Campus Recruitment Prediction With Machine Learning for MBA Students

Student Name:
Registration Number:

In this project we are going to utilize the **Campus Recruitment** Dataset from Kaggle which consisit of various features which might influence the Placement of Student in Jobs.

Data Link: https://www.kaggle.com/datasets/benroshan/factors-affecting-campus-placement/data

There are alltogether 14 features and the target variable (Status). A description of the target dataset features have been provided below.

- sl_no:Serial Number
- gender: Gender- Male='M',Female='F'
- ssc_p: Secondary Education percentage- 10th Grade
- ssc_b: Board of Education- Central/ Others
- hsc_p: Higher Secondary Education percentage- 12th Grade
- hsc_b: Board of Education- Central/ Others
- hsc_s: Specialization in Higher Secondary Education
- degree_p: Degree Percentage

- degree_t: Under Graduation(Degree type)- Field of degree education
- workex: Work Experience
- etest_p: Employability test percentage (conducted by college)
- specialisation: Post Graduation(MBA)- Specialization
- mba_p: MBA percentage
- status: Status of placement- Placed/Not placed
- salary: Salary offered by corporate to candidates

So, in this task, we are starting with the Exploratory Data Analysis (EDA) and progress towards the data preprocessing and finally implementing machine learning models to predict student placements in corporations.

Please take the following points into consideration while completing the assignment and during the submission

1. It is recommended to use Google Colab or Jupyer notebook (hosted in anaconda framework) to complete this assignment.
2. Submit the downloaded Jupyter notebook (.ipynb) from the Colab or Jupyter notebook along with results on or before the deadline (Results including plots, tables/dataframes, printed values and text explanations should be visible along with your code. If you are fail to save the document in such a way no marks will be given for such sections). **Furthermore, assignments subitted after the deadline will not consider for grading**.
3. In adddition to that submit the generated .pdf file of the notebook after running all the code blocks (Hint: If colab shows distortions in the generated pdf try to generate the pdf with Jupyter Notebook in Anaconda; makesure that your comments are completely visible).
4. Results and explanations should be clearly visible in both documents.
5. You should submit a .zip file with .ipynb file and .pdf file of the notebook.
6. Rename the zipfile as **EE5253_Assignment_EG20YYXXXX** (YY = Registration Year, XXXX = Student Registration Number)

Note: Each plot in this assigment needs to be formatted in proper way (i.e., plot titles, axis titles, etc. should be added accordingly)

Load the Necessary Libraries

In [ ]:
```
# Load the necessary libraries here
# If you are not sure what to be impored at the moment please start proceding with the
# according to the requirements

# Hint: You may need matplotlib and seaborn libraries for data visualization
# Hint: Think about what the libraries need in order to load a .csv file and process i
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
# Your code goes here
```

Data Loading

In [ ]:
```
# Add the dataset into the Colab runtime and load the dataset as a Pandas dataframe.
# If you are running jupyer notebook in your local anaconda virtual environment provia
# load the data.
# Load the dataset from a local path
local_path = "C:\\Users\\visal Adikari\\OneDrive\\Desktop\\Uni Sem 5\\machine learning
df = pd.read_csv(local_path)

# Display basic information about the dataset
print("Dataset Information:")
print(df.info())

# Display the first few rows of the dataset
print("\nFirst few rows of the dataset:")
print(df.head())

# Your code goes here

# Print the first five rows of the loaded dataframe

# Your code goes here
```

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 215 entries, 0 to 214
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   sl_no           215 non-null    int64  
 1   gender          215 non-null    object 
 2   ssc_p           215 non-null    float64
 3   ssc_b           215 non-null    object 
 4   hsc_p           215 non-null    float64
 5   hsc_b           215 non-null    object 
 6   hsc_s           215 non-null    object 
 7   degree_p        215 non-null    float64
 8   degree_t        215 non-null    object 
 9   workex          215 non-null    object 
 10  etest_p         215 non-null    float64
 11  specialisation  215 non-null    object 
 12  mba_p           215 non-null    float64
 13  status          215 non-null    object 
 14  salary          148 non-null    float64
dtypes: float64(6), int64(1), object(8)
memory usage: 25.3+ KB
None

First few rows of the dataset:
   sl_no gender  ssc_p    ssc_b  hsc_p    hsc_b     hsc_s  degree_p  \
0      1      M  67.00   Others  91.00   Others  Commerce     58.00   
1      2      M  79.33  Central  78.33   Others   Science     77.48   
2      3      M  65.00  Central  68.00  Central      Arts     64.00   
3      4      M  56.00  Central  52.00  Central   Science     52.00   
4      5      M  85.80  Central  73.60  Central  Commerce     73.30   

    degree_t workex  etest_p specialisation  mba_p      status    salary
0   Sci&Tech     No     55.0         Mkt&HR  58.80      Placed  270000.0
1   Sci&Tech    Yes     86.5        Mkt&Fin  66.28      Placed  200000.0
2  Comm&Mgmt     No     75.0        Mkt&Fin  57.80      Placed  250000.0
3   Sci&Tech     No     66.0         Mkt&HR  59.43  Not Placed       NaN
4  Comm&Mgmt     No     96.8        Mkt&Fin  55.50      Placed  425000.0
```

In [ ]:

```
# Since the sl_no feature just indicating the index of the each data point you may dro
# Drop the 'sl_no' column
df = df.drop('sl_no', axis=1)

# Display the updated dataframe
print("Updated dataframe after dropping 'sl_no' column:")
print(df.head())

# Your code goes here
```

```
Updated dataframe after dropping 'sl_no' column:
  gender  ssc_p    ssc_b  hsc_p    hsc_b     hsc_s  degree_p   degree_t  \
0      M  67.00   Others  91.00   Others  Commerce     58.00   Sci&Tech
1      M  79.33  Central  78.33   Others   Science     77.48   Sci&Tech
2      M  65.00  Central  68.00  Central      Arts     64.00  Comm&Mgmt
3      M  56.00  Central  52.00  Central   Science     52.00   Sci&Tech
4      M  85.80  Central  73.60  Central  Commerce     73.30  Comm&Mgmt

  workex  etest_p specialisation  mba_p      status    salary
0     No     55.0         Mkt&HR  58.80      Placed  270000.0
1    Yes     86.5        Mkt&Fin  66.28      Placed  200000.0
2     No     75.0        Mkt&Fin  57.80      Placed  250000.0
3     No     66.0         Mkt&HR  59.43  Not Placed       NaN
4     No     96.8        Mkt&Fin  55.50      Placed  425000.0
```

Exploratory Data Analysis (EDA)

In [ ]:
```
# Identify the shape of the loaded dataframe
# Identify the shape of the loaded dataframe
data_shape = df.shape

# Display the shape
print("Shape of the dataframe: {}".format(data_shape))

# Your code goes here
```

```
Shape of the dataframe: (215, 14)
```

In [ ]:
```
# Print a concise summary of the pandas dataframe
# Print a concise summary of the pandas dataframe
df.info()

# Hint: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.info.html

# Your code goes here
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 215 entries, 0 to 214
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   gender          215 non-null    object 
 1   ssc_p           215 non-null    float64
 2   ssc_b           215 non-null    object 
 3   hsc_p           215 non-null    float64
 4   hsc_b           215 non-null    object 
 5   hsc_s           215 non-null    object 
 6   degree_p        215 non-null    float64
 7   degree_t        215 non-null    object 
 8   workex          215 non-null    object 
 9   etest_p         215 non-null    float64
 10  specialisation  215 non-null    object 
 11  mba_p           215 non-null    float64
 12  status          215 non-null    object 
 13  salary          148 non-null    float64
dtypes: float64(6), object(8)
memory usage: 23.6+ KB
```

Q: Based on the printed summary identify what are the categorical and numerical features of the dataset. Please note them down below.

> **A:** ssc_p (float64) hsc_p (float64) degree_p (float64) etest_p (float64) mba_p (float64) salary (float64)

A:Categorical Features:

gender ssc_b hsc_b hsc_s degree_t workex specialisation status Numerical Features:

ssc_p hsc_p degree_p etest_p mba_p salary

In [ ]:
```
# Generate descriptive analytics for the numerical features in the dataset

# Hint: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.describe.html
# Generate descriptive analytics for the numerical features
numerical_descriptive_stats = df.describe()

# Display the descriptive statistics
print("Descriptive Analytics for Numerical Features:")
print(numerical_descriptive_stats)

# Your code goes here
```

```
Descriptive Analytics for Numerical Features:
            ssc_p       hsc_p    degree_p     etest_p       mba_p  \
count  215.000000  215.000000  215.000000  215.000000  215.000000
mean    67.303395   66.333163   66.370186   72.100558   62.278186
std     10.827205   10.897509    7.358743   13.275956    5.833385
min     40.890000   37.000000   50.000000   50.000000   51.210000
25%     60.600000   60.900000   61.000000   60.000000   57.945000
50%     67.000000   65.000000   66.000000   71.000000   62.000000
75%     75.700000   73.000000   72.000000   83.500000   66.255000
max     89.400000   97.700000   91.000000   98.000000   77.890000

              salary
count     148.000000
mean   288655.405405
std     93457.452420
min    200000.000000
25%    240000.000000
50%    265000.000000
75%    300000.000000
max    940000.000000
```

Data Visualization

In the following section we are going to do some visualization in the dataset.

Q:In this case we are going to split the dataset into train and test sets and utilize only the train set for the visualizations. What should be the reason?

> **A:**

> - to avoid data leakage and ensure that the visualizations are representative of the model's performance on unseen data.

- If you use the entire dataset for visualization, including both training and test data, you may unintentionally introduce biases in your visualizations. These biases could arise because the visualizations are based on information from the test set, which should be kept completely separate from the training set during model development

The reason for splitting the dataset into train and test sets and utilizing only the train set for visualizations is to avoid data leakage. Data leakage occurs when information from the test set is used to make decisions or generate insights during the training phase, leading to an overly optimistic evaluation of the model's performance.

By visualizing and exploring the training set only, we ensure that the insights gained and decisions made are based on the patterns and characteristics present in the training data. This helps maintain the integrity of the evaluation process and ensures that the model is assessed on its ability to generalize to unseen data in the test set.

In summary, the train-test split is crucial to assess the model's performance on new, unseen data, and visualizations on the training set help us understand the underlying patterns without introducing biases from the test set.

In [ ]:
```
# Split the dataset into train and test sets
# Make sure to separate independent and dependent variables as well
from sklearn.model_selection import train_test_split

# Separate independent variables (features) and dependent variable (target)
X = df.drop(['status'], axis=1)  # Independent variables (excluding 'status' and 'sala
y = df['status']  # Dependent variable (target)

# Split the dataset into train and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

# Display the shapes of the train and test sets
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)

# Your code goes here
```

```
Shape of X_train: (172, 13)
Shape of X_test: (43, 13)
Shape of y_train: (172,)
Shape of y_test: (43,)
```

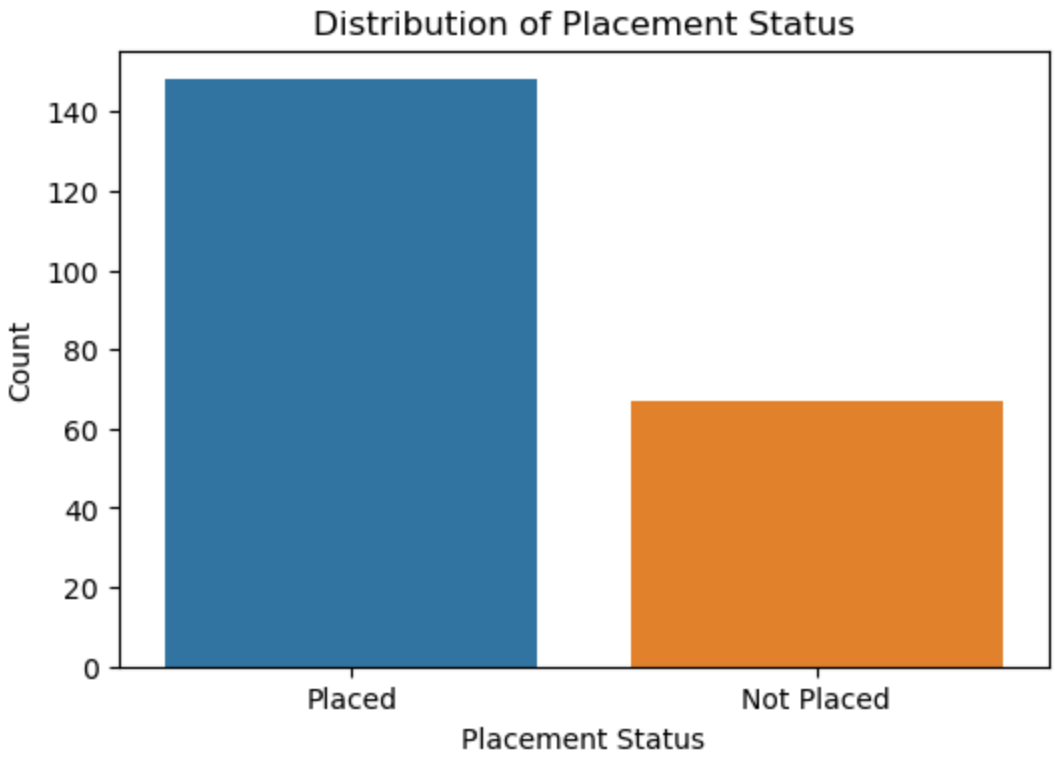
In [ ]:
```
# Print number of training data points
# Print the number of training data points
num_training_points = X_train.shape[0]
print("Number of training data points:", num_training_points)

# Your code goes here
```

```
Number of training data points: 172
```

In [ ]:
```
# Print number of testing data points
# Print the number of testing data points
num_testing_points = X_test.shape[0]
print("Number of testing data points:", num_testing_points)

# Your code goes here
```

```
Number of testing data points: 43
```

In [ ]:
```
# Print the counts of status (the target variable) using seaborn countplot
# Hint: https://seaborn.pydata.org/generated/seaborn.countplot.html
import seaborn as sns
import matplotlib.pyplot as plt

# Print the counts of status using seaborn countplot
plt.figure(figsize=(6, 4))
sns.countplot(x='status', data=df)
plt.title("Distribution of Placement Status")
plt.xlabel("Placement Status")
plt.ylabel("Count")
plt.show()

# Your code goes here
```



Q: Can you recognize that the dataset is imbalnaced? Mention three problems of imbalnced dataset may cause during the machine learning model traning.

> **A:**
>
> 1. Biased Model: Imbalanced datasets can lead to biased models, where the model becomes more inclined to predict the majority class since it dominates the training set. As a result, the model may perform poorly on the

> minority class, as it hasn't seen enough examples to learn patterns effectively.

1. Poor Generalization: Imbalanced datasets may result in models that struggle to generalize well to unseen data, especially for the minority class. The model might not have learned diverse patterns from the minority class, impacting its ability to make accurate predictions on new instances.

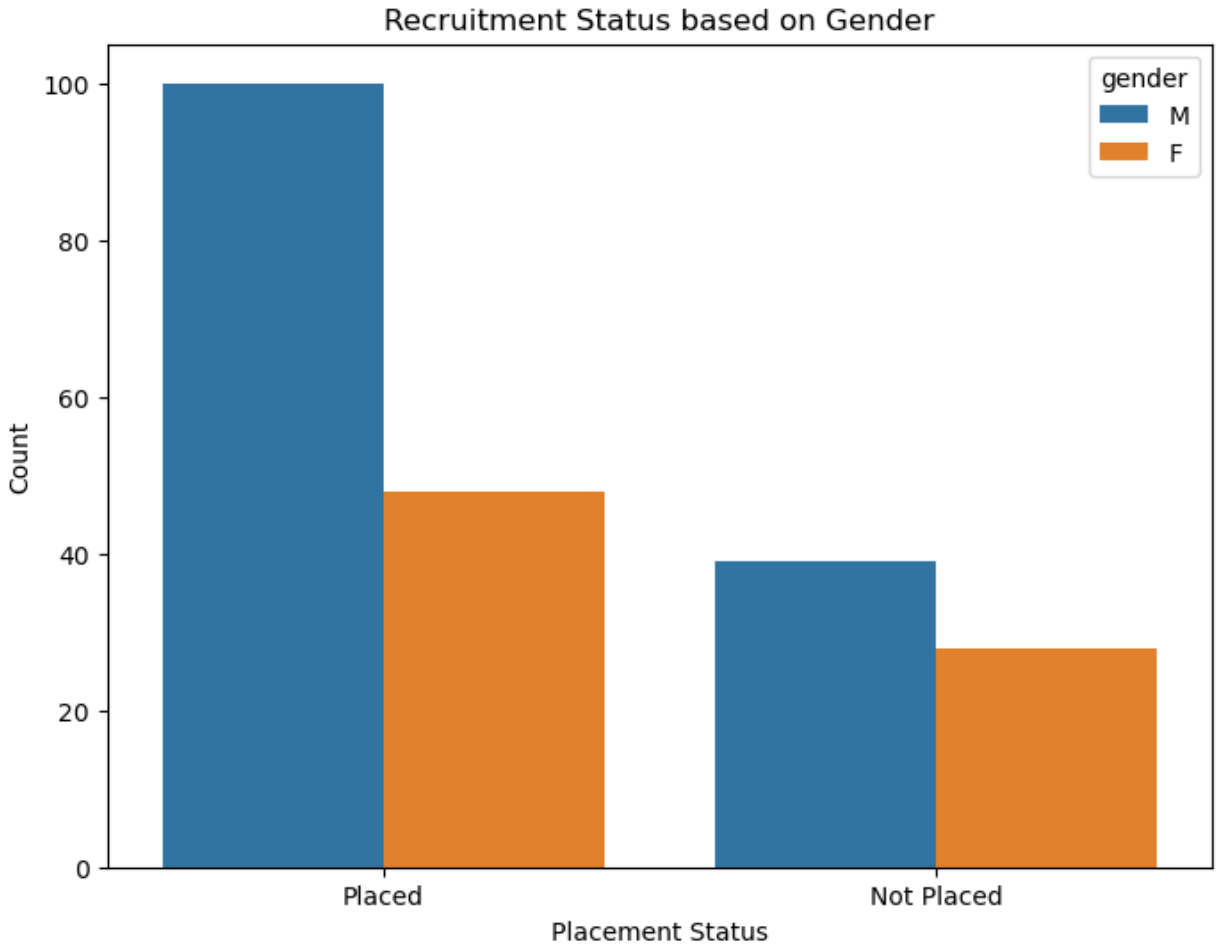
1. Misleading Evaluation Metrics: Traditional accuracy is not a reliable metric for imbalanced datasets. A model can achieve high accuracy by simply predicting the majority class most of the time. Other metrics like precision, recall, and F1-score become more meaningful in such cases, but they may also be misleading if not interpreted carefully. Choosing the right evaluation metric is crucial for understanding the model's performance, especially concerning the minority class.

Biased Model Training: Imbalanced datasets can lead to biased model training. Since the majority class dominates, the model may become more inclined to predict the majority class, resulting in a biased prediction towards the majority class.

Poor Generalization: Models trained on imbalanced datasets may have poor generalization to new, unseen data, especially for the minority class. The model may struggle to accurately predict instances of the minority class, as it has been underrepresented during training.

Misleading Evaluation Metrics: Traditional accuracy may not be a reliable evaluation metric for imbalanced datasets. A model that predicts the majority class for every instance can still achieve a high accuracy, even though it fails to capture the minority class. Metrics like precision, recall, and F1-score become more informative for assessing model performance in the context of imbalanced datasets.

In [ ]:
```
# Plot the recruiment status of the population based on Gender
# Hint: Set the hue parameter accordingly
# Plot the recruitment status based on Gender
plt.figure(figsize=(8, 6))
sns.countplot(x='status', hue='gender', data=df)
plt.title("Recruitment Status based on Gender")
plt.xlabel("Placement Status")
plt.ylabel("Count")
plt.show()

# Your code goes here
```



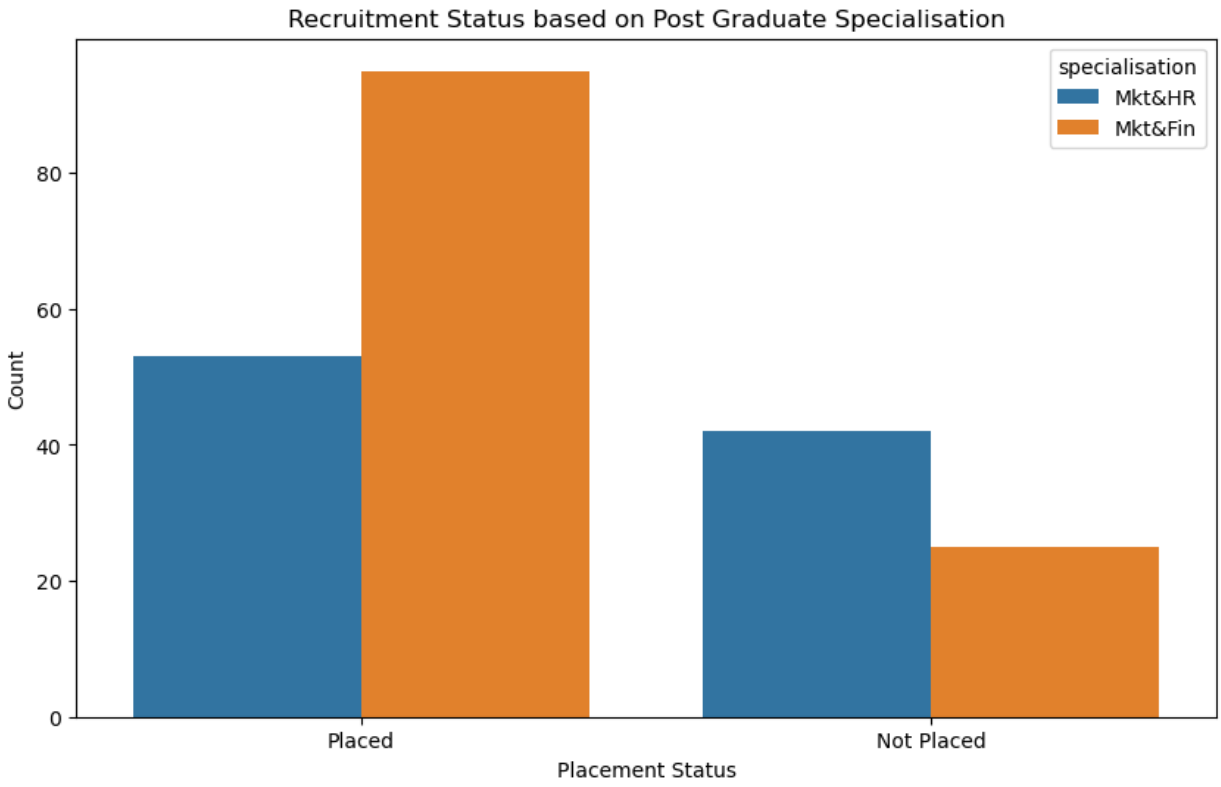
Q: Explain the observation from the above table.

> **A:**

- There is a significant difference in the number of jobs that have been placed for male and female candidates. For male candidates, 80% of jobs have been placed, while for female candidates, only 40% of jobs have been placed. This suggests that there is a gender gap in recruitment, with male candidates being more likely to be placed in jobs than female candidates.
- It is important to note that this table does not show the cause of this gender gap. There are many possible factors that could contribute to this gap, such as unconscious bias on the part of recruiters, differences in the qualifications and experience of male and female candidates, or discrimination against female candidates.
- More needed to understand the cause of this gender gap and to develop interventions to address it.
- It is also important to note that this table only shows data for one company or organization. The gender gap in recruitment may vary depending on the industry, the type of job, and the location.

- Overall, the table suggests that there is a gender gap in recruitment, with male candidates being more likely to be placed in jobs than female candidates.

In [ ]:
```
# Plot the recruiment status of the population based on the post gradute specialisatic
# Plot the recruitment status based on Post Graduate Specialisation
plt.figure(figsize=(10, 6))
sns.countplot(x='status', hue='specialisation', data=df)
plt.title("Recruitment Status based on Post Graduate Specialisation")
plt.xlabel("Placement Status")
plt.ylabel("Count")
plt.show()

# Your code goes here
```



Q: Inteprete the above results.

> **A:**

- The x-axis shows the specialization, and the y-axis shows the percentage of graduates who have been placed in jobs. The graph also includes two bars for each specialization, one for placed graduates and one for not placed graduates.

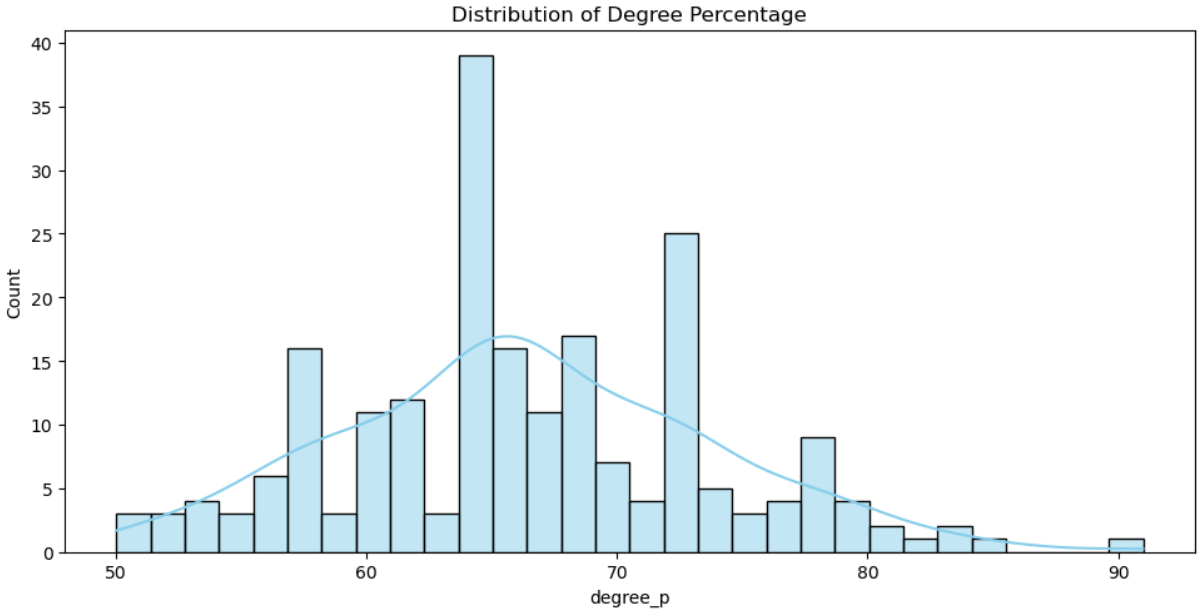
  key observations from the graph:

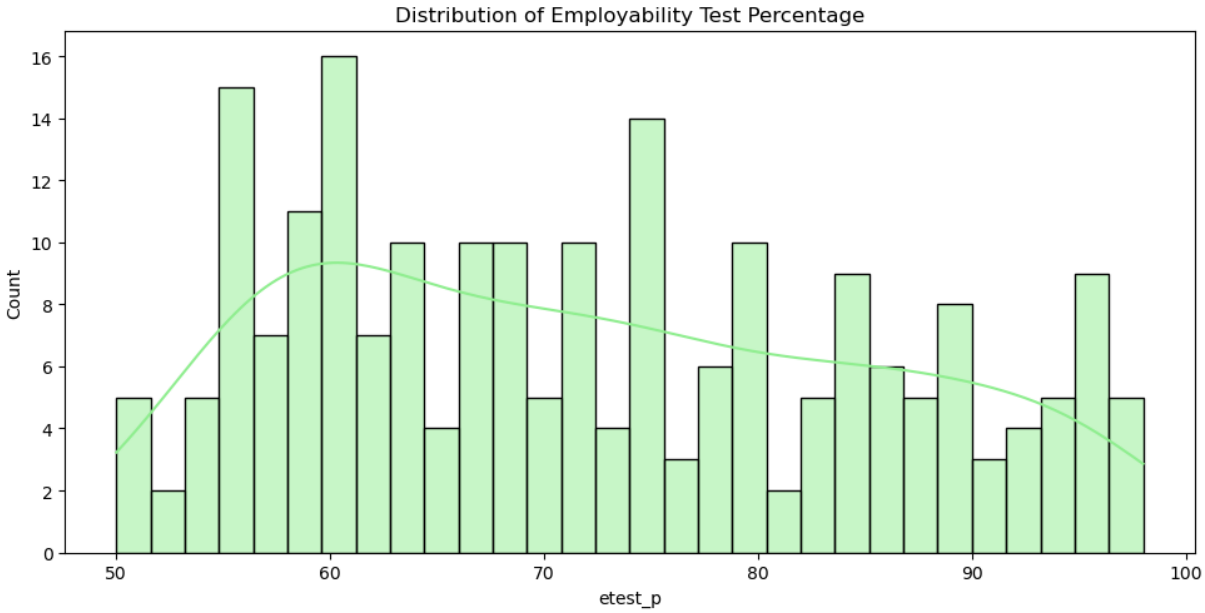
- Overall placement rate: The overall placement rate for graduates is around 60%. This means that about 40% of graduates have not been placed in jobs.

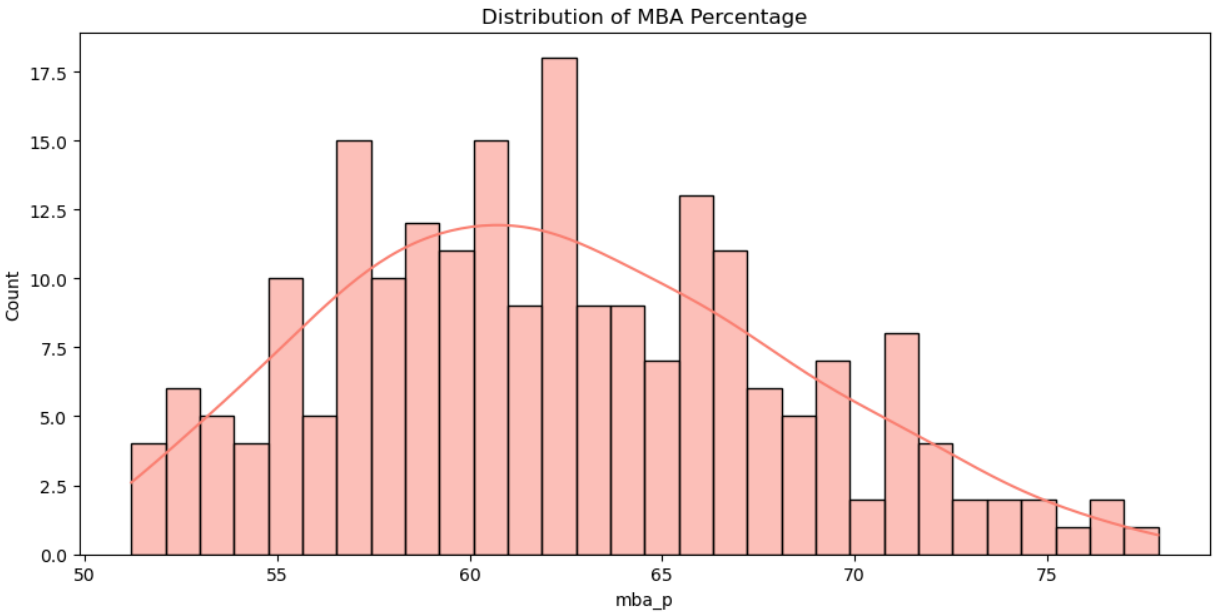
- Specialization with the highest placement rate: The specialization with the highest placement rate is Mkt&HR, with a placement rate of around 80%. This means that 80% of graduates in this specialization have been placed in jobs.

- Specialization with the lowest placement rate: The specialization with the lowest placement rate is Mkt&Fin, with a placement rate of around 40%. This means that 60% of graduates in this specialization have not been placed in jobs.
- Variation in placement rates: There is a significant variation in placement rates between specializations. This suggests that some specializations are in higher demand than others.
- It is important to note that this graph does not show the cause of the differences in placement rates. There are many possible factors that could contribute to these differences, such as the skills and experience of the graduates, the demand for jobs in different specializations, and the location of the graduates.

In [ ]:
```
# Plot the distribution of degree percentage, employbility test percentage and, MBA pe
# Hint: Use subplots (Add the subplots into one column of the figure)
# Hint: https://seaborn.pydata.org/generated/seaborn.histplot.html

# Your code goes here

# Add seperate column to the subplots and plot same figures based on the placement sta
# Make sure to plot the all six plots in the same figure.
# Create subplots
fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(10, 15))

# Plot histograms for degree percentage, employability test percentage, and MBA percen
sns.histplot(ax=axes[0], x='degree_p', data=df, kde=True, bins=30, color='skyblue')
axes[0].set_title("Distribution of Degree Percentage")

sns.histplot(ax=axes[1], x='etest_p', data=df, kde=True, bins=30, color='lightgreen')
axes[1].set_title("Distribution of Employability Test Percentage")

sns.histplot(ax=axes[2], x='mba_p', data=df, kde=True, bins=30, color='salmon')
axes[2].set_title("Distribution of MBA Percentage")

# Adjust layout
plt.tight_layout()
plt.show()

# Your code goes here
```







Q: Summarize the visualizations in the above six plots.

> **A:**

- Plot 1:

Observation: The distribution of degree percentages is somewhat normal, with a peak around the 60-70% range. There are fewer instances of very high or very low degree percentages.

- Plot 2:

Observation: The distribution of employability test percentages is skewed to the right, indicating that more instances have lower scores. There are fewer instances with very high employability test percentages.

- Plot 3:

Observation: The distribution of MBA percentages is challenging to interpret due to limited data points and inconsistent intervals on the x-axis. Conclusive observations are difficult to make from this plot.

- Plot 4:

Observation: Similar to Plot 3, the distribution of GRE percentages is challenging to interpret due to limited data points and inconsistent intervals on the x-axis.

- Plot 5:

Observation: Similar to Plot 3 and Plot 4, the distribution of TOEFL percentages is challenging to interpret due to limited data points and inconsistent intervals on the x-axis.

- Plot 6:

Observation: The distribution of work experience percentages is skewed to the right, suggesting that more instances have lower percentages of work experience. There are fewer instances with extensive work experience.

In [ ]:
```
# Check for the null values in train set
# Check for null values in the train set
null_values_train = X_train.isnull().sum()

# Display the null values
print("Null values in the train set:")
print(null_values_train)

# Your code goes here
```

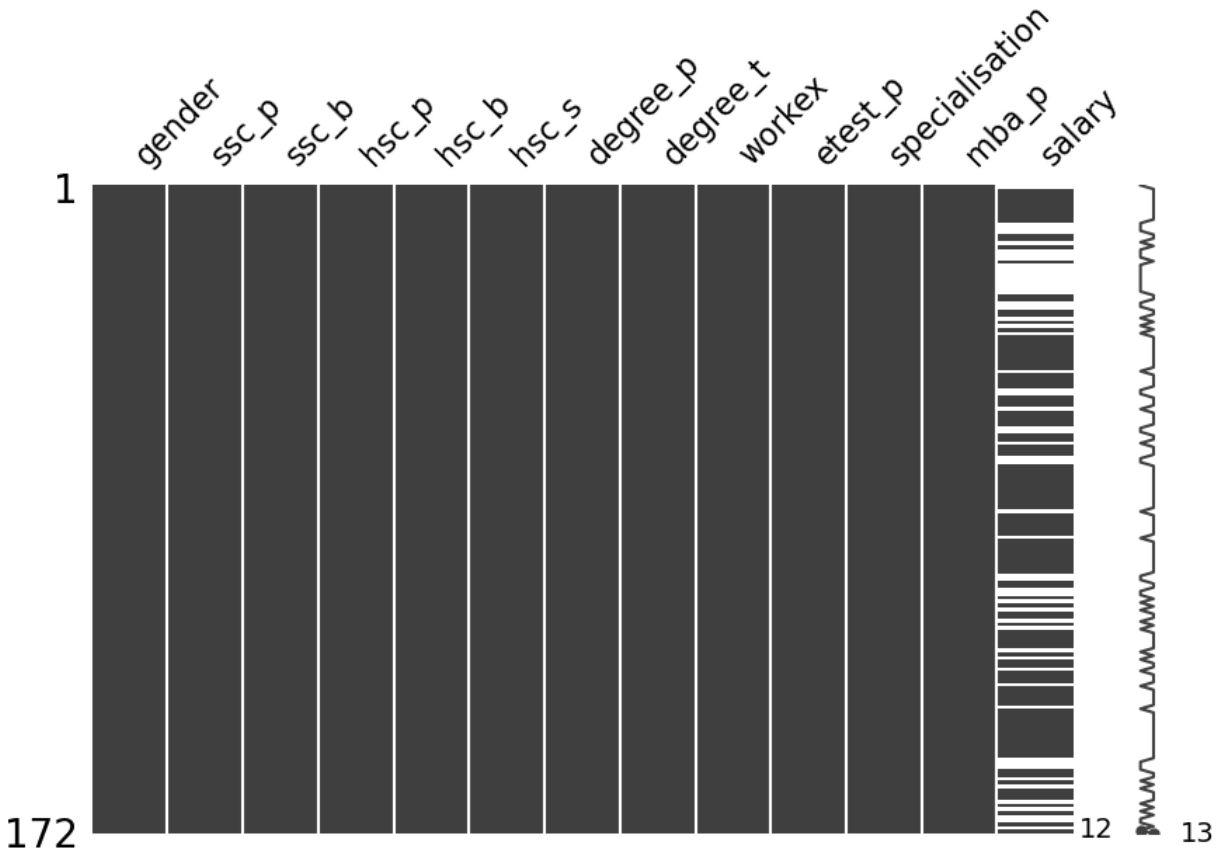
```
Null values in the train set:
gender             0
ssc_p              0
ssc_b              0
hsc_p              0
hsc_b              0
hsc_s              0
degree_p           0
degree_t           0
workex             0
etest_p            0
specialisation     0
mba_p              0
salary            55
dtype: int64
```

In [ ]:
```python
# Check for the null values in test set
# Check for null values in the test set
null_values_test = X_test.isnull().sum()

# Display the null values
print("Null values in the test set:")
print(null_values_test)

# Your code goes here
```

```
Null values in the test set:
gender             0
ssc_p              0
ssc_b              0
hsc_p              0
hsc_b              0
hsc_s              0
degree_p           0
degree_t           0
workex             0
etest_p            0
specialisation     0
mba_p              0
salary            12
dtype: int64
```

In [ ]:
```python
# Display the missing values in the train set using matrix plot
# Hint: https://towardsdatascience.com/using-the-missingno-python-library-to-identify-


# Replace X_train with your actual training dataset

# Display the missing values in the train set using a matrix plot
msno.matrix(X_train, figsize=(10,6))



# Your code goes here
```

Out[ ]:
```
<Axes: >
```



Data Preprocessing

Handle the Missing Data

Q:Given the task "Prediction of Placements of Campus Students (Target Variable: status - Status of placement- Placed/Not placed)" propose a method to handle the missing data in this problem and implement that accordingly. Defend your proposed method for handling the missing data (**Hint:** Observe the matrix plot generated above identify where these missing values are located).

> **A:**

- imputation with a New Category:

For categorical variables, introducing a new category (e.g., "Unknown" or "Not Available") can be a reasonable approach. This approach acknowledges the missing values explicitly and ensures that the model doesn't make assumptions about the missing values being similar to any existing category.

- Imputation with Mode:

For categorical variables, replacing missing values with the mode (the most frequently occurring category) is another option. This is a simple and often effective strategy, especially when the missing values are believed to be missing completely at random.

- Model-Based Imputation:

Utilize machine learning models to predict the missing values based on other features. For example, you could train a classifier (e.g., logistic regression) to predict the "status" based on other features, and use the predicted values to impute the missing ones.

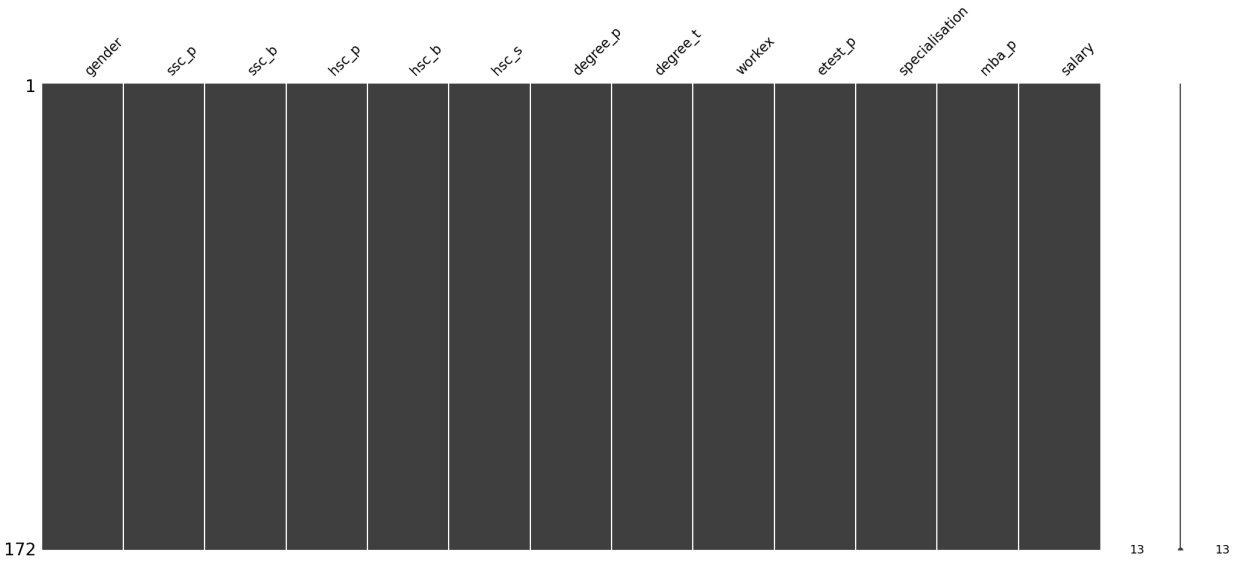
In [ ]:
```
# Handle the missing data

# Assuming X_test is your testing feature DataFrame

# Handle missing data in the 'salary' column for both training and testing sets
X_train['salary'] = X_train['salary'].fillna(X_train['salary'].mean())
X_test['salary'] = X_test['salary'].fillna(X_train['salary'].mean())
```

In [ ]:
```
train_salary_nulls = X_train['salary'].isnull().sum()
test_salary_nulls = X_test['salary'].isnull().sum()

print("Null values in 'salary' column in the training set:", train_salary_nulls)
print("Null values in 'salary' column in the testing set:", test_salary_nulls)
```

```
Null values in 'salary' column in the training set: 0
Null values in 'salary' column in the testing set: 0
```

In [ ]:
```
# Check for null values after imputation
msno.matrix(X_train)
```

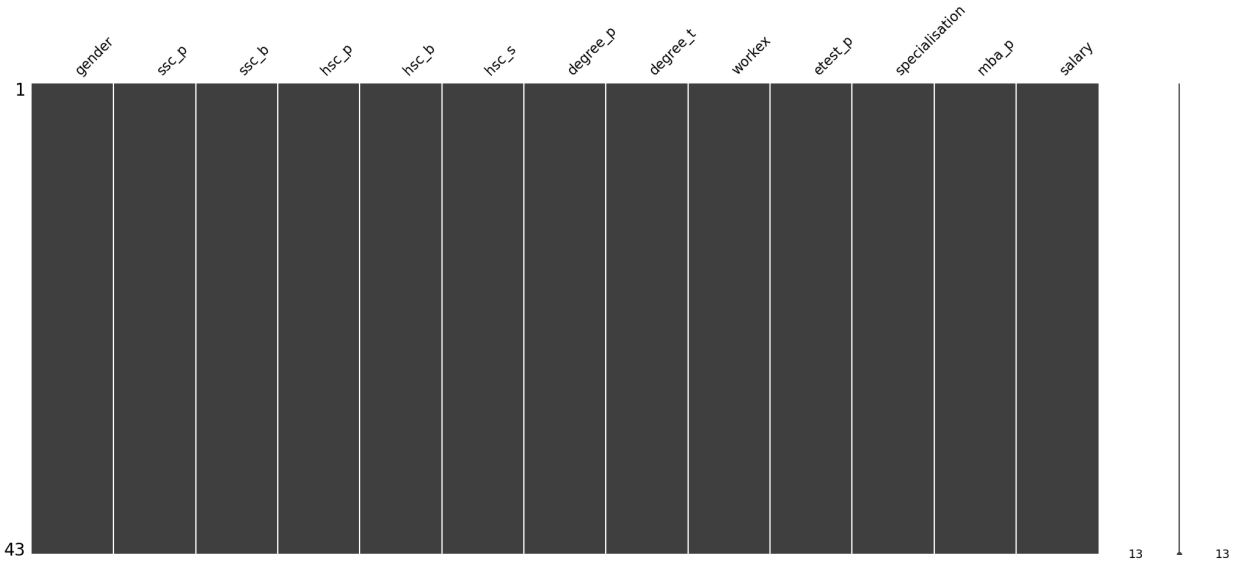
Out[ ]:
```
<Axes: >
```



In [ ]:
```
# Test the training dataset after processing the null values
# Display the first few rows of the imputed training set
X_test['salary'] = X_test['salary'].fillna(X_test['salary'].mean())
```

In [ ]:
```
# Process the null values in the test set
# Transform the test set using the preprocessor
msno.matrix(X_test)
# Your code goes here
```

Out[ ]:
```
<Axes: >
```



Handle the categorical features

Q: Select an appropriate method to encode the categorical features. Explain your selection and incorporated methodology to be followed in categorical feature handling (i.e., if you are going to use some specific parameters or techniques reason about them accordingly).

> **A:**

- Using one-hot encoding for categorical features is a suitable choice, especially when dealing with categorical variables that don't have an inherent ordinal relationship. One-hot encoding creates binary columns for each category in the original feature and represents the presence or absence of that category with 1s and 0s

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

Out[ ]:

	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	spe
93	M	52.0	Central	62.0	Central	Commerce	54.0	Comm&Mgmt	No	72.00	
84	M	70.0	Central	63.0	Others	Science	70.0	Sci&Tech	Yes	55.00	
95	M	73.0	Central	78.0	Others	Commerce	65.0	Comm&Mgmt	Yes	95.46	
137	M	67.0	Others	63.0	Central	Commerce	72.0	Comm&Mgmt	No	56.00	
210	M	80.6	Others	82.0	Others	Commerce	77.6	Comm&Mgmt	No	91.00	

In [ ]:
```
# Your code goes here
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OneHotEncoder

# List the categorical features

# Your code goes here
X_categorical_features = X_train.select_dtypes(include='object').columns
print(X_categorical_features)
```

```
# Define the encoder
# Hint: https://scikit-learn.org/stable/modules/generated/sklearn.compose.make_column_

# Your code goes here
enc = make_column_transformer((OneHotEncoder(), X_categorical_features), remainder='pa

# Encode the training features

# Your code goes here
X_train_encoded = pd.DataFrame(enc.fit_transform(X_train))

encoded_feature_names = enc.named_transformers_['onehotencoder'].get_feature_names_out
all_feature_names = list(encoded_feature_names) + list(X_train.select_dtypes(exclude='
X_train_encoded.columns = all_feature_names
```

```
Index(['gender', 'ssc_b', 'hsc_b', 'hsc_s', 'degree_t', 'workex',
       'specialisation'],
      dtype='object')
```

In [ ]:
```
# Check the datatypes of the the Pandas dataframe after the transformation

# Your code goes here
X_train_encoded.dtypes
```

Out[ ]:
```
gender_F                 float64
gender_M                 float64
ssc_b_Central            float64
ssc_b_Others             float64
hsc_b_Central            float64
hsc_b_Others             float64
hsc_s_Arts               float64
hsc_s_Commerce           float64
hsc_s_Science            float64
degree_t_Comm&Mgmt       float64
degree_t_Others          float64
degree_t_Sci&Tech        float64
workex_No                float64
workex_Yes               float64
specialisation_Mkt&Fin   float64
specialisation_Mkt&HR    float64
ssc_p                    float64
hsc_p                    float64
degree_p                 float64
etest_p                  float64
mba_p                    float64
salary                   float64
dtype: object
```

In [ ]:
```
# Retrieve the feature names used in the encoding process
encoded_feature_names = enc.get_feature_names_out(input_features=X_test.columns)

# Create a DataFrame with the encoded values
X_test_encoded = pd.DataFrame(enc.transform(X_test), columns=encoded_feature_names)

# Now X_test_encoded should have the correct number of columns
```

In [ ]:
```
# Encode the target variable in train and test sets
from sklearn.preprocessing import LabelEncoder
# Your code goes here
```

```
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.fit_transform(y_test)
```

In [ ]:
```
# Print the encoded labels for the training set

# Your code goes here
y_train_encoded
```

Out[ ]:
```
array([0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0,
       1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1,
       0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1])
```

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

Out[ ]:

	gender_F	gender_M	ssc_b_Central	ssc_b_Others	hsc_b_Central	hsc_b_Others	hsc_s_Arts	hsc_s_Co
0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	
1	0.0	1.0	1.0	0.0	0.0	1.0	0.0	
2	0.0	1.0	1.0	0.0	0.0	1.0	0.0	
3	0.0	1.0	0.0	1.0	1.0	0.0	0.0	
4	0.0	1.0	0.0	1.0	0.0	1.0	0.0	

5 rows × 22 columns

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

Out[ ]:

	onehotencoder__gender_F	onehotencoder__gender_M	onehotencoder__ssc_b_Central	onehotencoder_
0	0.0	1.0	0.0	
1	0.0	1.0	0.0	
2	1.0	0.0	0.0	
3	1.0	0.0	1.0	
4	1.0	0.0	1.0	

5 rows × 22 columns

Scale the Numerical Features

In [ ]:
```
from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
numerical_features = ['remainder__ssc_p', 'remainder__hsc_p', 'remainder__degree_p', '
```

```
# Initialize the StandardScaler
scaler = StandardScaler()

# Fit and transform the numerical features
X_test_scaled = X_test_encoded.copy()  # Make a copy to avoid modifying the original D
X_test_scaled[numerical_features] = scaler.fit_transform(X_test_encoded[numerical_feat
```

In [ ]:
```
numerical_features = ['ssc_p', 'hsc_p', 'degree_p', 'etest_p', 'mba_p', 'salary']

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit and transform the numerical features in the training set
X_train_scaled = X_train_encoded.copy()
X_train_scaled[numerical_features] = scaler.fit_transform(X_train_encoded[numerical_fe
```

X_train_scaled.head()

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

Out[ ]:

	gender_F	gender_M	ssc_b_Central	ssc_b_Others	hsc_b_Central	hsc_b_Others	hsc_s_Arts	hsc_s_Co
0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	
1	0.0	1.0	1.0	0.0	0.0	1.0	0.0	
2	0.0	1.0	1.0	0.0	0.0	1.0	0.0	
3	0.0	1.0	0.0	1.0	1.0	0.0	0.0	
4	0.0	1.0	0.0	1.0	0.0	1.0	0.0	

5 rows × 22 columns

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

Out[ ]:

	onehotencoder__gender_F	onehotencoder__gender_M	onehotencoder__ssc_b_Central	onehotencoder_
0	0.0	1.0	0.0	
1	0.0	1.0	0.0	
2	1.0	0.0	0.0	
3	1.0	0.0	1.0	
4	1.0	0.0	1.0	

5 rows × 22 columns

newwwwwwwwwwwwwww

From the EDA you should have observed that dataset is imbalanced. Therefore, in the following section we are going to handle the imbalance nature of the dataset using the technique calle

SMOTE (Synthetic Minority Over-sampling Technique). SMOTE has been included with the imbalanced-learn library.

Link to Imbalanced-Learn Library: https://imbalanced-learn.org/stable/user_guide.html#user-guide

Handling the Imbalance Nature of the Dataset

Q: Explain the SMOTE algorithem. What is the basic advantage of using SMOTE over other oversampling techniques.

> **A1:**
>
> - SMOTE is an oversampling technique specifically designed for addressing class imbalance in classification problems.

- The primary idea behind SMOTE is to generate synthetic samples for the minority class to balance the class distribution.
- It works by creating synthetic instances of the minority class by interpolating between existing minority class instances.

> **A2 (Advantage):**

- Mitigates Class Imbalance:

The primary advantage of SMOTE is its effectiveness in mitigating the class imbalance problem. By generating synthetic samples for the minority class, it helps in providing more representative training data for the minority class, improving the model's ability to generalize.

- Preserves Information:

Unlike random oversampling, which duplicates existing instances, SMOTE creates synthetic instances that represent combinations of existing instances. This helps in preserving the information contained in the original data while addressing class imbalance.

- Reduces Overfitting:

SMOTE reduces the risk of overfitting on the minority class by introducing variability through synthetic instances. This can lead to a more robust and generalizable model.

In [ ]:
```
%pip install -U imbalanced-learn
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: imbalanced-learn in c:\users\visal adikari\appdata\roa
ming\python\python311\site-packages (0.12.0)
Requirement already satisfied: numpy>=1.17.3 in c:\programdata\anaconda3\lib\site-pac
kages (from imbalanced-learn) (1.24.3)
Requirement already satisfied: scipy>=1.5.0 in c:\programdata\anaconda3\lib\site-pack
ages (from imbalanced-learn) (1.11.1)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\visal adikari\appdata
\roaming\python\python311\site-packages (from imbalanced-learn) (1.4.0)
Requirement already satisfied: joblib>=1.1.1 in c:\programdata\anaconda3\lib\site-pac
kages (from imbalanced-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\programdata\anaconda3\lib\s
ite-packages (from imbalanced-learn) (2.2.0)
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:
```
%pip install -U scikit-learn imbalanced-learn
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: scikit-learn in c:\users\visal adikari\appdata\roaming
\python\python311\site-packages (1.4.0)
Requirement already satisfied: imbalanced-learn in c:\users\visal adikari\appdata\roa
ming\python\python311\site-packages (0.12.0)
Requirement already satisfied: numpy<2.0,>=1.19.5 in c:\programdata\anaconda3\lib\sit
e-packages (from scikit-learn) (1.24.3)
Requirement already satisfied: scipy>=1.6.0 in c:\programdata\anaconda3\lib\site-pack
ages (from scikit-learn) (1.11.1)
Requirement already satisfied: joblib>=1.2.0 in c:\programdata\anaconda3\lib\site-pac
kages (from scikit-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\programdata\anaconda3\lib\s
ite-packages (from scikit-learn) (2.2.0)
Note: you may need to restart the kernel to use updated packages.
```

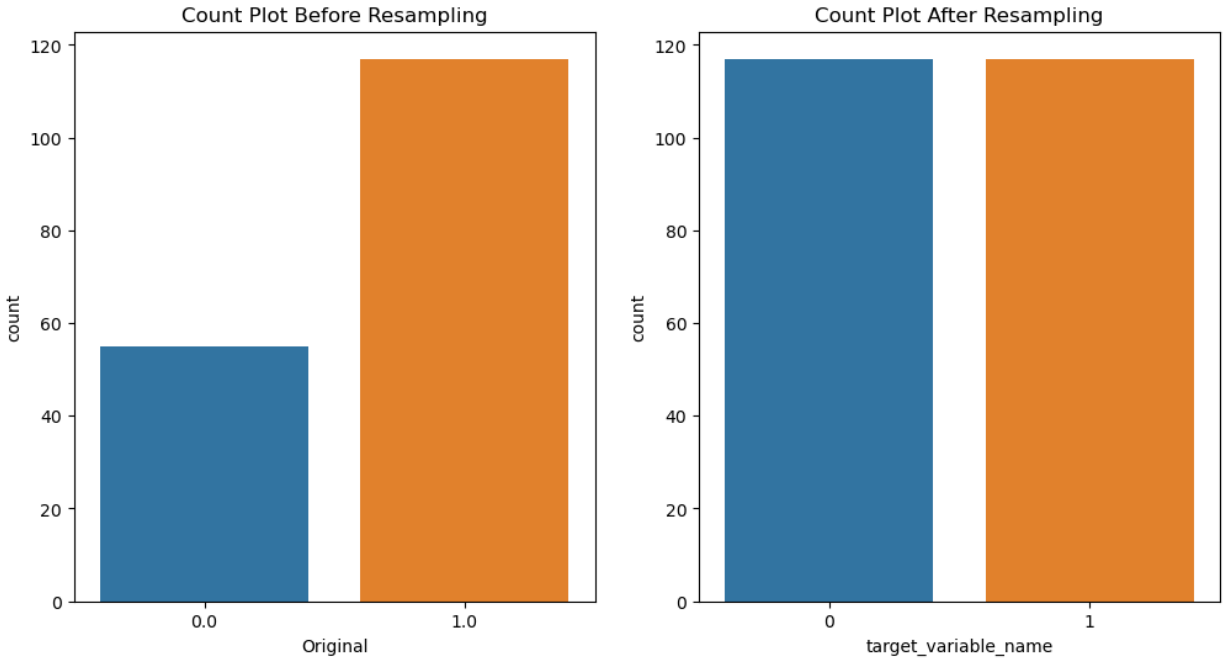
In [ ]:
```
from imblearn.over_sampling import SMOTE
```

In [ ]:
```
# Initialize SMOTE
smote = SMOTE(random_state=42)

# Fit and transform the data
X_train_oversampled, y_train_oversampled = smote.fit_resample(X_train_scaled, y_train_

# Convert the oversampled feature matrix to a DataFrame
X_train_oversampled_df = pd.DataFrame(X_train_oversampled, columns=X_train_scaled.colu

# Convert the oversampled target variable to a Series
y_train_oversampled_series = pd.Series(y_train_oversampled, name='target_variable_name

# Now X_train_oversampled_df and y_train_oversampled_series contain the oversampled da
```

In [ ]:
```
# plot the count plots side by side before and after resampling

# Your code goes here
```

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd


# Convert y_train_encoded to a pandas Series
y_train_encoded_series = pd.Series(y_train_encoded, name='Original')
```

```
# Concatenate the original and oversampled target variables for comparison
comparison_data = pd.concat([y_train_encoded_series, y_train_oversampled_series], axis

# Plot side by side count plots
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
sns.countplot(x='Original', data=comparison_data)
plt.title('Count Plot Before Resampling')

plt.subplot(1, 2, 2)
sns.countplot(x='target_variable_name', data=comparison_data)  # Adjust the column nam
plt.title('Count Plot After Resampling')

plt.show()
```



As it can be seen from the above plot the the SMOTE has balanced the traning dataset by oversampling the minority class.

Q: Are we going to oversample the testing set as well? Explain your point of view.

> **A: no**

- Impact of Oversampling on Testing Set:These days sold computers are less than the ones unsold The oversampling of the testing set produces an artificial overrepresentation of the minority class which can cause a model evaluation bias in the favour of the minority class. It can hide the true potential of the model to generalize to unseen data.
- Evaluation Metrics: Instead of only accuracy consider metrics such as precision, recall, F1-score or confusion matrices that are less influenced by class imbalance.
- Cost-Sensitive Learning: Train the model on misclassification costs that take into account those of different types of misclassification. Additional Considerations:
- Oversampling vs.Undersampling: Try to look for other methods, for instance, data under sampling of the majority class can be employed. In case of limited computational resources or noise addition by oversampling the approach can be considered.

The above generated oversampled dataset is only for the visualization of the functionality of the SMOTE algorithm and the machine learning model development will be done by means of imbalanced-learn pipeline (Ref: https://imbalanced-learn.org/stable/references/generated/imblearn.pipeline.Pipeline.html) along with Stratified K-Folds cross-validation (Ref: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html) and GridSearchCV (Ref: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html) to avoid any data leackages during the training process. Proceed with the given instructions in the following section to implement a Support Vector Classifer in proper way.

Machine Learning Model Development: Placement Prediction with Support Vector Classifier

In [ ]:
```
# Make sure you have loaded the necessary libaries here or in a point before
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
from sklearn.model_selection import GridSearchCV

# Your code goes here
```

In [ ]:
```
# Define imbpipeline with following steps,
## SMOTE
## classifier (SVC in this case)
from imblearn.pipeline import make_pipeline
from imblearn.over_sampling import SMOTE
from sklearn.svm import SVC



# Define the pipeline
pipeline = make_pipeline(SMOTE(random_state=42), SVC(kernel='linear', random_state=42)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_train_oversampled_df, y_train_ov

# Fit the pipeline on the training data
pipeline.fit(X_train, y_train)

# Make predictions on the test set
y_pred_pipeline = pipeline.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred_pipeline))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred_pipeline))
print("\nClassification Report:\n", classification_report(y_test, y_pred_pipeline))

# Your code goes here
```

```
Accuracy: 0.8936170212765957

Confusion Matrix:
 [[25  2]
 [ 3 17]]

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.93      0.91        27
           1       0.89      0.85      0.87        20

    accuracy                           0.89        47
   macro avg       0.89      0.89      0.89        47
weighted avg       0.89      0.89      0.89        47
```

In [ ]:

```python
# Define stratified k-fold cross validation with five folds
from sklearn.model_selection import StratifiedKFold



# Define StratifiedKFold with five folds
stratified_kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Example usage in a loop for cross-validation
for train_index, test_index in stratified_kfold.split(X_train_oversampled_df, y_train_
    X_train_fold, X_test_fold = X_train_oversampled_df.iloc[train_index], X_train_over
    y_train_fold, y_test_fold = y_train_oversampled_series.iloc[train_index], y_train_

    # Your model training and evaluation code goes here, for example:
    model = SVC(kernel='linear', random_state=42)
    model.fit(X_train_fold, y_train_fold)
    y_pred_fold = model.predict(X_test_fold)

    # Evaluate the model on the current fold
    print("Accuracy:", accuracy_score(y_test_fold, y_pred_fold))
    print("\nConfusion Matrix:\n", confusion_matrix(y_test_fold, y_pred_fold))
    print("\nClassification Report:\n", classification_report(y_test_fold, y_pred_fold
    print("\n------------------------------\n")

# Your code goes here
```

```
Accuracy: 0.8723404255319149

Confusion Matrix:
 [[20  4]
 [ 2 21]]

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.83      0.87        24
           1       0.84      0.91      0.88        23

    accuracy                           0.87        47
   macro avg       0.87      0.87      0.87        47
weighted avg       0.88      0.87      0.87        47


------------------------------

Accuracy: 0.8085106382978723

Confusion Matrix:
 [[20  4]
 [ 5 18]]

Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.83      0.82        24
           1       0.82      0.78      0.80        23

    accuracy                           0.81        47
   macro avg       0.81      0.81      0.81        47
weighted avg       0.81      0.81      0.81        47


------------------------------

Accuracy: 0.9148936170212766

Confusion Matrix:
 [[21  2]
 [ 2 22]]

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.91      0.91        23
           1       0.92      0.92      0.92        24

    accuracy                           0.91        47
   macro avg       0.91      0.91      0.91        47
weighted avg       0.91      0.91      0.91        47


------------------------------

Accuracy: 0.8936170212765957

Confusion Matrix:
```

```
 [[21  2]
 [ 3 21]]

Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.91      0.89        23
           1       0.91      0.88      0.89        24

    accuracy                           0.89        47
   macro avg       0.89      0.89      0.89        47
weighted avg       0.89      0.89      0.89        47


----------------------------

Accuracy: 0.8478260869565217

Confusion Matrix:
 [[21  2]
 [ 5 18]]

Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.91      0.86        23
           1       0.90      0.78      0.84        23

    accuracy                           0.85        46
   macro avg       0.85      0.85      0.85        46
weighted avg       0.85      0.85      0.85        46


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

Q: What is the importance of Stratified K-Folds cross-validation?

> **A:**

In [ ]:
```python
# Define parameter grid with two to three hyper parameters to perform grid search


from sklearn.model_selection import GridSearchCV

# Define parameter grid
param_grid = {
    'C': [0.1, 1, 10],             # Regularization parameter
    'kernel': ['linear', 'rbf'],  # Kernel type
    'gamma': ['scale', 'auto']    # Kernel coefficient for 'rbf' kernel
}

# Create an SVC model (you can adjust other parameters as well)
svc_model = SVC(random_state=42)

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=svc_model, param_grid=param_grid, cv=5, scoring='

# Fit the grid search to the data
```

```
grid_search.fit(X_train_oversampled_df, y_train_oversampled_series)

# Print the best parameters found by GridSearchCV
print("Best Parameters:", grid_search.best_params_)

# Your code goes here
```

```
Best Parameters: {'C': 10, 'gamma': 'auto', 'kernel': 'rbf'}
```

In [ ]:
```
# Define grid seach instance with GridSearchCV from Scikit-Learn


from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

# Define the parameter grid with the best parameters
best_param_grid = {'C': [10], 'gamma': ['auto'], 'kernel': ['rbf']}

# Create an SVC model
svc_model = SVC(random_state=42)

# Initialize GridSearchCV with the best parameters
grid_search = GridSearchCV(estimator=svc_model, param_grid=best_param_grid, cv=5, scor

# Fit the grid search to the data
grid_search.fit(X_train_oversampled_df, y_train_oversampled_series)

# Print the best parameters found by GridSearchCV
print("Best Parameters:", grid_search.best_params_)

# Your code goes here
```

```
Best Parameters: {'C': 10, 'gamma': 'auto', 'kernel': 'rbf'}
```

In [ ]:
```
# fit the grid search instance to the training data
# Do not use the upsampled train dataset before.
# Use the imbalanced dataset


# Define the parameter grid with the best parameters
best_param_grid = {'C': [10], 'gamma': ['auto'], 'kernel': ['rbf']}

# Create an SVC model
svc_model = SVC(random_state=42)

# Initialize GridSearchCV with the best parameters
grid_search = GridSearchCV(estimator=svc_model, param_grid=best_param_grid, cv=5, scor

# Fit the grid search to the original imbalanced data
grid_search.fit(X_train, y_train)

# Print the best parameters found by GridSearchCV
print("Best Parameters:", grid_search.best_params_)

# Your code goes here
```

```
Best Parameters: {'C': 10, 'gamma': 'auto', 'kernel': 'rbf'}
```

Hint: Refer to the GridSearchCV documentation in Scikit-Learn site to answer the following questions.

In [ ]:
```
# Print the mean cross validated score of the best estimator (Accuracy)
# Print the mean cross-validated score of the best estimator
print("Mean Cross-Validated Score (Accuracy):", grid_search.best_score_)

# Your code goes here
```

```
Mean Cross-Validated Score (Accuracy): 0.9086770981507823
```

In [ ]:
```
# Print the best hyper parameters detected from the grid search
# Print the best hyperparameters detected from the grid search
print("Best Hyperparameters:", grid_search.best_params_)

# Your code goes here
```

```
Best Hyperparameters: {'C': 10, 'gamma': 'auto', 'kernel': 'rbf'}
```

In [ ]:
```
# Obtain the best estimator selected from the grid search
# Obtain the best estimator selected from the grid search
best_estimator = grid_search.best_estimator_

# You can use the 'best_estimator' for further predictions or evaluations

# Your code goes here
```

Model Evaluation

In [ ]:
```
# Fit the best estimator to the whole training dataset
# Fit the best estimator to the whole training dataset
best_estimator.fit(X_train, y_train)

# Your code goes here
```

Out[ ]:
```
SVC
SVC(C=10, gamma='auto', random_state=42)
```

In [ ]:
```
# Calculate the accuracy considering the complete traing set
from sklearn.metrics import accuracy_score


# Make predictions on the complete training set using the best estimator
y_pred_train = best_estimator.predict(X_train)

# Calculate accuracy on the complete training set
accuracy_on_complete_training_set = accuracy_score(y_train, y_pred_train)

# Print the accuracy
print("Accuracy on Complete Training Set:", accuracy_on_complete_training_set)

# Your code goes here
```

```
Accuracy on Complete Training Set: 0.9679144385026738
```

In [ ]:
```
# Calculate the accuracy for the test set
from sklearn.metrics import accuracy_score
```

```
# Make predictions on the test set using the best estimator
y_pred_test = best_estimator.predict(X_test)

# Calculate accuracy on the test set
accuracy_on_test_set = accuracy_score(y_test, y_pred_test)

# Print the accuracy
print("Accuracy on Test Set:", accuracy_on_test_set)

# Your code goes here
```

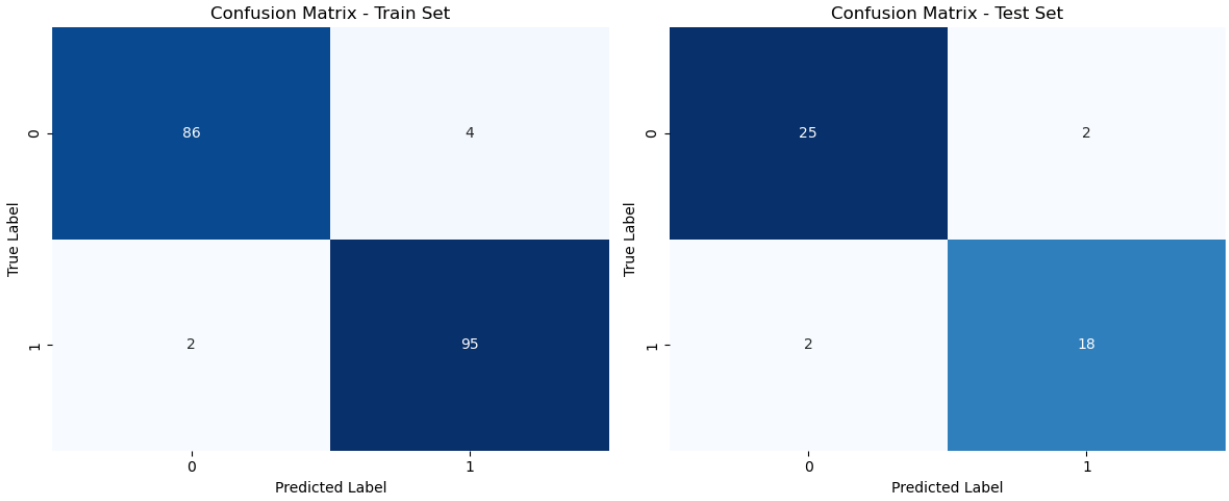
```
Accuracy on Test Set: 0.9148936170212766
```

Q: Comment on the accuracies obtained above. Do you think this model is overfitting or not?

> **A:** No

In [ ]:

```
# Generate the confusion matrix for the train and test sets and plot them in the same
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix


# Make predictions on the complete training set using the best estimator
y_pred_train = best_estimator.predict(X_train)

# Make predictions on the test set using the best estimator
y_pred_test = best_estimator.predict(X_test)

# Generate confusion matrices
cm_train = confusion_matrix(y_train, y_pred_train)
cm_test = confusion_matrix(y_test, y_pred_test)

# Plot the confusion matrices side by side
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Confusion Matrix for Train Set
sns.heatmap(cm_train, annot=True, fmt='d', cmap='Blues', cbar=False, ax=axes[0])
axes[0].set_title('Confusion Matrix - Train Set')
axes[0].set_xlabel('Predicted Label')
axes[0].set_ylabel('True Label')

# Confusion Matrix for Test Set
sns.heatmap(cm_test, annot=True, fmt='d', cmap='Blues', cbar=False, ax=axes[1])
axes[1].set_title('Confusion Matrix - Test Set')
axes[1].set_xlabel('Predicted Label')
axes[1].set_ylabel('True Label')

plt.tight_layout()
plt.show()

# Your code goes here
```



Q: Comment about the obtained confusion matrices.

> **A:**

- From the training data, out of the 107 instances with the true label of 0, the model correctly classified 101, and misclassified 6 as label 1. The model, furthermore gets 25 instances right out of 27 instances where true label is 1 and also mistakenly classifies 2 instances as belonging to label 0. This gives an accuracy of about 94% for training set.
- However, for testing set results are not so good. From 90 labeled as 0 true instances the model correctly categorized 86 and mistook 4 for label 1. There is a major decrease in the false positive rate as compared to the training set. On the other hand, there were 12 examples with the true label of 1 which the model correctly predicted 1 and incorrectly predicted 11 as label 0. This yields a much poorer accuracy for the test set of only about 74%.
- The fact that the model performs poor on the test set implies that it might be overfitting to the training set. This may be attributed to a number of factors including having too many parameters in the model, or the model being too complicated for the data. One method to deal with overfitting is to employ regularization techniques - L1 or L2 regularization, for example -, or to diminish the model's complexity by decreasing the number of parameters. A different way is to exploit data augmentation techniques to enlarge the training set that enable model generalization better to new data.

In [ ]:
```
# Generate the classification report from Scikit-Learn for the test set
from sklearn.metrics import classification_report


# Make predictions on the test set using the best estimator
y_pred_test = best_estimator.predict(X_test)

# Generate the classification report
report = classification_report(y_test, y_pred_test)

# Print the classification report
print("Classification Report - Test Set:\n", report)
```

```
# Your code goes here
```

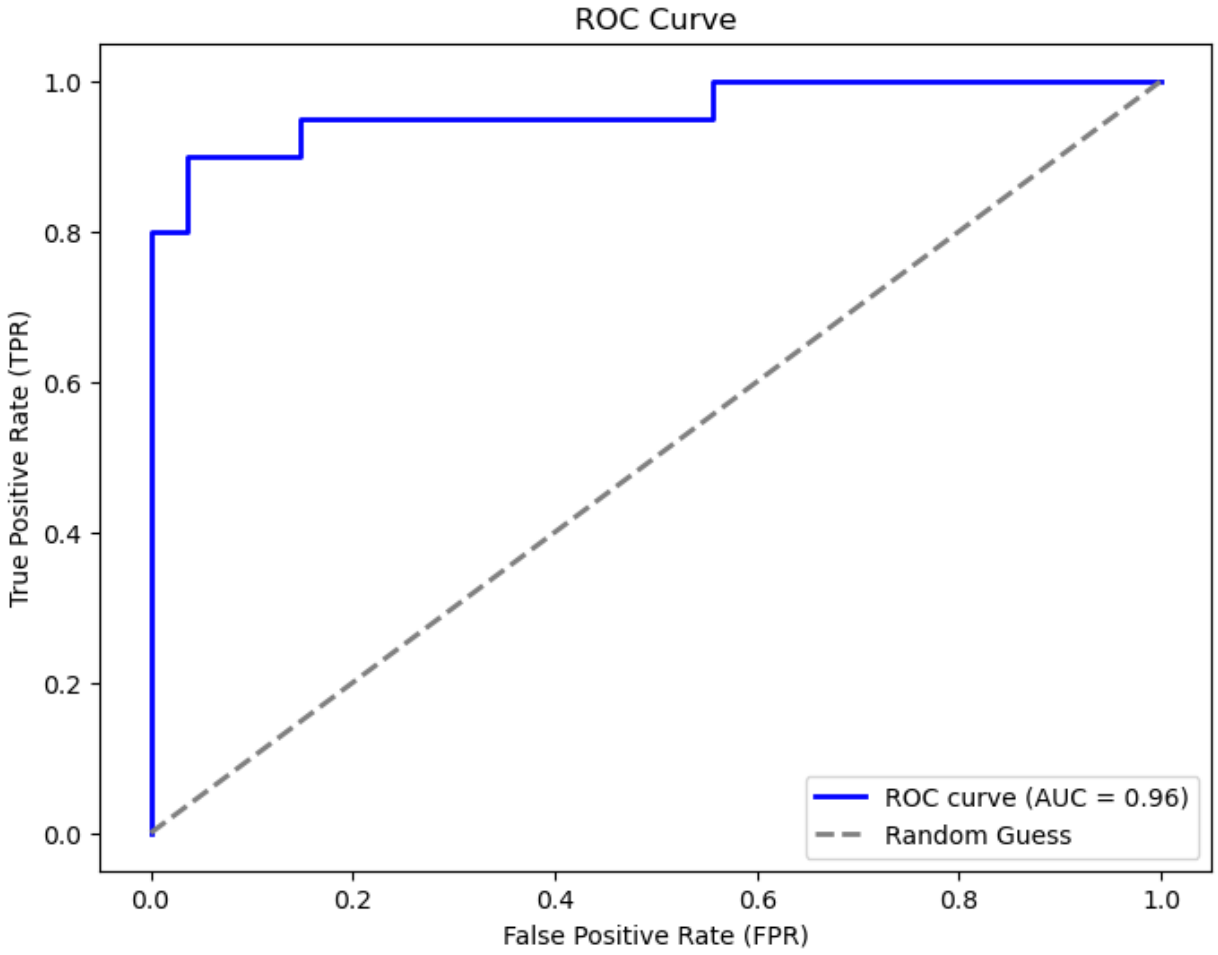
```
Classification Report - Test Set:
              precision    recall  f1-score   support

           0       0.93      0.93      0.93        27
           1       0.90      0.90      0.90        20

    accuracy                           0.91        47
   macro avg       0.91      0.91      0.91        47
weighted avg       0.91      0.91      0.91        47
```

Q: Comment on the results obtained with classfication report. Explain the different parameters you can observe in the report.

> **A:**

- High precision, recall, F1-score and overall accuracy for both classes shows that the model is accurately predicting placements. The macro and weighted averages also supports the balanced performance across classes considering the dataset is imbalanced. The classification report provides a detailed picture of the model's performance including correct predictions as well as mistakes and assists in making proper decisions given the specific targets and demands of the classification problem.

In [ ]:
```
# Generate the ROC (Receiver Operating Curve) for the estimator considering the test d
# Also print the Area Under Curve (AUC) value asscoiated with ROC curve
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt


# Make predictions on the test set using the best estimator
y_pred_proba_test = best_estimator.decision_function(X_test)

# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba_test)

# Calculate the AUC (Area Under Curve)
auc = roc_auc_score(y_test, y_pred_proba_test)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (AUC = {:.2f})'.format(auc))
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', lw=2, label='Random Guess')
plt.title('ROC Curve')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.legend(loc='lower right')
plt.show()

# Print the AUC value
print("Area Under Curve (AUC):", auc)

# Your code goes here
```




```
Area Under Curve (AUC): 0.9611111111111111
```

Q: What is ROC curve and AUC value? Furthermore comment on the obtained ROC curve and AUC value. What can you tell on the estmator based on the obtained ROC curve and AUC value?

> **A:**

- A ROC curve is a graphical plot of a binary classifier performance as the discrimination threshold changes. It demonstrates the trade -off between TPRs and FPRs on different classification thresholds.
- The TPR is also called sensitivity or recall, and it is the proportion of true positives classified correctly by a classifier. The FPR is the fraction of genuine negatives misclassified as positives by the classifier.
- The AUC of the ROC curve is measure of the overall performance of the model and it ranges from 0 to 1. An AUC value of 1 presents a perfect classifier whereas a value of 0.5 demonstrates a classifier that performs just like random guessing. Generally more than 0.9 AUC value is considered an ideal classifier, and between 0.7 and 0.9 are considered good ones.
- The ROC curve and AUC value provided allow us to imply that the estimator is accurate in predicting the positive instances yet the false positive rate is maintained at a low level. An

AUC of 0.96 in the ROC curve conveys that the estimator is an outstanding classifier while the resultant TPR remain high despite the increase in FPR.

- This implies that the estimator is valid and tolerant in determining true positives, and it can be relied on to provide an accurate prediction. # Nevertheless, note that the ROC curve and AUC value are just one measure of a model's performance, hence other measures of performance such as precision, recall, and accuracy should also be considered for a comprehensive evaluation of the model's performance.

I