Team name:
Team Speedwagon

Team members / email / contribution:
Andy Nguyen / andy_nguyen@csu.fullerton.edu / 33.3%
Rocio Salguero / salgueroroci@csu.fullerton.edu / 33.3%
Annie Chen / ann13ch3n@csu.fullerton.edu / 33.3%

Project #1 Report

For project #1, our team decided to develop predictive models of the given data set using Python language. We implemented our linear regression models using the scikit-learn library. Scikit-learn is a free machine learning library for the Python language. It was the ideal library to use for this project, since the library has built-in modules for linear modeling and computing performance metrics of the models. The scikit-learn library's linear regression computing is done by the least squares approach. Other packages we needed were matplotlib, seaborn, pandas, numpy, and math.

As we learned in class, the best model for a data set is a model that can accurately predict data outside of the training data well. In order to create a model that is trained well with the training data, while at the same time, being able to accurately predict new, unseen data (testing data), we decided to implement our modeling using the k-fold cross-validation method. With this method, the data set is split k times, where (k-1)/k of the data set will be used to train the model, while the remaining 1/k will be used for testing. We set k in the k-fold cross-validation to 10, ensuring the entire data set is fully utilized and creates the most accurate model.

In figures 1 through 6, the x-axis represents the actual values from the data set, while the y-axis represents the predicted values based off of the models. The blue line through the plot is where the scattered points would fall if the model predicted the actual values with 100% accuracy. Since we predicted both linear and non-linear models, we incremented the order of the features, Tm, Pr, Th, and Sv, up to the 6th order. As we learned from lecture, we want to make sure that there is a balance between generalization and overfitting of our models. We can see from figures 1 through 4 that the models become more accurate in predicting the actual Idx values as the polynomial order increases, being the most accurate in the 4th order. Starting at the 5th order, we see the prediction accuracy becomes much more inaccurate as order increases. In figures 7 and 8, the same trends are shown in the error measures and variance graphs. The error measurements, mean absolute error (MAE), mean squared error (MSE), and root mean squared error (RMSE) all decrease to the 4th order and increase starting at the 5th order. Variance also is lowest at the 4th order and becomes higher as order increases.

Thus, we can conclude that the 4th order polynomial model is the most balanced model, neither too general nor over-fitted. The 1st order, or linear, model yielded a R-squared value of 0.9698, meaning that 96.98% of the variation between the predicted and actual Idx value is explained by the model. From training models using the various polynomial orders, we saw that the 4th order model was the best non-linear model, with an R-squared value of roughly 0.9978.

Best models in standard math equations:

Order 1/Linear:

$Y = -0.54518287\ X_1 + -0.00155211 * X_2 + 3635.78620699 * X_3 + 0.05434339 * X_4 - 78.0742448822$

Order 4/Non-linear:

$Y = -0.06804886 * 1 + 0.34999684\ X_1 + 0.03684389 * X_2 + 0.00613381 * X_3 + -0.42246870 * X_4 + 0.01246965\ X_1^2 + -0.00005838\ X_1 * X_2 + 0.00272180\ X_1 * X_3 + -0.00565905\ X_1 * X_4 + (...) + 0.0000000004720326601814 76\ X_4^4 + 37.5339624054$

The 4th order equation was shortened due to having 70 coefficients. See figure 9 for all values.

Y = Chemical Index, X_1 = Temperature , X_2 = Pressure, X_3 = Thermal Conductivity, X_4 = Sound Velocity

Figures:

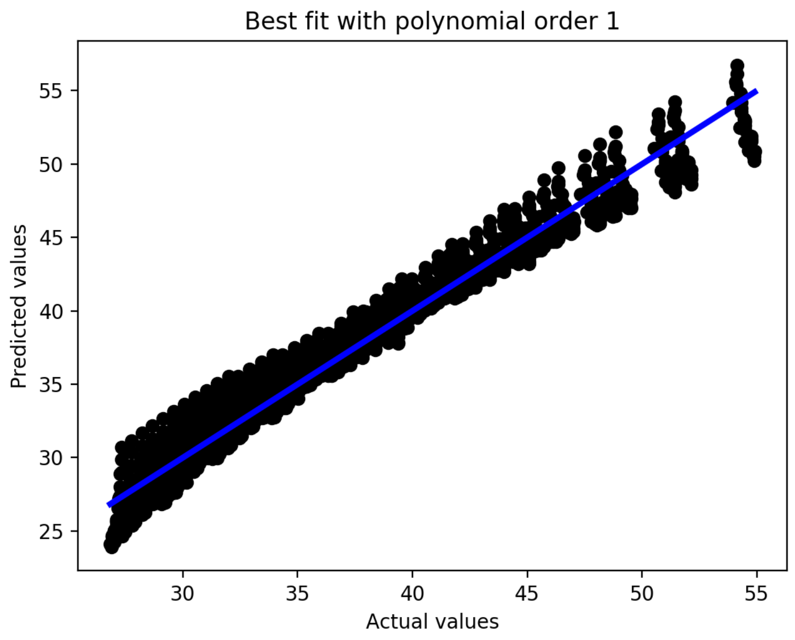


Figure 1: Best fit with polynomial order 1

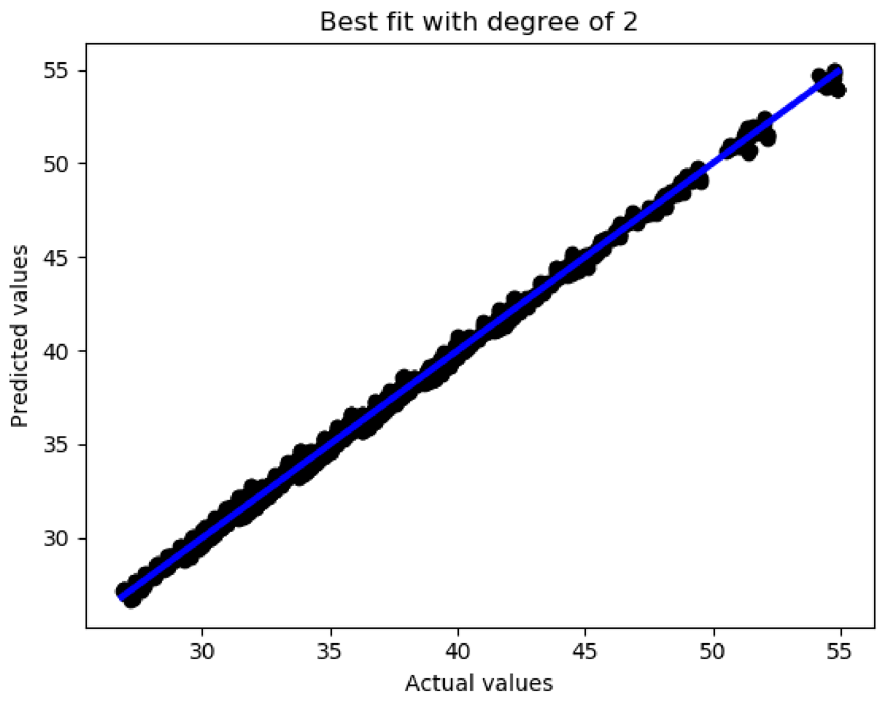


Figure 2: Best fit with polynomial order 2

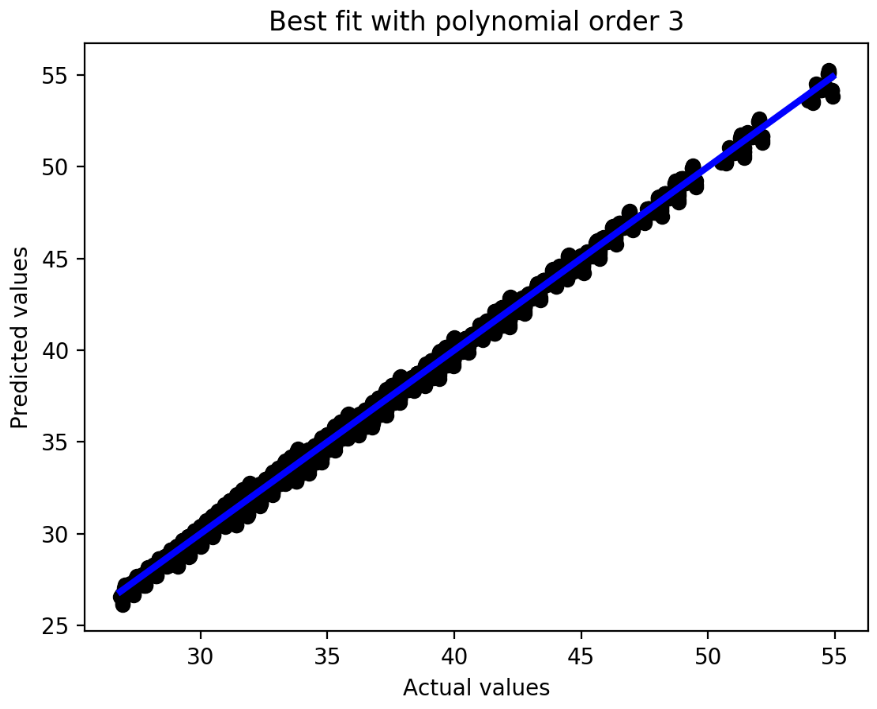


Figure 3: Best fit with polynomial order 3

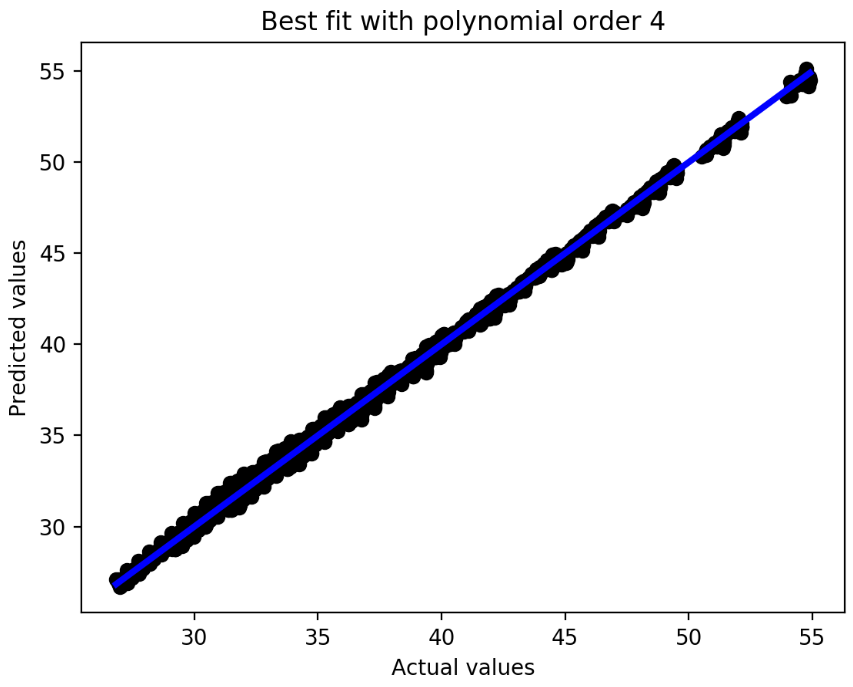


Figure 4: Best fit with polynomial order 4

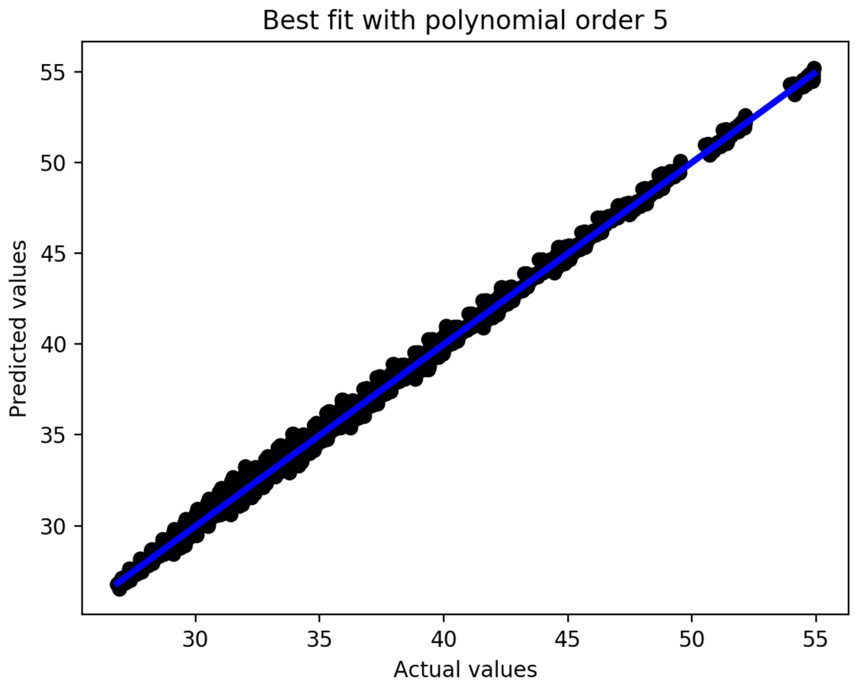


Figure 5: Best fit with polynomial order 5

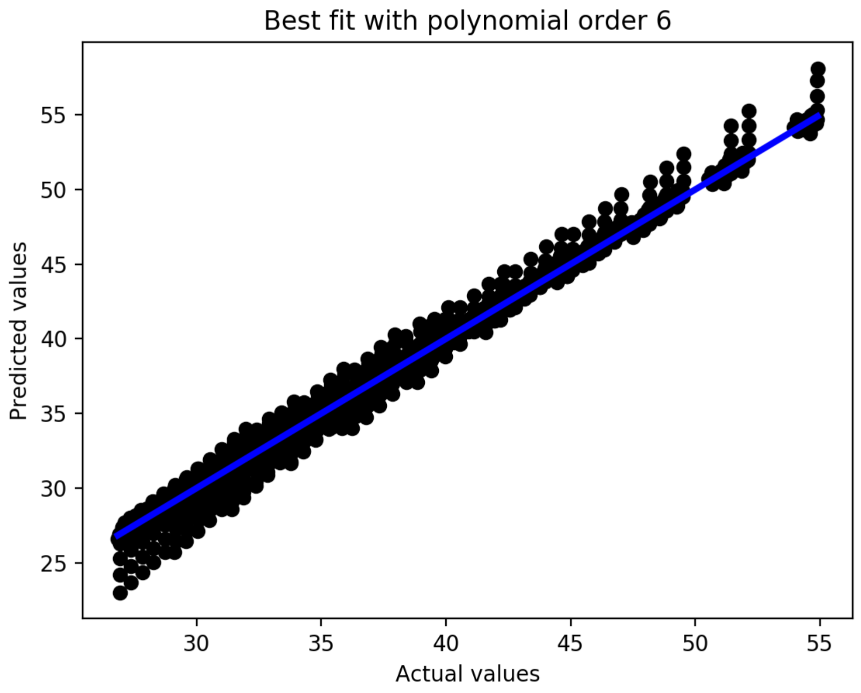


Figure 6: Best fit with polynomial order 6

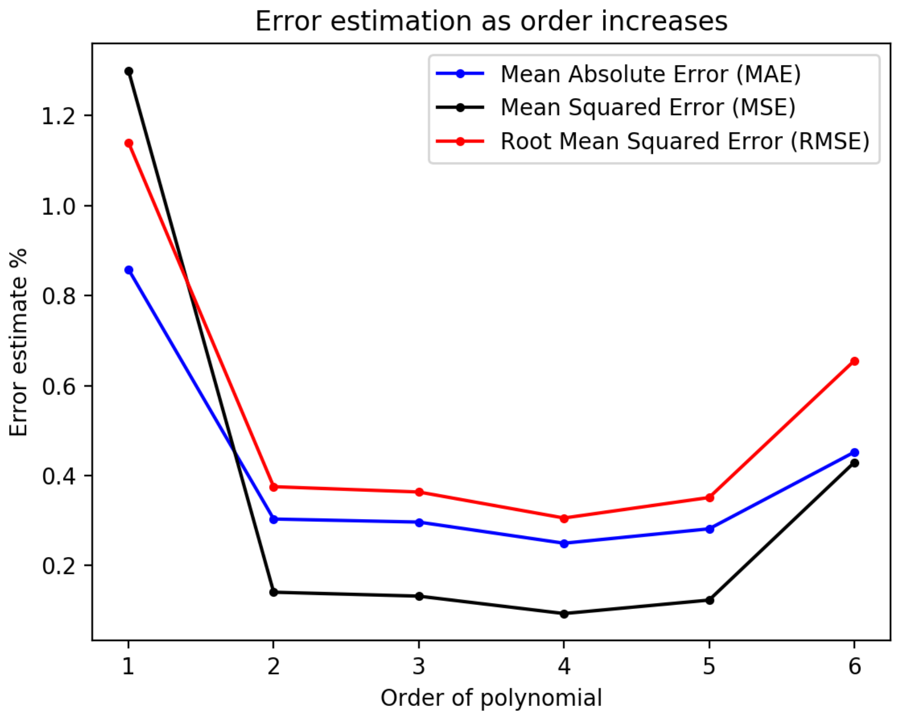


Figure 7: Error estimation as order increases

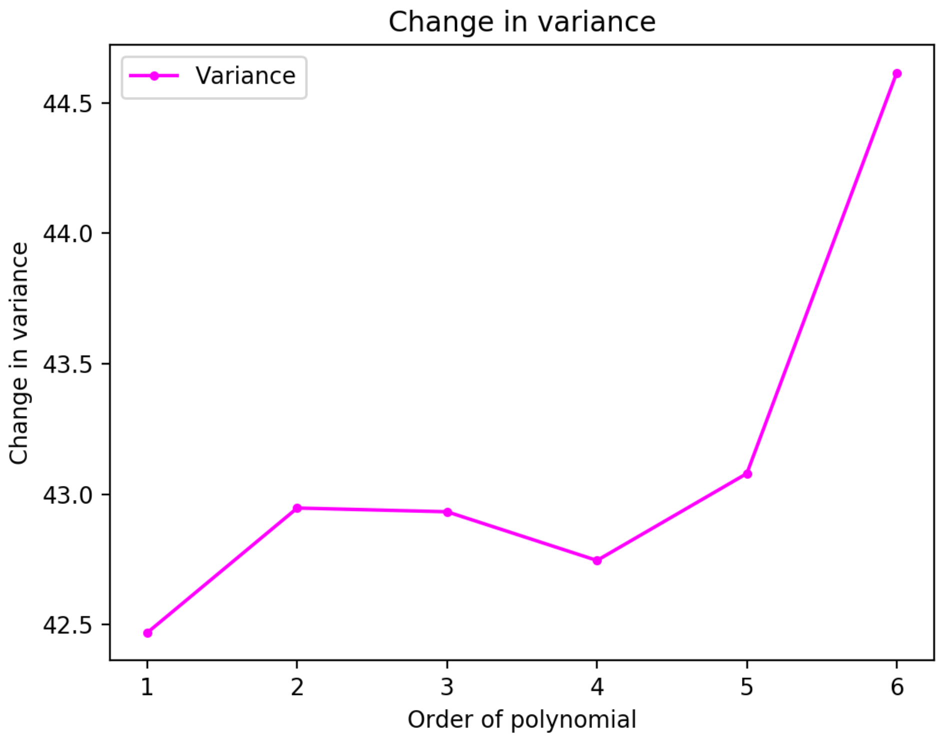
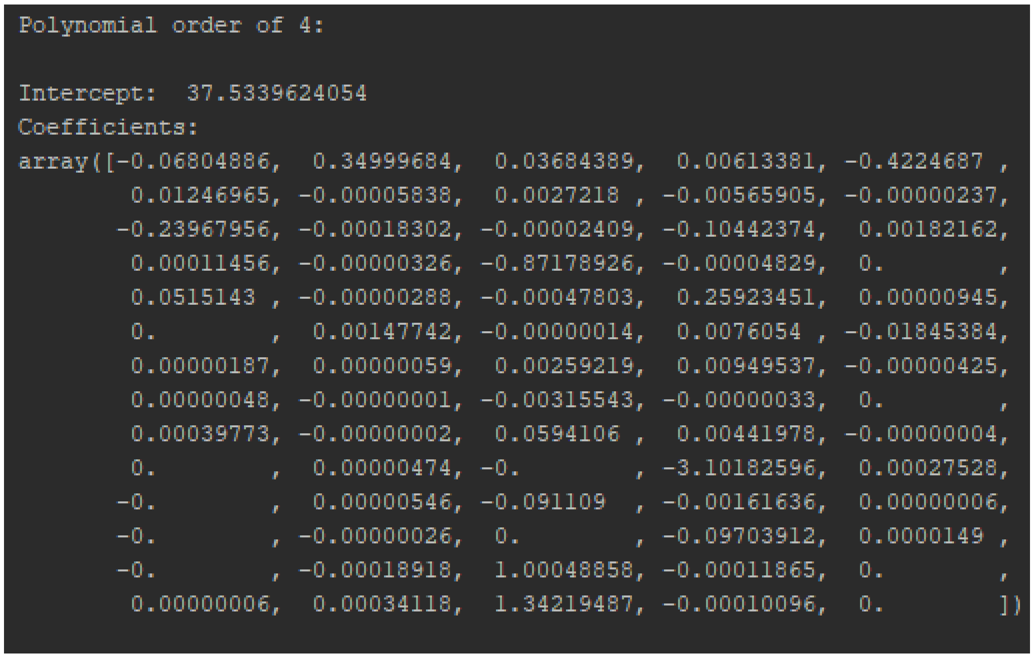


Figure 8: Change in variance



```
Polynomial order of 4:

Intercept:  37.5339624054
Coefficients:
array([-0.06804886,  0.34999684,  0.03684389,  0.00613381, -0.4224687 ,
        0.01246965, -0.00005838,  0.0027218 , -0.00565905, -0.00000237,
       -0.23967956, -0.00018302, -0.00002409, -0.10442374,  0.00182162,
        0.00011456, -0.00000326, -0.87178926, -0.00004829,  0.        ,
        0.0515143 , -0.00000288, -0.00047803,  0.25923451,  0.00000945,
        0.        ,  0.00147742, -0.00000014,  0.0076054 , -0.01845384,
        0.00000187,  0.00000059,  0.00259219,  0.00949537, -0.00000425,
        0.00000048, -0.00000001, -0.00315543, -0.00000033,  0.        ,
        0.00039773, -0.00000002,  0.0594106 ,  0.00441978, -0.00000004,
        0.        ,  0.00000474, -0.        , -3.10182596,  0.00027528,
       -0.        ,  0.00000546, -0.091109  , -0.00161636,  0.00000006,
       -0.        , -0.00000026,  0.        , -0.09703912,  0.0000149 ,
       -0.        , -0.00018918,  1.00048858, -0.00011865,  0.        ,
        0.00000006,  0.00034118,  1.34219487, -0.00010096,  0.        ])
```

Figure 9: Output of Intercept and Coefficients