Bias-Variance-Tradeoff

Calculation of the bias and variance after training two models.

Getting Started

Unzip the folder, run the scripts and view the saved tables and graphs for question 1 and 2 in the current directory.

```
$ unzip 33_assgn1.zip
$ cd 33_assgn1
$ python3 q1.py  # Graph and table will be displayed and saved in current
directory
$ eog graph1.png
$ eog table1.png
$ python3 q2.py  #Graph will be displayed and saved in the current
directory
$ eog graph2.png
$ eog table2.png
```

Alternately,

```
$ unzip 33_assgn1.zip
$ cd 33_assgn1
$ make all
$ make seeOne  # Data visualization for question 1
$ make seeTwo  # Data visualization for question 2
$ make clean
```

Libraries Used

```
# For data handling
import pickle
import numpy as np
import pandas as pd
# To generate training and testing data splits
from sklearn.model_selection import train_test_split
# For data visualization
import matplotlib.pyplot as plot
# To generate polynomial data and train the models
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
```

Question 1

Data Resampling

- 5000 entries of the form (x_i , y_i) are present are loaded using the `pickle` library, in numpy format.

```
file = open('Assignment/Q1_data/data.pkl', 'rb')
data =  pickle.load(file)
file.close()
```

- 90:10 split of data as training set and testing set is done once using the `train_test_split` from `sklearn`.

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.1)
```

- The training data is then reshuffled using the `np.random.shuffle` function.

```
training_data = np.column_stack((X_train, Y_train))
np.random.shuffle(training_data)
X_train = training_data[:, 0]
X_train = X_train[:, np.newaxis]
Y_train = training_data[:, 1]
```

- The training data is then split into 10 subsets using the `np.array_split` funtion. This is done because inorder to calculate the variance we need multiple realisations of the model. An easy way to achieve this is to randomly divide the training set (if feasible) into multiple subsets (here, `10`), so that we have `10` different realisations of model.

```
X_train_split = np.array_split(X_train, 10)
Y_train_split = np.array_split(Y_train, 10)
```

- `PolynomialFeatures(i)` is used to raise `x` data points to powers from 0 to `i` to generate data for polynomials of degree `i`.

Bias - Variance Tradeoff

> ***Variance*** is the variability of a model prediction for a given data point.

- We have repeated the entire model building process `10` times for each polynomial. The variance is how much the predictions for a given point vary between different realizations of the model.

This is calculated and averaged out for each polynomial as follows,

```
np.mean(np.var(poly_prediction, axis = 0))
```

- Here, `poly_prediction` is a [10, 500] matrix containing the predicted outputs using the 10 models for each data point.
- `axis = 0` specificies that we traverse the matrix column-wise to obtain different model predictions for a single test data point.

> ***Bias*** is the difference between the average prediction of our model and the correct value which we are trying to predict.

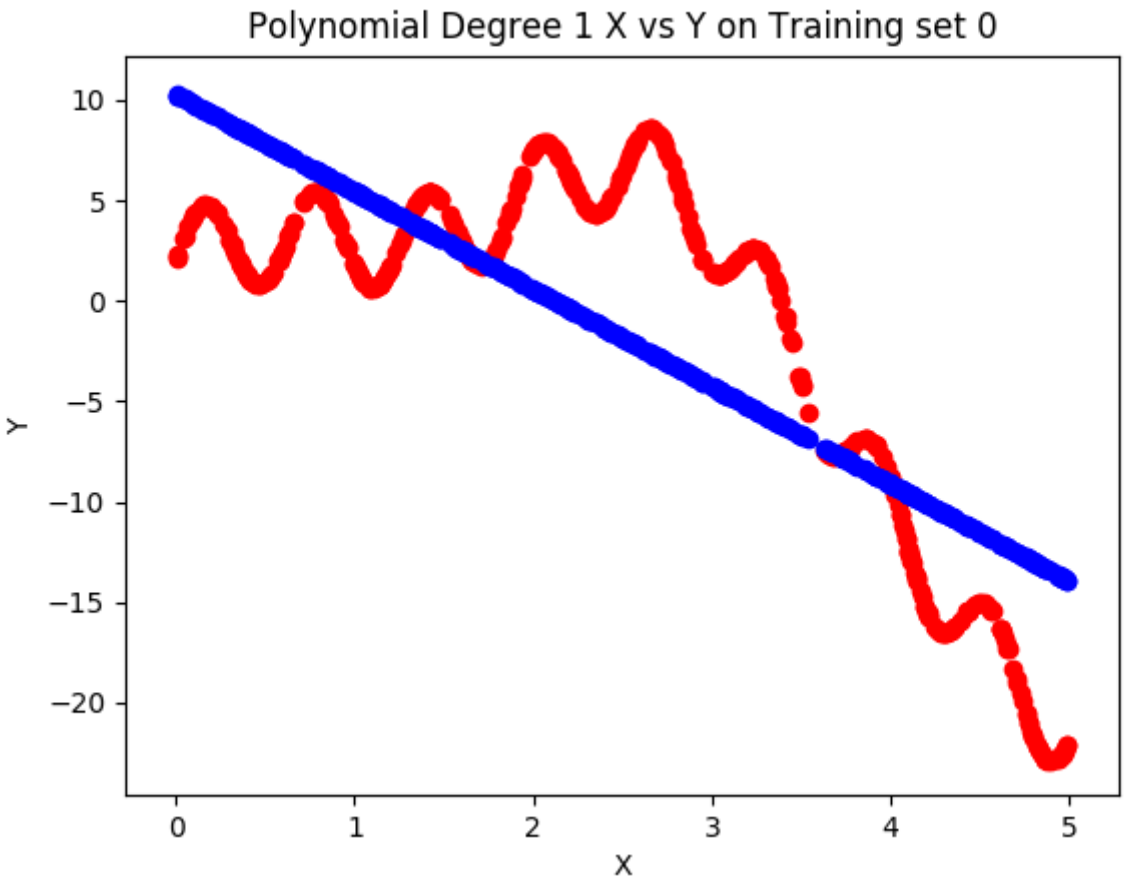
This is calculated and averaged out for each polynomial as follows,

```
(np.mean(poly_prediction, axis = 0) - Y_test)**2
```

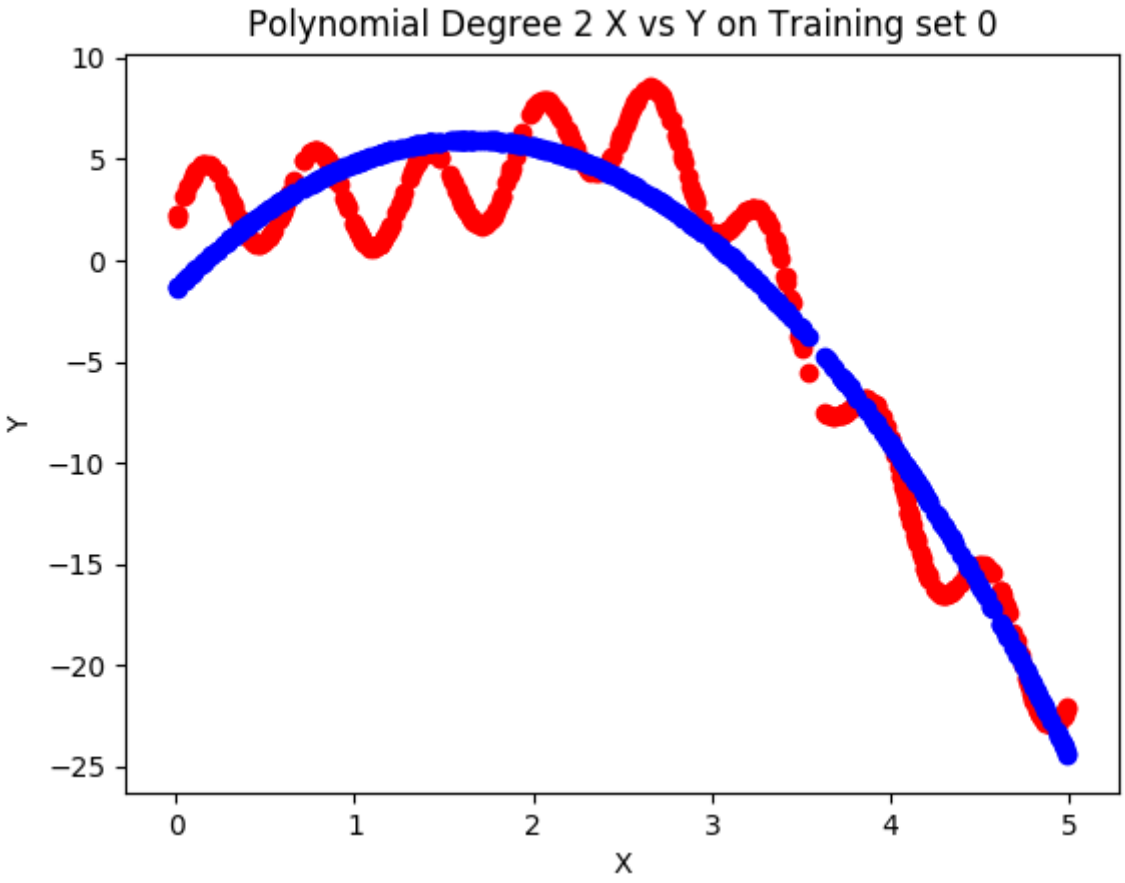
- Here, `Y_test` represents the correct output value for the input data point.

Listed are the plots for the models obtained using the `LinearRegression()` function using the first randomly generated test set for the respective polynomials.

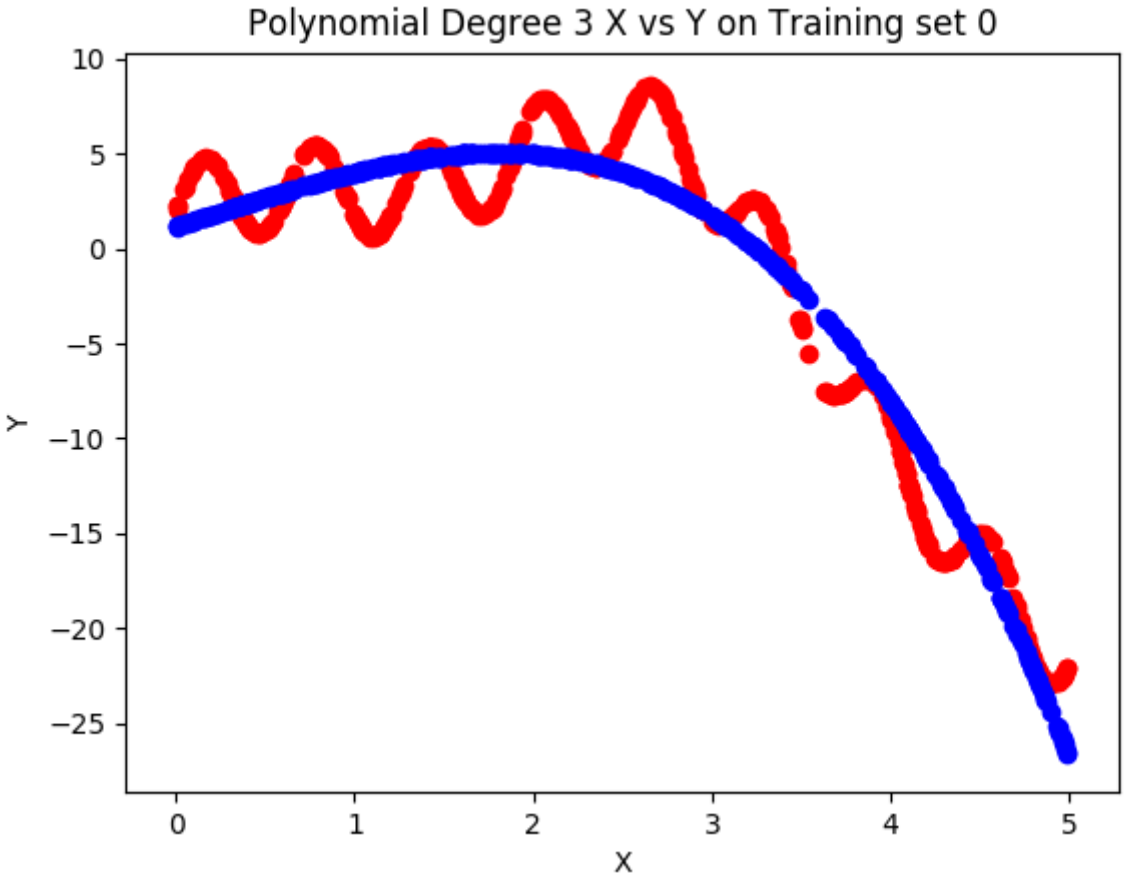
y = mx + c



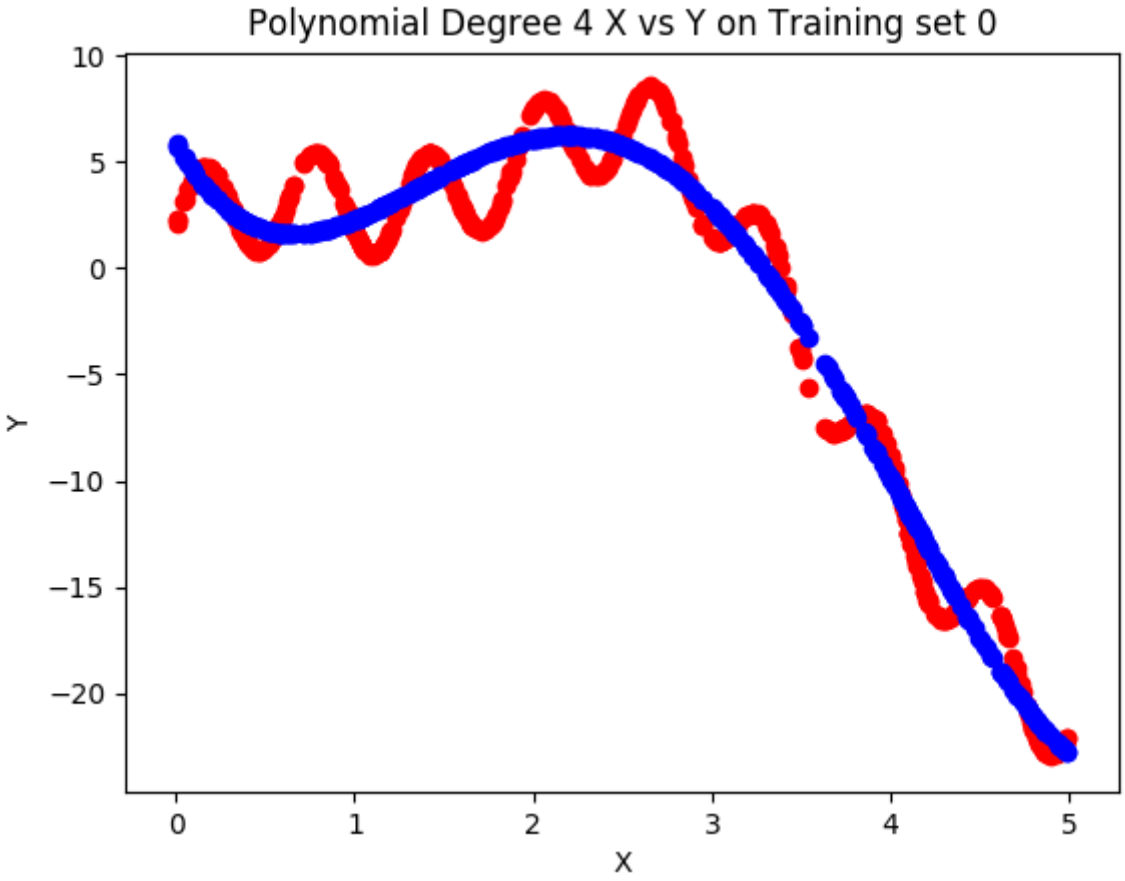
$y = ax^2 + bx + c$



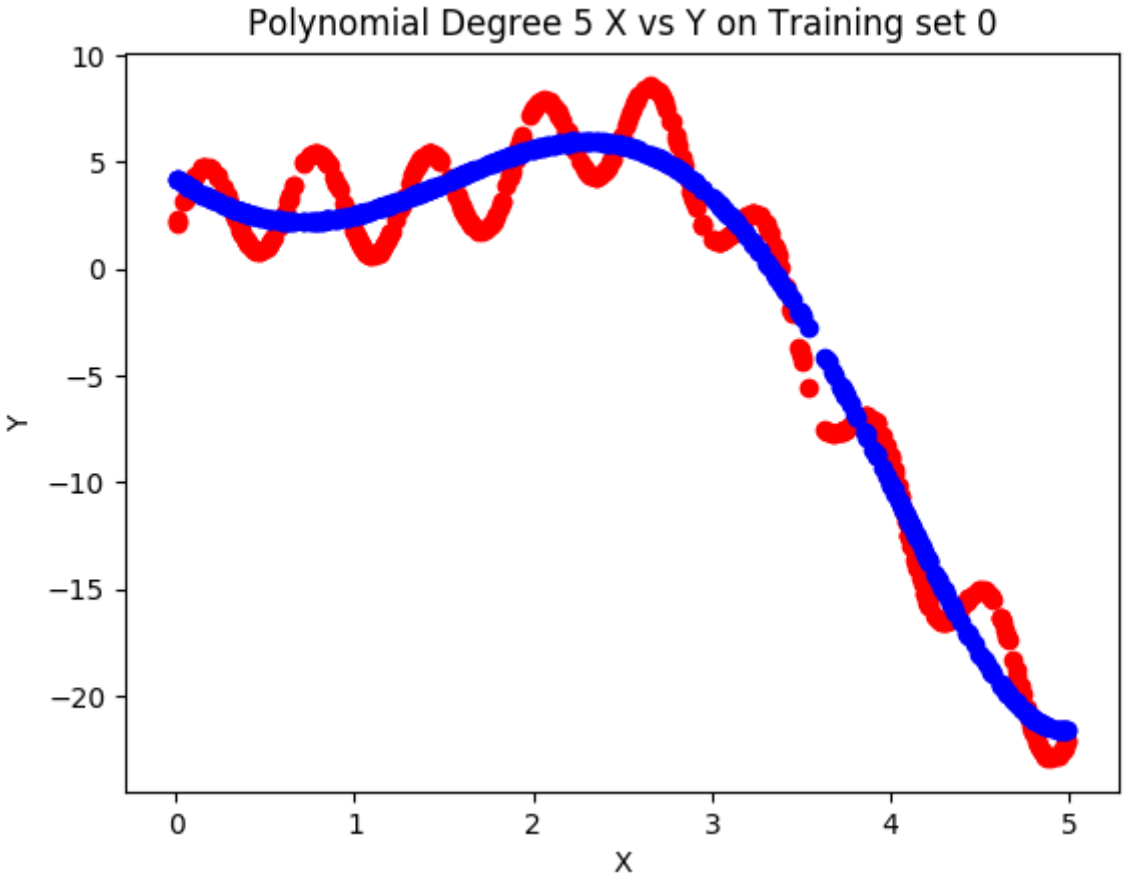
$y = ax^3 + bx^2 + cx + d$



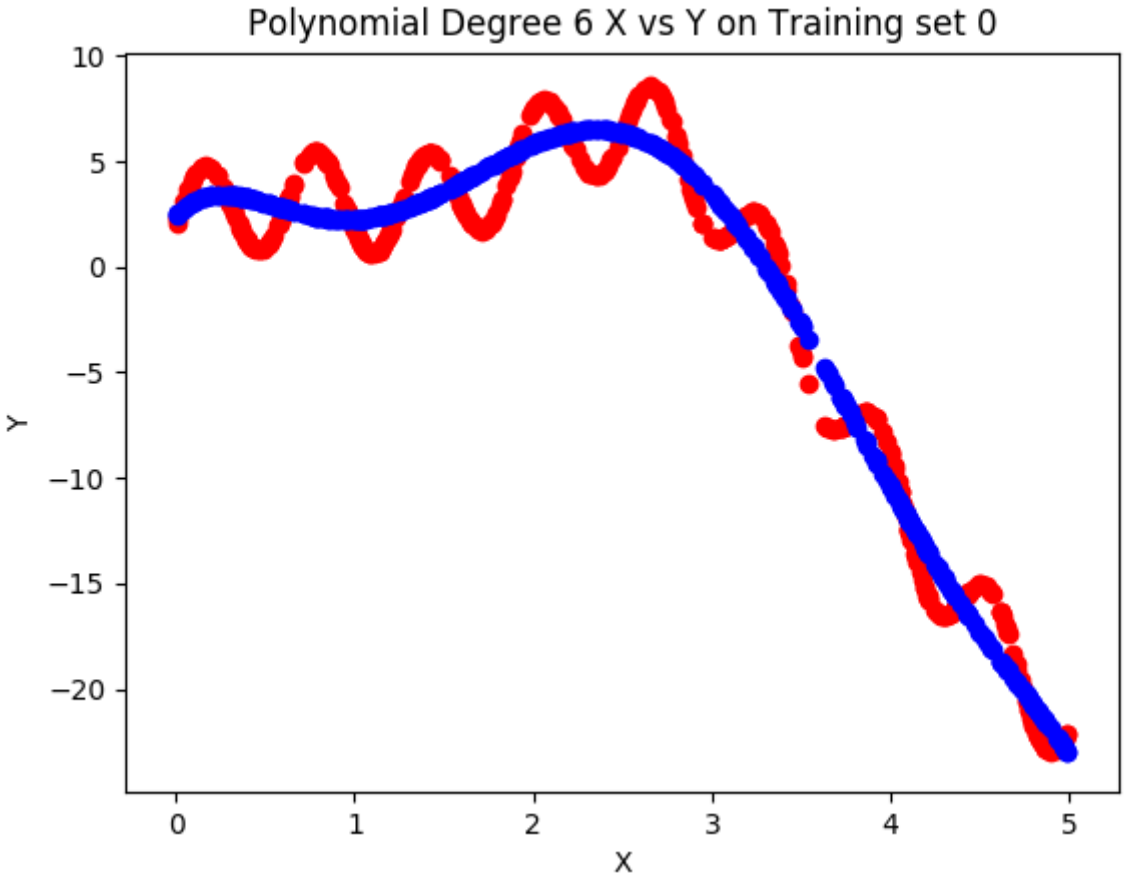
$y = ax^4 + bx^3 + cx^2 + dx + e$



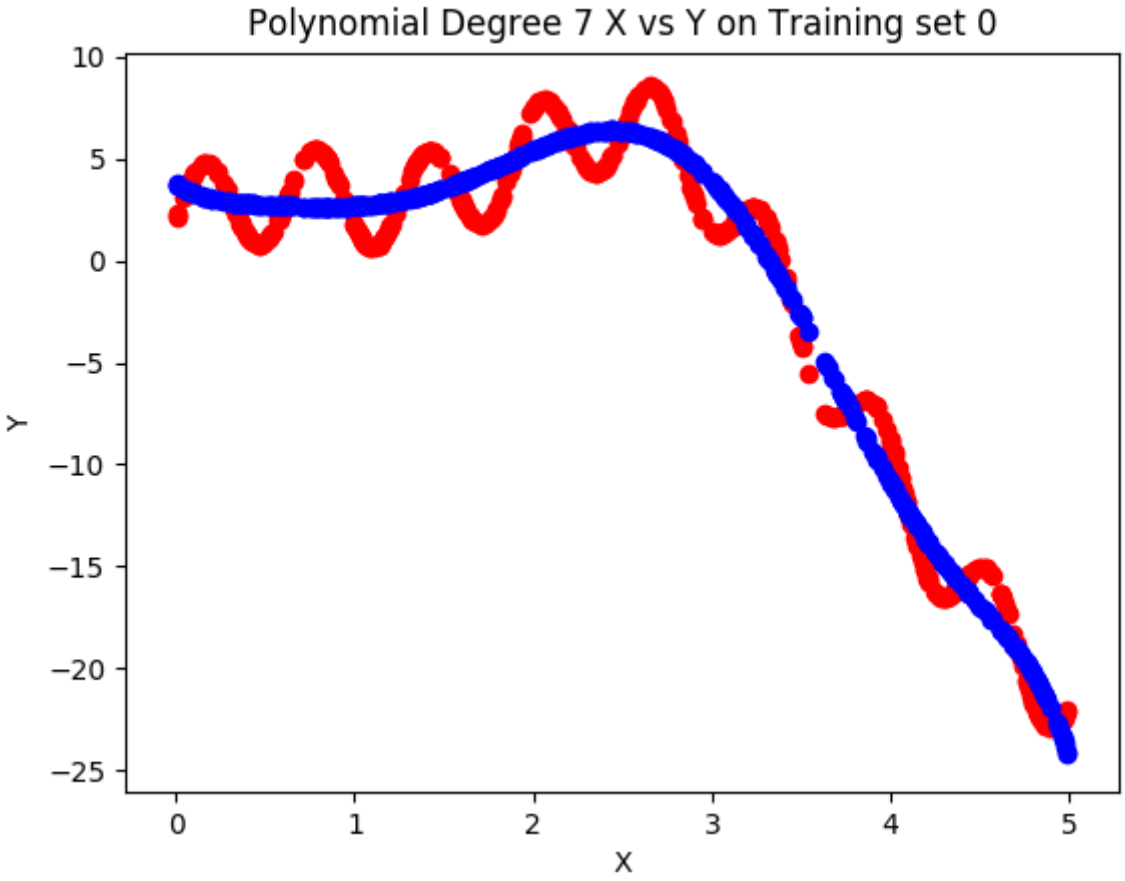
$y = ax^5 + bx^4 + cx^3 + dx^2 + ex + f$



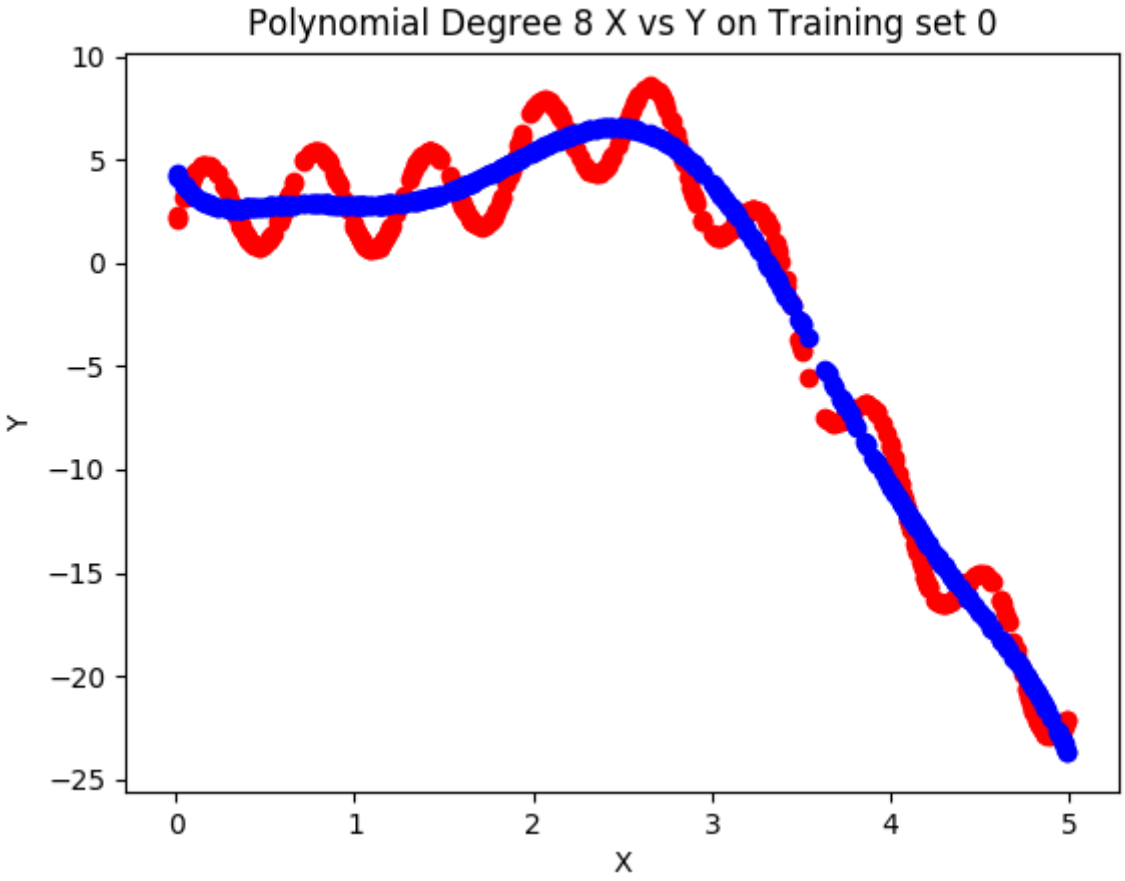
$y = ax^6 + bx^5 + cx^4 + dx^3 + ex^2 + fx + g$



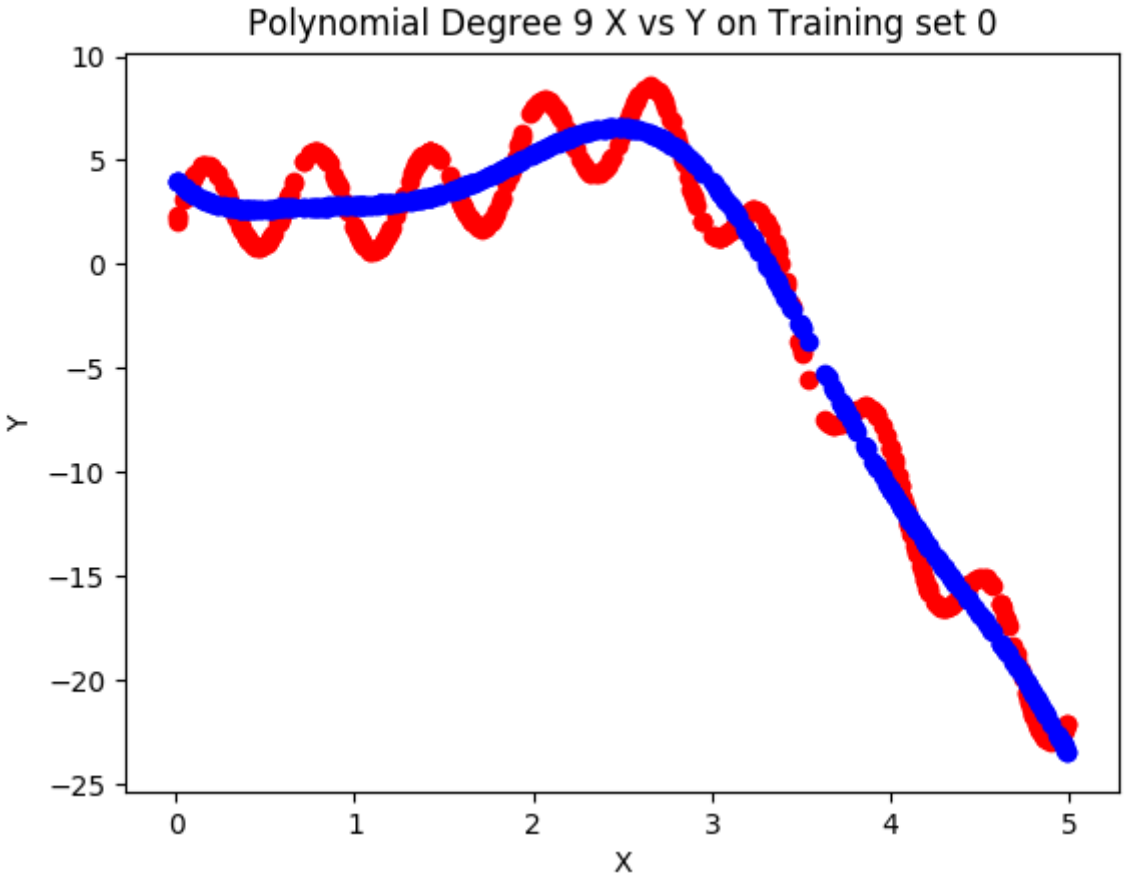
$y = ax^7 + bx^6 + cx^5 + dx^4 + ex^3 + fx^2 + gx + h$



$y = ax^8 + bx^7 + cx^6 + dx^5 + ex^4 + fx^3 + gx^2 + hx + i$



$y = ax^9 + bx^8 + cx^7 + dx^6 + ex^5 + fx^4 + gx^3 + hx^2 + ix + j$



Observervations

Here, it can be seen that the bias decreases as the complexity of the polynomial increases and the variance shows a slight increase as the degree of the polynomial increases.

This is in accordance with the Bias-Variance tradeoff where an optimum model is reached when total error is minimised.

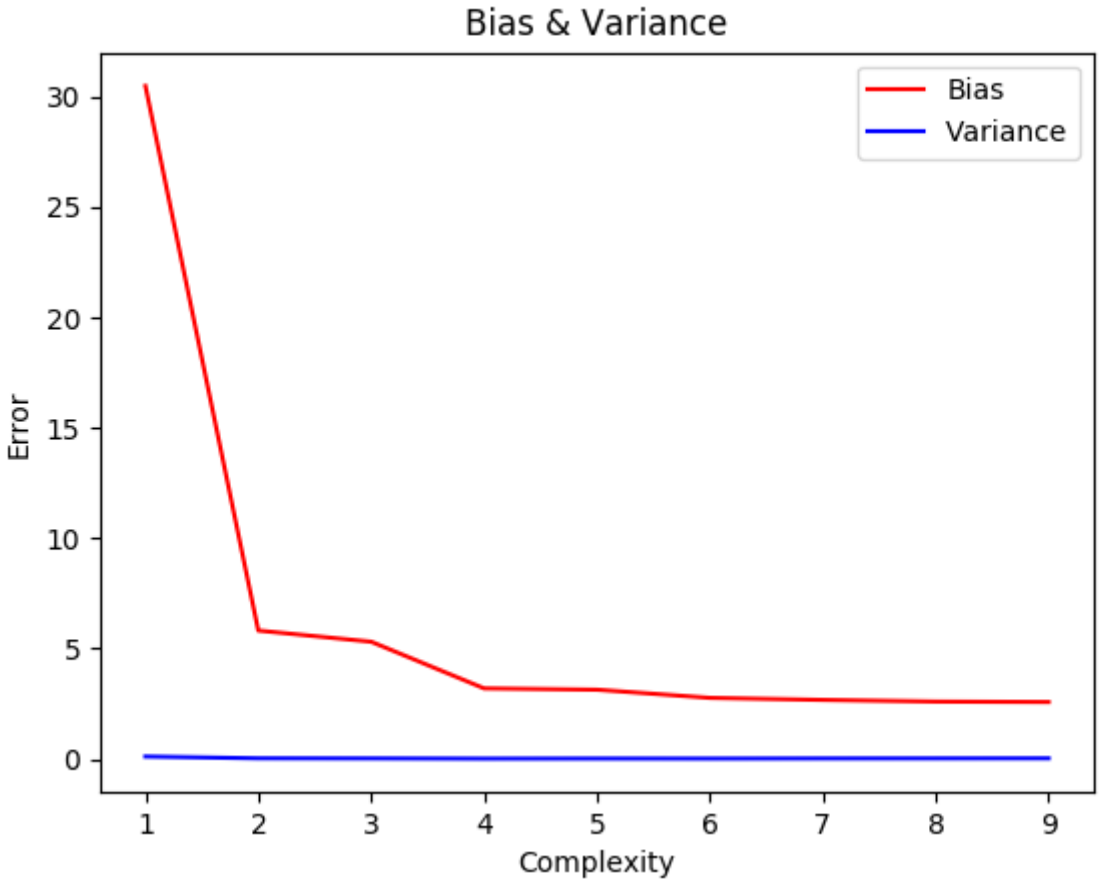
When our polynomial is too simple and has very few parameters then a high bias and low variance is observed. On the other hand, if our model has a large number of parameters then high variance and low bias is observed.

Tabulated Values

Bias & Variance

Degree	Bias	Bias^2	Variance
1.0	4.672592348241717	30.482826711097516	0.11812562043680355
2.0	1.9570501419909037	5.822203027204497	0.03814271938978563
3.0	1.8895720897934591	5.316436843982551	0.03340133186086311
4.0	1.5145105304789224	3.2037100641184426	0.02307642164845536
5.0	1.5302492631109714	3.1443310099139863	0.026953411970007644
6.0	1.4747381456739168	2.774018642981857	0.025060796485316252
7.0	1.46965330295769	2.6821348174835316	0.03194337063913668
8.0	1.4550045748222231	2.606215565346823	0.03473347132001201
9.0	1.446712891852262	2.5868953681079794	0.03701619068252883

Bias^2 Versus Variance Graph



Question 2

Data Resampling

- 20 subsets of 400 entries of X datapoints each are loaded as the training set and 1 set of 80 entries of X datapoints is loaded as the testing set using the `pickle` library, in numpy format. Corresponding Y sets are loaded as well.
- The training data has 20 subsets. This is now used to calculate the variance by generating multiple realisations of the model.
- Now, the outer loop selects the degree of the polynomial and the inner loop chooses the training set and a model is generated using the

Bias - Variance Tradeoff

> ***Variance*** is the variability of a model prediction for a given data point.

- We have repeated the entire model building process `20` times for each polynomial. The variance is how much the predictions for a given point vary between different realizations of the model.

This is calculated and averaged out for each polynomial as follows,

```
    np.mean(np.var(poly_prediction, axis = 0))
```

- Here, `poly_prediction` is a [20, 80] matrix containing the predicted outputs using the 20 models for each data point in the testing set of 80 points.
- `axis = 0` specificies that we traverse the matrix column-wise to obtain different model predictions for a single test data point.

> ***Bias*** is the difference between the average prediction of our model and the correct value which we are trying to predict.

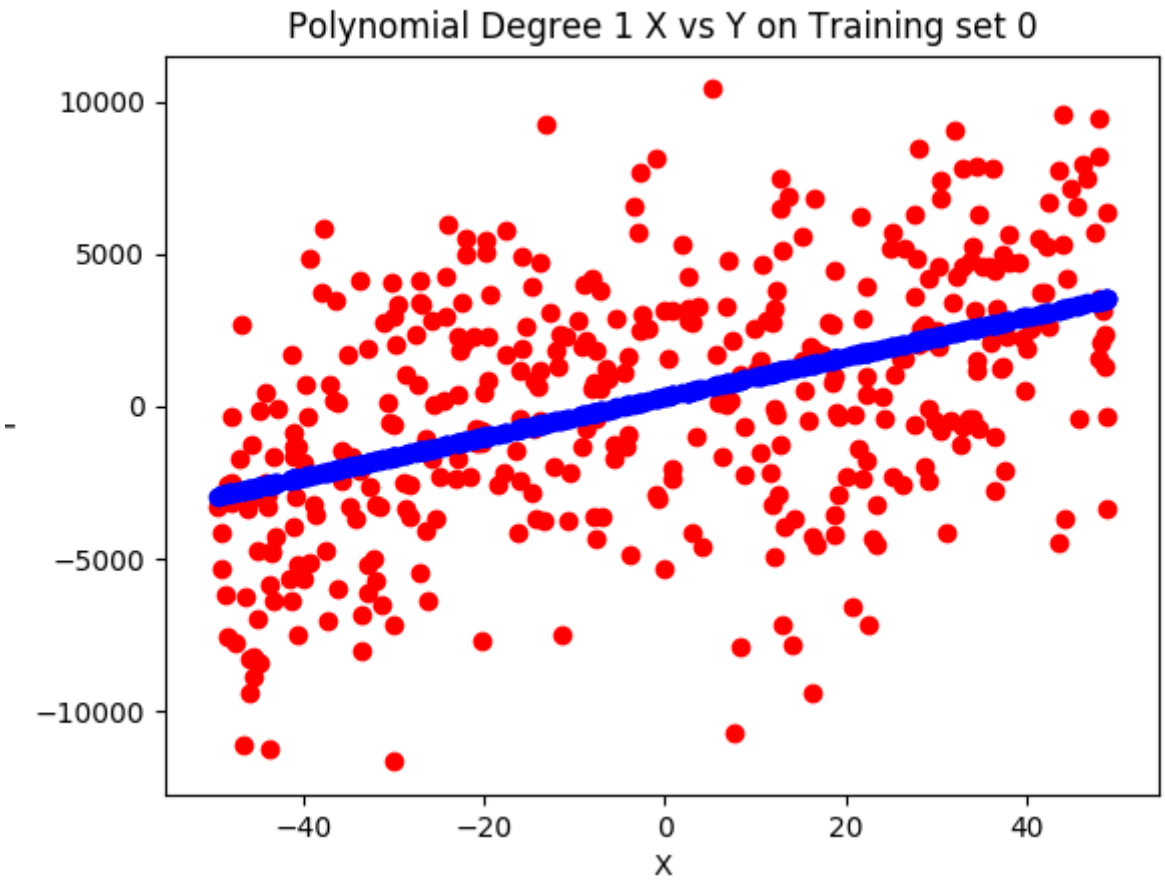
This is calculated and averaged out for each polynomial as follows,

```
    (np.mean(poly_prediction, axis = 0) - Y_test)**2
```

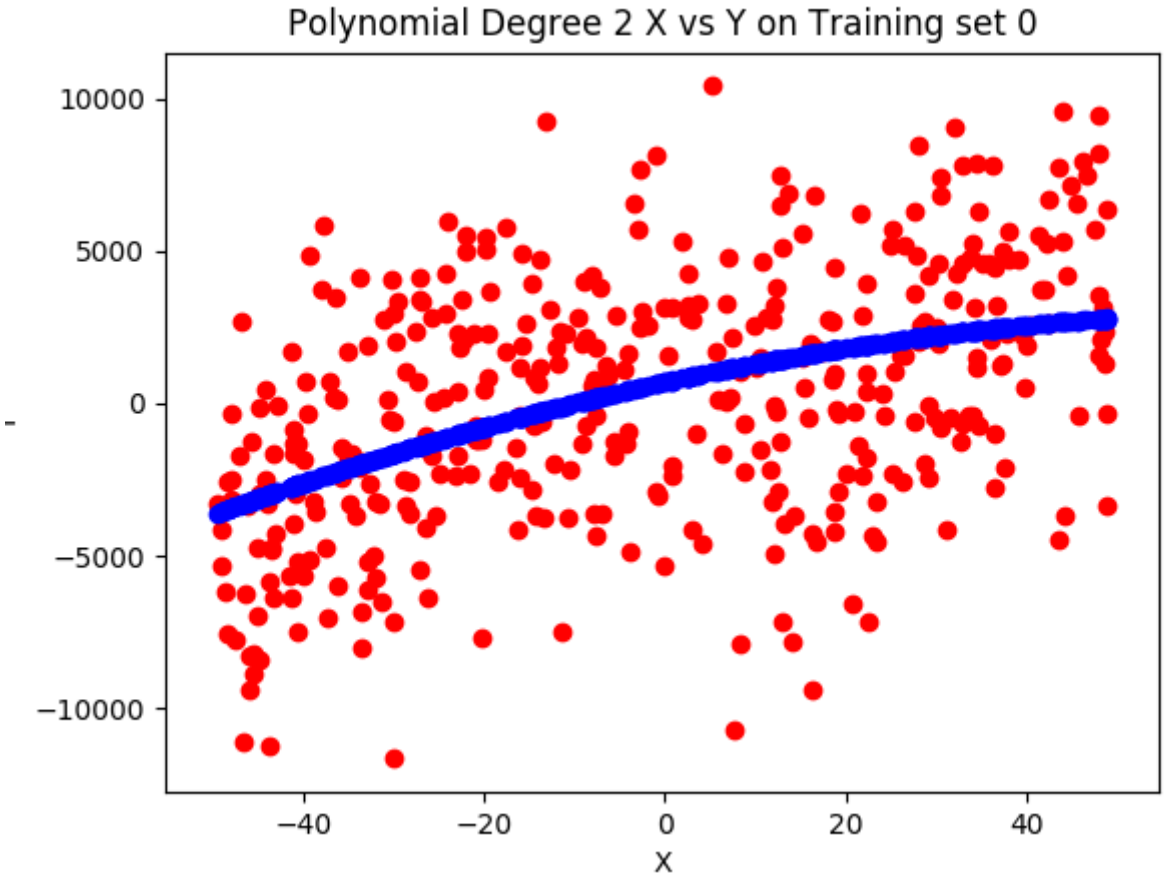
- Here, `Y_test` represents the correct output value for the input data point.

Listed are the plots for the models obtained using the `LinearRegression()` function using the first randomly generated test set for the respective polynomials.

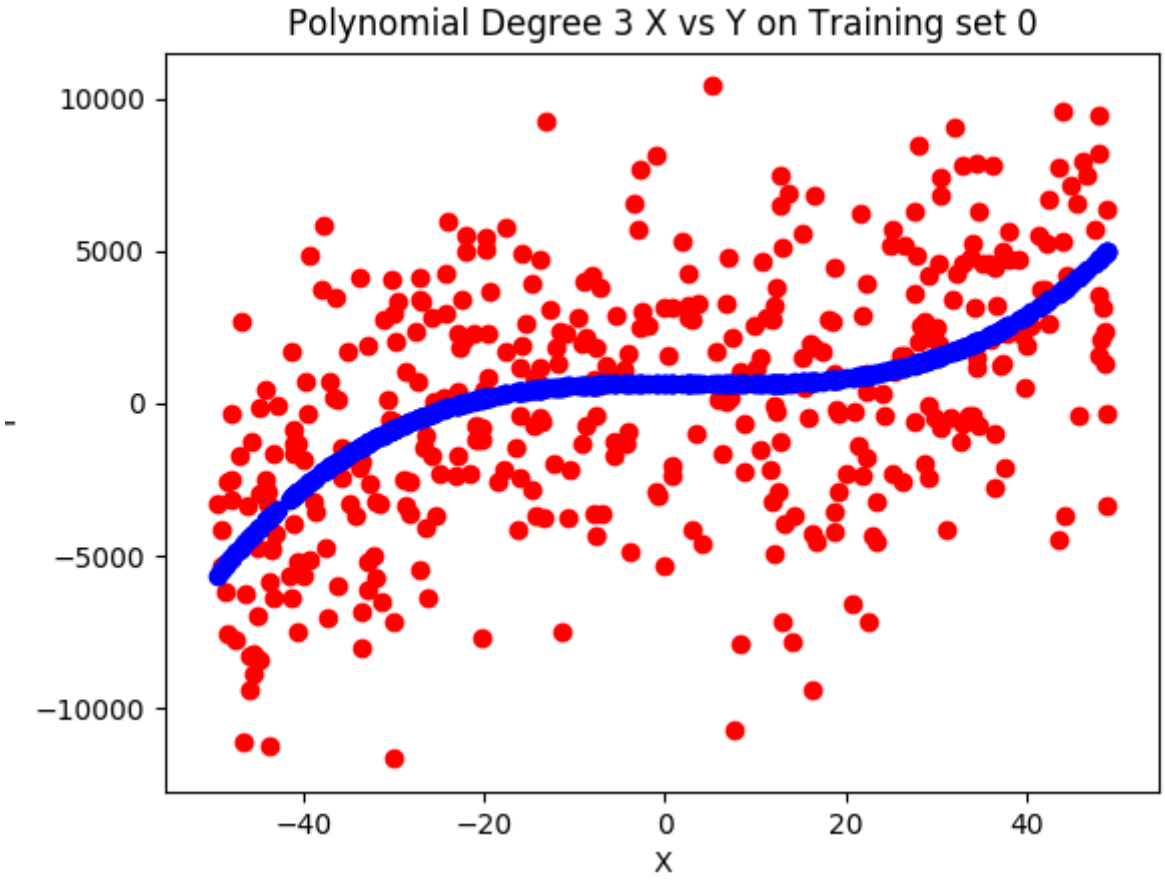
y = mx + c



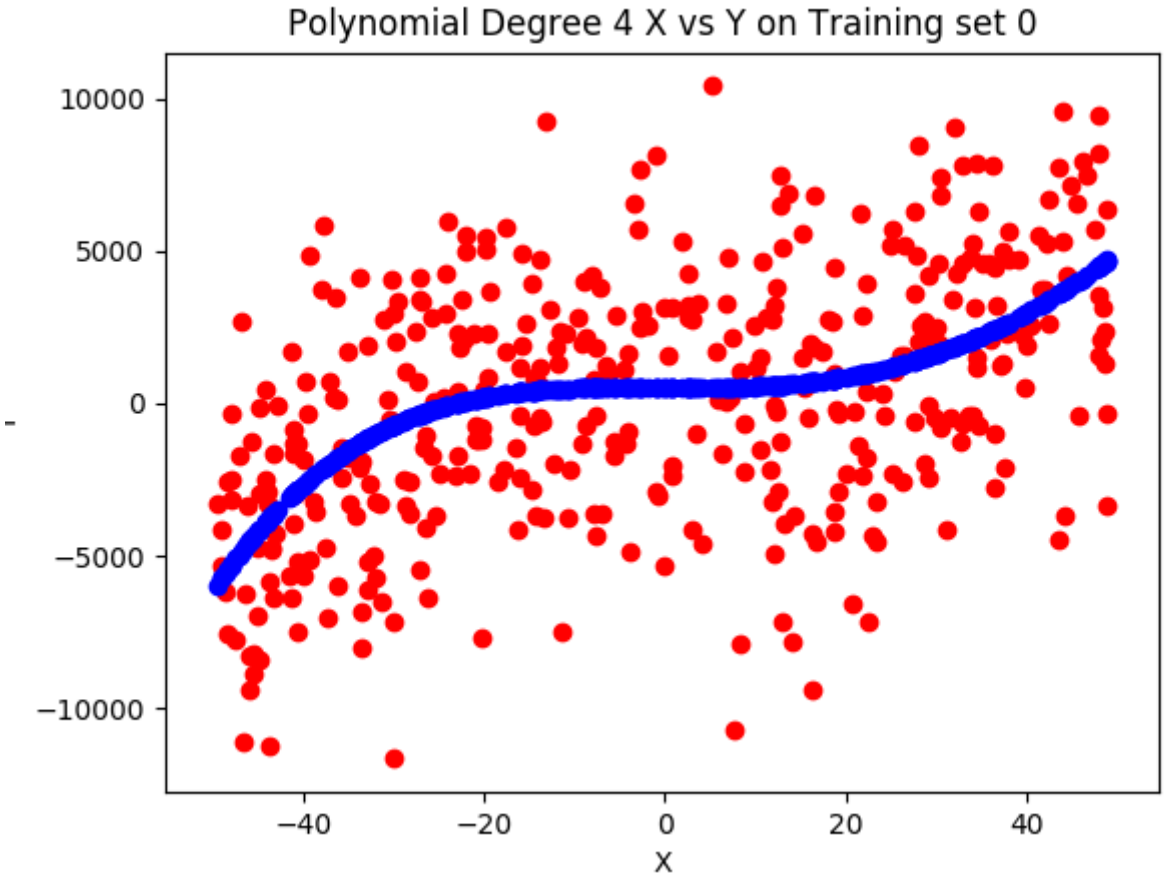
$y = ax^2 + bx + c$


Polynomial Degree 2 X vs Y on Training set 0
10000
5000
0
−5000
−10000
−40
−20
0
20
40
X

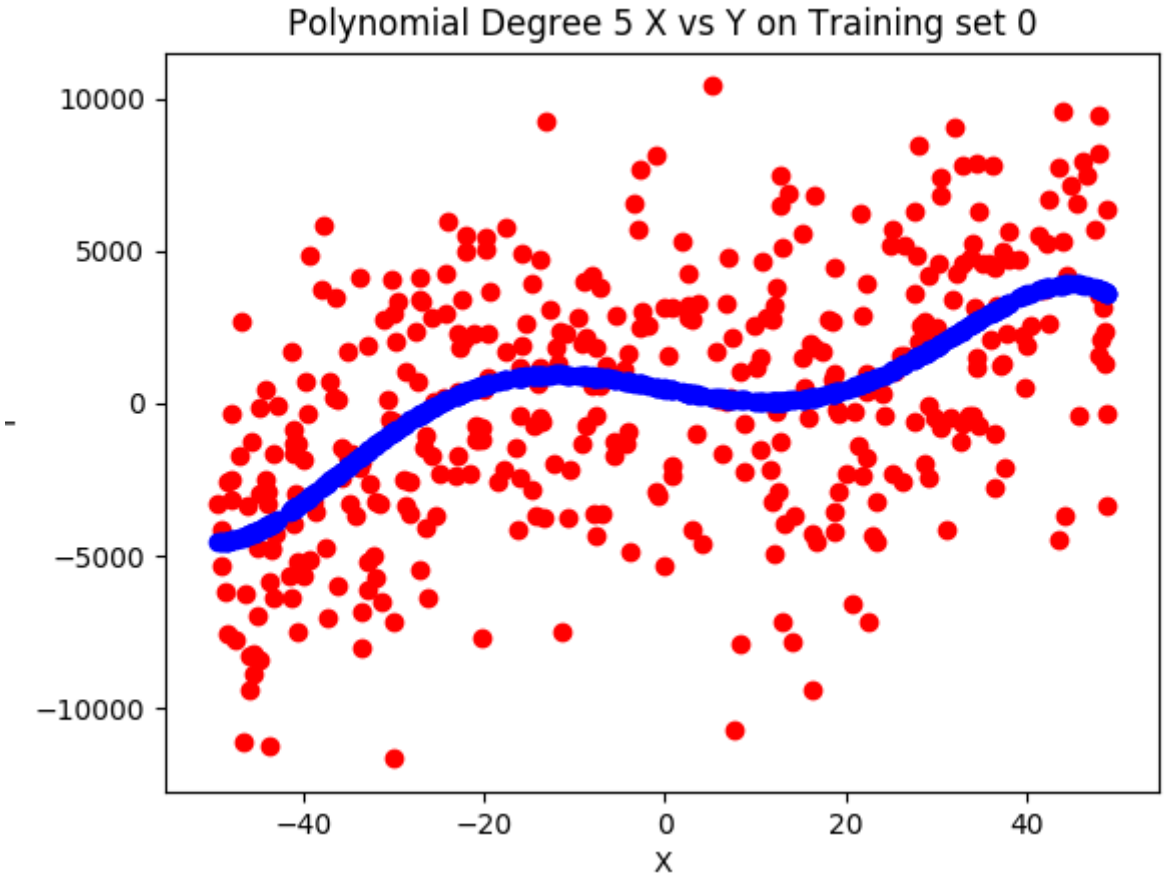
$y = ax^3 + bx^2 + cx + d$



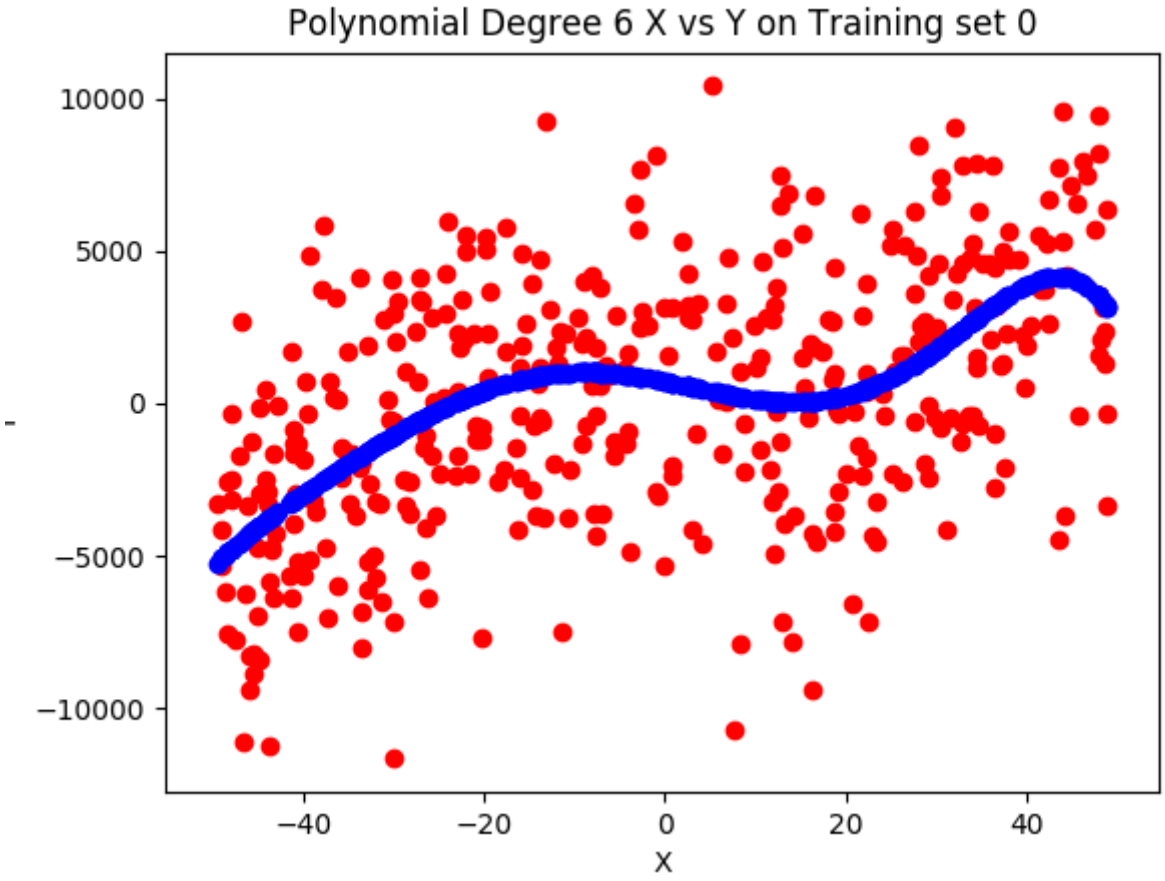
$y = ax^4 + bx^3 + cx^2 + dx + e$



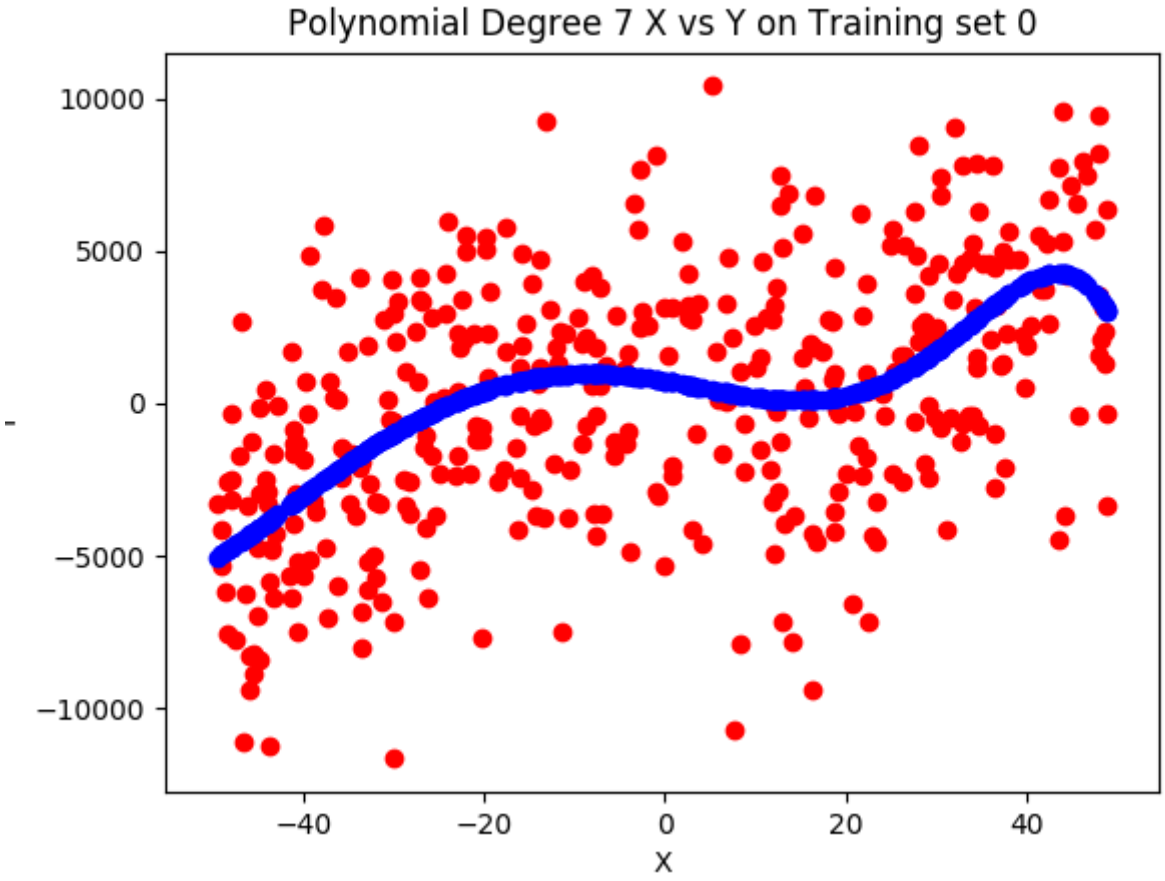
$y = ax^5 + bx^4 + cx^3 + dx^2 + ex + f$



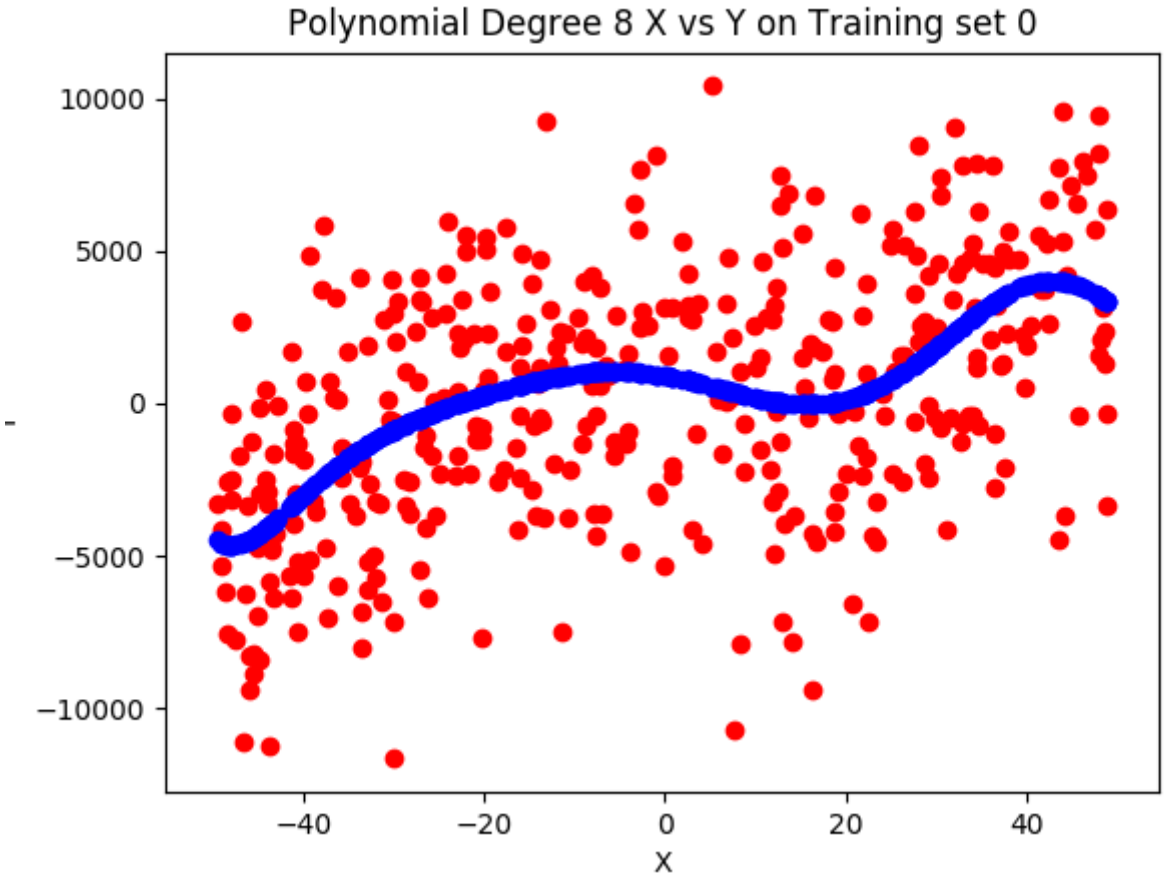
$y = ax^6 + bx^5 + cx^4 + dx^3 + ex^2 + fx + g$



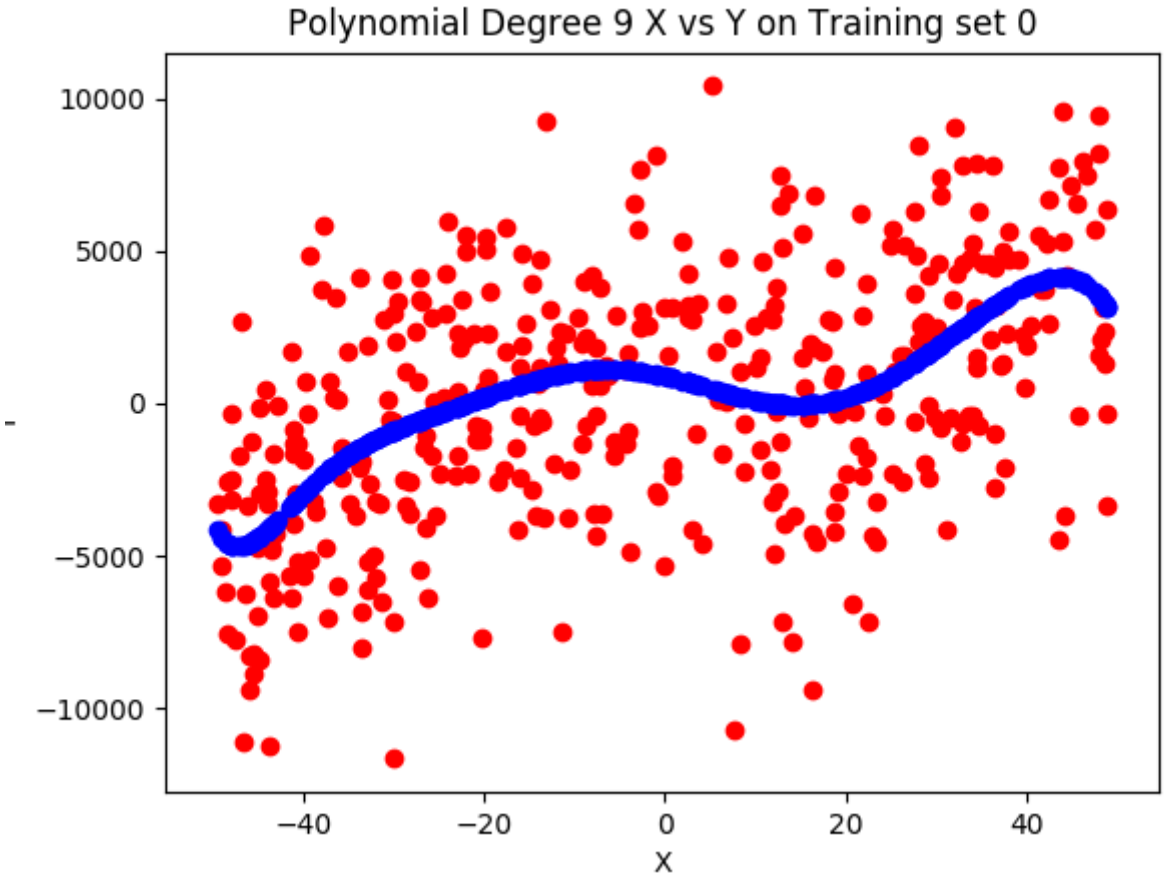
$y = ax^7 + bx^6 + cx^5 + dx^4 + ex^3 + fx^2 + gx + h$



$y = ax^8 + bx^7 + cx^6 + dx^5 + ex^4 + fx^3 + gx^2 + hx + i$



$y = ax^9 + bx^8 + cx^7 + dx^6 + ex^5 + fx^4 + gx^3 + hx^2 + ix + j$



Observervations

Here, it can be seen that the bias decreases as the complexity of the polynomial increases and the variance shows a slight increase as the degree of the polynomial increases.

This is in accordance with the Bias-Variance tradeoff where an optimum model is reached when total error is minimised.

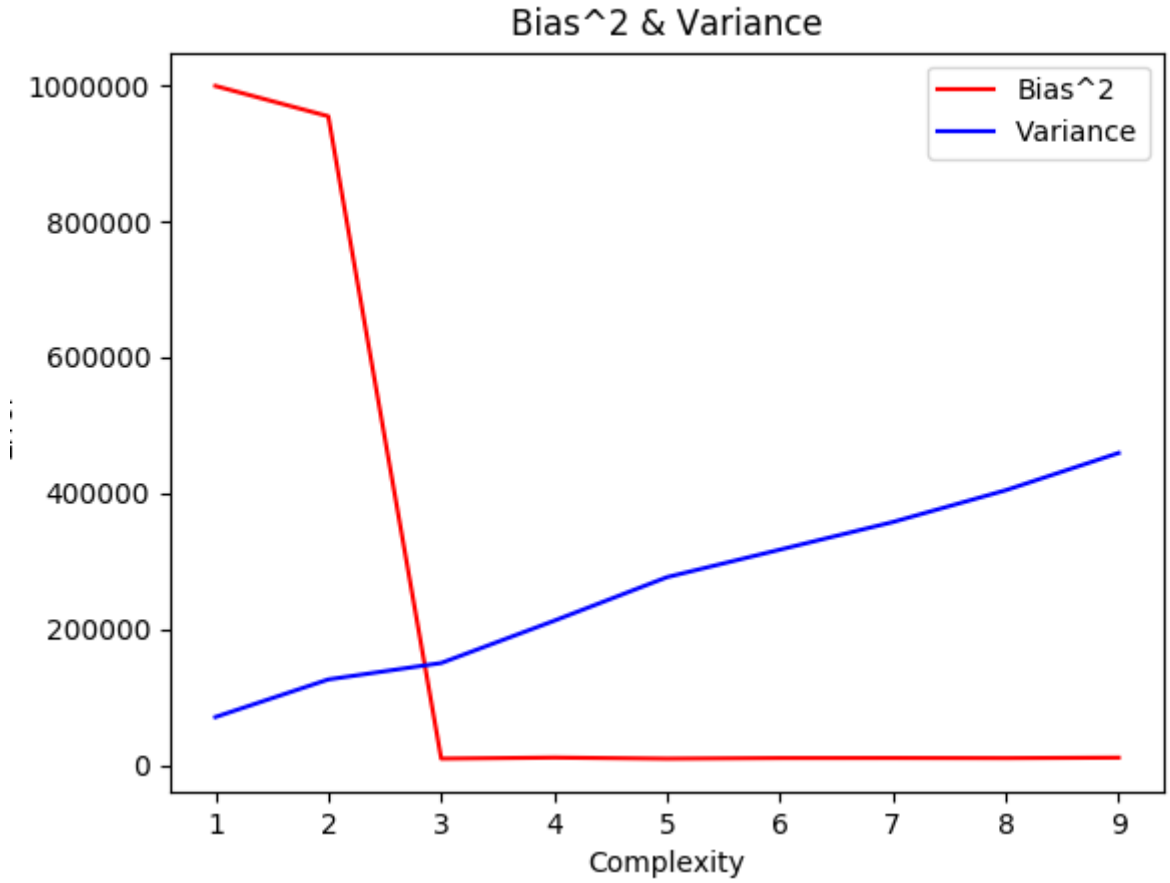
When our polynomial is too simple and has very few parameters then a high bias and low variance is observed. On the other hand, if our model has a large number of parameters then high variance and low bias is observed.

Tabulated Values

Bias^2 & Variance

Degree	Bias^2	Variance
1.0	999228.3968719237	70545.48914575046
2.0	954619.273794425	125870.85554877335
3.0	9389.730116791214	150073.7395464768
4.0	10907.34813407133	212235.70832526154
5.0	9339.194291326017	276388.4802547406
6.0	10248.585941147872	316863.49843748985
7.0	10335.2758616491	357510.98475735466
8.0	10149.419243937262	404286.670685786
9.0	10815.487036574234	459132.37837248633

Bias^2 Versus Variance Graph


Bias^2 & Variance
Bias^2
Variance
1000000
800000
600000
400000
200000
0
1
2
3
4
5
6
7
8
9
Complexity