4079675858.py:3: FutureWarning:
The default value of numeric_only in DataFrame.corr is deprecated. In a future ver
sion, it will default to False. Select only valid columns or specify the value of
numeric_only to silence this warning.
  sns.heatmap(employee_attrition.corr())
```

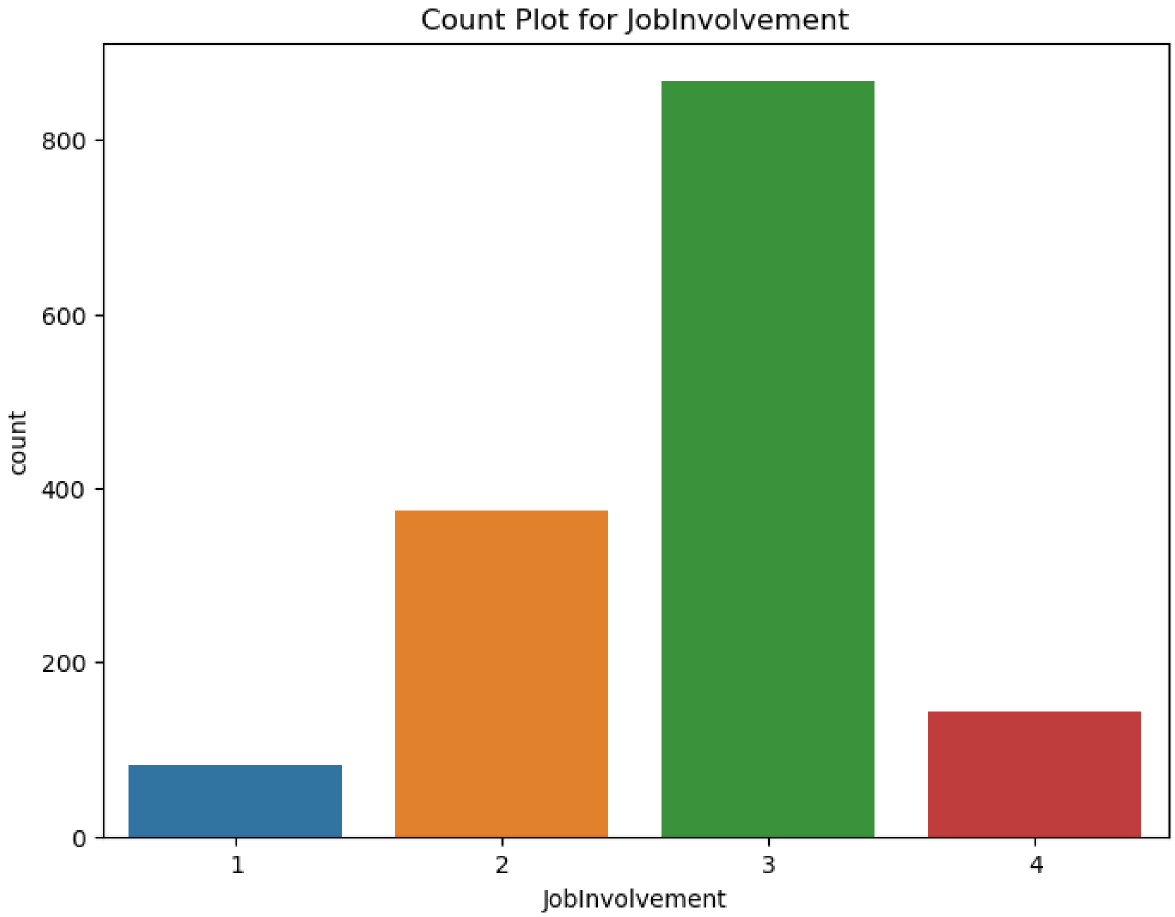
Out[119]: <Axes: >

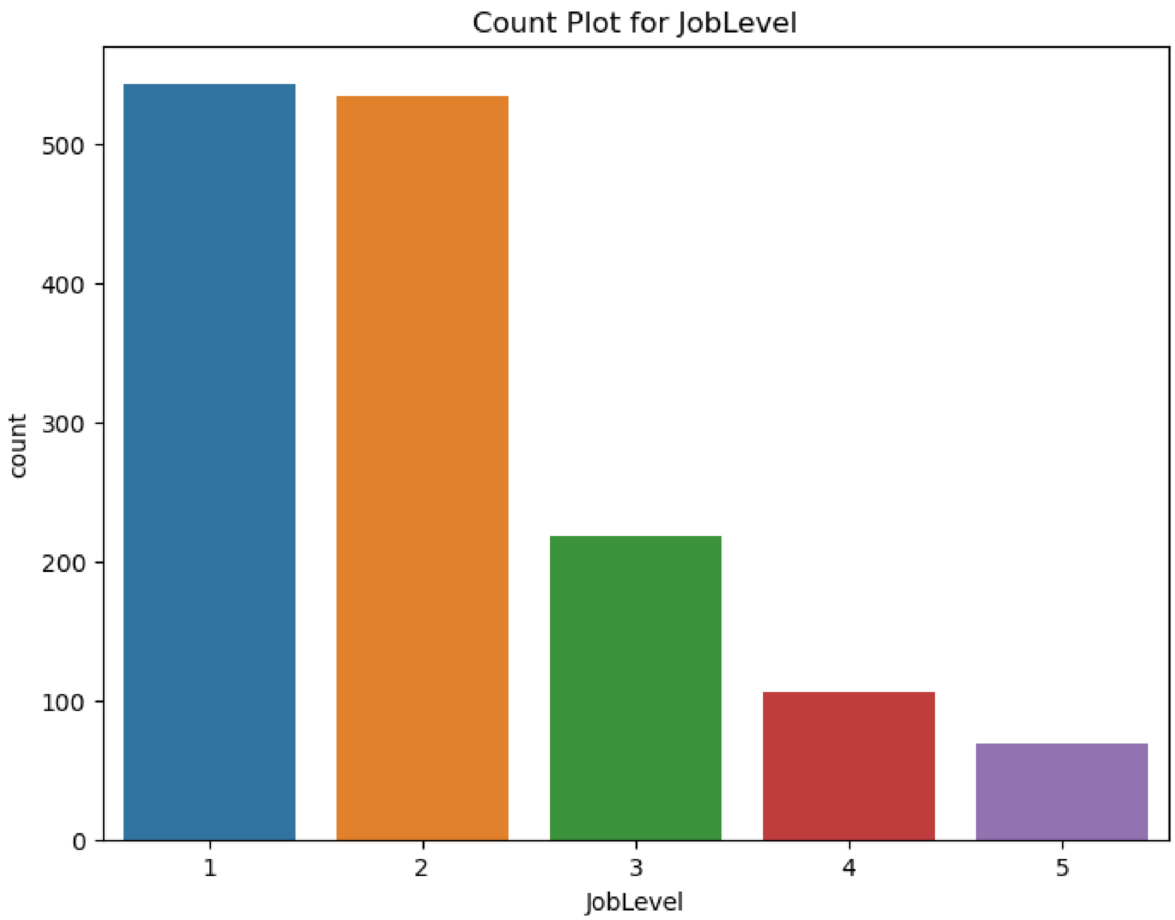


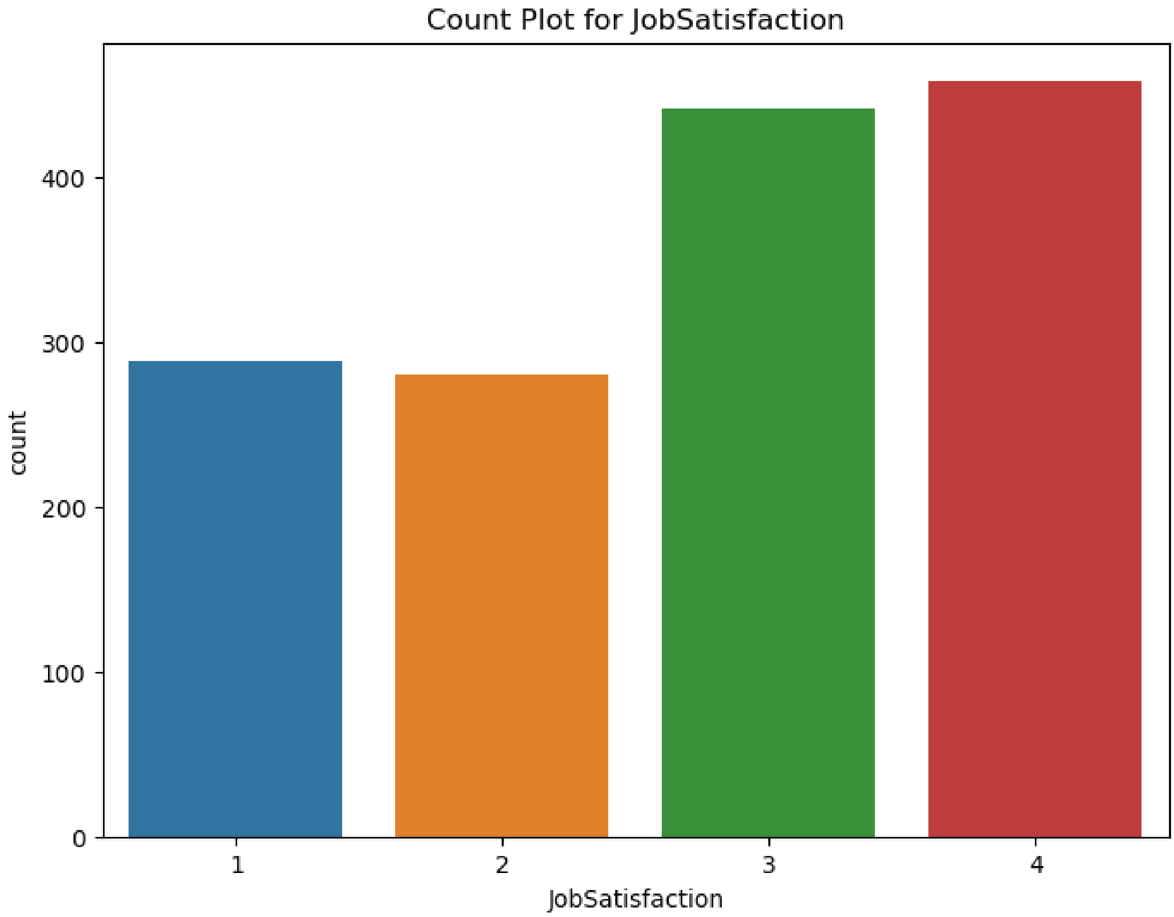
In [120…

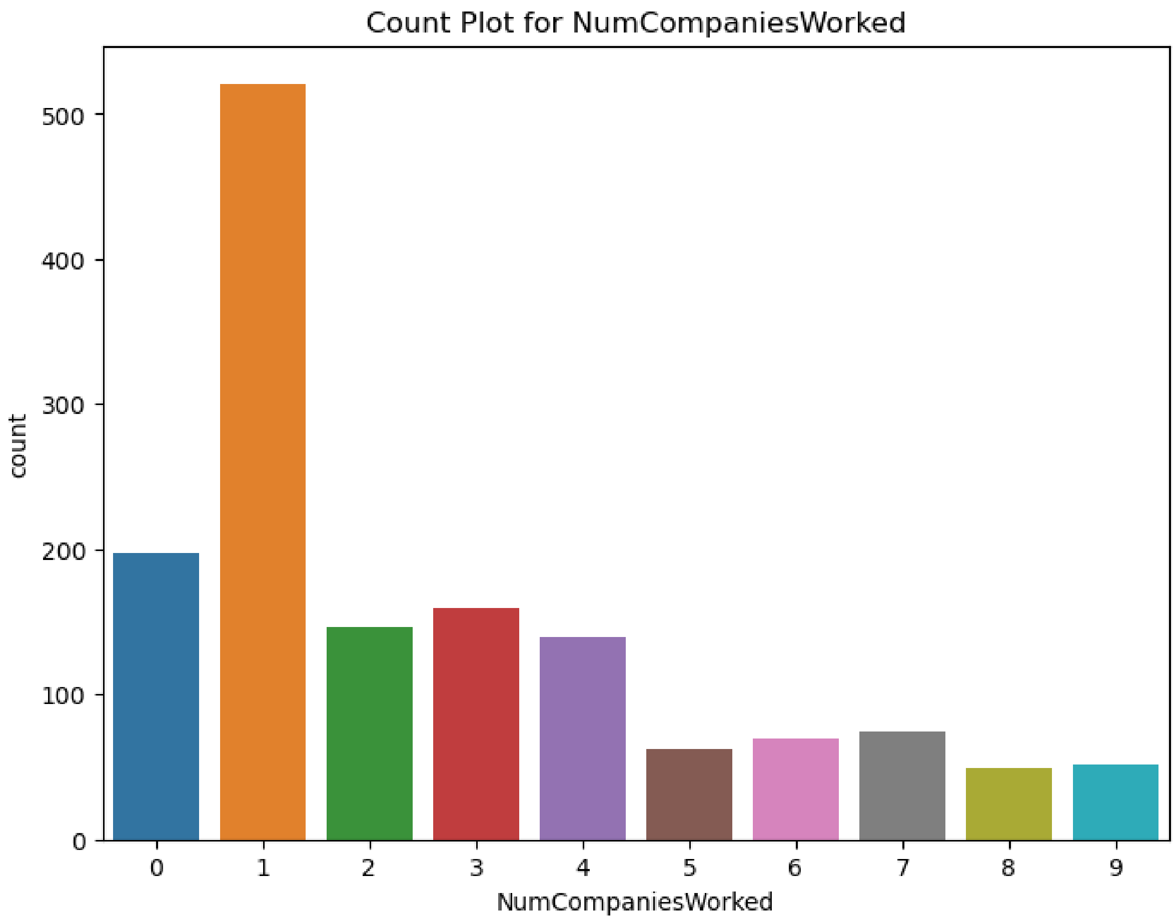
```
#for the count plot:
selected_columns = ['EnvironmentSatisfaction', 'JobInvolvement', 'JobLevel', 'JobSa
                    'NumCompaniesWorked', 'PercentSalaryHike', 'PerformanceRating'
                    'StandardHours', 'StockOptionLevel', 'TotalWorkingYears', 'Tra
                    'WorkLifeBalance', 'YearsAtCompany', 'YearsInCurrentRole', 'Ye
                    'YearsWithCurrManager'] #selected_columns for the visualisatic
for column in selected_columns:
    plt.figure(figsize=(8, 6))
    sns.countplot(x=column, data=employee_attrition)
    plt.title(f'Count Plot for {column}')
    plt.show()
```

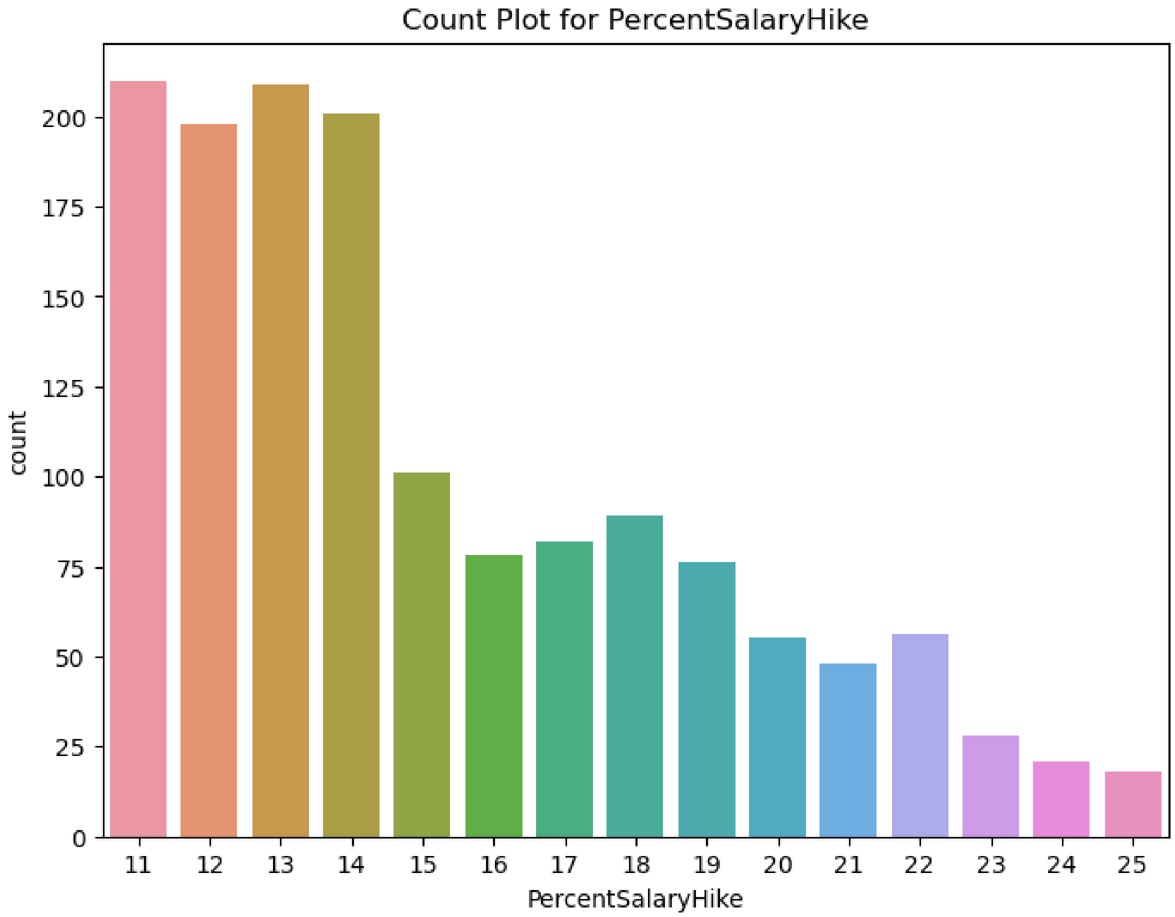


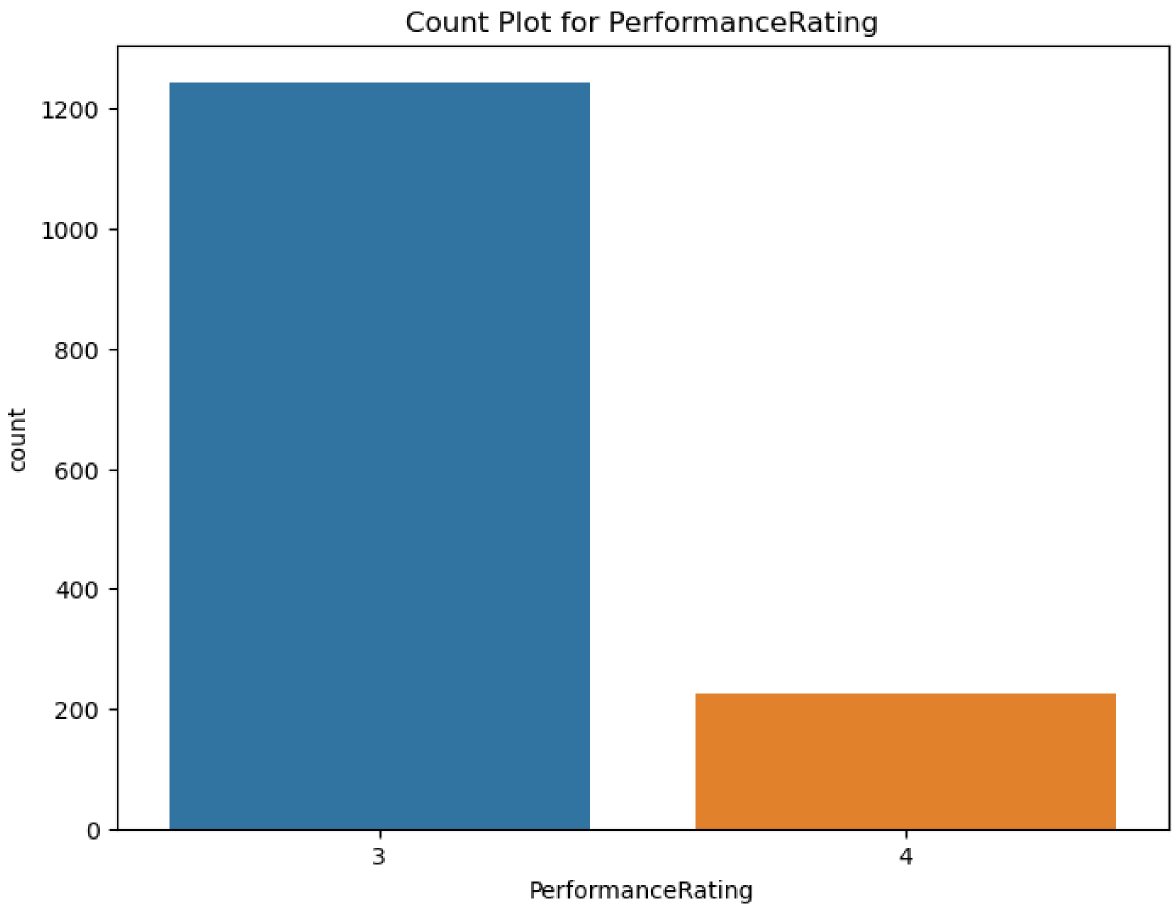

Count Plot for JobInvolvement
count
800
600
400
200
0
1
2
3
4
JobInvolvement

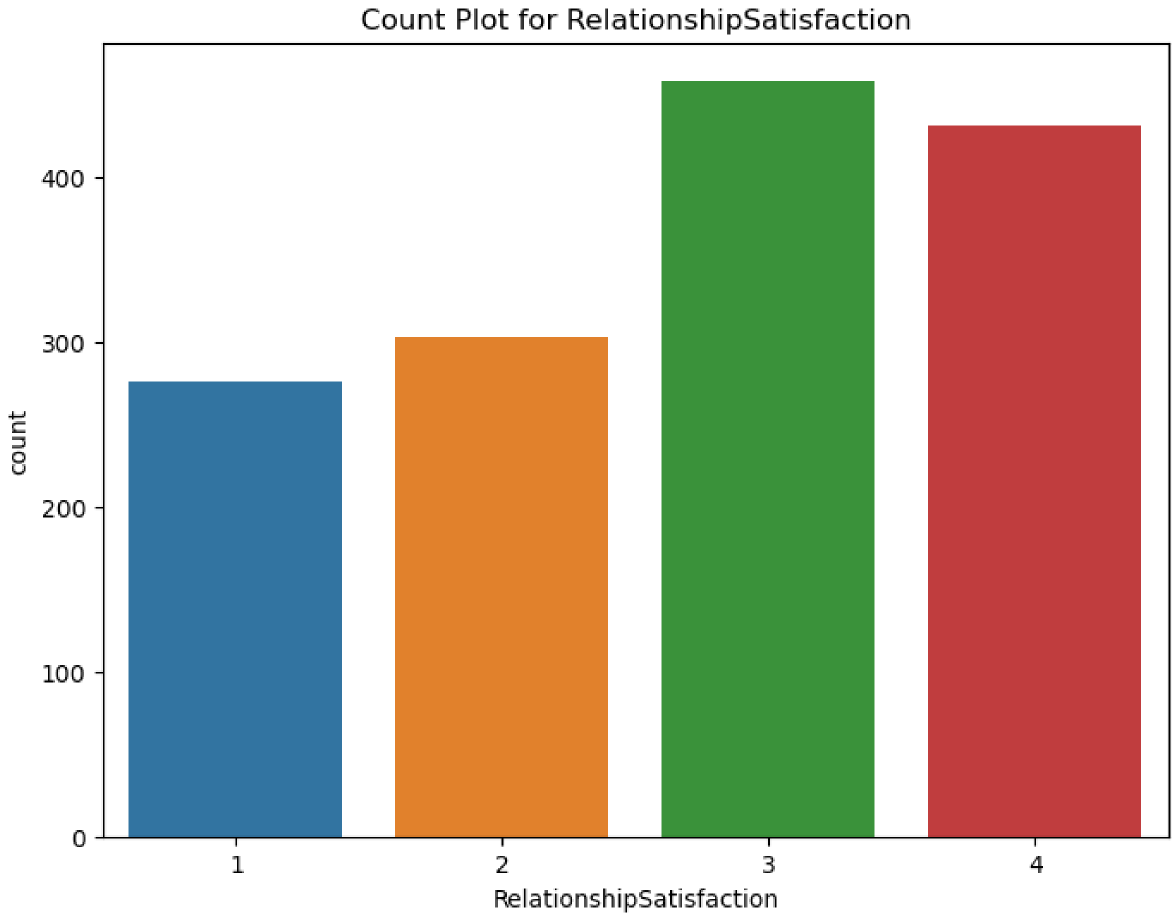

Count Plot for JobLevel
count
500
400
300
200
100
0
1
2
3
4
5
JobLevel

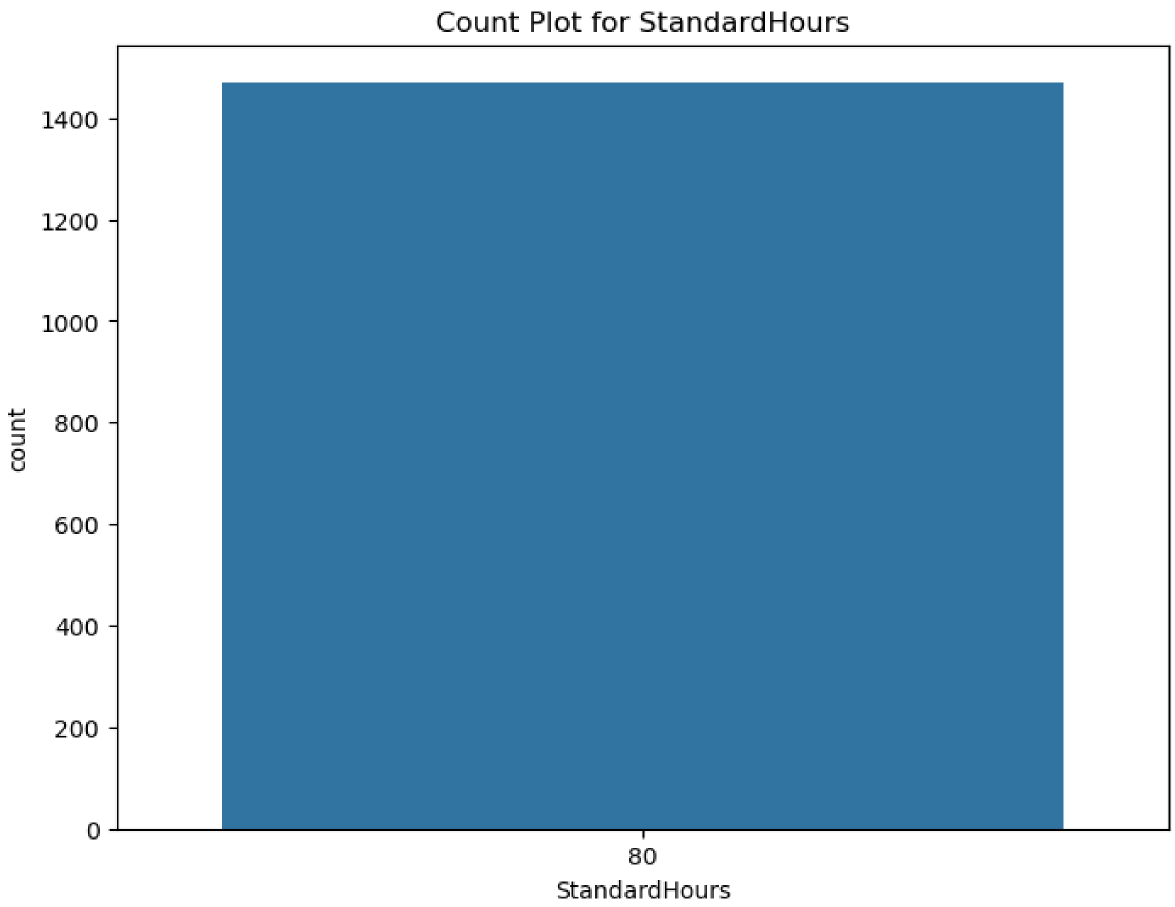

Count Plot for JobSatisfaction
count
0
100
200
300
400
1
2
3
4
JobSatisfaction

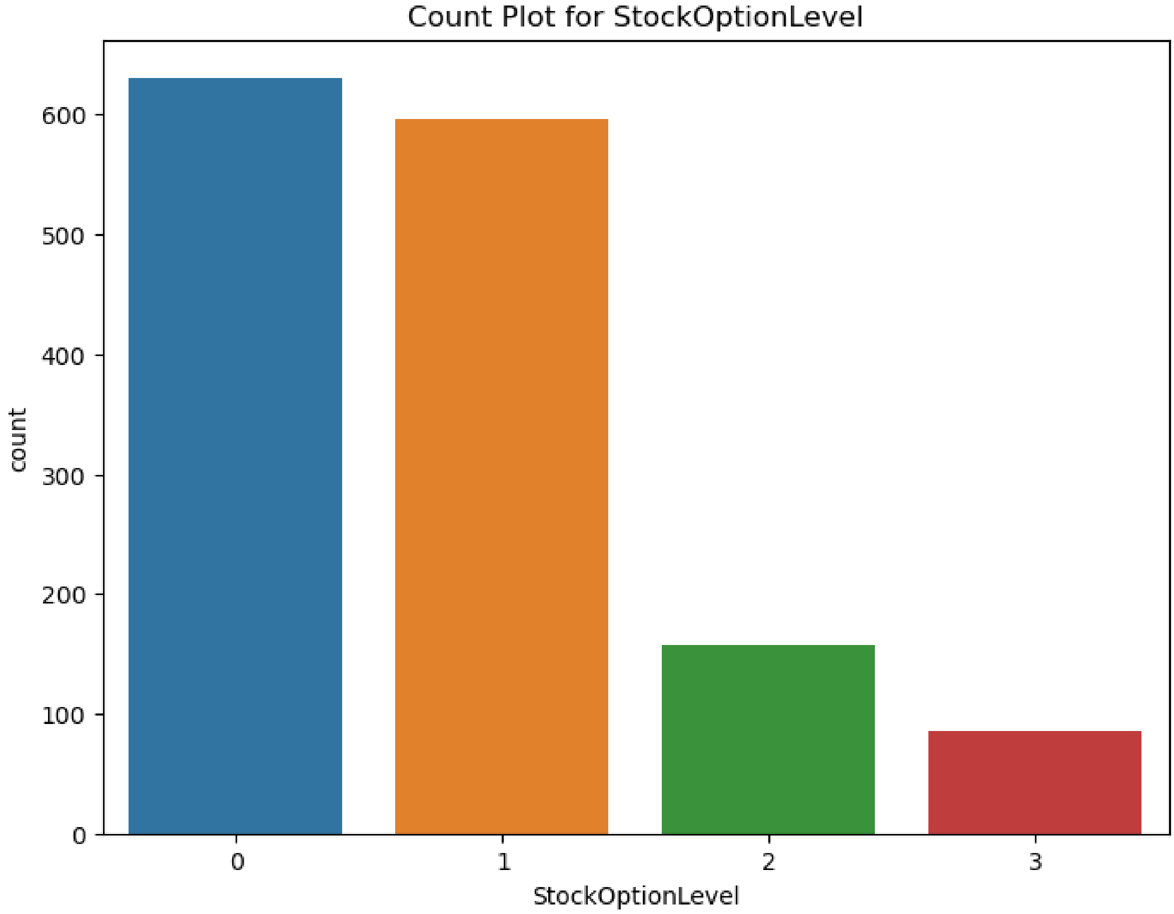

Count Plot for NumCompaniesWorked
count
0
100
200
300
400
500
0
1
2
3
4
5
6
7
8
9
NumCompaniesWorked

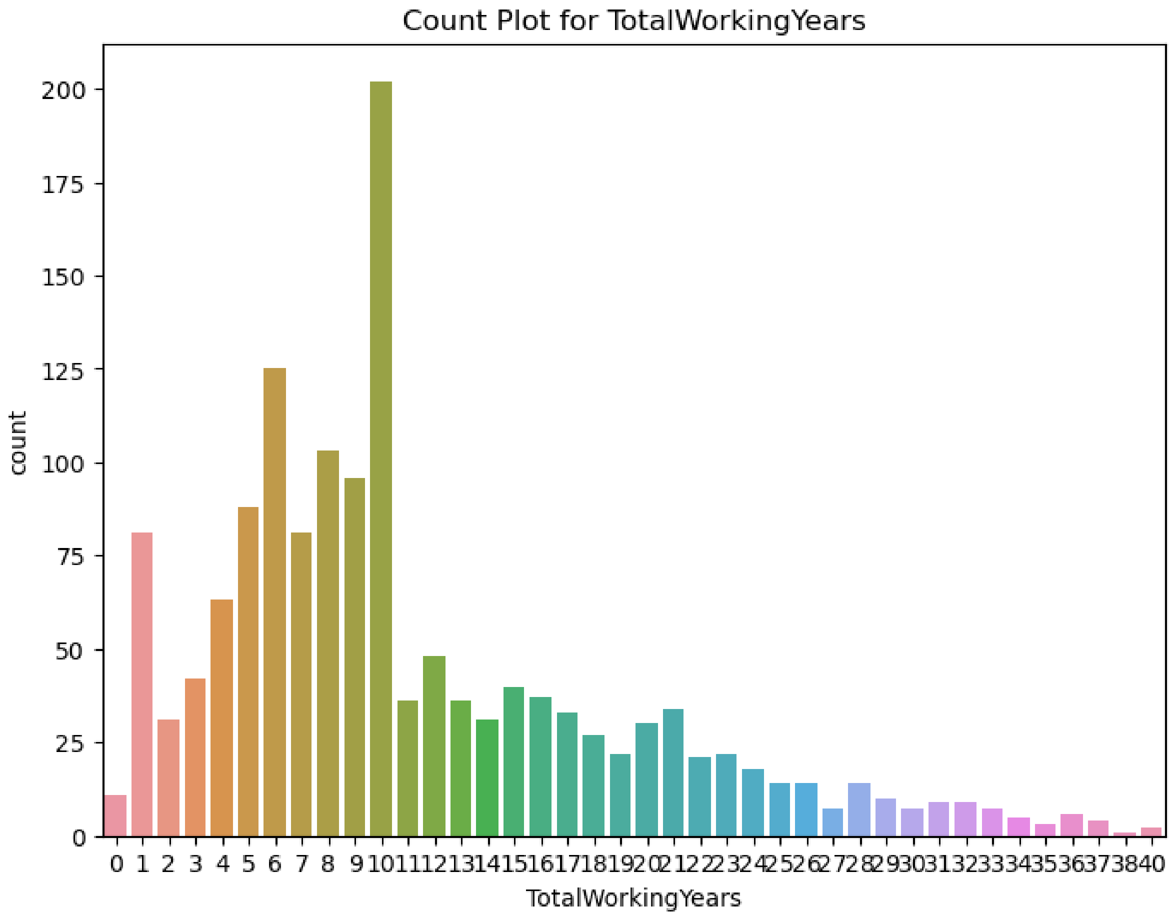

Count Plot for PercentSalaryHike
count
0
25
50
75
100
125
150
175
200
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
PercentSalaryHike

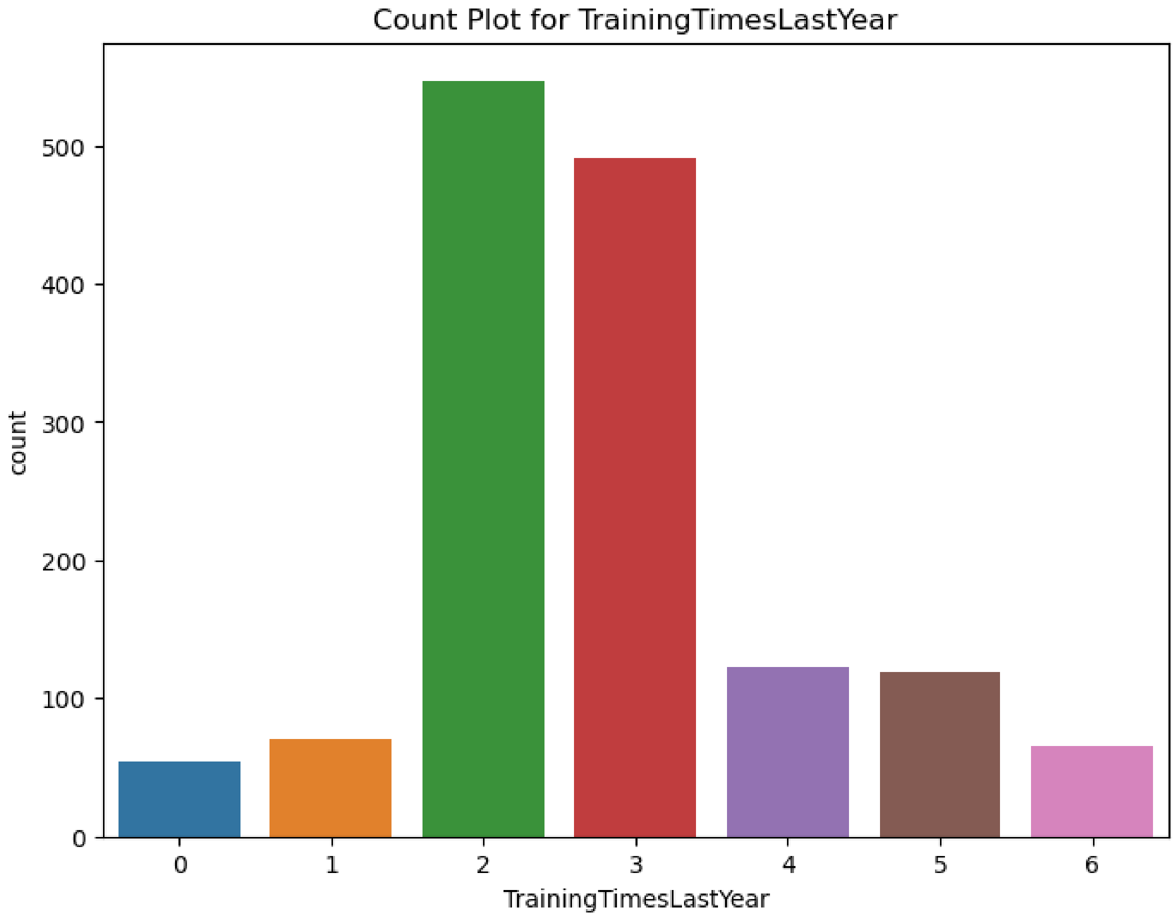

Count Plot for PerformanceRating
count
0
200
400
600
800
1000
1200
3
4
PerformanceRating

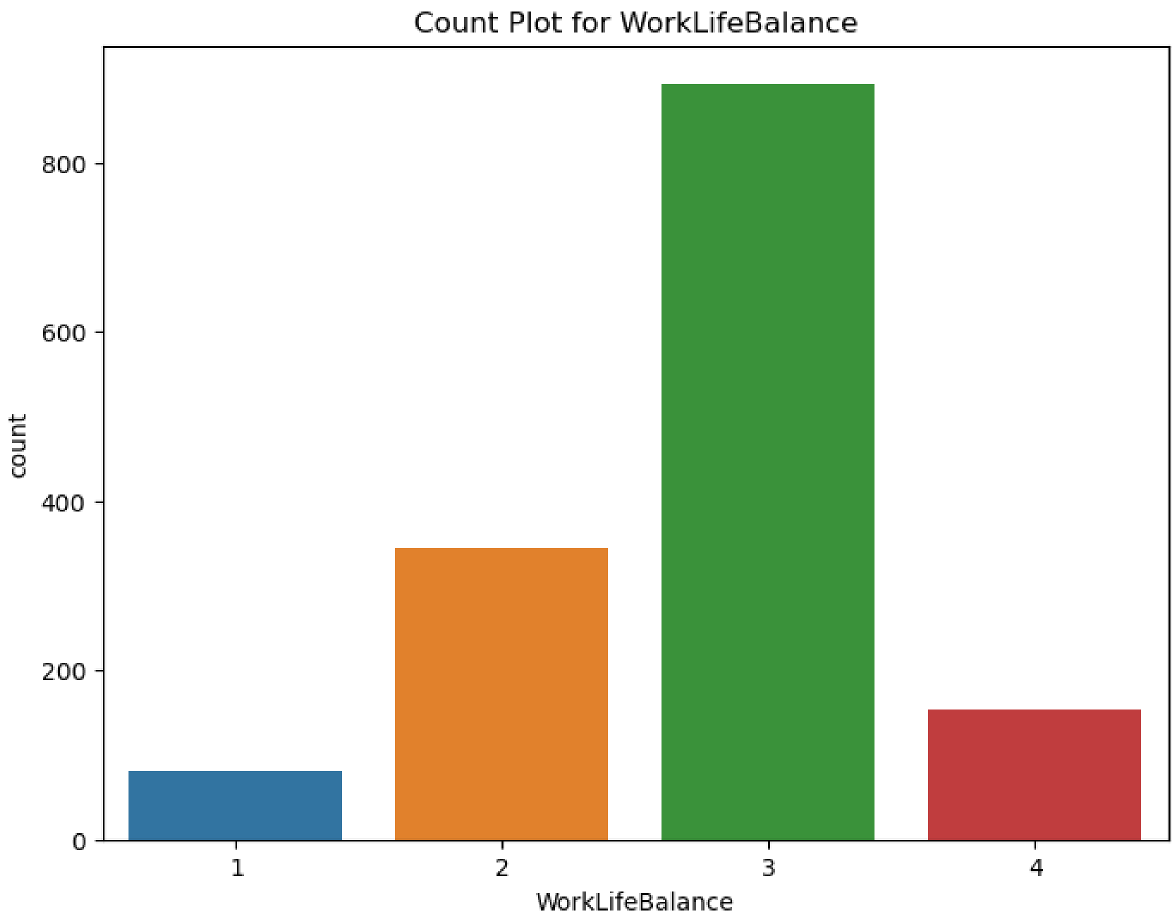

Count Plot for RelationshipSatisfaction
count
0
100
200
300
400
1
2
3
4
RelationshipSatisfaction

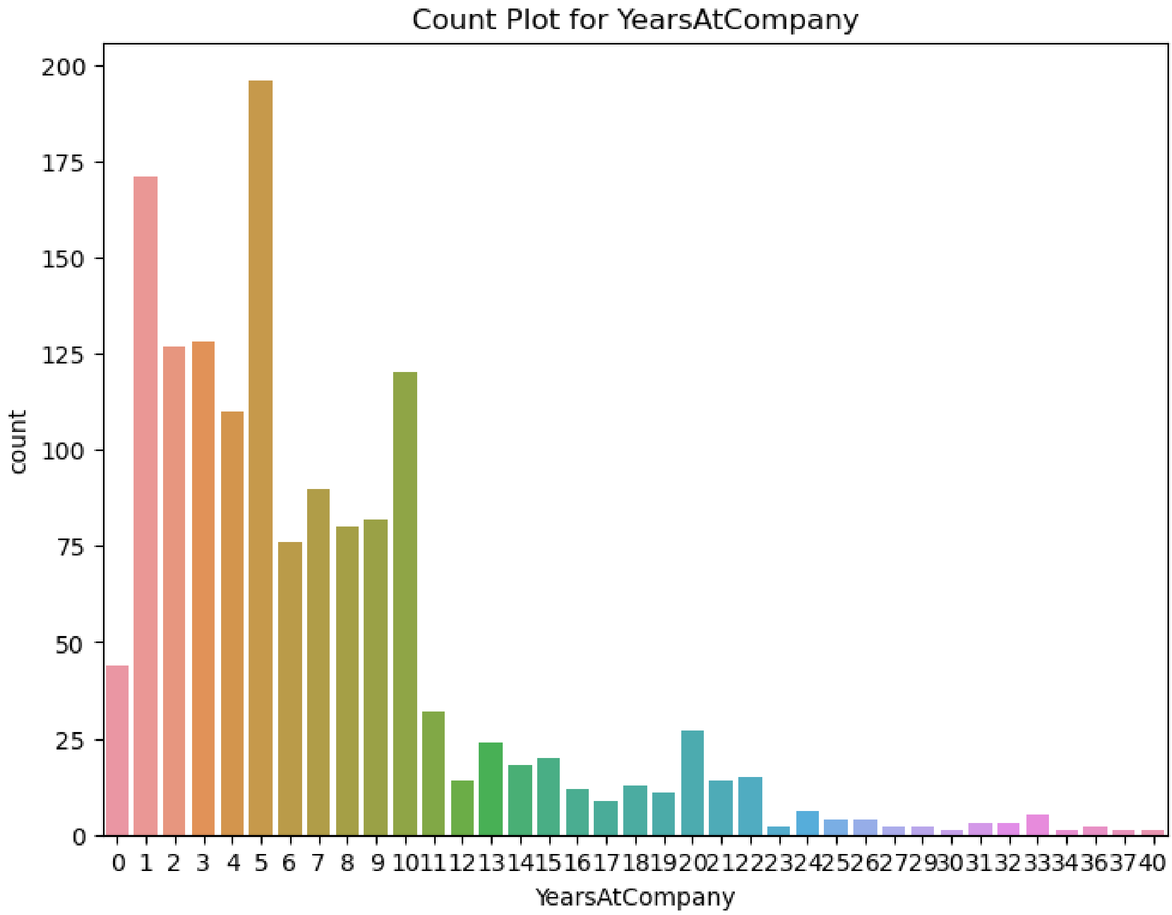

Count Plot for StandardHours
count
0
200
400
600
800
1000
1200
1400
80
StandardHours

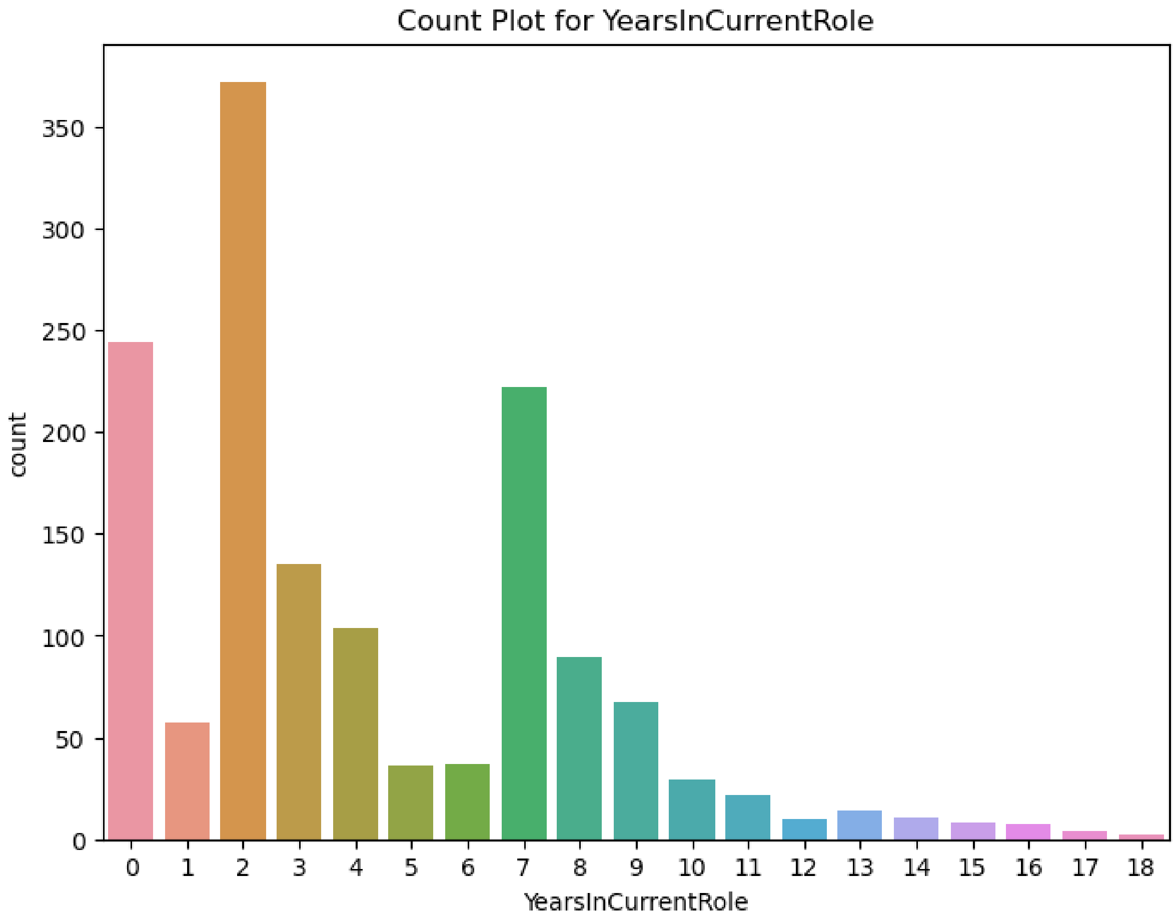

Count Plot for StockOptionLevel
count
600
500
400
300
200
100
0
0
1
2
3
StockOptionLevel

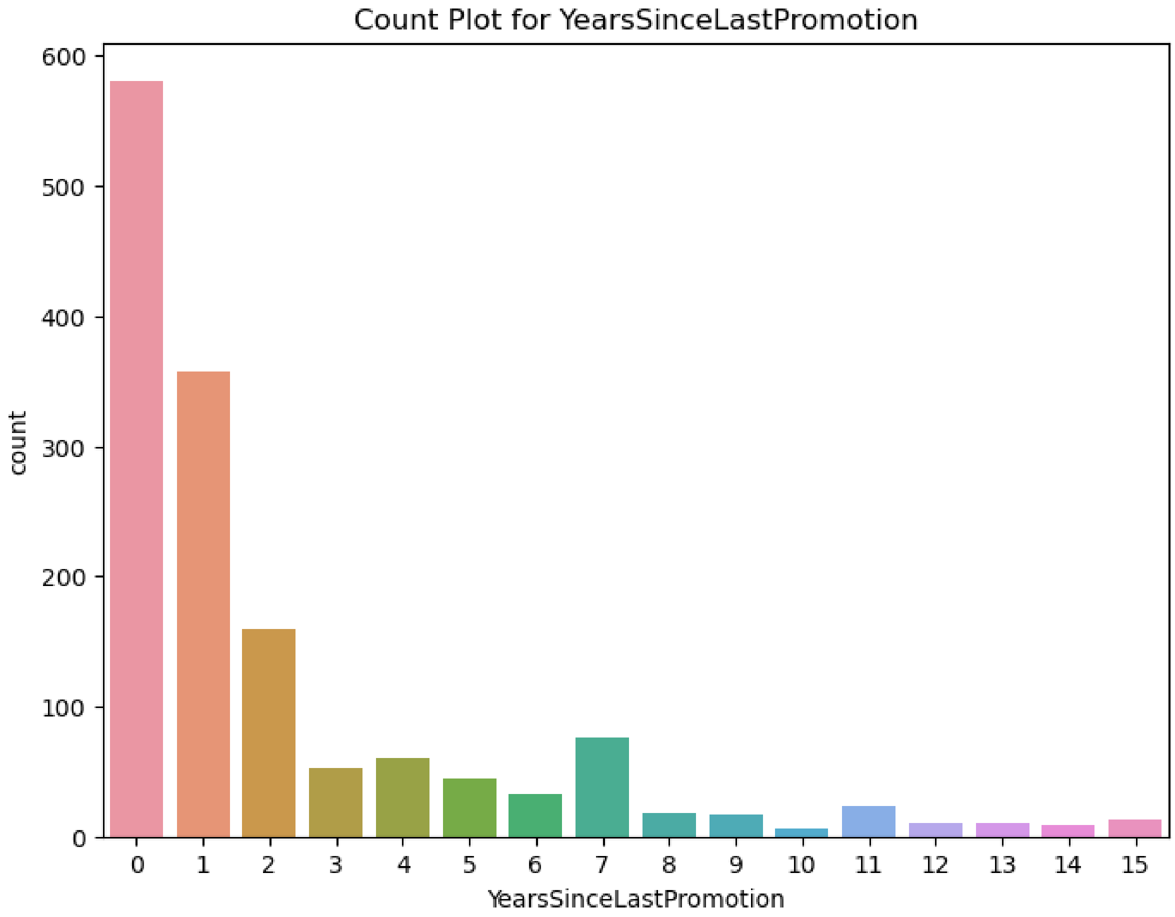

Count Plot for TotalWorkingYears
count
200
175
150
125
100
75
50
25
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 40
TotalWorkingYears

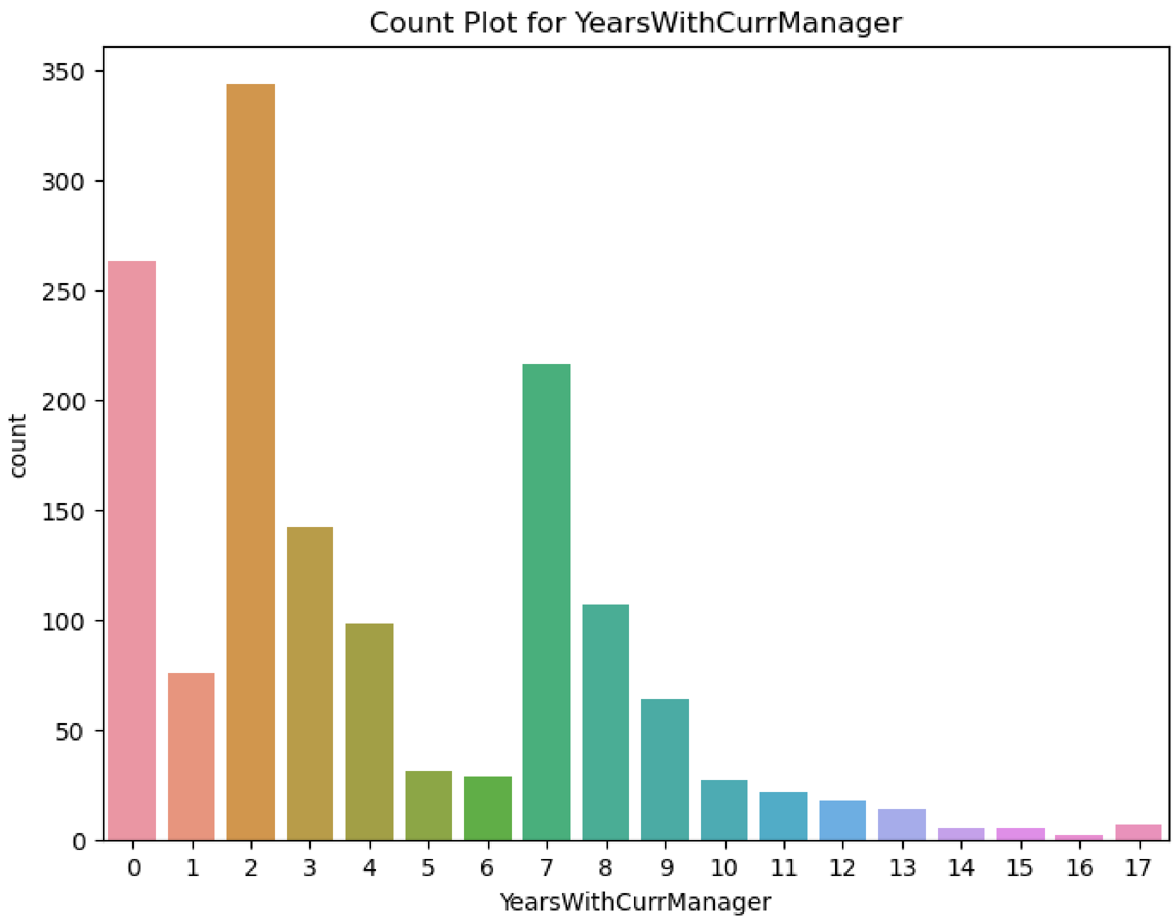

Count Plot for TrainingTimesLastYear
count
0
100
200
300
400
500
0
1
2
3
4
5
6
TrainingTimesLastYear


Count Plot for WorkLifeBalance
count
0
200
400
600
800
1
2
3
4
WorkLifeBalance


Count Plot for YearsAtCompany
200
175
150
125
100
75
50
25
0
count
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 29 30 31 32 33 34 36 37 40
YearsAtCompany


Count Plot for YearsInCurrentRole
350
300
250
200
150
100
50
0
count
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
YearsInCurrentRole

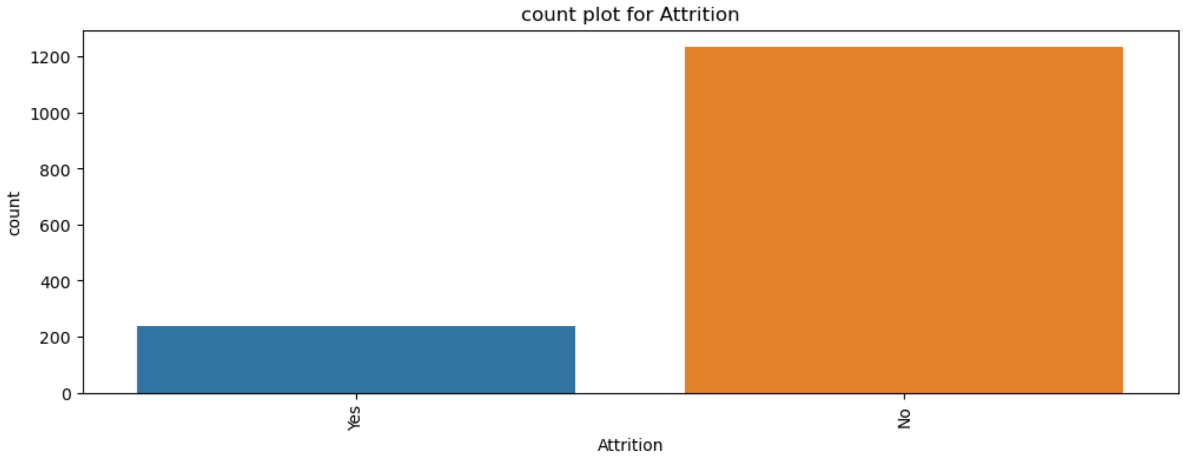


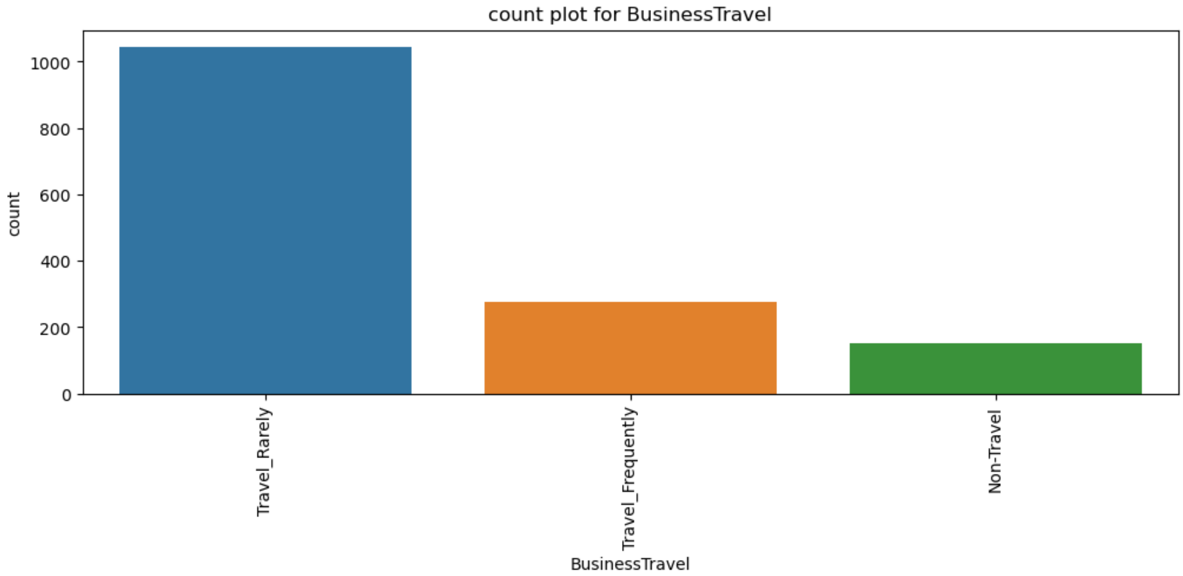


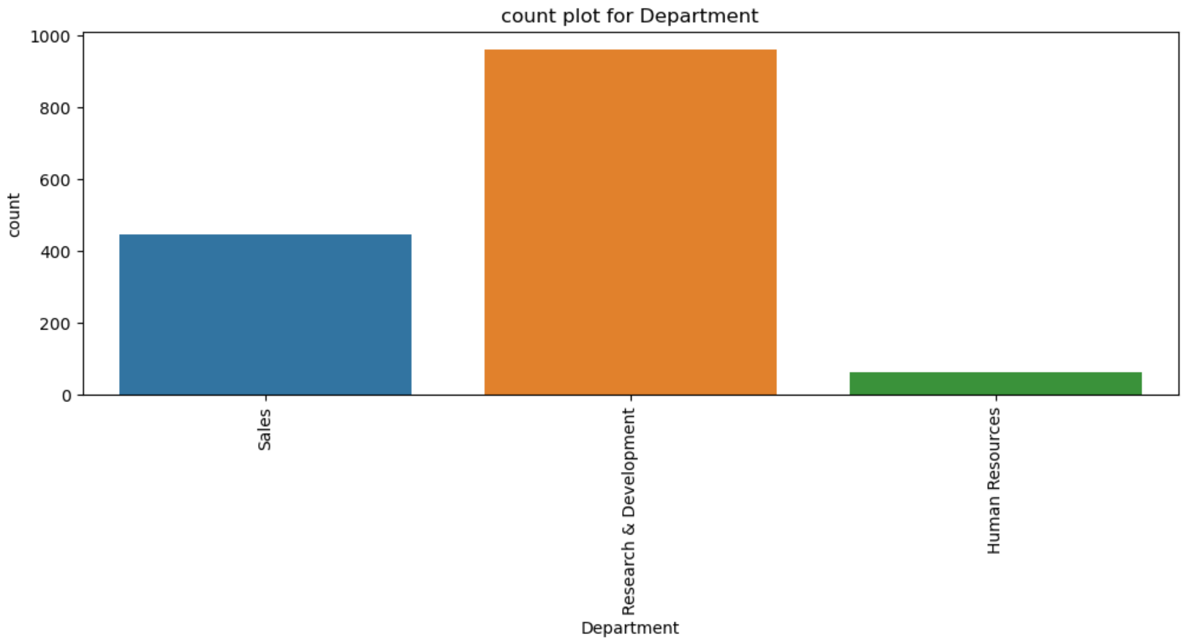
In [121…

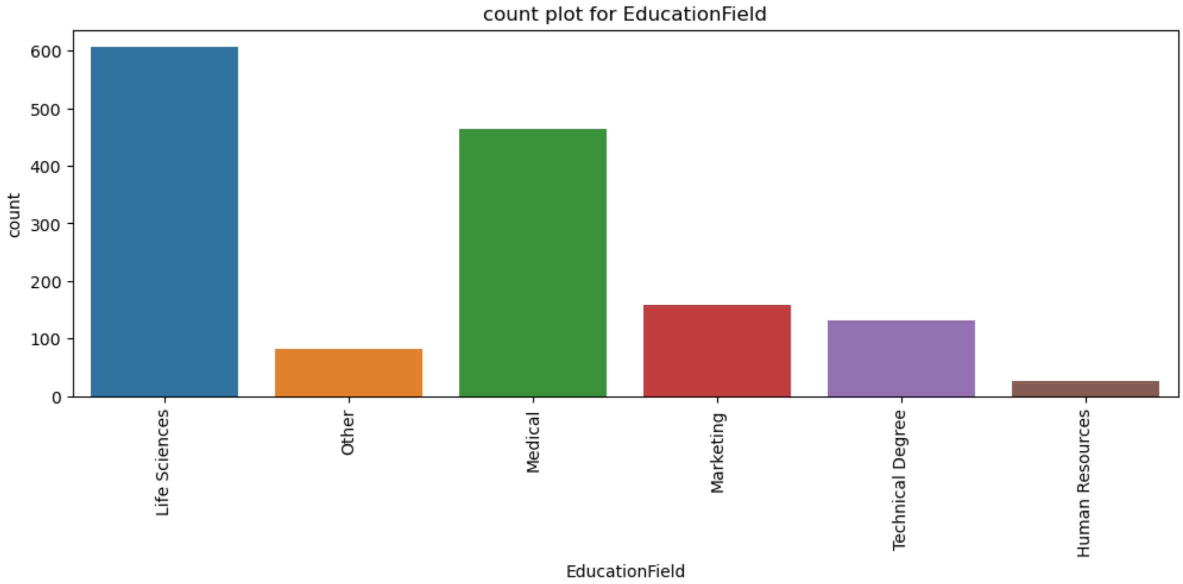
```
#from the info object, it is seen that the some data types are object. Therefore th

for column in employee_attrition.select_dtypes(include='object').columns:
    plt.figure(figsize=(12,4))
    plt.xticks(rotation=90)
```

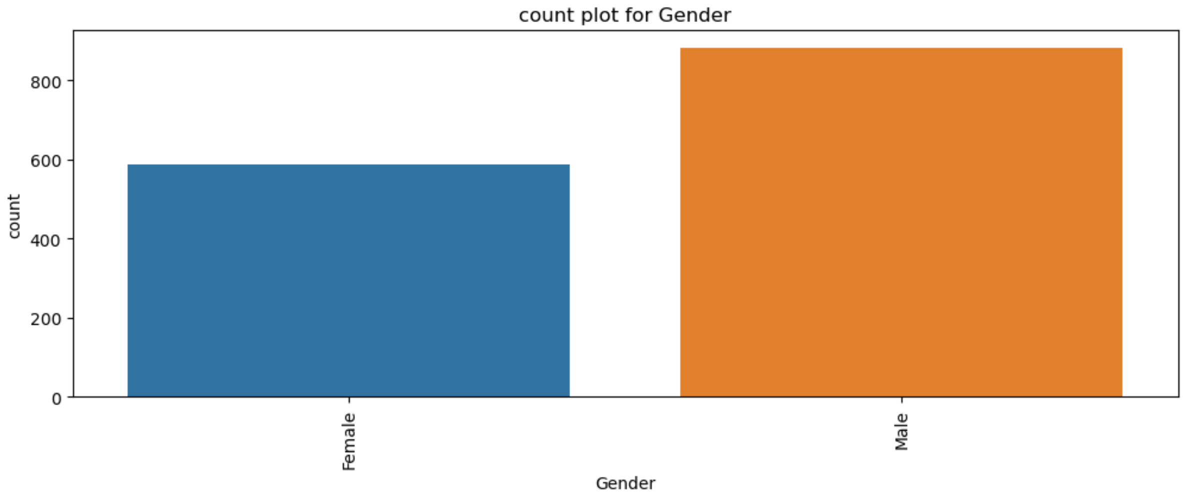
```
    sns.countplot(x=column, data=employee_attrition) #show the counts of observatic
    plt.title(f'count plot for {column}')
    plt.show()

#here include='object' is used to represent string or categorical variables in panc
```

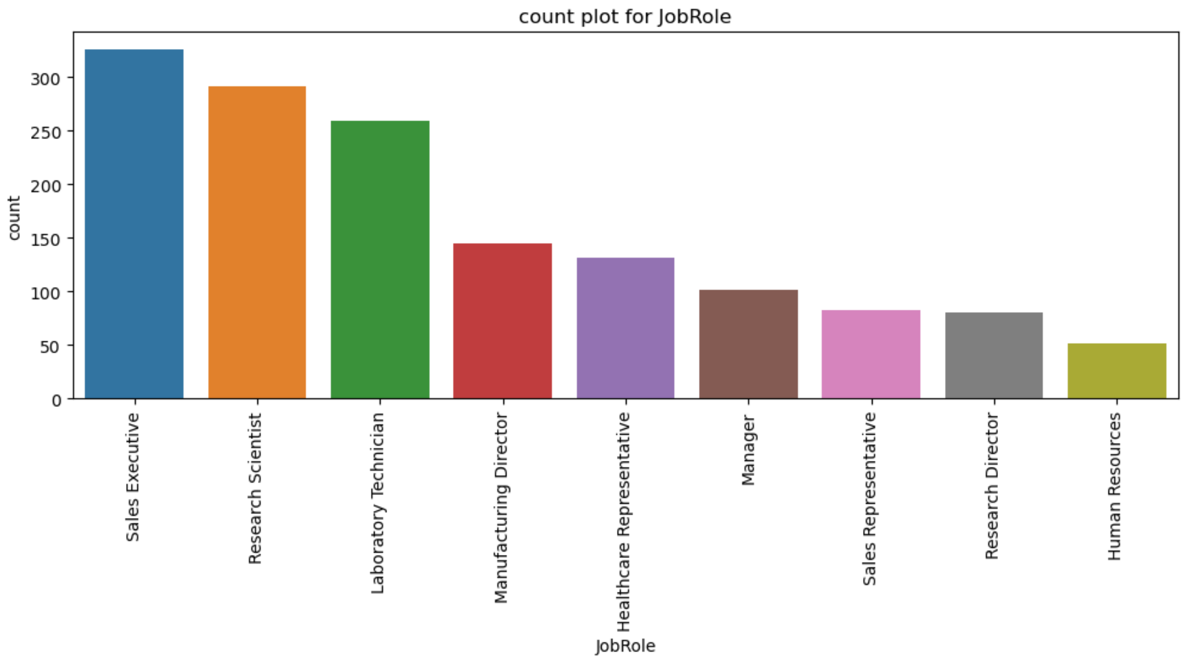


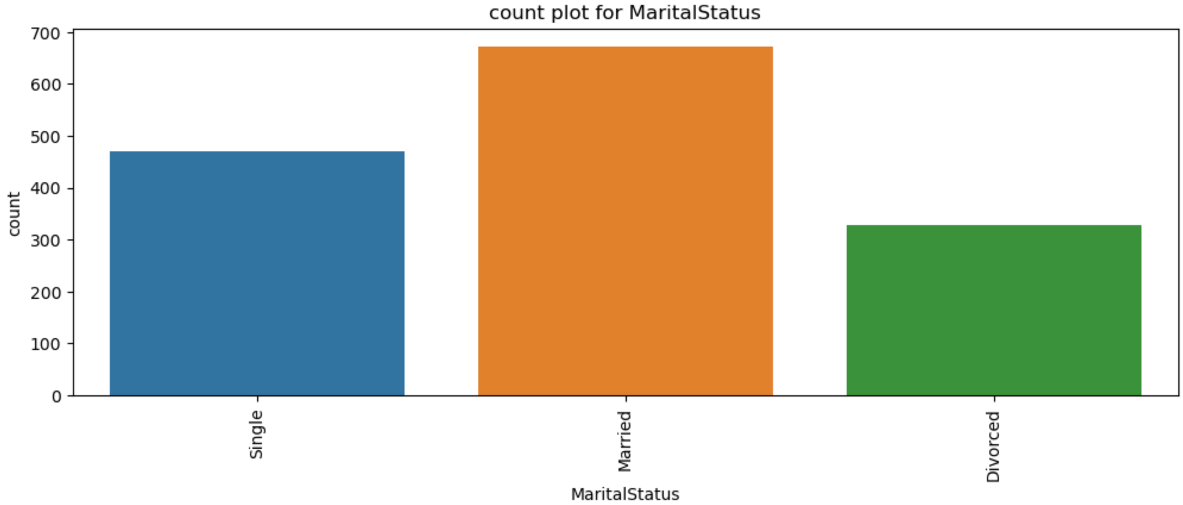


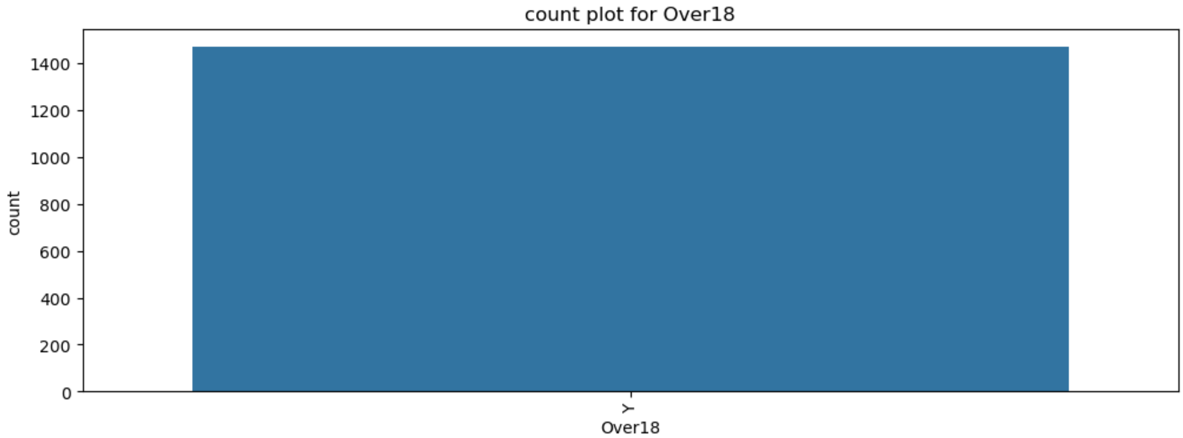


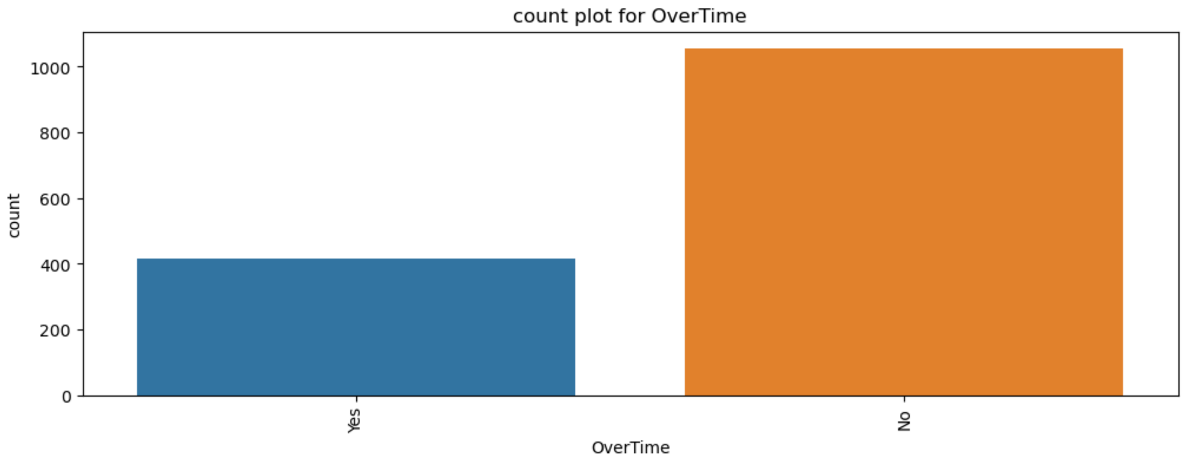

count plot for EducationField
count
0
100
200
300
400
500
600
Life Sciences
Other
Medical
Marketing
Technical Degree
Human Resources
EducationField


count plot for Gender
count
0
200
400
600
800
Female
Male
Gender


count plot for JobRole
count
0
50
100
150
200
250
300
Sales Executive
Research Scientist
Laboratory Technician
Manufacturing Director
Healthcare Representative
Manager
Sales Representative
Research Director
Human Resources
JobRole







In [122…

```
employee_attrition.columns
```

Out[122]:

```
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 [123…

```
object_columns = [col for col in employee_attrition.select_dtypes(include='object')
```

In [124…

```
object_columns
```

Out[124]:
```
['Attrition',
 'BusinessTravel',
 'Department',
 'EducationField',
 'Gender',
 'JobRole',
 'MaritalStatus',
 'Over18',
 'OverTime']
```

In [125…
```
selected_columns
```

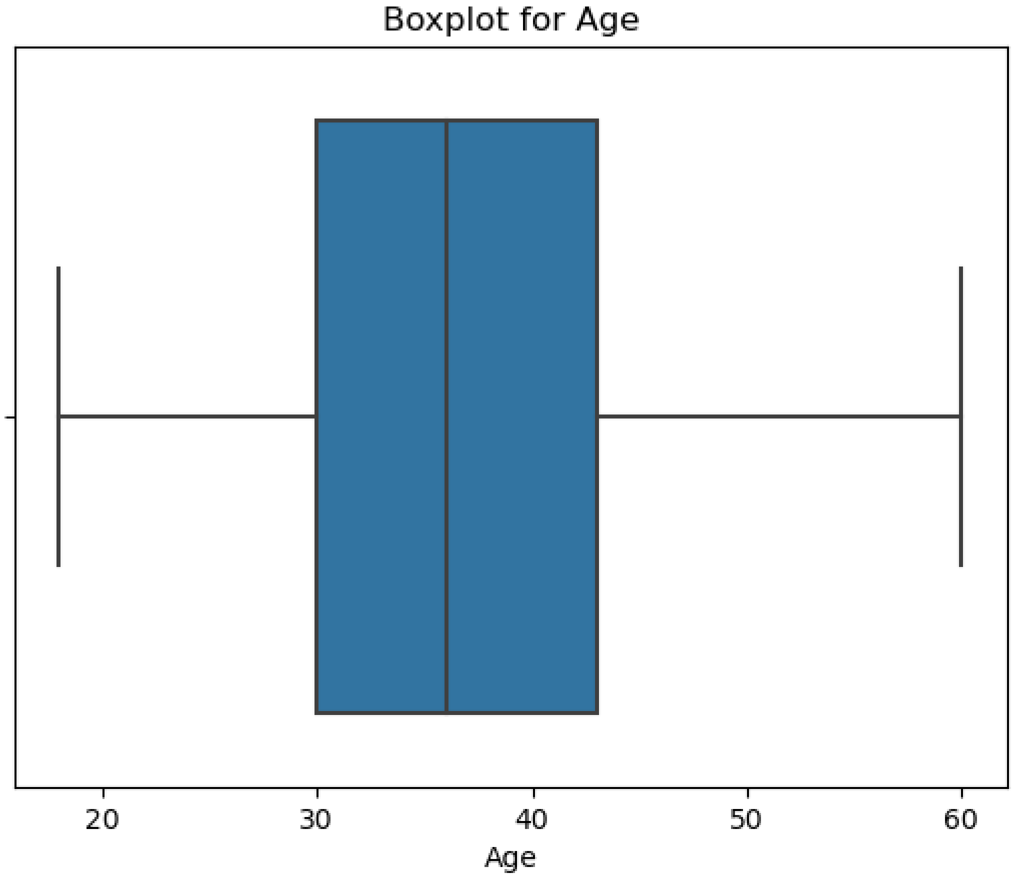
Out[125]:
```
['EnvironmentSatisfaction',
 'JobInvolvement',
 'JobLevel',
 'JobSatisfaction',
 'NumCompaniesWorked',
 'PercentSalaryHike',
 'PerformanceRating',
 'RelationshipSatisfaction',
 'StandardHours',
 'StockOptionLevel',
 'TotalWorkingYears',
 'TrainingTimesLastYear',
 'WorkLifeBalance',
 'YearsAtCompany',
 'YearsInCurrentRole',
 'YearsSinceLastPromotion',
 'YearsWithCurrManager']
```

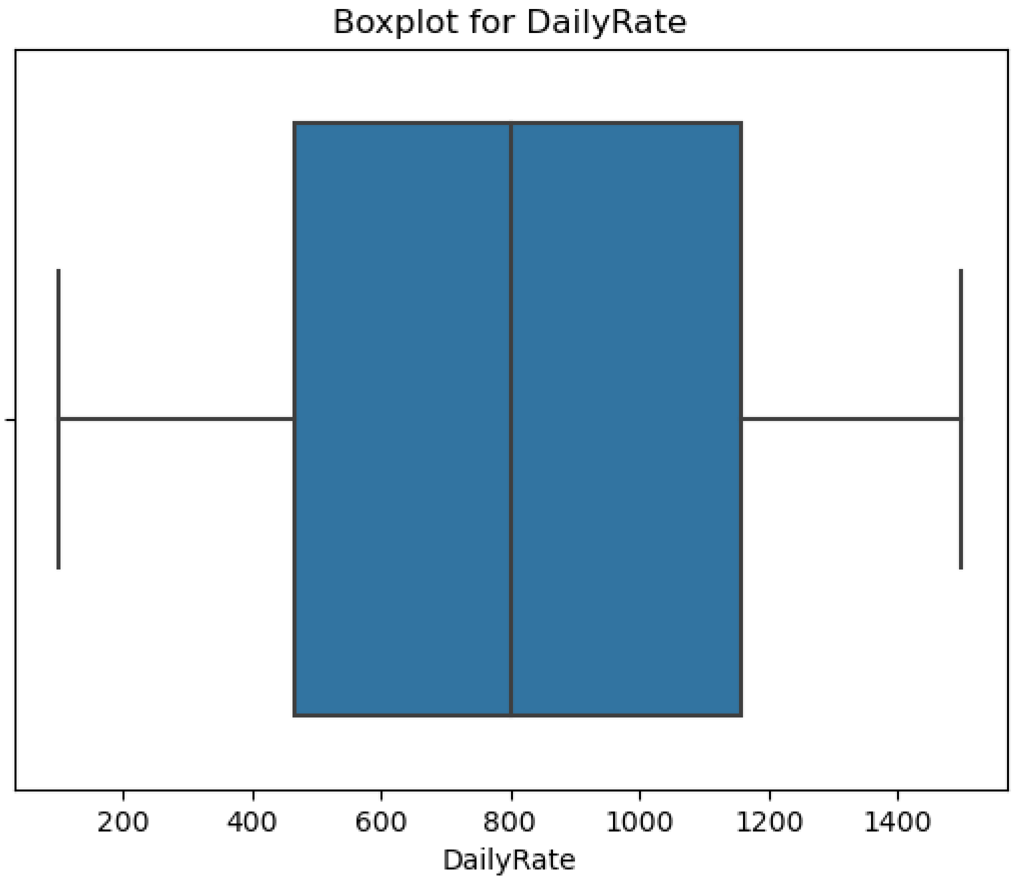
In [126…
```
other_columns = []
for col in employee_attrition.columns:
    if col not in selected_columns and employee_attrition[col].dtype != 'object':
        other_columns.append(col) #we will add the rest of the columns to the other
```

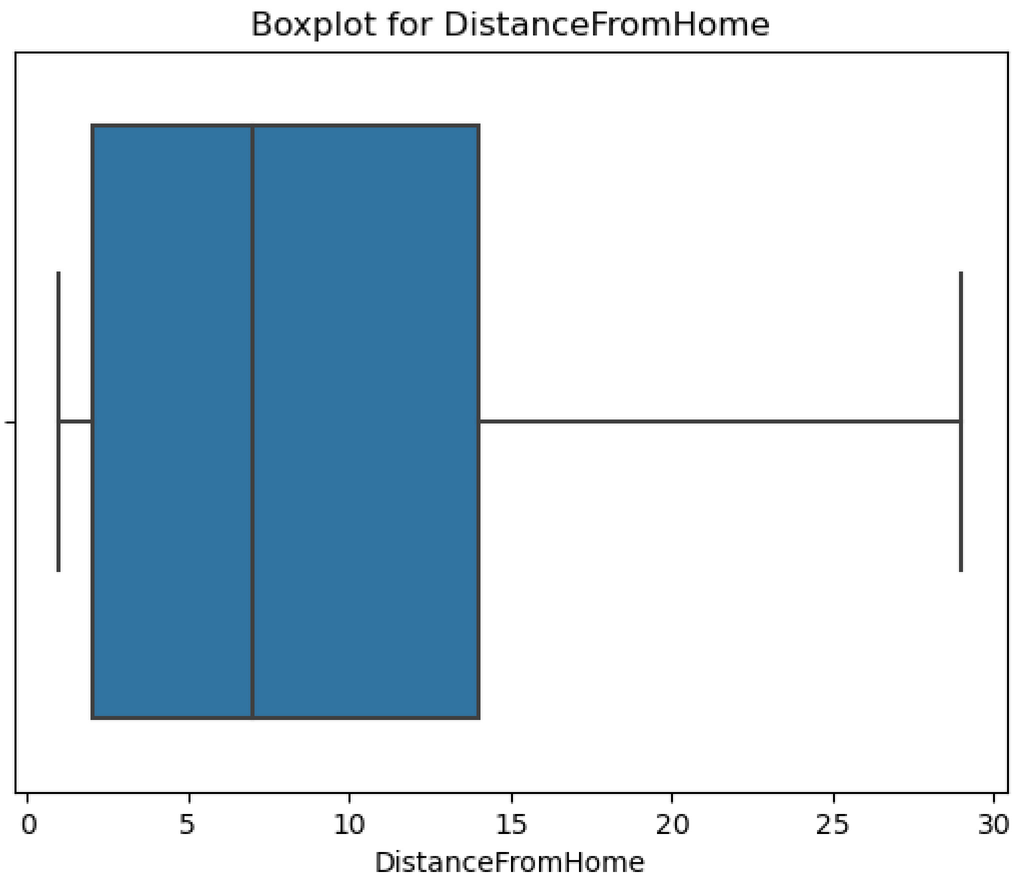
In [127…
```
other_columns #these columns were more suitable for boxplot representation
```

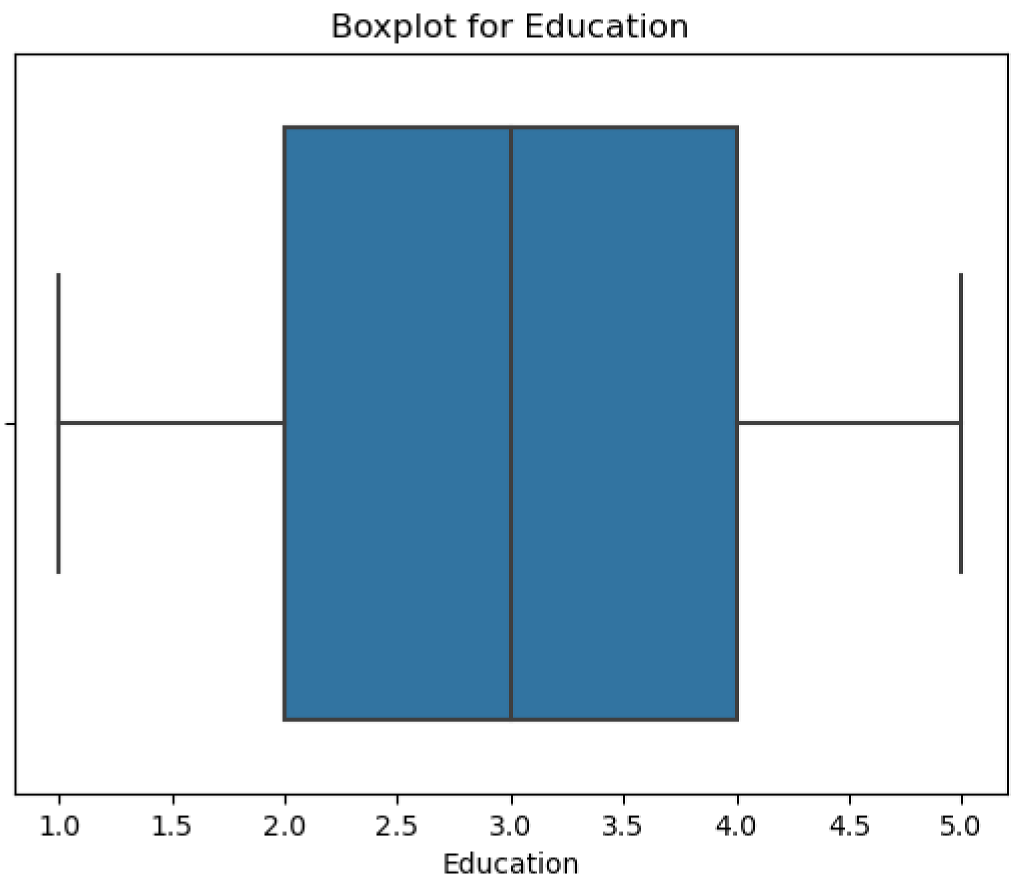
Out[127]:
```
['Age',
 'DailyRate',
 'DistanceFromHome',
 'Education',
 'EmployeeCount',
 'EmployeeNumber',
 'HourlyRate',
 'MonthlyIncome',
 'MonthlyRate']
```

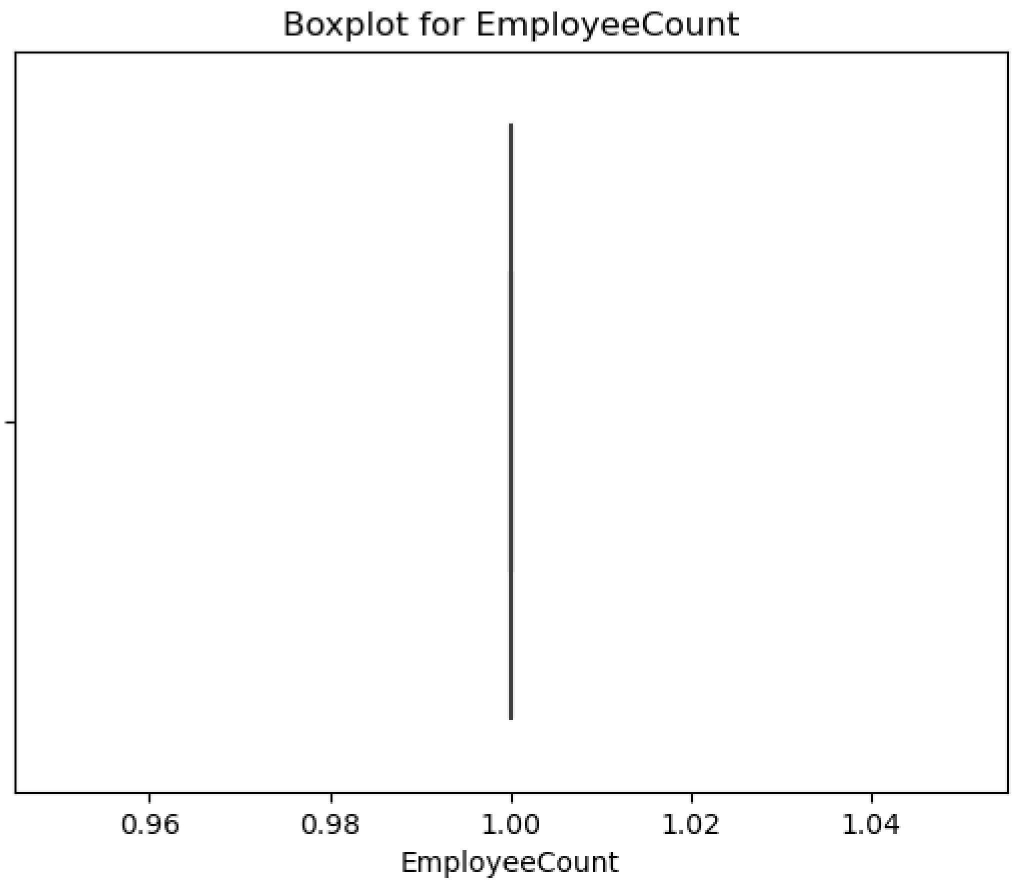
In [128…
```
for column in other_columns:
    sns.boxplot(x=column, data=employee_attrition)
    plt.title(f'Boxplot for {column}')
    plt.show()
```

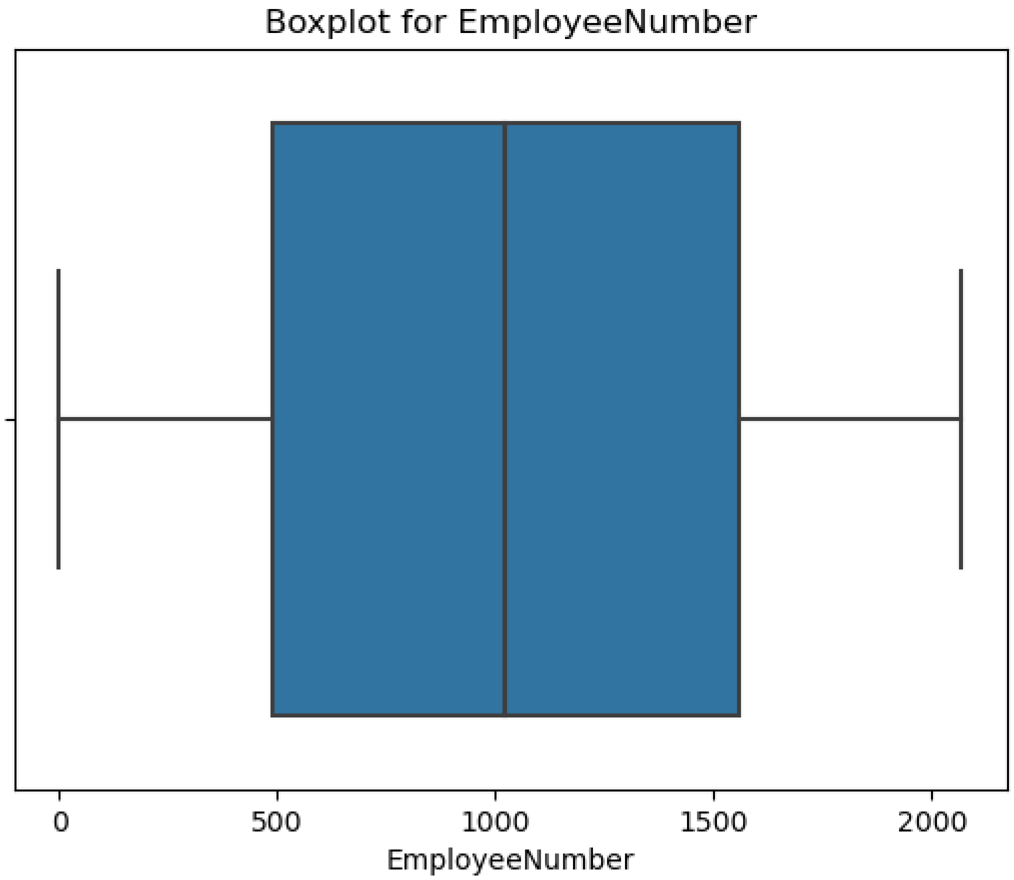

Boxplot for Age
20
30
40
50
60
Age

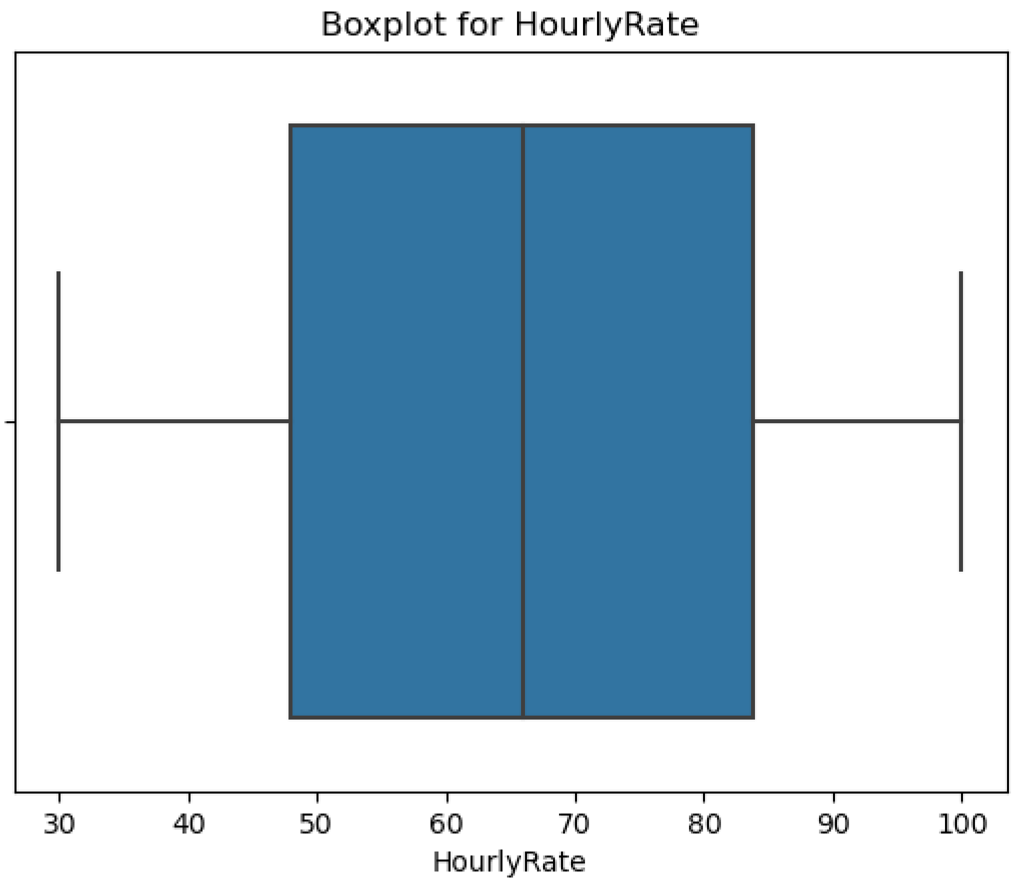

Boxplot for DailyRate
200
400
600
800
1000
1200
1400
DailyRate

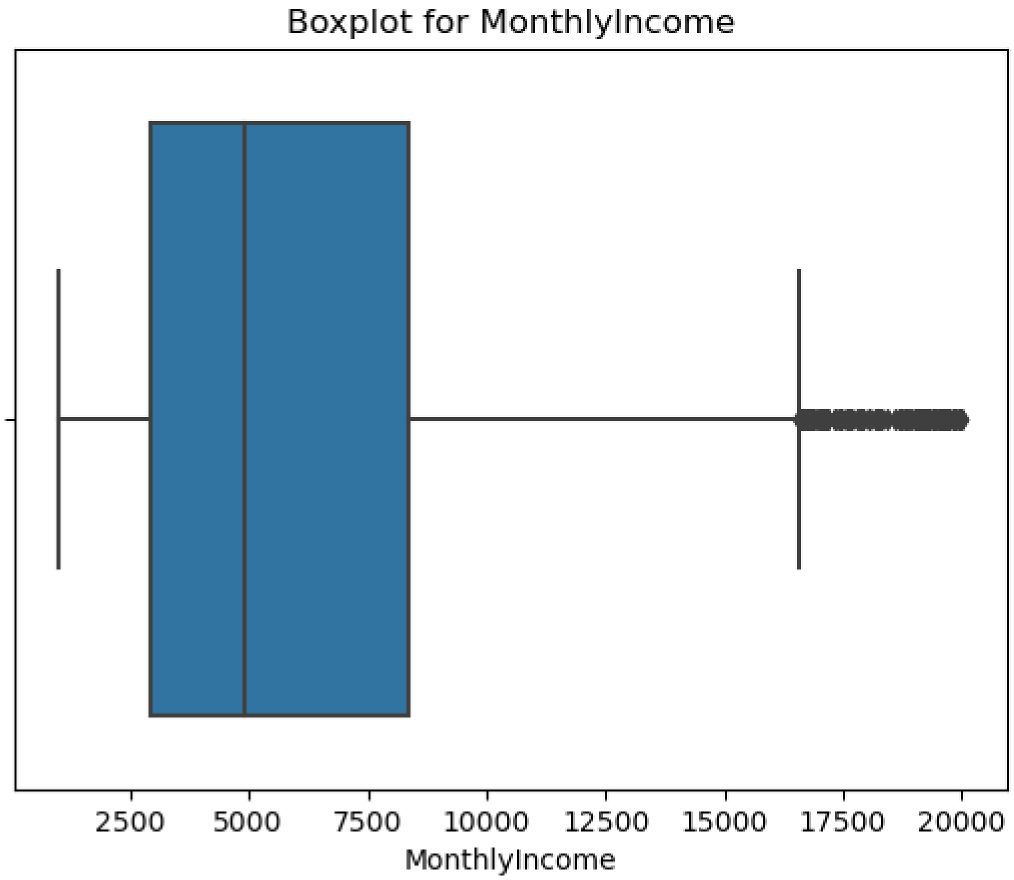

Boxplot for DistanceFromHome
0
5
10
15
20
25
30
DistanceFromHome

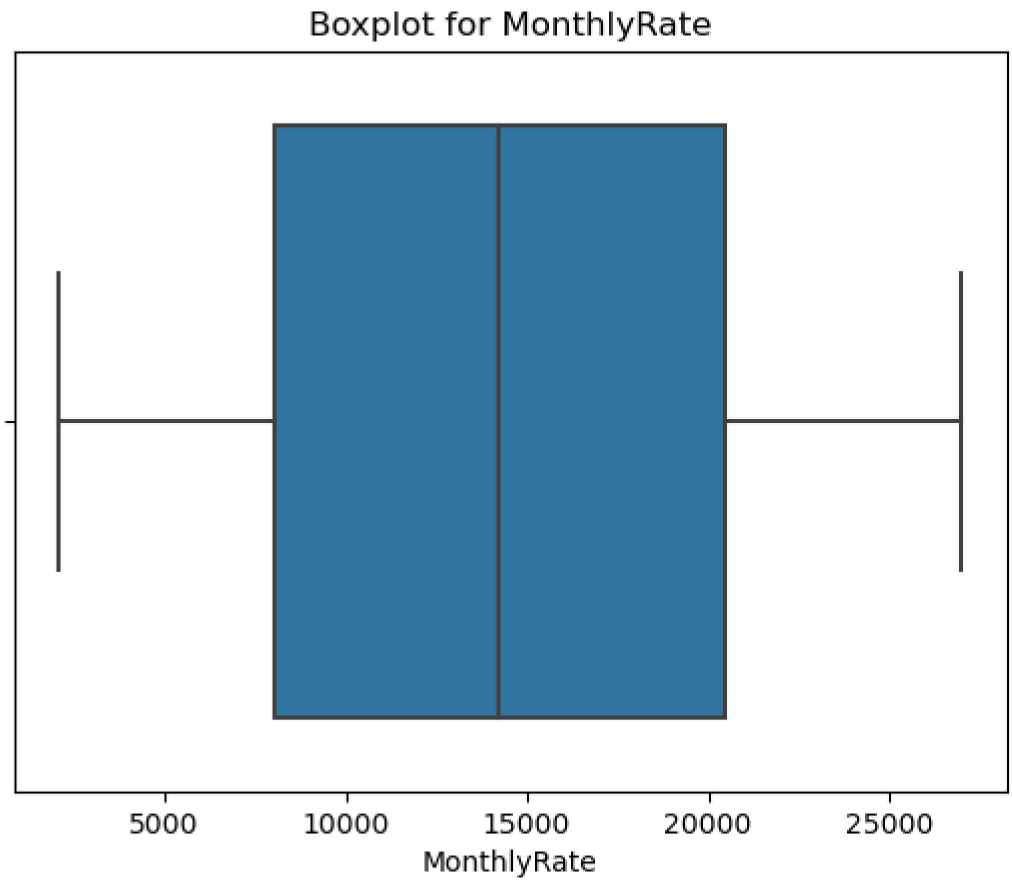

Boxplot for Education
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Education


Boxplot for EmployeeCount
0.96
0.98
1.00
1.02
1.04
EmployeeCount


Boxplot for EmployeeNumber
0
500
1000
1500
2000
EmployeeNumber


Boxplot for HourlyRate
30
40
50
60
70
80
90
100
HourlyRate


Boxplot for MonthlyIncome
2500
5000
7500
10000
12500
15000
17500
20000
MonthlyIncome



Data Processing

- missing values

In [129…

```
#handling missing values
missing_values = employee_attrition.isnull().sum()
print(missing_values) # we can see that there are no missing values
```

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

- encoding categorical variables

In [130…

```
# check the data types of each column
# previously the visualization is made possible by identifying data types from .inf
# but I will do it one more time

data_types = employee_attrition.dtypes
print(data_types)
```

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

In [131…

```
#these types of datas should be interpreted as int as well
print(data_types[data_types == 'object'].index)
```

```
Index(['Attrition', 'BusinessTravel', 'Department', 'EducationField', 'Gender',
       'JobRole', 'MaritalStatus', 'Over18', 'OverTime'],
      dtype='object')
```

In [132…

```
print(data_types[data_types == 'int64'].index)
```

```
Index(['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'],
      dtype='object')
```

In [133…

```
encoded_data = pd.get_dummies(employee_attrition, columns=object_columns, drop_firs
#Assume that all Yes are 1 and all 0 are No.
encoded_data.head()
```

Out[133]:

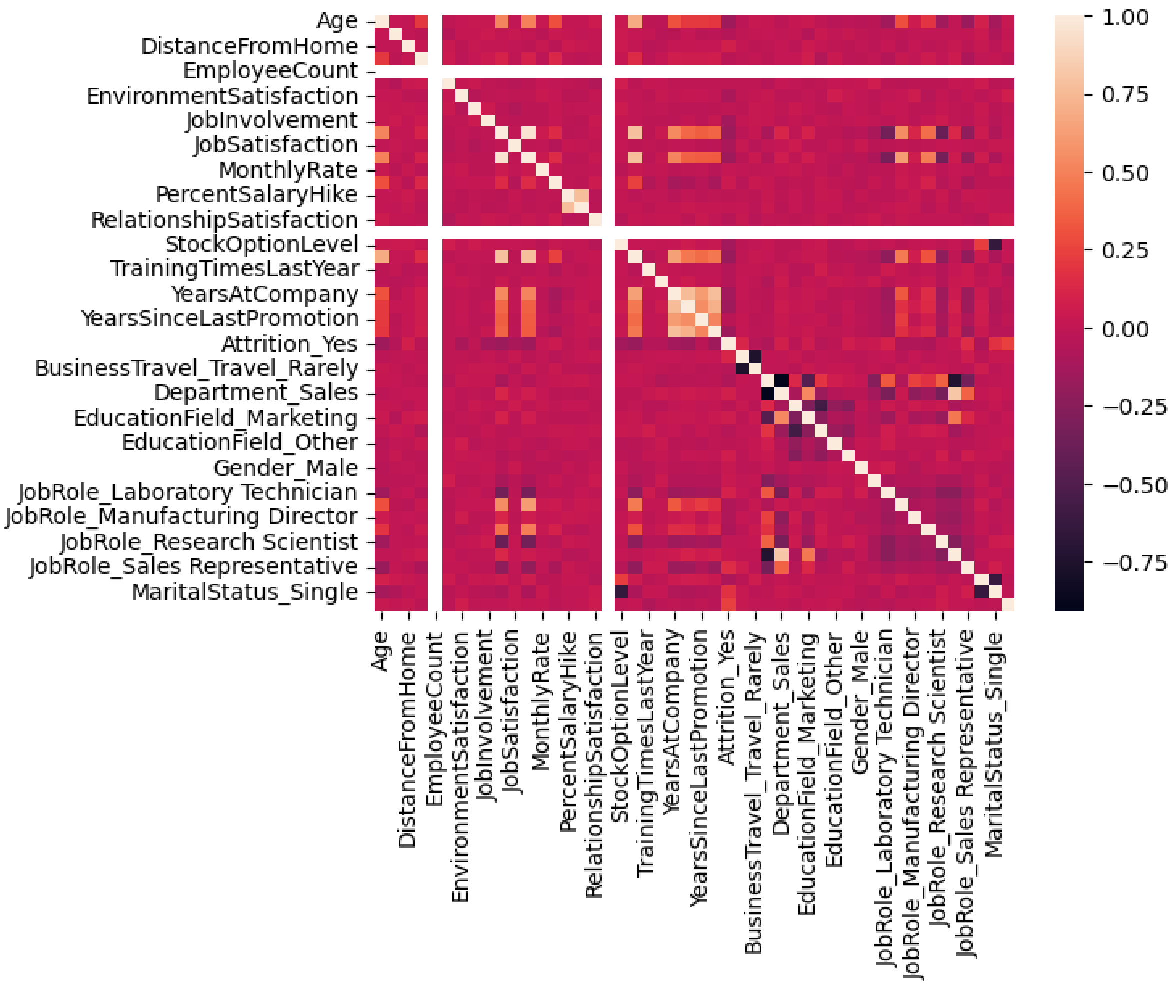
	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumber	Environmen
0	41	1102	1	2	1	1	
1	49	279	8	1	1	2	
2	37	1373	2	2	1	4	
3	33	1392	3	4	1	5	
4	27	591	2	1	1	7	

5 rows × 48 columns

In [134…

```
#Correlation plot
sns.heatmap(encoded_data.corr())
#from the figure the correlation seem to be mostly 0.
```

Out[134]: <Axes: >



In [135…

```
y = encoded_data['Attrition_Yes'].values #here is the y values in an array
X = encoded_data.drop('Attrition_Yes', axis=1).values
print(y)
print(X)
```

```
[1 0 1 ... 0 0 0]
[[  41 1102    1 ...    0    1    1]
 [  49  279    8 ...    1    0    0]
 [  37 1373    2 ...    0    1    1]
 ...
 [  27  155    4 ...    1    0    1]
 [  49 1023    2 ...    1    0    0]
 [  34  628    8 ...    1    0    0]]
```

- splitting the data as well as feature scaling with Standard Scaler

In [136…

```
#splitting data into final test set and remaining data(64/16/20 split)

#First the test set will be seperated from the training set by a 20%

X_remain, X_test, y_remain, y_test = train_test_split(X, y, test_size = 0.2, randon

print("X_test shape:" ,np.shape(X_test))
print("X_remain shape:" ,np.shape(X_remain))
print("y_test shape: ",np.shape(y_test))
print("y_remain shape:" ,np.shape(y_remain))

#After seperating train/test split, scaling is performed
#Fit on training data and transform training data

scaler = StandardScaler()
#fit_transform learns the transformation parameters from the training data and appl
X_train_scaled = scaler.fit_transform(X_remain)

#transform method applies the previously learned transformation to unseen data.
X_test_scaled = scaler.transform(X_test)
```

```
X_test shape: (294, 47)
X_remain shape: (1176, 47)
y_test shape:  (294,)
y_remain shape: (1176,)
```

5-Fold Cross Validation

In [137…

```
# Implementing 5-Fold Cross Validation
kf = KFold(n_splits = 5, shuffle = True, random_state = 42)
print(kf)

#for train_idx, cv_idx in kf.split(X_train_scaled, y_remain):
        #X_kf_train, X_kf_cv = X_train_scaled[train_idx], X_train_scaled[cv_idx]
        #y_kf_train, y_kf_cv = y_remain[train_idx], y_remain[cv_idx]
```

```
KFold(n_splits=5, random_state=42, shuffle=True)
```

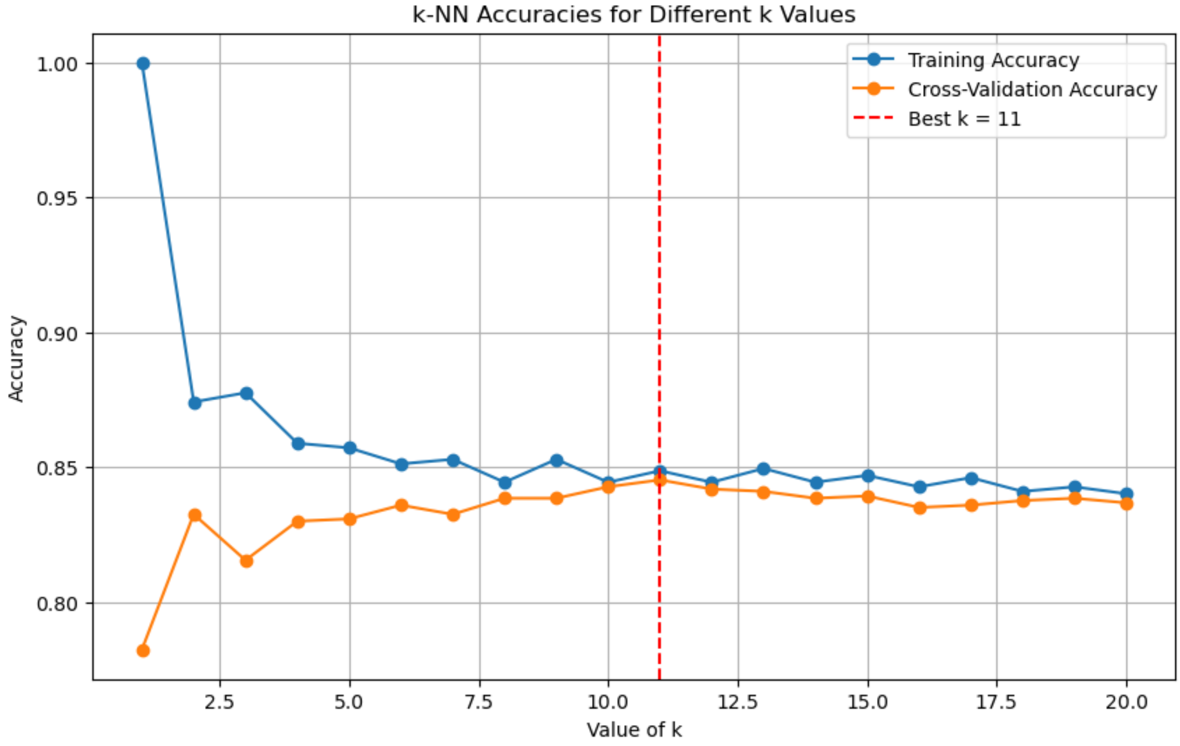
K-Nearest Neighbors (KNN)

In [138…

```
# first we can determine the k-value which represents the number of neighbors to cc

#this part is from the codes we went over in the lectures.
def calculate_accuracy(y_true, y_pred):
    "calculating the accuracy of true and predicted labels"
    return np.mean(y_true == y_pred)

cv_scores = []

train_scores = []
val_scores = []
```

```
#Looking for different k values
for k in range(1,21):
    knn = KNeighborsClassifier(n_neighbors = k)

    fold_scores = []

    for train_idx, cv_idx in kf.split(X_train_scaled, y_remain):
        X_kf_train, X_kf_cv = X_train_scaled[train_idx], X_train_scaled[cv_idx]
        y_kf_train, y_kf_cv = y_remain[train_idx], y_remain[cv_idx]

        knn.fit(X_kf_train, y_kf_train)
        y_pred_cv = knn.predict(X_kf_cv)
        fold_scores.append(calculate_accuracy(y_kf_cv, y_pred_cv))

    #print("k:", k ,"average CV accuracy for the corresponding k:" ,np.mean(fold_sc

    #taking the mean of the CV_results for the given k
    cv_scores.append(np.mean(fold_scores))

    #from the lecture notes
    # Training the model on the entire training set. I did include the val and trai
    knn.fit(X_train_scaled, y_remain)
    y_train_pred = knn.predict(X_train_scaled)
    train_scores.append(calculate_accuracy(y_remain, y_train_pred))

best_k = np.argmax(cv_scores) + 1
print("best_k: ", best_k)

# Plotting accuracies from the code in the lecture notes
plt.figure(figsize=(10, 6))
plt.plot(range(1, 21), train_scores, label='Training Accuracy', marker='o')
plt.plot(range(1, 21), cv_scores, label='Cross-Validation Accuracy', marker='o')

# Highlighting the best k value from CV
plt.axvline(x=best_k, color='r', linestyle='--', label=f'Best k = {best_k}')

plt.title('k-NN Accuracies for Different k Values')
plt.xlabel('Value of k')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```

```
best_k:  11
```



In [139…

```
# Now since we found the best_k we can test the accuracy now
best_knn = KNeighborsClassifier(n_neighbors = best_k) #best_k = 11

best_knn.fit(X_train_scaled, y_remain)

y_test_pred = best_knn.predict(X_test_scaled)

test_accuracy = calculate_accuracy(y_test, y_test_pred)

print(f"Test accuracy with best k: {k} is {test_accuracy:4f}")
```

```
Test accuracy with best k: 20 is 0.874150
```

Perceptron

In [140…

```
fold_scores = []
for train_idx, cv_idx in kf.split(X_train_scaled, y_remain):
    X_kf_train, X_kf_cv = X_train_scaled[train_idx], X_train_scaled[cv_idx]
    y_kf_train, y_kf_cv = y_remain[train_idx], y_remain[cv_idx]

    perceptron = Perceptron()
    perceptron.fit(X_kf_train, y_kf_train)
    y_train_pred = perceptron.predict(X_kf_cv)
    fold_scores.append(calculate_accuracy(y_kf_cv, y_train_pred))

print(np.mean(fold_scores))
#From the 5-Fold Cross Validation the robustness of our model can be seen.
```

```
0.8180165885322754
```

In [141…

```
# Create and train a perceptron model
perceptron = Perceptron()
perceptron.fit(X_train_scaled, y_remain)

# Make predictions on the test set
y_pred = perceptron.predict(X_test_scaled)

# Evaluate accuracy
```

```
accuracy = calculate_accuracy(y_test, y_pred)
print(f'Perceptron Accuracy: {accuracy}')
```

```
Perceptron Accuracy: 0.8639455782312925
```

Naive Bayes

In [145…

```
#We may need to split the categorical and numerical features
#considering that the features are independent. We can add them later.
from sklearn.naive_bayes import GaussianNB

#GaussianNB() will be used
nb_Gaussian = GaussianNB()
nb_Gaussian.fit(X_train_scaled, y_remain)

# Make predictions
y_pred = nb_Gaussian.predict(X_test_scaled)

# Evaluate the model
accuracy = calculate_accuracy(y_test, y_pred)
print(f'Gaussian Navie Bayes Accuracy: {accuracy}')
```

```
Gaussian Navie Bayes Accuracy: 0.6904761904761905
```

Part II: Gradient Descent Implementation

1. The "Vehicle Dataset" should be downloaded from the Kaggle website: https://www.kaggle.com/datasets/nehalbirla/vehicle-dataset-from-cardekho
2. Implement the gradient descent algorithm without using of any libraries except for Pandas and NumPy. [10 points]
3. How many iteration step needs to converge with learning rate [0.01, 0.1, 1]? Devise an intelligent strategy for choosing the learning rate to reduce the number of iterations required for convergence. Show how the learning rate that you propose impacts the convergence of the gradient descent algorithm. Show on the graph how the cost function changes with the number of iterations and how the gradient descent converges. [20 points]

In [ ]:

```
# Your code here (you can add more blocks as you need).
# Please add comments where you think necessary.

#we since we seperated the
nb_categorical = Mu
```

Your Discussion Here (You can double click and edit this block)

In [ ]:

```
#cite : https://towardsdatascience.com/what-is-feature-scaling-why-is-it-important-
```