CS 459 – Introduction to Machine Learning

Assignment No. 01

House Price Prediction In Sector F-7

Muhammad Nasir Khurshid

(04071913019)

1. Introduction

Predicting house prices is important for everyone involved in buying or selling a house. It helps buyers make better decisions, sellers set the right prices, and real estate agents provide better services. Machine learning techniques have made it easier to predict house prices using factors like location, size, and amenities.

For this assignment, I will predict house prices in Sector F-7, Islamabad, Pakistan, using a dataset scraped from zameen.com. I will apply three regression models: multivariate regression, polynomial regression, and normal equation, and identify the best one for predicting house prices.

2. Data Collection

To collect data for this project, I scraped information from zameen.com, a popular real estate website in Pakistan. I used Python ***requests*** library and ***BeautifulSoup*** to extract information from the website. The data I collected includes property details like price, location, number of baths, area, and number of bedrooms. I saved the data in an Excel workbook using the Pyhthon's ***openpyxl*** library. The data was scraped from 14 pages of the website, with each page containing multiple property listings.

3. Data Preprocessing

Preparing data for analysis is a crucial step in machine learning projects. It involves cleaning and transforming raw data into a usable format. To prepare the dataset of house listings in Sector F-7, Islamabad, Pakistan for analysis, I performed the following steps:

- **Load data:** The dataset was loaded into a pandas dataframe using the ***read_excel*** function.
- **Drop specified columns:** Columns that were not needed for analysis, such as 'Location', and 'House No.', were dropped from the dataframe using the ***drop*** function.
- **Remove rows with zero values:** All the rows which contained NULL values were dropped from the dataframe using ***dropna()*** function.
- **Split columns:** The 'Price' and 'Area' columns were split on the space character to separate their values and units.
- **Operation on columns:** The 'Price' column was converted to a numeric data type. The 'Arab' values in the 'Price' column were converted to 'Crore'. The 'Area' column was also converted to a numeric data type.
- **Drop columns:** The 'Price_Unit' and 'Area_Unit' columns were dropped from the dataframe using the ***drop*** function.

4. Modelling

To perfrom modelling, three Machine Learning techniques were used which are explained below:

a. Multivariate Regression

I utilized the SGDRegressor from the sklearn library in Python for this task. To optimize the model's performance, I experimented with various learning rates i.e.

[0.0001, 0.001, 0.01, 0.1, 0.5]. Additionally, I evaluated the model's performance with and without scaling the features. To scale the features, I employed the StandardScaler from the sklearn library.
The SGDRegressor utilizes Stochastic Gradient Descent, which involves estimating the gradient of the loss for each sample and updating the model incrementally with a decreasing learning rate.
The StandardScaler calculates the standard score of a sample by subtracting the mean 'u' of the training samples and dividing it by the standard deviation 's' of the training samples. The standard score of a sample 'x' is calculated as:

$$z = \frac{(x - u)}{s}$$

b. Polynomial Regression

I experimented with Polynomial features from degree 1 to 4 and trained SGDRegressor on each degree with and without scaling. For testing various degrees I used Python's PolynomialFeatures function. The PolynomialFeatures function creates a new feature matrix by computing all possible polynomial combinations of the original features up to a specified degree. As an example, if we have a three-dimensional input sample with the form [a, b, c], then the degree-2 polynomial features would be [1, a, b, c, a^2, ab, ac, b^2, bc, c^2]. This means that we not only include the original features [a, b, c], but also all possible combinations of these features up to degree 2.

c. Normal Equation

I implemented the Normal Equation using numpy library in Python, which provides various functions for matrix operations such as inverse, transpose, and dot product.

5. Results & Analysis

To check how well different models perform on the dataset, I used two metrics called R^2 Score and Mean Squared Error. Both metrics tell us how well the regression line fits the observed data points.
The R^2 score ranges from 0 to 1, where 1 represents a perfect fit and 0 indicates that the model does not explain any of the variance in the dependent variable. In simple terms, the closer the R^2 score is to 1, the better the model fits the data.
To calculate R^2 score in Python, I used the ***r2_score*** function from the sklearn library.

$$R^2 = 1 - (\frac{SS_{res}}{SS_{tot}})$$

where SS_res is the sum of squared residuals and SS_tot is the total sum of squares.

Mean Squared Error calculates the average squared difference between the predicted values and the true values of the dependent variable. A lower MSE value indicates a better fit of the regression line to the observed data points. To calculate Mean Squared Error in Python, I used the ***mean_squared_error*** function from the sklearn library.

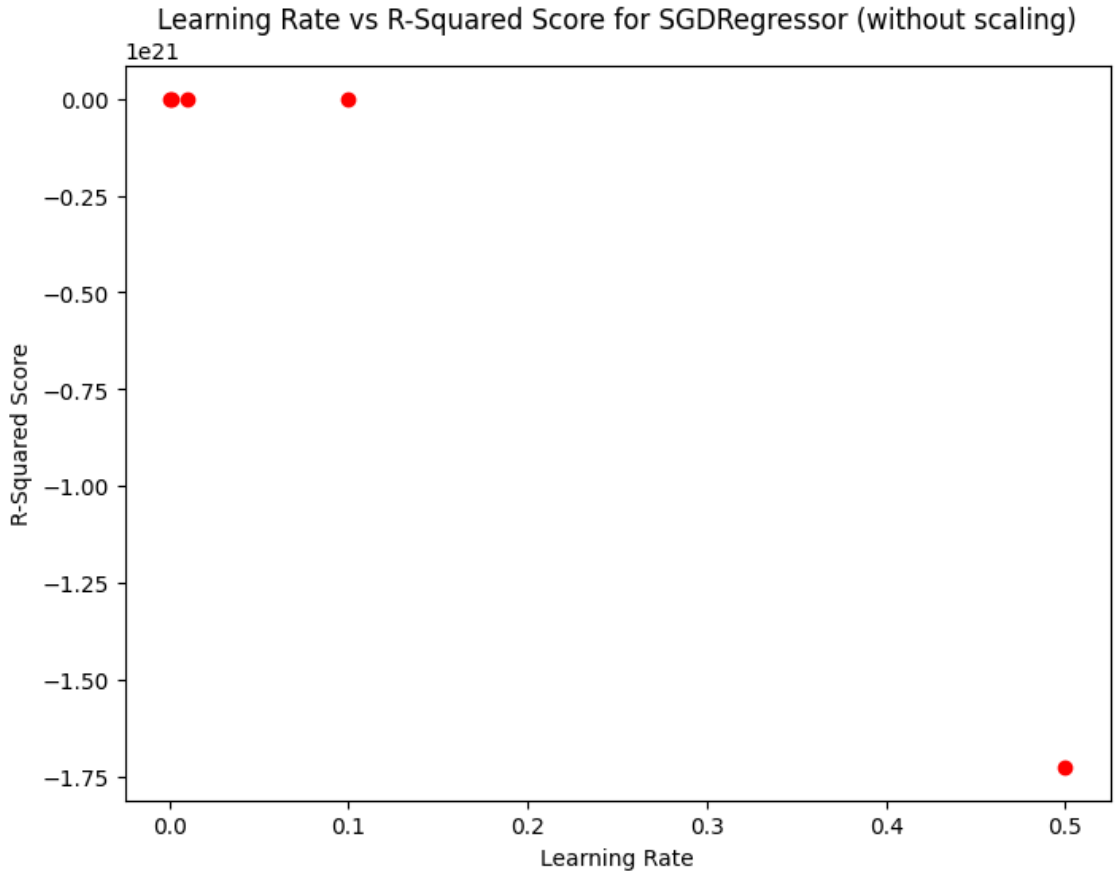
$$MSE = \frac{1}{n} * \sum(y_{true} - y_{pred})^2$$

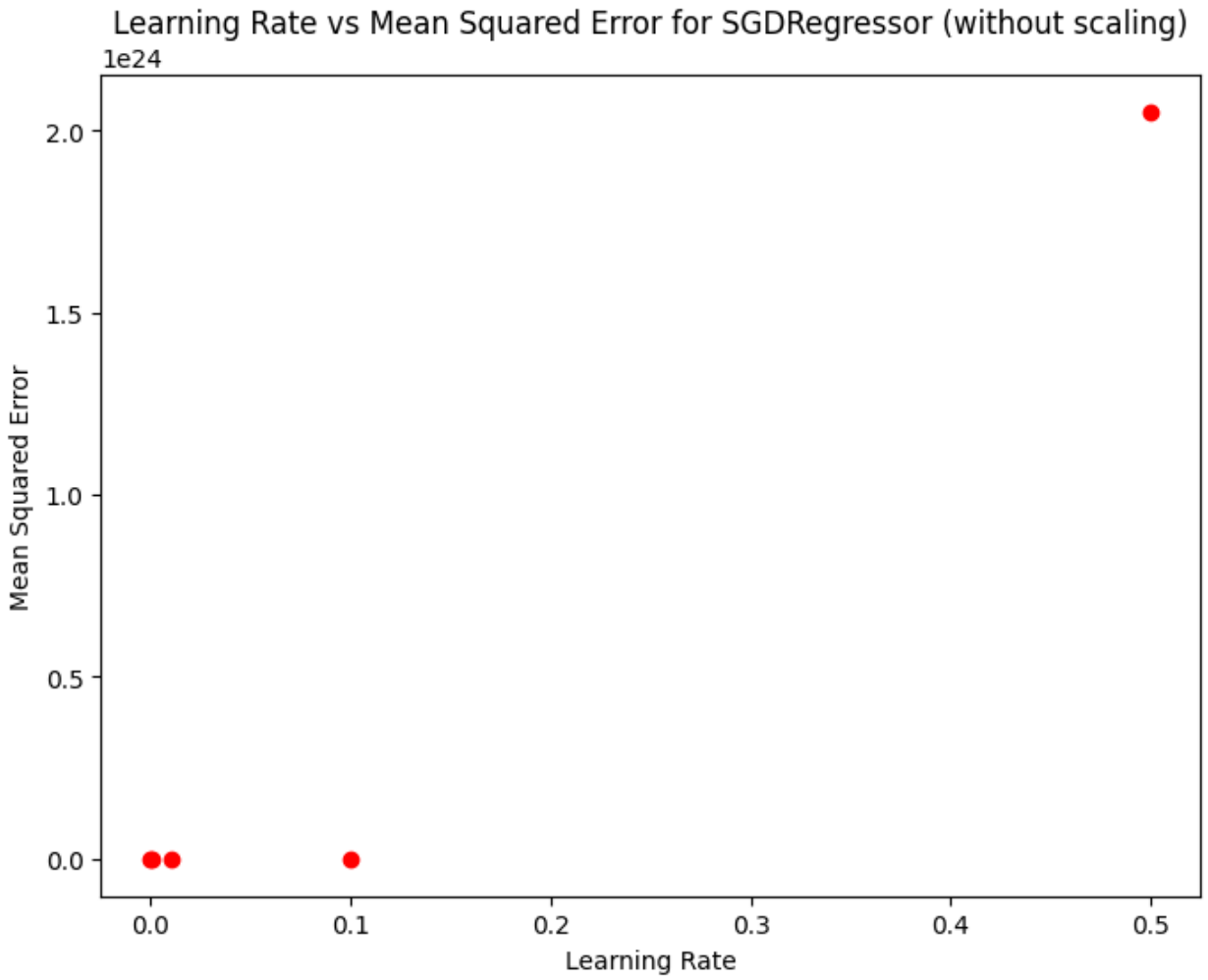
where n is the number of data points, y_true is the array of true values, and y_pred is the array of predicted values.

Evaluation of models is given below:

SGDRegressor (without scaling):

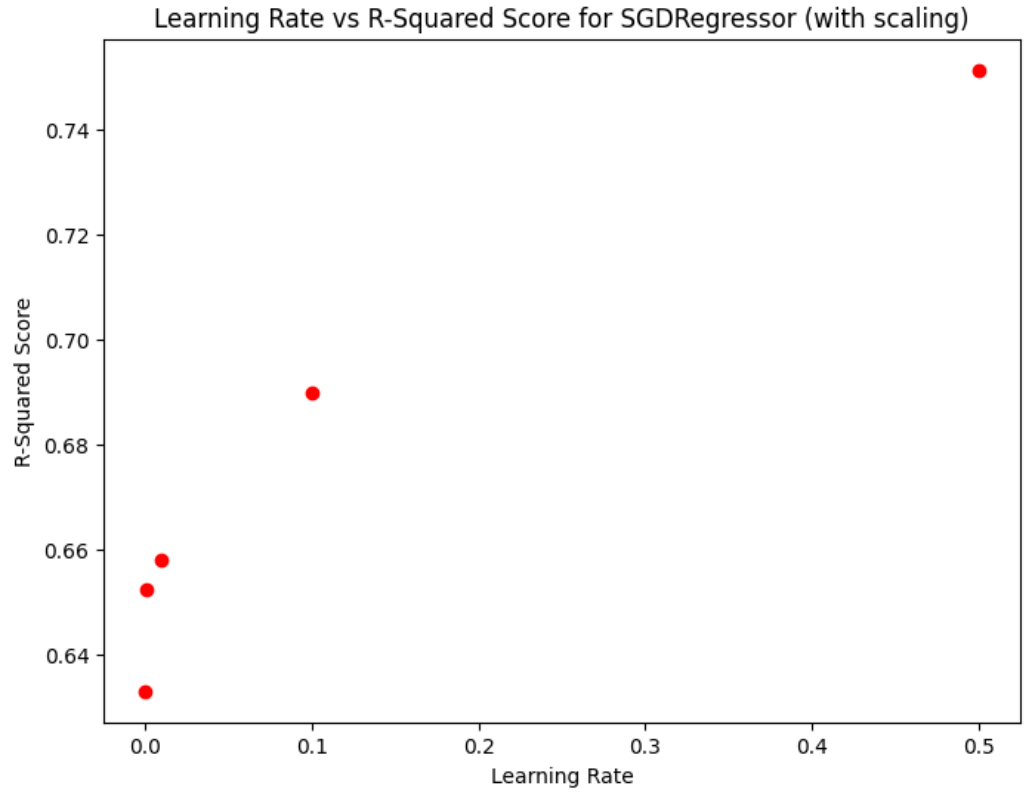
Learning Rate	R^2 Score	Mean Squared Error
0.0001	0.6065989441477064	467.80520833796936
0.001	0.6131182973564517	460.0528463636514
0.01	0. 5974906804894354	478.6361227823231
0.1	0. 67811101083814	382.76802615674626
0.5	-1.7257409004999134e+21	2.0521316987313e+24

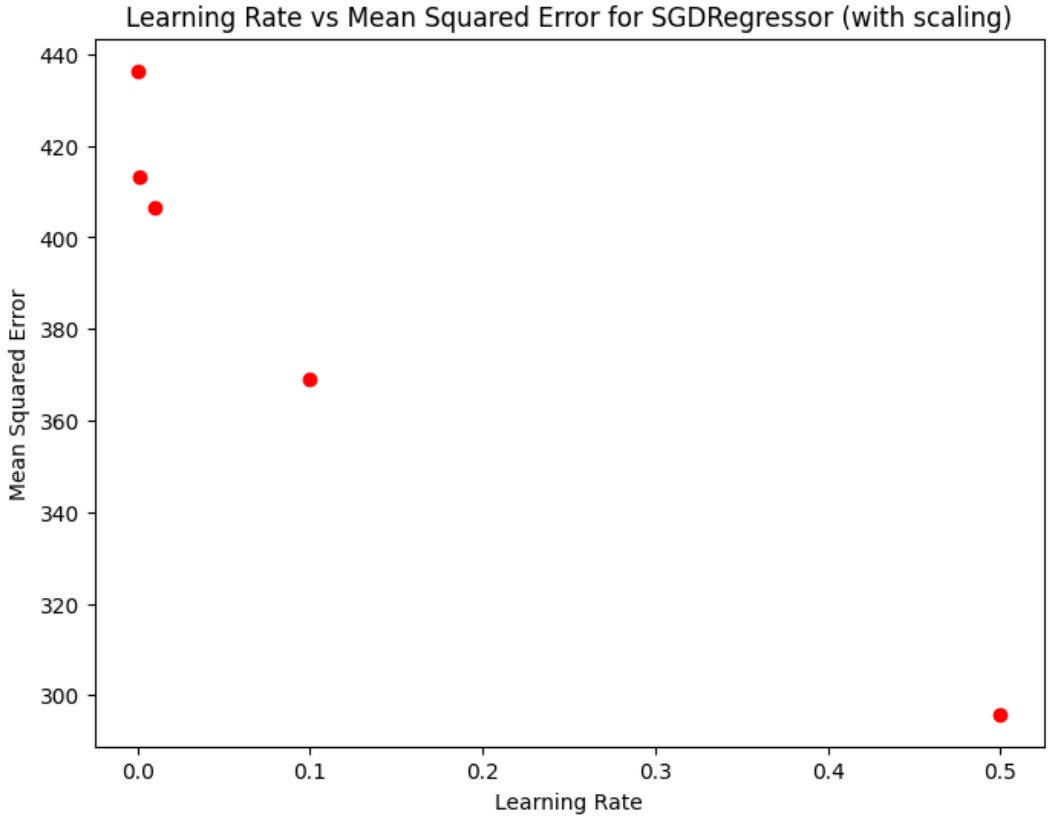




SGDRegressor (with scaling):

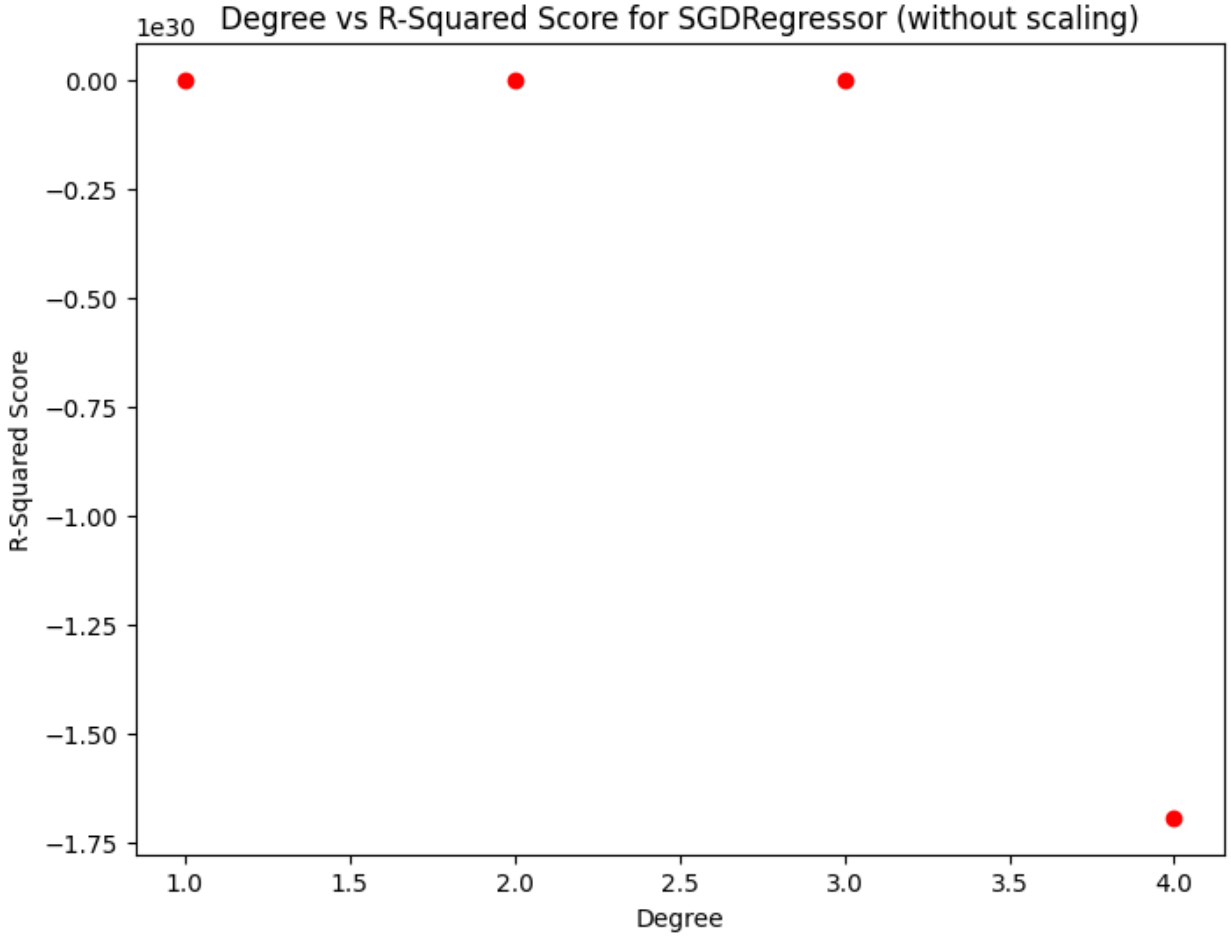
Learning Rate	R^2 Score	Mean Squared Error
0.0001	0.6330386750400825	436.3649169749696
0.001	0.6524324276013911	413.30321360017155
0.01	0.6580419696059836	406.6327359105431
0.1	0.6897373192715666	368.9428277796919
0.5	0.751217502159921	295.834864959323

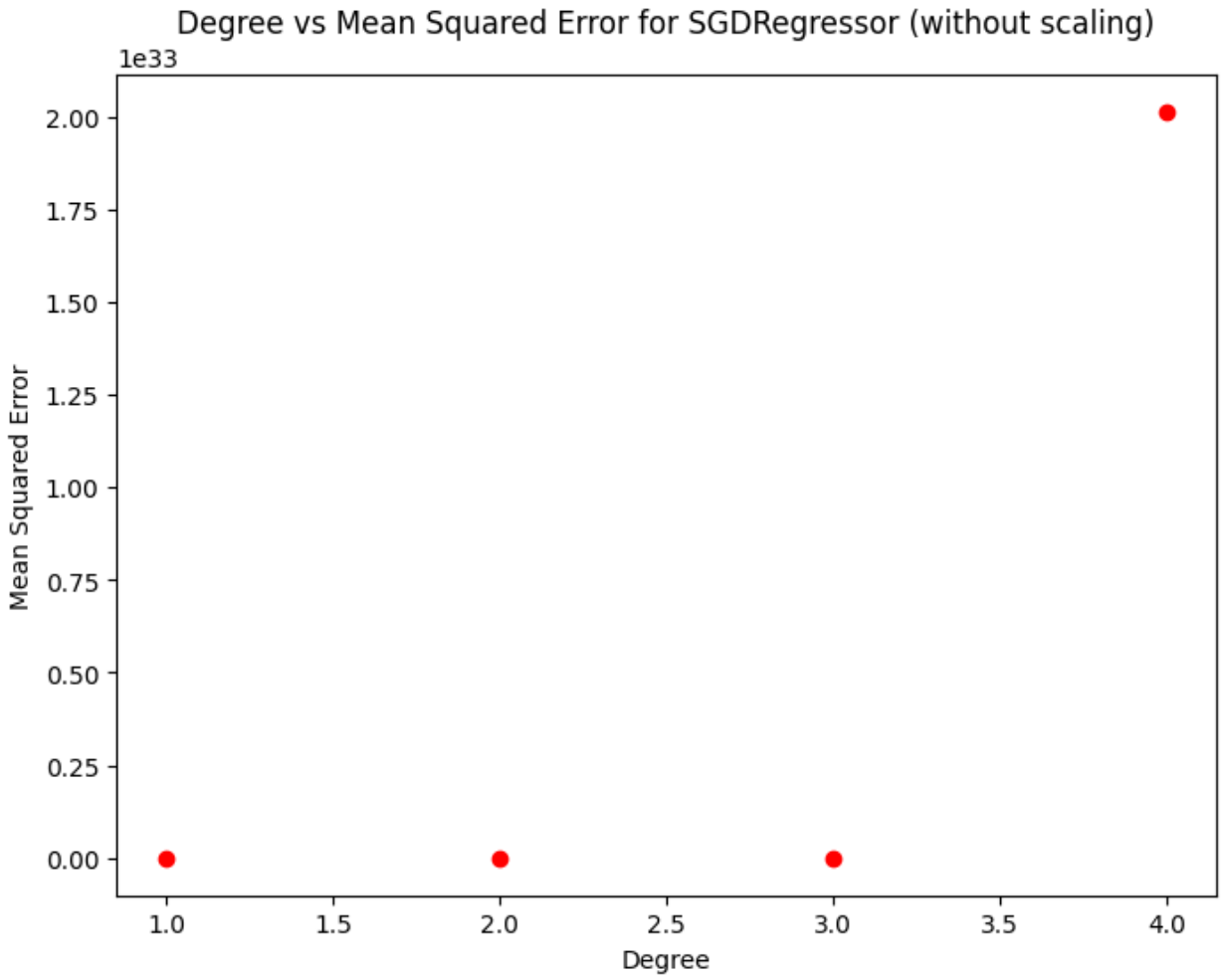

Learning Rate vs R-Squared Score for SGDRegressor (with scaling)
R-Squared Score
0.74
0.72
0.70
0.68
0.66
0.64
0.0
0.1
0.2
0.3
0.4
0.5
Learning Rate


Learning Rate vs Mean Squared Error for SGDRegressor (with scaling)
Mean Squared Error
440
420
400
380
360
340
320
300
0.0
0.1
0.2
0.3
0.4
0.5
Learning Rate

Polynomial Regression (without scaling):

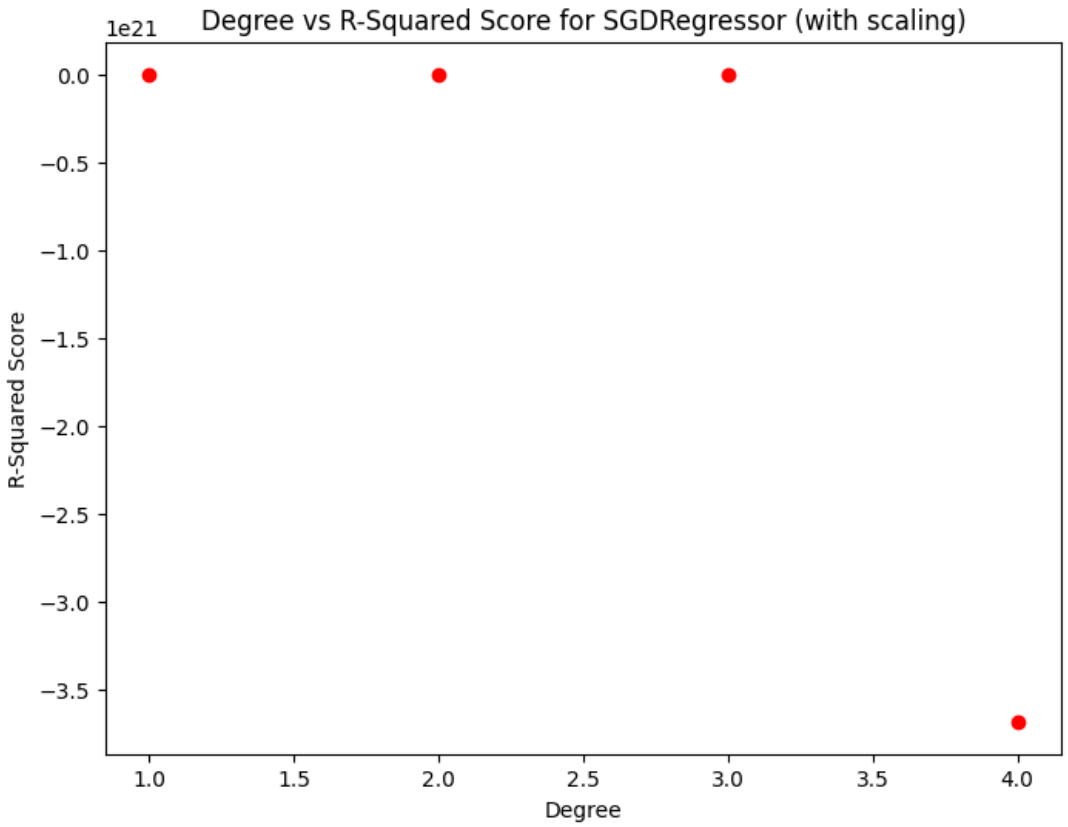
Learning Rate	R^2 Score	Mean Squared Error
1	0.597146138311038	479.0458284075328
2	-7.683874063604497e+21	9.137129282430889e+24
3	-2.752165843642945e+25	3.272684444317881e+28
4	-1.6936517103011005e+30	2.0139729657635724e+33

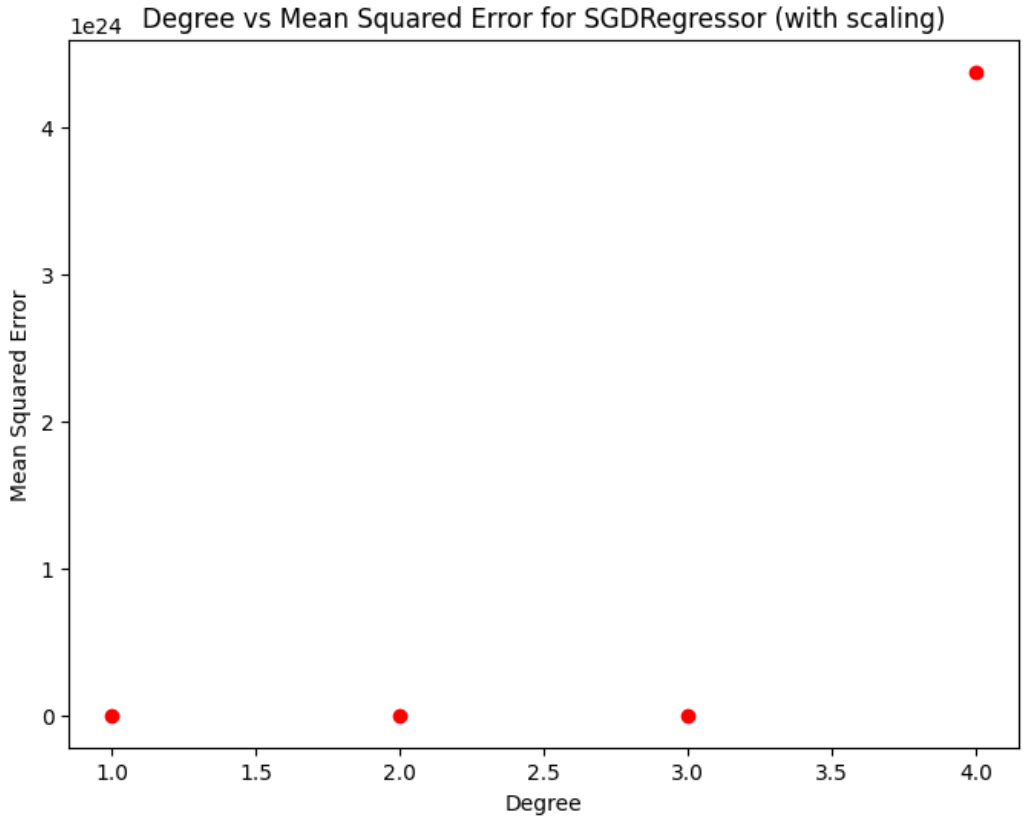




Polynomial Regression (with scaling):

Learning Rate	R^2 Score	Mean Squared Error
1	0.6579324951420813	406.7629153384618
2	0.5680675033293179	513.6241212042737
3	-9541333515424620.0	1.1345891035638516e+19
4	-3.68078908636878e+21	4.376938698515779e+24


Degree vs R-Squared Score for SGDRegressor (with scaling)
1e21
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
−3.5
R-Squared Score
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Degree


Degree vs Mean Squared Error for SGDRegressor (with scaling)
1e24
4
3
2
1
0
Mean Squared Error
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Degree

Normal Equation:

R^2 score: 0.6578367081334215

Mean Squared Error: 406.8768185953941

Conclusion

The findings indicate that scaling makes some models perform better, like SGDRegressor. But, Polynomial Regression didn't show much difference, whether scaling was applied or not. The most effective model was SGDRegressor with scaling and alpha=0.5, obtaining an R^2 score of 0.751.