MACHINE, DATA AND LEARNING

ASSIGNMENT 1: QUESTION 2

TEAM NUMBER: 86

TEAM MEMBERS:
Tathagata Raha (2018114017)
Arathy Rose Tony(2018101042)

TASK

You have been provided with a training data and a testing data. You need to fit the given data to polynomials of degree 1 to 9(both inclusive).
Specifically, you have been given 20 subsets of training data containing 400 samples each. For each polynomial, create 20 models trained on the 20 different subsets and find the variance of the predictions on the testing data. Also, find the bias of your trained models on the testing data. Finally plot the bias-variance trade-Off graph.
Write your observations in the report with respect to underfitting, overfitting and also comment on the type of data just by looking at the bias-variance plot.

SOME BASIC DEFINITIONS

Bias

Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. A model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to a high error on training and test data.

$$Bias^2 = (E[\hat{f}(x)] - f(x))^2$$

where $f(x)$ represents the true value, $\hat{f}(x)$ represents the predicted value

Variance

Variance is the variability of a model prediction for a given data point. The variance is how much the predictions for a given point vary between different realizations of the model.

$$Variance = E[\hat{f}(x) - E[\hat{f}(x)]]^2$$

where $f(x)$ represents the true value, $\hat{f}(x)$ represents the predicted value

Error

Noise is a unwanted distortion in data. Noise is anything that is spurious and extraneous to the original data, that is not intended to be present in the first place, but was introduced due to faulty capturing process.

Bias-Variance TradeOff

If our model is too simple and has very few parameters then it has high bias and low variance. On the other hand, if our model has a large number of parameters then it’s going to have high variance and low bias. So we need to find the right/good balance without overfitting and underfitting the data.

EXPLANATION OF THE CODE/APPROACH

Header files included

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
import numpy
import random
import pickle
import pandas
import matplotlib.pyplot as plt
```

Some global variables that can be used to check specific outputs

```
debug = 1  # 1 if you want to see the variable values during the program execution
graphing = 1  # 1 to see the graphs
```

STEP 1: LOADING THE DATASET AND VISUALISING IT

LOAD THE DATASET

Here we load the data_set from multiple files stored in the same directory as the current notebook.

Get x train dataset

```
f = open('X_train.pkl', 'rb')
X_train_data_sets = pickle.load(f)
f.close()
```

Get y train dataset

```
f = open('Y_train.pkl', 'rb')
Y_train_data_sets = pickle.load(f)
f.close()
```

Get x test dataset

```
f = open('X_test.pkl', 'rb')
xTest = pickle.load(f)
f.close()
```

Get y test dataset

```
f = open('Fx_test.pkl', 'rb')
yTest = pickle.load(f)
f.close()
```

Set all the sizes of the datasets

```
number_of_data_sets = len(X_train_data_sets)
train_size = len(X_train_data_sets[0])
test_size = len(xTest)
```

Get the list of all the x and y coordinates of the training dataset

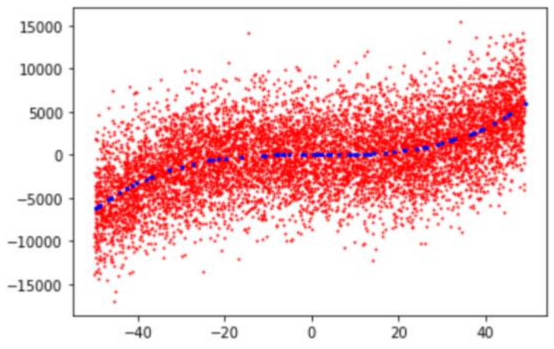
```
x = []
y = []
for i in range(number_of_data_sets):
    x.append(list(X_train_data_sets[i]))
    y.append(list(Y_train_data_sets[i]))
```

Graph the given dataset

Here we plot the given training dataset, just to get the feel of the dataset provided.

```
fig = plt.figure()
plt.plot(x, y, 'r.', markersize=2)
plt.plot(xTest, yTest, 'b.', markersize=4)
plt.show()
```

GRAPH OUTPUT



From the graph, it is obvious that the given dataset is really noisy in nature.

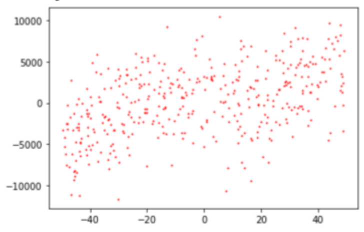
Graphing Each Of The Training Datasets

Here we graph each of the training datasets separately. (to check if the datasets are sampled properly)

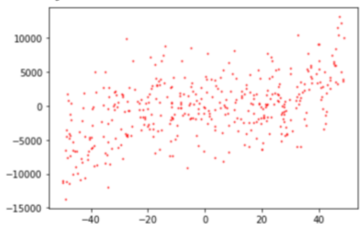
```
for i in range(20):
    print("Training set ",i)
    fig = plt.figure()
    plt.plot(X_train_data_sets[i], Y_train_data_sets[i], 'r.', markersize=2)
    plt.show()
```

GRAPH OUTPUT

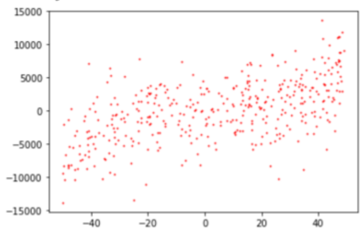
```
Training data  0
```

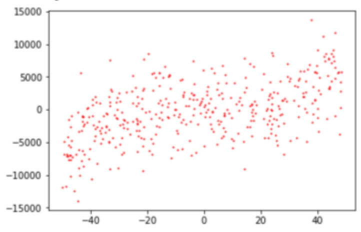


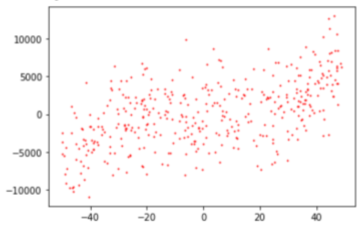
```
Training data  1
```

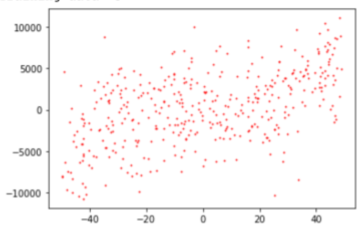


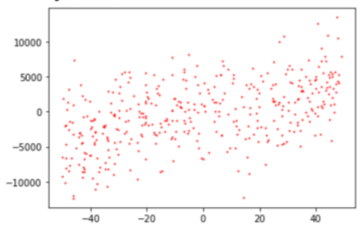
```
Training data  2
```

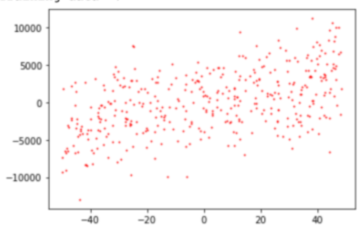


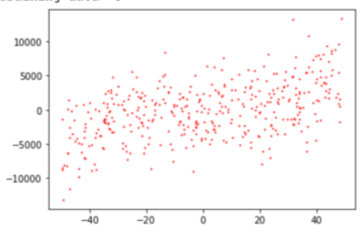
```
Training data  3
```


```
Training data  4
```


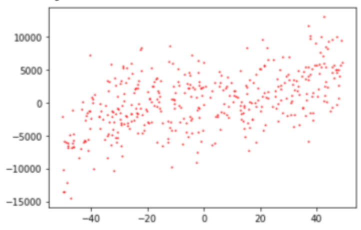
```
Training data  5
```


```
Training data  6
```


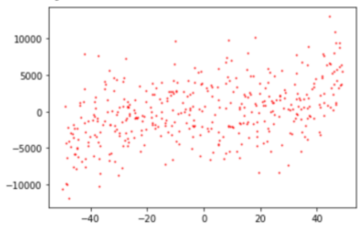
```
Training data  7
```


```
Training data  8
```


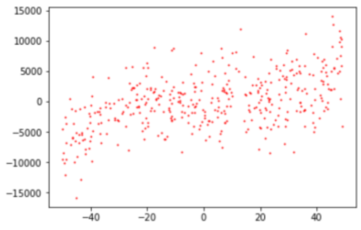
Training data 9



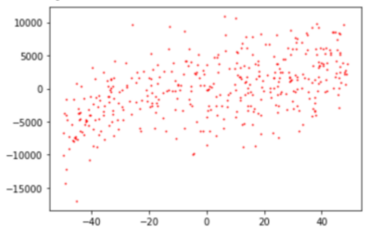
Training data 10



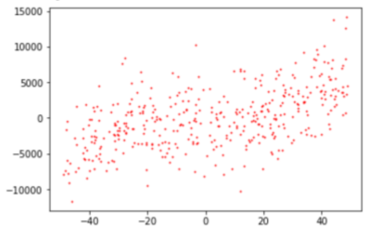
Training data 11



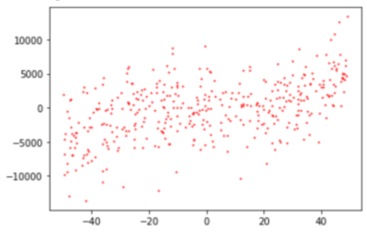
Training data 12



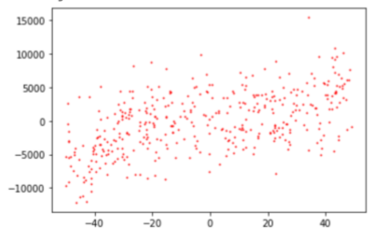
Training data 13



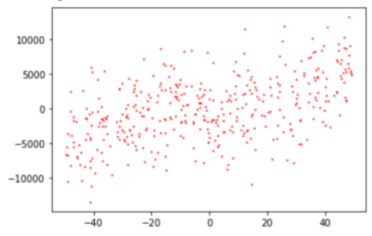
Training data 14



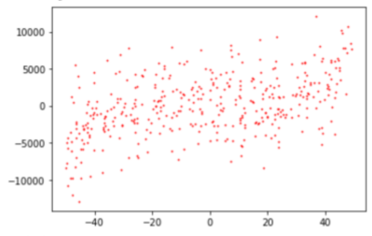
Training data 15



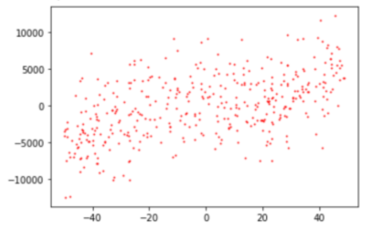
Training data 16



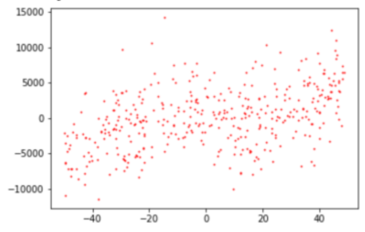
Training data 17



Training data 18



Training data 19



STEP 3: TRAINING A MODEL

Function just to store what exactly to be done to train a polynomial regression model of a given degree and given training dataset number

The following function returns a polynomial regression model for the given degree. I use this to find the lines required to make a model, of a given degree

```
def create_polynomial_regression_model(degree,i):
    i = 0
    # prepare the matrix of the powers of x
    poly_features = PolynomialFeatures(degree=degree)
```

```
  # transpose the x row matrix into a column matrix
  x = X_train_data_sets[i][:, numpy.newaxis]
  # get a matrix containing the higher powers of X in the format: [1 X X^2 X^3 ...]
  X_train_poly = poly_features.fit_transform(x)
  poly_model = LinearRegression()
  # fit the transformed features to Linear Regression
  poly_model.fit(X_train_poly, Y_train_data_sets[0])
  y_test_predict = poly_model.predict(
    poly_features.fit_transform(xTest))    # predicting on test data-set
  return poly_model
```

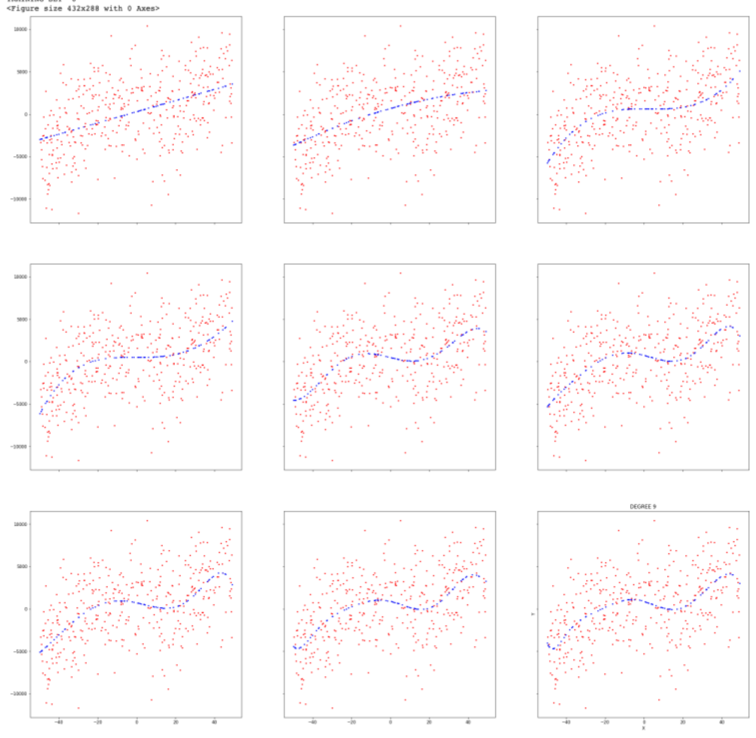
Plotting A Graph Of The Trained Polynomial Regression Model

Here, we take each of the training datasets, and plot the training dataset points and the values predicted by the model on the test dataset points, to visualise the provided data.
For each training set, 9 graphs are plotted, each corresponding to the model of each degree (from 0 to 9)

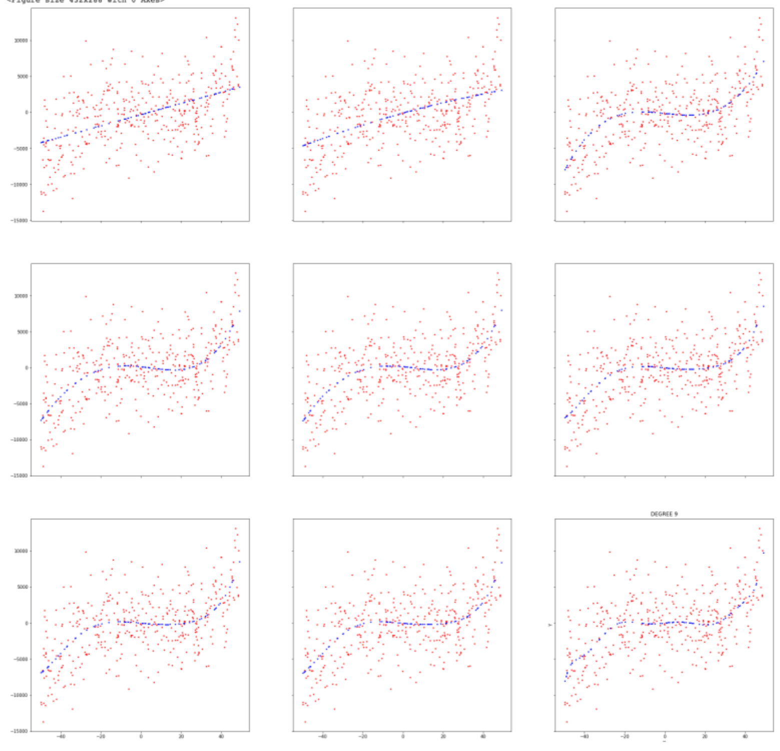
```
for i in range(10):
  print("TRAINING SET ", i)
  f = plt.figure()
  f, axes = plt.subplots(nrows=3, ncols=3, sharex=True, sharey=True, figsize=(30, 30))
  x = X_train_data_sets[i][:, numpy.newaxis]  # transposing it
  y = Y_train_data_sets[i]
  for degree in range(0, 9):
    axes[int(degree/3)][int(degree % 3)].plot(x, y, 'r.', markersize=4)
    poly_features = PolynomialFeatures(degree=degree+1)
    X_train_poly = poly_features.fit_transform(x)
    poly_model = LinearRegression()
    poly_model.fit(X_train_poly, Y_train_data_sets[i])
    y_test_predict = poly_model.predict(
      poly_features.fit_transform(xTest[:, numpy.newaxis]))
    axes[int(degree/3)][int(degree % 3)].plot(xTest[:, numpy.newaxis], y_test_predict, 'b.', markersize=4)
    plt.title("DEGREE "+str(degree+1))
    plt.xlabel("X")
    plt.ylabel("Y")
  plt.show()
```

GRAPH OUTPUT

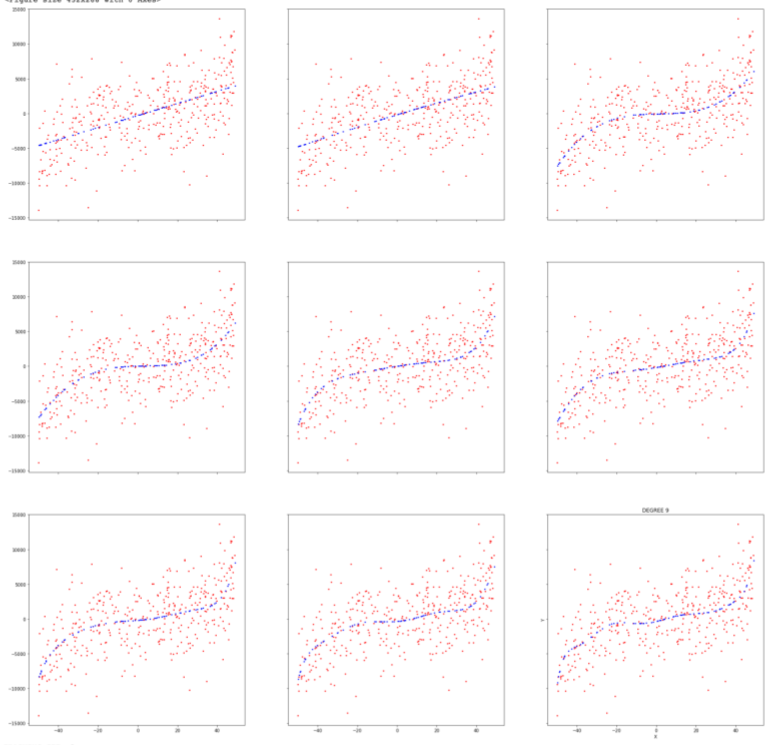
```
TRAINING SET  0
<Figure size 432x288 with 0 Axes>
```



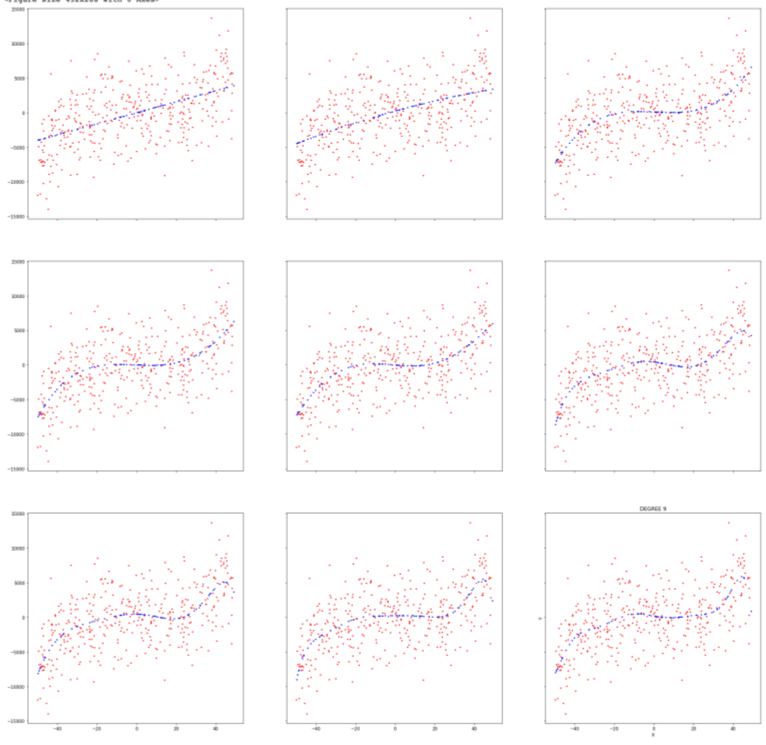
```
TRAINING SET  1
<Figure size 432x288 with 0 Axes>
```



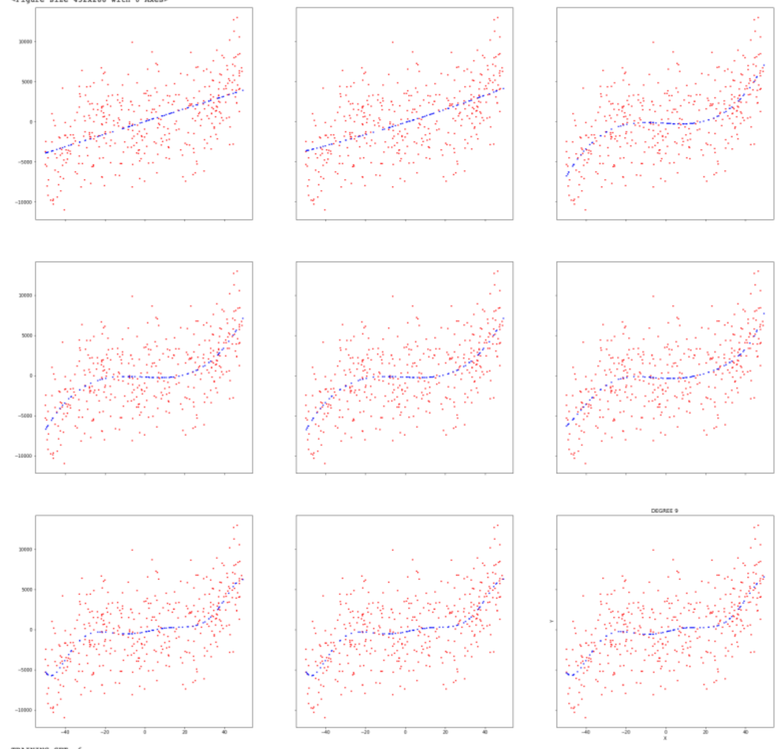
```
TRAINING SET  2
<Figure size 432x288 with 0 Axes>
```



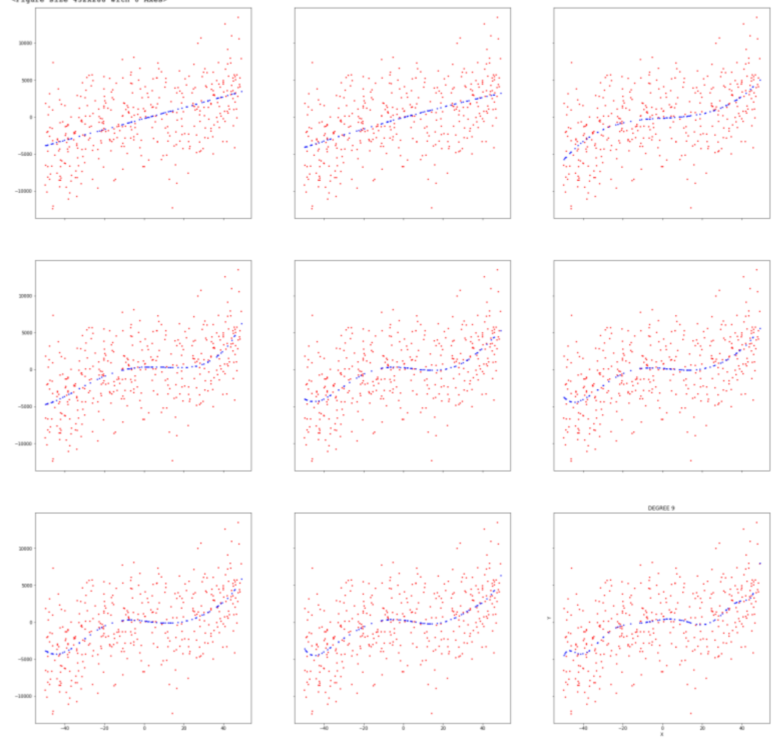
```
TRAINING SET  3
<Figure size 432x288 with 0 Axes>
```



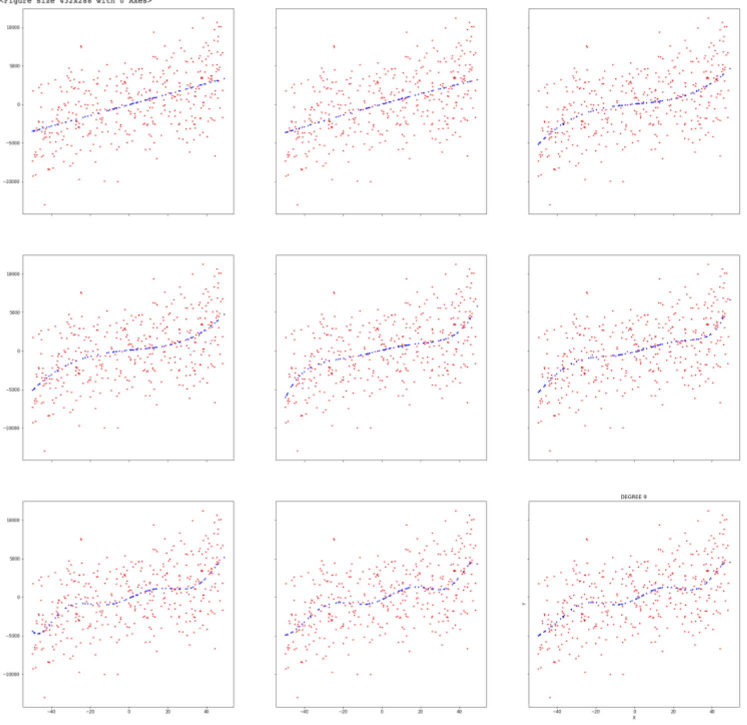
```
TRAINING SET  4
<Figure size 432x288 with 0 Axes>
```



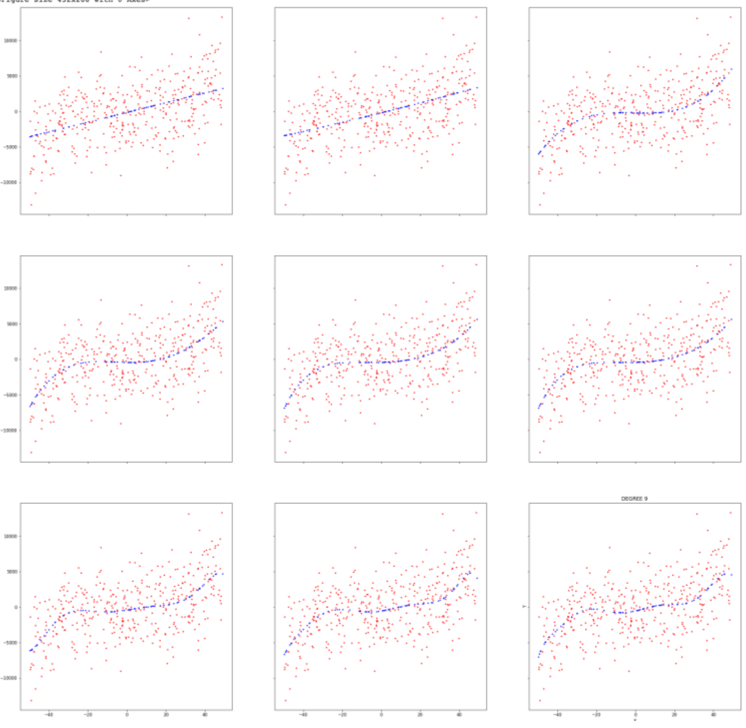
```
TRAINING SET  6
<Figure size 432x288 with 0 Axes>
```



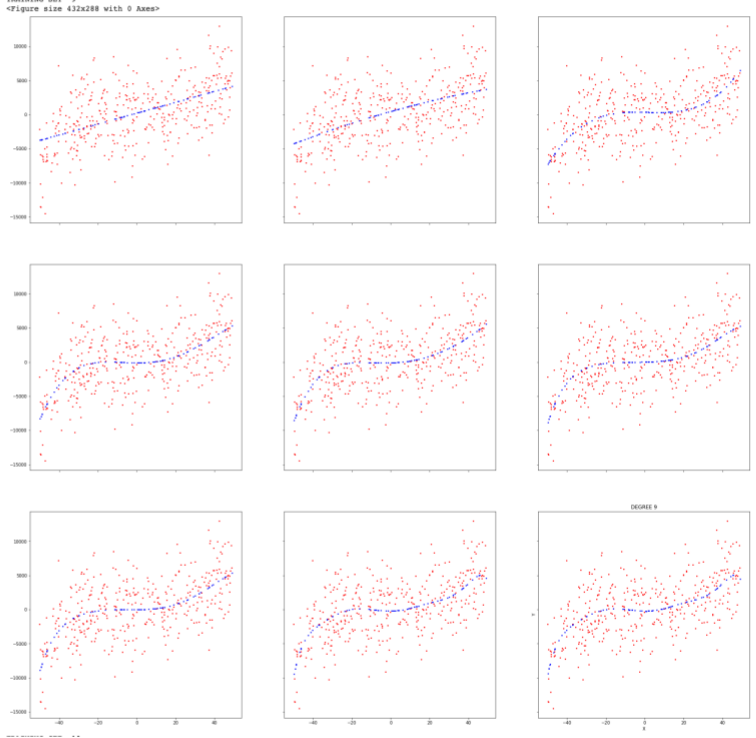
```
TRAINING SET  7
<Figure size 432x288 with 0 Axes>
```



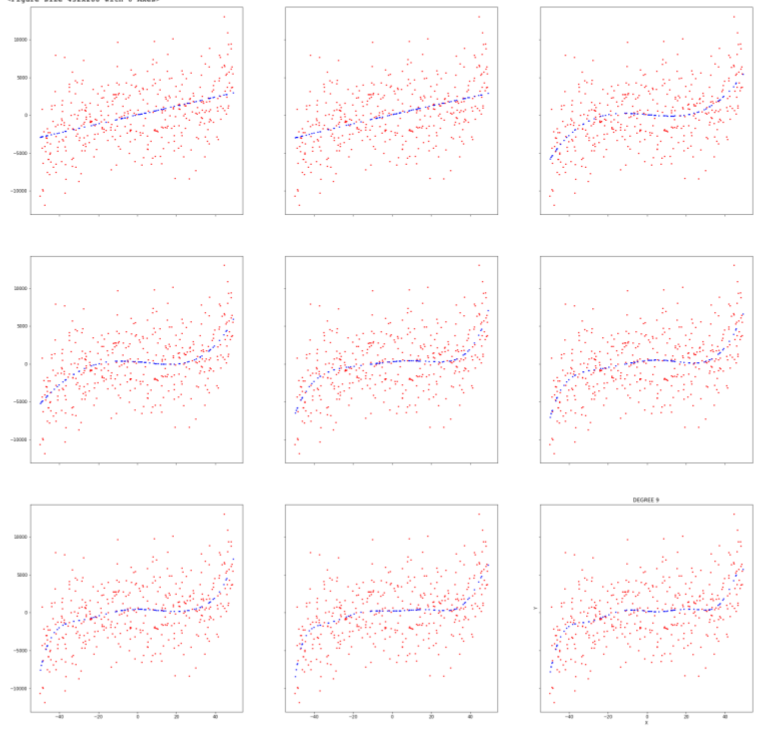
```
TRAINING SET  8
<Figure size 432x288 with 0 Axes>
```



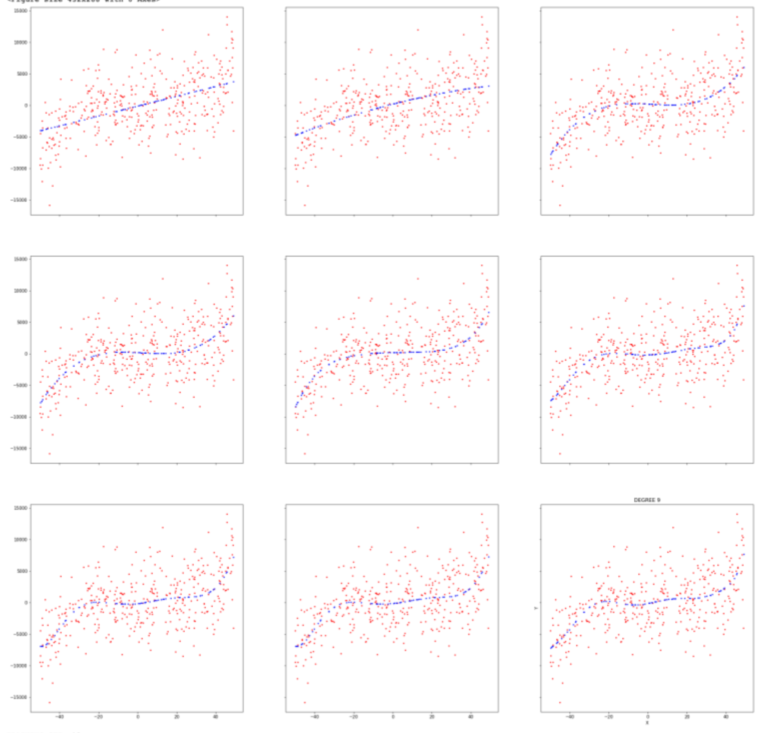
```
TRAINING SET  9
<Figure size 432x288 with 0 Axes>
```



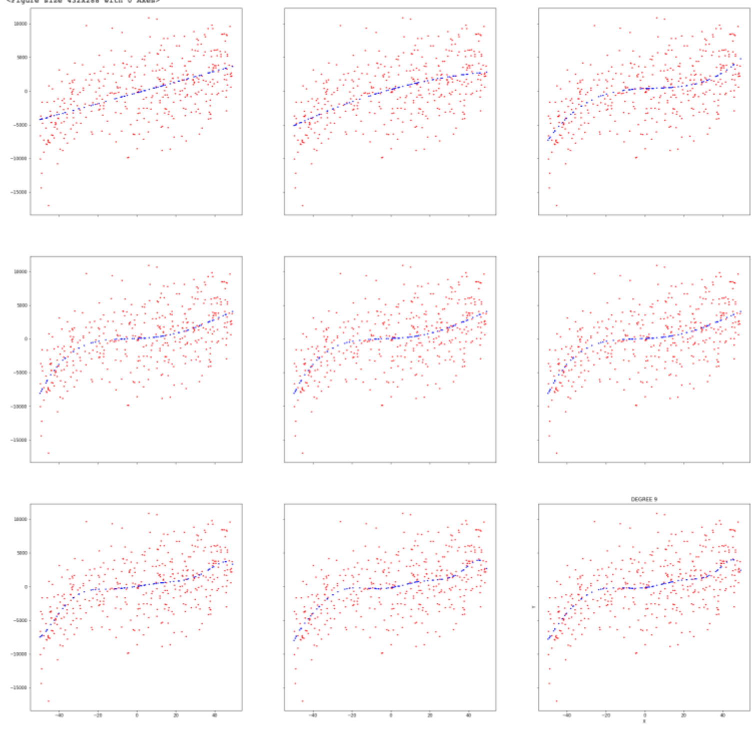
```
TRAINING SET  10
<Figure size 432x288 with 0 Axes>
```



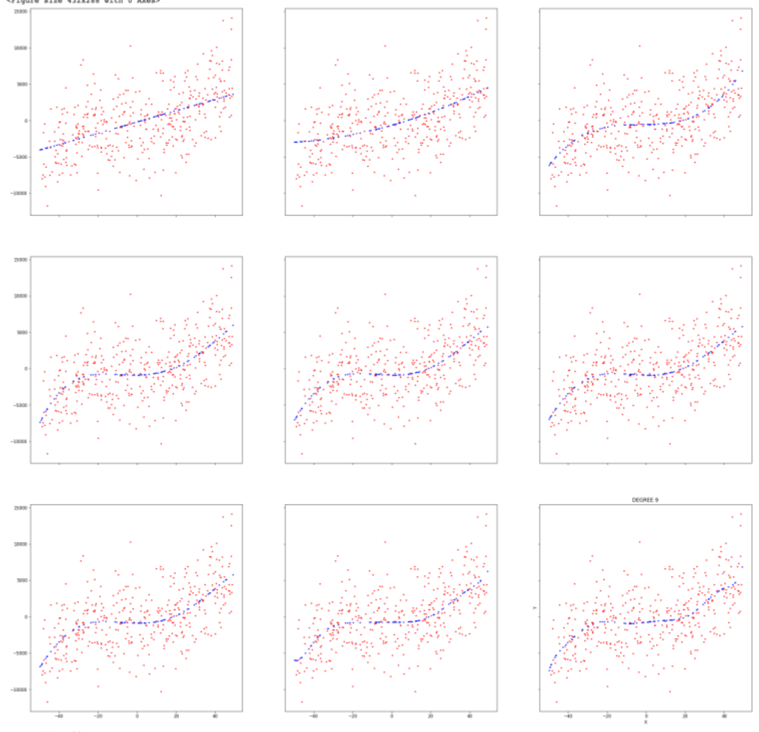
```
TRAINING SET  11
<Figure size 432x288 with 0 Axes>
```



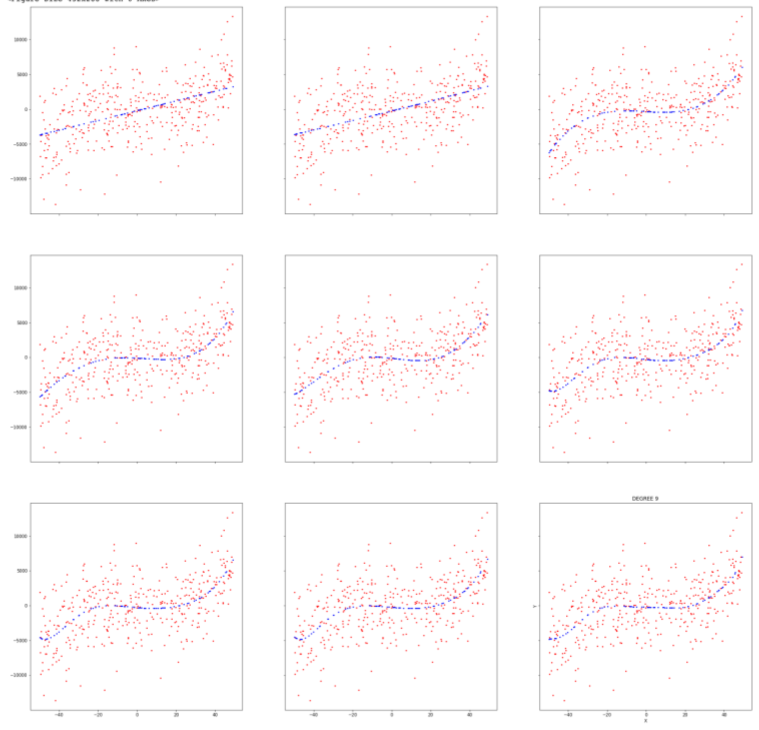
```
TRAINING SET  12
<Figure size 432x288 with 0 Axes>
```



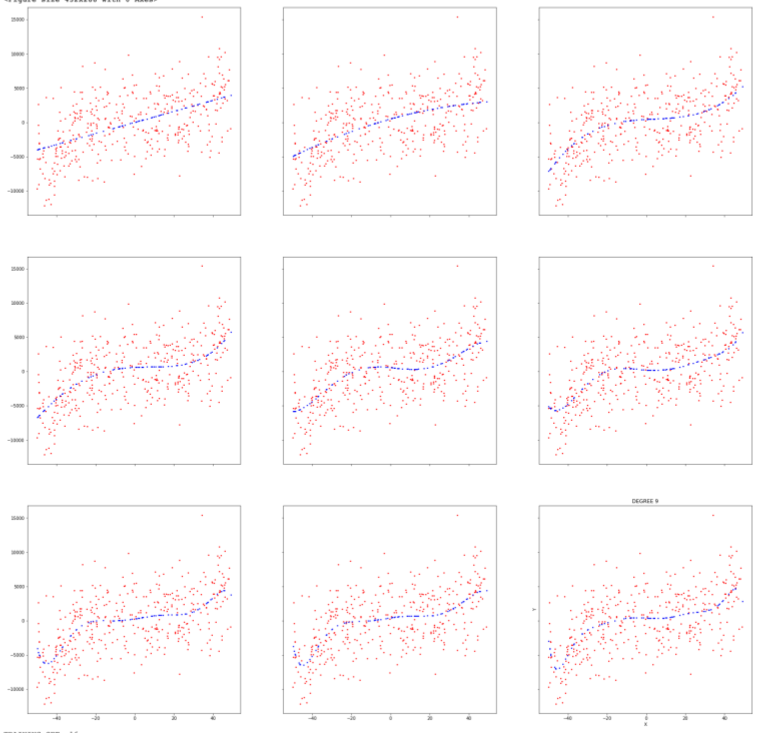
```
TRAINING SET  13
<Figure size 432x288 with 0 Axes>
```



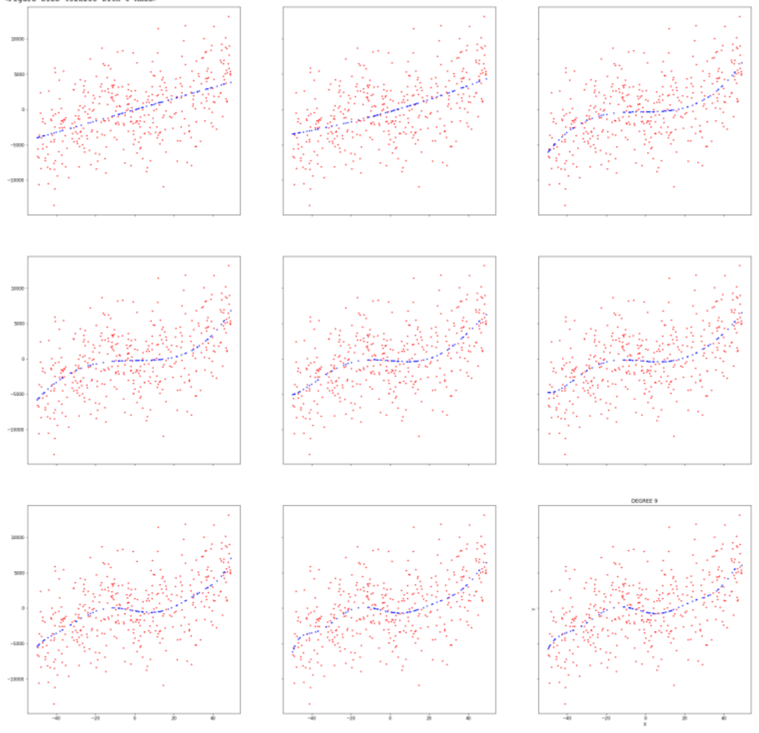
```
TRAINING SET  14
<Figure size 432x288 with 0 Axes>
```



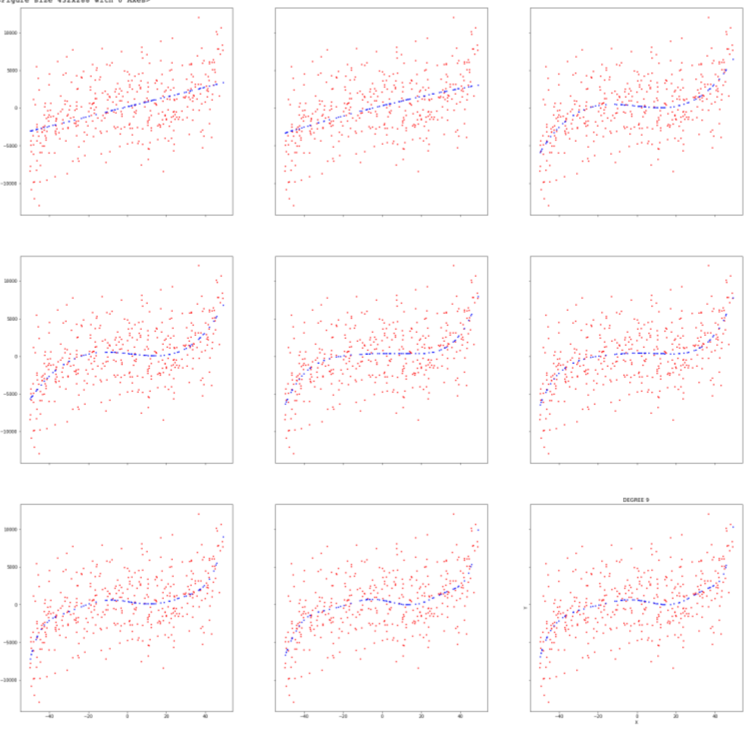
```
TRAINING SET  15
<Figure size 432x288 with 0 Axes>
```



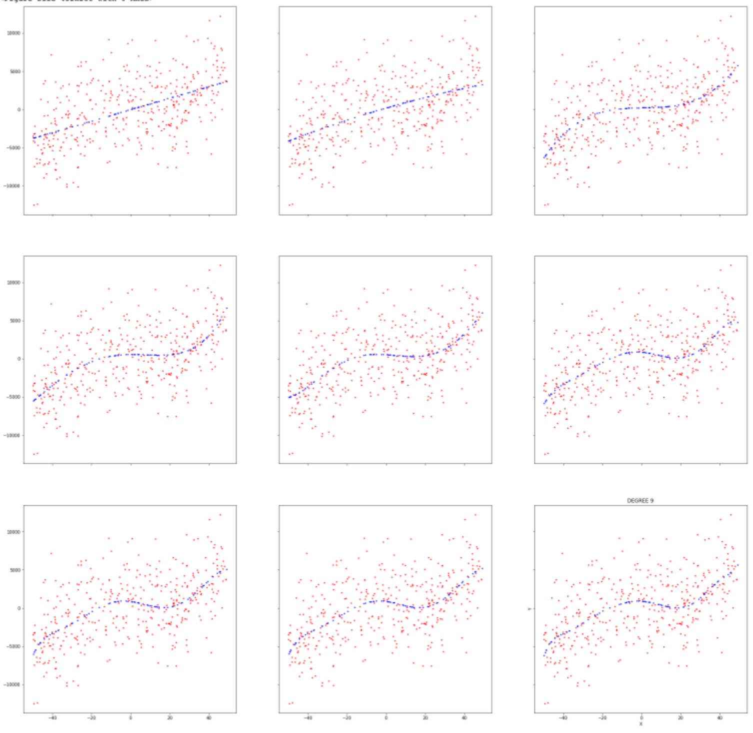
```
TRAINING SET  16
<Figure size 432x288 with 0 Axes>
```



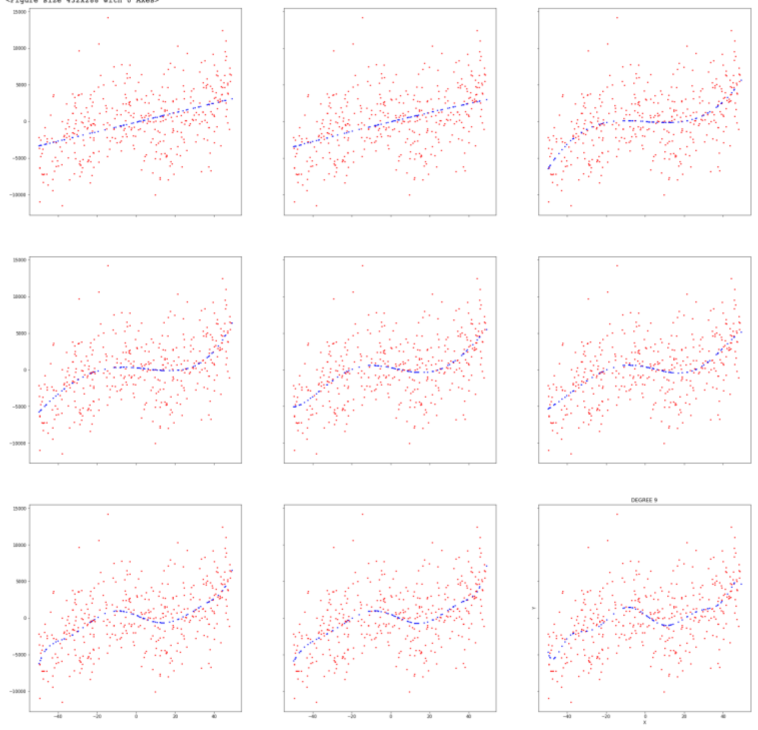
```
TRAINING SET  17
<Figure size 432x288 with 0 Axes>
```



```
TRAINING SET  18
<Figure size 432x288 with 0 Axes>
```



```
TRAINING SET  19
<Figure size 432x288 with 0 Axes>
```



STEP 4: CALCULATE THE BIAS AND VARIANCE OF THE MODEL

Get the list of all the predicted values

First we get the list of all the y predicted values for all the models and for all the degrees separately in a 2-D array.
Here, for each model of each degree, we get the predicted y values for the given test datasets.
The values are stored as follows: y[train_data_set_no][degree]

```
y_predicted = []
for i in range(10):
    x = X_train_data_sets[i][:, numpy.newaxis]  # transposing it
    y = Y_train_data_sets[i]
    temp = []
    for degree in range(0, 9):
        poly_features = PolynomialFeatures(degree=degree)
        X_train_poly = poly_features.fit_transform(x)
        poly_model = LinearRegression()
        poly_model.fit(X_train_poly, Y_train_data_sets[i])
        y_test_predict = poly_model.predict(
        poly_features.fit_transform(xTest[:, numpy.newaxis]))
      temp.append(y_test_predict)
    y_predicted.append(temp)
# y_predicted [5][3] means degree 3 and 5th model dataset
print(len(y_predicted[0][0]))
```

Function for calculating the bias and the variance

Then we calculate the bias and variance as follows:

- For a given degree we append the values of the y_predicted for each model to a list
- Convert this list to a numpy array y_predicted_part
- Calculate the bias of this list by subtracting the mean of the model from the testing dataset
- Bias corresponding to the models of a given degree is the mean of this list
- Similarly calculate the variance of this list
- Variance corresponding to the models of a given degree is the mean of this list

```
def find_bias_variance(order):
    y_predicted_part = []
    for i in range(20):
        y_predicted_part.append(y_predicted[i][order])
    y_predicted_part = numpy.asarray(y_predicted_part)
    bias = numpy.abs(numpy.mean(y_predicted_part, axis=0) - yTest)
    variance = numpy.var(y_predicted_part, axis=0)
    return(numpy.mean(bias), numpy.mean(variance))
```

Then we call the function as follows, in order to populate the lists, bias and variance.

```
Bias = []
variance = []
for i in range(9):
    b, v = find_bias_variance(i)
    bias.append(b)
    variance.append(v)
print("Bias:", bias)
print("Variance:", variance)
```

The lists, bias and variance, now contain the bias and variance corresponding to a particular degree.

OUTPUT

```
Bias: [1716.3588857838863, 819.8378604641326, 810.8402138398206, 67.63398967084957,
84.00189151242087, 79.18712744433826, 80.09962642308372, 84.97054138073908,
85.58629028495918]
```

```
Variance: [20935.166423224564, 70545.48914575046, 125870.85554877335,
150073.7395464768, 212235.70832526154, 276388.4802547406, 316863.49843748985,
357510.98475735466, 404286.670685786]
```

Tabulate the values

We use the pandas library in order to display the required items in a table format

```
final_table = dict()
final_table["DEGREE"] = range(1, 10)
final_table["BIAS"] = bias
final_table["BIAS^2"] = list(numpy.array(bias)**2)
final_table["VARIANCE"] = variance
final_table["MSE"] = list(numpy.array(final_table["BIAS^2"])+numpy.array(variance))
df = pandas.DataFrame(final_table)
print(df)
```

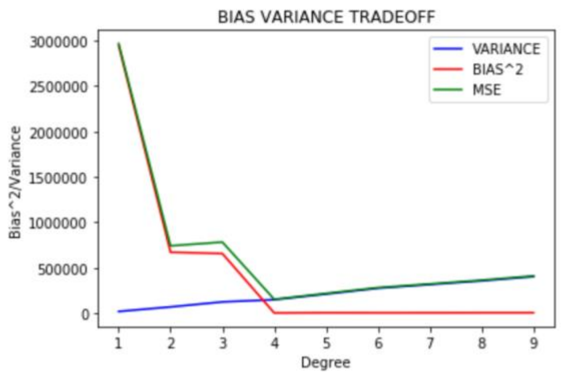
OUTPUT

```
   DEGREE         BIAS        BIAS^2       VARIANCE           MSE
0       1  1716.358886  2.945888e+06   20935.166423  2.966823e+06
1       2   819.837860  6.721341e+05   70545.489146  7.426796e+05
2       3   810.840214  6.574619e+05  125870.855549  7.833327e+05
3       4    67.633990  4.574357e+03  150073.739546  1.546481e+05
4       5    84.001892  7.056318e+03  212235.708325  2.192920e+05
5       6    79.187127  6.270601e+03  276388.480255  2.826591e+05
6       7    80.099626  6.415950e+03  316863.498437  3.232794e+05
7       8    84.970541  7.219993e+03  357510.984757  3.647310e+05
8       9    85.586290  7.325013e+03  404286.670686  4.116117e+05
```

Plot the bias-variance tradeoff

```
plt.plot(final_table["DEGREE"], final_table["VARIANCE"], color="blue")
plt.plot(final_table["DEGREE"], final_table["BIAS^2"], color="red")
plt.plot(final_table["DEGREE"], final_table["MSE"], color="green")
plt.title("BIAS VARIANCE TRADEOFF")
plt.xlabel("Degree")
plt.ylabel("Bias^2/Variance")
plt.legend(["VARIANCE", "BIAS^2", "MSE"])
plt.show()
```

GRAPH OUTPUT



OBSERVATIONS AND INFERENCES

From the bias variance tradeoff graph it is obvious that,

- as the degree of the polynomial regression that we try to fit the data with increases, the bias value decreases. This is because of overfitting; that is the model now trained, has a lot of additional features and hence is able to fit in more dataset points, leading to less bias.
- as the degree of the polynomial regression that we try to fit the data with increases, the variance value increases. This is because of overfitting; now, the model tries to predict the noise component of the dataset too, hence leading to higher values of variance

From the above plot, the given dataset is more likely to be of the degree 3, because of the following reasons:

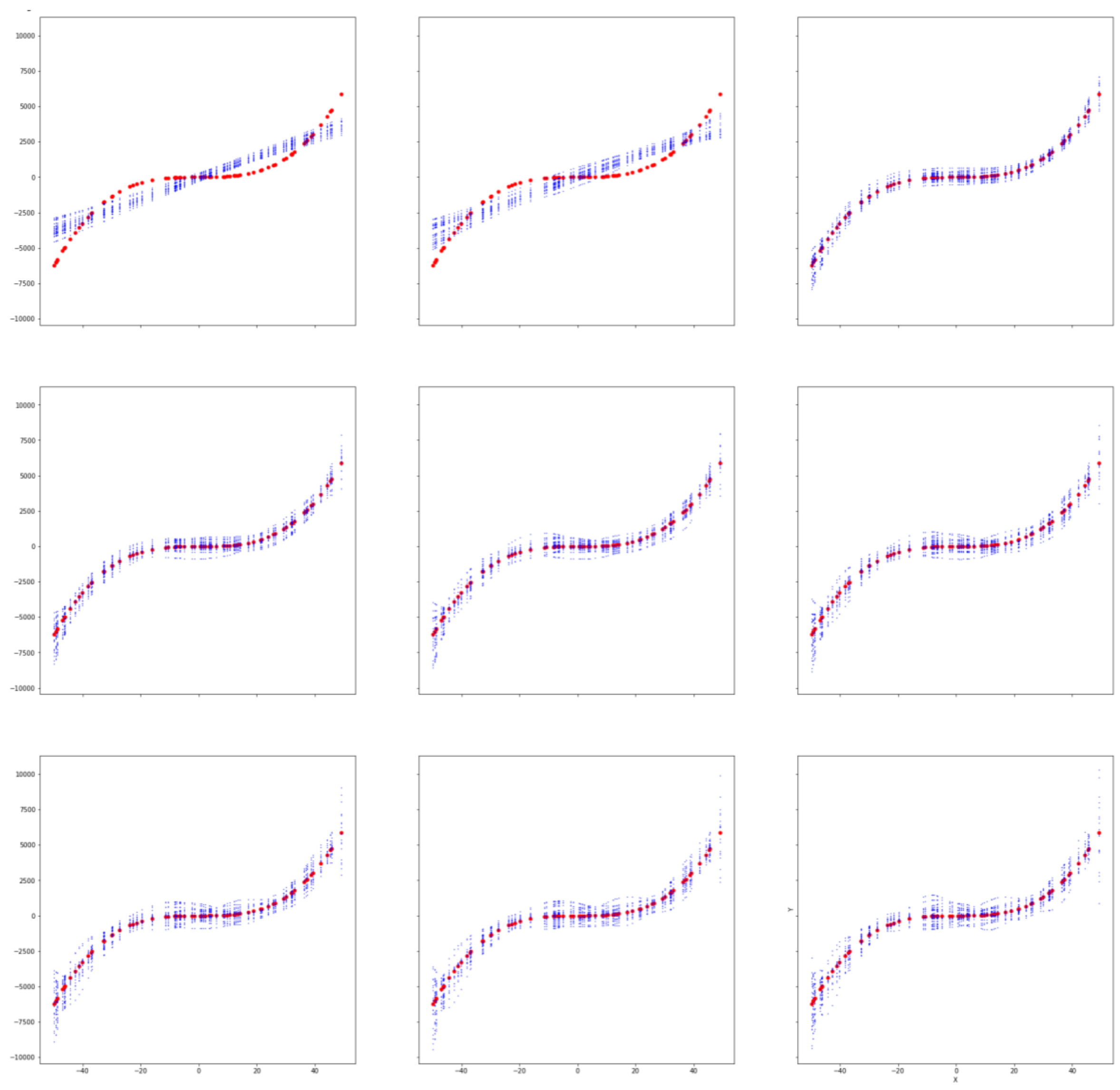
- from the bias-variance tradeoff graph, we see that the total error(MSE) keeps on decreasing till 4, after which it increases.
- The models with complexity above 3 degrees have high variance and low bias, and hence are overfit.
- The models with complexity below 3 degrees have low variance but high bias, and hence are underfit.
- since the dataset used is really noisy, the bias is lost really quickly with higher models.

FITTING THE TRAINED MODEL TO THE TESTING DATASET FOR DISPLAYING THE LINE OF BEST FIT

The code snippet

```
f = plt.figure()
f, axes = plt.subplots(nrows = 3, ncols = 3, sharex=True, sharey = True,figsize=(30,30))
for degree in range(0,9):
    xtemp=numpy.concatenate([xTest for i in range(20)])
    y_predicted_part=[]
    for i in range(20):
        y_predicted_part.append(y_predicted[i][degree])
    ytemp=numpy.array(y_predicted_part).reshape(-1)
    axes[int((degree)/3)][int((degree)%3)].plot(xTest, yTest, 'r.',markersize=10)
    axes[int((degree)/3)][int((degree)%3)].plot(xtemp, ytemp,'b.',markersize=1)
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

Output


10000
7500
5000
2500
0
−2500
−5000
−7500
−10000
−40
−20
0
20
40
X
Y