Jonothan Meyer
Machine Learning
Quiz 2

1. Generate 500 random X values from -3 to 3.

Achieved with the following python code:

```
data_points = 500
X = np.random.uniform(-3,3,data_points)
X = X.reshape((len(X), 1)) #Convert X to the proper matrix form
```

2. Generate 500 Y values using distribution "y = 0.5 * X^5 - X^3 - X^2 + 2 + a little bit randomness, both positive and negative.

Achieved with the following python code:

```
y = .5 * X**5 - X**3 - X**2 + 2 #+ np.random.normal(0,1,1)
for i in range(len(y)):
    randomnum = np.random.normal(0,10,1)
    y[i] = y[i] + randomnum #Done to add a bit more random variance
```

3. Use X and Y as the whole dataset and use 200 samples as testing + 300 samples as training.Testing and training sets must be disjoint.

```
testing_X = X[:200]
testing_y = y[:200]

training_X = X[200:500]
training_y = y[200:500]
```

4. Try Linear Regression and Polynomial Regression (PolynomialFeatures + LinearRegression) in SKLearn from degree 2 to 25 to fit the training data samples.

This python code borrowed heavily from 1_LinearRegressionSKLearn_ex_poly.py class:

```
def plot_nth_degree(x_data, y_data, nth_degree):
    model = LinearRegression()
     model.fit(x_data, y_data)
    y_pred = model.predict(x_data)
    model2 = LinearRegression()
     poly2_features = PolynomialFeatures(degree=nth_degree)
    X_poly2 = poly2_features.fit_transform(training_X)
    X_poly2_test = poly2_features.fit_transform(testing_X)
     model2.fit(X_poly2, training_y)

    poly_features = PolynomialFeatures(degree=nth_degree, include_bias=False)
     X_poly = poly_features.fit_transform(training_X)
    model = LinearRegression()
```

```
        model.fit(X_poly, training_y)
        Xplot = np.arange(-3, 3, .02)
        Xplot = Xplot.reshape(-1, 1)
        Xplot_poly = poly_features.fit_transform(Xplot)
        yplot_pred = model.predict(Xplot_poly)

        plt.scatter(x_data, y_data, color='red', label='Data Point', linewidths=1)
        label = "Poly Degree:" + str(nth_degree)
        plt.plot(x_data, y_pred, color='blue', linewidth=3, label='Poly Degree: 1')
        TrainL = MSE_history_training[nth_degree-1]
        TestL =  MSE_history_testing[nth_degree-1]
        plt.title("Machine Learnining, Quiz 2, Jonothan Meyer, TrainL:" + str(round(TrainL, 1))
            " + TestL:" + str(round(TestL,1)))
        plt.plot(Xplot, yplot_pred, color='green', linewidth=3, label=label)
        plt.legend(loc="upper left")
        plt.show()
```

5. Calculate the loss using mean squared error loss = average (sum (prediction - y)^2) for all the training samples after the model was selected.

```
    def MSE_array(x_data, y_data, nth_degree=25):
        iteration_array_history = []
        MSE_value_history = []
        for i in range(nth_degree):
        model2 = LinearRegression()
        poly2_features = PolynomialFeatures(degree=i)
        X_poly2 = poly2_features.fit_transform(x_data)
        model2.fit(X_poly2, y_data)
        y_pred2 = model2.predict(X_poly2)
        iteration_array_history.append(i)
        MSE = mean_squared_error(y_data, y_pred2)
        MSE_value_history.append(MSE)
        return iteration_array_history, MSE_value_history


    iterationValues_training, MSE_history_training = MSE_array(training_X,training_y)
    iterationValues_testing, MSE_history_testing = MSE_array(testing_X,testing_y)
```

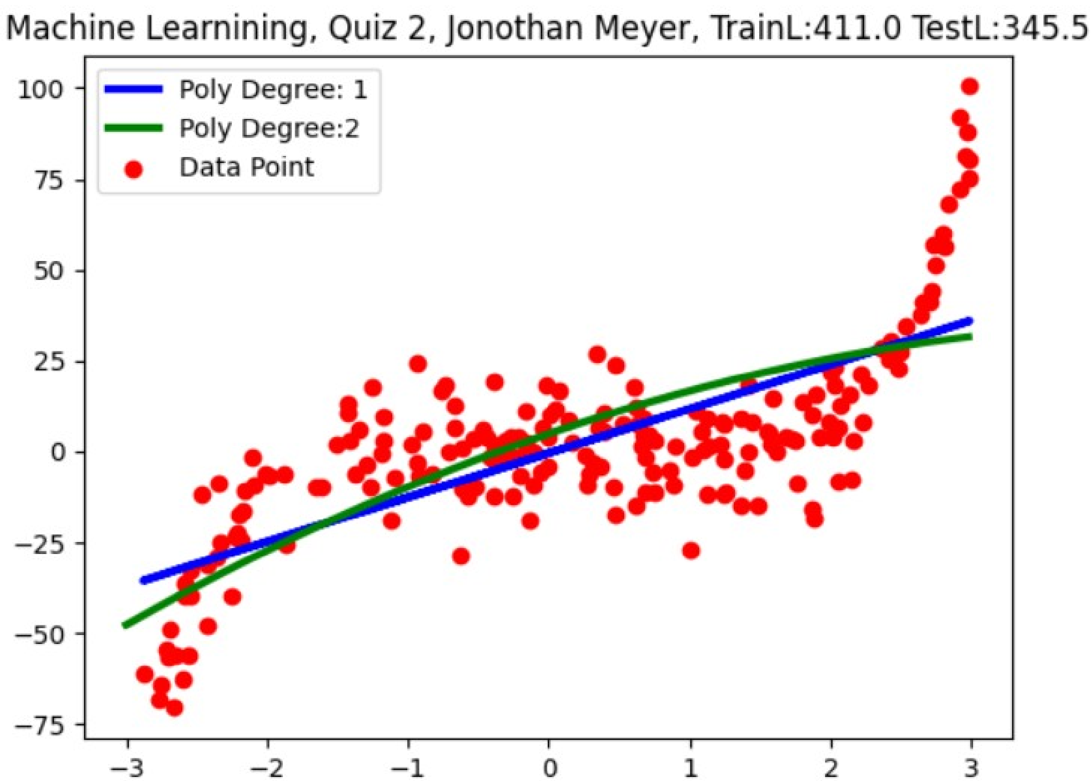
Sample Output of Testing Data:
[828.741816224535, 388.5171858575887, 388.51186958678807, 125.1684316151544, 124.5989620859292, 92.13321503800472, 91.70605737803653, 91.27311396292804, 91.17901784670637, 91.12158375939038, 90.0620522723897, 90.03802508345007, 87.96040949573735, 87.87916126738506, 87.19616866927727, 87.16401698338007, 87.15397830753807, 86.90358162678767, 86.90105837097106, 86.84977189954192, 86.55170474691433, 86.34302575963679, 86.26975833568856, 86.19778370491052, 85.00213721676833]

6. Calculate the loss using mean squared error loss = average (sum (prediction - y)2) for all the testing samples after the model was selected and used for testing data predictions.

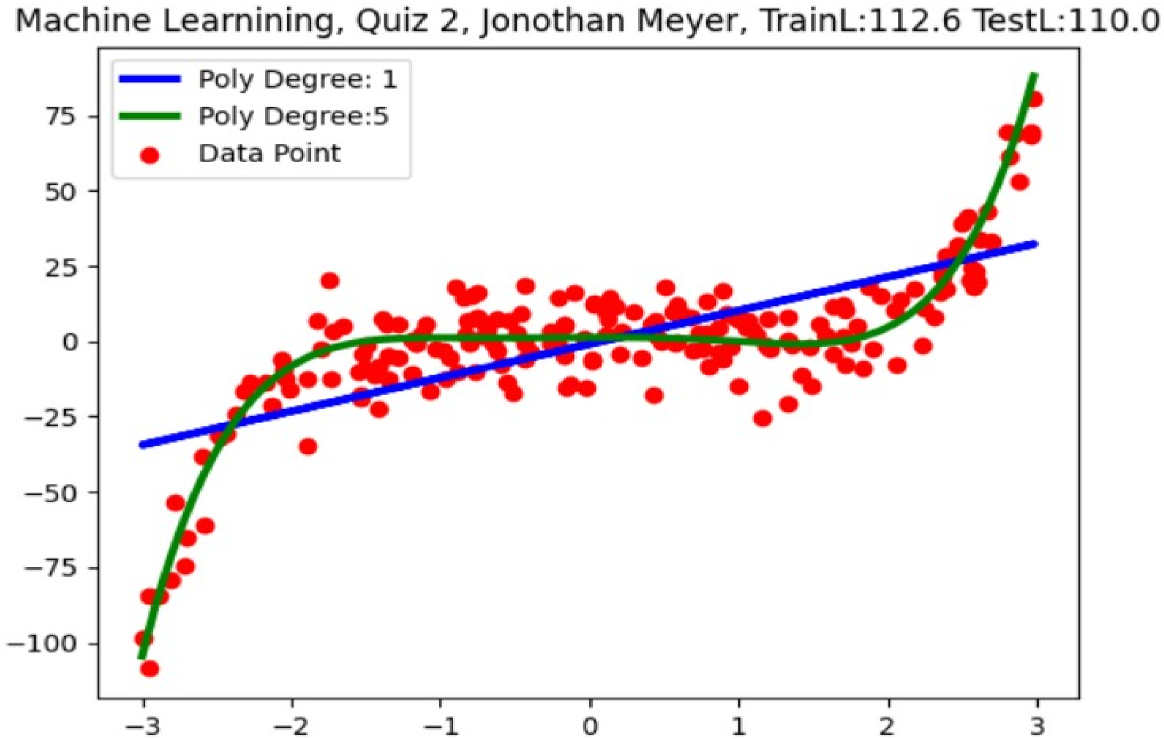
Sample Output of Training Data:
[766.8261380708556, 374.5182072211934, 373.1685021631236, 123.4259368589275, 123.21185420652914, 102.6036684002086, 102.46016365075745, 102.16735464379873, 102.1624964028619, 101.94363876813328, 101.94350115765, 100.08335797658106, 99.95951705231826, 99.93965840703915, 99.86954796865281, 99.86677492945974, 99.85141163806917, 99.39598630312264, 98.7861833657249, 95.91031069661592, 95.6050085418232, 95.52713090250239, 95.46301149231911, 95.33763654106227, 95.08845899644936]

7. Plot the figures showing your predictions using degree =2, 5, 8, 10, 20. Include these figures in your report.

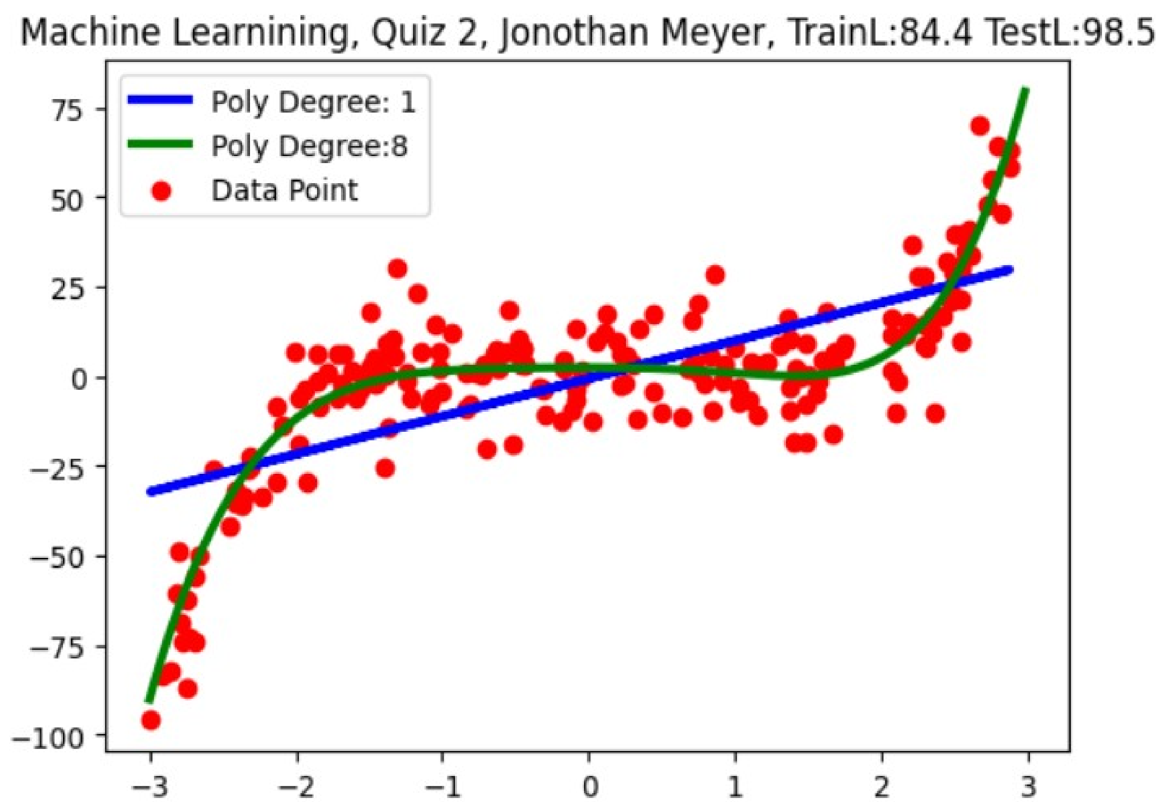
Degree 2:



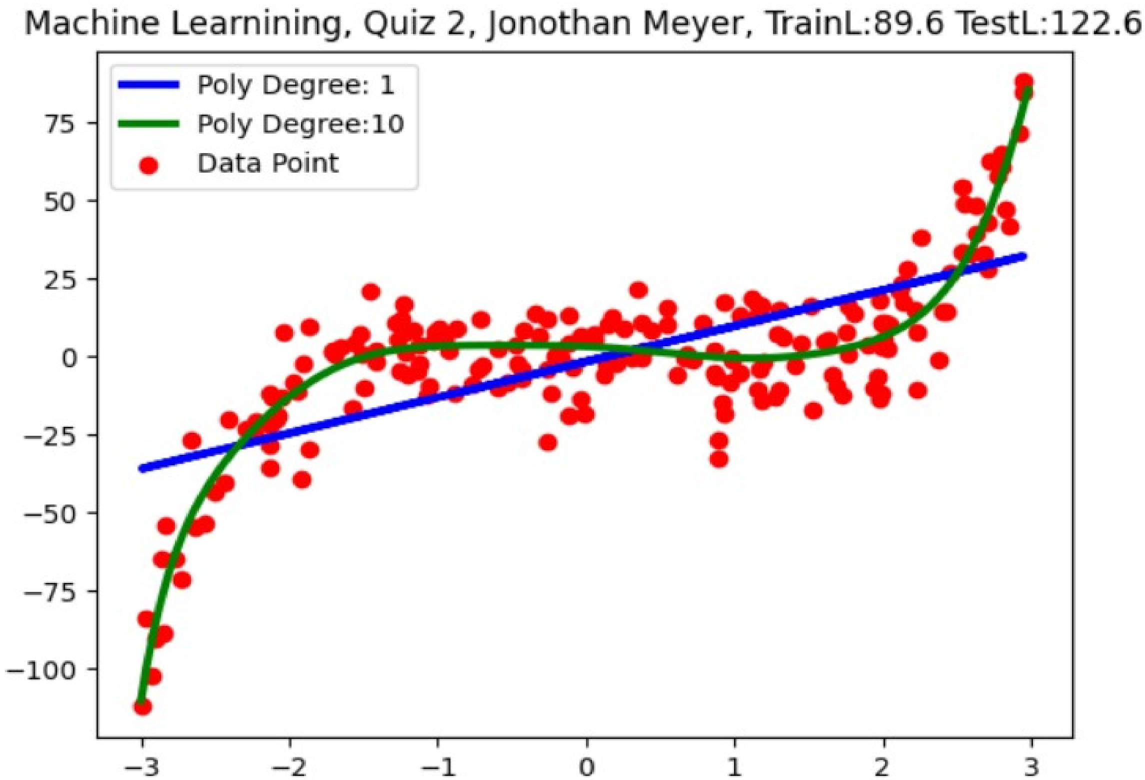
Degree 5:



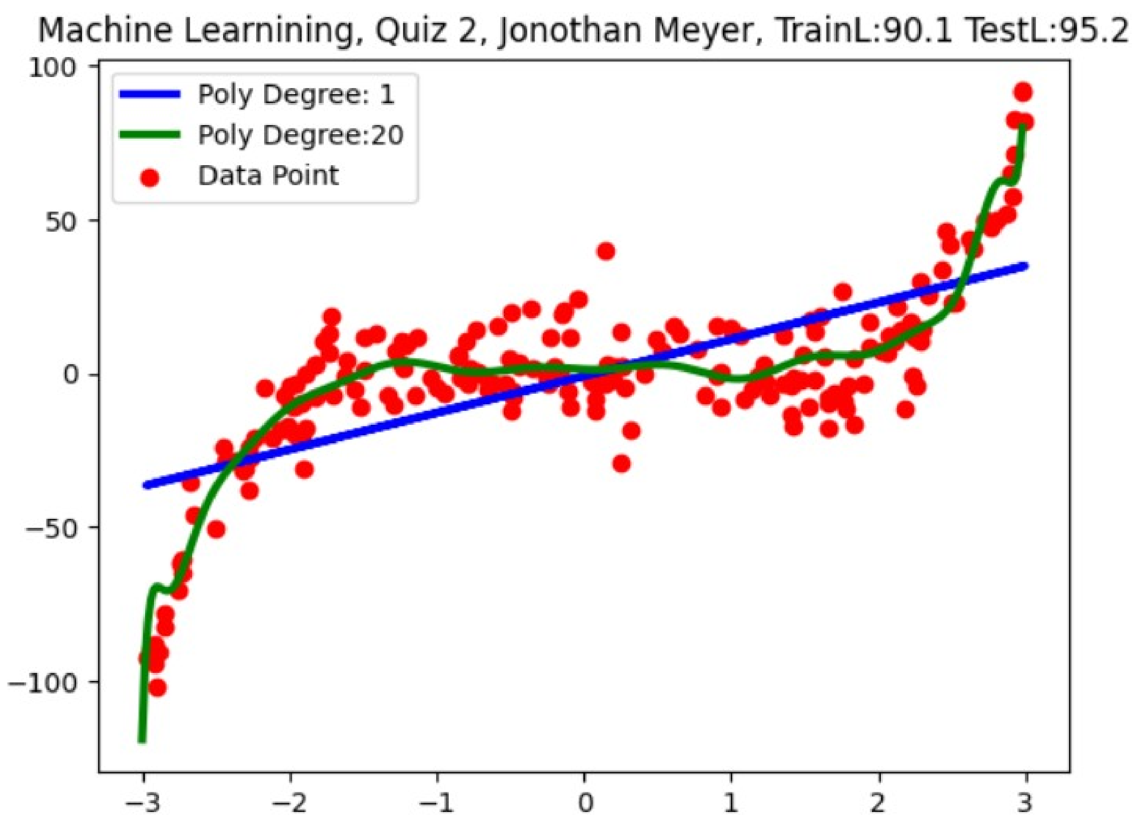
Degree 8:



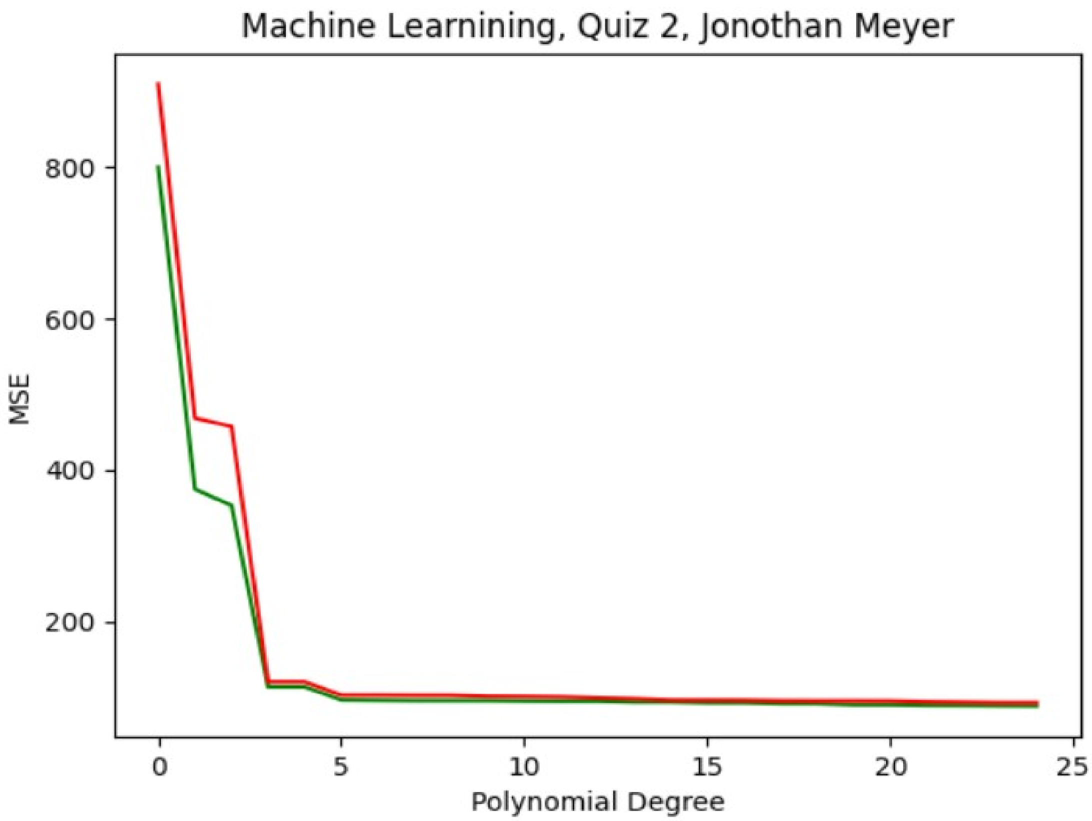
Degree 10:



Degree 20:



8. Plot a figure tracking the change of training and testing loss using different models (degree = 2....25). Include this figure in your report.



9. Which degree number(s) is/are the best and why? Provide your answer to this question in your report.

The Mean Squared Error lowers dramatically for the first 4 degrees of polynomial. After that its still lowered by increasing the degree, however it is marginally insignificant after the 4th degree. For this reason I think a polynomial degree of 4 or 5 is best because those degree's provide the best fit with the least amount of processing. Any more degree's and the model becomes over-fit and is doing more work than it receives MSE loss.