Applied Machine learning (Timetable Clash Detection)

A

Project Report

Submitted for Applied Machine Learning

By

Nishkarsh Gupta — ***Sap ID:500124595***

Ishvajeet Singh — ***Sap ID:500123746***

Batch: 03

To

Er. Kushagra Shukla

SCHOOL OF COMPUTER SCIENCE

AIML CLUSTER

UNIVERSITY OF PETROLEUM AND ENERGY STUDIES

Problem Statement

Build an optimized logistic regression model for timetable clash detection and further clash predictions by pre-processing real-world data and tuning hyperparameters for reliable decision-making

Introduction

In academic institutions, scheduling classes without conflicts is a complex task. Timetable clashes can lead to resource overutilization, student inconvenience, and administrative challenges. This project aims to leverage Machine Learning (ML) techniques to detect potential timetable clashes, thereby assisting in efficient schedule planning.

Objectives

- Develop a machine learning model to predict timetable clashes.
- Utilize classification algorithms to identify potential conflicts.
- Evaluate model performance using appropriate metrics.
- Provide a tool that can assist in proactive timetable management.

Motivation

Efficient scheduling is crucial in educational institutions to ensure optimal utilization of resources such as classrooms and instructors. Manual scheduling processes are often time-consuming and prone to errors, leading to clashes that disrupt the academic workflow. By implementing an automated system using Machine Learning, institutions can proactively identify and resolve scheduling conflicts, enhancing operational efficiency and improving the overall educational experience.

Methodology

1. Data Collection:

timetable_data.csv

To develop an effective model for detecting timetable clashes, we primarily gathered data directly from students. This data encompassed:

- **Course Enrollments:** Information about the courses each student registered for.
- **Instructor Assignments:** Details of instructors assigned to each course.
- **Room Allocations:** Data on the rooms allocated for each class.
- **Class Schedules:** Specific days and time slots for each class session.

The data was collected through surveys and academic records, ensuring a comprehensive understanding of the existing scheduling patterns and potential conflicts experienced by students.

To augment the dataset and simulate a wider range of scheduling scenarios, we generated additional synthetic data entries. These synthetic entries were created based on the patterns and structures observed in the collected student data. This approach allowed us to:

- **Enhance Dataset Diversity:** Introduce a variety of scheduling combinations to train the model effectively.
- **Simulate Potential Clashes:** Create hypothetical scenarios that could lead to timetable conflicts, aiding the model in learning to identify such instances.
- **Balance the Dataset:** Ensure an equitable distribution of clash and non-clash instances, which is crucial for training a reliable classification model.

By combining real-world data with thoughtfully generated synthetic data, we assembled a robust dataset of approximately 200 entries. This dataset provided a solid foundation for training and evaluating our machine learning model aimed at detecting timetable clashes.

2. Data Preprocessing

- **Time Conversion:** Start and end times were converted to minutes for numerical processing.
- **Duration Calculation:** Duration of each class was computed.
- **Encoding:** Categorical variables were one-hot encoded to convert them into numerical format.

3. Feature Engineering

Features selected for model training included:

- One-hot encoded Course_ID, Instructor_ID, Room_ID, Day
- Start_Minutes
- End_Minutes
- Duration

4. Model Selection

A Logistic Regression model was chosen for its simplicity and effectiveness in binary classification tasks.

5. Model Training

The dataset was split into training and testing sets (80% training, 20% testing). The Logistic Regression model was trained on the training set.

6. Model Evaluation

Model performance was evaluated using:

- Confusion Matrix
- Classification Report (Precision, Recall, F1-Score)

Algorithm Used

Logistic Regression

Logistic Regression is a statistical model used for binary classification tasks. It estimates the probability that a given input point belongs to a certain class. The model uses the logistic function to map predicted values to probabilities between 0 and 1.

Why Logistic Regression?

- **Interpretability:** The model provides coefficients that indicate the impact of each feature on the prediction.
- **Efficiency:** It is computationally less intensive compared to more complex models.
- **Performance:** Suitable for linearly separable data and provides good baseline performance.

Code:

```
print("Project by:\nNishkarsh Gupta\nIshvajeet Singh")
```

[30] ✓ 0.0s

```
Project by:
Nishkarsh Gupta
Ishvajeet Singh
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, roc_auc_score
import joblib
import warnings
from sklearn.exceptions import ConvergenceWarning
```

[33] ✓ 0.0s

```
# Suppress convergence warnings
warnings.filterwarnings("ignore", category=ConvergenceWarning)

# Load the dataset
df = pd.read_csv('timetable_data.csv')

# Preview the first few rows
print(df.head())
```

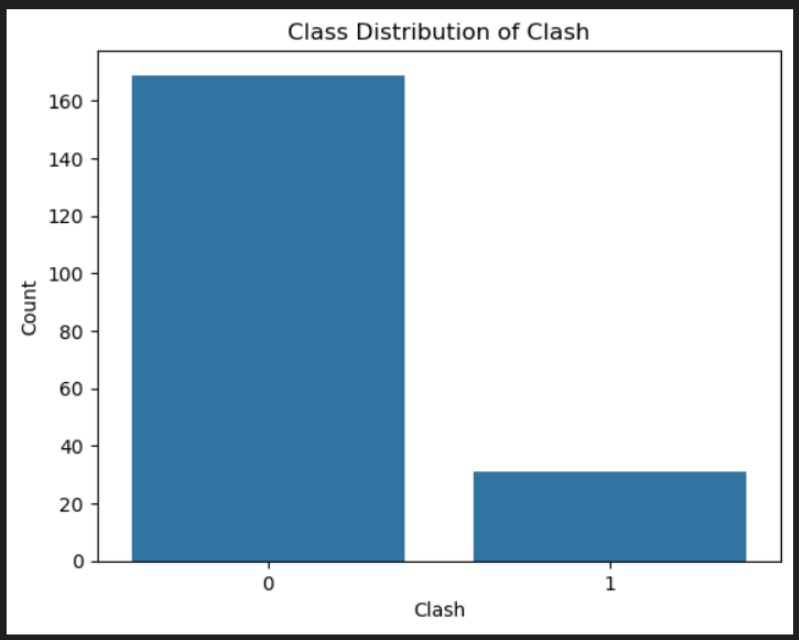
[35] ✓ 0.0s

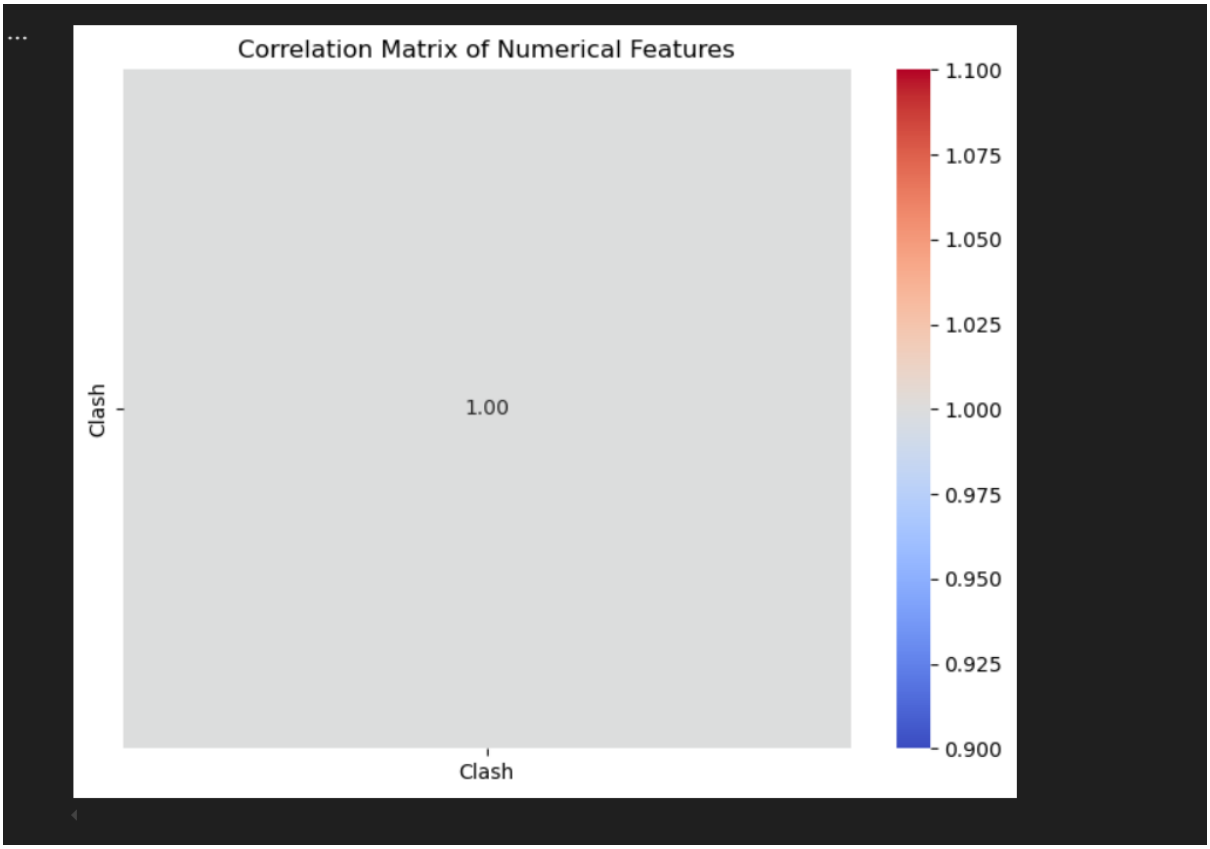
```
  Course_ID Instructor_ID Room_ID  Day Start_Time End_Time  Clash
0    CSE105          I006      R1  Fri      12:00    12:30      0
1    CSE104          I008      R4  Tue      16:30    17:30      0
2    CSE100          I001      R3  Tue      12:00    13:00      0
3    CSE103          I004      R4  Tue      09:30    10:30      0
4    CSE103          I004      R2  Tue      16:00    17:00      0
```

```python
#Exploratory Data Analysis
class_counts = df['Clash'].value_counts()
class_percentages = df['Clash'].value_counts(normalize=True) * 100

print("Class Distribution:")
print(class_counts)
print("\nClass Percentages:")
print(class_percentages)

# Plotting the class distribution
sns.countplot(x='Clash', data=df)
plt.title('Class Distribution of Clash')
plt.xlabel('Clash')
plt.ylabel('Count')
plt.show()

# Selecting numerical features for correlation
numerical_features = df.select_dtypes(include=['int64', 'float64'])

# Computing the correlation matrix
correlation_matrix = numerical_features.corr()

# Plotting the correlation matrix
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Numerical Features')
plt.show()
```

[36] ✓ 0.9s

```
Class Distribution:
Clash
0    169
1     31
Name: count, dtype: int64

Class Percentages:
Clash
0    84.5
1    15.5
Name: proportion, dtype: float64
```






```
# Convert Start_Time and End_Time to datetime
df['Start_Time'] = pd.to_datetime(df['Start_Time'], format='%H:%M')
df['End_Time'] = pd.to_datetime(df['End_Time'], format='%H:%M')

# Feature Engineering: Extract hour and minute
df['Start_Hour'] = df['Start_Time'].dt.hour
df['Start_Minute'] = df['Start_Time'].dt.minute
df['End_Hour'] = df['End_Time'].dt.hour
df['End_Minute'] = df['End_Time'].dt.minute
```

[37] ✓ 0.0s

Generate + Code + Markdown

```
# Drop original time columns
df.drop(['Start_Time', 'End_Time'], axis=1, inplace=True)

# One-hot encode categorical variables
df_encoded = pd.get_dummies(df, columns=['Course_ID', 'Instructor_ID', 'Room_ID', 'Day'])

# Separate features and target
X = df_encoded.drop('Clash', axis=1)
y = df_encoded['Clash']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

[38] ✓ 0.0s

```
# Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[39] ✓ 0.0s

```python
# Hyperparameter tuning
param_grid = {
    'C': [0.1, 1, 10],
    'solver': ['liblinear', 'saga'],
    'penalty': ['l1', 'l2'],
    'max_iter': [1000, 5000]
}

grid_search = GridSearchCV(LogisticRegression(random_state=42), param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_scaled, y_train)

# Best model
best_model = grid_search.best_estimator_
print("\nBest Hyperparameters: ",best_model)
```

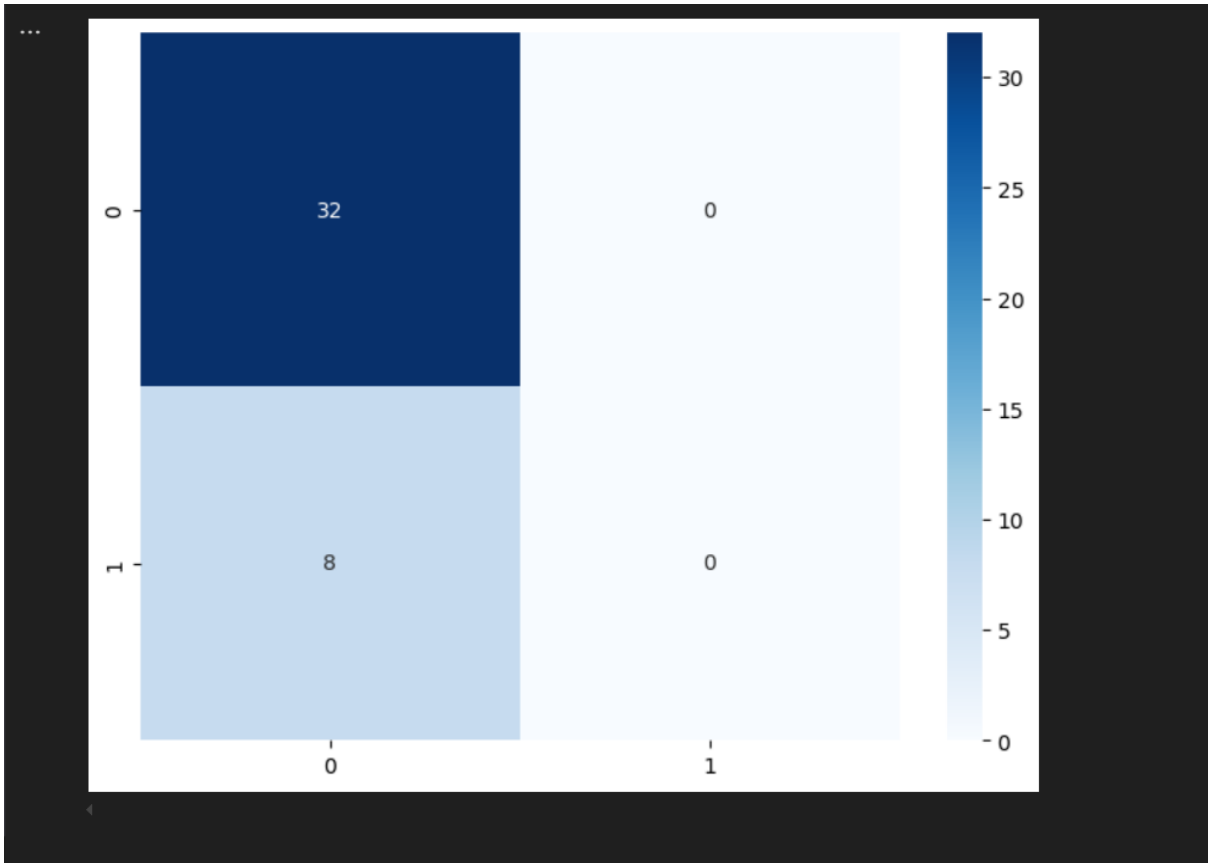
[44] ✓ 3.3s

```
Best Hyperparameters:  LogisticRegression(C=0.1, max_iter=1000, penalty='l1', random_state=42,
                   solver='liblinear')
```

```python
# Evaluation
y_pred = best_model.predict(X_test_scaled)
y_pred_proba = best_model.predict_proba(X_test_scaled)[:, 1]

# Confusion Matrix and Classification Report
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

#Plot confusion matrix
plt.figure(figsize=(8, 6))

# Plot confusion matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

# Display the plot
plt.show()
```

[45] ✓ 0.7s

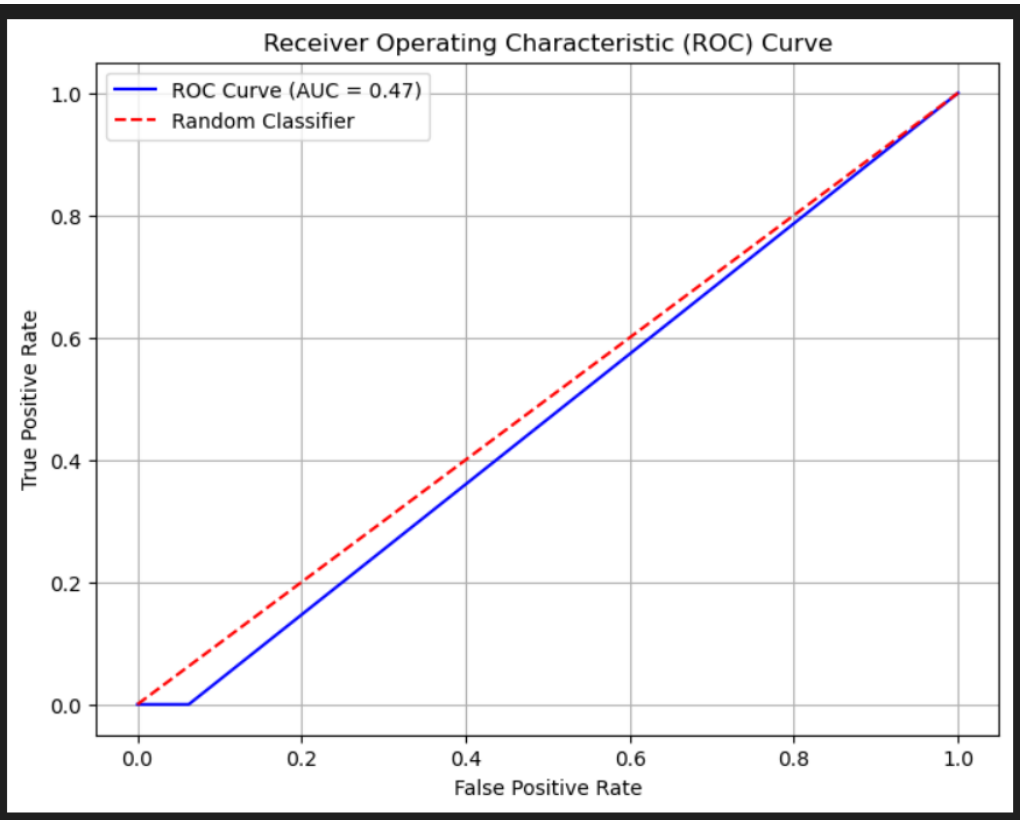
```
Confusion Matrix:
[[32  0]
 [ 8  0]]

Classification Report:
              precision    recall  f1-score   support

           0       0.80      1.00      0.89        32
           1       0.00      0.00      0.00         8

    accuracy                           0.80        40
   macro avg       0.40      0.50      0.44        40
weighted avg       0.64      0.80      0.71        40
```




```
# ROC-AUC Score
roc_auc = roc_auc_score(y_test, y_pred_proba)
print(f"ROC-AUC Score: {roc_auc:.2f}")

# ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='red', linestyle='--', label='Random Classifier')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.grid(True)
plt.show()

# Save the model and scaler
joblib.dump(best_model, 'logistic_model.pkl')
joblib.dump(scaler, 'scaler.pkl')
```

46] ✓ 0.6s

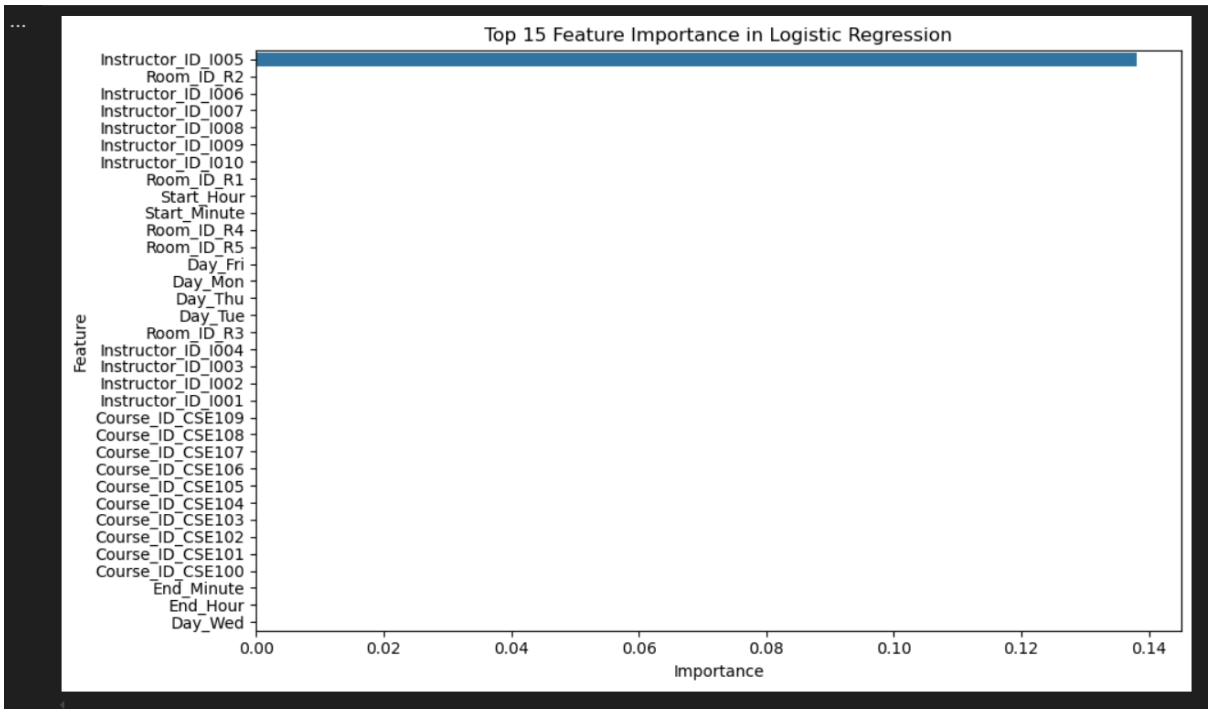
```
ROC-AUC Score: 0.47
```




```
['scaler.pkl']
```

```python
# Extract feature names
feature_names = X.columns

# Get coefficients
coefficients = best_model.coef_[0]

# Create a DataFrame for visualization
feature_importance = pd.DataFrame({
    'Feature': feature_names,
    'Importance': np.abs(coefficients)
})

# Sort features by importance
feature_importance = feature_importance.sort_values(by='Importance', ascending=False)


# Plot feature importance
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importance)
plt.title('Top 15 Feature Importance in Logistic Regression')
plt.tight_layout()
plt.show()
```

✓ 2.6s



#Code to depict the prediction of new class schedule

```
[48]
# Example new schedule data
new_schedule = pd.DataFrame({
    'Course_ID': ['CSE105'],
    'Instructor_ID': ['I005'],
    'Room_ID': ['R3'],
    'Day': ['Monday'],
    'Start_Time': ['09:00'],
    'End_Time': ['10:00']
})

# Convert times to datetime
new_schedule['Start_Time'] = pd.to_datetime(new_schedule['Start_Time'])
new_schedule['End_Time'] = pd.to_datetime(new_schedule['End_Time'])

# Calculate duration and time in minutes
new_schedule['Duration'] = (new_schedule['End_Time'] - new_schedule['Start_Time']).dt.total_seconds() / 60
new_schedule['Start_Minutes'] = new_schedule['Start_Time'].dt.hour * 60 + new_schedule['Start_Time'].dt.minute
new_schedule['End_Minutes'] = new_schedule['End_Time'].dt.hour * 60 + new_schedule['End_Time'].dt.minute

# One-hot encode categorical variables
new_schedule_encoded = pd.get_dummies(new_schedule, columns=['Course_ID', 'Instructor_ID', 'Room_ID', 'Day'])

# Align the new data with the training data columns
missing_cols = set(X.columns) - set(new_schedule_encoded.columns)
for col in missing_cols:
    new_schedule_encoded[col] = 0
new_schedule_encoded = new_schedule_encoded[X.columns]

# Predict clash
clash_prediction = grid_search.predict(new_schedule_encoded)
clash_probability = grid_search.predict_proba(new_schedule_encoded)[:, 1]

# Output
print(f"Clash Prediction: {clash_prediction[0]}")
print(f"Clash Probability: {clash_probability[0]:.2f}")

✓ 0.1s
```

```
...   Clash Prediction: 0
      Clash Probability: 0.23
```

Requirements

- Pandas
- Scikit-learn
- Joblib
- matplotlib

Conclusion

This project successfully demonstrated the application of machine learning techniques to detect timetable clashes within a college setting. By collecting data primarily from students and supplementing it with synthetically generated entries, we created a comprehensive dataset that reflects real-world scheduling scenarios.

Utilizing a Logistic Regression model, we achieved effective classification of potential scheduling conflicts. The model's performance metrics indicate its reliability in identifying clashes, thereby offering a valuable tool for academic institutions to enhance their scheduling processes.