Documentation

Libraries Used

1. **pandas:** Handles Table data (Loading, Saving, Modifying tables)
2. **numpy:** Handles Matrix data (Define matrices and perform operations with them)
3. **matplotlib.pyplot:** Plot Graph (Scatter, Surface, curve, etc.)

Data Preprocessing

1. **Loading the dataset into a DataFrame:** Using the pandas library we load the given dataset into a pandas DataFrame.
2. **Normalizing the feature variable:** We normalize the feature variables by utilizing the formula: $X' = (X - \mu) / \sigma$ where μ represents the mean of the feature column, and σ represents the standard deviation of the feature column.
3. **Filling Null Values:** *–Only in Part B–* We predict the null values in cells using the mean of the existing values of the corresponding feature.
4. **Shuffle split the dataset into training and testing sets:** We shuffle the dataset and split the dataset into training and testing sets; using 80% data for training and 20% for testing.

Polynomial Regression

Polynomial Transformation and Dummy Ones

Using an input degree, we transform the input data by adding higher degree terms. We add a column of 1s at the start of the dataset to facilitate matrix operations.

Error Function

We use Mean squared error as our error metric and create a function for later use.

Batch Gradient Descent

We use matrix operations and the following formula to perform batch gradient descent:

```
for iteration in range(max_iterations):
  Y_pred = X @ W
  gradient = X.T @ (Y_pred – Y)
  W –= (learning_rate/n)*gradient
```

where W is our weight vector (includes Wo – bias term) and is initialized to 0. @ denotes matrix multiplication. X is the matrix of training points and features; while Y is the true target value of those points.

Stochastic Gradient Descent

We use matrix operations and the following formula to perform stochastic gradient descent:

```
for iteration in range(max_iterations):
  nextrow = random.randint(0, len(train)–1)
  x_i = X[nextrow]
  y_i = Y[nextrow]
  y_pred = x_i @ W
  gradient = x_i * (y_pred – y_i)
  W –= (learning_rate/n) * gradient
```

where W is our weight vector (includes Wo – bias term) and is initialized to 0. @ denotes matrix multiplication. nextrow is a randomly selected row from our dataset, containing a combination of features and their target value. X is the matrix of training points and features; while Y is the true target value of those points. x_i is the feature vector of nextrow and y_i is its target value.

Regularized Linear Regression

To implement the gradient descent algorithms; we must first differentiate both loss functions (to find the gradient) and then write them in matrix form (for faster operations). Differentiating the equation (w.r.t. W) we get:

SUM(tn – wT * Xn) + lambda * (0.5 * q) * SUM(|wj|^(q–1))

as a matrix equation:

Y – (W.T * X) + lambda * 0.5 * q * (W**(q–1)) (for q = 2 or 4)

Y – (W.T * X) + lambda * 0.5 * q * (abs(W)**(–0.5)) (for q = 0.5)

For q = 1:

SUM(tn – wT * Xn) + lambda * 0.5 * SUM(sign(wj))

as a matrix equation:

Y – (W.T * X) + lambda * 0.5 * (sign(W))

To modify our above fuctions to use these, we just need to change the gradient as follows:

Batch Gradient Descent

if q = 0.0: gradient = X.T @ (Y_pred – Y)

q = 0.5: gradient = X.T @ (Y_pred – Y) + (lmbda * 0.5 * q) * (np.abs(W) ** (–0.5))

q = 1.0: gradient = X.T @ (Y_pred – Y) + (lmbda * 0.5) * np.sign(W)

else: gradient = X.T @ (Y_pred – Y) + (lmbda * 0.5 * q) * (W ** (q–1))

Stochastic Gradient Descent

if q = 0.0: gradient = x_i * (y_pred – y_i)

q = 0.5: gradient = x_i * (y_pred – y_i) + (lmbda * 0.5 * q) * (np.abs(W) ** (–0.5))

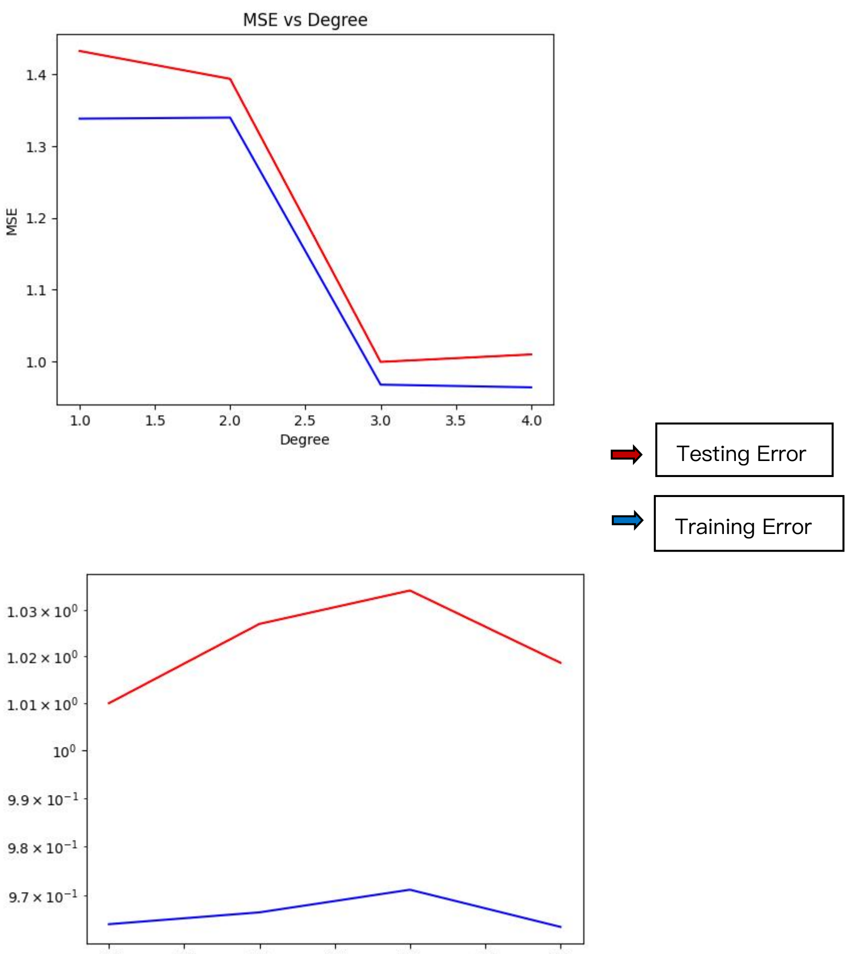
q = 1: gradient = x_i * (y_pred – y_i) + (lmbda * 0.5) * np.sign(W)

else: gradient = x_i * (y_pred – y_i) + (lmbda * 0.5 * q) * (W ** (q–1))

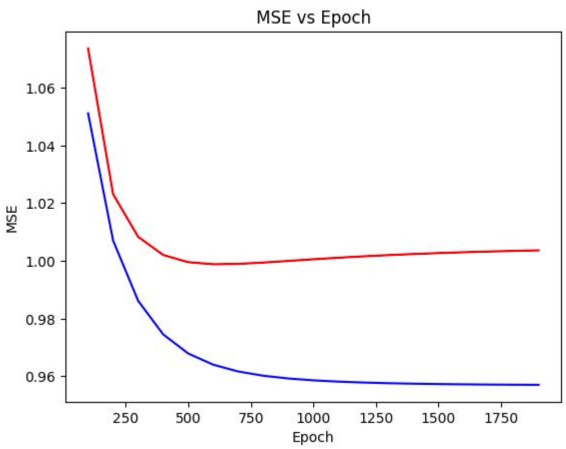
Graph Plotting

1-A

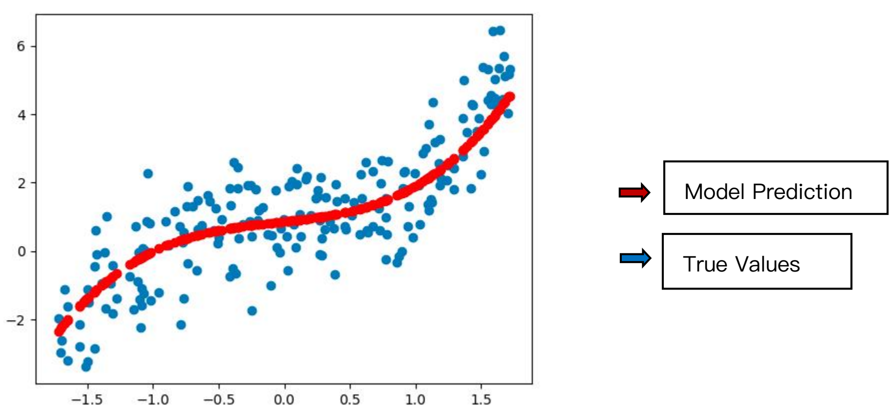
Training and Testing Error vs Degree of polynomial



Training and Testing Error vs Epoch



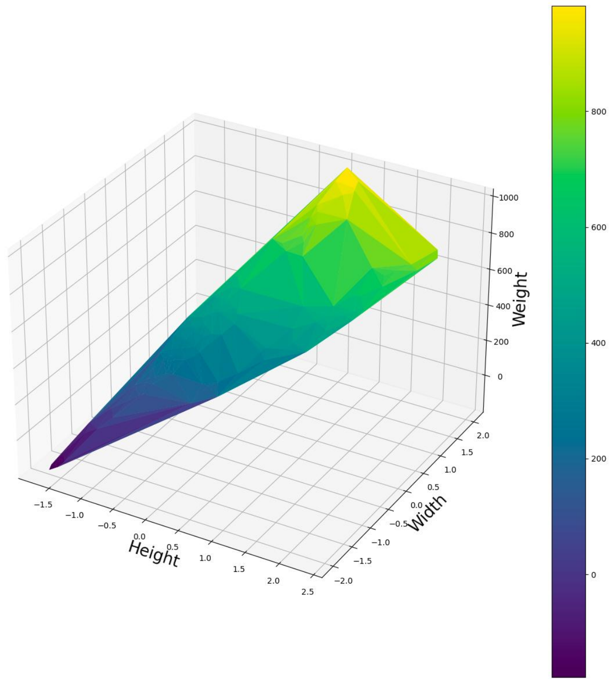
Plotting the best fit curve



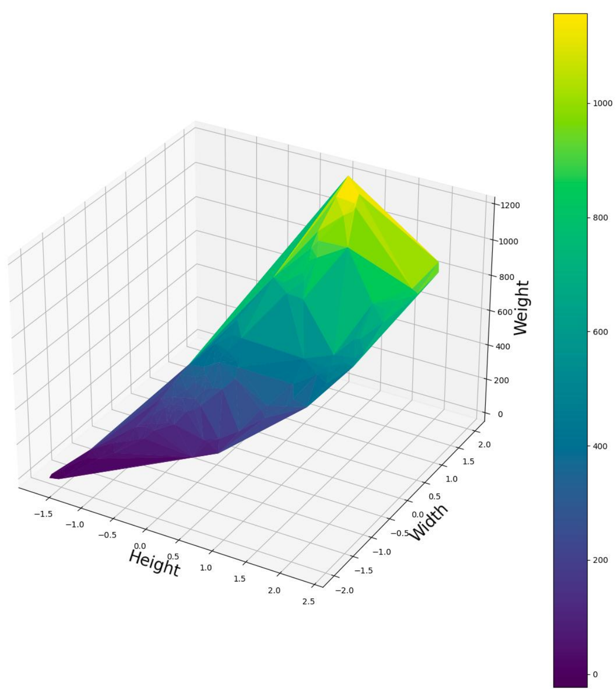
1-B

Polynomial Regression Surface plot

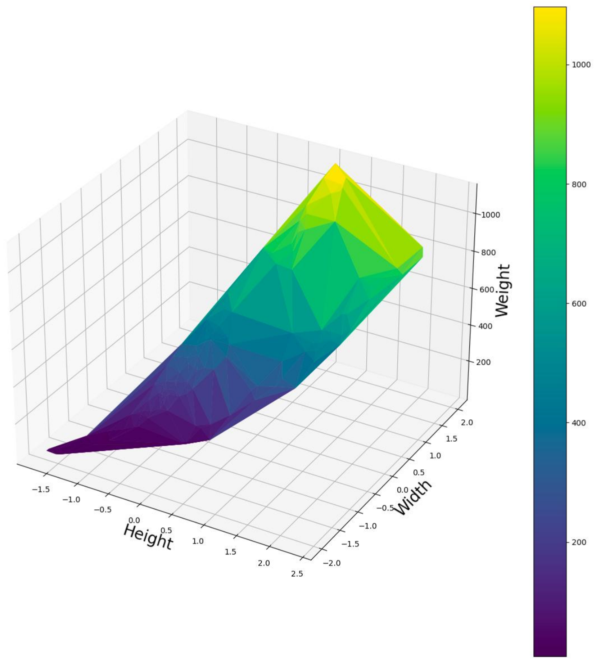
Degree 1 best fit



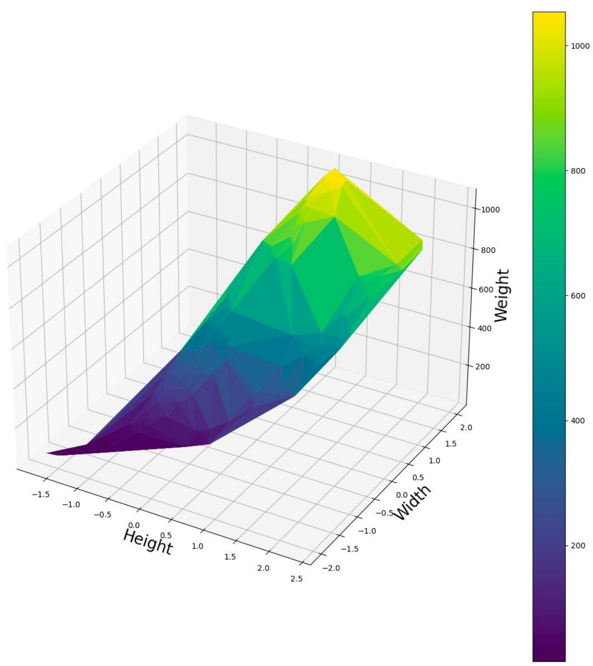
Degree 2 Best Fit



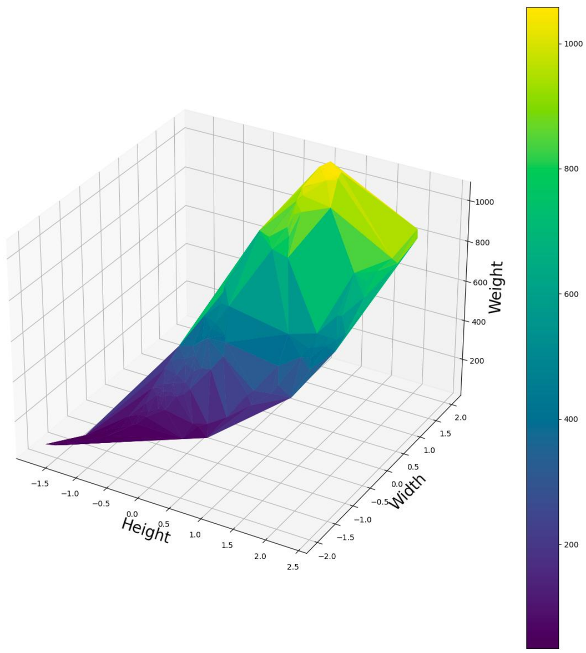
Degree 3 Best Fit



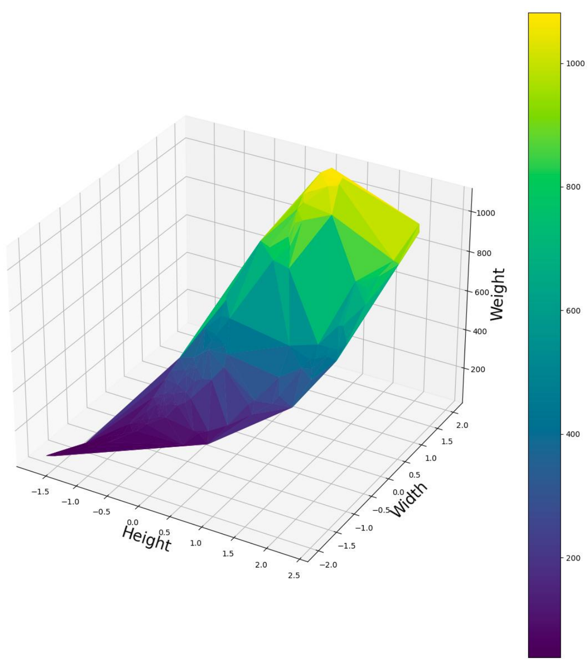
Degree 4 Best Fit



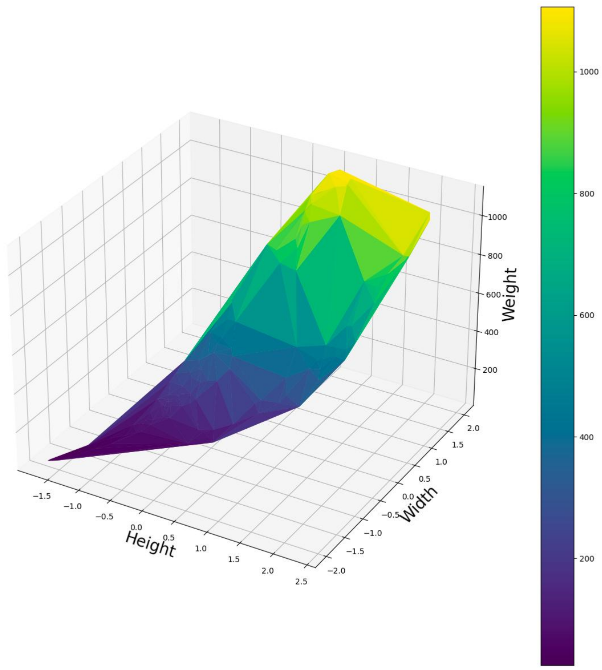
Degree 5 Best Fit



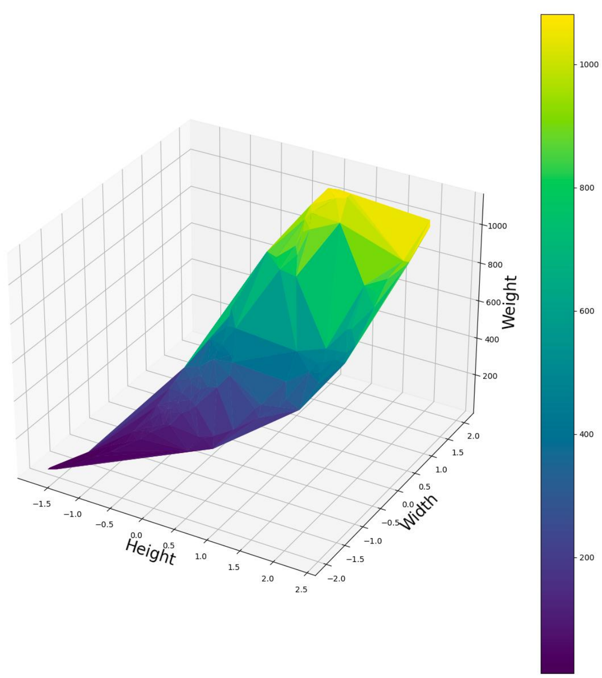
Degree 6 Best Fit



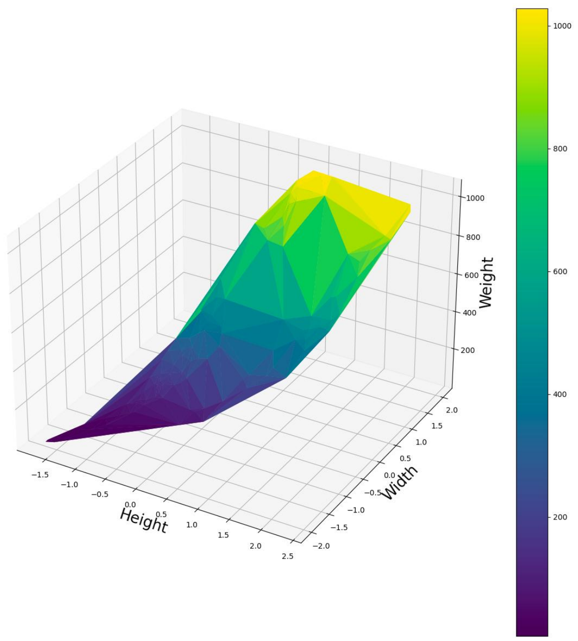
Degree 7 Best Fit



Degree 8 Best Fit

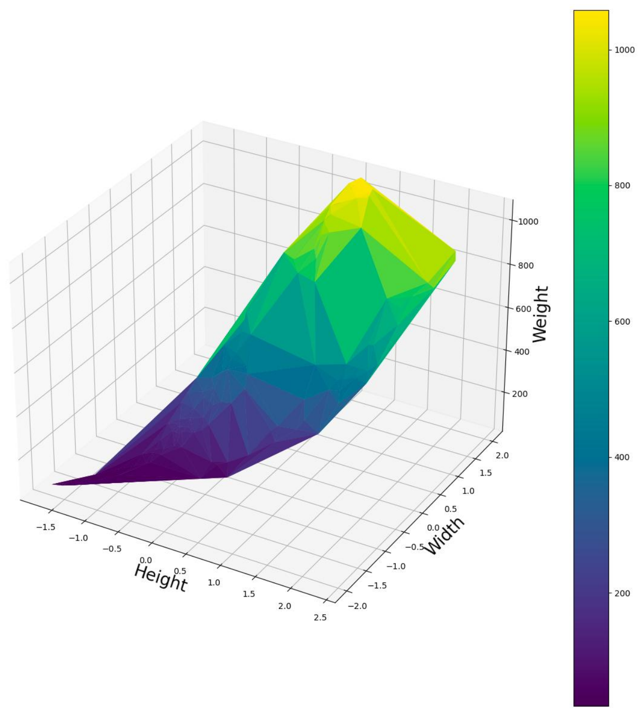


Degree 9 Best Fit

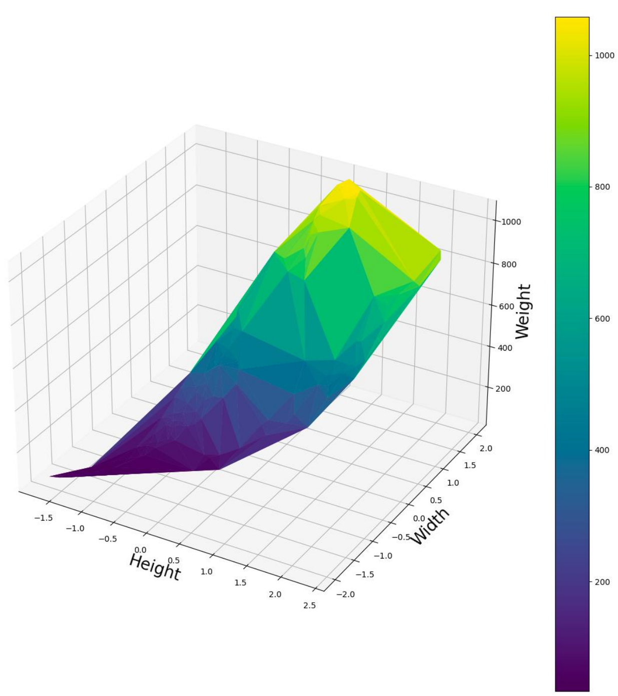


Optimal Regularized Linear Regression Model Surface Plot

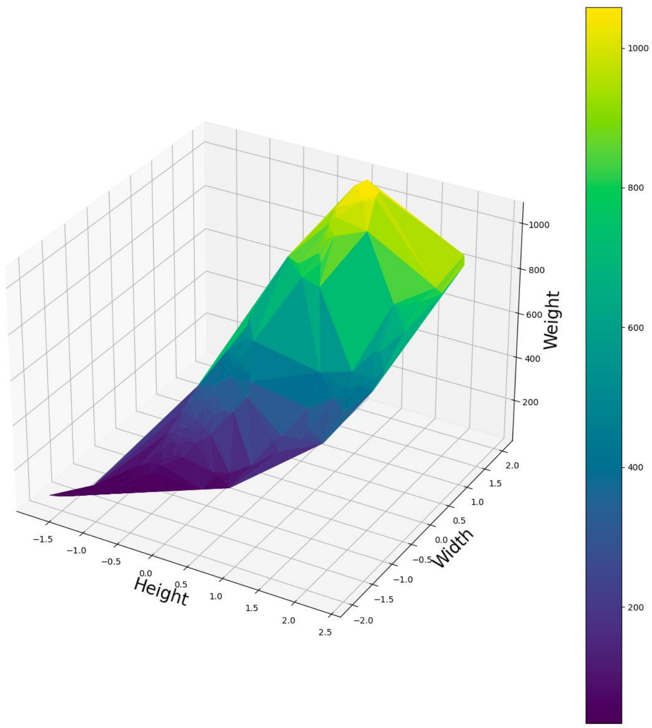
Q = 0.5



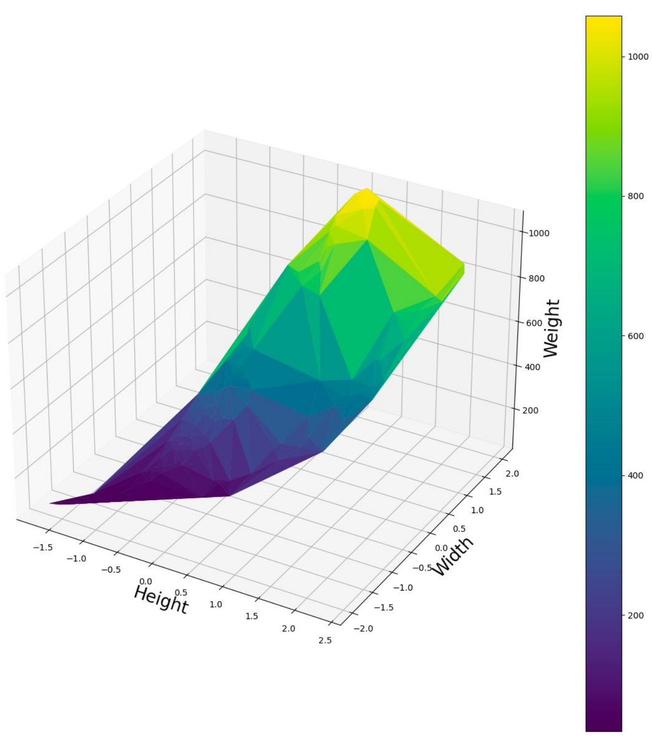
Q = 1



Q = 2



Q = 4



Comparative Analysis

1-A

We observe that a polynomial model of degree 3, with learning rate of 0.01; trained with batch gradient descent algorithm for 600 iterations gives us the best result of 0.9988638271467665 testing error and 0.9640454936180258 training error out of all polynomial models.

1-B

<u>Tabulation of MSE vs Degree</u>

Q	Degree	Batch Test Error	Batch Train Error	Stochastic Test Error	Stochastic Train Error
0	1	54929.7961	20953.2505	51559.6331	22073.3716
	2	48393.6692	13907.1528	44622.4110	15285.5285
	3	46574.5700	12922.6224	42194.1122	14242.0322
	4	45299.5922	12405.5280	40691.8961	13862.1792
	5	45460.6792	12233.4381	40220.3483	14803.3771
	6	46192.7053	12176.7766	43016.1331	18953.8453
	7	46834.5874	12166.2085	81165.9661	39570.2850
	8	47042.6682	12160.4672	2.5522019e+118	1.3328255e+118
	9	46850.2169	12150.4923	inf	inf

Q	Degree	Batch Test Error	Batch Train Error	Stochastic Test Error	Stochastic Train Error
0.5	1	54929.7961	20953.2505	51559.6331	22073.3716
	2	48393.6692	13907.1528	44622.4110	15285.5285
	3	46574.5700	12922.6224	42194.1122	14242.0322
	4	45299.5922	12405.5280	40691.8961	13862.1792
	5	45460.6792	12233.4381	40220.3483	14803.3771
	6	46192.7053	12176.7766	43016.133	18953.8453
	7	46834.5874	12166.2085	81165.9661	39570.2850
	8	47042.6682	12160.4672	2.55220e+118	1.332825e+118
	9	46850.2169	12150.49231	inf	inf

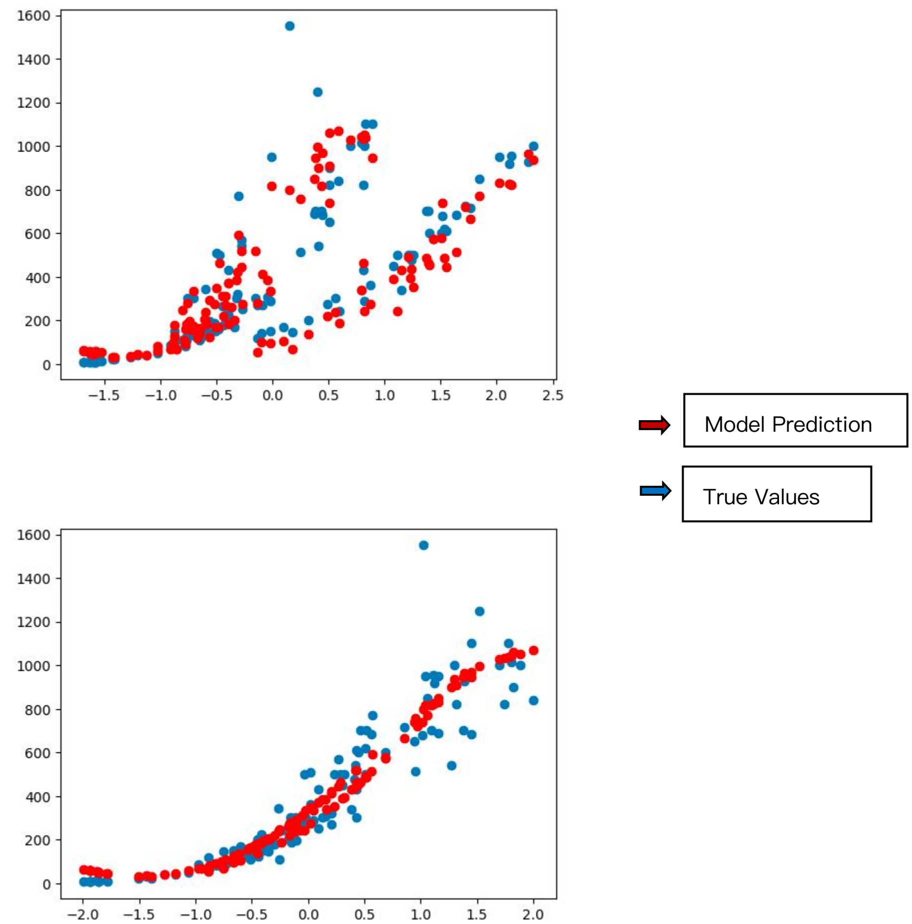
Q	Degree	Batch Test Error	Batch Train Error	Stochastic Test Error	Stochastic Train Error
1	1	54929.7961	20953.2505	51559.6331	22073.3716
	2	48393.6692	13907.1528	44622.4110	15285.5285
	3	46574.5700	12922.6224	42194.1122	14242.0322
	4	45299.5922	12405.5280	40691.8961	13862.1792
	5	45460.6792	12233.4381	40220.3483	14803.3771
	6	46192.7053	12176.7766	43016.1331	18953.8453
	7	46834.5874	12166.2085	81165.9661	39570.2850
	8	47042.6682	12160.4672	2.552201e+118	1.332825e+118
	9	46850.2169	12150.4923	inf	inf

Q	Degree	Batch Test Error	Batch Train Error	Stochastic Test Error	Stochastic Train Error
2	1	54929.7961	20953.2505	51559.6331	22073.3716
	2	48393.6692	13907.1528	44622.4110	15285.5285
	3	46574.5700	12922.6224	42194.1122	14242.0322
	4	45299.5922	12405.5280	40691.8961	13862.1792
	5	45460.6792	12233.4381	40220.3483	14803.3771
	6	46192.7053	12176.7766	43016.1331	18953.8453
	7	46834.5874	12166.2085	81165.9661	39570.2850
	8	47042.6682	12160.4672	2.552201e+118	1.332825e+118
	9	46850.2169	12150.4923	inf	inf

Q	Degree	Batch Test Error	Batch Train Error	Stochastic Test Error	Stochastic Train Error
4	1	54929.7961	20953.2505	51559.6331	22073.3716
	2	48393.6692	13907.1528	44622.4110	15285.5285
	3	46574.5700	12922.6224	42194.1122	14242.0322
	4	45299.5922	12405.5280	40691.8961	13862.1792
	5	45460.6792	12233.4381	40220.3483	14803.3771
	6	46192.7053	12176.7766	43016.1331	18953.8453
	7	46834.5874	12166.2085	81165.9661	39570.2850
	8	47042.6682	12160.4672	inf	inf
	9	inf	inf	inf	inf

The best plot

We observe that a regularized polynomial model of degree 5, q 2 and lambda 10^(−20), with learning rate of 7; trained with stochastic gradient descent algorithm for 50000 iterations gives us the best result of 40220.3483 testing error and 14803.3771 training error out of all models.



Team Members

1. Aryan Gupta – 2021A7PS0162H
2. Subal Tankwal – 2021A7PS1407H