Predict Student Exam Score

Objective: Predict the exam scores of students based on the number of hours they studied.

Steps to Solve the Problem

- Load the Data:

Read the data from a CSV file or use a hardcoded dataset.

- Explore the Data:

Check for missing values, data types, basic statistics, and visualize the relationship between Hours and Scores. Prepare the Data:

- Split the data into training and test sets.
- Train the Model:

Fit a linear regression model using the training data. Evaluate the Model:

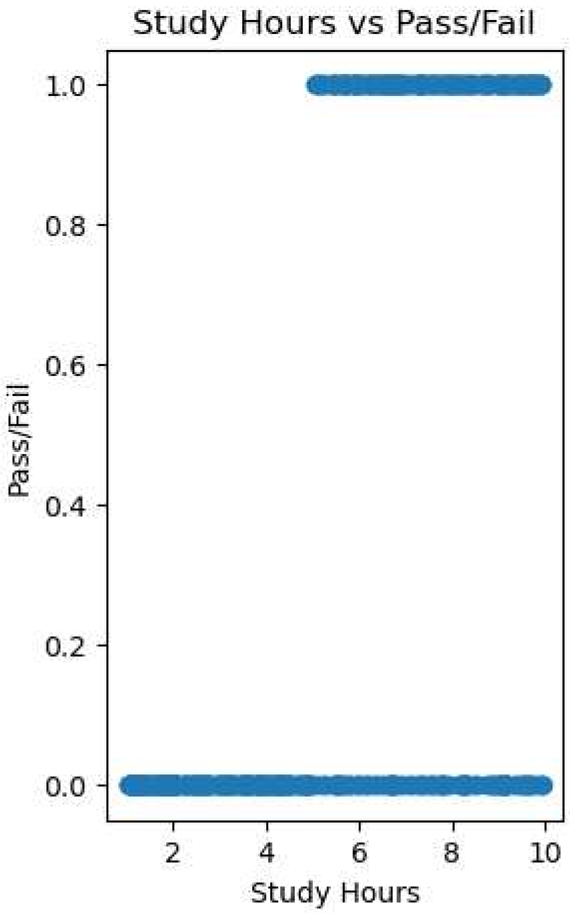
- Use the test data to evaluate the model's performance using metrics such as Mean Squared Error (MSE) and R^2 score. Visualize the Results:
- Plot the regression line along with the data points.

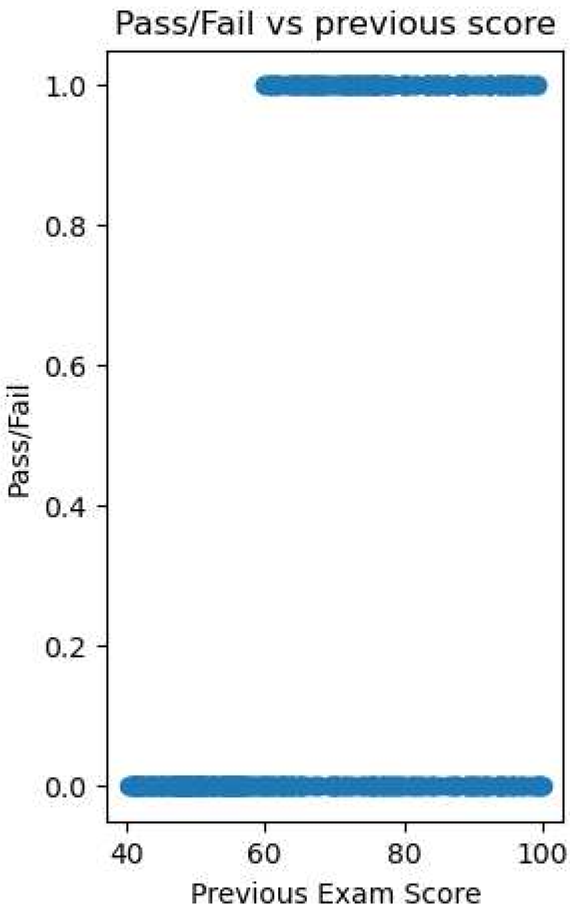
In [ ]:

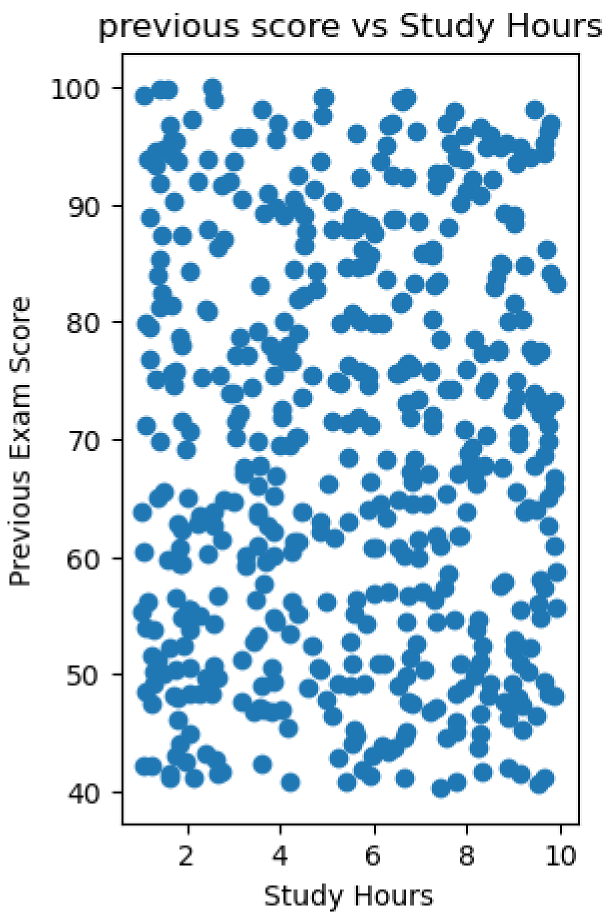
```
# Scatter plot for Pass/Fail vs previous score
plt.figure(figsize=(10, 5))
plt.subplot(1, 3, 1)
plt.scatter(df['Previous Exam Score'], df['Pass/Fail'])
plt.title('Pass/Fail vs previous score')
plt.xlabel('Previous Exam Score')
plt.ylabel('Pass/Fail')

# Calculat Pearson Correlation Coefficient to see how linear Pass/Fail vs pr
correlation_matrix = np.corrcoef(df['Previous Exam Score'], df['Pass/Fail']]
pearson_correlation = correlation_matrix[0, 1]
print("Pearson Correlation Coefficient for Pass/Fail vs previous score:", pe

# Scatter plot for previous score vs Study Hours
plt.figure(figsize=(10, 5))
plt.subplot(1, 3, 1)
plt.scatter(df['Study Hours'], df['Previous Exam Score'])
plt.title('previous score vs Study Hours')
plt.xlabel('Study Hours')
plt.ylabel('Previous Exam Score')

# Calculat Pearson Correlation Coefficient to see how linear previous score
correlation_matrix = np.corrcoef(df['Study Hours'], df['Previous Exam Score
pearson_correlation = correlation_matrix[0, 1]
print("Pearson Correlation Coefficient for previous score vs Study Hours:",
#the previous score vs Study Hours result on Pearson Correlation Coefficient
#which means there no relationship between the two columns
```

In [33]:
```
#import librairies
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler

#Dataset downloaded from Kaggle, prepared in CSV file

df = pd.read_csv("student_exam_data.csv",
                   delimiter=',',
                   na_values=['NA', 'n/a'],
                   encoding='utf-8' )


# Handle missing values by filling with a default value
df.fillna(0, inplace=True)

# Explore the data
print(df.head())
print(df.describe())


# Scatter plot for Pass/Fail vs hours

plt.figure(figsize=(10, 5))
plt.subplot(1, 3, 1)
plt.scatter(df['Study Hours'], df['Pass/Fail'])
plt.title('Study Hours vs Pass/Fail')
plt.xlabel('Study Hours')
plt.ylabel('Pass/Fail')

# Calculat Pearson Correlation Coefficient to see how linear Pass/Fail vs ho
correlation_matrix = np.corrcoef(df['Study Hours'], df['Pass/Fail'])
pearson_correlation = correlation_matrix[0, 1]
print("Pearson Correlation Coefficient for Pass/Fail vs hours:", pearson_cor
```

```
   Study Hours  Previous Exam Score  Pass/Fail
0     4.370861            81.889703          0
1     9.556429            72.165782          1
2     7.587945            58.571657          0
3     6.387926            88.827701          1
4     2.404168            81.083870          0
       Study Hours  Previous Exam Score   Pass/Fail
count   500.000000           500.000000  500.000000
mean      5.487055            68.917084    0.368000
std       2.688196            17.129607    0.482744
min       1.045554            40.277921    0.000000
25%       3.171517            53.745955    0.000000
50%       5.618474            68.309294    0.000000
75%       7.805124            83.580209    1.000000
max       9.936683            99.983060    1.000000
Pearson Correlation Coefficient for Pass/Fail vs hours: 0.5835049389186294
Pearson Correlation Coefficient for Pass/Fail vs previous score: 0.4437055
706139683
Pearson Correlation Coefficient for previous score vs Study Hours: 0.01035
420402828343
```




Pass/Fail vs previous score
1.0
0.8
0.6
0.4
0.2
0.0
Pass/Fail
40
60
80
100
Previous Exam Score


previous score vs Study Hours
100
90
80
70
60
50
40
Previous Exam Score
2
4
6
8
10
Study Hours

In [ ]:

In [34]:
```
#Prepare data for training

X = df[['Study Hours', 'Previous Exam Score']]
y = df['Pass/Fail']

#Split data into training and testing

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

#Train the linear regression model

model = LinearRegression()
model.fit(X_train, y_train)

#Make prediction

y_pred = model.predict(X_test)

#Evaluate the model
#use the R² score to evaluate the model's performance
#We need to see high R2 (how much the model explains the variance)
#we need to see low MSE (low MSE means predictions are near to mean,measures
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error with more features: {mse} /n")
print(f"R² Score with more features: {r2}/n")
# MSE : 0.12176568902308751 means prediction accuracy of my model is good
#R2 :0.4715030858372937 means
```

```
Mean Squared Error with more features: 0.12176568902308751 /n
R² Score with more features: 0.4715030858372937/n
```

- 1 - Interpretation of Mean Squared Error (MSE)

Mean Squared Error measures the average of the squares of the errors—that is, the average squared difference between the estimated values (predictions) and the actual value. A lower MSE indicates that the predictions are closer to the actual values.

Value: 0.12176568902308751 Interpretation: This is a relatively low MSE, suggesting that the model's predictions are close to the actual scores. The closer the MSE is to 0, the better the model's performance in terms of prediction accuracy.

- 2 - Interpretation of R^2 Score

R^2 Score (Coefficient of Determination) measures the proportion of the variance in the dependent variable that is predictable from the independent variables. It ranges from 0 to 1, where 0 indicates that the model explains none of the variability of the response data around its mean, and 1 indicates that the model explains all the variability.

Value: 0.4715030858372937 Interpretation: This indicates that approximately 47.15% of the variance in the scores can be explained by the features in the model. While this is a significant improvement over a low R^2, it also suggests that more than half of the variance is still unexplained by the model, implying that there might be other influential factors not captured by the current features.

Model improvements

In [24]:
```python
#Improve the model Using Ridge Regression

from sklearn.linear_model import Ridge

# Train the model with Ridge regression
model_ridge = Ridge(alpha=1.0)
model_ridge.fit(X_train, y_train)

# Make predictions and evaluate
y_pred_ridge = model_ridge.predict(X_test)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)

print(f"Mean Squared Error with Ridge regression: {mse_ridge}")
print(f"R² Score with Ridge regression: {r2_ridge}")
```

```
Mean Squared Error with Ridge regression: 0.12176699298219824
R² Score with Ridge regression: 0.4714974262925423
```

In [27]:
```python
#Improve the model Adding Polynomial Features

from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

# Create polynomial features
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(X)

# Train the model with polynomial features
model_poly = LinearRegression()
model_poly.fit(X_poly, y)

# Make predictions and evaluate
y_pred_poly = model_poly.predict(poly.transform(X_test))
mse_poly = mean_squared_error(y_test, y_pred_poly)
r2_poly = r2_score(y_test, y_pred_poly)

print(f"Mean Squared Error with polynomial features: {mse_poly}")
print(f"R² Score with polynomial features: {r2_poly}")
```

```
Mean Squared Error with polynomial features: 0.07239247842558766
R² Score with polynomial features: 0.6857965346111646
```

Interpretation of the Results

- 1 - Mean Squared Error (MSE) Value: 0.07239247842558766

Interpretation: This is a lower MSE compared to the previous model without polynomial features (which had an MSE of 0.12176568902308751 0.12176568902308751). A lower MSE indicates that the model's predictions are closer to the actual values, showing an improvement in prediction accuracy.

- 2 - R^2 Score Value: 0.6857965346111646

Interpretation: This R^2 score indicates that approximately 68.58% of the variance in the target variable (scores) can be explained by the model with polynomial features. This is a significant improvement from the previous R^2 score of 0.4715030858372937 0.4715030858372937. An R^2 score closer to 1 indicates a better fit of the model to the data.

Why Polynomial Features Improved the Model

Capturing Non-Linearity: The polynomial features allow the model to capture non-linear relationships between the independent variables and the target variable, which a simple linear model might miss. Higher-Order Interactions: Polynomial features can represent higher-order interactions between variables, providing a more flexible model that can fit the data better.

Cross-Validation Evaluation:

Validation: Use cross-validation to ensure that the model's performance is consistent across different subsets of the data.

Cross-validation is a powerful technique for evaluating the performance of a model by splitting the data into multiple subsets, training the model on some subsets, and validating it on others. This helps ensure that the model generalizes well to unseen data.

Using cross_val_score in Scikit-Learn

In [30]:
```
#Using cross validation to Evaluate my model

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.metrics import mean_squared_error, make_scorer

X = df.drop(columns=['Pass/Fail'])
y = df['Pass/Fail']

# Polynomial Features
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)

# Define the Model
model = Ridge(alpha=1.0)  # You can also use LinearRegression() or any other

# Define the Scoring Metric
mse_scorer = make_scorer(mean_squared_error, greater_is_better=False)

# Perform Cross-Validation
cv_scores = cross_val_score(model, X_poly, y, cv=5, scoring=mse_scorer)
mse_scores = -cv_scores  # Since we used neg_mean_squared_error, negate the

print("Cross-Validation MSE Scores:", mse_scores)
print("Mean MSE:", np.mean(mse_scores))
print("Standard Deviation of MSE:", np.std(mse_scores))

r2_scores = cross_val_score(model, X_poly, y, cv=5, scoring='r2')
print("Cross-Validation R² Scores:", r2_scores)
print("Mean R²:", np.mean(r2_scores))
print("Standard Deviation of R²:", np.std(r2_scores))
```

```
Cross-Validation MSE Scores: [0.06010635 0.06289143 0.06432938 0.08381233
0.07377929]
Mean MSE: 0.06898375592199321
Standard Deviation of MSE: 0.008723208106276603
Cross-Validation R² Scores: [0.75152398 0.68803855 0.73753823 0.63159415
0.68348653]
Mean R²: 0.6984362868756333
Standard Deviation of R²: 0.04277088730418352
```

Interpretaion

1 - Mean Squared Error (MSE) Scores: Cross-Validation MSE Scores: [0.06010635, 0.06289143, 0.06432938, 0.08381233, 0.07377929] Mean MSE: 0.06898375592199321 Standard Deviation of MSE: 0.008723208106276603

The MSE scores represent the mean squared error for each fold in your 5-fold cross-validation process. Mean MSE: The average MSE across all folds is 0.06898, which indicates that, on average, your model's predictions are about 0.06898 units away from the actual values. Lower values of MSE indicate better model performance. Standard Deviation of MSE: The standard deviation (0.00872) shows how much the MSE scores vary across different folds. A lower standard deviation suggests that the model's performance is consistent across folds.

2- R^2 (R-squared) Scores: Cross-Validation R^2 Scores: [0.75152398, 0.68803855, 0.73753823, 0.63159415, 0.68348653] Mean R^2: 0.6984362868756333 Standard Deviation of R^2: 0.04277088730418352

The R^2 scores represent the coefficient of determination for each fold in your cross-validation. Mean R^2: The average R^2 across all folds is 0.6984, indicating that approximately 69.84% of the variance in the target variable (scores) is explained by your model with polynomial features. Higher values of R^2 indicate that the model fits the data well. Standard Deviation of R^2: The standard deviation (0.04277) shows how much the R^2 scores vary across different folds. A lower standard deviation suggests that the model's explanatory power is consistent across folds.

Summary and Implications:

Model Performance: The results suggest that my model, with polynomial features and Ridge regression, performs reasonably well.

MSE: The low average MSE indicates that your model's predictions are close to the actual values on average.

R^2: The average R^2 of 0.6984 suggests that your model explains a substantial portion of the variance in the scores, demonstrating good predictive capability.

Consistency: The low standard deviations for both MSE and R^2 indicate that the model's performance is stable across different subsets of data.