Problem Statement

Business context

Employee Promotion means the ascension of an employee to higher ranks, this aspect of the job is what drives employees the most. The ultimate reward for dedication and loyalty towards an organization and the HR team plays an important role in handling all these promotion tasks based on ratings and other attributes available.

The HR team in JMD company stored data on the promotion cycle last year, which consists of details of all the employees in the company working last year and also if they got promoted or not, but every time this process gets delayed due to so many details available for each employee - it gets difficult to compare and decide.

Objective

For the upcoming appraisal cycle, the HR team wants to utilize the stored data and leverage machine learning to make a model that will predict if a person is eligible for promotion or not. You, as a data scientist at JMD company, need to come up with the best possible model that will help the HR team to predict if a person is eligible for promotion or not.

Data Description

- **employee_id: Unique ID for the 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 training completed in the 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**
- **awards_ won: if awards won during the previous year then 1 else 0**
- **avg *training* score: Average score in current training evaluations**
- **is_promoted: (Target) Recommended for promotion**

Please read the instructions carefully before starting the project.

This is a commented Jupyter IPython Notebook file in which all the instructions and tasks to be performed are mentioned.

- **Blanks '___' are provided in the notebook that needs to be filled with an appropriate code to get the correct result. With every '___' blank, there is a comment that briefly describes what needs to be filled in the blank space.**
- **Identify the task to be performed correctly, and only then proceed to write the required code.**
- **Fill the code wherever asked by the commented lines like "# write your code here" or "# complete the code". Running incomplete code may throw error.**
- **Please run the codes in a sequential manner from the beginning to avoid any unnecessary errors.**
- **Add the results/observations (wherever mentioned) derived from the analysis in the presentation and submit the same.**

Importing necessary libraries

In [3]:

```
# Installing the libraries with the specified version.
```

```
# uncomment and run the following line if Google Colab is being used
# !pip install scikit-learn==1.2.2 seaborn==0.13.1 matplotlib==3.7.1 numpy==1.25.2 pandas
==1.5.3 imbalanced-learn==0.10.1 xgboost==2.0.3 -q --user
```

In [4]:

```
# Installing the libraries with the specified version.
# uncomment and run the following lines if Jupyter Notebook is being used
# !pip install scikit-learn==1.2.2 seaborn==0.13.1 matplotlib==3.7.1 numpy==1.25.2 pandas
==1.5.3 imbalanced-learn==0.10.1 xgboost==2.0.3 -q --user
# !pip install --upgrade -q threadpoolctl
```

In [5]:

```
# Libraries to help with reading and manipulating data
import pandas as pd
import numpy as np

# Libaries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# To tune model, get different metric scores, and split data
from sklearn.metrics import (
    f1_score,
    accuracy_score,
    recall_score,
    precision_score,
    confusion_matrix,
    roc_auc_score,
    ConfusionMatrixDisplay,
)
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score

# To be used for data scaling and one hot encoding
from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder

# To oversample and undersample data
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler

# To do hyperparameter tuning
from sklearn.model_selection import RandomizedSearchCV

# To impute missing values
from sklearn.impute import SimpleImputer

# To define maximum number of columns to be displayed in a dataframe
pd.set_option("display.max_columns", None)

# To supress scientific notations for a dataframe
pd.set_option("display.float_format", lambda x: "%.3f" % x)

# To help with model building
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import (
    AdaBoostClassifier,
    GradientBoostingClassifier,
    RandomForestClassifier,
    BaggingClassifier,
)
from xgboost import XGBClassifier

# To suppress scientific notations
pd.set_option("display.float_format", lambda x: "%.3f" % x)

# To supress warnings
import warnings

warnings.filterwarnings("ignore")
```

Loading the dataset

In [6]:

```
promotion = pd.read_csv("employee_promotion.csv")
```

Data Overview

The initial steps to get an overview of any dataset is to:

- **observe the first few rows of the dataset, to check whether the dataset has been loaded properly or not**
- **get information about the number of rows and columns in the dataset**
- **find out the data types of the columns to ensure that data is stored in the preferred format and the value of each property is as expected.**
- **check the statistical summary of the dataset to get an overview of the numerical columns of the data**

Checking the shape of the dataset

In [7]:

```
# Checking the number of rows and columns in the training data
promotion.shape ##  Complete the code to view dimensions of the train data
```

Out[7]:

```
(54808, 13)
```

In [8]:

```
# let's create a copy of the data
data = promotion.copy()
```

Displaying the first few rows of the dataset

In [9]:

```
# let's view the first 5 rows of the data
data.head() ##  Complete the code to view top 5 rows of the data
```

Out[9]:

	employee_id	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_year_rating	l
0	65438	Sales & Marketing	region_7	Master's & above	f	sourcing	1	35	5.000	
1	65141	Operations	region_22	Bachelor's	m	other	1	30	5.000	
2	7513	Sales & Marketing	region_19	Bachelor's	m	sourcing	1	34	3.000	
3	2542	Sales & Marketing	region_23	Bachelor's	m	other	2	39	1.000	
4	48945	Technology	region_26	Bachelor's	m	other	1	45	3.000	

In [10]:

```
# let's view the last 5 rows of the data
data.tail() ##  Complete the code to view last 5 rows of the data
```

Out[10]:

	employee_id	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_year_ratir

	employee_id	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_year_ratir
54803	3030	Technology	region_14	Bachelor's	m	sourcing	1	48	3.0
54804	74592	Operations	region_27	Master's & above	f	other	1	37	2.0
54805	13918	Analytics	region_1	Bachelor's	m	other	1	27	5.0
54806	13614	Sales & Marketing	region_9	NaN	m	sourcing	1	29	1.0
54807	51526	HR	region_22	Bachelor's	m	other	1	27	1.0

Checking the data types of the columns for the dataset

In [11]:

```
# let's check the data types of the columns in the dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54808 entries, 0 to 54807
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   employee_id           54808 non-null  int64  
 1   department            54808 non-null  object 
 2   region                54808 non-null  object 
 3   education             52399 non-null  object 
 4   gender                54808 non-null  object 
 5   recruitment_channel   54808 non-null  object 
 6   no_of_trainings       54808 non-null  int64  
 7   age                   54808 non-null  int64  
 8   previous_year_rating  50684 non-null  float64
 9   length_of_service     54808 non-null  int64  
 10  awards_won            54808 non-null  int64  
 11  avg_training_score    52248 non-null  float64
 12  is_promoted           54808 non-null  int64  
dtypes: float64(2), int64(6), object(5)
memory usage: 5.4+ MB
```

Checking for duplicate values

In [12]:

```
# let's check for duplicate values in the data
data.duplicated().sum()  ##  Complete the code to check duplicate entries in the data
```

Out[12]:

```
0
```

Checking for missing values

In [13]:

```
# let's check for missing values in the data
data.isnull().sum()  ##  Complete the code to check missing entries in the train data
```

Out[13]:

```
employee_id                0
department                 0
region                     0
education               2409
gender                     0
recruitment_channel        0
no_of_trainings            0
age                        0
previous_year_rating    4124
```

```
previous_year_rating    4124
length_of_service          0
awards_won                 0
avg_training_score      2560
is_promoted                0
dtype: int64
```

Statistical summary of the dataset

In [14]:

```
# let's view the statistical summary of the numerical columns in the data
data.describe() ##  Complete the code to print the statitical summary of the train data
```

Out[14]:

	employee_id	no_of_trainings	age	previous_year_rating	length_of_service	awards_won	avg_training_score	is_pr
count	54808.000	54808.000	54808.000	50684.000	54808.000	54808.000	52248.000	54
mean	39195.831	1.253	34.804	3.329	5.866	0.023	63.712	
std	22586.581	0.609	7.660	1.260	4.265	0.150	13.522	
min	1.000	1.000	20.000	1.000	1.000	0.000	39.000	
25%	19669.750	1.000	29.000	3.000	3.000	0.000	51.000	
50%	39225.500	1.000	33.000	3.000	5.000	0.000	60.000	
75%	58730.500	1.000	39.000	4.000	7.000	0.000	77.000	
max	78298.000	10.000	60.000	5.000	37.000	1.000	99.000	

In [15]:

```
# let's view the statistical summary of the numerical columns in the data
data.describe().T
```

Out[15]:

	count	mean	std	min	25%	50%	75%	max
employee_id	54808.000	39195.831	22586.581	1.000	19669.750	39225.500	58730.500	78298.000
no_of_trainings	54808.000	1.253	0.609	1.000	1.000	1.000	1.000	10.000
age	54808.000	34.804	7.660	20.000	29.000	33.000	39.000	60.000
previous_year_rating	50684.000	3.329	1.260	1.000	3.000	3.000	4.000	5.000
length_of_service	54808.000	5.866	4.265	1.000	3.000	5.000	7.000	37.000
awards_won	54808.000	0.023	0.150	0.000	0.000	0.000	0.000	1.000
avg_training_score	52248.000	63.712	13.522	39.000	51.000	60.000	77.000	99.000
is_promoted	54808.000	0.085	0.279	0.000	0.000	0.000	0.000	1.000

Let's check the number of unique values in each column

In [16]:

```
data.nunique()
```

Out[16]:

```
employee_id           54808
department                9
region                   34
education                 3
gender                    2
recruitment_channel       3
no_of_trainings          10
age                      41
```

```
age                          41
previous_year_rating          5
length_of_service            35
awards_won                    2
avg_training_score           59
is_promoted                   2
dtype: int64
```

In [17]:

```
for i in data.describe(include=["object"]).columns:
    print("Unique values in", i, "are :")
    print(data[i].value_counts())
    print("*" * 50)
```

```
Unique values in department are :
department
Sales & Marketing    16840
Operations           11348
Technology            7138
Procurement           7138
Analytics             5352
Finance               2536
HR                    2418
Legal                 1039
R&D                    999
Name: count, dtype: int64
**************************************************
Unique values in region are :
region
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: count, dtype: int64
**************************************************
Unique values in education are :
education
Bachelor's          36669
Master's & above    14925
Below Secondary       805
Name: count, dtype: int64
**************************************************
```

```
Unique values in gender are :
gender
m    38496
f    16312
Name: count, dtype: int64
**************************************************
Unique values in recruitment_channel are :
recruitment_channel
other       30446
sourcing    23220
referred     1142
Name: count, dtype: int64
**************************************************
```

In [18]:

```
# ID column consists of uniques ID for clients and hence will not add value to the modeli
ng
data.drop(columns="employee_id", inplace=True)
```

In [19]:

```
data["is_promoted"].value_counts(1)
```

Out[19]:

```
is_promoted
0    0.915
1    0.085
Name: proportion, dtype: float64
```

Exploratory Data Analysis

The below functions need to be defined to carry out the Exploratory Data Analysis.

In [20]:

```
# function to plot a boxplot and a histogram along the same scale.


def histogram_boxplot(data, feature, figsize=(12, 7), kde=False, bins=None):
    """
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    kde: whether to the show density curve (default False)
    bins: number of bins for histogram (default None)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=data, x=feature, ax=ax_box2, showmeans=True, color="violet"
    )  # boxplot will be created and a triangle will indicate the mean value of the colum
n
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins, palette="winter"
    ) if bins else sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data[feature].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
```

```python
    ax_hist2.axvline(
        data[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

In [21]:

```python
# function to create labeled barplots


def labeled_barplot(data, feature, perc=False, n=None):
    """
    Barplot with percentage at the top

    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of count (default is False)
    n: displays the top n category levels (default is None, i.e., display all levels)
    """

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 1, 5))
    else:
        plt.figure(figsize=(n + 1, 5))

    plt.xticks(rotation=90, fontsize=15)
    ax = sns.countplot(
        data=data,
        x=feature,
        palette="Paired",
        order=data[feature].value_counts().index[:n].sort_values(),
    )

    for p in ax.patches:
        if perc == True:
            label = "{:.1f}%".format(
                100 * p.get_height() / total
            )  # percentage of each class of the category
        else:
            label = p.get_height()  # count of each level of the category

        x = p.get_x() + p.get_width() / 2  # width of the plot
        y = p.get_height()  # height of the plot

        ax.annotate(
            label,
            (x, y),
            ha="center",
            va="center",
            size=12,
            xytext=(0, 5),
            textcoords="offset points",
        )  # annotate the percentage

    plt.show()  # show the plot
```

In [22]:

```python
# function to plot stacked bar chart

def stacked_barplot(data, predictor, target):
    """
    Print the category counts and plot a stacked bar chart

    data: dataframe
    predictor: independent variable
    target: target variable
    """
    count = data[predictor].nunique()
    sorter = data[target].value_counts().index[-1]
```

```
    tab1 = pd.crosstab(data[predictor], data[target], margins=True).sort_values(
        by=sorter, ascending=False
    )
    print(tab1)
    print("-" * 120)
    tab = pd.crosstab(data[predictor], data[target], normalize="index").sort_values(
        by=sorter, ascending=False
    )
    tab.plot(kind="bar", stacked=True, figsize=(count + 1, 5))
    plt.legend(
        loc="lower left", frameon=False,
    )
    plt.legend(loc="upper left", bbox_to_anchor=(1, 1))
    plt.show()
```

In [23]:

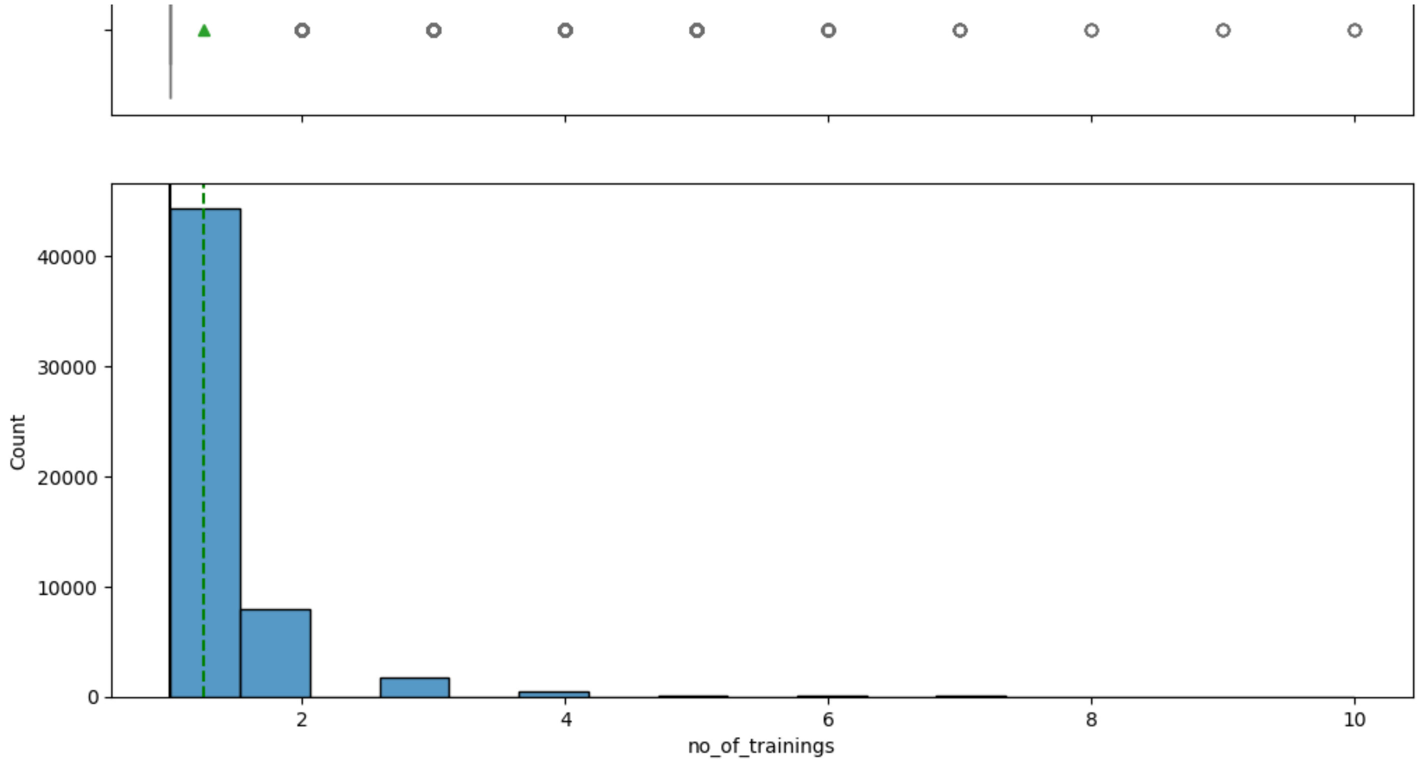
```
### Function to plot distributions

def distribution_plot_wrt_target(data, predictor, target):

    fig, axs = plt.subplots(2, 2, figsize=(12, 10))

    target_uniq = data[target].unique()

    axs[0, 0].set_title("Distribution of target for target=" + str(target_uniq[0]))
    sns.histplot(
        data=data[data[target] == target_uniq[0]],
        x=predictor,
        kde=True,
        ax=axs[0, 0],
        color="teal",
    )

    axs[0, 1].set_title("Distribution of target for target=" + str(target_uniq[1]))
    sns.histplot(
        data=data[data[target] == target_uniq[1]],
        x=predictor,
        kde=True,
        ax=axs[0, 1],
        color="orange",
    )

    axs[1, 0].set_title("Boxplot w.r.t target")
    sns.boxplot(data=data, x=target, y=predictor, ax=axs[1, 0], palette="gist_rainbow")

    axs[1, 1].set_title("Boxplot (without outliers) w.r.t target")
    sns.boxplot(
        data=data,
        x=target,
        y=predictor,
        ax=axs[1, 1],
        showfliers=False,
        palette="gist_rainbow",
    )

    plt.tight_layout()
    plt.show()
```

Univariate analysis

Observations on No. of Trainings

In [24]:

```
histogram_boxplot(data, "no_of_trainings")
```

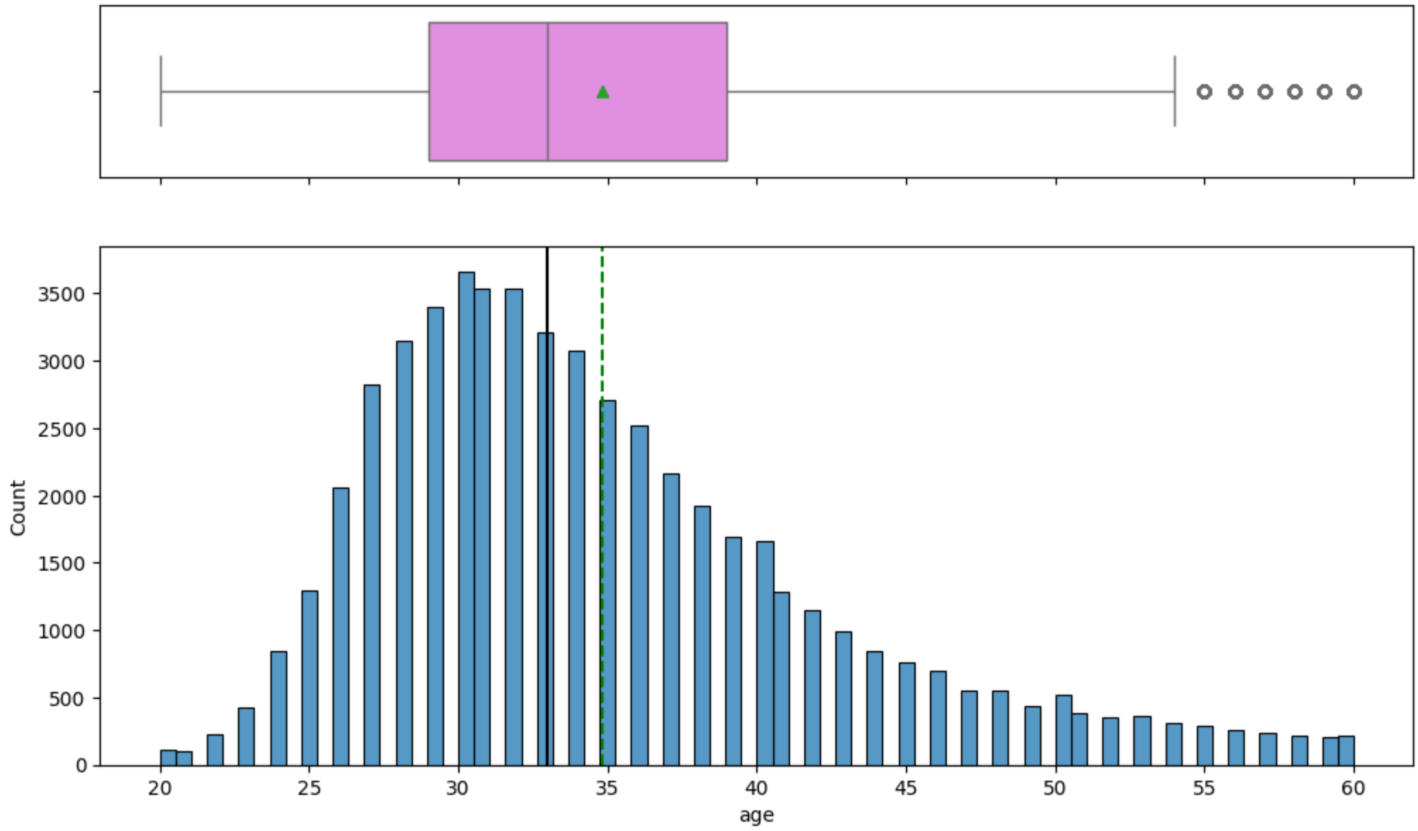


Let's see the distribution of age of employee

Observations on Age

In [25]:

```
histogram_boxplot(data, "age")  ## Complete the code to create histogram_boxplot for 'age
'
```

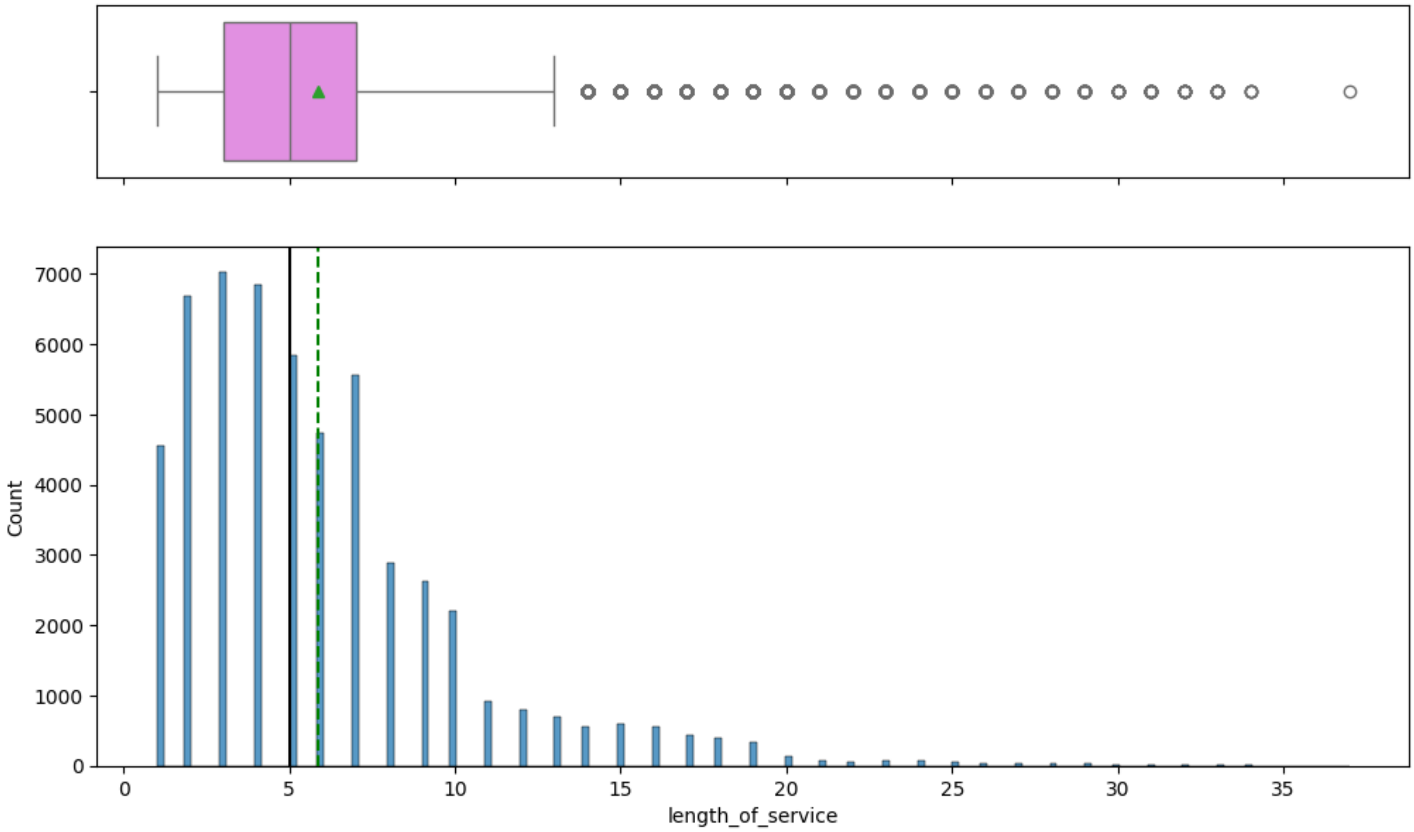


Observations on Length of Service

In [26]:

```
histogram_boxplot(data, "length_of_service")  ## Complete the code to create histogram_bo
```

```
xplot for 'length_of_service'
```

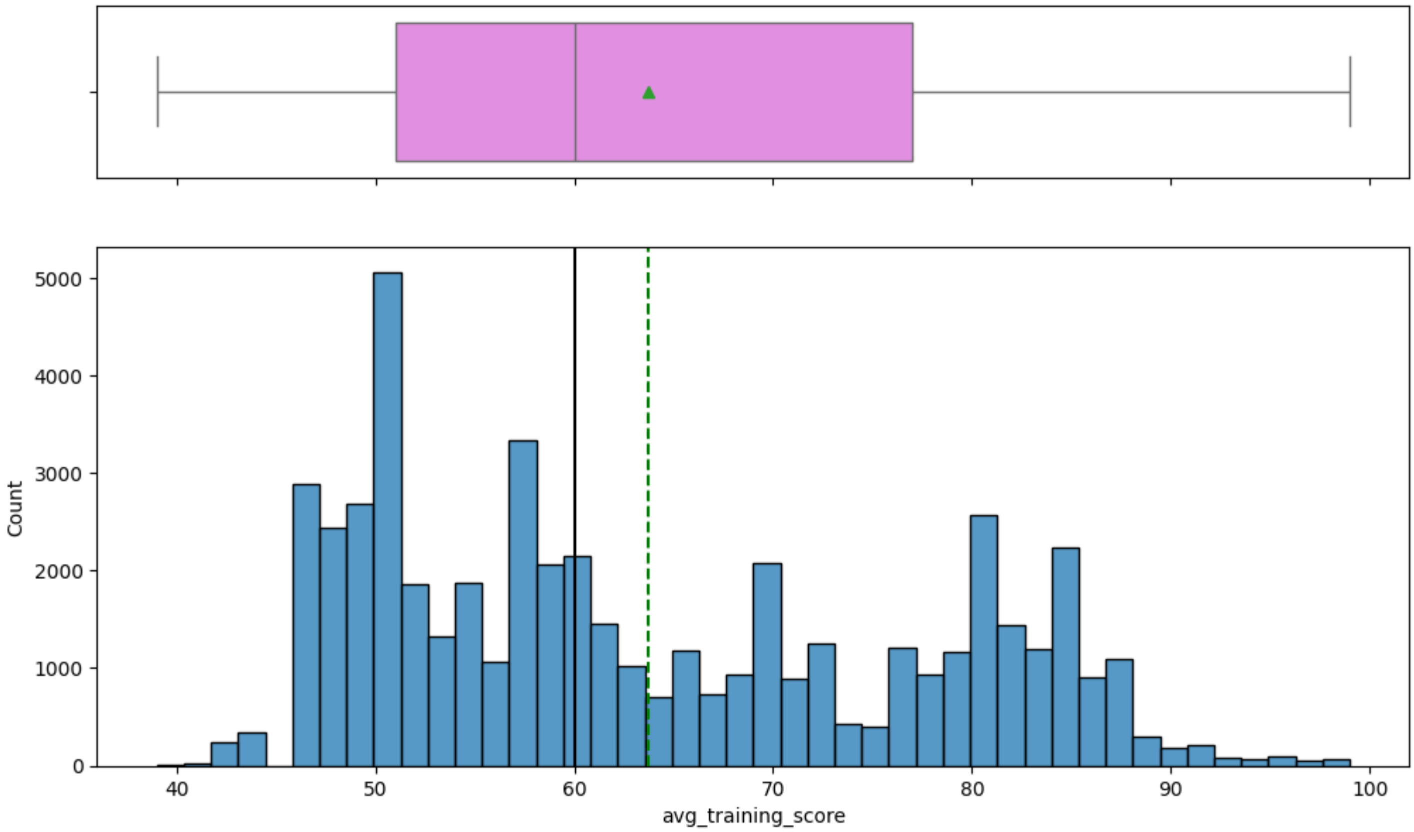


Let's see the distribution of average training score of employee

Observations on Average Training Score

In [27]:

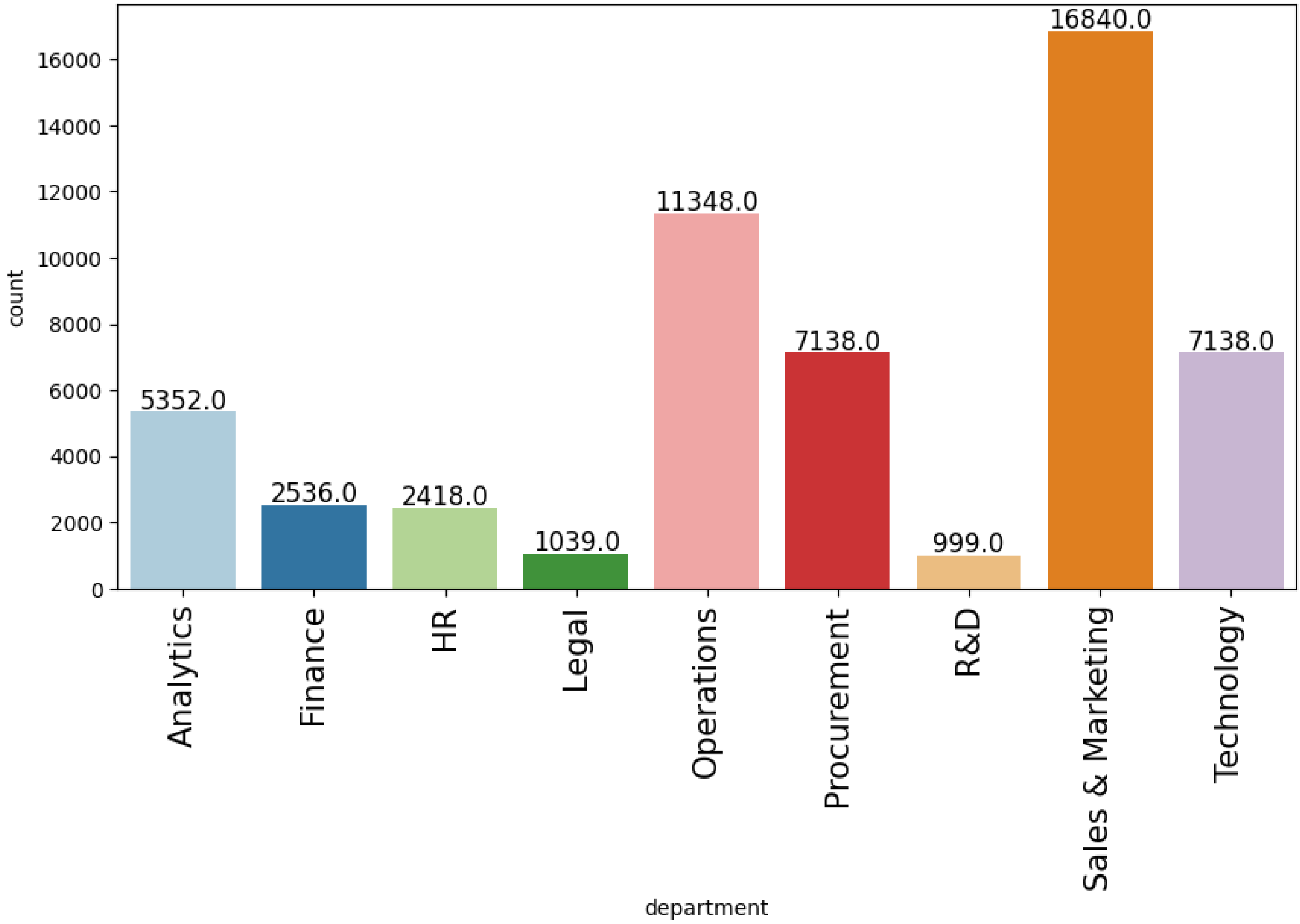
```
histogram_boxplot(data, "avg_training_score")  ## Complete the code to create histogram_b
oxplot for 'avg_training_score'
```



Observations on Department

In [28]:

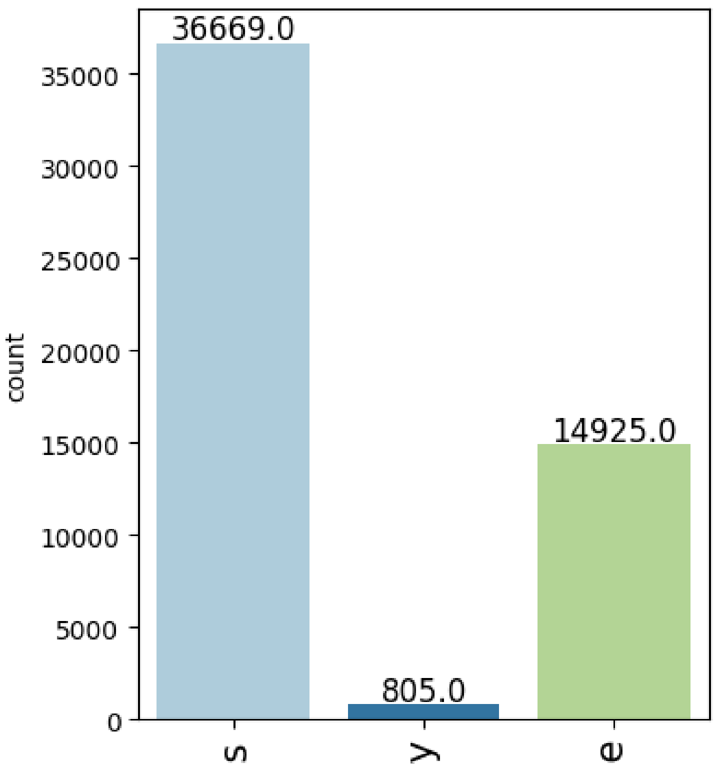
```
labeled_barplot(data, "department")
```

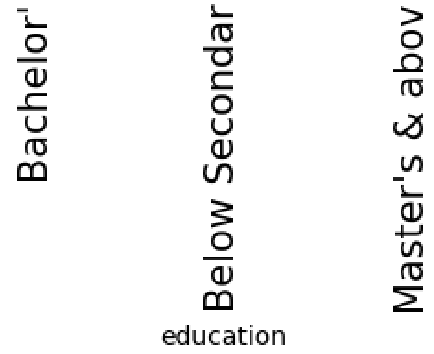


Observations on Education

In [29]:

```
labeled_barplot(data, "education") ## Complete the code to create labeled_barplot for 'ed
ucation'
```

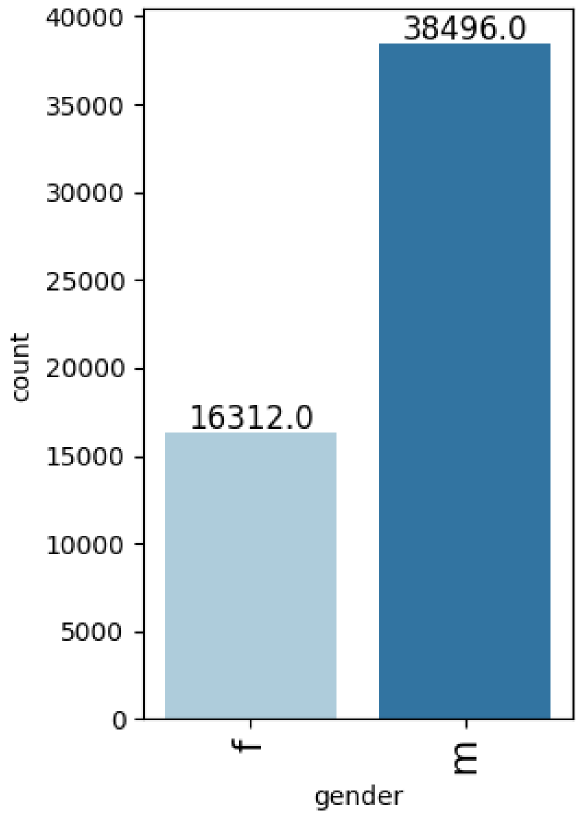




Observations on Gender

In [30]:

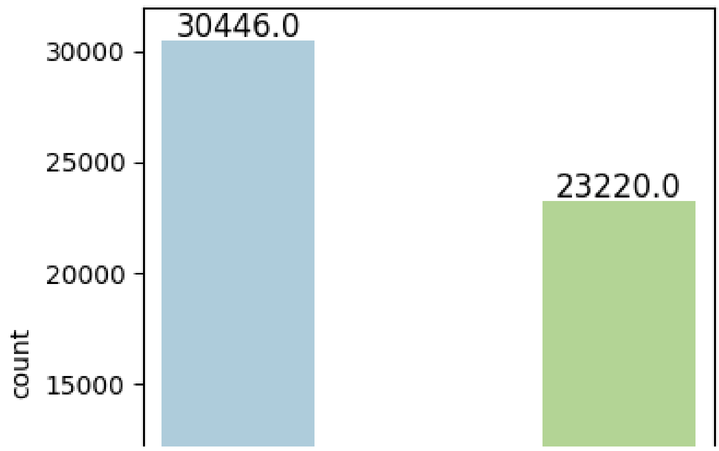
```
labeled_barplot(data, "gender") ## Complete the code to create labeled_barplot for 'gende
r'
```

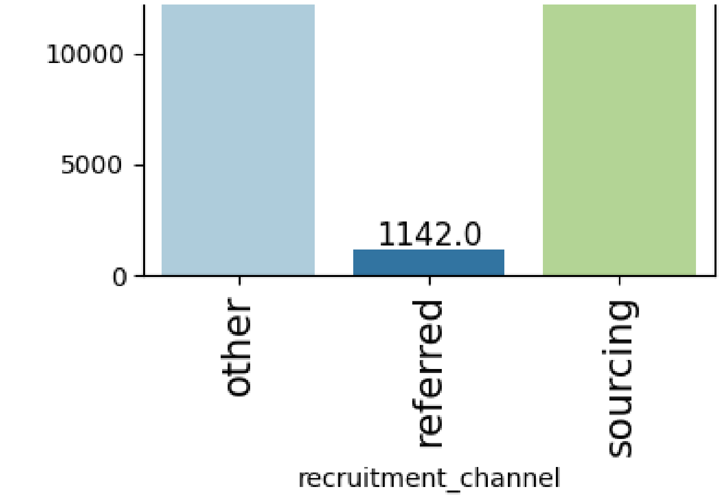


Observations on Recruitment Channel

In [31]:

```
labeled_barplot(data, "recruitment_channel") ## Complete the code to create labeled_barpl
ot for 'recruitment_channel'
```

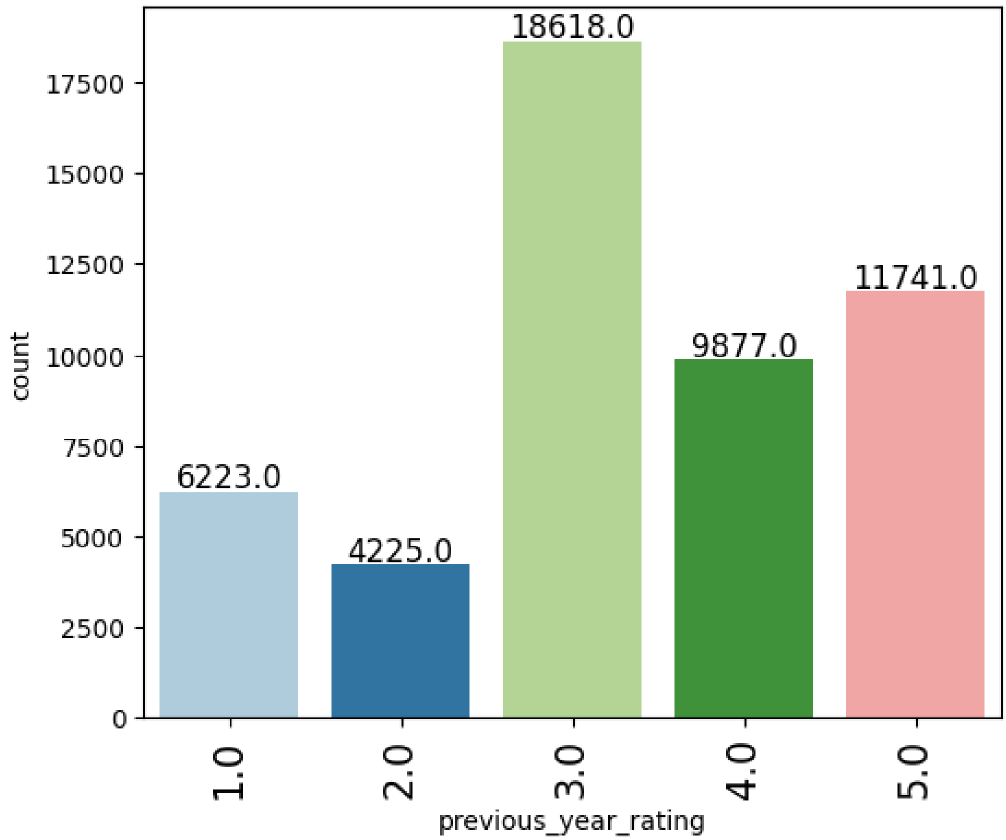




Observations on Previous Year Rating

In [32]:

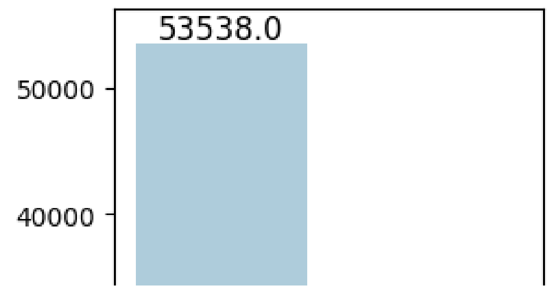
```
labeled_barplot(data, "previous_year_rating") ## Complete the code to create labeled_barp
lot for 'previous_year_rating'
```

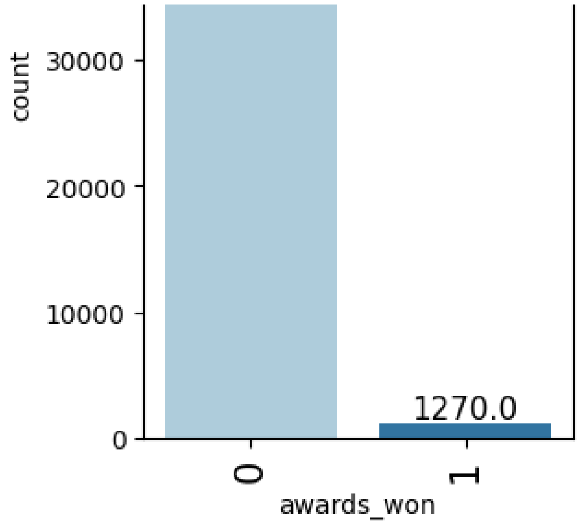


Observations on Awards Won

In [33]:

```
labeled_barplot(data, "awards_won") ## Complete the code to create labeled_barplot for 'a
wards_won'
```

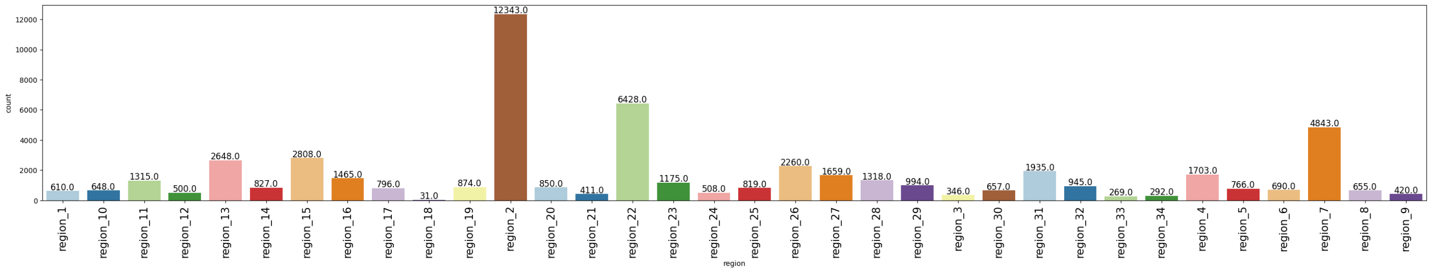




Observations on Region

In [34]:

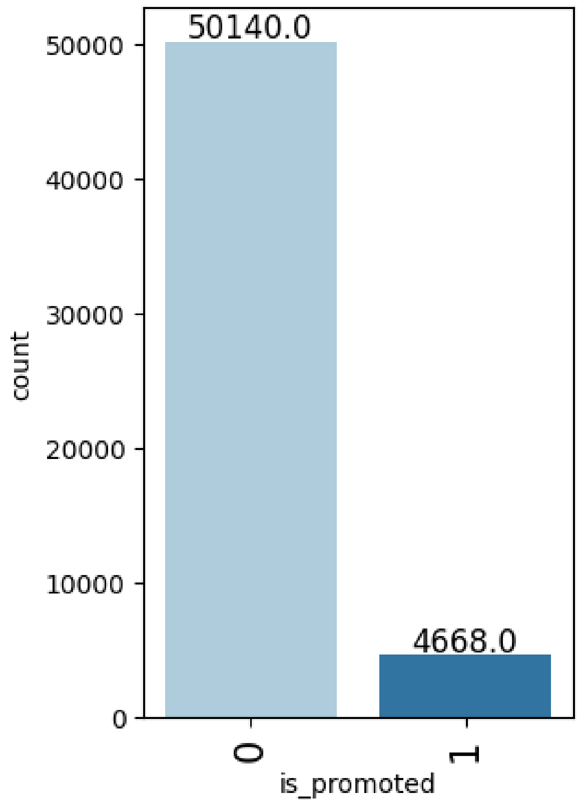
```
labeled_barplot(data, "region") ## Complete the code to create labeled_barplot for 'region'
```



Observations on target variable

In [35]:

```
labeled_barplot(data, "is_promoted") ## Complete the code to create labeled_barplot for 'is_promoted'
```



Bivariate Analysis

In [36]:

```
sns.pairplot(data, hue="is_promoted")
```

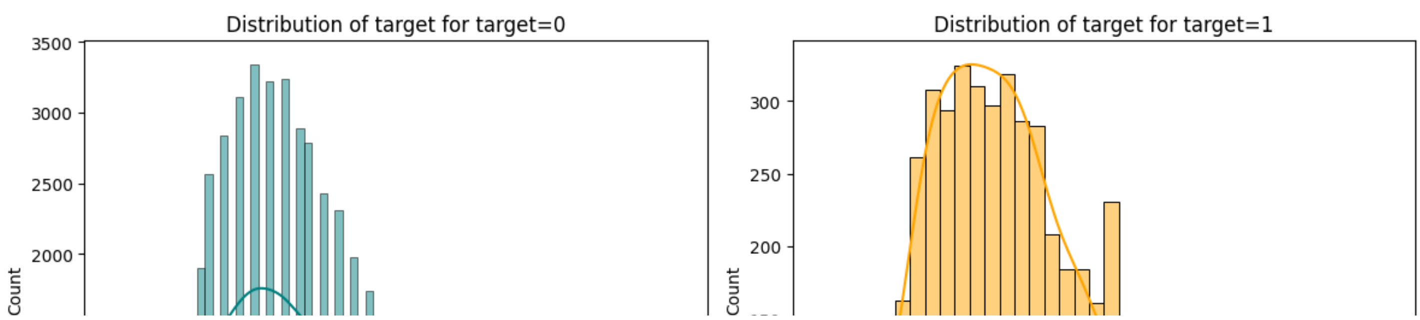
Out[36]:

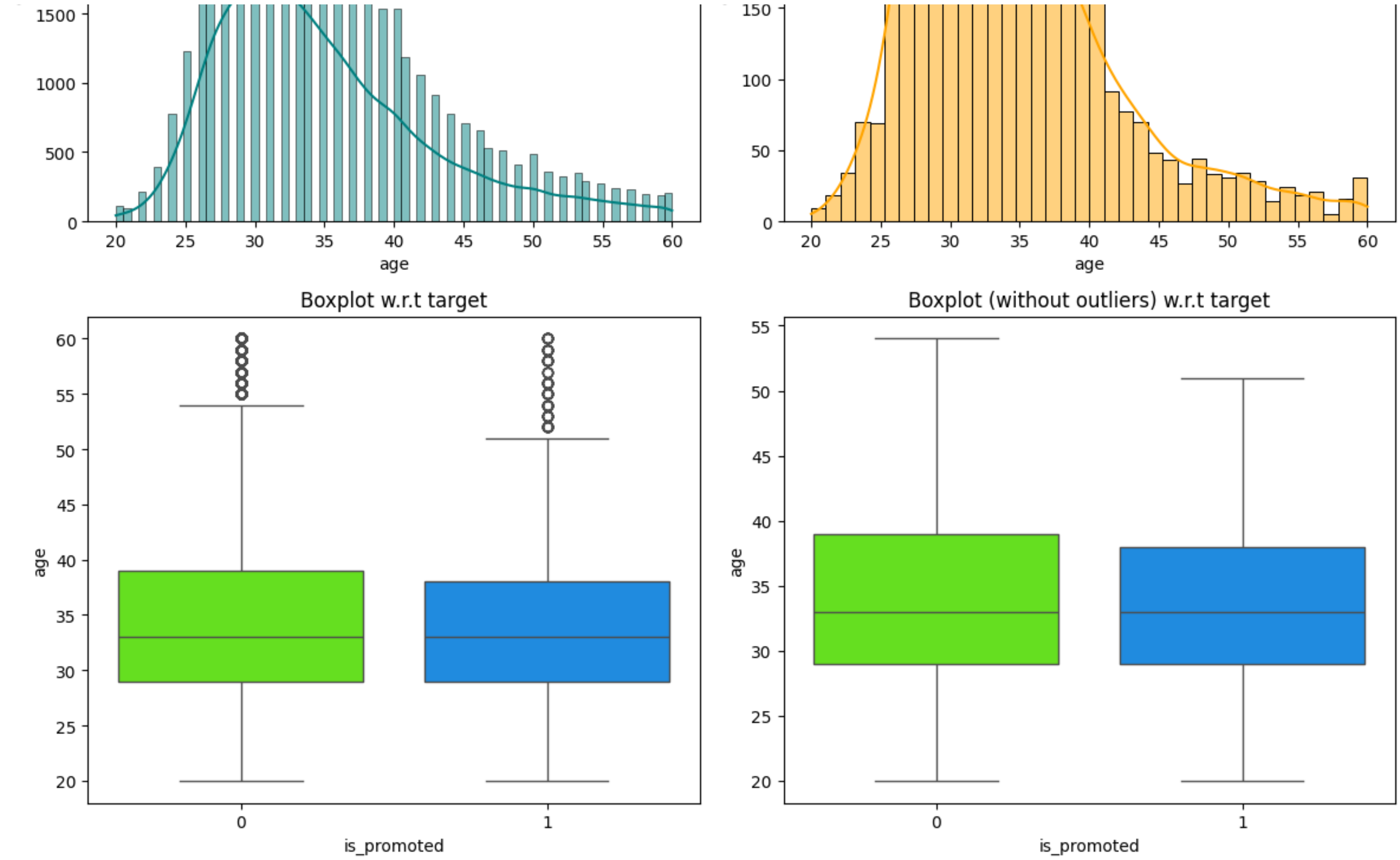
```
<seaborn.axisgrid.PairGrid at 0x227014b78e0>
```

Target variable vs Age

In [37]:

```
distribution_plot_wrt_target(data, "age", "is_promoted")
```



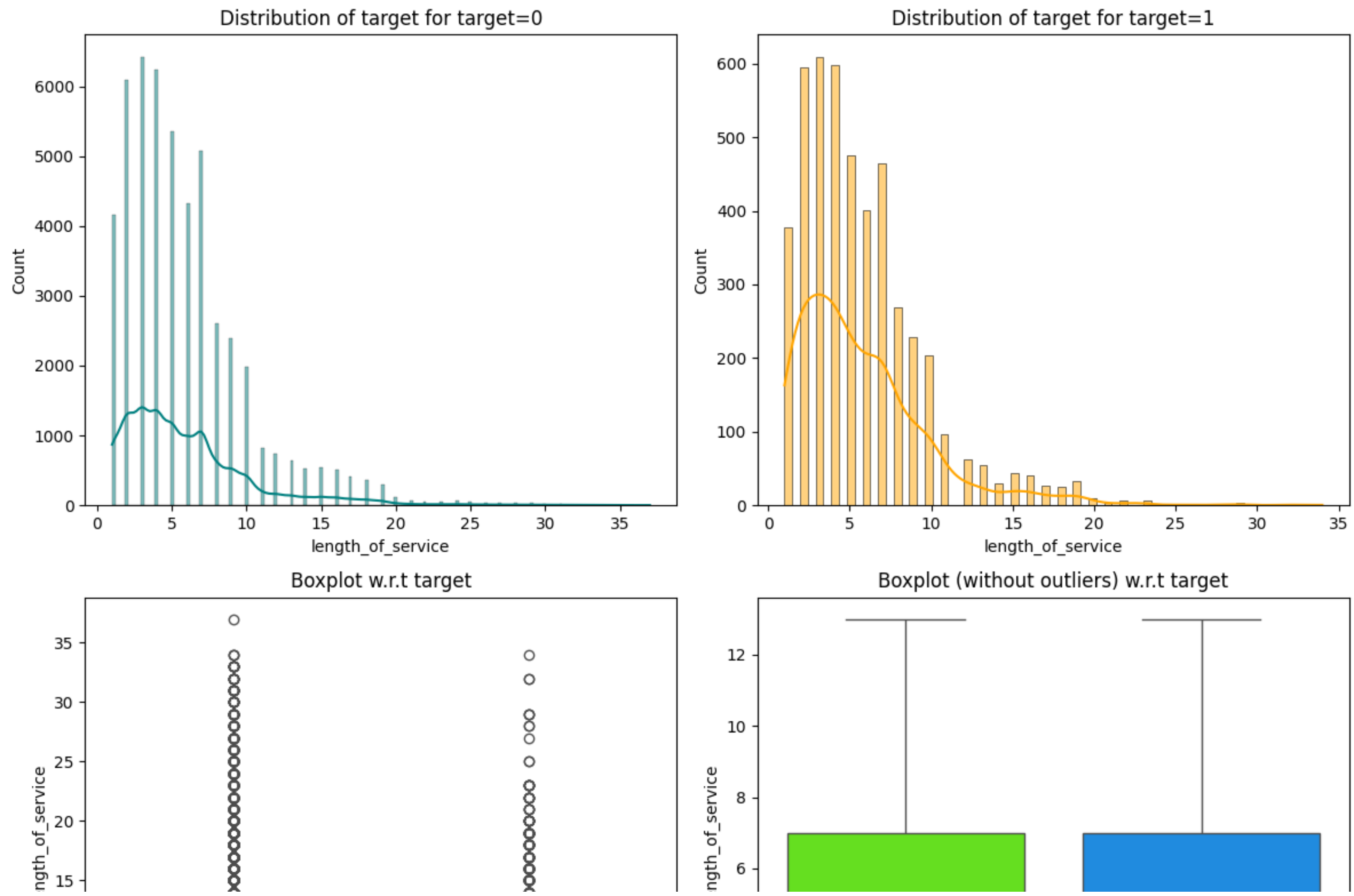


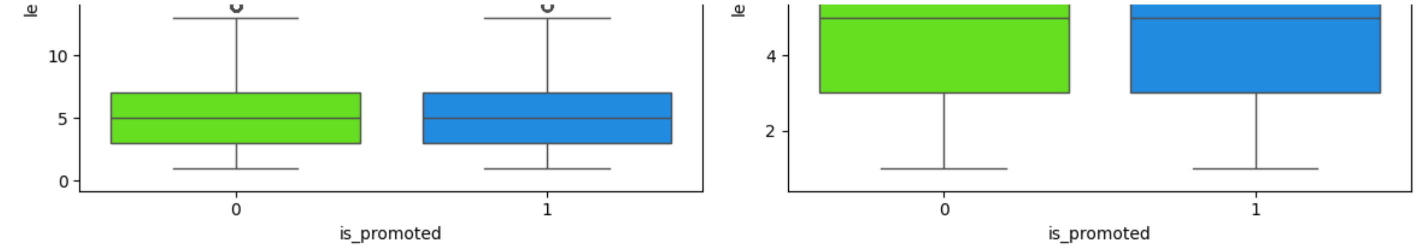
Let's see the change in length of service (length_of_service) vary by the employee's promotion status (is_promoted)?

Target variable vs Length of Service

In [38]:

```
distribution_plot_wrt_target(data, "length_of_service", "is_promoted") ## Complete the code to create distribution_plot for length_of_service vs is_promoted
```

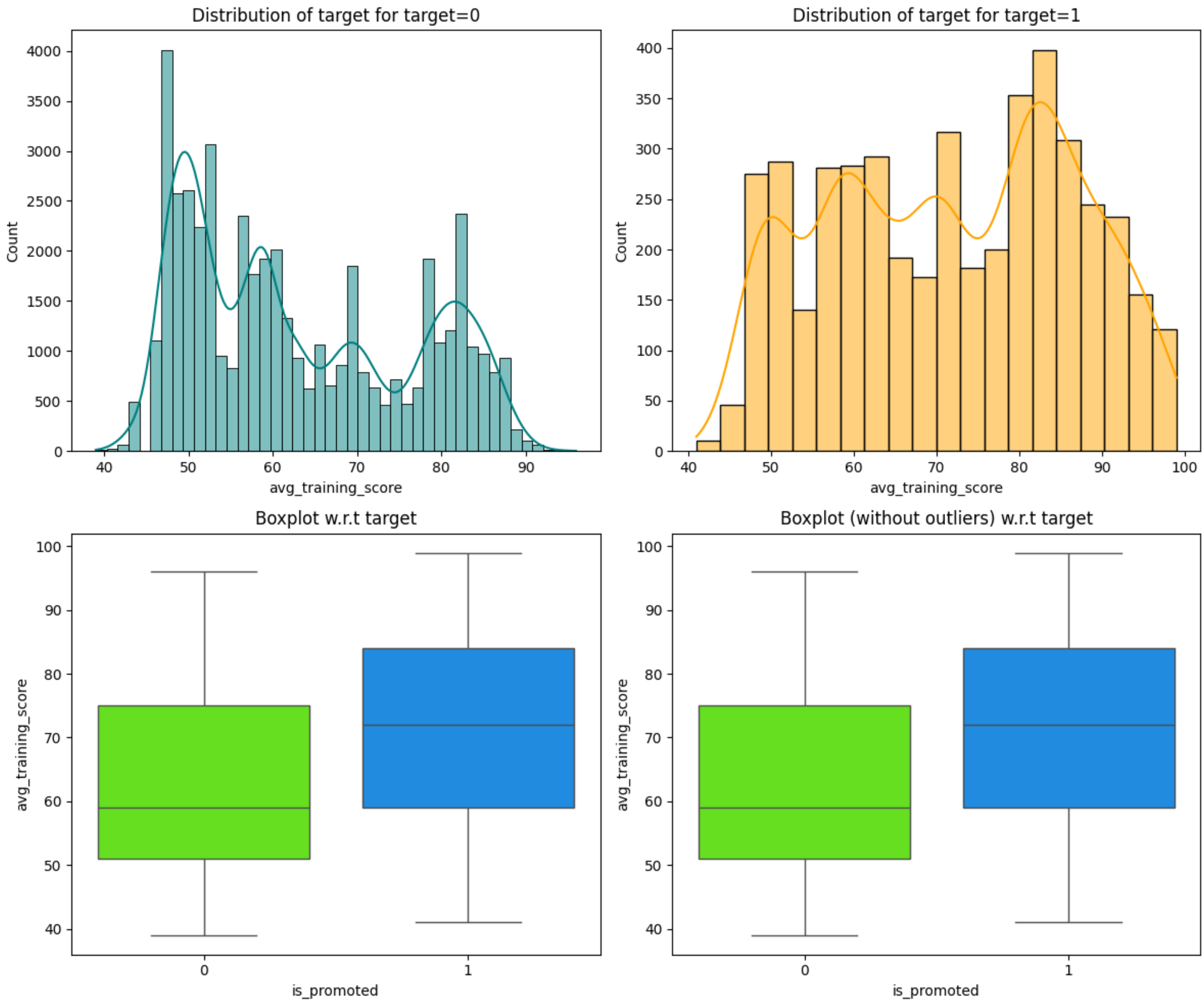




Target variable vs Average Training Score

In [39]:

```
distribution_plot_wrt_target(data, "avg_training_score", "is_promoted") ## Complete the c
ode to create distribution_plot for avg_training_score vs is_promoted
```



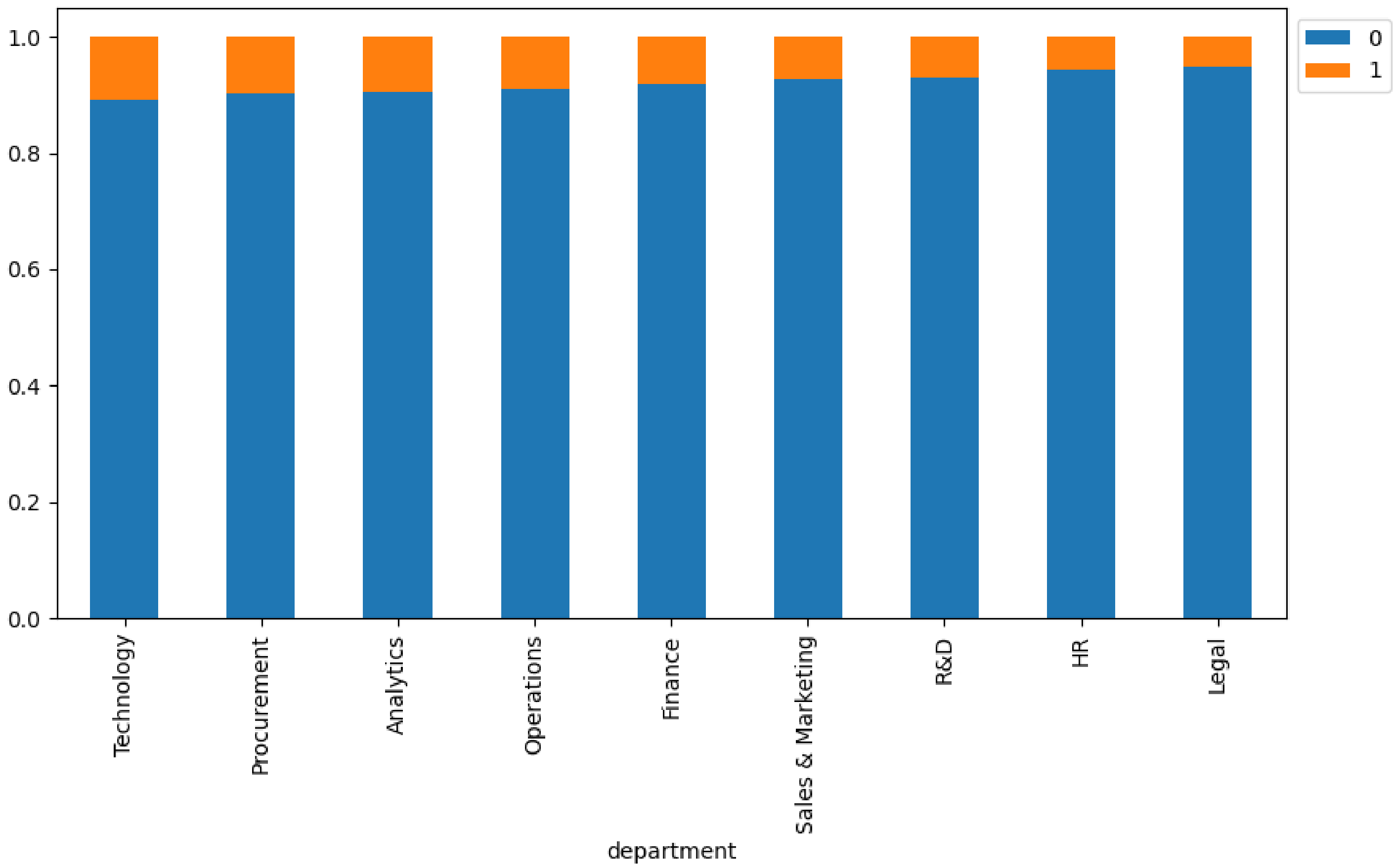
Target variable vs Department

In [40]:

```
stacked_barplot(data, "department", "is_promoted")
```

```
is_promoted           0     1    All
department
All               50140  4668  54808
Sales & Marketing 15627  1213  16840
Operations        10325  1023  11348
Technology         6370   768   7138
Procurement        6450   688   7138
Analytics          4840   512   5352
```

```
Finance              2330   206   2536
HR                   2282   136   2418
R&D                   930    69    999
Legal                 986    53   1039
------------------------------------------------------------------------------------------
-------------------------------------
```



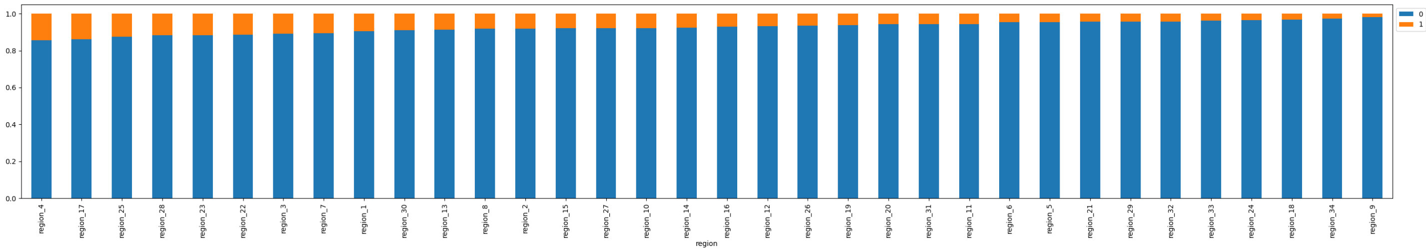
Target variable vs Region

In [41]:

```
stacked_barplot(data, "region", "is_promoted")
```

```
is_promoted      0     1    All
region
All          50140  4668  54808
region_2     11354   989  12343
region_22     5694   734   6428
region_7      4327   516   4843
region_4      1457   246   1703
region_13     2418   230   2648
region_15     2586   222   2808
region_28     1164   154   1318
region_26     2117   143   2260
region_23     1038   137   1175
region_27     1528   131   1659
region_31     1825   110   1935
region_17      687   109    796
region_25      716   103    819
region_16     1363   102   1465
region_11     1241    74   1315
region_14      765    62    827
region_30      598    59    657
region_1       552    58    610
region_19      821    53    874
region_8       602    53    655
region_10      597    51    648
region_20      801    49    850
region_29      951    43    994
region_32      905    40    945
region_3       309    37    346
```

```
region_5          731     35     766
region_12         467     33     500
region_6          658     32     690
region_24         490     18     508
region_21         393     18     411
region_33         259     10     269
region_34         284      8     292
region_9          412      8     420
region_18          30      1      31
------------------------------------------------------------------------------------------
---------------------------------
```

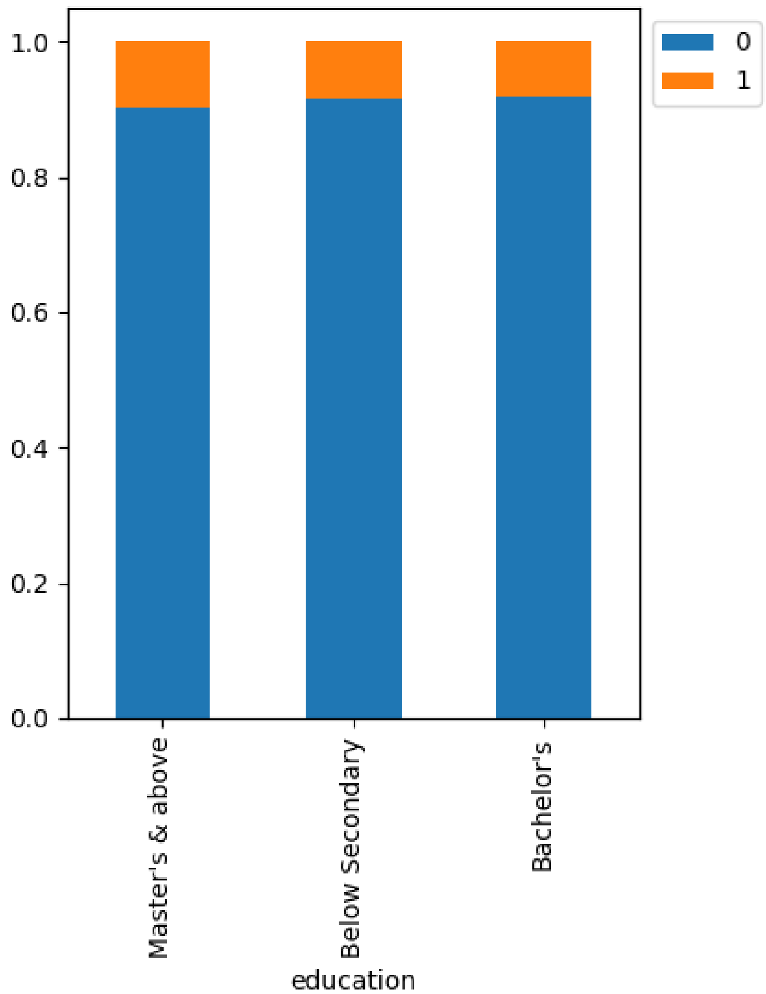


Target variable vs Education

In [42]:

```
stacked_barplot(data,"education", "is_promoted") ## Complete the code to create distribut
ion_plot for education vs is_promoted
```

```
is_promoted            0      1     All
education
All                47853   4546   52399
Bachelor's         33661   3008   36669
Master's & above   13454   1471   14925
Below Secondary      738     67     805
------------------------------------------------------------------------------------------
---------------------------------
```

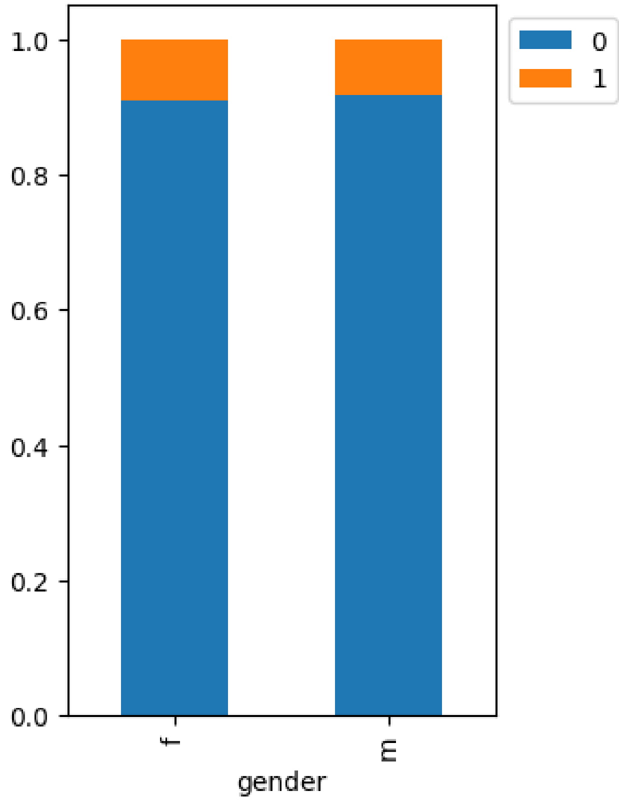


Target variable vs Gender

In [43]:

```
stacked_barplot(data,"gender", "is_promoted") ## Complete the code to create distribution_plot for gender vs is_promoted
```

```
is_promoted      0     1    All
gender
All          50140  4668  54808
m            35295  3201  38496
f            14845  1467  16312
------------------------------------------------------------------------------------------------------------------------
```

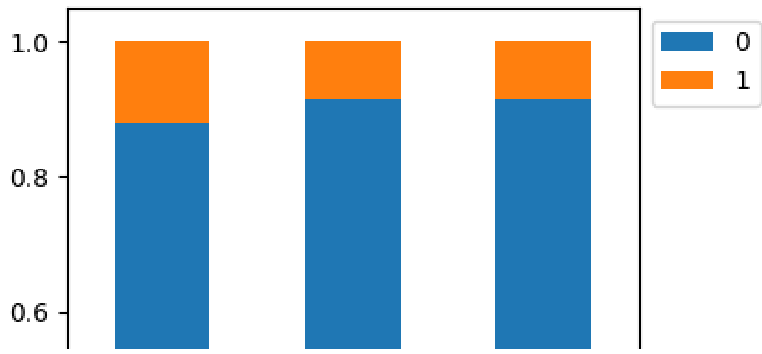


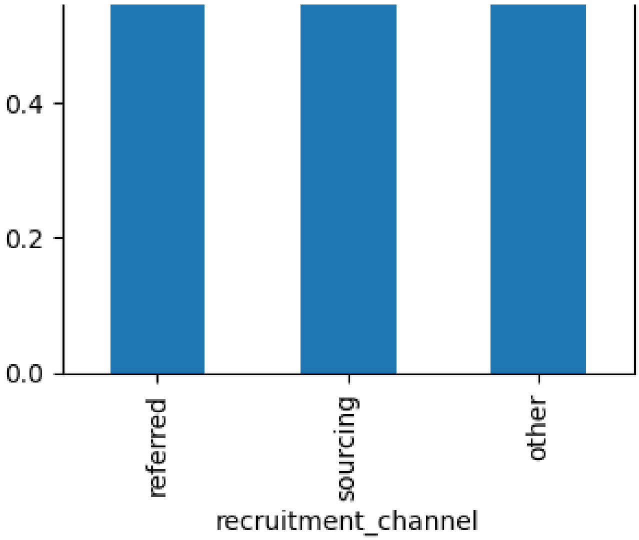
Target variable vs Recruitment Channel

In [44]:

```
stacked_barplot(data,"recruitment_channel", "is_promoted") ## Complete the code to create distribution_plot for recruitment_channel vs is_promoted
```

```
is_promoted              0     1    All
recruitment_channel
All                  50140  4668  54808
other                27890  2556  30446
sourcing             21246  1974  23220
referred              1004   138   1142
------------------------------------------------------------------------------------------------------------------------
```





Let's see the previous rating(previous_year_rating) vary by the employee's promotion status (is_promoted)

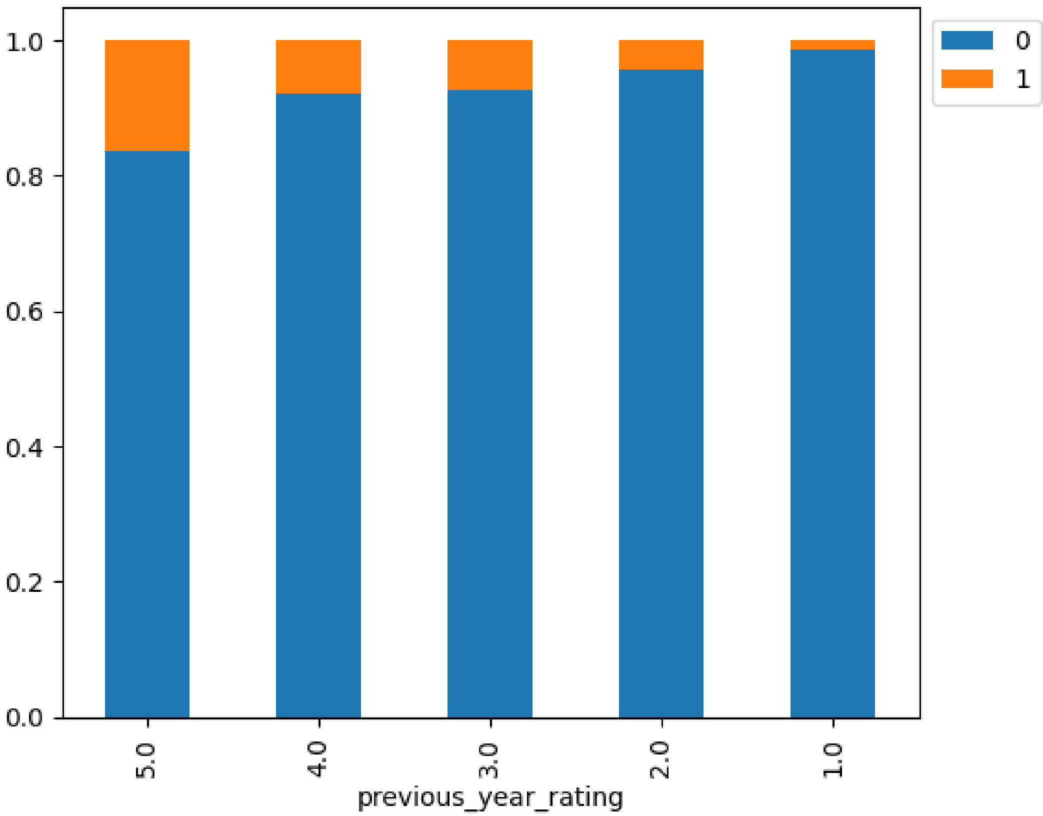
Target variable vs Previous Year Rating

In [45]:

```
stacked_barplot(data,"previous_year_rating", "is_promoted") ## Complete the code to creat
e distribution_plot for previous_year_rating vs is_promoted
```

is_promoted previous_year_rating	0	1	All
All	46355	4329	50684
5.0	9820	1921	11741
3.0	17263	1355	18618
4.0	9093	784	9877
2.0	4044	181	4225
1.0	6135	88	6223

```
------------------------------------------------------------------------------------------
----------------------------------
```

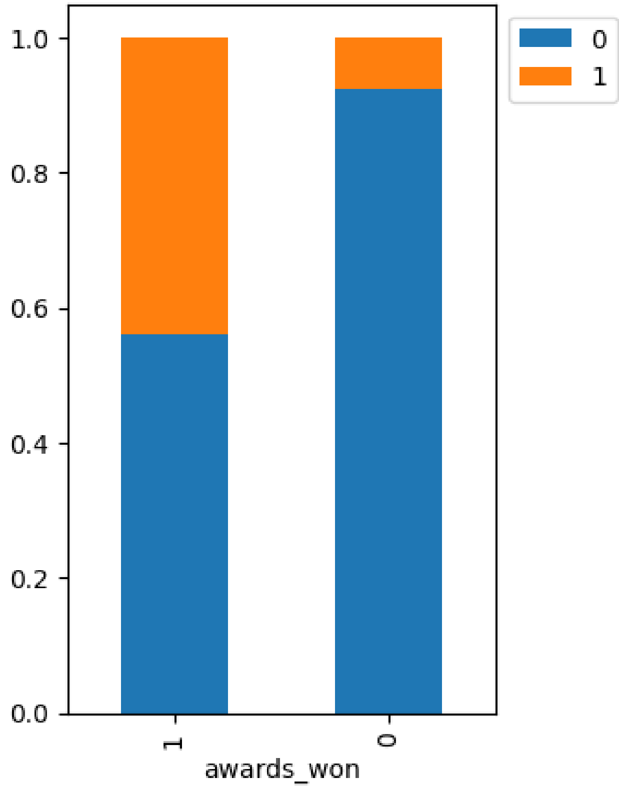


Target variable vs Awards Won

In [46]:

```
stacked_barplot(data,"awards_won", "is_promoted") ## Complete the code to create distribu
tion_plot for awards_won vs is_promoted
```

```
is_promoted      0     1    All
awards_won
All          50140  4668  54808
0            49429  4109  53538
1              711   559   1270
------------------------------------------------------------------------------------------
----------------------------------
```

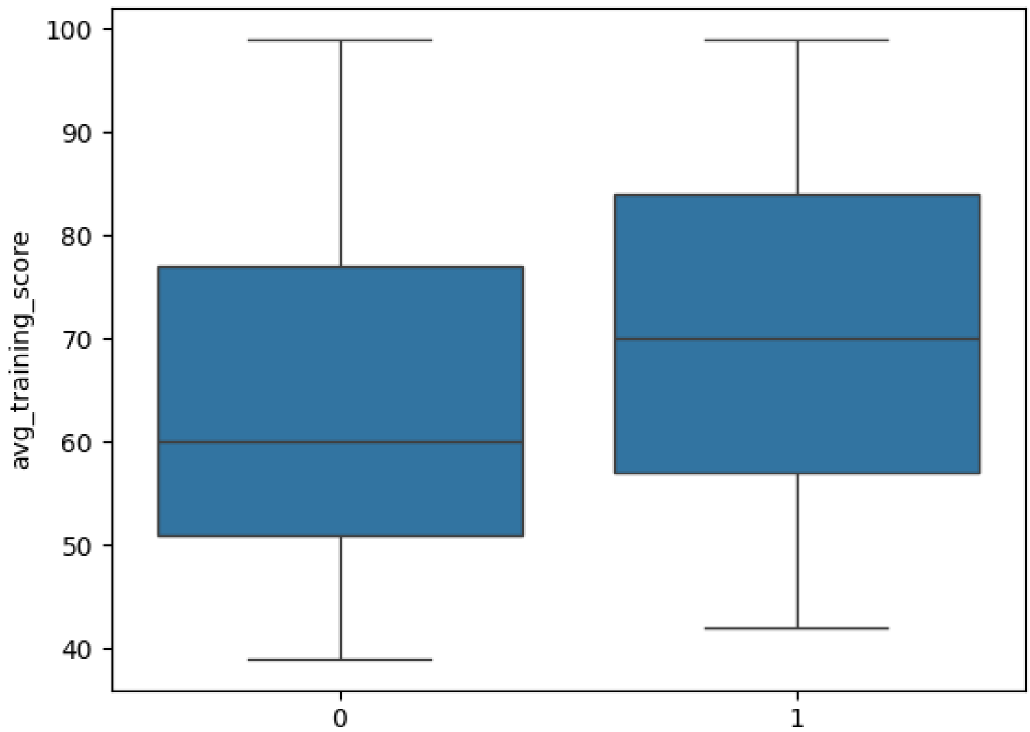


In [47]:

```
sns.boxplot(data=data, x="awards_won", y="avg_training_score")
```

Out[47]:

```
<Axes: xlabel='awards_won', ylabel='avg_training_score'>
```



awards_won

Let's see the attributes that have a strong correlation with each other

Correlation Heatmap

In [48]:

```
# plt.figure(figsize=(15, 7))
# sns.heatmap(data.corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral")
# plt.show()


# Create a new DataFrame that only includes numeric columns
numeric_data = data.select_dtypes(include=[np.number])

# Generate the correlation matrix heatmap with only numeric data
plt.figure(figsize=(15, 7))
sns.heatmap(numeric_data.corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral"
)
plt.show()
```

Data Preprocessing

In [49]:

```
data1 = data.copy()
```

Train-Test Split

In [50]:

```
X = data1.drop(["is_promoted"], axis=1)
y = data1["is_promoted"]
```

In [51]:

```
# Splitting data into training and validation set:
```

```
X_train, X_temp, y_train, y_temp = train_test_split(data.drop('is_promoted', axis=1), data['is_promoted'], test_size=0.20, random_state=42) ## Complete the code to split the data into train test in the ratio 80:20

X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp, test_size=0.25, random_state=42) ## Complete the code to split the data into train test in the ratio 75:25

print(X_train.shape, X_val.shape, X_test.shape)
```

```
(43846, 11) (2741, 11) (8221, 11)
```

Missing value imputation

In [52]:

```
# Defining the imputers for numerical and categorical variables
imputer_mode = SimpleImputer(strategy="most_frequent")
imputer_median = SimpleImputer(strategy="median")
```

In [53]:

```
# Fit and transform the train data
X_train[["education"]] = imputer_mode.fit_transform(X_train[["education"]])

# Transform the validation data
X_val[["education"]]  =  imputer_mode.transform(X_val[["education"]]) ## Complete the code to impute missing values in X_val

# Transform the test data
X_test[["education"]] = imputer_mode.transform(X_test[["education"]]) ## Complete the code to impute missing values in X_test
```

In [54]:

```
# Fit and transform the train data
X_train[["previous_year_rating", "avg_training_score"]] = imputer_median.fit_transform(
    X_train[["previous_year_rating", "avg_training_score"]]
)

# Transform the validation data
X_val[["previous_year_rating", "avg_training_score"]]  =  imputer_median.transform(X_val[["previous_year_rating", "avg_training_score"]]) ## Complete the code to impute missing values in X_val

# Transform the test data
X_test[["previous_year_rating", "avg_training_score"]] =  imputer_median.transform(X_test[["previous_year_rating", "avg_training_score"]]) ## Complete the code to impute missing values in X_test
```

In [55]:

```
# Checking that no column has missing values in train, validation and test sets
print(X_train.isna().sum())
print("-" * 30)
print(X_val.isna().sum())
print("-" * 30)
print(X_test.isna().sum())
```

```
department              0
region                  0
education               0
gender                  0
recruitment_channel     0
no_of_trainings         0
age                     0
previous_year_rating    0
length_of_service       0
awards_won              0
avg_training_score      0
```

```
dtype: int64
------------------------------
department              0
region                  0
education               0
gender                  0
recruitment_channel     0
no_of_trainings         0
age                     0
previous_year_rating    0
length_of_service       0
awards_won              0
avg_training_score      0
dtype: int64
------------------------------
department              0
region                  0
education               0
gender                  0
recruitment_channel     0
no_of_trainings         0
age                     0
previous_year_rating    0
length_of_service       0
awards_won              0
avg_training_score      0
dtype: int64
```

Encoding categorical variables

In [56]:

```python
# X_train = pd.get_dummies(X_train, drop_first=True)
# X_val = '_______'  ## Complete the code to impute missing values in X_val
# X_test = '_______'  ## Complete the code to impute missing values in X_val
# print(X_train.shape, X_val.shape, X_test.shape)
# print(X_train.shape, X_test.shape)

# Encoding categorical variables for the train, validation, and test data
X_train = pd.get_dummies(X_train, drop_first=True)
X_val = pd.get_dummies(X_val, drop_first=True)
X_test = pd.get_dummies(X_test, drop_first=True)

# Print the shapes of the datasets
print(X_train.shape, X_val.shape, X_test.shape)
```

```
(43846, 52) (2741, 52) (8221, 52)
```

Building the model

Model evaluation criterion

Model can make wrong predictions as:

- **Predicting an employee should get promoted when he/she should not get promoted**
- **Predicting an employee should not get promoted when he/she should get promoted**

Which case is more important?

- **Both cases are important here as not promoting a deserving employee might lead to less productivity and the company might lose a good employee which affects the company's growth. Further, giving promotion to a non-deserving employee would lead to loss of monetary resources and giving such employee higher responsibility might again affect the company's growth.**

How to reduce this loss i.e need to reduce False Negatives as well as False Positives?

- **Bank would want `F1-score` to be maximized, as both classes are important here. Hence, the focus should**

- Bank would want `F1-score` to be maximized, as both classes are important here. Hence, the focus should be on increasing the F1-score rather than focusing on just one metric i.e. Recall or Precision.

First, let's create two functions to calculate different metrics and confusion matrix, so that we don't have to use the same code repeatedly for each model.

In [57]:

```
# defining a function to compute different metrics to check performance of a classificati
on model built using sklearn
def model_performance_classification_sklearn(model, predictors, target):
    """
    Function to compute different metrics to check classification model performance

    model: classifier
    predictors: independent variables
    target: dependent variable
    """

    # predicting using the independent variables
    pred = model.predict(predictors)

    acc = accuracy_score(target, pred)  # to compute Accuracy
    recall = recall_score(target, pred)  # to compute Recall
    precision = precision_score(target, pred)  # to compute Precision
    f1 = f1_score(target, pred, average="macro")  # to compute F1-score

    # creating a dataframe of metrics
    df_perf = pd.DataFrame(
        {
            "Accuracy": acc,
            "Recall": recall,
            "Precision": precision,
            "F1": f1,
        },
        index=[0],
    )

    return df_perf
```

In [58]:

```
def confusion_matrix_sklearn(model, predictors, target):
    """
    To plot the confusion_matrix with percentages

    model: classifier
    predictors: independent variables
    target: dependent variable
    """
    y_pred = model.predict(predictors)
    cm = confusion_matrix(target, y_pred)
    labels = np.asarray(
        [
            ["{0:0.0f}".format(item) + "\n{0:.2%}".format(item / cm.flatten().sum())]
            for item in cm.flatten()
        ]
    ).reshape(2, 2)

    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=labels, fmt="")
    plt.ylabel("True label")
    plt.xlabel("Predicted label")
```

Model Building - Original Data

In [59]:

```
# models = []  # Empty list to store all the models
```

```
# # Appending models into the list
# models.append(("Bagging", BaggingClassifier(random_state=1)))
# '_______' ## Complete the code to append remaining 4 models in the list models

# results1 = []  # Empty list to store all model's CV scores
# names = []  # Empty list to store name of the models


# # loop through all models to get the mean cross validated score
# print("\n" "Cross-Validation Cost:" "\n")

# for name, model in models:
#     kfold = StratifiedKFold(
#         n_splits=5, shuffle=True, random_state=1
#     )  # Setting number of splits equal to 5
#     cv_result = cross_val_score(
#         estimator=model, X=X_train, y=y_train, scoring=scorer, cv=kfold
#     )
#     results1.append(cv_result)
#     names.append(name)
#     print("{}: {}".format(name, cv_result.mean()))

# print("\n" "Validation Performance:" "\n")

# for name, model in models:
#     model.fit(X_train, y_train)
#     scores = recall_score(y_val, model.predict(X_val))
#     print("{}: {}".format(name, scores))
```

In [60]:

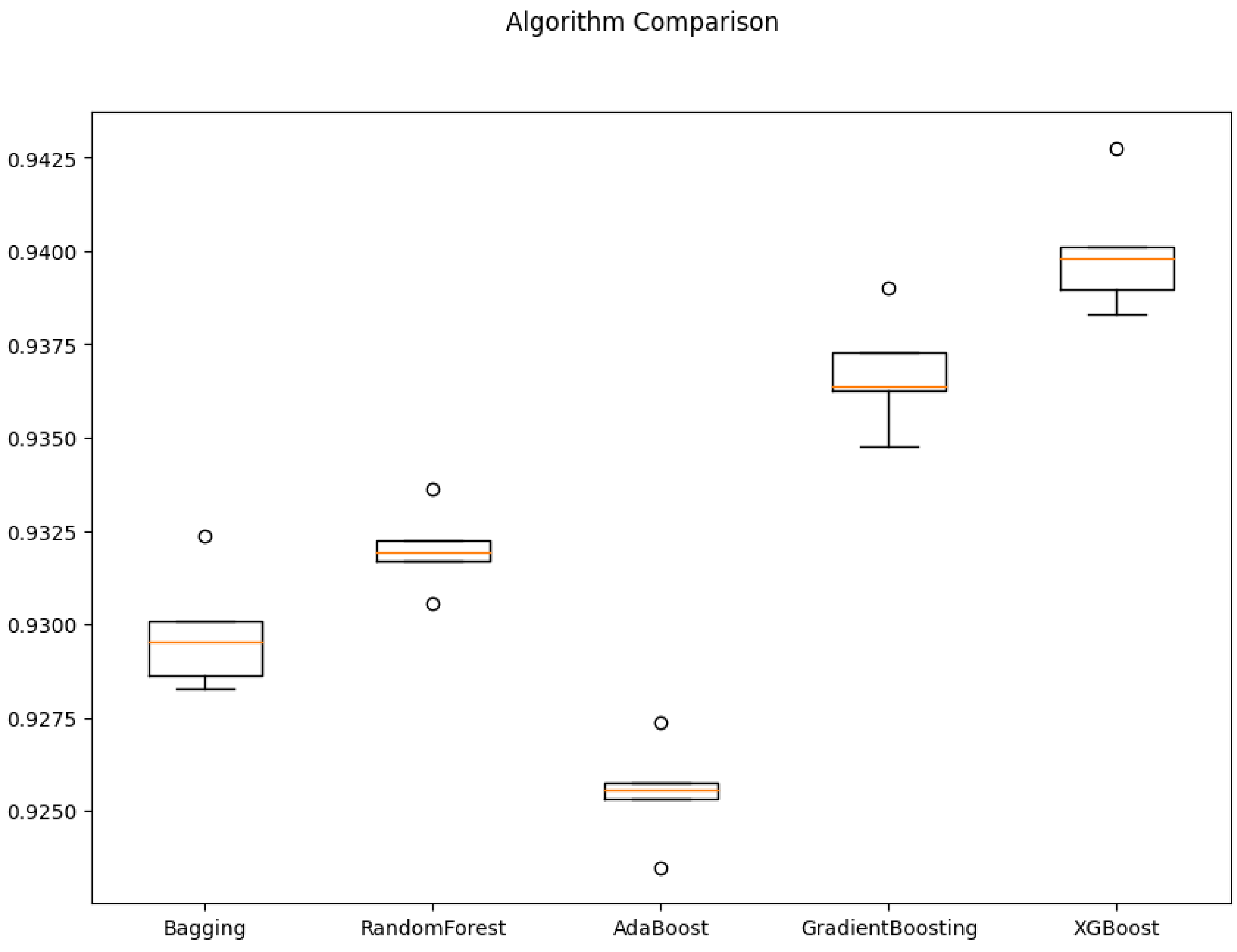
```
models = []  # Empty list to store all the models

# Appending models into the list
models.append(("Bagging", BaggingClassifier(random_state=1)))
models.append(("RandomForest", RandomForestClassifier(random_state=1)))
models.append(("AdaBoost", AdaBoostClassifier(random_state=1)))
models.append(("GradientBoosting", GradientBoostingClassifier(random_state=1)))
models.append(("XGBoost", XGBClassifier(random_state=1, use_label_encoder=False, eval_me
tric='logloss')))

results1 = []  # Empty list to store all model's CV scores
names = []  # Empty list to store name of the models

# Assuming 'scorer' is previously defined, for example:
scorer = 'accuracy'  # You might want to customize this based on your specific evaluation
needs

# loop through all models to get the mean cross validated score
print("\n" "Cross-Validation Cost:" "\n")

for name, model in models:
    kfold = StratifiedKFold(
        n_splits=5, shuffle=True, random_state=1
    )  # Setting number of splits equal to 5
    cv_result = cross_val_score(
        estimator=model, X=X_train, y=y_train, scoring=scorer, cv=kfold
    )
    results1.append(cv_result)
    names.append(name)
    print("{}: {}".format(name, cv_result.mean()))

print("\n" "Validation Performance:" "\n")

for name, model in models:
    model.fit(X_train, y_train)
    scores = recall_score(y_val, model.predict(X_val))
    print("{}: {}".format(name, scores))
```

```
Cross-Validation Cost:
```

```
Bagging: 0.92977688871450728
RandomForest: 0.9320120050769705
AdaBoost: 0.925489211047573
GradientBoosting: 0.9367330597199397
XGBoost: 0.9399944632362397

Validation Performance:

Bagging: 0.34051724137931033
RandomForest: 0.28448275862068967
AdaBoost: 0.19827586206896552
GradientBoosting: 0.28448275862068967
XGBoost: 0.34051724137931033
```

In [61]:

```
# Plotting boxplots for CV scores of all models defined above
fig = plt.figure(figsize=(10, 7))

fig.suptitle("Algorithm Comparison")
ax = fig.add_subplot(111)

plt.boxplot(results1)
ax.set_xticklabels(names)

plt.show()
```



Model Building - Oversampled Data

In [62]:

```
print("Before Oversampling, counts of label 'Yes': {}".format(sum(y_train == 1)))
print("Before Oversampling, counts of label 'No': {} \n".format(sum(y_train == 0)))
```

```
sm = SMOTE(
    sampling_strategy=1, k_neighbors=5, random_state=1
)  # Synthetic Minority Over Sampling Technique
X_train_over, y_train_over = sm.fit_resample(X_train, y_train)


print("After Oversampling, counts of label 'Yes': {}".format(sum(y_train_over == 1)))
print("After Oversampling, counts of label 'No': {} \n".format(sum(y_train_over == 0)))


print("After Oversampling, the shape of train_X: {}".format(X_train_over.shape))
print("After Oversampling, the shape of train_y: {} \n".format(y_train_over.shape))
```

```
Before Oversampling, counts of label 'Yes': 3760
Before Oversampling, counts of label 'No': 40086

After Oversampling, counts of label 'Yes': 40086
After Oversampling, counts of label 'No': 40086

After Oversampling, the shape of train_X: (80172, 52)
After Oversampling, the shape of train_y: (80172,)
```

In [63]:

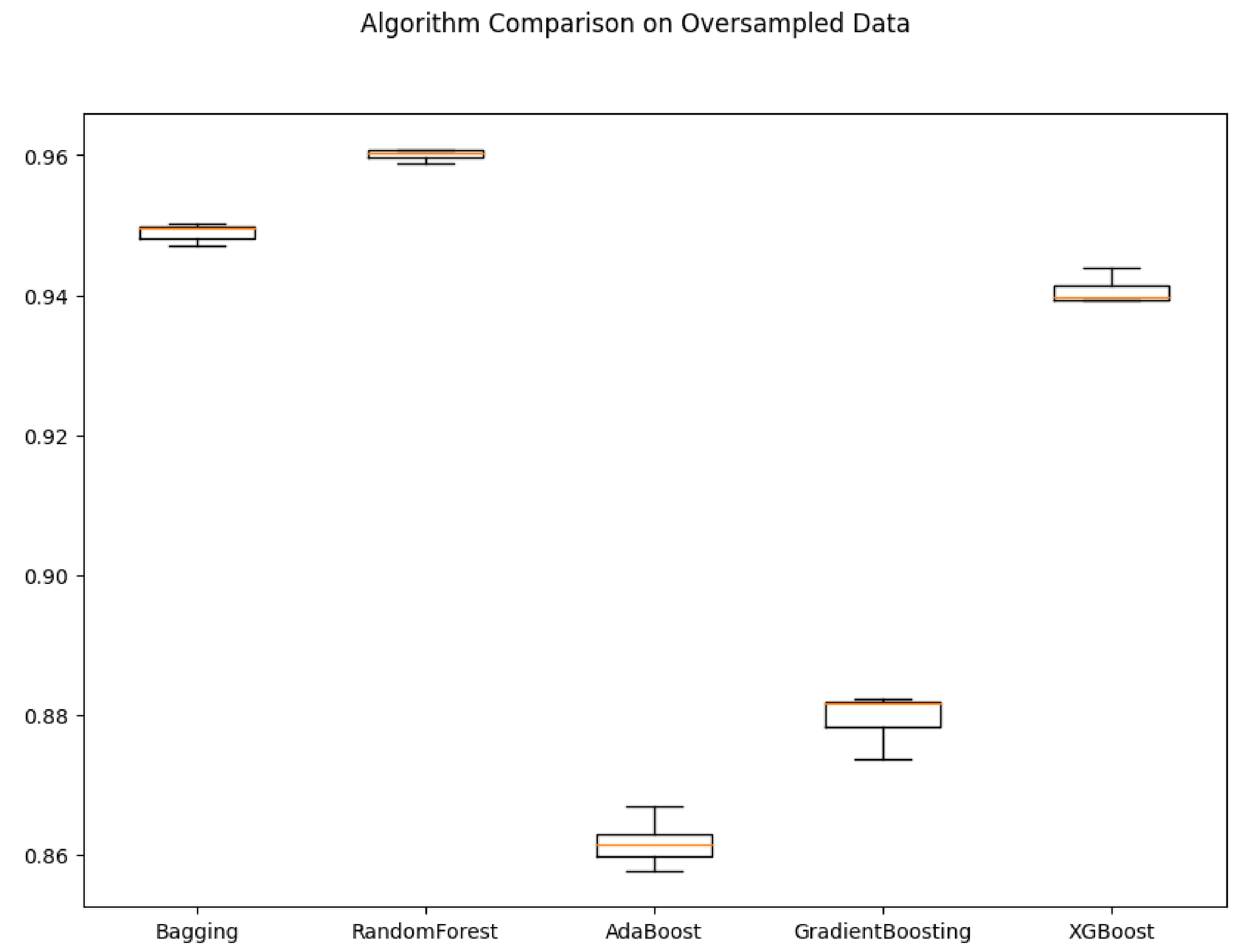
```
# '_______' ## Complete the code to build models on oversampled data
# ## Note - Take reference from the original models built above


# Assuming 'models' list contains previously initialized models

print("\n" "Validation Performance on Oversampled Data:" "\n")

# Loop to fit each model on the oversampled data and evaluate it
for name, model in models:
    model.fit(X_train_over, y_train_over)  # Fit model on oversampled data
    scores = recall_score(y_val, model.predict(X_val))  # Evaluating using the validatio
n set
    print("{}: {}".format(name, scores))
```

```
Validation Performance on Oversampled Data:

Bagging: 0.2974137931034483
RandomForest: 0.29310344827586204
AdaBoost: 0.4482758620689655
GradientBoosting: 0.4827586206896552
XGBoost: 0.375
```

In [64]:

```
# # Plotting boxplots for CV scores of all models defined above
# '_______' ## Write the code to create boxplot to check model performance on oversampled
data


results1_oversampled = []  # Empty list to store all model's CV scores for oversampled da
ta
names_oversampled = []  # Empty list to store the names of the models

# Assuming you have a models list set up as follows
models = [
    ("Bagging", BaggingClassifier(random_state=1)),
    ("RandomForest", RandomForestClassifier(random_state=1)),
    ("AdaBoost", AdaBoostClassifier(random_state=1)),
    ("GradientBoosting", GradientBoostingClassifier(random_state=1)),
    ("XGBoost", XGBClassifier(random_state=1, use_label_encoder=False, eval_metric='logl
oss'))
]

for name, model in models:
    kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)
```

```
    cv_results = cross_val_score(model, X_train_over, y_train_over, cv=kfold, scoring='a
ccuracy')   # Adjust scoring method as needed
    results1_oversampled.append(cv_results)
    names_oversampled.append(name)

# Now plot the boxplot
fig = plt.figure(figsize=(10, 7))
fig.suptitle("Algorithm Comparison on Oversampled Data")
ax = fig.add_subplot(111)

plt.boxplot(results1_oversampled)
ax.set_xticklabels(names_oversampled)

plt.show()
```



Model Building - Undersampled Data

In [65]:

```
rus = RandomUnderSampler(random_state=1)
X_train_un, y_train_un = rus.fit_resample(X_train, y_train)
```

In [66]:

```
print("Before Under Sampling, counts of label 'Yes': {}".format(sum(y_train == 1)))
print("Before Under Sampling, counts of label 'No': {} \n".format(sum(y_train == 0)))

print("After Under Sampling, counts of label 'Yes': {}".format(sum(y_train_un == 1)))
print("After Under Sampling, counts of label 'No': {} \n".format(sum(y_train_un == 0)))

print("After Under Sampling, the shape of train_X: {}".format(X_train_un.shape))
print("After Under Sampling, the shape of train_y: {} \n".format(y_train_un.shape))
```

```
Before Under Sampling, counts of label 'Yes': 3760
```

```
Before Under Sampling, counts of label 'Yes': 3760
Before Under Sampling, counts of label 'No': 40086

After Under Sampling, counts of label 'Yes': 3760
After Under Sampling, counts of label 'No': 3760

After Under Sampling, the shape of train_X: (7520, 52)
After Under Sampling, the shape of train_y: (7520,)
```

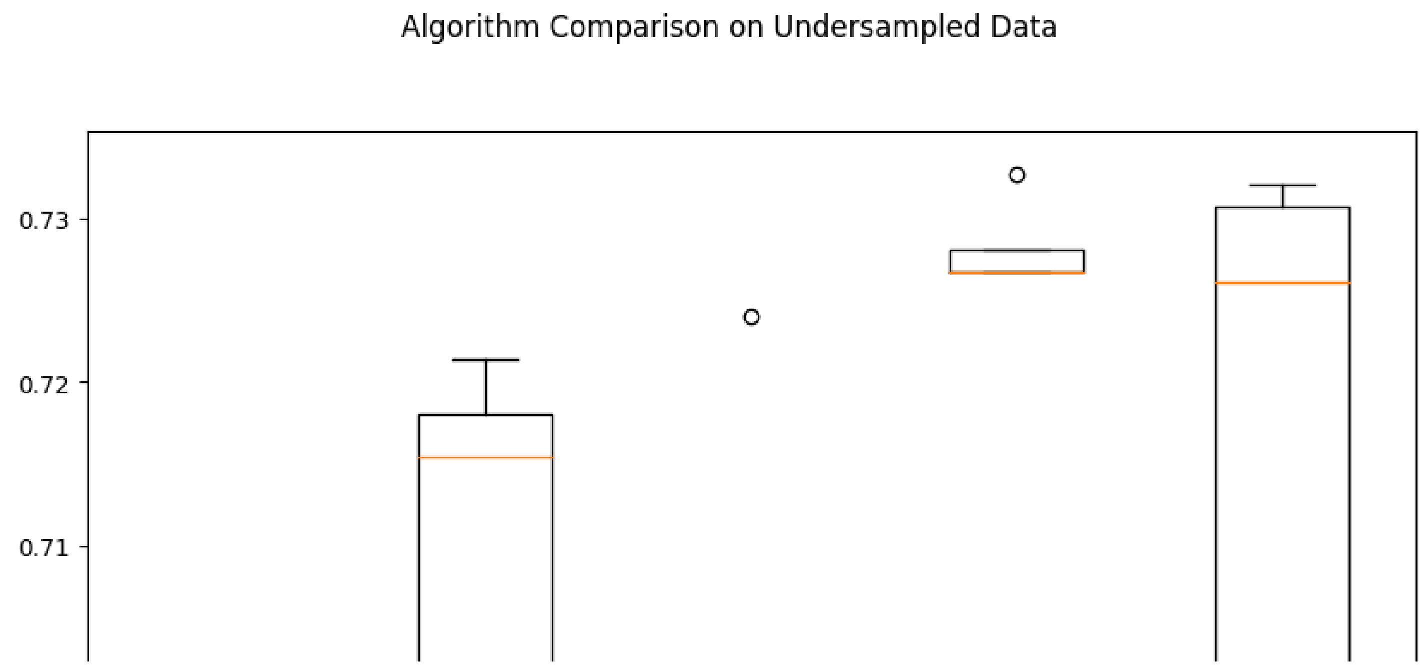
In [67]:

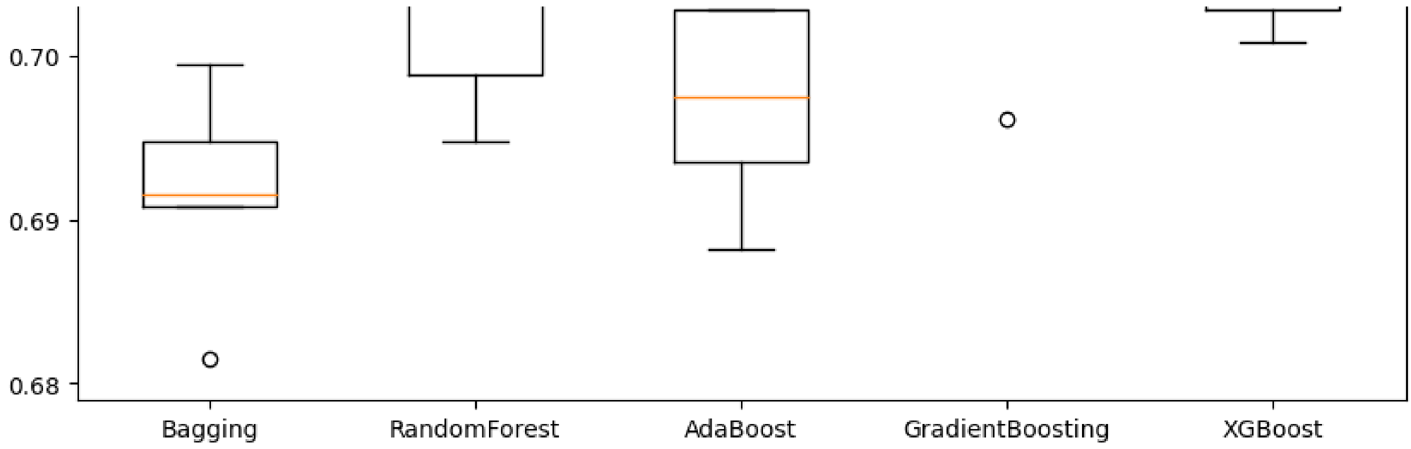
```
'_______' ## Complete the code to build models on undersampled data
## Note - Take reference from the original models built above

# Loop to fit each model on the undersampled data
for name, model in models:
    model.fit(X_train_un, y_train_un)  # Fit model on the undersampled training data
    print(f"{name} has been trained on the undersampled data.")
```

```
Bagging has been trained on the undersampled data.
RandomForest has been trained on the undersampled data.
AdaBoost has been trained on the undersampled data.
GradientBoosting has been trained on the undersampled data.
XGBoost has been trained on the undersampled data.
```

In [68]:

```
# Plotting boxplots for CV scores of all models defined above
# '_______' ## Write the code to create boxplot to check model performance on undersample
d data

# Assuming 'models' list exists and 'X_train_un', 'y_train_un' are the undersampled data

results_un = []  # List to store cross-validation results for undersampled data
names_un = []  # List to store the names of the models

for name, model in models:
    cv_results = cross_val_score(model, X_train_un, y_train_un, cv=StratifiedKFold(n_spl
its=5), scoring='accuracy')
    results_un.append(cv_results)
    names_un.append(name)  # Storing names for x-axis labels in the plot

# Plotting the boxplots
fig = plt.figure(figsize=(10, 7))
fig.suptitle("Algorithm Comparison on Undersampled Data")
ax = fig.add_subplot(111)
plt.boxplot(results_un)
ax.set_xticklabels(names_un)
plt.show()
```





Hyperparameter Tuning

Note

1. **Sample parameter grid has been provided to do necessary hyperparameter tuning. One can extend/reduce the parameter grid based on execution time and system configuration to try to improve the model performance further wherever needed.**
2. **The models chosen in this notebook are based on test runs. One can update the best models as obtained upon code execution and tune them for best performance.**

Tuning AdaBoost using Undersampled data

In [69]:

```
%%time

# defining model
Model = AdaBoostClassifier(random_state=1)

# Parameter grid to pass in RandomSearchCV
param_grid = {
    "n_estimators": np.arange(10, 110, 10),
    "learning_rate": [0.1, 0.01, 0.2, 0.05, 1],
    "base_estimator": [
        DecisionTreeClassifier(max_depth=1, random_state=1),
        DecisionTreeClassifier(max_depth=2, random_state=1),
        DecisionTreeClassifier(max_depth=3, random_state=1),
    ],
}

# Type of scoring used to compare parameter combinations
scorer = 'f1_macro'

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=Model, param_distributions=param_grid, n_job
s = -1, n_iter=50, scoring=scorer, cv=5, random_state=1)

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train_un, y_train_un)  # This is the line that was needed to complete
the code
 ## Complete the code to fit the model on undersampled data

print("Best parameters are {} with CV score={}:" .format(randomized_cv.best_params_,rand
omized_cv.best_score_))
```

```
Best parameters are {'n_estimators': 50, 'learning_rate': 1, 'base_estimator': DecisionTr
eeClassifier(max_depth=2, random_state=1)} with CV score=0.7194193297216376:
CPU times: total: 2.86 s
Wall time: 1min 3s
```

In [70]:

```
# # Creating new pipeline with best parameters
# tuned_adb1 = AdaBoostClassifier( random_state=___,
#     n_estimators= _______, learning_rate= _______, base_estimator= DecisionTreeClassifi
er(max_depth=_______, random_state=1)
# ) ## Complete the code with the best parameters obtained from tuning

# tuned_adb1.'_______' ## Complete the code to fit the model on undersampled data

# Creating new pipeline with best parameters
tuned_adb1 = AdaBoostClassifier(
    random_state=1,
    n_estimators=50,  # Replace with the best n_estimators found
    learning_rate=0.1,  # Replace with the best learning_rate found
    base_estimator=DecisionTreeClassifier(max_depth=2, random_state=1)  # Replace with t
he best max_depth found
)

# Fitting the model on undersampled data
tuned_adb1.fit(X_train_un, y_train_un)
```

Out[70]:

```
▸ AdaBoostClassifier
▸ base_estimator: DecisionTreeClassifier
▸ DecisionTreeClassifier
```

In [71]:

```
# adb1_train = '_______' ## Complete the code to check the performance on training set
# adb1_train


from sklearn.metrics import accuracy_score, classification_report

# Predict on the training set
adb1_train_predictions = tuned_adb1.predict(X_train_un)

# Evaluate the predictions using accuracy as an example metric
adb1_train_accuracy = accuracy_score(y_train_un, adb1_train_predictions)

# You can also use classification_report to get a detailed performance report
adb1_train_report = classification_report(y_train_un, adb1_train_predictions)

# Store the accuracy in adb1_train and print it
adb1_train = adb1_train_accuracy

# Print the accuracy and the detailed report
print("Accuracy on training set:", adb1_train)
print("Detailed classification report:\n", adb1_train_report)
```

```
Accuracy on training set: 0.7122340425531914
Detailed classification report:
               precision    recall  f1-score   support

           0       0.67      0.85      0.75      3760
           1       0.79      0.57      0.67      3760

    accuracy                           0.71      7520
   macro avg       0.73      0.71      0.71      7520
weighted avg       0.73      0.71      0.71      7520
```

In [72]:

```
# # Checking model's performance on validation set
# adb1_val =  '_______' ## Complete the code to check the performance on validation set
# adb1_val
```

```
from sklearn.metrics import accuracy_score, classification_report

# Predict on the validation set
adb1_val_predictions = tuned_adb1.predict(X_val)

# Evaluate the predictions using accuracy as an example metric
adb1_val_accuracy = accuracy_score(y_val, adb1_val_predictions)

# You can also use classification_report to get a detailed performance report
adb1_val_report = classification_report(y_val, adb1_val_predictions)

# Store the accuracy in adb1_val and print it
adb1_val = adb1_val_accuracy

# Print the accuracy and the detailed report
print("Accuracy on validation set:", adb1_val)
print("Detailed classification report:\n", adb1_val_report)
```

```
Accuracy on validation set: 0.8314483765049252
Detailed classification report:
               precision    recall  f1-score   support

           0       0.96      0.85      0.90      2509
           1       0.28      0.62      0.38       232

    accuracy                           0.83      2741
   macro avg       0.62      0.73      0.64      2741
weighted avg       0.90      0.83      0.86      2741
```

Tuning AdaBoost using original data

In [73]:

```
%%time

# defining model
Model = AdaBoostClassifier(random_state=1)

# Parameter grid to pass in RandomSearchCV
param_grid = {
    "n_estimators": np.arange(10, 110, 10),
    "learning_rate": [0.1, 0.01, 0.2, 0.05, 1],
    "base_estimator": [
        DecisionTreeClassifier(max_depth=1, random_state=1),
        DecisionTreeClassifier(max_depth=2, random_state=1),
        DecisionTreeClassifier(max_depth=3, random_state=1),
    ],
}

# Type of scoring used to compare parameter combinations
scorer = 'f1_macro'

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=Model, param_distributions=param_grid, n_job
s = -1, n_iter=50, scoring=scorer, cv=5, random_state=1)

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train, y_train)#'_______' ## Complete the code to fit the model on or
iginal data

print("Best parameters are {} with CV score={}:" .format(randomized_cv.best_params_,rand
omized_cv.best_score_))
```

```
Best parameters are {'n_estimators': 90, 'learning_rate': 1, 'base_estimator': DecisionTr
eeClassifier(max_depth=2, random_state=1)} with CV score=0.7223671392095827:
CPU times: total: 13.2 s
Wall time: 5min 17s
```

In [74]:

```python
# # Creating new pipeline with best parameters
# tuned_adb2 = AdaBoostClassifier( random_state=___,
#     n_estimators= _______, learning_rate= _______, base_estimator= DecisionTreeClassifi
er(max_depth=_______, random_state=1)
# ) ## Complete the code with the best parameters obtained from tuning

# tuned_adb2.'_______' ## Complete the code to fit the model on original data


# Creating new pipeline with best parameters
tuned_adb2 = AdaBoostClassifier(
    random_state=1,  # Use the random state from the tuning process
    n_estimators=randomized_cv.best_params_['n_estimators'],  # Use the best number of es
timators
    learning_rate=randomized_cv.best_params_['learning_rate'],  # Use the best learning r
ate
    base_estimator=DecisionTreeClassifier(
        max_depth=randomized_cv.best_params_['base_estimator'].max_depth,
        random_state=1
    )
)

# Fitting the model on the original data
tuned_adb2.fit(X_train, y_train)
```

Out[74]:

```
▸          AdaBoostClassifier
▸ base_estimator: DecisionTreeClassifier
 ▸        DecisionTreeClassifier
```

In [75]:

```python
# adb2_train = '_______' ## Complete the code to check the performance on training set
# adb2_train

from sklearn.metrics import accuracy_score, classification_report

# Predict on the training set
adb2_train_predictions = tuned_adb2.predict(X_train)

# Evaluate the predictions using accuracy as an example metric
adb2_train_accuracy = accuracy_score(y_train, adb2_train_predictions)

# You can also use classification_report to get a detailed performance report
adb2_train_report = classification_report(y_train, adb2_train_predictions)

# Store the accuracy in adb2_train and print it
adb2_train = adb2_train_accuracy

# Print the accuracy and the detailed report
print("Accuracy on training set:", adb2_train)
print("Detailed classification report:\n", adb2_train_report)
```

```
Accuracy on training set: 0.939652419833052
Detailed classification report:
               precision    recall  f1-score   support

           0       0.94      1.00      0.97     40086
           1       0.91      0.33      0.48      3760

    accuracy                           0.94     43846
   macro avg       0.92      0.66      0.73     43846
weighted avg       0.94      0.94      0.93     43846
```

In [76]:

```
# # Checking model's performance on validation set
# adb2_val =  '_______' ## Complete the code to check the performance on validation set
# adb2_val

from sklearn.metrics import accuracy_score, classification_report

# Predict on the validation set
adb2_val_predictions = tuned_adb2.predict(X_val)

# Evaluate the predictions using accuracy as an example metric
adb2_val_accuracy = accuracy_score(y_val, adb2_val_predictions)

# You can also use classification_report to get a detailed performance report
adb2_val_report = classification_report(y_val, adb2_val_predictions)

# Store the accuracy in adb2_val and print it
adb2_val = adb2_val_accuracy

# Print the accuracy and the detailed report
print("Accuracy on validation set:", adb2_val)
print("Detailed classification report:\n", adb2_val_report)
```

```
Accuracy on validation set: 0.9398029916089019
Detailed classification report:
               precision    recall  f1-score   support

           0       0.94      1.00      0.97      2509
           1       0.89      0.33      0.48       232

    accuracy                           0.94      2741
   macro avg       0.92      0.66      0.72      2741
weighted avg       0.94      0.94      0.93      2741
```

Tuning Gradient Boosting using undersampled data

In [77]:

```
%%time

#Creating pipeline
Model = GradientBoostingClassifier(random_state=1)

#Parameter grid to pass in RandomSearchCV
param_grid = {
    "init": [AdaBoostClassifier(random_state=1),DecisionTreeClassifier(random_state=1)],
    "n_estimators": np.arange(75,150,25),
    "learning_rate": [0.1, 0.01, 0.2, 0.05, 1],
    "subsample":[0.5,0.7,1],
    "max_features":[0.5,0.7,1],
}

# Type of scoring used to compare parameter combinations
scorer = 'f1_macro'

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=Model, param_distributions=param_grid, n_ite
r=50, scoring=scorer, cv=5, random_state=1, n_jobs = -1)

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train_un, y_train_un) ## Complete the code to fit the model on under
sampled data

print("Best parameters are {} with CV score={}:" .format(randomized_cv.best_params_,rand
omized_cv.best_score_))
```

```
Best parameters are {'subsample': 0.7, 'n_estimators': 125, 'max_features': 0.7, 'learnin
g_rate': 0.2, 'init': AdaBoostClassifier(random_state=1)} with CV score=0.728005306035405
7:
CPU times: total: 2.78 s
Wall time: 47.6 s
```

In [78]:

```
# Creating new pipeline with best parameters
# tuned_gbm1 = GradientBoostingClassifier(
#     max_features=_______,
#     init=AdaBoostClassifier(random_state=1),
#     random_state=1,
#     learning_rate=_______,
#     n_estimators=_______,
#     subsample=_______,
# )## Complete the code with the best parameters obtained from tuning

# tuned_gbm1.fit(X_train_un, y_train_un)

# Replace the placeholders with the best parameters obtained from the RandomizedSearchCV
best_params = randomized_cv.best_params_

tuned_gbm1 = GradientBoostingClassifier(
    max_features=best_params['max_features'],
    init=AdaBoostClassifier(random_state=1),  # Assuming the best `init` was AdaBoostClas
sifier
    random_state=1,
    learning_rate=best_params['learning_rate'],
    n_estimators=best_params['n_estimators'],
    subsample=best_params['subsample']
)

# Fitting the model on undersampled data
tuned_gbm1.fit(X_train_un, y_train_un)

# Verify the model is fitted
print("Model is trained with the best parameters from hyperparameter tuning on the unders
ampled data.")
```

```
Model is trained with the best parameters from hyperparameter tuning on the undersampled
data.
```

In [79]:

```
# gbm1_train = '_______' ## Complete the code to check the performance on oversampled tra
in set
# gbm1_train


from sklearn.metrics import accuracy_score, classification_report

# Predict on the oversampled training set
gbm1_train_predictions = tuned_gbm1.predict(X_train_over)

# Evaluate the predictions using accuracy as an example metric
gbm1_train_accuracy = accuracy_score(y_train_over, gbm1_train_predictions)

# You can also use classification_report to get a detailed performance report
gbm1_train_report = classification_report(y_train_over, gbm1_train_predictions)

# Store the accuracy in gbm1_train and print it
gbm1_train = gbm1_train_accuracy

# Print the accuracy and the detailed report
print("Accuracy on oversampled training set:", gbm1_train)
print("Detailed classification report:\n", gbm1_train_report)
```

```
Accuracy on oversampled training set: 0.7931447388115551
Detailed classification report:
               precision    recall  f1-score   support

           0       0.79      0.79      0.79     40086
           1       0.79      0.79      0.79     40086

    accuracy                           0.79     80172
   macro avg       0.79      0.79      0.79     80172
```

```
weighted avg       0.79      0.79      0.79     80172
```

In [80]:

```
# gbm1_val = '_______' ## Complete the code to check the performance on validation set
# gbm1_val


from sklearn.metrics import accuracy_score, classification_report

# Predict on the validation set
gbm1_val_predictions = tuned_gbm1.predict(X_val)

# Evaluate the predictions using accuracy as an example metric
gbm1_val_accuracy = accuracy_score(y_val, gbm1_val_predictions)

# You can also use classification_report to get a detailed performance report
gbm1_val_report = classification_report(y_val, gbm1_val_predictions)

# Store the accuracy in gbm1_val and print it
gbm1_val = gbm1_val_accuracy

# Print the accuracy and the detailed report
print("Accuracy on validation set:", gbm1_val)
print("Detailed classification report:\n", gbm1_val_report)
```

```
Accuracy on validation set: 0.7697920466982853
Detailed classification report:
               precision    recall  f1-score   support

           0       0.97      0.77      0.86      2509
           1       0.23      0.72      0.35       232

    accuracy                           0.77      2741
   macro avg       0.60      0.75      0.60      2741
weighted avg       0.91      0.77      0.82      2741
```

Tuning Gradient Boosting using original data

In [81]:

```
%%time

#defining model
Model = GradientBoostingClassifier(random_state=1)

#Parameter grid to pass in RandomSearchCV
param_grid = {
    "init": [AdaBoostClassifier(random_state=1),DecisionTreeClassifier(random_state=1)],
    "n_estimators": np.arange(75,150,25),
    "learning_rate": [0.1, 0.01, 0.2, 0.05, 1],
    "subsample":[0.5,0.7,1],
    "max_features":[0.5,0.7,1],
}

# Type of scoring used to compare parameter combinations
scorer = 'f1_macro'

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=Model, param_distributions=param_grid, n_ite
r=50, scoring=scorer, cv=5, random_state=1, n_jobs = -1)

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train, y_train) ## Complete the code to fit the model on original dat
a

print("Best parameters are {} with CV score={}:" .format(randomized_cv.best_params_,rand
omized_cv.best_score_))
```

```
Best parameters are {'subsample': 0.7, 'n_estimators': 125, 'max_features': 0.7, 'learnin
g_rate': 0.2, 'init': AdaBoostClassifier(random_state=1)} with CV score=0.721238400832193
7:
CPU times: total: 11.9 s
Wall time: 4min 34s
```

In [82]:

```
# # Creating new pipeline with best parameters
# tuned_gbm2 = GradientBoostingClassifier(
#     max_features=_______,
#     init=AdaBoostClassifier(random_state=1),
#     random_state=1,
#     learning_rate=_______,
#     n_estimators=_______,
#     subsample=_______,
# )## Complete the code with the best parameters obtained from tuning

# tuned_gbm2.fit(X_train, y_train)

# Assuming that randomized_cv is the RandomizedSearchCV object and its best_params_ attri
bute contains the optimal values.
best_params = randomized_cv.best_params_

tuned_gbm2 = GradientBoostingClassifier(
    max_features=best_params['max_features'],
    init=AdaBoostClassifier(random_state=1),  # Assuming the best `init` was AdaBoostClas
sifier; replace if necessary.
    random_state=1,
    learning_rate=best_params['learning_rate'],
    n_estimators=best_params['n_estimators'],
    subsample=best_params['subsample']
)

# Fitting the model on original data
tuned_gbm2.fit(X_train, y_train)

# Verify the model is fitted
print("Model is trained with the best parameters from hyperparameter tuning on the origin
al data.")
```

```
Model is trained with the best parameters from hyperparameter tuning on the original data
.
```

In [83]:

```
# gbm2_train = '_______' ## Complete the code to check the performance on original data
# gbm2_train


from sklearn.metrics import accuracy_score, classification_report

# Predict on the original training set
gbm2_train_predictions = tuned_gbm2.predict(X_train)

# Evaluate the predictions using accuracy as an example metric
gbm2_train_accuracy = accuracy_score(y_train, gbm2_train_predictions)

# You can also use classification_report to get a detailed performance report
gbm2_train_report = classification_report(y_train, gbm2_train_predictions)

# Store the accuracy in gbm2_train and print it
gbm2_train = gbm2_train_accuracy

# Print the accuracy and the detailed report
print("Accuracy on training set:", gbm2_train)
print("Detailed classification report:\n", gbm2_train_report)
```

```
Accuracy on training set: 0.9409980385896091
Detailed classification report:
              precision    recall  f1-score   support
```

```
           0       0.94      1.00      0.97     40086
           1       0.96      0.33      0.49      3760

    accuracy                           0.94     43846
   macro avg       0.95      0.66      0.73     43846
weighted avg       0.94      0.94      0.93     43846
```

In [84]:

```
# gbm2_val = '_______' ## Complete the code to check the performance on validation set
# gbm2_val\


from sklearn.metrics import accuracy_score, classification_report

# Predict on the validation set
gbm2_val_predictions = tuned_gbm2.predict(X_val)

# Evaluate the predictions using accuracy as an example metric
gbm2_val_accuracy = accuracy_score(y_val, gbm2_val_predictions)

# You can also use classification_report to get a detailed performance report
gbm2_val_report = classification_report(y_val, gbm2_val_predictions)

# Store the accuracy in gbm2_val and print it
gbm2_val = gbm2_val_accuracy

# Print the accuracy and the detailed report
print("Accuracy on validation set:", gbm2_val)
print("Detailed classification report:\n", gbm2_val_report)
```

```
Accuracy on validation set: 0.9398029916089019
Detailed classification report:
               precision    recall  f1-score   support

           0       0.94      1.00      0.97      2509
           1       0.89      0.33      0.48       232

    accuracy                           0.94      2741
   macro avg       0.91      0.66      0.73      2741
weighted avg       0.94      0.94      0.93      2741
```

Model Comparison and Final Model Selection

In [85]:

```
# training performance comparison

# models_train_comp_df = pd.concat(
#     [
#         gbm1_train.T,
#         gbm2_train.T,
#         adb1_train.T,
#         adb2_train.T,
#     ],
#     axis=1,
# )
# models_train_comp_df.columns = [
#     "Gradient boosting trained with Undersampled data",
#     "Gradient boosting trained with Original data",
#     "AdaBoost trained with Undersampled data",
#     "AdaBoost trained with Original data",
# ]
# print("Training performance comparison:")
# models_train_comp_df


import pandas as pd
```

```
# Assuming gbm1_train, gbm2_train, adb1_train, and adb2_train are scalar values like accu
racy
models_train_comp_df = pd.DataFrame(
    {
        "Gradient boosting trained with Undersampled data": [gbm1_train],
        "Gradient boosting trained with Original data": [gbm2_train],
        "AdaBoost trained with Undersampled data": [adb1_train],
        "AdaBoost trained with Original data": [adb2_train],
    },
    index=["Training Performance"]
)

print("Training performance comparison:")
print(models_train_comp_df)
```

```
Training performance comparison:
                      Gradient boosting trained with Undersampled data  \
Training Performance                                             0.793

                      Gradient boosting trained with Original data  \
Training Performance                                         0.941

                      AdaBoost trained with Undersampled data  \
Training Performance                                    0.712

                      AdaBoost trained with Original data
Training Performance                                0.940
```

In [86]:

```
# validation performance comparison

#'_______' ## Write the code to compare the performance on validation set


# Assuming gbm1_val, gbm2_val, adb1_val, and adb2_val are also scalar values
models_val_comp_df = pd.DataFrame(
    {
        "Gradient boosting trained with Undersampled data": [gbm1_val],
        "Gradient boosting trained with Original data": [gbm2_val],
        "AdaBoost trained with Undersampled data": [adb1_val],
        "AdaBoost trained with Original data": [adb2_val],
    },
    index=["Validation Performance"]
)

print("Validation performance comparison:")
print(models_val_comp_df)
```

```
Validation performance comparison:
                        Gradient boosting trained with Undersampled data  \
Validation Performance                                             0.770

                        Gradient boosting trained with Original data  \
Validation Performance                                         0.940

                        AdaBoost trained with Undersampled data  \
Validation Performance                                    0.831

                        AdaBoost trained with Original data
Validation Performance                                0.940
```

Now we have our final model, so let's find out how our final model is performing on unseen test data.

In [87]:

```
# Let's check the performance on test set
#'_______' ## Write the code to check the performance of best model on test data
```

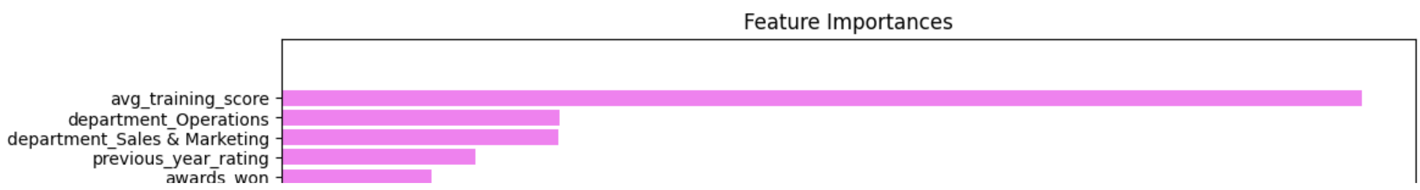
```
from sklearn.metrics import accuracy_score, classification_report

# Assuming 'tuned_gbm2' is the Gradient Boosting model trained with the original data
# Predict on the test set
test_predictions = tuned_gbm2.predict(X_test)

# Evaluate the predictions using accuracy
test_accuracy = accuracy_score(y_test, test_predictions)

# Detailed performance report
test_report = classification_report(y_test, test_predictions)

print("Accuracy on test set:", test_accuracy)
print("Detailed classification report:\n", test_report)
```

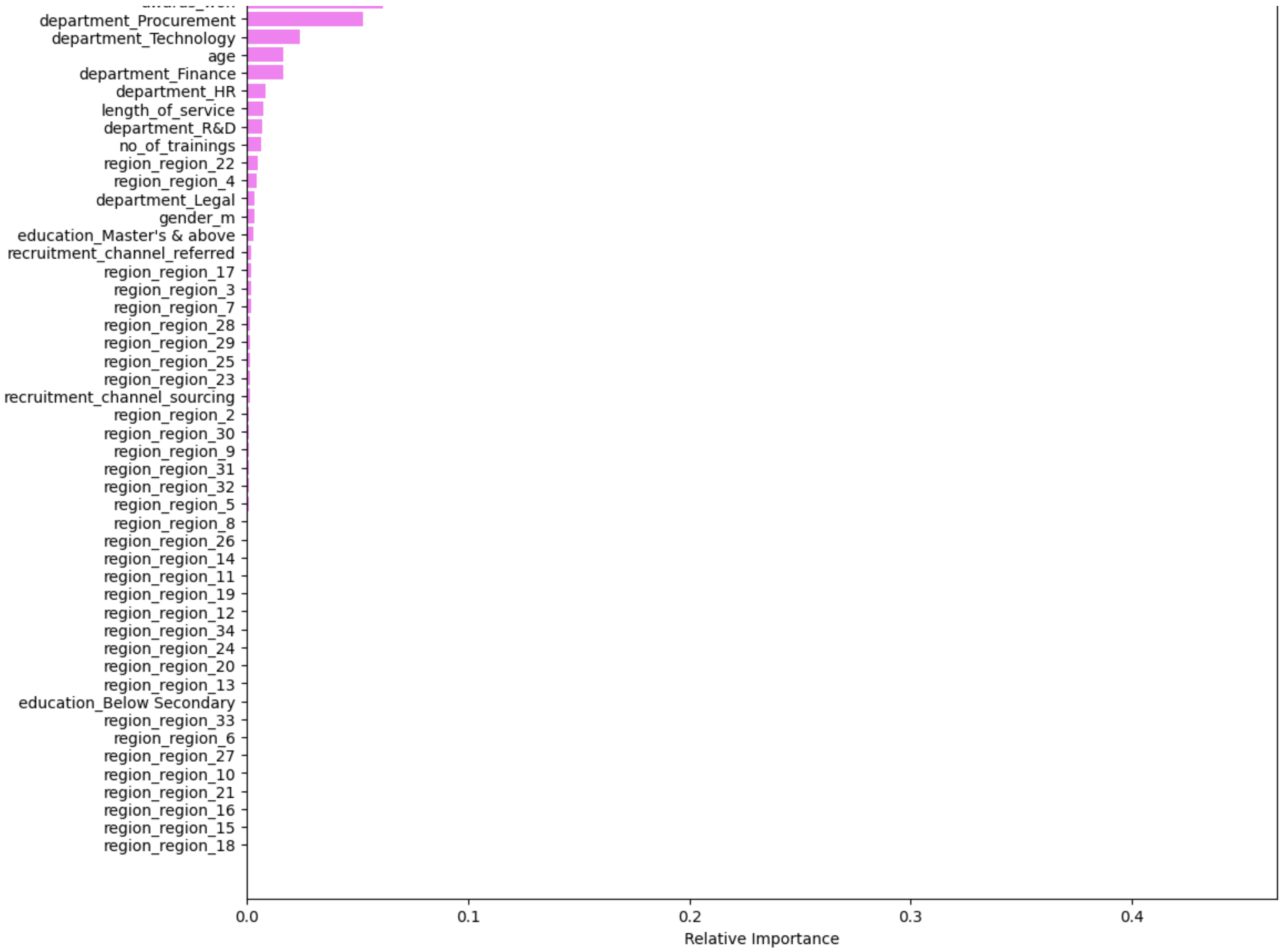
```
Accuracy on test set: 0.9400316263228318
Detailed classification report:
               precision    recall  f1-score   support

           0       0.94      1.00      0.97      7545
           1       0.91      0.30      0.45       676

    accuracy                           0.94      8221
   macro avg       0.93      0.65      0.71      8221
weighted avg       0.94      0.94      0.93      8221
```

Feature Importances

In [89]:

```
# feature_names = X_train.columns
# importances =  '_______' ## Complete the code to check the feature importance of the be
st model
# indices = np.argsort(importances)

# plt.figure(figsize=(12, 12))
# plt.title("Feature Importances")
# plt.barh(range(len(indices)), importances[indices], color="violet", align="center")
# plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
# plt.xlabel("Relative Importance")
# plt.show()


import numpy as np
import matplotlib.pyplot as plt

# Feature names from the training set
feature_names = X_train.columns

# Assuming 'tuned_gbm2' is the trained Gradient Boosting model
importances = tuned_gbm2.feature_importances_  # This retrieves the feature importances f
rom the model

# Sort the feature importances in ascending order
indices = np.argsort(importances)

# Create a horizontal bar chart to display feature importance
plt.figure(figsize=(12, 12))
plt.title("Feature Importances")
plt.barh(range(len(indices)), importances[indices], color="violet", align="center")
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```





Business Insights and Conclusions

-