University of Regina

ENEL 890AN: POWER SYSTEM ELECTRONICS

ASSIGNMENT 2

SUBMITTED BY: Hitarthi Gandhi (200428713)

1. **Implement the gradient descent algorithm (regression_gradient_descent) from scratch to compute coefficients for multiple regression. [Make sure you add a 'constant' column in the feature matrix. Implement functions for output prediction, feature derivative and RSS calculations.]**

To implement to regression gradient decent we used Fish.csv dataset. Then we divided that into 80% training and 20% training. For splitting the data we used scikit learn inbuilt function train_test_split. Here we used Length1 column as our inout feature and weight column as our output. We add one extra column constant in our feature matrix. We can consider it as intercept. below function return the first column as inout feature and another it as output array.

```
def getdata_numpy(Dataset, features, output):

    features_matrix = np.array([np.ones(len(Dataset))] + [Dataset[feature] for feature in features])

    output_array = np.array(Dataset[output])

    return features_matrix, output_array
```

Now next step is to predict the output for the feature matrix and as we know predicated output means the dot product of feature matrix transpose and weights.

Below function predict the prediction 'errors' (predictions - output) then the derivative of the regression cost function with respect to the weight of 'feature' is just twice the dot product between 'feature' and 'errors'. IT gives the derivation of errors and features.

```
def feature_derivation(errors, feature):

    return 2 * feature.dot(errors)
```

Here we created gradient desecent function for the linear regression. In this gradient descent algorithm at each step computes the magnitude/length of the gradient. After this it derivates the function and the When the magnitude of the gradient is smaller than theinput tolerance returns the final weight vector. Below code return the weights for gradient decent.

```
def regression_gradient_descent(feature_matrix, output, initial_weights, step_size, tolerance):

    converged = False

    weights = np.array(initial_weights)

    while not converged

        predictions = output_prediction(feature_matrix, weights) # Calculate predictions based on feature_matrix and weights:
```

```
    errors = predictions - output # calculate the error:


    gradient_sum_squares = 0

    for i in range(len(weights)):

        derivative = feature_derivation(errors, feature_matrix.T[:,i]) # calculate the derivative for weight[i]

        gradient_sum_squares = derivative ** 2 #Gradient Magnitude

        weights[i] = weights[i] - step_size * derivative # update the weight based on step size and derivative

    gradient_magnitude = np.sqrt(gradient_sum_squares)

    if gradient_magnitude < tolerance:

        converged = True

  return(weights)
```

2. **Use the regression_gradient_descent function implemented in Q1 to compute the model coefficients and test error (RSS) for each of the following cases. Calculate training and test RSS for each step of the gradient descent and then plot it for each of the cases given below.**

In this we use our gradient descent function to find out the coeffieicnt and RSS value. As we already divided our dataset into training and testing. Here we calculate the model coefficient and RSS value for the given cases. Where Weight is our output and input_frature attribute are changing for each cases.

Case 1:

Below code is calculate the model coefficient for feature length1 and output weight. This function taking argument from the above created function for gradient descent.

```
Model_coefficient_Length1 = regression_gradient_descent(S_feature_matrix,
output,initial_weights, step_size,tolerance)

print(Model_coefficient_Length1)
```

Output:

```
[-7.30922984 17.19865988]
```

```
test_simple_feature_matrix, train_output = getdata_numpy(Test_data, features,
my_output)

test_predictions = output_prediction(test_simple_feature_matrix,
Model_coefficient_Length1)

print(test_predictions)

test_rss = np.sum((test_predictions - test_output) ** 2)

print(test_rss)
```

output:

```
336.66396773 677.19743332 345.26329766 336.66396773 594.6438659
539.60815429 190.47535876 955.81572335 176.71643086 682.35703128
336.66396773 281.62825611 551.64721621 319.46530785 371.06128748
336.66396773 515.53003046 532.72869034 577.44520602 955.81572335
312.5858439  324.62490581 319.46530785 305.70637994 453.6148549
453.6148549  224.87267852 405.45860724 319.46530785 353.8626276
273.02892618 486.29230867
```

RSS Value for testdata:
1986971.9239113512

train_simple_feature_matrix, train_output = getdata_numpy(Train_data, features, my_output)

train_predictions = output_prediction(train_simple_feature_matrix, Model_coefficient_Length1)

print(test_predictions)

*train_rss = np.sum((train_predictions - train_output) ** 2)*

print(train_rss)

Output:

542.96445977, 406.55260785, 169.67060618, 598.72875277,
185.35557735, 516.5366192 , 304.56828675, 218.83143337,
638.77845635, 293.52592407, 501.44397165, 430.18075035,
238.05444637, 499.83775015, 321.95113703, 121.65772743,
374.60466664, 546.5904277 , 468.68487037, 528.57384469,
491.70832214, 482.98226274, 353.20234238, 276.55346353,
184.22765281, 597.22165695, 273.53748796, 346.07896671,
468.08116347, 817.39240742, 703.50708783, 407.58416946,
175.93611606, 371.79908066, 431.08827334, 1000.0862917 ,
584.9277086 , 422.7933976 , 484.07328971, 349.4213312 ,
183.76486319, 633.33980261, 534.02858202, 727.65511511,
162.16942053, 383.0532143 , 434.86006264, 437.96435685,
298.97007319, 363.0694835 , 634.94779298, 392.43734813,
430.0111836 , 588.76094095, 647.25585225, 582.33312356,
524.58361148, 346.07896671, 618.87499988, 344.93141581,
398.62749278, 357.85025249, 573.49518888, 532.9314784 ,
756.19180581, 399.96836585, 502.22019177, 387.79768109,
321.35861966, 576.69859872, 490.36393136, 163.7832938 ,
560.29117865, 406.70706983, 320.34943211, 471.72027856,
389.60526906, 326.53133368, 690.71641381, 636.24646185,
356.72625322, 672.94082247, 626.77244429, 634.0159333 ,
425.26474915, 686.20922221, 590.82072517, 150.11237492,
228.04992071, 293.47214784, 345.77986095, 456.2509986 ,
309.9313815 , 332.63041201, 707.84429507, 451.57051375,
408.26893073, 543.12716482, 289.06623328, 557.27009521,
547.75819626, 206.94866985, 521.41671168, 665.80430967,
197.3929966 , 371.35866509, 631.89098956, 398.27520208,
214.6414088 , 337.3933326 , 320.9170265 , 501.11247988,

430.18075035, 620.92221821, 502.74753959, 230.21981133,
627.87446203, 881.98443942, 604.67258406, 370.99832507,
535.1143174 , 263.74845976, 445.58840353, 551.51132798,
672.40502278, 190.80718449, 250.91617283])

RSS Value for traindata:
5454820.09853312

We calculate train and test rss for each step of gradient descent. RSS is the difference between prediction output and actual output.

Case 2:

Length1_width_Model_coefficient : [-8.32453822 16.59406427 3.92527246]

Test Prediction:

337.3933326 680.68421247 346.84949769 339.23821066 597.16552555
543.43098091 187.97746058 945.05993488 174.24530745 677.23251661
337.3933326 282.71593752 561.58279708 320.11862609 371.75904288
336.93211309 516.88165065 533.80936308 582.81671713 945.05993488
314.35476603 325.81281506 319.8941005 306.14860144 455.73777174
457.1159349 224.8274758 406.83365736 319.94552157 354.38384939
272.65096266 486.82372311

Test RSS: (2003456.176977193, 1986971.9239113512)

Train Prediction:
[337.3933326 , 680.68421247, 346.84949769, 339.23821066,
597.16552555, 543.43098091, 187.97746058, 945.05993488,
174.24530745, 677.23251661, 337.3933326 , 282.71593752,
561.58279708, 320.11862609, 371.75904288, 336.93211309,
516.88165065, 533.80936308, 582.81671713, 945.05993488,
314.35476603, 325.81281506, 319.8941005 , 306.14860144,
455.73777174, 457.1159349 , 224.8274758 , 406.83365736,
319.94552157, 354.38384939, 272.65096266, 486.82372311]

Train RSS: (5392552.831522416, 5454820.09853312)

Case 3:

Length1,Width, Height Coefficient : [-13.05796564, 12.94243719, 5.16870649, 11.0
3515654]

Test Prediction:

[331.4352749 671.34506115 349.62267606 327.38141248 672.00860791
600.89723128 166.05106259 849.4125522 155.01602228 618.36689112
326.24875132 277.8929028 574.64122044 321.62662317 361.7950777
330.82795188 571.69764629 597.58446291 582.53643831 849.4125522
314.73988979 325.44572012 344.08988 300.88302697 509.37224857
517.55537026 249.2531664 457.53366306 315.24437642 335.06190939
293.43725412 541.86588451]

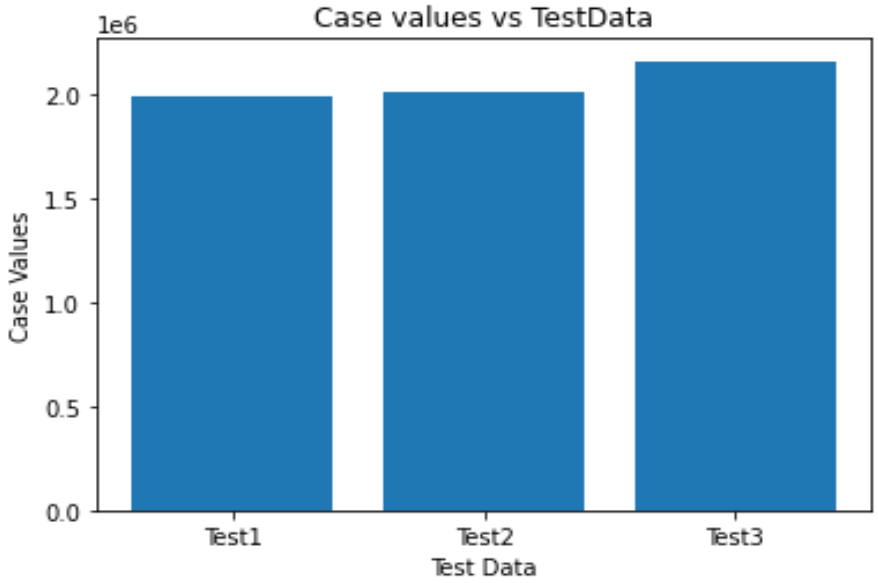
Test RSS:
(2156293.070428636, 2003456.176977193, 1986971.9239113512)

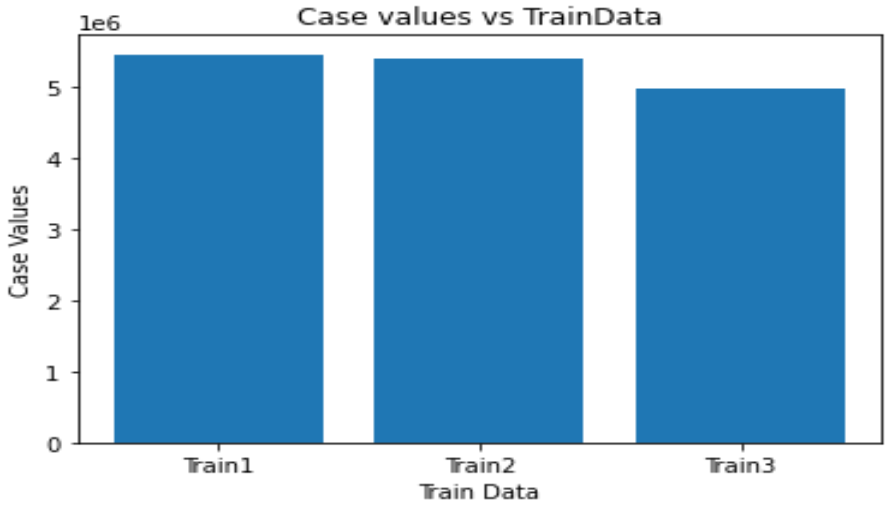
Train Prediction:
[607.9101409 , 444.89248961, 152.90939981, 550.49943087,
164.74745988, 572.09964514, 292.90271105, 200.16875297,
711.50855453, 321.12672821, 566.65181645, 417.17745545,
256.56073825, 559.20018005, 319.75549621, 114.5941026 ,
375.92702747, 613.13247105, 523.73574497, 577.4825191 ,
537.97879224, 538.28159276, 348.19943966, 273.54435691,
164.25789301, 671.56276454, 271.40789375, 345.94417219,
508.20996745, 742.46088621, 695.95252642, 418.76004371,
155.19292421, 374.64195899, 419.64812448, 908.51986535,
527.50949315, 411.02208576, 538.42793511, 339.30738256,
163.64850252, 624.25144602, 481.69524122, 657.21286113,
144.12343444, 375.27703528, 439.3328467 , 430.18077148,
289.70796436, 353.7034728 , 641.0072316 , 435.10978046,
431.49025515, 585.40191703, 706.15997334, 578.8991946 ,
585.91105092, 335.61526567, 697.3010114 , 334.42671072,
390.77425981, 394.42529829, 663.8361115 , 587.77279474,
679.30018858, 406.81717548, 501.09081435, 374.71791051,
314.62220053, 586.86585957, 508.28026798, 142.66955703,
629.87106421, 457.13000623, 354.16202832, 455.44412614,
428.39502501, 322.52815372, 688.39873047, 639.7490751 ,
350.29354548, 603.0017274 , 620.96630115, 632.59850081,
429.72552252, 682.34487337, 584.31781209, 131.9093673 ,
207.61798676, 289.97429503, 343.80565851, 442.75648609,
340.23583536, 366.963112 , 632.63507264, 493.77424685,
410.69978412, 610.33391194, 287.5094777 , 622.67209688,
547.29722003, 197.98780946, 523.47426425, 668.45003897,
175.16343736, 371.39814334, 632.88283059, 385.40111708,

211.38710157, 328.84201311, 314.04072105, 548.494775 ,
415.90179135, 613.26286099, 454.10638716, 220.31640031,
635.81380571, 813.97665929, 544.00657997, 366.75567746,
593.21551722, 255.97134651, 490.83484683, 498.33051034,
607.31719519, 171.74306836, 245.35841303])

Train RSS:
(4971406.468135323, 5392552.831522416, 5454820.09853312)

This bar graph is comparing the Test and train Rss values for the three different cases.When input feature is increasing train data error is decreasing and test error is increasing.





3. **Use in-built linear regression functions of Scikit Learn library to compute higher polynomial regression models for degrees 2, 3, 4, 5 and 6. Use 'Lenght1' as the input feature and 'Weight' as output. For each of the model, compute the RSS (on the train and test dataset), and plot the model through the training data.**

In Regression our main aim is to plot best fit line our the data. Polynomial regression is mainly finding the relation between the data point and find the best fit line. Here we have to create polynomial regression model for the degree 2,3,4,5 and 6. As the degree is increasing model try to give best fit line. Here we also calculated values Train and Test RSS.

Below function takes the degrees for the polynomial features.

```
Test_RSS={}

Train_RSS={}

def perform_polynomial_structure(features, poly_degree):

 polynomial_features = PolynomialFeatures(poly_degree)

 features_poly = polynomial_features.fit_transform(features.reshape(-1,1))

 return features_poly
```

Now next step is to calculate the rss values for the each polynomial regression model. As we know rss is the sum of actual output minus predicated output.

```
def calculate_rss_value(output_train_poly_prediction, output_test_poly_prediction,
output_train, output_test, poly_degree):

    Train_RSS[poly_degree] = np.sum((output_train-output_train_poly_prediction)**2)

    Test_RSS[poly_degree] = np.sum((output_test-output_test_poly_prediction)**2)
```

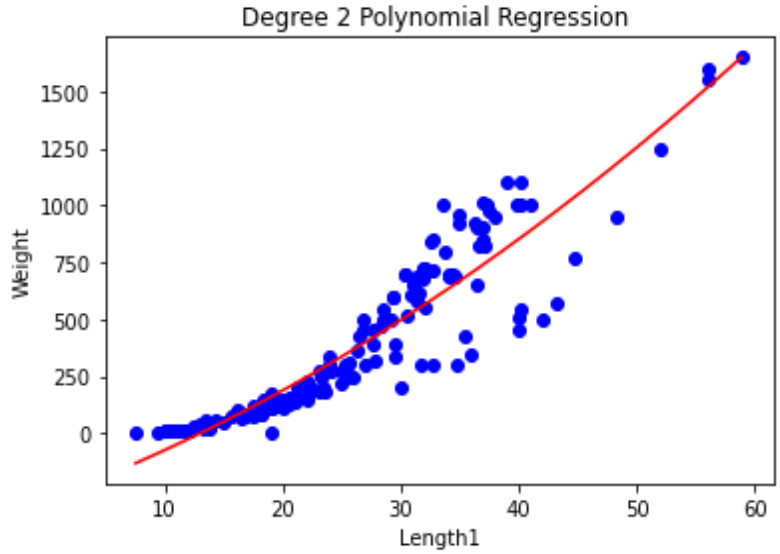
We perform polynomial regression to fit the best fit line for the data.

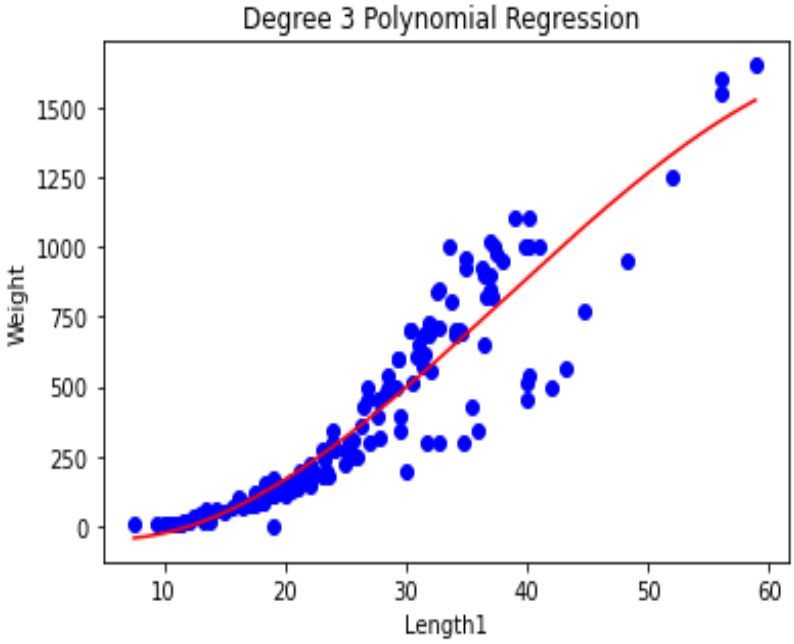
```
def perform_polynomial_regression(train_features,test_features, output_train,
output_test, poly_degree):

    features_poly = perform_polynomial_structure(train_features, poly_degree)

    model = LinearRegression()
```

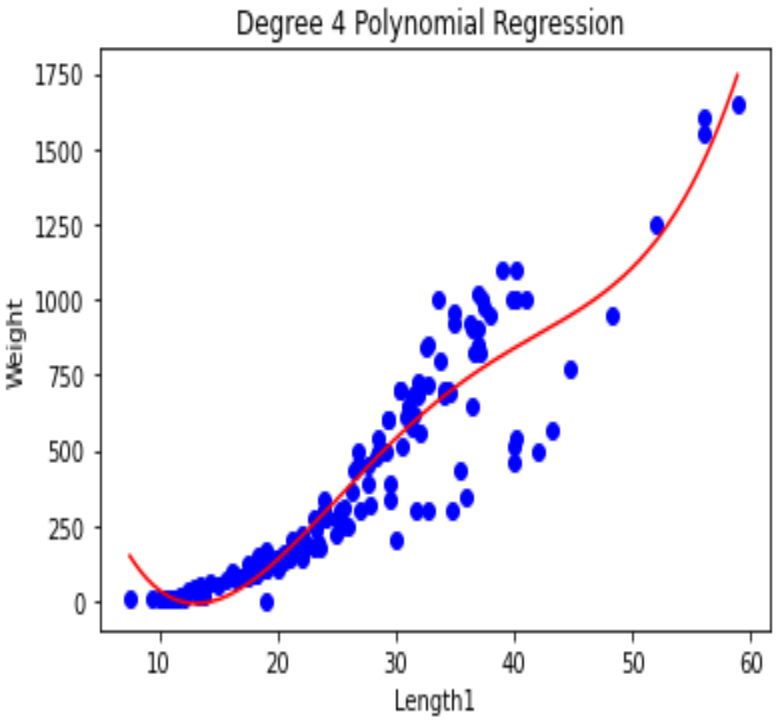
model.fit(features_poly, output_train)

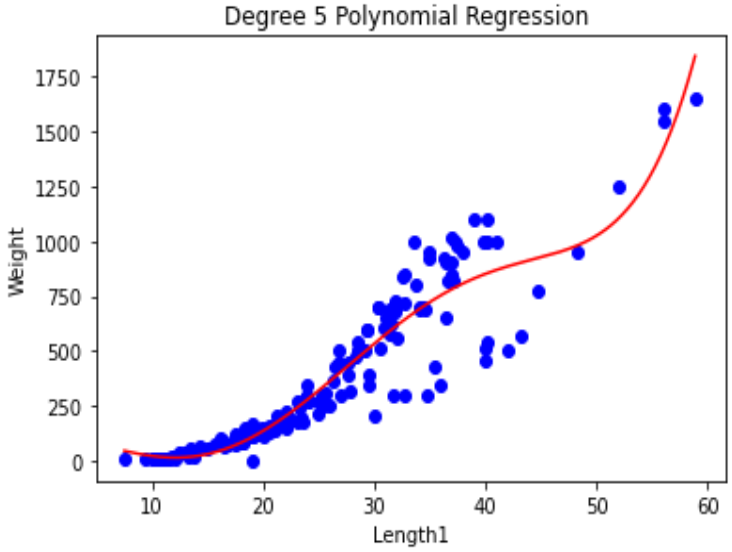
predict_output_values(train_features, test_features, model, poly_degree, output_train,output_test)

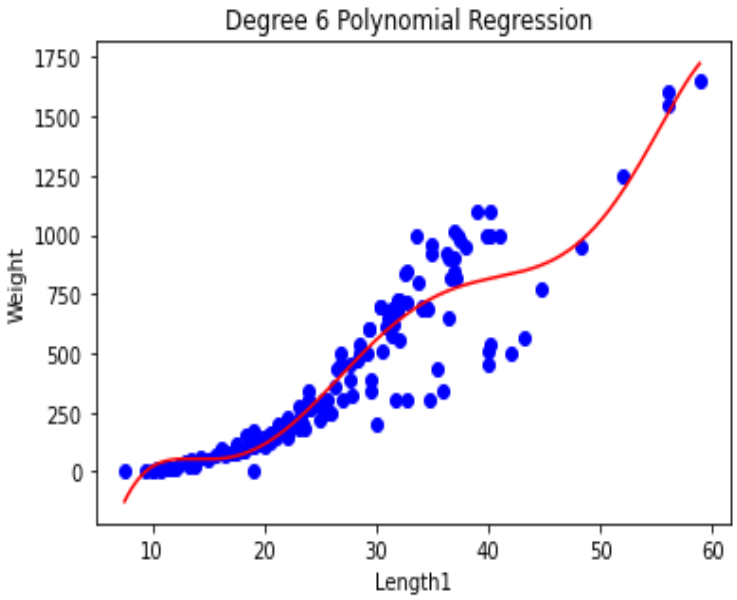
Below is the graph for polynomial regression degree 2,3,4,5 and 6.






Degree 4 Polynomial Regression
Weight
Length1
1750
1500
1250
1000
750
500
250
0
10
20
30
40
50
60


Degree 5 Polynomial Regression
Weight
Length1
1750
1500
1250
1000
750
500
250
0
10
20
30
40
50
60



4. **From all the models in Q2 and Q3, which model is the best and why? What criteria you will use**?

   When we are working with regression our main aim is to draw a best fit line our the data point. As per my research and implementation, when we got the minimum train error and test data that will be the best model. Moreover, in best model there is low bias and low variance. In polynomial regression, model is dependended on the number of degrees. Higher the number of degree model can give more accurate result.

   As per my opinion, degree 6 graph is best because there is low train and test error. Bias is depended on the training dataset and variance is depended on test dataset. In polynomial regression, initially we are using less training data to build our model and that means there is high bias and low variance and model is underfitting. Then we increasing our polynomial degree value that means model used more data in training in compression of testing. So initially, training error is high and testing error. There is comes a point where in the model less training error and training error and where model try to fit the best fit line. After that train error is decreasing and test error is increasing and it creates overfitting problem. To stop overfitting, we are using more data points to build our model.

   In gradient decent algotihm there is more error in comparison of polynomial regression. So,Here degree 6 model is giving more accurate result in comparison of other model.

References

[1] desicochrane/ml-regression-uni-washington. (2020). Retrieved 29 October 2020, from https://github.com/desicochrane/ml-regression-uni-washington/blob/master/Week_2/Week2_Assignment2.ipynb

[2] Polynomial Regression, Retrived From https://towardsdatascience.com/polynomial-regression-bbe8b9d97491