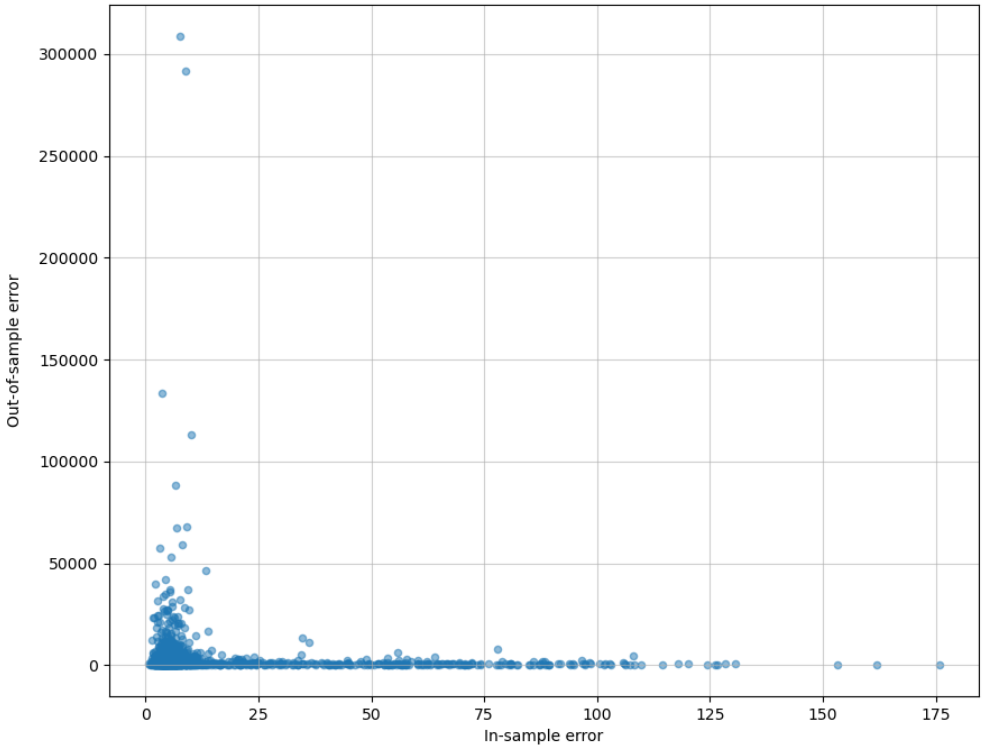
Question 10: Report



Explanation:

The first thing we would probably notice is that, there's 2 points in the upper left corner of the plot, which indicates a small in-sample error, and a relatively large out of sample error. I think this is the result that I would expect, since we only choose 32 from all the 8192 examples, using such a small subset of the dataset would make the model vary greatly, depending on the subset we choose, and if the subset contains outliers, or the examples chosen are not representative enough for the entire dataset, then overfitting to such examples would cause high out of sample error, meaning that the model is nor well generalized. In addition to these 2 points, we can also see that in about 10 experiments, we have out of sample error greater than 50000, which is also

quite high. Most of the other points in the lower left corner still shows that the out of sample error is not that small, this result verifies our intuition.

Next, we can see that in the lower middle part, and the lower right corner, there are experiments having nearly 0 out of sample error but higher in sample errors (compared to those having only in sample errors less than 25.) For this part, this result actually matches that in theory, higher in sample error (due to underfitting) would often end up with lower out of sample error, because models like this generalize better than those that overfit. However, I still think that it's surprising to use 32 points to generalize well on the remaining points, but the amount of experiments like this is small, so I think maybe it's reasonable due to probability.

Last, observe the whole plot, we can see that the distribution varies a lot over all of the experiments. Even though we don't know the correlations between the 8192 examples, randomly choosing 32 from them still results in a high variability (since there would be C(8192, 32) possibilities.) As mentioned before, this would cause our distribution to depend highly on the 32 samples we choose, so I don't think we can tell more by only conducting 1126 experiments.

Code:

Read in the data:

```
def read_libsvm_format(file_path):
    X, y = [], []
    with open(file_path, 'r') as f:
        for line in f:
            parts = line.strip().split()
            y.append(float(parts[0]))
            features = {}
            for item in parts[1:]:
                index, value = item.split(":")
                features[int(index)] = float(value)
            X.append(features)
    return X, y

# aim: read the data from the file 'cpusmall_scale'
X, y = read_libsvm_format('cpusmall_scale')

# explain: the data X returned is in the form of a list of dictionaries, where each dictionary contains the features of a data point
# ex: X[0] = {1: -0.993496, 2: -0.993043, 3: -0.850291, 4: -0.963479, 5: -0.960727, 6: -0.900596, 7: -0.96642, 8: -0.863996, 9: -0.606175, 10: -0.999291, 11: 0.08118

# explain: the label y is a list of floats
# ex: y[0] = 90.0

```

Calculate w_{LIN} and E_{in}:

```python
N = 32
feature_amount = 12
in_out_sample_error = []
# aim: perform linear regression for 1126 times, calculate the in-sample error and estimate the out of sample error
for experiment in tqdm(range(1126)):
    np.random.seed(experiment)

    random_sample_indices = np.random.choice(len(X), N, replace=False)
    X_sample = [X[i] for i in random_sample_indices]
    y_sample = [y[i] for i in random_sample_indices]

    # subaim: convert each sample data points (which is a dictionary) to a ndarray and save in X_sample_array
    X_sample_array = []
    for sample in X_sample:
        input_vector = np.concatenate((np.array([1]), np.zeros(feature_amount)))
        # explain: originally we need subtract 1 because indices in libsvm format start from 1, but now we add the 0-th element (1) to each input vector
        for index, value in sample.items():
            input_vector[index] = value
        X_sample_array.append(input_vector)

    # subaim: 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)

    # subaim: perform linear regression
    # explain: we do linear regression by using the normal equation (w = (X^TX)^(-1)X^Ty)
    # explain: we use np.linalg.inv() calculate the inverse of X^TX, and use @ to denote matrix multiplication
    w = np.linalg.inv(X_mat.T @ X_mat) @ X_mat.T @ y_array

    # subaim: calculate the in-sample error
    # explain: the in-sample error is the mean squared error of the N points
    in_sample_error = np.mean((X_mat @ w - y_array) ** 2)

```

Estimate E_{out}:

```python
    # subaim: estimate the out of sample error
    # explain: the out of sample error is estimated by averaging the squared error of the rest of the 8192 - N data points
    out_of_sample_error = 0
    for i in range(len(X)):
        if i not in random_sample_indices:
            unseen_input_vector = np.concatenate((np.array([1]), np.zeros(feature_amount)))
            for index, value in X[i].items():
                unseen_input_vector[index] = value
            out_of_sample_error += (unseen_input_vector @ w - y[i]) ** 2
    out_of_sample_error /= (len(X) - N)
    in_out_sample_error.append((in_sample_error, out_of_sample_error))
```