Import the necessary packages

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import rcParams
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns
```

--

Load the Dataset

In [2]:

```
performance=pd.read_excel("C:\\Users\DELL\Desktop\Datamites projects\IABAC Mar2020\INX_Futu
performance.head()
```

Out[2]:

shipSatisfaction	TotalWorkExperienceInYears	TrainingTimesLastYear	EmpWorkLifeBalance	Experie
4	10	2	2	
4	20	2	3	
3	20	2	3	
2	23	2	2	
4	10	1	3	

head() function is used to return top n rows of a data frame or series.

In [3]:

```
performance.tail()
```

Out[3]:

	EmpNumber	Age	Gender	EducationBackground	MaritalStatus	EmpDepartment	EmpJob
1195	E100992	27	Female	Medical	Divorced	Sales	Exe
1196	E100993	37	Male	Life Sciences	Single	Development	S Deve
1197	E100994	50	Male	Medical	Married	Development	S Deve
1198	E100995	34	Female	Medical	Single	Data Science	Sci
1199	E100998	24	Female	Life Sciences	Single	Sales	Exe

5 rows × 28 columns

The tail() function is used to return the last n rows.

Perform Exploratory Data Analysis steps

Checking the datatypes of data

In [4]:

```
performance.dtypes
```

Out[4]:

```
EmpNumber                        object
Age                               int64
Gender                           object
EducationBackground              object
MaritalStatus                    object
EmpDepartment                    object
EmpJobRole                       object
BusinessTravelFrequency          object
DistanceFromHome                  int64
EmpEducationLevel                 int64
EmpEnvironmentSatisfaction        int64
EmpHourlyRate                     int64
EmpJobInvolvement                 int64
EmpJobLevel                       int64
EmpJobSatisfaction                int64
NumCompaniesWorked                int64
OverTime                         object
EmpLastSalaryHikePercent          int64
EmpRelationshipSatisfaction       int64
TotalWorkExperienceInYears        int64
TrainingTimesLastYear             int64
EmpWorkLifeBalance                int64
ExperienceYearsAtThisCompany      int64
ExperienceYearsInCurrentRole      int64
YearsSinceLastPromotion           int64
YearsWithCurrManager              int64
Attrition                        object
PerformanceRating                 int64
dtype: object
```

--

Data Cleaning

Find out the names of the columns in the data.

In [5]:

```
performance.columns
```

Out[5]:

```
Index(['EmpNumber', 'Age', 'Gender', 'EducationBackground', 'MaritalStatus',
       'EmpDepartment', 'EmpJobRole', 'BusinessTravelFrequency',
       'DistanceFromHome', 'EmpEducationLevel', 'EmpEnvironmentSatisfactio
n',
       'EmpHourlyRate', 'EmpJobInvolvement', 'EmpJobLevel',
       'EmpJobSatisfaction', 'NumCompaniesWorked', 'OverTime',
       'EmpLastSalaryHikePercent', 'EmpRelationshipSatisfaction',
       'TotalWorkExperienceInYears', 'TrainingTimesLastYear',
       'EmpWorkLifeBalance', 'ExperienceYearsAtThisCompany',
       'ExperienceYearsInCurrentRole', 'YearsSinceLastPromotion',
       'YearsWithCurrManager', 'Attrition', 'PerformanceRating'],
      dtype='object')
```

Display the information of all the fields present in the data

In [6]:

```
performance.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1200 entries, 0 to 1199
Data columns (total 28 columns):
 #   Column                        Non-Null Count  Dtype 
---  ------                        --------------  ----- 
 0   EmpNumber                     1200 non-null   object
 1   Age                           1200 non-null   int64 
 2   Gender                        1200 non-null   object
 3   EducationBackground           1200 non-null   object
 4   MaritalStatus                 1200 non-null   object
 5   EmpDepartment                 1200 non-null   object
 6   EmpJobRole                    1200 non-null   object
 7   BusinessTravelFrequency       1200 non-null   object
 8   DistanceFromHome              1200 non-null   int64 
 9   EmpEducationLevel             1200 non-null   int64 
 10  EmpEnvironmentSatisfaction    1200 non-null   int64 
 11  EmpHourlyRate                 1200 non-null   int64 
 12  EmpJobInvolvement             1200 non-null   int64 
 13  EmpJobLevel                   1200 non-null   int64 
 14  EmpJobSatisfaction            1200 non-null   int64 
 15  NumCompaniesWorked            1200 non-null   int64 
 16  OverTime                      1200 non-null   object
 17  EmpLastSalaryHikePercent      1200 non-null   int64 
 18  EmpRelationshipSatisfaction   1200 non-null   int64 
 19  TotalWorkExperienceInYears    1200 non-null   int64 
 20  TrainingTimesLastYear         1200 non-null   int64 
 21  EmpWorkLifeBalance            1200 non-null   int64 
 22  ExperienceYearsAtThisCompany  1200 non-null   int64 
 23  ExperienceYearsInCurrentRole  1200 non-null   int64 
 24  YearsSinceLastPromotion       1200 non-null   int64 
 25  YearsWithCurrManager          1200 non-null   int64 
 26  Attrition                     1200 non-null   object
 27  PerformanceRating             1200 non-null   int64 
dtypes: int64(19), object(9)
memory usage: 262.6+ KB
```

The describe() function computes a summary of statistics pertaining to the DataFrame columns

In [7]:

```
performance.describe()
```

Out[7]:

	Age	DistanceFromHome	EmpEducationLevel	EmpEnvironmentSatisfaction	Empl
count	1200.000000	1200.000000	1200.00000	1200.000000	12
mean	36.918333	9.165833	2.89250	2.715833	
std	9.087289	8.176636	1.04412	1.090599	
min	18.000000	1.000000	1.00000	1.000000	
25%	30.000000	2.000000	2.00000	2.000000	
50%	36.000000	7.000000	3.00000	3.000000	
75%	43.000000	14.000000	4.00000	4.000000	
max	60.000000	29.000000	5.00000	4.000000	

Shape is a tuple that gives dimensions of the array

In [8]:

```
performance.shape
```

Out[8]:

```
(1200, 28)
```

Checking for the null values in the data

In [9]:

```
performance.isnull().sum().to_frame().T
```

Out[9]:

	EmpNumber	Age	Gender	EducationBackground	MaritalStatus	EmpDepartment	EmpJobRo
0	0	0	0	0	0	0	

1 rows × 28 columns

In [10]:

```
performance.isnull().sum().to_frame().any()
```

Out[10]:

```
0    False
dtype: bool
```

Count() function returns count of how many times a given object occurs in list

In [11]:

```
performance.count()
```

Out[11]:

```
EmpNumber                       1200
Age                             1200
Gender                          1200
EducationBackground             1200
MaritalStatus                   1200
EmpDepartment                   1200
EmpJobRole                      1200
BusinessTravelFrequency         1200
DistanceFromHome                1200
EmpEducationLevel               1200
EmpEnvironmentSatisfaction      1200
EmpHourlyRate                   1200
EmpJobInvolvement               1200
EmpJobLevel                     1200
EmpJobSatisfaction              1200
NumCompaniesWorked              1200
OverTime                        1200
EmpLastSalaryHikePercent        1200
EmpRelationshipSatisfaction     1200
TotalWorkExperienceInYears      1200
TrainingTimesLastYear           1200
EmpWorkLifeBalance              1200
ExperienceYearsAtThisCompany    1200
ExperienceYearsInCurrentRole    1200
YearsSinceLastPromotion         1200
YearsWithCurrManager            1200
Attrition                       1200
PerformanceRating               1200
dtype: int64
```

---

1) Department wise Performance Analysis

In [12]:

```
performance.groupby('EmpDepartment')['PerformanceRating'].count()
```

Out[12]:

```
EmpDepartment
Data Science               20
Development               361
Finance                    49
Human Resources            54
Research & Development    343
Sales                     373
Name: PerformanceRating, dtype: int64
```

In [13]:

```
performance.groupby(by='EmpDepartment')['PerformanceRating'].mean()
```

Out[13]:

```
EmpDepartment
Data Science              3.050000
Development               3.085873
Finance                   2.775510
Human Resources           2.925926
Research & Development    2.921283
Sales                     2.860590
Name: PerformanceRating, dtype: float64
```

1.1) Visualization carried out for Department wise Performance Analysis

Performance Rating Analysis of each department

In [14]:

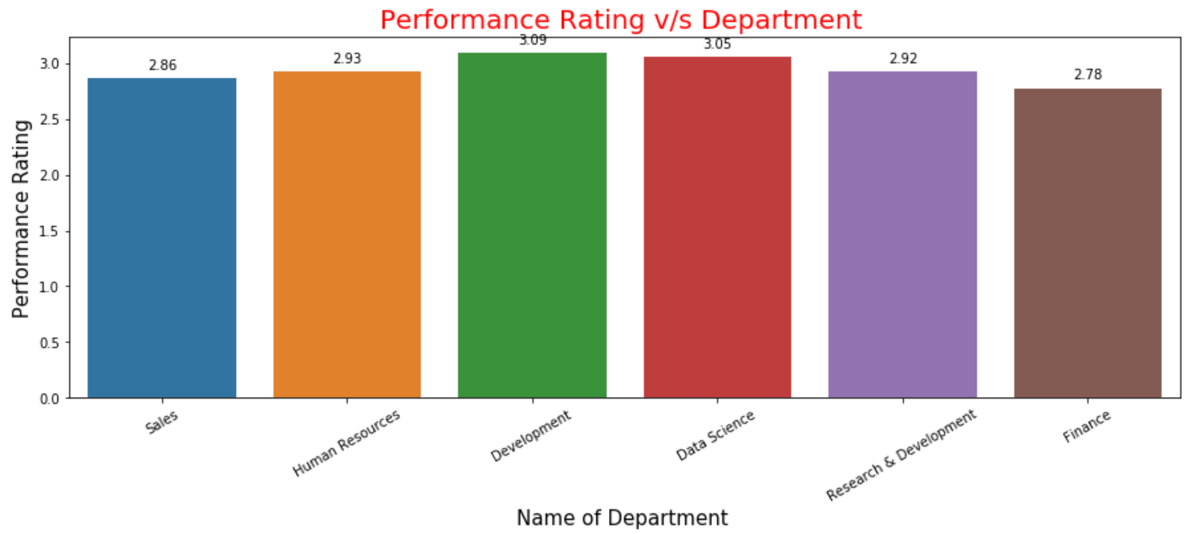
```
performance.groupby(by=['EmpDepartment'])['PerformanceRating'].mean()
```

Out[14]:

```
EmpDepartment
Data Science              3.050000
Development               3.085873
Finance                   2.775510
Human Resources           2.925926
Research & Development    2.921283
Sales                     2.860590
Name: PerformanceRating, dtype: float64
```

In [15]:

```
plt.figure(figsize=(15,5))
splot=sns.barplot(performance['EmpDepartment'],performance['PerformanceRating'],ci=None)
plt.xticks(rotation=30)
plt.xlabel("Name of Department ",fontsize=15,color='black')
plt.ylabel(" Performance Rating ",fontsize=15,color='black')
plt.title("Performance Rating v/s Department ",fontdict={'fontsize':20,'color':'Red'})
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he
```



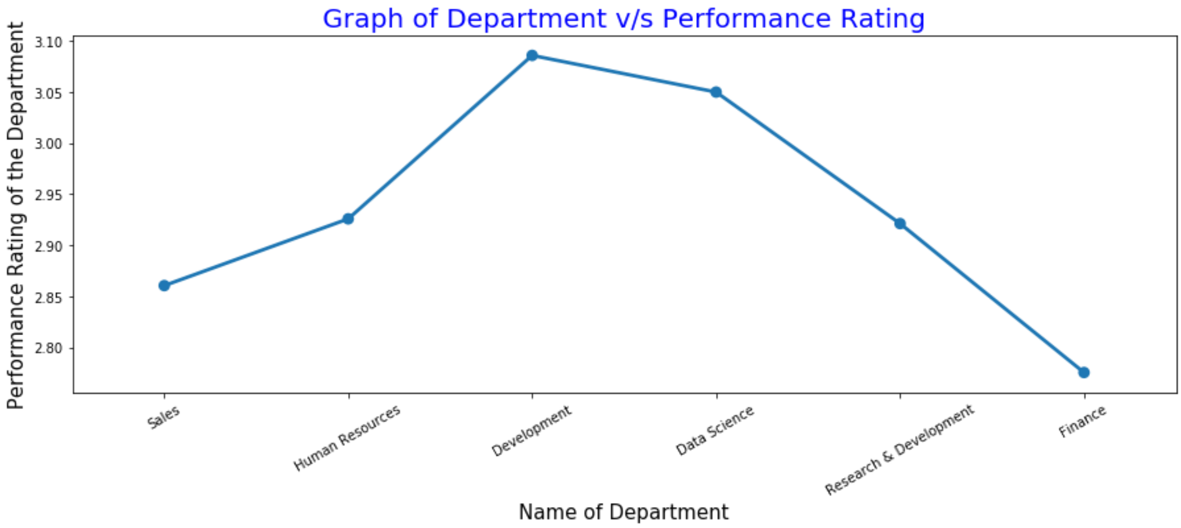
Performance Rating of '**Development**' Department is the highest and '**Finance**' Department is the lowest.

In [16]:

```
plt.figure(figsize=(15,5))
splot=sns.pointplot(performance['EmpDepartment'],performance['PerformanceRating'],ci=None)
plt.xticks(rotation=30)
plt.xlabel("Name of Department",fontsize=15,color='black')
plt.ylabel(" Performance Rating of the Department ",fontsize=15,color='black')
plt.title(" Graph of Department v/s Performance Rating ",fontdict={'fontsize':20,'color':'B
```

Out[16]:

```
Text(0.5, 1.0, ' Graph of Department v/s Performance Rating ')
```



Here '**Development**' department has the highest mean Performance Rating and '**Finance**' department has the lowest mean Performance Rating.

Performance Rating Analysis of each department with respect to Male and Female

In [17]:

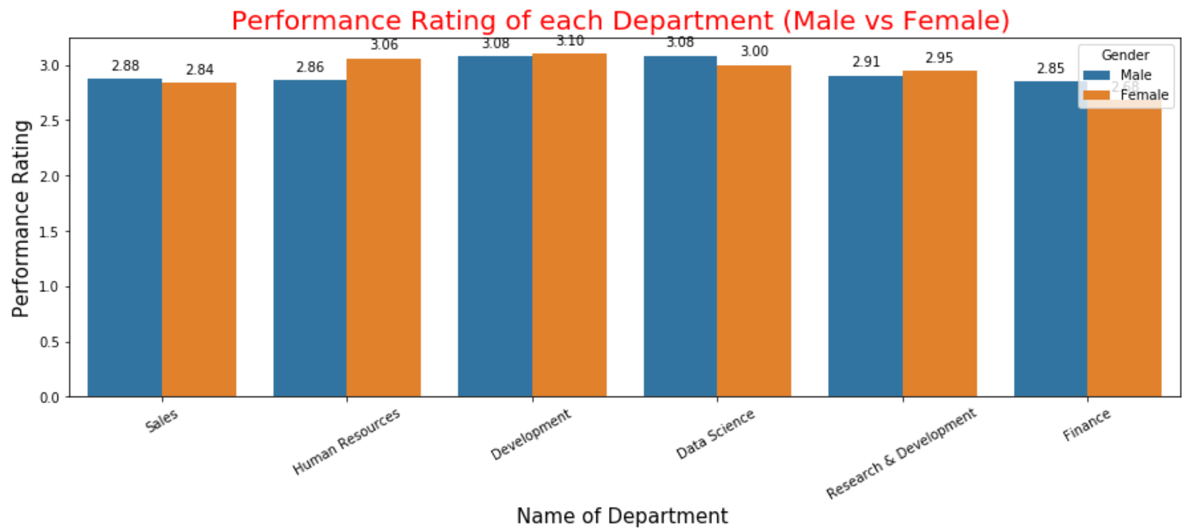
```
performance.groupby(by=['EmpDepartment' ,'Gender'])['PerformanceRating'].mean()
```

Out[17]:

```
EmpDepartment           Gender
Data Science            Female    3.000000
                        Male      3.083333
Development             Female    3.098592
                        Male      3.077626
Finance                 Female    2.681818
                        Male      2.851852
Human Resources         Female    3.058824
                        Male      2.864865
Research & Development  Female    2.945736
                        Male      2.906542
Sales                   Female    2.840764
                        Male      2.875000
Name: PerformanceRating, dtype: float64
```

In [18]:

```
plt.figure(figsize=(15,5))
splot=sns.barplot(performance['EmpDepartment'],performance['PerformanceRating'],performance
plt.xticks(rotation=30)
plt.xlabel("Name of Department ",fontsize=15,color='black')
plt.ylabel(" Performance Rating ",fontsize=15,color='black')
plt.title("Performance Rating of each Department (Male vs Female) ",fontdict={'fontsize':20
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he
```



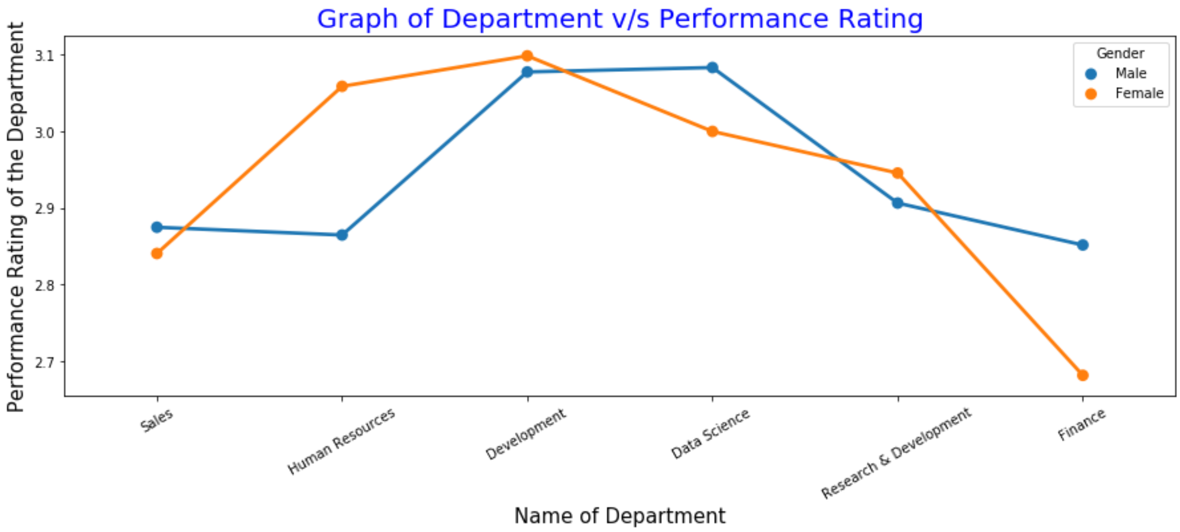
Here **Females** performed better than males in **HumanResources**, **Development**, **Research and Development** departments. **Males** performed better than **Females** in **Sales**,**Data Science** and **Finance** departments.

In [19]:

```
plt.figure(figsize=(15,5))
sns.pointplot(performance['EmpDepartment'],performance['PerformanceRating'],performance['Ge
plt.xticks(rotation=30)
plt.xlabel("Name of Department",fontsize=15,color='black')
plt.ylabel(" Performance Rating of the Department ",fontsize=15,color='black')
plt.title(" Graph of Department v/s Performance Rating ",fontdict={'fontsize':20,'color':'B
```

Out[19]:

```
Text(0.5, 1.0, ' Graph of Department v/s Performance Rating ')
```



Analysis of each Department with respect to Performance Rating Index 2,3,4

In [20]:

```
dept=pd.get_dummies(performance['EmpDepartment'])
ratings=pd.DataFrame(performance['PerformanceRating'])
```

In [21]:

```
rating_index=pd.concat([dept,ratings],axis=1)
rating_index.head()
```

Out[21]:

	Data Science	Development	Finance	Human Resources	Research & Development	Sales	PerformanceRating
0	0	0	0	0	0	1	3
1	0	0	0	0	0	1	3
2	0	0	0	0	0	1	4
3	0	0	0	1	0	0	3
4	0	0	0	0	0	1	3

In [22]:

```
rating_index.groupby(by=['PerformanceRating'])['Sales'].mean()
```

Out[22]:

```
PerformanceRating
2    0.448454
3    0.287185
4    0.265152
Name: Sales, dtype: float64
```

In [23]:

```
rating_index.groupby(by=['PerformanceRating'])['Development'].mean()
```

Out[23]:

```
PerformanceRating
2    0.067010
3    0.347826
4    0.333333
Name: Development, dtype: float64
```

In [24]:

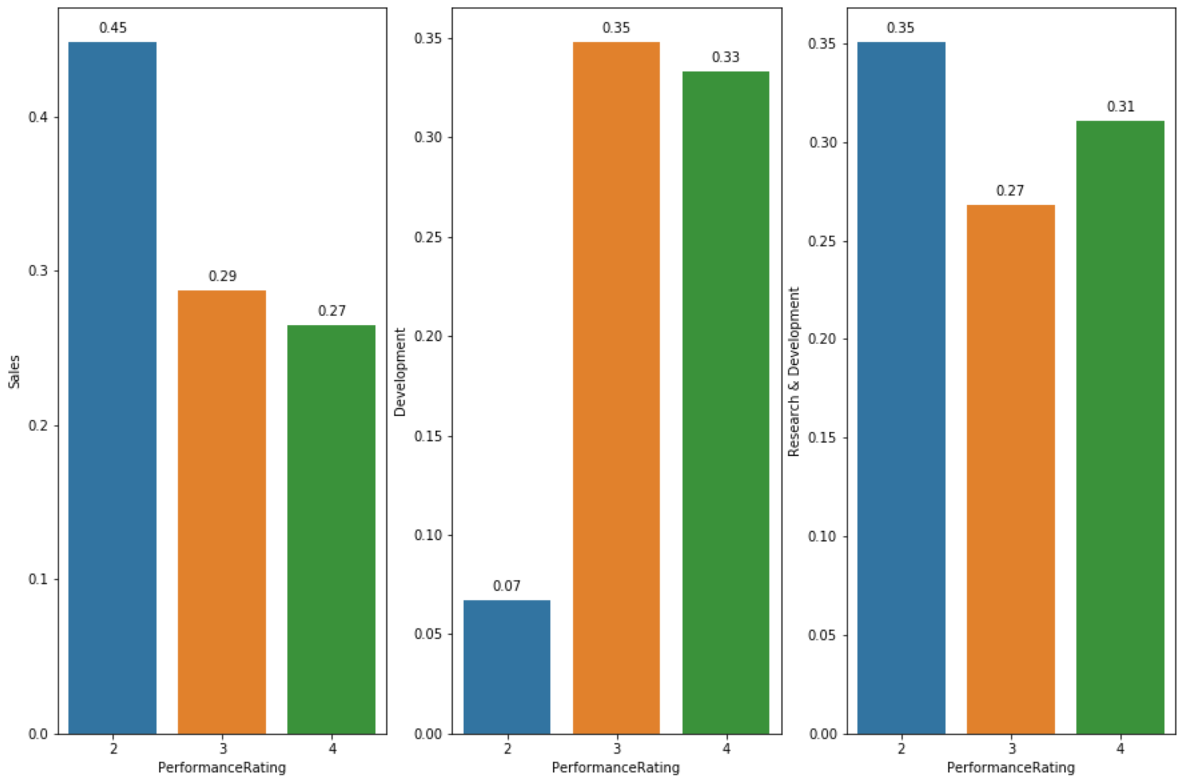
```
rating_index.groupby(by=['PerformanceRating'])['Research & Development'].mean()
```

Out[24]:

```
PerformanceRating
2    0.350515
3    0.267735
4    0.310606
Name: Research & Development, dtype: float64
```

In [25]:

```
plt.figure(figsize=(15,10))
plt.subplot(1,3,1)
splot=sns.barplot(rating_index['PerformanceRating'],rating_index['Sales'],ci=None)
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he

plt.subplot(1,3,2)
splot=sns.barplot(rating_index['PerformanceRating'],rating_index['Development'],ci=None)

for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he


plt.subplot(1,3,3)
splot=sns.barplot(rating_index['PerformanceRating'],rating_index['Research & Development'],

for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he

plt.show()
```



Here PerformanceRating of 2 is highest in **Sales, Research and Development** departments. 3 is highest in **Development** department.

In [26]:

```
rating_index.groupby(by=['PerformanceRating'])['Human Resources'].mean()
```

Out[26]:

```
PerformanceRating
2    0.051546
3    0.043478
4    0.045455
Name: Human Resources, dtype: float64
```

In [27]:

```
rating_index.groupby(by=['PerformanceRating'])['Finance'].mean()
```

Out[27]:

```
PerformanceRating
2    0.077320
3    0.034325
4    0.030303
Name: Finance, dtype: float64
```

In [28]:

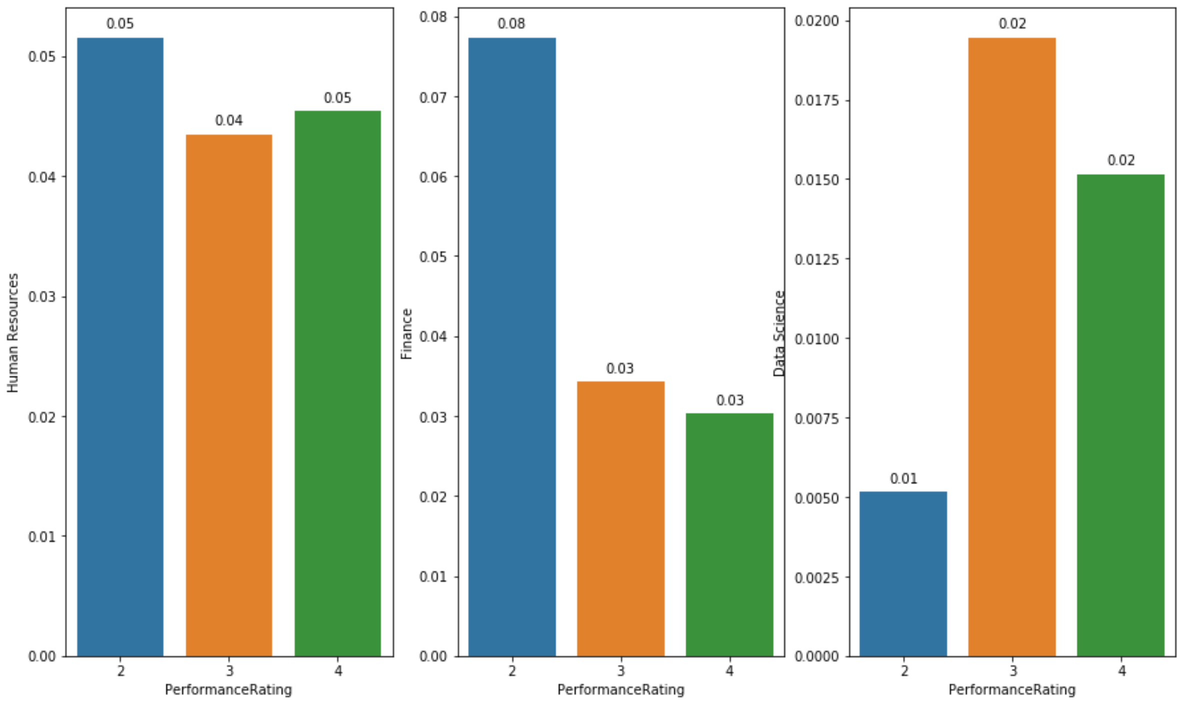
```
rating_index.groupby(by=['PerformanceRating'])['Data Science'].mean()
```

Out[28]:

```
PerformanceRating
2    0.005155
3    0.019451
4    0.015152
Name: Data Science, dtype: float64
```

In [29]:

```
plt.figure(figsize=(15,9))
plt.subplot(1,3,1)
splot=sns.barplot(rating_index['PerformanceRating'],rating_index['Human Resources'],ci=None
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he

plt.subplot(1,3,2)
splot=sns.barplot(rating_index['PerformanceRating'],rating_index['Finance'],ci=None)
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he

plt.subplot(1,3,3)
splot=sns.barplot(rating_index['PerformanceRating'],rating_index['Data Science'],ci=None)
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he

plt.show()
```



Here PerformanceRating 2 is highest in **Human Resources, Finance** departments. PerformanceRating 3 is highest in **Data Science** department. PerformanceRating 4 is 2nd highest in **Development, Research and Development, Human Resources and Data Science** departments.

--

2) Top 3 important factors effecting the employee performance

In [30]:

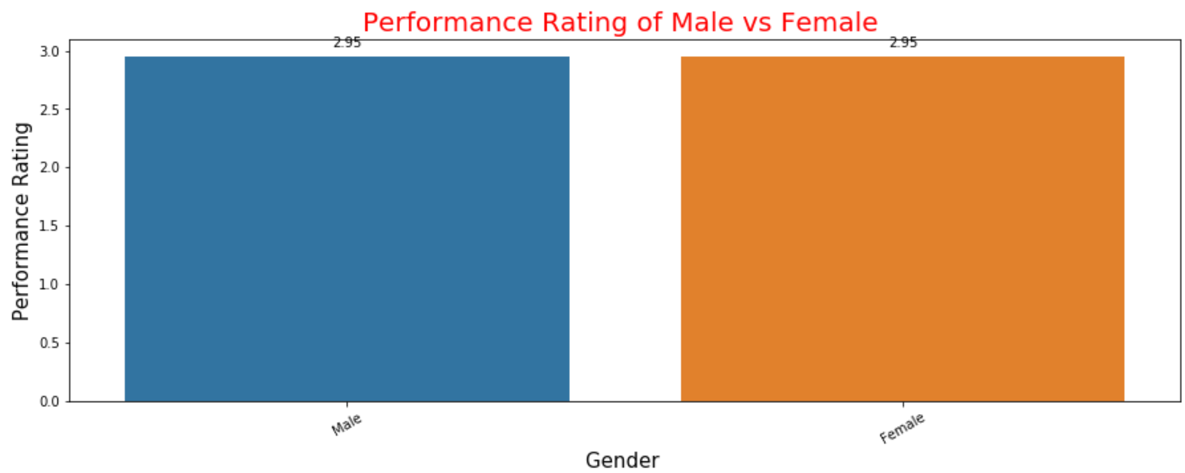
```
performance.groupby(by=['Gender'])['PerformanceRating'].mean()
```

Out[30]:

```
Gender
Female    2.949474
Male      2.947586
Name: PerformanceRating, dtype: float64
```

In [31]:

```
plt.figure(figsize=(15,5))
splot=sns.barplot(performance['Gender'],performance['PerformanceRating'],ci=None)
plt.xticks(rotation=30)
plt.xlabel("Gender ",fontsize=15,color='black')
plt.ylabel(" Performance Rating ",fontsize=15,color='black')
plt.title("Performance Rating of Male vs Female ",fontdict={'fontsize':20,'color':'Red'})
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he
```



PerformanceRating of Female is better than Males.

In [32]:

```
performance1=pd.DataFrame(performance,columns=['Age','Gender','BusinessTravelFrequency','Di
                                                'EmpJobLevel','EmpJobSatisfaction','OverTime','EmpLas
                                                        'EmpRelationshipSatisfaction','TrainingTimesL
                                                         'ExperienceYearsAtThisCompany','ExperienceYe
                                                        'YearsWithCurrManager','Attrition','Performan
performance1.head()
```

Out[32]:

	Age	Gender	BusinessTravelFrequency	DistanceFromHome	EmpEducationLevel	EmpEnviror
0	32	Male	Travel_Rarely	10	3	
1	47	Male	Travel_Rarely	14	4	
2	40	Male	Travel_Frequently	5	4	
3	41	Male	Travel_Rarely	10	4	
4	60	Male	Travel_Rarely	16	4	

5 rows × 22 columns

In [33]:

```
performance1.tail()
```

Out[33]:

	Age	Gender	BusinessTravelFrequency	DistanceFromHome	EmpEducationLevel	EmpEnv
1195	27	Female	Travel_Frequently	3	1	
1196	37	Male	Travel_Rarely	10	2	
1197	50	Male	Travel_Rarely	28	1	
1198	34	Female	Travel_Rarely	9	3	
1199	24	Female	Travel_Rarely	3	2	

5 rows × 22 columns

In [34]:

```
performance1.isna().sum().to_frame()
```

Out[34]:

	0
Age	0
Gender	0
BusinessTravelFrequency	0
DistanceFromHome	0
EmpEducationLevel	0
EmpEnvironmentSatisfaction	0
EmpHourlyRate	0
EmpJobInvolvement	0
EmpJobLevel	0
EmpJobSatisfaction	0
OverTime	0
EmpLastSalaryHikePercent	0
NumCompaniesWorked	0
EmpRelationshipSatisfaction	0
TrainingTimesLastYear	0
EmpWorkLifeBalance	0
ExperienceYearsAtThisCompany	0
ExperienceYearsInCurrentRole	0
YearsSinceLastPromotion	0
YearsWithCurrManager	0
Attrition	0
PerformanceRating	0

In [35]:

```
performance1.groupby('PerformanceRating').mean()
```

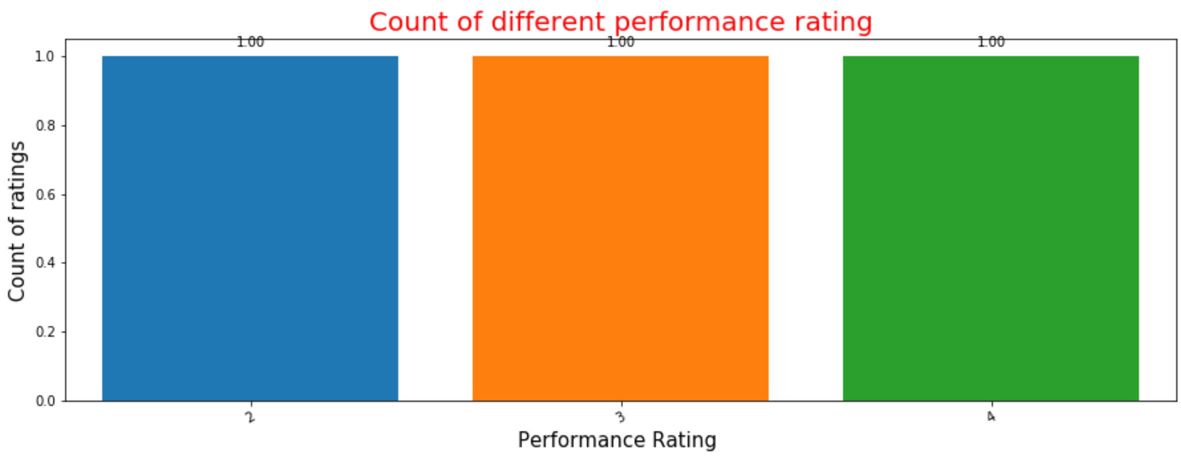
Out[35]:

	Age	DistanceFromHome	EmpEducationLevel	EmpEnvironmentSatisfac
PerformanceRating				
2	37.804124	9.835052	2.829897	1.582
3	36.784897	9.137300	2.905034	2.911
4	36.500000	8.371212	2.901515	3.083

Visualizations carried out for different factors affecting Employee Performance

In [36]:

```
plt.figure(figsize=(15,5))
splot=sns.countplot(performance1.groupby('PerformanceRating').count().index[:20],saturation
plt.xticks(rotation=30)
plt.xlabel("Performance Rating ",fontsize=15,color='black')
plt.ylabel(" Count of ratings ",fontsize=15,color='black')
plt.title("Count of different performance rating",fontdict={'fontsize':20,'color':'Red'})
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he
```



2.1) Performance Rating based on BusinessTravelFrequency

In [37]:

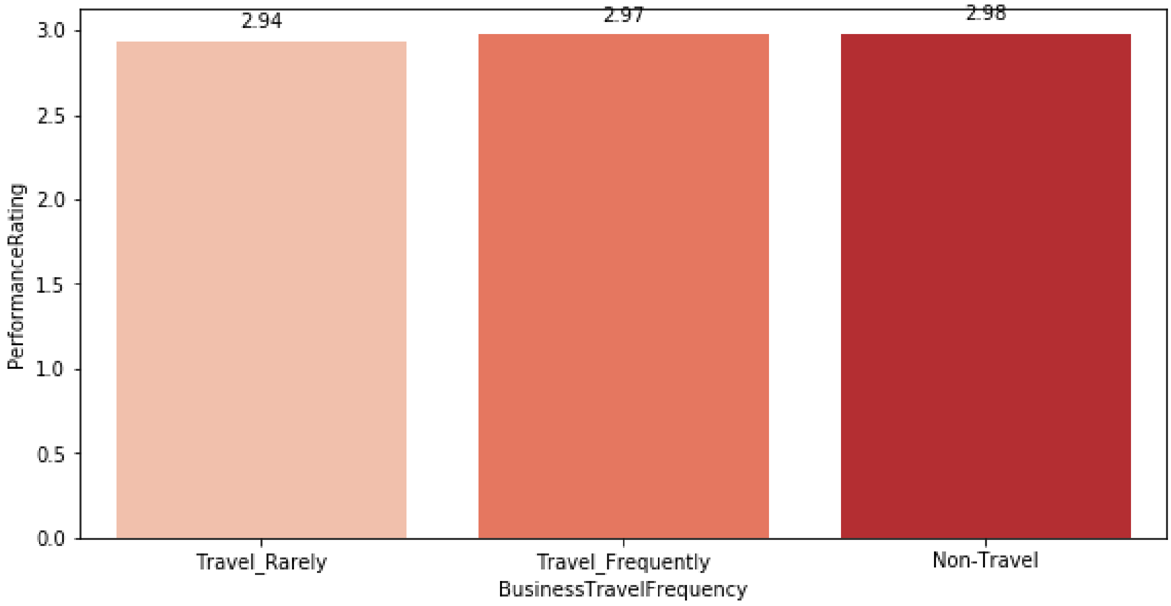
```
performance1.groupby(by=['BusinessTravelFrequency'])['PerformanceRating'].mean()
```

Out[37]:

```
BusinessTravelFrequency
Non-Travel           2.977273
Travel_Frequently    2.972973
Travel_Rarely        2.937352
Name: PerformanceRating, dtype: float64
```

In [38]:

```
plt.figure(figsize=(10,5))
splot=sns.barplot(performance1['BusinessTravelFrequency'],performance1['PerformanceRating']
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he
```



Here in terms of BusinessTravelFrequency , **'Non-Travel'** employees performed better.

2.2) Performance Rating based on DistanceFromHome

In [39]:

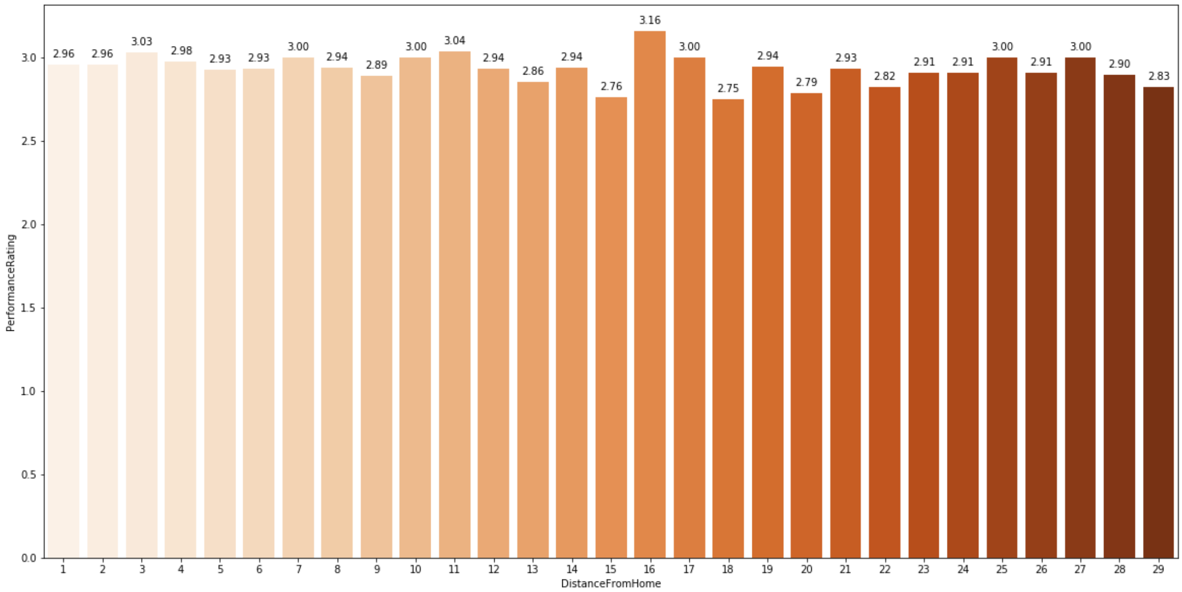
```
performance1.groupby(by=['DistanceFromHome'])['PerformanceRating'].mean()
```

Out[39]:

```
DistanceFromHome
1     2.958824
2     2.956522
3     3.029851
4     2.980392
5     2.925926
6     2.934783
7     3.000000
8     2.942029
9     2.893939
10    3.000000
11    3.040000
12    2.937500
13    2.857143
14    2.941176
15    2.764706
16    3.160000
17    3.000000
18    2.750000
19    2.944444
20    2.789474
21    2.933333
22    2.823529
23    2.909091
24    2.913043
25    3.000000
26    2.909091
27    3.000000
28    2.900000
29    2.826087
Name: PerformanceRating, dtype: float64
```

In [40]:

```
plt.figure(figsize=(20,10))
splot=sns.barplot(performance1['DistanceFromHome'], performance1['PerformanceRating'],palet
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he
```



Here employees staying 16km away from home performed better.

2.3) Performance Rating based on EmpEducationLevel

In [41]:

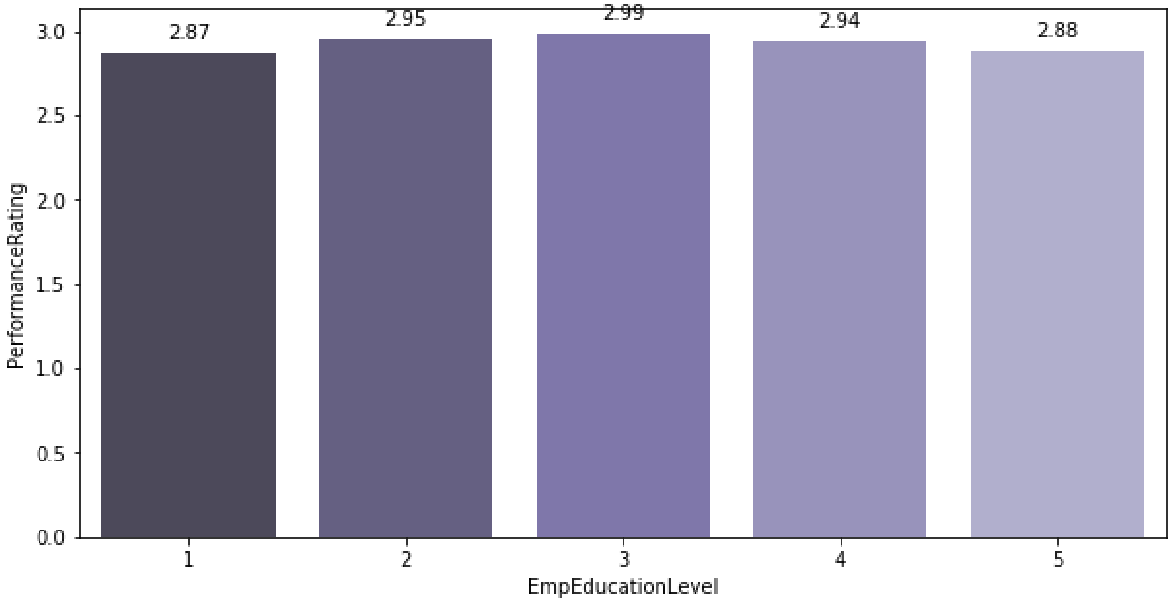
```
performance1.groupby(by=['EmpEducationLevel'])['PerformanceRating'].mean()
```

Out[41]:

```
EmpEducationLevel
1    2.871622
2    2.949791
3    2.986637
4    2.937888
5    2.880952
Name: PerformanceRating, dtype: float64
```

In [42]:

```
plt.figure(figsize=(10,5))
splot=sns.barplot(performance1['EmpEducationLevel'], performance1['PerformanceRating'],pale
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he
```



Here Employees having education level 3 is the highest.

2.4) Performance Rating based on EmpEnvironmentSatisfaction

In [43]:

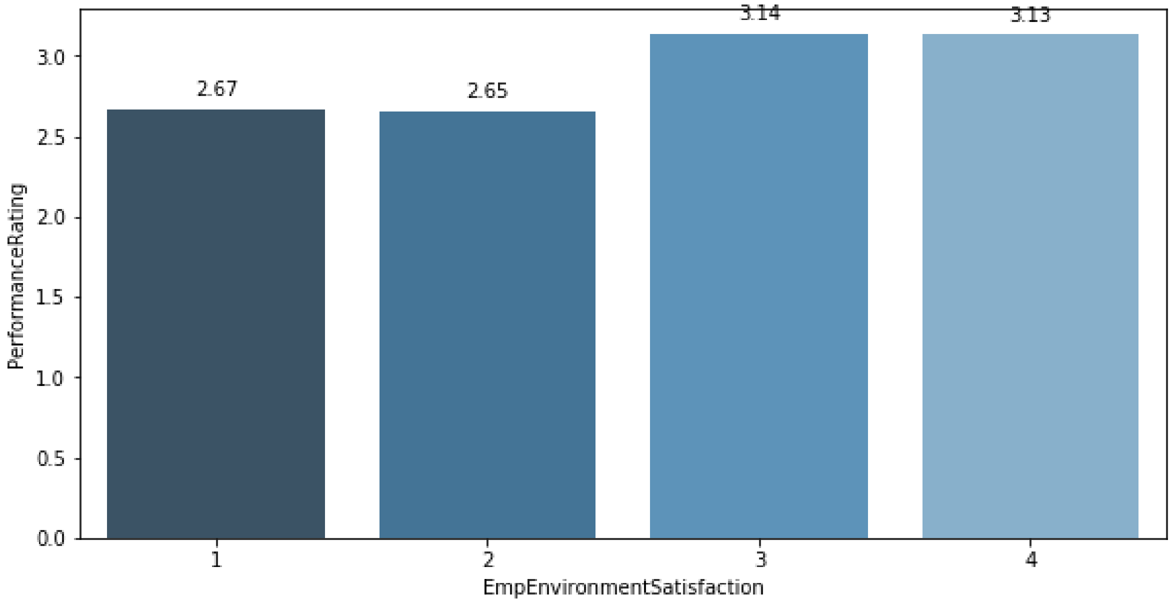
```
performance1.groupby(by=['EmpEnvironmentSatisfaction'])['PerformanceRating'].mean()
```

Out[43]:

```
EmpEnvironmentSatisfaction
1    2.665217
2    2.652893
3    3.138965
4    3.132964
Name: PerformanceRating, dtype: float64
```

In [44]:

```
plt.figure(figsize=(10,5))
splot=sns.barplot(performance1['EmpEnvironmentSatisfaction'], performance1['PerformanceRati
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he
```



Here Environment Satisfaction Level 3 performed better.

2.5) Performance Rating based on EmpJobInvolvement

In [45]:

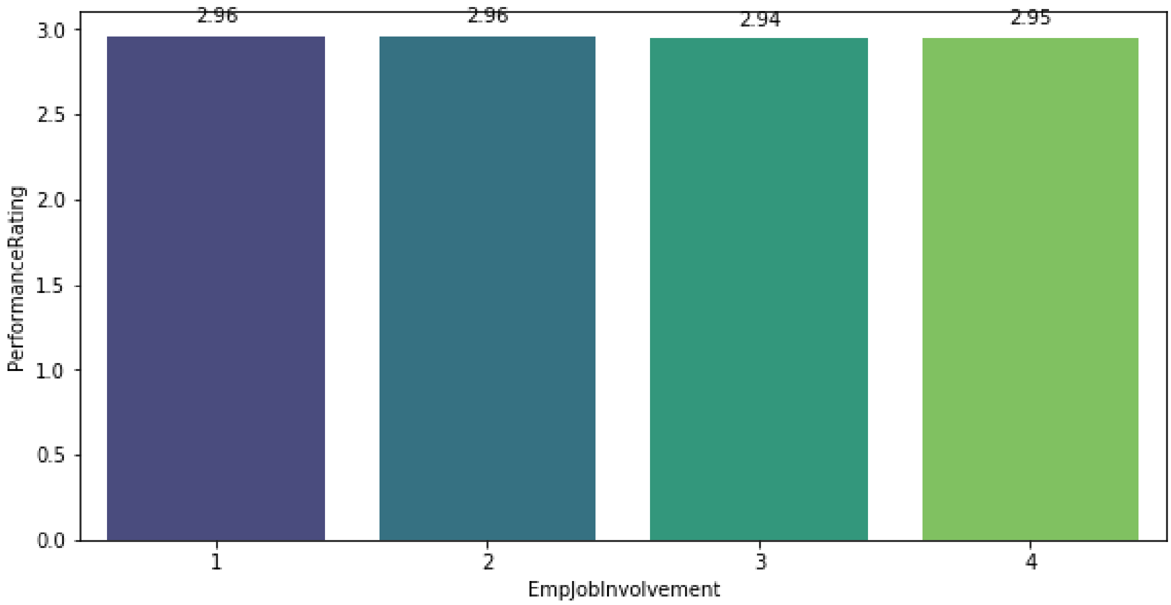
```
performance1.groupby(by=['EmpJobInvolvement'])['PerformanceRating'].mean()
```

Out[45]:

```
EmpJobInvolvement
1    2.957143
2    2.959184
3    2.943370
4    2.946429
Name: PerformanceRating, dtype: float64
```

In [46]:

```
plt.figure(figsize=(10,5))
splot=sns.barplot(performance1['EmpJobInvolvement'], performance1['PerformanceRating'],pale
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he
```



Here Employees Job Involvement level 2 performed better.

2.6) Performance Rating based on EmpJobLevel

In [47]:

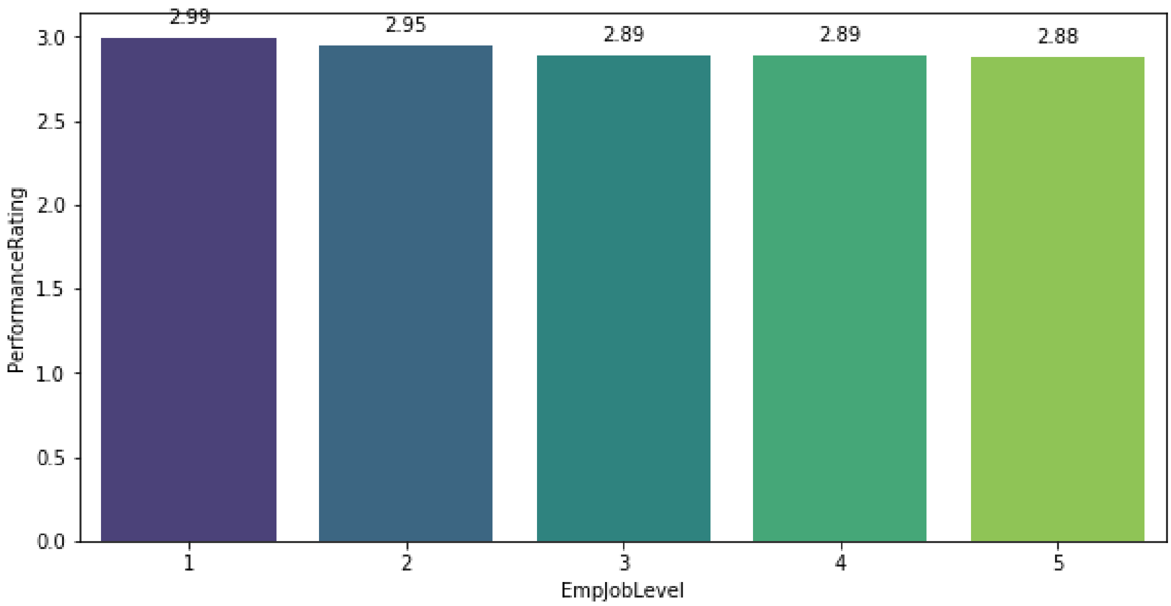
```
performance1.groupby(by=['EmpJobLevel'])['PerformanceRating'].mean()
```

Out[47]:

```
EmpJobLevel
1    2.993182
2    2.947846
3    2.890173
4    2.888889
5    2.875000
Name: PerformanceRating, dtype: float64
```

In [48]:

```
plt.figure(figsize=(10,5))
splot=sns.barplot(performance1['EmpJobLevel'], performance1['PerformanceRating'],palette ="
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he
```



Here Employee Job Level 1 performed better.

2.7) Performance Rating based on EmpJobSatisfaction

In [49]:

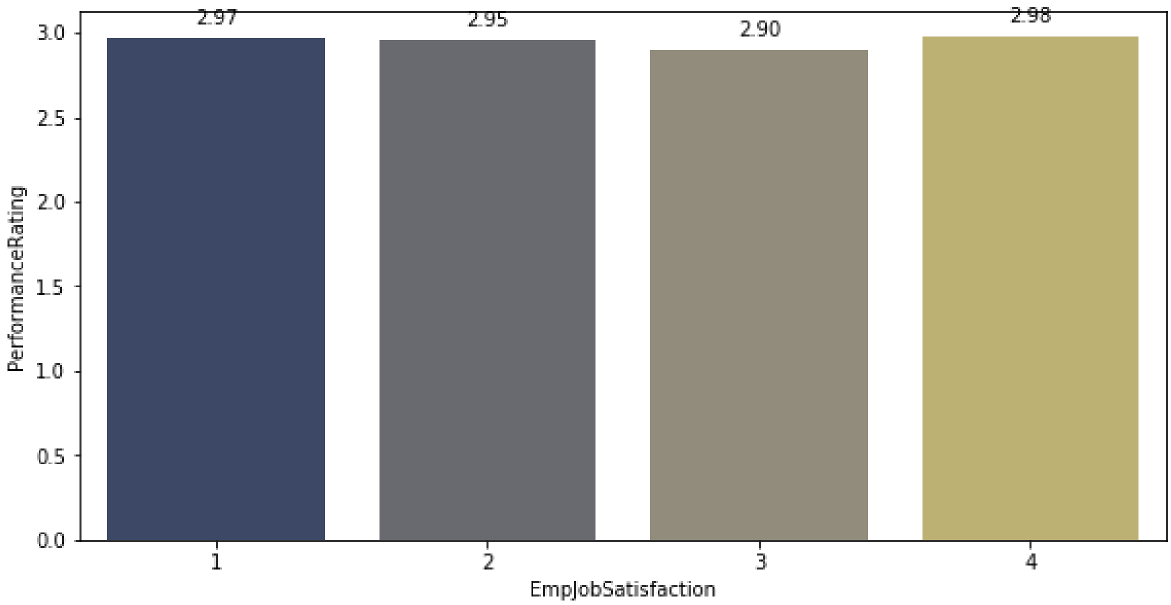
```
performance1.groupby(by=['EmpJobSatisfaction'])['PerformanceRating'].mean()
```

Out[49]:

```
EmpJobSatisfaction
1    2.969697
2    2.953586
3    2.898305
4    2.978836
Name: PerformanceRating, dtype: float64
```

In [50]:

```
plt.figure(figsize=(10,5))
splot=sns.barplot(performance1['EmpJobSatisfaction'], performance1['PerformanceRating'],pal
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he
```



Here Employee Job Satisfaction level 4 performed better.

2.8) Performance Rating based on OverTime

In [51]:

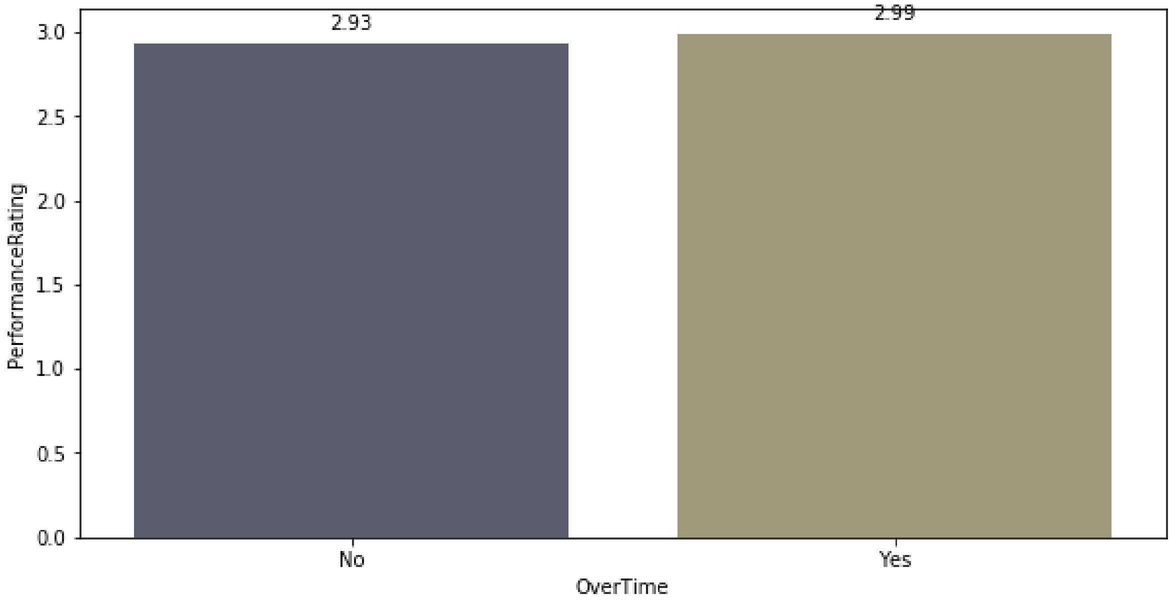
```
performance1.groupby(by=['OverTime'])['PerformanceRating'].mean()
```

Out[51]:

```
OverTime
No     2.931523
Yes    2.988669
Name: PerformanceRating, dtype: float64
```

In [52]:

```
plt.figure(figsize=(10,5))
splot=sns.barplot(performance1['OverTime'], performance1['PerformanceRating'],palette ="civ
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he
```



Here employees working overtime performed better.

2.9) Performance Rating based on EmpLastSalaryHikePercent

In [53]:

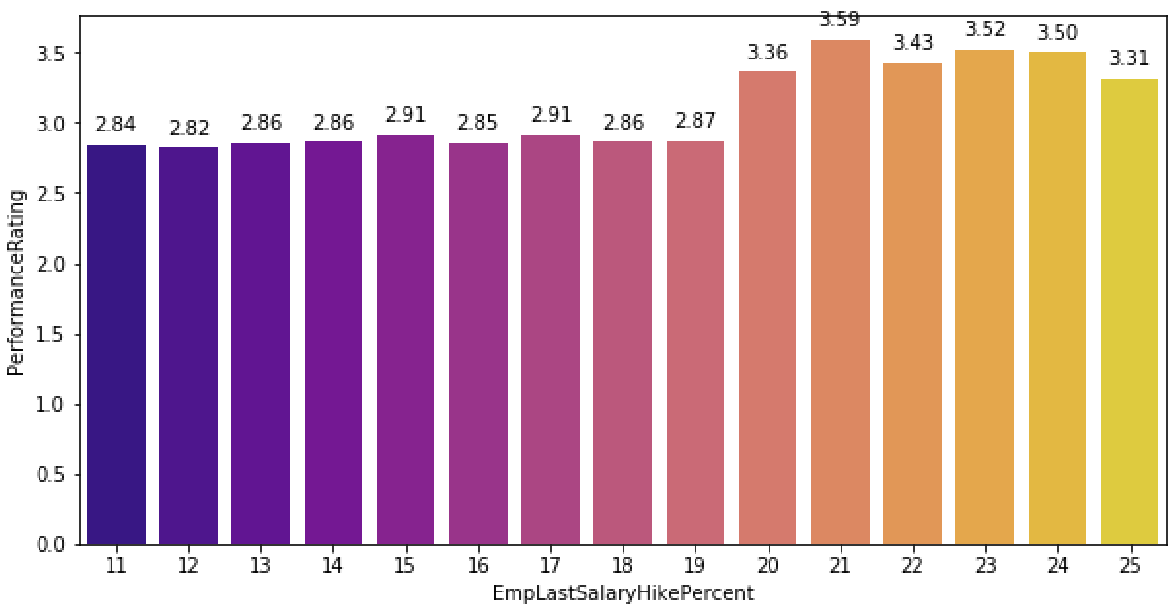
```
performance1.groupby(by=['EmpLastSalaryHikePercent'])['PerformanceRating'].mean()
```

Out[53]:

```
EmpLastSalaryHikePercent
11    2.840237
12    2.819355
13    2.857143
14    2.860465
15    2.914634
16    2.852941
17    2.910448
18    2.863014
19    2.873016
20    3.360000
21    3.588235
22    3.425532
23    3.523810
24    3.500000
25    3.307692
Name: PerformanceRating, dtype: float64
```

In [54]:

```
plt.figure(figsize=(10,5))
splot=sns.barplot(performance1['EmpLastSalaryHikePercent'], performance1['PerformanceRating
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he
```



Here employees who received 21% hike in the salary has better performance.

2.10) Performance Rating based on NumCompaniesWorked

In [55]:

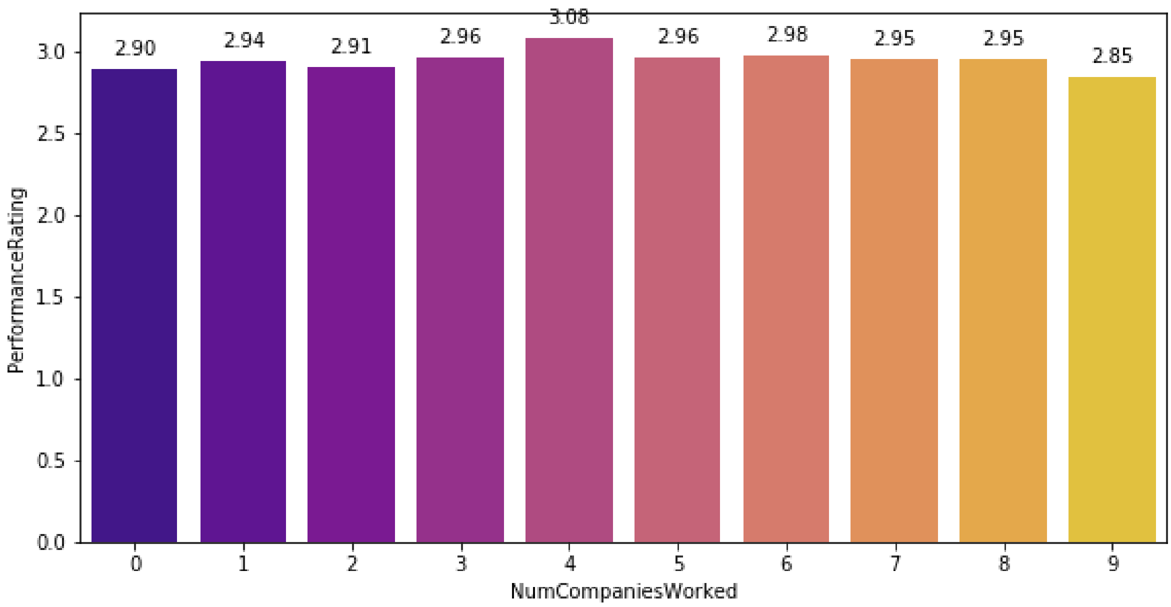
```
performance1.groupby(by=['NumCompaniesWorked'])['PerformanceRating'].mean()
```

Out[55]:

```
NumCompaniesWorked
0    2.897436
1    2.942263
2    2.910569
3    2.962406
4    3.084112
5    2.962264
6    2.982143
7    2.950000
8    2.950000
9    2.846154
Name: PerformanceRating, dtype: float64
```

In [56]:

```
plt.figure(figsize=(10,5))
splot=sns.barplot(performance1['NumCompaniesWorked'], performance1['PerformanceRating'],pal
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he
```



Here employees who worked in 4 different companies performed better on the job.

2.11) Performance Rating based on EmpRelationshipSatisfaction

In [57]:

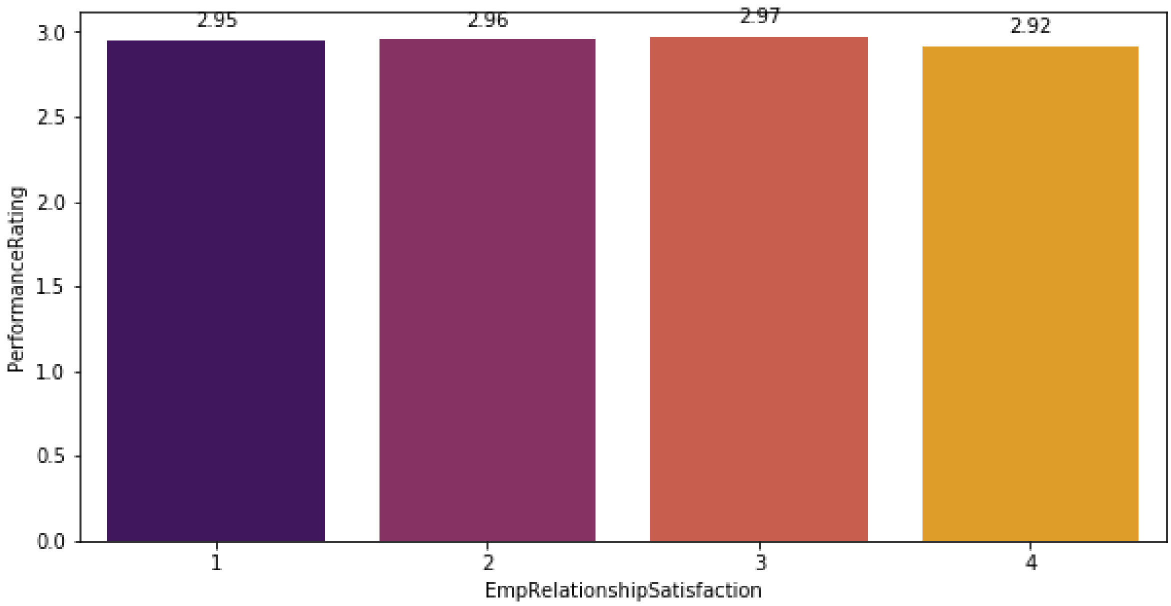
```
performance1.groupby(by=['EmpRelationshipSatisfaction'])['PerformanceRating'].mean()
```

Out[57]:

```
EmpRelationshipSatisfaction
1    2.949772
2    2.955466
3    2.970976
4    2.918310
Name: PerformanceRating, dtype: float64
```

In [58]:

```
plt.figure(figsize=(10,5))
splot=sns.barplot(performance1['EmpRelationshipSatisfaction'], performance1['PerformanceRat
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he
```



Here employee having Relationship Satisfaction level 3 has performed better.

2.12) Performance Rating based on TrainingTimesLastYear

In [59]:

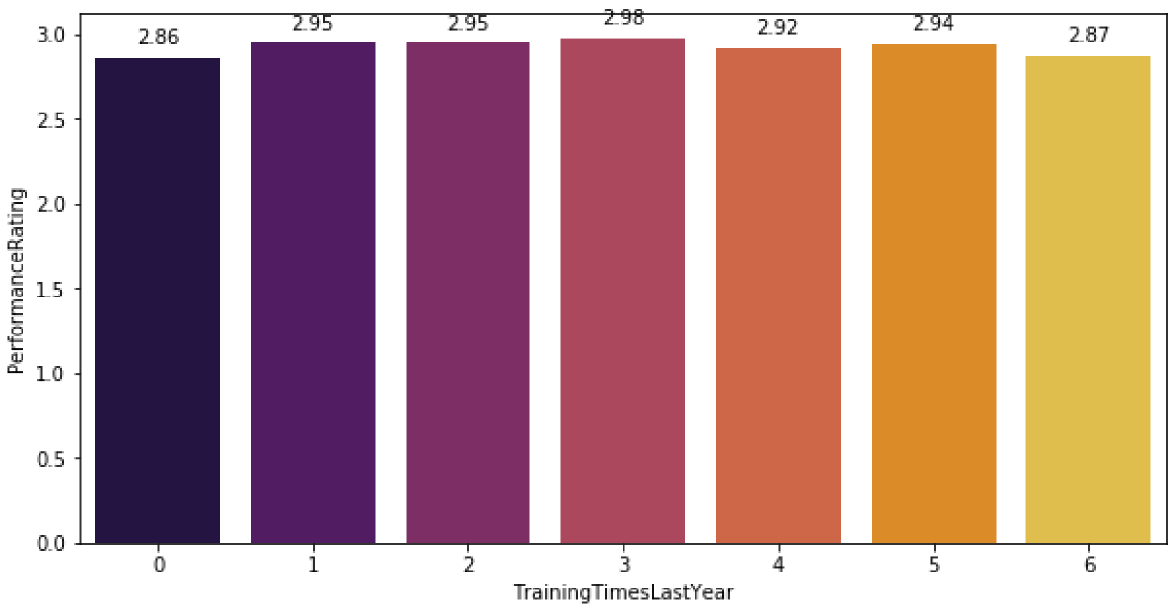
```
performance1.groupby(by=['TrainingTimesLastYear'])['PerformanceRating'].mean()
```

Out[59]:

```
TrainingTimesLastYear
0    2.863636
1    2.946429
2    2.948315
3    2.975787
4    2.918367
5    2.938776
6    2.869565
Name: PerformanceRating, dtype: float64
```

In [60]:

```
plt.figure(figsize=(10,5))
splot=sns.barplot(performance1['TrainingTimesLastYear'], performance1['PerformanceRating'],
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he
```



Here employees who have been trained 3 times last year have performed better.

2.13) Performance Rating based on EmpWorkLifeBalance

In [61]:

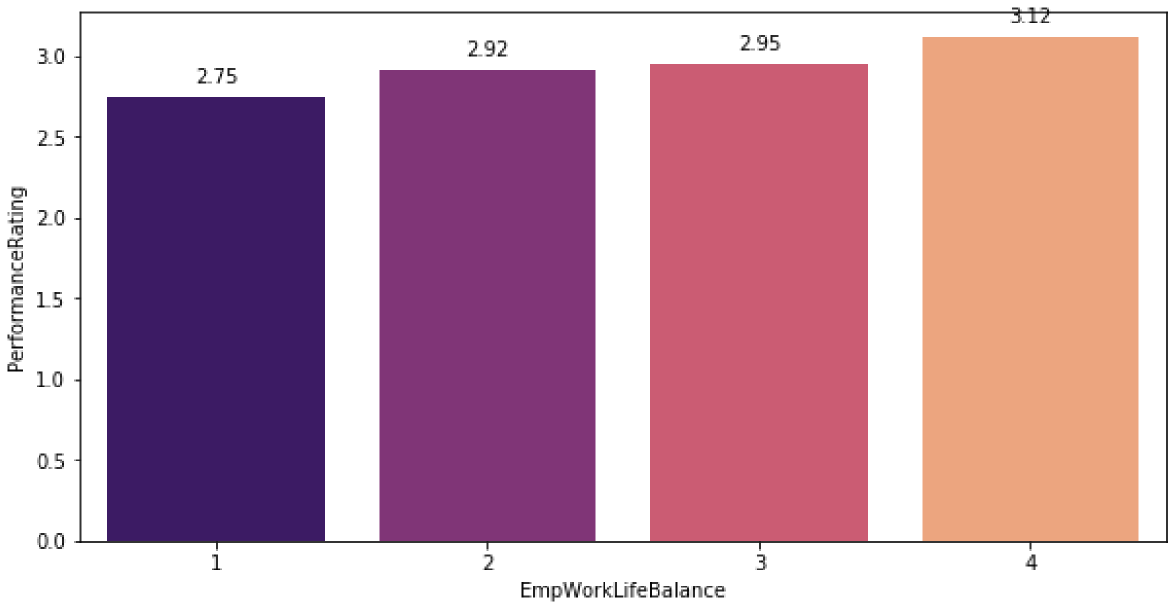
```
performance1.groupby(by=['EmpWorkLifeBalance'])['PerformanceRating'].mean()
```

Out[61]:

```
EmpWorkLifeBalance
1    2.750000
2    2.918367
3    2.950481
4    3.121739
Name: PerformanceRating, dtype: float64
```

In [62]:

```
plt.figure(figsize=(10,5))
splot=sns.barplot(performance1['EmpWorkLifeBalance'], performance1['PerformanceRating'],pal
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he
```



Here employees having Worklife Balance level 4 has highest performance rating.

2.14) Performance Rating based on ExperienceYearsAtThisCompany

In [63]:

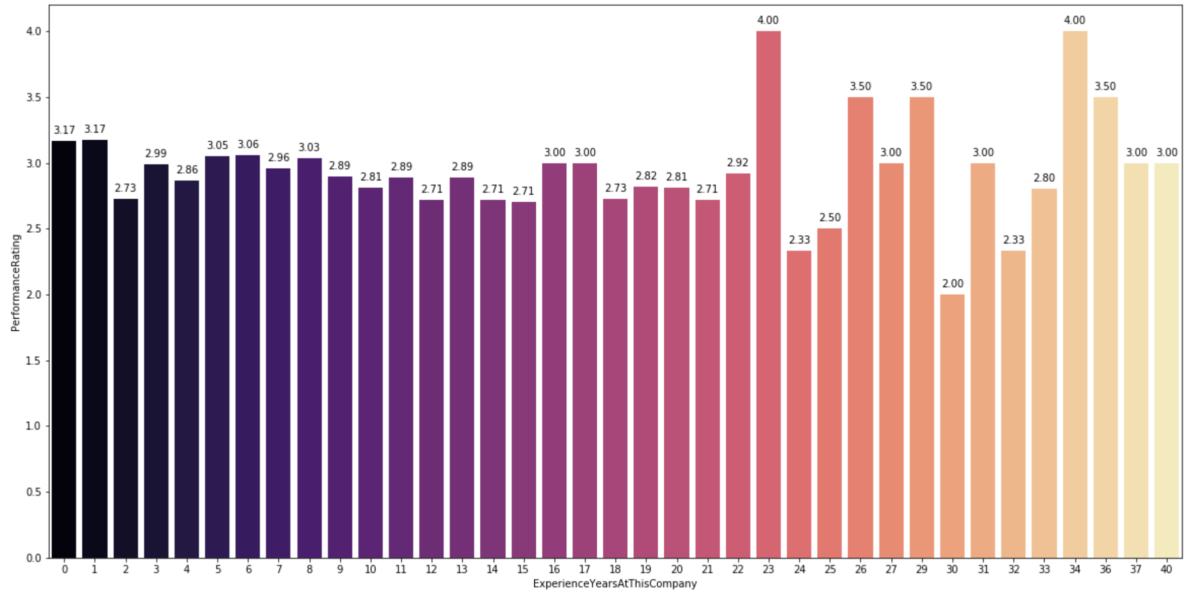
```
performance1.groupby(by=['ExperienceYearsAtThisCompany'])['PerformanceRating'].mean()
```

Out[63]:

```
ExperienceYearsAtThisCompany
0     3.166667
1     3.173913
2     2.728972
3     2.990476
4     2.863636
5     3.052632
6     3.060606
7     2.958904
8     3.031746
9     2.893939
10    2.810000
11    2.888889
12    2.714286
13    2.888889
14    2.714286
15    2.705882
16    3.000000
17    3.000000
18    2.727273
19    2.818182
20    2.809524
21    2.714286
22    2.916667
23    4.000000
24    2.333333
25    2.500000
26    3.500000
27    3.000000
29    3.500000
30    2.000000
31    3.000000
32    2.333333
33    2.800000
34    4.000000
36    3.500000
37    3.000000
40    3.000000
Name: PerformanceRating, dtype: float64
```

In [64]:

```
plt.figure(figsize=(20,10))
splot=sns.barplot(performance1['ExperienceYearsAtThisCompany'], performance1['PerformanceRa
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he
```



Here employees who have 23 and 34 years of experience in this company performed better.

2.15) Performance Rating based on ExperienceYearsInCurrentRole

In [65]:

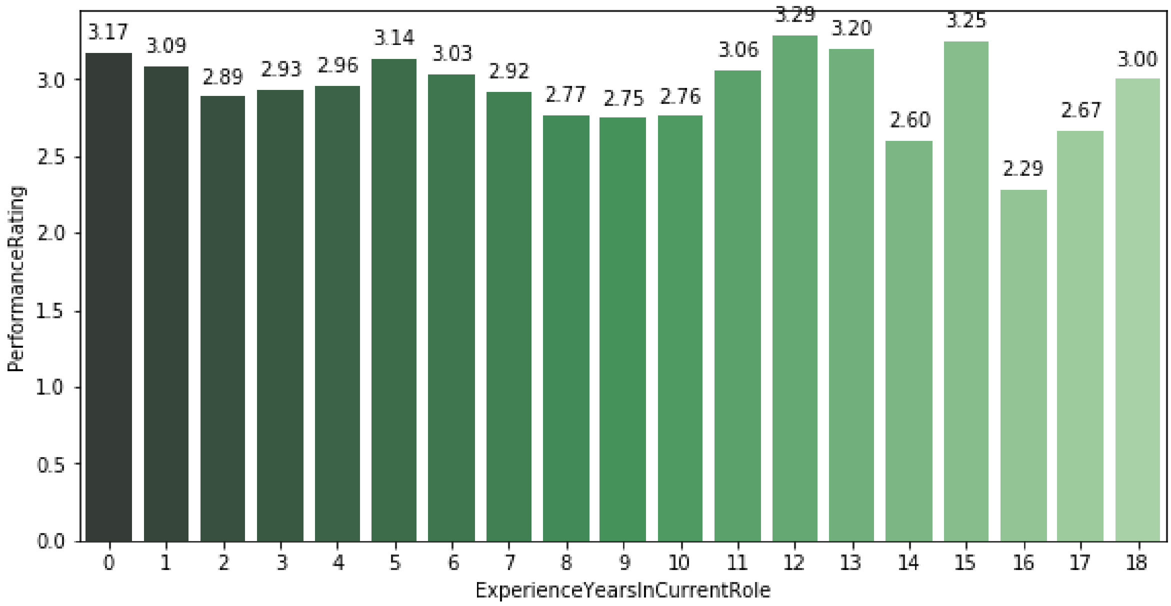
```
performance1.groupby(by=['ExperienceYearsInCurrentRole'])['PerformanceRating'].mean()
```

Out[65]:

```
ExperienceYearsInCurrentRole
0     3.168421
1     3.086957
2     2.887789
3     2.934579
4     2.956522
5     3.137931
6     3.033333
7     2.920455
8     2.769231
9     2.746032
10    2.760000
11    3.055556
12    3.285714
13    3.200000
14    2.600000
15    3.250000
16    2.285714
17    2.666667
18    3.000000
Name: PerformanceRating, dtype: float64
```

In [66]:

```
plt.figure(figsize=(10,5))
splot=sns.barplot(performance1['ExperienceYearsInCurrentRole'], performance1['PerformanceRa
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he
```



Here employees having 12 years of experience in the current role performed better.

2.16) Performance Rating based on

YearsSinceLastPromotion

In [67]:

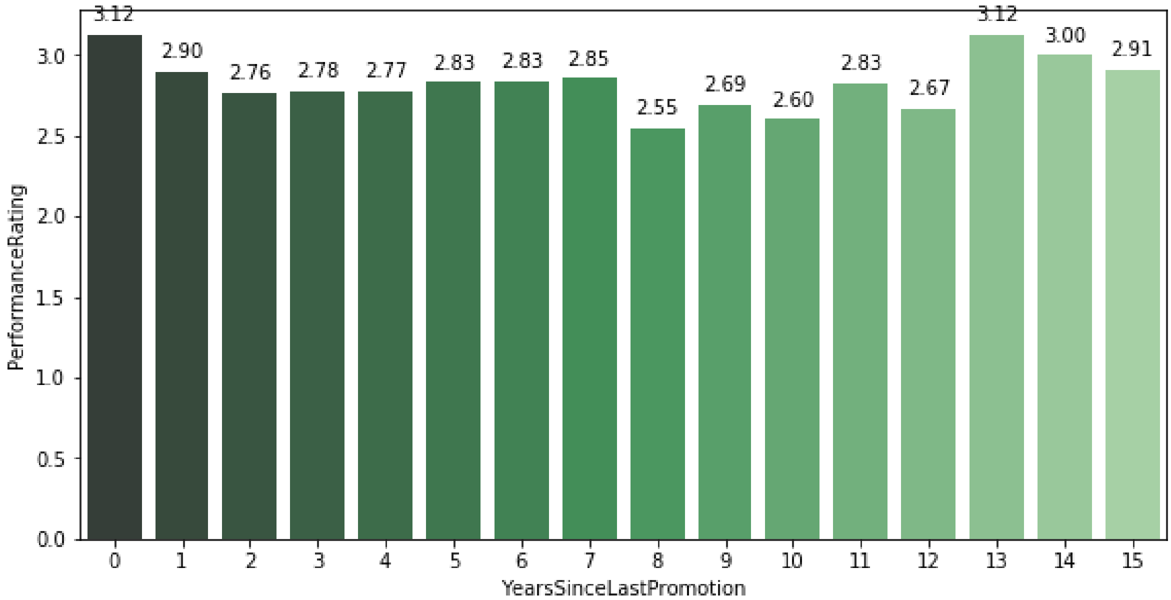
```
performance1.groupby(by=['YearsSinceLastPromotion'])['PerformanceRating'].mean()
```

Out[67]:

```
YearsSinceLastPromotion
0     3.123667
1     2.898990
2     2.763780
3     2.777778
4     2.773585
5     2.828571
6     2.833333
7     2.854839
8     2.545455
9     2.687500
10    2.600000
11    2.826087
12    2.666667
13    3.125000
14    3.000000
15    2.909091
Name: PerformanceRating, dtype: float64
```

In [68]:

```
plt.figure(figsize=(10,5))
splot=sns.barplot(performance1['YearsSinceLastPromotion'], performance1['PerformanceRating'
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he
```



Here employees who haven't been promoted since 13 years after their last promotion have performed better.

2.17) Performance Rating based on YearsWithCurrManager

In [69]:

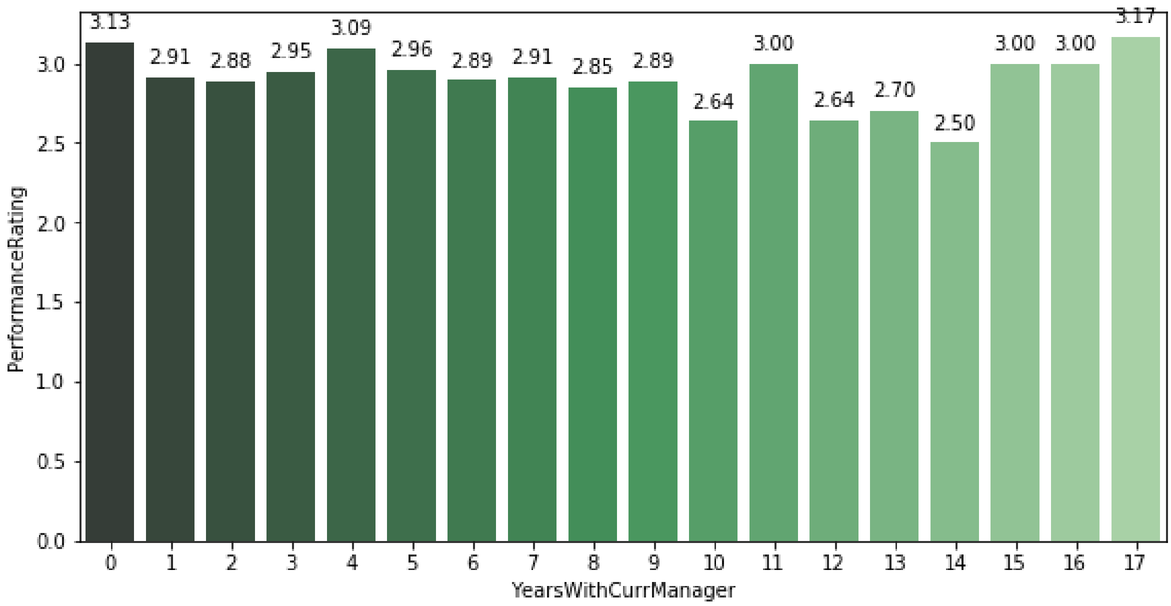
```
performance1.groupby(by=['YearsWithCurrManager'])['PerformanceRating'].mean()
```

Out[69]:

```
YearsWithCurrManager
0     3.134884
1     2.910448
2     2.882562
3     2.951456
4     3.094118
5     2.961538
6     2.892857
7     2.909091
8     2.850575
9     2.886792
10    2.636364
11    3.000000
12    2.642857
13    2.700000
14    2.500000
15    3.000000
16    3.000000
17    3.166667
Name: PerformanceRating, dtype: float64
```

In [70]:

```
plt.figure(figsize=(10,5))
splot=sns.barplot(performance1['YearsWithCurrManager'], performance1['PerformanceRating'],p
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he
```



Here employees who spent 17 years with the current manager has better performance rating.

2.18) Performance Rating based on EmpHourlyRate

In [71]:

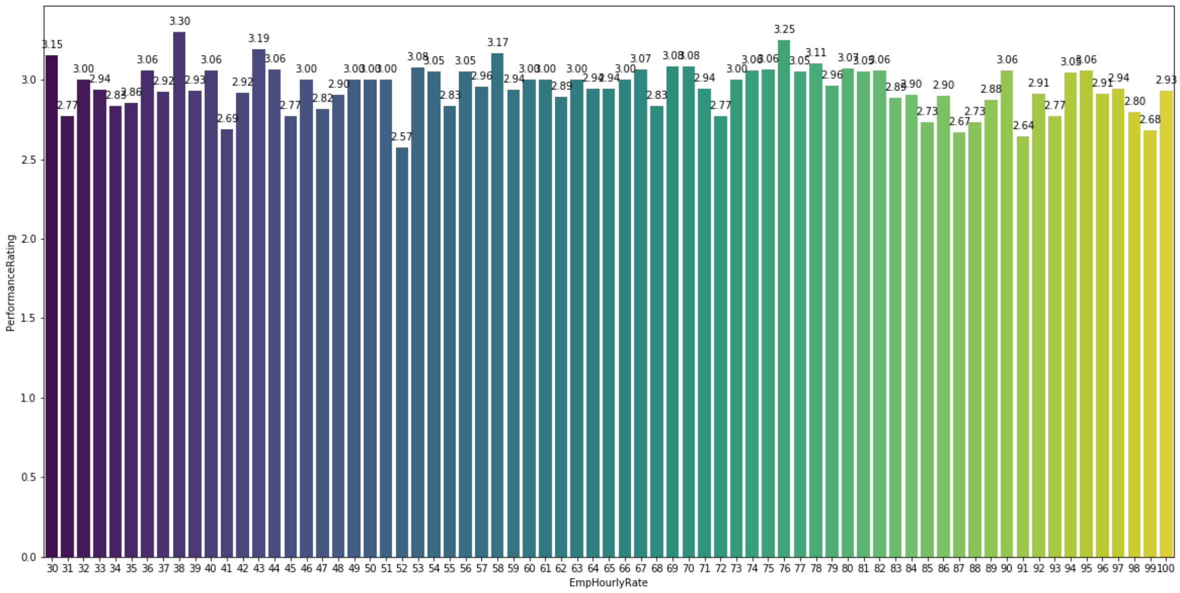
```
performance1.groupby(by=['EmpHourlyRate'])['PerformanceRating'].mean()
```

Out[71]:

```
EmpHourlyRate
30     3.153846
31     2.769231
32     3.000000
33     2.937500
34     2.833333
         ...
96     2.909091
97     2.944444
98     2.800000
99     2.684211
100    2.928571
Name: PerformanceRating, Length: 71, dtype: float64
```

In [72]:

```
plt.figure(figsize=(20,10))
splot=sns.barplot(performance1['EmpHourlyRate'], performance1['PerformanceRating'],palette
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he
```



Here employees working at the rate of 38 hours have performed better.

2.19) Performance Rating based on Attrition

In [73]:

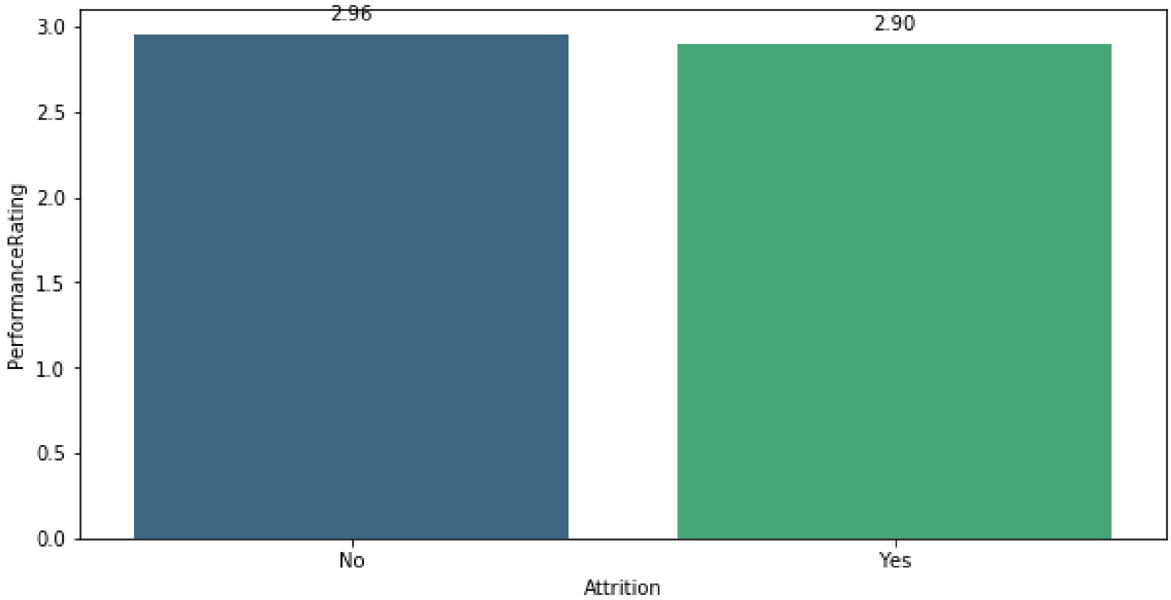
```
performance1.groupby(by=['Attrition'])['PerformanceRating'].mean()
```

Out[73]:

```
Attrition
No     2.956947
Yes    2.898876
Name: PerformanceRating, dtype: float64
```

In [74]:

```
plt.figure(figsize=(10,5))
splot=sns.barplot(performance1['Attrition'], performance1['PerformanceRating'],palette ="vi
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he
```



Here employees who haven't resigned from many jobs have performed better.

2.20) Performance Rating based on Gender

In [75]:

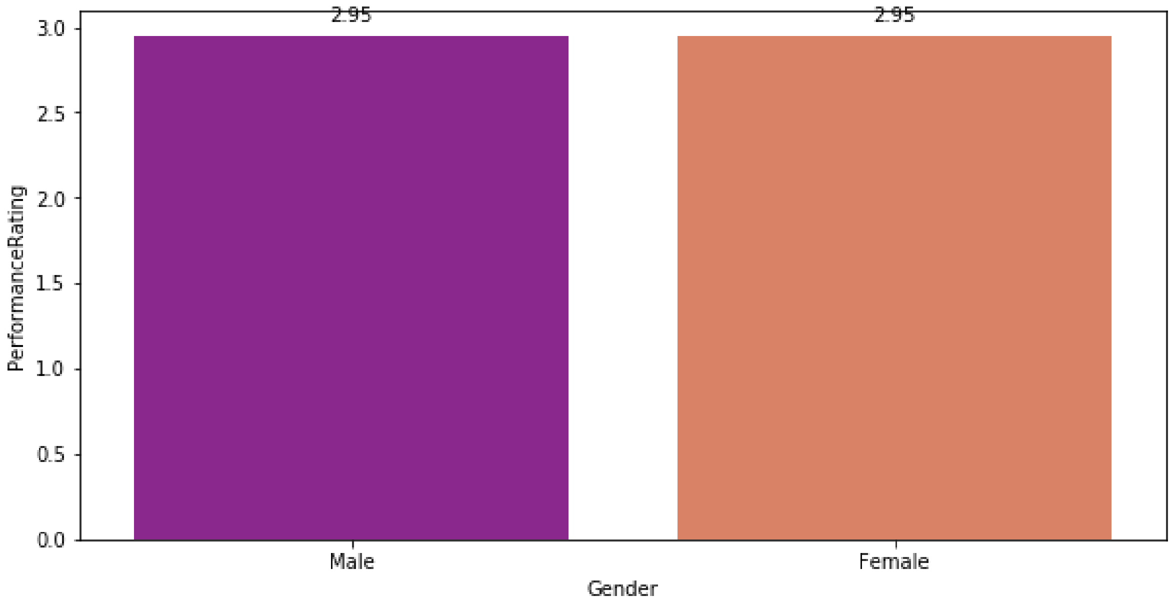
```
performance1.groupby(by=['Gender'])['PerformanceRating'].mean()
```

Out[75]:

```
Gender
Female    2.949474
Male      2.947586
Name: PerformanceRating, dtype: float64
```

In [76]:

```
plt.figure(figsize=(10,5))
splot=sns.barplot(performance1['Gender'], performance1['PerformanceRating'],palette ="plasm
for p in splot.patches:
    splot.annotate(format(p.get_height(), '.2f'), (p.get_x() + p.get_width() / 2., p.get_he
```



Females performed better than males.

Correlation Matrix

It is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables

In [77]:

```
corr=performance.corr()
corr
```

Out[77]:

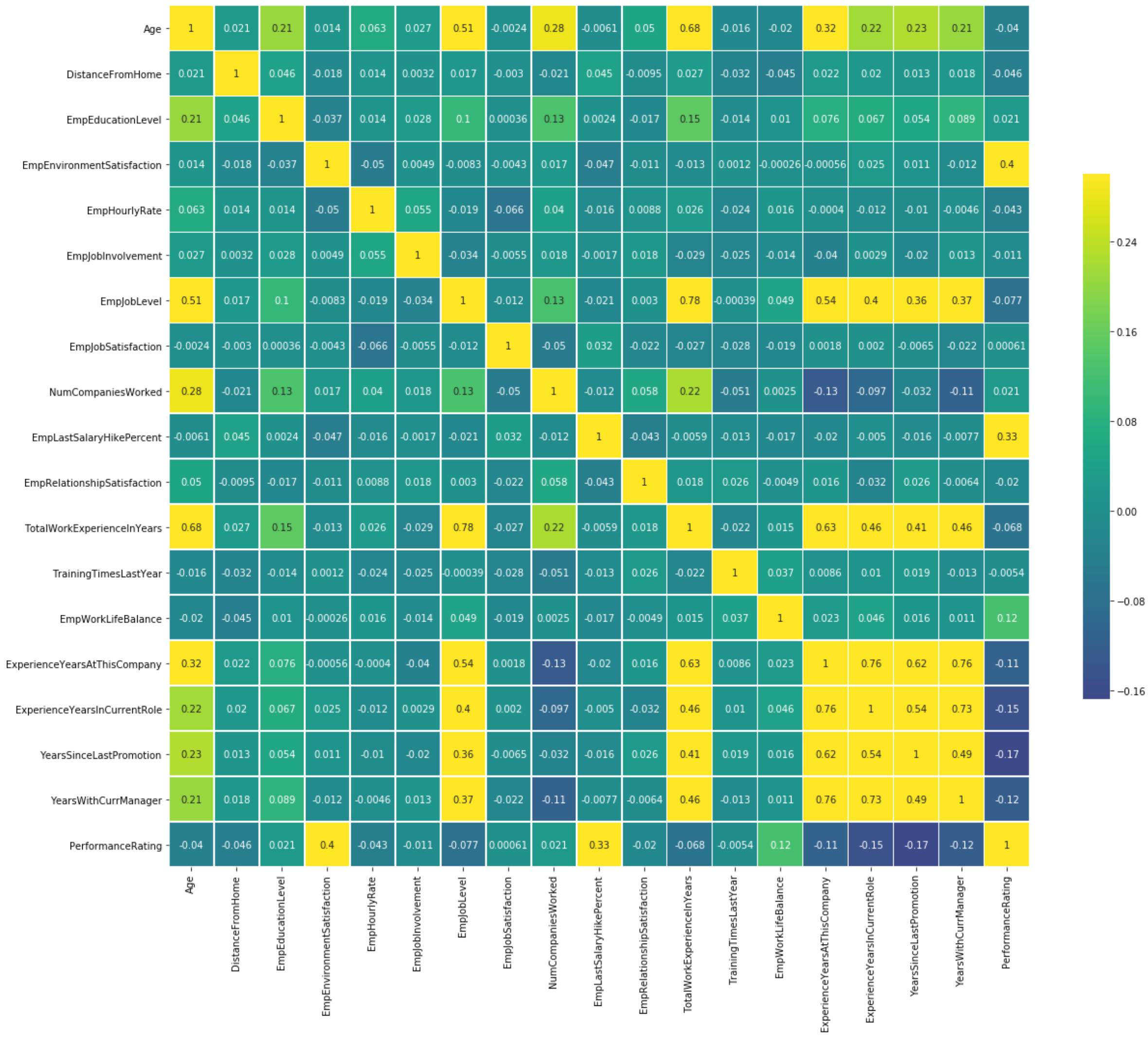
	Age	DistanceFromHome	EmpEducationLevel	EmpEnviron
Age	1.000000	0.020937	0.207313	
DistanceFromHome	0.020937	1.000000	0.045856	
EmpEducationLevel	0.207313	0.045856	1.000000	
EmpEnvironmentSatisfaction	0.013814	-0.017719	-0.037103	
EmpHourlyRate	0.062867	0.013730	0.014095	
EmpJobInvolvement	0.027216	0.003231	0.027544	
EmpJobLevel	0.509139	0.017270	0.100734	
EmpJobSatisfaction	-0.002436	-0.003036	0.000357	
NumCompaniesWorked	0.284408	-0.021411	0.128674	
EmpLastSalaryHikePercent	-0.006105	0.044974	0.002358	
EmpRelationshipSatisfaction	0.049749	-0.009509	-0.016690	
TotalWorkExperienceInYears	0.680886	0.027306	0.151062	
TrainingTimesLastYear	-0.016053	-0.032082	-0.013674	
EmpWorkLifeBalance	-0.019563	-0.044788	0.010276	
ExperienceYearsAtThisCompany	0.318852	0.021908	0.076332	
ExperienceYearsInCurrentRole	0.217163	0.019898	0.066672	
YearsSinceLastPromotion	0.228199	0.013246	0.054313	
YearsWithCurrManager	0.205098	0.017860	0.088988	
PerformanceRating	-0.040164	-0.046142	0.020529	

In [78]:

```
plt.figure(figsize=(20,20))
sns.heatmap(corr, cmap='viridis', vmax=.3, center=0,
            square=True, linewidths=.6, cbar_kws={"shrink": .5}, annot=True)
```

Out[78]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xb481948>
```



After the visualiation and correlation matrix, it clearly indicates that the top 3 factors which affect the employee performance are

1) Employee Environment Satisfation --> 39.5561%

2) Employee Last Salary Hike Percent --> 33.3722%

3) Years Since Last Promotion --> 16.7629%

--

In [79]:

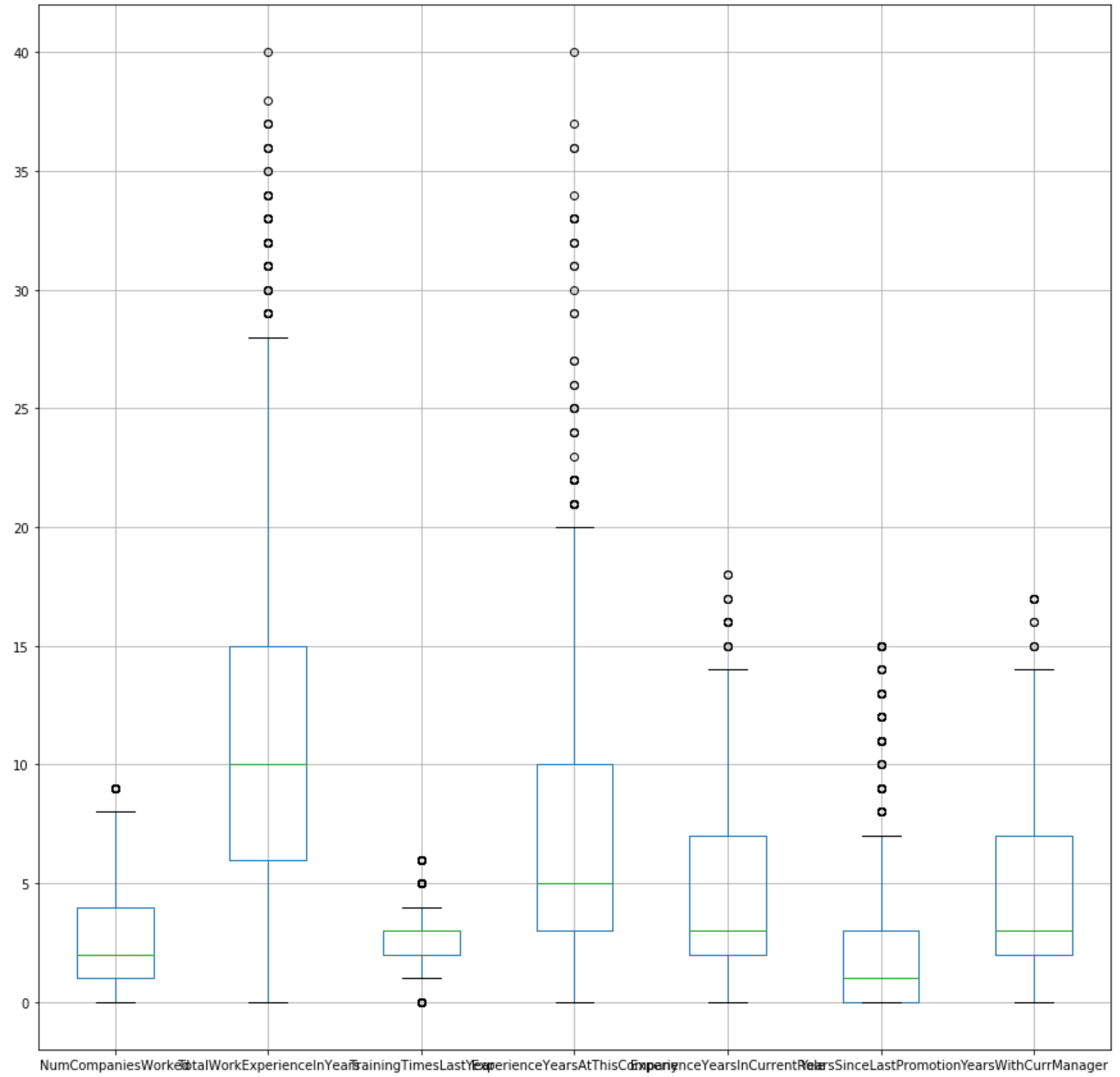
```
performance.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1200 entries, 0 to 1199
Data columns (total 28 columns):
 #   Column                        Non-Null Count  Dtype 
---  ------                        --------------  ----- 
 0   EmpNumber                     1200 non-null   object
 1   Age                           1200 non-null   int64 
 2   Gender                        1200 non-null   object
 3   EducationBackground           1200 non-null   object
 4   MaritalStatus                 1200 non-null   object
 5   EmpDepartment                 1200 non-null   object
 6   EmpJobRole                    1200 non-null   object
 7   BusinessTravelFrequency       1200 non-null   object
 8   DistanceFromHome              1200 non-null   int64 
 9   EmpEducationLevel             1200 non-null   int64 
 10  EmpEnvironmentSatisfaction    1200 non-null   int64 
 11  EmpHourlyRate                 1200 non-null   int64 
 12  EmpJobInvolvement             1200 non-null   int64 
 13  EmpJobLevel                   1200 non-null   int64 
 14  EmpJobSatisfaction            1200 non-null   int64 
 15  NumCompaniesWorked            1200 non-null   int64 
 16  OverTime                      1200 non-null   object
 17  EmpLastSalaryHikePercent      1200 non-null   int64 
 18  EmpRelationshipSatisfaction   1200 non-null   int64 
 19  TotalWorkExperienceInYears    1200 non-null   int64 
 20  TrainingTimesLastYear         1200 non-null   int64 
 21  EmpWorkLifeBalance            1200 non-null   int64 
 22  ExperienceYearsAtThisCompany  1200 non-null   int64 
 23  ExperienceYearsInCurrentRole  1200 non-null   int64 
 24  YearsSinceLastPromotion       1200 non-null   int64 
 25  YearsWithCurrManager          1200 non-null   int64 
 26  Attrition                     1200 non-null   object
 27  PerformanceRating             1200 non-null   int64 
dtypes: int64(19), object(9)
memory usage: 262.6+ KB
```

Checking for outliers

In [80]:

```
plt.figure(figsize=(15,15))
performance[['NumCompaniesWorked','TotalWorkExperienceInYears','TrainingTimesLastYear','Exp
            'ExperienceYearsInCurrentRole','YearsSinceLastPromotion','YearsWithCurrManager'
```



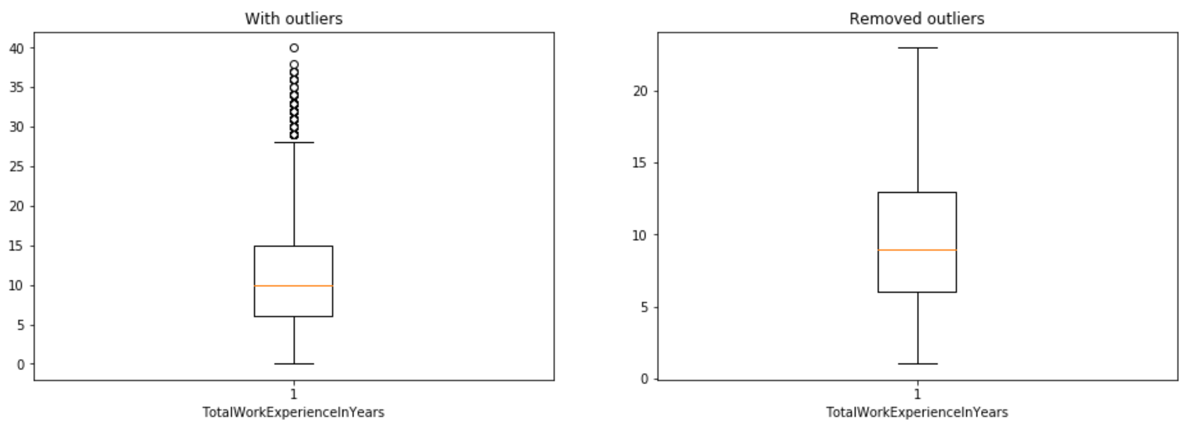
Note : Here more outliers are present in the fields TotalWorkExperienceInYears, ExperienceYearsAtThisCompany, YearsSinceLastPromotion. So we only remove them from these fields. The other fields will remain as it is.

Removing outliers for TotalWorkExperienceInYears

In [81]:

```
X=performance.TotalWorkExperienceInYears
removed_outliers_TotalWorkExperienceInYears = X.between(X.quantile(0.01),X.quantile(0.90))
print(str(X[removed_outliers_TotalWorkExperienceInYears].size)+"/"+str(X.size)+" data point
plt.boxplot(X[removed_outliers_TotalWorkExperienceInYears]);
plt.xlabel("TotalWorkExperienceInYears")
figure,axis=plt.subplots(1,2,figsize=(16,5))
axis[0].boxplot(X);
axis[1].boxplot(X[removed_outliers_TotalWorkExperienceInYears]);
axis[0].set_title("With outliers")
axis[0].set_xlabel("TotalWorkExperienceInYears")
axis[1].set_title("Removed outliers")
axis[1].set_xlabel("TotalWorkExperienceInYears")

performance['clean_TotalWorkExperienceInYears']=X[removed_outliers_TotalWorkExperienceInYea
```

```
1082/1200 data points remain
```

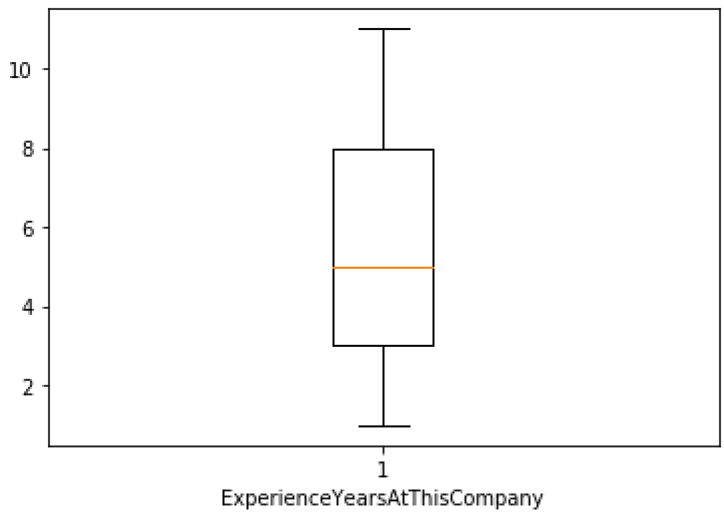


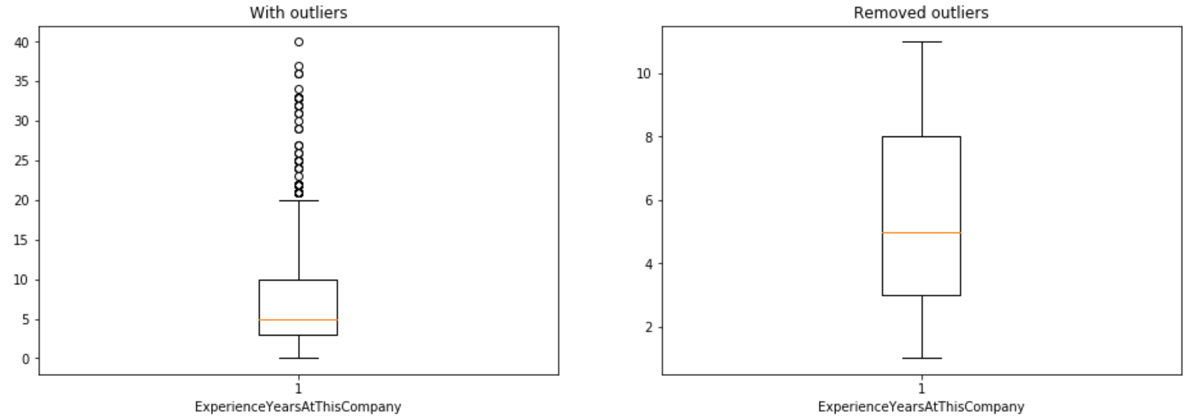
Removing outliers for ExperienceYearsAtThisCompany

In [82]:

```
X=performance.ExperienceYearsAtThisCompany
removed_outliers_ExperienceYearsAtThisCompany = X.between(X.quantile(0.05),X.quantile(0.85)
print(str(X[removed_outliers_ExperienceYearsAtThisCompany].size)+"/"+str(X.size)+" data poi
plt.boxplot(X[removed_outliers_ExperienceYearsAtThisCompany]);
plt.xlabel("ExperienceYearsAtThisCompany")
figure,axis=plt.subplots(1,2,figsize=(16,5))
axis[0].boxplot(X);
axis[1].boxplot(X[removed_outliers_ExperienceYearsAtThisCompany]);
axis[0].set_title("With outliers")
axis[0].set_xlabel("ExperienceYearsAtThisCompany")
axis[1].set_title("Removed outliers")
axis[1].set_xlabel("ExperienceYearsAtThisCompany")

performance['clean_ExperienceYearsAtThisCompany']=X[removed_outliers_ExperienceYearsAtThisC
```

```
985/1200 data points remain
```



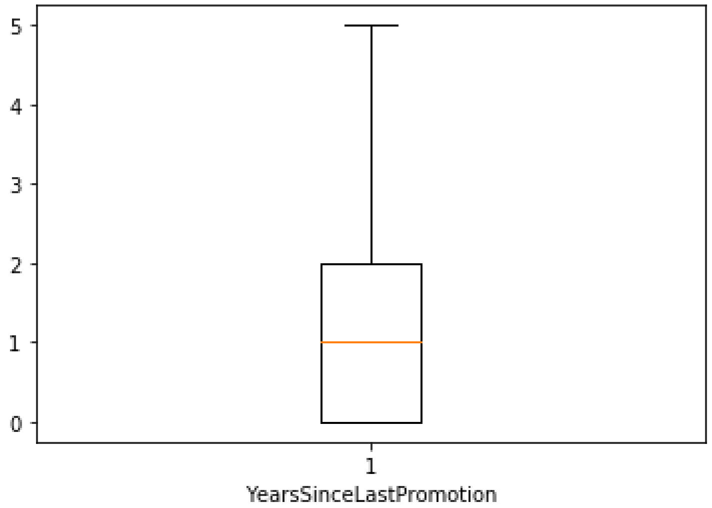


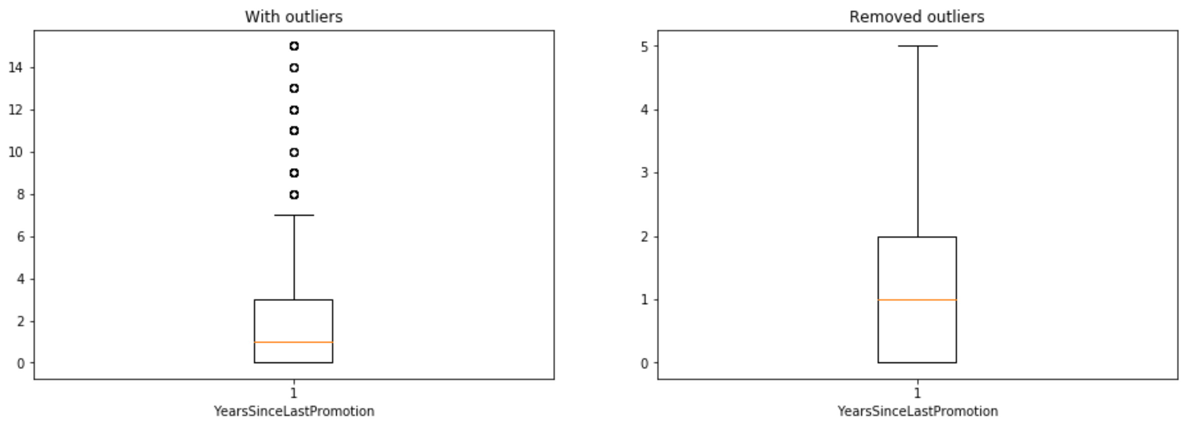
Removing outliers for YearsSinceLastPromotion

In [83]:

```
X=performance.YearsSinceLastPromotion
removed_outliers_YearsSinceLastPromotion = X.between(X.quantile(0.09),X.quantile(0.85))
print(str(X[removed_outliers_YearsSinceLastPromotion].size)+"/"+str(X.size)+" data points r
plt.boxplot(X[removed_outliers_YearsSinceLastPromotion]);
plt.xlabel("YearsSinceLastPromotion")
figure,axis=plt.subplots(1,2,figsize=(16,5))
axis[0].boxplot(X);
axis[1].boxplot(X[removed_outliers_YearsSinceLastPromotion]);
axis[0].set_title("With outliers")
axis[0].set_xlabel("YearsSinceLastPromotion")
axis[1].set_title("Removed outliers")
axis[1].set_xlabel("YearsSinceLastPromotion")

performance['clean_YearsSinceLastPromotion']=X[removed_outliers_YearsSinceLastPromotion]
```

```
1026/1200 data points remain
```





In [84]:

```
performance.head()
```

Out[84]:

	EmpNumber	Age	Gender	EducationBackground	MaritalStatus	EmpDepartment	EmpJobRo
0	E1001000	32	Male	Marketing	Single	Sales	Sale Executi\
1	E1001006	47	Male	Marketing	Single	Sales	Sale Executi\
2	E1001007	40	Male	Life Sciences	Married	Sales	Sale Executi\
3	E1001009	41	Male	Human Resources	Divorced	Human Resources	Manag
4	E1001010	60	Male	Marketing	Single	Sales	Sale Executi\

5 rows × 31 columns

In [85]:

```
performance.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1200 entries, 0 to 1199
Data columns (total 31 columns):
 #   Column                              Non-Null Count  Dtype  
---  ------                              --------------  -----  
 0   EmpNumber                           1200 non-null   object 
 1   Age                                 1200 non-null   int64  
 2   Gender                              1200 non-null   object 
 3   EducationBackground                 1200 non-null   object 
 4   MaritalStatus                       1200 non-null   object 
 5   EmpDepartment                       1200 non-null   object 
 6   EmpJobRole                          1200 non-null   object 
 7   BusinessTravelFrequency             1200 non-null   object 
 8   DistanceFromHome                    1200 non-null   int64  
 9   EmpEducationLevel                   1200 non-null   int64  
 10  EmpEnvironmentSatisfaction          1200 non-null   int64  
 11  EmpHourlyRate                       1200 non-null   int64  
 12  EmpJobInvolvement                   1200 non-null   int64  
 13  EmpJobLevel                         1200 non-null   int64  
 14  EmpJobSatisfaction                  1200 non-null   int64  
 15  NumCompaniesWorked                  1200 non-null   int64  
 16  OverTime                            1200 non-null   object 
 17  EmpLastSalaryHikePercent            1200 non-null   int64  
 18  EmpRelationshipSatisfaction         1200 non-null   int64  
 19  TotalWorkExperienceInYears          1200 non-null   int64  
 20  TrainingTimesLastYear               1200 non-null   int64  
 21  EmpWorkLifeBalance                  1200 non-null   int64  
 22  ExperienceYearsAtThisCompany        1200 non-null   int64  
 23  ExperienceYearsInCurrentRole        1200 non-null   int64  
 24  YearsSinceLastPromotion             1200 non-null   int64  
 25  YearsWithCurrManager                1200 non-null   int64  
 26  Attrition                           1200 non-null   object 
 27  PerformanceRating                   1200 non-null   int64  
 28  clean_TotalWorkExperienceInYears    1082 non-null   float64
 29  clean_ExperienceYearsAtThisCompany  985 non-null    float64
 30  clean_YearsSinceLastPromotion       1026 non-null   float64
dtypes: float64(3), int64(19), object(9)
memory usage: 290.8+ KB
```

Drop the columns from where the outliers are removed

In [86]:

```
performance.drop(columns=['TotalWorkExperienceInYears','ExperienceYearsAtThisCompany','Year
```

In [87]:

```
performance.head()
```

Out[87]:

	EmpNumber	Age	Gender	EducationBackground	MaritalStatus	EmpDepartment	EmpJobRo
0	E1001000	32	Male	Marketing	Single	Sales	Sale Executiv
1	E1001006	47	Male	Marketing	Single	Sales	Sale Executiv
2	E1001007	40	Male	Life Sciences	Married	Sales	Sale Executiv
3	E1001009	41	Male	Human Resources	Divorced	Human Resources	Manag
4	E1001010	60	Male	Marketing	Single	Sales	Sale Executiv

5 rows × 28 columns

In [88]:

```
performance.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1200 entries, 0 to 1199
Data columns (total 28 columns):
 #   Column                              Non-Null Count  Dtype  
---  ------                              --------------  -----  
 0   EmpNumber                           1200 non-null   object 
 1   Age                                 1200 non-null   int64  
 2   Gender                              1200 non-null   object 
 3   EducationBackground                 1200 non-null   object 
 4   MaritalStatus                       1200 non-null   object 
 5   EmpDepartment                       1200 non-null   object 
 6   EmpJobRole                          1200 non-null   object 
 7   BusinessTravelFrequency             1200 non-null   object 
 8   DistanceFromHome                    1200 non-null   int64  
 9   EmpEducationLevel                   1200 non-null   int64  
 10  EmpEnvironmentSatisfaction          1200 non-null   int64  
 11  EmpHourlyRate                       1200 non-null   int64  
 12  EmpJobInvolvement                   1200 non-null   int64  
 13  EmpJobLevel                         1200 non-null   int64  
 14  EmpJobSatisfaction                  1200 non-null   int64  
 15  NumCompaniesWorked                  1200 non-null   int64  
 16  OverTime                            1200 non-null   object 
 17  EmpLastSalaryHikePercent            1200 non-null   int64  
 18  EmpRelationshipSatisfaction         1200 non-null   int64  
 19  TrainingTimesLastYear               1200 non-null   int64  
 20  EmpWorkLifeBalance                  1200 non-null   int64  
 21  ExperienceYearsInCurrentRole        1200 non-null   int64  
 22  YearsWithCurrManager                1200 non-null   int64  
 23  Attrition                           1200 non-null   object 
 24  PerformanceRating                   1200 non-null   int64  
 25  clean_TotalWorkExperienceInYears    1082 non-null   float64
 26  clean_ExperienceYearsAtThisCompany  985 non-null    float64
 27  clean_YearsSinceLastPromotion       1026 non-null   float64
dtypes: float64(3), int64(16), object(9)
memory usage: 262.6+ KB
```

--

Using Label Encoder to convert categorical data to numerical data

Data Processing / Data Munging

It is the process of transforming and mapping data from one raw data form into another format for making it valuable for analytics.

Import the package

In [89]:

```
from sklearn.preprocessing import LabelEncoder,scale, StandardScaler
```

In [90]:

```
performance.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1200 entries, 0 to 1199
Data columns (total 28 columns):
 #   Column                              Non-Null Count  Dtype  
---  ------                              --------------  -----  
 0   EmpNumber                           1200 non-null   object 
 1   Age                                 1200 non-null   int64  
 2   Gender                              1200 non-null   object 
 3   EducationBackground                 1200 non-null   object 
 4   MaritalStatus                       1200 non-null   object 
 5   EmpDepartment                       1200 non-null   object 
 6   EmpJobRole                          1200 non-null   object 
 7   BusinessTravelFrequency             1200 non-null   object 
 8   DistanceFromHome                    1200 non-null   int64  
 9   EmpEducationLevel                   1200 non-null   int64  
 10  EmpEnvironmentSatisfaction          1200 non-null   int64  
 11  EmpHourlyRate                       1200 non-null   int64  
 12  EmpJobInvolvement                   1200 non-null   int64  
 13  EmpJobLevel                         1200 non-null   int64  
 14  EmpJobSatisfaction                  1200 non-null   int64  
 15  NumCompaniesWorked                  1200 non-null   int64  
 16  OverTime                            1200 non-null   object 
 17  EmpLastSalaryHikePercent            1200 non-null   int64  
 18  EmpRelationshipSatisfaction         1200 non-null   int64  
 19  TrainingTimesLastYear               1200 non-null   int64  
 20  EmpWorkLifeBalance                  1200 non-null   int64  
 21  ExperienceYearsInCurrentRole        1200 non-null   int64  
 22  YearsWithCurrManager                1200 non-null   int64  
 23  Attrition                           1200 non-null   object 
 24  PerformanceRating                   1200 non-null   int64  
 25  clean_TotalWorkExperienceInYears    1082 non-null   float64
 26  clean_ExperienceYearsAtThisCompany  985 non-null    float64
 27  clean_YearsSinceLastPromotion       1026 non-null   float64
dtypes: float64(3), int64(16), object(9)
memory usage: 262.6+ KB
```

In [91]:

```
enc=LabelEncoder()
performance_label=performance.apply(enc.fit_transform)
```

In [92]:

```
performance_label.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1200 entries, 0 to 1199
Data columns (total 28 columns):
 #   Column                             Non-Null Count  Dtype
---  ------                             --------------  -----
 0   EmpNumber                          1200 non-null   int32
 1   Age                                1200 non-null   int64
 2   Gender                             1200 non-null   int32
 3   EducationBackground                1200 non-null   int32
 4   MaritalStatus                      1200 non-null   int32
 5   EmpDepartment                      1200 non-null   int32
 6   EmpJobRole                         1200 non-null   int32
 7   BusinessTravelFrequency            1200 non-null   int32
 8   DistanceFromHome                   1200 non-null   int64
 9   EmpEducationLevel                  1200 non-null   int64
 10  EmpEnvironmentSatisfaction         1200 non-null   int64
 11  EmpHourlyRate                      1200 non-null   int64
 12  EmpJobInvolvement                  1200 non-null   int64
 13  EmpJobLevel                        1200 non-null   int64
 14  EmpJobSatisfaction                 1200 non-null   int64
 15  NumCompaniesWorked                 1200 non-null   int64
 16  OverTime                           1200 non-null   int32
 17  EmpLastSalaryHikePercent           1200 non-null   int64
 18  EmpRelationshipSatisfaction        1200 non-null   int64
 19  TrainingTimesLastYear              1200 non-null   int64
 20  EmpWorkLifeBalance                 1200 non-null   int64
 21  ExperienceYearsInCurrentRole       1200 non-null   int64
 22  YearsWithCurrManager               1200 non-null   int64
 23  Attrition                          1200 non-null   int32
 24  PerformanceRating                  1200 non-null   int64
 25  clean_TotalWorkExperienceInYears   1200 non-null   int64
 26  clean_ExperienceYearsAtThisCompany 1200 non-null   int64
 27  clean_YearsSinceLastPromotion      1200 non-null   int64
dtypes: int32(9), int64(19)
memory usage: 220.4 KB
```

In [93]:

```
performance_label.head()
```

Out[93]:

	EmpNumber	Age	Gender	EducationBackground	MaritalStatus	EmpDepartment	EmpJobRo
0	0	14	1	2	2	5	1
1	1	29	1	2	2	5	1
2	2	22	1	1	1	5	1
3	3	23	1	0	0	3	
4	4	42	1	2	2	5	1

5 rows × 28 columns

In [94]:

```
performance_label.isna().sum().to_frame().T
```

Out[94]:

	EmpNumber	Age	Gender	EducationBackground	MaritalStatus	EmpDepartment	EmpJobRo
0	0	0	0	0	0	0	

1 rows × 28 columns

In [95]:

```
performance_label.shape
```

Out[95]:

(1200, 28)

Define X and y variables

In [96]:

```
X=performance_label.iloc[:,performance.columns!='PerformanceRating']
y=performance_label.PerformanceRating
```

In [97]:

```
X.head()
```

Out[97]:

	EmpNumber	Age	Gender	EducationBackground	MaritalStatus	EmpDepartment	EmpJobRo
0	0	14	1	2	2	5	1
1	1	29	1	2	2	5	1
2	2	22	1	1	1	5	1
3	3	23	1	0	0	3	
4	4	42	1	2	2	5	1

5 rows × 27 columns

Use Train-Test split to divide test and train data

It splits arrays or matrices into random train and test subsets

In [98]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=10)
```

In [99]:

```
print("X_train shape = ",X_train.shape)
print("X_test shape = ",X_test.shape)
print("y_train shape = ",y_train.shape)
print("y_test shape = ",y_test.shape)
```

```
X_train shape =  (960, 27)
X_test shape =  (240, 27)
y_train shape =  (960,)
y_test shape =  (240,)
```

Scaling technique to be used for standardizing the dataset along X axis

In [100]:

```
X=scale(X)
```

In [101]:

```
print("X_train shape = ",X_train.shape)
print("X_test shape = ",X_test.shape)
```

```
X_train shape =  (960, 27)
X_test shape =  (240, 27)
```

Using StandardScaler

It standardizes features by removing the mean and scaling to unit variance.

In [102]:

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

In [103]:

```
print("X_train shape = ",X_train.shape)
print("X_test shape = ",X_test.shape)
print("y_train shape = ",y_train.shape)
print("y_test shape = ",y_test.shape)
```

```
X_train shape =  (960, 27)
X_test shape =  (240, 27)
y_train shape =  (960,)
y_test shape =  (240,)
```

--

3) Different Machine Learning Algorithms to train and predict the model

3.1) Using Random Forest Classifier

Import the necessary packages

In [104]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,precision_score,confusion_matrix,classification_
```

Define and Train the model

Here **model.fit()** fits training data. It accepts two arguments :the data X_train and the labels y_train.

In [105]:

```
model=RandomForestClassifier(n_estimators=250,random_state=10,criterion='gini')
model.fit(X_train,y_train)
print(" Model Feature Importances = " ,model.feature_importances_)
```

```
 Model Feature Importances =  [0.05695015 0.02787464 0.00507008 0.01069518
0.01036765 0.03260104
 0.03402633 0.01018268 0.02580976 0.013669   0.20529141 0.03229078
 0.01053392 0.01160763 0.01248842 0.01657791 0.00530697 0.20557607
 0.01235847 0.01728891 0.02524584 0.03681119 0.02589432 0.00431312
 0.02815546 0.02476927 0.09824382]
```

Predict the model

It generates output predictions for the input samples.

In [106]:

```
y_train_predict=model.predict(X_train)
y_predict=model.predict(X_test)
```

Calculating the confusion matrix

A confusion matrix is a matrix used to measure the performance of an machine learning algorithm mainly a supervised learning one. Each row of the confusion matrix represents the instances of an actual class and each column represents the instances of a predicted class.

In [107]:

```
confusion_matrix(y_test,y_predict)
```

Out[107]:

```
array([[ 35,   3,   0],
       [  2, 174,   3],
       [  1,   5,  17]], dtype=int64)
```

Generating crosstab

A crosstab is a table showing the relationship between two or more variables.

The table only shows the relationship between two categorical variables.

It is also known as a contingency table.

In this case, it is displaying the details of confusion matrix.

In [108]:

```
pd.crosstab(y_test,y_predict)
```

Out[108]:

col_0	0	1	2
PerformanceRating			
0	35	3	0
1	2	174	3
2	1	5	17

Calculating the accuracy score

It is the ratio of correctly predicted observation to the total observations.

Accuracy = TP+TN/TP+FP+FN+TN where,

True Positives (TP) - Correctly predicted positive values which means that the value of actual class is yes and the value of predicted class is also yes.

True Negatives (TN) - Correctly predicted negative values which means that the value of actual class is no and value of predicted class is also no.

False positives and false negatives, these values occur when your actual class contradicts with the predicted class.

False Positives (FP) – It occurs when actual class is no and predicted class is yes. False Negatives (FN) – It occurs when actual class is yes but predicted class in no.

In [109]:

```
print("\n Accuracy of Training = " ,accuracy_score(y_train,y_train_predict)*100)
print("\n Accuracy of Testing =  ", accuracy_score(y_test,y_predict)*100)
```

```
 Accuracy of Training =  100.0

 Accuracy of Testing =   94.16666666666667
```

Calculating Precision score, recall score and F1 score

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations.

Precision = TP/TP+FP

Recall (Sensitivity) is the ratio of correctly predicted positive observations to the all observations in actual class - yes.

Recall = TP/TP+FN

F1 score is the weighted average of Precision and Recall. So, this score takes both false positives and false negatives into account.

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

In [110]:

```
print("Precision score = ",(precision_score(y_test,y_predict,average='weighted')*100))
print("Recall score = ",(recall_score(y_test,y_predict,average='weighted')*100))
print("F1 score = ",(f1_score(y_test,y_predict,average='weighted')*100))
```

```
Precision score =  94.03411172161174
Recall score =  94.16666666666667
F1 score =  94.05835963838605
```

Genarating classification report

It builds a text report showing the main classification metrics

In [111]:

```
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       0.92      0.92      0.92        38
           1       0.96      0.97      0.96       179
           2       0.85      0.74      0.79        23

    accuracy                           0.94       240
   macro avg       0.91      0.88      0.89       240
weighted avg       0.94      0.94      0.94       240
```

3.2) Using Gradient Boosting Classifier

Importing the package

In [112]:

```
from sklearn.ensemble import GradientBoostingClassifier
```

Using train-test split and standard scaler

In [113]:

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=10)
```

In [114]:

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

Define and Train the model

In [115]:

```
model=GradientBoostingClassifier(learning_rate=0.1,n_estimators=100,subsample=1.0,max_depth
model.fit(X_train,y_train)
print(" Model Feature Importances = " ,model.feature_importances_)
```

```
 Model Feature Importances =  [0.04866955 0.0050496  0.00043461 0.00057209
0.00045569 0.06290332
 0.01481567 0.00246649 0.00738162 0.00141899 0.28589074 0.00785092
 0.00047714 0.00120089 0.00205885 0.00137072 0.00159579 0.23808559
 0.00092834 0.00730749 0.04842203 0.04007265 0.01041203 0.00032288
 0.00622111 0.00707224 0.19654298]
```

Predict the model

In [116]:

```
y_train_predict=model.predict(X_train)
y_predict=model.predict(X_test)
```

Calculating the confusion matrix

In [117]:

```
confusion_matrix(y_test,y_predict)
```

Out[117]:

```
array([[ 35,   3,   0],
       [  2, 176,   1],
       [  0,   3,  20]], dtype=int64)
```

Generating crosstab

In [118]:

```
pd.crosstab(y_test,y_predict)
```

Out[118]:

col_0	0	1	2
PerformanceRating			
0	35	3	0
1	2	176	1
2	0	3	20

Calculating the accuracy score

In [119]:

```
print("\n Accuracy of Training = " ,accuracy_score(y_train,y_train_predict)*100)
print("\n Accuracy of Testing =  ", accuracy_score(y_test,y_predict)*100)
```

```
 Accuracy of Training =  99.6875

 Accuracy of Testing =   96.25
```

Calculating Precision score, recall score and F1 score

In [120]:

```
print("Precision score = ",(precision_score(y_test,y_predict,average='weighted')*100))
print("Recall score = ",(recall_score(y_test,y_predict,average='weighted')*100))
print("F1 score = ",(f1_score(y_test,y_predict,average='weighted')*100))
```

```
Precision score =  96.22900372900372
Recall score =  96.25
F1 score =  96.21381404068383
```

Generating classification report

In [121]:

```
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       0.95      0.92      0.93        38
           1       0.97      0.98      0.98       179
           2       0.95      0.87      0.91        23

    accuracy                           0.96       240
   macro avg       0.96      0.92      0.94       240
weighted avg       0.96      0.96      0.96       240
```

--

Different techniques used in the model

1] Feature Engineering

Import the package

In [122]:

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

Sort the values as per the correlation with respect to Performance Rating

In [123]:

```
performance_label.corr()['PerformanceRating'].sort_values()
```

Out[123]:

```
EmpDepartment                        -0.162615
ExperienceYearsInCurrentRole         -0.147638
YearsWithCurrManager                 -0.122313
EmpJobRole                           -0.096209
EmpJobLevel                          -0.076632
clean_ExperienceYearsAtThisCompany   -0.076598
clean_YearsSinceLastPromotion        -0.071304
DistanceFromHome                     -0.046142
EmpHourlyRate                        -0.043116
Age                                  -0.040164
Attrition                            -0.039796
clean_TotalWorkExperienceInYears     -0.031405
BusinessTravelFrequency              -0.031025
EmpRelationshipSatisfaction          -0.019502
EmpJobInvolvement                    -0.010539
TrainingTimesLastYear                -0.005443
EmpNumber                            -0.003163
Gender                               -0.001780
EmpJobSatisfaction                    0.000606
EducationBackground                   0.005607
EmpEducationLevel                     0.020529
NumCompaniesWorked                    0.020980
MaritalStatus                         0.024172
OverTime                              0.050206
EmpWorkLifeBalance                    0.124429
EmpLastSalaryHikePercent              0.333722
EmpEnvironmentSatisfaction            0.395561
PerformanceRating                     1.000000
Name: PerformanceRating, dtype: float64
```

Define X and y variables

In [124]:

```
X = performance_label.loc[:,performance.columns!='PerformanceRating']
y = performance_label.PerformanceRating
```

Use train-test split to divide test and train data

In [125]:

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=10)
```

In [126]:

```
print("X_train shape = ",X_train.shape)
print("X_test shape = ",X_test.shape)
print("y_train shape = ",y_train.shape)
print("y_test shape = ",y_test.shape)
```

```
X_train shape =  (960, 27)
X_test shape =  (240, 27)
y_train shape =  (960,)
y_test shape =  (240,)
```

Define the Model

In [127]:

```
model = RandomForestClassifier(n_estimators=20)
model.fit(X_train,y_train)
pd.DataFrame(model.feature_importances_,index=X.columns).sort_values(0,ascending=False)
```

Out[127]:

	0
EmpEnvironmentSatisfaction	0.198675
EmpLastSalaryHikePercent	0.181176
clean_YearsSinceLastPromotion	0.091609
EmpNumber	0.063455
EmpDepartment	0.040217
EmpHourlyRate	0.040096
ExperienceYearsInCurrentRole	0.035169
Age	0.033780
EmpJobRole	0.031434
clean_TotalWorkExperienceInYears	0.029958
clean_ExperienceYearsAtThisCompany	0.028322
DistanceFromHome	0.027593
EmpWorkLifeBalance	0.024659
YearsWithCurrManager	0.019427
NumCompaniesWorked	0.019002
EmpEducationLevel	0.017008
EducationBackground	0.016079
EmpRelationshipSatisfaction	0.015867
TrainingTimesLastYear	0.015042
EmpJobLevel	0.012963
EmpJobSatisfaction	0.011942
EmpJobInvolvement	0.009508
BusinessTravelFrequency	0.009490
MaritalStatus	0.008161
OverTime	0.007772
Gender	0.006651
Attrition	0.004947

Predict the Model

In [128]:

```
y_train_predict=model.predict(X_train)
y_predict=model.predict(X_test)
```

Calculating the confusion matrix and generating crosstab

In [129]:

```
print(confusion_matrix(y_test,y_predict))
pd.crosstab(y_test,y_predict)
```

```
[[ 34   4   0]
 [  2 174   3]
 [  0   8  15]]
```

Out[129]:

col_0	0	1	2
PerformanceRating			
0	34	4	0
1	2	174	3
2	0	8	15

Calculating the accuracy score, precision score, recall score and F1 score

In [130]:

```
print("Accuracy of Training = " ,accuracy_score(y_train,y_train_predict)*100)
print("Accuracy of Testing =  ", accuracy_score(y_test,y_predict)*100)
print("Precision score = ",(precision_score(y_test,y_predict,average='weighted')*100))
print("Recall score = ",(recall_score(y_test,y_predict,average='weighted')*100))
print("F1 score = ",(f1_score(y_test,y_predict,average='weighted')*100))
```

```
Accuracy of Training =  99.89583333333333
Accuracy of Testing =   92.91666666666667
Precision score =  92.71132019115889
Recall score =  92.91666666666667
F1 score =  92.67133371259668
```

Generating classification report

In [131]:

```
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       0.94      0.89      0.92        38
           1       0.94      0.97      0.95       179
           2       0.83      0.65      0.73        23

    accuracy                           0.93       240
   macro avg       0.90      0.84      0.87       240
weighted avg       0.93      0.93      0.93       240
```

--

2] Using GridSearch Cross Validation(CV) in Random Forest Classifier

Import the package

In [132]:

```
from sklearn.model_selection import GridSearchCV
```

Using train-test split and standard scaler

In [133]:

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=10)
```

In [134]:

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

Define and Train the model

In [135]:

```
model=RandomForestClassifier(max_depth=3,
                             criterion='gini',
                             n_estimators=10,
                             random_state=5)
parameters={'max_depth': [2,3],
              'n_estimators':[10,20],
              'random_state' : [5]}
grid=GridSearchCV(model,parameters,scoring='accuracy',cv=15)
grid.fit(X_train,y_train)
```

Out[135]:

```
GridSearchCV(cv=15, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=3,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=10, n_jobs=None,
                                              oob_score=False, random_state=
5,
                                              verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [2, 3], 'n_estimators': [10, 20],
                         'random_state': [5]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

best_score_: Mean cross-validated score of the best_estimator.

best_params_: Parameter setting that gave the best results on the hold out data.

In [136]:

```
print("Best Score =  ",grid.best_score_)
print("Best Params = ",grid.best_params_)
```

```
Best Score =   0.7666666666666667
Best Params =  {'max_depth': 3, 'n_estimators': 20, 'random_state': 5}
```

Predict the model

In [137]:

```
y_train_predict=grid.predict(X_train)
y_predict=grid.predict(X_test)
```

Calculating the confusion matrix and generating crosstab

In [138]:

```
print(confusion_matrix(y_test,y_predict))
pd.crosstab(y_test,y_predict)
```

```
[[  2  36   0]
 [  0 177   2]
 [  0  10  13]]
```

Out[138]:

col_0	0	1	2
PerformanceRating			
0	2	36	0
1	0	177	2
2	0	10	13

Calculating Accuracy Score,Precision score, recall score and F1 score

In [139]:

```
print("Accuracy Score of Training = ",(accuracy_score(y_train,y_train_predict)*100))
print("Accuracy Score of Testing = ",(accuracy_score(y_test,y_predict)*100))
print("Precision score = ",(precision_score(y_test,y_predict,average='weighted')*100))
print("Recall score = ",(recall_score(y_test,y_predict,average='weighted')*100))
print("F1 score = ",(f1_score(y_test,y_predict,average='weighted')*100))
```

```
Accuracy Score of Training =  79.27083333333333
Accuracy Score of Testing =  80.0
Precision score =  83.33731938216242
Recall score =  80.0
F1 score =  73.81821157371039
```

Generating classification report

In [140]:

```
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       1.00      0.05      0.10        38
           1       0.79      0.99      0.88       179
           2       0.87      0.57      0.68        23

    accuracy                           0.80       240
   macro avg       0.89      0.54      0.55       240
weighted avg       0.83      0.80      0.74       240
```

3] Using Randomized Search Cross Validation(CV) in Random Forest Classifier

Import the package

In [141]:

```
from sklearn.model_selection import RandomizedSearchCV
```

Using train-test split and standard scaler

In [142]:

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=10)
```

In [143]:

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

Define and Train the model

In [144]:

```
model=RandomForestClassifier(max_depth=3,
                             criterion='gini',
                             n_estimators=10,
                             random_state=5)
parameters={'max_depth': [2,3],
              'n_estimators':[10,20],
              'random_state' : [5]}
randomized=RandomizedSearchCV(model,parameters,scoring='accuracy',cv=5)
randomized.fit(X_train,y_train)
```

Out[144]:

```
RandomizedSearchCV(cv=5, error_score=nan,
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    ccp_alpha=0.0,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=3,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=0.
0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf
=0.0,
                                                    n_estimators=10,
                                                    n_jobs=None,
                                                    oob_score=False,
                                                    random_state=5, verbose=
0,
                                                    warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'max_depth': [2, 3],
                                        'n_estimators': [10, 20],
                                        'random_state': [5]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring='accuracy', verbose=0)
```

In [145]:

```
print("Best Score =  ",randomized.best_score_)
print("Best Params = ",randomized.best_params_)
```

```
Best Score =   0.7614583333333333
Best Params =  {'random_state': 5, 'n_estimators': 20, 'max_depth': 3}
```

Predict the model

In [146]:

```
y_train_predict=randomized.predict(X_train)
y_predict=randomized.predict(X_test)
```

Calculating the confusion matrix and generating crosstab

In [147]:

```
print(confusion_matrix(y_test,y_predict))
pd.crosstab(y_test,y_predict)
```

```
[[  2  36   0]
 [  0 177   2]
 [  0  10  13]]
```

Out[147]:

col_0	0	1	2
PerformanceRating			
0	2	36	0
1	0	177	2
2	0	10	13

Calculating Accuracy Score,Precision score, recall score and F1 score

In [148]:

```
print("Accuracy Score of Training = ",(accuracy_score(y_train,y_train_predict)*100))
print("Accuracy Score of Testing= ",(accuracy_score(y_test,y_predict)*100))
print("Precision score = ",(precision_score(y_test,y_predict,average='weighted')*100))
print("Recall score = ",(recall_score(y_test,y_predict,average='weighted')*100))
print("F1 score = ",(f1_score(y_test,y_predict,average='weighted')*100))
```

```
Accuracy Score of Training =  79.27083333333333
Accuracy Score of Testing=  80.0
Precision score =  83.33731938216242
Recall score =  80.0
F1 score =  73.81821157371039
```

Generating classification report

In [149]:

```
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       1.00      0.05      0.10        38
           1       0.79      0.99      0.88       179
           2       0.87      0.57      0.68        23

    accuracy                           0.80       240
   macro avg       0.89      0.54      0.55       240
weighted avg       0.83      0.80      0.74       240
```

--

4] Using Synthetic Minority Over-sampling Technique(SMOTE) in Random Forest Classifier

In [150]:

```
from collections import Counter
```

In [151]:

```
Counter(performance_label.PerformanceRating)
```

Out[151]:

```
Counter({1: 874, 2: 132, 0: 194})
```

Import the necessary package

In [152]:

```
from imblearn.over_sampling import SMOTE
smote=SMOTE(random_state=5)
```

```
Using TensorFlow backend.
```

Using train-test split

In [153]:

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=10)
```

Define and Train the model

In [154]:

```
X_train_smote,y_train_smote=smote.fit_sample(X_train.astype('float'),y_train)
print(Counter(y_train))
print(Counter(y_train_smote))
```

```
Counter({1: 695, 0: 156, 2: 109})
Counter({1: 695, 2: 695, 0: 695})
```

In [155]:

```
model.fit(X_train_smote,y_train_smote)
```

Out[155]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=3, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=5, verbose
=0,
                       warm_start=False)
```

In [156]:

```
X_train_smote.shape
```

Out[156]:

```
(2085, 27)
```

In [157]:

```
y_train_smote.shape
```

Out[157]:

```
(2085,)
```

Predict the model

In [158]:

```
y_train_predict=model.predict(X_train)
y_predict=model.predict(X_test)
```

In [159]:

```
y_predict.shape
```

Out[159]:

```
(240,)
```

Calculating the confusion matrix and generating crosstab

In [160]:

```
print(confusion_matrix(y_test,y_predict))
pd.crosstab(y_test,y_predict)
```

```
[[ 35   3   0]
 [ 10 160   9]
 [  0   4  19]]
```

Out[160]:

col_0	0	1	2
PerformanceRating			
0	35	3	0
1	10	160	9
2	0	4	19

Calculating the accuracy score, precision score, recall score and F1 score

In [161]:

```
print("Accuracy score of Training = ",accuracy_score(y_train,y_train_predict)*100)
print("Accuracy score of Testing = ",accuracy_score(y_test,y_predict)*100)
print("Precision score = ",(precision_score(y_test,y_predict,average='weighted')*100))
print("Recall score = ",(recall_score(y_test,y_predict,average='weighted')*100))
print("F1 score = ",(f1_score(y_test,y_predict,average='weighted')*100))
```

```
Accuracy score of Training =  88.85416666666667
Accuracy score of Testing =  89.16666666666667
Precision score =  90.27487683363431
Recall score =  89.16666666666667
F1 score =  89.47274192542584
```

Generating classification report

In [162]:

```
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       0.78      0.92      0.84        38
           1       0.96      0.89      0.92       179
           2       0.68      0.83      0.75        23

    accuracy                           0.89       240
   macro avg       0.80      0.88      0.84       240
weighted avg       0.90      0.89      0.89       240
```

Displaying X_train and y_train data after performing Synthetic Minority Over-sampling Technique(SMOTE)

In [163]:

```
pd.DataFrame(X_train_smote).head()
```

Out[163]:

	EmpNumber	Age	Gender	EducationBackground	MaritalStatus	EmpDepartment	EmpJobRo
0	1011.0	0.0	1.0	1.0	2.0	1.0	3
1	924.0	25.0	1.0	3.0	0.0	1.0	18
2	654.0	18.0	1.0	0.0	2.0	3.0	6
3	716.0	29.0	1.0	3.0	2.0	4.0	9
4	681.0	31.0	1.0	1.0	1.0	4.0	9

5 rows × 27 columns

In [164]:

```
pd.DataFrame(X_train_smote).tail()
```

Out[164]:

	EmpNumber	Age	Gender	EducationBackground	MaritalStatus	EmpDepartment	I
2080	114.227772	17.783522	1.000000	3.216478	2.000000	1.000000	
2081	855.504562	18.392531	0.392531	0.607469	0.000000	1.785063	
2082	52.484588	15.979211	0.783513	1.000000	1.000000	1.000000	
2083	756.196770	24.803230	1.000000	1.000000	0.000000	3.560646	
2084	1187.009764	17.882930	0.151219	2.302439	1.848781	4.395123	

5 rows × 27 columns

In [165]:

```
pd.DataFrame(y_train_smote).head()
```

Out[165]:

	PerformanceRating
0	1
1	1
2	2
3	0
4	0

In [166]:

```
pd.DataFrame(y_train_smote).tail()
```

Out[166]:

	PerformanceRating
2080	2
2081	2
2082	2
2083	2
2084	2

--

Other Machine Learning Algorithms

3.3) Using eXtreme Gradient Boosting(XGBoosting) Classifier

Import the packages

In [167]:

```
from xgboost import XGBClassifier
```

Using train-test split and standard scaler

In [168]:

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=10)
```

In [169]:

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

Define and Train the model

In [170]:

```
model=XGBClassifier(max_depth=3,learning_rate=0.1,test_size=0.2,n_estimators=100,n_jobs=1,r
```

In [171]:

```
model.fit(X_train,y_train)
```

Out[171]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0.1,
              learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
              nthread=None, objective='multi:softprob', random_state=10,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, test_size=0.2, verbosity=1)
```

Predict the model

In [172]:

```
y_train_predict=model.predict(X_train)
y_predict=model.predict(X_test)
```

In [173]:

```
y_predict.shape
```

Out[173]:

```
(240,)
```

Calculating confusion matrix and generating crosstab

In [174]:

```
print(confusion_matrix(y_test,y_predict))
pd.crosstab(y_test,y_predict)
```

```
[[ 35   3   0]
 [  2 176   1]
 [  0   5  18]]
```

Out[174]:

col_0	0	1	2
PerformanceRating			
0	35	3	0
1	2	176	1
2	0	5	18

Calculating Accuracy score,Precision score, recall score and F1 score

In [175]:

```
print("Accuracy score of Training = ",accuracy_score(y_train,y_train_predict)*100)
print("Accuracy score of Testing = ",accuracy_score(y_test,y_predict)*100)
print("Precision score = ",(precision_score(y_test,y_predict,average='weighted')*100))
print("Recall score = ",(recall_score(y_test,y_predict,average='weighted')*100))
print("F1 score = ",(f1_score(y_test,y_predict,average='weighted')*100))
```

```
Accuracy score of Training =  98.85416666666667
Accuracy score of Testing =  95.41666666666667
Precision score =  95.39700455604347
Recall score =  95.41666666666667
F1 score =  95.31529581529583
```

Generating classification report

In [176]:

```
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       0.95      0.92      0.93        38
           1       0.96      0.98      0.97       179
           2       0.95      0.78      0.86        23

    accuracy                           0.95       240
   macro avg       0.95      0.90      0.92       240
weighted avg       0.95      0.95      0.95       240
```

--

3.4) Using Artificial Neural Networks

Import the necessary package

In [177]:

```
from sklearn.neural_network import MLPClassifier
```

Using train-test split

In [178]:

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=10)
```

Define and Train the model

In [179]:

```
model=MLPClassifier(hidden_layer_sizes=10,batch_size=10,learning_rate_init=0.01,random_stat
model.fit(X_train,y_train)
```

Out[179]:

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size=10, beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=10, learning_rate='constant',
              learning_rate_init=0.01, max_fun=15000, max_iter=200,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=5, shuffle=True, solver='adam',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
```

Predict the model

In [180]:

```
y_train_predict=model.predict(X_train)
y_predict=model.predict(X_test)
```

In [181]:

```
y_predict.shape
```

Out[181]:

```
(240,)
```

Calculating confusion matrix and generating crosstab

In [182]:

```
print(confusion_matrix(y_test,y_predict))
pd.crosstab(y_test,y_predict)
```

```
[[ 13  25   0]
 [ 10 167   2]
 [  0  10  13]]
```

Out[182]:

col_0	0	1	2
PerformanceRating			
0	13	25	0
1	10	167	2
2	0	10	13

Calculate the accuracy score, precision score, recall score and F1 score

In [183]:

```
print("Accuracy score of Training = ",accuracy_score(y_train,y_train_predict)*100)
print("Accuracy score of Testing = ",accuracy_score(y_test,y_predict)*100)
print("Precision score = ",(precision_score(y_test,y_predict,average='weighted')*100))
print("Recall score = ",(recall_score(y_test,y_predict,average='weighted')*100))
print("F1 score = ",(f1_score(y_test,y_predict,average='weighted')*100))
```

```
Accuracy score of Training =  79.58333333333333
Accuracy score of Testing =  80.41666666666667
Precision score =  78.9153094657292
Recall score =  80.41666666666667
F1 score =  78.68841607805926
```

Generating classification report

In [184]:

```
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       0.57      0.34      0.43        38
           1       0.83      0.93      0.88       179
           2       0.87      0.57      0.68        23

    accuracy                           0.80       240
   macro avg       0.75      0.61      0.66       240
weighted avg       0.79      0.80      0.79       240
```

--

3.5) Using K- Nearest Neighbors

Import the necessary packages

In [185]:

```
from sklearn.neighbors import KNeighborsClassifier
```

Using train-test split

In [186]:

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=10)
```

Define and Train the model

In [187]:

```
model= KNeighborsClassifier(n_neighbors=20, metric='euclidean')
model.fit(X_train,y_train)
```

Out[187]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='euclidean',
                     metric_params=None, n_jobs=None, n_neighbors=20, p=2,
                     weights='uniform')
```

Predict the Model

In [188]:

```
y_train_predict=model.predict(X_train)
y_predict = model.predict(X_test)
```

Calculating confusion matrix and generating crosstab

In [189]:

```
print(confusion_matrix(y_test,y_predict))
pd.crosstab(y_test,y_predict)
```

```
[[  2  36   0]
 [  0 179   0]
 [  1  22   0]]
```

Out[189]:

col_0	0	1
PerformanceRating		
0	2	36
1	0	179
2	1	22

Calculate the accuracy score, precision score, recall score and F1 score

In [190]:

```
print("Accuracy score of Training = ",accuracy_score(y_train,y_train_predict)*100)
print("Accuracy score of Testing = ",accuracy_score(y_test,y_predict)*100)
print("Precision score = ",(precision_score(y_test,y_predict,average='weighted')*100))
print("Recall score = ",(recall_score(y_test,y_predict,average='weighted')*100))
print("F1 score = ",(f1_score(y_test,y_predict,average='weighted')*100))
```

```
Accuracy score of Training =  72.5
Accuracy score of Testing =  75.41666666666667
Precision score =  66.88642756680731
Recall score =  75.41666666666667
F1 score =  65.72941095997498
```

Generating classification report

In [191]:

```
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       0.67      0.05      0.10        38
           1       0.76      1.00      0.86       179
           2       0.00      0.00      0.00        23

    accuracy                           0.75       240
   macro avg       0.47      0.35      0.32       240
weighted avg       0.67      0.75      0.66       240
```

--

3.6) Using Logistic Regression

Import the necessary packages

In [192]:

```
from sklearn.linear_model import LogisticRegression
```

Using train-test split and standard scaler

In [193]:

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=10)
```

In [194]:

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

Define and Train the model

In [195]:

```
model=LogisticRegression()
model.fit(X_train,y_train)
```

Out[195]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Predict the model

In [196]:

```
y_train_predict=model.predict(X_train)
y_predict= model.predict(X_test)
```

Calculating confusion matrix and generating crosstab

In [197]:

```
print(confusion_matrix(y_test,y_predict))
pd.crosstab(y_test,y_predict)
```

```
[[ 18  18   2]
 [  9 167   3]
 [  2   6  15]]
```

Out[197]:

col_0	0	1	2
PerformanceRating			
0	18	18	2
1	9	167	3
2	2	6	15

Calculate the accuracy score, precision score, recall score and F1 score

In [198]:

```
print("Accuracy score of Training = ",accuracy_score(y_train,y_train_predict)*100)
print("Accuracy score of Testing = ",accuracy_score(y_test,y_predict)*100)
print("Precision score = ",(precision_score(y_test,y_predict,average='micro')*100))
print("Recall score = ",(recall_score(y_test,y_predict,average='micro')*100))
print("F1 score = ",(f1_score(y_test,y_predict,average='micro')*100))
```

```
Accuracy score of Training =  82.8125
Accuracy score of Testing =  83.33333333333334
Precision score =  83.33333333333334
Recall score =  83.33333333333334
F1 score =  83.33333333333334
```

Generating classification report

In [199]:

```
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       0.62      0.47      0.54        38
           1       0.87      0.93      0.90       179
           2       0.75      0.65      0.70        23

    accuracy                           0.83       240
   macro avg       0.75      0.69      0.71       240
weighted avg       0.82      0.83      0.83       240
```

--

3.7) Using Support Vector Machine

Import the necessary package

In [200]:

```
from sklearn.svm import SVC
```

Using train-test split and standard scaler

In [201]:

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=10)
```

In [202]:

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

Define and Train the model

In [203]:

```
model=SVC(C=130,kernel = 'rbf',random_state=10)
model.fit(X_train,y_train)
```

Out[203]:

```
SVC(C=130, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=10, shrinking=True, tol=0.0
01,
    verbose=False)
```

Predict the model

In [204]:

```
y_train_predict=model.predict(X_train)
y_predict=model.predict(X_test)
```

Calculating confusion matrix and generating crosstab

In [205]:

```
print(confusion_matrix(y_test,y_predict))
pd.crosstab(y_test,y_predict)
```

```
[[ 22  16   0]
 [ 14 158   7]
 [  1   8  14]]
```

Out[205]:

col_0	0	1	2
PerformanceRating			
0	22	16	0
1	14	158	7
2	1	8	14

Calculate the accuracy score, precision score, recall score and F1 score

In [206]:

```
print("Accuracy score of Training = ",accuracy_score(y_train,y_train_predict)*100)
print("Accuracy score of Testing = ",accuracy_score(y_test,y_predict)*100)
print("Precision score = ",(precision_score(y_test,y_predict,average='micro')*100))
print("Recall score = ",(recall_score(y_test,y_predict,average='micro')*100))
print("F1 score = ",(f1_score(y_test,y_predict,average='micro')*100))
```

```
Accuracy score of Training =  100.0
Accuracy score of Testing =  80.83333333333333
Precision score =  80.83333333333333
Recall score =  80.83333333333333
F1 score =  80.83333333333333
```

Generating classification report

In [207]:

```
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       0.59      0.58      0.59        38
           1       0.87      0.88      0.88       179
           2       0.67      0.61      0.64        23

    accuracy                           0.81       240
   macro avg       0.71      0.69      0.70       240
weighted avg       0.81      0.81      0.81       240
```

3.8) Using Decision Tree Classifier

Import the necessary package

In [208]:

```
from sklearn.tree import DecisionTreeClassifier
```

Using train-test split and standard scaler

In [209]:

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=10)
```

In [210]:

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

Define and Train the model

In [211]:

```
model=DecisionTreeClassifier(splitter='best',random_state=10,criterion='gini')
model.fit(X_train,y_train)
print(" Model Feature Importances = " ,model.feature_importances_)
```

```
 Model Feature Importances =  [0.06704579 0.00109898 0.00477187 0.01132664
0.         0.08641163
 0.00238593 0.00290864 0.01423053 0.00902506 0.22243216 0.00617658
 0.00377773 0.         0.00897979 0.01132996 0.00409017 0.20666338
 0.00238593 0.02036627 0.04258213 0.04994978 0.01102692 0.0021587
 0.01818392 0.00835077 0.18234073]
```

Predict the model

In [212]:

```
y_train_predict=model.predict(X_train)
y_predict=model.predict(X_test)
```

Calculating confusion matrix and generating crosstab

In [213]:

```
print(confusion_matrix(y_test,y_predict))
pd.crosstab(y_test,y_predict)
```

```
[[ 35   3   0]
 [  9 168   2]
 [  1   4  18]]
```

Out[213]:

col_0	0	1	2
PerformanceRating			
0	35	3	0
1	9	168	2
2	1	4	18

Calculate the accuracy score, precision score, recall score and F1 score

In [214]:

```
print("Accuracy score of Training = ",accuracy_score(y_train,y_train_predict)*100)
print("Accuracy score of Testing = ",accuracy_score(y_test,y_predict)*100)
print("Precision score = ",(precision_score(y_test,y_predict,average='weighted')*100))
print("Recall score = ",(recall_score(y_test,y_predict,average='weighted')*100))
print("F1 score = ",(f1_score(y_test,y_predict,average='weighted')*100))
```

```
Accuracy score of Training =  100.0
Accuracy score of Testing =  92.08333333333333
Precision score =  92.53981481481482
Recall score =  92.08333333333333
F1 score =  92.16762992054936
```

Generating classification report

In [215]:

```
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       0.78      0.92      0.84        38
           1       0.96      0.94      0.95       179
           2       0.90      0.78      0.84        23

    accuracy                           0.92       240
   macro avg       0.88      0.88      0.88       240
weighted avg       0.93      0.92      0.92       240
```

--

Downloading the final processed data

In [216]:

```
performance.to_excel("INX_Future_Inc_Employee_Performance_Final_Processed_Data.xls")
```

In [218]:

```
performance.head()
```

Out[218]:

	EmpNumber	Age	Gender	EducationBackground	MaritalStatus	EmpDepartment	EmpJobRo
0	E1001000	32	Male	Marketing	Single	Sales	Sale Executi\
1	E1001006	47	Male	Marketing	Single	Sales	Sale Executi\
2	E1001007	40	Male	Life Sciences	Married	Sales	Sale Executi\
3	E1001009	41	Male	Human Resources	Divorced	Human Resources	Manag
4	E1001010	60	Male	Marketing	Single	Sales	Sale Executi\

5 rows × 28 columns