HR ANALYTICS and MODELING :

Problem Statement :

HR analytics is revolutionising the way human resources departments operate, leading to higher efficiency and better results overall. Human resources has been using analytics for years. However, the collection, processing and analysis of data has been largely manual, and given the nature of human resources dynamics and HR KPIs, the approach has been constraining HR. Therefore, it is surprising that HR departments woke up to the utility of machine learning so late in the game. Here is an opportunity to try predictive analytics in identifying the employees most likely to get promoted.

Dataset Description

Variable	Definition
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	no of other trainings 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) >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

Solution :

Problem Understanding :

1. HR are mostly following manual approach for collecting data,processing data and analysis of data which is really difficult for any big organisation to do all these things manually and come up with correct result.
2. HR team also want to make the things automated to save the time as well correctness just like many organisation is currently using chatbot to help their customers(saving customer care member time for basic queries and definitely chatbot works 24/7).
3. From last line of problem,I am able to observe that it is classification problem because employee will be either promoted or not promoted.

Sequential steps followed during problem solving

1. Exploring the data
2. Variables Identification
3. Missing Value Treatment
4. Univariate Analysis (4.1 Univariate analysis 4.2 Bivariate Analysis)
5. Bivariate Analysis (5.1 Continuous-Continuous variables 5.2 Continuous-Categorical Variables 5.3 Categorical-Categorical Variables)
6. Outliers Treatment (No requirement for our dataset)
7. Feature Engineering
8. Modeling and Evaluation (8.1 Directly applying algorithms 8.2 Downsampling 8.3 Upsampling 8.4 SMOTE)
9. Saving the models

What are the challenges and how to overcome ?

1. How to do preprocessing part if I have separate train and test data? (merge or concatenate or treat independently).

2. Missing value treatment for columns education and previous_year_rating (not directly imputed mode value or dropping these column will be fine).
3. Separating of continuous and categorical variables (sometimes it is hard when categorical variable is in numerical form and is present in large numbers like in this problem 'no_of_trainings', 'avg_training_score').
4. Target column,'is_promoted' has imbalanced(91:9) data (so to deal such type of data advance technique required but still it is better to first go through brute force approach).
5. For imbalance data,accuracy metrics(biased towards majority class value) is not the good metrics to check the performance of model (precision,recall,f1_score metrics will work for such data).
6. Feature engineering is challenging for some variables if there is ambiguity that, is this variable is nominal or ordinal in nature.
7. After simply applying four algorithms accuracy goes approx,92% (in all four cases as expected due to unbalanced class) while f1 score are 34% (in case of Logistic Regression) and 41% (in case of Random Forest).
8. To overcome from above problem (point 7),three new techniques tried that are Downsampling, Upsampling and SMOTE .
9. Downsampling, Upsampling works well compare to SMOTE and Logistic Regression and KNN consistantly perform well (accuracy,precision,recall and f1 score are constant around 75%) for Downsampling, Upsampling techniques.

Observations/Insights during preprocessing part :

1. Train dataset consist of approx 55k observations and 14 features while test dataset has 23.5k observations and 13 features.
2. education and previous_year_rating columns have missing values but very less in number so dropping is not good option.
3. Some of the graph is little bit skewed but still fine.
4. The data in train and test is almost similar,we can observe each column corresponding graph for both dataset.
5. For maximum organisation, sales and marketing department play huge role(this department is bringing customers).In our data also 35%(highest) people belong to this department only.
6. 70% have Bachelor's degree in our data and It is true for almost many organisation
7. Again we know that Male number is higher in maximum industry and same our data is also giving.70% people are male.
8. There are only few people(4%) who come in industry through reference.
9. Approx 65% people have KPI<80% and which is really correct if we see data of any company.
10. Only 2% people won the award.
11. Approx 85% people are not promoted and only 15% people are promoted.From this data we can observe class is imbalance and type 1 or type 2 error occur.We can't use accuracy here.
12. The person with age 27-35(binning option) has higher chance of promotion.

In [1]:

```
# importing common used library

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:

```
# reading the datasets

train=pd.read_csv("train.csv")
test=pd.read_csv("test.csv")

# copying the datasets

train_copy=train.copy()
test_copy=test.copy()

# checking dimension of datasets

print("train data dimension : ",train.shape)
print("test data dimension : ",test.shape)
```

```
train data dimension :  (54808, 14)
test data dimension :  (23490, 13)
```

In [3]:

```
# accessing first five rows of train dataset

print("*"*71)
print("first five rows of train data :-")
print("*"*71)
```

```
print("*"*71)
train.head()
```

```
***********************************************************************
first five rows of train data :-
***********************************************************************
```

Out[3]:

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

In [4]:

```
# accessing first five rows of test dataset

print("*"*71)
print("first five rows of test data:-")
print("*"*71)
test.head()
```

```
***********************************************************************
first five rows of test data:-
***********************************************************************
```

Out[4]:

	employee_id	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_year_rating	length_of_
0	8724	Technology	region_26	Bachelor's	m	sourcing	1	24	NaN	
1	74430	HR	region_4	Bachelor's	f	other	1	31	3.0	
2	72255	Sales & Marketing	region_13	Bachelor's	m	other	1	31	1.0	
3	38562	Procurement	region_2	Bachelor's	f	other	3	31	2.0	
4	64486	Finance	region_29	Bachelor's	m	sourcing	1	30	4.0	

Can I merge(side by side) or concatenate(on top or bottom) or treat both datasets independently during preprocessing?

Case 1: Both datasets have same columns except target column.So merge will not work.

Case 2: concatenate will work upto some extent but still have problem due to target column.

Case 3: It will work If I treat both datasets independently during preprocessing.

2. Variables Identification:

In [5]:

```
# checking datatype of each columns and any missing values

print(train.info())
print("*"*71)
print("*"*71)
print(test.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54808 entries, 0 to 54807
Data columns (total 14 columns):
employee_id             54808 non-null int64
department              54808 non-null object
region                  54808 non-null object
education               52399 non-null object
gender                  54808 non-null object
recruitment_channel     54808 non-null object
no_of_trainings         54808 non-null int64
age                     54808 non-null int64
previous_year_rating    50684 non-null float64
length_of_service       54808 non-null int64
KPIs_met >80%           54808 non-null int64
awards_won?             54808 non-null int64
avg_training_score      54808 non-null int64
is_promoted             54808 non-null int64
dtypes: float64(1), int64(8), object(5)
memory usage: 5.9+ MB
None
***********************************************************************
***********************************************************************
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23490 entries, 0 to 23489
Data columns (total 13 columns):
employee_id             23490 non-null int64
department              23490 non-null object
region                  23490 non-null object
education               22456 non-null object
gender                  23490 non-null object
recruitment_channel     23490 non-null object
no_of_trainings         23490 non-null int64
age                     23490 non-null int64
previous_year_rating    21678 non-null float64
length_of_service       23490 non-null int64
KPIs_met >80%           23490 non-null int64
awards_won?             23490 non-null int64
avg_training_score      23490 non-null int64
dtypes: float64(1), int64(7), object(5)
memory usage: 2.3+ MB
None
```

Observations :

1.I have observed both train and test data have missing values for these two columns (that is education and previous_year_rating) and all other columns don't have any missing values.

3. Missing Value Treatment:

In [6]:

```
#Check the percentage of null values per variable

print("% of null values for each column:\n",train.isnull().sum()/train.shape[0]*100)
print("*"*71)
print("*"*71)
print("% of null values for each column:\n",test.isnull().sum()/test.shape[0]*100)
```

```
% of null values for each column:
 employee_id             0.000000
department              0.000000
region                  0.000000
education               4.395344
gender                  0.000000
recruitment_channel     0.000000
no_of_trainings         0.000000
age                     0.000000
previous_year_rating    7.524449
length_of_service       0.000000
KPIs_met >80%           0.000000
awards_won?             0.000000
```

```
avg_training_score      0.000000
is_promoted             0.000000
dtype: float64
***********************************************************************
***********************************************************************
% of null values for each column:
 employee_id             0.000000
department              0.000000
region                  0.000000
education               4.401873
gender                  0.000000
recruitment_channel     0.000000
no_of_trainings         0.000000
age                     0.000000
previous_year_rating    7.713921
length_of_service       0.000000
KPIs_met >80%           0.000000
awards_won?             0.000000
avg_training_score      0.000000
dtype: float64
```

Observations :

1. Number of value or % of missing value in columns education and previous_year_rating is very less so I can't drop any of these columns
2. Both columns are catgorical in nature so definitely we can't use mean or median
3. We can use mode but if we directly use mode for both columns then there may be chance of biased(why see next point)
4. education column is more related to department hence we find mode for each department corresponding to education column from dataset (which don't contain null value)
5. previous_year_rating is more related to no_of_trainings(it also belong to previous year as given in data dictionary) hence we find mode for each no_of_trainings corresponding to previous_year_rating column from dataset (which don't contain null value)
6. Points 4 and 5 become more clear from next few lines of code

In [7]:

```
# checking which values appears how much in terms of percentage

print("For train data :")
print(train['education'].value_counts(normalize=True))
print("*"*71)
print(train['previous_year_rating'].value_counts(normalize=True))

print("*"*71)
print("*"*71)

print("For test data :")
print(test['education'].value_counts(normalize=True))
print("*"*71)
print(test['previous_year_rating'].value_counts(normalize=True))
```

```
For train data :
Bachelor's          0.699803
Master's & above    0.284834
Below Secondary     0.015363
Name: education, dtype: float64
***********************************************************************
3.0    0.367335
5.0    0.231651
4.0    0.194874
1.0    0.122780
2.0    0.083360
Name: previous_year_rating, dtype: float64
***********************************************************************
***********************************************************************
For test data :
Bachelor's          0.693712
Master's & above    0.289633
Below Secondary     0.016655
Name: education, dtype: float64
***********************************************************************
3.0    0.365393
5.0    0.235123
```

```
4.0     0.196005
1.0     0.123628
2.0     0.079851
Name: previous_year_rating, dtype: float64
```

Observations :

1. Mode value corresponding to column eucation is Bachelor's for both train and test data
2. Mode value corresponding to column previous_year_rating is 3.0 for both train and test data
3. These above two mode are calculated from whole dataset (separately for both datasets)

In [8]:

```
# checking occurence using crosstab for train and test data

train_not_null=train[train['education'].isnull()!=True]

pd.crosstab(index=train_not_null["education"], columns=train_not_null["department"])
```

Out[8]:

department	Analytics	Finance	HR	Legal	Operations	Procurement	R&D	Sales & Marketing	Technology
education									
Bachelor's	3978	1895	1525	814	7781	4393	542	11099	4642
Below Secondary	0	106	128	65	176	129	0	0	201
Master's & above	1037	499	733	156	3165	2544	429	4166	2196

Observations :

1. For each department,"Bachelor's" education is dominating so mode value is coming same after even performing some logic.

In [9]:

```
train_not_null_1=train[train['previous_year_rating'].isnull()!=True]


pd.crosstab(index=train_not_null_1["previous_year_rating"],
columns=train_not_null_1["no_of_trainings"])
```

Out[9]:

no_of_trainings	1	2	3	4	5	6	7	8	9	10
previous_year_rating										
1.0	4864	989	253	76	28	10	1	1	0	1
2.0	3508	539	123	40	10	2	2	0	1	0
3.0	14584	3047	719	188	55	14	3	2	2	4
4.0	8300	1235	252	61	19	7	2	1	0	0
5.0	9904	1514	267	46	7	1	2	0	0	0

Observations :

1. For each no_of_trainings,"3.0" rating is dominating so mode value is coming same after even performing some logic.

In [10]:

```
test_not_null=test[test['education'].isnull()!=True]

pd.crosstab(index=test_not_null["education"], columns=test_not_null["department"])
```

Out[10]:

department education	Analytics	Finance	HR	Legal	Operations	Procurement	R&D	Sales & Marketing	Technology
Bachelor's	1703	788	685	336	3298	1807	227	4818	1916
Below Secondary	0	49	58	35	73	72	0	0	87
Master's & above	461	240	328	72	1301	1101	201	1834	966

Observations :

1. For each department,"Bachelor's" education is dominating so mode value is coming same after even performing some logic.Same is coming for train data.

In [11]:

```
test_not_null_1=test[test['education'].isnull()!=True]

pd.crosstab(index=test_not_null_1["previous_year_rating"],
columns=test_not_null_1["no_of_trainings"])
```

Out[11]:

no_of_trainings previous_year_rating	1	2	3	4	5	6	7	8	9
1.0	2008	401	96	30	2	2	2	0	0
2.0	1385	207	48	15	1	2	0	0	0
3.0	5836	1332	283	76	20	12	5	0	1
4.0	3430	569	85	22	6	1	0	1	1
5.0	4131	656	127	18	4	2	1	1	0

Observations :

1. For each no_of_trainings,"3.0" rating is dominating so mode value is coming same after even performing some logic.Same is coming for train data.

In [12]:

```
# now filling mode value in both columns where value is missing

train['education']=train['education'].fillna("Bachelor's")
train['previous_year_rating']=train['previous_year_rating'].fillna(3.0)
test['education']=train['education'].fillna("Bachelor's")
test['previous_year_rating']=test['previous_year_rating'].fillna(3.0)

# checking missing value
print(train.info())
print("*"*71)
print("*"*71)
print(test.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54808 entries, 0 to 54807
Data columns (total 14 columns):
employee_id             54808 non-null int64
department              54808 non-null object
region                  54808 non-null object
education               54808 non-null object
gender                  54808 non-null object
recruitment_channel     54808 non-null object
no_of_trainings         54808 non-null int64
age                     54808 non-null int64
```

```
previous_year_rating    54808 non-null float64
length_of_service       54808 non-null int64
KPIs_met >80%           54808 non-null int64
awards_won?             54808 non-null int64
avg_training_score      54808 non-null int64
is_promoted             54808 non-null int64
dtypes: float64(1), int64(8), object(5)
memory usage: 5.9+ MB
None
***********************************************************************
***********************************************************************
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23490 entries, 0 to 23489
Data columns (total 13 columns):
employee_id             23490 non-null int64
department              23490 non-null object
region                  23490 non-null object
education               23490 non-null object
gender                  23490 non-null object
recruitment_channel     23490 non-null object
no_of_trainings         23490 non-null int64
age                     23490 non-null int64
previous_year_rating    23490 non-null float64
length_of_service       23490 non-null int64
KPIs_met >80%           23490 non-null int64
awards_won?             23490 non-null int64
avg_training_score      23490 non-null int64
dtypes: float64(1), int64(7), object(5)
memory usage: 2.3+ MB
None
```

4. Univariate Analysis :

In [13]:

```
# dropping employee_id from both dataset as it is like primary key so it will not help

train=train.drop('employee_id',1)
test=test.drop('employee_id',1)
```

In [14]:

```
# first deciding which variable are continuous and which are categorical in nature for
# further analysis

for i in train.columns.tolist():
    print("Column : ",i)
    print("Number of unique value in each column : ",train[i].nunique())
    print(train[i].value_counts())
    print("*"*71)

print("*"*71)
print("*"*71)

for i in test.columns.tolist():
    print("Column : ",i)
    print("Number of unique value in each column : ",test[i].nunique())
    print(test[i].value_counts(normalize=True))
    print("*"*71)
```

```
Column :  department
Number of unique value in each column :  9
Sales & Marketing    16840
Operations           11348
Procurement           7138
Technology            7138
Analytics             5352
Finance               2536
HR                    2418
Legal                 1039
R&D                    999
Name: department, dtype: int64
***********************************************************************
```

```
Column :  region
Number of unique value in each column :  34
region_2     12343
region_22     6428
region_7      4843
region_15     2808
region_13     2648
region_26     2260
region_31     1935
region_4      1703
region_27     1659
region_16     1465
region_28     1318
region_11     1315
region_23     1175
region_29      994
region_32      945
region_19      874
region_20      850
region_14      827
region_25      819
region_17      796
region_5       766
region_6       690
region_30      657
region_8       655
region_10      648
region_1       610
region_24      508
region_12      500
region_9       420
region_21      411
region_3       346
region_34      292
region_33      269
region_18       31
Name: region, dtype: int64
**********************************************************************
Column :  education
Number of unique value in each column :  3
Bachelor's          39078
Master's & above    14925
Below Secondary       805
Name: education, dtype: int64
**********************************************************************
Column :  gender
Number of unique value in each column :  2
m    38496
f    16312
Name: gender, dtype: int64
**********************************************************************
Column :  recruitment_channel
Number of unique value in each column :  3
other       30446
sourcing    23220
referred     1142
Name: recruitment_channel, dtype: int64
**********************************************************************
Column :  no_of_trainings
Number of unique value in each column :  10
1     44378
2      7987
3      1776
4       468
5       128
6        44
7        12
10        5
9         5
8         5
Name: no_of_trainings, dtype: int64
**********************************************************************
Column :  age
Number of unique value in each column :  41
30    3665
31    3534
32    3534
```

```
29    3405
33    3210
28    3147
34    3076
27    2827
35    2711
36    2517
37    2165
26    2060
38    1923
39    1695
40    1663
25    1299
41    1289
42    1149
43     992
44     847
24     845
45     760
46     697
47     557
48     557
50     521
49     441
23     428
51     389
53     364
52     351
54     313
55     294
56     264
57     238
22     231
60     217
58     213
59     209
20     113
21      98
Name: age, dtype: int64
***********************************************************************
Column :  previous_year_rating
Number of unique value in each column :  5
3.0    22742
5.0    11741
4.0     9877
1.0     6223
2.0     4225
Name: previous_year_rating, dtype: int64
***********************************************************************
Column :  length_of_service
Number of unique value in each column :  35
3     7033
4     6836
2     6684
5     5832
7     5551
6     4734
1     4547
8     2883
9     2629
10    2193
11     916
12     794
13     687
15     593
14     549
16     548
17     432
18     392
19     329
20     128
21      78
24      70
23      65
22      61
25      51
26      41
```

```
26      41
27      36
28      30
29      30
31      20
30      12
32      10
33       9
34       4
37       1
Name: length_of_service, dtype: int64
**********************************************************************
Column :  KPIs_met >80%
Number of unique value in each column :  2
0    35517
1    19291
Name: KPIs_met >80%, dtype: int64
**********************************************************************
Column :  awards_won?
Number of unique value in each column :  2
0    53538
1     1270
Name: awards_won?, dtype: int64
**********************************************************************
Column :  avg_training_score
Number of unique value in each column :  61
50    2716
49    2681
48    2437
51    2347
60    2155
59    2064
58    1898
61    1879
52    1856
47    1746
62    1450
82    1447
57    1437
81    1357
53    1324
80    1206
83    1198
84    1168
79    1160
46    1136
85    1072
56    1070
70    1055
63    1021
69    1018
54     997
68     935
78     933
86     912
71     898
      ...
67     728
72     725
64     708
77     697
45     681
87     655
65     599
66     580
73     523
76     516
88     444
74     433
75     403
44     335
89     301
90     185
43     176
91     117
92      99
93      84
94      65
```

```
94      65
42      62
97      49
96      48
95      45
98      37
99      35
41      26
40       5
39       2
Name: avg_training_score, Length: 61, dtype: int64
***********************************************************************
Column :  is_promoted
Number of unique value in each column :  2
0    50140
1     4668
Name: is_promoted, dtype: int64
***********************************************************************
***********************************************************************
***********************************************************************
Column :  department
Number of unique value in each column :  9
Sales & Marketing    0.311409
Operations           0.202810
Procurement          0.128565
Technology           0.128182
Analytics            0.098723
Finance              0.046445
HR                   0.046190
Legal                0.018944
R&D                  0.018731
Name: department, dtype: float64
***********************************************************************
Column :  region
Number of unique value in each column :  34
region_2     0.225585
region_22    0.116603
region_7     0.084376
region_13    0.049681
region_15    0.048106
region_26    0.043040
region_31    0.035930
region_4     0.032993
region_27    0.030226
region_28    0.025330
region_16    0.025117
region_11    0.024308
region_23    0.021967
region_32    0.018433
region_29    0.017625
region_19    0.017454
region_17    0.015368
region_14    0.014900
region_5     0.014559
region_25    0.014347
region_20    0.013878
region_6     0.012686
region_30    0.011622
region_8     0.011452
region_10    0.011452
region_1     0.010132
region_24    0.009323
region_12    0.009153
region_9     0.007663
region_21    0.007620
region_34    0.006599
region_3     0.006258
region_33    0.005364
region_18    0.000851
Name: region, dtype: float64
***********************************************************************
Column :  education
Number of unique value in each column :  3
Bachelor's          0.711069
Master's & above    0.274415
Below Secondary     0.014517
Name: education, dtype: float64
***********************************************************************
```

```
*********************************************************************
Column :  gender
Number of unique value in each column :  2
m    0.706513
f    0.293487
Name: gender, dtype: float64
*********************************************************************
Column :  recruitment_channel
Number of unique value in each column :  3
other       0.556748
sourcing    0.424053
referred    0.019200
Name: recruitment_channel, dtype: float64
*********************************************************************
Column :  no_of_trainings
Number of unique value in each column :  9
1    0.805790
2    0.150873
3    0.032141
4    0.008089
5    0.001533
6    0.001022
7    0.000383
9    0.000085
8    0.000085
Name: no_of_trainings, dtype: float64
*********************************************************************
Column :  age
Number of unique value in each column :  41
30    0.067901
31    0.064964
32    0.063048
29    0.061984
33    0.059046
28    0.057982
34    0.056237
27    0.052533
35    0.049766
36    0.043508
37    0.038272
38    0.036696
26    0.036484
39    0.031971
40    0.028736
25    0.024947
41    0.024904
42    0.020221
43    0.018519
44    0.015155
24    0.014304
45    0.012899
46    0.011792
47    0.010898
48    0.009366
23    0.009110
49    0.008897
50    0.008727
52    0.007109
51    0.006854
53    0.006556
54    0.005917
55    0.005747
56    0.005151
58    0.004427
22    0.004172
57    0.004044
60    0.003789
59    0.003533
20    0.002171
21    0.001660
Name: age, dtype: float64
*********************************************************************
Column :  previous_year_rating
Number of unique value in each column :  5
3.0    0.414347
5.0    0.216986
4.0    0.180885
1.0    0.114001
```

```
1.0    0.114091
2.0    0.073691
Name: previous_year_rating, dtype: float64
************************************************************************
Column :  length_of_service
Number of unique value in each column :  34
3     0.129119
4     0.123925
2     0.122180
5     0.110345
7     0.099787
6     0.086760
1     0.084802
8     0.053768
9     0.045551
10    0.040060
11    0.014985
12    0.013963
13    0.011537
14    0.011111
15    0.010217
16    0.009025
17    0.008685
18    0.007067
19    0.006045
20    0.002639
23    0.001320
21    0.001235
22    0.001149
25    0.001022
24    0.000766
27    0.000681
26    0.000596
28    0.000511
29    0.000426
31    0.000298
30    0.000255
34    0.000085
33    0.000043
32    0.000043
Name: length_of_service, dtype: float64
************************************************************************
Column :  KPIs_met >80%
Number of unique value in each column :  2
0    0.641166
1    0.358834
Name: KPIs_met >80%, dtype: float64
************************************************************************
Column :  awards_won?
Number of unique value in each column :  2
0    0.977224
1    0.022776
Name: awards_won?, dtype: float64
************************************************************************
Column :  avg_training_score
Number of unique value in each column :  61
50    0.052235
49    0.048702
48    0.046147
51    0.043550
60    0.037846
59    0.037335
52    0.033674
58    0.033589
47    0.033504
61    0.032695
57    0.026948
62    0.026905
81    0.024138
80    0.023968
53    0.023457
83    0.023457
84    0.023287
82    0.022435
46    0.021541
70    0.020732
79    0.019838
```

```
69     0.019072
85     0.018433
63     0.018178
54     0.018050
78     0.016943
71     0.016943
86     0.016816
56     0.016773
55     0.016645
         ...
64     0.014474
45     0.013495
67     0.013325
77     0.012899
72     0.012686
87     0.012473
73     0.010175
65     0.010175
66     0.009621
76     0.009579
88     0.007365
74     0.007237
75     0.007237
44     0.006854
89     0.005619
90     0.003491
43     0.002895
91     0.001916
92     0.001533
94     0.001320
96     0.001149
93     0.001107
95     0.000979
97     0.000937
42     0.000681
98     0.000596
99     0.000553
41     0.000468
40     0.000128
39     0.000043
Name: avg_training_score, Length: 61, dtype: float64
***********************************************************************
```

In [15]:

```
# separating categorical and continuous variables

con_var=['no_of_trainings','age','length_of_service','avg_training_score']
cat_var=['department','region','education','gender','recruitment_channel',
         'previous_year_rating','KPIs_met >80%','awards_won?','is_promoted']

# this cat_var_1 is only for test data(it doesn't contain target variable)
cat_var_1=['department','region','education','gender','recruitment_channel',
         'previous_year_rating','KPIs_met >80%','awards_won?']

tra_con=train[con_var]
tra_cat=train[cat_var]

tes_con=test[con_var]
tes_cat=test[cat_var_1]
```

4.1 Continuous Variables :

In [16]:

```
# checking summary statistics

tra_con.describe()
```

Out[16]:

	no_of_trainings	age	length_of_service	avg_training_score

	no_of_trainings	age	length_of_service	avg_training_score
count	54808.000000	54808.000000	54808.000000	54808.000000
mean	1.253011	34.803915	5.865512	63.386750
std	0.609264	7.660169	4.265094	13.371559
min	1.000000	20.000000	1.000000	39.000000
25%	1.000000	29.000000	3.000000	51.000000
50%	1.000000	33.000000	5.000000	60.000000
75%	1.000000	39.000000	7.000000	76.000000
max	10.000000	60.000000	37.000000	99.000000

Observations :

1. From min,max,mean and 50% ,we can little bit describe related to skewness but histogram give crystal clear picture

In [17]:

```
tes_con.describe()
```

Out[17]:

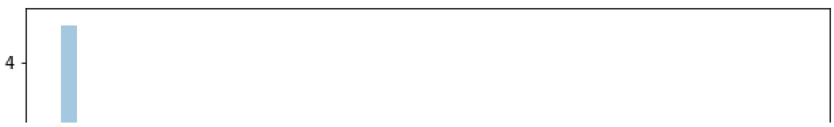
	no_of_trainings	age	length_of_service	avg_training_score
count	23490.000000	23490.000000	23490.000000	23490.000000
mean	1.254236	34.782929	5.810387	63.263133
std	0.600910	7.679492	4.207917	13.411750
min	1.000000	20.000000	1.000000	39.000000
25%	1.000000	29.000000	3.000000	51.000000
50%	1.000000	33.000000	5.000000	60.000000
75%	1.000000	39.000000	7.000000	76.000000
max	9.000000	60.000000	34.000000	99.000000

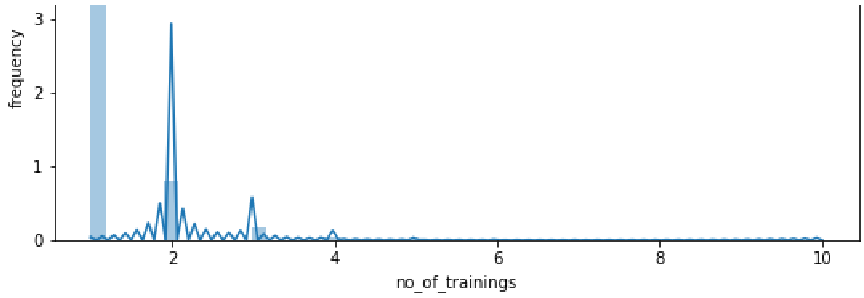
In [18]:

```
print("For train data :")
for i in tra_con.columns.tolist():
    plt.figure(figsize=(9,4))
    sns.distplot(tra_con[i])
    plt.ylabel('frequency')
    plt.show()
    print("Skewness : ",tra_con[i].skew())

print("*"*71)

print("For test data :")
for i in tes_con.columns.tolist():
    plt.figure(figsize=(9,4))
    sns.distplot(tes_con[i])
    plt.ylabel('frequency')
    plt.show()
    print("Skewness : ",tes_con[i].skew())
```

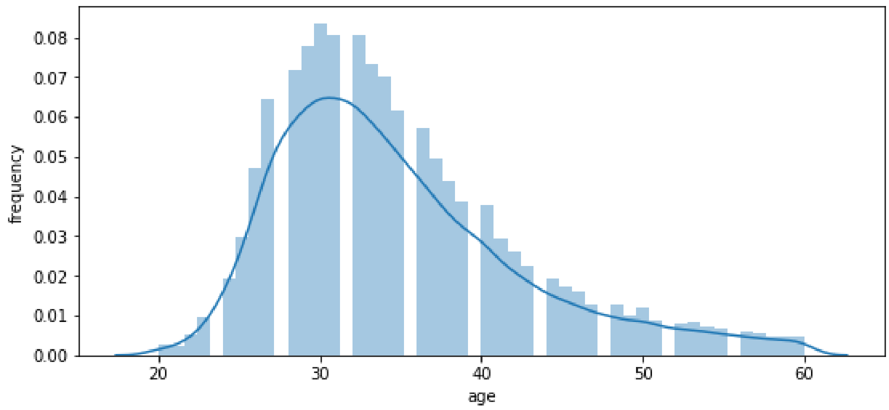
```
For train data :
```

```
C:\Users\Purushottam\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

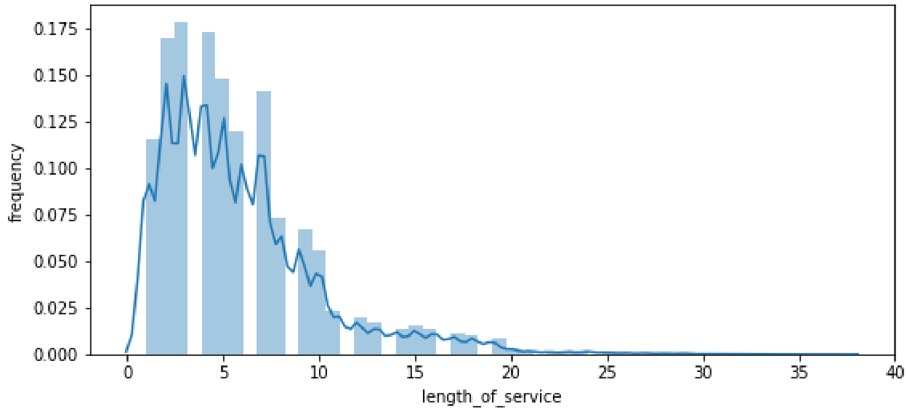




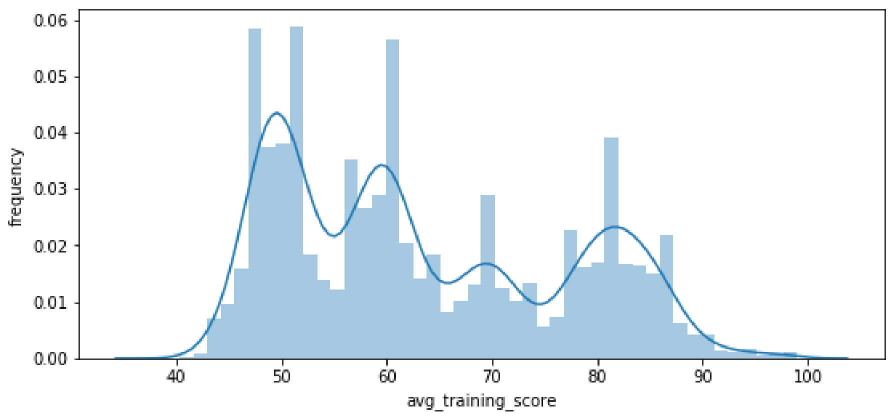

```
Skewness :  3.445433937567454
```



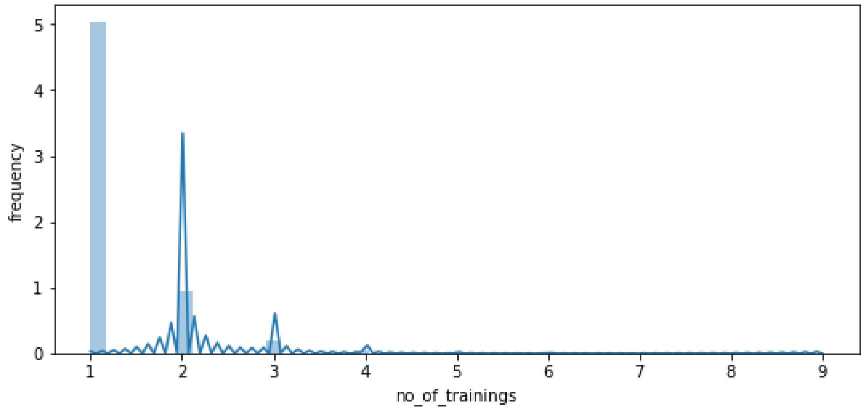

```
Skewness :  1.0074317710382241
```



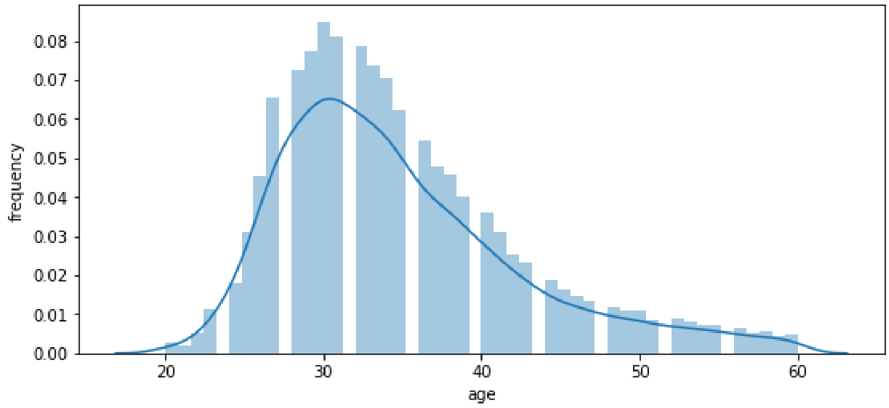

```
Skewness :  1.738061458740809
```



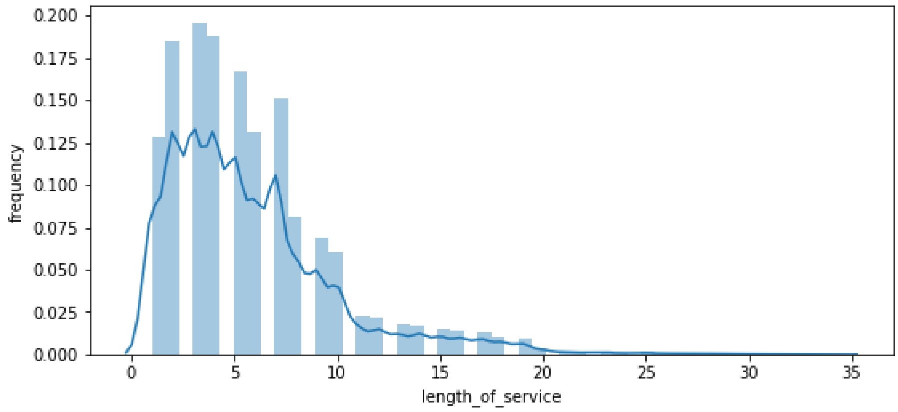

```
Skewness :  0.45190808551707995
*************************************************************************
For test data :
```



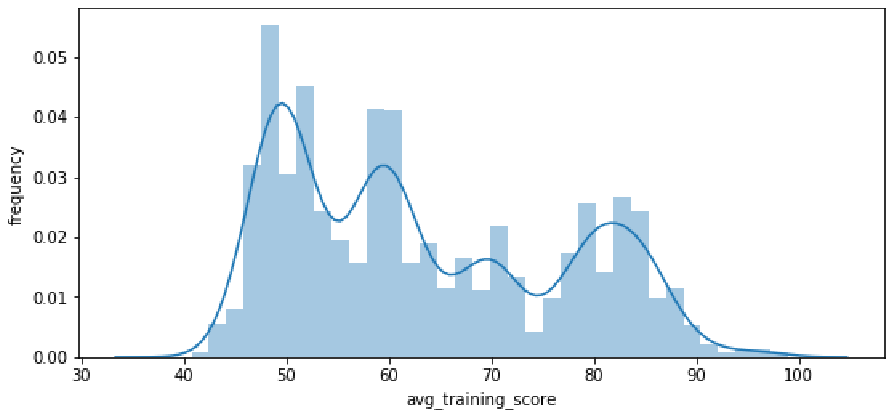

```
Skewness :  3.299829240353573
```




```
Skewness :  1.0117773507112382
```




```
Skewness :  1.7030008581343499
```



```
Skewness :  0.4581131346053442
```

Observations :

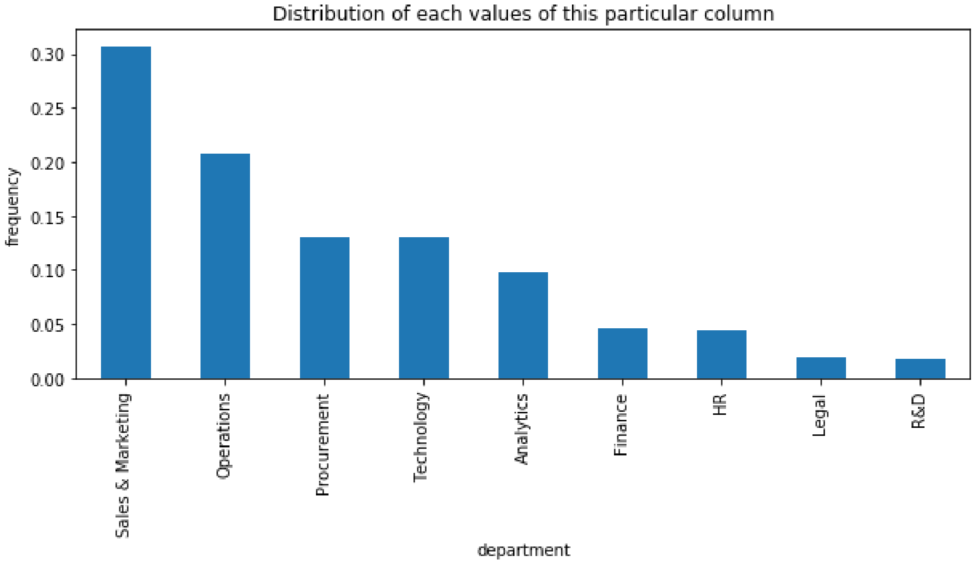
1. Some of the graph is little bit skewed but still fine.
2. The data in train and test is almost similar,we can observe each column corresponding graph for both dataset.

4.2 Categorical Variables :

In [19]:

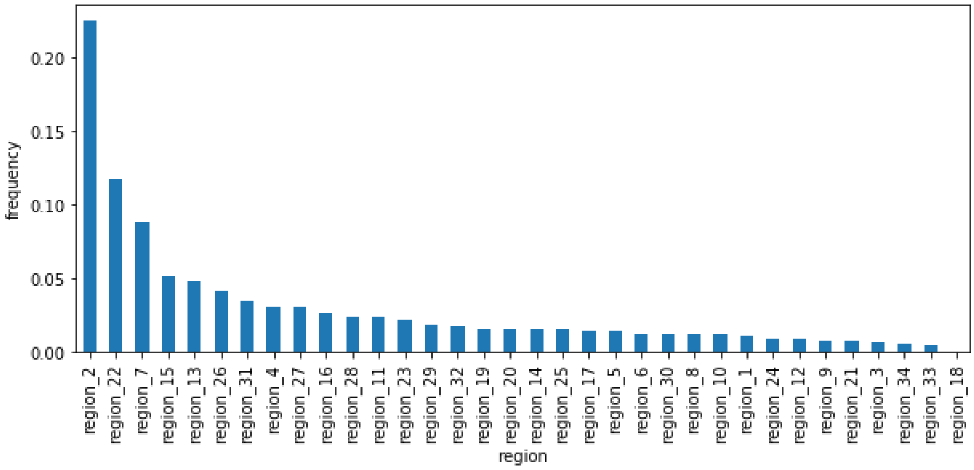
```
# plotting barplot to observe the % of value for each column

print("train categorical data:\n")
for i in tra_cat.columns.tolist():
    print("Column : ",i)
    plt.figure(figsize=(10,4))
    tra_cat[i].value_counts(normalize=True).plot.bar()
    plt.title("Distribution of each values of this particular column")
    plt.xlabel(i)
    plt.ylabel("frequency")
    plt.show()
    print("*"*71)


print("*"*71)
print("*"*71)

print("test categorical data:\n")
for i in tes_cat.columns.tolist():
    print("Column : ",i)
    plt.figure(figsize=(10,4))
    tes_cat[i].value_counts(normalize=True).plot.bar()
    plt.title("Distribution of each values of this particular column")
    plt.xlabel(i)
    plt.ylabel("frequency")
    plt.show()
    print("*"*71)
```

```
train categorical data:

Column :  department
```



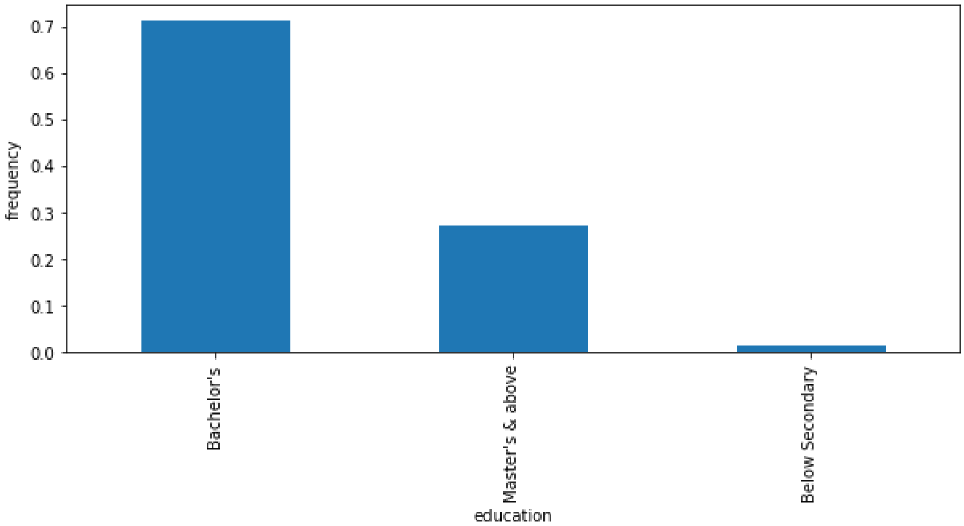

```
***********************************************************************
Column :  region
```

Distribution of each values of this particular column



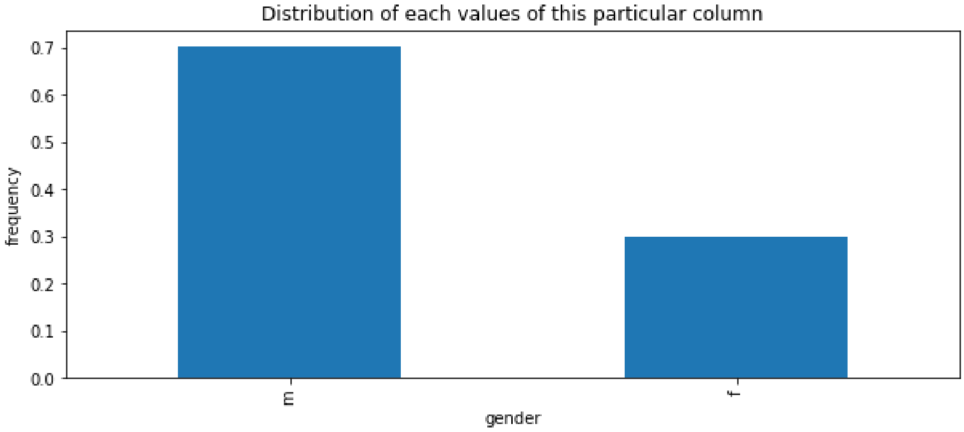

```
************************************************************************
Column :  education
```

Distribution of each values of this particular column



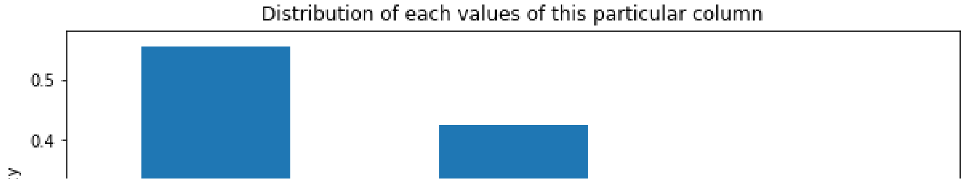

```
************************************************************************
Column :  gender
```

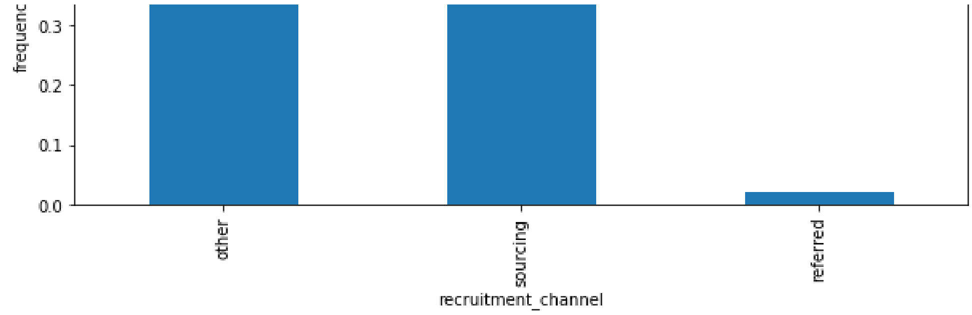
Distribution of each values of this particular column



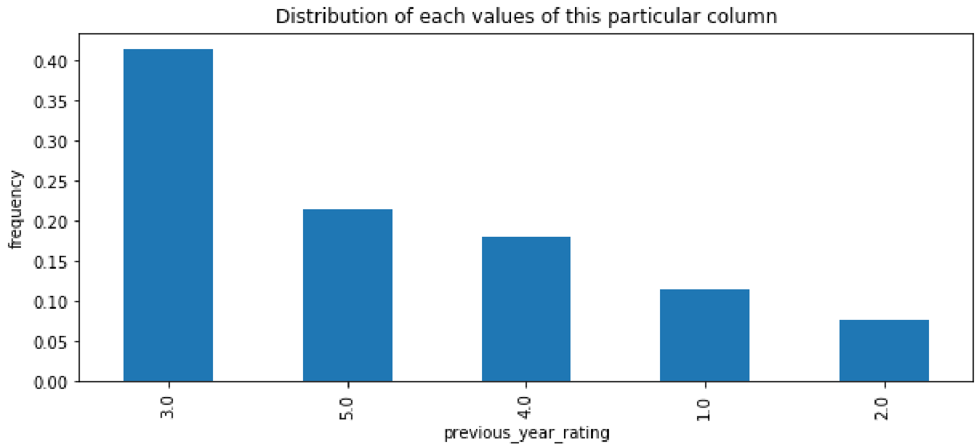

```
************************************************************************
Column :  recruitment_channel
```

Distribution of each values of this particular column

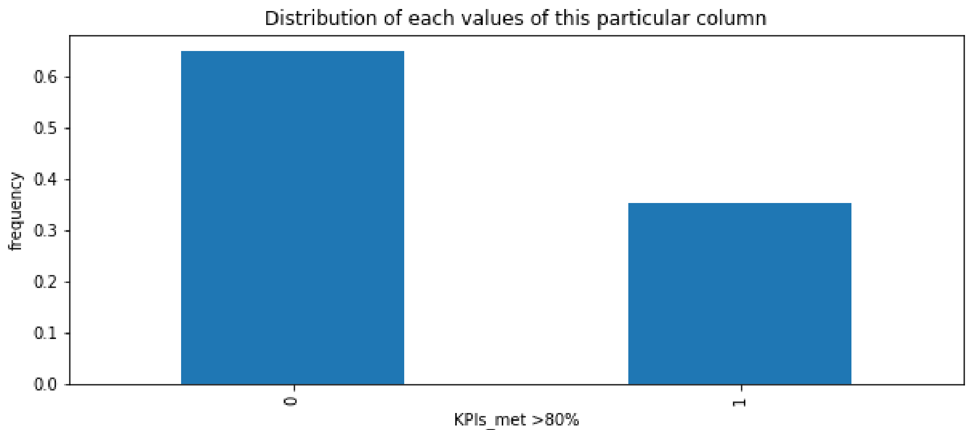




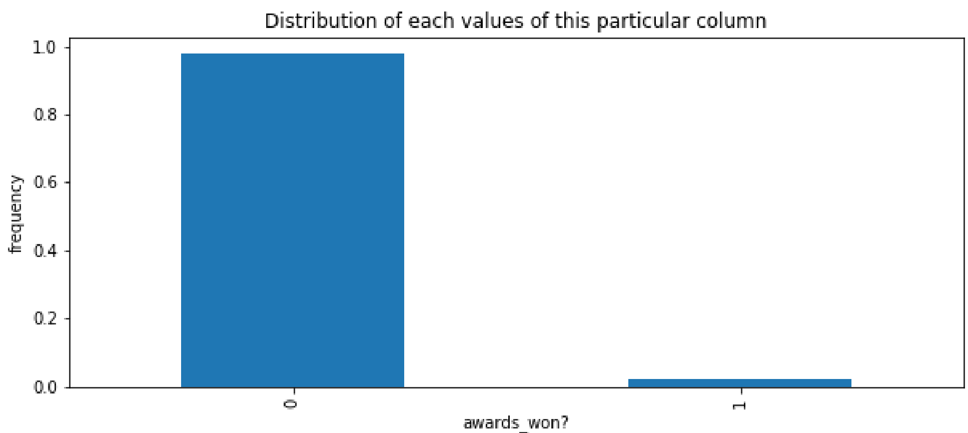

```
************************************************************************
Column :  previous_year_rating
```



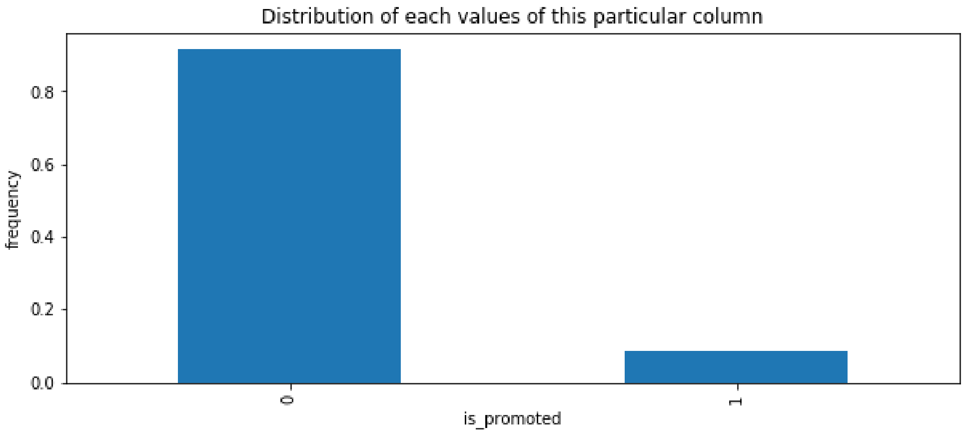

```
************************************************************************
Column :  KPIs_met >80%
```



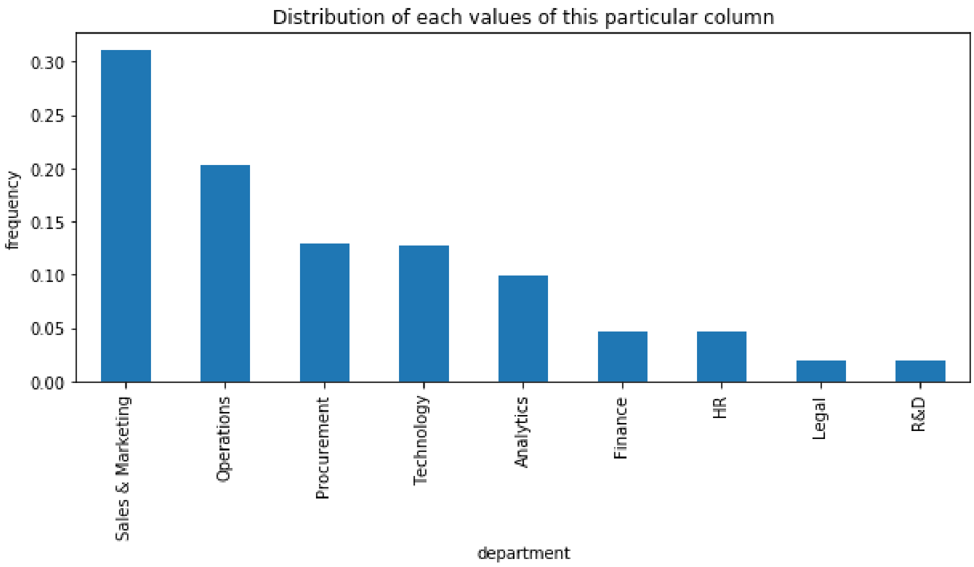

```
************************************************************************
Column :  awards_won?
```



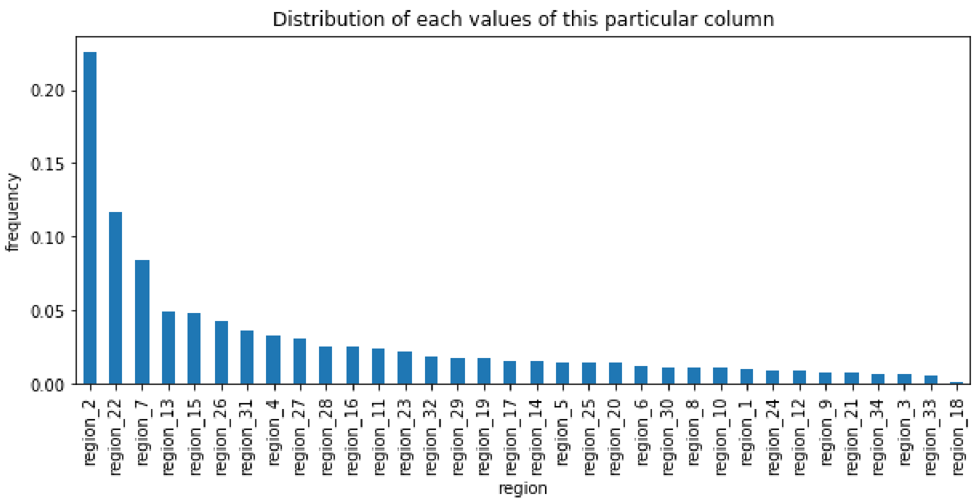
```
***********************************************************************
Column :  is_promoted
```




```
***********************************************************************
***********************************************************************
***********************************************************************
test categorical data:

Column :  department
```

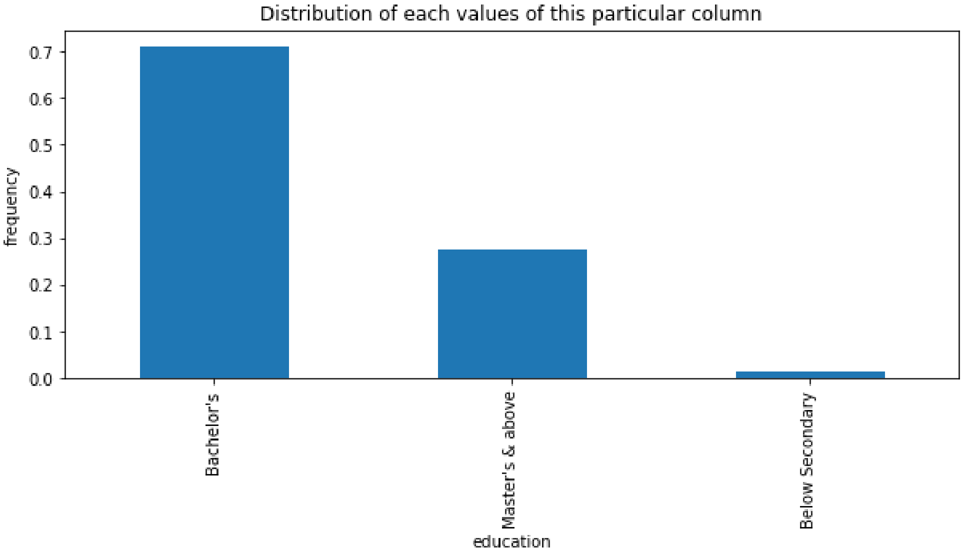



```
***********************************************************************
Column :  region
```

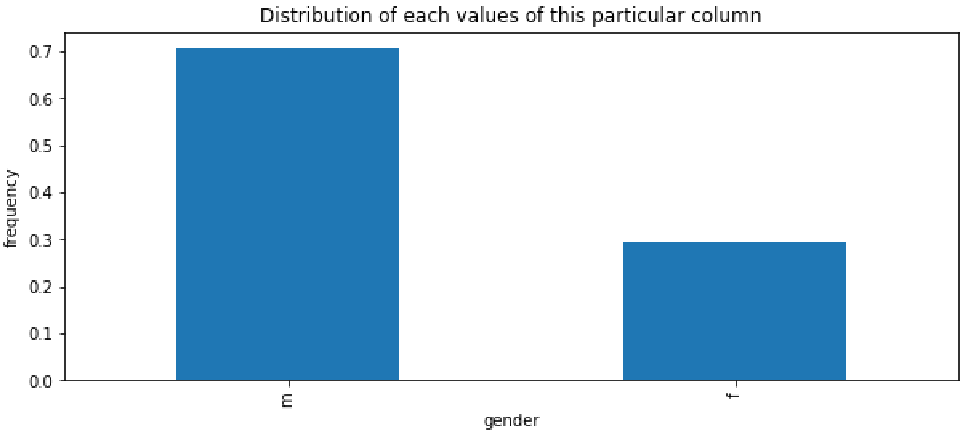



```
***********************************************************************
```

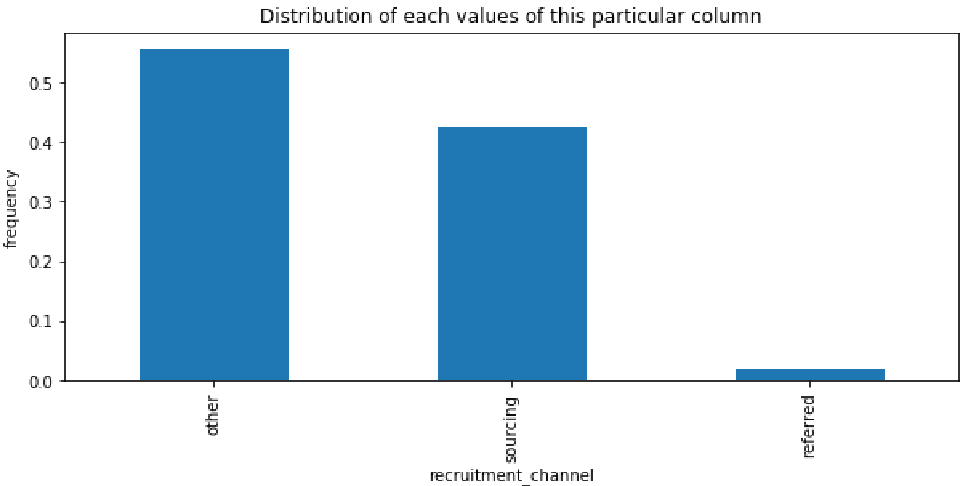
```
*************************************************************************
Column :  education
```



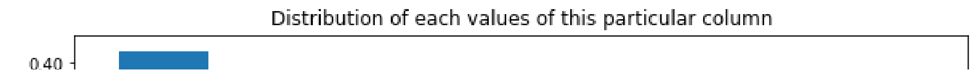

```
*************************************************************************
Column :  gender
```

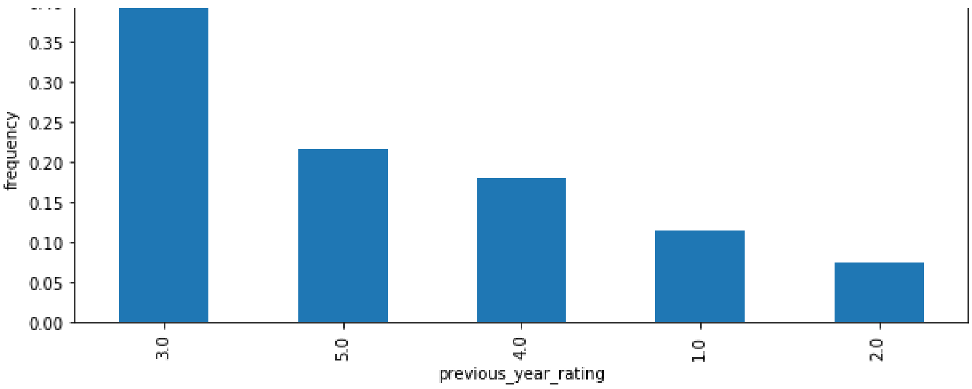



```
*************************************************************************
Column :  recruitment_channel
```

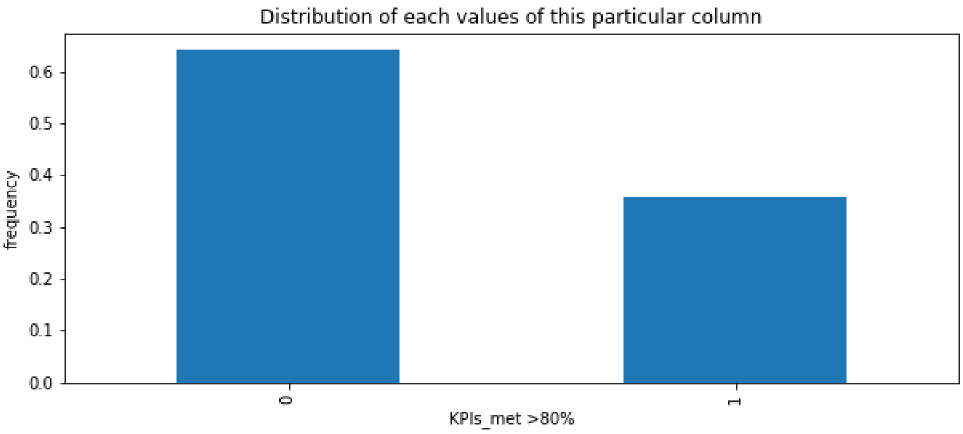



```
*************************************************************************
Column :  previous_year_rating
```

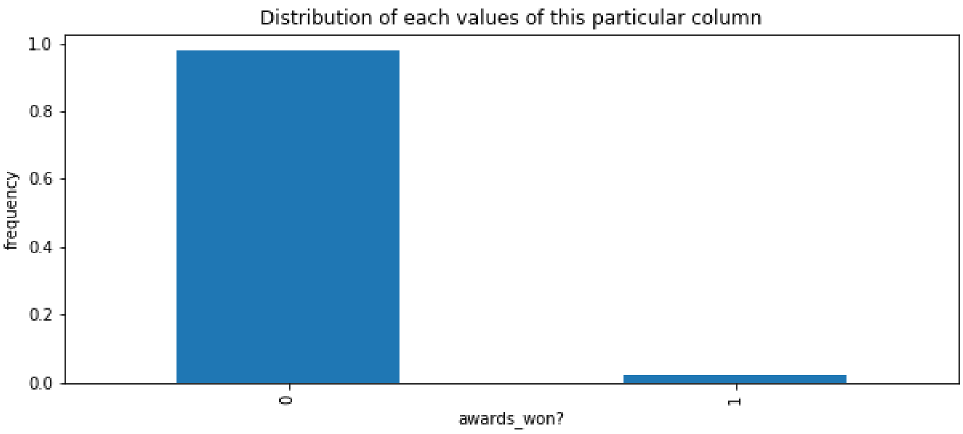





```
************************************************************************
Column :  KPIs_met >80%
```




```
************************************************************************
Column :  awards_won?
```




```
************************************************************************
```

Observations :

1. As we know, for maximum organisation sales and marketing department play huge role(this department is bringing customers).In our data also 35%(highest) people belong to this department only.
2. Again 25% people belong to region_2 only.
3. 70% have Bachelor's degree
4. Again we know that Male number is higher in maximum industry and same our data is also giving.70% people are male.
5. There are only few people(4%) who come in industry through reference.
6. Approx 40% people have previous_year_rating 3
7. Approx 65% people have KPI<80% and which is really correct if we see data of any company.
8. Only 2% people won the award.
9. Approx 85% people are not promoted and only 15% people are promoted.From this data we can observe class is imbalance and type 1 or type 2 error occur.We can't use accuracy here.

type 1 or type 2 error occur.We can't use accuracy here.

5. Bivariate Analysis :

5.1 Continuous-Continuous Variables:

In [20]:

```
# instead of using tra_con,I am using train because .corr() gives correlation between any numerical
# data present in our dataset

train.corr()
```

Out[20]:

	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_met >80%	awards_won?	avg_training_score	i
no_of_trainings	1.000000	-0.081278	-0.061564	-0.057275	-0.045576	-0.007628	0.042517	
age	-0.081278	1.000000	0.026810	0.657111	-0.025592	-0.008169	-0.048380	
previous_year_rating	-0.061564	0.026810	1.000000	0.023504	0.337367	0.026587	0.071926	
length_of_service	-0.057275	0.657111	0.023504	1.000000	-0.077693	-0.039927	-0.038122	
KPIs_met >80%	-0.045576	-0.025592	0.337367	-0.077693	1.000000	0.097000	0.078391	
awards_won?	-0.007628	-0.008169	0.026587	-0.039927	0.097000	1.000000	0.072138	
avg_training_score	0.042517	-0.048380	0.071926	-0.038122	0.078391	0.072138	1.000000	
is_promoted	-0.024896	-0.017166	0.153230	-0.010670	0.221582	0.195871	0.181147	

In [21]:

```
test.corr()
```

Out[21]:

	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_met >80%	awards_won?	avg_training_score
no_of_trainings	1.000000	-0.085509	-0.047281	-0.061095	-0.040020	0.001892	0.048121
age	-0.085509	1.000000	0.037746	0.644515	-0.027661	-0.005457	-0.035823
previous_year_rating	-0.047281	0.037746	1.000000	0.027444	0.334821	0.026392	0.060751
length_of_service	-0.061095	0.644515	0.027444	1.000000	-0.078121	-0.042083	-0.028643
KPIs_met >80%	-0.040020	-0.027661	0.334821	-0.078121	1.000000	0.108288	0.072981
awards_won?	0.001892	-0.005457	0.026392	-0.042083	0.108288	1.000000	0.073857
avg_training_score	0.048121	-0.035823	0.060751	-0.028643	0.072981	0.073857	1.000000

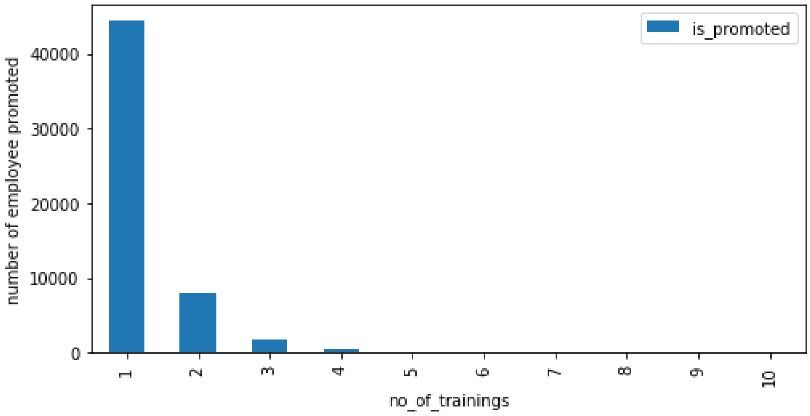
Observations :

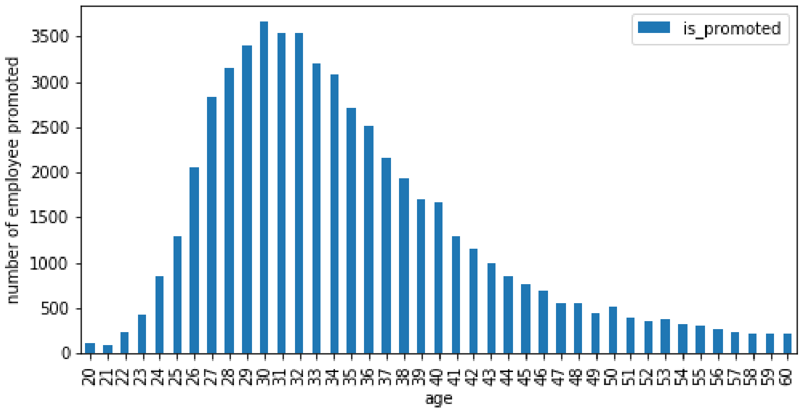
1. The correlation of any numerical variables with target variable is very less and this may create problem in terms of performance

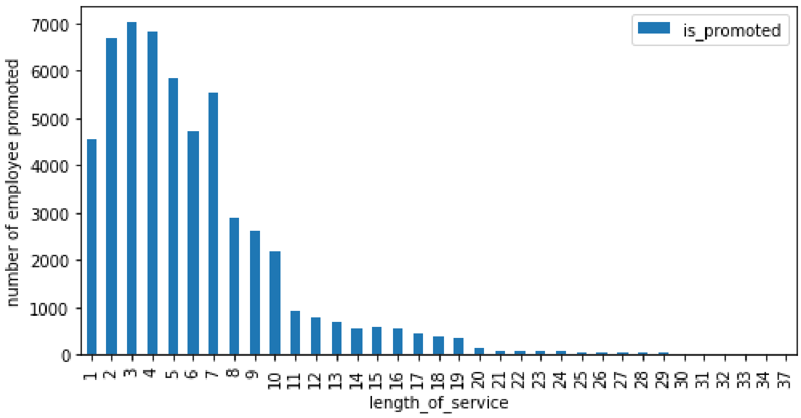
5.2 Continuous-Categorical Variables:

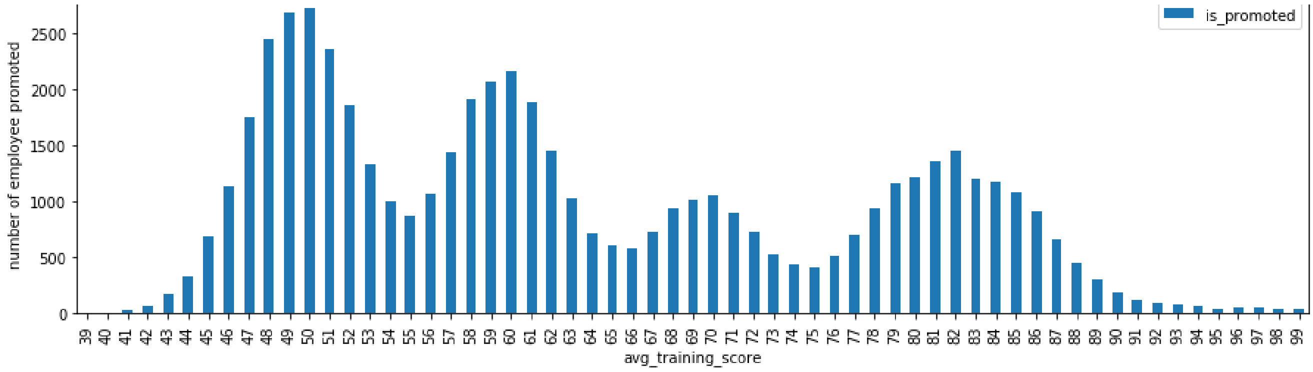
In [22]:

```
# checking % of employee promoted with respect to other continuous variables
#Note that value on y-axis is average because pivot_table by default uses aggfunct=average

pd.pivot_table(data=train,index='no_of_trainings',values='is_promoted',aggfunc=len).plot(kind='bar'
,figsize=(8,4))
plt.ylabel('number of employee promoted')
plt.show()

pd.pivot_table(data=train,index='age',values='is_promoted',aggfunc=len).plot(kind='bar',figsize=(8,
4))
plt.ylabel('number of employee promoted')
plt.show()

pd.pivot_table(data=train,index='length_of_service',values='is_promoted',aggfunc=len).plot(kind='ba
r',figsize=(8,4))
plt.ylabel('number of employee promoted')
plt.show()

pd.pivot_table(data=train,index='avg_training_score',values='is_promoted',aggfunc=len).plot(kind='b
ar',figsize=(15,4))
plt.ylabel('number of employee promoted')
plt.show()
```









Observations :

1. Number of training equal to 1 has higher chance of promotion
2. The person with age 27-35 has higher chance of promotion
3. The person with length of service 3 has higher chance of promotion
4. The curve is quad model
5. After observing 'age' value carefully thinking to group them(next line of code is used for binning)

In [23]:

```
# for train data

bins = [19,20, 27, 35, 46, 60]
names = ['<=20','20-27', '28-35', '36-46', '47-60']

train['Age'] = pd.cut(train['age'], bins, labels=names)

train.head()
```

Out[23]:

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

In [24]:

```
test['Age'] = pd.cut(test['age'], bins, labels=names)

train.head()
```

Out[24]:

	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_ >(
0	Sales & Marketing	region_7	Master's & above	f	sourcing	1	35	5.0	8	
1	Operations	region_22	Bachelor's	m	other	1	30	5.0	4	
2	Sales & Marketing	region_19	Bachelor's	m	sourcing	1	34	3.0	7	
3	Sales & Marketing	region_23	Bachelor's	m	other	2	39	1.0	10	

	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_>8
4	Technology	region_26	Bachelor's	m	other	1	45	3.0	2	

Observation :

1. Must play around boundary value of bins and labels to get the idea of boundary case.
2. Must first check lowest and highest value for that particular column(in both datasets) before creating bins.
3. Age column now become categorical column.
4. How to convert continuous into categorical : https://stackoverflow.com/questions/15891038/change-data-type-of-columns-in-pandas

In [25]:

```
# now dropping age column
# now age becomes categorical variables

train=train.drop('age',1)
test=test.drop('age',1)

#renaming 'Age' to 'age'

train.rename({'Age':'age'},axis=1,inplace=True)
test.rename({'Age':'age'},axis=1,inplace=True)
```

5.3 Categorical-Categorical Variables:

In [26]:

```
# next two statement (line) related to this code only but due to I want to see what result each st
eps
# are giving,I had breaked into more line

depart_count = train.groupby(["gender", "is_promoted"])["is_promoted"].count()
depart_count
```

Out[26]:

```
gender  is_promoted
f       0              14845
        1               1467
m       0              35295
        1               3201
Name: is_promoted, dtype: int64
```

In [27]:

```
depart_count.unstack()#.plot(kind='bar', stacked=True)
```

Out[27]:

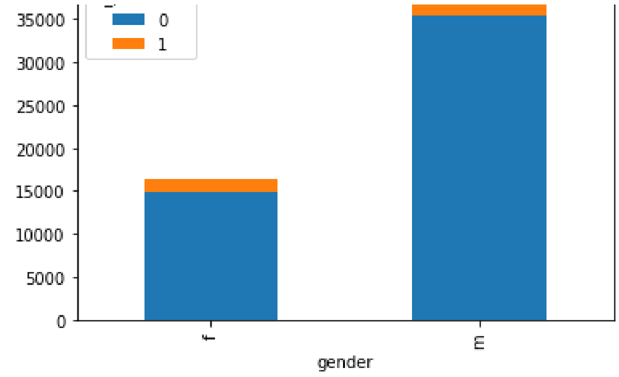
is_promoted	0	1
gender		
f	14845	1467
m	35295	3201

In [28]:

```
depart_count.unstack().plot(kind='bar', stacked=True)
```

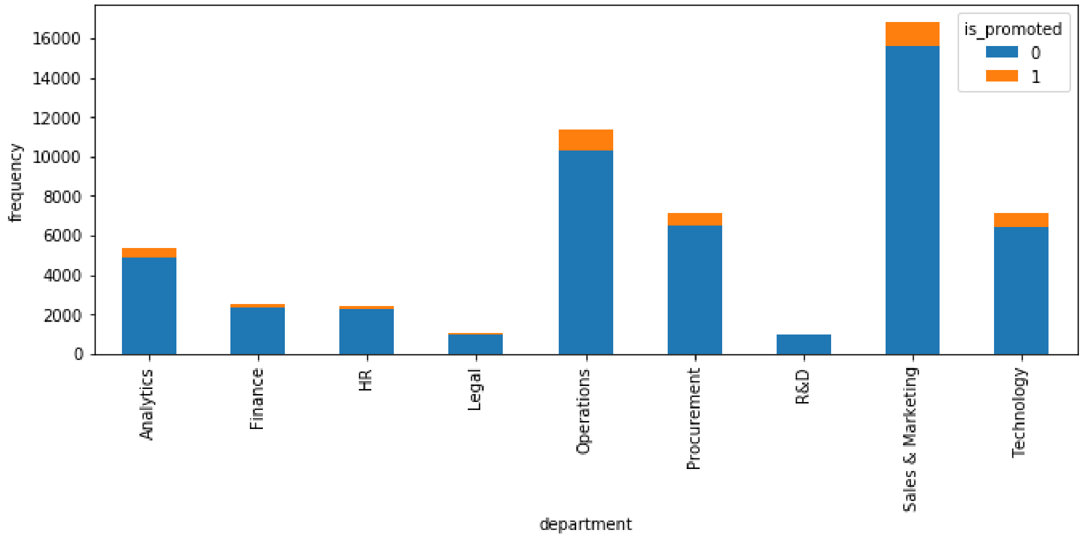
Out[28]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21c5e8befd0>
```

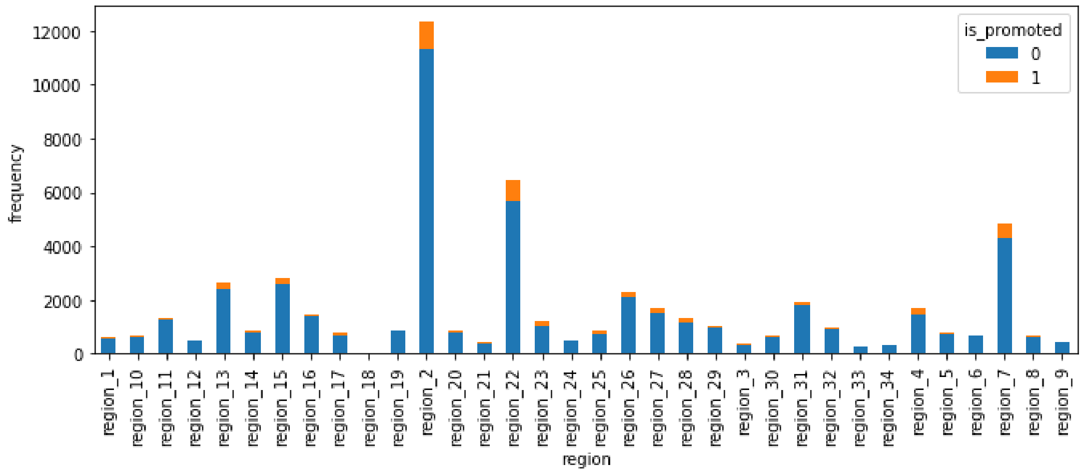


In [29]:

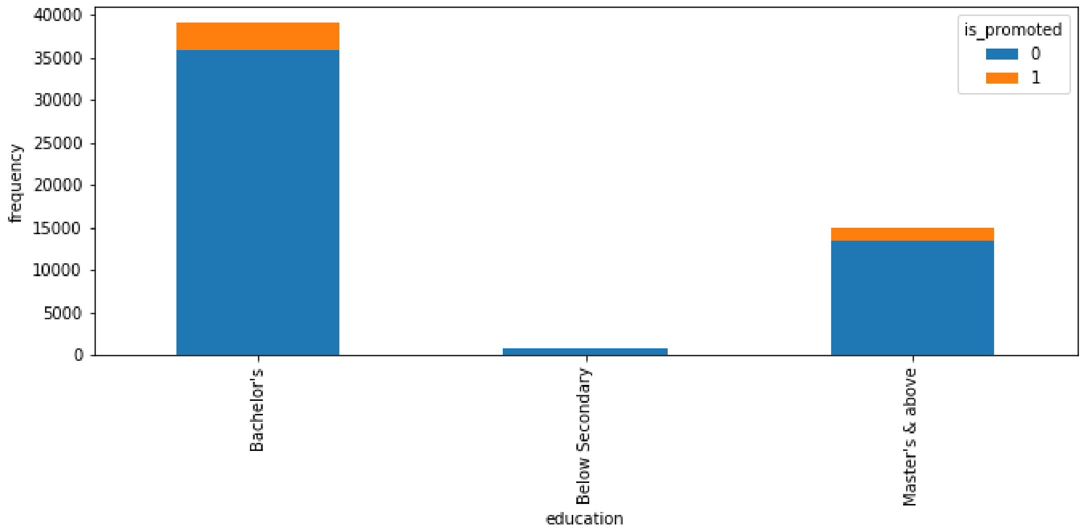
```
for i in tes_cat.columns.tolist():
    #plt.figure(figsize=(11,4))
    train.groupby([i,'is_promoted'])['is_promoted'].count().unstack().plot(kind='bar',stacked=True
,figsize=(11,4))
    plt.ylabel('frequency')
    plt.show()
    print('*'*95)

train.groupby(['age','is_promoted'])['is_promoted'].count().unstack().plot(kind='bar',stacked=True
,figsize=(11,4))
plt.ylabel('frequency')
plt.show()
```



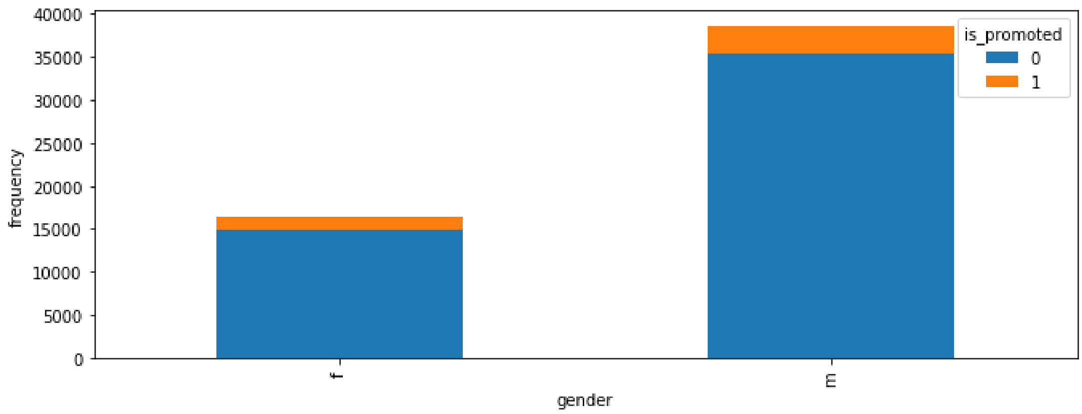

```
***********************************************************************************************
```



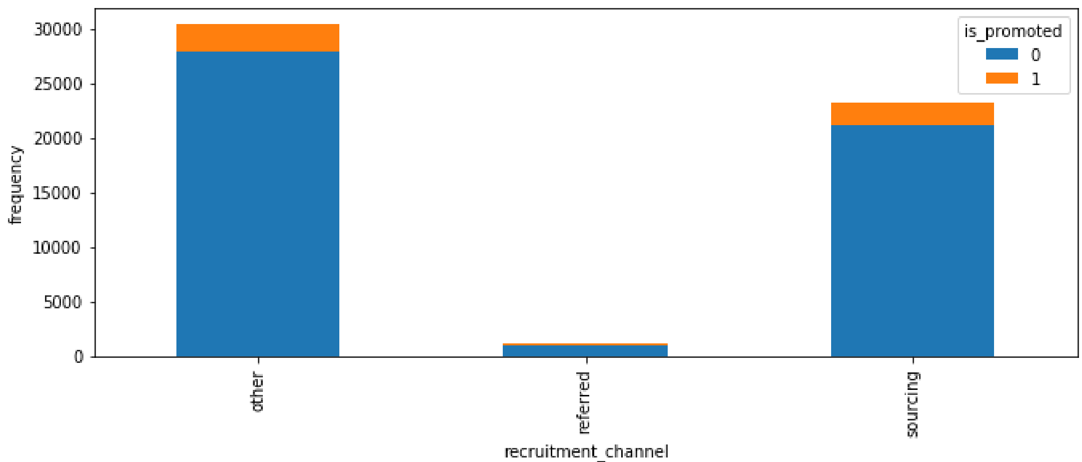

```
***********************************************************************************************
```


40000
35000
30000
25000
20000
15000
10000
5000
0
frequency
is_promoted
0
1
Bachelor's
Below Secondary
Master's & above
education

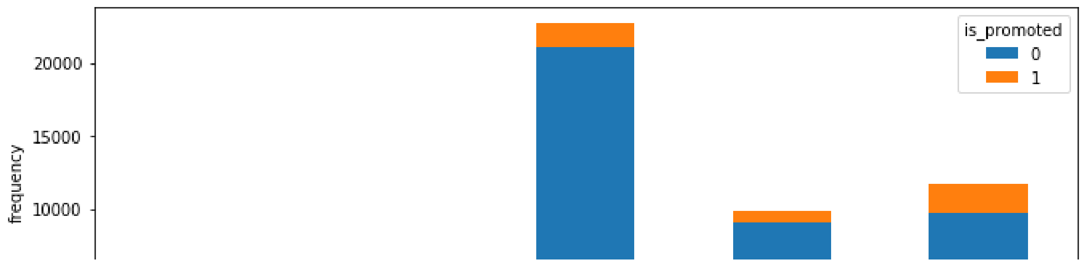
****

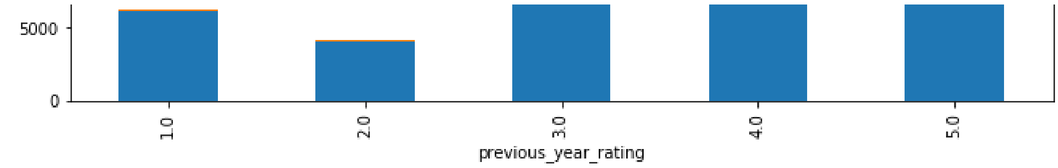

40000
35000
30000
25000
20000
15000
10000
5000
0
frequency
is_promoted
0
1
f
m
gender

****

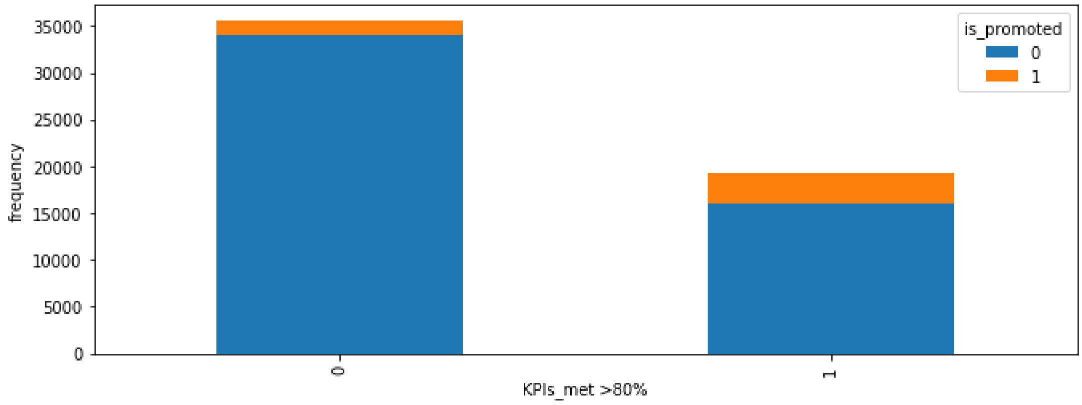

30000
25000
20000
15000
10000
5000
0
frequency
is_promoted
0
1
other
referred
sourcing
recruitment_channel

****

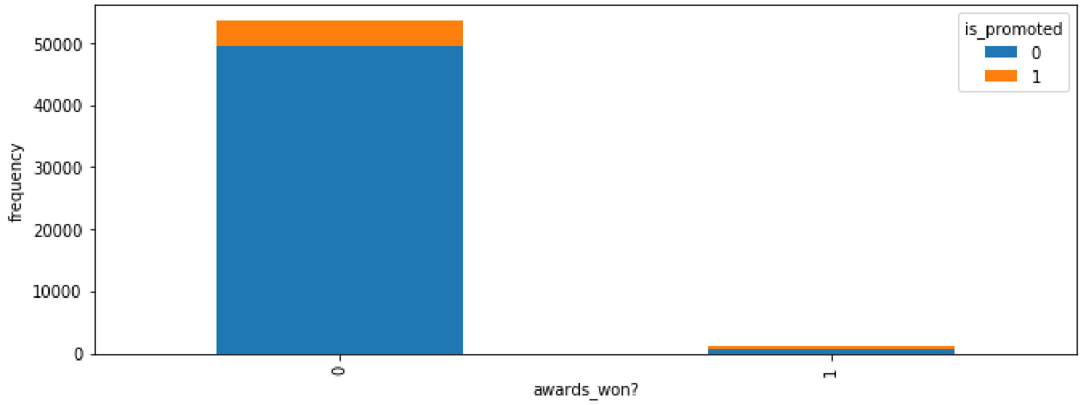

20000
15000
10000
frequency
is_promoted
0
1



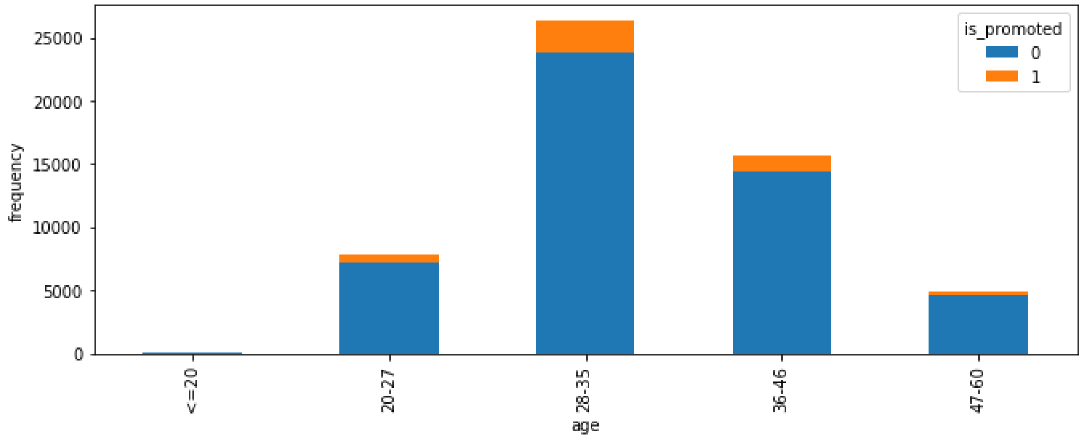
****



****



****



Reference :

1. https://robertmitchellv.com/blog-bar-chart-annotations-pandas-mpl.html

6. Outliers Treatment:

1. No need of this step as all values lies in certain range

Important links :

1. https://www.analyticsvidhya.com/blog/2016/01/12-pandas-techniques-python-data-manipulation/
2. https://data-flair.training/blogs/pandas-function-applications/

7. Feature Engineering :

In [30]:

```
# first doing this thing for train test(in multi step)
# then combine all steps into single for test data

# separating nominal(dummies) and ordinal(label encoder or manually) categorical
# Because both requires different method

ord_var=['education','age']
nom_var=['department','region','gender','recruitment_channel']
```

In [31]:

```
# ordinal part
# this step also change datatype of these variables from object to int

train['education']=train['education'].map({'Below Secondary':0,"Bachelor's":1,"Master's & above":2}
)
train['age']=train['age'].map({'<=20':0,'20-27':1,'28-35':2,'36-46':3,'47-60':4})
```

In [32]:

```
# dummies part

def categorical_to_dummies(train):
    list_to_drop = []
    for col in train.columns:
        if train[col].dtype == 'object':
            print("Converting....", col)
            list_to_drop.append(col)
            train = pd.concat([train, pd.get_dummies(train[col], prefix=col, prefix_sep='_',
drop_first=True)], axis=1)
    return train, list_to_drop
```

In [33]:

```
train.head()
```

Out[33]:

	department	region	education	gender	recruitment_channel	no_of_trainings	previous_year_rating	length_of_service	KPIs_met >80%
0	Sales & Marketing	region_7	2	f	sourcing	1	5.0	8	1
1	Operations	region_22	1	m	other	1	5.0	4	0
2	Sales & Marketing	region_19	1	m	sourcing	1	3.0	7	0
3	Sales & Marketing	region_23	1	m	other	2	1.0	10	0
4	Technology	region_26	1	m	other	1	3.0	2	0

In [34]:

```
train, list_to_drop =categorical_to_dummies(train)
train.head()
```

```
Converting.... department
Converting.... region
Converting.... gender
Converting.... recruitment_channel
```

Out[34]:

	department	region	education	gender	recruitment_channel	no_of_trainings	previous_year_rating	length_of_service	KPIs_met >80%
0	Sales & Marketing	region_7	2	f	sourcing	1	5.0	8	1
1	Operations	region_22	1	m	other	1	5.0	4	0
2	Sales & Marketing	region_19	1	m	sourcing	1	3.0	7	0
3	Sales & Marketing	region_23	1	m	other	2	1.0	10	0
4	Technology	region_26	1	m	other	1	3.0	2	0

5 rows × 57 columns

In [35]:

```
print("Total shape of Data :", train.shape)
print("Columns which need to be dropped :", list_to_drop)
train = train.drop(list_to_drop, axis = 1)
print("Total shape of Data :", train.shape)
```

```
Total shape of Data : (54808, 57)
Columns which need to be dropped : ['department', 'region', 'gender', 'recruitment_channel']
Total shape of Data : (54808, 53)
```

In [36]:

```
train.head()
```

Out[36]:

	education	no_of_trainings	previous_year_rating	length_of_service	KPIs_met >80%	awards_won?	avg_training_score	is_promoted	age
0	2	1	5.0	8	1	0	49	0	2
1	1	1	5.0	4	0	0	60	0	2
2	1	1	3.0	7	0	0	50	0	2
3	1	2	1.0	10	0	0	50	0	3
4	1	1	3.0	2	0	0	73	0	3

5 rows × 53 columns

In [37]:

```
# checking now datatype of each column

train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54808 entries, 0 to 54807
Data columns (total 53 columns):
education                    54808 non-null int64
no_of_trainings              54808 non-null int64
previous_year_rating         54808 non-null float64
length_of_service            54808 non-null int64
KPIs_met >80%                54808 non-null int64
awards_won?                  54808 non-null int64
avg_training_score           54808 non-null int64
is_promoted                  54808 non-null int64
age                          54808 non-null int64
```

```
department_Finance               54808 non-null uint8
department_HR                    54808 non-null uint8
department_Legal                 54808 non-null uint8
department_Operations            54808 non-null uint8
department_Procurement           54808 non-null uint8
department_R&D                   54808 non-null uint8
department_Sales & Marketing     54808 non-null uint8
department_Technology            54808 non-null uint8
region_region_10                 54808 non-null uint8
region_region_11                 54808 non-null uint8
region_region_12                 54808 non-null uint8
region_region_13                 54808 non-null uint8
region_region_14                 54808 non-null uint8
region_region_15                 54808 non-null uint8
region_region_16                 54808 non-null uint8
region_region_17                 54808 non-null uint8
region_region_18                 54808 non-null uint8
region_region_19                 54808 non-null uint8
region_region_2                  54808 non-null uint8
region_region_20                 54808 non-null uint8
region_region_21                 54808 non-null uint8
region_region_22                 54808 non-null uint8
region_region_23                 54808 non-null uint8
region_region_24                 54808 non-null uint8
region_region_25                 54808 non-null uint8
region_region_26                 54808 non-null uint8
region_region_27                 54808 non-null uint8
region_region_28                 54808 non-null uint8
region_region_29                 54808 non-null uint8
region_region_3                  54808 non-null uint8
region_region_30                 54808 non-null uint8
region_region_31                 54808 non-null uint8
region_region_32                 54808 non-null uint8
region_region_33                 54808 non-null uint8
region_region_34                 54808 non-null uint8
region_region_4                  54808 non-null uint8
region_region_5                  54808 non-null uint8
region_region_6                  54808 non-null uint8
region_region_7                  54808 non-null uint8
region_region_8                  54808 non-null uint8
region_region_9                  54808 non-null uint8
gender_m                         54808 non-null uint8
recruitment_channel_referred     54808 non-null uint8
recruitment_channel_sourcing     54808 non-null uint8
dtypes: float64(1), int64(8), uint8(44)
memory usage: 6.1 MB
```

In [38]:

```
test['education']=test['education'].map({'Below Secondary':0,"Bachelor's":1,"Master's & above":2})
test['age']=test['age'].map({'<=20':0,'20-27':1,'28-35':2,'36-46':3,'47-60':4})

# dummies part

def categorical_to_dummies(test):
    list_to_drop = []
    for col in test.columns:
        if test[col].dtype == 'object':
            print("Converting....", col)
            list_to_drop.append(col)
            test = pd.concat([test, pd.get_dummies(test[col], prefix=col, prefix_sep='_',
drop_first=True)], axis=1)
    return test, list_to_drop
```

In [39]:

```
test, list_to_drop =categorical_to_dummies(test)
print("Total shape of Data :", test.shape)
print("Columns which need to be dropped :", list_to_drop)
test = test.drop(list_to_drop, axis = 1)
print("Total shape of Data :", test.shape)
```

```
Converting.... department
Converting.... region
```

```
Converting.... gender
Converting.... recruitment_channel
Total shape of Data : (23490, 56)
Columns which need to be dropped : ['department', 'region', 'gender', 'recruitment_channel']
Total shape of Data : (23490, 52)
```

In [40]:

```
test.head()
```

Out[40]:

	education	no_of_trainings	previous_year_rating	length_of_service	KPIs_met >80%	awards_won?	avg_training_score	age	department_F
0	2	1	3.0	1	1	0	77	1	
1	1	1	3.0	5	0	0	51	2	
2	1	1	1.0	4	0	0	47	2	
3	1	3	2.0	9	0	0	65	2	
4	1	1	4.0	7	0	0	61	2	

5 rows × 52 columns

8. Modeling

In [41]:

```
# this step is added after forming models(all four) first time and I got good accuracy but poor re
call
# ,precision,f1_score due to unbalanced value in target column

# so after I taught to go for upsampling and downsampling

train_1=train.copy()
train_2=train.copy()
train_3=train.copy()
train_4=train.copy()
train_5=train.copy() # extra created,if in future,I will create more model
```

8.1 Brute Force Technique

In [42]:

```
# splitting data into train and test

y=train.pop('is_promoted')
x=train

from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.8,random_state=51)
print("x_train dimension : ",x_train.shape)
print("x_test dimension : ",x_test.shape)
print("y_train dimension : ",y_train.shape)
print("y_test dimension : ",y_test.shape)
```

```
x_train dimension :  (43846, 52)
x_test dimension :  (10962, 52)
y_train dimension :  (43846,)
y_test dimension :  (10962,)
```

```
C:\Users\Purushottam\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:2026:
FutureWarning: From version 0.21, test_size will always complement train_size unless both are
specified.
  FutureWarning)
```

In [43]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

lr = LogisticRegression()
knn = KNeighborsClassifier()
dtc = DecisionTreeClassifier()
rfc = RandomForestClassifier()
```

```
C:\Users\Purushottam\Anaconda3\lib\site-packages\sklearn\ensemble\weight_boosting.py:29:
DeprecationWarning: numpy.core.umath_tests is an internal NumPy module and should not be imported.
It will be removed in a future NumPy release.
  from numpy.core.umath_tests import inner1d
```

In [44]:

```
fit1 = lr.fit(x_train,y_train) #fitting training data to Logistic Regression
fit2 = knn.fit(x_train,y_train) #fitting training data to knn
fit3 = dtc.fit(x_train,y_train) #fitting training data to Decision Tree Classifier
fit4 = rfc.fit(x_train,y_train) #fitting training data to Random Forest Classifier
```

In [45]:

```
print("Accuracy Score of Logistic regression on train set",fit1.score(x_train,y_train)*100)
print("Accuracy Score of knn on train set",fit2.score(x_train,y_train)*100)
print("Accuracy Score of Decision Tree on train set",fit3.score(x_train,y_train)*100)
print("Accuracy Score of Random Forest on train set",fit4.score(x_train,y_train)*100)
```

```
Accuracy Score of Logistic regression on train set 93.0027824659034
Accuracy Score of knn on train set 93.47260867581991
Accuracy Score of Decision Tree on train set 99.86771883410117
Accuracy Score of Random Forest on train set 98.8528029922912
```

In [46]:

```
# this is equivalent to accuracy_score but this I had shown just to show result

print("Accuracy Score of Logistic regression on test set",fit1.score(x_test,y_test)*100)
print("Accuracy Score of knn on test set",fit2.score(x_test,y_test)*100)
print("Accuracy Score of Decision Tree on test set",fit3.score(x_test,y_test)*100)
print("Accuracy Score of Random Forest on test set",fit4.score(x_test,y_test)*100)
```

```
Accuracy Score of Logistic regression on test set 93.06695858419997
Accuracy Score of knn on test set 92.56522532384601
Accuracy Score of Decision Tree on test set 90.12041598248494
Accuracy Score of Random Forest on test set 93.05783616128444
```

In [47]:

```
# predicting target variable value for x_test dataset

y_pred1=fit1.predict(x_test)
y_pred2=fit2.predict(x_test)
y_pred3=fit3.predict(x_test)
y_pred4=fit4.predict(x_test)
```

In [48]:

```
# importing performance metrics

from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
```

In [49]:

```
print("Accuracy Score of Logistic regression on test set",accuracy_score(y_test,y_pred1)*100)
print("Accuracy Score of knn on test set",accuracy_score(y_test,y_pred2)*100)
```

```
print("Accuracy Score of Decision Tree on test set",accuracy_score(y_test,y_pred3)*100)
print("Accuracy Score of Random Forest on test set",accuracy_score(y_test,y_pred4)*100)
```

```
Accuracy Score of Logistic regression on test set 93.06695858419997
Accuracy Score of knn on test set 92.56522532384601
Accuracy Score of Decision Tree on test set 90.12041598248494
Accuracy Score of Random Forest on test set 93.05783616128444
```

In [50]:

```
print("Precision Score of Logistic regression on test set",precision_score(y_test,y_pred1)*100)
print("Precision Score of knn on test set",precision_score(y_test,y_pred2)*100)
print("Precision Score of Decision Tree on test set",precision_score(y_test,y_pred3)*100)
print("Precision Score of Random Forest on test set",precision_score(y_test,y_pred4)*100)
```

```
Precision Score of Logistic regression on test set 84.87394957983193
Precision Score of knn on test set 72.11155378486056
Precision Score of Decision Tree on test set 42.3712342079689
Precision Score of Random Forest on test set 71.2082262210797
```

In [51]:

```
print("Recall Score of Logistic regression on test set",recall_score(y_test,y_pred1)*100)
print("Recall Score of knn on test set",recall_score(y_test,y_pred2)*100)
print("Recall Score of Decision Tree on test set",recall_score(y_test,y_pred3)*100)
print("Recall Score of Random Forest on test set",recall_score(y_test,y_pred4)*100)
```

```
Recall Score of Logistic regression on test set 21.814254859611232
Recall Score of knn on test set 19.546436285097194
Recall Score of Decision Tree on test set 47.084233261339094
Recall Score of Random Forest on test set 29.913606911447083
```

In [52]:

```
print("F1 Score of Logistic regression on test set",f1_score(y_test,y_pred1)*100)
print("F1 Score of knn on test set",f1_score(y_test,y_pred2)*100)
print("F1 Score of Decision Tree on test set",f1_score(y_test,y_pred3)*100)
print("F1 Score of Random Forest on test set",f1_score(y_test,y_pred4)*100)
```

```
F1 Score of Logistic regression on test set 34.70790378006873
F1 Score of knn on test set 30.756159728122345
F1 Score of Decision Tree on test set 44.60358056265984
F1 Score of Random Forest on test set 42.12927756653992
```

In [54]:

```
# this step is only to bring all in one dataframe

a_lr=accuracy_score(y_test,y_pred1)*100
a_knn=accuracy_score(y_test,y_pred2)*100
a_dtc=accuracy_score(y_test,y_pred3)*100
a_rfc=accuracy_score(y_test,y_pred4)*100

p_lr=precision_score(y_test,y_pred1)*100
p_knn=precision_score(y_test,y_pred2)*100
p_dtc=precision_score(y_test,y_pred3)*100
p_rfc=precision_score(y_test,y_pred4)*100

r_lr=recall_score(y_test,y_pred1)*100
r_knn=recall_score(y_test,y_pred2)*100
r_dtc=recall_score(y_test,y_pred3)*100
r_rfc=recall_score(y_test,y_pred4)*100

f_lr=f1_score(y_test,y_pred1)*100
f_knn=f1_score(y_test,y_pred2)*100
f_dtc=f1_score(y_test,y_pred3)*100
f_rfc=f1_score(y_test,y_pred4)*100

performance_df1=pd.DataFrame({'Accuracy':[a_lr,a_knn,a_dtc,a_rfc],
                            'Precision':[p_lr,p_knn,p_dtc,p_rfc],
```

```
                              'Recall':[r_lr,r_knn,r_dtc,r_rfc],
                              'F1 score':[f_lr,f_knn,f_dtc,f_rfc]},
                             index=['Logistic Regression', 'KNN', 'Decision Tree','Random Forest'])
performance_df1
```

Out[54]:

	Accuracy	Precision	Recall	F1 score
Logistic Regression	93.066959	84.873950	21.814255	34.707904
KNN	92.565225	72.111554	19.546436	30.756160
Decision Tree	90.120416	42.371234	47.084233	44.603581
Random Forest	93.057836	71.208226	29.913607	42.129278

In [55]:

```
# it is similar like .predict (it predict label for each observation) while
# .predict_proba (it predict probability for each observation)

y_pred_proba1=fit1.predict_proba(x_test)
y_pred_proba2=fit2.predict_proba(x_test)
y_pred_proba3=fit3.predict_proba(x_test)
y_pred_proba4=fit4.predict_proba(x_test)
```

In [56]:

```
from sklearn.metrics import roc_curve,auc
```

In [57]:

```
# just checking length of predicted value

len(y_pred_proba1[:,1])
```

Out[57]:

```
10962
```

In [58]:

```
print("For Logistic Regression:")
false_positive_rate1,true_positive_rate1,thresholds1=roc_curve(y_test,y_pred_proba1[:,1])
roc_auc1=auc(false_positive_rate1,true_positive_rate1)
print("area under the roc curve :",roc_auc1)

print("For KNN:")
false_positive_rate2,true_positive_rate2,thresholds2=roc_curve(y_test,y_pred_proba2[:,1])
roc_auc2=auc(false_positive_rate2,true_positive_rate2)
print("area under the roc curve :",roc_auc2)

print("For Decision tree:")
false_positive_rate3,true_positive_rate3,thresholds3=roc_curve(y_test,y_pred_proba3[:,1])
roc_auc3=auc(false_positive_rate3,true_positive_rate3)
print("area under the roc curve :",roc_auc3)

print("For Random Forest:")
false_positive_rate4,true_positive_rate4,thresholds4=roc_curve(y_test,y_pred_proba4[:,1])
roc_auc4=auc(false_positive_rate4,true_positive_rate4)
print("area under the roc curve :",roc_auc4)
```

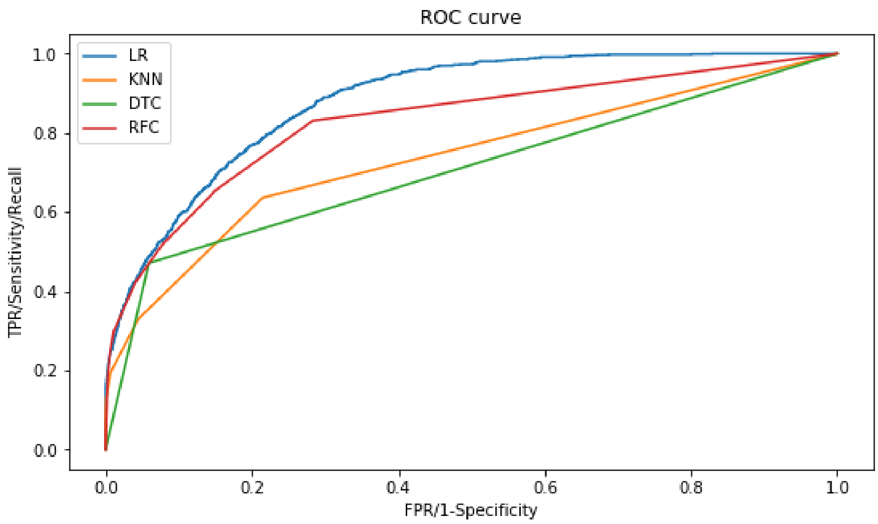
```
For Logistic Regression:
area under the roc curve : 0.8811796431335314
For KNN:
area under the roc curve : 0.7355626117467399
For Decision tree:
area under the roc curve : 0.7059375664454615
For Random Forest:
area under the roc curve : 0.8278185034954079
```

In [59]:

```
plt.figure(figsize=(9,5))
plt.plot(false_positive_rate1,true_positive_rate1,label = "LR")
plt.plot(false_positive_rate2,true_positive_rate2,label = "KNN")
plt.plot(false_positive_rate3,true_positive_rate3,label = "DTC")
plt.plot(false_positive_rate4,true_positive_rate4,label = "RFC")
plt.ylabel("TPR/Sensitivity/Recall")
plt.xlabel("FPR/1-Specificity")
plt.title("ROC curve")
plt.legend()
```

Out[59]:

```
<matplotlib.legend.Legend at 0x21c651879b0>
```



8.2 Down Sampling

In [60]:

```
from sklearn.utils import resample

# Separate majority and minority classes
df_majority = train_1[train_1.is_promoted==0]
df_minority = train_1[train_1.is_promoted==1]

# downsample majority class
df_minority_upsampled = resample(df_majority,
                                 replace=True,     # sample with replacement
                                 n_samples=4668,    # to match minority class
                                 random_state=123) # reproducible results

# Combine minority class with downsampled majority class
train_1= pd.concat([df_minority, df_minority_upsampled])

# Display new class counts
train_1.is_promoted.value_counts()
```

Out[60]:

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

In [61]:

```
import warnings
warnings.filterwarnings("ignore")

# splitting data into train and test
```

```
y=train_1.pop('is_promoted')
x=train_1


x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.8,random_state=51)
print("x_train dimension : ",x_train.shape)
print("x_test dimension : ",x_test.shape)
print("y_train dimension : ",y_train.shape)
print("y_test dimension : ",y_test.shape)
```

```
x_train dimension :  (7468, 52)
x_test dimension :  (1868, 52)
y_train dimension :  (7468,)
y_test dimension :  (1868,)
```

In [62]:

```
# there is no requirement of importing same libraries

fit1 = lr.fit(x_train,y_train) #fitting training data to logistic regression
fit2 = knn.fit(x_train,y_train) #fitting training data to knn
fit3 = dtc.fit(x_train,y_train) #fitting training data to Decision Tree Classifier
fit4 = rfc.fit(x_train,y_train) #fitting training data to Random Forest Classifier
```

In [63]:

```
print("Accuracy Score of Logistic regression on train set",fit1.score(x_train,y_train)*100)
print("Accuracy Score of knn on train set",fit2.score(x_train,y_train)*100)
print("Accuracy Score of Decision Tree on train set",fit3.score(x_train,y_train)*100)
print("Accuracy Score of Random Forest on train set",fit4.score(x_train,y_train)*100)
```

```
Accuracy Score of Logistic regression on train set 79.45902517407606
Accuracy Score of knn on train set 82.67273701124799
Accuracy Score of Decision Tree on train set 99.93304767005891
Accuracy Score of Random Forest on train set 98.87520085698982
```

In [64]:

```
# predicting target variable value for x_test dataset

y_pred1=fit1.predict(x_test)
y_pred2=fit2.predict(x_test)
y_pred3=fit3.predict(x_test)
y_pred4=fit4.predict(x_test)
```

In [65]:

```
from sklearn.metrics import confusion_matrix
```

In [66]:

```
print("For Logistic Regression:")

confusion=confusion_matrix(y_test,y_pred1)
print(confusion)
TN=confusion[0,0]
FP=confusion[0,1]
FN=confusion[1,0]
TP=confusion[1,1]

print("Accuracy :",(TP+TN)/(TP+TN+FP+FN))
print("Precision :",TP/(TP+FP))
print("Recall :",TP/(TP+FN))
precision=TP/(TP+FP)
recall=TP/(TP+FN)
print("F1 Score :",(2*precision*recall)/(precision+recall))

print("*"*71)

print("For KNN:")
```

```
confusion=confusion_matrix(y_test,y_pred2)
TN=confusion[0,0]
FP=confusion[0,1]
FN=confusion[1,0]
TP=confusion[1,1]

print("Accuracy :",(TP+TN)/(TP+TN+FP+FN))
print("Precision :",TP/(TP+FP))
print("Recall :",TP/(TP+FN))
precision=TP/(TP+FP)
recall=TP/(TP+FN)
print("F1 Score :",(2*precision*recall)/(precision+recall))

print("*"*71)

print("For Decision Tree:")

confusion=confusion_matrix(y_test,y_pred3)
TN=confusion[0,0]
FP=confusion[0,1]
FN=confusion[1,0]
TP=confusion[1,1]

print("Accuracy :",(TP+TN)/(TP+TN+FP+FN))
print("Precision :",TP/(TP+FP))
print("Recall :",TP/(TP+FN))
precision=TP/(TP+FP)
recall=TP/(TP+FN)
print("F1 Score :",(2*precision*recall)/(precision+recall))

print("*"*71)

print("For Random Forest:")

confusion=confusion_matrix(y_test,y_pred4)
TN=confusion[0,0]
FP=confusion[0,1]
FN=confusion[1,0]
TP=confusion[1,1]

print("Accuracy :",(TP+TN)/(TP+TN+FP+FN))
print("Precision :",TP/(TP+FP))
print("Recall :",TP/(TP+FN))
precision=TP/(TP+FP)
recall=TP/(TP+FN)
print("F1 Score :",(2*precision*recall)/(precision+recall))
```

```
For Logistic Regression:
[[690 243]
 [157 778]]
Accuracy : 0.7858672376873662
Precision : 0.761998041136141
Recall : 0.8320855614973262
F1 Score : 0.7955010224948875
***********************************************************************
For KNN:
Accuracy : 0.7168094218415417
Precision : 0.71875
Recall : 0.7133689839572193
F1 Score : 0.7160493827160493
***********************************************************************
For Decision Tree:
Accuracy : 0.7692719486081371
Precision : 0.7751091703056768
Recall : 0.7593582887700535
F1 Score : 0.7671528903295516
***********************************************************************
For Random Forest:
Accuracy : 0.771948608137045
Precision : 0.7704569606801275
Recall : 0.7754010695187166
F1 Score : 0.7729211087420041
```

In [67]:

```python
# this step is only to bring all in one dataframe

a_lr=accuracy_score(y_test,y_pred1)*100
a_knn=accuracy_score(y_test,y_pred2)*100
a_dtc=accuracy_score(y_test,y_pred3)*100
a_rfc=accuracy_score(y_test,y_pred4)*100

p_lr=precision_score(y_test,y_pred1)*100
p_knn=precision_score(y_test,y_pred2)*100
p_dtc=precision_score(y_test,y_pred3)*100
p_rfc=precision_score(y_test,y_pred4)*100

r_lr=recall_score(y_test,y_pred1)*100
r_knn=recall_score(y_test,y_pred2)*100
r_dtc=recall_score(y_test,y_pred3)*100
r_rfc=recall_score(y_test,y_pred4)*100

f_lr=f1_score(y_test,y_pred1)*100
f_knn=f1_score(y_test,y_pred2)*100
f_dtc=f1_score(y_test,y_pred3)*100
f_rfc=f1_score(y_test,y_pred4)*100

performance_df2=pd.DataFrame({'Accuracy':[a_lr,a_knn,a_dtc,a_rfc],
                            'Precision':[p_lr,p_knn,p_dtc,p_rfc],
                            'Recall':[r_lr,r_knn,r_dtc,r_rfc],
                            'F1 score':[f_lr,f_knn,f_dtc,f_rfc]},
                           index=['Logistic Regression', 'KNN', 'Decision Tree','Random Forest'])
performance_df2
```

Out[67]:

	Accuracy	Precision	Recall	F1 score
Logistic Regression	78.586724	76.199804	83.208556	79.550102
KNN	71.680942	71.875000	71.336898	71.604938
Decision Tree	76.927195	77.510917	75.935829	76.715289
Random Forest	77.194861	77.045696	77.540107	77.292111

In [68]:

```python
# it is similar like .predict (it predict label for each observation) while
# .predict_proba (it predict probability for each observation)

y_pred_proba1=fit1.predict_proba(x_test)
y_pred_proba2=fit2.predict_proba(x_test)
y_pred_proba3=fit3.predict_proba(x_test)
y_pred_proba4=fit4.predict_proba(x_test)
```

In [69]:

```python
print("For Logistic Regression:")
false_positive_rate1,true_positive_rate1,thresholds1=roc_curve(y_test,y_pred_proba1[:,1])
roc_auc1=auc(false_positive_rate1,true_positive_rate1)
print("area under the roc curve :",roc_auc1)

print("For KNN:")
false_positive_rate2,true_positive_rate2,thresholds2=roc_curve(y_test,y_pred_proba2[:,1])
roc_auc2=auc(false_positive_rate2,true_positive_rate2)
print("area under the roc curve :",roc_auc2)

print("For Decision tree:")
false_positive_rate3,true_positive_rate3,thresholds3=roc_curve(y_test,y_pred_proba3[:,1])
roc_auc3=auc(false_positive_rate3,true_positive_rate3)
print("area under the roc curve :",roc_auc3)

print("For Random Forest:")
false_positive_rate4,true_positive_rate4,thresholds4=roc_curve(y_test,y_pred_proba4[:,1])
roc_auc4=auc(false_positive_rate4,true_positive_rate4)
print("area under the roc curve :",roc_auc4)
```

```
For Logistic Regression:
```

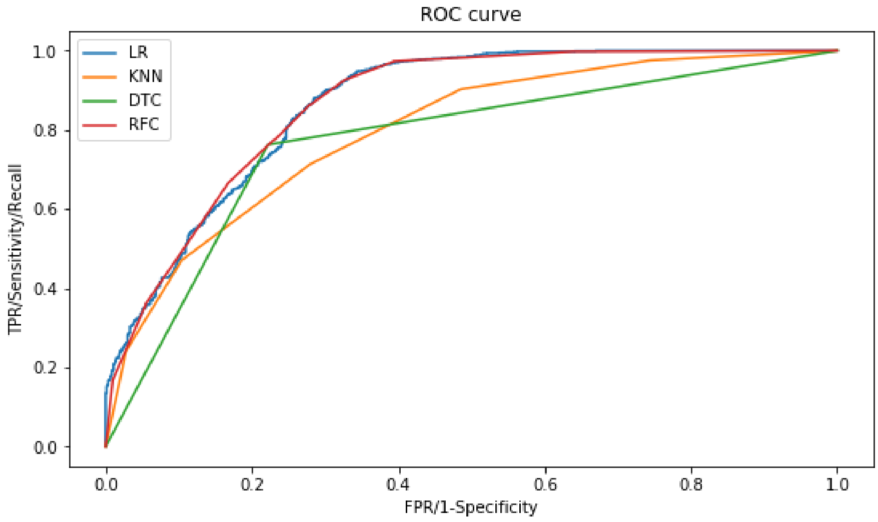
```
area under the roc curve : 0.8639229442142248
For KNN:
area under the roc curve : 0.7963237443471981
For Decision tree:
area under the roc curve : 0.7704036774019752
For Random Forest:
area under the roc curve : 0.8641355870029976
```

In [70]:

```python
plt.figure(figsize=(9,5))
plt.plot(false_positive_rate1,true_positive_rate1,label = "LR")
plt.plot(false_positive_rate2,true_positive_rate2,label = "KNN")
plt.plot(false_positive_rate3,true_positive_rate3,label = "DTC")
plt.plot(false_positive_rate4,true_positive_rate4,label = "RFC")
plt.ylabel("TPR/Sensitivity/Recall")
plt.xlabel("FPR/1-Specificity")
plt.title("ROC curve")
plt.legend()
```

Out[70]:

```
<matplotlib.legend.Legend at 0x21c651fd908>
```



8.3 Up Sampling

In [71]:

```python
# Separate majority and minority classes
df_majority = train_2[train_2.is_promoted==0]
df_minority = train_2[train_2.is_promoted==1]

# Upsample minority class
df_minority_upsampled = resample(df_minority,
                                 replace=True,     # sample with replacement
                                 n_samples=50140,    # to match majority class
                                 random_state=123) # reproducible results

# Combine majority class with upsampled minority class
train_2= pd.concat([df_majority, df_minority_upsampled])

# Display new class counts
train_2.is_promoted.value_counts()
```

Out[71]:

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

In [72]:

In [72]:

```
# splitting data into train and test

y=train_2.pop('is_promoted')
x=train_2


x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.8,random_state=51)
print("x_train dimension : ",x_train.shape)
print("x_test dimension : ",x_test.shape)
print("y_train dimension : ",y_train.shape)
print("y_test dimension : ",y_test.shape)
```

```
x_train dimension :  (80224, 52)
x_test dimension :  (20056, 52)
y_train dimension :  (80224,)
y_test dimension :  (20056,)
```

In [73]:

```
# there is no requirement of importing same libraries

fit1 = lr.fit(x_train,y_train) #fitting training data to logistic regression
fit2 = knn.fit(x_train,y_train) #fitting training data to knn
fit3 = dtc.fit(x_train,y_train) #fitting training data to Decision Tree Classifier
fit4 = rfc.fit(x_train,y_train) #fitting training data to Random Forest Classifier
```

In [74]:

```
print("Accuracy Score of Logistic regression on train set",fit1.score(x_train,y_train)*100)
print("Accuracy Score of knn on train set",fit2.score(x_train,y_train)*100)
print("Accuracy Score of Decision Tree on train set",fit3.score(x_train,y_train)*100)
print("Accuracy Score of Random Forest on train set",fit4.score(x_train,y_train)*100)
```

```
Accuracy Score of Logistic regression on train set 79.32289589150379
Accuracy Score of knn on train set 93.70387913841245
Accuracy Score of Decision Tree on train set 99.88656761069007
Accuracy Score of Random Forest on train set 99.85540486637416
```

In [75]:

```
# predicting target variable value for x_test dataset

y_pred1=fit1.predict(x_test)
y_pred2=fit2.predict(x_test)
y_pred3=fit3.predict(x_test)
y_pred4=fit4.predict(x_test)
```

In [76]:

```
from sklearn.metrics import classification_report

print("For Logistic Regression:")
print(classification_report(y_test,y_pred1))

print("*"*71)

print("For KNN:")
print(classification_report(y_test,y_pred2))

print("*"*71)

print("For Decision Tree:")
print(classification_report(y_test,y_pred3))

print("*"*71)

print("For Random Forest:")
print(classification_report(y_test,y_pred4))
```

```
For Logistic Regression:
```

```
             precision    recall  f1-score   support

          0       0.81      0.77      0.79     10058
          1       0.78      0.82      0.80      9998

avg / total       0.80      0.79      0.79     20056

***********************************************************************
For KNN:
             precision    recall  f1-score   support

          0       1.00      0.82      0.90     10058
          1       0.85      1.00      0.92      9998

avg / total       0.92      0.91      0.91     20056

***********************************************************************
For Decision Tree:
             precision    recall  f1-score   support

          0       1.00      0.93      0.96     10058
          1       0.93      1.00      0.97      9998

avg / total       0.97      0.96      0.96     20056

***********************************************************************
For Random Forest:
             precision    recall  f1-score   support

          0       1.00      0.96      0.98     10058
          1       0.96      1.00      0.98      9998

avg / total       0.98      0.98      0.98     20056
```

In [77]:

```
# this step is only to bring all in one dataframe

a_lr=accuracy_score(y_test,y_pred1)*100
a_knn=accuracy_score(y_test,y_pred2)*100
a_dtc=accuracy_score(y_test,y_pred3)*100
a_rfc=accuracy_score(y_test,y_pred4)*100

p_lr=precision_score(y_test,y_pred1)*100
p_knn=precision_score(y_test,y_pred2)*100
p_dtc=precision_score(y_test,y_pred3)*100
p_rfc=precision_score(y_test,y_pred4)*100

r_lr=recall_score(y_test,y_pred1)*100
r_knn=recall_score(y_test,y_pred2)*100
r_dtc=recall_score(y_test,y_pred3)*100
r_rfc=recall_score(y_test,y_pred4)*100

f_lr=f1_score(y_test,y_pred1)*100
f_knn=f1_score(y_test,y_pred2)*100
f_dtc=f1_score(y_test,y_pred3)*100
f_rfc=f1_score(y_test,y_pred4)*100

performance_df3=pd.DataFrame({'Accuracy':[a_lr,a_knn,a_dtc,a_rfc],
                            'Precision':[p_lr,p_knn,p_dtc,p_rfc],
                            'Recall':[r_lr,r_knn,r_dtc,r_rfc],
                            'F1 score':[f_lr,f_knn,f_dtc,f_rfc]},
                           index=['Logistic Regression', 'KNN', 'Decision Tree','Random Forest'])
performance_df3
```

Out[77]:

	Accuracy	Precision	Recall	F1 score
Logistic Regression	79.467491	77.772530	82.346469	79.994170
KNN	90.840646	84.718795	99.589918	91.554411
Decision Tree	96.499801	93.447373	99.989998	96.608040

	Accuracy	Precision	Recall	F1 score
Random Forest	98.160152	96.449590	99.990098	98.187890

In [78]:

```
# it is similar like .predict (it predict label for each observation) while
# .predict_proba (it predict probability for each observation)

y_pred_proba1=fit1.predict_proba(x_test)
y_pred_proba2=fit2.predict_proba(x_test)
y_pred_proba3=fit3.predict_proba(x_test)
y_pred_proba4=fit4.predict_proba(x_test)
```

In [79]:

```
print("For Logistic Regression:")
false_positive_rate1,true_positive_rate1,thresholds1=roc_curve(y_test,y_pred_proba1[:,1])
roc_auc1=auc(false_positive_rate1,true_positive_rate1)
print("area under the roc curve :",roc_auc1)

print("For KNN:")
false_positive_rate2,true_positive_rate2,thresholds2=roc_curve(y_test,y_pred_proba2[:,1])
roc_auc2=auc(false_positive_rate2,true_positive_rate2)
print("area under the roc curve :",roc_auc2)

print("For Decision tree:")
false_positive_rate3,true_positive_rate3,thresholds3=roc_curve(y_test,y_pred_proba3[:,1])
roc_auc3=auc(false_positive_rate3,true_positive_rate3)
print("area under the roc curve :",roc_auc3)

print("For Random Forest:")
false_positive_rate4,true_positive_rate4,thresholds4=roc_curve(y_test,y_pred_proba4[:,1])
roc_auc4=auc(false_positive_rate4,true_positive_rate4)
print("area under the roc curve :",roc_auc4)
```

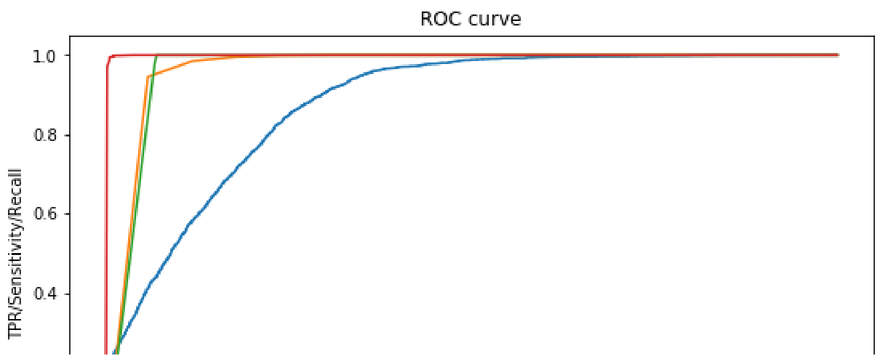
```
For Logistic Regression:
area under the roc curve : 0.877302424095875
For KNN:
area under the roc curve : 0.9667710883596485
For Decision tree:
area under the roc curve : 0.9657875003117545
For Random Forest:
area under the roc curve : 0.9990003518699364
```

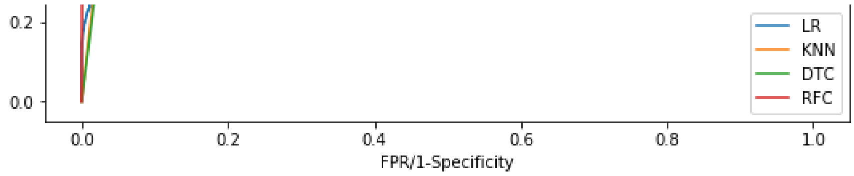
In [80]:

```
plt.figure(figsize=(9,5))
plt.plot(false_positive_rate1,true_positive_rate1,label = "LR")
plt.plot(false_positive_rate2,true_positive_rate2,label = "KNN")
plt.plot(false_positive_rate3,true_positive_rate3,label = "DTC")
plt.plot(false_positive_rate4,true_positive_rate4,label = "RFC")
plt.ylabel("TPR/Sensitivity/Recall")
plt.xlabel("FPR/1-Specificity")
plt.title("ROC curve")
plt.legend()
```

Out[80]:

```
<matplotlib.legend.Legend at 0x21c60bce630>
```





8.4 SMOTE :

In [81]:

```
# splitting data into train and test

y=train_3.pop('is_promoted')
x=train_3


x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.8,random_state=51)
print("x_train dimension : ",x_train.shape)
print("x_test dimension : ",x_test.shape)
print("y_train dimension : ",y_train.shape)
print("y_test dimension : ",y_test.shape)
```

```
x_train dimension :  (43846, 52)
x_test dimension :  (10962, 52)
y_train dimension :  (43846,)
y_test dimension :  (10962,)
```

In [82]:

```
print("Before OverSampling, counts of label '1': {}".format(sum(y_train == 1)))
print("Before OverSampling, counts of label '0': {} \n".format(sum(y_train == 0)))

# import SMOTE module from imblearn library

# pip install imblearn (if you don't have imblearn in your system)

from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state = 123)
x_train, y_train= sm.fit_sample(x_train, y_train)

print('After OverSampling, the shape of train_X: {}'.format(x_train.shape))
print('After OverSampling, the shape of train_y: {} \n'.format(y_train.shape))

print("After OverSampling, counts of label '1': {}".format(sum(y_train == 1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train == 0)))
```

```
Before OverSampling, counts of label '1': 3742
Before OverSampling, counts of label '0': 40104

After OverSampling, the shape of train_X: (80208, 52)
After OverSampling, the shape of train_y: (80208,)

After OverSampling, counts of label '1': 40104
After OverSampling, counts of label '0': 40104
```

In [83]:

```
# there is no requirement of importing same libraries

fit1 = lr.fit(x_train,y_train) #fitting training data to logistic regression
fit2 = knn.fit(x_train,y_train) #fitting training data to knn
fit3 = dtc.fit(x_train,y_train) #fitting training data to Decision Tree Classifier
fit4 = rfc.fit(x_train,y_train) #fitting training data to Random Forest Classifier
```

In [84]:

```
# predicting target variable value for x_test dataset

y_pred1=fit1.predict(x_test)
```

```
y_pred2=fit2.predict(x_test)
y_pred3=fit3.predict(x_test)
y_pred4=fit4.predict(x_test)
```

In [85]:

```
a_lr=accuracy_score(y_test,y_pred1)*100
a_knn=accuracy_score(y_test,y_pred2)*100
a_dtc=accuracy_score(y_test,y_pred3)*100
a_rfc=accuracy_score(y_test,y_pred4)*100

p_lr=precision_score(y_test,y_pred1)*100
p_knn=precision_score(y_test,y_pred2)*100
p_dtc=precision_score(y_test,y_pred3)*100
p_rfc=precision_score(y_test,y_pred4)*100

r_lr=recall_score(y_test,y_pred1)*100
r_knn=recall_score(y_test,y_pred2)*100
r_dtc=recall_score(y_test,y_pred3)*100
r_rfc=recall_score(y_test,y_pred4)*100

f_lr=f1_score(y_test,y_pred1)*100
f_knn=f1_score(y_test,y_pred2)*100
f_dtc=f1_score(y_test,y_pred3)*100
f_rfc=f1_score(y_test,y_pred4)*100

performance_df4=pd.DataFrame({'Accuracy':[a_lr,a_knn,a_dtc,a_rfc],
                             'Precision':[p_lr,p_knn,p_dtc,p_rfc],
                             'Recall':[r_lr,r_knn,r_dtc,r_rfc],
                             'F1 score':[f_lr,f_knn,f_dtc,f_rfc]},
                            index=['Logistic Regression', 'KNN', 'Decision Tree','Random Forest'])
performance_df4
```

Out[85]:

	Accuracy	Precision	Recall	F1 score
Logistic Regression	77.257800	24.353519	80.345572	37.377543
KNN	75.871191	21.050859	67.494600	32.092426
Decision Tree	89.518336	39.143135	43.412527	41.167435
Random Forest	92.391899	60.952381	27.645788	38.038633

In [86]:

```
# it is similar like .predict (it predict label for each observation) while
# .predict_proba (it predict probability for each observation)

y_pred_proba1=fit1.predict_proba(x_test)
y_pred_proba2=fit2.predict_proba(x_test)
y_pred_proba3=fit3.predict_proba(x_test)
y_pred_proba4=fit4.predict_proba(x_test)

print("For Logistic Regression:")
false_positive_rate1,true_positive_rate1,thresholds1=roc_curve(y_test,y_pred_proba1[:,1])
roc_auc1=auc(false_positive_rate1,true_positive_rate1)
print("area under the roc curve :",roc_auc1)

print("For KNN:")
false_positive_rate2,true_positive_rate2,thresholds2=roc_curve(y_test,y_pred_proba2[:,1])
roc_auc2=auc(false_positive_rate2,true_positive_rate2)
print("area under the roc curve :",roc_auc2)

print("For Decision tree:")
false_positive_rate3,true_positive_rate3,thresholds3=roc_curve(y_test,y_pred_proba3[:,1])
roc_auc3=auc(false_positive_rate3,true_positive_rate3)
print("area under the roc curve :",roc_auc3)

print("For Random Forest:")
false_positive_rate4,true_positive_rate4,thresholds4=roc_curve(y_test,y_pred_proba4[:,1])
roc_auc4=auc(false_positive_rate4,true_positive_rate4)
print("area under the roc curve :",roc_auc4)
```

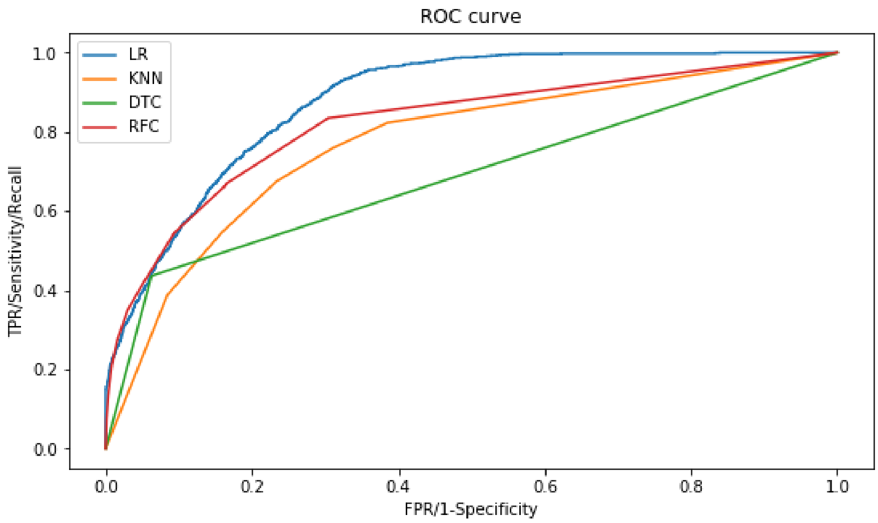
```
For Logistic Regression:
area under the roc curve : 0.8792149019469436
For KNN:
area under the roc curve : 0.7713947391980663
For Decision tree:
area under the roc curve : 0.6868802010386797
For Random Forest:
area under the roc curve : 0.8221179671110569
```

In [87]:

```
plt.figure(figsize=(9,5))
plt.plot(false_positive_rate1,true_positive_rate1,label = "LR")
plt.plot(false_positive_rate2,true_positive_rate2,label = "KNN")
plt.plot(false_positive_rate3,true_positive_rate3,label = "DTC")
plt.plot(false_positive_rate4,true_positive_rate4,label = "RFC")
plt.ylabel("TPR/Sensitivity/Recall")
plt.xlabel("FPR/1-Specificity")
plt.title("ROC curve")
plt.legend()
```

Out[87]:

```
&lt;matplotlib.legend.Legend at 0x21c608d5a58&gt;
```



Comparision of results from all four techniques:

In [88]:

```
performance_df1
```

Out[88]:

	Accuracy	Precision	Recall	F1 score
Logistic Regression	93.066959	84.873950	21.814255	34.707904
KNN	92.565225	72.111554	19.546436	30.756160
Decision Tree	90.120416	42.371234	47.084233	44.603581
Random Forest	93.057836	71.208226	29.913607	42.129278

In [89]:

```
performance_df2
```

Out[89]:

	Accuracy	Precision	Recall	F1 score
Logistic Regression	78.586724	76.199804	83.208556	79.550102
KNN	71.680942	71.875000	71.336898	71.604938
Decision Tree	76.927195	77.510917	75.935829	76.715289
Random Forest	77.194861	77.045696	77.540107	77.292111

In [90]:

```
performance_df3
```

Out[90]:

	Accuracy	Precision	Recall	F1 score
Logistic Regression	79.467491	77.772530	82.346469	79.994170
KNN	90.840646	84.718795	99.589918	91.554411
Decision Tree	96.499801	93.447373	99.989998	96.608040
Random Forest	98.160152	96.449590	99.989998	98.187890

In [91]:

```
performance_df4
```

Out[91]:

	Accuracy	Precision	Recall	F1 score
Logistic Regression	77.257800	24.353519	80.345572	37.377543
KNN	75.871191	21.050859	67.494600	32.092426
Decision Tree	89.518336	39.143135	43.412527	41.167435
Random Forest	92.391899	60.952381	27.645788	38.038633

9. Saving the models

In [92]:

```
import pickle
```

In [93]:

```
pickle.dump(fit1, open('logistic.pkl','wb'))
pickle.dump(fit2, open('knn.pkl','wb'))
pickle.dump(fit3, open('dtc.pkl','wb'))
pickle.dump(fit4, open('rfc.pkl','wb'))
```

In [94]:

```
logistic = pickle.load(open('logistic.pkl','rb'))
knn = pickle.load(open('knn.pkl','rb'))
dtc = pickle.load(open('dtc.pkl','rb'))
rfc = pickle.load(open('rfc.pkl','rb'))
```