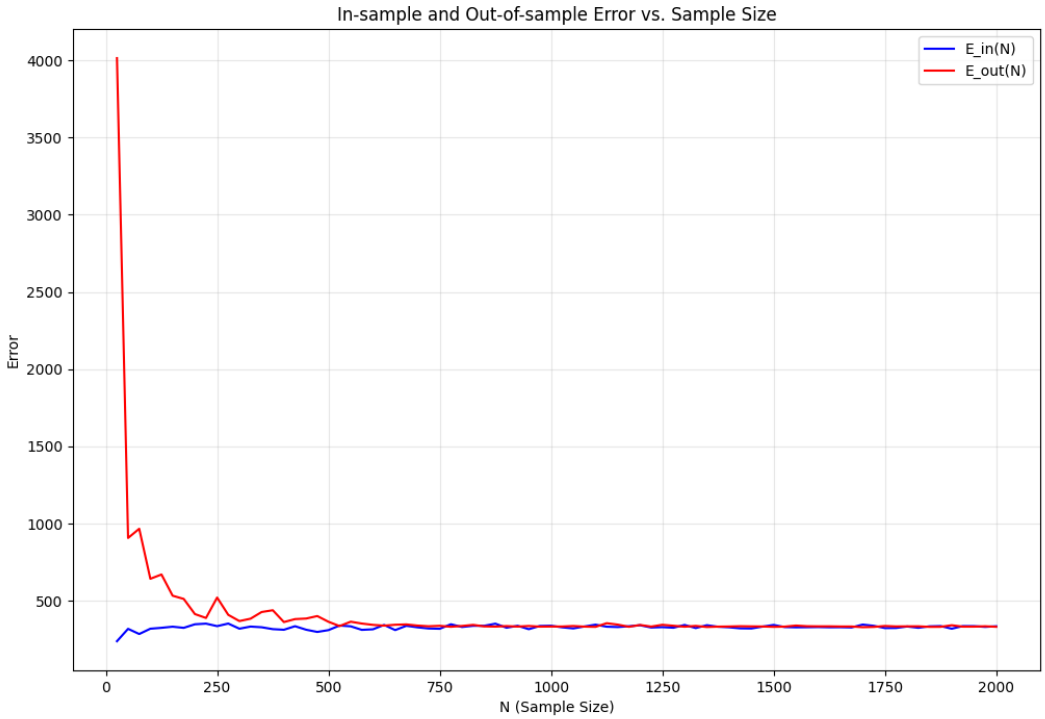
Question 12: Report



Explanations:

The difference between this question and the previous question 11 is that we only preserve the first 2 features instead of using all the 12 features. We could see that in the resulting figure, the highest out of sample value is lower comparing to the previous question. Although they both gradually decrease as the sample size increases.

This is because when we're in a higher dimensional space, we would have to face the problem of generalization. Although using all the 12 features results in a space that can form more flexible models, meaning that is more possible to find a hyperplane that can separate our training data, making the in sample error to be about 0, when the model is used to evaluate the

unseen data, over-parameterized models may cause overfitting. In this question we decrease the amount of features to 2, indicates a choice toward generalization than approximation.

However, we can see that the in sample error is a nonzero value, even though the value is low for over all possible N. This is also the result as we reduce the dimension of the space to 3 (including the fixed zeroth coordinate), our free parameter is reduced to 2, so that it is hard to make the training data linearly separable.

Nonetheless, reducing the number of features would make $\mathrm{d_{VC}}$ small, and as the amount of data needed is proportional to the VC dimension, we could reach the same level of generalization error using less examples, and this is reflected in the drastic decrease in the out of sample error in a low value of N.

Code:

The initialization part (getting the dataset, and generate random indices is the same as the previous 2 questions), so the first difference occurs when we save the input vectors, as we only save the first 2 features instead of 12:

```
38          # subsubaim: convert each sample data points (which is a dictionary) to a ndarray and save in X_sample_array
39          X_sample_array = []
40          for sample in X_sample:
41              input_vector = [1, sample.get(1, 0), sample.get(2, 0)]
42              X_sample_array.append(input_vector)
```

The following part is almost the same as the previous question, except modifying it to be more clear and simple, but the concept behind is all the same:

```
        # subsubaim: convert X_sample_array to a matrix, and y_sample to an array
        X_mat = np.array(X_sample_array)
        y_array = np.array(y_sample)

        # subsubaim: perform linear regression
        w = np.linalg.inv(X_mat.T @ X_mat) @ X_mat.T @ y_array

        # subsubaim: calculate the in-sample error
        in_sample_error = np.mean((X_mat @ w - y_array) ** 2)

        # subsubaim: estimate the out of sample error
        out_sample_indices = list(set(range(len(X))) - set(random_sample_indices))
        X_out = np.array([[1, X[i].get(1, 0), X[i].get(2, 0)] for i in out_sample_indices])
        y_out = np.array([y[i] for i in out_sample_indices])
        out_of_sample_error = np.mean((X_out @ w - y_out) ** 2)
        in_out_sample_error.append((in_sample_error, out_of_sample_error))

    # subaim: calculate the average in-sample error and out of sample error for each N
    avg_in_sample_error = np.mean([error[0] for error in in_out_sample_error])
    avg_out_of_sample_error = np.mean([error[1] for error in in_out_sample_error])

    avg_in_error_for_each_N.append(avg_in_sample_error)
    avg_out_error_for_each_N.append(avg_out_of_sample_error)
```