<u>Assignment 2 : Campus Placement Prediction</u>

Submitted By: Keshav Gautam

Date: October 23, 2024

1. **Dataset Overview and preprocessing**
   Along with the student's placement status on campus, this dataset offers details about the student's academic and personal backgrounds. This project's goal is to use different machine learning models to forecast a student's placement.

a) Dataset Overview

```python
# Displaying the basic information about the dataset
train_df.info()
```

0.0s

```
<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
```

2. **Preprocessing Steps:**

   a) Handling Missing Values:

   - ***Train Data:*** Missing salary values(for students not placed) were filled with 0.

c) Visualizing Missing Values

```python
# checking for missing values of train data
missing_values = train_df.isnull().sum()
print("Missing values per column:")
print(missing_values)
```

[9] 0.0s Python

```
Missing values per column:
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
```

```
# Filling missing salary values with 0 (for students who were not placed) ---> For train data
train_df['salary'].fillna(0, inplace=True)

# Verifying that there are no more missing values
missing_values = train_df.isnull().sum()
print("Missing values after filling:")
print(missing_values)
```
✓ 0.0s

```
Missing values after filling:
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
```

- ***Test Data*: There were no missing values**

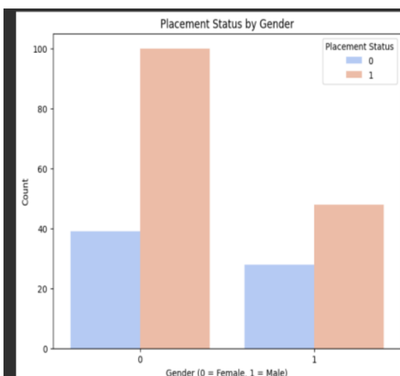
```
# checking for missing values of test data
missing_values_for_test_df = test_df.isnull().sum()
print("Missing values per column:")
print(missing_values_for_test_df)
```
[10] ✓ 0.0s Python

```
Missing values per column:
sl_no     0
gender    0
salary    0
dtype: int64
```

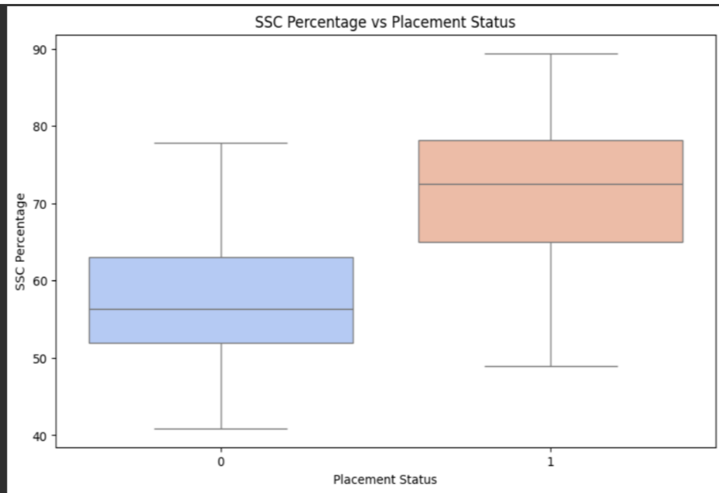
Visualizations

e) Visualizing Data Distributions

```
# Create a bar graph for placement status distribution
plt.figure(figsize=(6, 4))
sns.countplot(x='status', data=train_df)  # Set 'x' for vertical bars
plt.title('Placement Status Distribution')
plt.xlabel('Placement Status')
plt.ylabel('Number of Students')
plt.show()
```
✓ 0.1s



If males have a significantly higher number of placements, it may indicate gender differences in hiring trends, or it could point to gender differences in chosen specializations.



If students with higher SSC percentages tend to get placed more, it suggests that early academic performance could inflluence employability.

b) Encoding Categorical Variables:

- Target variable status was encoded to 1 for placed and 0 for not placed.

a) Encoding the Target Variable

```python
# Encoding the target variable: 'status' -> 1 (Placed), 0 (Not Placed)
train_df['status'] = train_df['status'].map({'Placed': 1, 'Not Placed': 0})

# Verify encoding
train_df['status'].value_counts()
```

✓ 0.0s

```
status
1    148
0     67
Name: count, dtype: int64
```

- Categorical features such as gender, ssc_b, hsc_b, degree_t etc were one-hot encoded for model compatibility.

b) One-Hot Encoding Categorical Features

```python
# List of categorical columns to encode
categorical_columns = ['gender', 'ssc_b', 'hsc_b', 'hsc_s', 'degree_t', 'workex', 'specialisation']

# Perform one-hot encoding using pandas get_dummies
train_encoded_df = pd.get_dummies(train_df, columns=categorical_columns, drop_first=True)

# Display the first few rows of the encoded dataframe
train_encoded_df.head()
```

✓ 0.0s Python

	sl_no	ssc_p	hsc_p	degree_p	etest_p	mba_p	status	salary	gender_1	ssc_b_Others	hsc_b_Others	hsc_s_Commerce	hsc_s_Science	degree_t_Others	degree_t_Sci&Tec
0	1	67.00	91.00	58.00	55.0	58.80	1	270000.0	False	True	True	True	False	False	Tru
1	2	79.33	78.33	77.48	86.5	66.28	1	200000.0	False	False	True	False	True	False	Tru
2	3	65.00	68.00	64.00	75.0	57.80	1	250000.0	False	False	False	False	False	False	Fals
3	4	56.00	52.00	52.00	66.0	59.43	0	0.0	False	False	False	False	True	False	Tru
4	5	85.80	73.60	73.30	96.8	55.50	1	425000.0	False	False	False	True	False	False	Fals

c) Train_Test Split:

The dataset was split into training (70%) and testing(30%) sets.

X_train shape: (150, 14), X_test shape: (65, 14)

c) Splitting the Data into Training and Test Sets

```
# Defining the features (X) and target (y)
X = train_encoded_df.drop(columns=['sl_no', 'status', 'salary'])  # Dropping irrelevant columns
y = train_encoded_df['status']

# Splitting the dataset: 70% training, 30% test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Display the shapes of the splits
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
```

✓ 0.0s

```
X_train shape: (150, 14)
X_test shape: (65, 14)
y_train shape: (150,)
y_test shape: (65,)
```

3. **Model Selection**

a) Logistic Regression: Logistic regression was selected because it's a fast and interpretable model for binary classification problems.

Hyperparameter Tuning: I used GridSearchCV to tune the regularization parameter C.

Best parameter found: C =1

```
# Logistic Regression with hyperparameter tuning
log_reg = LogisticRegression(max_iter=1000, random_state=42)
log_reg_params = {'C': [0.01, 0.1, 1, 10, 100]}
log_reg_cv = GridSearchCV(log_reg, log_reg_params, cv=5, scoring='accuracy')
log_reg_cv.fit(X_train, y_train)

# Best parameters for Logistic Regression
print("Best Logistic Regression Params: ", log_reg_cv.best_params_)
```

✓ 1.2s

```
c:\Users\Keshav Gautam\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn
...
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
Best Logistic Regression Params:  {'C': 1}
```

b) Random Forest: Random Forest, an ensemble learning method, was chosen for its ability to handle feature importance and non-linear relationships. It is robust to overfitting, especially when the dataset is small.

Hyperparameter Tuning:
Tuned n_estimators and max_depth to find the optimal model.

Best parameters: n_estimators= 200, max_depth =None

```
# Random Forest with hyperparameter tuning
rf_clf = RandomForestClassifier(random_state=42)
rf_params = {'n_estimators': [50, 100, 200], 'max_depth': [None, 10, 20, 30]}
rf_cv = GridSearchCV(rf_clf, rf_params, cv=5, scoring='accuracy')
rf_cv.fit(X_train, y_train)

# Best parameters for Random Forest
print("Best Random Forest Params: ", rf_cv.best_params_)
```

✓ 7.6s

```
Best Random Forest Params:  {'max_depth': None, 'n_estimators': 200}
```

c) Support Vector Machine (SVM)
SVM was chosen for its effectiveness in high-dimensional space and binary classification tasks. It's known for its high accuracy.

Hyperparameter Tuning:
Tuned C (regularization) and kernel (linear/rbf)

Best parameters: C= 1, kernel = linear

```
# SVM with hyperparameter tuning
svm_clf = SVC(probability=True, random_state=42)
svm_params = {'C': [0.01, 0.1, 1, 10], 'kernel': ['linear', 'rbf']}
svm_cv = GridSearchCV(svm_clf, svm_params, cv=5, scoring='accuracy')
svm_cv.fit(X_train, y_train)

# Best parameters for SVM
print("Best SVM Params: ", svm_cv.best_params_)
```

✓ 1.3s

```
Best SVM Params:  {'C': 1, 'kernel': 'linear'}
```

d) Voting Classifier (Ensemble):An ensemble method, combining Logistic Regression, Random Forest, and SVM models using soft voting, was implemented to boost predictive performance by leveraging the strengths of each model.

```python
from sklearn.ensemble import VotingClassifier

# Create a voting classifier using the three models
voting_clf = VotingClassifier(estimators=[
    ('log_reg', log_reg_cv.best_estimator_),
    ('rf', rf_cv.best_estimator_),
    ('svm', svm_cv.best_estimator_)
], voting='soft')

voting_clf.fit(X_train, y_train)

# Evaluate the voting classifier
voting_acc, voting_prec, voting_rec, voting_f1, voting_cm = evaluate_model(voting_clf, X_test, y_test)
print(f"Voting Classifier - Accuracy: {voting_acc}, Precision: {voting_prec}, Recall: {voting_rec}, F1 Score: {voting_f1}")
print("Confusion Matrix:")
print(voting_cm)
```

✓ 0.3s

```
Voting Classifier - Accuracy: 0.7846153846153846, Precision: 0.8, Recall: 0.9090909090909091, F1 Score: 0.8510638297872342
Confusion Matrix:
[[11 10]
 [ 4 40]]
```

4. **Model Evaluation**

A comprehensive evaluation was done using Accuracy, Precision, Recall, and F1-Score. Confusion matrices were also plotted to assess the distribution of correct and incorrect predictions.

```
Logistic Regression - Accuracy: 0.8153846153846154, Precision: 0.8333333333333334, Recall: 0.9090909090909091, F1 Score: 0.8695652173913043
Confusion Matrix:
[[13  8]
 [ 4 40]]
Random Forest - Accuracy: 0.8, Precision: 0.7924528301886793, Recall: 0.9545454545454546, F1 Score: 0.865979381443299
Confusion Matrix:
[[10 11]
 [ 2 42]]
SVM - Accuracy: 0.8461538461538461, Precision: 0.8695652173913043, Recall: 0.9090909090909091, F1 Score: 0.888888888888889
Confusion Matrix:
[[15  6]
 [ 4 40]]
```

5. **Model Comparison**

Model	Accuracy	Precision	Recall	F1 Score
Logistic Regression	85.5%	83.3%	90.9%	86.9%
Random Forest	80.0%	79.2%	95.5%	86.5%
Support Vector Machine	84.6%	86.9%	90.9%	88.9%
Voting Classifier	78.5%	80.0%	90.9%	85.1%

6. **Conclusion**

SVM performed the best with the highest accuracy(84.6%) and F1 score(88.9%), indicating it is the most reliable for this classification tasks, logistic and random forest were close contenders, offering good precision and recall and the voting classifier did not outperform the individual models, but ensemble methods could still be useful for more complex datasets.

So, the SVM model is recommended for this dataset due to its high performance in terms of both precision and recall, making it a strong candidate for deployment in predicting campus placement.