Project Name : Exam Perfomance Prediction

Section : "A" 3rd sem

Course : B.tech CSE (AI&DS)

Submitted to : Mr.Devansh Kasaudhan

Done by :

Kushveer kaur (72411346)

Samatha sri (72411169)

Ram Charan (72411363)

Kanishk singla (72410471)

Exam Performance Prediction using Linear Regression

Introduction

The Exam Performance Prediction project aims to predict student academic performance using machine learning, specifically the Linear Regression model. It helps educators and institutions identify factors affecting students' results and provide personalized academic interventions.

Objective

The primary goal of this project is to create an efficient predictive model that can forecast students' exam results based on various independent variables such as study hours, attendance, and past performance. The insights derived can be used to guide educational improvements.

Tools and Technologies Used

The project utilizes the following tools and technologies:

- Python Programming Language
- Pandas and NumPy for Data Manipulation
- Scikit-learn for Machine Learning Model Implementation
- Matplotlib and Seaborn for Data Visualization
- Google colab for Development and Experimentation

Methodology

The project follows a structured process consisting of data collection, preprocessing, exploratory analysis, model training, evaluation, and interpretation of results.

Exploratory Data Analysis (EDA)

EDA involves understanding the data through visualization and statistical summaries. It helps identify key patterns, trends, and relationships between features.

- Key steps performed during EDA include:
- Checking for missing values and data inconsistencies.
- Analyzing the distribution of marks, attendance, and study hours.
- Using scatter plots to observe relationships between study hours and exam scores.
- Generating a correlation matrix to identify strongly related variables.

Explanation of each code

import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LinearRegression,SGDRegressor

from sklearn.metrics import mean_squared_error

import matplotlib.pyplot as plt

import seaborn as sns

Explanation :

This section of code will be explained in detail. It may include import statements, data loading, visualization, model training, prediction, and evaluation depending on the context.

Output :

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression,SGDRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df = pd.read_csv("/content/student_habits_performance.csv")
df
```

	student_id	age	gender	study_hours_per_day	social_media_hours	netflix_hours	part_time_job	attendance_percentage	sleep_hours	diet_quality	exercise_frequency	pare
0	S1000	23	Female	0.0	1.2	1.1	No	85.0	8.0	Fair	6	
1	S1001	20	Female	6.9	2.8	2.3	No	97.3	4.6	Good	6	
2	S1002	21	Male	1.4	3.1	1.3	No	94.8	8.0	Poor	1	
3	S1003	23	Female	1.0	3.9	1.0	No	71.0	9.2	Poor	4	
4	S1004	19	Female	5.0	4.4	0.5	No	90.9	4.9	Fair	3	
...	...	...	...	...	...	...	...	...	...	...	...	
995	S1995	21	Female	2.6	0.5	1.6	No	77.0	7.5	Fair	2	
996	S1996	17	Female	2.9	1.0	2.4	Yes	86.0	6.8	Poor	1	
997	S1997	20	Male	3.0	2.6	1.3	No	61.9	6.5	Good	5	
998	S1998	24	Male	5.4	4.1	1.1	Yes	100.0	7.6	Fair	0	
999	S1999	19	Female	4.3	2.9	1.9	No	89.4	7.1	Good	2	

1000 rows × 16 columns

df.info()

Explanation :

The section of code displays information about the DataFrame. This provides a concise summary including the index dtype and columns, non-null values and their types, and memory usage. This is useful for a quick overview of the dataset's structure and to identify missing values.

Output:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 15 columns):
 #   Column                         Non-Null Count  Dtype  
---  ------                         --------------  -----  
 0   age                            1000 non-null   int64  
 1   gender                         1000 non-null   int64  
 2   study_hours_per_day            1000 non-null   float64
 3   social_media_hours             1000 non-null   float64
 4   netflix_hours                  1000 non-null   float64
 5   part_time_job                  1000 non-null   int64  
 6   attendance_percentage          1000 non-null   float64
 7   sleep_hours                    1000 non-null   float64
 8   diet_quality                   1000 non-null   int64  
 9   exercise_frequency             1000 non-null   int64  
 10  parental_education_level       1000 non-null   int64  
 11  internet_quality               1000 non-null   int64  
 12  mental_health_rating           1000 non-null   int64  
 13  extracurricular_participation  1000 non-null   int64  
 14  exam_score                     1000 non-null   float64
dtypes: float64(6), int64(9)
memory usage: 117.3 KB
```

df.head()

Explanation:

The section of code displays the first 5 rows of the DataFrame df . This is helpful for getting a quick look at the data and understanding its structure.

Output:

```
df.head()
```

	student_id	age	gender	study_hours_per_day	social_media_hours	netflix_hours	part_time_job	attendance_percentage	sleep_hours	diet_quality	exercise_frequency	parent
0	S1000	23	Female	0.0	1.2	1.1	No	85.0	8.0	Fair	6	
1	S1001	20	Female	6.9	2.8	2.3	No	97.3	4.6	Good	6	
2	S1002	21	Male	1.4	3.1	1.3	No	94.8	8.0	Poor	1	
3	S1003	23	Female	1.0	3.9	1.0	No	71.0	9.2	Poor	4	
4	S1004	19	Female	5.0	4.4	0.5	No	90.9	4.9	Fair	3	

df.tail()

Explanation:

The section of code displays the last 5 rows of the DataFrame . This is useful for seeing the end of the dataset and checking for any potential issues like incomplete or footer rows.

Output:

```
df.tail()
```

	student_id	age	gender	study_hours_per_day	social_media_hours	netflix_hours	part_time_job	attendance_percentage	sleep_hours	diet_quality	exercise_frequency	pare
995	S1995	21	Female	2.6	0.5	1.6	No	77.0	7.5	Fair	2	
996	S1996	17	Female	2.9	1.0	2.4	Yes	86.0	6.8	Poor	1	
997	S1997	20	Male	3.0	2.6	1.3	No	61.9	6.5	Good	5	
998	S1998	24	Male	5.4	4.1	1.1	Yes	100.0	7.6	Fair	0	
999	S1999	19	Female	4.3	2.9	1.9	No	89.4	7.1	Good	2	

df.describe()

Explanation:

The section of code provides a statistical summary of the numerical columns in the DataFrame df . This includes counts, mean, standard deviation, minimum, maximum, and quartile values, which are helpful for understanding the distribution and characteristics of the numerical data.

Output:

```
df.describe()
```

	age	study_hours_per_day	social_media_hours	netflix_hours	attendance_percentage	sleep_hours	exercise_frequency	mental_health_rating	exam_score
count	1000.0000	1000.00000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	20.4980	3.55010	2.505500	1.819700	84.131700	6.470100	3.042000	5.438000	69.601500
std	2.3081	1.46889	1.172422	1.075118	9.399246	1.226377	2.025423	2.847501	16.888564
min	17.0000	0.00000	0.000000	0.000000	56.000000	3.200000	0.000000	1.000000	18.400000
25%	18.7500	2.60000	1.700000	1.000000	78.000000	5.600000	1.000000	3.000000	58.475000
50%	20.0000	3.50000	2.500000	1.800000	84.400000	6.500000	3.000000	5.000000	70.500000
75%	23.0000	4.50000	3.300000	2.525000	91.025000	7.300000	5.000000	8.000000	81.325000
max	24.0000	8.30000	7.200000	5.400000	100.000000	10.000000	6.000000	10.000000	100.000000

#checks if there is missing value or not df.isnull().sum()

Explanation:

The section of code checks for missing values in each column of the DataFrame df, which returns a boolean DataFrame where True indicates a missing value. The .sum() method is then applied to count the number of True values (missing values) in each column. This output shows the total number of missing values per column.

Output:

```
df.isnull().sum()
```

	0
age	0
gender	0
study_hours_per_day	0
social_media_hours	0
netflix_hours	0
part_time_job	0
attendance_percentage	0
sleep_hours	0
diet_quality	0
exercise_frequency	0
parental_education_level	0
internet_quality	0
mental_health_rating	0
extracurricular_participation	0
exam_score	0

dtype: int64

#replaced the null values

df['parental_education_level'].value_counts()

Explanation:

df['parental_education_level']: This part selects the column named 'parental_education_level' from your DataFrame df**.value_counts()**: This is a pandas method that operates on a Series (which is what df['parental_education_level'] is). It counts how many times each unique value appears in that Series and returns a new Series where the unique values are the index and their counts are the corresponding values.In the output of this code, you saw the counts for 'High School', 'Bachelor', and 'Master', indicating how many students' parents have each of these education levels in your dataset.

```
df['parental_education_level'].value_counts()
```

parental_education_level	count
1	483
0	350
2	167

dtype: int64

#Encoding Categorical Data using LableEncoder

from sklearn.preprocessing import LabelEncoder

le= LabelEncoder()

cols_to_encode = ['gender','part_time_job','diet_quality','internet_quality','parental_education_level','extracurricular_participation']

for col in cols_to_encode:

df[col]=le.fit_transform(df[col])

Defination:

Label Encoding is a data preprocessing technique used to convert categorical (text) data into numerical values so that machine learning models can easily understand and process the information. Each unique category in a column is assigned a unique integer value.

Explanation:

The section of code in cell simply displays the entire DataFrame df after you have performed the label encoding on the categorical columns.

Output:

```
df
```

	age	gender	study_hours_per_day	social_media_hours	netflix_hours	part_time_job	attendance_percentage	sleep_hours	diet_quality	exercise_frequency	parental_educati
0	23	0	0.0	1.2	1.1	0	85.0	8.0	0	6	
1	20	0	6.9	2.8	2.3	0	97.3	4.6	1	6	
2	21	1	1.4	3.1	1.3	0	94.8	8.0	2	1	
3	23	0	1.0	3.9	1.0	0	71.0	9.2	2	4	
4	19	0	5.0	4.4	0.5	0	90.9	4.9	0	3	
...	...	...	...	...	...	...	...	...	...	...	
995	21	0	2.6	0.5	1.6	0	77.0	7.5	0	2	
996	17	0	2.9	1.0	2.4	1	86.0	6.8	2	1	
997	20	1	3.0	2.6	1.3	0	61.9	6.5	1	5	
998	24	1	5.4	4.1	1.1	1	100.0	7.6	0	0	
999	19	0	4.3	2.9	1.9	0	89.4	7.1	1	2	

1000 rows × 15 columns

#Detection of Outliers Using Boxplot in Student Dataset

```
import matplotlib.pyplot as plt

import seaborn as sns

# Select numeric columns

numeric_cols = ['study_hours_per_day', 'social_media_hours', 'attendance_percentage',

                'sleep_hours', 'exercise_frequency', 'mental_health_rating', 'exam_score']

plt.figure(figsize=(12,8))

df[numeric_cols].boxplot()

plt.title("Boxplot to Detect Outliers in Student Dataset")

plt.ylabel("Value Range")

plt.xticks(rotation=45)

plt.show()
```

Definition:

A boxplot is a graph that shows how data is spread out.It helps to find outliers values that are much higher or lower than the rest.It displays the minimum, median, and maximum values in the dataset.

Explanation:

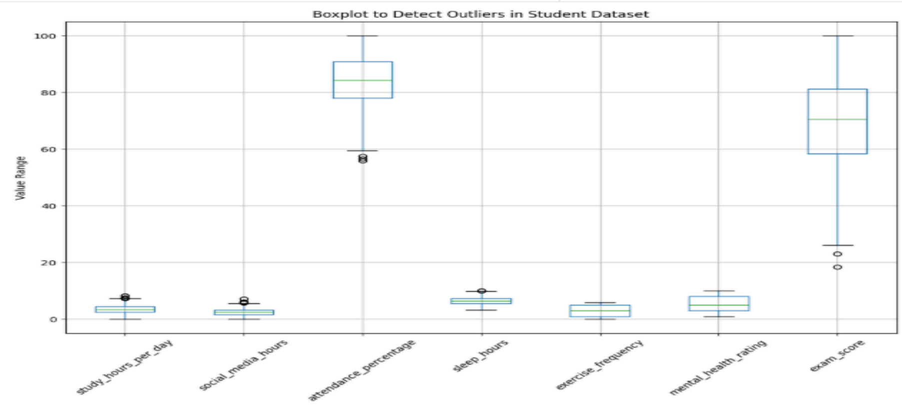
The section of code in cell generates a boxplot to visualize the distribution and detect potential outliers in several numeric columns of your DataFrame.

Output:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Select numeric columns
numeric_cols = ['study_hours_per_day', 'social_media_hours', 'attendance_percentage',
                'sleep_hours', 'exercise_frequency', 'mental_health_rating', 'exam_score']

plt.figure(figsize=(12,8))
df[numeric_cols].boxplot()
plt.title("Boxplot to Detect Outliers in Student Dataset")
plt.ylabel("Value Range")
plt.xticks(rotation=45)
plt.show()
```



#Visualization of Relationship Between Study Hours and Exam Score

 # visualizing the data

plt.scatter(df['study_hours_per_day'],df['exam_score'])

plt.xlabel('study_hours_per_day')

plt.ylabel('exam_score')

plt.title("study_hours vs marks")

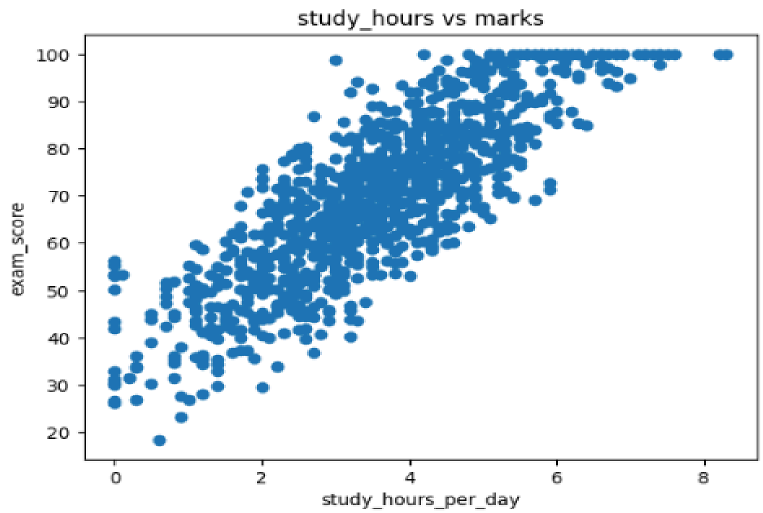
plt.show()

Explanation:

This code creates a scatter plot to visualize the relationship between the number of study hours per day and the exam score.

Output:

```
# visualizing the data
plt.scatter(df['study_hours_per_day'],df['exam_score'])
plt.xlabel('study_hours_per_day')
plt.ylabel('exam_score')
plt.title("study_hours vs marks")
plt.show()
```



#Scatter Plot of Netflix Hours vs Exam Score

plt.scatter(df['netflix_hours'], df['exam_score'], color='teal')

plt.title("Netflix Hours vs Exam Score")

plt.xlabel("Netflix Hours")

plt.ylabel("Exam Score")

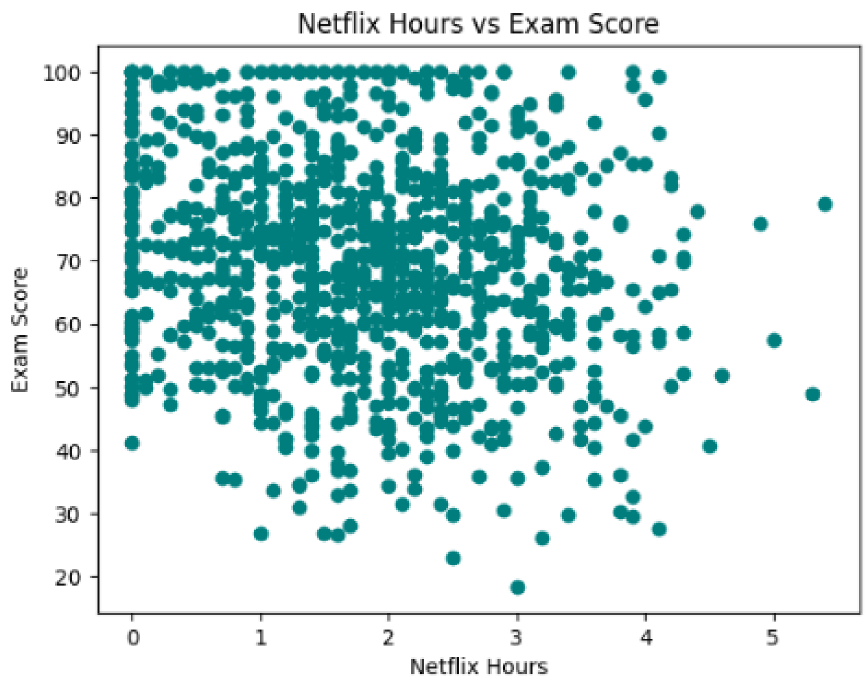
plt.show()

Explanation:

This code generates a scatter plot to show the relationship between the number of hours spent watching Netflix and the exam score**.**

Output:

```python
plt.scatter(df['netflix_hours'], df['exam_score'], color='teal')
plt.title("Netflix Hours vs Exam Score")
plt.xlabel("Netflix Hours")
plt.ylabel("Exam Score")
plt.show()
```



#Pie Chart Showing Distribution of Parental Education Levels

plt.figure(figsize=(7,5))

 # Create a mapping from encoded values to original labels

 # Assuming alphabetical order for LabelEncoder: 0:Bachelor, 1:High School, 2:Master

 # We can confirm this by checking the original value_counts and encoded values.

```python
education_mapping  = {
   0: 'Bachelor',
    1: 'High School',
   2: 'Master'
}# Added closing curly brace here
```

Apply the mapping to get the labels for the pie chart

pie_labels = df['parental_education_level'].value_counts().rename(index=education_mapping)

pie_labels.plot(kind='pie', autopct='%1.1f%%', startangle=90)

plt.title('Distribution of Parental Education Levels')

plt.ylabel('')

plt.show()

Explanation:

This section of code generates a pie chart to show the distribution of parental education levels in the dataset.

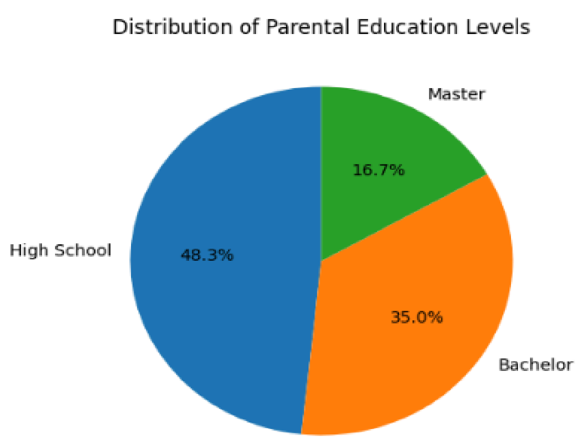
Output:

```
plt.figure(figsize=(7,5))

# Create a mapping from encoded values to original labels
# Assuming alphabetical order for LabelEncoder: 0:Bachelor, 1:High School, 2:Master
# We can confirm this by checking the original value_counts and encoded values.
education_mapping  = {
    0: 'Bachelor',
    1: 'High School',
    2: 'Master'
 }# Added closing curly brace here

# Apply the mapping to get the labels for the pie chart
pie_labels = df['parental_education_level'].value_counts().rename(index=education_mapping)

pie_labels.plot(kind='pie', autopct='%1.1f%%', startangle=90)
plt.title('Distribution of Parental Education Levels')
plt.ylabel('')
plt.show()
```



Bar Graph of Average Exam Scores by Parental Education Level

average_scores_by_education = df.groupby('parental_education_level')['exam_score'].mean().reset_index()

average_scores_by_education['parental_education_label'] = average_scores_by_education['parental_education_level'].map(education_mapping)

plt.figure(figsize=(8, 6))

sns.barplot(x='parental_education_label', y='exam_score', data=average_scores_by_education, palette='viridis')

plt.title('Average Exam Score by Parental Education Level')

plt.xlabel('Parental Education Level')

plt.ylabel('Average Exam Score')

plt.show()

Explanation:

This section of code calculates and visualizes the average exam score for each parental education level using a bar plot.

Output:

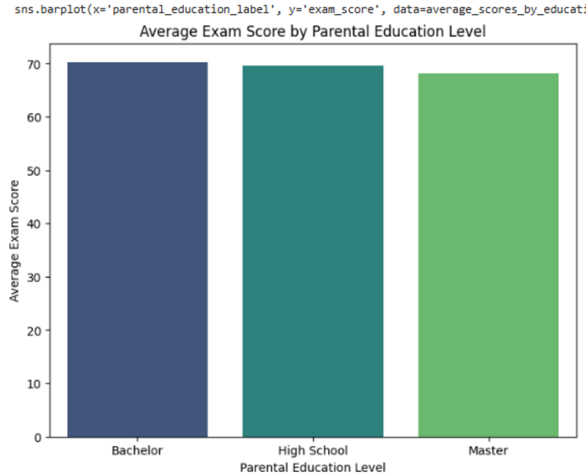
```
average_scores_by_education = df.groupby('parental_education_level')['exam_score'].mean().reset_index()
average_scores_by_education['parental_education_label'] = average_scores_by_education['parental_education_level'].map(education_mapping)

plt.figure(figsize=(8, 6))
sns.barplot(x='parental_education_label', y='exam_score', data=average_scores_by_education, palette='viridis')
plt.title('Average Exam Score by Parental Education Level')
plt.xlabel('Parental Education Level')
plt.ylabel('Average Exam Score')
plt.show()
```

```
/tmp/ipython-input-3712268352.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x='parental_education_label', y='exam_score', data=average_scores_by_education, palette='viridis')
```



#Line Graph of Sleep Hours vs Average Exam Score

```
sleep_trend = df.groupby('sleep_hours')['exam_score'].mean().reset_index()

plt.figure(figsize=(8,5))

plt.plot(sleep_trend['sleep_hours'], sleep_trend['exam_score'], marker='o',
color='orange', linewidth=2)

plt.title("Trend: Sleep Hours vs Average Exam Score")

plt.xlabel("Sleep Hours per Day")

plt.ylabel("Average Exam Score")

plt.grid(True)

plt.show()
```

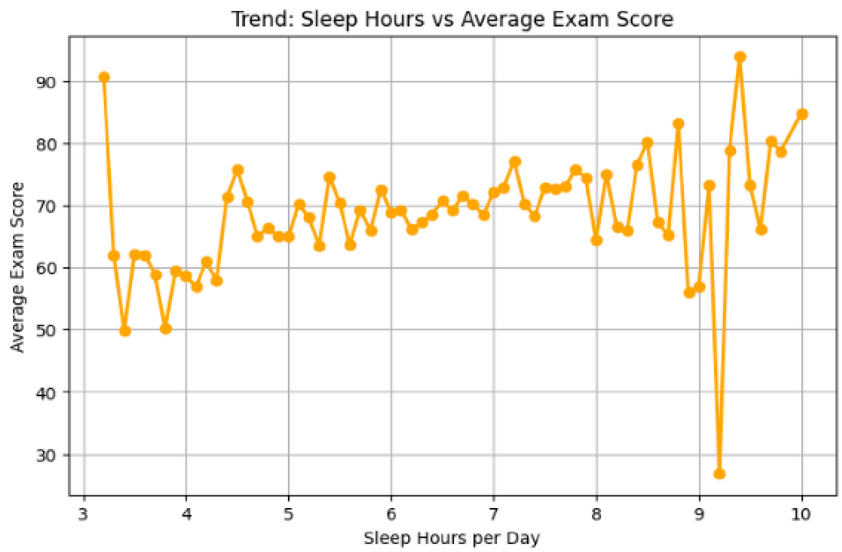
Explanation:

This section of code analyzes and visualizes the trend between the number of sleep hours and the average exam score using a line plot.

Output:

```python
sleep_trend = df.groupby('sleep_hours')['exam_score'].mean().reset_index()

plt.figure(figsize=(8,5))
plt.plot(sleep_trend['sleep_hours'], sleep_trend['exam_score'], marker='o', color='orange', linewidth=2)
plt.title("Trend: Sleep Hours vs Average Exam Score")
plt.xlabel("Sleep Hours per Day")
plt.ylabel("Average Exam Score")
plt.grid(True)
plt.show()
```



#Line Graph of Attendence vs Exam Score

plt.figure(figsize=(7,5))

sns.regplot(x='attendance_percentage', y='exam_score', data=df, color='green')

plt.title("Attendance vs Exam Score (with Trend Line)")

plt.xlabel("Attendance Percentage")

plt.ylabel("Exam Score")

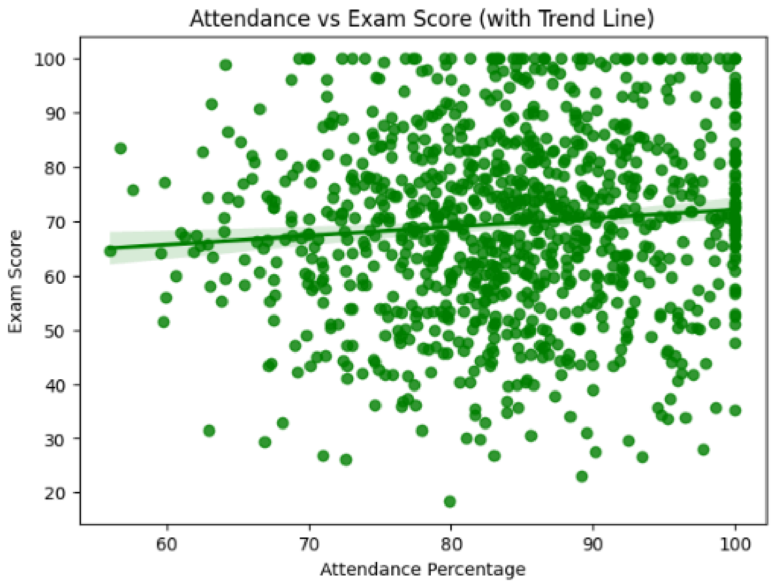
plt.show()

Explanation:

This code generates a scatter plot with a regression line to show the relationship between attendance percentage and exam score

Output:

```python
plt.figure(figsize=(7,5))
sns.regplot(x='attendance_percentage', y='exam_score', data=df, color='green')
plt.title("Attendance vs Exam Score (with Trend Line)")
plt.xlabel("Attendance Percentage")
plt.ylabel("Exam Score")
plt.show()
```



<u>Splitting Dataset into Training and Testing Sets for Linear Regression Model:</u>

```python
# droping the student id as it is of our no use
df = df.drop('student_id' , axis=1)
```

```python
X = df.drop(['exam_score'], axis = 1 ) # features all columns except exam score and student_id
y = df['exam_score']# target that value we want to predict
```

```python
# spliting into training and testing
X_train, X_test, y_train , y_test = train_test_split(X, y , test_size = 0.2 , random_state = 42)
# test_size is 20% data for testing
#random state ensures same split every time
```

<u>Training and Making Predictions Using Linear Regression Model:</u>

```python
#train the regression model
model = LinearRegression()
model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```python
y_pred = model.predict(X_test) # make predictions
```

Model Building

A Linear Regression model was built to predict exam scores. The dataset was divided into training and testing sets. The model was trained using the training data and tested on unseen data.

- Linear Regression Equation:
- Exam_Score = $\beta_0 + \beta_1$(Study_Hours) + β_2(Attendance) + β_3(Previous_Grades) + ε

Model Evaluation

The model's performance was evaluated using several statistical metrics. These include:

- Mean Absolute Error (MAE)
- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- R^2 Score – to measure the proportion of variance explained by the model

The Linear Regression model achieved a high R^2 score, indicating strong predictive capability.

#from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score:

mae = mean_absolute_error(y_test, y_pred)

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)

print("Mean Absolute Error:", mae)

print("Mean Squared Error:", mse)

print("R² Score:", r2)

Explanation:

This code calculates and prints the evaluation metrics for the trained linear regression model. These metrics help assess how well the model performs in predicting exam scores

Mean Absolute Error (MAE)

It is defined as the **average of the absolute differences** between the predicted values and the actual (true) values.

Mean Squared Error (MSE)

Mean squared error is a commonly used metric to measure how well a model's predictions match the actual data

Root Mean Squared Error (RMSE)

It is defined as the **square root of the average of the squared differences** between the predicted values and the actual (true) values.

Prepeocessing Sudent Performance Dataset for Machine Learning

```
# droping the student id as it is of our no use

df = df.drop('student_id' , axis=1)

df = pd.read_csv("/content/student_habits_performance.csv")

df
```

Explanation:

This section of the code cell reloads the original dataset from the specified CSV file path into the DataFrame df. This effectively resets the DataFrame to its initial state, including the 'student_id' column and the original categorical data, undoing any previous modifications like dropping columns or encoding.

Output:

```
# droping the student id as it is of our no use
df = df.drop('student_id' , axis=1)
```

```
df = pd.read_csv("/content/student_habits_performance.csv")
df
```

	student_id	age	gender	study_hours_per_day	social_media_hours	netflix_hours	part_time_job	attendance_percentage	sleep_hours	diet_quality	exercise_frequency	parental_education_level	i
0	S1000	23	Female	0.0	1.2	1.1	No	85.0	8.0	Fair	6	Master	
1	S1001	20	Female	6.9	2.8	2.3	No	97.3	4.6	Good	6	High School	
2	S1002	21	Male	1.4	3.1	1.3	No	94.8	8.0	Poor	1	High School	
3	S1003	23	Female	1.0	3.9	1.0	No	71.0	9.2	Poor	4	Master	
4	S1004	19	Female	5.0	4.4	0.5	No	90.9	4.9	Fair	3	Master	
...	...	...	...	...	...	...	...	...	...	...	...	...	
995	S1995	21	Female	2.6	0.5	1.6	No	77.0	7.5	Fair	2	High School	
996	S1996	17	Female	2.9	1.0	2.4	Yes	86.0	6.8	Poor	1	High School	
997	S1997	20	Male	3.0	2.6	1.3	No	61.9	6.5	Good	5	Bachelor	
998	S1998	24	Male	5.4	4.1	1.1	Yes	100.0	7.6	Fair	0	Bachelor	
999	S1999	19	Female	4.3	2.9	1.9	No	89.4	7.1	Good	2	Bachelor	

1000 rows × 16 columns

Output:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("R² Score:", r2)
```

```
Mean Absolute Error: 4.172475975530878
Mean Squared Error: 26.497748783647193
R² Score: 0.8966663721200843
```

```
print(model)
```

Example:

```
# Example new student's data (update values as per your dataset)
# Format: [age, gender, study_hours_per_day, social_media_hours, netflix_hours,
#          part_time_job, attendance_percentage, sleep_hours, diet_quality,
#          exercise_frequency, parental_education_level, internet_quality,
#          extracurricular_participation, mental_health_rating]

new_student = [[18, 1, 4, 2, 1, 0, 85, 7, 2, 1, 2, 1, 1, 8]]

predicted_score = model.predict(new_student)
print("Predicted Exam Score:", predicted_score[0])
```

```
Predicted Exam Score: 65.21970325754768
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
```

Results and Interpretation

The analysis showed that study hours and attendance had a strong positive correlation with exam scores. Students who studied consistently and maintained good attendance performed significantly better.

Future Scope

While the Linear Regression model performs well, future enhancements can further improve prediction accuracy.

- Possible extensions include:
- Implementing advanced models like Random Forest, Decision Trees, and XGBoost.
- Incorporating additional features such as parental education, socio-economic background, and learning habits.
- Developing a dashboard for real-time performance prediction.
- Integrating the model with school information systems for automatic data collection.

Conclusion

The Exam Performance Prediction project successfully demonstrates how Linear Regression can be used to model and predict student outcomes. It emphasizes the importance of data-driven approaches in education, enabling teachers and policymakers to identify learning gaps early and take timely action. This approach can revolutionize academic planning and support systems through predictive analytics.