

# Employee Promotion Prediction

## Predicting whether an employee is promoted or not based on various factors

The aim is to analyze the various factors that can contribute to the promotion of an employee. Based on the analysis, predict which employees will be promoted.

The following details for an employee is given in the dataset :-

- Department - department of the employee
- Region - region as designated by the company
- Education - qualification of the employee
- Gender - gender of the employee
- Recruitment channel - means via which employee was recruited
- No of trainings - total number of trainings undergone by the employee
- Age - age of the employee
- Previous year ratings - previous year performance ratings of the employee
- Length of service - total years worked for the company
- KPIs met(1 if >80%) - total KPIs met in the tenure
- Average training score - average score on trainings
- Awards won - Awards won if any

The target column is the *is\_promoted* column. The column is binary and specifies whether the employee was promoted or not.

```
In [36]: import pandas as pd
from pandas.api.types import CategoricalDtype
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, cross_validate, validation_curve
from imblearn.over_sampling import RandomOverSampler
from sklearn.metrics import classification_report
```

```
C:\Users\richa\AppData\Local\Temp\ipykernel_20512\4008325993.py:18: UserWarning: DelftStack
ack
warnings.warn('DelftStack')
```

```
In [37]: df = pd.read_csv("Promotion.csv")
df.head(5)
```

Out[37]:

	employee_id	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_
0	65438	Sales & Marketing	region_7	Master's & above	f	sourcing	1	35	
1	65141	Operations	region_22	Bachelor's	m	other	1	30	
2	7513	Sales & Marketing	region_19	Bachelor's	m	sourcing	1	34	
3	2542	Sales & Marketing	region_23	Bachelor's	m	other	2	39	
4	48945	Technology	region_26	Bachelor's	m	other	1	45	

Right off the bat, some column names have to be changed for ease of access. The following colum names have been changed :-

- KPIs\_met >80% - KPIs\_met
- awards\_won? - awards\_won

In [38]:

```
cols = ['employee_id', 'department', 'region', 'education', 'gender',
        'recruitment_channel', 'no_of_trainings', 'age', 'previous_year_rating',
        'length_of_service', 'KPIs_met', 'awards_won',
        'avg_training_score', 'is_promoted']
df.columns = cols
```

In [39]:

```
df.describe()
```

Out[39]:

	employee_id	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_met	awa
count	54808.000000	54808.000000	54808.000000	50684.000000	54808.000000	54808.000000	5480
mean	39195.830627	1.253011	34.803915	3.329256	5.865512	0.351974	
std	22586.581449	0.609264	7.660169	1.259993	4.265094	0.477590	
min	1.000000	1.000000	20.000000	1.000000	1.000000	0.000000	
25%	19669.750000	1.000000	29.000000	3.000000	3.000000	0.000000	
50%	39225.500000	1.000000	33.000000	3.000000	5.000000	0.000000	
75%	58730.500000	1.000000	39.000000	4.000000	7.000000	1.000000	
max	78298.000000	10.000000	60.000000	5.000000	37.000000	1.000000	

In [40]:

```
df.isnull().sum()
```

Out[40]:

employee_id	0
department	0
region	0
education	2409
gender	0
recruitment_channel	0
no_of_trainings	0
age	0
previous_year_rating	4124
length_of_service	0
KPIs_met	0
awards_won	0
avg_training_score	0
is_promoted	0
dtype:	int64

The `describe()` method gives a general view of all the numerical columns in the dataset.

The `isnull().sum()` chaining counts the number of Null values per column.

Only the *education* and *previous\_year\_rating* columns have Null values in them. The *previous\_year\_rating* column gives the rating employee receives each year on the Likert scale (here 1 to 5). A new employee joining in the current year would not have a previous year rating, to prove this assumption, the previous year rating which are Null are compared to the *length\_of\_service* column which gives the number of years employee has now worked for the company. The length of service has the minimum value of 1, this implies that any new joiner is also by default said to have worked for the company for one year.

```
In [41]: df.previous_year_rating.value_counts(dropna=False)
```

```
Out[41]: 3.0    18618
         5.0    11741
         4.0     9877
         1.0     6223
         2.0     4225
         NaN     4124
         Name: previous_year_rating, dtype: int64
```

```
In [42]: service_filter = df[df.length_of_service == 1]
         print(
             "Null rating counts of employees with length of service 1\n",
             service_filter.previous_year_rating.isnull().sum()
         )
         print(
             "Null rating counts of employees with length of service 1 and promoted\n",
             service_filter[service_filter.is_promoted == 1].previous_year_rating.isnull().sum()
         )
         df.previous_year_rating = df.previous_year_rating.fillna(0)
```

```
Null rating counts of employees with length of service 1
4124
```

```
Null rating counts of employees with length of service 1 and promoted
339
```

The 4124 Null values were all new employees. From these, 339 employees have actually received a promotion. This reason makes a compelling argument not to remove these rows. Thus the *previous\_year\_ratings* are filled with 0s instead, making it the lowest score as well as indicating absence of value (rating) all together.

It is odd that the *education* column has so many Null values. Qualifications mean a great deal and hence it is imperative the Null values are analyzed. Out of the total 2409 missing rows, 122 of those employees have received a promotion, thus the Null values have to be imputed.

```
In [43]: df[df.education.isna()].is_promoted.value_counts()
```

```
Out[43]: 0    2287
         1     122
         Name: is_promoted, dtype: int64
```

One solution to impute the *education* column is to fill the Null values with the mode of the column. The mode of the column gives the qualification that is most frequent among the employees. The mode is a good option since during hiring process, the company looks into the employee's qualifications and hence recruits them. It can be assumed that every department has a different criterion for qualification, for example An engineer hired would mostly be a *Bachelor's* whereas an employee working in management would mostly be a *Master's*.

Analyzing the *Department* column based on the above argument, there are 9 unique departments in the company.

```
In [44]: depts = df.department.unique()

for dept in depts:
    edu = df[df.department == dept].education.mode()[0]
    print(dept, " : ", edu)
```

```
Sales & Marketing : Bachelor's
Operations : Bachelor's
Technology : Bachelor's
Analytics : Bachelor's
R&D : Bachelor's
Procurement : Bachelor's
Finance : Bachelor's
HR : Bachelor's
Legal : Bachelor's
```

The above arguments are not in line with the data. Hence simply filling the education with *Bachelor's* would suffice for the analysis.

```
In [45]: df.education = df.education.fillna(df.education.mode()[0])
```

Viewing the number of unique values in each column gives a good idea as to which columns are categorical and which are continuous.

```
In [46]: df.nunique()
```

```
Out[46]: employee_id      54808
department           9
region              34
education            3
gender              2
recruitment_channel  3
no_of_trainings     10
age                 41
previous_year_rating  6
length_of_service   35
KPIs_met            2
awards_won          2
avg_training_score   61
is_promoted         2
dtype: int64
```

The dataset seems clean with all the Null values removed. The aim of the project is to analyze which factors contribute to an employee's promotion. Over the next few markdowns, a detailed analysis has been done for each column that seems relevant.

Firstly looking into two columns *awards\_won* and *KPIs\_met*. These columns directly describe the performance of the employee. KPIs are Key Performance Indicators that evaluate their success at reaching targets. In this case, any employee with > 80% success rate at reaching the target has a value of 1 otherwise a value of 0. Similarly, *awards\_won* is 1 if the employee has won an award from the company.

```
In [47]: plt.style.use('fivethirtyeight')
plt.subplots(figsize=(16,8))
sns.set_style('dark')
plt.subplot(1,2,1)
```

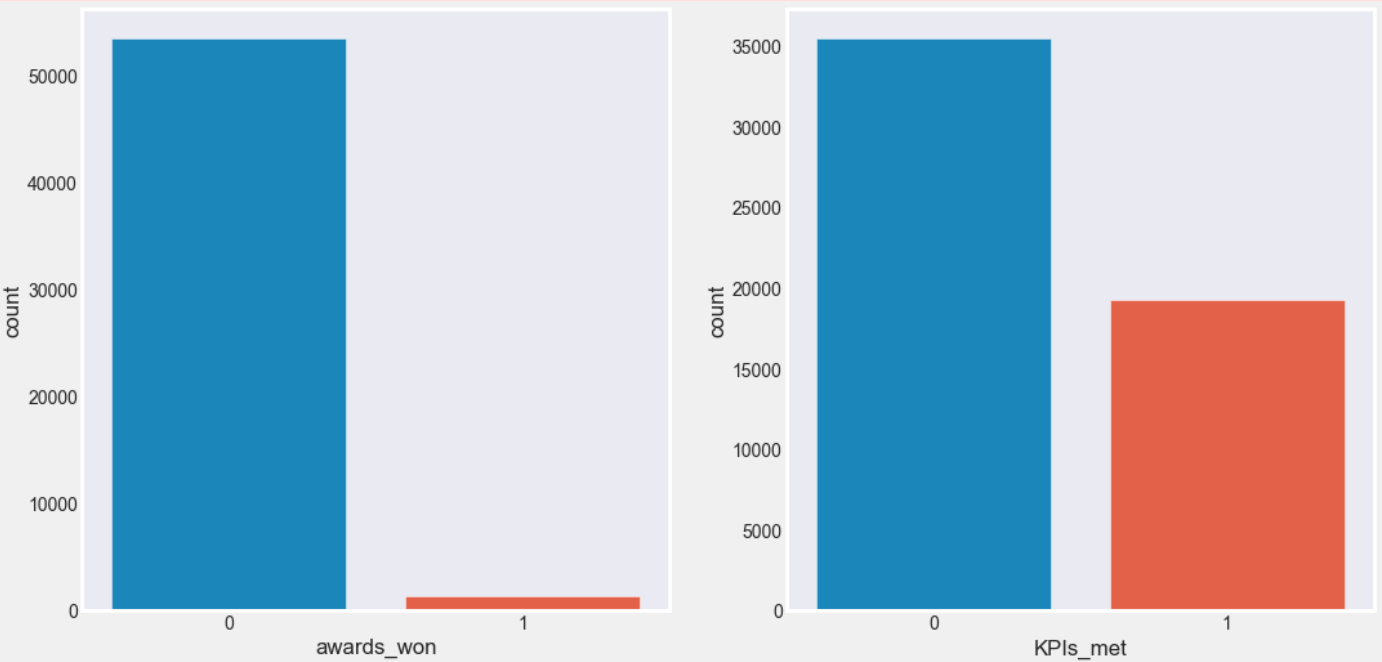
```
sns.countplot(df['awards_won'])
plt.subplot(1,2,2)
sns.countplot(df['KPIs_met'])
plt.show()
```

C:\Users\richa\anaconda3\lib\site-packages\seaborn\\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

```
warnings.warn(
```

C:\Users\richa\anaconda3\lib\site-packages\seaborn\\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

```
warnings.warn(
```



A very small set of employees have actually won awards, around 1270 people. Whereas a good percentage of people have greater than 80% success rate on reaching targets (KPIs). A closer look into the employees having both won and award and achieved targets or either of the two.

```
In [48]: star_ems = df[(df.KPIs_met == 1) & (df.awards_won == 1)]
```

```
plt.style.use('fivethirtyeight')
plt.subplots(figsize=(12,8))
plt.subplot(2,2,1)
sns.countplot(df[df.is_promoted == 1].awards_won)
plt.subplot(2,2,2)
sns.countplot(df[df.is_promoted == 1].KPIs_met)
plt.subplot(2,2,3)
sns.countplot(star_ems.is_promoted)
```

```
C:\Users\richa\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
s the following variable as a keyword arg: x. From version 0.12, the only valid position
al argument will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
```

```
warnings.warn(
```

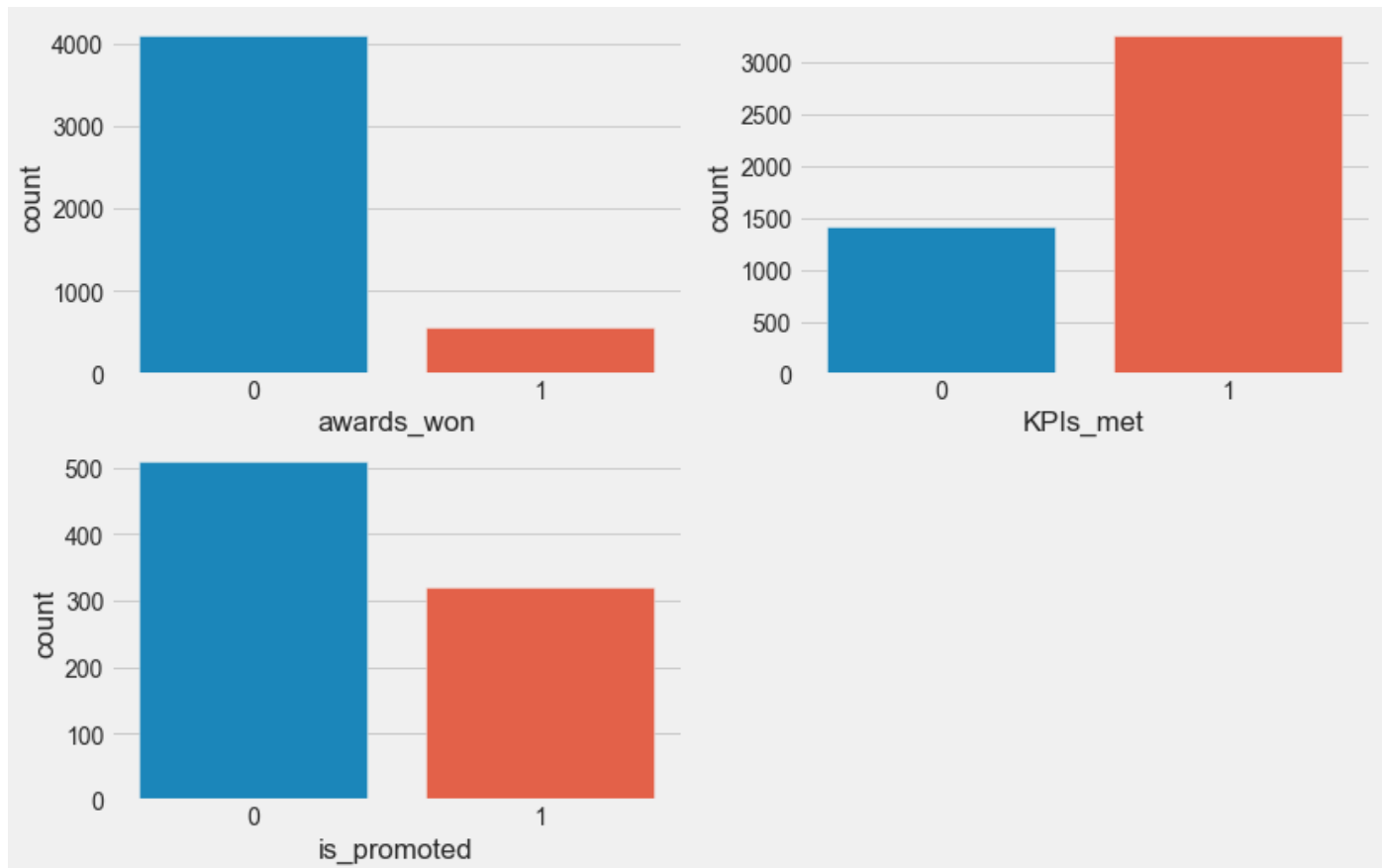
```
C:\Users\richa\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
s the following variable as a keyword arg: x. From version 0.12, the only valid position
al argument will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
```

```
warnings.warn(
```

```
C:\Users\richa\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
s the following variable as a keyword arg: x. From version 0.12, the only valid position
al argument will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
```

```
warnings.warn(
```

```
Out[48]: <AxesSubplot:xlabel='is_promoted', ylabel='count'>
```



The *KPIs\_met* and *awards\_won* columns are inconclusive individually in the count plots. For the above plots - the first two plots convey the distribution of *awards\_won* (left) and *KPIs\_met* (right) respectively for the employees who were promoted. The third (bottom) plot shows the distribution of *is\_promoted* for all employees who have both won and award and met the KPIs. The conclusions drawn :-

- A less percentage of employees who were promoted have won award.
- An employee who meets the targets has a high chance of promotion
- Employees who have both the award and met targets are likely to be promoted.

For the ease of analysis, the two columns are combined into a single column *good\_performance* using the `any()` function. Basically any employee who has either won an award or has met KPIs has shown good performance.

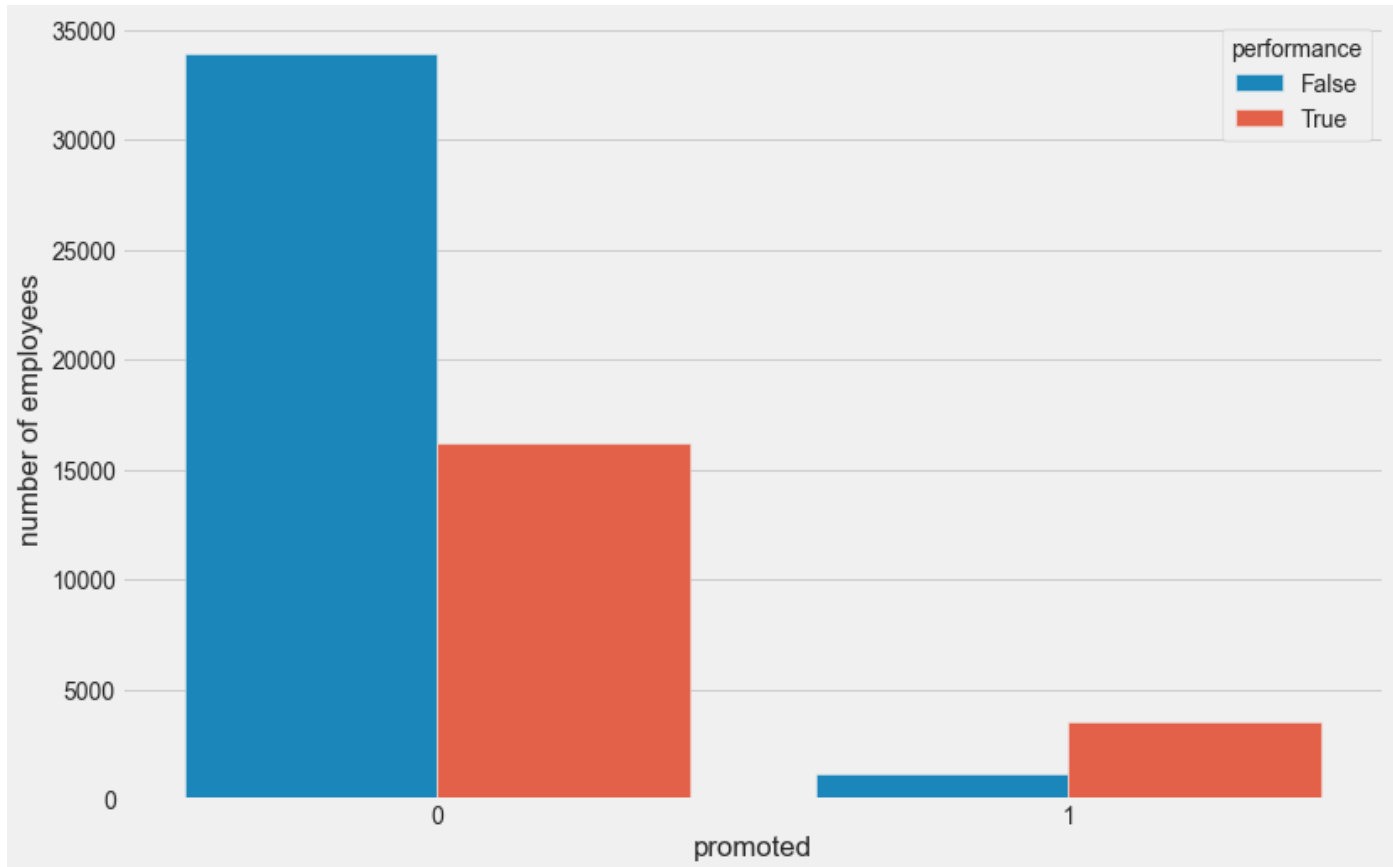
```
In [49]: df['performance'] = df[['KPIs_met', 'awards_won']].any(axis=1, skipna=False)
```

```
plt.figure(figsize=(12,8))
sns.countplot(df.is_promoted,hue=df.performance)
plt.xlabel('promoted')
plt.ylabel('number of employees')
```

C:\Users\richa\anaconda3\lib\site-packages\seaborn\\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

```
warnings.warn(
```

Out[49]: Text(0, 0.5, 'number of employees')



The plot above compares counts of performance of employees who were promoted and performance of employees who were not promoted. The plot draws the following conclusions :-

- Out of the employees who were promoted, most of them had shown good performance.
- The employees who were not promoted have a high number of non-performing employees.

There exist a great number of employees who had performed but yet were not promoted. This could be due to various other reasons. This proves a compelling argument to analyze the other factors as well.

Various companies have been reported to be gender bias. It is good to be sure if such case exists for the company under analysis, this can gives markers for predictions later on. For the dataset under analysis, the number of male employees are more than double the female employees.

```
In [50]: df.gender.value_counts(normalize=True)
```

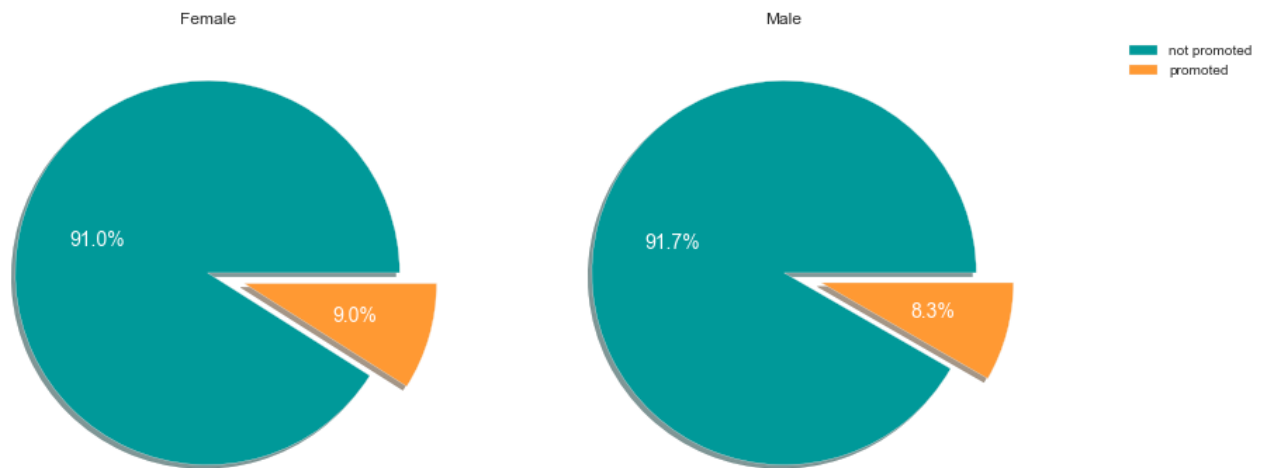
```
Out[50]: m    0.702379
         f    0.297621
         Name: gender, dtype: float64
```

```
In [51]: plt.style.use('seaborn')
         plt.subplots(figsize=(12,8))
```

```

plt.pie(
    x=df[df.gender=='f'].is_promoted.value_counts(normalize=True),
    labels=['not promoted', 'promoted'],
    explode=[0, 0.2],
    autopct="%1.1f%%",
    shadow=True,
    textprops=dict(color='w', fontsize=14),
    colors=['#009999', '#ff9933']
)
plt.title("Female")
plt.subplot(1, 2, 2)
plt.pie(
    x=df[df.gender=='m'].is_promoted.value_counts(normalize=True),
    labels=['not promoted', 'promoted'],
    explode=[0, 0.2],
    autopct="%1.1f%%",
    shadow=True,
    textprops=dict(color='w', fontsize=14),
    colors=['#009999', '#ff9933']
)
plt.title("Male")
plt.legend(['not promoted', 'promoted'], loc='upper right', bbox_to_anchor=(1, 0.5, 0.5, 0
Out[51]: <matplotlib.legend.Legend at 0x193328a7f70>

```



Opposite to the assumption, the females have more promotions as compared to the males. The pie charts concludes that the two genders have equal proportions of promotions. This does not mean that equal number of females and males were promoted. As established earlier, the population of males is far greater than female. The proportions calculated are with respect to their population.

The columns *no\_of\_trainings* and *avg\_training\_score* specify the number of company organized workshops and trainings has the employee attended and the average training score the employee recieved for the said trainings. Trainings and workshops are integral for an employee as they are hosted for skill development of the employees. These trainings scores help the company understand which employees are improving. The two columns individually do not provide a good assessment since they cannot be compared between employees.

Say employee A scored an average training score of 60 but has attended only one workshop, and employee B achieves an average score of 50 but over 3 workshops. From the current perspective, employee A has a better score if the comparision is made on *avg\_training\_score*, but in reality, employee B has amassed a total of 150 training points whereas employee A has only 60.

Another case, employee A has undergone only 1 training session and recieved an average training score of



100. Employee B underwent 3 training sessions and recieved an average training score of 25. By comparison on *no\_of\_trainings* employee A trumps employee B. In reality employee A has a total score of 100 whereas employee B has just 75. Number of trainings can be less for a recently joined employee as compared to a veteran in the company.

The two above scenarios give rise to a third column - *total\_score*, which gives an estimate of the total score recieved by an employee. This column is a good factor to compare and differentiate between employees who have shown improvement and those who havent.

```
In [52]: df['total_score'] = df.no_of_trainings * df.avg_training_score
df.head(5)
```

```
Out[52]:
```

	employee_id	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_
0	65438	Sales & Marketing	region_7	Master's & above	f	sourcing	1	35	
1	65141	Operations	region_22	Bachelor's	m	other	1	30	
2	7513	Sales & Marketing	region_19	Bachelor's	m	sourcing	1	34	
3	2542	Sales & Marketing	region_23	Bachelor's	m	other	2	39	
4	48945	Technology	region_26	Bachelor's	m	other	1	45	

The *total\_score* column is on the ratio scale i.e. is numeric. The aim is to compare the *total\_score* with the *is\_promoted* column to identify if any relationships exist. For the purpose the *total\_score* column is divided into 3 bins (categories) :-

- Low - 65 or lower
- Mediocre - 65 to 145 points
- High - 145 or higher

The bins have been selected based on the distribution of the *total\_score* column for the employees that were promoted. Using the `pd.cut()` functions the column is split into bins and stored in *total\_score\_label* column for further analysis.

```
In [53]: df['total_score_label'] = pd.cut(df.total_score, bins=[0, 65, 145, 1000], labels=['Low', 'Medi']
df.head(5)
```

```
Out[53]:
```

	employee_id	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_
0	65438	Sales & Marketing	region_7	Master's & above	f	sourcing	1	35	
1	65141	Operations	region_22	Bachelor's	m	other	1	30	
2	7513	Sales & Marketing	region_19	Bachelor's	m	sourcing	1	34	
3	2542	Sales & Marketing	region_23	Bachelor's	m	other	2	39	
4	48945	Technology	region_26	Bachelor's	m	other	1	45	

```
In [54]: scores = df.pivot_table(values='is_promoted', index='total_score_label')
scores
```

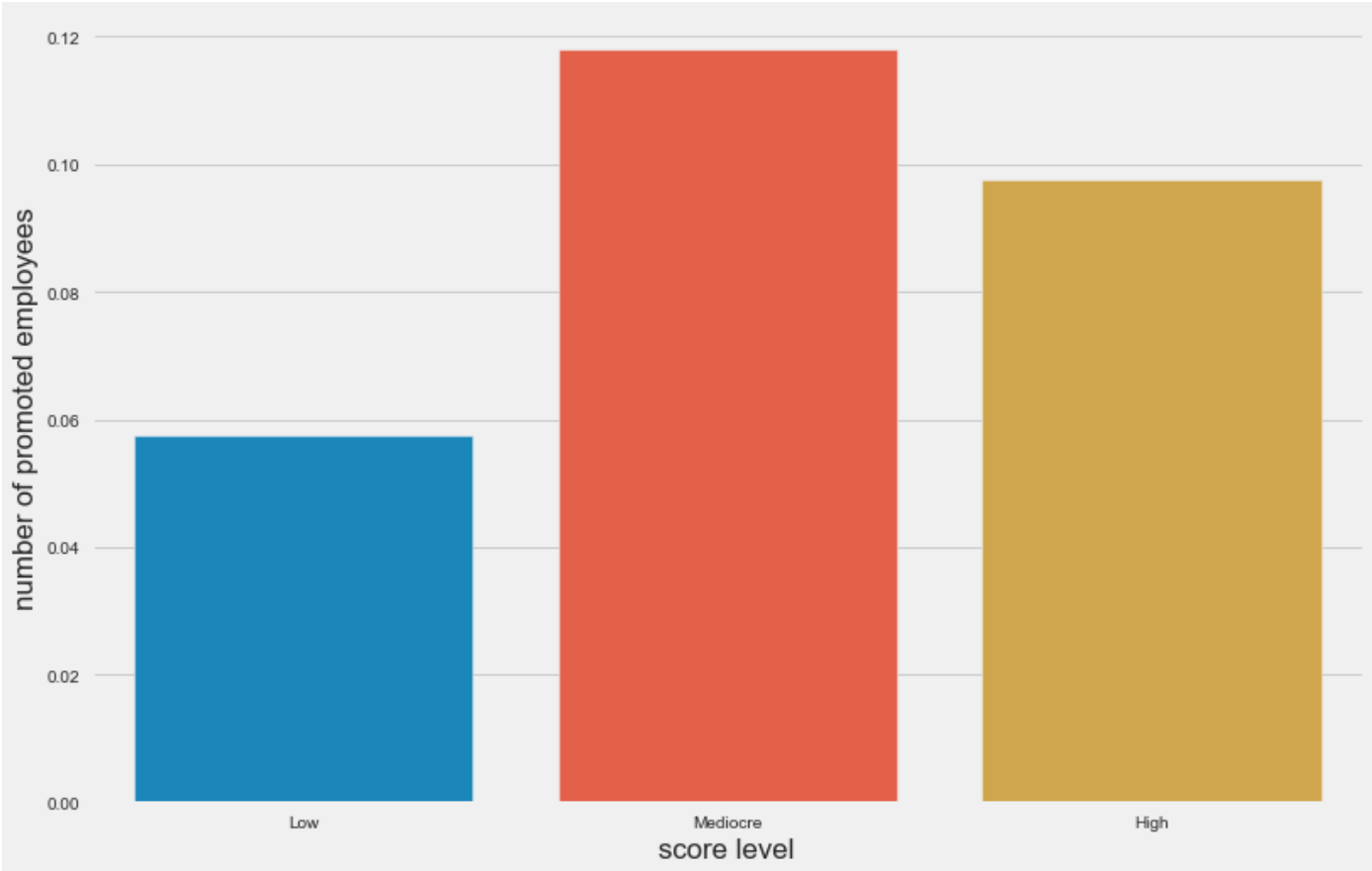
Out[54]:

	is_promoted
Low	0.057549
Mediocre	0.117858
High	0.097444

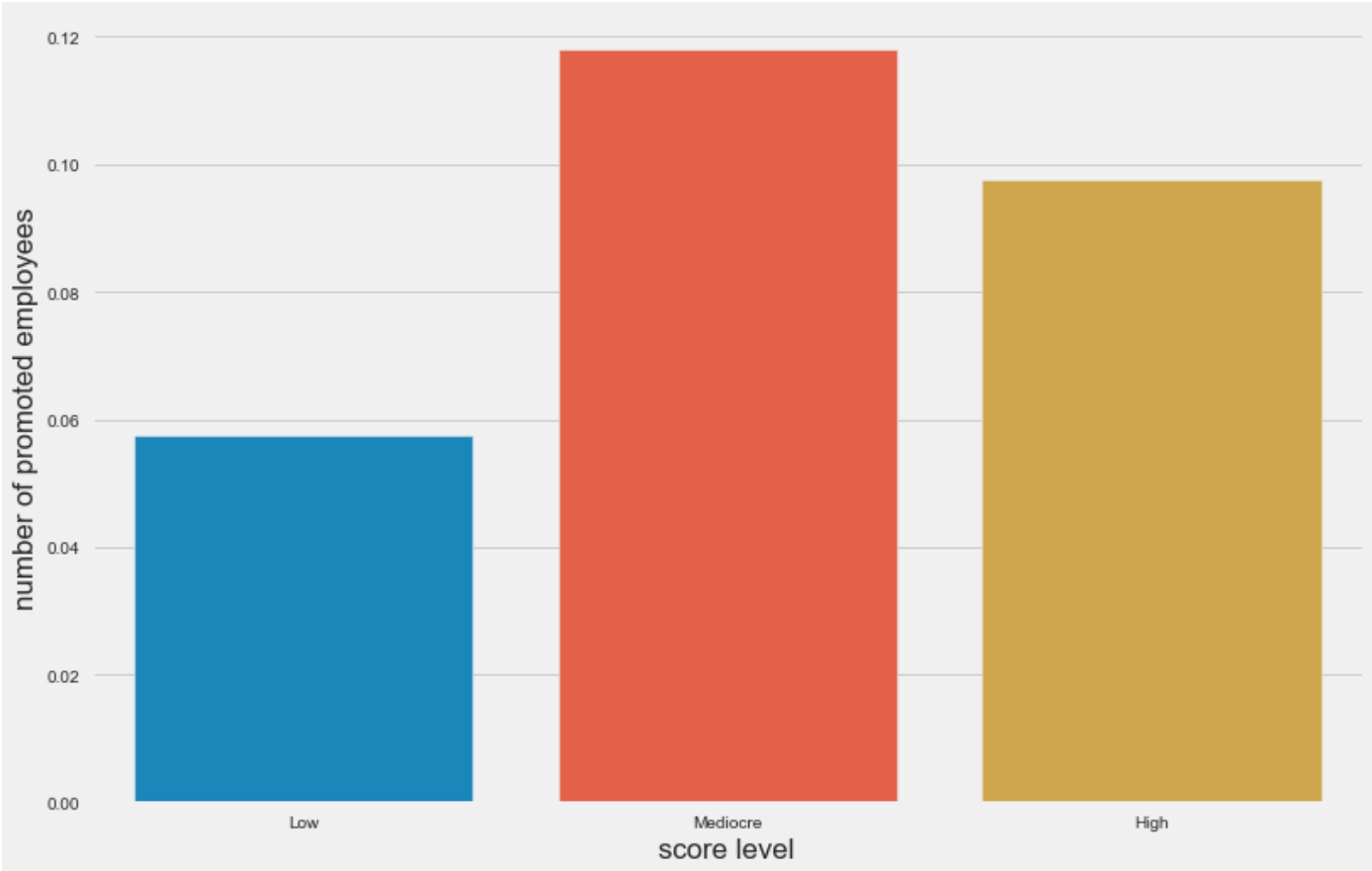
total_score_label	
Low	0.057549
Mediocre	0.117858
High	0.097444

```
In [55]: plt.style.use('fivethirtyeight')
plt.figure(figsize=(12,8))
sns.barplot(
    x=scores.index,
    y=scores.is_promoted
)
plt.ylabel('number of promoted employees')
plt.xlabel('score level')
```

Out[55]:



score level	number of promoted employees
Low	0.057549
Mediocre	0.117858
High	0.097444



The following conclusions can be drawn from the plot :-

- Employees promoted have scores in the Mediocre and High range i.e. 65 and greater
- Employees with scores in the Low range have also been promoted a good percentage of times.
- Mediocre score level having the highest percentage of promoted employees means total score is not the only criterion for promotion.

A similar approach is taken for the *length\_of\_service* column. An employee who has been in a company longer is more likely to be promoted rather than a new joinee. Since the column is numeric, it is converted to categorical via binning into the following categories.

- New - 0 to 2 years
- - 2 to 7 years

- Experienced - 7 to 10 years
- Veteran - 10 years or more

The categories have been chosen based on general trend. (Logic)

```
In [56]: df['service_catg'] = pd.cut(
    df.length_of_service,
    bins=[0,2,7,10,37],
    labels=['New', 'Established', 'Experienced', 'Veteran']
)
df.head(5)
```

```
Out[56]:
```

	employee_id	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_
0	65438	Sales & Marketing	region_7	Master's & above	f	sourcing	1	35	
1	65141	Operations	region_22	Bachelor's	m	other	1	30	
2	7513	Sales & Marketing	region_19	Bachelor's	m	sourcing	1	34	
3	2542	Sales & Marketing	region_23	Bachelor's	m	other	2	39	
4	48945	Technology	region_26	Bachelor's	m	other	1	45	

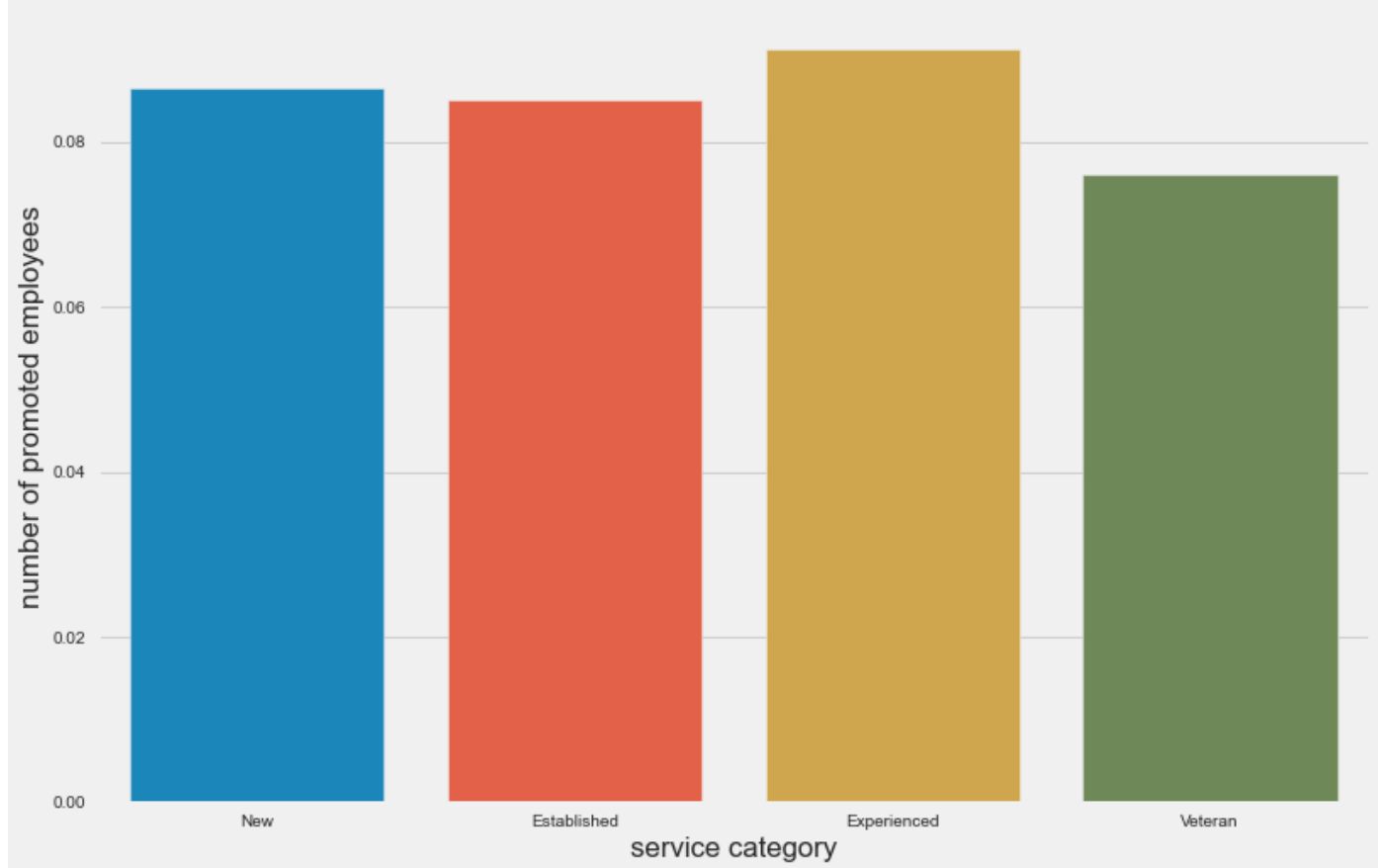
```
In [57]: service = df.pivot_table(values='is_promoted', index='service_catg')
service
```

```
Out[57]:
```

	is_promoted
service_catg	
New	0.086546
Established	0.084940
Experienced	0.091110
Veteran	0.075943

```
In [58]: plt.style.use('fivethirtyeight')
plt.figure(figsize=(12,8))
sns.barplot(
    x=service.index,
    y=service.is_promoted
)
plt.ylabel('number of promoted employees')
plt.xlabel('service category')
```

```
Out[58]: Text(0.5, 0, 'service category')
```



The plot above concludes that :-

- Experienced employees are likely to get promoted than other categories, due to their experience and understanding of the company
- New and Established employees have almost equal likelihood of getting promoted.
- Veteran employees generally get less promotions. Due to their number of years given to the company, majority of them might have reached the pinnacle.

The *age* column is very similar to the service length, in terms that both define a temporal factor for the employee. The column is numeric. Assuming that an employee's career starts at the age of 20, and hence is binned into the following categories :-

- Young - Less than 25 years
- Middle - 25 to 40 years
- Senior - 40 to 50 years
- Elder - 50 years or older

On these categories, the percentage of people who were promoted from each category is calculated.

```
In [59]: df['age_label'] = pd.cut(df.age, bins=[0, 25, 40, 50, 100], labels=['Young', 'Middle', 'Senior',  
df.head(5)
```

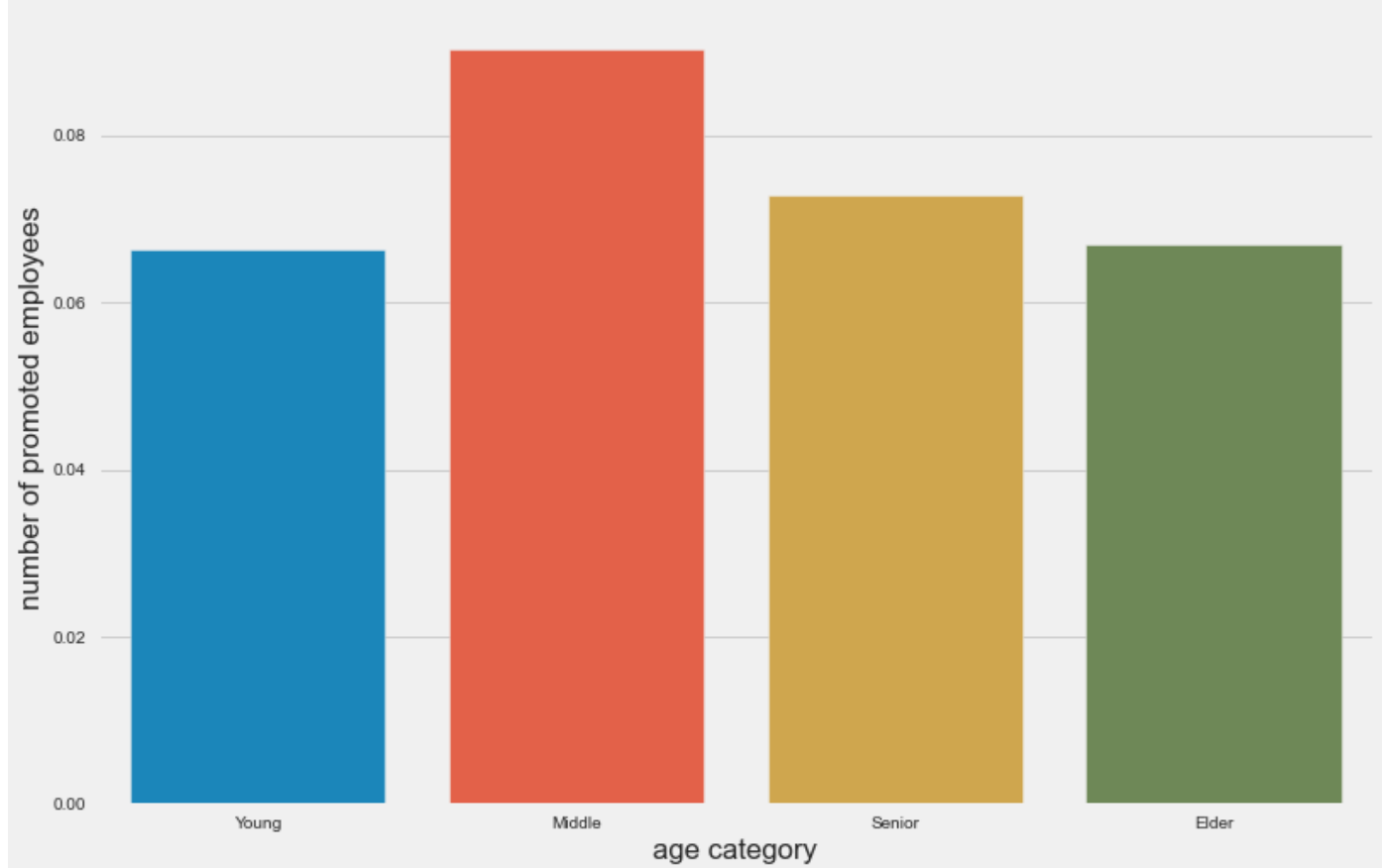
Out[59]:	employee_id	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_
0	65438	Sales & Marketing	region_7	Master's & above	f	sourcing	1	35	
1	65141	Operations	region_22	Bachelor's	m	other	1	30	
2	7513	Sales & Marketing	region_19	Bachelor's	m	sourcing	1	34	
3	2542	Sales & Marketing	region_23	Bachelor's	m	other	2	39	
4	48945	Technology	region_26	Bachelor's	m	other	1	45	

```
In [60]: ages = df.pivot_table(values='is_promoted', index='age_label')
ages
```

Out[60]:	is_promoted
age_label	
Young	0.066357
Middle	0.090173
Senior	0.072727
Elder	0.066971

```
In [61]: plt.style.use('fivethirtyeight')
plt.figure(figsize=(12,8))
sns.barplot(
    x=ages.index,
    y=ages.is_promoted
)
plt.ylabel('number of promoted employees')
plt.xlabel('age category')
```

```
Out[61]: Text(0.5, 0, 'age category')
```



The plot clearly states the following :-

- An employee aged between 25 and 40 years is likely to be promoted.
- A senior employee aged 40 to 50 years has a good chance of getting promoted.
- A Middle aged or Senior employee form the perfect age interval (25 to 50) where an employee is said to at the peak of his/her career.
- Younger or Elder employees have lesser chances of being promoted as the former has just begun their career and the latter is coming towards an end to their career.

The *service\_catg* shows that a *New* employee is also likelier to get promoted. A *New* employee is not necessarily a *Young* employee. Since *age\_label* has been defined, this can be analyzed.

```
In [62]: new_emps = df[df.service_catg == 'New']

new_age = new_emps.pivot_table(values='is_promoted', index='age_label')
new_age
```

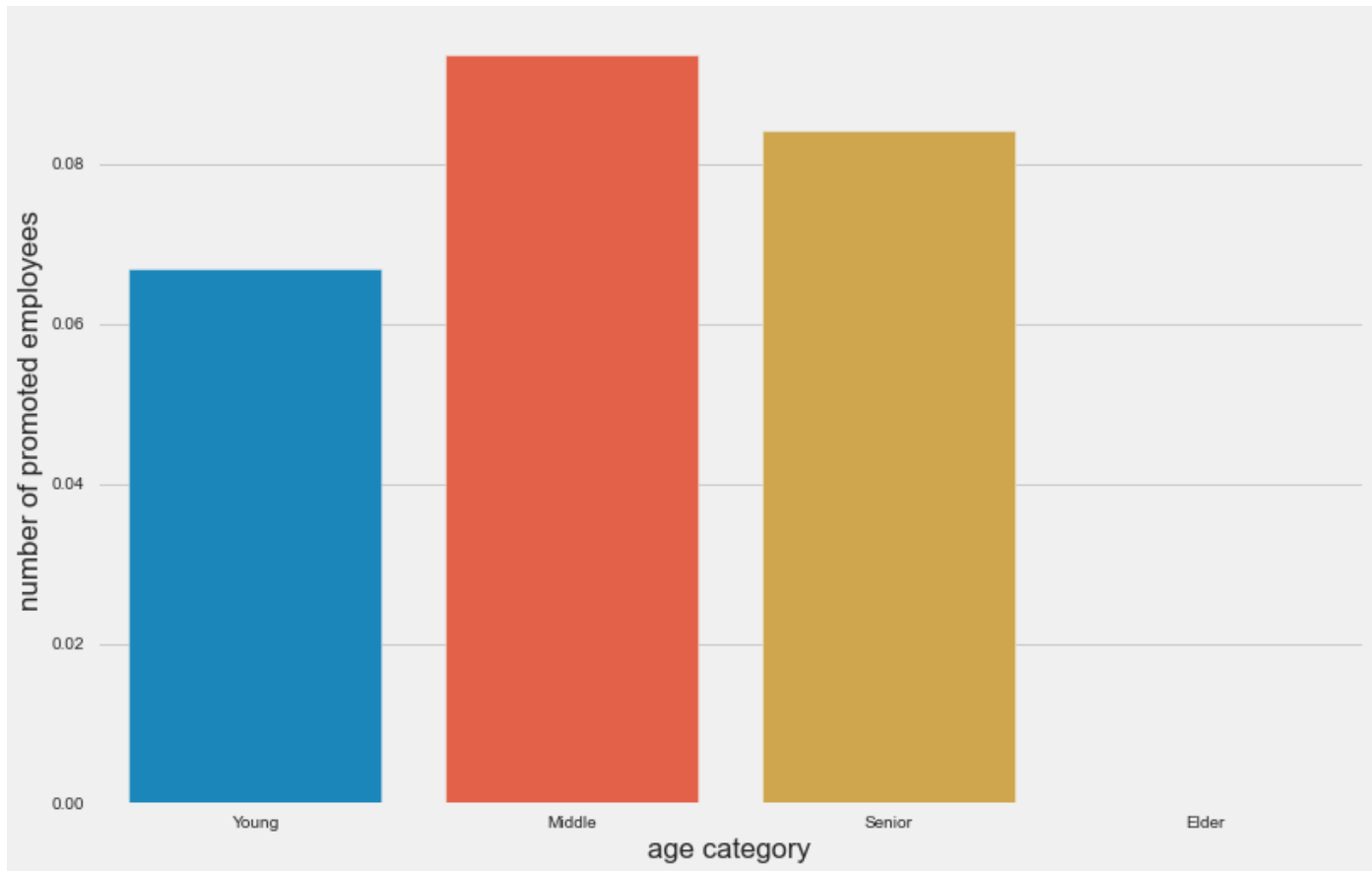
Out[62]:

is_promoted	
age_label	
Young	0.066873
Middle	0.093564
Senior	0.084112
Elder	0.000000

```
In [63]: plt.style.use('fivethirtyeight')
plt.figure(figsize=(12,8))
sns.barplot(
    x=new_age.index,
```

```
)
plt.ylabel('number of promoted employees')
plt.xlabel('age category')
```

Out[63]: Text(0.5, 0, 'age category')



The plot is conclusive of the facts :-

- No elder employees are new joiners, this is due to the fact that their end of career is nearing.
- Among the new joiners, the Middle aged and Senior employees are likelier to be promoted.
- The Young employees being promoted, means there must be a few other factors discussed above, that must be great for them.

The *previous\_year\_rating* describes the rating an employee received in the internal evaluations of the company in the previous year. These ratings give a clear differentiation between employees. By intuition, an employee with a good rating paired with other factors is likelier to be promoted. To prove this hypothesis, the *previous\_year\_rating* column is first encoded into labels for understanding.

- New - rating 0, new employee
- Minimum - rating 1
- Fair - rating 2
- Improving - rating 3
- Good - rating 4
- Very good - rating 5

NOTE: These rating labels are chosen to be subtle about it.

```
In [64]: def decode(val):
          if val == 0:
              return 'New'
          elif val == 1:
```

```

        return 'Minimum'
    elif val == 2:
        return 'Fair'
    elif val == 3:
        return 'Improving'
    elif val == 4:
        return 'Good'
    else:
        return 'Very good'

```

```

df['rating_label'] = df.previous_year_rating.apply(decode)
df.head(5)

```

```

Out[64]:

```

	employee_id	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_
0	65438	Sales & Marketing	region_7	Master's & above	f	sourcing	1	35	
1	65141	Operations	region_22	Bachelor's	m	other	1	30	
2	7513	Sales & Marketing	region_19	Bachelor's	m	sourcing	1	34	
3	2542	Sales & Marketing	region_23	Bachelor's	m	other	2	39	
4	48945	Technology	region_26	Bachelor's	m	other	1	45	

```

In [65]: ratings = df.pivot_table(values='is_promoted', index='rating_label')
ratings

```

```

Out[65]:

```

	is_promoted
rating_label	
Fair	0.042840
Good	0.079376
Improving	0.072779
Minimum	0.014141
New	0.082202
Very good	0.163615

```

In [66]: plt.style.use('fivethirtyeight')
plt.figure(figsize=(12,8))
sns.barplot(
    x=ratings.index,
    y=ratings.is_promoted
)
plt.ylabel('number of promoted employees')
plt.xlabel('ratings')

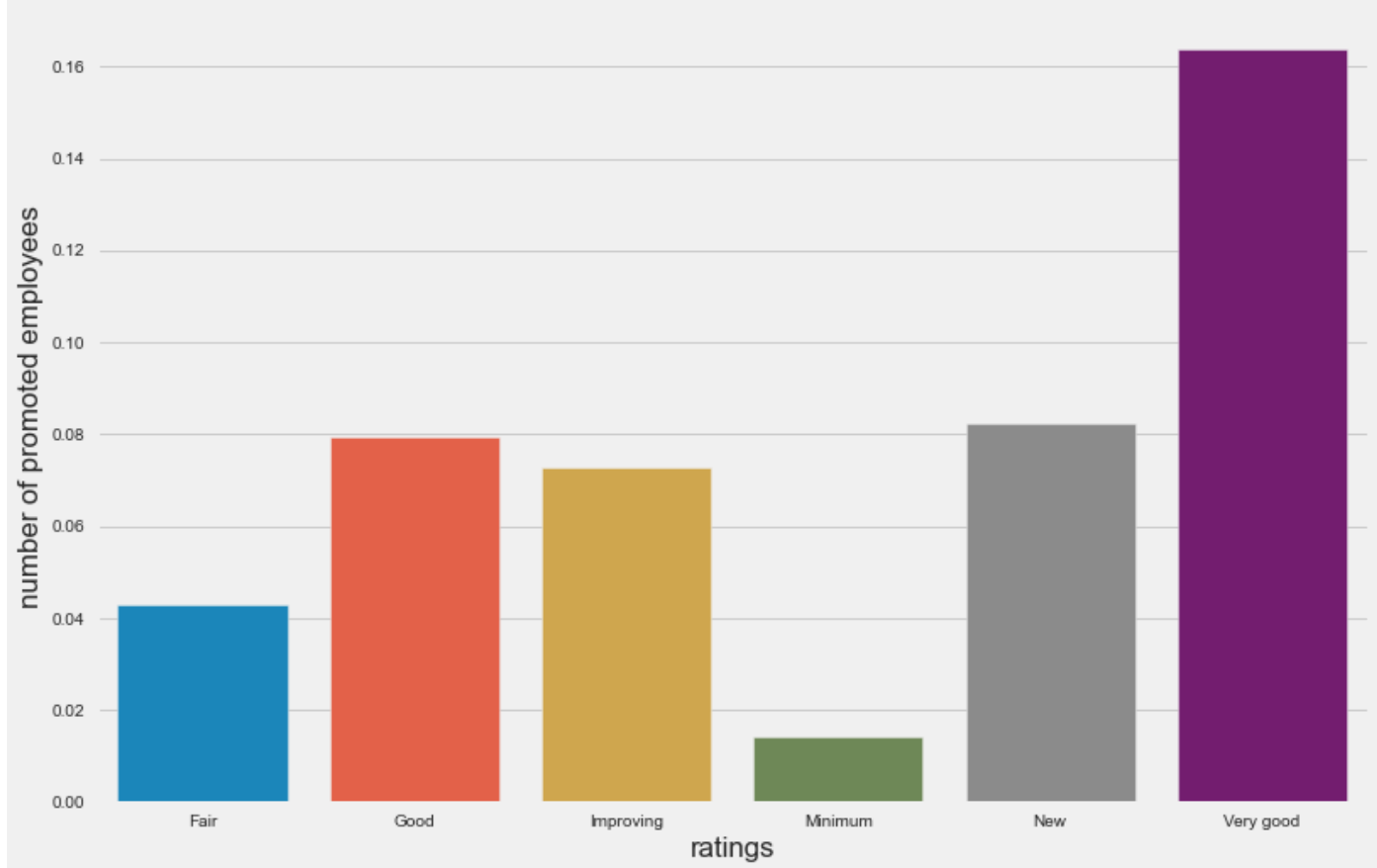
```

```

Out[66]: Text(0.5, 0, 'ratings')

```





The plot proves the hypothesis coined earlier.

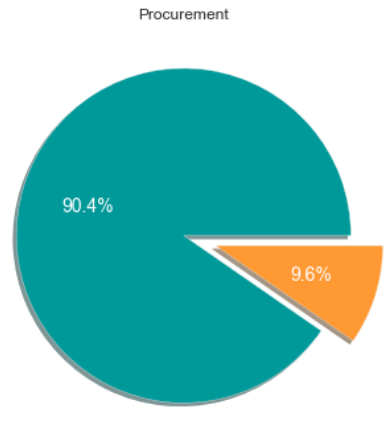
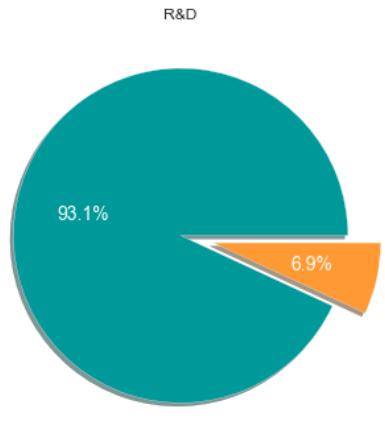
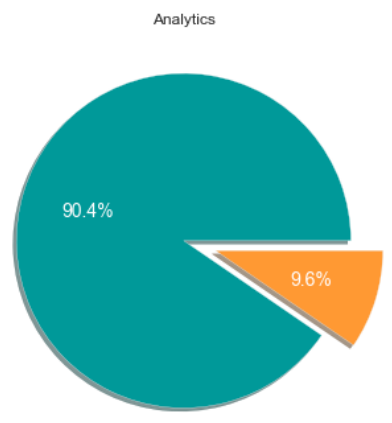
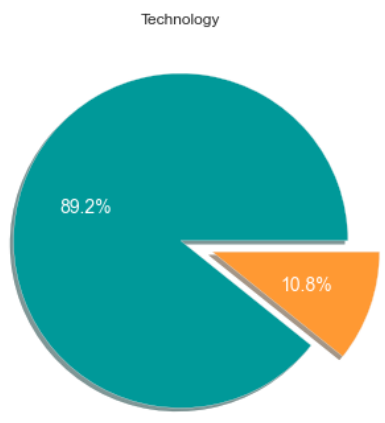
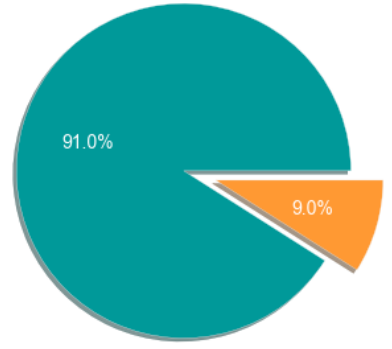
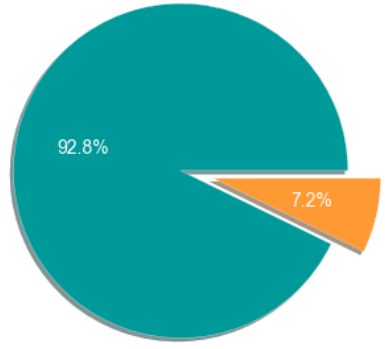
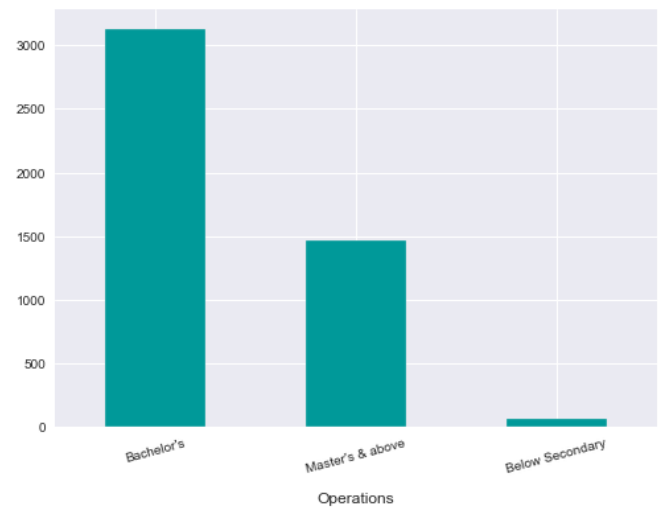
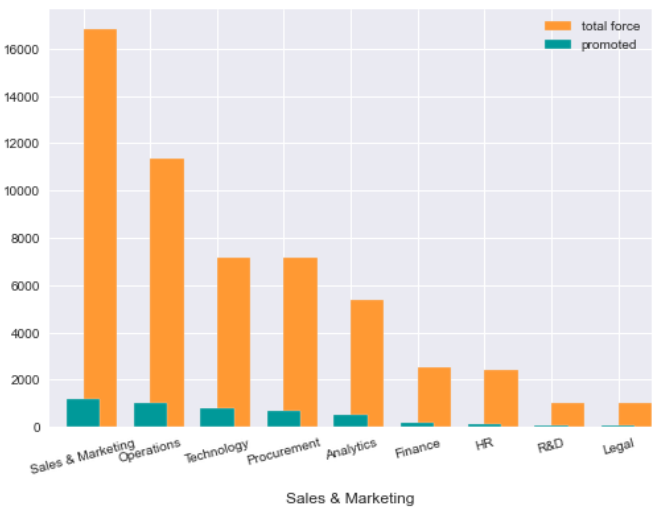
- Employees with 'Very good' rating (5) are most likely to be promoted.
- The trend in the number of people promoted is upwards from 'Minimum' to 'Very good' rating.
- 'New' (no previous year rating) employees shows good percentage of people being promoted. For these employees, other factors are dominant.

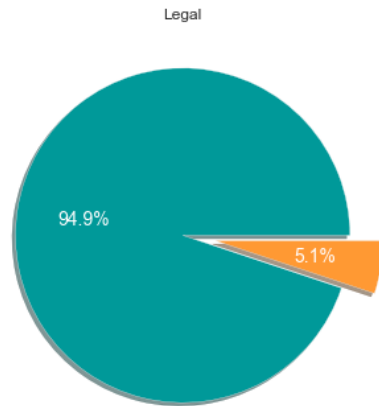
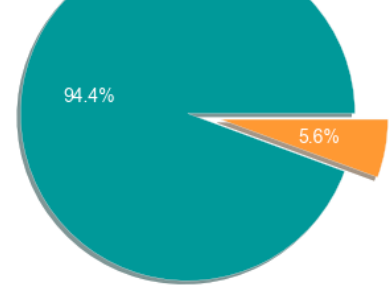
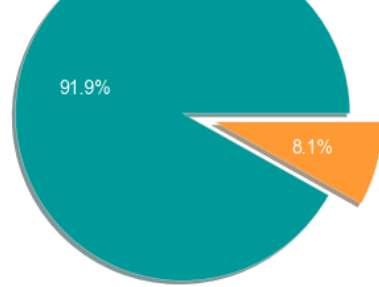
The last columns to analyze are the *department* and *education*. Intuitively the education of an employee is very important when recruiting, but once the employee has joined, the performance within the company is what should matter. Promotions happen within each departments, meaning a sales employee is promoted but stays in the sales department. This can be considered to generally true, with ofcourse a few exceptions. By these arguments, the two columns offer less inuition when analyzing which factors affect an employee's promotion.

```
In [67]: plt.style.use('seaborn')
fig, ax = plt.subplots(2,2,figsize=(16,38))
plt.subplot(6,2,1)
df.department.value_counts().plot.bar(color='#ff9933',align='edge')
df[df.is_promoted == 1].department.value_counts().plot.bar(color='#009999',rot=15,align=
plt.legend(['total force', 'promoted'])
plt.subplot(6,2,2)
df[df.is_promoted == 1].education.value_counts().plot.bar(rot=15,color='#009999')

for ind,dept in enumerate(df.department.unique()):
    plt.subplot(6,2,ind+3)
    plt.title(dept)
    plt.pie(
        x=df[df.department==dept].is_promoted.value_counts(normalize=True),
        labels=['not promoted', 'promoted'],
        explode=[0,0.2],
        autopct="%1.1f%%",
        shadow=True,
```

```
textprops=dict(color='w', fontsize=14),  
colors=['#009999', '#ff9933']  
)
```





This concludes :-

- The Sales & Marketing team has the highest number of promoted employees according to the bar plots, this can be because the Sales & Marketing team also has the highest total of employees.
- From the promoted employees, most of them hold a Bachelor's qualification. Bachelor's qualification is also the most frequent in the data and hence this can be accounted for.
- Across the different departments, the percentage of employees promoted are very close to each other.

These conclusions prove that department and education are indecisive when it comes to promotion.

The extensive analysis conducted, led to the following conclusions on this dataset :-

- The performance column which is an amalgamation of KPIs\_met and awards\_won is a good factor to predict which employees get promoted, but it is not the only factor.
- The average training score and the number of trainings attended combine to give the total training score. Employees with Mediocre or High score are likely to be promoted.
- Gender as a factor has no effect on the promotions of the employee, both the genders had about equal percentages of representatives from the two sets.
- The Age and Service categories give a temporal factor for the employees. Employees who are Middle aged and Established or Experienced have high chances of being promoted. The other categories within these proves that Age and Service length are not the only factors affecting promotion.
- The most important factor amongst all is the previous year rating the employee recieved. The chances of promotions increase as the rating increases for an employee. New employees, that have 0 ratings also have a chance at promotion as, these ratings are not the only contributing factor.
- Department and Education of an employee hardly gives much insight into which employee is likely to be promoted.

From the above conclusions, it can be said with confidence that no factor alone is responsible for the promotion of an employee. The following factors together can be considered for predictive modeling.

- performance
- age
- service length
- previous year rating
- total score

The rest of the factors are removed from the dataset.

```
In [68]: rm_cols = [
    'employee_id',
    'department',
    'region',
    'education',
    'gender',
    'recruitment_channel',
    'KPIs_met',
    'awards_won',
    'total_score_label',
    'service_catg',
    'age_label',
    'rating_label'
]

df.drop(rm_cols,axis=1,inplace=True)
df.head(5)
```

```
Out[68]:
```

	no_of_trainings	age	previous_year_rating	length_of_service	avg_training_score	is_promoted	performance
0	1	35	5.0	8	49	0	True
1	1	30	5.0	4	60	0	False
2	1	34	3.0	7	50	0	False
3	2	39	1.0	10	50	0	False
4	1	45	3.0	2	73	0	False

The dataframe above is the final dataset with the factors that affect promotion. This will be now used for model building.

The performance column has to be converted to numeric bool.

```
In [69]: df.performance = df.performance.apply(lambda x: 1 if x else 0)
```

The dataset is scaled using the `MinMaxScaler` from the `sklearn.preprocessing` module. This way all columns are on the same scale. Before model building, the scaled dataset is split into Training set and Testing set with 80:20 ratio.

```
In [70]: print(df.is_promoted.value_counts())

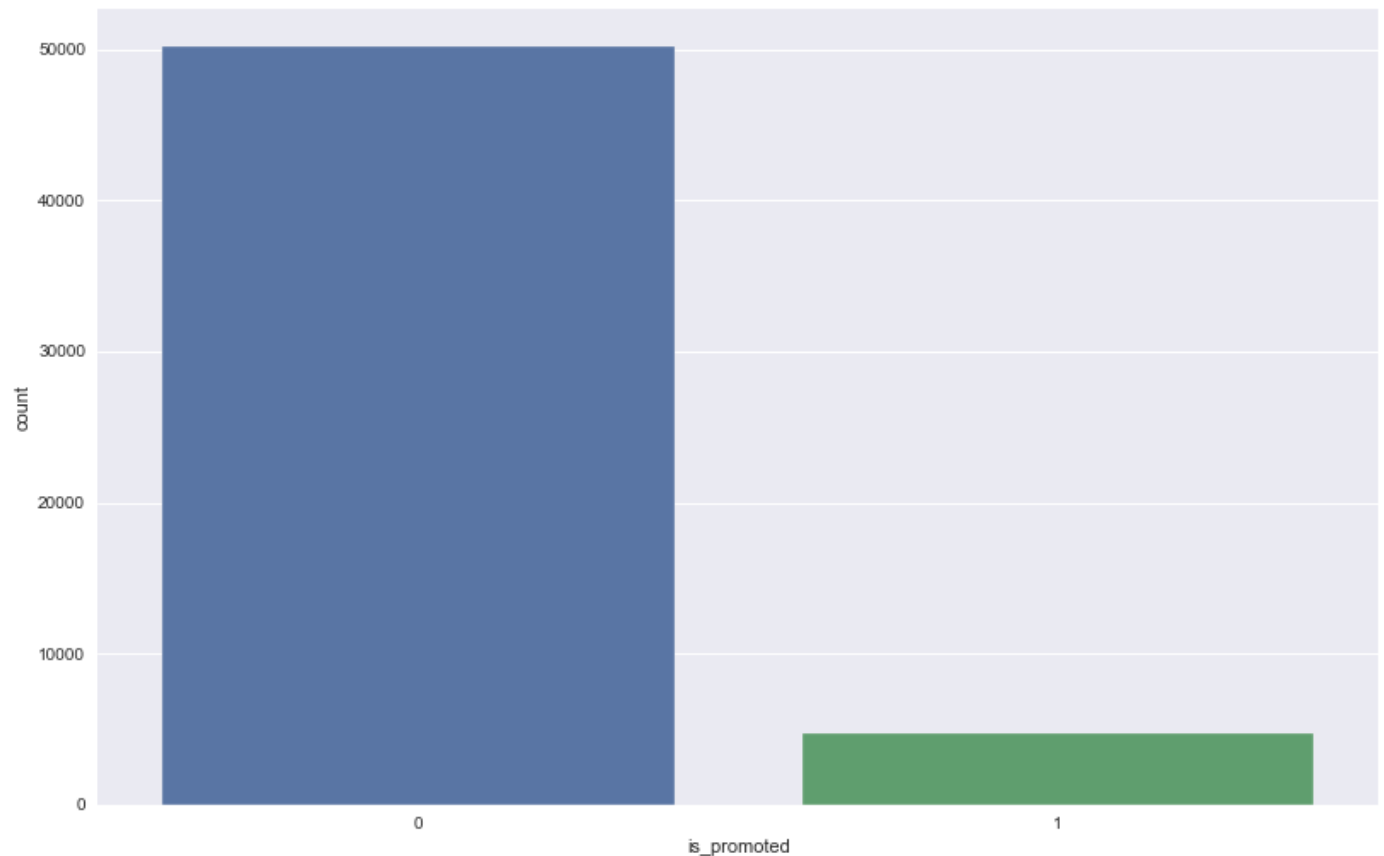
plt.figure(figsize=(12,8))
sns.countplot(df.is_promoted)

0    50140
1     4668
Name: is_promoted, dtype: int64
```

C:\Users\richa\anaconda3\lib\site-packages\seaborn\\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

```
warnings.warn(
```

Out[70]: <AxesSubplot:xlabel='is\_promoted', ylabel='count'>



The dataset has a huge imbalance in the target classes. Training a classifier on imbalanced classes can prove problems in precision and recall of the model. Depending on which class is lesser than the other :-

- low recall + high precision : the model can't detect the class well but is highly trustable when it does.
- high recall + low precision : the class is well detected but the model also include points of other classes in it

To overcome this problem, two solutions are suggested. Either the dataset is undersampled to match the classes or the dataset is oversampled and the class with lesser representatives is increased.

Undersampling can cause loss of data and in this case since class 1 has only 4668 data points, the total depth of the data would be reduced to around 9000. Effectively the model would lose 41000 data points.

Oversampling seems like the better option. The function `RandomOverSampler` from `imblearn.over_sampling` module is used.

```
In [172... X = df.drop('is_promoted', axis=1)
y = df.is_promoted

cols = X.columns

resampler = RandomOverSampler(random_state=1)
X_res, y_res = resampler.fit_resample(X, y)
print(X_res.shape)
print(y_res.value_counts())
```

```
(100280, 7)
1    50140
0    50140
Name: is_promoted, dtype: int64
```

From around 50000 data points, there are about 1,00,000 data points. Both the classes now have equal representatives, about 50140.

```
In [173]: scaler = MinMaxScaler()
X_res = pd.DataFrame(scaler.fit_transform(X_res))
X_res.head(5)
```

```
Out[173]:
```

	0	1	2	3	4	5	6
0	0.000000	0.375	1.0	0.194444	0.166667	1.0	0.014903
1	0.000000	0.250	1.0	0.083333	0.350000	0.0	0.031297
2	0.000000	0.350	0.6	0.166667	0.183333	0.0	0.016393
3	0.111111	0.475	0.2	0.250000	0.183333	0.0	0.090909
4	0.000000	0.625	0.6	0.027778	0.566667	0.0	0.050671

```
In [174]: X_train,X_test,y_train,y_test = train_test_split(X_res,y_res,random_state=0)
```

Using the `sklearn.model_selection - cross_validate` function, the entire dataset will be cross-validated on the entire dataset to see how the model performs on this dataset. The cross validation gives a good method for model selection. The model which performs good on cross validation is then chosen for further training, testing and hyper-parameter tuning.

The ML model chosen here is `GradientBoostingClassifier`. This model is a good classification model and removes any bias in the data. It is robust to over fitting. Since it uses ensemble techniques, various trees are used as predictors. One tree can develop bias, but the combination of outputs of many trees give a stable result.

```
In [185]: scores = cross_validate(
    GradientBoostingClassifier(max_depth=10,random_state=0),
    X_res,
    y_res,
    cv=5,
    scoring=['accuracy','precision_macro','recall_macro','f1'],
    verbose=2
)
print('Avg Test score: ',np.mean(scores['test_accuracy']))
print('Avg Precision: ',np.mean(scores['test_precision_macro']))
print('Avg Recall: ',np.mean(scores['test_recall_macro']))
print('Avg F1 score: ',np.mean(scores['test_f1']))
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[CV] .....
[CV] ..... , total= 19.6s
[CV] .....
```

```
[Parallel(n_jobs=1)]: Done 1 out of 1 | elapsed: 19.6s remaining: 0.0s
```

```
[CV] ..... , total= 18.4s
[CV] .....
[CV] ..... , total= 18.8s
[CV] .....
[CV] ..... , total= 18.6s
[CV] .....
[CV] ..... , total= 18.7s
Avg Test score: 0.8729158356601516
Avg Precision: 0.8845925043898679
Avg Recall: 0.8729158356601516
Avg F1 score: 0.883097997957426
```

```
[Parallel(n_jobs=1)]: Done 5 out of 5 | elapsed: 1.6min finished
```

Cross-validation on the resampled dataset shows excellent results. The average Test set accuracy is about 87% with precision and recall around the same neighbourhood. f1 scores give a holistic view of the precision and recall, which is very good in this case, about 88%. Thus it can be concluded that the `GradientBoostingClassifier` is a good predictor for this dataset.

The next step is hyper-parameter tuning. This tunes the parameters of the model to the best set that can maximize the accuracy of the model. Letting the learning rate be fixed at 0.1, the `max_depth` of the trees (estimators) will be varied between 10 and 25 to get the optimal value. For this parameter a `validation curve` is made. The validation curve is a plot of the train and test accuracies vs a parameter under optimization.

```
In [201]: params = np.arange(10,26,1)
scores = validation_curve(
    GradientBoostingClassifier(),
    X_res,
    y_res,
    param_name='max_depth',
    param_range=params,
    scoring='accuracy',
    verbose=3,
    cv=3
)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[CV] max_depth=10 .....
[CV] .... max_depth=10, score=(train=0.896, test=0.871), total= 17.2s
[CV] max_depth=11 .....
[CV] .... max_depth=11, score=(train=0.930, test=0.898), total= 19.4s
[CV] max_depth=12 .....
[Parallel(n_jobs=1)]: Done 2 out of 2 | elapsed: 37.2s remaining: 0.0s
```

```
[Parallel(n_jobs=1)]: Done 1 out of 1 | elapsed: 17.5s remaining: 0.0s
```

```
[CV] .... max_depth=11, score=(train=0.930, test=0.898), total= 19.4s
[CV] max_depth=12 .....
[Parallel(n_jobs=1)]: Done 2 out of 2 | elapsed: 37.2s remaining: 0.0s
```

```
[Parallel(n_jobs=1)]: Done 2 out of 2 | elapsed: 37.2s remaining: 0.0s
```



```

[CV] .... max_depth=12, score=(train=0.947, test=0.915), total= 24.1s
[CV] max_depth=13 .....
[CV] .... max_depth=13, score=(train=0.964, test=0.931), total= 28.9s
[CV] max_depth=14 .....
[CV] .... max_depth=14, score=(train=0.971, test=0.939), total= 36.4s
[CV] max_depth=15 .....
[CV] .... max_depth=15, score=(train=0.974, test=0.945), total= 42.9s
[CV] max_depth=16 .....
[CV] .... max_depth=16, score=(train=0.974, test=0.948), total= 53.4s
[CV] max_depth=17 .....
[CV] .... max_depth=17, score=(train=0.974, test=0.948), total= 59.3s
[CV] max_depth=18 .....
[CV] .... max_depth=18, score=(train=0.974, test=0.949), total= 1.2min
[CV] max_depth=19 .....
[CV] .... max_depth=19, score=(train=0.974, test=0.949), total= 1.2min
[CV] max_depth=20 .....
[CV] .... max_depth=20, score=(train=0.974, test=0.949), total= 1.3min
[CV] max_depth=21 .....
[CV] .... max_depth=21, score=(train=0.974, test=0.949), total= 1.3min
[CV] max_depth=22 .....
[CV] .... max_depth=22, score=(train=0.974, test=0.949), total= 1.2min
[CV] max_depth=23 .....
[CV] .... max_depth=23, score=(train=0.974, test=0.948), total= 1.2min
[CV] max_depth=24 .....
[CV] .... max_depth=24, score=(train=0.974, test=0.948), total= 1.1min
[CV] max_depth=25 .....
[CV] .... max_depth=25, score=(train=0.974, test=0.947), total= 1.1min
[CV] max_depth=10 .....
[CV] .... max_depth=10, score=(train=0.898, test=0.867), total= 16.1s
[CV] max_depth=11 .....
[CV] .... max_depth=11, score=(train=0.923, test=0.888), total= 18.9s
[CV] max_depth=12 .....
[CV] .... max_depth=12, score=(train=0.950, test=0.913), total= 22.9s
[CV] max_depth=13 .....
[CV] .... max_depth=13, score=(train=0.965, test=0.929), total= 28.6s
[CV] max_depth=14 .....
[CV] .... max_depth=14, score=(train=0.973, test=0.938), total= 36.6s
[CV] max_depth=15 .....
[CV] .... max_depth=15, score=(train=0.974, test=0.941), total= 42.2s
[CV] max_depth=16 .....
[CV] .... max_depth=16, score=(train=0.974, test=0.944), total= 49.6s
[CV] max_depth=17 .....
[CV] .... max_depth=17, score=(train=0.974, test=0.946), total= 57.9s
[CV] max_depth=18 .....
[CV] .... max_depth=18, score=(train=0.974, test=0.947), total= 1.2min
[CV] max_depth=19 .....
[CV] .... max_depth=19, score=(train=0.974, test=0.947), total= 1.2min
[CV] max_depth=20 .....
[CV] .... max_depth=20, score=(train=0.974, test=0.947), total= 1.3min
[CV] max_depth=21 .....
[CV] .... max_depth=21, score=(train=0.974, test=0.947), total= 1.2min
[CV] max_depth=22 .....
[CV] .... max_depth=22, score=(train=0.974, test=0.946), total= 1.2min
[CV] max_depth=23 .....
[CV] .... max_depth=23, score=(train=0.974, test=0.946), total= 1.2min
[CV] max_depth=24 .....
[CV] .... max_depth=24, score=(train=0.974, test=0.945), total= 1.1min
[CV] max_depth=25 .....
[CV] .... max_depth=25, score=(train=0.974, test=0.945), total= 1.1min
[CV] max_depth=10 .....
[CV] .... max_depth=10, score=(train=0.900, test=0.878), total= 15.5s
[CV] max_depth=11 .....
[CV] .... max_depth=11, score=(train=0.931, test=0.904), total= 18.9s
[CV] max_depth=12 .....

```

```

[CV] .... max_depth=12, score=(train=0.950, test=0.921), total= 23.2s
[CV] max_depth=13 .....
[CV] .... max_depth=13, score=(train=0.962, test=0.932), total= 28.3s
[CV] max_depth=14 .....
[CV] .... max_depth=14, score=(train=0.971, test=0.942), total= 34.5s
[CV] max_depth=15 .....
[CV] .... max_depth=15, score=(train=0.973, test=0.946), total= 41.2s
[CV] max_depth=16 .....
[CV] .... max_depth=16, score=(train=0.973, test=0.948), total= 48.9s
[CV] max_depth=17 .....
[CV] .... max_depth=17, score=(train=0.973, test=0.950), total= 57.0s
[CV] max_depth=18 .....
[CV] .... max_depth=18, score=(train=0.973, test=0.951), total= 1.1min
[CV] max_depth=19 .....
[CV] .... max_depth=19, score=(train=0.973, test=0.951), total= 1.2min
[CV] max_depth=20 .....
[CV] .... max_depth=20, score=(train=0.973, test=0.952), total= 1.3min
[CV] max_depth=21 .....
[CV] .... max_depth=21, score=(train=0.973, test=0.951), total= 1.3min
[CV] max_depth=22 .....
[CV] .... max_depth=22, score=(train=0.973, test=0.951), total= 1.3min
[CV] max_depth=23 .....
[CV] .... max_depth=23, score=(train=0.973, test=0.950), total= 1.2min
[CV] max_depth=24 .....
[CV] .... max_depth=24, score=(train=0.973, test=0.950), total= 1.2min
[CV] max_depth=25 .....
[CV] .... max_depth=25, score=(train=0.973, test=0.948), total= 1.2min
[Parallel(n_jobs=1)]: Done 48 out of 48 | elapsed: 43.2min finished

```

In [204]:

```

train_scores = scores[0]
test_scores = scores[1]

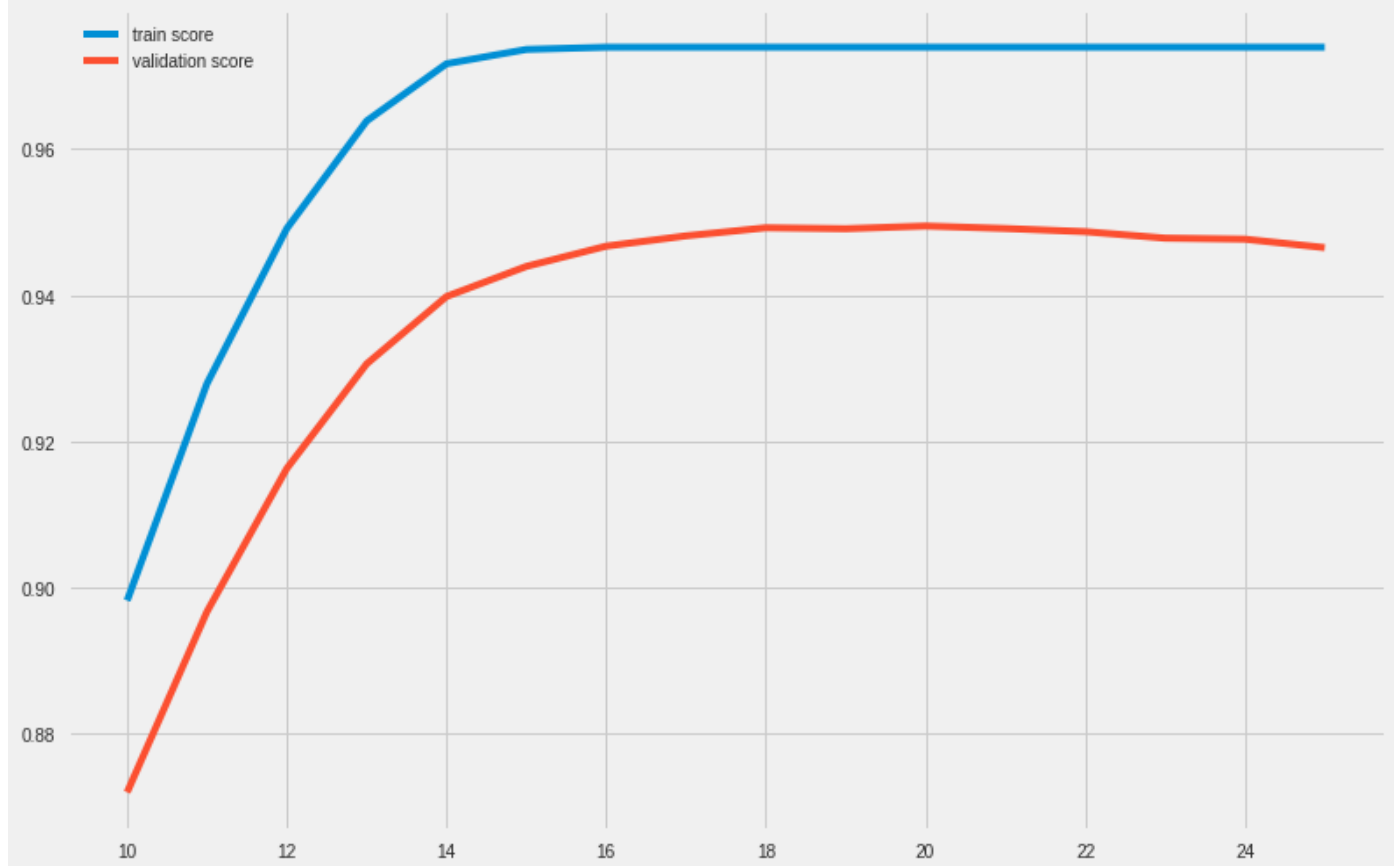
train_means = np.mean(train_scores,axis=1)
test_means = np.mean(test_scores,axis=1)

plt.style.use('fivethirtyeight')
plt.figure(figsize=(12,8))
sns.lineplot(x=params,y=train_means)
sns.lineplot(x=params,y=test_means)
plt.legend(['train score','validation score'])

```

Out[204]:

<matplotlib.legend.Legend at 0x7f46c15db7d0>



The validation curve for the test accuracy identifies the optimal value of `max_depth` for classifier. In this case the `max_depth` value of 20 yields a average test accuracy of 94.94%. This accuracy is quite good and shows that the model has learnt the classes well and is highly trustable in its predictions.

The validation curve is intuitive when plotting for a single parameter. The

`GradientBoostingClassifier` has many parameters such as `learning_rate`, `max_depth`, `n_estimators` etc. When the best value for these have to be found together, `GridSearchCV` is used. For every combination of these parameters cross validation is performed. The combination giving the best score is returned along with the parameters in the combination.

```
In [210... model = GradientBoostingClassifier(max_depth=20, random_state=0)
grid_vals = {
    'learning_rate': np.arange(0.1, 0.5, 0.1),
    'n_estimators': np.arange(100, 500, 100)
}
grid_mod = GridSearchCV(model, param_grid=grid_vals, cv=3, verbose=1)
grid_mod.fit(X_train, y_train)
print(grid_mod.best_params_)
print(grid_mod.best_score_)
```

Fitting 3 folds for each of 16 candidates, totalling 48 fits

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 48 out of 48 | elapsed: 53.4min finished
{'learning_rate': 0.4, 'n_estimators': 300}
0.9405132296237203
```

After performing a `Grid search` the optimal paramters for the model are returned. Using these parameters, the model is finally trained on the train set and tested as well as make predictions on the test set. This model becomes the final output from the analysis.

```
In [211... #Applying Gradient Boosting Classifier
model = GradientBoostingClassifier(learning_rate=0.4, n_estimators=300, max_depth=20, random_state=0)
model.fit(X_train, y_train)
```

```
print(model.score(X_train,y_train))  
print(model.score(X_test,y_test))
```

```
0.9738066746443292
```

```
0.9570003988831273
```

The above model does extremely well due to the hyper-parameter tuning executed on the model. The model gives the best fit possible with a test set accuracy of 95.7%. Generating a `classification_report` to gain more insights into the model.

The `classification_report` gives all the important metrics for a classifier. The model achieves a 100% precision and 92% recall on the test set. The model is now ready to predict which employees get promoted and which do not based on various factors. This concludes the project.