Importing the Libraries

In [ ]:

```
# for basic manipulations
import numpy as np

# for DataFrames
import pandas as pd

# for plotting graphs
import matplotlib.pyplot as plt
import seaborn as sns

# required algorithms for traning models
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

# for converting String / Object data into numeric
from sklearn.preprocessing import LabelEncoder

# for spliting data for testing and traning
from sklearn.model_selection import train_test_split

# for measuring accuracy of model
from sklearn.metrics import confusion_matrix
from sklearn import metrics

# [Optional] for saving trained object
import joblib
```

Loading DataSet

In [ ]:

```
data_df = pd.read_csv("C:\\Users\\91911\\Downloads\\train.csv")
test_df = pd.read_csv("C:\\Users\\91911\\Downloads\\test.csv")
```

Cleaning

In [ ]:

```
# removing employee_id column
test_df.drop(columns=[data_df.columns[0]], axis=1, inplace=True)
data_df.drop(columns=[data_df.columns[0]], axis=1, inplace=True)
data_df.info()
```

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

```
 12  is_promoted           54808 non-null  int64
dtypes: float64(1), int64(7), object(5)
memory usage: 5.4+ MB
```

In [ ]:

```
# replace all empty cells with numpy.NaN
for df in (data_df, test_df):
    for i in df.columns:
        for j in range(len(df[i])):
            if df[i][j] == " ?":
                df[i][j] = np.NaN
```

In [ ]:

```
# droping NaN values
for df in (data_df, test_df):
    df.fillna(df.mean())
    df.dropna(axis=0, inplace=True)
```

In [ ]:

```
# droping Infinit values
for df in (data_df, test_df):
    df.replace([np.inf, -np.inf], np.nan).dropna(axis=1)
```

In [ ]:

```
# coverting `label` columns to `numeric`
le = LabelEncoder()
cat_cols = ["department", "region", "education", "gender", "recruitment_channel"]

for col in cat_cols:
    for df in (data_df, test_df):
        le.fit(df[col])
        df[col]= le.transform(df[col])

data_df = data_df.astype({"previous_year_rating": "float32"})
test_df = test_df.astype({"previous_year_rating": "float32"})
```

Training Model

In [ ]:

```
data_df.describe()
```

Out[ ]:

	department	region	education	gender	recruitment_channel	no_of_trainings
count	48660.000000	48660.000000	48660.000000	48660.000000	48660.000000	48660.000000
mean	4.963913	15.397801	0.617633	0.695684	0.868598	1.251993
std	2.484464	8.821645	0.918913	0.460122	0.980710	0.604994
min	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	4.000000	11.000000	0.000000	0.000000	0.000000	1.000000
50%	5.000000	14.000000	0.000000	1.000000	0.000000	1.000000
75%	7.000000	21.000000	2.000000	1.000000	2.000000	1.000000
max	8.000000	33.000000	2.000000	1.000000	2.000000	10.000000

In [ ]:

```
test_df.describe()
```

Out[ ]:

	department	region	education	gender	recruitment_channel	no_of_trainings
count	20819.000000	20819.000000	20819.000000	20819.000000	20819.000000	20819.000000
mean	4.957058	15.420001	0.628416	0.700562	0.864211	1.251261
std	2.488127	8.768036	0.922687	0.458023	0.980870	0.595103
min	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	4.000000	11.000000	0.000000	0.000000	0.000000	1.000000
50%	5.000000	14.000000	0.000000	1.000000	0.000000	1.000000
75%	7.000000	21.000000	2.000000	1.000000	2.000000	1.000000
max	8.000000	33.000000	2.000000	1.000000	2.000000	9.000000

In [ ]:

```
# creating new dataframes for input and output of model
X = data_df.drop(columns=['is_promoted'], axis=1).dropna(axis=1) # input
y = data_df['is_promoted'].dropna() # output

# creating random train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_stat
```

Using DecisionTreeClassifier

In [ ]:

```
# instantiation of Model with optimization for classification
model = DecisionTreeClassifier(criterion="entropy", max_depth=12)
model.fit(X_train, y_train)

# [optional] saving trained model
joblib.dump(model, "trained-model.joblib")
```

Out[ ]:

```
['trained-model.joblib']
```

In [ ]:

```
# measuring Accuracy
y_pred = model.predict(X_test)
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9349910946705028
```

Using RandomForestClassifier

In [ ]:

```
# instantiation of Model with optimization for classification
fmodel = RandomForestClassifier(n_estimators=12)
fmodel.fit(X_train, y_train)
```

Out[ ]:

```
RandomForestClassifier(n_estimators=12)
```

In [ ]:

```
# measuring Accuracy
print("Accuracy:", fmodel.score(X_test, y_test))
```

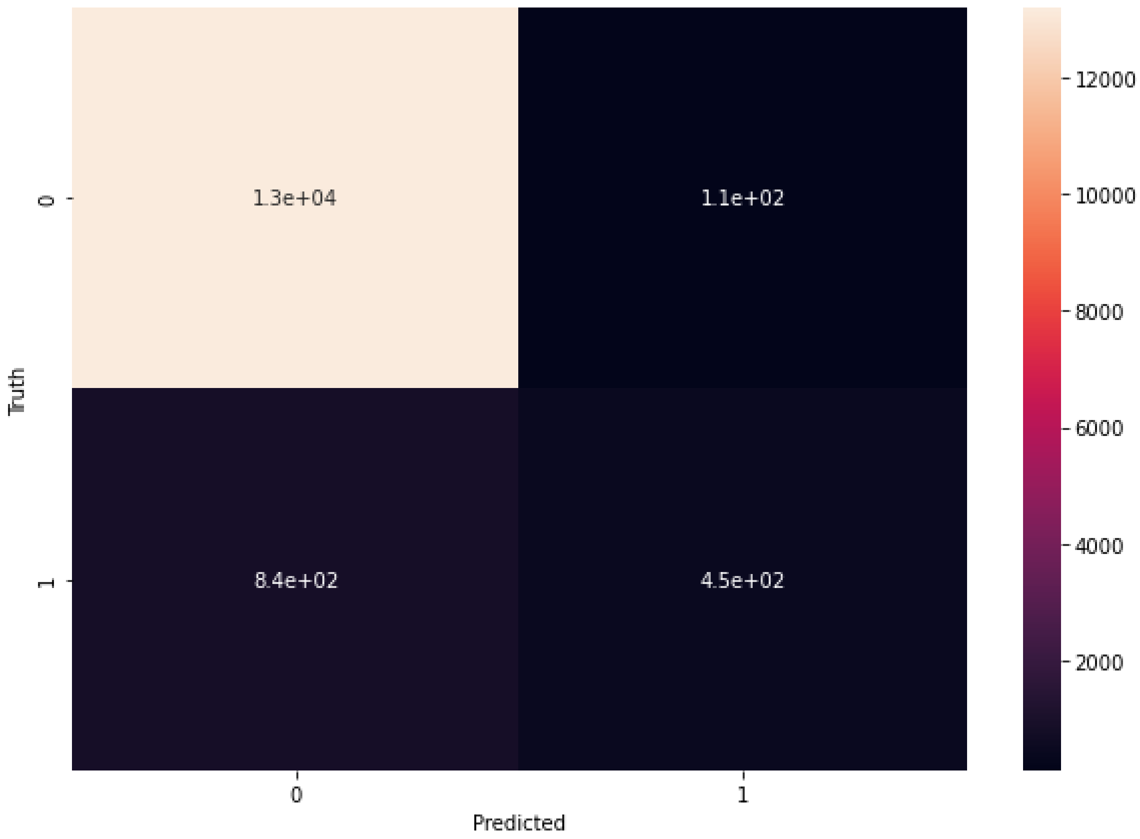
```
Accuracy: 0.9228661460474038
```

In [ ]:
```
y_predicted = model.predict(X_test)

# confusion matrix for random forest
cm = confusion_matrix(y_test, y_predicted)
cm
```

Out[ ]:
```
array([[13197,   108],
       [  841,   452]], dtype=int64)
```

In [ ]:
```
# ploting confusion matrix
## definition : A confusion matrix is a technique for summarizing the performance of
plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Out[ ]:
```
Text(69.0, 0.5, 'Truth')
```



In [ ]:
```
# making prediction on the test.csv data
decision_predict = model.predict(test_df)
rand_forest_predict = fmodel.predict(test_df)

decision_predict, rand_forest_predict
```

Out[ ]:
```
(array([0, 0, 0, ..., 0, 0, 1], dtype=int64),
 array([0, 0, 0, ..., 0, 0, 0], dtype=int64))
```

EDA

In [ ]:
```
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
```

In [ ]:
```
train.head()
```

Out[ ]:

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

In [ ]:

```
train.columns
```

Out[ ]:

```
Index(['employee_id', 'department', 'region', 'education', 'gender',
       'recruitment_channel', 'no_of_trainings', 'age', 'previous_year_rating',
       'length_of_service', 'KPIs_met >80%', 'awards_won?',
       'avg_training_score', 'is_promoted'],
      dtype='object')
```

In [ ]:

```
train.info()
```

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

In [ ]:

```
train.isnull().sum()
```

Out[ ]:

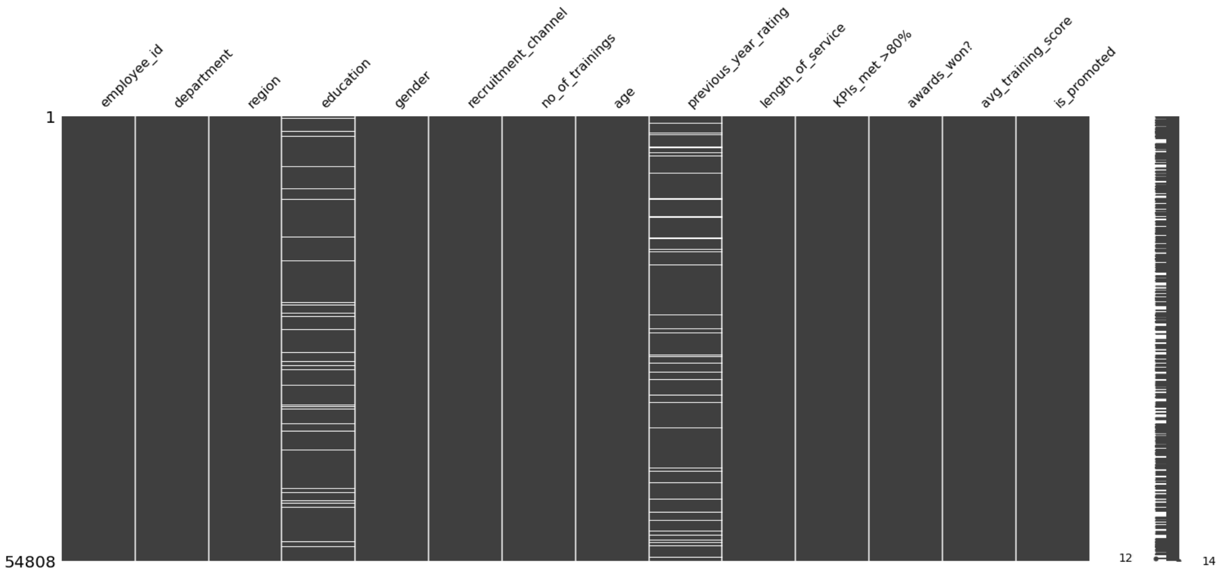
```
employee_id                0
department                 0
region                     0
education               2409
gender                     0
recruitment_channel        0
no_of_trainings            0
age                        0
previous_year_rating    4124
length_of_service          0
KPIs_met >80%              0
awards_won?                0
avg_training_score         0
```

```
is_promoted            0
dtype: int64
```

In [ ]:

```
# Visualizing the null values using missingo function

import missingno as msno
msno.matrix(train)
```

Out[ ]: <AxesSubplot:>



In [ ]:

```
test.head()
```

Out[ ]:

	employee_id	department	region	education	gender	recruitment_channel	no_of_trainings	a
0	8724	Technology	region_26	Bachelor's	m	sourcing	1	
1	74430	HR	region_4	Bachelor's	f	other	1	
2	72255	Sales & Marketing	region_13	Bachelor's	m	other	1	
3	38562	Procurement	region_2	Bachelor's	f	other	3	
4	64486	Finance	region_29	Bachelor's	m	sourcing	1	

In [ ]:

```
test.info() ### Check all information in the datasets
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23490 entries, 0 to 23489
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   employee_id           23490 non-null  int64
 1   department            23490 non-null  object
 2   region                23490 non-null  object
 3   education             22456 non-null  object
 4   gender                23490 non-null  object
 5   recruitment_channel   23490 non-null  object
 6   no_of_trainings       23490 non-null  int64
 7   age                   23490 non-null  int64
 8   previous_year_rating  21678 non-null  float64
 9   length_of_service     23490 non-null  int64
```

```
 10  KPIs_met >80%          23490 non-null  int64
 11  awards_won?            23490 non-null  int64
 12  avg_training_score     23490 non-null  int64
dtypes: float64(1), int64(7), object(5)
memory usage: 2.3+ MB
```

In [ ]:

```
test.isnull().sum()
```

Out[ ]:

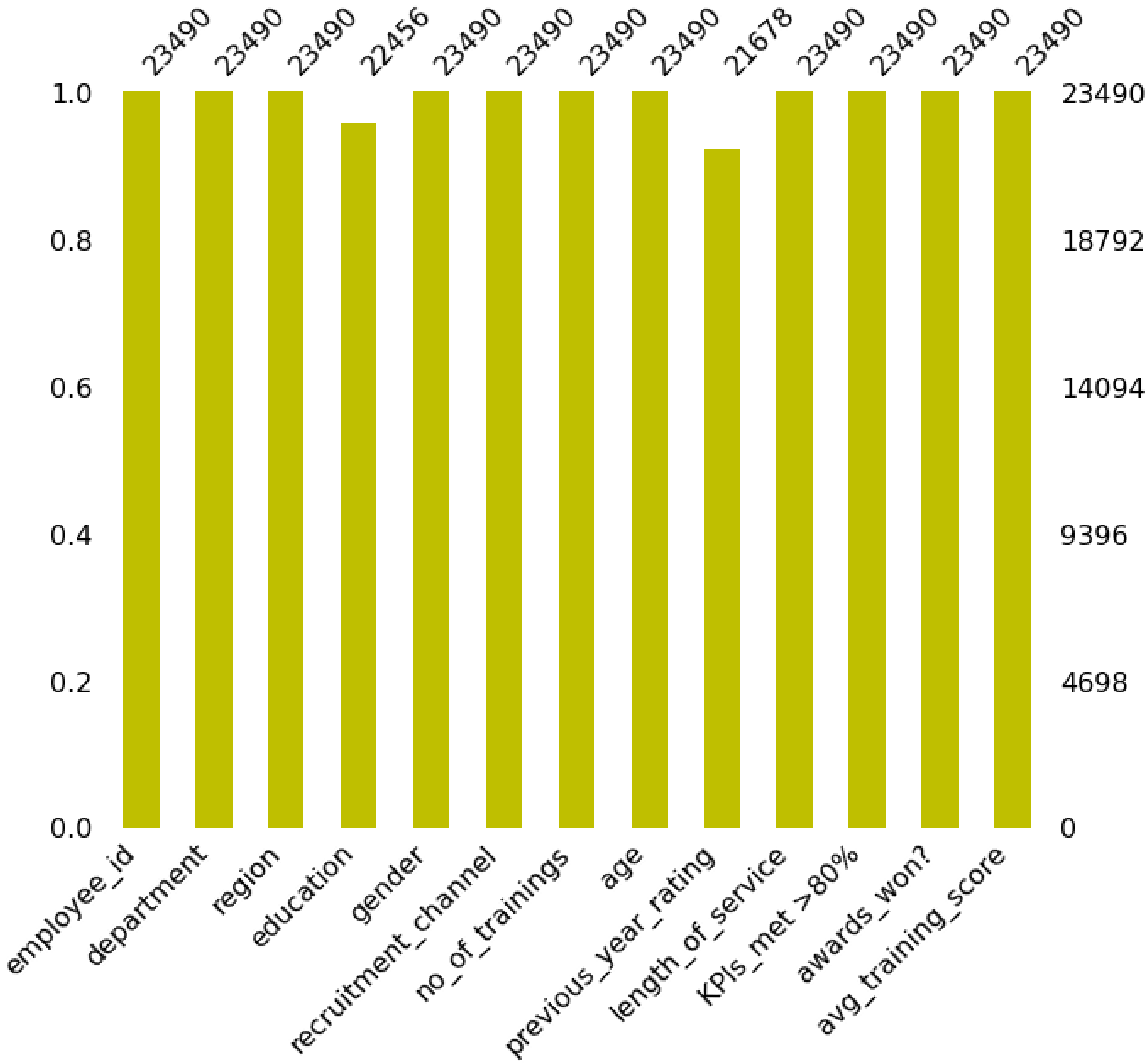
```
employee_id                0
department                 0
region                     0
education               1034
gender                     0
recruitment_channel        0
no_of_trainings            0
age                        0
previous_year_rating    1812
length_of_service          0
KPIs_met >80%              0
awards_won?                0
avg_training_score         0
dtype: int64
```

In [ ]:

```
##A barplot (or barchart) is one of the most common types of graphic.
##It shows the relationship between a numeric and a categoric variable.
##Each entity of the categoric variable is represented as a bar. The size of the bar
msno.bar(test, color = 'y', figsize = (10,8))  #### Check the missing values in test
```

Out[ ]:

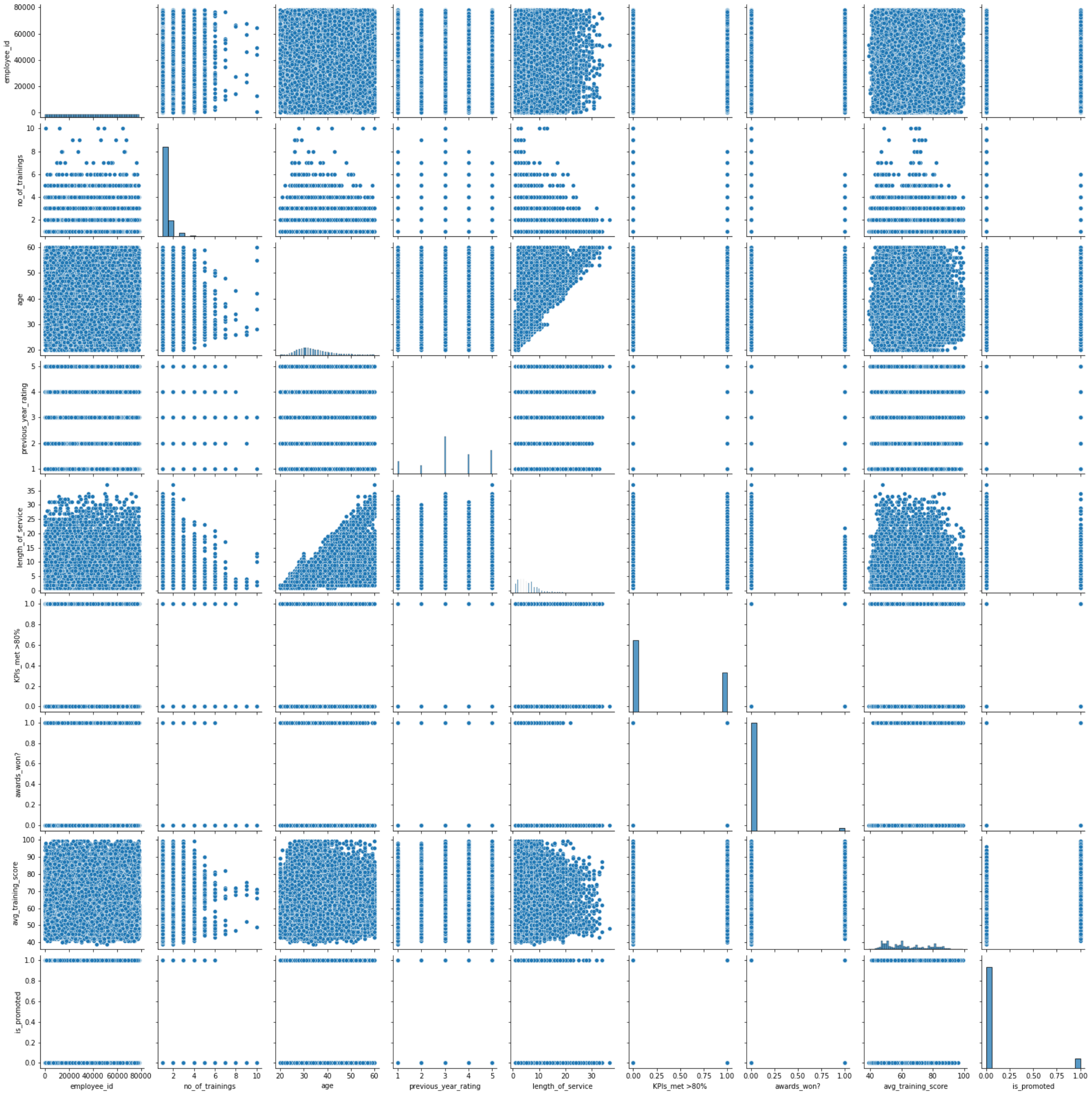
```
<AxesSubplot:>
```



In [ ]:

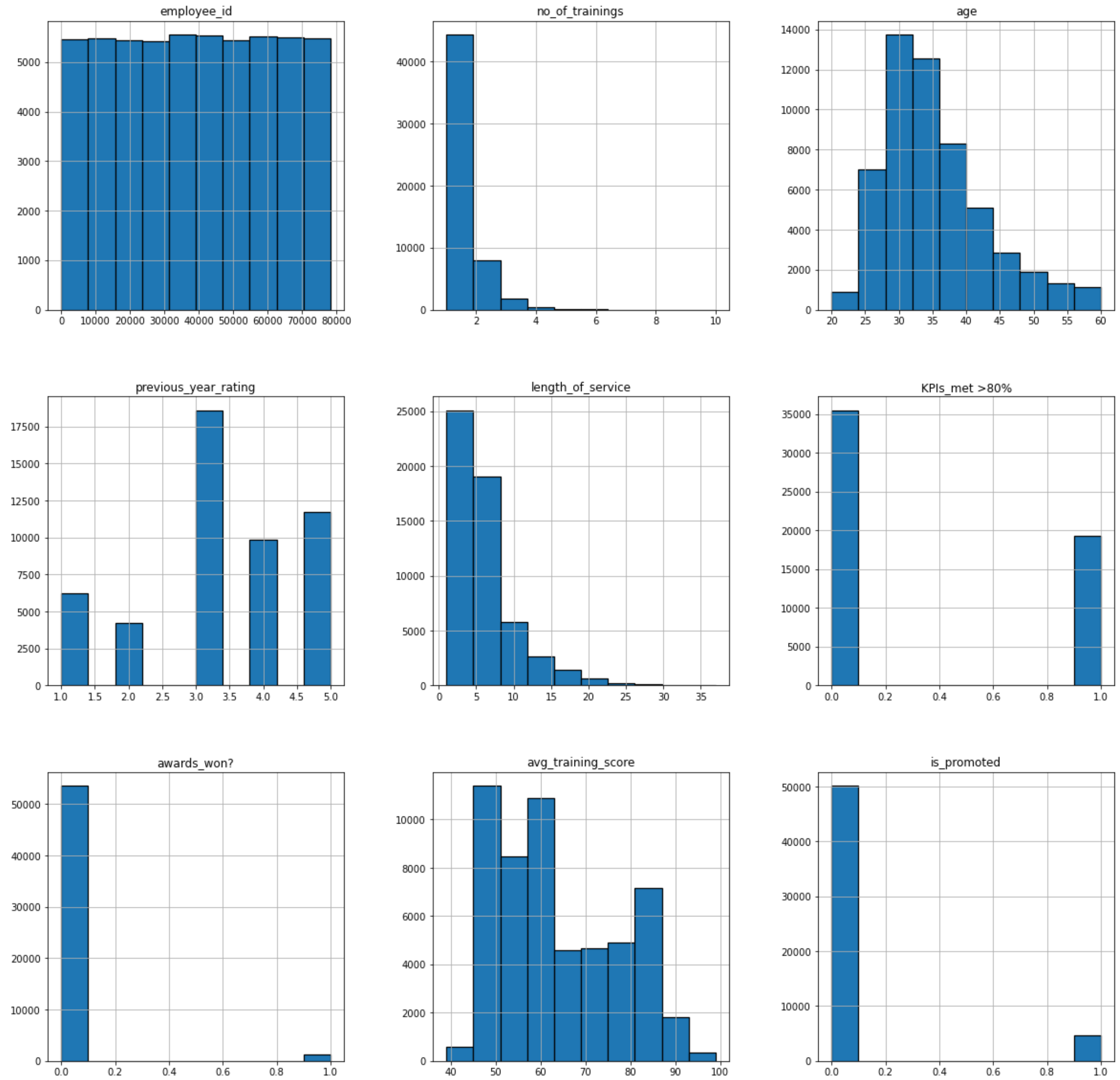
```
### Pairplot using seaborn library
sns.pairplot(train)
```

Out[ ]: <seaborn.axisgrid.PairGrid at 0x1b7e59ebc10>



In [ ]:

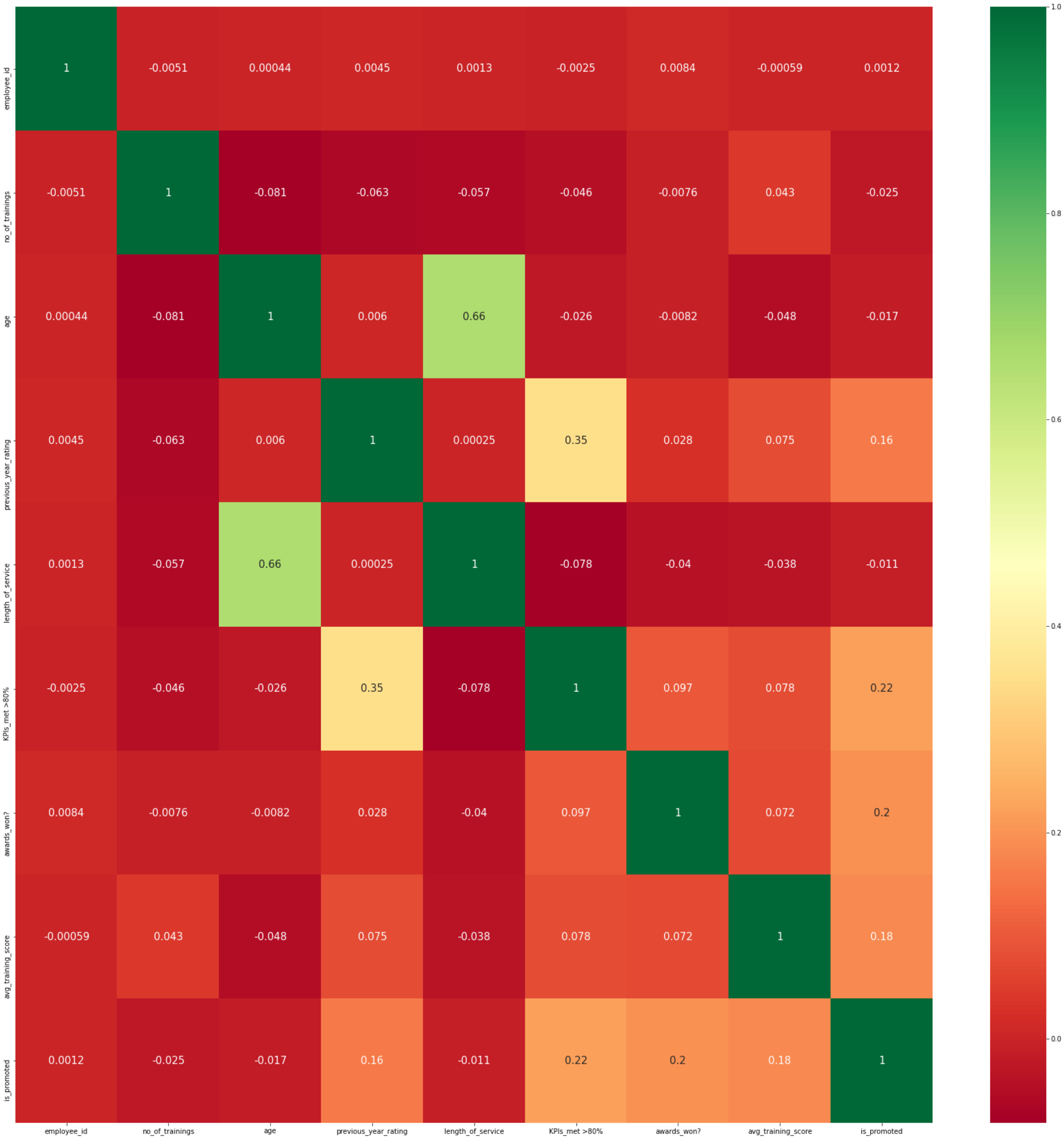
```
# Visulazing the distibution of the data for every feature
train.hist(edgecolor='black', linewidth=1.2, figsize=(20, 20));
```



In [ ]:

```
##heat map : a representation of data in the form of a map or diagram in which data
plt.figure(figsize=(30, 30))
sns.heatmap(train.corr(), annot=True, cmap="RdYlGn", annot_kws={"size":15})
```

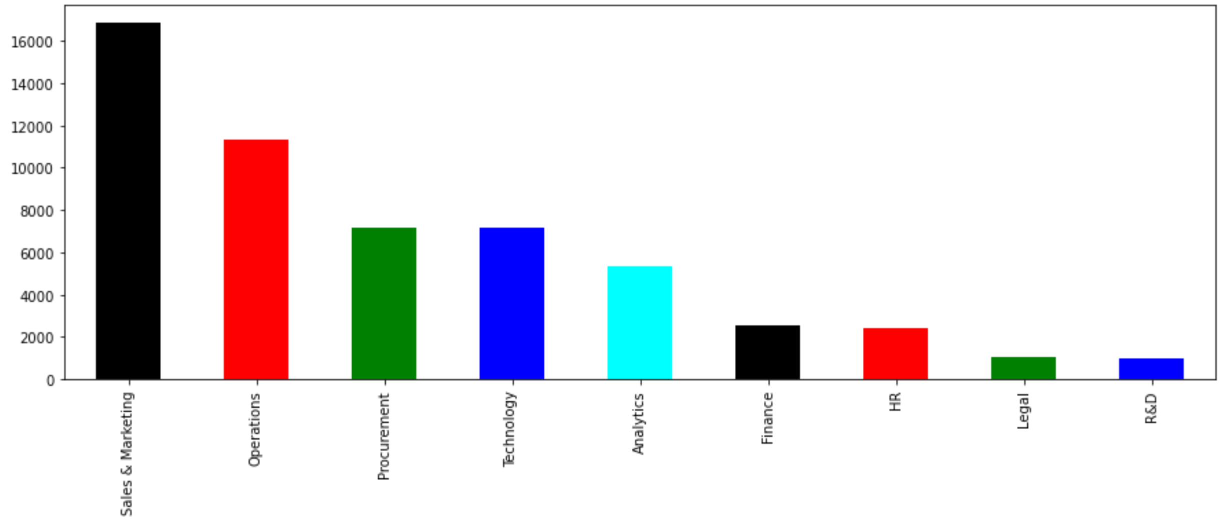
Out[ ]: <AxesSubplot:>



In [ ]:
```
train['department'].value_counts()
```

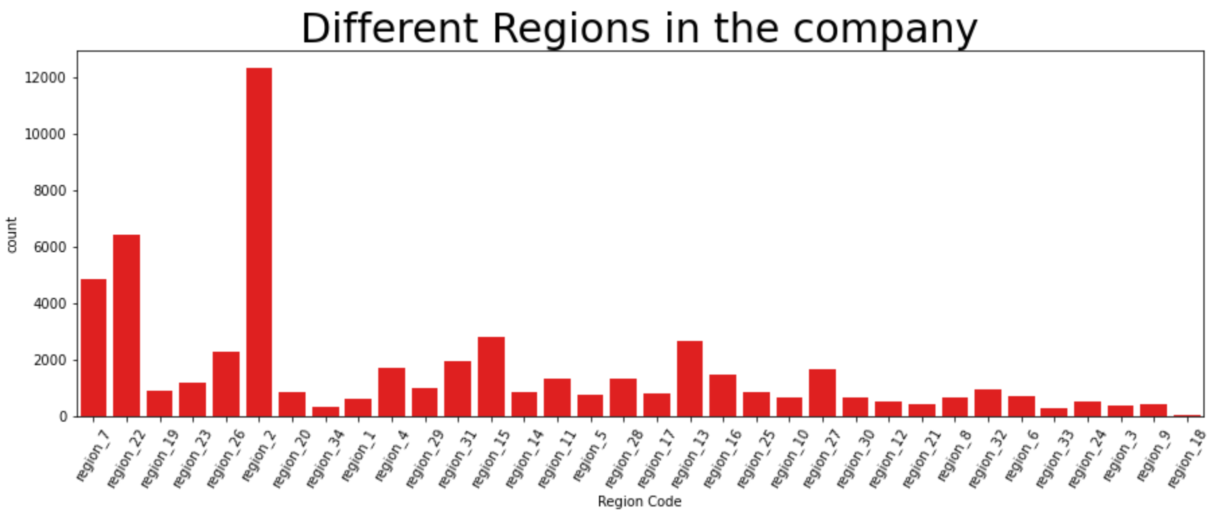
Out[ ]:
```
Sales & Marketing    16840
Operations           11348
Procurement           7138
Technology            7138
Analytics             5352
Finance               2536
HR                    2418
Legal                 1039
R&D                    999
Name: department, dtype: int64
```

In [ ]:
```
# visualizing the different groups in the dataset
plt.subplots(figsize=(15,5))
train['department'].value_counts(normalize = True)
train['department'].value_counts(dropna = False).plot.bar(color=['black', 'red', 'gr
plt.show()
```



In [ ]:

```
# checking the different regions of the company
plt.subplots(figsize=(15,5))
sns.countplot(train['region'], color = 'red')
plt.title('Different Regions in the company', fontsize = 30)
plt.xticks(rotation = 60)
plt.xlabel('Region Code')
plt.ylabel('count')
plt.show()
```



In [ ]:

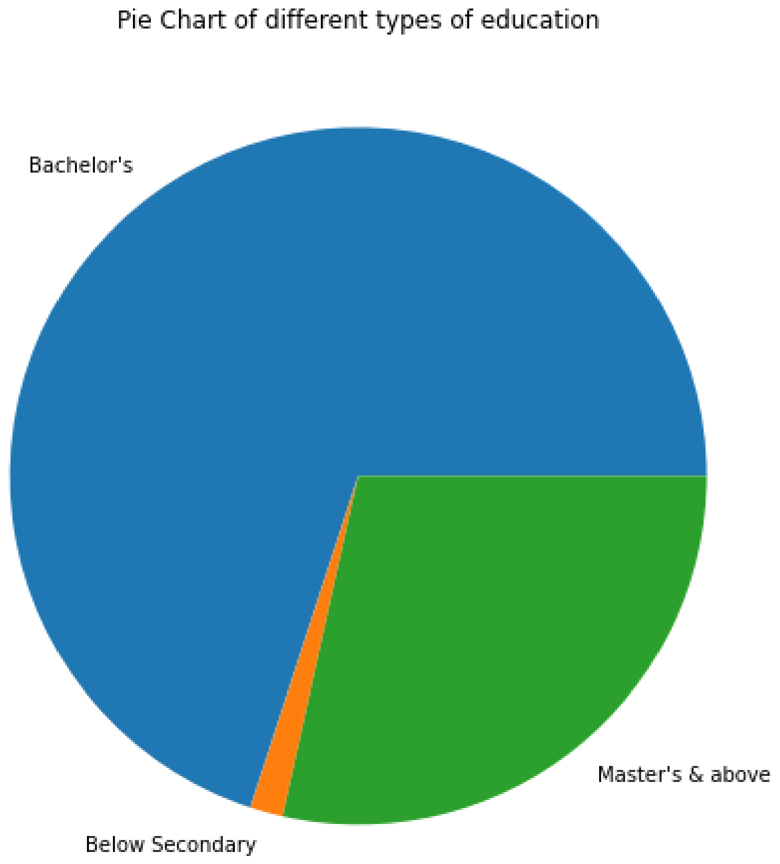
```
train['education'].value_counts()
```

Out[ ]:

```
Bachelor's          36669
Master's & above    14925
Below Secondary       805
Name: education, dtype: int64
```

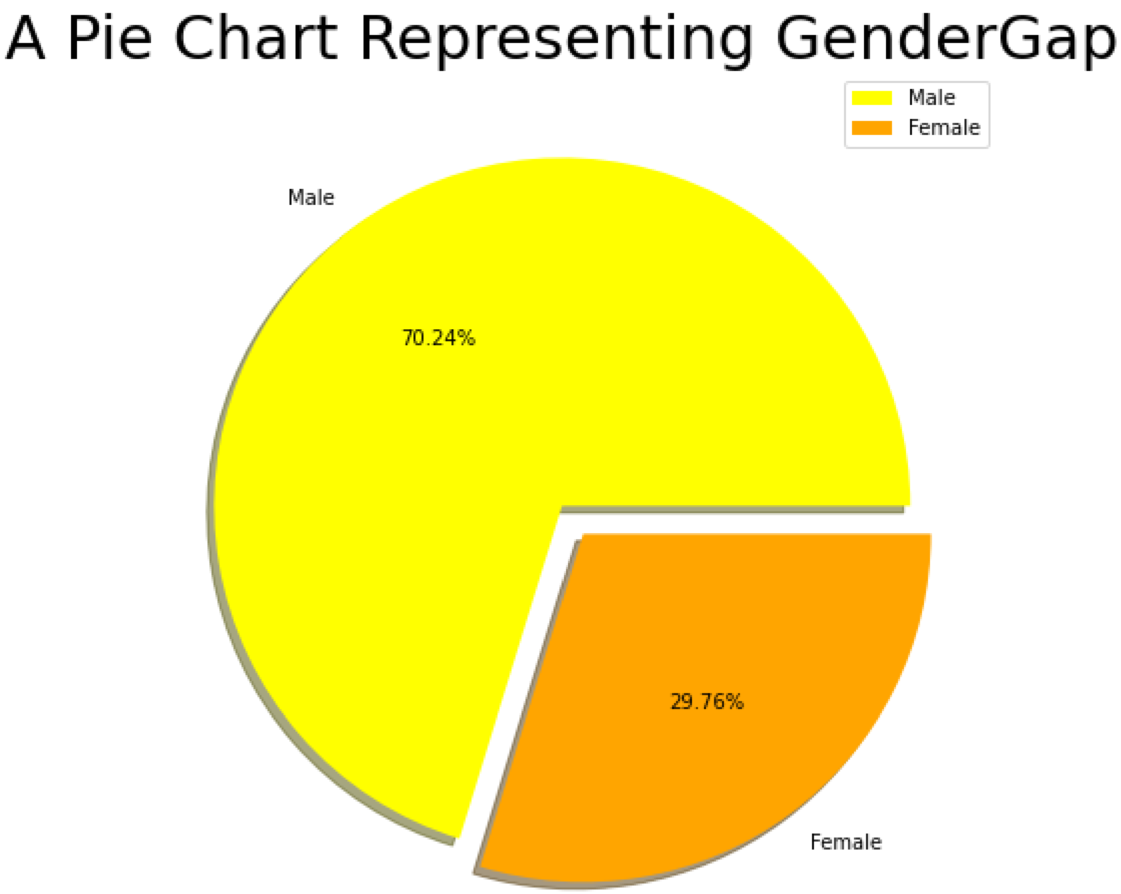
In [ ]:

```
##pie chart : a type of graph in which a circle is divided into sectors that each re
# Prepare Data
df = train.groupby('education').size()

# Make the plot with pandas
df.plot(kind='pie', subplots=True, figsize=(15, 8))
plt.title("Pie Chart of different types of education")
plt.ylabel("")
plt.show()
```



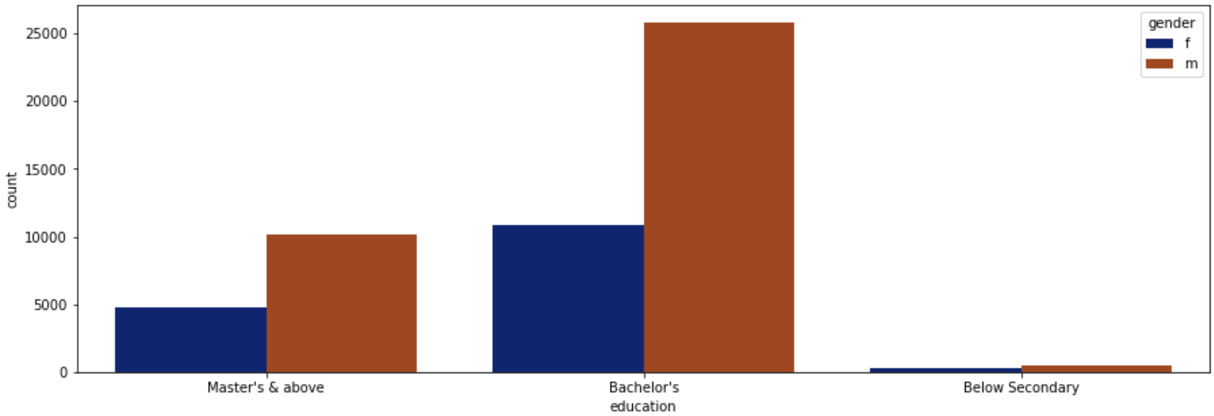
In [ ]:

```
# plotting a pie chart

size = [38496, 16312]
labels = "Male", "Female"
colors = ['yellow', 'orange']
explode = [0, 0.1]

plt.subplots(figsize=(8,8))
plt.pie(size, labels = labels, colors = colors, explode = explode, shadow = True, au
plt.title('A Pie Chart Representing GenderGap', fontsize = 30)
plt.axis('off')
plt.legend()
plt.show()
```



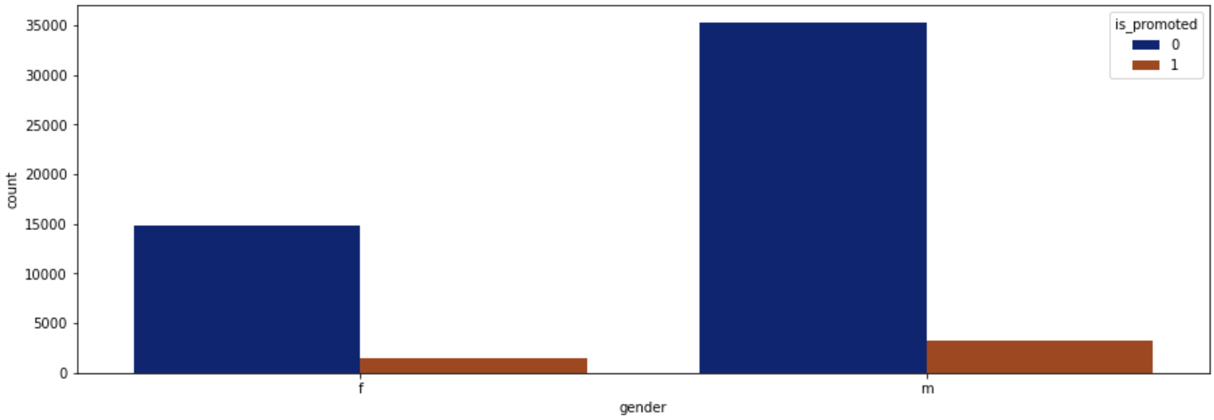
In [ ]:

```
# comparison of promoted gender male & female
plt.subplots(figsize=(15,5))
sns.countplot(x = 'education', data = train, hue = 'gender', palette = 'dark')
plt.show()
```



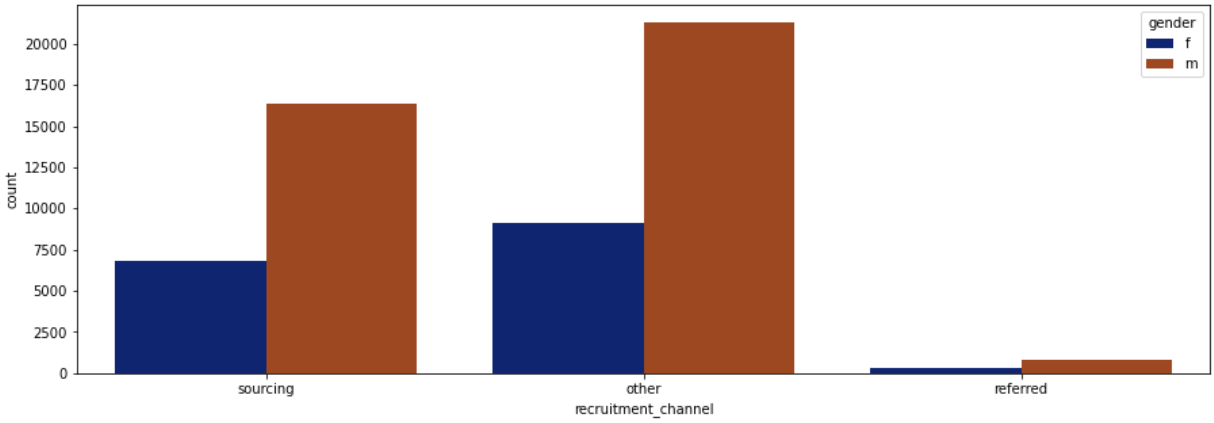
In [ ]:

```
# comparison of promoted gender male & female
plt.subplots(figsize=(15,5))
sns.countplot(x = 'gender', data = train, hue = 'is_promoted', palette = 'dark')
plt.show()
```



In [ ]:

```
# comparison of requirement gender male & female
plt.subplots(figsize=(15,5))
sns.countplot(x = 'recruitment_channel', data = train, hue = 'gender', palette = 'da
plt.show()
```



In [ ]:

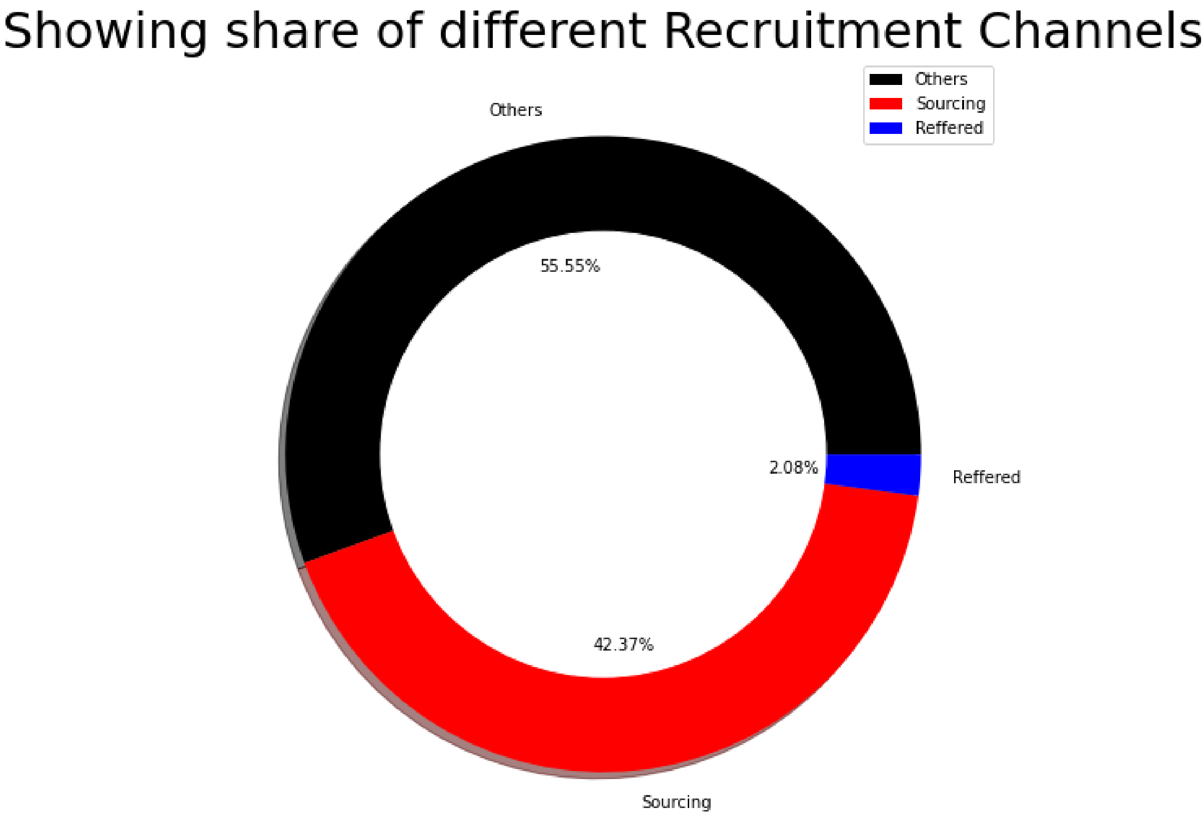
```
train['recruitment_channel'].value_counts()
```

Out[ ]:

```
other       30446
sourcing    23220
referred     1142
Name: recruitment_channel, dtype: int64
```

In [ ]:

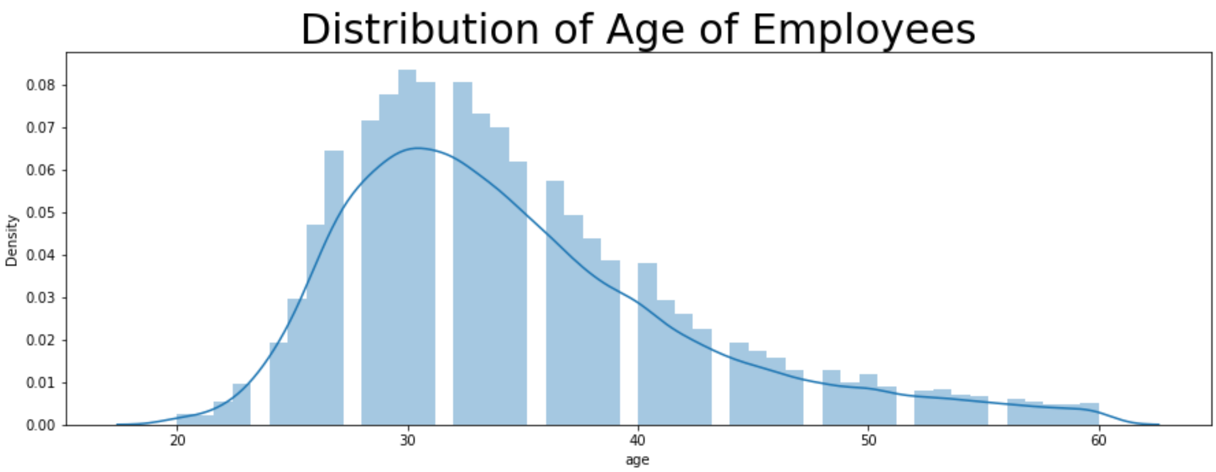
```
## A donut chart is essentially a Pie Chart with an area of the centre cut out.
## these are more efficent than pie charts
# plotting a donut chart for visualizing each of the recruitment channel's share

size = [30446, 23220, 1142]
colors = ['black', 'red', 'blue']
labels = "Others", "Sourcing", "Reffered"

my_circle = plt.Circle((0, 0), 0.7, color = 'white')

plt.rcParams['figure.figsize'] = (9, 9)
plt.pie(size, colors = colors, labels = labels, shadow = True, autopct = '%.2f%%')
plt.title('Showing share of different Recruitment Channels', fontsize = 30)
p = plt.gcf()
p.gca().add_artist(my_circle)
plt.legend()
plt.show()
```



In [ ]:

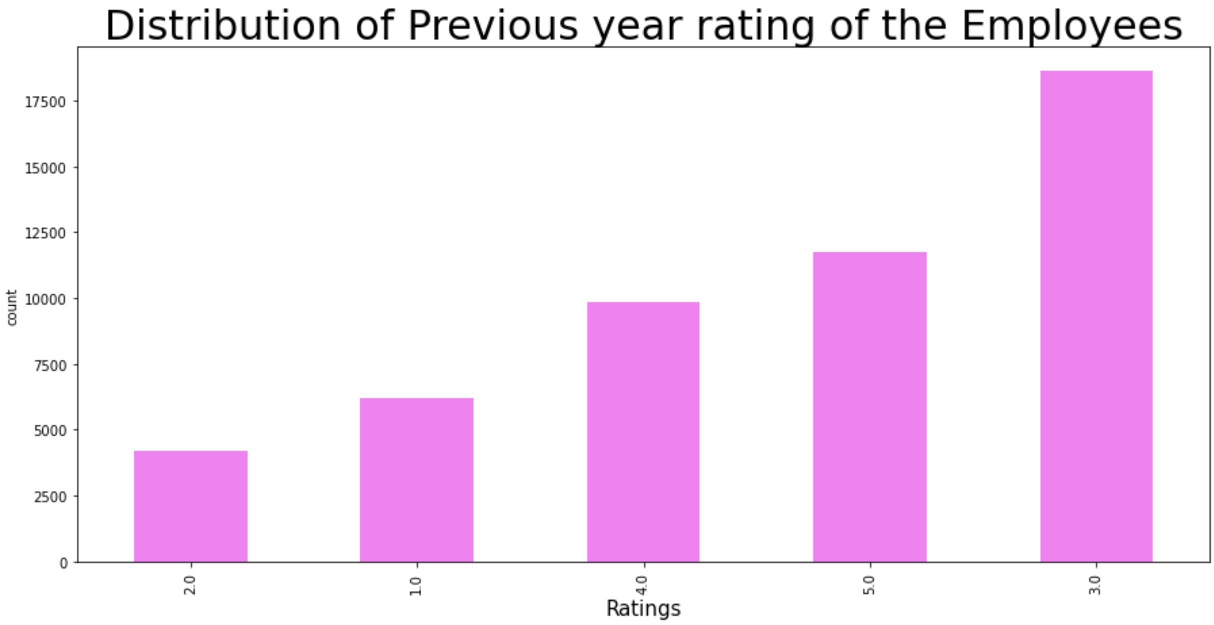
```
plt.subplots(figsize=(15,5))
sns.distplot(train['age'])
plt.title('Distribution of Age of Employees', fontsize = 30)
```

Out[ ]: Text(0.5, 1.0, 'Distribution of Age of Employees')



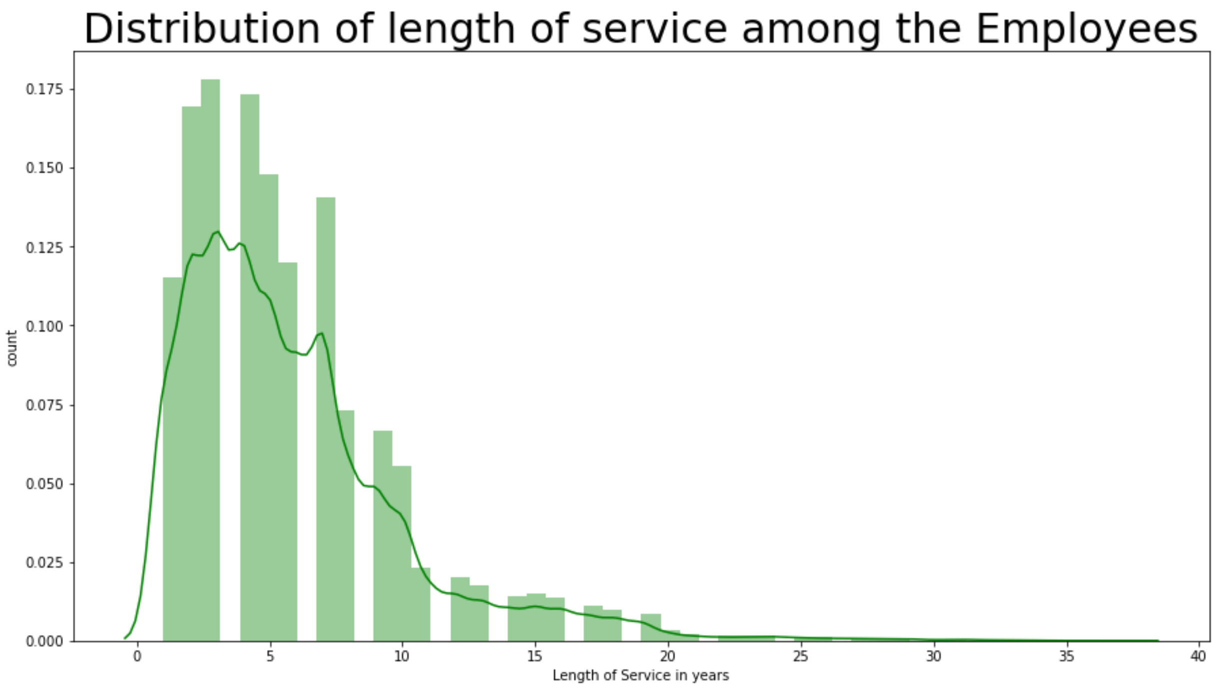
In [ ]:

```
train['previous_year_rating'].value_counts().sort_values().plot.bar(color = 'violet'
plt.title('Distribution of Previous year rating of the Employees', fontsize = 30)
plt.xlabel('Ratings', fontsize = 15)
plt.ylabel('count')
plt.show()
```



In [ ]:

```
# checking the distribution of length of service
plt.subplots(figsize=(15,8))
sns.distplot(train['length_of_service'], color = 'green')
plt.title('Distribution of length of service among the Employees', fontsize = 30)
plt.xlabel('Length of Service in years')
plt.ylabel('count')
plt.show()
```



In [ ]:

```
train['KPIs_met >80%'].value_counts()
```

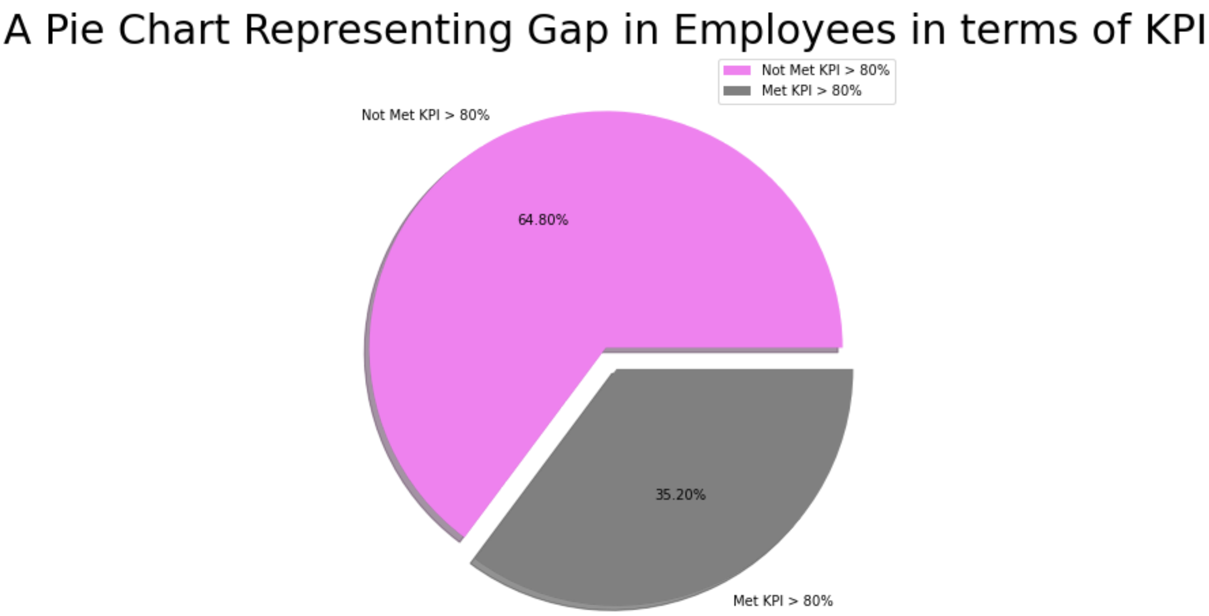
Out[ ]:

```
0    35517
1    19291
Name: KPIs_met >80%, dtype: int64
```

In [ ]:

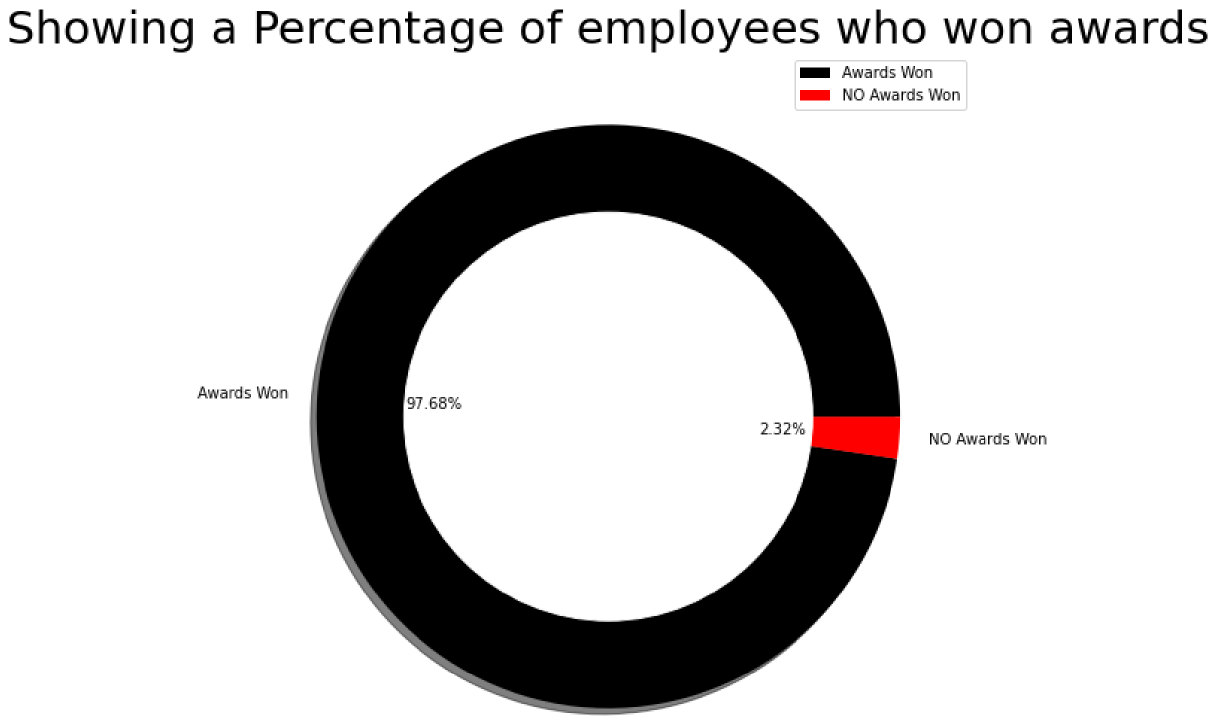
```
# plotting a pie chart


size = [35517, 19291]
labels = "Not Met KPI > 80%", "Met KPI > 80%"
```

```
colors = ['violet', 'grey']
explode = [0, 0.1]

plt.rcParams['figure.figsize'] = (8, 8)
plt.pie(size, labels = labels, colors = colors, explode = explode, shadow = True, au
plt.title('A Pie Chart Representing Gap in Employees in terms of KPI', fontsize = 30
plt.axis('off')
plt.legend()
plt.show()
```



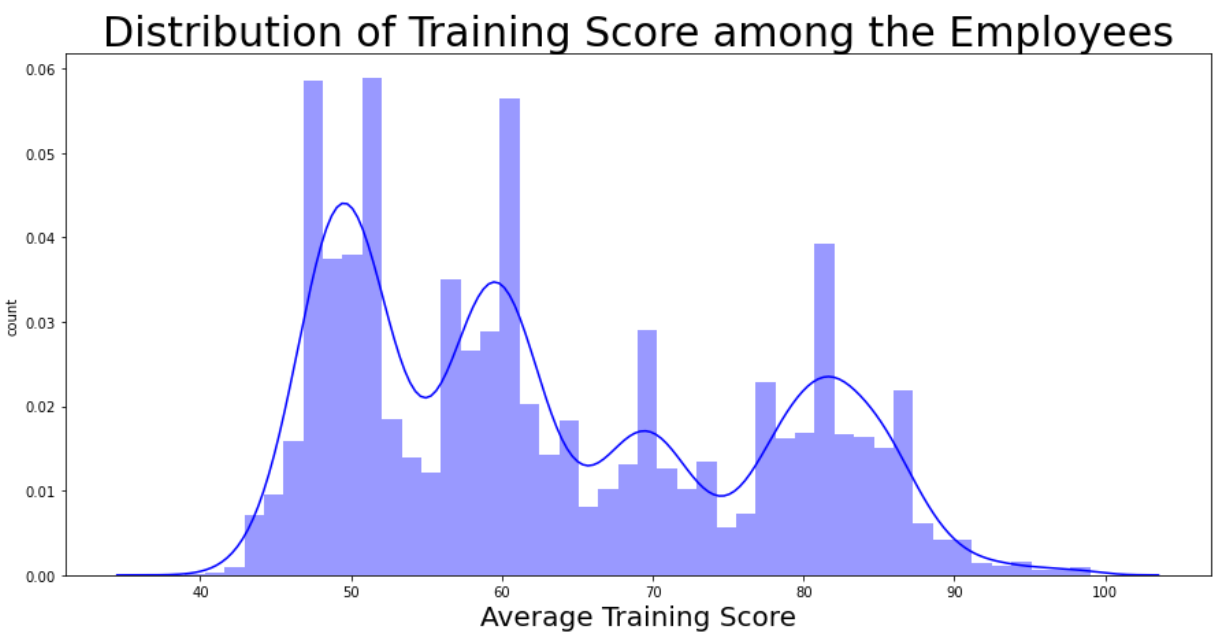
In [ ]:

```
# plotting a donut chart for visualizing each of the recruitment channel's share

size = [53538, 1270]
colors = ['black', 'red']
labels = "Awards Won", "NO Awards Won"

my_circle = plt.Circle((0, 0), 0.7, color = 'white')

plt.rcParams['figure.figsize'] = (9, 9)
plt.pie(size, colors = colors, labels = labels, shadow = True, autopct = '%.2f%%')
plt.title('Showing a Percentage of employees who won awards', fontsize = 30)
p = plt.gcf()
p.gca().add_artist(my_circle)
plt.legend()
plt.show()
```



In [ ]:

```
# checking the distribution of the avg_training score of the Employees

plt.subplots(figsize=(15,7))
sns.distplot(train['avg_training_score'], color = 'blue')
plt.title('Distribution of Training Score among the Employees', fontsize = 30)
plt.xlabel('Average Training Score', fontsize = 20)
plt.ylabel('count')
plt.show()
```



In [ ]:

```
train['is_promoted'].value_counts()
```

Out[ ]:

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

In [ ]:

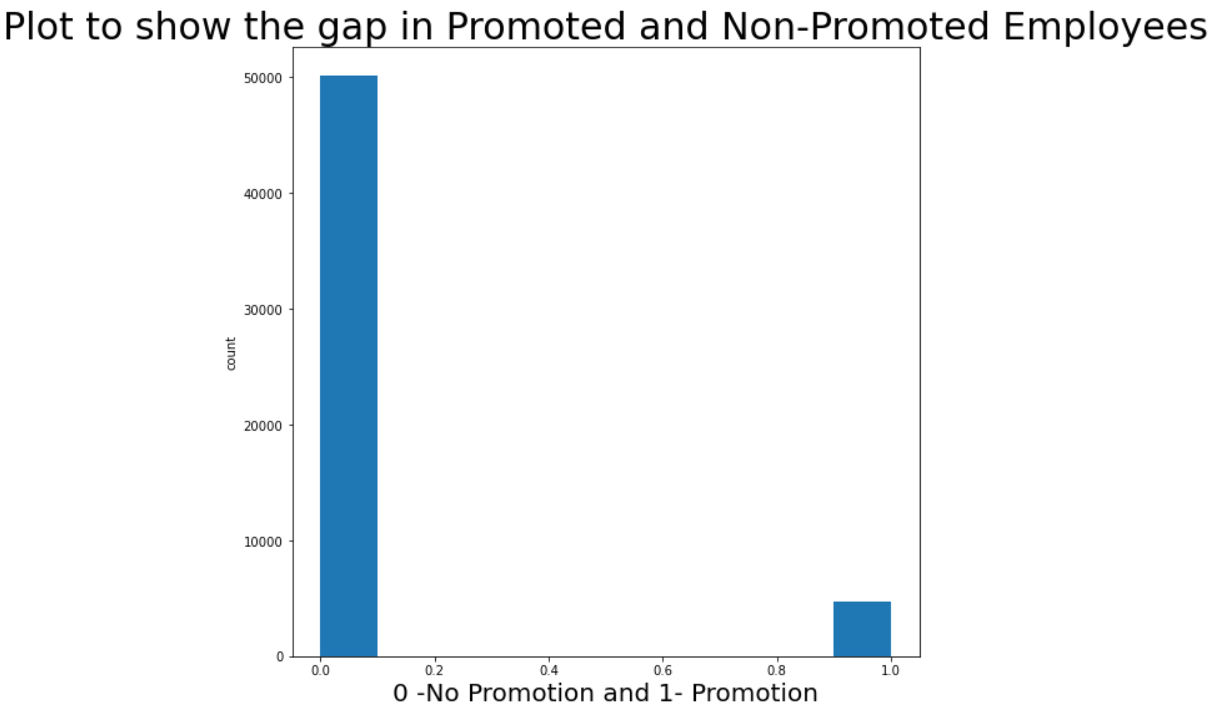
```
# finding the %age of people promoted
```

```
promoted = (4668/54808)*100
print("Percentage of Promoted Employees is {:.2f}%".format(promoted))
```

Percentage of Promoted Employees is 8.52%

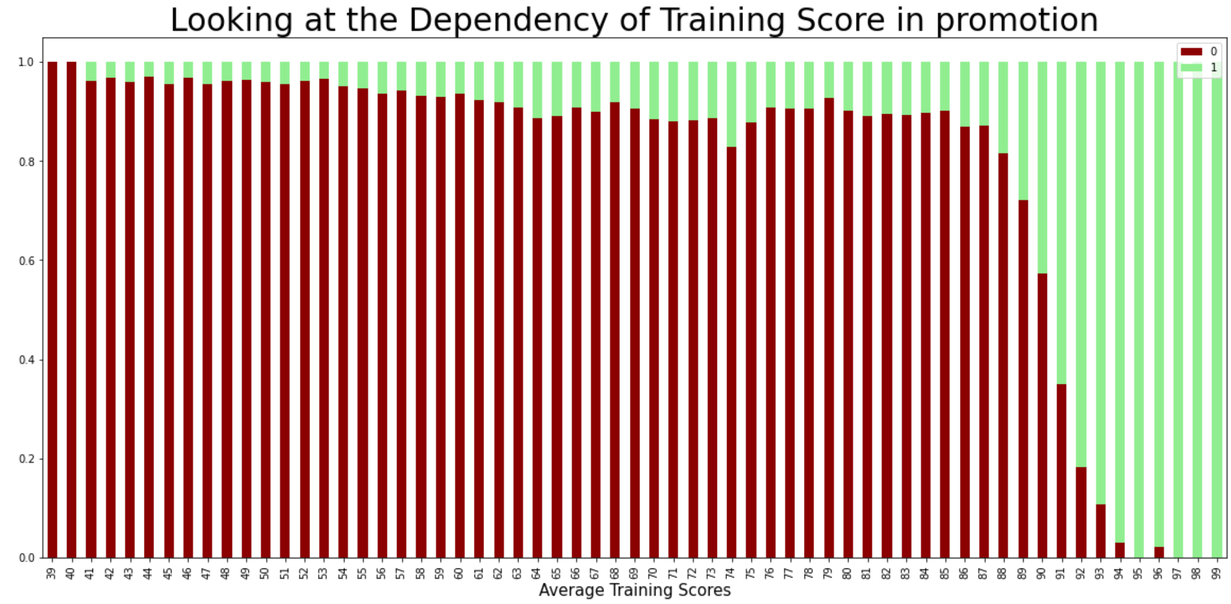
In [ ]:

```
##A histogram is a bar graph-like representation of data that buckets a range of out
##The y-axis represents the number count or percentage of occurrences in the data fo
plt.hist(train['is_promoted'])
plt.title('Plot to show the gap in Promoted and Non-Promoted Employees', fontsize =
plt.xlabel('0 -No Promotion and 1- Promotion', fontsize = 20)
plt.ylabel('count')
plt.show()
```



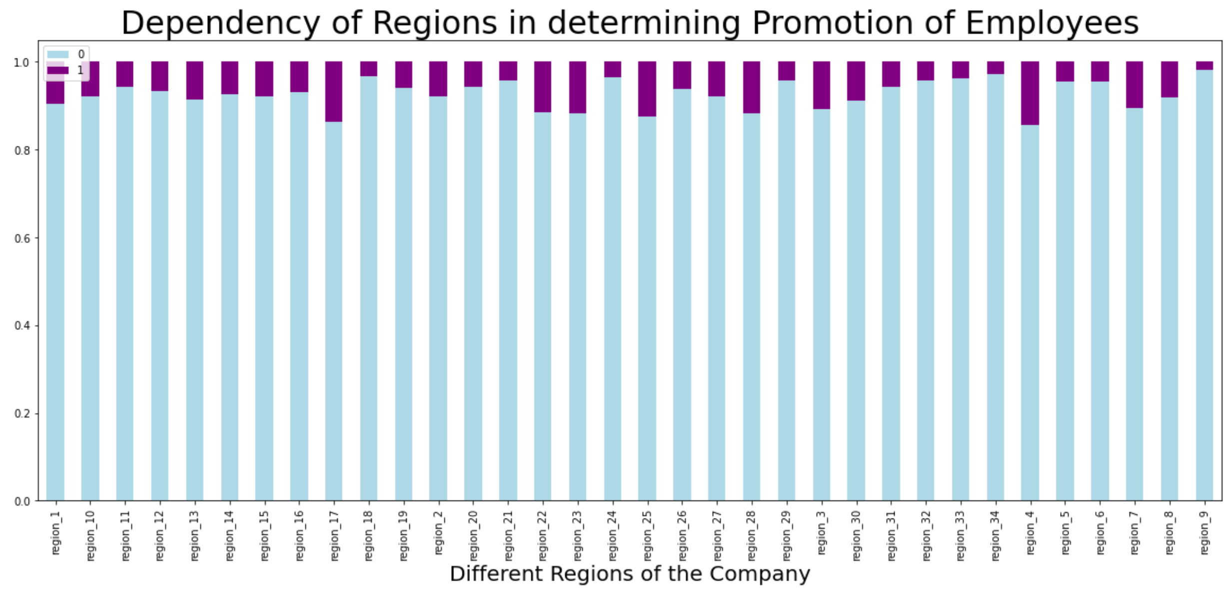
In [ ]:

```
##Scatter plots are the graphs that present the relationship between two variables i
##It represents data points on a two-dimensional plane or on a Cartesian system
# scatter plot between average training score and is_promoted

data = pd.crosstab(train['avg_training_score'], train['is_promoted'])
data.div(data.sum(1).astype(float), axis = 0).plot(kind = 'bar', stacked = True, fig

plt.title('Looking at the Dependency of Training Score in promotion', fontsize = 30)
plt.xlabel('Average Training Scores', fontsize = 15)
plt.legend()
plt.show()
```



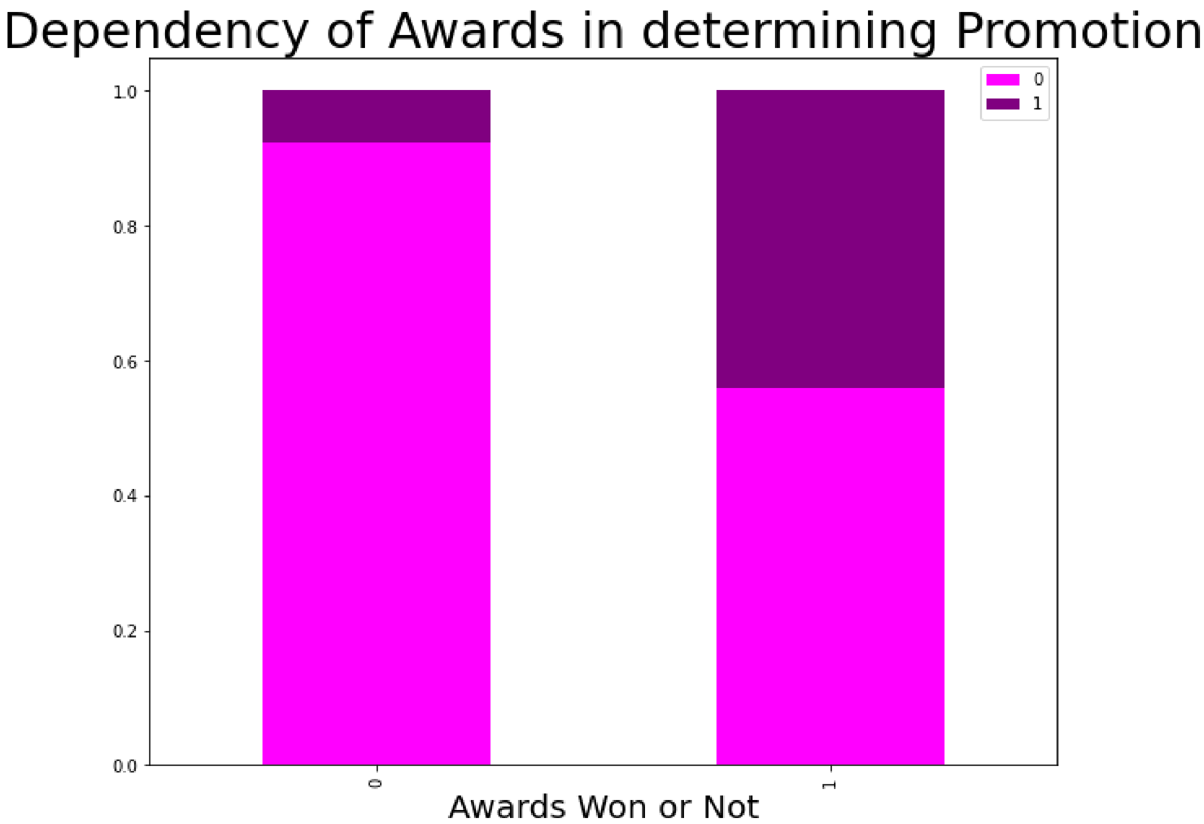
In [ ]:

```
# checking dependency of different regions in promotion

data = pd.crosstab(train['region'], train['is_promoted'])
data.div(data.sum(1).astype('float'), axis = 0).plot(kind = 'bar', stacked = True, f

plt.title('Dependency of Regions in determining Promotion of Employees', fontsize =
plt.xlabel('Different Regions of the Company', fontsize = 20)
plt.legend()
plt.show()
```



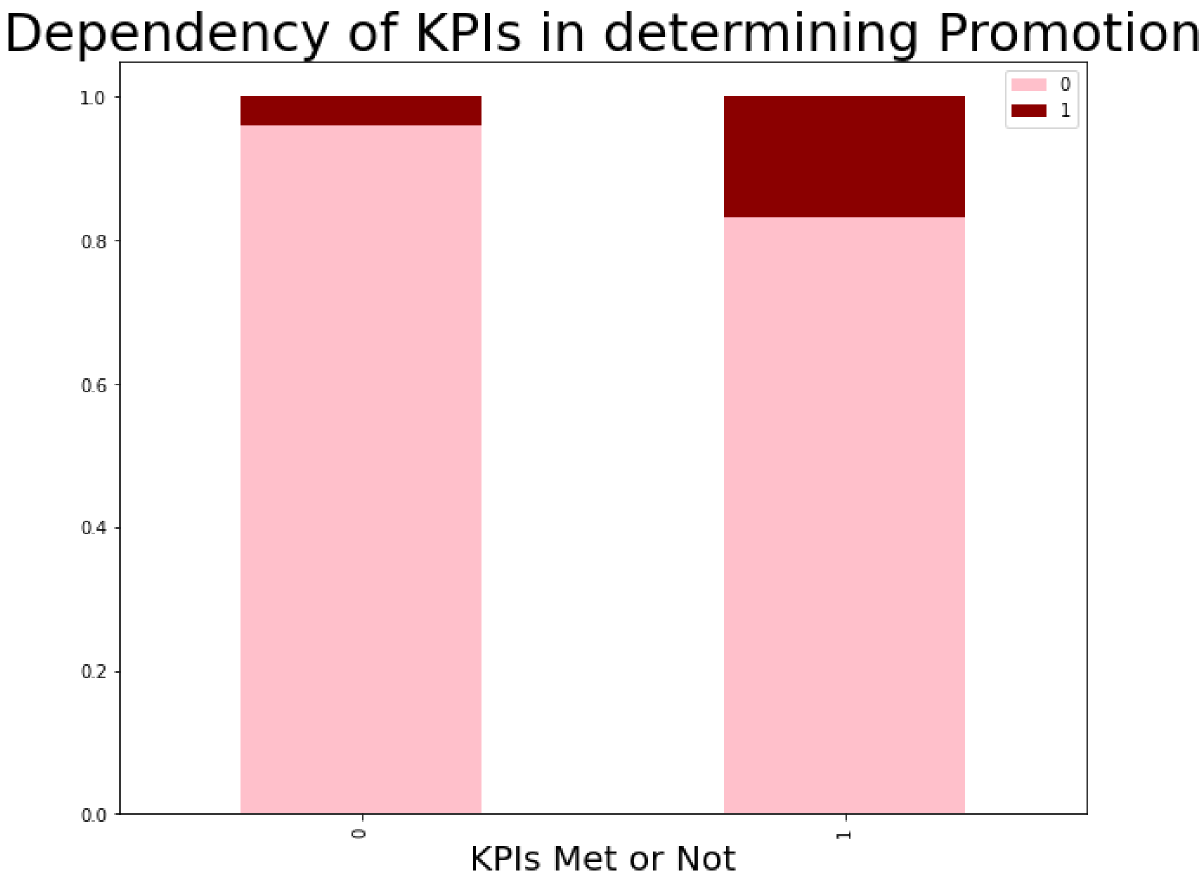
In [ ]:

```
# dependency of awards won on promotion

data = pd.crosstab(train['awards_won?'], train['is_promoted'])
data.div(data.sum(1).astype('float'), axis = 0).plot(kind = 'bar', stacked = True, f

plt.title('Dependency of Awards in determining Promotion', fontsize = 30)
plt.xlabel('Awards Won or Not', fontsize = 20)
plt.legend()
plt.show()
```



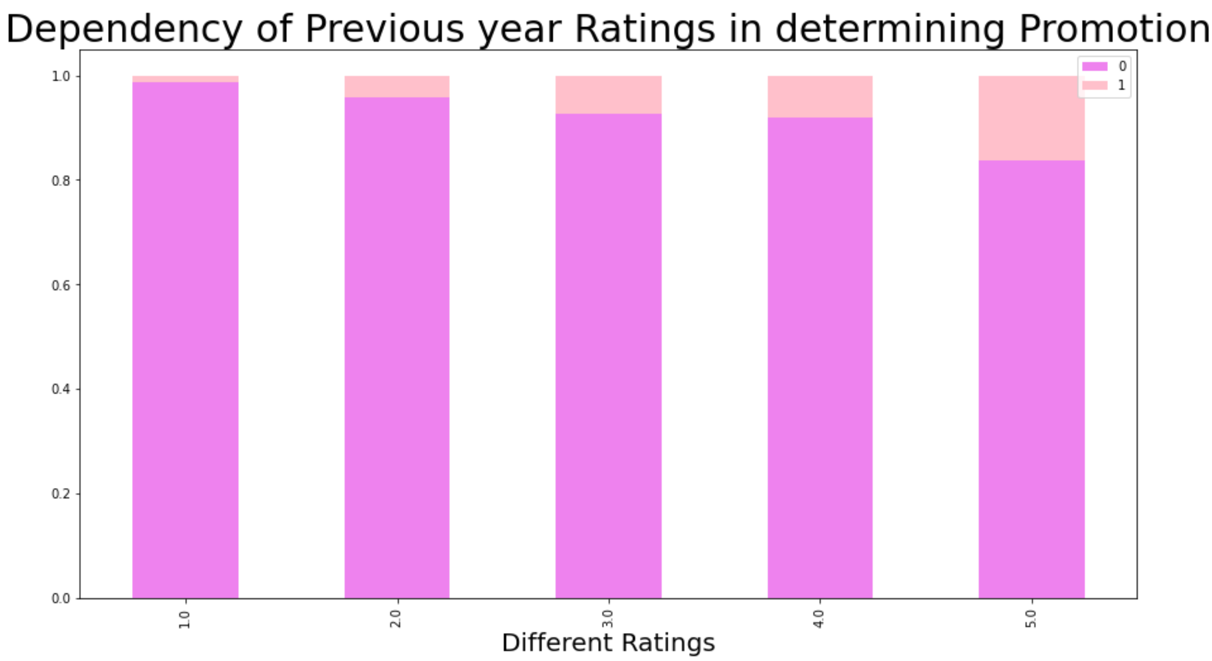
In [ ]:

```
#dependency of KPIs with Promotion

data = pd.crosstab(train['KPIs_met >80%'], train['is_promoted'])
data.div(data.sum(1).astype('float'), axis = 0).plot(kind = 'bar', stacked = True, f

plt.title('Dependency of KPIs in determining Promotion', fontsize = 30)
plt.xlabel('KPIs Met or Not', fontsize = 20)
plt.legend()
plt.show()
```



In [ ]:

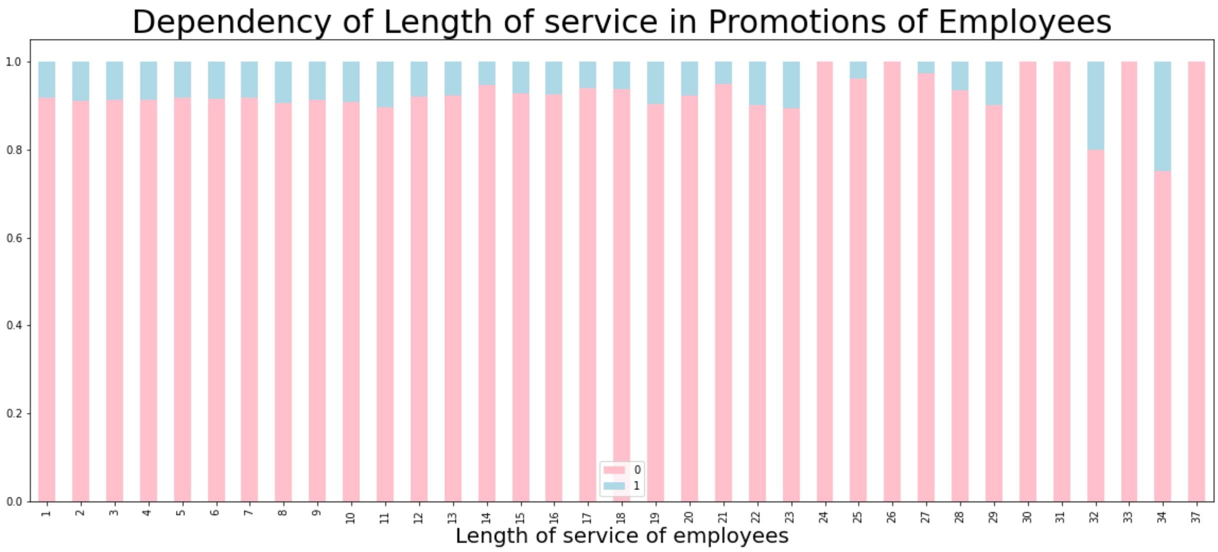
```
# checking dependency on previous years' ratings

data = pd.crosstab(train['previous_year_rating'], train['is_promoted'])
data.div(data.sum(1).astype('float'), axis = 0).plot(kind = 'bar', stacked = True, f

plt.title('Dependency of Previous year Ratings in determining Promotion', fontsize =
plt.xlabel('Different Ratings', fontsize = 20)
plt.legend()
plt.show()
```



In [ ]:

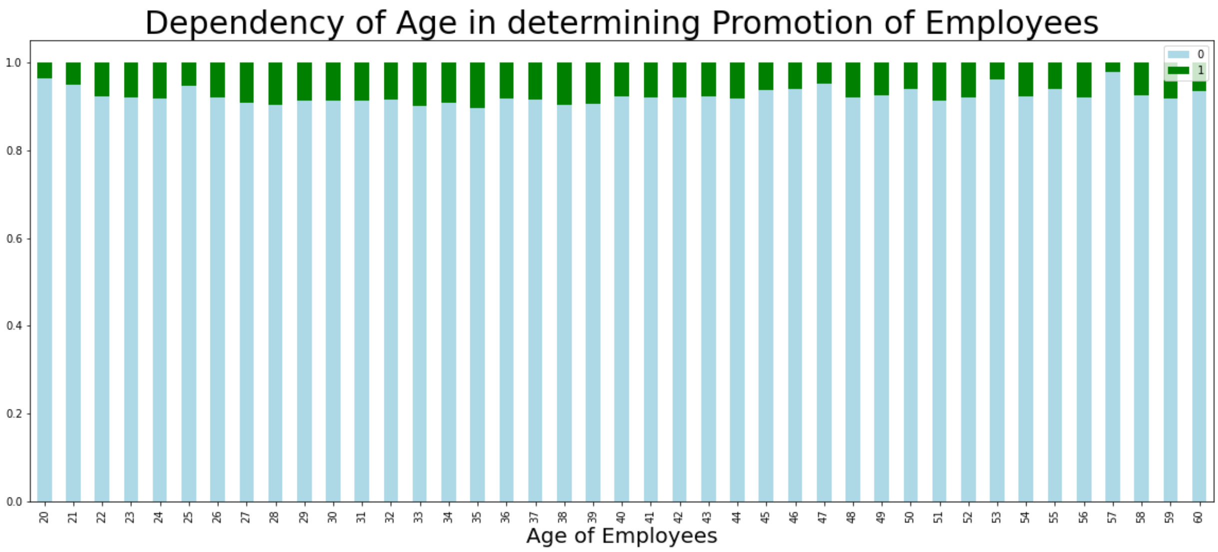
```
# checking how length of service determines the promotion of employees

data = pd.crosstab(train['length_of_service'], train['is_promoted'])
```

```
data.div(data.sum(1).astype('float'), axis = 0).plot(kind = 'bar', stacked = True, f

plt.title('Dependency of Length of service in Promotions of Employees', fontsize = 3
plt.xlabel('Length of service of employees', fontsize = 20)
plt.legend()
plt.show()
```



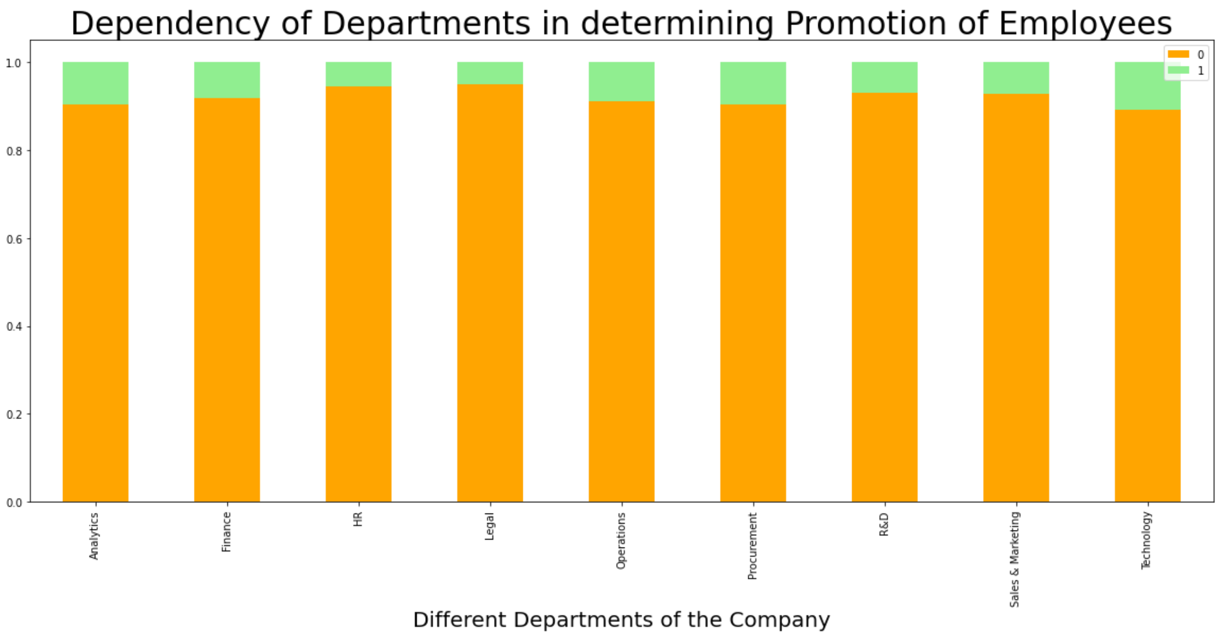
In [ ]:

```
# checking dependency of age factor in promotion of employees

data = pd.crosstab(train['age'], train['is_promoted'])
data.div(data.sum(1).astype('float'), axis = 0).plot(kind = 'bar', stacked = True, f

plt.title('Dependency of Age in determining Promotion of Employees', fontsize = 30)
plt.xlabel('Age of Employees', fontsize = 20)
plt.legend()
plt.show()
```



In [ ]:

```
# checking which department got most number of promotions

data = pd.crosstab(train['department'], train['is_promoted'])
data.div(data.sum(1).astype('float'), axis = 0).plot(kind = 'bar', stacked = True, f

plt.title('Dependency of Departments in determining Promotion of Employees', fontsiz
plt.xlabel('Different Departments of the Company', fontsize = 20)
plt.legend()
plt.show()
```



In [ ]:

```
# checking dependency of gender over promotion

data = pd.crosstab(train['gender'], train['is_promoted'])
data.div(data.sum(1).astype('float'), axis = 0).plot(kind = 'bar', stacked = True, f

plt.title('Dependency of Genders in determining Promotion of Employees', fontsize =
plt.xlabel('Gender', fontsize = 20)
plt.legend()
```

Out[ ]: <matplotlib.legend.Legend at 0x1b7f4c33610>