Campus Placement Data

Objective

The primary objective of this project is to develop a classification model to predict whether a student will be placed or not in a campus placement scenario.

Dataset Description

The dataset contains placement data of students in a XYZ campus. Key features include secondary,higher secondary school percentages, specialization, degree specialization, degree type, work experience, and salary offers to placed students. The target variable is binary, indicating whether a student is placed or not.

Model building using Logistic Regression

In this project, we employ the Logistic Regression algorithm for binary classification. Logistic Regression is well-suited for scenarios where the outcome variable has two classes. The model utilizes the logistic function (sigmoid) to map input features to a probability range of [0, 1].

Outline

1. Importing the Libraries
2. Reading the Dataset
3. Data pre-processing and EDA
4. Splitting the Dependent and Independent Variables
5. Splitting the dataset into Train and Test
6. Model Building (Without handling Imbalanced data)
7. HyperParameter Tuning using GridSearchCV
8. Model Building after balancing the data using SMOTE
9. Conclusion

Importing libraries

In [1]:
```
import os, sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
%matplotlib inline
sns.set_theme(style='whitegrid')

from warnings import filterwarnings
filterwarnings('ignore')
```

Reading the Dataset

In [2]:
```
df = pd.read_csv(r"C:\Users\abhis\Downloads\Placement_Data_Full_Class.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etes
0	1	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	No	5
1	2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Yes	8
2	3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	No	7
3	4	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	No	6
4	5	M	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	No	9

Basic Information about the dataset

In [4]:
```
df.shape
```

Out[4]:
```
(215, 15)
```

In [5]:
```
print('*'*30 + ' No. of rows in the dataset are :', df.shape[0], '*'*30)
print('*'*30 + ' No. of columns in the dataset are :', df.shape[1], '*'*28)
```

```
****************************** No. of rows in the dataset are : 215 **************
****************
****************************** No. of columns in the dataset are : 15 ************
****************
```

In [6]:
```
df.info()
```

```
<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
```

In [7]:
```
print(df.columns.to_list())
```

```
['sl_no', 'gender', 'ssc_p', 'ssc_b', 'hsc_p', 'hsc_b', 'hsc_s', 'degree_p', 'degr
ee_t', 'workex', 'etest_p', 'specialisation', 'mba_p', 'status', 'salary']
```

Check for missing values

In [8]:
```
df.isnull().sum()

# Salary column has 67 missing values for students who did not get the placed.
```

Out[8]:
```
sl_no              0
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
status             0
salary            67
dtype: int64
```

In [9]:
```
df[(df['salary'].isnull()) & (df['status'] == 'Placed')]
```

Out[9]:

sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	specialisa

Checking for duplicate records

In [10]:
```
df.duplicated().sum()
```

Out[10]:
```
0
```

Statistical Analysis

In [11]:
```
df.describe().transpose().style
```

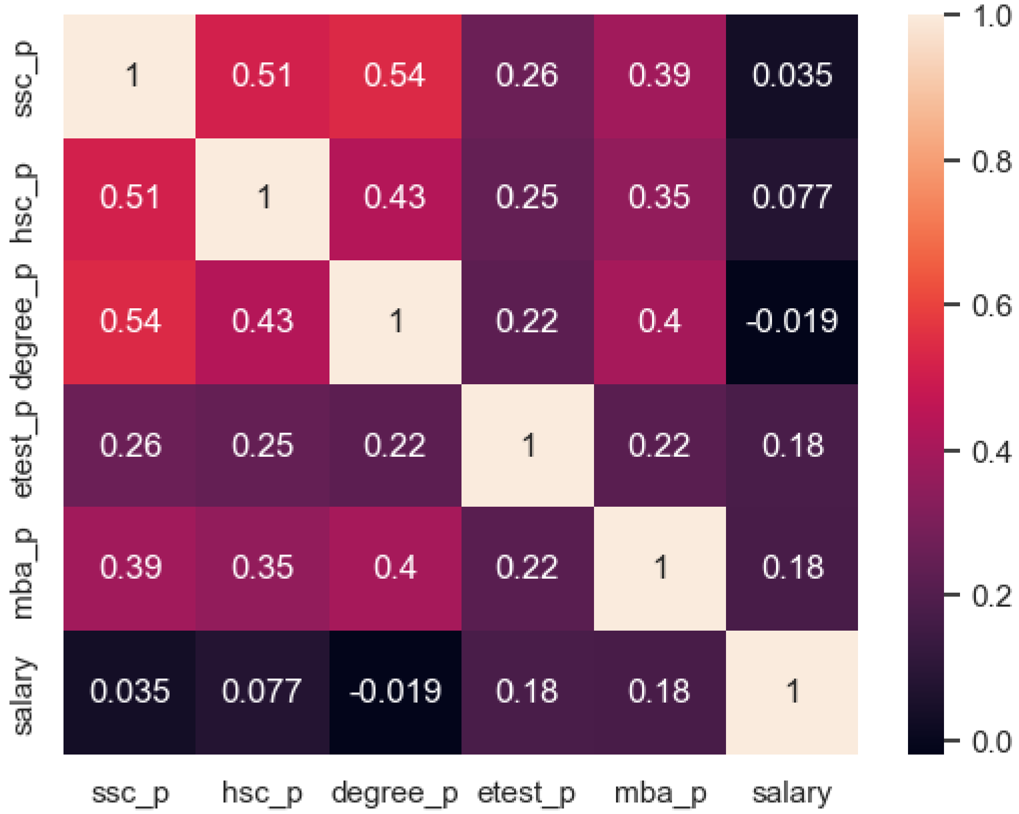
Out[11]:

	count	mean	std	min	25%	50%
sl_no	215.000000	108.000000	62.209324	1.000000	54.500000	108.000000
ssc_p	215.000000	67.303395	10.827205	40.890000	60.600000	67.000000
hsc_p	215.000000	66.333163	10.897509	37.000000	60.900000	65.000000
degree_p	215.000000	66.370186	7.358743	50.000000	61.000000	66.000000
etest_p	215.000000	72.100558	13.275956	50.000000	60.000000	71.000000
mba_p	215.000000	62.278186	5.833385	51.210000	57.945000	62.000000
salary	148.000000	288655.405405	93457.452420	200000.000000	240000.000000	265000.000000

In [12]:
```
#Removing the sl_no from the dataframe

df.drop('sl_no', axis=1, inplace=True)
```

Exploratory Data Analysis

In [13]:
```
# Checking correlation between the variables using heatmap

sns.heatmap(df.corr(), annot=True)
plt.show()
```



Checking for outliers

In [14]:
```
Numerical_cols = df.select_dtypes(exclude='object')
Categorical_cols = df.select_dtypes(include = 'object')
```

In [15]:
```
Numerical_cols
```

Out[15]:

	ssc_p	hsc_p	degree_p	etest_p	mba_p	salary
0	67.00	91.00	58.00	55.0	58.80	270000.0
1	79.33	78.33	77.48	86.5	66.28	200000.0
2	65.00	68.00	64.00	75.0	57.80	250000.0
3	56.00	52.00	52.00	66.0	59.43	NaN
4	85.80	73.60	73.30	96.8	55.50	425000.0
...	...	...	...	...	...	...
210	80.60	82.00	77.60	91.0	74.49	400000.0
211	58.00	60.00	72.00	74.0	53.62	275000.0
212	67.00	67.00	73.00	59.0	69.72	295000.0
213	74.00	66.00	58.00	70.0	60.23	204000.0
214	62.00	58.00	53.00	89.0	60.22	NaN

215 rows × 6 columns

In [16]:

```
Categorical_cols
```

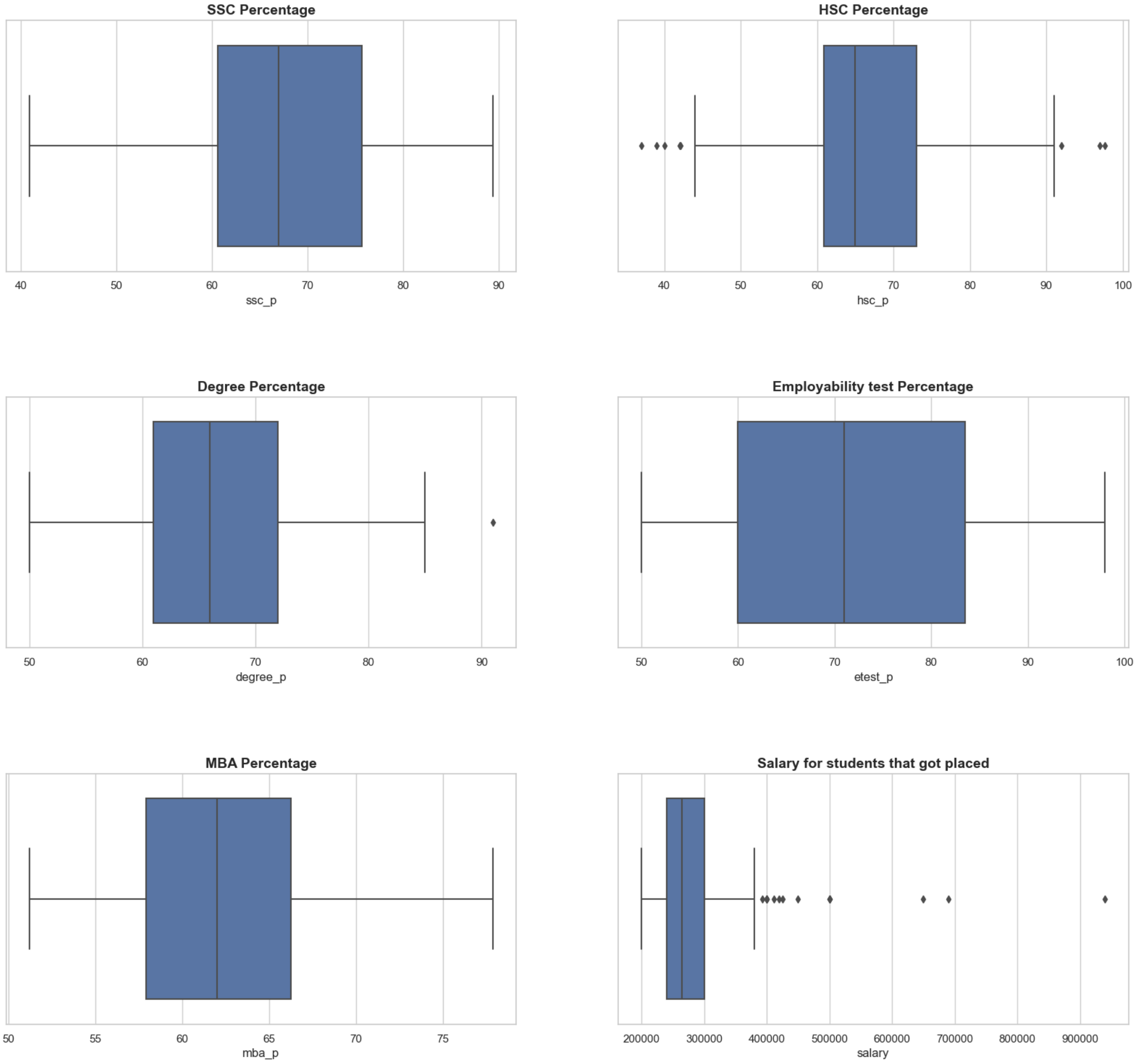
Out[16]:

	gender	ssc_b	hsc_b	hsc_s	degree_t	workex	specialisation	status
0	M	Others	Others	Commerce	Sci&Tech	No	Mkt&HR	Placed
1	M	Central	Others	Science	Sci&Tech	Yes	Mkt&Fin	Placed
2	M	Central	Central	Arts	Comm&Mgmt	No	Mkt&Fin	Placed
3	M	Central	Central	Science	Sci&Tech	No	Mkt&HR	Not Placed
4	M	Central	Central	Commerce	Comm&Mgmt	No	Mkt&Fin	Placed
...	...	...	...	...	...	...	...	...
210	M	Others	Others	Commerce	Comm&Mgmt	No	Mkt&Fin	Placed
211	M	Others	Others	Science	Sci&Tech	No	Mkt&Fin	Placed
212	M	Others	Others	Commerce	Comm&Mgmt	Yes	Mkt&Fin	Placed
213	F	Others	Others	Commerce	Comm&Mgmt	No	Mkt&HR	Placed
214	M	Central	Others	Science	Comm&Mgmt	No	Mkt&HR	Not Placed

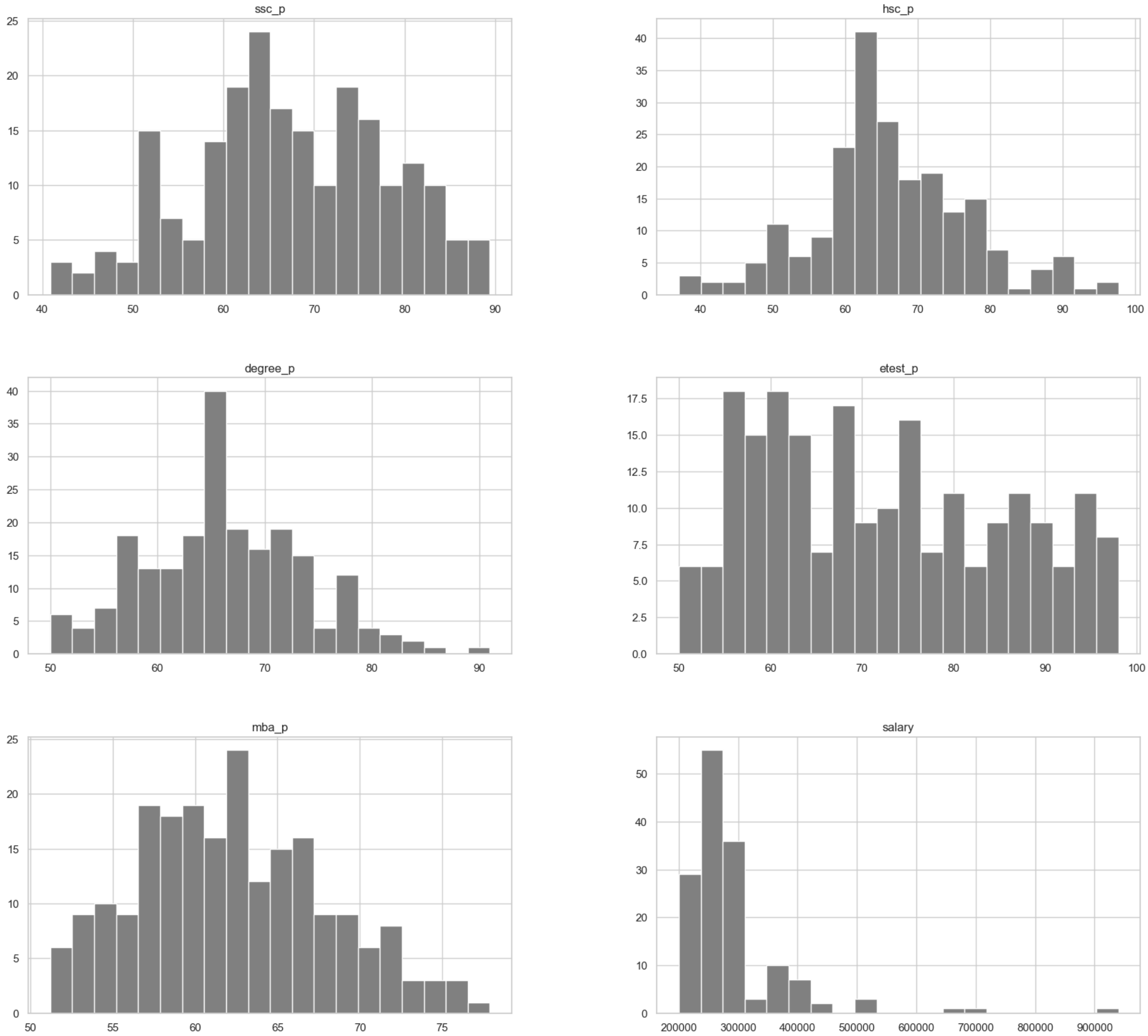
215 rows × 8 columns

In [17]:

```
fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(20,18))
fig.subplots_adjust(hspace=0.5)
sns.boxplot(df['ssc_p'], ax=axes[0,0]).set_title('SSC Percentage', fontweight='bolc
sns.boxplot(df['hsc_p'], ax=axes[0,1]).set_title('HSC Percentage',fontweight='bold'
sns.boxplot(df['degree_p'], ax=axes[1,0]).set_title('Degree Percentage',fontweight=
sns.boxplot(df['etest_p'], ax=axes[1,1]).set_title('Employability test Percentage',
sns.boxplot(df['mba_p'], ax=axes[2,0]).set_title('MBA Percentage',fontweight='bold'
sns.boxplot(df['salary'], ax=axes[2,1]).set_title('Salary for students that got pla
plt.show()
```



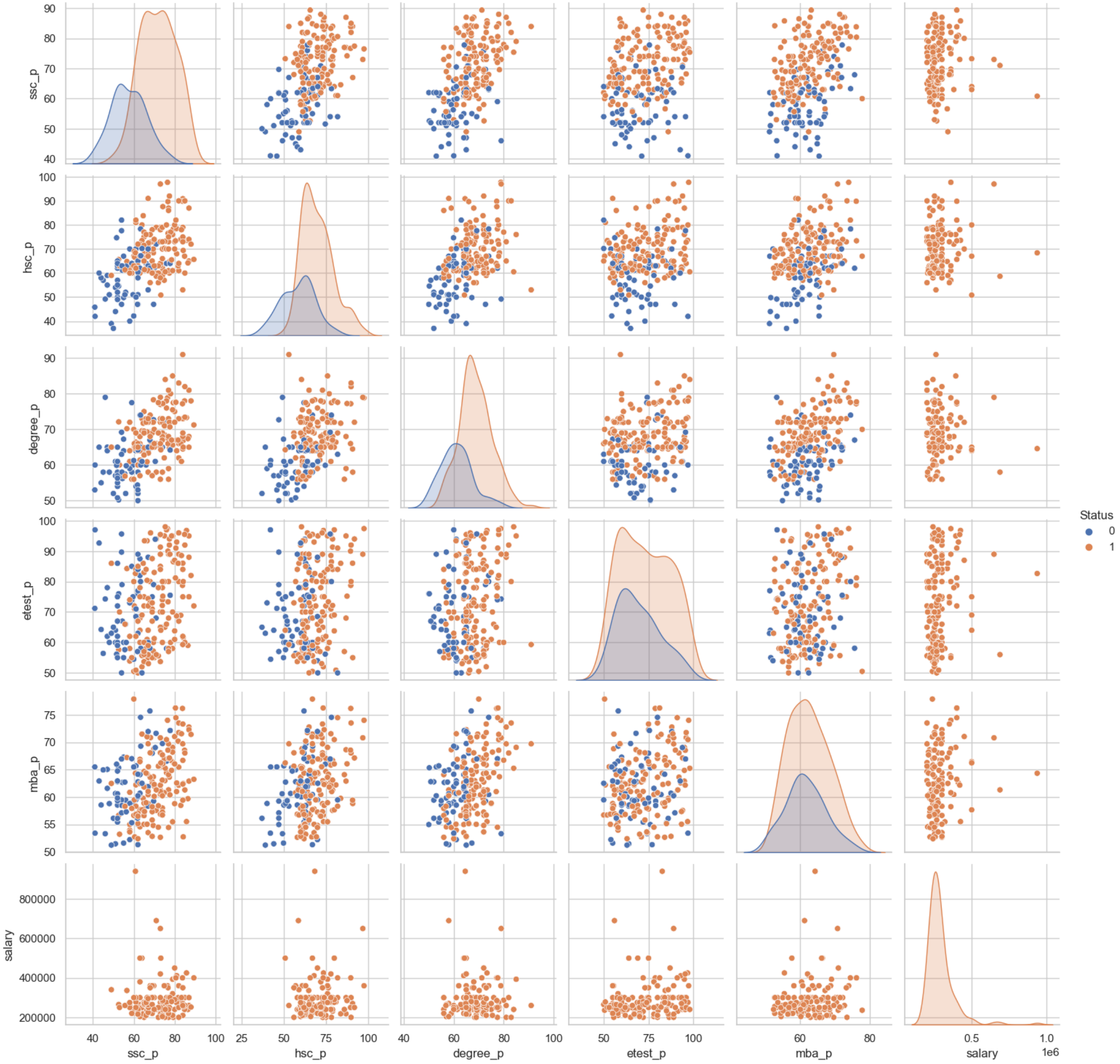
In [18]:
```
df.hist(figsize=(20,18), bins=20, color='grey')
plt.show()
```



In [19]:
```
# Adding the status variable in numerical format for better visualization

df['Status'] = df['status'].replace({'Placed': 1, 'Not Placed' : 0})
```

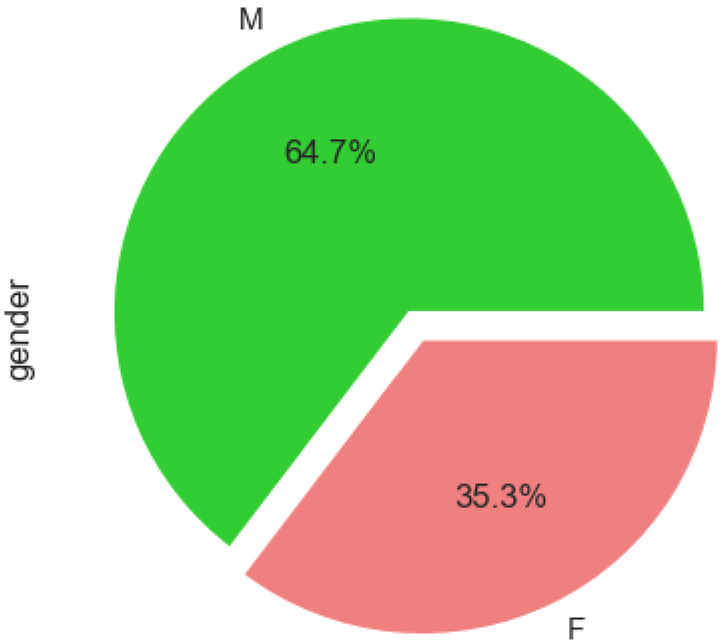
In [20]:
```
plt.figure(figsize=(12,10))
sns.pairplot(data=df, hue='Status')
plt.show()
```

<Figure size 1200x1000 with 0 Axes>

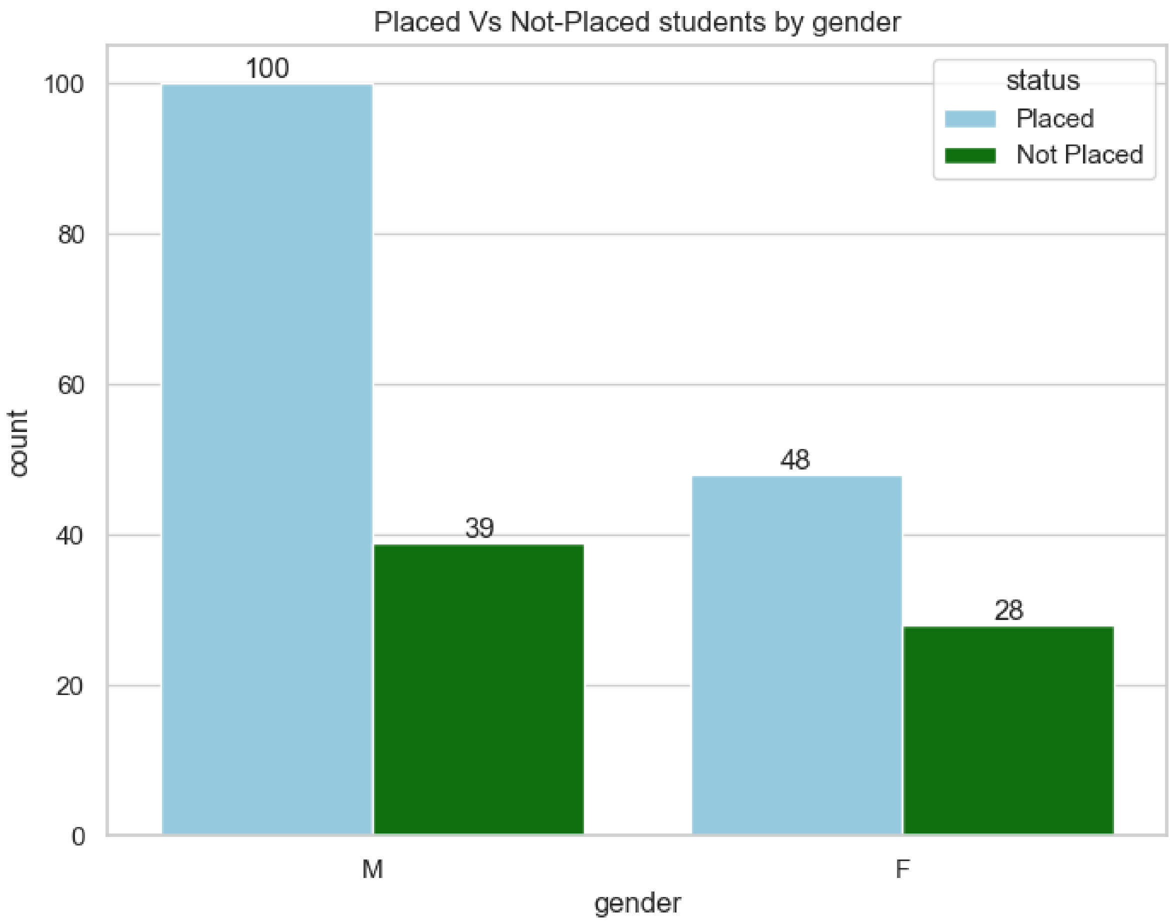


Gender Distribution

In [21]:
```
df['gender'].value_counts().plot(kind='pie', autopct='%0.1f%%', explode=(0.05,0.05)
plt.show()
```

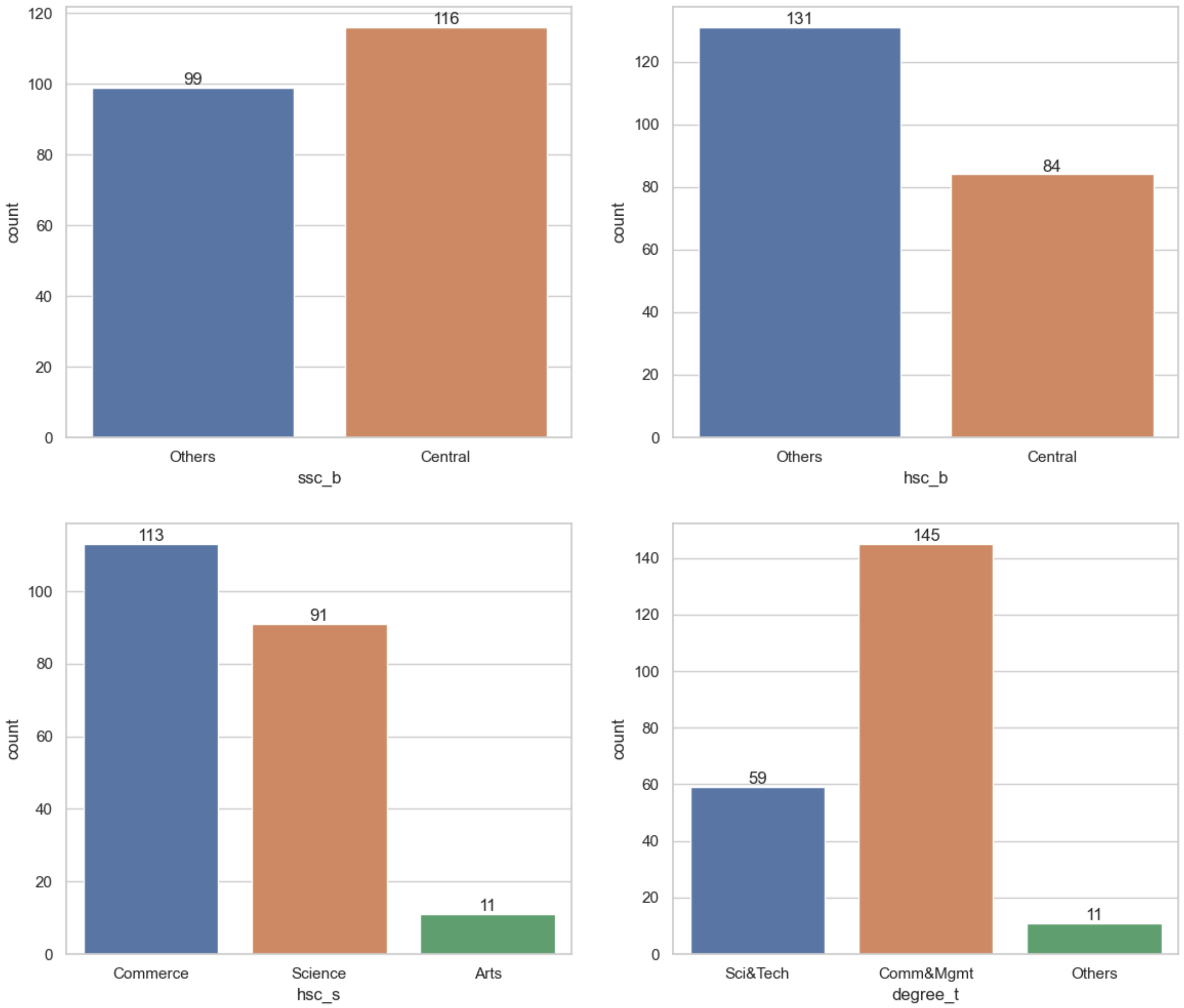


In [22]:
```
plt.figure(figsize=(8,6))
ax = sns.countplot(df['gender'], hue=df['status'], palette=['skyblue','green'])
for i in ax.containers:
    ax.bar_label(i)
plt.title('Placed Vs Not-Placed students by gender')
plt.show()
```

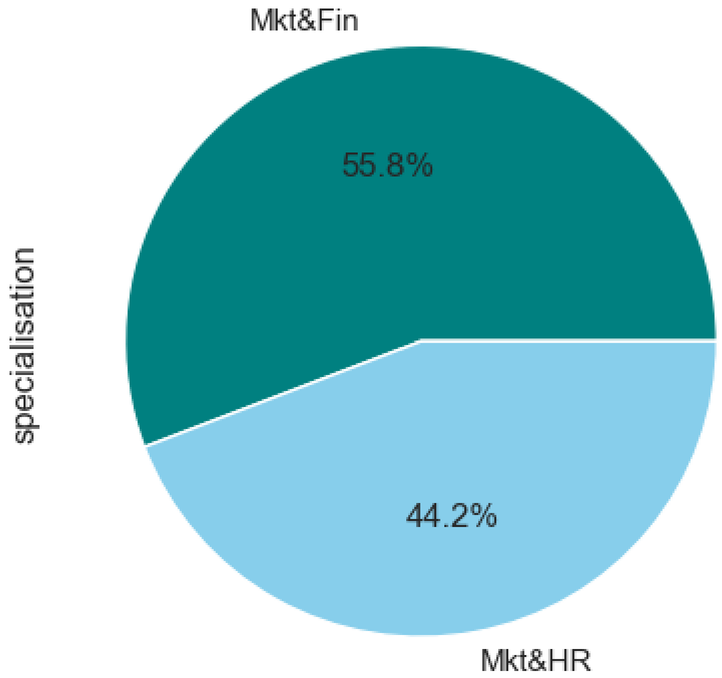


In [23]:
```
# Student Distribution across all categories
```

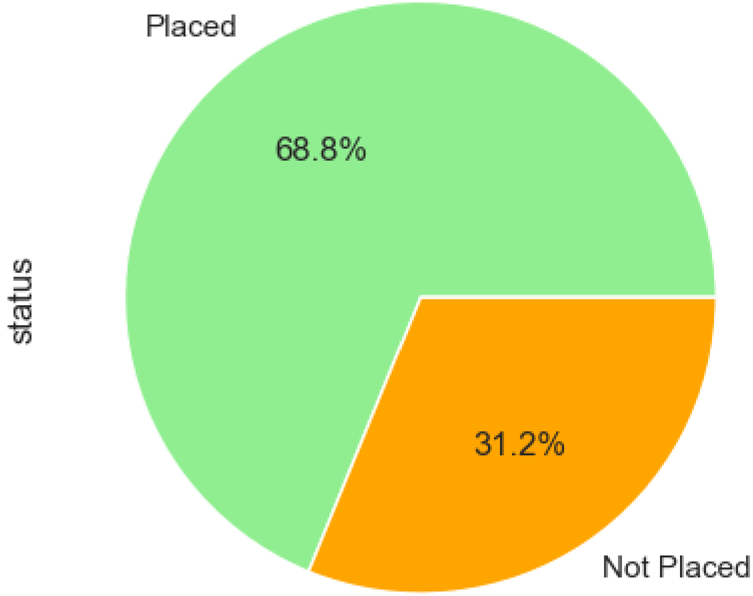
```
fig, axes = plt.subplots(nrows= 2, ncols=2, figsize=(14,12))
ax=sns.countplot(df['ssc_b'], ax=axes[0,0])
for i in ax.containers:
    ax.bar_label(i)
ax1=sns.countplot(df['hsc_b'], ax=axes[0,1])
for i in ax1.containers:
    ax1.bar_label(i)
ax2=sns.countplot(df['hsc_s'], ax=axes[1,0])
for i in ax2.containers:
    ax2.bar_label(i)
ax3=sns.countplot(df['degree_t'], ax=axes[1,1])
for i in ax3.containers:
    ax3.bar_label(i)
```



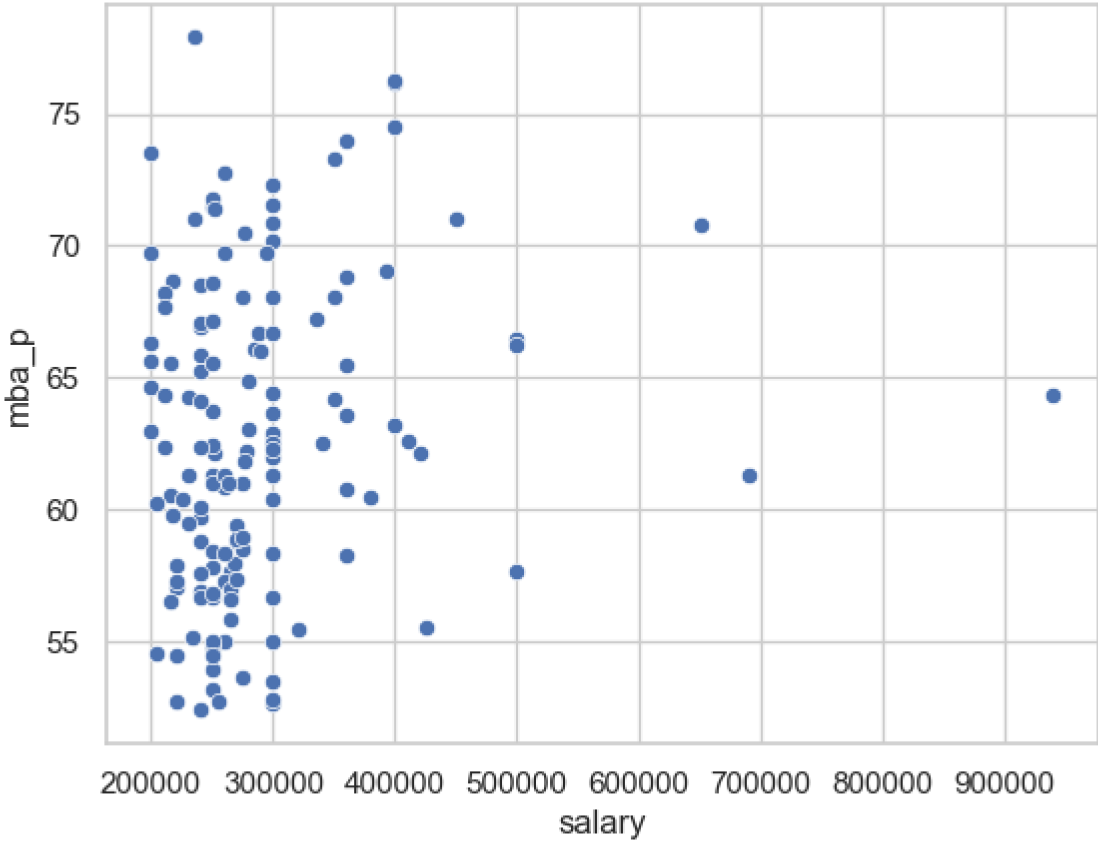
In [24]:
```
df['specialisation'].value_counts().plot(kind='pie', autopct='%0.1f%%', colors=['te
plt.show()
```



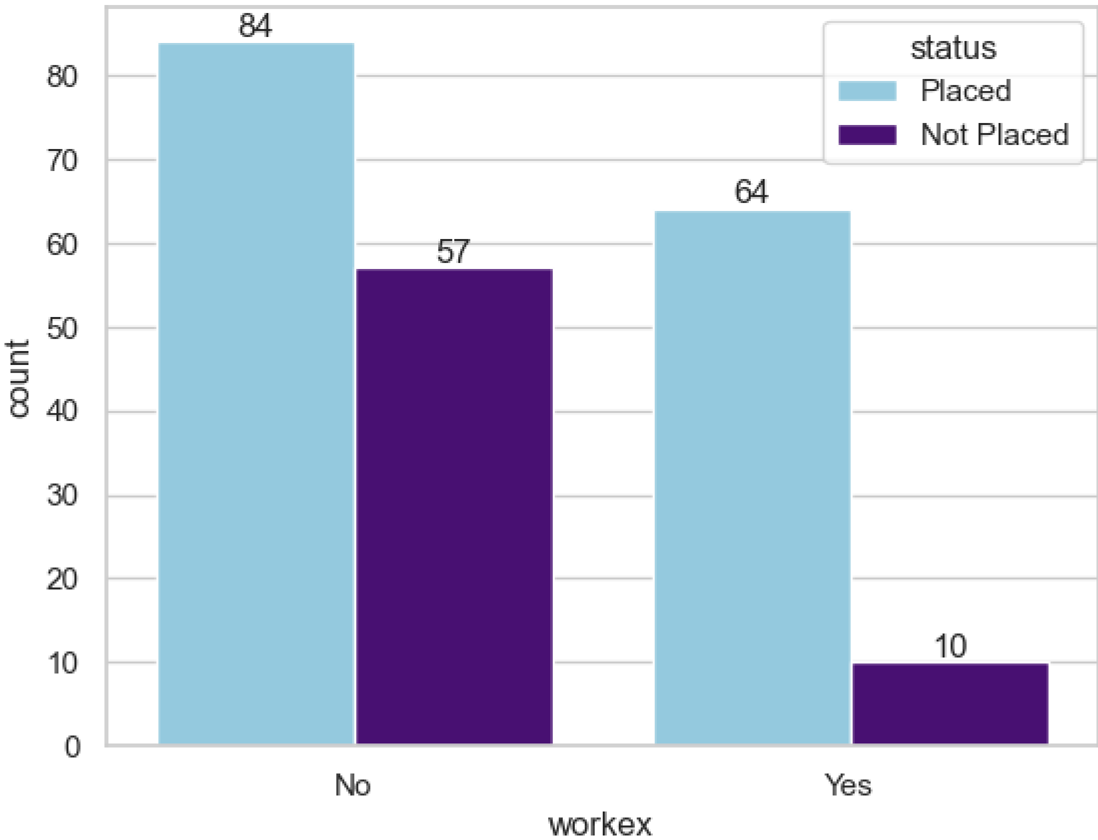
In [25]:
```
df['status'].value_counts().plot(kind='pie', autopct='%0.1f%%', colors=['lightgreer
plt.show()

# 69% students got placed while 31% did not get the placement.
```



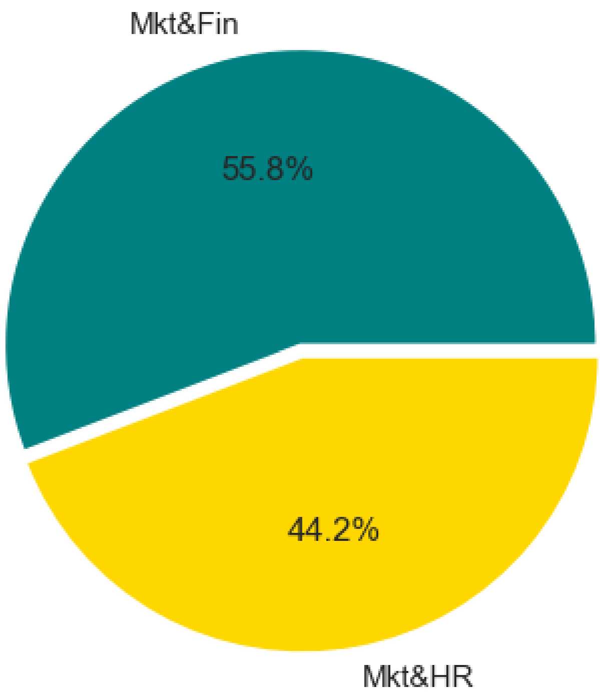
In [26]:
```
# Relationship between MBA percentage and salary:

sns.scatterplot(x=df['salary'],y=df['mba_p'])
plt.show()
```



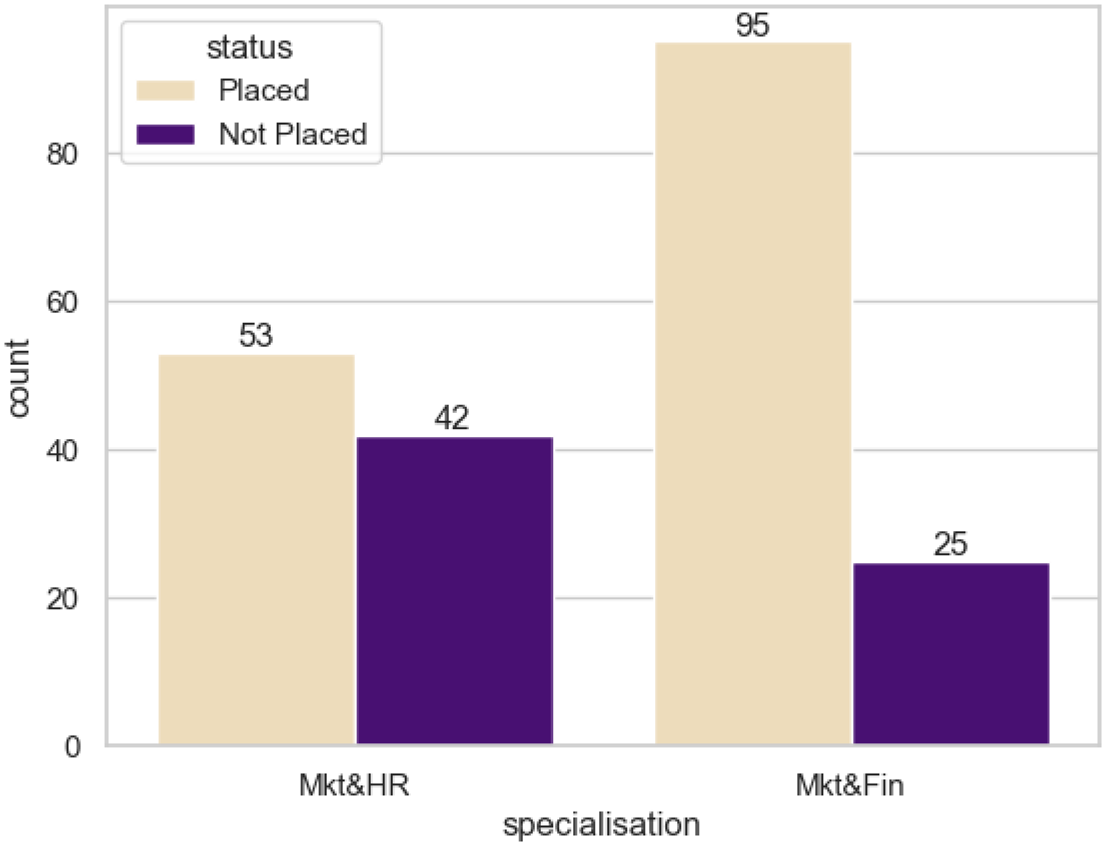
In [27]:
```
ax = sns.countplot(df['workex'], hue=df['status'], palette=['skyblue','indigo'])
for i in ax.containers:
    ax.bar_label(i)
```



In [28]:
```
plt.pie(df['specialisation'].value_counts(), labels=df['specialisation'].value_cour
        explode=(0.02,0.02), colors=['teal','gold'])
plt.show()
```



In [29]:
```
ax = sns.countplot(df['specialisation'], hue=df['status'], palette=['wheat','indigo
for i in ax.containers:
    ax.bar_label(i)
```



In [30]:
```
data = pd.crosstab(df['specialisation'], df['status'])
data['Total'] = data['Placed']+ data['Not Placed']
data['% of students placed in each specialisation'] = data['Placed']*100/data['Tota
data
```

Out[30]:

status	Not Placed	Placed	Total	% of students placed in each specialisation
specialisation				
Mkt&Fin	25	95	120	79.166667
Mkt&HR	42	53	95	55.789474

In [31]:
```
pd.crosstab(index=[df['degree_t'], df['specialisation']], columns=df['status'])
```

Out[31]:

	status	Not Placed	Placed
degree_t	**specialisation**		
Comm&Mgmt	**Mkt&Fin**	18	68
	Mkt&HR	25	34
Others	**Mkt&Fin**	2	2
	Mkt&HR	4	3
Sci&Tech	**Mkt&Fin**	5	25
	Mkt&HR	13	16

Encoding the categorical variables

In [32]:
```
from sklearn.preprocessing import LabelEncoder
```

In [33]:
```
le = LabelEncoder()
```

In [34]:
```
columns = Categorical_cols.columns
columns
```

Out[34]:
```
Index(['gender', 'ssc_b', 'hsc_b', 'hsc_s', 'degree_t', 'workex',
       'specialisation', 'status'],
      dtype='object')
```

In [35]:
```
df['gender'] = le.fit_transform(df['gender'])
df['ssc_b'] = le.fit_transform(df['ssc_b'])
df['hsc_b'] = le.fit_transform(df['hsc_b'])
df['hsc_s'] = le.fit_transform(df['hsc_s'])
df['degree_t'] = le.fit_transform(df['degree_t'])
df['workex'] = le.fit_transform(df['workex'])
df['specialisation'] = le.fit_transform(df['specialisation'])
```

Dropping Salary column as the students who did not get placed have the salary value as Null. This will create bias while model building as it is representing similar information as the Target variable 'status'*

In [36]:
```
new_df = df.drop(['salary','status'], axis=1)
new_df.head()
```

Out[36]:

	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	specialisation
0	1	67.00	1	91.00	1	1	58.00	2	0	55.0	1
1	1	79.33	0	78.33	1	2	77.48	2	1	86.5	0
2	1	65.00	0	68.00	0	0	64.00	0	0	75.0	0
3	1	56.00	0	52.00	0	2	52.00	2	0	66.0	1
4	1	85.80	0	73.60	0	1	73.30	0	0	96.8	0

In [37]:
```
new_df.isnull().sum()
```

Out[37]:
```
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
Status            0
dtype: int64
```

Splitting the dataset into Train and test

In [38]:
```
from sklearn.model_selection import train_test_split
```

In [39]:
```
X = new_df.iloc[:,:-1]
```

In [40]:
```
y = new_df.iloc[:,-1]
```

In [41]:
```
X.head()
```

Out[41]:

	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	specialisation
0	1	67.00	1	91.00	1	1	58.00	2	0	55.0	1
1	1	79.33	0	78.33	1	2	77.48	2	1	86.5	0
2	1	65.00	0	68.00	0	0	64.00	0	0	75.0	0
3	1	56.00	0	52.00	0	2	52.00	2	0	66.0	1
4	1	85.80	0	73.60	0	1	73.30	0	0	96.8	0

In [42]:
```
y.head()
```

Out[42]:
```
0    1
1    1
2    1
3    0
4    1
Name: Status, dtype: int64
```

In [43]:
```
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.2, random_state=1(
```

In [44]:
```
x_train.shape, x_test.shape
```

Out[44]:
```
((172, 12), (43, 12))
```

Case 1 : Model Building without handling the Imbalance in data

Model Building

Logistic Regression without Hyperparameter Tuning

In [45]:
```
from sklearn.linear_model import LogisticRegression
```

In [46]:
```
log_it = LogisticRegression(random_state=32)
```

In [47]:
```
log_it.fit(x_train,y_train)
```

Out[47]:
```
LogisticRegression(random_state=32)
```

In [48]:
```
y_pred_train = log_it.predict(x_train)
y_pred_test = log_it.predict(x_test)
```

In [49]:
```
# Evaluating the model

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

In [50]:
```
accuracy_score(y_test,y_pred_test) # Accuracy of Test data
```

Out[50]:
```
0.8837209302325582
```

In [51]:
```
accuracy_score(y_train,y_pred_train) # Accuracy of Train data
```

Out[51]:
```
0.8895348837209303
```

In [52]:
```
# Evaluating Test data

print(classification_report(y_test,y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.82      0.88      0.85        16
           1       0.92      0.89      0.91        27

    accuracy                           0.88        43
   macro avg       0.87      0.88      0.88        43
weighted avg       0.89      0.88      0.88        43
```

In [53]:
```
confusion_matrix(y_test,y_pred_test)
```

Out[53]:
```
array([[14,  2],
       [ 3, 24]], dtype=int64)
```

In [54]:
```
# Evaluating Train data

print(classification_report(y_train, y_pred_train))
```

```
              precision    recall  f1-score   support

           0       0.83      0.78      0.81        51
           1       0.91      0.93      0.92       121

    accuracy                           0.89       172
   macro avg       0.87      0.86      0.87       172
weighted avg       0.89      0.89      0.89       172
```

In [55]:
```
confusion_matrix(y_train,y_pred_train)
```

Out[55]:
```
array([[ 40,  11],
       [  8, 113]], dtype=int64)
```

Logistic Regression with Hyperparameter Tuning using GridSearchCV

In [56]:
```
from sklearn.model_selection import GridSearchCV
```

In [57]:
```
# Specifying the parameters that we want to Hypertune

parameters = {'penalty': ['l1','l2','elasticnet'], 'C': [1,2,3,5,10,20,30,50], 'max
```

In [58]:
```
log_it_grid = GridSearchCV(log_it, param_grid=parameters, scoring = 'accuracy', cv=
```

In [59]:
```
log_it_grid.fit(x_train,y_train)
```

Out[59]:
```
GridSearchCV(cv=10, estimator=LogisticRegression(random_state=32),
             param_grid={'C': [1, 2, 3, 5, 10, 20, 30, 50],
                         'max_iter': [100, 200, 300],
                         'penalty': ['l1', 'l2', 'elasticnet']},
             scoring='accuracy')
```

In [60]:
```
print(log_it_grid.best_params_)
```

```
{'C': 30, 'max_iter': 300, 'penalty': 'l2'}
```

In [61]:
```
y_pred_grid_test = log_it_grid.predict(x_test)
```

In [62]:
```
y_pred_grid_train = log_it_grid.predict(x_train)
```

In [63]:
```
accuracy_score(y_test,y_pred_grid_test) # Accuracy score of test data
```

Out[63]:
```
0.8837209302325582
```

In [64]:
```
accuracy_score(y_train,y_pred_grid_train) # Accuracy score of train data
```

Out[64]:
```
0.8895348837209303
```

In [65]:
```
# Evaluating Test data

print(classification_report(y_test,y_pred_grid_test))
```

```
              precision    recall  f1-score   support

           0       0.87      0.81      0.84        16
           1       0.89      0.93      0.91        27

    accuracy                           0.88        43
   macro avg       0.88      0.87      0.87        43
weighted avg       0.88      0.88      0.88        43
```

In [66]:
```
confusion_matrix(y_test,y_pred_grid_test)
```

Out[66]:
```
array([[13,  3],
       [ 2, 25]], dtype=int64)
```

In [67]:
```
# Evaluating Train data

print(classification_report(y_train, y_pred_grid_train))
```

```
              precision    recall  f1-score   support

           0       0.83      0.78      0.81        51
           1       0.91      0.93      0.92       121

    accuracy                           0.89       172
   macro avg       0.87      0.86      0.87       172
weighted avg       0.89      0.89      0.89       172
```

In [68]:
```
confusion_matrix(y_train,y_pred_grid_train)
```

Out[68]:
```
array([[ 40,  11],
       [  8, 113]], dtype=int64)
```

Plotting Area Under Receiver Operating Curve (AUROC)

AUC (Area Under the Curve): A metric that represents overall performance of a binary classification model based on the area under its ROC curve.

ROC Curve (Receiver Operating Characteristic Curve): It is a graphical plot illustrating the trade-off between True Positive Rate and False Positive Rate at various classification thresholds.

True Positive Rate (Sensitivity / tpr): Proportion of actual positives correctly identified by the model.

False Positive Rate (fpr) : The model incorrectly classifies the proportion of actual negatives as positives.

In [69]:
```
from sklearn.metrics import roc_auc_score, roc_curve
```

In [70]:
```
roc_auc_score(y_test,y_pred_test)
```

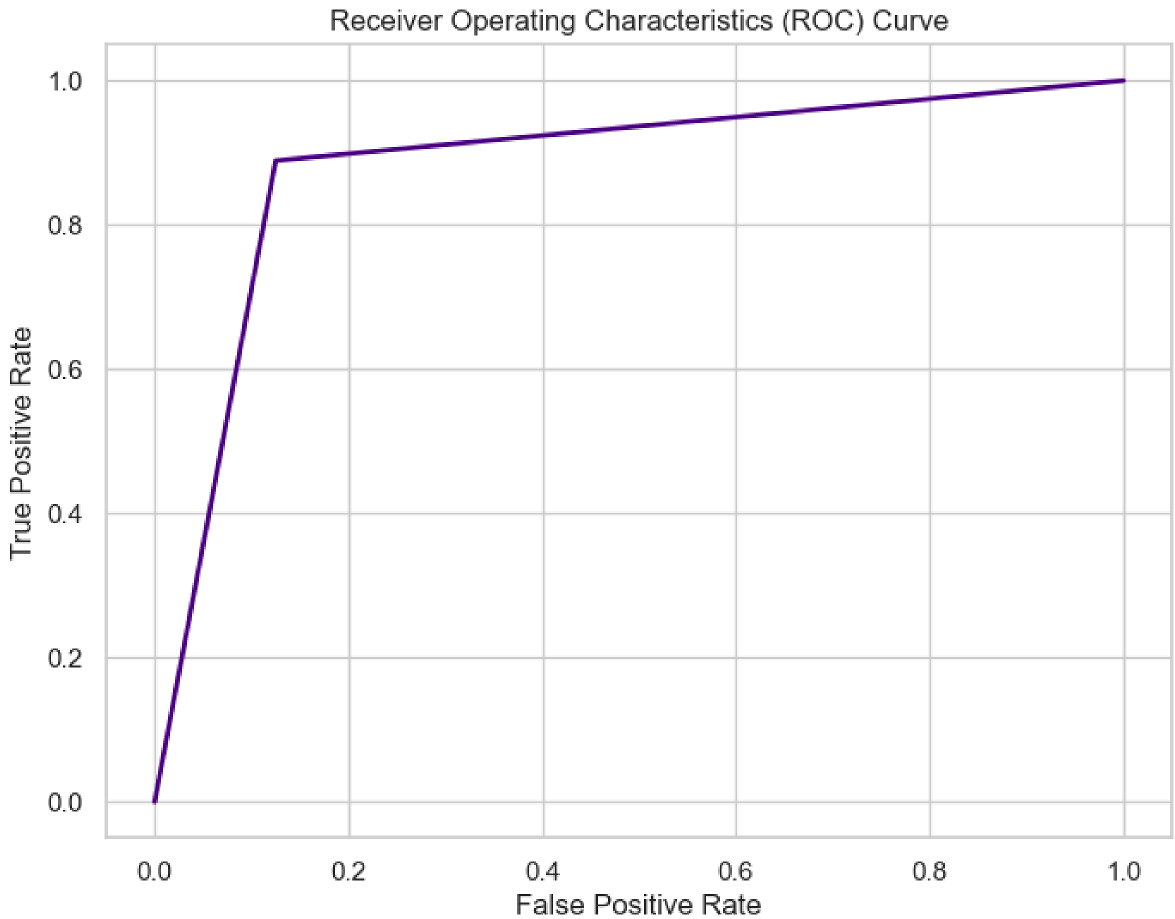
Out[70]:
```
0.8819444444444444
```

In [71]:
```
roc_auc_score(y_train,y_pred_train)
```

Out[71]:
```
0.8590990115054286
```

In [72]:
```
fpr, tpr, thresholds = roc_curve(y_test,y_pred_test)
```

In [73]:
```
plt.figure(figsize=(8,6))
plt.plot(fpr,tpr, color='indigo', lw=2)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristics (ROC) Curve')
plt.show()
```



AUC represents the degree or measure of separability. AUC score of the model is coming out to be 0.88 which is quite good, as higher the area under the curve better the model is at distinguishing the target class.

Case 2 : Model Building after handling Imbalance data

In [74]:
```
# Handling class Imbalance with SMOTE (Synthetic Minorty Over-sampling Technique)

from imblearn.over_sampling import SMOTE
```

In [75]:
```
smote = SMOTE()
```

In [76]:
```
y.value_counts()
```

Out[76]:
```
1    148
0     67
Name: Status, dtype: int64
```

In [77]:
```
x_smote,y_smote = smote.fit_resample(X,y)
```

In [78]:
```
y_smote.value_counts()
```

Out[78]:
```
1    148
0    148
Name: Status, dtype: int64
```

In [79]:
```
# Splitting the data to train and test

xtrain,xtest,ytrain,ytest=train_test_split(x_smote.values,y_smote,test_size=0.2,rar
```

In [80]:
```
log_it.fit(xtrain,ytrain)
```

Out[80]:
```
LogisticRegression(random_state=32)
```

In [81]:
```
ys_pred_test = log_it.predict(xtest)
ys_pred_train= log_it.predict(xtrain)
```

In [82]:
```
accuracy_score(ytest,ys_pred_test) #Accuracy score of Test data
```

Out[82]:
```
0.85
```

In [83]:
```
accuracy_score(ytrain,ys_pred_train) #Accuracy score of Train data
```

Out[83]:
```
0.8898305084745762
```

In [84]:
```
# Evaluating Test data

print(classification_report(ytest,ys_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.88      0.79      0.84        29
           1       0.82      0.90      0.86        31

    accuracy                           0.85        60
   macro avg       0.85      0.85      0.85        60
weighted avg       0.85      0.85      0.85        60
```

In [85]:
```
confusion_matrix(ytest,ys_pred_test)
```

Out[85]:
```
array([[23,  6],
       [ 3, 28]], dtype=int64)
```

In [86]:
```
# Evaluating Train data

print(classification_report(ytrain,ys_pred_train))
```

```
              precision    recall  f1-score   support

           0       0.90      0.88      0.89       119
           1       0.88      0.90      0.89       117

    accuracy                           0.89       236
   macro avg       0.89      0.89      0.89       236
weighted avg       0.89      0.89      0.89       236
```

In [87]:
```
confusion_matrix(ytrain,ys_pred_train)
```

Out[87]:
```
array([[105,  14],
       [ 12, 105]], dtype=int64)
```

Conclusion

Initial Model Evaluation

1. **Without Handling Imbalance in Data:**
    - The accuracy of the Logistic Regression model, without addressing class imbalance, is approximately 88% for both the Train and Test dataset.
    - This initial assessment provides a baseline understanding of the model's performance on the original, imbalanced dataset.

Hyperparameter Tuning

1. **Post Hyperparameter Tuning:**
    - After hyperparameter tuning the model, we observed that the accuracy remains consistent, hovering around 88% for both Train and Test datasets.
    - While hyperparameter tuning fine-tunes the model, it does not lead to a substantial improvement in accuracy in this case.

Impact of Handling Imbalanced Data

1. **Handling Imbalanced Data:**
    - There is change in the performance of the model when addressing class imbalance using SMOTE
    - The train accuracy increases to approximately 89%, indicating better capturing of patterns in the majority and minority classes.
    - The test accuracy is around 85% which is quite stable.

Generalization without Overfitting or Underfitting

1. **Absence of Overfitting or Underfitting:**
    - Notably, throughout these model iterations, there is neither the case of overfitting nor underfitting.
    - The model demonstrates consistent and reliable performance across both training and testing datasets.

In [ ]: