**1. Importing Libraries:**

python

Copy code

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

import matplotlib.pyplot as plt

import seaborn as sns

* **pandas (pd)**: A powerful data manipulation and analysis library, used here to load and manage the dataset.
* **train\_test\_split**: A function from sklearn that splits the dataset into training and testing sets.
* **LinearRegression**: A class from sklearn used to perform linear regression, predicting a continuous outcome (in this case, GPA).
* **mean\_squared\_error, r2\_score**: Functions from sklearn used to evaluate the performance of the regression model.
* **matplotlib.pyplot (plt)**: A plotting library used to create visualizations.
* **seaborn (sns)**: An additional visualization library built on top of matplotlib, often used for creating more complex plots.

**2. Loading Data:**

python

Copy code

csv\_student\_path = r'C:\Users\Sarah Son Kim\class24\NU-VIRT-DATA-PT-02-2024-U-LOLC\02-Homework\Project4\_Team5\Resources\Student\_performance\_data.csv'

data = pd.read\_csv(csv\_student\_path)

* **csv\_student\_path**: A string containing the file path to the dataset.
* **pd.read\_csv**: Reads the CSV file into a pandas DataFrame, making it easy to manipulate and analyze the data.

**3. Defining Features and Target:**

python

Copy code

X = data[['StudentID', 'Age','Gender', 'Ethnicity', 'ParentalEducation', 'StudyTimeWeekly', 'Absences',

'Tutoring', 'ParentalSupport', 'Extracurricular', 'Sports', 'Music', 'Volunteering','GradeClass']]

y = data['GPA']

* **X**: A DataFrame containing the independent variables (features) that will be used to predict the target variable. In this case, it includes various student-related factors.
* **y**: A Series containing the dependent variable (target) that we want to predict, which is the GPA in this case.

**4. Splitting Data into Training and Testing Sets:**

python

Copy code

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

* **train\_test\_split**: This function splits the data into two sets:
  + **X\_train, y\_train**: The training set (70% of the data), used to train the model.
  + **X\_test, y\_test**: The testing set (30% of the data), used to evaluate the model's performance.
* **test\_size=0.3**: Specifies that 30% of the data should be used for testing, and the remaining 70% for training.
* **random\_state=42**: Ensures reproducibility by setting a seed for the random number generator.

**5. Normalizing the Features (Optional):**

python

Copy code

X\_train = (X\_train - X\_train.mean()) / X\_train.std()

X\_test = (X\_test - X\_test.mean()) / X\_test.std()

* **Normalization**: This step scales the features so that they have a mean of 0 and a standard deviation of 1. This can improve the performance of the model, especially for algorithms sensitive to the scale of input data.

**6. Training the Linear Regression Model:**

python

Copy code

model = LinearRegression()

model.fit(X\_train, y\_train)

* **model = LinearRegression()**: Initializes the linear regression model.
* **model.fit(X\_train, y\_train)**: Trains the model on the training data. The model learns the relationship between the features (X\_train) and the target variable (y\_train).

**7. Making Predictions on the Test Data:**

python

Copy code

y\_pred = model.predict(X\_test)

* **model.predict(X\_test)**: Uses the trained model to predict GPA values for the testing set. The predicted values are stored in y\_pred.

**8. Evaluating the Model:**

python

Copy code

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f'Mean Squared Error: {mse}')

print(f'R-squared: {r2}')

* **mean\_squared\_error(y\_test, y\_pred)**: Calculates the Mean Squared Error (MSE), a common metric for regression models that measures the average squared difference between the actual and predicted values.
* **r2\_score(y\_test, y\_pred)**: Calculates the R-squared (R²) value, which indicates how well the model's predictions match the actual data. An R² value closer to 1 indicates a better fit.
* **print(f'Mean Squared Error: {mse}')** and **print(f'R-squared: {r2}')**: Output the MSE and R² values to evaluate the model's performance.

**9. Visualizing the Actual vs Predicted Values:**

python

Copy code

plt.scatter(y\_test, y\_pred)

plt.xlabel('Actual GPA')

plt.ylabel('Predicted GPA')

plt.title('Actual vs Predicted GPA')

plt.plot([y.min(), y.max()], [y.min(), y.max()], color='red', lw=2) # Line for perfect prediction

plt.show()

* **plt.scatter(y\_test, y\_pred)**: Creates a scatter plot showing the actual GPA values (y\_test) against the predicted values (y\_pred).
* **plt.plot([y.min(), y.max()], [y.min(), y.max()], color='red', lw=2)**: Adds a red line representing perfect prediction (where actual values equal predicted values).
* **plt.show()**: Displays the plot.

This code provides a complete workflow for predicting GPA using linear regression, including loading data, preprocessing, training, evaluation, and visualization.