Placement Classification Using Feature Selection & Machine Learning

To make prediction on college student's placement status

Prepared by: Rupayan Dey

Objective:

The project *"Placement Classification Using Feature Selection & Machine Learning,"* is to analyze student-related academic and skill-based factors, visualize patterns within the placement dataset, and identify the most significant features influencing placement outcomes. By applying effective feature selection and engineering techniques, the project aims to reduce irrelevant variables, create more meaningful inputs, and enhance the overall predictive capability of the model. Machine learning classification algorithms are trained and evaluated using metrics such as accuracy, precision, recall, and F1-score to determine the most reliable model. Ultimately, the goal is to develop a robust and data-driven placement prediction system that helps institutions understand student employability and supports students in improving their placement readiness.

Why choose this project:

This project is valuable because placement prediction is a real-world problem faced by almost every educational institution, and solving it provides meaningful insights for both students and administrators. With the rising importance of data-driven decision-making, using machine learning to understand which factors truly influence placement outcomes helps colleges improve training programs, identify skill gaps, and support students more effectively. At the same time, students gain a clearer understanding of how their academic performance, skills, and experiences impact employability. The project also offers hands-on experience in essential ML concepts such as feature selection, visualization, classification modeling, and evaluation metrics—making it highly relevant for academic learning, resume building, and practical applications in campus analytics. Overall, this project combines real impact, modern technology, and practical skill development, making it an ideal choice.

About Dataset:[1]

This dataset data representing engineering students from Indian colleges. It is designed for machine learning, data analysis, and educational research practice. Each row represents a student and includes academic performance, technical skills, lifestyle factors, and placement outcomes. Logical relationships have been built into the data to make it realistic and useful for predictive modeling.

This dataset is ideal for:
• Placement prediction models
• Salary regression models
• Feature importance analysis
• Student performance analytics

This dataset contains detailed academic, personal, behavioural, and skill-related attributes of engineering students.
These features are used as inputs for training the ML model.
Key Feature Categories
A. Academic Performance

- cgpa
- tenth_percentage
- twelfth_percentage
- backlogs

B. Study Behaviour & Engagement

- study_hours_per_day
- attendance_percentage
- projects_completed
- internships_completed
- technical_skill_rating
- communication_skill_rating

C. Aptitude & Practical Exposure

- aptitude_skill_rating
- hackathons_participated
- certifications_count

D. Extra Skills & Activities

- co_curricular_participation
- coding_skills
- problem_solving_skills
- logical_reasoning_skill
- extracurricular_involvement

E. Socio-Personal Factors

- gender
- branch
- hostel_resident
- internet_access

[1] The link of the data set is link

Code Implementation:

Let’s drive in the code implementation section ,This section presents the core Python code used for data processing, model training, and evaluation. The implementation uses scikit-learn for machine learning and pandas for data manipulation.

Environment Setup

The project requires standard data science libraries: NumPy and pandas for data handling, matplotlib and seaborn for visualization, and scikit-learn with XGBoost for machine learning.

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from sklearn.preprocessing import StandardScaler, LabelEncoder
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, roc_auc_score, roc_curve
from xgboost import XGBClassifier
```

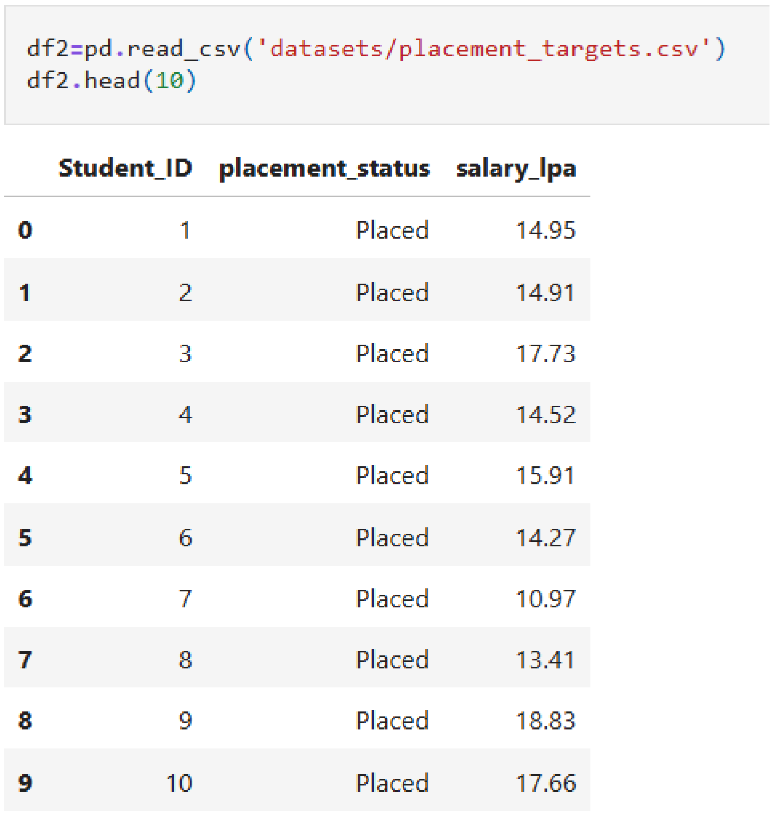
Data Loading:

This section works with the data loading from the csv from the csv files.

```python
df = pd.read_csv('datasets/indian_engineering_student_placement.csv')
df.head(10)
```

	Student_ID	gender	branch	cgpa	tenth_percentage	twelfth_percentage	backlogs	study_hours_per_day	attendance_percentage	projects_completed	...
0	1	Male	ECE	8.74	74.0	75.0	0	3.8	71.1	7	...
1	2	Female	ECE	7.80	75.3	69.7	0	6.3	69.5	5	...
2	3	Female	IT	6.95	62.8	68.3	0	1.5	62.5	8	...
3	4	Male	ECE	7.46	57.9	51.4	1	4.7	64.6	6	...
4	5	Male	IT	6.86	61.3	73.5	2	5.2	75.9	3	...
5	6	Male	CSE	7.46	70.5	70.4	3	3.6	81.7	5	...
6	7	Male	CE	8.26	81.2	74.8	0	1.8	59.6	4	...
7	8	Female	IT	7.01	63.9	61.2	0	2.4	64.7	7	...
8	9	Female	CSE	7.97	71.4	73.8	1	7.6	75.5	7	...
9	10	Female	ECE	8.79	70.1	77.1	0	5.9	77.2	7	...

10 rows × 23 columns

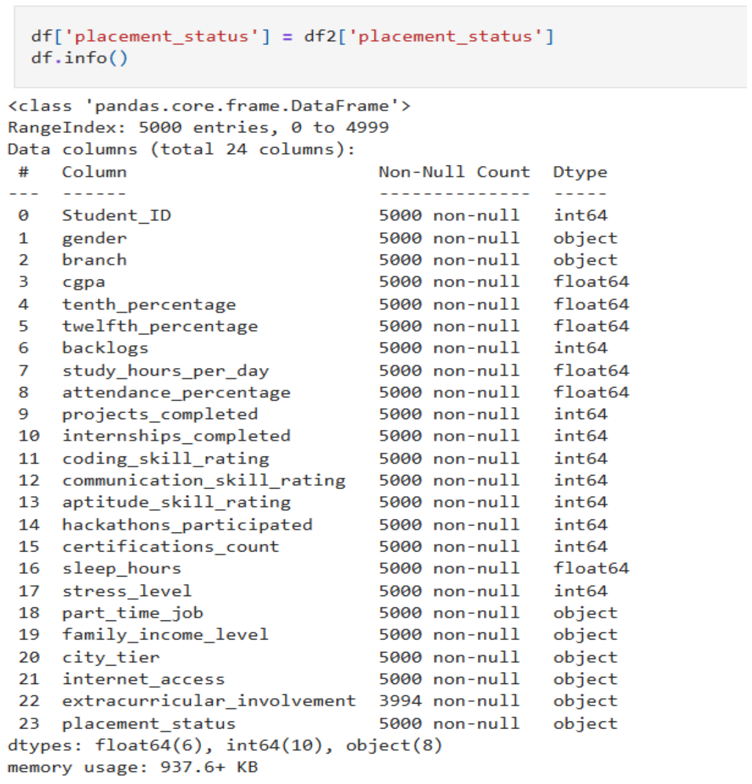


```python
df2=pd.read_csv('datasets/placement_targets.csv')
df2.head(10)
```

	Student_ID	placement_status	salary_lpa
0	1	Placed	14.95
1	2	Placed	14.91
2	3	Placed	17.73
3	4	Placed	14.52
4	5	Placed	15.91
5	6	Placed	14.27
6	7	Placed	10.97
7	8	Placed	13.41
8	9	Placed	18.83
9	10	Placed	17.66

Dataset Overview :

Using df.info(), the dataset's composition—numeric, categorical, and binary columns—is reviewed to understand data cleanliness and type-specific preprocessing needs, similar to the method applied in the reference fraud detection report



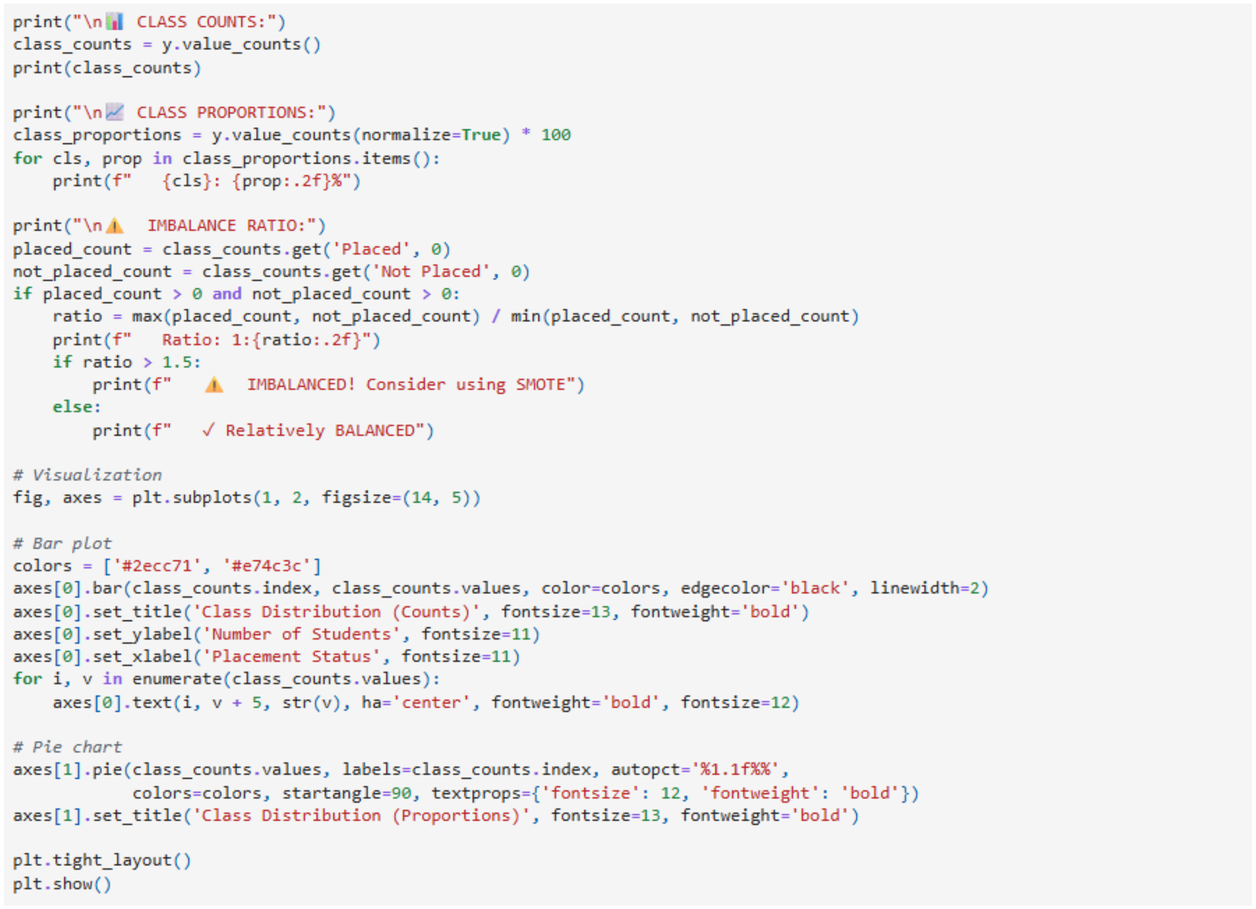
```python
df['placement_status'] = df2['placement_status']
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 24 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   Student_ID                  5000 non-null   int64  
 1   gender                      5000 non-null   object 
 2   branch                      5000 non-null   object 
 3   cgpa                        5000 non-null   float64
 4   tenth_percentage            5000 non-null   float64
 5   twelfth_percentage          5000 non-null   float64
 6   backlogs                    5000 non-null   int64  
 7   study_hours_per_day         5000 non-null   float64
 8   attendance_percentage       5000 non-null   float64
 9   projects_completed          5000 non-null   int64  
 10  internships_completed       5000 non-null   int64  
 11  coding_skill_rating         5000 non-null   int64  
 12  communication_skill_rating  5000 non-null   int64  
 13  aptitude_skill_rating       5000 non-null   int64  
 14  hackathons_participated     5000 non-null   int64  
 15  certifications_count        5000 non-null   int64  
 16  sleep_hours                 5000 non-null   float64
 17  stress_level                5000 non-null   int64  
 18  part_time_job               5000 non-null   object 
 19  family_income_level         5000 non-null   object 
 20  city_tier                   5000 non-null   object 
 21  internet_access             5000 non-null   object 
 22  extracurricular_involvement  3994 non-null   object 
 23  placement_status            5000 non-null   object 
dtypes: float64(6), int64(10), object(8)
memory usage: 937.6+ KB
```

Statement on Extracurricular Activity Missing Values :

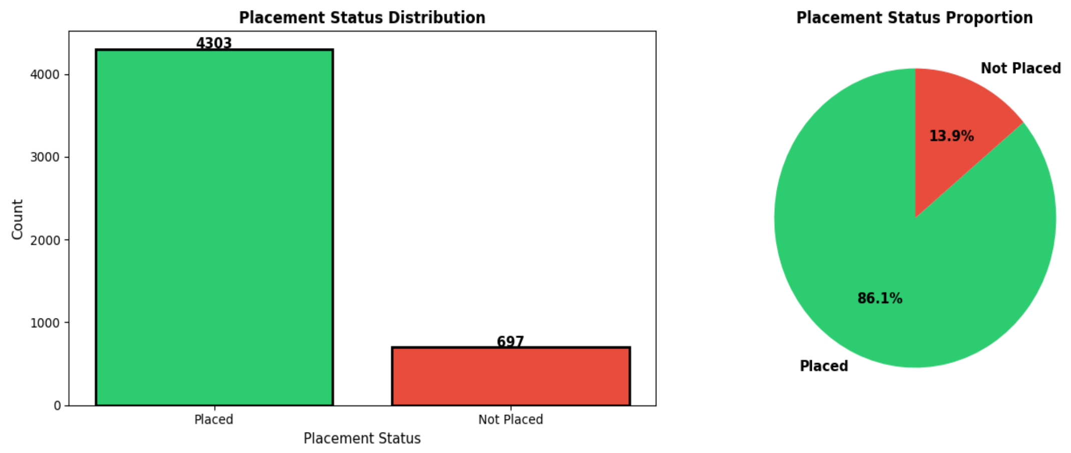
The *extracurricular_activity* column contains missing values because not every student necessarily participates in extracurricular activities. These missing entries do not represent data errors but rather valid real-world scenarios. Therefore, applying **dropna()** to remove such rows would incorrectly eliminate genuine student records and reduce dataset quality. For this reason, missing values in this column are **not removed**; instead, they are retained and handled appropriately during preprocessing.

Check whether the Dataset is imbalanced or not :



```
print("\n📊 CLASS COUNTS:")
class_counts = y.value_counts()
print(class_counts)

print("\n📈 CLASS PROPORTIONS:")
class_proportions = y.value_counts(normalize=True) * 100
for cls, prop in class_proportions.items():
    print(f"   {cls}: {prop:.2f}%")

print("\n⚠️  IMBALANCE RATIO:")
placed_count = class_counts.get('Placed', 0)
not_placed_count = class_counts.get('Not Placed', 0)
if placed_count > 0 and not_placed_count > 0:
    ratio = max(placed_count, not_placed_count) / min(placed_count, not_placed_count)
    print(f"   Ratio: 1:{ratio:.2f}")
    if ratio > 1.5:
        print(f"   ⚠️  IMBALANCED! Consider using SMOTE")
    else:
        print(f"   ✓ Relatively BALANCED")

# Visualization
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Bar plot
colors = ['#2ecc71', '#e74c3c']
axes[0].bar(class_counts.index, class_counts.values, color=colors, edgecolor='black', linewidth=2)
axes[0].set_title('Class Distribution (Counts)', fontsize=13, fontweight='bold')
axes[0].set_ylabel('Number of Students', fontsize=11)
axes[0].set_xlabel('Placement Status', fontsize=11)
for i, v in enumerate(class_counts.values):
    axes[0].text(i, v + 5, str(v), ha='center', fontweight='bold', fontsize=12)

# Pie chart
axes[1].pie(class_counts.values, labels=class_counts.index, autopct='%1.1f%%',
            colors=colors, startangle=90, textprops={'fontsize': 12, 'fontweight': 'bold'})
axes[1].set_title('Class Distribution (Proportions)', fontsize=13, fontweight='bold')

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

After checking the imbalance condition of the data set regarding the target value We found that the data is imbalance as the placed results are found out to be more ***Placed*** than the ***Not Placed*** Status so in future



after train test split the smote will be applied in order to make sure the data is perfect for prediction and the train test leakage will not happen.

EDA And Feature Selection Part :

Step 1: At first we have to differentiate the numerical and categorical features so that the EDA and the Feature Selection part works well.

```
numerical_features = [feat for feat in X.columns if df[feat].dtype != 'O']
categorical_features = [feat for feat in X.columns if df[feat].dtype == 'O']

print("Numerical Features:")
print(numerical_features)
print("\nCategorical Features:")
print(categorical_features)
```

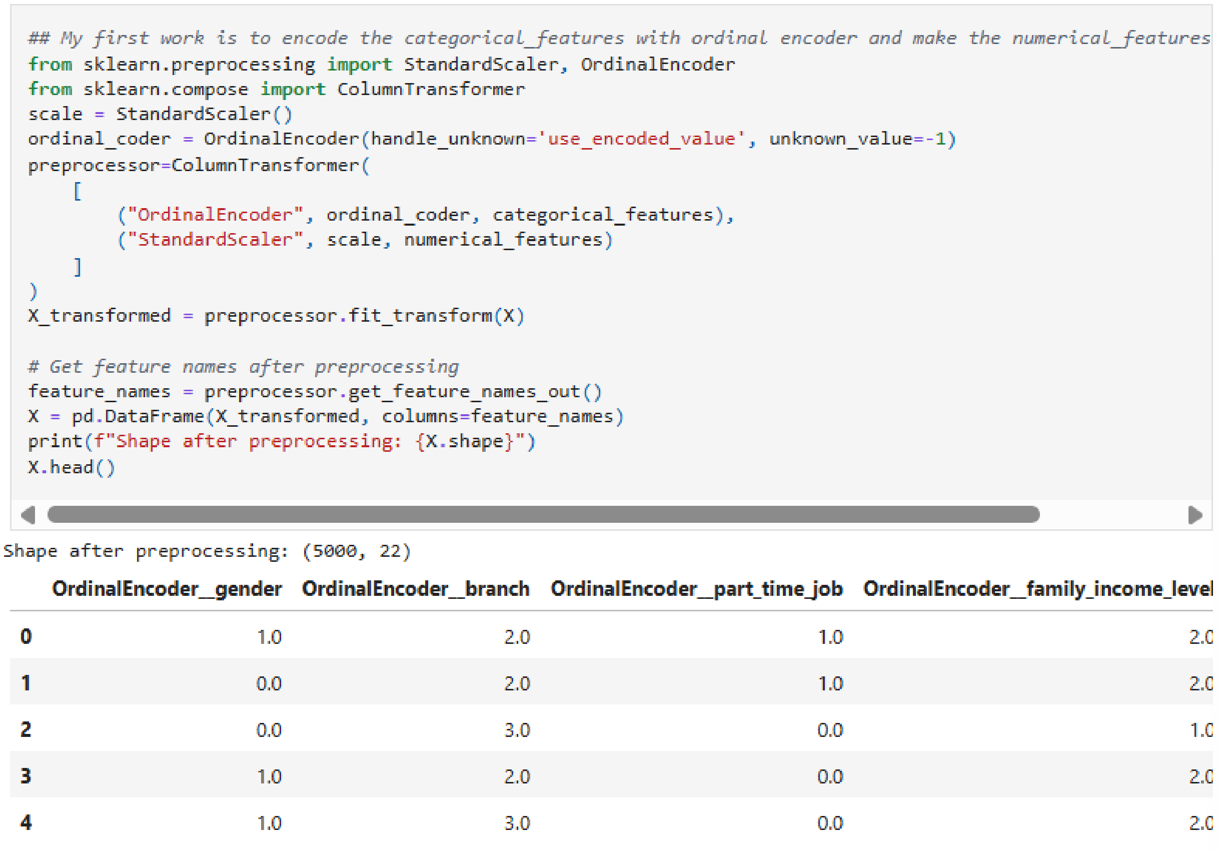
```
Numerical Features:
['cgpa', 'tenth_percentage', 'twelfth_percentage', 'backlogs', 'study_hours_per_day', 'attendance_percentag
e', 'projects_completed', 'internships_completed', 'coding_skill_rating', 'communication_skill_rating', 'apti
tude_skill_rating', 'hackathons_participated', 'certifications_count', 'sleep_hours', 'stress_level']

Categorical Features:
['gender', 'branch', 'part_time_job', 'family_income_level', 'city_tier', 'internet_access', 'extracurricular
_involvement']
```

Now in **Step 2:** checking how many unique values are there in the categorical feature to make it easy to select the encoding technique we are going to use

```
for feature in categorical_features:
    print(f"{feature} has uniques of : {df[feature].unique()}")
```

```
gender has uniques of : ['Male' 'Female']
branch has uniques of : ['ECE' 'IT' 'CSE' 'CE' 'ME']
part_time_job has uniques of : ['Yes' 'No']
family_income_level has uniques of : ['Medium' 'Low' 'High']
city_tier has uniques of : ['Tier 2' 'Tier 3' 'Tier 1']
internet_access has uniques of : ['Yes' 'No']
extracurricular_involvement has uniques of : ['Medium' 'Low' 'High' nan]
```

Step 3 : as most of the section has more than two unique and we have to further make it useful for visualization so we will use the ***OrdinalEncoding*** technique. Along with we will make the numerical feature to standardize for the better visualization .

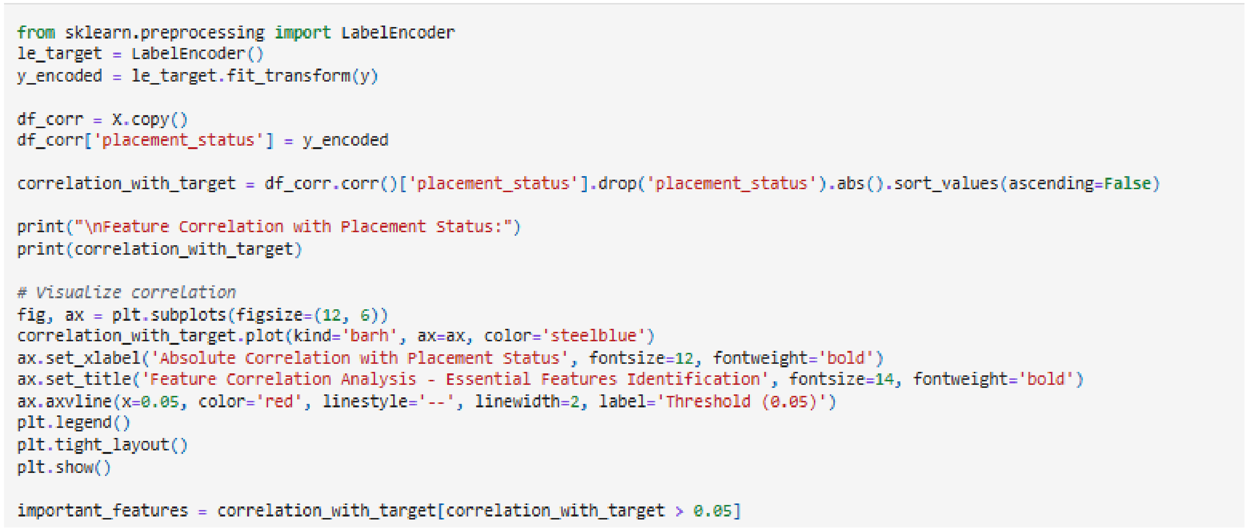


```
## My first work is to encode the categorical_features with ordinal encoder and make the numerical_features
from sklearn.preprocessing import StandardScaler, OrdinalEncoder
from sklearn.compose import ColumnTransformer
scale = StandardScaler()
ordinal_coder = OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1)
preprocessor=ColumnTransformer(
    [
        ("OrdinalEncoder", ordinal_coder, categorical_features),
        ("StandardScaler", scale, numerical_features)
    ]
)
X_transformed = preprocessor.fit_transform(X)

# Get feature names after preprocessing
feature_names = preprocessor.get_feature_names_out()
X = pd.DataFrame(X_transformed, columns=feature_names)
print(f"Shape after preprocessing: {X.shape}")
X.head()
```

```
Shape after preprocessing: (5000, 22)
```

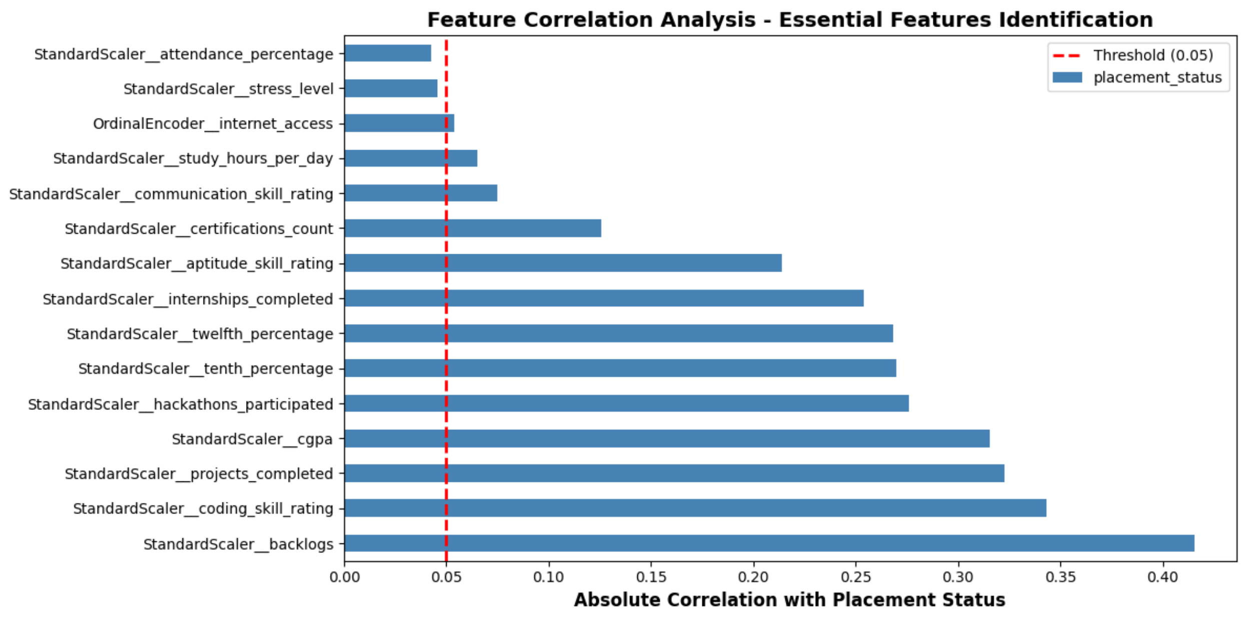
	OrdinalEncoder__gender	OrdinalEncoder__branch	OrdinalEncoder__part_time_job	OrdinalEncoder__family_income_level
0	1.0	2.0	1.0	2.0
1	0.0	2.0	1.0	2.0
2	0.0	3.0	0.0	1.0
3	1.0	2.0	0.0	2.0
4	1.0	3.0	0.0	2.0

step 4: now comes the feature selection part where the features with the best correlation with the target will be chosen

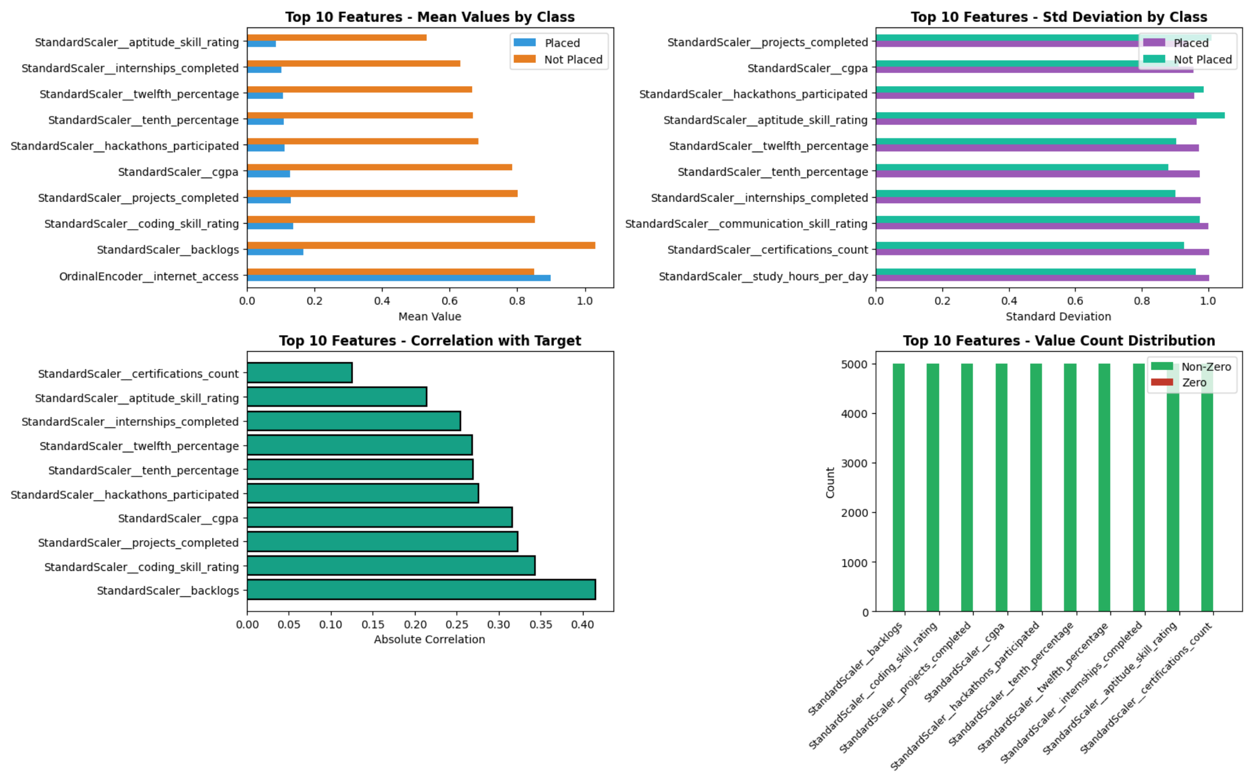


```
from sklearn.preprocessing import LabelEncoder
le_target = LabelEncoder()
y_encoded = le_target.fit_transform(y)

df_corr = X.copy()
df_corr['placement_status'] = y_encoded

correlation_with_target = df_corr.corr()['placement_status'].drop('placement_status').abs().sort_values(ascending=False)

print("\nFeature Correlation with Placement Status:")
print(correlation_with_target)

# Visualize correlation
fig, ax = plt.subplots(figsize=(12, 6))
correlation_with_target.plot(kind='barh', ax=ax, color='steelblue')
ax.set_xlabel('Absolute Correlation with Placement Status', fontsize=12, fontweight='bold')
ax.set_title('Feature Correlation Analysis - Essential Features Identification', fontsize=14, fontweight='bold')
ax.axvline(x=0.05, color='red', linestyle='--', linewidth=2, label='Threshold (0.05)')
plt.legend()
plt.tight_layout()
plt.show()

important_features = correlation_with_target[correlation_with_target > 0.05]
```

The bar chart represents the absolute correlation of each feature with the target variable *placement_status*, helping identify which attributes are most influential in predicting whether a student gets placed. A red dotted vertical line at 0.05 marks the minimum correlation threshold, meaning any feature with a correlation value below this line is considered weak or insignificant for the model. From the visualization, features such as backlogs, coding skill rating, projects completed, CGPA, hackathons participated, tenth and twelfth percentage, internships completed, and aptitude skill rating show strong correlations, making them highly relevant for placement prediction. In contrast, features like attendance percentage, stress level, and internet access fall below or very close to the threshold, indicating minimal impact on placement outcomes. Overall, this chart reveals the most essential features that should be prioritized during feature selection and highlights which variables can be ignored or assigned lower importance in the model.

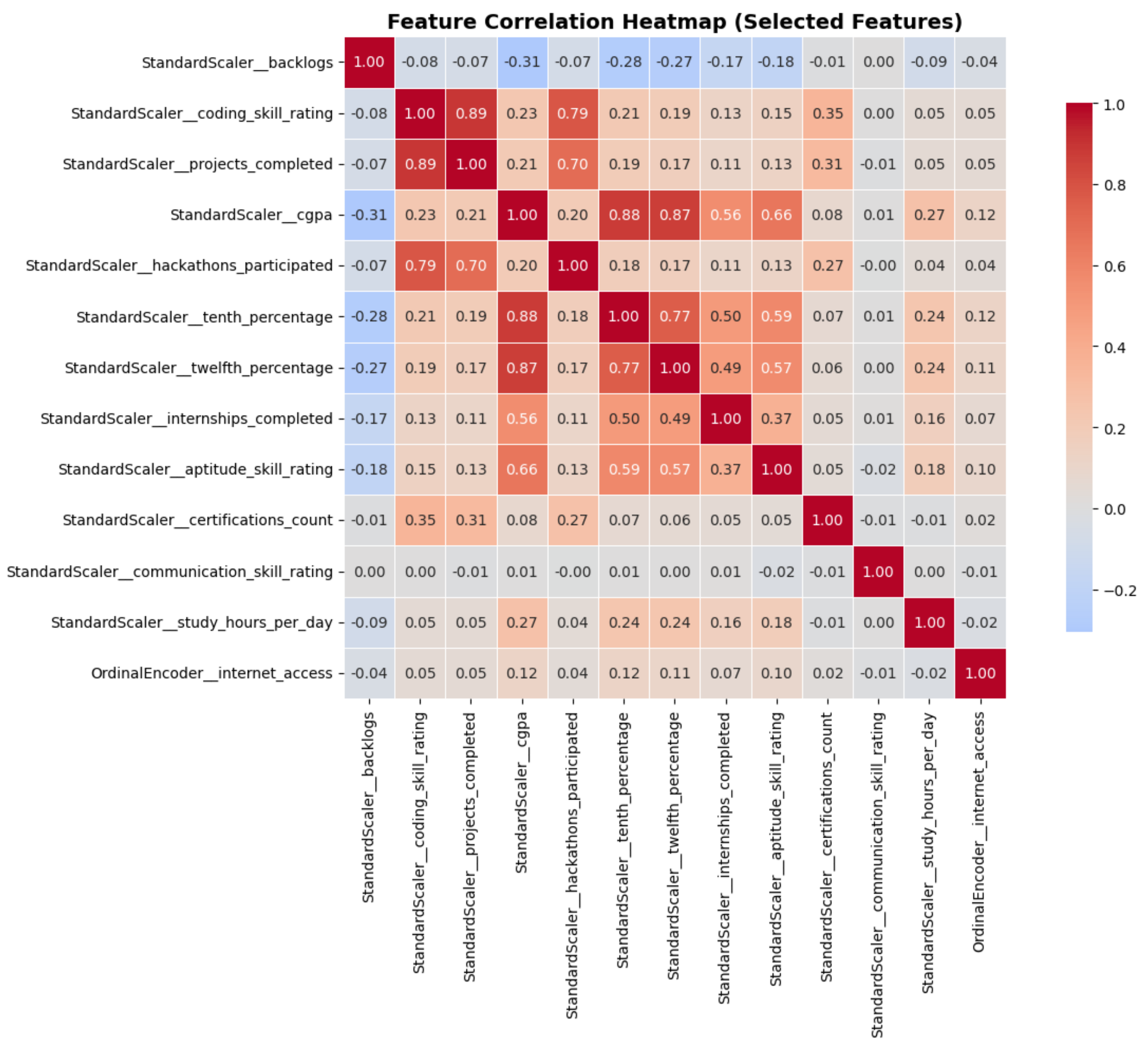


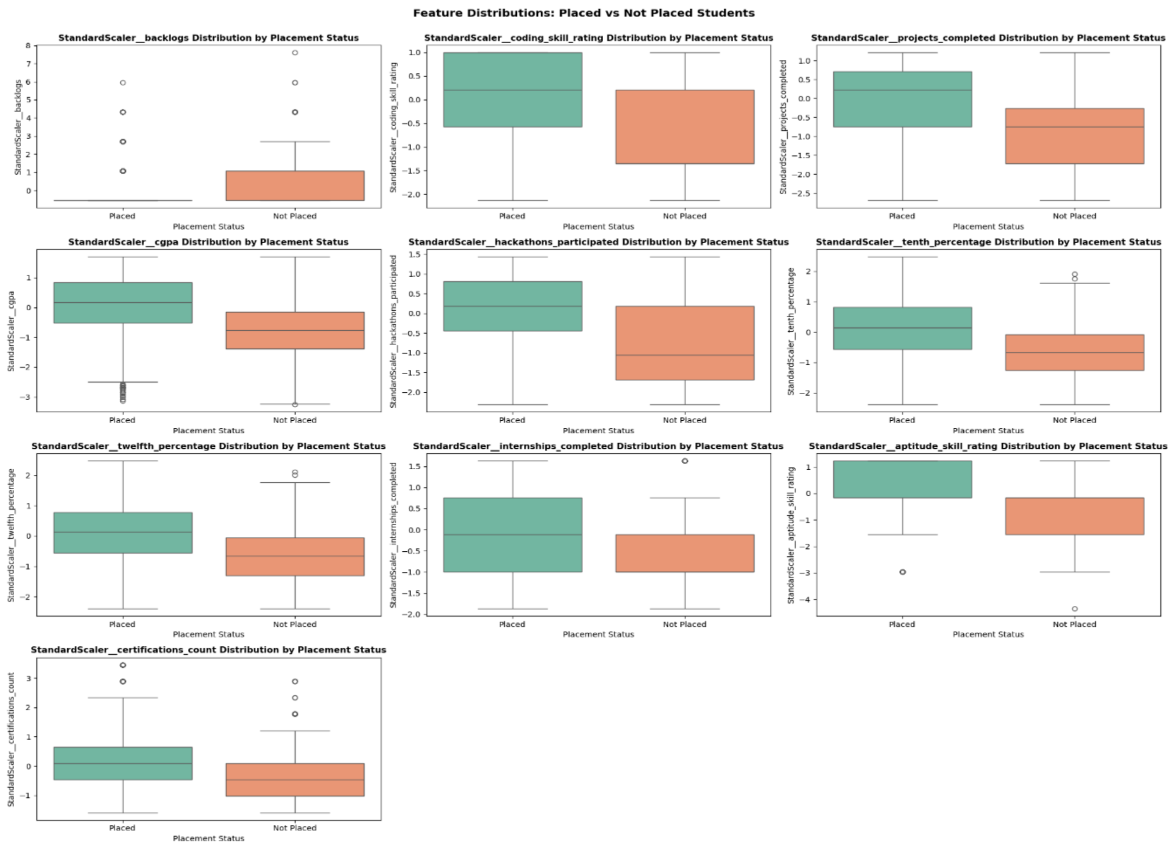
<u>Visualizations Analysis:</u>



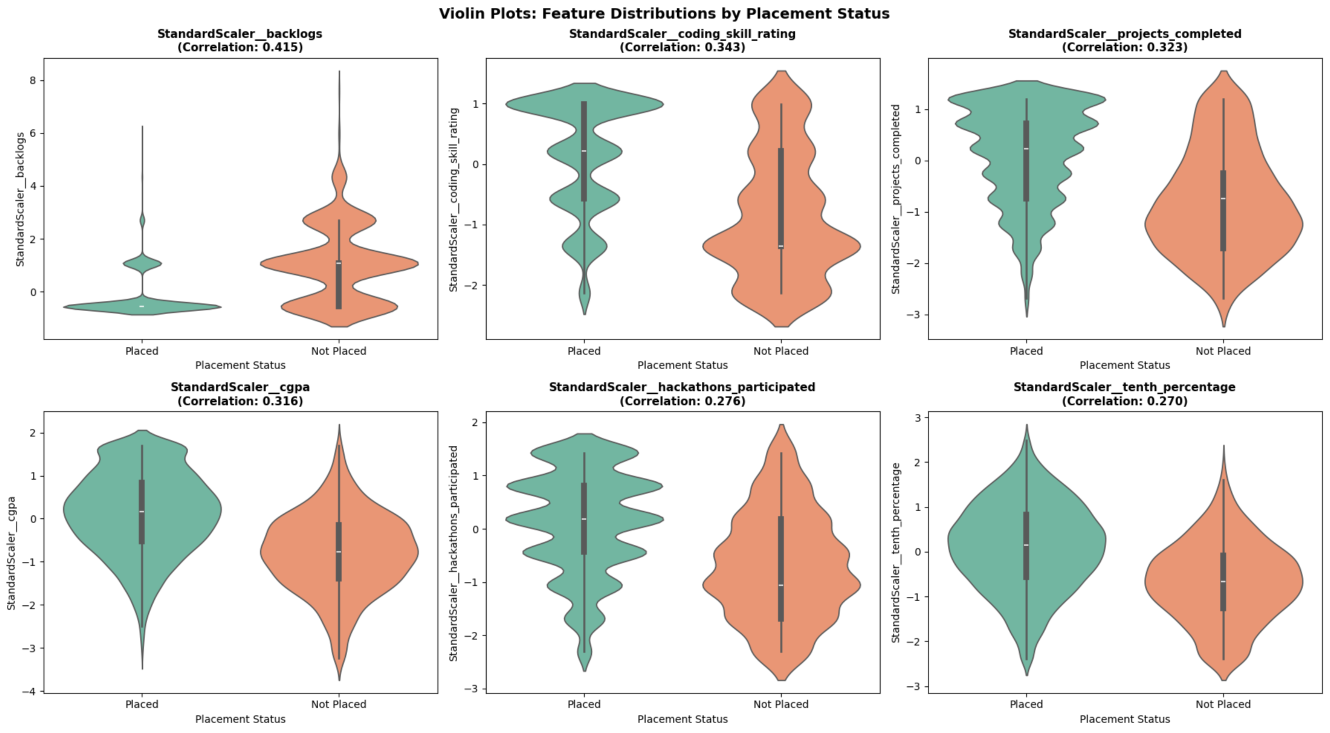
The visualizations collectively present a comprehensive feature analysis aimed at identifying factors that most strongly influence placement outcomes. The **Feature Correlation Analysis bar chart** ranks all features by their absolute correlation with placement status, with a red threshold line helping highlight truly impactful variables such as backlogs, coding skills, projects completed, CGPA, internships, and academic percentages. The second composite figure provides a deeper breakdown: the **mean value comparison plot** shows how placed students consistently score higher across key features like aptitude, internships, projects, and skill ratings; the **standard deviation plot** reveals how variability differs between placed and non-placed groups, indicating consistent performance among placed students; the **correlation bar chart** reiterates the top contributors; and the **value count plot** ensures these features are not sparse and hold meaningful variation across the dataset.

The **correlation heatmap** visualizes relationships among selected features, showing strong internal correlations between academic indicators (10th %, 12th %, CGPA) and moderate associations between skills and participation metrics, helping detect redundancy and multicollinearity.





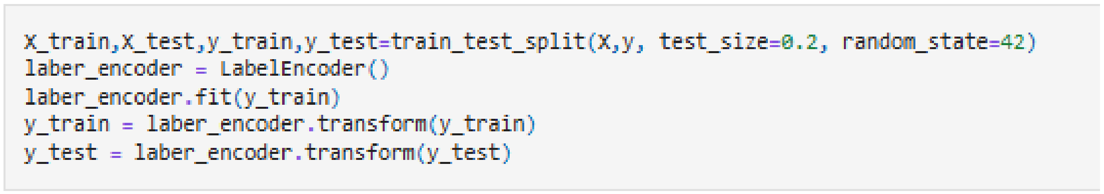
The **boxplots** contrast the distribution of top features across placed vs. not-placed groups, clearly illustrating higher medians and lower dispersion among placed students, especially in CGPA, projects, internships, and skill scores, while also exposing outliers such as high backlogs or extremely low skill values. Finally, the **violin plots** offer a smoother visualization of feature density, highlighting distinct



distribution shapes between the two classes and reinforcing that placed students consistently cluster around higher values for coding skill, CGPA, projects completed, hackathons, and academic percentages. Together, these visualizations provide strong evidence of which attributes meaningfully differentiate placed from non-placed students and justify their selection as key predictors in the machine-learning model.

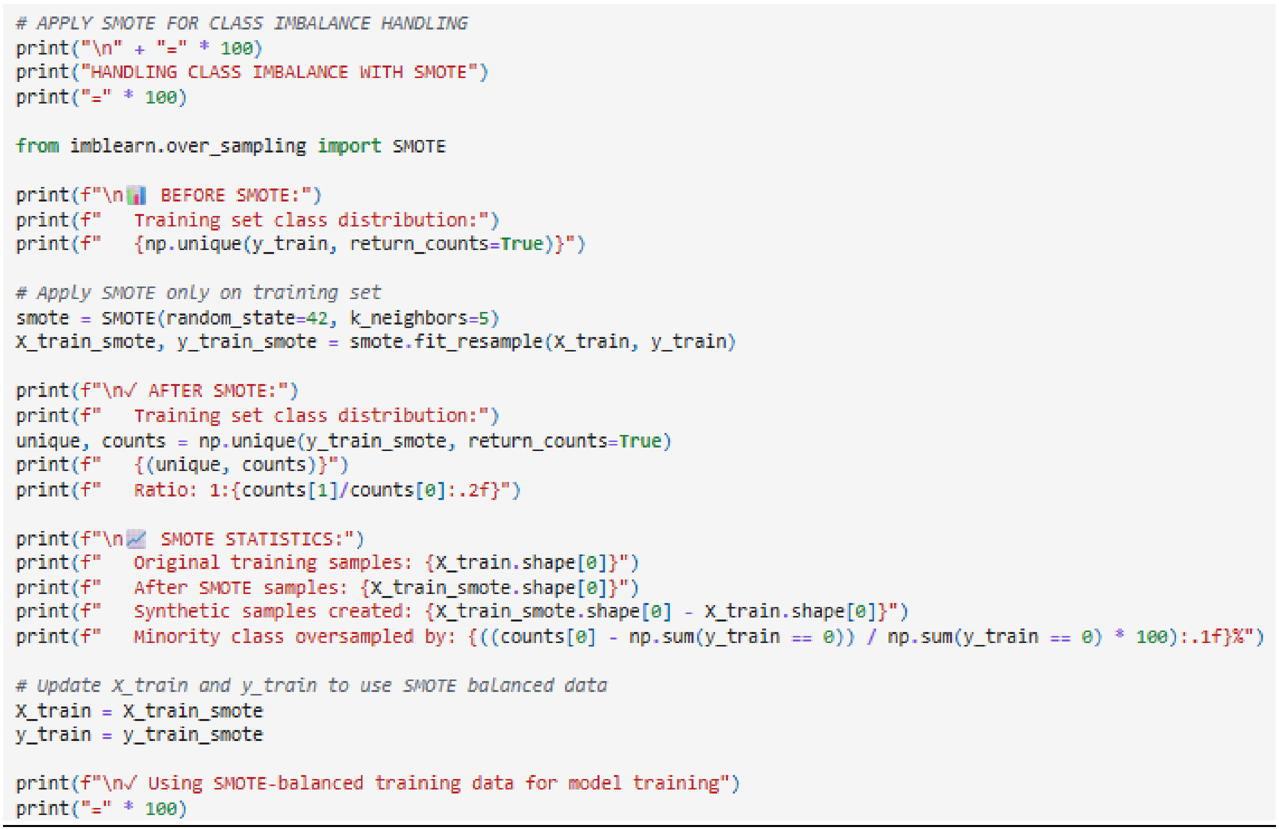
Model selection and Hyperparameter Tuning:

Test & test splitting :



```python
X_train,X_test,y_train,y_test=train_test_split(X,y, test_size=0.2, random_state=42)
laber_encoder = LabelEncoder()
laber_encoder.fit(y_train)
y_train = laber_encoder.transform(y_train)
y_test = laber_encoder.transform(y_test)
```

Appling Smote Technique:



```python
# APPLY SMOTE FOR CLASS IMBALANCE HANDLING
print("\n" + "=" * 100)
print("HANDLING CLASS IMBALANCE WITH SMOTE")
print("=" * 100)

from imblearn.over_sampling import SMOTE

print(f"\n📊 BEFORE SMOTE:")
print(f"   Training set class distribution:")
print(f"   {np.unique(y_train, return_counts=True)}")

# Apply SMOTE only on training set
smote = SMOTE(random_state=42, k_neighbors=5)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

print(f"\n✓ AFTER SMOTE:")
print(f"   Training set class distribution:")
unique, counts = np.unique(y_train_smote, return_counts=True)
print(f"   {(unique, counts)}")
print(f"   Ratio: 1:{counts[1]/counts[0]:.2f}")

print(f"\n📈 SMOTE STATISTICS:")
print(f"   Original training samples: {X_train.shape[0]}")
print(f"   After SMOTE samples: {X_train_smote.shape[0]}")
print(f"   Synthetic samples created: {X_train_smote.shape[0] - X_train.shape[0]}")
print(f"   Minority class oversampled by: {((counts[0] - np.sum(y_train == 0)) / np.sum(y_train == 0) * 100):.1f}%")

# Update X_train and y_train to use SMOTE balanced data
X_train = X_train_smote
y_train = y_train_smote

print(f"\n✓ Using SMOTE-balanced training data for model training")
print("=" * 100)
```

Evaluation metrics we are going are going to use :

1. **ROC-AUC(Receiver Operating Characteristic -Area Under the curve) :**
   The ROC curve is a graphical evaluation metric that illustrates the performance of a classification model across all possible probability thresholds. It is created by plotting:
   - True Positive Rate (TPR) on the Y-axis

- False Positive Rate (FPR) on the X-axis

The curve shows how well the model distinguishes between the two classes (*Placed* vs *Not Placed*) as the decision threshold changes.

Mathematical Expression of ROC-AUC:

$$AUC = \int_{0}^{1} TPR(FPR)d(FPR)$$

Where :

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{\text{FP}}{FP + TN}$$

2. Accuracy

You used accuracy to measure the overall correctness of the model.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

3. Precision

You used precision to measure how many students predicted as *Placed* were truly placed.

$$\text{Precision} = \frac{TP}{TP + FP}$$

4. Recall

Your notebook uses recall to determine how many truly placed students were correctly identified.

$$\text{Recall} = \frac{TP}{TP + FN}$$

5. F1-Score

You also used the F1-score, which balances precision and recall and is especially helpful for classification tasks.

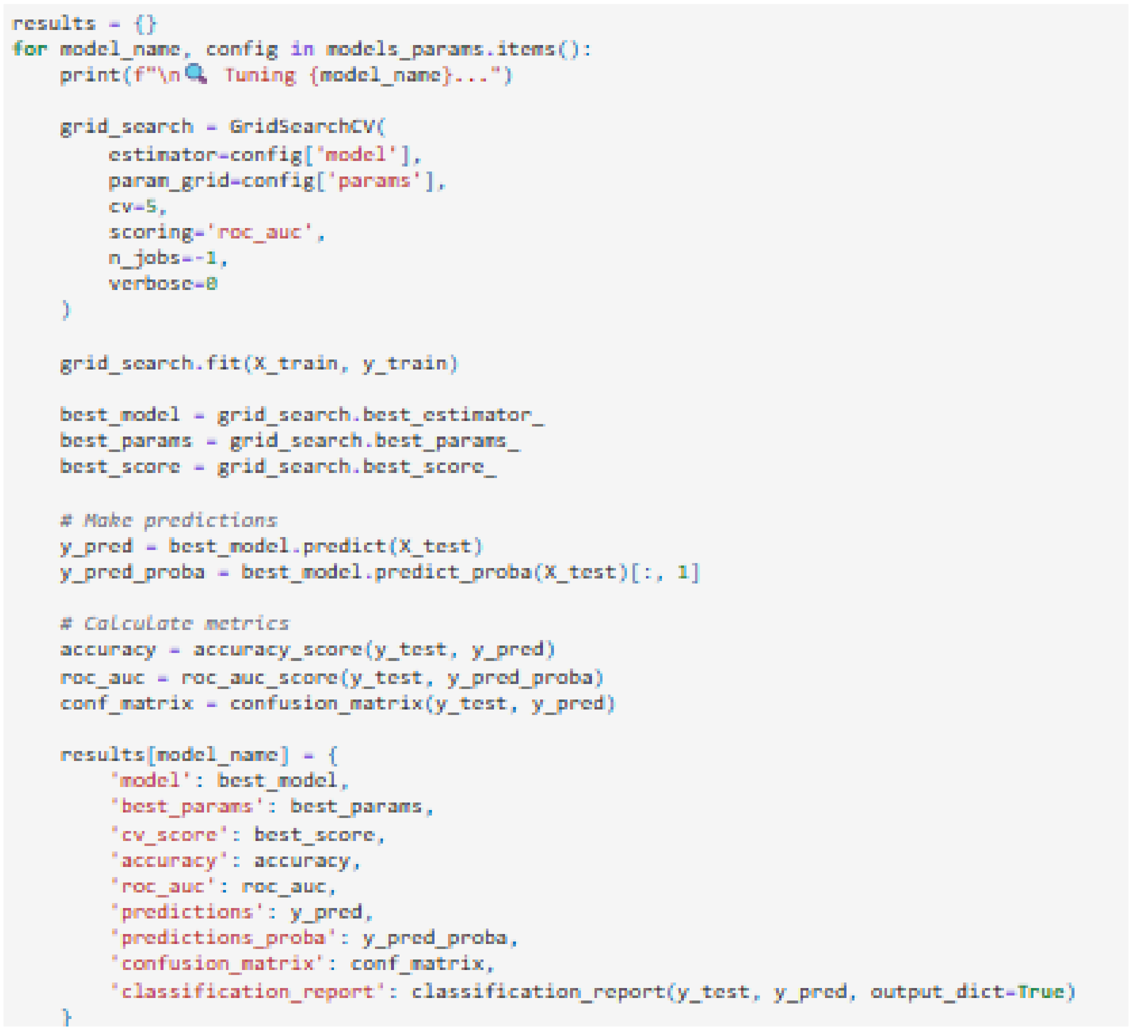
$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Which Evaluation matric is useful for this problem?

- In this placement classification project, evaluation metrics such as accuracy, precision, recall, F1-score, and the confusion matrix are used to understand different aspects of the model's performance, including how well it identifies truly placed students, how reliably it predicts placement, and where it makes errors. However, ROC–AUC stands out as the most meaningful metric for this problem because it measures the model's ability to distinguish between *Placed* and *Not Placed* students across all probability thresholds, rather than relying on a single cutoff value. Since placement prediction involves ranking students by their employability potential and ensuring that genuinely placeable students are not overlooked, ROC-AUC provides a more complete and unbiased view of the model's discrimination power. It remains reliable even when the dataset is imbalanced and reflects how effectively the model separates high-potential students from low-potential ones, making it the most suitable metric for evaluating placement prediction performance.

Model Selection With GridSearchCV():

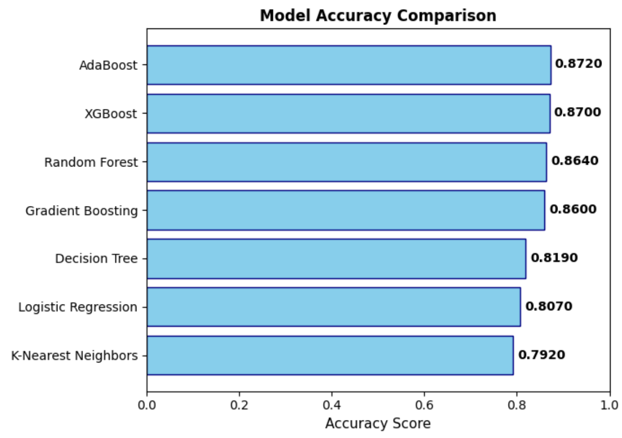
Model selection in this project was carried out by training and evaluating multiple classification algorithms to determine which model best predicts student placement outcomes. The selection process followed a systematic approach: after preprocessing the data (scaling, encoding, and preparing balanced training samples), four machine-learning models—**Logistic Regression, Decision Tree, Random Forest, and XGBoost**, **AdaBoost**, **Gradient Boosting**, **K-Nearest Neighbors** were trained on the transformed dataset. Each model was evaluated using key performance metrics such as **Accuracy, Precision, Recall, F1-Score, Confusion Matrix**, and **ROC-AUC**, ensuring a fair and comprehensive comparison.

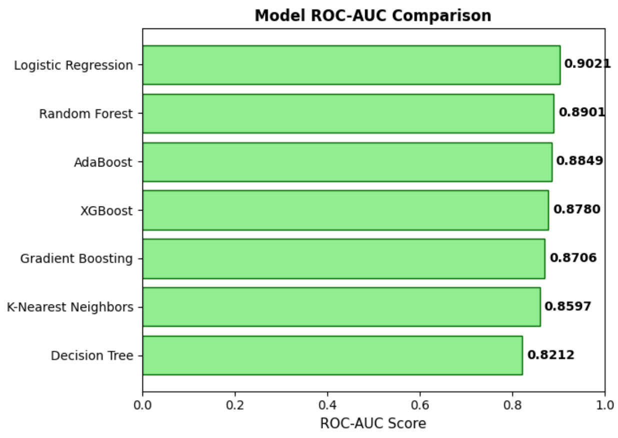


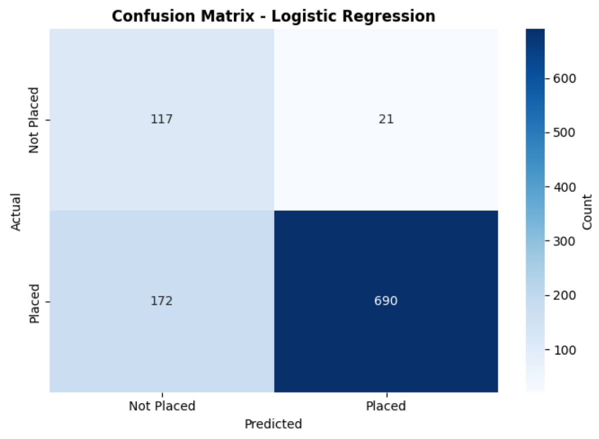
```
results = {}
for model_name, config in models_params.items():
    print(f"\n🔍 Tuning {model_name}...")

    grid_search = GridSearchCV(
        estimator=config['model'],
        param_grid=config['params'],
        cv=5,
        scoring='roc_auc',
        n_jobs=-1,
        verbose=0
    )

    grid_search.fit(X_train, y_train)

    best_model = grid_search.best_estimator_
    best_params = grid_search.best_params_
    best_score = grid_search.best_score_

    # Make predictions
    y_pred = best_model.predict(X_test)
    y_pred_proba = best_model.predict_proba(X_test)[:, 1]

    # Calculate metrics
    accuracy = accuracy_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred_proba)
    conf_matrix = confusion_matrix(y_test, y_pred)

    results[model_name] = {
        'model': best_model,
        'best_params': best_params,
        'cv_score': best_score,
        'accuracy': accuracy,
        'roc_auc': roc_auc,
        'predictions': y_pred,
        'predictions_proba': y_pred_proba,
        'confusion_matrix': conf_matrix,
        'classification_report': classification_report(y_test, y_pred, output_dict=True)
    }
```

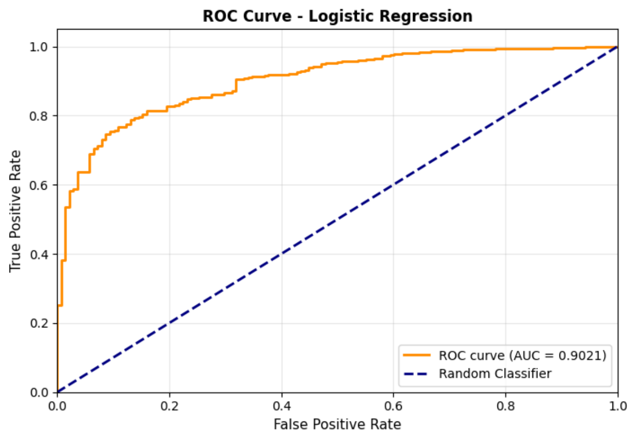
Results :

The hyperparameter tuning results show clear statistical differences in the performance of the tested models across both cross-validation and test evaluation. Logistic Regression achieved a respectable cross-validation ROC-AUC of 0.9104 and a test ROC-AUC of 0.9021, indicating stable but moderate predictive strength. The Decision Tree model showed a high cross-validation ROC-AUC of 0.9376, but its test ROC-AUC dropped to 0.8212, suggesting overfitting. Random Forest delivered one of the strongest performances, with an excellent cross-validation ROC-AUC of 0.9886 and a high test accuracy of 0.8640, supported by a robust test ROC-AUC of 0.8901. Gradient Boosting and AdaBoost also performed competitively, achieving cross-validation ROC-AUC scores of 0.9872 and 0.9768, and test ROC-AUC values of 0.8706 and 0.8849, respectively, demonstrating consistent generalization. K-Nearest Neighbors, despite a strong cross-validation ROC-AUC of 0.9806, lagged on the test set with an accuracy of 0.7920 and ROC-AUC of 0.8597, showing sensitivity to unseen data. XGBoost maintained stable performance with a cross-validation ROC-AUC of 0.9880 and test ROC-AUC of 0.8780, placing it among the top performers. Overall, the statistical results indicate that Random Forest, AdaBoost, and XGBoost offer the most balanced and reliable predictive power for the placement classification task, while Decision Tree and KNN show higher variance between training and testing results.









Conclusion:

The tuning results show that all models exhibited strong performance, but with notable differences between training behavior and generalization to unseen data. Logistic Regression delivered one of the most balanced outcomes, achieving a cross-validation ROC-AUC of 0.9104 and a test ROC-AUC of 0.9021, indicating highly stable and consistent performance across both training splits and test data.

Model	CV ROC-AUC	Test Accuracy	Test ROC-AUC
Logistic Regression	0.9104	0.8070	0.9021
Decision Tree	0.9376	0.8190	0.8212
Random Forest	0.9886	0.8640	0.8901
Gradient Boosting	0.9872	0.8600	0.8706
AdaBoost	0.9768	0.8720	0.8849
K-Nearest Neighbors	0.9806	0.7920	0.8597
XGBoost	0.9880	0.8700	0.8780

The Decision Tree model, despite a strong CV ROC-AUC of 0.9376, dropped significantly to 0.8212 on the test set, revealing overfitting. Ensemble methods such as Random Forest, Gradient Boosting, AdaBoost, and XGBoost achieved excellent cross-validation ROC-AUC scores above 0.97, but their test ROC-AUC values ranged between 0.87–0.89, showing strong but slightly less stable generalization than Logistic Regression. Among them, Random Forest and AdaBoost performed the best on test accuracy (0.8640 and 0.8720, respectively), yet neither surpassed Logistic Regression in test ROC-AUC performance. K-Nearest Neighbors, although achieving a high CV score (0.9806), showed the weakest test performance (0.792 accuracy, 0.8597 ROC-AUC), confirming sensitivity to unseen data. Overall, the results indicate that while complex models may appear powerful during cross-validation, Logistic Regression offers the most reliable, consistent, and generalizable performance, making it the strongest choice for the placement classification task.