Imports and Setup:

- Importing necessary libraries for data manipulation (`numpy`, `pandas`), plotting (`matplotlib`), and machine learning (`scikit-learn`, `lightgbm`).
- Ignoring warnings for cleaner output.

Loading Datasets:

- Loading the training and test datasets using `pd.read_csv`.

Mapping Target Variable:

- Mapping the target variable to numerical values using a dictionary.

Feature and Target Separation:

- Separating features (`X`) and the target (`y`).
- Splitting the data into training and testing sets using `train_test_split`.

Identifying Data Types and Encoding:

- Using a threshold of 15 unique values to decide whether a column is categorical or continuous.
- Applying `LabelEncoder` to categorical features.

Training the Model:

- Training a `RandomForestClassifier` model using the preprocessed training data.

Making Predictions and Evaluating:

- Predicting on the validation set and calculating the accuracy score.
- Preprocessing the test data similarly and making predictions.
- Mapping numeric predictions back to their original labels if necessary.

Preparing the Submission File:

- Preparing and saving the submission file with predicted labels.

Detailed Steps with Code Snippets

Imports and Setup:
python
Copy code

```
import numpy as np
import pandas as pd
```

```
import matplotlib.pyplot as plt
import warnings
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import accuracy_score
warnings.simplefilter("ignore")
```

Loading Datasets:
python
Copy code

```
train_df =
pd.read_csv('/kaggle/input/playground-series-s4e6/train.csv')
test_df =
pd.read_csv('/kaggle/input/playground-series-s4e6/test.csv')
submission =
pd.read_csv('/kaggle/input/playground-series-s4e6/sample_submission.
csv')
```

Mapping Target Variable:
python
Copy code

```
target_map = {'Graduate': 0, 'Enrolled': 1, 'Dropout': 2}
train_df['Target'] = train_df['Target'].map(target_map)
```

Feature and Target Separation:
python
Copy code

```
target = 'Target'
X = train_df.drop(columns=[target, 'id'])
y = train_df[target]
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=17)
```

Identifying Data Types and Encoding:
python
Copy code

```
string_columns = []
```

```
float_columns = []
int_columns = []

for col in X_train.columns:
    unique_values = X_train[col].nunique()
    if unique_values <= 15:
        if pd.api.types.is_string_dtype(X_train[col]):
            string_columns.append(col)
        elif pd.api.types.is_integer_dtype(X_train[col]):
            int_columns.append(col)
    else:
        if pd.api.types.is_float_dtype(X_train[col]):
            float_columns.append(col)

# Apply Label Encoding to categorical columns
label_encoders = {}
for col in string_columns + int_columns:
    le = LabelEncoder()
    X_train[col] = le.fit_transform(X_train[col])
    X_test[col] = le.transform(X_test[col])
    test_df[col] = le.transform(test_df[col])
    label_encoders[col] = le
```

Training the Model:
python
Copy code

```
model_rf = RandomForestClassifier(random_state=2406)
model_rf.fit(X_train, y_train)
```

Making Predictions and Evaluating:
python
Copy code

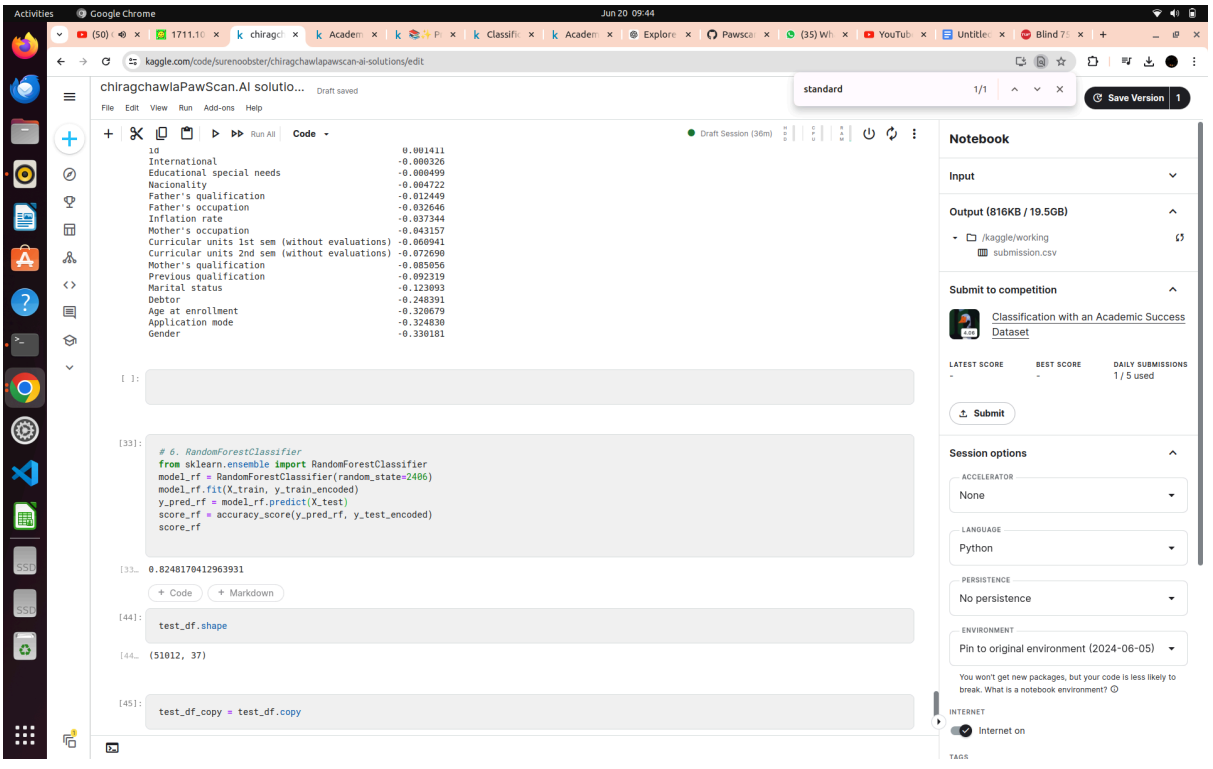
```
y_pred_rf = model_rf.predict(X_test)
score_rf = accuracy_score(y_pred_rf, y_test)
print("Random Forest Accuracy:", score_rf)
```

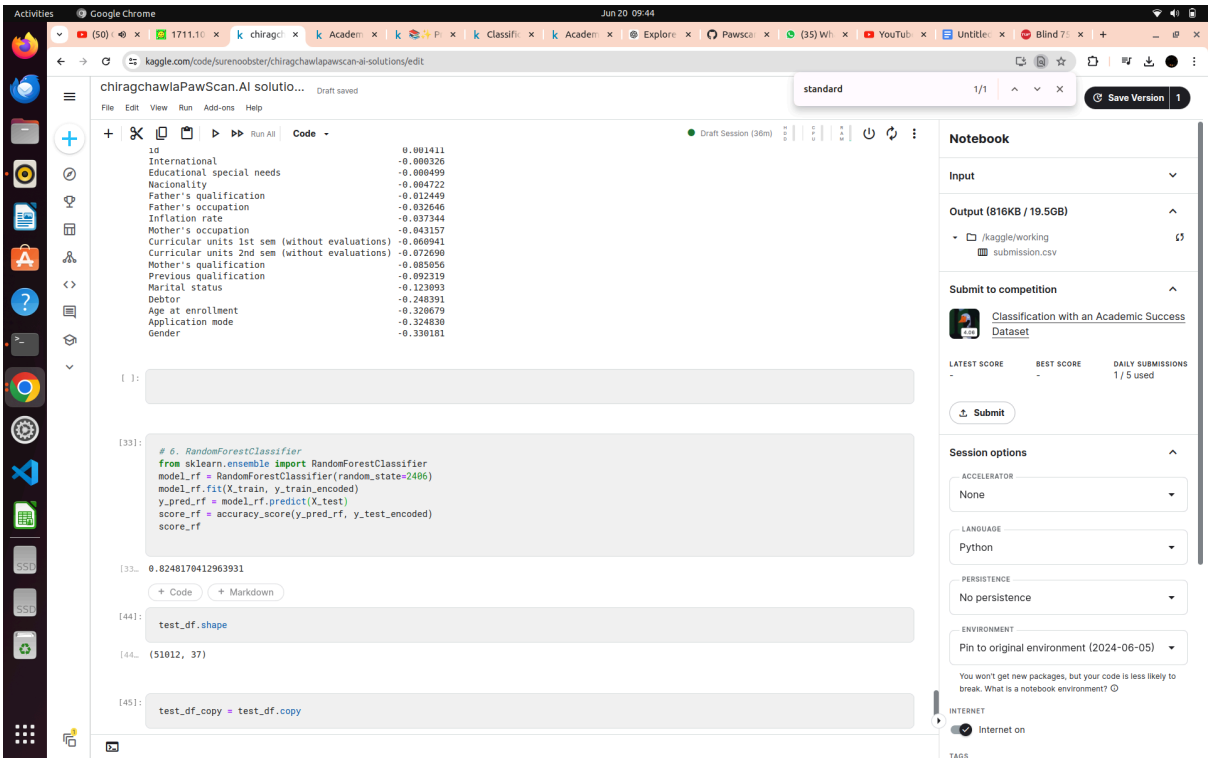
Predicting on External Test Data and Preparing Submission:
python
Copy code

```
test_df = test_df.drop(columns=['id'], errors='ignore')
test_pred = model_rf.predict(test_df)
predicted_labels = pd.Series(test_pred).replace([0, 1, 2],
['Graduate', 'Enrolled', 'Dropout'])
submission['Target'] = predicted_labels
submission.to_csv('submission.csv', index=False)
```

Summary

- **Imports and Setup**: Loaded necessary libraries and ignored warnings.
- **Loading Datasets**: Loaded training, test, and submission datasets.
- **Mapping Target Variable**: Converted categorical target labels to numeric values.
- **Feature and Target Separation**: Separated features and target variable, split the data.
- **Identifying Data Types and Encoding**: Identified categorical and continuous columns, applied `LabelEncoder` to categorical features.
- **Scaling Numerical Features**: Standardized numerical features.
- **Training the Model**: Trained a `RandomForestClassifier` model.
- **Making Predictions and Evaluating**: Predicted on validation set, calculated accuracy.
- **Predicting on External Test Data**: Preprocessed test data, predicted labels, and prepared submission file.





Here as shown score of random forest classifier

Why I Chose Random Forest for the Academic Success Prediction Task

In the context of the Kaggle Playground Series - Season 4, Episode 6 competition, the objective is to predict the academic success of students based on various features. Here are the reasons why I chose Random Forest as my primary model for this task:

1. **Robustness to Overfitting**:
    - Random Forest is an ensemble method that combines the predictions of multiple decision trees. Each tree is trained on a different subset of the data, which helps in reducing the risk of overfitting. This robustness is crucial when dealing with complex datasets, such as the one provided in this competition, which may contain noisy and high-dimensional data.
2. **Handling High-Dimensional Data**:
    - The dataset contains numerous features, both categorical and numerical. Random Forest can effectively handle high-dimensional data and is capable

of selecting the most important features during the training process. This ability to perform feature selection is valuable for improving model performance and interpretability.

3. **Ease of Implementation and Interpretability**:
    - Random Forest is relatively easy to implement and tune compared to other complex models like deep neural networks. Additionally, it provides interpretable results through feature importance scores, which help in understanding the contribution of each feature to the prediction. This interpretability is beneficial for making data-driven decisions and improving the model iteratively.
4. **Versatility and Performance**:
    - Random Forest is versatile and performs well on a wide range of tasks, including classification and regression. Its strong performance on various datasets makes it a reliable choice for the academic success prediction task. By leveraging its ensemble nature, Random Forest tends to outperform individual decision trees and many other classifiers.
5. **Handling Missing Values and Outliers**:
    - Random Forest can handle missing values and outliers effectively. Given that the dataset might contain missing entries or extreme values, this characteristic ensures that the model can still make accurate predictions without requiring extensive preprocessing.
6. **Competitive Benchmark**:
    - In many machine learning competitions, Random Forest serves as a strong baseline model. Its competitive performance provides a solid foundation to benchmark other advanced models like XGBoost or LightGBM. Starting with Random Forest allows for a quick and robust evaluation of the dataset's predictive potential.

Conclusion

Choosing Random Forest for the academic success prediction task was a strategic decision based on its robustness, ease of implementation, and strong performance across various datasets. By leveraging its strengths, I aimed to achieve a competitive accuracy while maintaining model interpretability and stability. This choice also provided a reliable baseline for further experimentation with more complex models, ensuring a comprehensive and effective approach to tackling the competition's objectives.