WORCESTER POLYTECHNIC INSTITUTE
DATA SCIENCE PROGRAM

Case Study 4
HR Analytics

SUBMITTED BY
Sirshendu Ganguly
Enbo Tian
Dang Tran

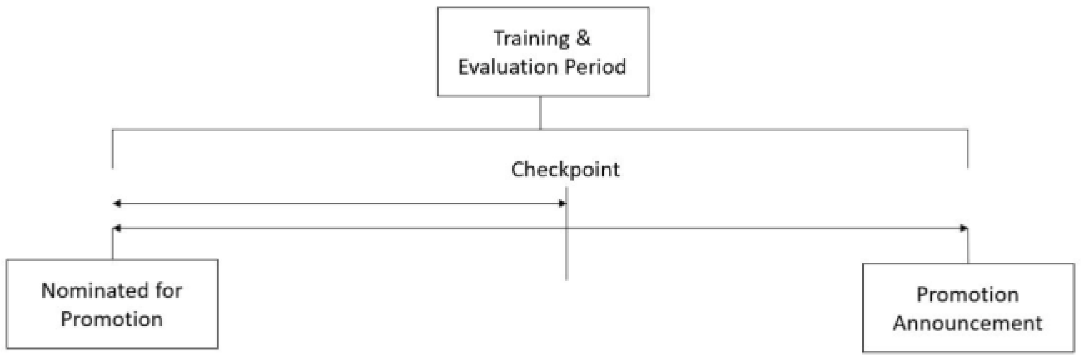
Date Submitted	**: 4/20/2022**
Date Completed	**: 4/20/2022**
Course Instructor	**: Prof. Ngan**

Motivation and Background

There's a MNC that is spending too much money and resources on determining which employees deserve a promotion and they're looking for a solution. The current process entails:

1. First identify a set of employees based on recommendations/past performance.
2. Have the selected employees go through a training and evaluation program based on the department they're in.
3. At the end of the program, based on various factors such as training performances, KPI completion, etc., employee get promotion.

For the above process, the final promotions are only announced after the final evaluation and this leads to delay in transitioning the employees to their new roles.



Since the MNC have a lot of data on their past employees who have received promotions, we decided to take advantage of that for our solution. We propose to use a ML model that can be used to predict who is worthy of promotion at a certain checkpoint before the final evaluation period.

Data Sources

In this case study, we used the Employee dataset from an ongoing Data Science Competition.

The competition can be found at https://datahack.analyticsvidhya.com/contest/wns-analytics-hackathon-2018-1/#ProblemStatement.

This dataset contains 54,808 different employees that were considered for promotion in the past with 14 attributes. These attributes include:

employee_id: Unique ID for employee

department: Department of employee

region: Region of employment (unordered)

education: Education Level

gender: Gender of Employee

recruitment_channel: Channel of recruitment for employee

no_of_trainings: number of other training completed in previous year on soft skills, technical skills, etc.

age: Age of Employee

previous_year_rating: Employee Rating for the previous year

length_of_service: Length of service in years

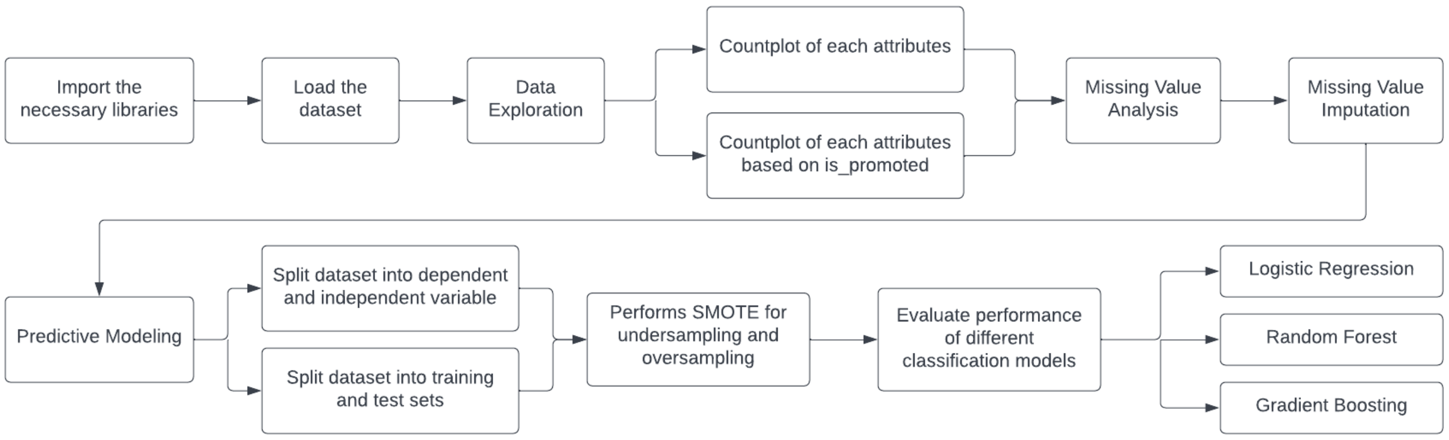
KPIs_met >80%: if percent of KPIs (Key performance Indicators) >is larger than 80% then 1, else 0

awards_won?: if awards won during previous year then 1, else 0

avg_training_score: Average score in current training evaluations

is_promoted (Target): Recommended for promotion then 1, else 0

Methodology



Import the necessary libraries
Load the dataset
Data Exploration
Countplot of each attributes
Countplot of each attributes based on is_promoted
Missing Value Analysis
Missing Value Imputation
Predictive Modeling
Split dataset into dependent and independent variable
Split dataset into training and test sets
Performs SMOTE for undersampling and oversampling
Evaluate performance of different classification models
Logistic Regression
Random Forest
Gradient Boosting

Math Modeling

1. Logistic Regression

   The equation for Logistic Regression is :

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

To implement logistic regression, we would like to transform this formula to:

$$\frac{p(x)}{1 - p(x)} = e^{\beta_0 + \beta_1 x}$$

$$ln\frac{p(x)}{1 - p(x)} = \beta_0 + \beta_1 x$$

Since we have multiple explanatory variables, $\beta_0 + \beta_1 x$ can be revised to $\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_m x_m$.

2. Random Forest

For each decision tree, Scikit-learn calculates a nodes importance using Gini Importance, assuming only two child nodes (binary tree):

$$ni_j = w_j C_j - w_{left(j)} C_{left(j)} - w_{right(j)} C_{right(j)}$$

where

ni_j = the importance of node j

w_j = weighted number of samples reaching node j

C_j = the impurity value of node j

$left(j)$ = child node from left split on node j

$right(j)$ = child node from right split on node j

The importance of each feature on a decision tree is then calculated as:

$$fi_i = \frac{\sum_{\text{j: node j splits on feature}} ni_j}{\sum_{k \in \text{all nodes}} ni_k}$$

where

fi_i = the importance of feature i

ni_j = the importance of node j

These can then be normalized to a value between 0 and 1 by dividing by the sum of all feature importance values:

$$normfi_i = \frac{fi_i}{\sum_{j \in \text{all features}} fi_j}$$

Thus, we can now get our random forest function:

$$RFfi_i = \frac{\sum_{j \in \text{all trees}} normfi_j}{T}$$

where T is the total number of trees.

To implement this, For each decision tree, Spark calculates a feature's importance by summing the gain, scaled by the number of samples passing through the node:

$$fi_i = \sum_{j:nodes\ j\ splits\ on\ feature\ i} s_j C_j$$

Sj is number of samples reaching node j

Cj is the impurity value of node j

Then the feature importance for each tree is normalized in relation to the tree:

$$normfi_i = \frac{fi_i}{\sum_{j \in \text{all features}} fi_j}$$

and feature importance values from each tree are summed normalized:

$$RFfi_i = \frac{\sum_j normfi_{ij}}{\sum_{j \in \text{all features}, k \in \text{all trees}} normfi_{jk}}$$

3. Gradient Boosting

The goal is to find some function $\hat{F}(x)$ that best approximates the output variable y from the values of input variables x.
This is formalized by introducing some loss function and minimizing it:

$$\hat{F} = \arg\min_{F} \mathbb{E}_{x,y}[L(y, F(x))]$$

The gradient boosting method assumes a real-valued *y* and seeks an approximation $\hat{F}(x)$ in the form of a weighted sum of functions $h_i(x)$ from some class H, called base (or weak) learners:

$$\hat{F}(x) = \sum_{i=1}^{M} \gamma_i h_i(x) + \text{const}$$

To implement this, we Start with a model, consisting of a constant function $F_0(x)$, and incrementally expands it in a greedy fashion:

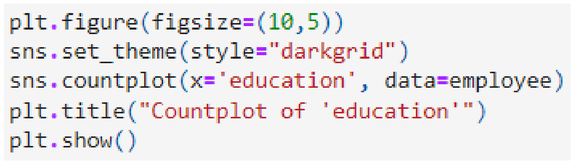
$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$$
$$F_m(x) = F_{m-1}(x) + \arg\min_{h_m \in \mathcal{H}} \left[\sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + h_m(x_i)) \right]$$

Data Extraction and Preparation

- To begin, we needed to import some necessary libraries. These include pandas, numpy, matplotlib.pyplot, seaborn, sklearn, typing, statsmodels.api, imblearn, and math.
- Load the csv Employee dataset into a panda DataFrame.

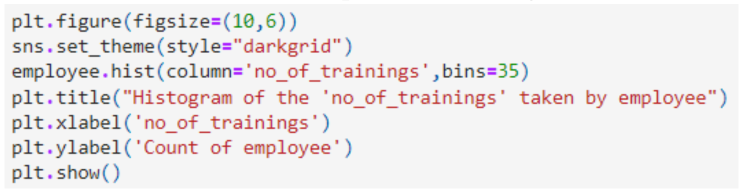
Data Exploration

- Used panda head() to see a preview of the dataset.
- Ran df.info() to get the number of data in the dataset based on employee_id.
    - Identified the attributes with missing values if their non-null count is lower than that of employee_id.
- Explored all the attributes with the exception of employee_id
    - Categorical attributes:
        - Get the number of unique variables in each of these attributes and display their values.
        - Made a countplot of each attribute itself and of each attribute based on is_promoted.
        - Utilized seaborn and matplotlib for countplot creation.



```
plt.figure(figsize=(10,5))
sns.set_theme(style="darkgrid")
sns.countplot(x='education', data=employee)
plt.title("Countplot of 'education'")
plt.show()
```

    - Numerical attributes:
        - Get the number of unique variables in each of these attributes.
        - Plot a histogram of the attribute with the count of all employees in the dataset.
        - Utilized seaborn and matplotlib for histogram creation.



```
plt.figure(figsize=(10,6))
sns.set_theme(style="darkgrid")
employee.hist(column='no_of_trainings',bins=35)
plt.title("Histogram of the 'no_of_trainings' taken by employee")
plt.xlabel('no_of_trainings')
plt.ylabel('Count of employee')
plt.show()
```

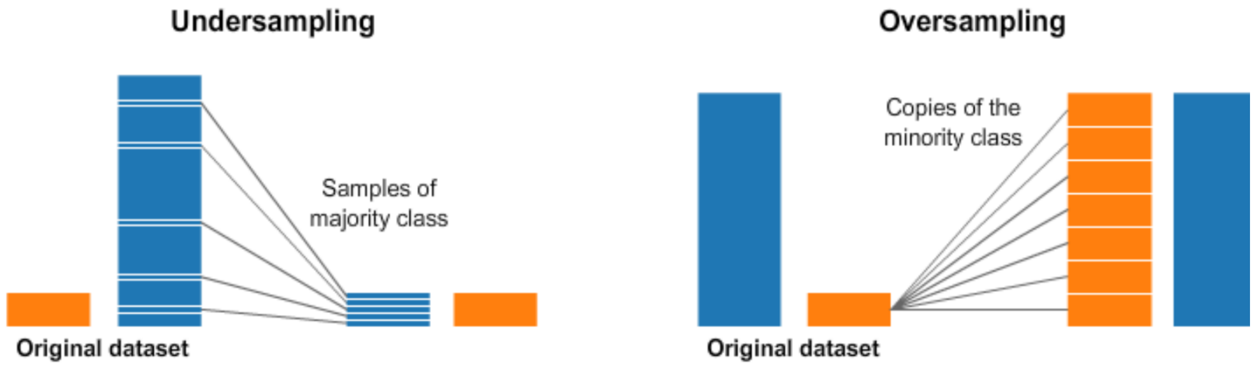
Missing Value Analysis

- Plot a heatmap of the dataset to confirms the attributes with missing value we found earlier
    - “Education” and “previous_year_rating” are confirmed to have missing values.
- Plot a heatmap of the correlation among the attributes
- Decided to impute the attributes with missing values.

- Display the value counts and countplot of each of these attributes for before and after imputation comparison.
- Imputed “previous_year_rating” by simply replacing the missing data with the most common value in that column (3.0).
- Imputed “education” using the RandomForest classification model.
  - Wrote a function to impute a target attribute within a DataFrame.
  - Excluded “employee_id” and “is_promoted” from imputation.
  - Used RandomForest with an n_estimators value of 1000.
  - Fit and resampled the dataset with SMOTE.
  - Tested the performance of the model and displayed the confusion matrix of the imputation
  - Process the DataFrame for prediction.
  - Encode values for prediction.
  - Make predictions.
  - Add the predictions to their respective columns.

Predictive Modeling

- Set “is_promoted” as our dependent column.
  - Split the dataset with “is_promoted” as the dependent variable and everything else as the independent variable.
- Split into training and testing sets with test_size of 0.3 and random_state of 2.
- Needs to deal with unbalanced target variable (“is_promoted”).
  - Used SMOTE (Synthetic Minority Over-sampling Technique) for undersampling and oversampling.



- Wrote a function to test the performance of a classification model.
  - Output the Accuracy K-Fold value and the F1 Score.
  - Creates a confusion matrix of the model performance.
- Ran the above function with Logistic Regression, RandomForest, and GradientBoosting.
  - For each model, tested the performance without SMOTE, and with undersampling and oversampling SMOTE.

Results

We firstly loaded the data to see the information of the data set:

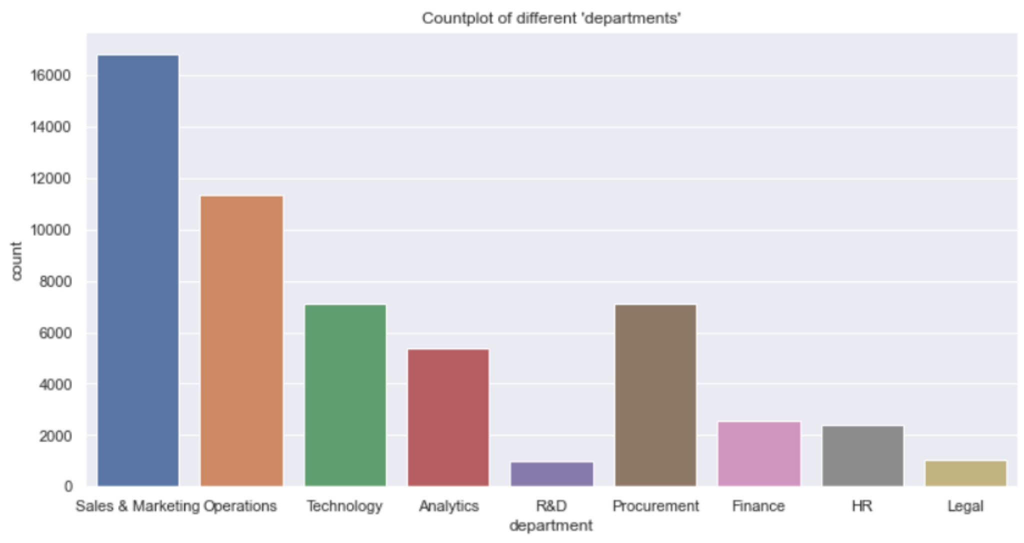
	employee_id	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_year_rating	length_of_service
0	65438	Sales & Marketing	region_7	Master's & above	f	sourcing	1	35	5.0	8
1	65141	Operations	region_22	Bachelor's	m	other	1	30	5.0	4
2	7513	Sales & Marketing	region_19	Bachelor's	m	sourcing	1	34	3.0	7
3	2542	Sales & Marketing	region_23	Bachelor's	m	other	2	39	1.0	10
4	48945	Technology	region_26	Bachelor's	m	other	1	45	3.0	2

KPIs_met >80%	awards_won?	avg_training_score	is_promoted
1	0	49	0
0	0	60	0
0	0	50	0
0	0	50	0
0	0	73	0

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54808 entries, 0 to 54807
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   employee_id           54808 non-null  int64
 1   department            54808 non-null  object
 2   region                54808 non-null  object
 3   education             52399 non-null  object
 4   gender                54808 non-null  object
 5   recruitment_channel   54808 non-null  object
 6   no_of_trainings       54808 non-null  int64
 7   age                   54808 non-null  int64
 8   previous_year_rating  50684 non-null  float64
 9   length_of_service     54808 non-null  int64
 10  KPIs_met >80%         54808 non-null  int64
 11  awards_won?           54808 non-null  int64
 12  avg_training_score    54808 non-null  int64
 13  is_promoted           54808 non-null  int64
dtypes: float64(1), int64(8), object(5)
memory usage: 5.9+ MB
```

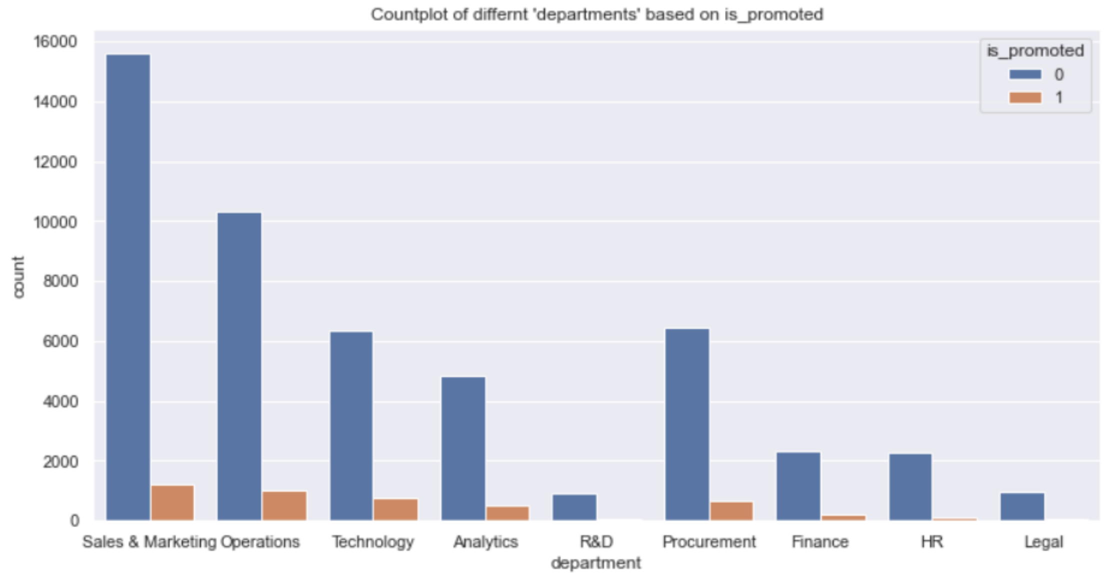
We have 54808 employees, however we have missing values on education and previous year rating.

We explore the data of departement, with Plotting the count plot of the different departments:



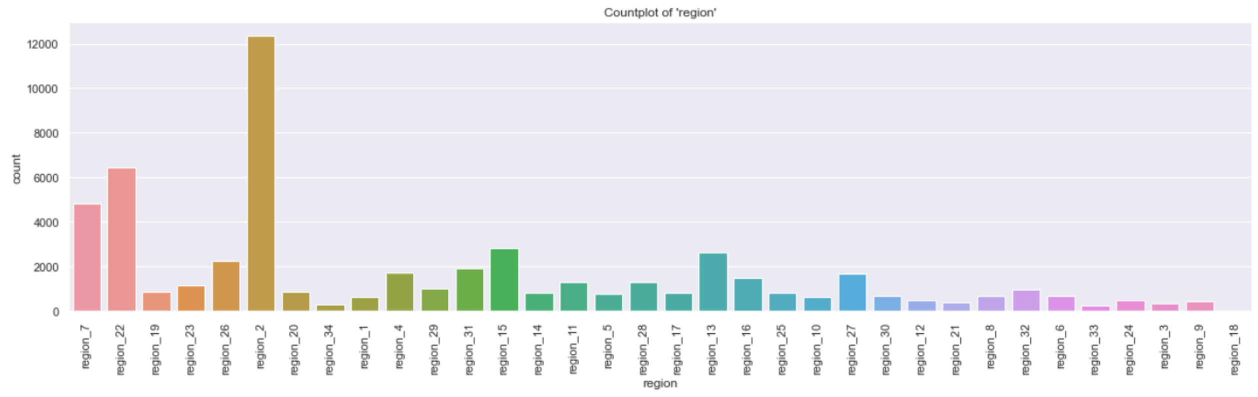
```
No of unique department:  9

The unique departments are:
['Sales & Marketing' 'Operations' 'Technology' 'Analytics' 'R&D'
 'Procurement' 'Finance' 'HR' 'Legal']
```

We plot the count plot of the different departments based on is_promoted:

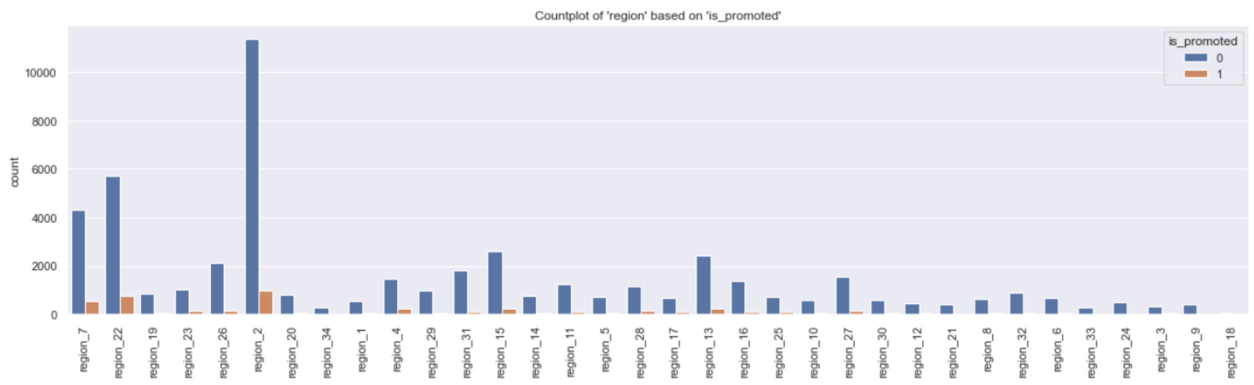


We use the same way to explore the data of region:

```
No of unique region:  34

The unique region are:
['region_7' 'region_22' 'region_19' 'region_23' 'region_26' 'region_2'
 'region_20' 'region_34' 'region_1' 'region_4' 'region_29' 'region_31'
 'region_15' 'region_14' 'region_11' 'region_5' 'region_28' 'region_17'
 'region_13' 'region_16' 'region_25' 'region_10' 'region_27' 'region_30'
 'region_12' 'region_21' 'region_8' 'region_32' 'region_6' 'region_33'
 'region_24' 'region_3' 'region_9' 'region_18']
```

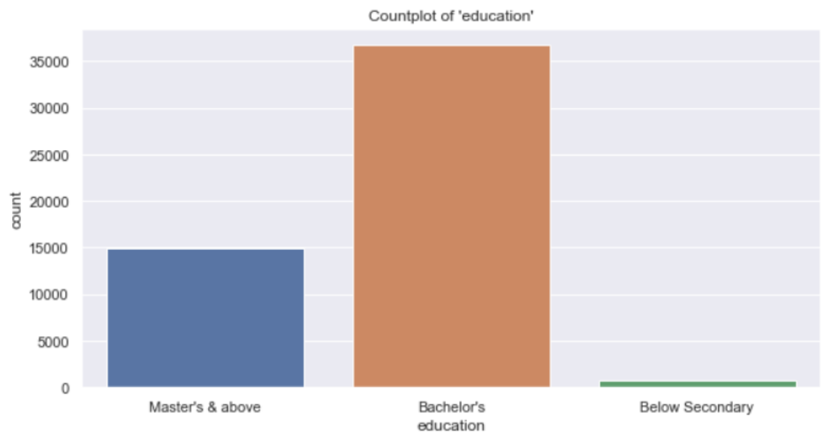


Plotting the count plot of the different regions based on is_promoted:

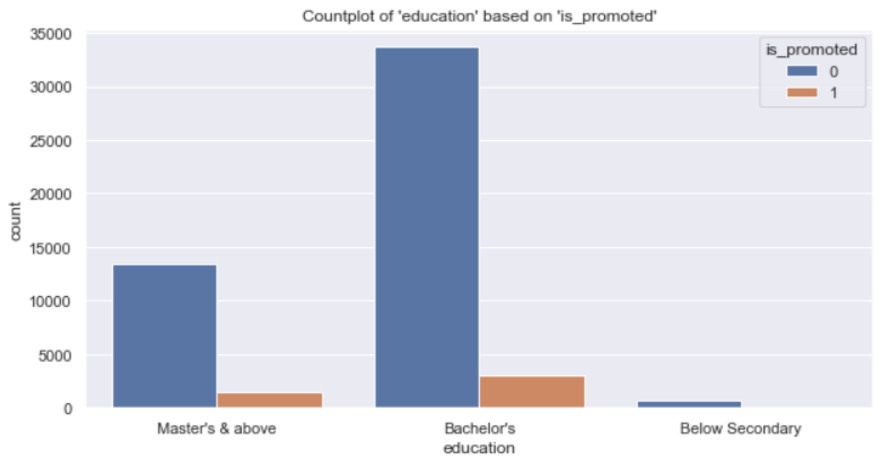


Exploring the data of 'education':

```
 No of unique education:  3

 The unique education are:
 ["Master's & above" "Bachelor's" nan 'Below Secondary']
```

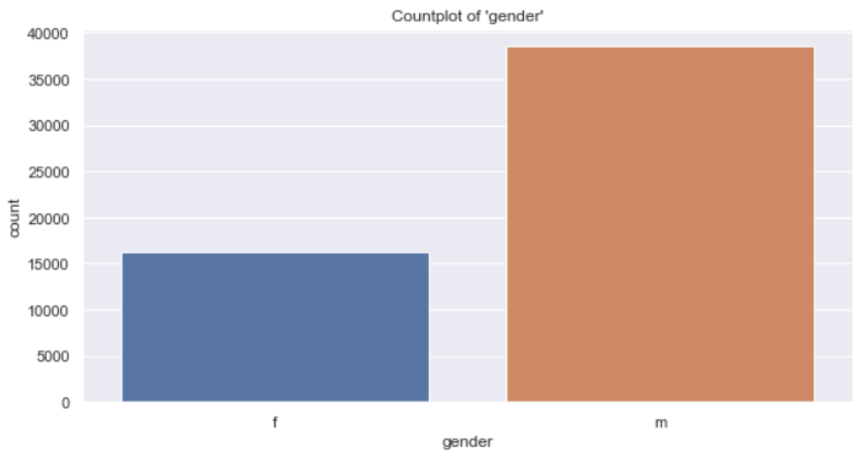


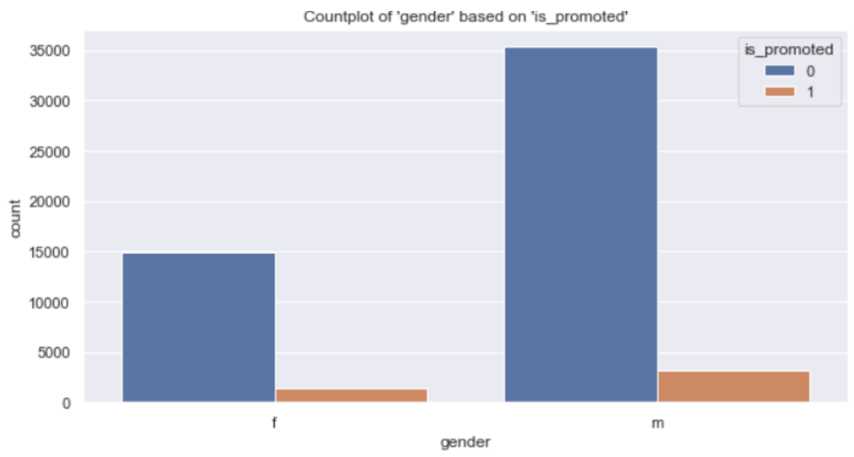
Plotting the count plot of the different education based on is_promoted:



Exploring the data of 'gender', and plot the count plot of the different gender based on is_promoted:

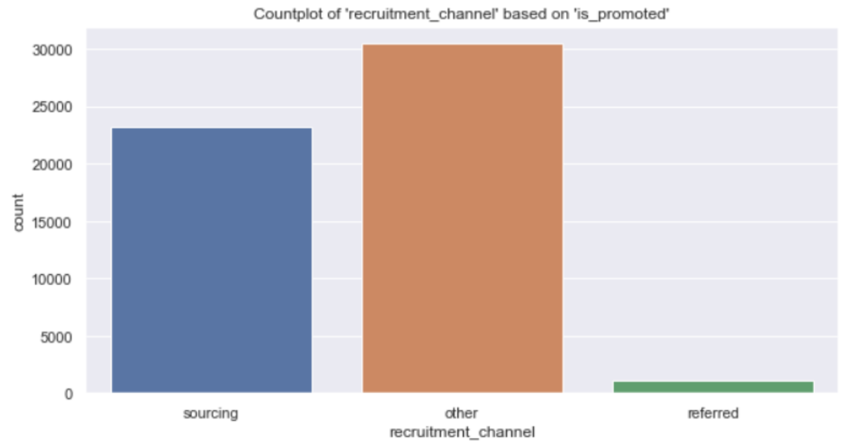
```
No of unique gender:  2

The unique gender are:
['f' 'm']
```

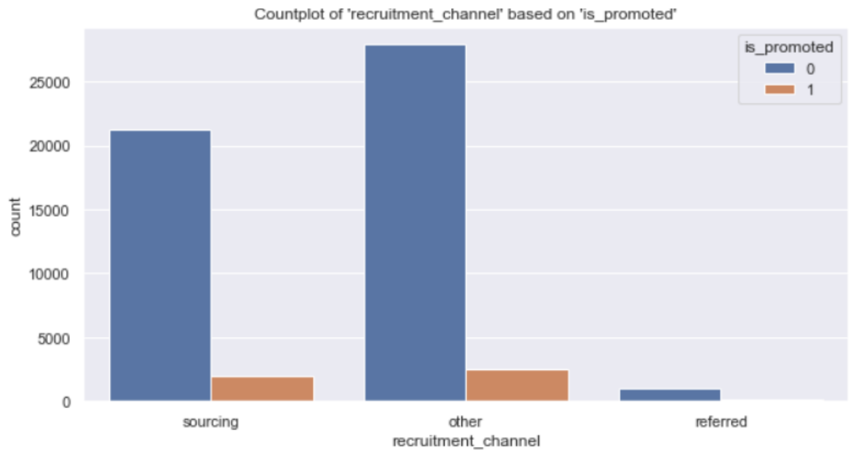




Exploring the data of 'recruitment_channel', and plot the count plot of the different recruitment_channel based on is_promoted:

```
No of unique recruitment_channel:  3

The unique recruitment_channel are:
['sourcing' 'other' 'referred']
```

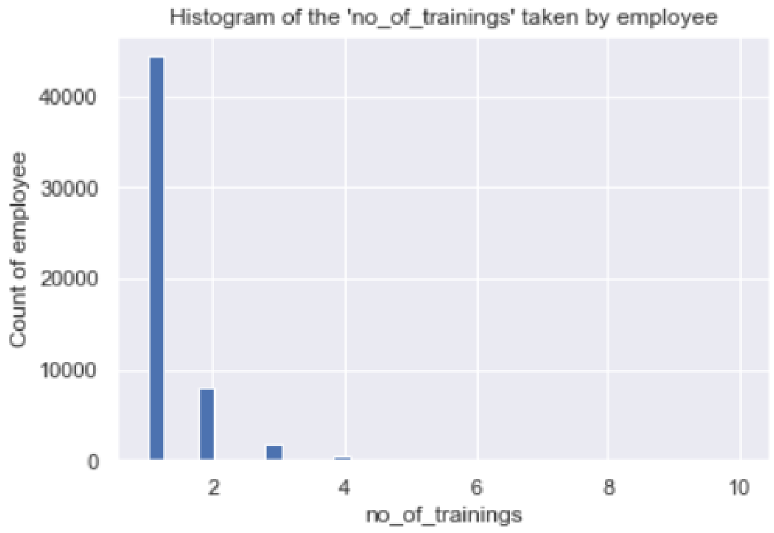




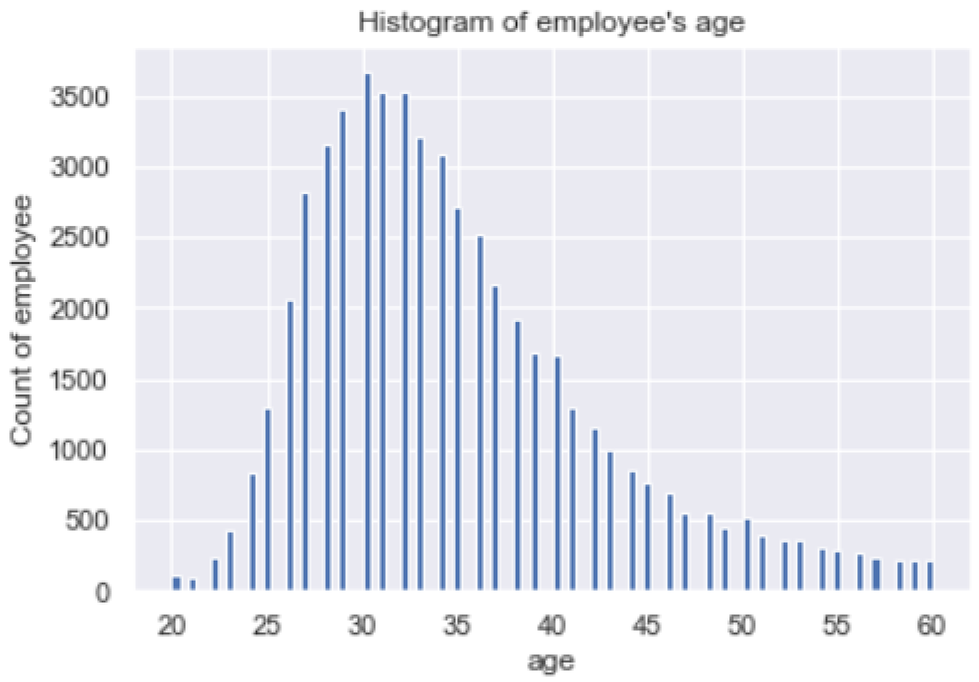
Exploring the data of 'no_of_trainings':

```
No of unique no_of_trainings:  10
```

```
<Figure size 720x432 with 0 Axes>
```

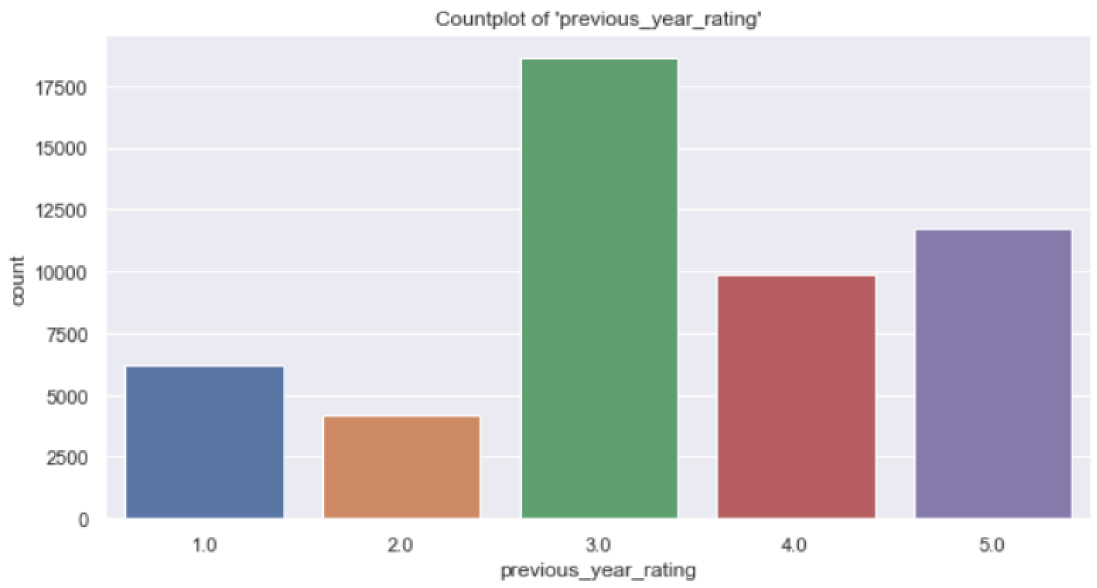


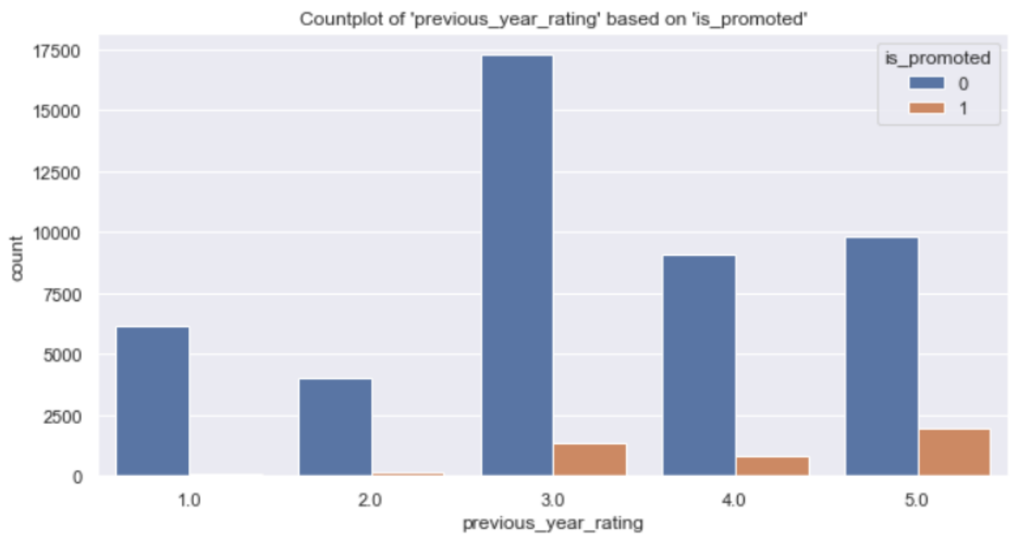
We have 41 group of age:



Exploring 'previous_year_rating', and Plotting the count plot of the different previous_year_rating based on is_promoted:

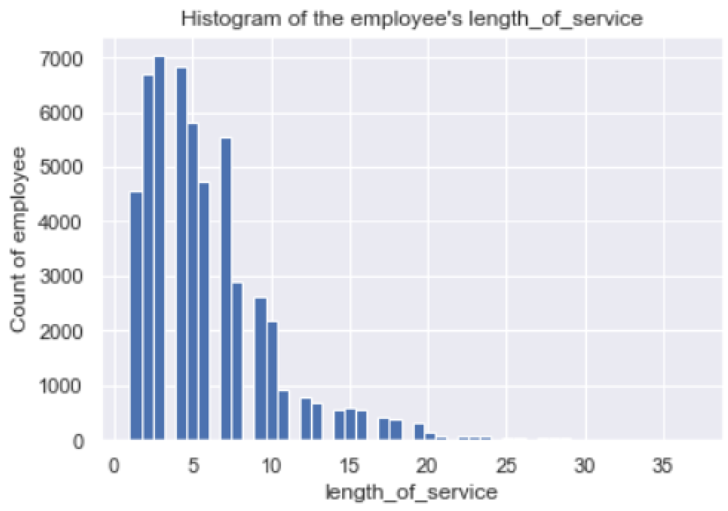
```
No of unique previous_year_rating:  5

The unique previous_year_rating are:
[ 5.   3.   1.   4.  nan  2.]
```



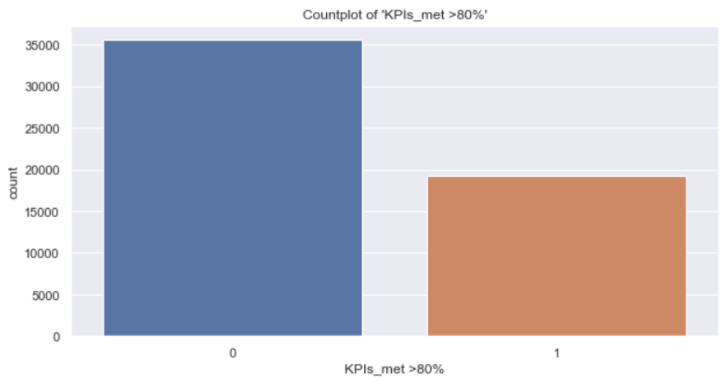


We have 35 'length_of_service':

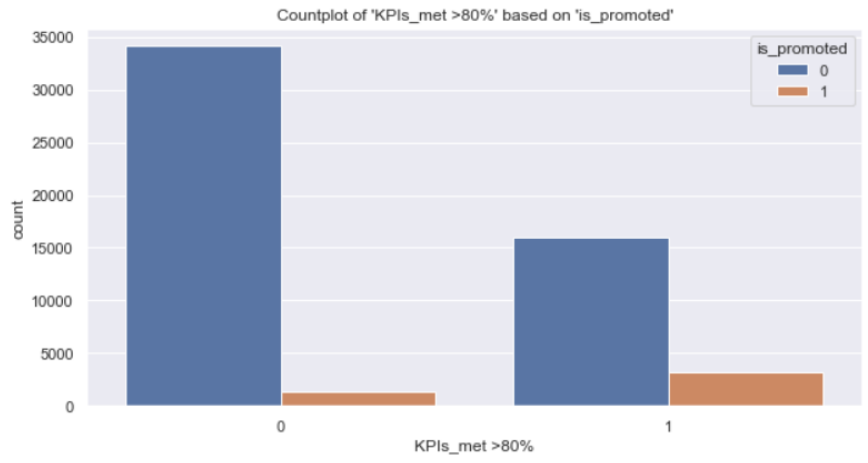
```
<Figure size 720x432 with 0 Axes>
```



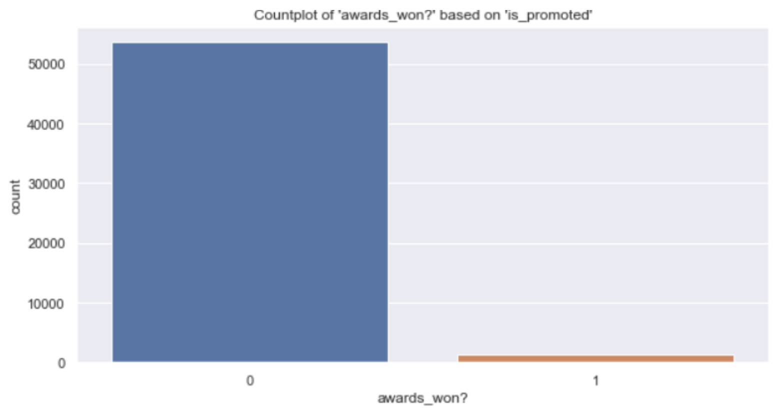
Exploring 'KPIs_met >80%':

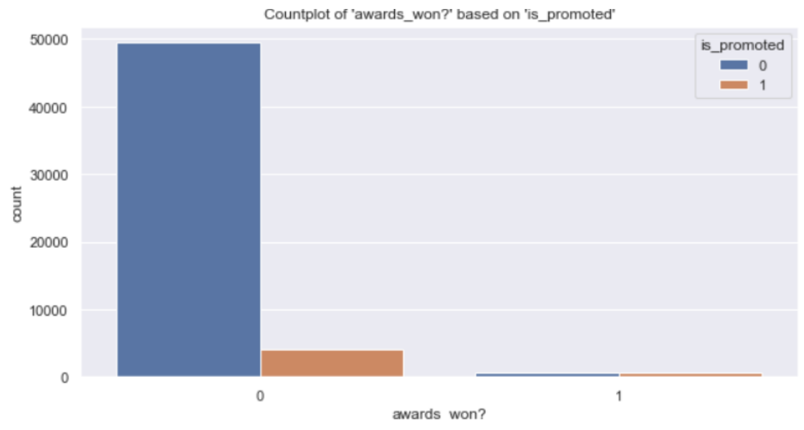


Plotting the count plot of the different KPIs_met >80% based on is_promoted:

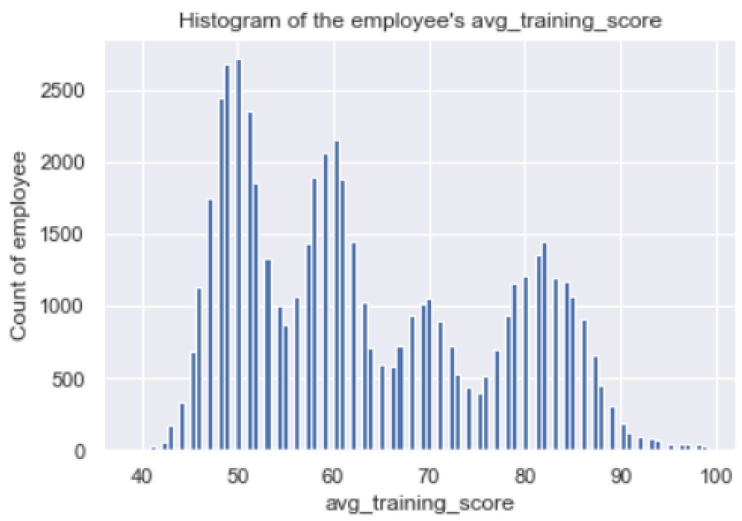


Exploring 'department' and plotting the count plot of the different awards_won? based on is_promoted:

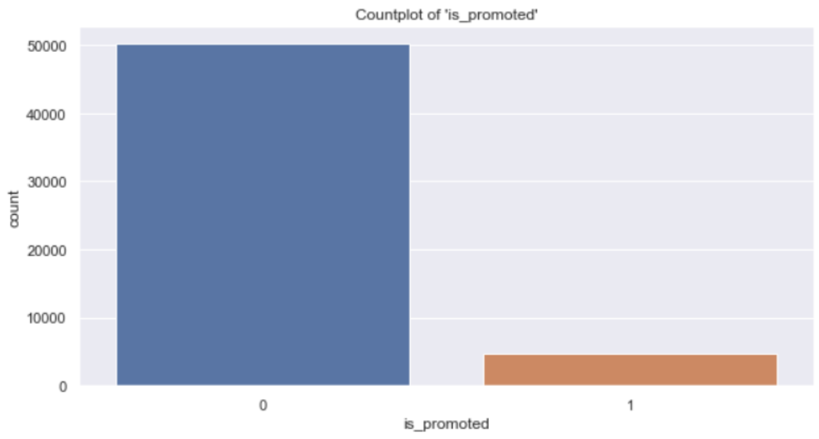




Exploring 61 'avg_training_score':

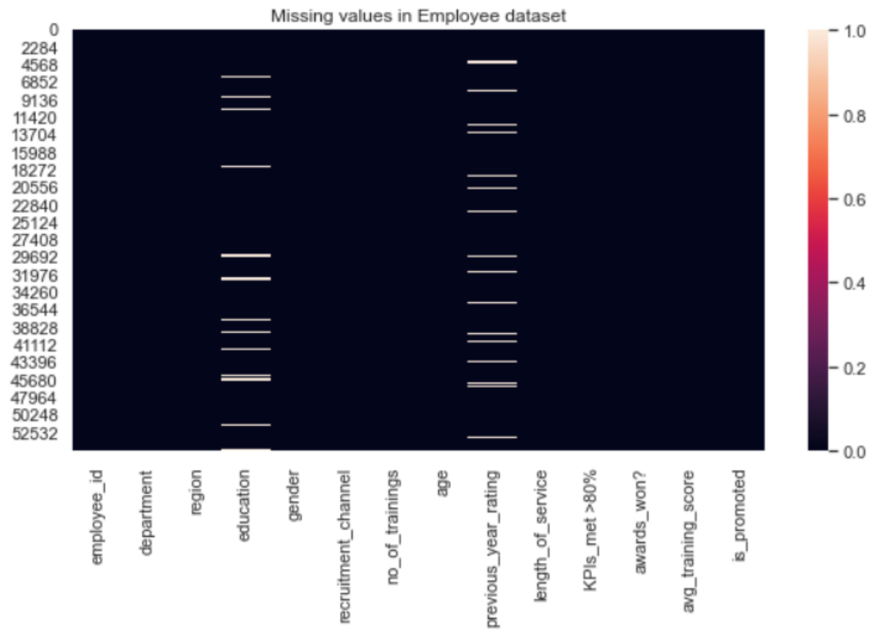


Exploring 'is_promoted':



The next step is to analyze the missing values.

Identifying missing value:

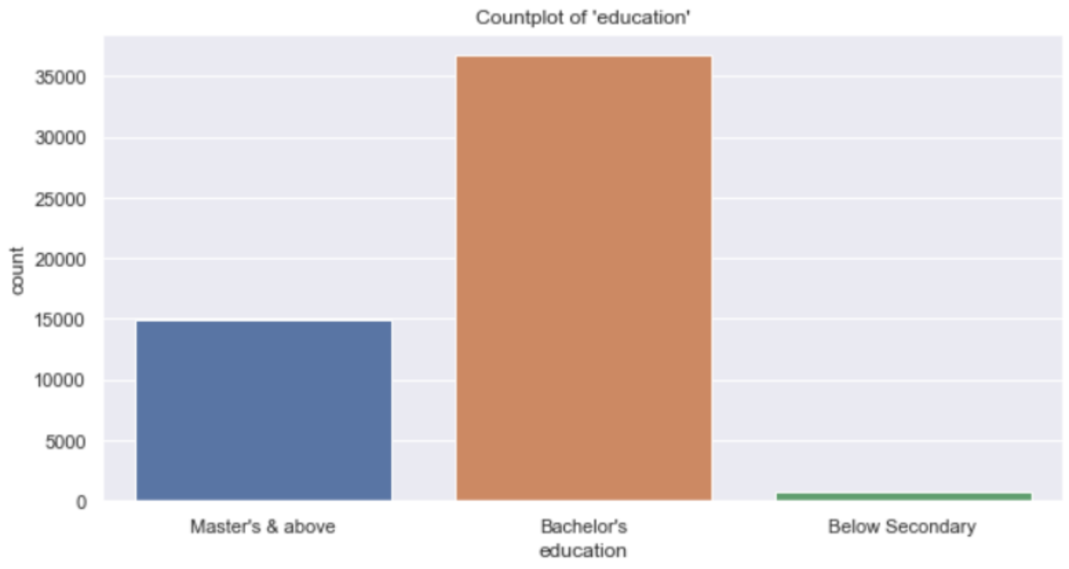


There are two missing values: education and previous_year_rating.

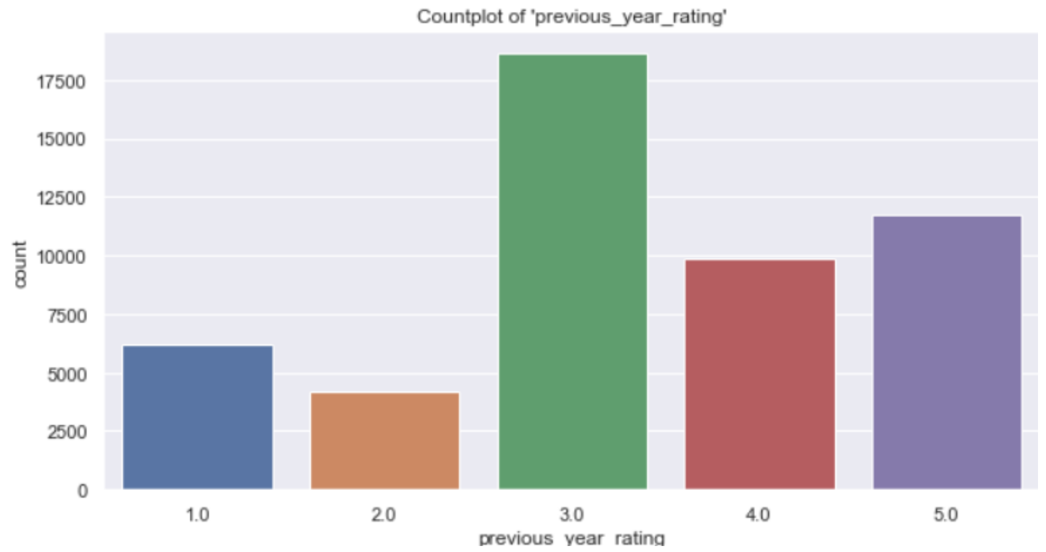
Percentage of missing value in 'education': 4.395343745438622 %

Percentage of missing value in 'previous_year_rating': 7.524448985549554 %

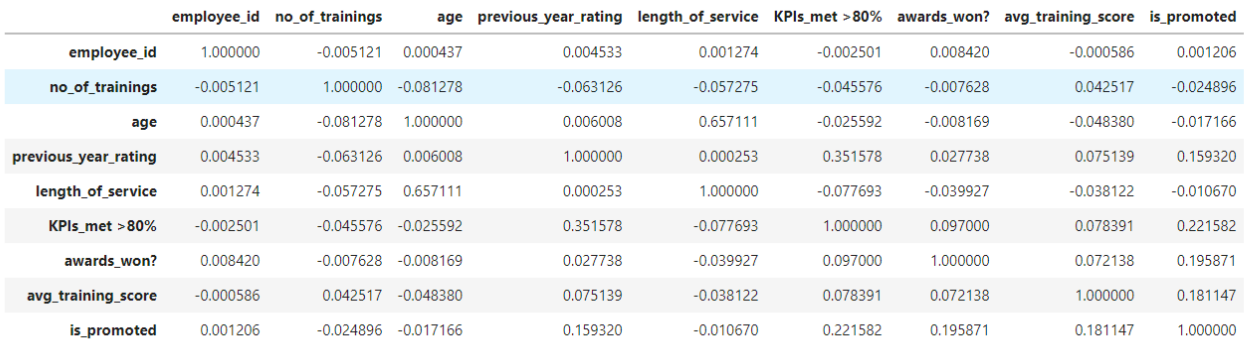
Plotting the count plot of the different education:



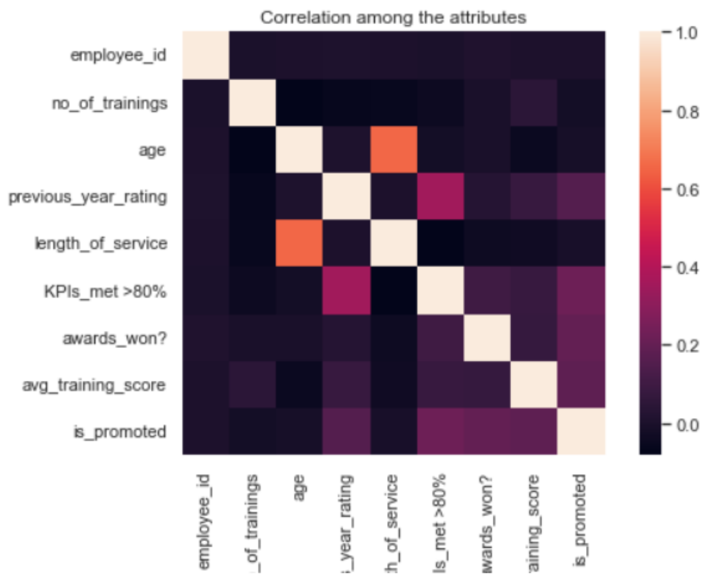
Plotting the count plot of the different previous_year_rating:



We then find the correlation of the employees:



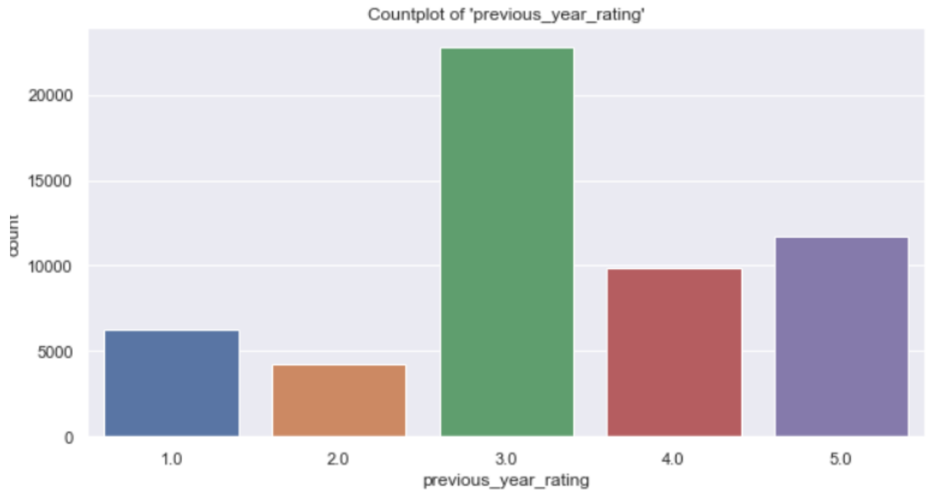
	employee_id	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_met >80%	awards_won?	avg_training_score	is_promoted
employee_id	1.000000	-0.005121	0.000437	0.004533	0.001274	-0.002501	0.008420	-0.000586	0.001206
no_of_trainings	-0.005121	1.000000	-0.081278	-0.063126	-0.057275	-0.045576	-0.007628	0.042517	-0.024896
age	0.000437	-0.081278	1.000000	0.006008	0.657111	-0.025592	-0.008169	-0.048380	-0.017166
previous_year_rating	0.004533	-0.063126	0.006008	1.000000	0.000253	0.351578	0.027738	0.075139	0.159320
length_of_service	0.001274	-0.057275	0.657111	0.000253	1.000000	-0.077693	-0.039927	-0.038122	-0.010670
KPIs_met >80%	-0.002501	-0.045576	-0.025592	0.351578	-0.077693	1.000000	0.097000	0.078391	0.221582
awards_won?	0.008420	-0.007628	-0.008169	0.027738	-0.039927	0.097000	1.000000	0.072138	0.195871
avg_training_score	-0.000586	0.042517	-0.048380	0.075139	-0.038122	0.078391	0.072138	1.000000	0.181147
is_promoted	0.001206	-0.024896	-0.017166	0.159320	-0.010670	0.221582	0.195871	0.181147	1.000000



We are going to impute 'education' and 'previous_year_rating' both of which are categorical variables:

```
Value counts of 'previous_year_rating' before inputation
3.0    18618
5.0    11741
4.0     9877
1.0     6223
2.0     4225
Name: previous_year_rating, dtype: int64
```

Plotting the count plot of the different previous_year_rating after imputation:

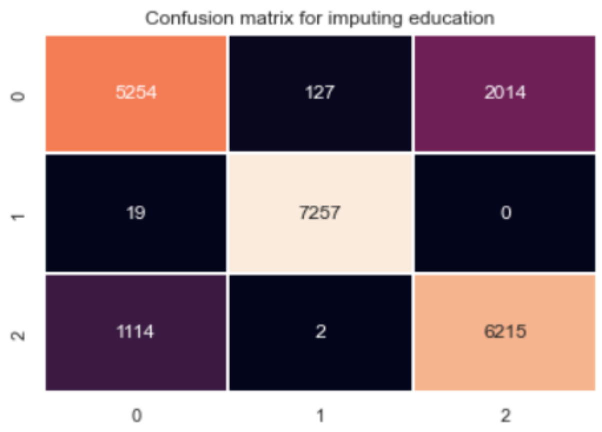


```
Value counts of 'previous_year_rating' after imputation
3.0    22742
5.0    11741
4.0     9877
1.0     6223
2.0     4225
Name: previous_year_rating, dtype: int64
```

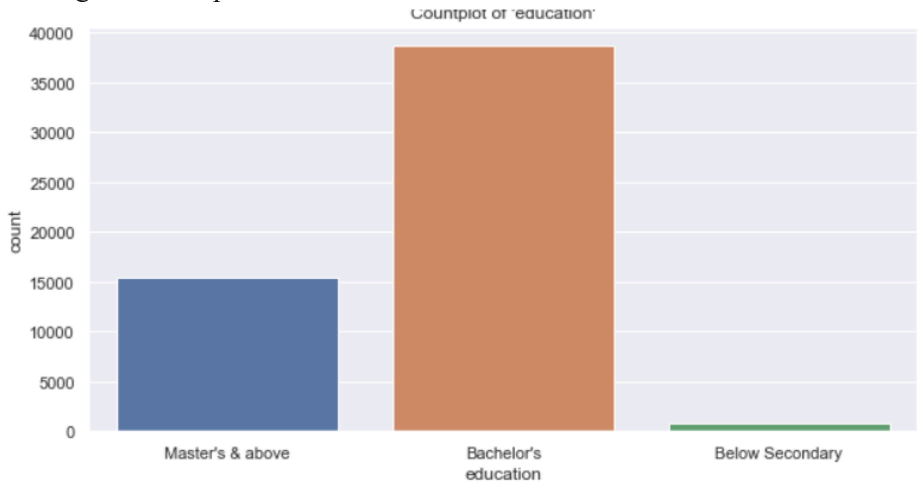
```
Value counts of 'education' before inmputation
Bachelor's          36669
Master's & above    14925
Below Secondary       805
Name: education, dtype: int64
```

We calculate the Accuracy and the F1 score then make a confusion matrix for education:

```
Accuracy score:  0.8511044450504499
F1 score:  0.8503976113455516
Classifcation report score:
              precision    recall  f1-score   support

           0       0.82      0.71      0.76      7395
           1       0.98      1.00      0.99      7276
           2       0.76      0.85      0.80      7331

    accuracy                           0.85     22002
   macro avg       0.85      0.85      0.85     22002
weighted avg       0.85      0.85      0.85     22002
```



Plotting the count plot of the different education

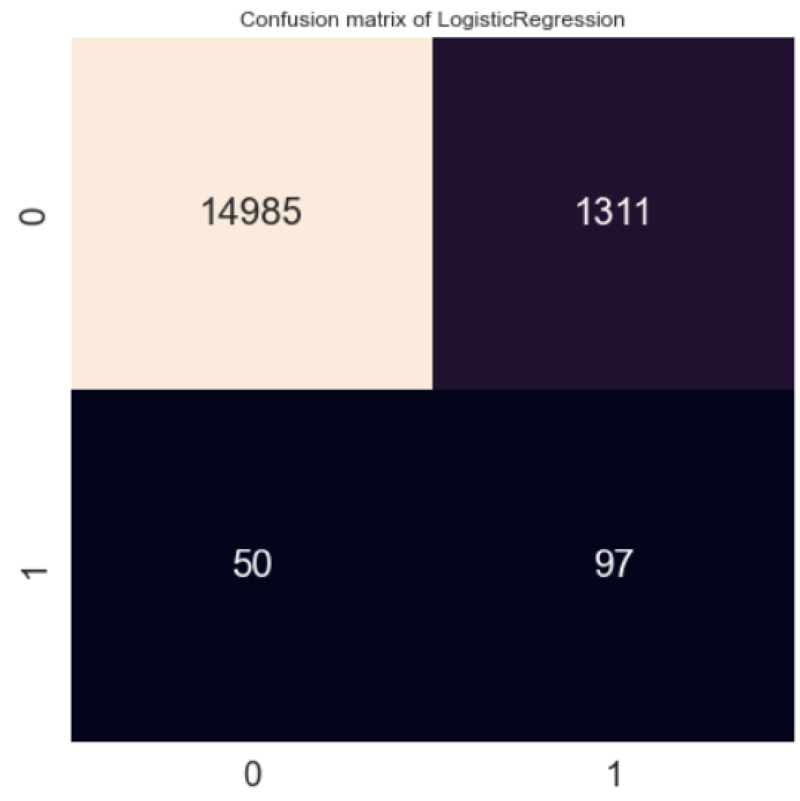


```
Value counts of 'education' after imputation
Bachelor's          38629
Master's & above    15354
Below Secondary       825
Name: education, dtype: int64
```

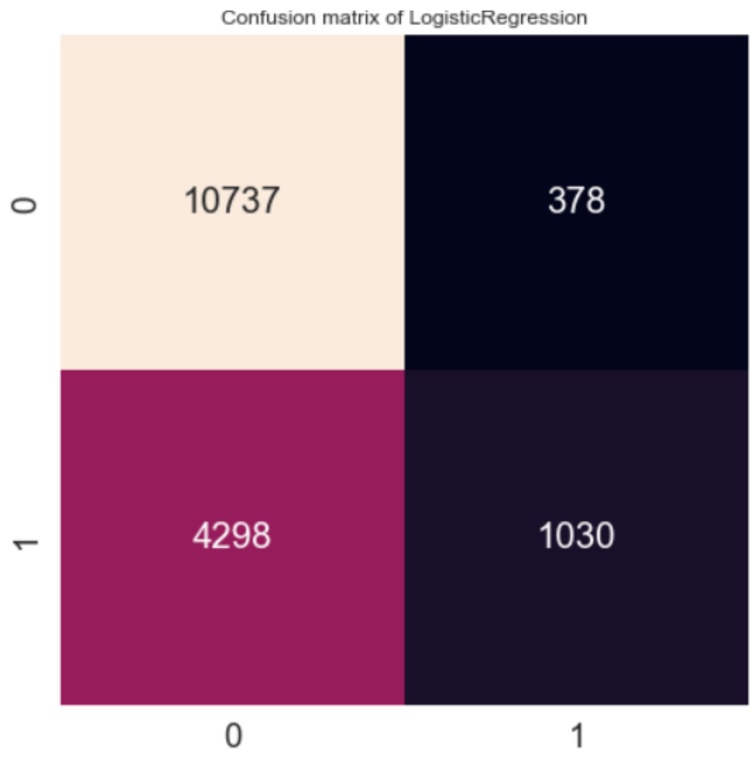
We fit the transform with Performing SMOTE for oversampling:

```
Count of 'is_promoted' before SMOTE : Counter({0: 35105, 1: 3260})
Count of 'is_promoted' after SMOTE(Undersampling) : Counter({0: 3260, 1: 3260})
Count of 'is_promoted' after SMOTE(Oversampling) : Counter({1: 35105, 0: 35105})
```

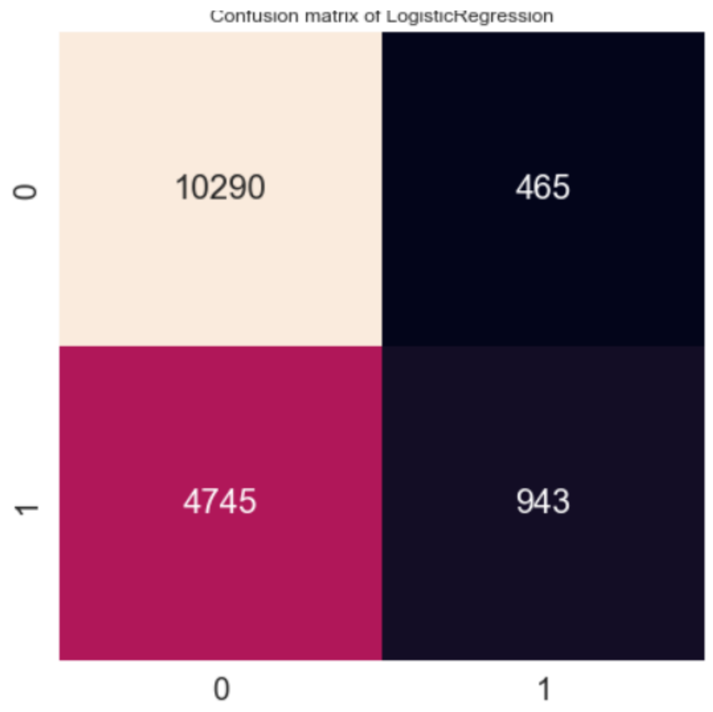
By using Logistic Regression without SMOTE:
The Accuracy K-Fold is 0.91729441095877
and the F1 Score is 0.12475884244372991
.



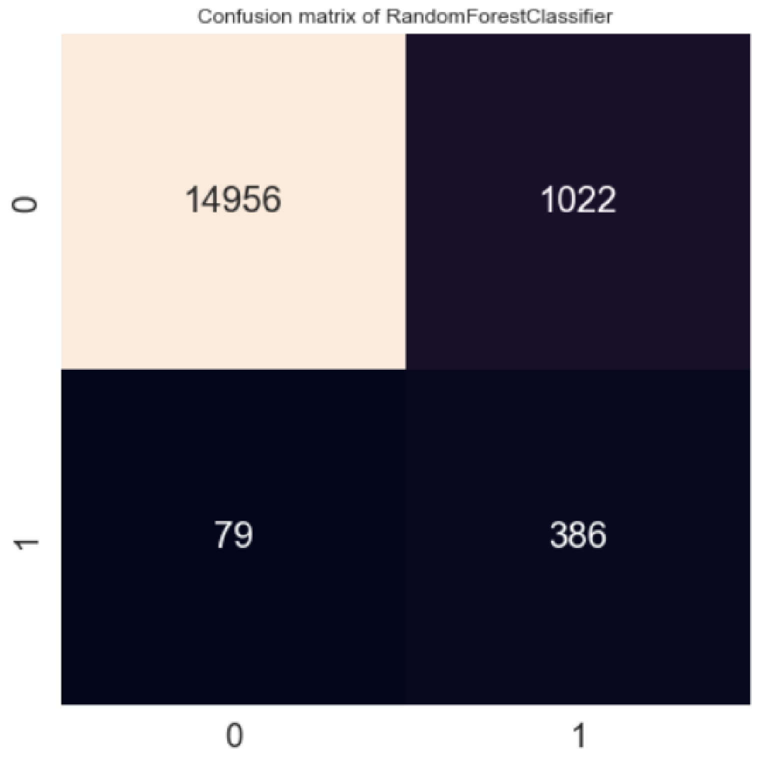
By using Logistic Regression with SMOTE(undersampling):
The Accuracy K-Fold is 0.91729441095877
and the F1 Score is 0.3058194774346793.



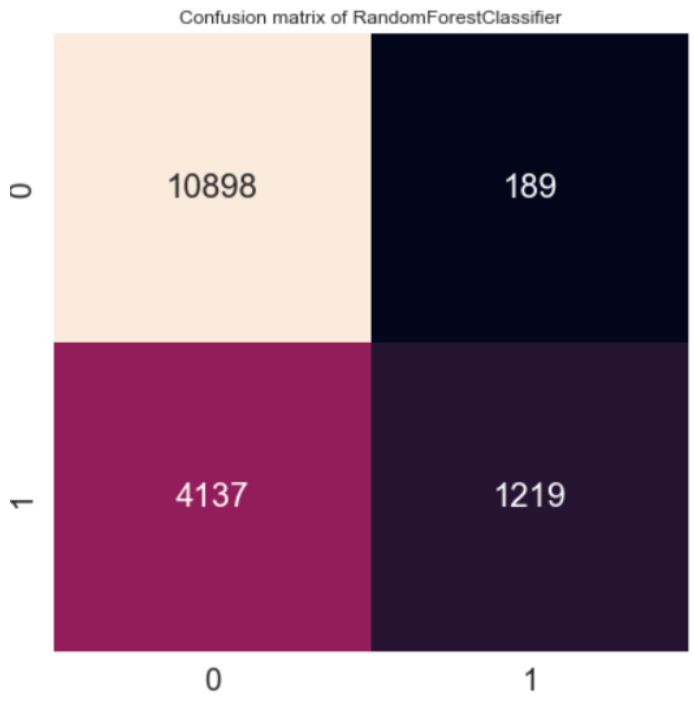
Logistic Regression with SMOTE(Oversampling):
The Accuracy K-Fold is 0.91729441095877
and the F1 Score is 0.26578354002254795



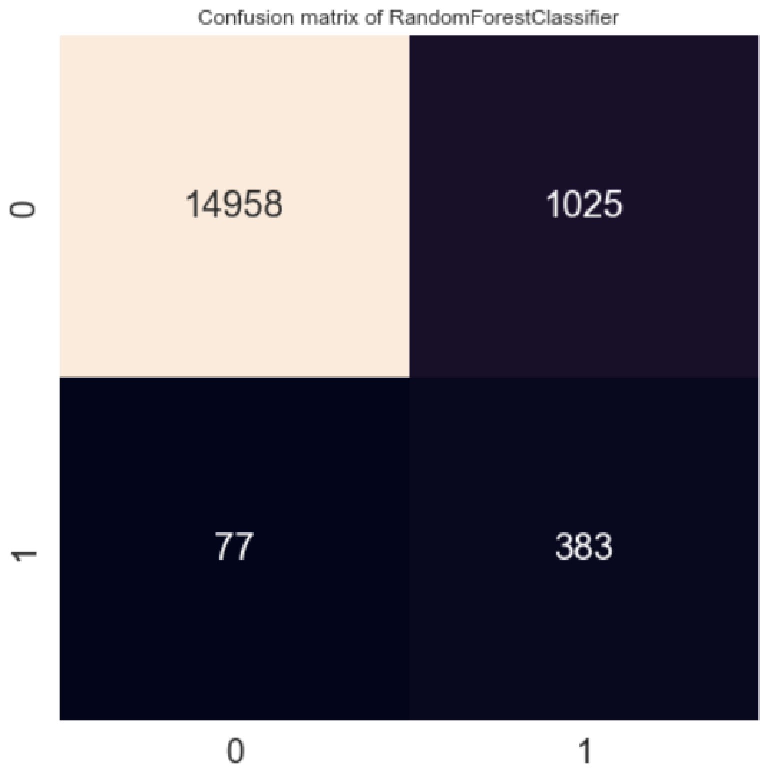
RandomForest without SMOTE:
The Accuracy K-Fold is 0.9332725944055509
and the F1 Score is 0.4121729845168179



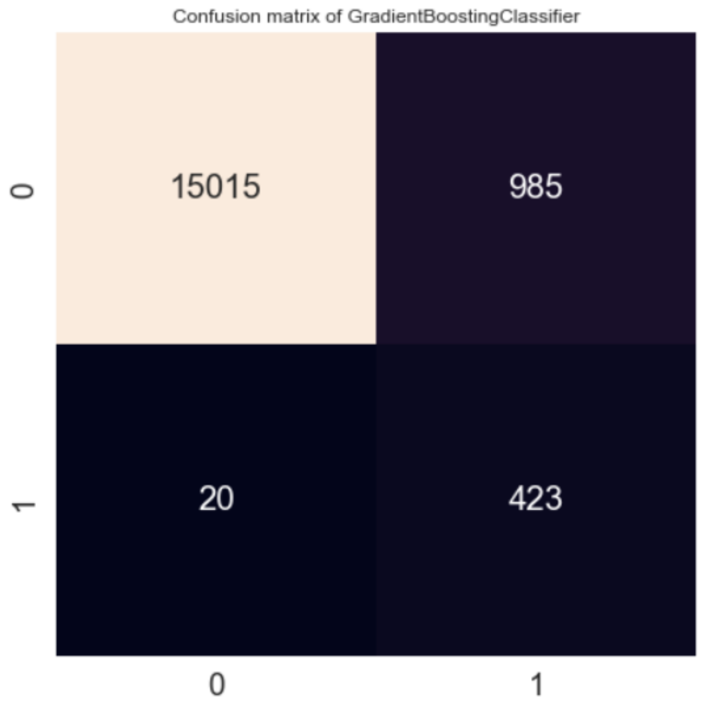
RandomForest with SMOTE(Undersampling):
The Accuracy K-Fold is 0.7599693251533742
and the F1 Score is 0.3604376108811354



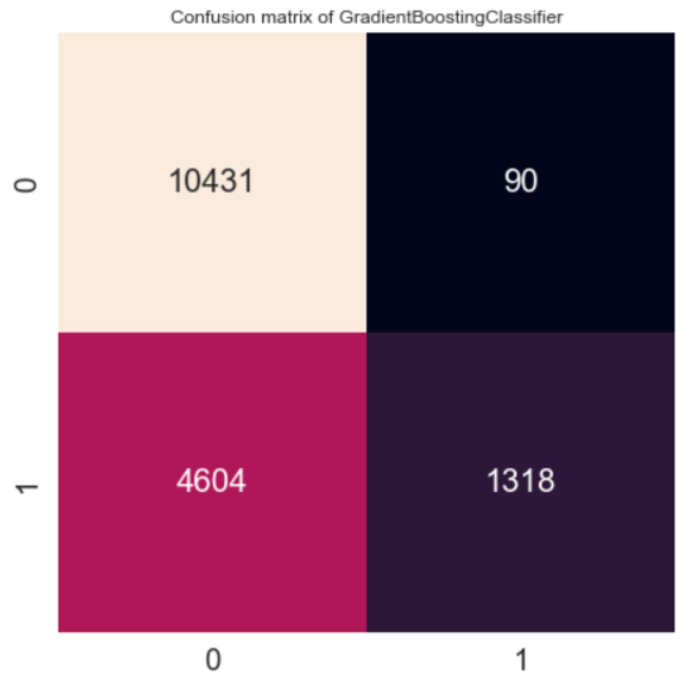
RandomForest with SMOTE(Oversampling)
The Accuracy K-Fold is 0.9331422638852315
and the F1 Score is 0.41006423982869383



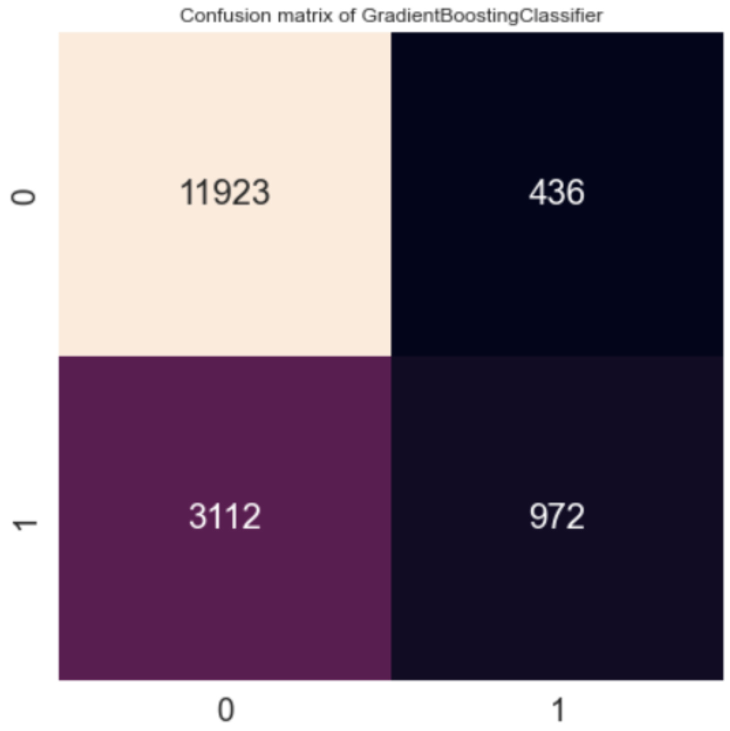
Gradient Boosting Classifier without SMOTE:
ACCURACY K-Fold: 0.9376515925420751
F1 SCORE: 0.4570502431118314



Gradient Boosting Classifier with SMOTE(Undersampling):
ACCURACY K-Fold: 0.7693251533742331
F1 SCORE: 0.35961800818553885



Gradient Boosting Classifier with SMOTE(Oversampling)
ACCURACY K-Fold: 0.7900014242985329
F1 SCORE: 0.35396941005098326



Conclusion

In conclusion, from the correlation of each parameter we can see that the promotion is not related with training, age and service. The most related is the employee’s KPI over 80%. By comparing the three methods of logistic Regression, Random Forest, and Gradient Boosting, with SMOTE or not, under or over sampling, we can say that the best idea is Gradient Boosting Classifier without SMOTE. Since it has the largest Accuray value, and the largest F1 score.