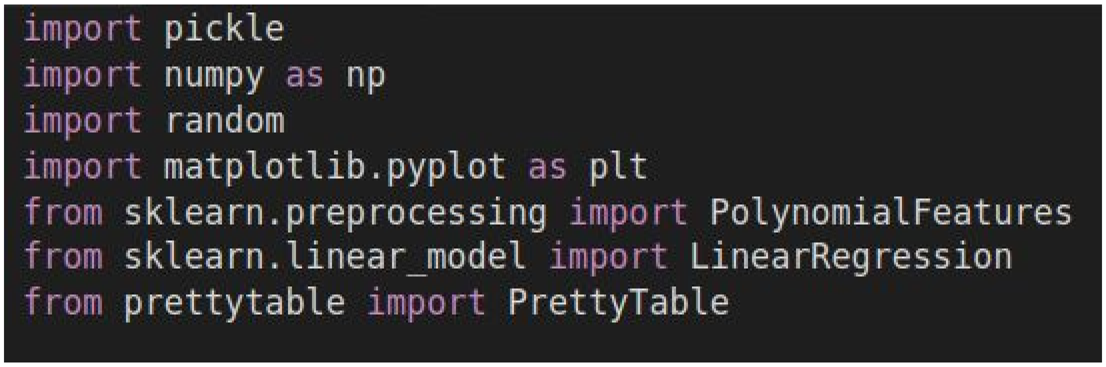
Assignment 1
Machine, Data and Learning

Kushagra Agarwal - 2018113012
Shreeya Pahune - 2018113011

Q1: Calculate Bias and Variance

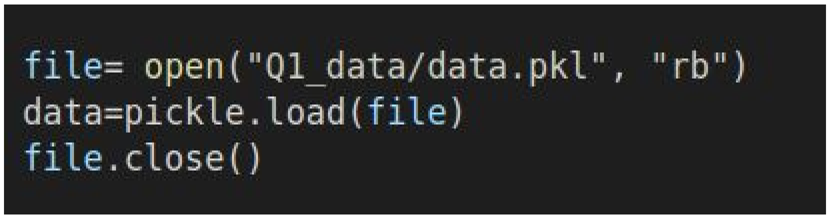
Code Implementation

Imported all the necessary libraries:



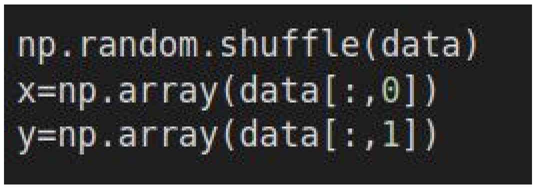
```
import pickle
import numpy as np
import random
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from prettytable import PrettyTable
```

Opened file and loaded the content into ‘data’:



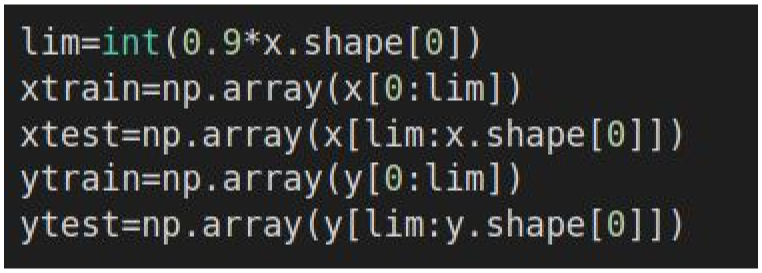
```
file= open("Q1_data/data.pkl", "rb")
data=pickle.load(file)
file.close()
```

Randomised data and segregated the features and values:



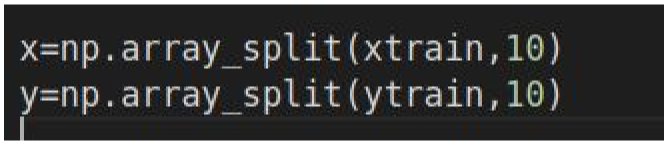
```
np.random.shuffle(data)
x=np.array(data[:,0])
y=np.array(data[:,1])
```

Split the train and test data with ratio 0.9:0.1 (Does same task as sklearn's train_test_split):



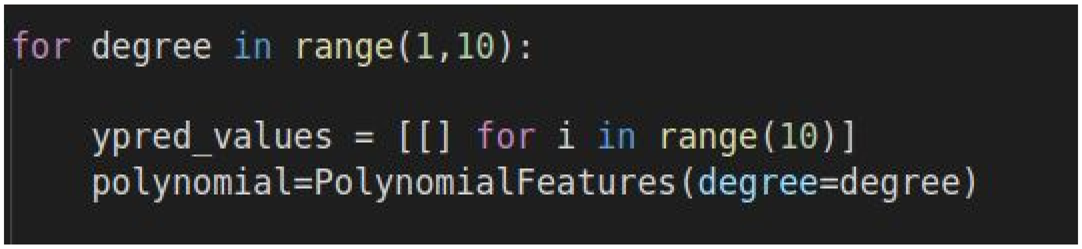
```
lim=int(0.9*x.shape[0])
xtrain=np.array(x[0:lim])
xtest=np.array(x[lim:x.shape[0]])
ytrain=np.array(y[0:lim])
ytest=np.array(y[lim:y.shape[0]])
```

Split the training data into 10 subsets:



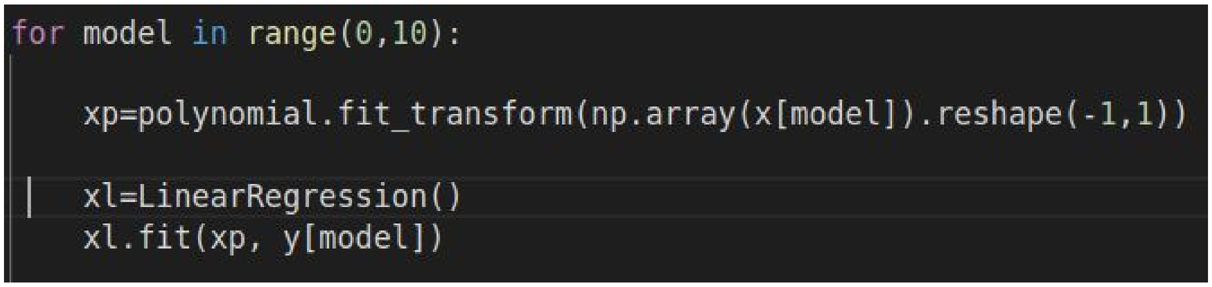
```
x=np.array_split(xtrain,10)
y=np.array_split(ytrain,10)
```

Outer for loop to range the polynomials from degree 1 to 9.
Ypred stores the predicted set of values for xtest.
polynomial Features returns a polynomial of input degree:



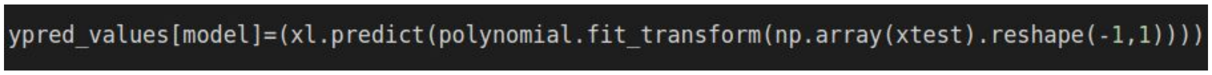
```
for degree in range(1,10):

    ypred_values = [[] for i in range(10)]
    polynomial=PolynomialFeatures(degree=degree)
```

Inner for loop iterates over the each of the 10 subsets of the training data.
Fit_transform performs a fit (calculates the parameters and saves them as an internal objects state) and then a transform (method to apply the transformation to the set of examples)
reshape(-1, 1) to vectorise x[model]
LinerRegression fits a linear model with coefficients to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation, and the .fit(a, b) fits the data to a = training data and b = target data.



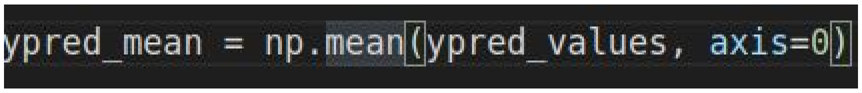
```
for model in range(0,10):

    xp=polynomial.fit_transform(np.array(x[model]).reshape(-1,1))

    xl=LinearRegression()
    xl.fit(xp, y[model])
```

(Inside inner for loop)
Ypred_values (10x500) is a 2D array in which each 1D array contains the predicted values (calculated from xtest) for a particular model
.predicts Predicts the sample(input) using the linear model and returns predicted values.



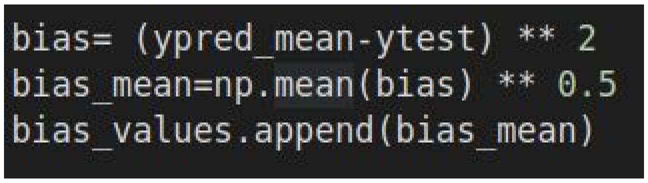
```
ypred_values[model]=(xl.predict(polynomial.fit_transform(np.array(xtest).reshape(-1,1))))
```

(Inside outer for loop)
This calculates column-wise (axis=0) mean that is ypred_mean (500x1) will contain at each indice the mean corresponding to that xtest.



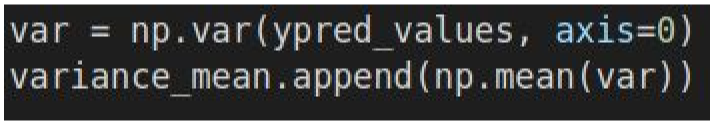
```
ypred_mean = np.mean(ypred_values, axis=0)
```

(Inside outer for loop)
Calculating Bias_sq as per formula
Calculating the mean bias for a particular polynomial
Append to final bias array



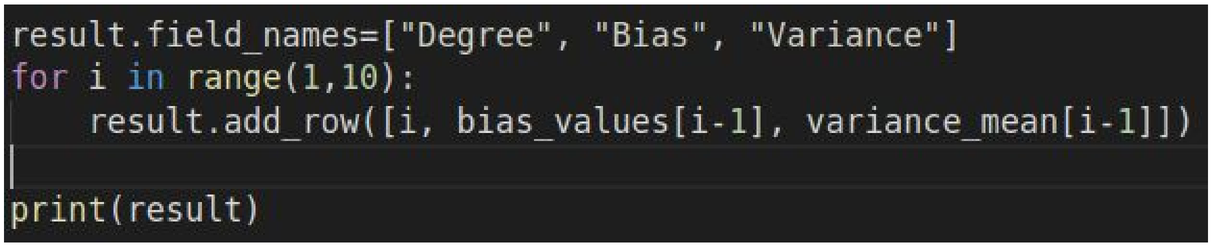
```
bias= (ypred_mean-ytest) ** 2
bias_mean=np.mean(bias) ** 0.5
bias_values.append(bias_mean)
```

(Inside outer for loop)
This calculates column-wise (axis=0) variance that is var will contain at each indice the variance corresponding to that xtest.
Append the mean variance for a particular polynomial to final variance array



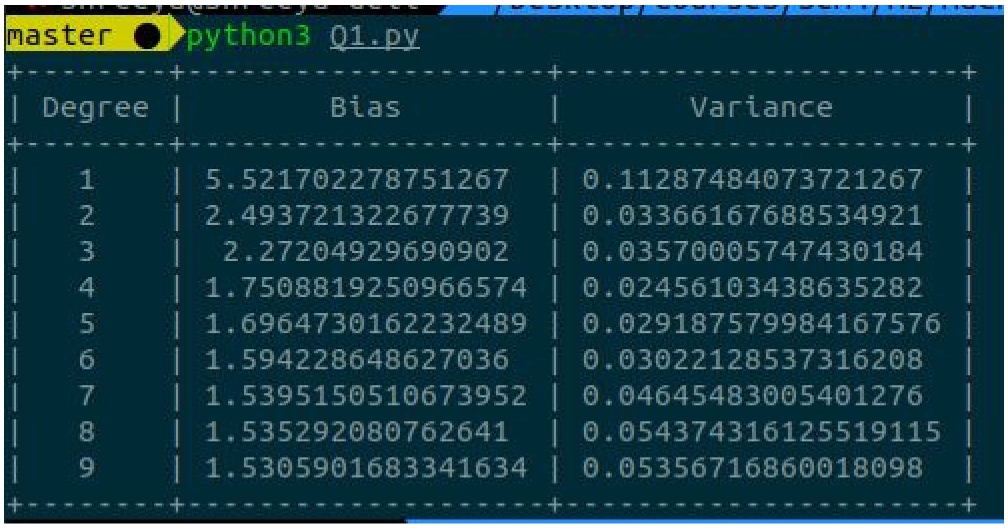
```
var = np.var(ypred_values, axis=0)
variance_mean.append(np.mean(var))
```

Display result as table



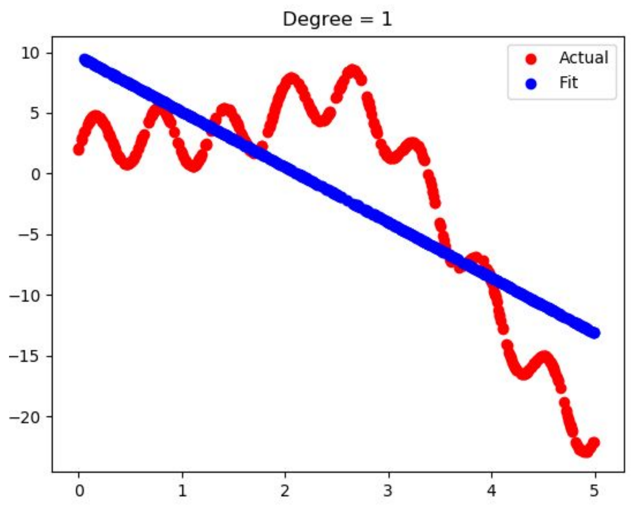
```
result.field_names=["Degree", "Bias", "Variance"]
for i in range(1,10):
    result.add_row([i, bias_values[i-1], variance_mean[i-1]])

print(result)
```

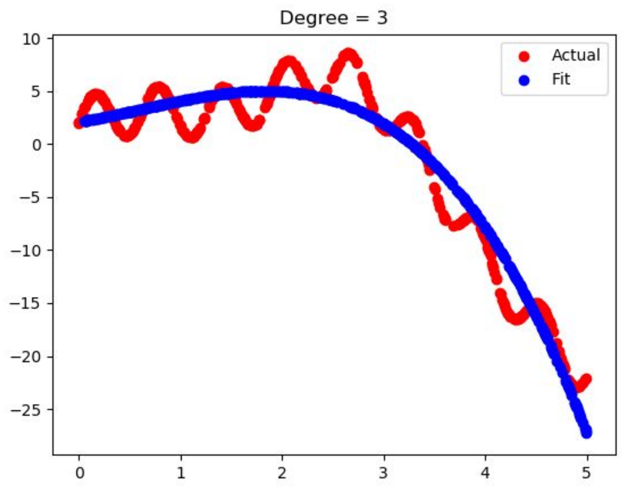
Results

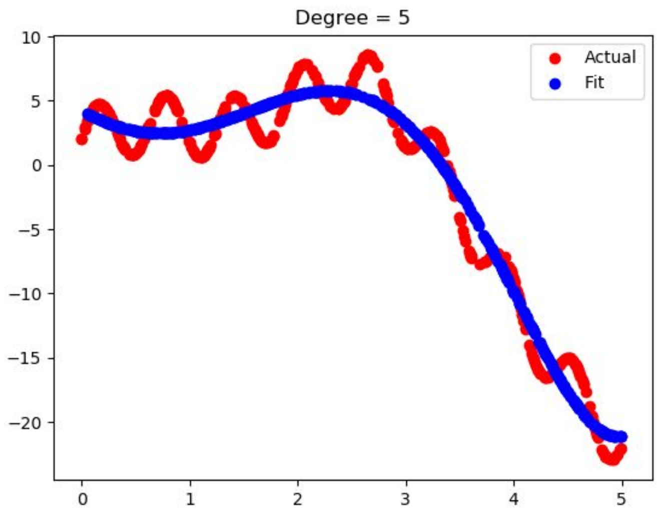


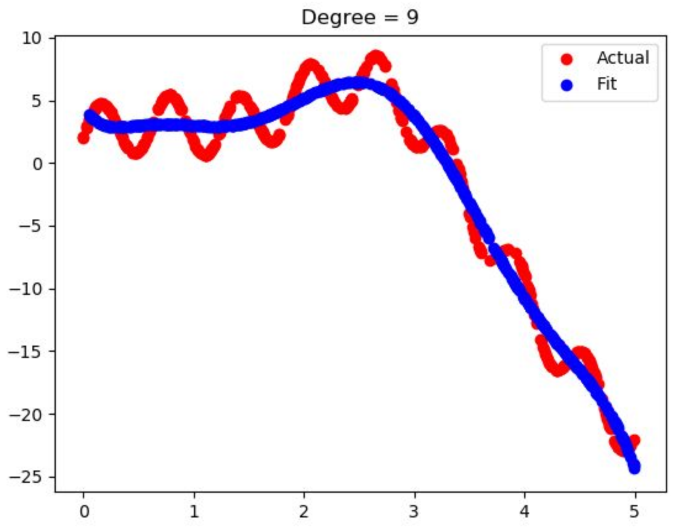
```
master ● python3 Q1.py
+--------+---------------------+----------------------+
| Degree |         Bias        |       Variance       |
+--------+---------------------+----------------------+
|   1    |  5.521702278751267  | 0.11287484073721267  |
|   2    |  2.493721322677739  | 0.03366167688534921  |
|   3    |   2.27204929690902  | 0.035700057747430184 |
|   4    |  1.7508819250966574 | 0.02456103438635282  |
|   5    |  1.6964730162232489 | 0.029187579984167576 |
|   6    |  1.594228648627036  | 0.03022128537316208  |
|   7    |  1.5395150510673952 | 0.04645483005401276  |
|   8    |  1.535292080762641  | 0.054374316125519115 |
|   9    |  1.5305901683341634 | 0.05356716860018098  |
+--------+---------------------+----------------------+
```

Plots of Testing Data vs Predicted Values



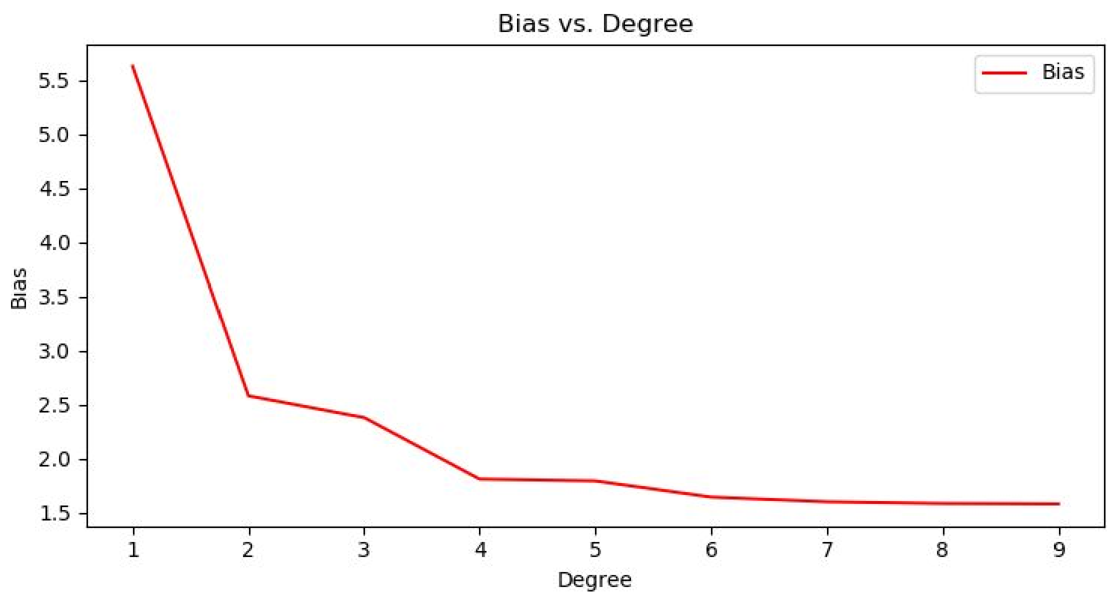




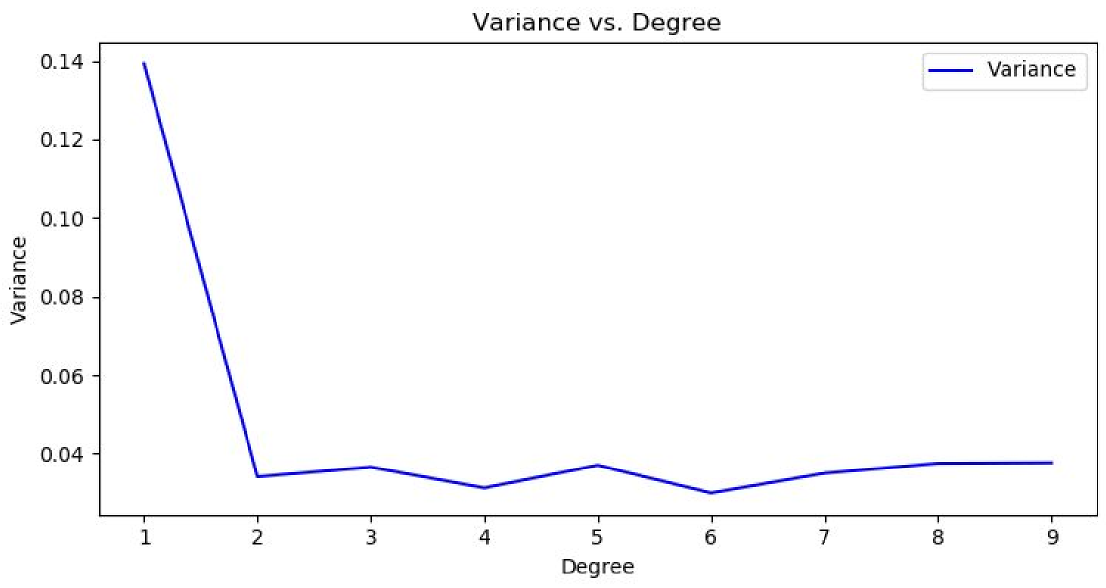


Observations

Bias is constantly decreasing as degree becomes complex, as complexity i.e Degree of the model increases the Bias decreases because we are increasing the number of features. Hence the model is fitting the curve even better:



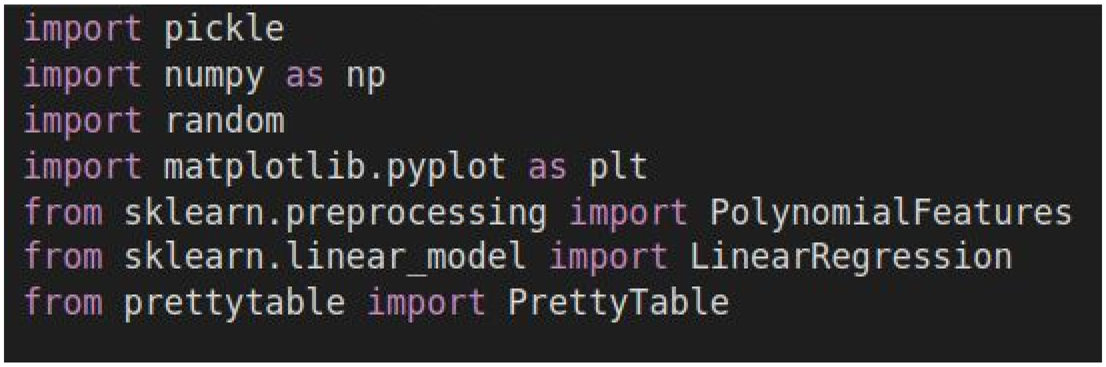
Variance is **overall** decreasing as degree becomes complex, as complexity increases variance decreases initially and then it increases because of overfitting as the curve generated by the model tends to follow only the training set:



Q2: Bias Variance Tradeoff Plot

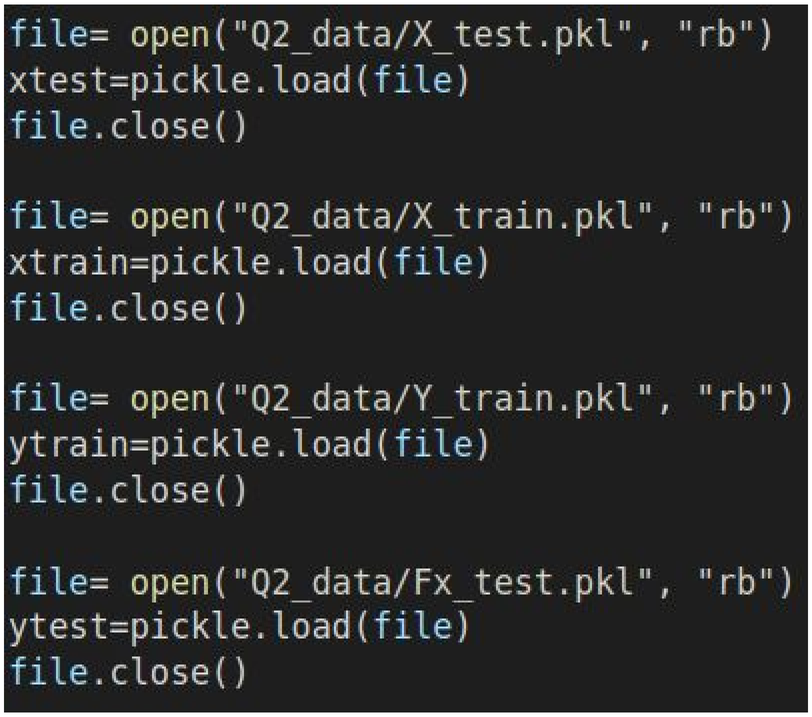
Code Implementation

Imported all the necessary libraries:



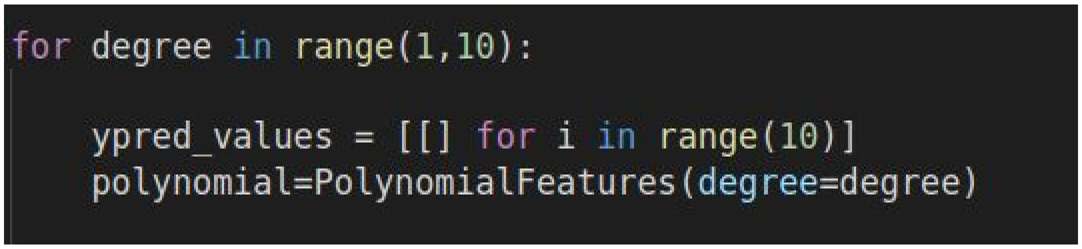
```
import pickle
import numpy as np
import random
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from prettytable import PrettyTable
```

Opened file and loaded the content into ‘data’:



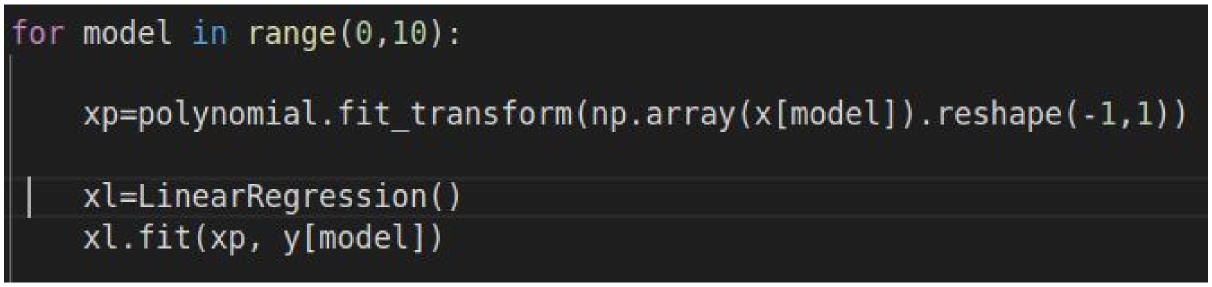
```
file= open("Q2_data/X_test.pkl", "rb")
xtest=pickle.load(file)
file.close()

file= open("Q2_data/X_train.pkl", "rb")
xtrain=pickle.load(file)
file.close()

file= open("Q2_data/Y_train.pkl", "rb")
ytrain=pickle.load(file)
file.close()

file= open("Q2_data/Fx_test.pkl", "rb")
ytest=pickle.load(file)
file.close()
```

Outer for loop to range the polynomials from degree 1 to 9.
Ypred stores the predicted set of values for xtest.
polynomial Features returns a polynomial of input degree:



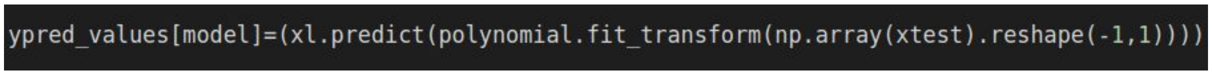
```
for degree in range(1,10):

    ypred_values = [[] for i in range(10)]
    polynomial=PolynomialFeatures(degree=degree)
```

Inner for loop iterates over the each of the 10 subsets of the training data.
Fit_transform performs a fit (calculates the parameters and saves them as an internal objects state) and then a transform (method to apply the transformation to the set of examples)
reshape(-1, 1) to vectorise x[model]
LinerRegression fits a linear model with coefficients to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation, and the .fit(a, b) fits the data to a = training data and b = target data.



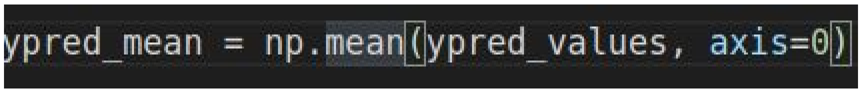
```
for model in range(0,10):

    xp=polynomial.fit_transform(np.array(x[model]).reshape(-1,1))

    xl=LinearRegression()
    xl.fit(xp, y[model])
```

(Inside inner for loop)
Ypred_values (10x500) is a 2D array in which each 1D array contains the predicted values (calculated from xtest) for a particular model
.predicts Predicts the sample(input) using the linear model and returns predicted values.



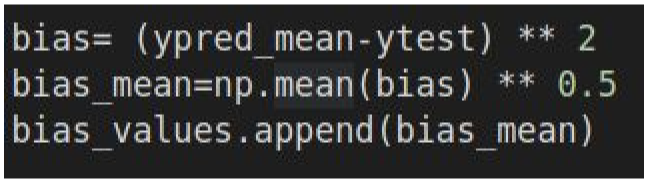
```
ypred_values[model]=(xl.predict(polynomial.fit_transform(np.array(xtest).reshape(-1,1))))
```

(Inside outer for loop)
This calculates column-wise (axis=0) mean that is ypred_mean (500x1) will contain at each indice the mean corresponding to that xtest.



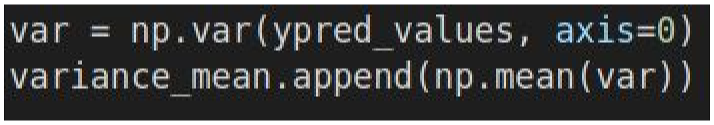
```
ypred_mean = np.mean(ypred_values, axis=0)
```

(Inside outer for loop)
Calculating Bias_sq as per formula
Calculating the mean bias for a particular polynomial
Append to final bias array



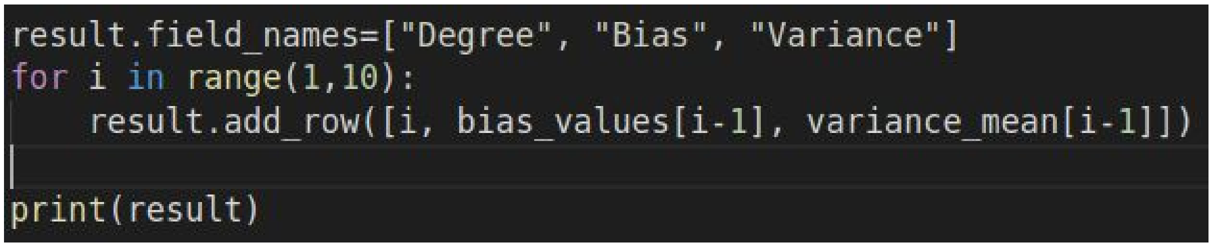
```
bias= (ypred_mean-ytest) ** 2
bias_mean=np.mean(bias) ** 0.5
bias_values.append(bias_mean)
```

(Inside outer for loop)
This calculates column-wise (axis=0) variance that is var will contain at each indice the variance corresponding to that xtest.
Append the mean variance for a particular polynomial to final variance array



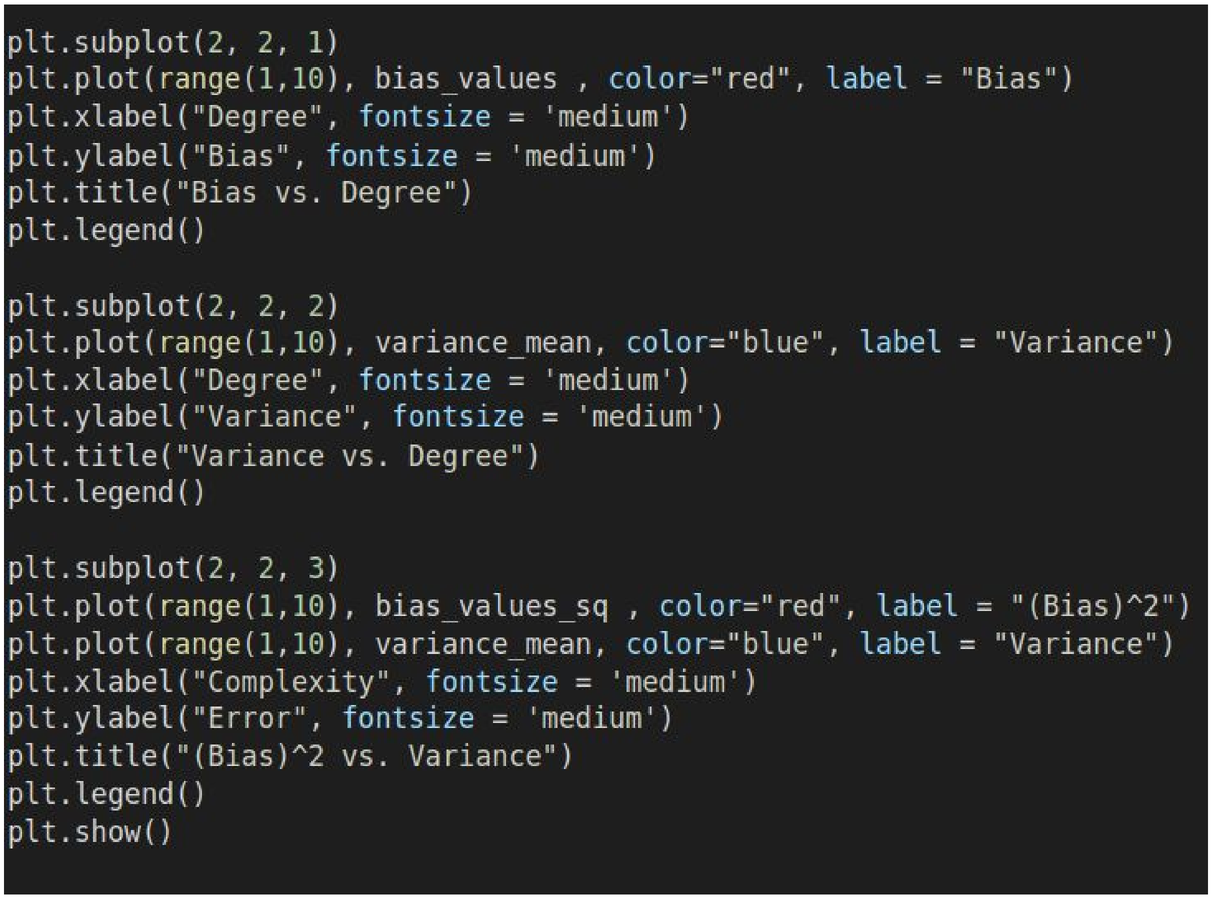
```
var = np.var(ypred_values, axis=0)
variance_mean.append(np.mean(var))
```

Display result as table



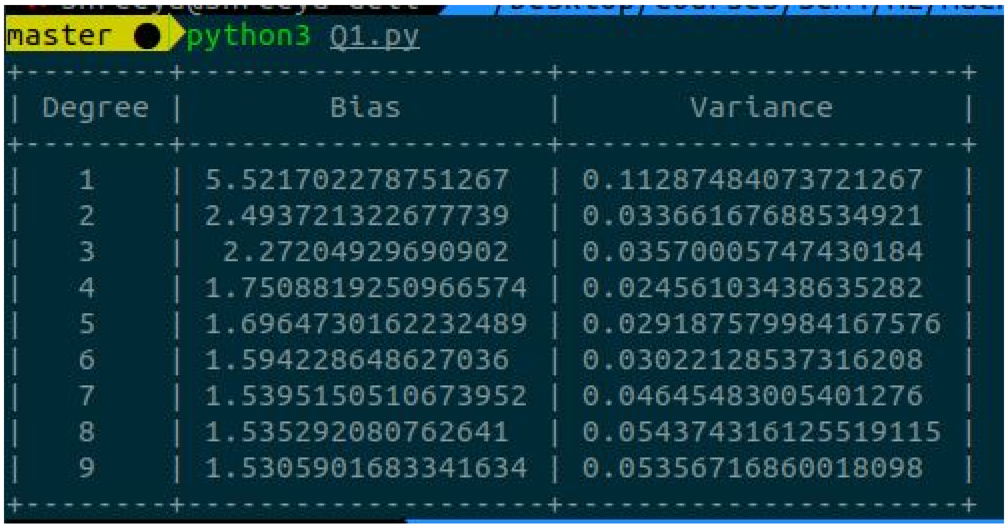
```
result.field_names=["Degree", "Bias", "Variance"]
for i in range(1,10):
    result.add_row([i, bias_values[i-1], variance_mean[i-1]])

print(result)
```

Plot graphs as subplots in a figure:



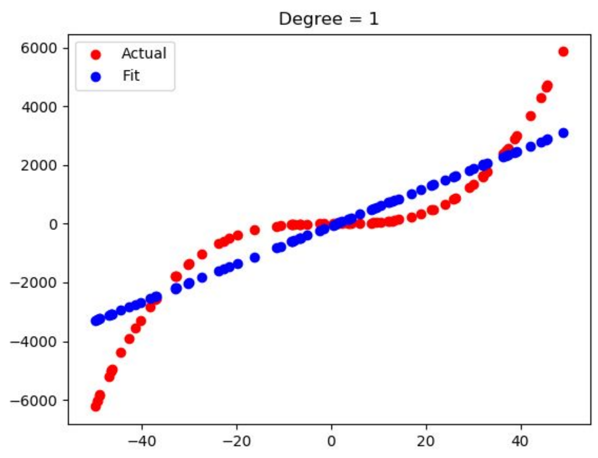
```
plt.subplot(2, 2, 1)
plt.plot(range(1,10), bias_values , color="red", label = "Bias")
plt.xlabel("Degree", fontsize = 'medium')
plt.ylabel("Bias", fontsize = 'medium')
plt.title("Bias vs. Degree")
plt.legend()

plt.subplot(2, 2, 2)
plt.plot(range(1,10), variance_mean, color="blue", label = "Variance")
plt.xlabel("Degree", fontsize = 'medium')
plt.ylabel("Variance", fontsize = 'medium')
plt.title("Variance vs. Degree")
plt.legend()

plt.subplot(2, 2, 3)
plt.plot(range(1,10), bias_values_sq , color="red", label = "(Bias)^2")
plt.plot(range(1,10), variance_mean, color="blue", label = "Variance")
plt.xlabel("Complexity", fontsize = 'medium')
plt.ylabel("Error", fontsize = 'medium')
plt.title("(Bias)^2 vs. Variance")
plt.legend()
plt.show()
```

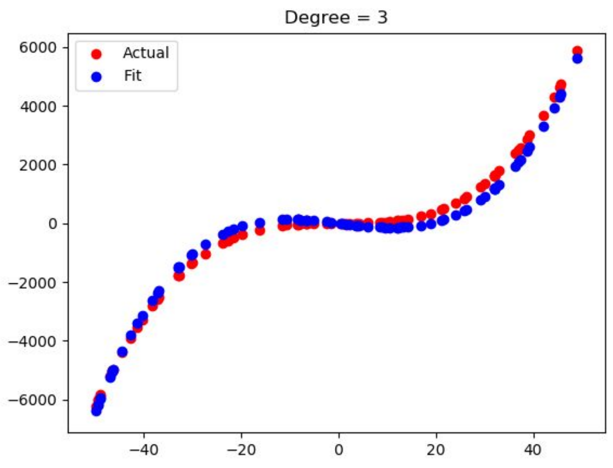
Results

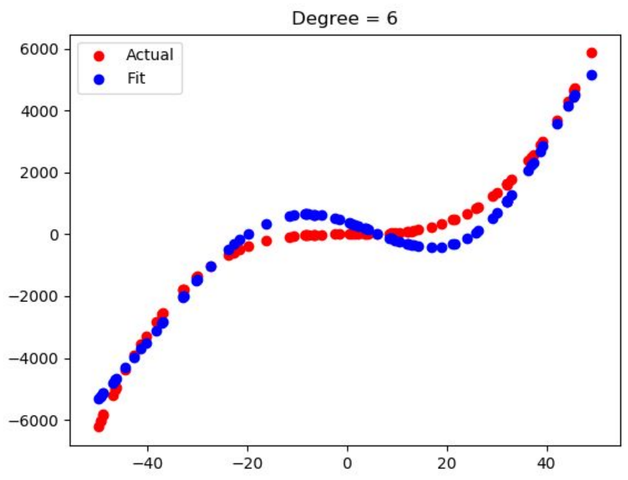


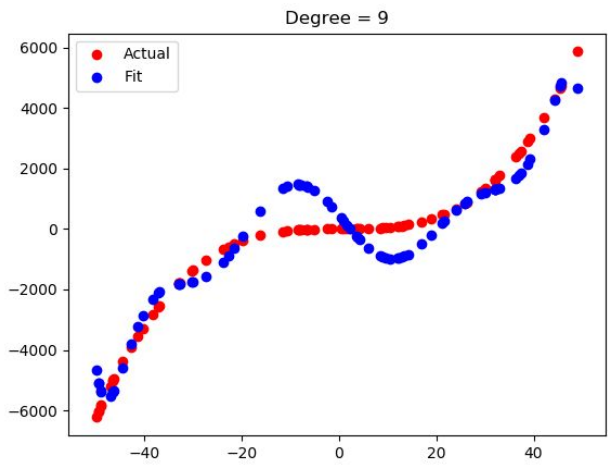
```
master ● python3 Q1.py
+--------+---------------------+----------------------+
| Degree |         Bias        |       Variance       |
+--------+---------------------+----------------------+
|   1    | 5.521702278751267   | 0.11287484073721267  |
|   2    | 2.493721322677739   | 0.03366167688534921  |
|   3    |  2.27204929690902   | 0.03570005747430184  |
|   4    | 1.7508819250966574  | 0.02456103438635282  |
|   5    | 1.6964730162232489  | 0.029187579984167576 |
|   6    | 1.594228648627036   | 0.03022128537316208  |
|   7    | 1.5395150510673952  | 0.04645483005401276  |
|   8    | 1.535292080762641   | 0.054374316125519115 |
|   9    | 1.5305901683341634  | 0.05356716860018098  |
+--------+---------------------+----------------------+
```

Plots of Testing Data vs Predicted Values



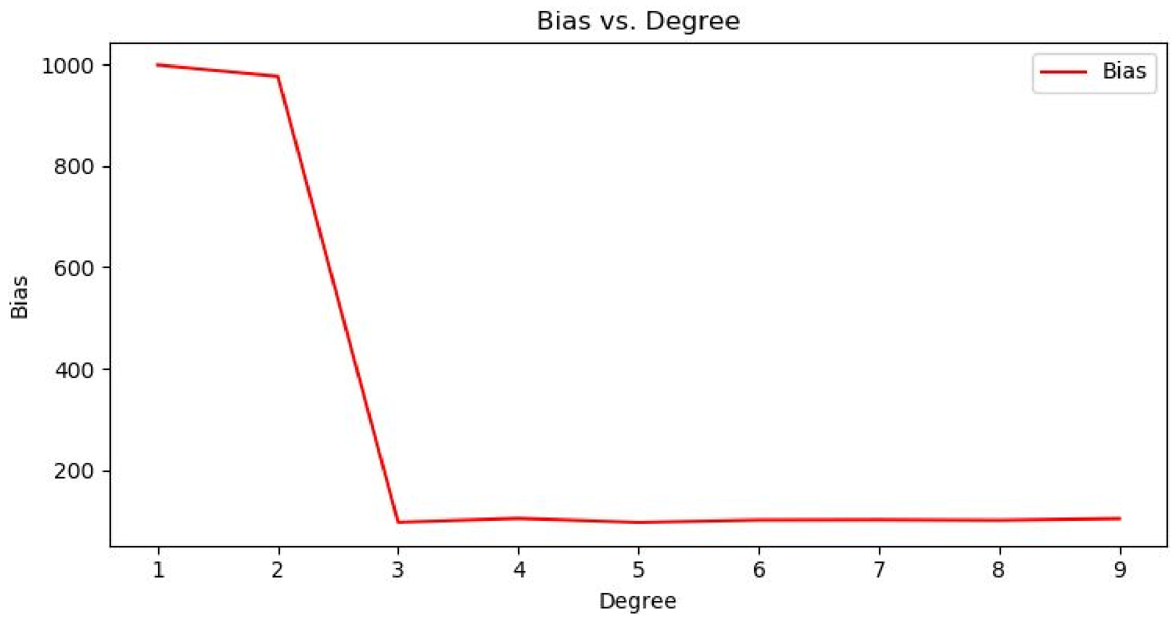




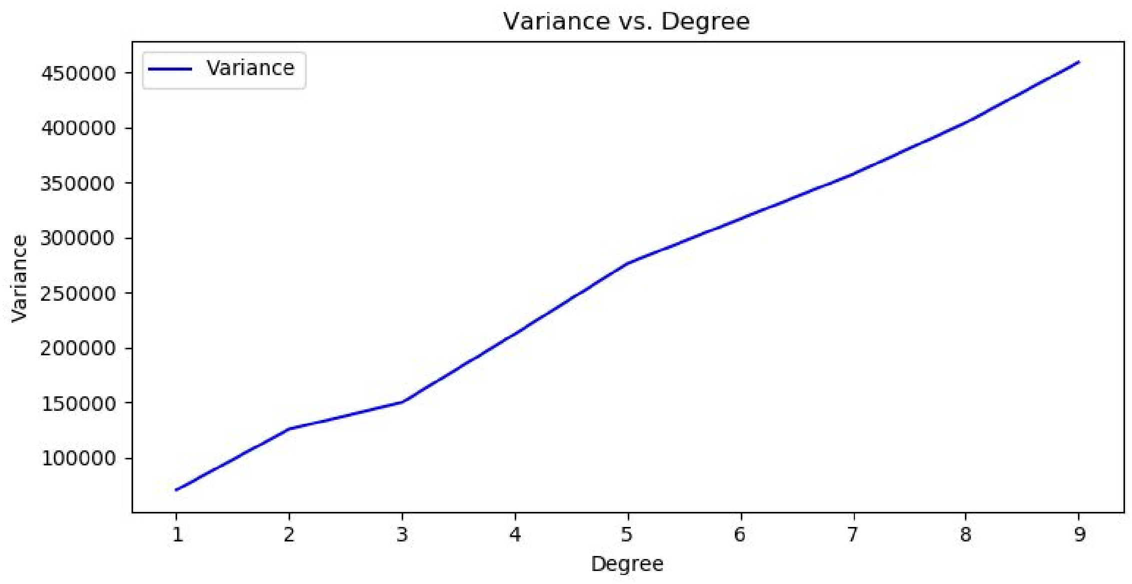


Observations

Bias initially sharply decreases and then nearly remains constant, the bias decreases initially but after n = 3 , the value becomes almost constant, because at cubic polynomial the model almost fits the data and further complex models aren't much different from this cubic model:

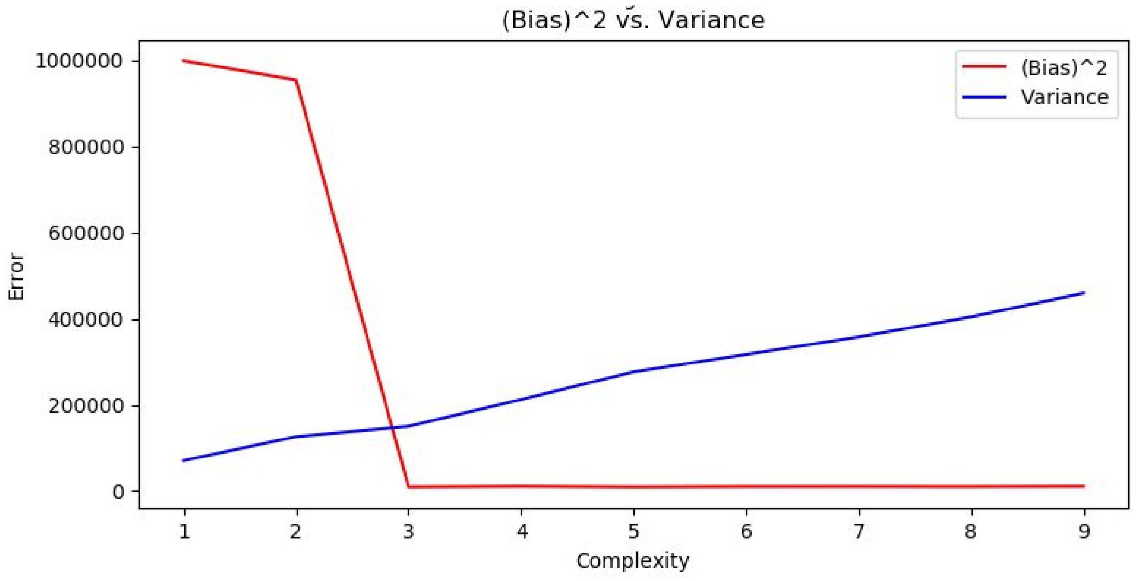


Variance is increasing with degree:



Complexity vs Error:
As we can see, error is least near degree = 3 (Bias-variance tradeoff)

High Bias - Low Variance = Underfitting (As Complexity of model is less i.e. low degree polynomial)
Low Bias - Low Variance = Perfect fit (Bias-variance tradeoff is balanced i.e. simultaneously minimize these two sources of error)
Low Bias - High Variance = Overfitting (As Complexity of model is more i.e. high degree polynomial)


(Bias)^2 vs. Variance
(Bias)^2
Variance
Error
Complexity