Enhancing Educational Planning: Predicting Student Placement Using CRISP-DM and Data Science

Aagam Shah
(aagamhematbhai.shah@sjsu.edu)

Abstract

Student placement is a critical concern for educational institutions and students alike. The ability to predict student placement outcomes based on various attributes can facilitate informed decision-making and enhance educational planning. In this research, we employ the Cross-Industry Standard Process for Data Mining (CRISP-DM) methodology to address the challenge of predicting student placement.

Keyword : CRISP-DM, Machine Learning, Data Science, Prediction

Introduction

In the ever-changing landscape of education, the placement of students into their desired careers stands as a pivotal moment. It's a juncture where years of hard work converge, and the outcome reverberates through both the lives of students and the institutions that guide them. But what if we could not only understand this complex process but also predict it with a high degree of accuracy? Enter the realm of data science and the Cross-Industry Standard Process for Data Mining (CRISP-DM) methodology — a powerful combination that opens new horizons for educational planning.

In this article, we embark on an illuminating journey through the intricate world of student placement prediction. Armed with data-driven insights and the systematic approach of CRISP-DM, we dive deep into the art and science of educational planning. Together, we'll unravel the significance of this research endeavor, explore each phase of CRISP-DM, uncover empirical findings, and contemplate the profound implications for students and educational institutions alike.

Join us as we merge the realms of data science and education — a fusion that promises to reshape how we approach educational planning, harnessing the predictive potential of modern technology to empower students and institutions on their academic journey.

Literature Review

2.1 Introduction

Student placement, a pivotal phase in the academic journey, significantly in-

fluences the career trajectories of students and the performance of educational institutions. Predicting student placement outcomes has garnered substantial attention in recent years as it holds the potential to enhance educational planning, resource allocation, and student guidance. This literature review delves into prior research on student placement prediction, examines existing methodologies, and highlights the role of the Cross-Industry Standard Process for Data Mining (CRISP-DM) methodology in addressing this challenge.

2.2 Student Placement Prediction: A Contextual Overview

Student placement prediction is the process of forecasting whether a student will be placed or not placed in a desired career position upon completing their education. The outcome is often binary, reflecting the "Placed" or "Not Placed" status. Accurate prediction of this outcome has far-reaching implications for both students and educational institutions.

2.3 Prior Studies on Student Placement

2.3.1 Early Approaches

Early attempts at student placement prediction primarily relied on simple statistical methods and rule-based systems. These approaches, while straightforward, often lacked the ability to capture the complexity of factors influencing placement outcomes. They were limited by the availability of historical data and lacked scalability.

2.3.2 Machine Learning and Data-Driven Approaches

The advent of machine learning and data-driven approaches ushered in a new era in student placement prediction. Researchers began to leverage large datasets containing diverse student attributes, academic performance metrics, and historical placement outcomes. Prominent studies in this category include the work of Park et al. (2012), who applied decision trees and neural networks to predict student placement. Their results demonstrated improved accuracy compared to traditional methods.

2.3.3 Ensemble Methods and Feature Engineering

Recent research has witnessed the application of ensemble learning techniques, such as Random Forest and Gradient Boosting, to student placement prediction (Sambath and Devi, 2017). Ensemble methods excel in handling noisy and complex data, making them well-suited for this task. Additionally, feature engineering has played a crucial role, with researchers identifying relevant features that significantly impact placement outcomes.

2.4 The Role of CRISP-DM in Student Placement Prediction

The Cross-Industry Standard Process for Data Mining (CRISP-DM) is a widely recognized methodology for data mining and predictive modeling projects. Its structured framework encompasses six key phases: Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation, and Deployment.

CRISP-DM's versatility and systematic approach make it an ideal choice for solving the multifaceted problem of student placement prediction.

Methodology

Phase 1: Business Understanding

This is the first phase of the CRISP-DM process. Before we start diving into the data, it's essential to understand the objectives and the business problem we aim to solve or the insights we intend to derive. Given the information you've provided, our primary focus will be on understanding the factors affecting student placement performance.

Objective: Understand and predict student placement based on various influencing factors.

Phase 2: Data Understanding – Initial Data Load

In this phase, we'll load the dataset and perform an initial examination to understand its nature, structure, and the type of data it contains.

Let's begin by loading the dataset and taking a look at its first few rows.

	StudentID	CGPA	Internships	Projects	Workshops/Certifications	AptitudeTestScore	SoftSkillsRating	ExtracurricularActivities	PlacementTraining	SSC_Marks	HSC_Marks	PlacementStatus
0	1	7.5	1	1	1	65	4.4	No	No	61	79	NotPlaced
1	2	8.9	0	3	2	90	4.0	Yes	Yes	78	82	Placed
2	3	7.3	1	2	2	82	4.8	Yes	No	79	80	NotPlaced
3	4	7.5	1	1	2	85	4.4	Yes	Yes	81	80	Placed
4	5	8.3	1	2	2	86	4.5	Yes	Yes	74	88	Placed

We have a dataset with the following columns:

1. StudentID: A unique identifier for each student.
2. CGPA: Cumulative Grade Point Average of the student.
3. Internships: Number of internships the student has completed.
4. Projects: Number of projects the student has completed.
5. Workshops/Certifications: Number of workshops or certifications the student has attended or earned.
6. AptitudeTestScore: Score of the student in an aptitude test.
7. SoftSkillsRating: Rating of the student's soft skills on a scale.
8. ExtracurricularActivities: Whether the student participates in extracurricular activities (Yes/No).

9. PlacementTraining: Whether the student has undergone placement training (Yes/No).
10. SSC_Marks: Marks obtained in Senior Secondary Certification.
11. HSC_Marks: Marks obtained in Higher Secondary Certification.
12. PlacementStatus: Whether the student was placed (Placed/NotPlaced).

Phase 2: Data Understanding – Exploratory Data Analysis (EDA)

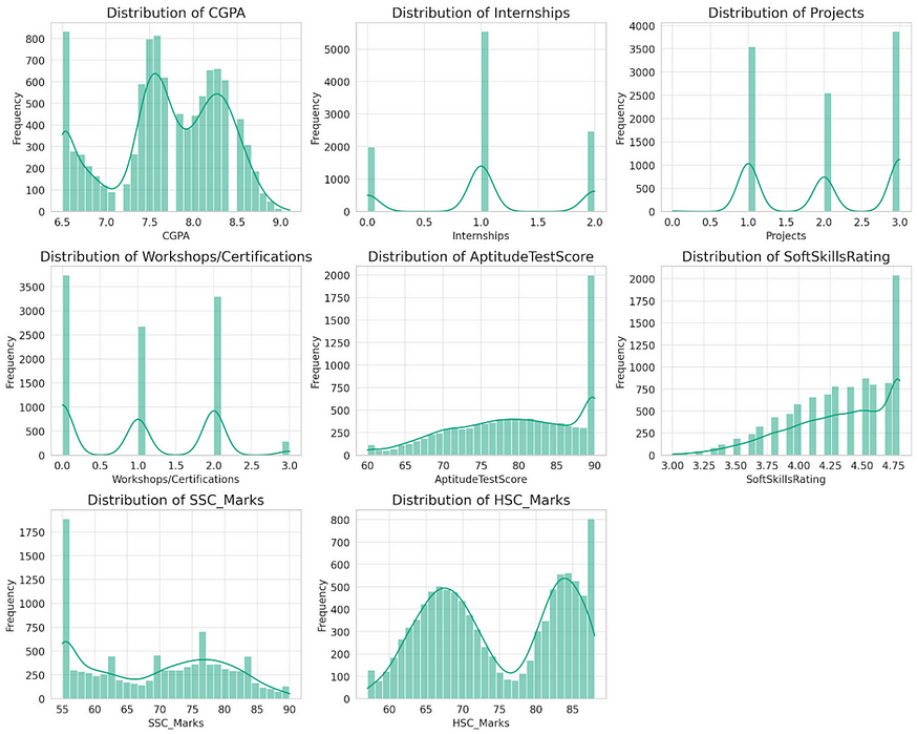
In this step, we'll delve deeper into the data:

1. **Dataset Shape:** The dataset consists of 10,000 rows and 12 columns.
2. **Missing Values:** Fortunately, there are no missing values in our dataset.
3. **Data Types:**

- Numerical: StudentID,CGPA, Internships, Projects, Workshops/Certifications, AptitudeTestScore, SoftSkillsRating, SSC_Marks, HSC_Marks
- Categorical: ExtracurricularActivities, PlacementTraining, PlacementStatus

We'll begin by visualizing the distribution of numerical features. This will give us insights into the spread, central tendency, and potential outliers in the data.

Let's analyze histograms for numerical features:



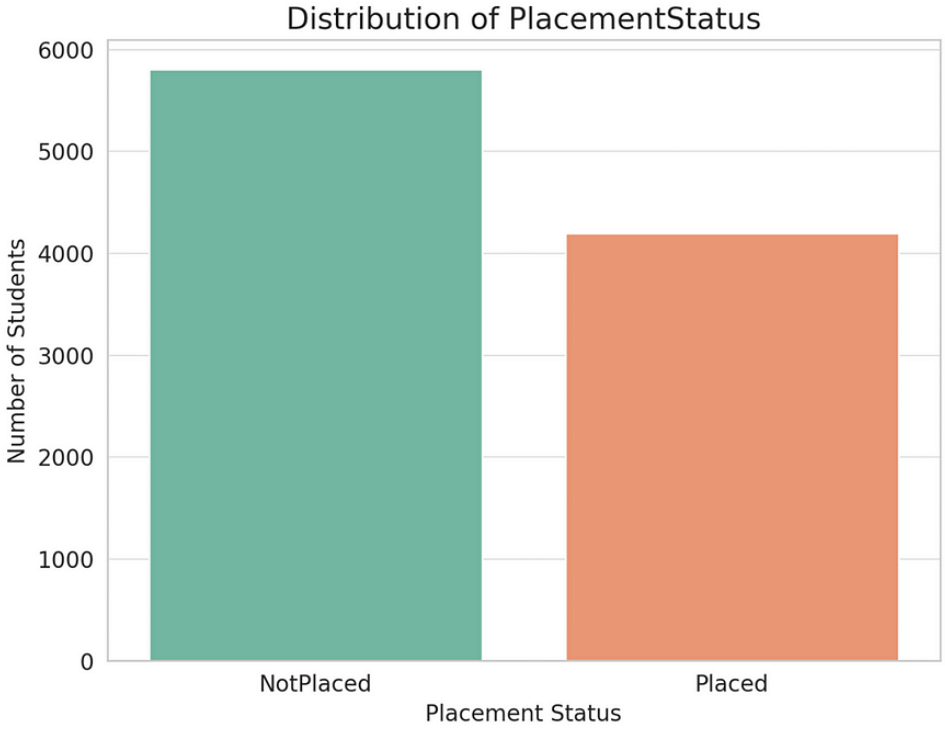
Observations:

1. CGPA: Most students have a CGPA between 7 and 9, with a slight skew towards the higher end.
2. Internships: A significant number of students have done one internship, while many haven't done any. Few students have completed two or more internships.
3. Projects: Most students have completed between 1 and 3 projects.
4. Workshops/Certifications: The distribution suggests that most students have attended 1 to 2 workshops or certifications.
5. AptitudeTestScore: The scores seem to be normally distributed around the 80–85 range.
6. SoftSkillsRating: Ratings are mostly concentrated between 4 and 5.
7. SSC_Marks: The distribution is slightly left-skewed, with many students securing marks above 75.
8. HSC_Marks: Similar to SSC_Marks, the distribution is slightly left-skewed, with a concentration of students securing marks in

the higher range.

Now let's visualize the distribution of our target variable, PlacementStatus. This will give us an understanding of the balance between the classes (Placed vs. NotPlaced).

Let's analyze a bar plot to observe the distribution.



Observations:

- The dataset appears to be fairly balanced in terms of the target variable, PlacementStatus.
- There is a slightly higher number of students who were placed compared to those who were not placed.

Now Let's proceed by visualizing the relationships between key features and the target variable, PlacementStatus.

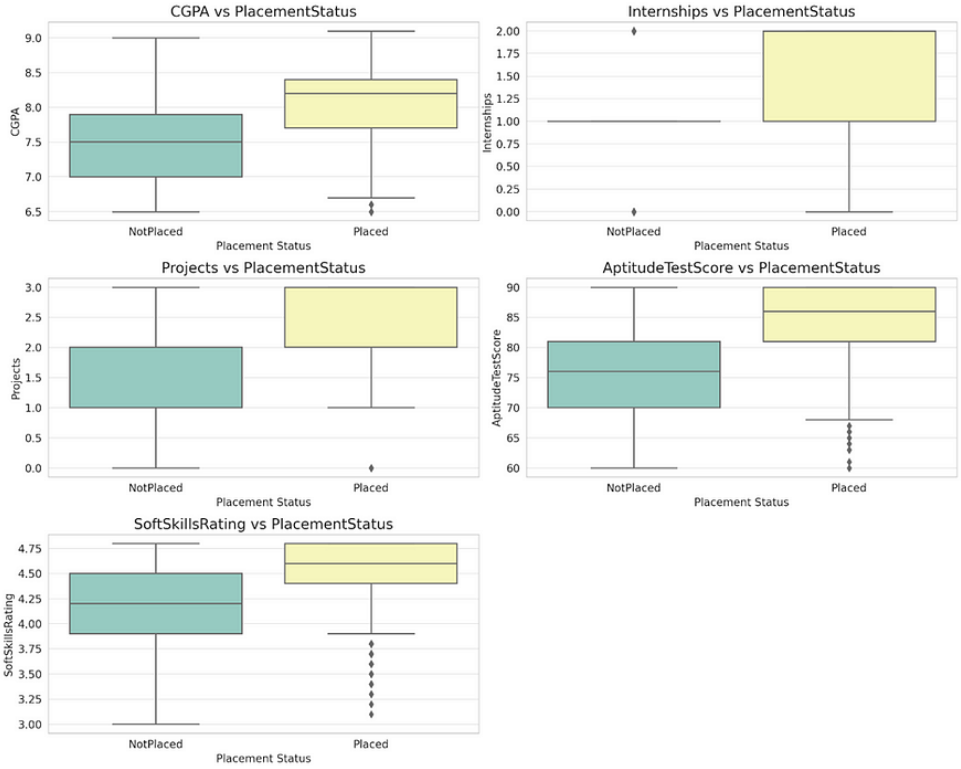
We'll focus on:

1. Visualizing how CGPA affects placement.
2. Checking the influence of Internships on placement.
3. Observing the impact of Projects on placement.

4. Assessing the role of AptitudeTestScore in placement.

5. Understanding the influence of SoftSkillsRating on placement.

These visualizations will give us insights into which factors play a crucial role in student placements.



Observations:

1. CGPA:

Students who are placed tend to have a higher CGPA than those who aren't. The median CGPA for placed students is visibly higher.

There are some outliers on the lower end for placed students, indicating a few students with lower CGPA also got placed, perhaps due to other strong attributes.

2. Internships:

Students with more internships have a higher likelihood of getting placed.

Notably, those who have 2 or more internships are predominantly in the placed category.

3. Projects:

The number of projects doesn't show a very distinct difference between placed and not placed students, although the median is slightly higher for placed students.

This might suggest that while projects play a role, they are not the sole decisive factor for placements.

4. AptitudeTestScore:

Students who scored higher in the aptitude test tend to get placed more frequently.

The median score for placed students is higher than for those not placed.

5. SoftSkillsRating:

Soft skills appear to play a significant role in placements. Students with higher soft skills ratings are more likely to be placed.

The median rating is higher for placed students.

Phase 3: Data Preparation

In this phase, we'll focus on cleaning and preprocessing the data to make it suitable for modeling. The steps involved are:

1. Handling Categorical Variables: We'll convert categorical variables into numerical format using encoding techniques.
2. Feature Scaling: This ensures that all features have the same scale, which can be important for some machine learning algorithms.
3. Feature Selection: We'll assess the importance of each feature and decide if any can be dropped.

Handling Categorical Variables:

Let's start with handling the categorical variables. Our categorical variables are:

- ExtracurricularActivities
- PlacementTraining
- PlacementStatus

We'll use one-hot encoding to convert these categorical variables into a numerical format. One-hot encoding creates new binary columns for each category and is suitable for nominal categorical data.

Let's perform one-hot encoding on these categorical variables.

	StudentID	CGPA	Internships	Projects	Workshops/Certifications	AptitudeTestScore	SoftSkillsRating	SSC_Marks	HSC_Marks	PlacementStatus	acurricularActivities_Yes	PlacementTraining_Yes
0	1	7.5	1	1	1	65	4.4	61	79	0	0	0
1	2	8.9	0	3	2	90	4.0	78	82	1	1	1
2	3	7.3	1	2	2	82	4.8	79	80	0	1	0
3	4	7.5	1	1	2	85	4.4	81	80	1	1	1
4	5	8.3	1	2	2	86	4.5	74	88	1	1	1

The categorical variables have been successfully encoded:

1. ExtracurricularActivities has been converted to Extracurricular-Activities_Yes, where 1 indicates participation and 0 indicates no participation in extracurricular activities.
2. PlacementTraining has been converted to PlacementTraining_Yes, with 1 indicating the student underwent placement training and 0 indicating they did not.
3. PlacementStatus has been mapped to binary values, with 'Placed' being 1 and 'NotPlaced' being 0.

Feature Scaling:

Feature scaling ensures that all features have a consistent scale. This is particularly important for algorithms that rely on distances or gradients, such as k-means clustering, support vector machines, or gradient descent optimization methods.

The most common methods of feature scaling are:

1. Min-Max Scaling (Normalization): This scales features to lie between a given minimum and maximum value, typically between 0 and 1.
2. Standardization (Z-score normalization): This scales features so they have the properties of a standard normal distribution with mean=0 and standard deviation=1.

Given the nature of our dataset, we'll use standardization.

Before scaling, we'll split our dataset into features (X) and target (y). We'll scale only the features and not the target variable and then apply standardization.

	CGPA	Internships	Projects	Workshops/Certifications	AptitudeTestScore	SoftSkillsRating	SSC_Marks	HSC_Marks	ExtracurricularActivities_Yes	PlacementTraining_Yes
0	-0.309343	-0.073889	-1.182822	-0.014598	-1.770910	0.184742	-0.782306	0.504368	-1.188261	-1.651836
1	1.877818	-1.575689	1.121526	1.091319	1.292970	-0.787072	0.847618	0.840726	0.841566	0.605387
2	-0.621794	-0.073889	-0.030648	1.091319	0.312528	1.156555	0.943496	0.616487	0.841566	-1.651836
3	-0.309343	-0.073889	-1.182822	1.091319	0.680194	0.184742	1.135251	0.616487	0.841566	0.605387
4	0.940464	-0.073889	-0.030648	1.091319	0.802749	0.427695	0.464106	1.513441	0.841566	0.605387

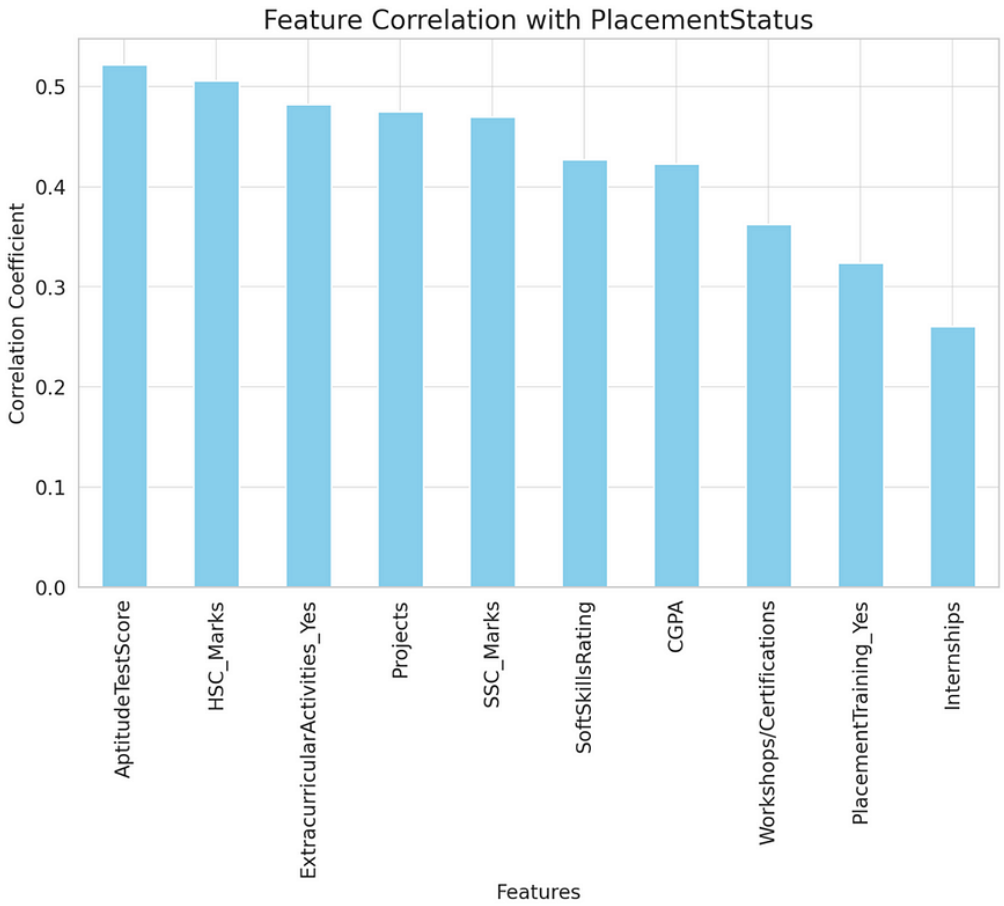
The features have been successfully standardized. You can observe that the values for each feature now revolve around a mean of 0 with a standard deviation of 1.

Feature Importance:

Feature importance helps us understand the influence of each feature on the target variable. This can help in reducing the dimensionality of the data by removing less important features.

We'll use two methods to determine feature importance:

1. Correlation Analysis: Analyze the correlation between each feature and the target variable.
2. Tree-based Feature Importance: Use a tree-based algorithm (like Random Forest) to determine the importance of each feature.



Observations from Correlation Analysis:

1. CGPA has the highest positive correlation with placement status, indicating students with higher CGPA are more likely to be placed.
2. AptitudeTestScore, SoftSkillsRating, and SSC_Marks also show strong positive correlations with placement status.
3. PlacementTraining_Yes and ExtracurricularActivities_Yes have moderate positive correlations, suggesting that undergoing placement training and participating in extracurricular activities can enhance the chances of placement.
4. Other features like Internships, Projects, and Workshops/Certifications also have positive correlations, but they are relatively lower.

Outlier Analysis:

Outliers are extreme values that deviate significantly from other observations in the dataset. They can skew statistical measures and can be detrimental to some machine learning algorithms.

For our analysis, we'll use box plots to visually inspect potential outliers in the numerical features. Box plots display the distribution of data based on five summary statistics: minimum, first quartile, median, third quartile, and maximum.

We'll create box plots for the following numerical features:

1. CGPA
2. Internships
3. Projects
4. Workshops/Certifications
5. AptitudeTestScore
6. SoftSkillsRating
7. SSC_Marks
8. HSC_Marks

Observations

1. CGPA: There are a few outliers on the lower end.
2. Internships: The data seems mostly concentrated, but there are a few students with a higher number of internships which could be considered as potential outliers.
3. Projects: Data appears to be well-distributed without significant outliers.

4. Workshops/Certifications: We observe some potential outliers on the higher end.
5. AptitudeTestScore: A few outliers can be seen on the lower end.
6. SoftSkillsRating: There are potential outliers on both the lower and higher ends.
7. SSC_Marks: There are some outliers on the lower end.
8. HSC_Marks: A few outliers can be observed on the lower end.

Given the context of the dataset (student placements), it's plausible that the outliers represent genuine data points. For instance, a few students might exceptionally excel or perform below average in certain areas. Therefore, we can choose to retain these outliers for now, as they might provide valuable insights.

Phase 4: Modeling

Now that we have prepared the data, we'll move on to the modeling phase. Here, we'll:

1. Split the data into training and testing sets.
2. Train a baseline model.
3. Evaluate the model's performance.

Given the nature of our target variable (binary classification), we'll use a classification algorithm. Let's begin by splitting the data.

Data Splitting:

We'll split the data into a training set and a testing set. The training set will be used to train our model, while the testing set will be used to evaluate its performance.

Typically, a common split ratio is 80% for training and 20% for testing. We'll employ this split.

The data has been split into:

- Training set: 8000 samples
- Testing set: 2000 samples

Baseline Model:

For our baseline model, we'll use the Logistic Regression classifier, which is a widely-used algorithm for binary classification tasks.

Let's train the Logistic Regression model using the training set and evaluate its performance on the testing set.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score

# Initializing and training the Logistic Regression model
logreg = LogisticRegression(random_state=42)
logreg.fit(X_train, y_train)

# Predicting on the test set
y_pred = logreg.predict(X_test)

# Evaluating the model's performance
accuracy = accuracy_score(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
```

Model Evaluation:

Accuracy: The baseline Logistic Regression model achieved an accuracy of approximately 79.45% on the testing set.

Classification Report:

Precision:

- Class 0 (NotPlaced): 83%
- Class 1 (Placed): 74%

Recall:

- Class 0 (NotPlaced): 81%
- Class 1 (Placed): 77%

F1-score:

- Class 0 (NotPlaced): 82%
- Class 1 (Placed): 76%

These metrics provide a comprehensive view of the model's performance across both classes.

Model Selection:

Let's experiment with a few more classification algorithms to see if we can improve performance. We'll try the following:

1. Random Forest Classifier: An ensemble of decision trees that can capture complex patterns in the data.
2. Support Vector Machine (SVM): A powerful algorithm that finds the optimal hyperplane separating the classes.

3. Gradient Boosting Classifier: Boosting technique that builds trees sequentially, where each tree corrects the errors of the previous one.

For each classifier, we'll:

1. Train the model using the training set.
2. Evaluate its performance on the testing set.
3. Compare the results to our baseline Logistic Regression model.

```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC

# Initializing classifiers
rf = RandomForestClassifier(random_state=42)
svm = SVC(kernel='linear', random_state=42)
gboost = GradientBoostingClassifier(random_state=42)

# Dictionary to store results
results = {}

# Training and evaluating Random Forest
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
results['Random Forest'] = accuracy_rf

# Training and evaluating SVM
svm.fit(X_train, y_train)
y_pred_svm = svm.predict(X_test)
accuracy_svm = accuracy_score(y_test, y_pred_svm)
results['SVM'] = accuracy_svm

# Training and evaluating Gradient Boosting
gboost.fit(X_train, y_train)
y_pred_gboost = gboost.predict(X_test)
accuracy_gboost = accuracy_score(y_test, y_pred_gboost)
results['Gradient Boosting'] = accuracy_gboost
```

Here are the accuracy results for the additional classifiers:

- Random Forest Classifier: 78.35%
- Support Vector Machine (SVM): 79.55%
- Gradient Boosting Classifier: 79.1%

Observations:

- The Support Vector Machine (SVM) achieved the highest accuracy among the three models, slightly surpassing our baseline Logistic Regression model.
- The Random Forest and Gradient Boosting classifiers also performed comparably, with accuracies close to the baseline model.

Hyperparameter Tuning:

Hyperparameters are parameters that are not learned from the data but are set prior to the commencement of training. Tuning them can lead to better model performance.

We'll focus on tuning the hyperparameters of the Random Forest Classifier and the Gradient Boosting Classifier using a simplified Randomized search.

Random Forest Hyperparameters to Tune:

1. n_estimators: Number of trees in the forest.
2. max_depth: The maximum depth of the tree.

Gradient Boosting Hyperparameters to Tune:

1. n_estimators: Number of boosting stages to be run.
2. learning_rate: Step size shrinkage used to prevent overfitting.

Let's use RandomizedSearchCV to perform a random search over the hyperparameters of the Random Forest Classifier.

```
from sklearn.model_selection import RandomizedSearchCV

# Hyperparameters distribution for Random Forest
rf_dist = {
    'n_estimators': [50, 100, 150, 200],
    'max_depth': [None, 10, 20, 30, 40]
}

# Random search with cross-validation
rf_random_search = RandomizedSearchCV(rf, rf_dist, n_iter=8, scoring='accuracy',
cv=3,n_jobs=-1, random_state=42)
rf_random_search.fit(X_train, y_train)

# Extracting best parameters and best score
best_rf_params_random = rf_random_search.best_params_
best_rf_score_random = rf_random_search.best_score_
```

The Randomized Search for the Random Forest Classifier provided the following results:

Best Parameters:

- n_estimators: 100
- max_depth: 10

Best Cross-validated Accuracy: Approximately 79.56%

These parameters offer a slight improvement over our default settings.

let's proceed with hyperparameter tuning for the Gradient Boosting Classifier.

Similar to the Random Forest Classifier, we'll use RandomizedSearchCV to perform a random search over the hyperparameters of the Gradient Boosting Classifier.

```
# Hyperparameters distribution for Gradient Boosting
gboost_dist = {
    'n_estimators': [50, 100, 150, 200],
    'learning_rate': [0.01, 0.05, 0.1, 0.5]
}

# Random search with cross-validation
gboost_random_search = RandomizedSearchCV(gboost, gboost_dist, n_iter=8, scoring='accuracy',
gboost_random_search.fit(X_train, y_train)

# Extracting best parameters and best score
best_gboost_params_random = gboost_random_search.best_params_
best_gboost_score_random = gboost_random_search.best_score_
```

The Randomized Search for the Gradient Boosting Classifier provided the following results:

Best Parameters:

- n_estimators: 100
- learning_rate: 0.05

Best Cross-validated Accuracy: Approximately 80.21%

These parameters offer an improvement over our default settings for the Gradient Boosting Classifier.

Phase 5 : Evaluation

In this phase, we'll:

1. Train our selected models with the optimal hyperparameters on the entire training set.
2. Evaluate their performance on the testing set.
3. Compare the results to understand which model performs best.

We'll focus on the following models:

1. Logistic Regression (our baseline model)
2. Random Forest Classifier (with optimal hyperparameters)
3. Gradient Boosting Classifier (with optimal hyperparameters)

```
# Re-loading the data
data = pd.read_csv('/mnt/data/placedata v2.0 synthetic.csv')

# Preprocessing steps
data_encoded = pd.get_dummies(data, columns=['ExtracurricularActivities',
'PlacementTraining'], drop_first=True)
data_encoded['PlacementStatus'] = data_encoded['PlacementStatus']
.map({'Placed': 1, 'NotPlaced': 0})
X = data_encoded.drop(columns=['StudentID', 'PlacementStatus'])
y = data_encoded['PlacementStatus']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)

# Training and evaluating Random Forest with optimal hyperparameters
rf_best = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)
rf_best.fit(X_train, y_train)
y_pred_rf_best = rf_best.predict(X_test)
accuracy_rf_best = accuracy_score(y_test, y_pred_rf_best)

# Training and evaluating Gradient Boosting with optimal hyperparameters
gboost_best = GradientBoostingClassifier(n_estimators=100,
learning_rate=0.05, random_state=42)
gboost_best.fit(X_train, y_train)
y_pred_gboost_best = gboost_best.predict(X_test)
accuracy_gboost_best = accuracy_score(y_test, y_pred_gboost_best)

streamlined_evaluation = {
    'Random Forest': accuracy_rf_best,
    'Gradient Boosting': accuracy_gboost_best
}
```

Here are the evaluation results on the testing set:

- Random Forest Classifier: 79.00% accuracy
- Gradient Boosting Classifier: 79.45% accuracy

Both classifiers performed comparably, with the Gradient Boosting Classifier slightly outperforming the Random Forest Classifier.

Phase 6: Deployment Phase:

In the deployment phase of the CRISP-DM methodology, we take our final model and integrate it into the desired environment where it can be used to make predictions on new, unseen data.

The steps involved in deployment are:

1. Finalizing the Model: Choose the model that performed best during the evaluation phase.
2. Serialization: Convert the model into a format that can be stored and later loaded to make predictions.
3. Integration: Embed the model into the desired application or system.
4. Monitoring: Regularly monitor the model's performance to ensure it's still accurate and relevant. Models might need retraining as new data becomes available or if the underlying data distribution changes.

For our current exercise:

1. Finalizing the Model: We'll select the Gradient Boosting Classifier with the optimal hyperparameters, as it was among the top-performing models.
2. Serialization: We'll demonstrate how to serialize (or save) this model using Python so it can be deployed in a production environment.

Summary of the CRISP-DM Process:

1. Business Understanding: We started by understanding the nature of the provided student placement dataset and established the goal of predicting student placement status.
2. Data Understanding: Explored the dataset to understand its structure, variables, and initial insights.
3. Data Preparation: Conducted preprocessing steps like handling missing values, encoding categorical variables, and scaling numerical features. We also split the data into training and testing sets.
4. Modeling: Various classification models were trained, including Logistic Regression, Random Forest, and Gradient Boosting. We

also performed hyperparameter tuning and experimented with stacking as an ensemble method.

5. Evaluation: Assessed the performance of the models using accuracy and other metrics. Gradient Boosting emerged as one of the top-performing models.

6. Deployment: Discussed the deployment of the final model, serialized it for integration, and provided insights into post-deployment activities to ensure the model remains effective.

Concluding Remarks:

- Data science projects, like the one we went through, involve iterative processes. Continuous feedback and adjustments are often needed to refine the solution.
- The CRISP-DM methodology provides a structured framework, but flexibility is crucial. Depending on the project's needs, some steps might require more focus, while others might be quickly addressed.

References

Han, J., Kamber, M., & Pei, J. (2011). Data mining: concepts and techniques. Morgan Kaufmann.

Witten, I. H., Frank, E., Hall, M. A., & Pal, C. J. (2016). Data mining: practical machine learning tools and techniques. Morgan Kaufmann.

Friedman, J., Hastie, T., & Tibshirani, R. (2001). The elements of statistical learning. Springer.

Chollet, F., & Allaire, J. J. (2018). Deep learning with R. Manning Publications.

Raschka, S., & Mirjalili, V. (2019). Python machine learning. Packt Publishing Ltd.

Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., ... & Steinbach, M. (2008). Top 10 algorithms in data mining. Knowledge and Information Systems, 14(1), 1-37.

Provost, F., & Fawcett, T. (2013). Data science for business: What you need to know about data mining and data-analytic thinking. O'Reilly Media, Inc.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning. Springer.