a) Which model gives you better performance: Multiple Linear Regression or Polynomial Regression?
Provide evidence from your analysis (such as R-squared, Mean Squared Error, or other performance metrics) and explain your reasoning. Be sure to compare the results from both models and justify why one might perform better than the other based on the dataset and problem you're solving.

Answer:
Multiple Linear Regression gives better performance than Polynomial Regression. I compared R-squared (R^2), Mean Squared Error (MSE), and Mean Absolute Error (MAE).
Linear Regression achieved a Test R^2 score of about 0.41, 41% of the variation in house prices.
Polynomial Regression with degree 2 gave a lower Test R^2 score of 0.25.
Polynomial Regression with degree 7/8 is overfitting.
Even though it has $R^2 = 1.0$ on the training data, its Test $R^2 = -2.27$ shows it performs very poorly
Even though Polynomial Regression slightly reduced MAE, it failed to improve R^2.
Therefore, Multiple Linear Regression is the better model in terms of both performance and generalization.

b) How do you decide the optimal degree for Polynomial Regression in this case?
Explain how you determined the degree of the polynomial and what criteria you used to decide whether a higher degree improves the model's performance. Discuss the potential risks of choosing too high or too low a degree for the polynomial.

Answer:
I tried polynomial degrees from 2 to 8 and used train-test split to test model performance on new data. I compared the Test R^2 scores for each degree. Degree 2 gave the best performance ($R^2 = 0.25$). Higher degrees (like 8) overfitted the training data and didn't generalize well. So I chose degree 2 as the optimal degree for Polynomial Regression in this case.
The potential risks of choosing too high or too low a degree for the polynomial are:

- A too low degree may underfit the data — the model is too simple and cannot capture important patterns or curves in the relationship.
- A too high degree may overfit — the model becomes too complex, learns noise in the training data, and performs badly on new data.

To avoid these risks, I think it is important to use validation techniques, such as train-test split or cross-validation, and compare performance on unseen data.