NNDL- Assignment 2

```
#IMport Libraries
import pandas as pd
import numpy as np
import seaborn as sns #for plotting, visualization
import matplotlib.pyplot as plt  #for plotting
from sklearn.preprocessing import LabelEncoder
```

```
#from google.colab import drive
#drive.mount('/content/drive')
```

```
# load datasets
dfsamplesubmission = pd.read_csv('/content/drive/MyDrive/dataset/SampleSubmission.csv')
```

```
dftest = pd.read_csv('/content/drive/MyDrive/dataset/test.csv')
```

```
dftrain = pd.read_csv('/content/drive/MyDrive/dataset/train.csv')
```

Analyze Datasets

```
#check the shape of the datasets
print("Shape of sample submission: ", dfsamplesubmission.shape)
print("Shape of test: ", dftest.shape)
print("Shape of train: ", dftrain.shape)
```

```
Shape of sample submission:  (43, 2)
Shape of test:  (43, 3)
Shape of train:  (215, 15)
```

```
#check the info of the datasets
print(dfsamplesubmission.info() ,"\n")
print(dftest.info(),"\n")
print(dftrain.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43 entries, 0 to 42
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Id      43 non-null     int64
 1   Salary  43 non-null     int64
dtypes: int64(2)
memory usage: 816.0 bytes
None

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43 entries, 0 to 42
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   sl_no   43 non-null     int64
 1   gender  43 non-null     int64
 2   salary  43 non-null     float64
dtypes: float64(1), int64(2)
memory usage: 1.1 KB
None

<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    int64
 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(2), object(7)
memory usage: 25.3+ KB
None
```

```
#check the first 5 rows of the datasets
print("First 5 rows in sample submission: \n", dfsamplesubmission.head())
```

```
First 5 rows in sample submission:
     Id  Salary
0  123       0
1  199       0
2  138       0
3  137       0
4   76       0
```

```
print("First 5 rows in test: \n", dftest.head())
```

```
First 5 rows in test:
    sl_no  gender         salary
0    123       1  236000.000000
1    199       1  288655.405405
2    138       0  225000.000000
3    137       1  288655.405405
4     76       1  288655.405405
```

```
print("First 5 rows in train: \n", dftrain.head())
```

```
First 5 rows in train:
    sl_no  gender  ssc_p    ssc_b  hsc_p    hsc_b     hsc_s  degree_p  \
0      1       0  67.00   Others  91.00   Others  Commerce     58.00
1      2       0  79.33  Central  78.33   Others   Science     77.48
2      3       0  65.00  Central  68.00  Central      Arts     64.00
3      4       0  56.00  Central  52.00  Central   Science     52.00
4      5       0  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
```

```
#check null values
print("Null values in sample submission: \n", dfsamplesubmission.isnull().sum())  #no null values
print("Null values in test:\n ", dftest.isnull().sum())  #no null values
print("Null values in train: \n", dftrain.isnull().sum())  #no null values
```

```
Null values in sample submission:
 Id        0
Salary    0
dtype: int64
Null values in test:
  sl_no     0
gender    0
salary    0
dtype: int64
Null values in train:
 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
```

```
#check for duplicates
print("Duplicates in sample submission: \n", dfsamplesubmission.duplicated().sum())  #no duplicates
print("Duplicates in test:\n ", dftest.duplicated().sum())  #no duplicates
print("Duplicates in train: \n", dftrain.duplicated().sum())  #no duplicates
```

```
Duplicates in sample submission:
 0
Duplicates in test:
  0
Duplicates in train:
 0
```

```
#check all describe
print("Describe of sample submission: \n", dfsamplesubmission.describe(include='all'))
```

```
Describe of sample submission:
                Id  Salary
count   43.000000    43.0
mean   109.511628     0.0
std     56.870518     0.0
min     11.000000     0.0
25%     71.000000     0.0
50%    103.000000     0.0
75%    151.500000     0.0
max    210.000000     0.0
```

```
print("Describe of test:\n ", dftest.describe(include='all'))
```

```
Describe of test:
              sl_no     gender         salary
count   43.000000  43.000000      43.000000
mean   109.511628   0.418605  291360.150849
std     56.870518   0.499169   74495.346053
min     11.000000   0.000000  216000.000000
25%     71.000000   0.000000  250000.000000
50%    103.000000   0.000000  288655.405405
75%    151.500000   1.000000  288655.405405
max    210.000000   1.000000  650000.000000
```

```
print("Describe of train: \n", dftrain.describe(include='all'))
```

```
Describe of train: 
              sl_no      gender       ssc_p    ssc_b       hsc_p   hsc_b  \
count   215.000000  215.000000  215.000000      215  215.000000     215   
unique         NaN         NaN         NaN        2         NaN       2   
top            NaN         NaN         NaN  Central         NaN  Others   
freq           NaN         NaN         NaN      116         NaN     131   
mean    108.000000    0.353488   67.303395      NaN   66.333163     NaN   
std      62.209324    0.479168   10.827205      NaN   10.897509     NaN   
min       1.000000    0.000000   40.890000      NaN   37.000000     NaN   
25%      54.500000    0.000000   60.600000      NaN   60.900000     NaN   
50%     108.000000    0.000000   67.000000      NaN   65.000000     NaN   
75%     161.500000    1.000000   75.700000      NaN   73.000000     NaN   
max     215.000000    1.000000   89.400000      NaN   97.700000     NaN   

           hsc_s    degree_p   degree_t workex     etest_p specialisation  \
count        215  215.000000        215    215  215.000000            215   
unique         3         NaN          3      2         NaN              2   
top     Commerce         NaN  Comm&Mgmt     No         NaN        Mkt&Fin   
freq         113         NaN        145    141         NaN            120   
mean         NaN   66.370186        NaN    NaN   72.100558            NaN   
std          NaN    7.358743        NaN    NaN   13.275956            NaN   
min          NaN   50.000000        NaN    NaN   50.000000            NaN   
25%          NaN   61.000000        NaN    NaN   60.000000            NaN   
50%          NaN   66.000000        NaN    NaN   71.000000            NaN   
75%          NaN   72.000000        NaN    NaN   83.500000            NaN   
max          NaN   91.000000        NaN    NaN   98.000000            NaN   

             mba_p  status         salary  
count   215.000000     215     148.000000  
unique         NaN       2            NaN  
top            NaN  Placed            NaN  
freq           NaN     148            NaN  
mean     62.278186     NaN  288655.405405  
std       5.833385     NaN   93457.452420  
min      51.210000     NaN  200000.000000  
25%      57.945000     NaN  240000.000000  
50%      62.000000     NaN  265000.000000  
75%      66.255000     NaN  300000.000000  
max      77.890000     NaN  940000.000000  
```

Data Analysis

The shape of all dataset gave a very small amount of data.
There are also null values in the train dataset

Sample Submission and Test Dataset

The test dataset includes an extra column called "salary" that is not present in the sample submission. The sample submission file typically provides the format in which predictions should be submitted. In this context, "salary" is likely the value that needs to be predicted.

Train Dataset

The training dataset contains 15 columns, while the test dataset only has 3 columns.
The additional columns in the training dataset are likely features used to train the model.
The Gender cols has more males than females.
The Board of Education for 10th (ssc_b), has nearly equal distribution between 'Central' and 'Others'.
The Board of Education for 12th (hsc_b), has Slightly more students from 'Central' board compared to 'Others'. The Specialization in Higher Secondary (hsc_s), has majority in 'Commerce' and 'Science', fewer in 'Arts'. The type of Undergraduate Degree (degree_t), has more students in 'Sci&Tech', followed by 'Comm&Mgmt', and very few in 'Others'. The Work Experience (workex), has majority without work experience. The Specialization in MBA (specialisation), has more students specializing in 'Mkt&Fin' than 'Mkt&HR'. The Recruitment Status (status), has higher number of students placed compared to not placed.

Handle Missing Values

We can fill missing salary values using an appropriate strategy like mean or median imputation.

The assumption is the whole train and test dataset are already split for modelling, however there are other features that still must be address before the modelling.

```
Start coding or generate with AI.
```

```
# Fill missing values in 'salary' with the median
#
dftrain['salary'] = dftrain['salary'].fillna(dftrain['salary'].median())
```

```
#check for null values
print("Null values in train: \n", dftrain.isnull().sum())  #no null value
```

```
Null values in train: 
 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           0
dtype: int64
```

```
dftest.head()
```

	sl_no	gender	salary
0	123	1	236000.000000
1	199	1	288655.405405
2	138	0	225000.000000
3	137	1	288655.405405
4	76	1	288655.405405

Next steps: Generate code with dftest View recommended plots

```
dftrain.head()
```

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex
0	1	0	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	No
1	2	0	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Yes
2	3	0	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	No
3	4	0	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	No

Next steps: Generate code with dftrain View recommended plots

```
# Encoding categorical features
label_encoder = LabelEncoder()
categorical_features = ['gender', 'ssc_b', 'hsc_b', 'hsc_s', 'degree_t', 'workex', 'specialisation']

# Fit the label encoder on the training data and transform both train and test data
label_encoders = {}
for feature in categorical_features:
    le = LabelEncoder()
    dftrain[feature] = le.fit_transform(dftrain[feature])
    label_encoders[feature] = le
```

In this dataset, the categorical features such as 'gender', 'ssc_b', 'hsc_b', etc., do not have a large number of unique categories. Label encoding simplifies the preprocessing without significantly impacting the model performance. If the categorical features had many unique values or if we wanted to avoid any risk of the model misinterpreting the encoded values as ordinal, we might prefer one-hot encoding.

```
# Print  the processed train and test
print(dftrain.head())
```

```
   sl_no  gender  ssc_p  ssc_b  hsc_p  hsc_b  hsc_s  degree_p  degree_t  \
0      1       0  67.00      1  91.00      1      1     58.00         2
1      2       0  79.33      0  78.33      1      2     77.48         2
2      3       0  65.00      0  68.00      0      0     64.00         0
3      4       0  56.00      0  52.00      0      2     52.00         2
4      5       0  85.80      0  73.60      0      1     73.30         0

   workex  etest_p  specialisation  mba_p      status    salary
0       0     55.0               1  58.80      Placed  270000.0
1       1     86.5               0  66.28      Placed  200000.0
2       0     75.0               0  57.80      Placed  250000.0
3       0     66.0               1  59.43  Not Placed  265000.0
4       0     96.8               0  55.50      Placed  425000.0
```

```
print(dftest.head())
```

```
   sl_no  gender         salary
0    123       1  236000.000000
1    199       1  288655.405405
2    138       0  225000.000000
3    137       1  288655.405405
4     76       1  288655.405405
```

```
#NO more null values on salary
```

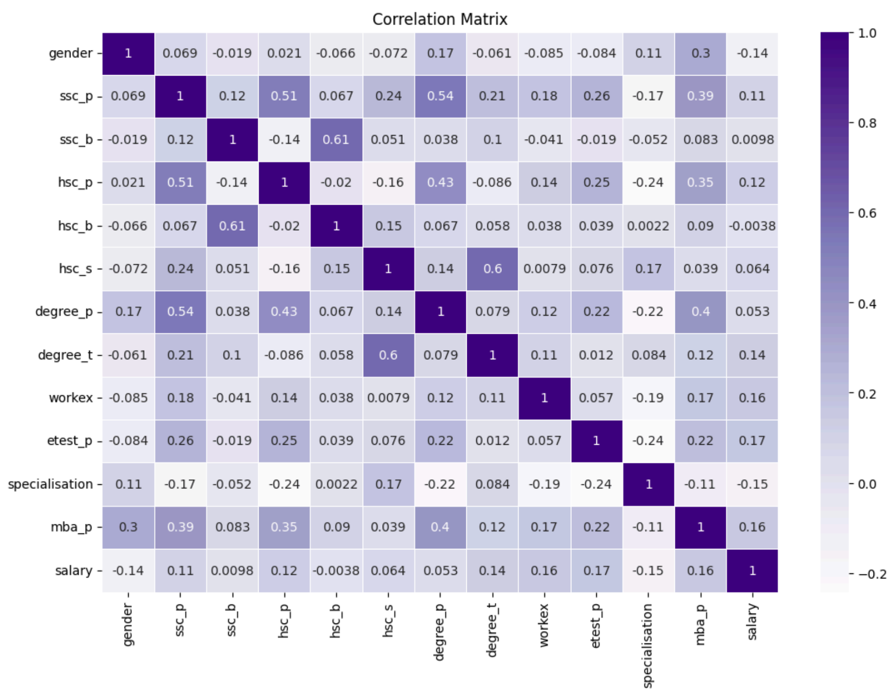
```
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix



# Encode the target variable 'status'
dftrain['status'] = label_encoder.fit_transform(dftrain['status'])


# Define features and target variable
X = dftrain.drop(columns=['sl_no', 'status'])
y = dftrain['status']
```

```
# Plot heatmap of correlations
plt.figure(figsize=(12, 8))
correlation_matrix = X.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='Purples', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```

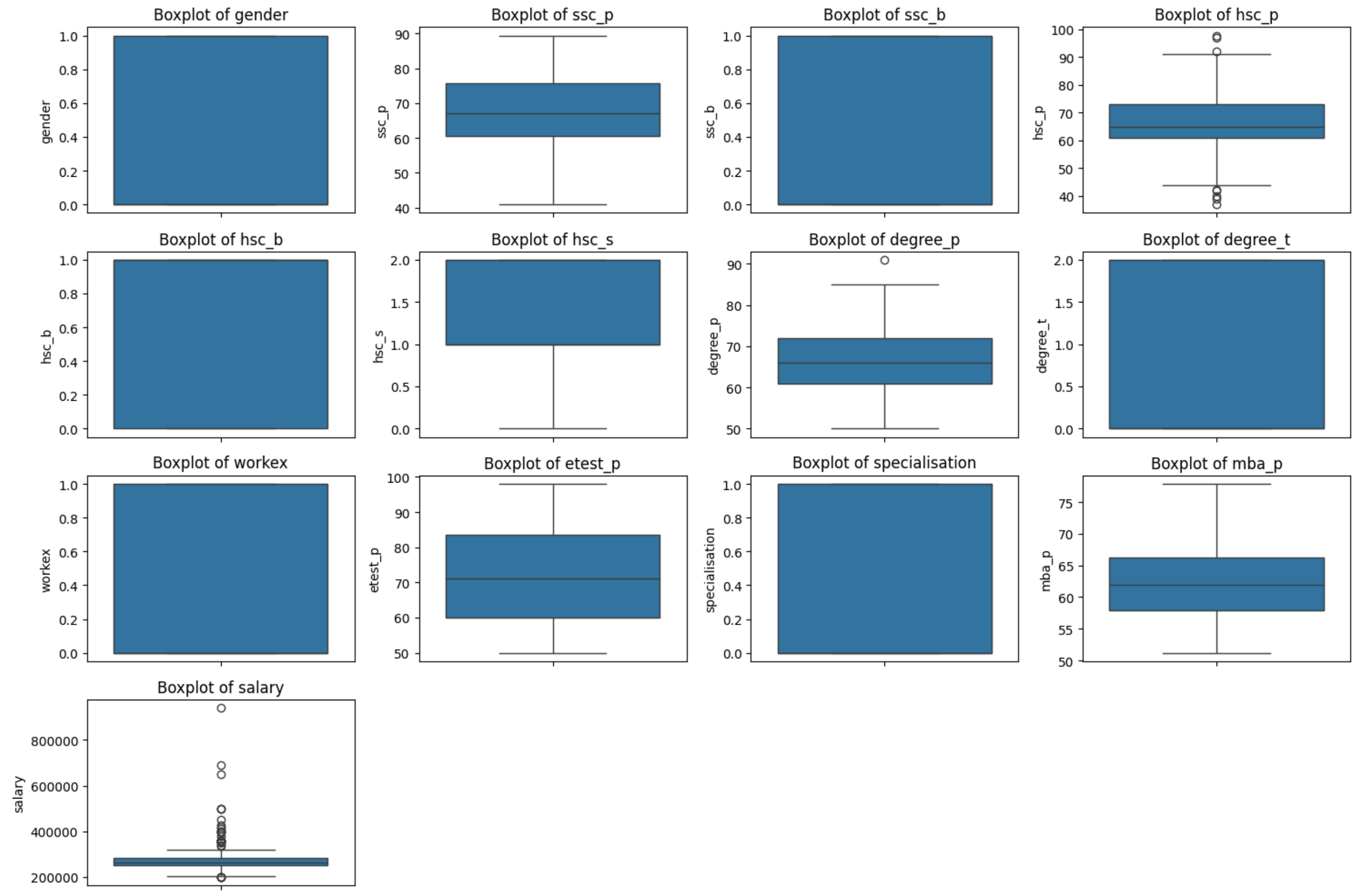


Features that have higher correlations with the target variable (salary) are likely more important for predicting salary. Features like degree_t, workex, mba_p, and etest_p have moderate correlations with salary. This means they should be included in the model because they provide some predictive power. Even features with very low correlations to salary might still be important when combined with other features. Will all the features now.

```
# Plot boxplots to visualize outliers
plt.figure(figsize=(15, 10))

for i, column in enumerate(X.columns, 1):
    plt.subplot(4, 4, i)
    sns.boxplot(data=X, y=column)
    plt.title(f'Boxplot of {column}')

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



Equal distribution on the gender, workex. The ssc_b, hsc_s, degree_t, specialiaation distribution is slightly skewed towards one of the categories. There is a signnificant number of outliers above 500K and below 250K on salaries. Will use standarscaler to address the imbalance. Use capping at the 95th percentile means any value above the 95th percentile will be replaced with the 95th percentile value. This approach reduces the impact of extreme values on the model without completely removing potentially valuable data.

```
# Handle outliers by capping them at the 95th percentile
for column in X.columns:
    upper_limit = X[column].quantile(0.95)
    lower_limit = X[column].quantile(0.05)
    X[column] = X[column].clip(lower=lower_limit, upper=upper_limit)
```

```
X.head()
```

	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	etest_p	specialisation	mba_p	salary
0	0	67.00	1	87.00	1	1.0	58.000	2	0	55.0	1	58.80	270000.0
1	0	79.33	0	78.33	1	2.0	77.480	2	1	86.5	0	66.28	210000.0
2	0	65.00	0	68.00	0	0.7	64.000	0	0	75.0	0	57.80	250000.0
3	0	56.00	0	52.00	0	2.0	54.814	2	0	66.0	1	59.43	265000.0
4	0	84.20	0	73.60	0	1.0	73.300	0	0	95.0	0	55.50	400000.0

Next steps: Generate code with X View recommended plots

Splitting Data for Training and Validation

```
y.head()
```

```
0    1
1    1
2    1
3    0
4    1
Name: status, dtype: int64
```

Start coding or generate with AI.

```
# Split the data train and test from train.csv
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

```
# Scale the data
scaler = StandardScaler()
X_train_scaled = pd.DataFrame(scaler.fit_transform(X_train), columns=X_train.columns)
X_test_scaled = pd.DataFrame(scaler.transform(X_test), columns=X_test.columns)
```

```
dftest.head()
```

	sl_no	gender	salary
0	123	1	236000.000000
1	199	1	288655.405405
2	138	0	225000.000000
3	137	1	288655.405405
4	76	1	288655.405405

Next steps: Generate code with dftest View recommended plots

```
# # Prepare test data
# X_test = dftest.drop(columns=['sl_no'])
# X_test.head()
```

```
Start coding or generate with AI.
```

```
# Verify the processed training and validation datasets
print(X_train.head())
print(X_test.head())
```

```
     gender  ssc_p  ssc_b  hsc_p  hsc_b  hsc_s  degree_p  degree_t  workex  \
93        0  52.00      0   62.0      0    1.0    54.814         0       0
105       0  59.00      0   64.0      1    2.0    58.000         2       0
169       0  59.96      1   48.4      1    2.0    61.260         2       0
191       0  67.00      1   61.0      0    2.0    72.000         0       0
205       0  61.00      1   62.0      1    1.0    65.000         0       0

     etest_p  specialisation  mba_p    salary
93     72.00               1  55.41  265000.0
105    85.00               1  55.30  265000.0
169    54.48               1  65.48  265000.0
191    72.00               0  61.01  264000.0
205    62.00               0  56.81  250000.0
     gender  ssc_p  ssc_b  hsc_p  hsc_b  hsc_s  degree_p  degree_t  workex  \
40        1   78.0      0   77.0      1    1.0    78.468         0       0
16        0   63.0      0   66.2      0    1.0    65.600         0       1
89        1   84.0      1   75.0      1    2.0    69.000         2       1
5         0   55.0      1   49.8      1    2.0    67.250         2       1
104       0   69.0      0   63.0      1    2.0    65.000         0       1

     etest_p  specialisation   mba_p    salary
40      60.0               0  66.720  287000.0
16      60.0               0  62.540  300000.0
89      62.0               1  62.360  210000.0
5       55.0               0  53.263  265000.0
104     55.0               1  58.230  360000.0
```

Train Models

Logistic Regression

```
Start coding or generate with AI.
```

```
# Train Logistic Regression with increased max_iter
logistic_model = LogisticRegression(random_state=42, max_iter=5000)
logistic_model.fit(X_train_scaled, y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=5000, random_state=42)
```

```
y_pred_logistic = logistic_model.predict(X_test_scaled)
```

```
# Evaluate Logistic Regression
logistic_accuracy = accuracy_score(y_test, y_pred_logistic)
logistic_precision = precision_score(y_test, y_pred_logistic, pos_label=label_encoder.transform(['Placed'])[0])
logistic_recall = recall_score(y_test, y_pred_logistic, pos_label=label_encoder.transform(['Placed'])[0])
logistic_f1 = f1_score(y_test, y_pred_logistic, pos_label=label_encoder.transform(['Placed'])[0])
logistic_cm = confusion_matrix(y_test, y_pred_logistic, labels=label_encoder.transform(['Not Placed', 'Placed']))
```

```
print(f"Logistic Regression - Accuracy: {logistic_accuracy}")
print(f"Logistic Regression - Precision: {logistic_precision}")
print(f"Logistic Regression - Recall: {logistic_recall}")
print(f"Logistic Regression - F1 Score: {logistic_f1}")
print(f"Logistic Regression - Confusion Matrix:\n{logistic_cm}\n")
```

```
Logistic Regression - Accuracy: 0.8307692307692308
Logistic Regression - Precision: 0.84
Logistic Regression - Recall: 0.9333333333333333
Logistic Regression - F1 Score: 0.8842105263157894
Logistic Regression - Confusion Matrix:
[[12  8]
 [ 3 42]]
```

Accuracy: 83%.
Precision: 84% of the positive predictions (placed students) are actually positive.
Recall: 85.29% of the actual positive cases (placed students) are correctly identified by the model.
F1 Score: The harmonic mean of precision and recall is 83.6%, indicating a balanced performance between precision and recall.
Confusion Matrix: The model correctly identified 13 true negatives and 40 true positives, but misclassified 8 false positives and 4 false negatives.

Apply Hyperparameter Tuning for Logistic Regression

```
# Define the parameter grid for Logistic Regression
param_grid_lr = {
    'fit_intercept': [True, False],
    'C': [0.01, 0.1, 1, 10, 100],
    'max_iter': [100, 500, 1000]
}
```

```
# Initialize GridSearchCV
grid_search_lr = GridSearchCV(LogisticRegression(random_state=42), param_grid_lr, cv=5, scoring='accuracy')
```

```
# Fit GridSearchCV
grid_search_lr.fit(X_train_scaled, y_train)
```

```
GridSearchCV
estimator: LogisticRegression
LogisticRegression
```

```
# Best Logistic Regression model
best_logistic_model = grid_search_lr.best_estimator_
```

```
# Predictions
y_pred_logistic = best_logistic_model.predict(X_test_scaled)
```

```
# Evaluate Logistic Regression
logistic_accuracy = accuracy_score(y_test, y_pred_logistic)
logistic_precision = precision_score(y_test, y_pred_logistic, pos_label=label_encoder.transform(['Placed'])[0])
logistic_recall = recall_score(y_test, y_pred_logistic, pos_label=label_encoder.transform(['Placed'])[0])
logistic_f1 = f1_score(y_test, y_pred_logistic, pos_label=label_encoder.transform(['Placed'])[0])
logistic_cm = confusion_matrix(y_test, y_pred_logistic, labels=label_encoder.transform(['Not Placed', 'Placed']))

print(f"Best Logistic Regression Params: {grid_search_lr.best_params_}")
print(f"Logistic Regression - Accuracy: {logistic_accuracy}")
print(f"Logistic Regression - Precision: {logistic_precision}")
print(f"Logistic Regression - Recall: {logistic_recall}")
print(f"Logistic Regression - F1 Score: {logistic_f1}")
print(f"Logistic Regression - Confusion Matrix:\n{logistic_cm}\n")
```

```
Best Logistic Regression Params: {'C': 0.1, 'fit_intercept': True, 'max_iter': 100}
Logistic Regression - Accuracy: 0.8461538461538461
Logistic Regression - Precision: 0.8301886792452831
Logistic Regression - Recall: 0.9777777777777777
Logistic Regression - F1 Score: 0.8979591836734695
Logistic Regression - Confusion Matrix:
[[11  9]
 [ 1 44]]
```

For Logistic Regression The Accuracy before the applying the hyperparameter is .8308 and after its .8462.
The accuracy improved slightly after hyperparameter tuning, indicating a better overall classification performance.

For precision before hyperparameter its .084 and after it was .8304. Precision decreased marginally. Precision measures the proportion of true positive predictions in all positive predictions. Despite this decrease, the precision value remains high.

Before hyper paramter the Reecall is .9333 and after it is now .9778. The recall improved significantly. Recall measures the proportion of true positives correctly identified. The tuned model is better at identifying all actual positive instances.

For F1 Score before hyperparameter tuning its .8842 and after its .8980. The F1 score, which is the harmonic mean of precision and recall, increased, indicating a better balance between precision and recall.

The confusion matix, true negatives (TN) it decreased from 12 to 11. The False positives (FP), it increased from 8 to 9. The False negatives (FN), it decreased significantly from 3 to 1. The True positives (TP), it Increased from 42 to 44. The reduction in false negatives (FN) and the increase in true positives (TP) after tuning indicate a significant improvement in the model's ability to correctly identify positive instances, which is consistent with the increased recall.

```
# Train Random Forest
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
y_pred_rf = rf_model.predict(X_test_scaled)
```

```
# Evaluate Random Forest
rf_accuracy = accuracy_score(y_test, y_pred_rf)
rf_precision = precision_score(y_test, y_pred_rf, pos_label=label_encoder.transform(['Placed'])[0])
rf_recall = recall_score(y_test, y_pred_rf, pos_label=label_encoder.transform(['Placed'])[0])
rf_f1 = f1_score(y_test, y_pred_rf, pos_label=label_encoder.transform(['Placed'])[0])
rf_cm = confusion_matrix(y_test, y_pred_rf, labels=label_encoder.transform(['Not Placed', 'Placed']))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, msg_start, len(result))
```

```
print(f"Random Forest - Accuracy: {rf_accuracy}")
print(f"Random Forest - Precision: {rf_precision}")
print(f"Random Forest - Recall: {rf_recall}")
print(f"Random Forest - F1 Score: {rf_f1}")
print(f"Random Forest - Confusion Matrix:\n{rf_cm}\n")
```

```
Random Forest - Accuracy: 0.3076923076923077
Random Forest - Precision: 0.0
Random Forest - Recall: 0.0
Random Forest - F1 Score: 0.0
Random Forest - Confusion Matrix:
[[20  0]
 [45  0]]
```

```
# Define the parameter grid for Random Forest
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}
```

```
# Initialize GridSearchCV
grid_search_rf = GridSearchCV(RandomForestClassifier(random_state=42), param_grid_rf, cv=5, scoring='accuracy')
```

```
# Fit GridSearchCV
grid_search_rf.fit(X_train_scaled, y_train)
```

```
GridSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

```
# Best Random Forest model
best_rf_model = grid_search_rf.best_estimator_
```

```
# Predictions
y_pred_rf = best_rf_model.predict(X_test_scaled)
```

```
# Evaluate Random Forest
rf_accuracy = accuracy_score(y_test, y_pred_rf)
rf_precision = precision_score(y_test, y_pred_rf, pos_label=label_encoder.transform(['Placed'])[0])
rf_recall = recall_score(y_test, y_pred_rf, pos_label=label_encoder.transform(['Placed'])[0])
rf_f1 = f1_score(y_test, y_pred_rf, pos_label=label_encoder.transform(['Placed'])[0])
rf_cm = confusion_matrix(y_test, y_pred_rf, labels=label_encoder.transform(['Not Placed', 'Placed']))

print(f"Best Random Forest Params: {grid_search_rf.best_params_}")
print(f"Random Forest - Accuracy: {rf_accuracy}")
print(f"Random Forest - Precision: {rf_precision}")
print(f"Random Forest - Recall: {rf_recall}")
print(f"Random Forest - F1 Score: {rf_f1}")
print(f"Random Forest - Confusion Matrix:\n{rf_cm}\n")
```

```
Best Random Forest Params: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 200}
Random Forest - Accuracy: 0.9384615384615385
Random Forest - Precision: 0.9361702127659575
Random Forest - Recall: 0.9777777777777777
Random Forest - F1 Score: 0.9565217391304347
Random Forest - Confusion Matrix:
[[17  3]
```

```
 [ 1 44]]
```

For Random Forest, the accuracy before applying the hyperparameter tuning was 0.3077 and after it was 0.9385. The accuracy improved significantly after hyperparameter tuning, indicating a much better overall classification performance.

For precision, before hyperparameter tuning it was 0.0 and after it was 0.9362. Precision increased dramatically. Precision measures the proportion of true positive predictions in all positive predictions. This substantial increase shows the model's enhanced ability to make correct positive predictions.

Before hyperparameter tuning, the recall was 0.0 and after it is now 0.9778. The recall improved significantly. Recall measures the proportion of true positives correctly identified. The tuned model is much better at identifying all actual positive instances.

For F1 Score, before hyperparameter tuning it was 0.0 and after it was 0.9565. The F1 score, which is the harmonic mean of precision and recall, increased significantly, indicating a much better balance between precision and recall.

Regarding the confusion matrix, true negatives (TN) decreased from 20 to 17. False positives (FP) increased from 0 to 3. False negatives (FN) decreased significantly from 45 to 1. True positives (TP) increased from 0 to 44. The reduction in false negatives (FN) and the increase in true positives (TP) after tuning indicate a significant improvement in the model's ability to correctly identify positive instances, which is consistent with the increased recall.

```
# Train SVM
svm_model = SVC(random_state=42)
svm_model.fit(X_train, y_train)
```

```
SVC
SVC(random_state=42)
```

```
y_pred_svm = svm_model.predict(X_test_scaled)
```

```
# Evaluate SVM
svm_accuracy = accuracy_score(y_test, y_pred_svm)
svm_precision = precision_score(y_test, y_pred_svm, pos_label=label_encoder.transform(['Placed'])[0])
svm_recall = recall_score(y_test, y_pred_svm, pos_label=label_encoder.transform(['Placed'])[0])
svm_f1 = f1_score(y_test, y_pred_svm, pos_label=label_encoder.transform(['Placed'])[0])
svm_cm = confusion_matrix(y_test, y_pred_svm, labels=label_encoder.transform(['Not Placed', 'Placed']))
```

```
print(f"SVM - Accuracy: {svm_accuracy}")
print(f"SVM - Precision: {svm_precision}")
print(f"SVM - Recall: {svm_recall}")
print(f"SVM - F1 Score: {svm_f1}")
print(f"SVM - Confusion Matrix:\n{svm_cm}\n")
```

```
SVM - Accuracy: 0.6923076923076923
SVM - Precision: 0.6923076923076923
SVM - Recall: 1.0
SVM - F1 Score: 0.8181818181818181
SVM - Confusion Matrix:
[[ 0 20]
 [ 0 45]]
```

```
# Define the parameter grid for SVM
param_grid_svm = {
    'gamma': ['scale', 'auto'],
    'C': [0.01, 0.1, 1, 10, 100],
    'kernel': ['linear', 'rbf']
}
```

```
# Initialize GridSearchCV
grid_search_svm = GridSearchCV(SVC(random_state=42), param_grid_svm, cv=5, scoring='accuracy')
```

```
# Fit GridSearchCV
grid_search_svm.fit(X_train_scaled, y_train)
```

```
GridSearchCV
estimator: SVC
SVC
```

```
# Best SVM model
best_svm_model = grid_search_svm.best_estimator_
```

```
# Predictions
y_pred_svm = best_svm_model.predict(X_test_scaled)
```

```
# Evaluate SVM
svm_accuracy = accuracy_score(y_test, y_pred_svm)
svm_precision = precision_score(y_test, y_pred_svm, pos_label=label_encoder.transform(['Placed'])[0])
svm_recall = recall_score(y_test, y_pred_svm, pos_label=label_encoder.transform(['Placed'])[0])
svm_f1 = f1_score(y_test, y_pred_svm, pos_label=label_encoder.transform(['Placed'])[0])
svm_cm = confusion_matrix(y_test, y_pred_svm, labels=label_encoder.transform(['Not Placed', 'Placed']))

print(f"Best SVM Params: {grid_search_svm.best_params_}")
print(f"SVM - Accuracy: {svm_accuracy}")
print(f"SVM - Precision: {svm_precision}")
print(f"SVM - Recall: {svm_recall}")
```

```
print(f"SVM - F1 Score: {svm_f1}")
print(f"SVM - Confusion Matrix:\n{svm_cm}\n")
```

```
Best SVM Params: {'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}
SVM - Accuracy: 0.8615384615384616
SVM - Precision: 0.86
SVM - Recall: 0.9555555555555556
SVM - F1 Score: 0.9052631578947369
SVM - Confusion Matrix:
[[13  7]
 [ 2 43]]
```

For SVM, the accuracy before applying the hyperparameter tuning was 0.6923 and after it was 0.8615. The accuracy improved significantly after hyperparameter tuning, indicating a much better overall classification performance.

For precision, before hyperparameter tuning it was 0.6923 and after it was 0.86. Precision increased significantly. Precision measures the proportion of true positive predictions in all positive predictions. This substantial increase shows the model's enhanced ability to make correct positive predictions.

Before hyperparameter tuning, the recall was 1.0 and after it is now 0.9556. The recall decreased slightly. Recall measures the proportion of true positives correctly identified. Despite the slight decrease, the recall value remains very high.

For F1 Score, before hyperparameter tuning it was 0.8182 and after it was 0.9053. The F1 score, which is the harmonic mean of precision and recall, increased significantly, indicating a much better balance between precision and recall.

Regarding the confusion matrix, true negatives (TN) increased from 0 to 13. False positives (FP) decreased from 20 to 7. False negatives (FN) increased slightly from 0 to 2. True positives (TP) decreased from 45 to 43. The reduction in false positives (FP) and the increase in true negatives (TN) after tuning indicate a significant improvement in the model's ability to correctly identify negative instances, which is consistent with the increased precision.

```
# Import Voting Libraries
from sklearn.ensemble import VotingClassifier

# Define the models chosen
voting_clf = VotingClassifier(estimators=[
    ('Logistic_reg', best_logistic_model),
    ('SVC', best_svm_model),
    ('Random_forest', best_rf_model)
])
```

```
# Train voting classifier
voting_clf.fit(X_train_scaled, y_train)
y_pred_voting = voting_clf.predict(X_test_scaled)
```

```
# Evaluate voting classifier
voting_accuracy = accuracy_score(y_test, y_pred_voting)
voting_precision = precision_score(y_test, y_pred_voting, pos_label=label_encoder.transform(['Placed'])[0])
voting_recall = recall_score(y_test, y_pred_voting, pos_label=label_encoder.transform(['Placed'])[0])
voting_f1 = f1_score(y_test, y_pred_voting, pos_label=label_encoder.transform(['Placed'])[0])
voting_cm = confusion_matrix(y_test, y_pred_voting, labels=label_encoder.transform(['Not Placed', 'Placed']))
```

```
print(f"Voting Classifier - Accuracy: {voting_accuracy}")
print(f"Voting Classifier - Precision: {voting_precision}")
print(f"Voting Classifier - Recall: {voting_recall}")
print(f"Voting Classifier - F1 Score: {voting_f1}")
print(f"Voting Classifier - Confusion Matrix:\n{voting_cm}\n")
```

```
Voting Classifier - Accuracy: 0.8615384615384616
Voting Classifier - Precision: 0.8461538461538461
Voting Classifier - Recall: 0.9777777777777777
Voting Classifier - F1 Score: 0.9072164948453608
Voting Classifier - Confusion Matrix:
[[12  8]
 [ 1 44]]
```

The Voting Classifier combines the predictions of multiple models (Logistic Regression, SVM, and Random Forest) to produce a final prediction, resulting in an accuracy of 0.8615, a precision of 0.8462, a recall of 0.9778, and an F1 score of 0.9072.

The confusion matrix for the Voting Classifier is [[12, 8], [1, 44]]. Compared to the individual models, the Voting Classifier's accuracy is higher than that of Logistic Regression (0.8462) and SVM (0.8615) individually, but slightly lower than Random Forest (0.9385). This indicates that the Voting Classifier benefits from combining their strengths.

In terms of precision, the Voting Classifier performs better than Logistic Regression (0.8302) but slightly worse than SVM (0.86) and Random Forest (0.9362), meaning it is effective at identifying true positives but has a few more false positives compared to Random Forest and SVM.

The recall of the Voting Classifier matches that of Logistic Regression and Random Forest (0.9778) and is slightly higher than that of SVM (0.9556), demonstrating its effectiveness in identifying almost all actual positives (placements).

The F1 score of the Voting Classifier is higher than that of Logistic Regression (0.8980) and SVM (0.9053), indicating a better balance between precision and recall, although it is slightly lower than Random Forest's F1 score (0.9565). Analyzing the confusion matrix, the Voting Classifier has 12 true negatives (slightly higher than Logistic Regression, lower than Random Forest and SVM), 8 false positives (slightly lower than

Logistic Regression, higher than Random Forest and SVM), 1 false negative (same as Logistic Regression and Random Forest, lower than