Assignment 1: Artificial Neural Networks

Student Name : Tazmeen Afroz
Roll No : 22P-9252

April 29, 2025

Task 0

1. Identify which columns are most suitable to provide a linear/multilinear/polynomial relationship to price?

Most Suitable Columns for Linear/Polynomial Relationship with Price:

Carat, Depth, X (length), Y (width), Z (depth), and Table are more suitable for establishing a regression relationship with price.

- **Carat:** This feature shows a strong and direct relationship with price. It increases sharply, almost exponentially, as carat increases. While a linear model might work to some extent, a polynomial or exponential model would likely perform better. The choice depends on the desired accuracy and how well the output fits different degrees of polynomial equations.
- **Depth:** The relationship seems to be parabolic in nature, so a polynomial regression may yield better results.
- **X, Y, Z (dimensions):** These behave similarly to carat. As the values increase, the price increases rapidly. The steep change suggests a polynomial or exponential relationship, making them suitable for such regression models.
- **Table:** The relationship is not as direct or consistent. There might be some linear pattern, but it's not very strong. So, while linear regression may be applied, its effectiveness would be limited.

2. Explain why some plots are appearing as vertical or horizontal lines?

These are categorical features represented by discrete numerical values. For example, Cut has only five distinct classes. Since they represent categories, applying a linear or polynomial model directly to their encoded values may lead to misleading results.

Task 1

1. How many iterations would be needed for Learning Rate 0.01 to make its linear regression line similar to Learning Rate 0.1?

100 iterations would be needed.

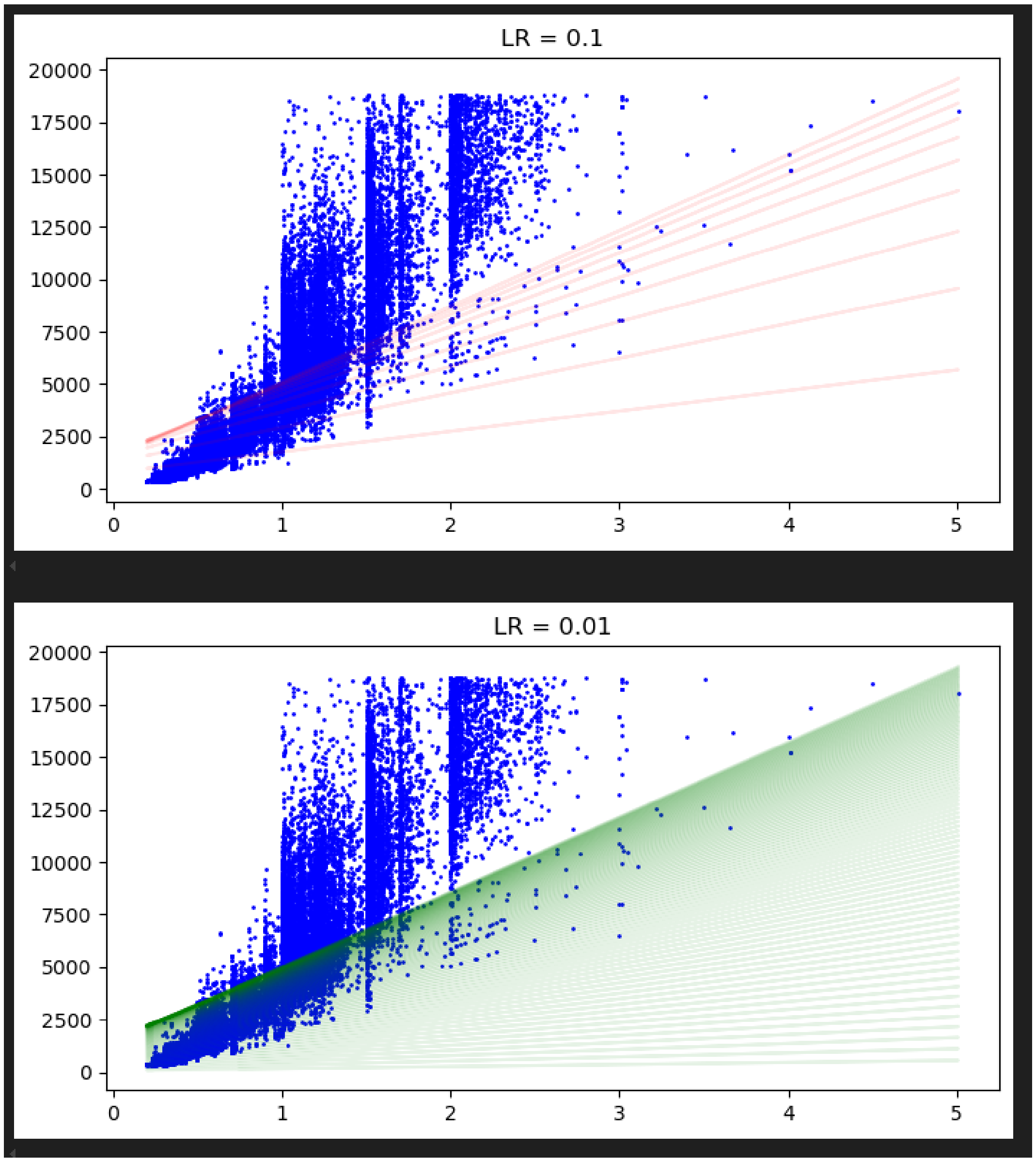


Figure 1: Comparison of Learning Rates 0.1 and 0.01

Modify your code so that the cost function for Learning Rate 0.1 and 0.01 is determined. Show your code change.

```
cost_0_1 = []
b_0_1 = 0
m_0_1 = 0

# For LR = 0.1
for i in range(10):
    m_gradient = 0
    b_gradient = 0
    N = float(len(points))
    total_cost = 0

    for j in range(len(points)):
        x_j = points[j, 0]
        y_j = points[j, 1]
        m_gradient += (2/N) * x_j * (y_j - (m_0_1 * x_j + b_0_1))
        b_gradient += (2/N) * (y_j - (m_0_1 * x_j + b_0_1))
        total_cost += (y_j - (m_0_1 * x_j + b_0_1)) ** 2

    cost_0_1.append(total_cost/N)

    m_0_1 = m_0_1 + 0.1 * m_gradient
    b_0_1 = b_0_1 + 0.1 * b_gradient
```

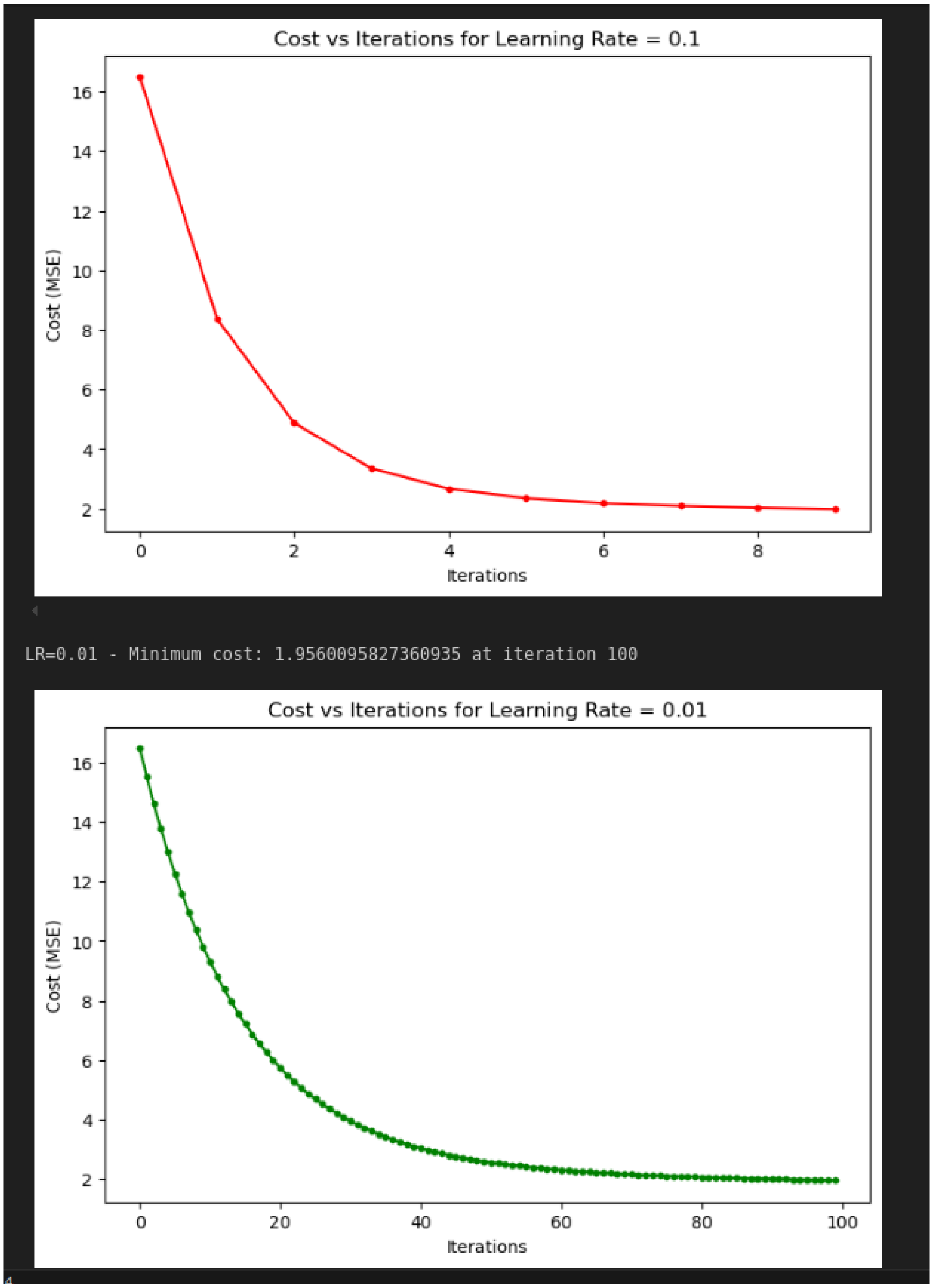
Listing 1: Cost Function Implementation

```
plt.figure(figsize=(8, 5))
plt.plot(range(10), cost_0_1, 'r-', label='LR=0.1')
plt.scatter(range(10), cost_0_1, color='r', s=10, label='Points')
plt.xlabel('Iterations')
plt.ylabel('Cost (MSE)')
plt.title('Cost vs Iterations for Learning Rate = 0.1')
plt.show()
```

Listing 2: Plotting the Cost Function

What is the minimum error in both cases?

- LR=0.1 - Minimum cost: 1.9798318842272402 at iteration 10
- LR=0.01 - Minimum cost: 1.9560095827360935 at iteration 100

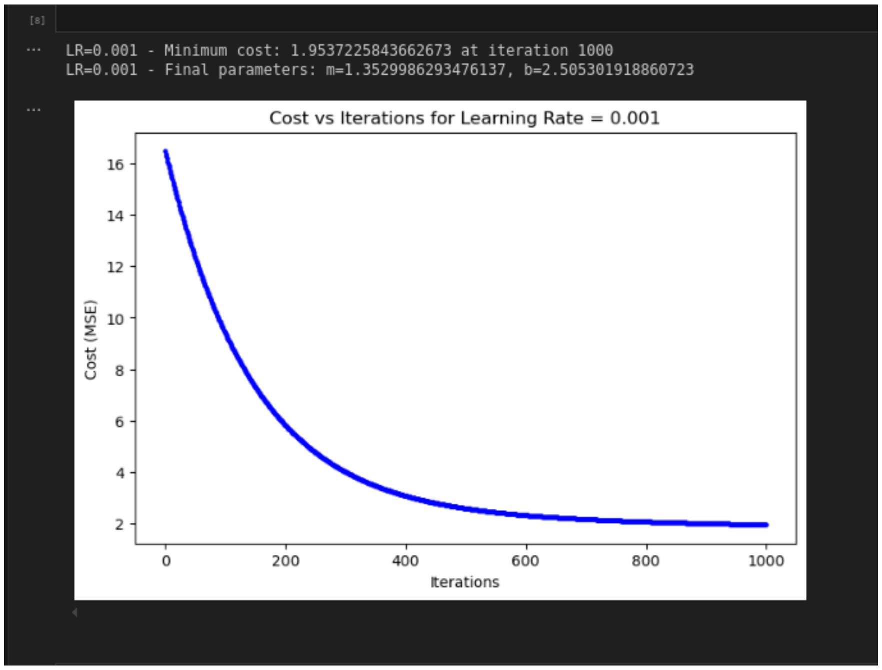


```
LR=0.01 - Minimum cost: 1.9560095827360935 at iteration 100
```

Figure 2: Cost vs Iterations for Different Learning Rates

Modify your code so that Learning Rate of 0.001 is used, and it gives exactly the same output as Learning Rate 0.1. Comment on the nature of execution.

1000 iterations are needed as the step size is very low. It finds weights more precisely but takes more iterations, resulting in slow convergence. I would not be willing to try a learning rate of 0.0001 due to the extremely slow convergence.

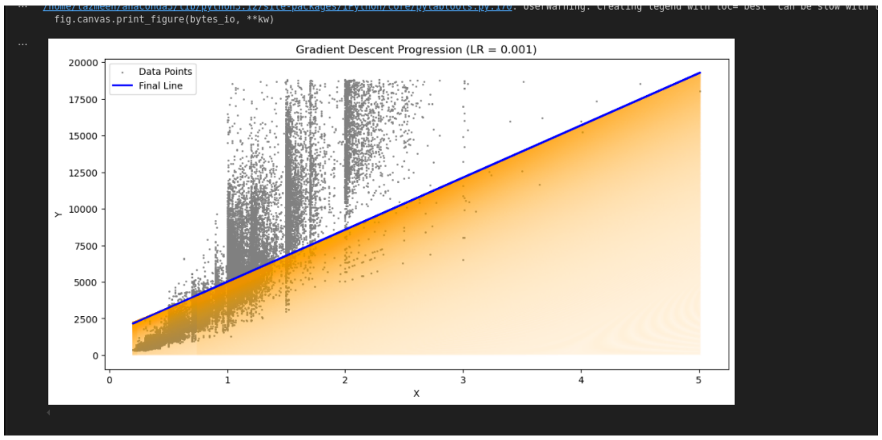


```
LR=0.001 - Minimum cost: 1.9537225843662673 at iteration 1000
LR=0.001 - Final parameters: m=1.3529986293476137, b=2.505301918860723
```

Figure 3: Learning Rate 0.001 with 1000 Iterations

What is the behavior of the linear regression line? Is it smoothly progressing in one direction or is it oscillating?

It is smoothly progressing in one direction.



```
/home/tazmeen/anaconda3/lib/python3.12/site-packages/IPython/core/pylabtools.py:170: UserWarning: Creating legend with loc="best" can be slow with
  fig.canvas.print_figure(bytes_io, **kw)
```

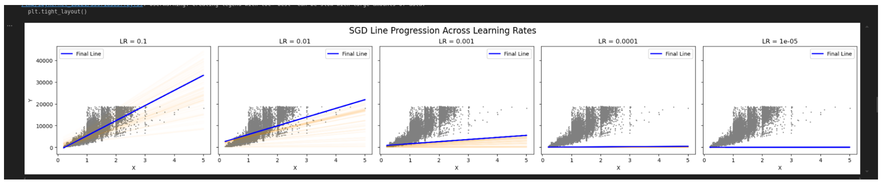
Figure 4: Behavior of Linear Regression Line

Task 2: Converting to Stochastic Gradient Descent

1. What is the behavior of the linear regression line? Is it smoothly progressing in one direction or is it oscillating?

It is oscillating.

2. Determine the stochastic gradient solution for learning rates 0.1, 0.01, 0.001, 0.0001, 0.00001, and show the output graphs.



```
plt.tight_layout()
```

Figure 5: Stochastic Gradient Descent Solutions for Different Learning Rates

3. Compute the total cost and plot the cost graph for all learning rates.

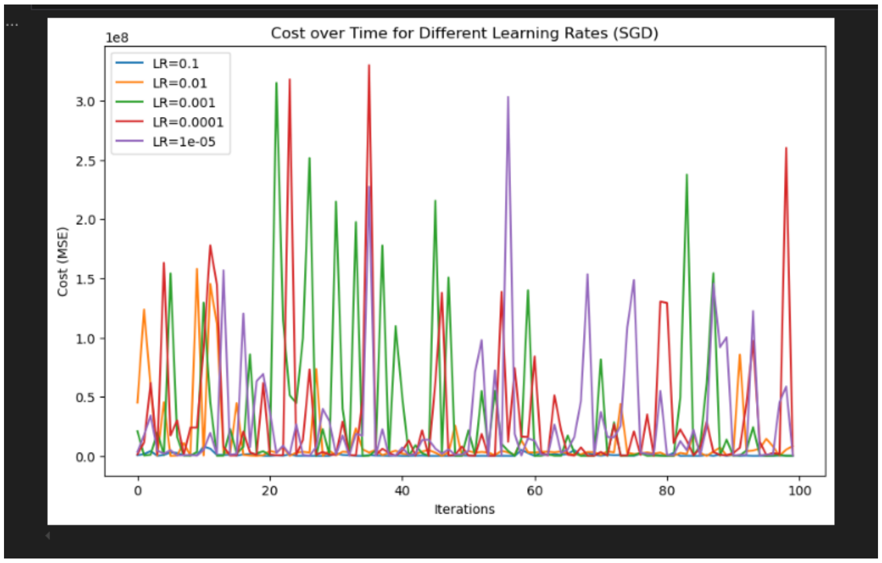


Figure 6: Cost vs Iterations for Different Learning Rates in SGD

4. From 3 above, explain why these costs are not smoothly progressing and why are they oscillating?

The costs are oscillating because at each iteration, random points are selected instead of using all data points as in batch gradient descent. This randomness in point selection causes fluctuations in the cost function, making the convergence path noisy rather than smooth.

Task 3: Fitting to Polynomial Regression

1. Add the cost/error calculation and show the convergence of the error + the fitted polynomial line.

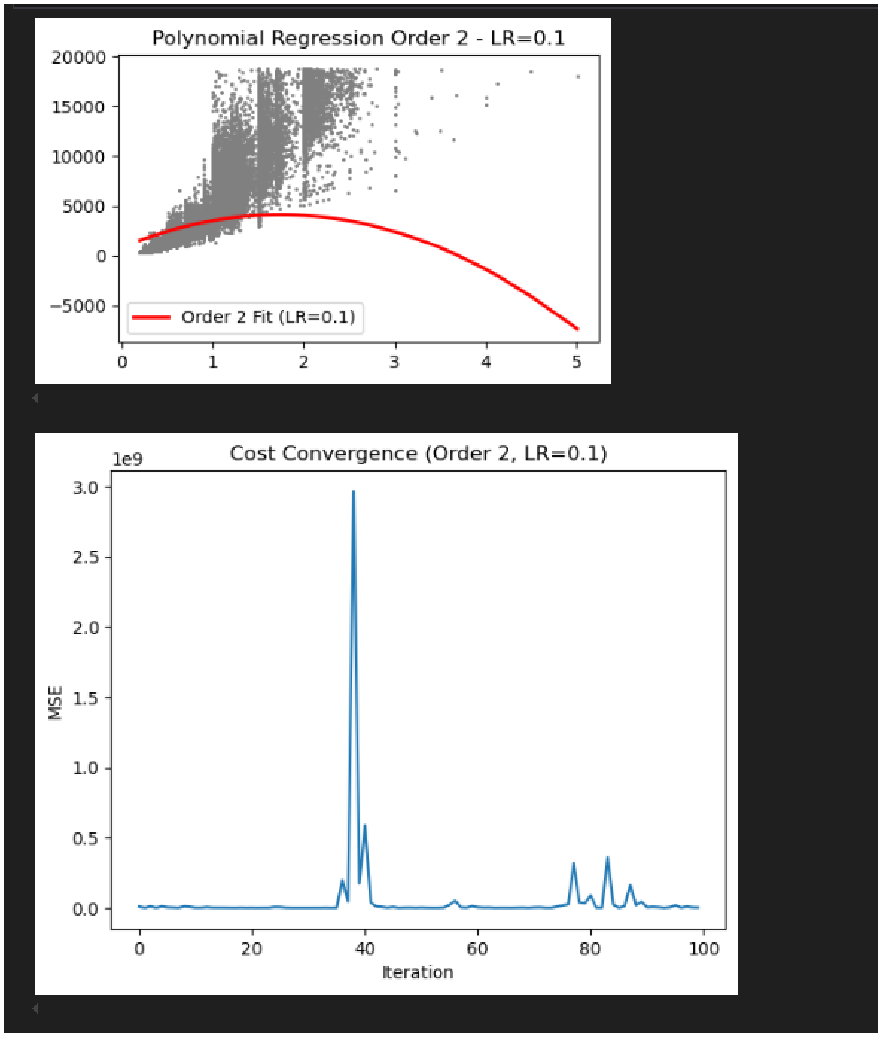


Figure 7: Error Convergence for Polynomial Regression

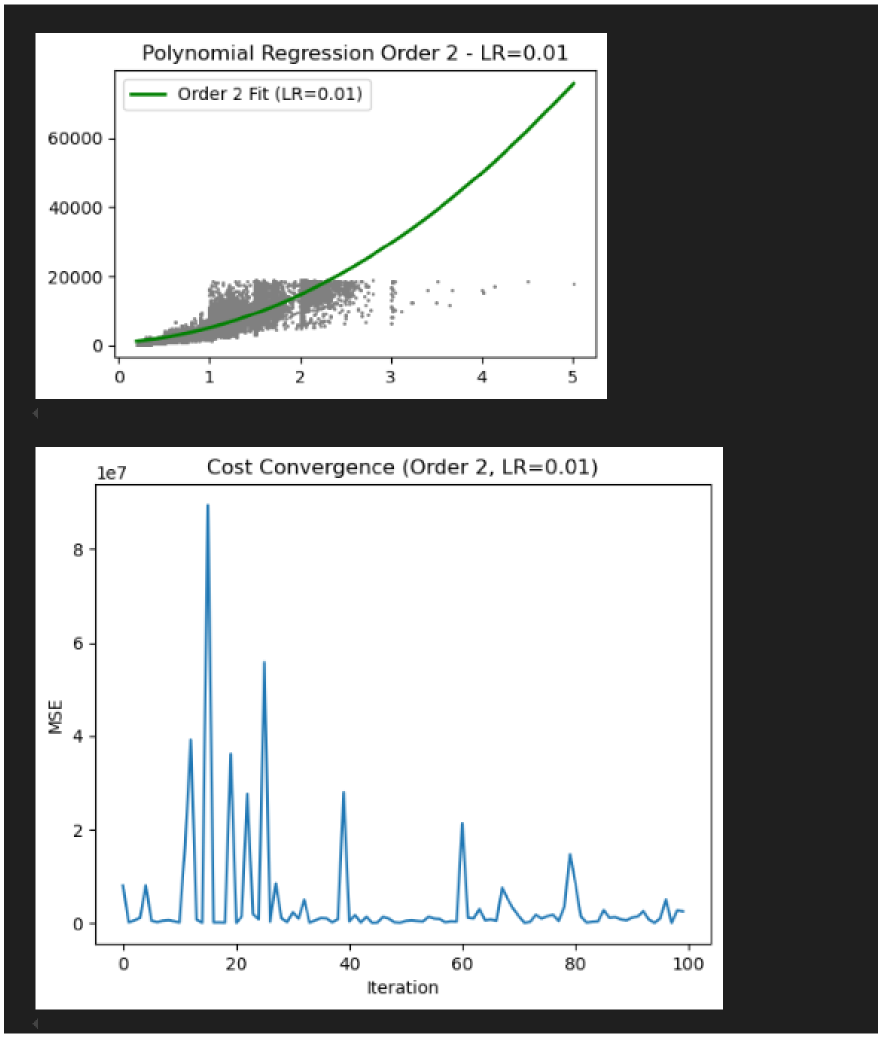


Figure 8: Fitted Polynomial Line - Order 2

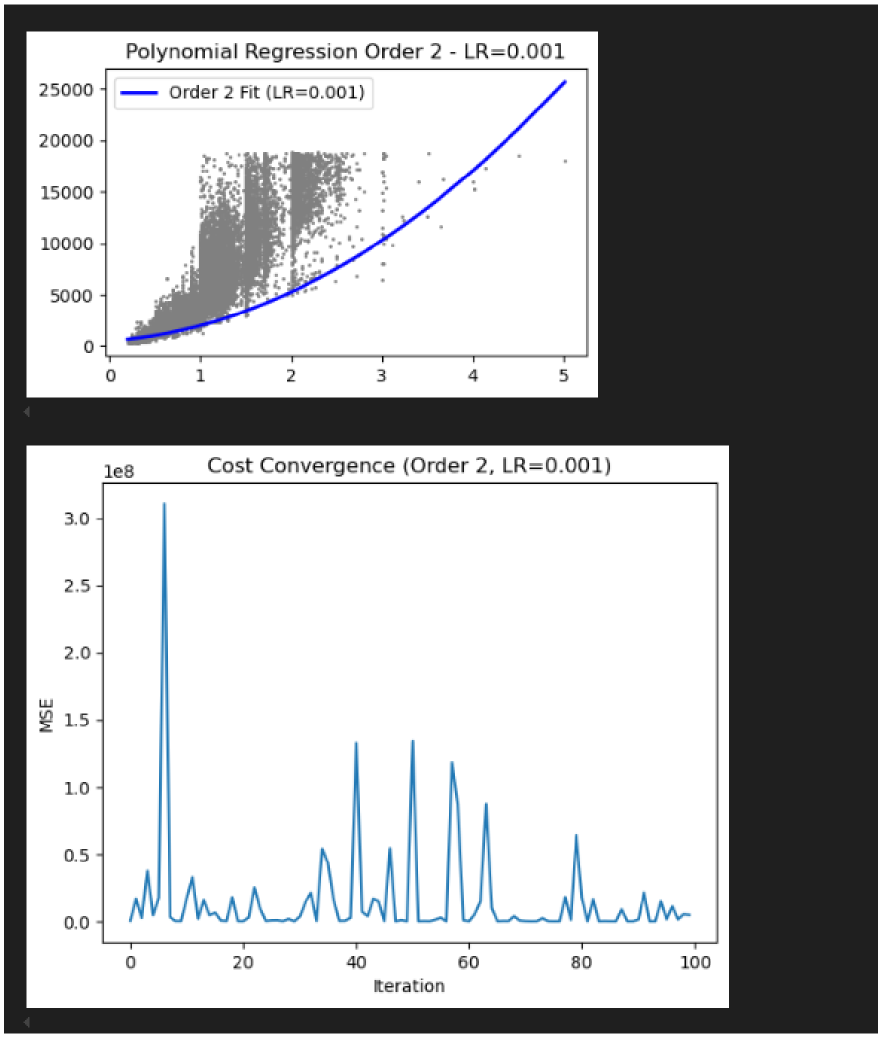


Figure 9: Error vs Iterations for Polynomial Regression

2. Convert the code into an order 3 polynomial and show the convergence of the error + the fitted polynomial line.

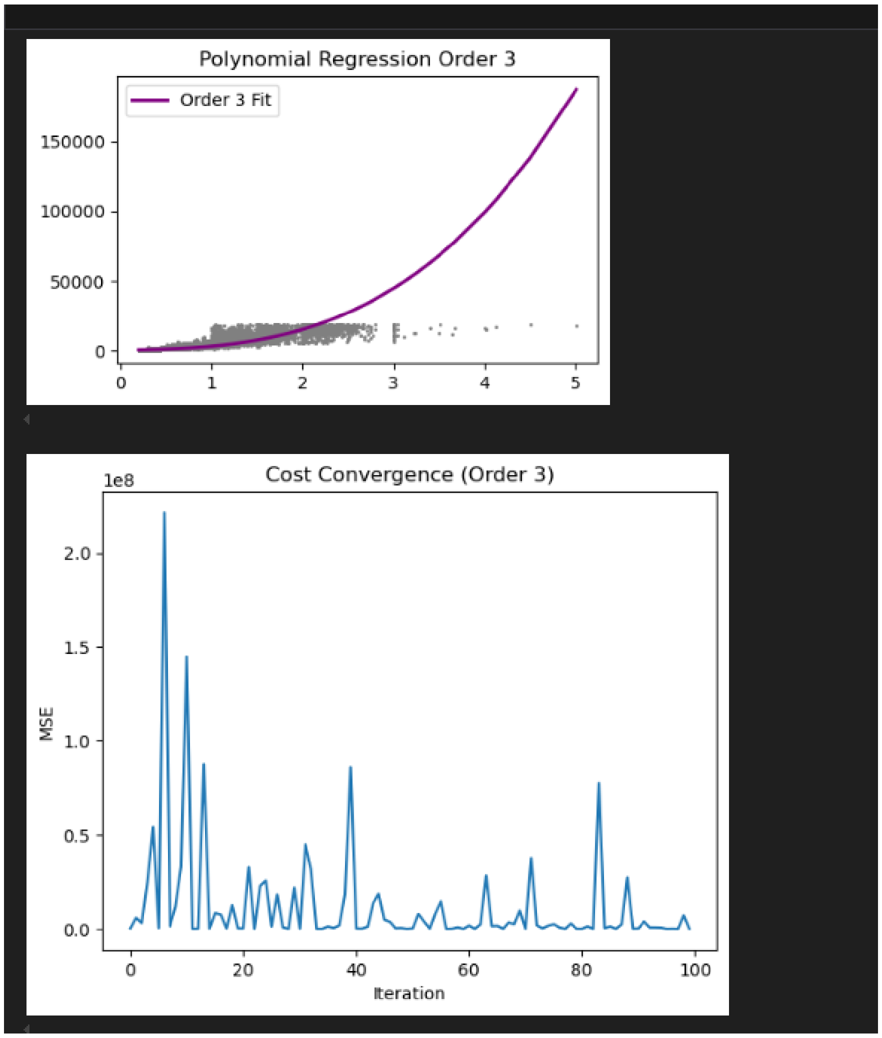


Figure 10: Order 3 Polynomial Regression Results

3. Convert the code into an order 4 polynomial and show the convergence of the error + the fitted polynomial line.

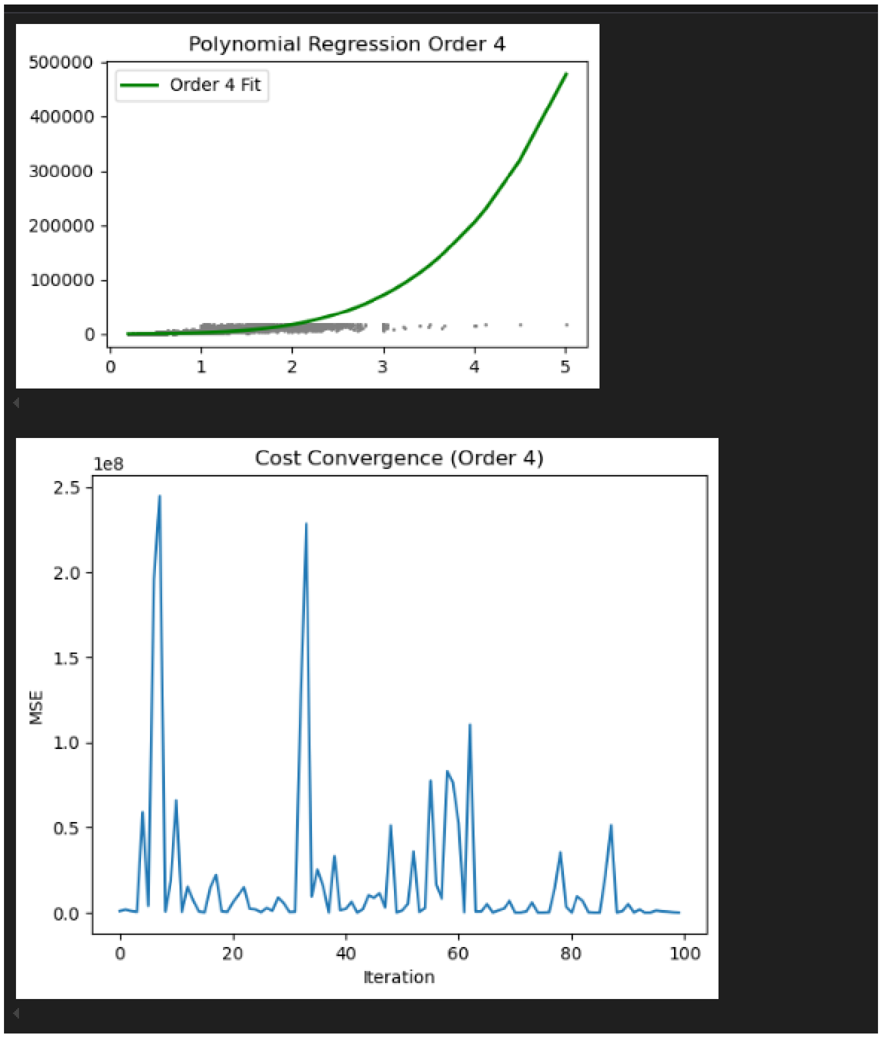


Figure 11: Order 4 Polynomial Regression Results