Report for Data Visualization(q2.py)

Explanation of libraries same as q1.py

```
class linear_regression:
    def __init__(self):
        self.test_data_x = []
        self.test_data_y = []
        self.split_train_data_x = []
        self.split_train_data_y = []
        self.split_test_data_x = []
        self.split_test_data_y = []
        self.bias = []
        self.variance = []
        self.b_final = []
        self.v_final = []
```

The above code is for storing the empty lists to the variables.

```
if __name__ == '__main__':
        main()
```

The above is the main code

```
def main():
    ob=linear_regression()
    ob.data_refactoring()
    ob.model_training()
    ob.b_v_averaging(ob.bias,ob.variance)
    ob.plot_check(ob.b_final,ob.v_final)
```

The above is the code snippet for main function.

```
def bias_variance_calculation(self, x_test, y_test, y_predicted, j):
        bias_total = 0
        variance_total = 0
        E_y_predicted = y_predicted.mean()
        for i in range(80):
            bias = (y_predicted[i]- y_test[i])**2
            bias_total += bias
        variance_total = statistics.variance(y_predicted)
        bias_total /= 80
        # if j == 0: bias_total = None; variance_total=None
        print(j,"degree :-",bias_total," | ",variance_total)
        self.bias.append(bias_total)
        self.variance.append(variance_total)
```

The above function is to calculate the bias and variance:

At first bias_total and varaince_total are intialized to 0.Then mean() is statistics module function that used to calculate average of numbers and list and the value of y_predicted.mean() is stored into E_y_predicted variable(Expected y_predicted value)

In a range of 0-80 we will calculate the (bias)^2 and we add bias value to bias_total

statistics.variance(y_predicted)-This function helps to calculate the variance from a sample of data(here y_predicted can be taken as a list or matrix) and that value is stored into variance_total variable.

print(j,"degree :-",bias_total," | ",variance_total)-this is for printing in the format <jvalue> degree:-<bias_total value> | <variance_total value>

self.bias.append(bias_total)

self.variance.append(variance_total)

The above is for appending the bias_total value to bias list and variance_total value to variance list

---

```
def plot_check(self, x, y):
        plt.plot(range(10),x)
        plt.title('number of parameters vs Bias')
        plt.xlabel('Number of parameters')
        plt.ylabel('Bias^2')
        plt.show()
        plt.plot(range(10),y)
        plt.title('number of parameters vs Variance')
        plt.xlabel('Number of parameters')
        plt.ylabel('Variance')
        plt.show()

    def model_training(self):
        for i in range(10):
            for j in range(20):
                self.test_data_x = self.split_test_data_x
                self.test_data_y = self.split_test_data_y
                model = lr()
                poly=PolynomialFeatures(degree=i)
                x=self.split_train_data_x[j][...,np.newaxis]
                y=self.split_train_data_y[j][..., np.newaxis]
                x_=poly.fit_transform(x)
                x_test=poly.fit_transform(self.test_data_x[...,np.newaxis])
                model.fit(x_, y)
                predicted_y=model.predict(x_test)
                plt.plot(self.test_data_x[...,np.newaxis],self.test_data_y[...,np.newaxis],'o')
                plt.title('X vs Y')
                plt.xlabel('x')
                plt.ylabel('y')
                plt.plot(self.test_data_x,predicted_y.flatten(), 'o', color='black')
                plt.show()
                self.bias_variance_calculation(self.test_data_x, self.test_data_y,predicted_y.flatten(),i)
```

Train all 10 degrees on all 20 sets and average bias and variance over all 20 sets for each degree.

The above code snippet is about the information of the data used for plotting the graph

---

```
def b_v_averaging(self,b,v):
    for i in range(10):
        b=np.asarray(b)
        bias_total=b[i*20:(i+1)*20].mean()
        v=np.asarray(v)
        variance_total=v[i*20:(i+1)*20].mean()
        self.b_final.append(bias_total)
        self.v_final.append(variance_total)
        print(bias_total," | ",variance_total)
```

The function of b_v_averaging is Averages bias and Variance for 20 values,20 sets for each degree.

```
def data_refactoring(self):
    Y_test = open('./Q2_data/Fx_test.pkl', 'rb')
    X_test = open('./Q2_data/X_test.pkl','rb')
    Y_train = open('./Q2_data/Y_train.pkl','rb')
    X_train = open('./Q2_data/X_train.pkl','rb')
    self.split_test_data_x = pickle.load(X_test)
    self.split_test_data_y = pickle.load(Y_test)
    self.split_train_data_x = pickle.load(X_train)
    self.split_train_data_y = pickle.load(Y_train)
```

The above code is for data refactoring…

The process of loading a pickled file Fx_test.pkl,X_test.pkl,Y_train.pkl,X_train.pkl which are in Q2_data file into a python program and stored into Y_test,X_test,Y_train,X_train respectively. And 'rb' denotes r for read mode and b for byte mode.

```
def plot_check(self, x, y):
    plt.plot(range(10),x)
    plt.title('number of parameters vs Bias')
    plt.xlabel('Number of parameters')
    plt.ylabel('Bias^2')
    plt.show()
    plt.plot(range(10),y)
    plt.title('number of parameters vs Variance')
    plt.xlabel('Number of parameters')
    plt.ylabel('Variance')
    plt.show()
```

*The above code snippet is for plotting the graph (**number of parameters vs Bias and number of parameters vs Variance)***

plt.title('number of parameters vs Bias')

For keeping the title of the graph as ' ***number of parameters vs Bias'***

plt.xlabel('Number of parameters')

For labeling the x-axis as ' ***Number of parameters'***

plt.ylabel('Bias^2')

*For labeling the y-axis as '****Bias^2'***

plt.show()

For executing the graph

Similarly for the graph 'number of parameters vs Variance'

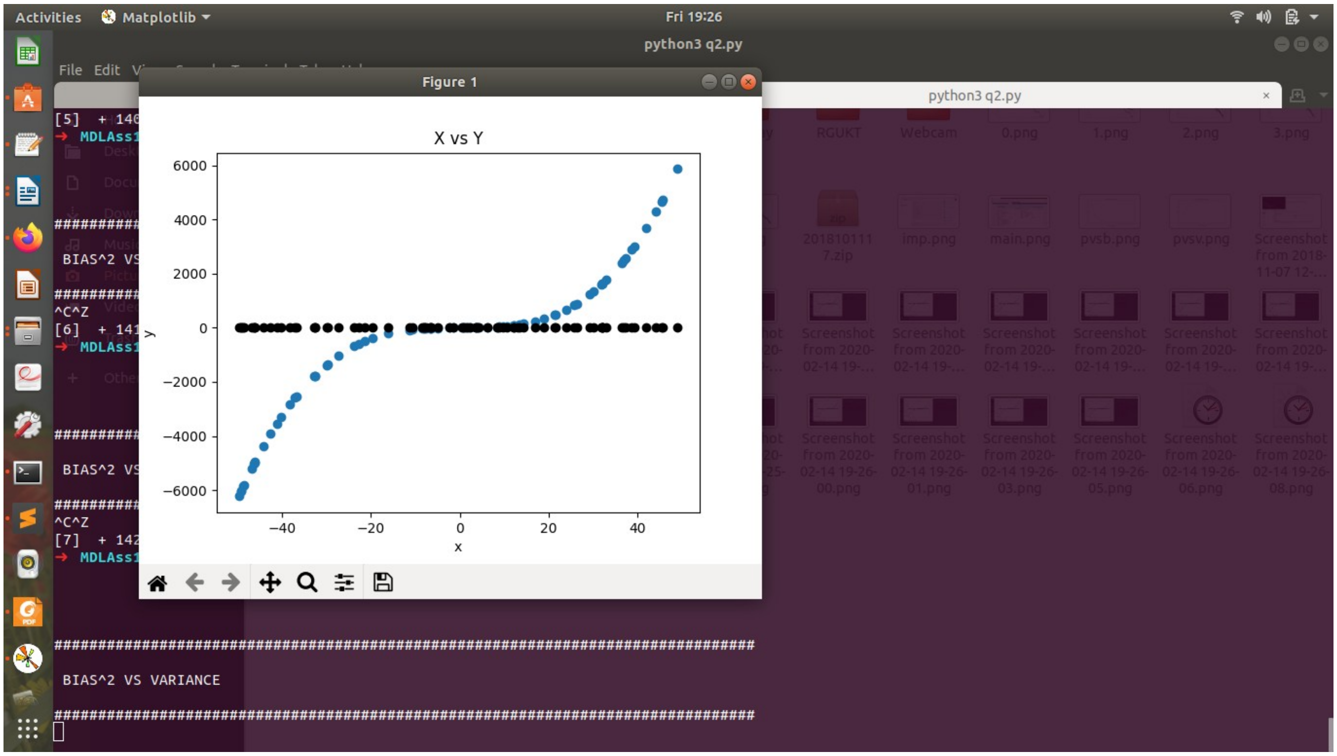


Figure 1
X vs Y
x
y
BIAS^2 VS VARIANCE

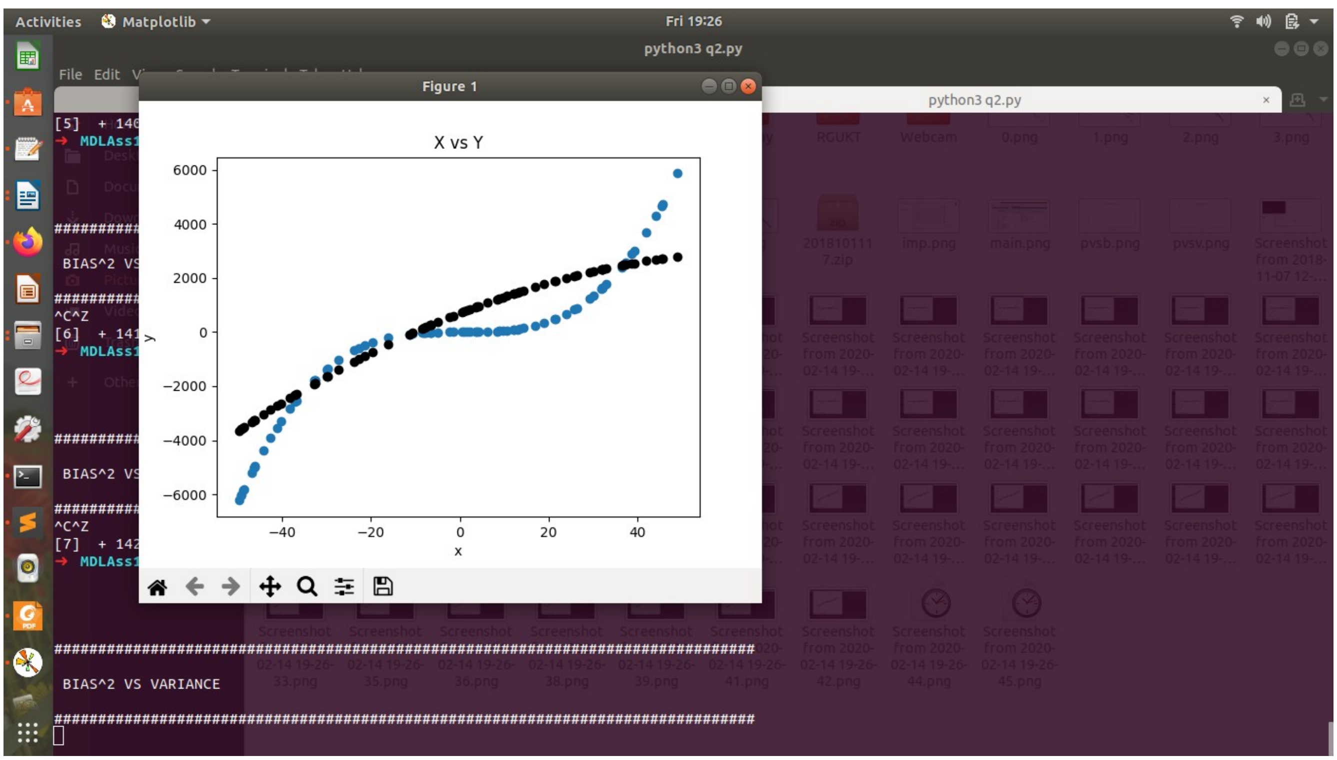
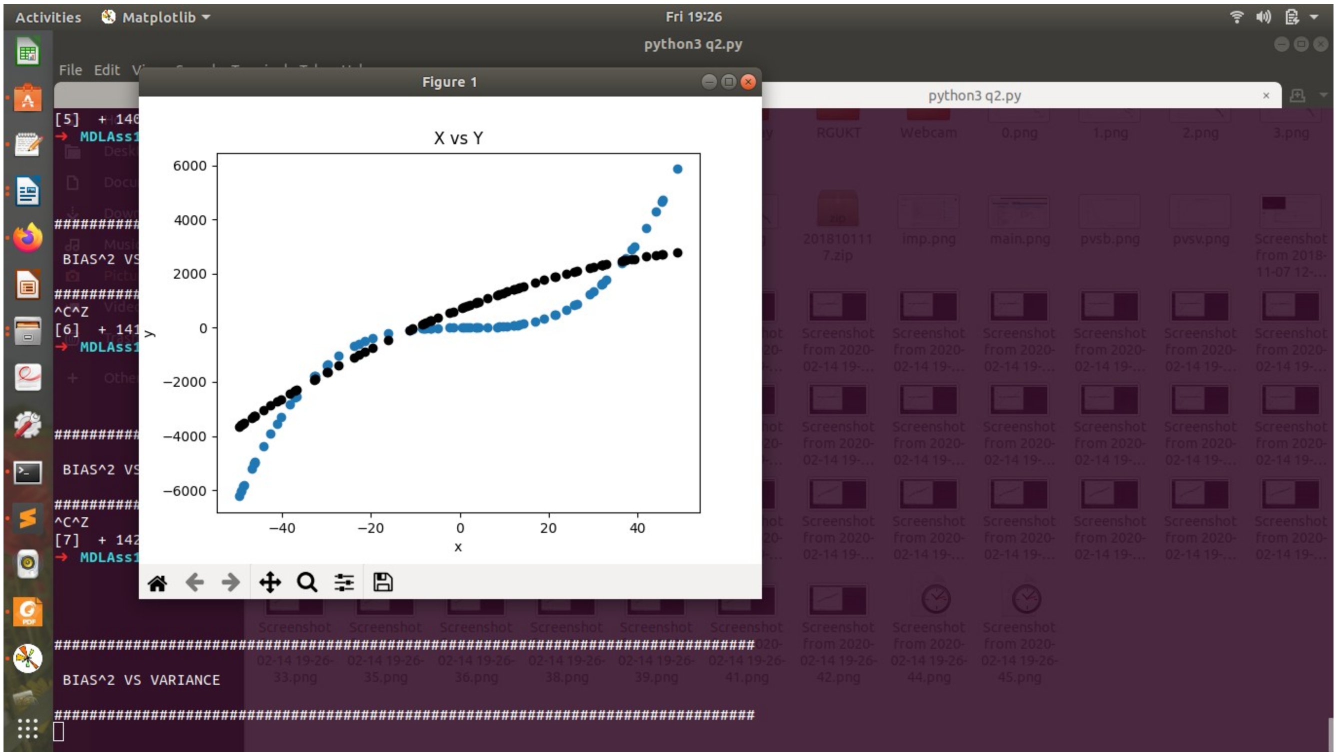
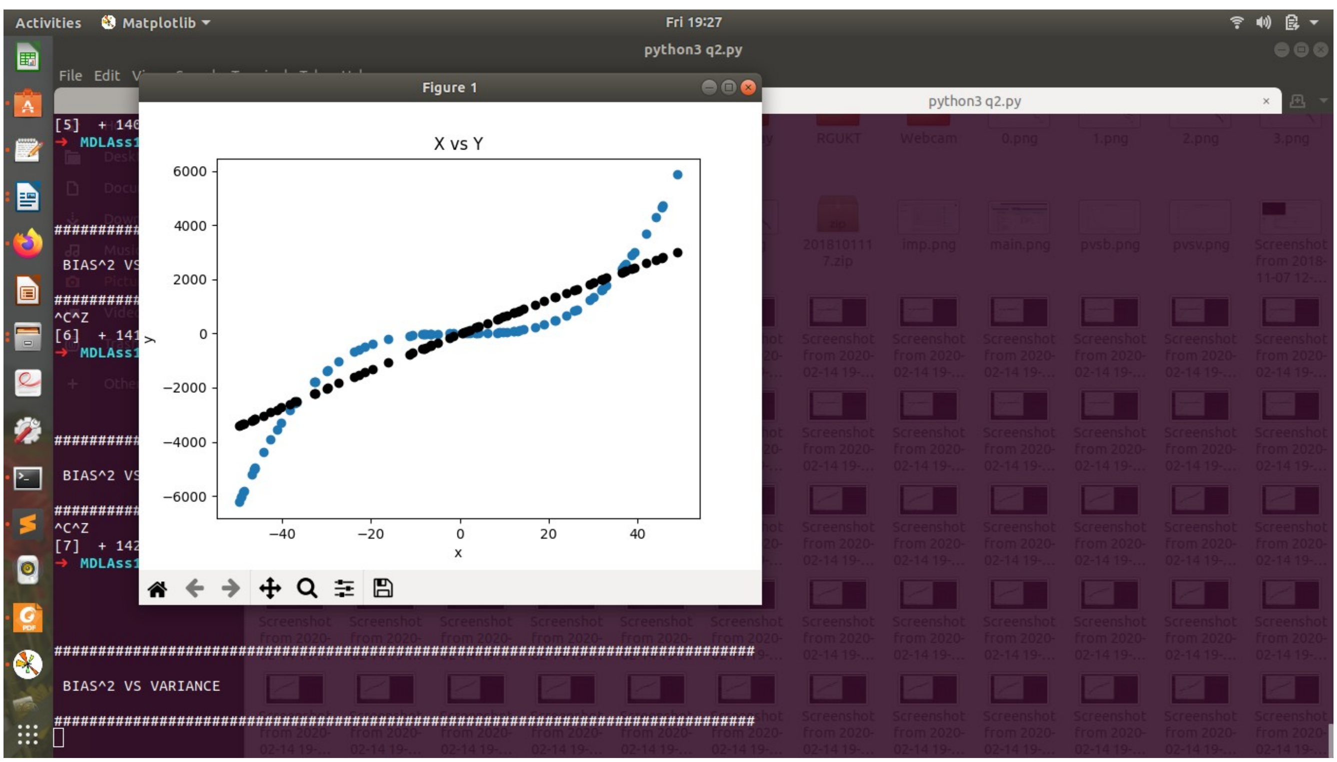
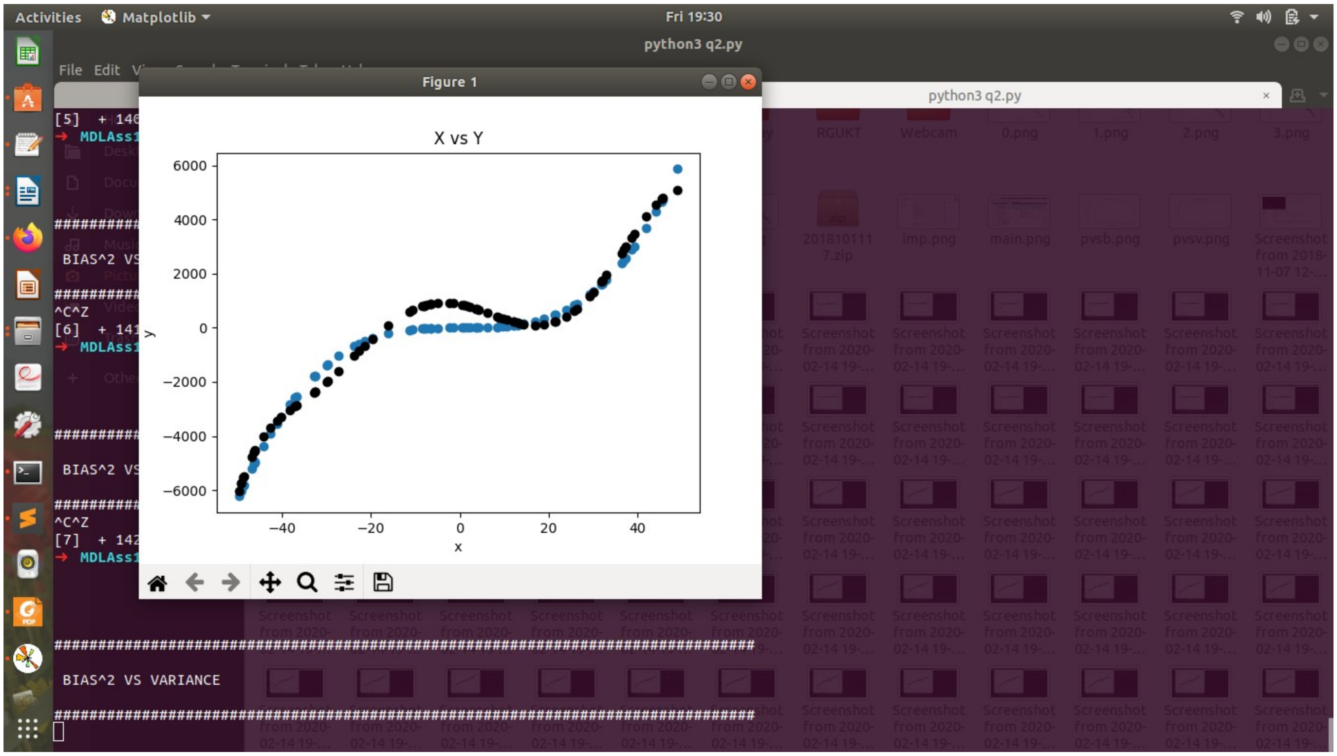


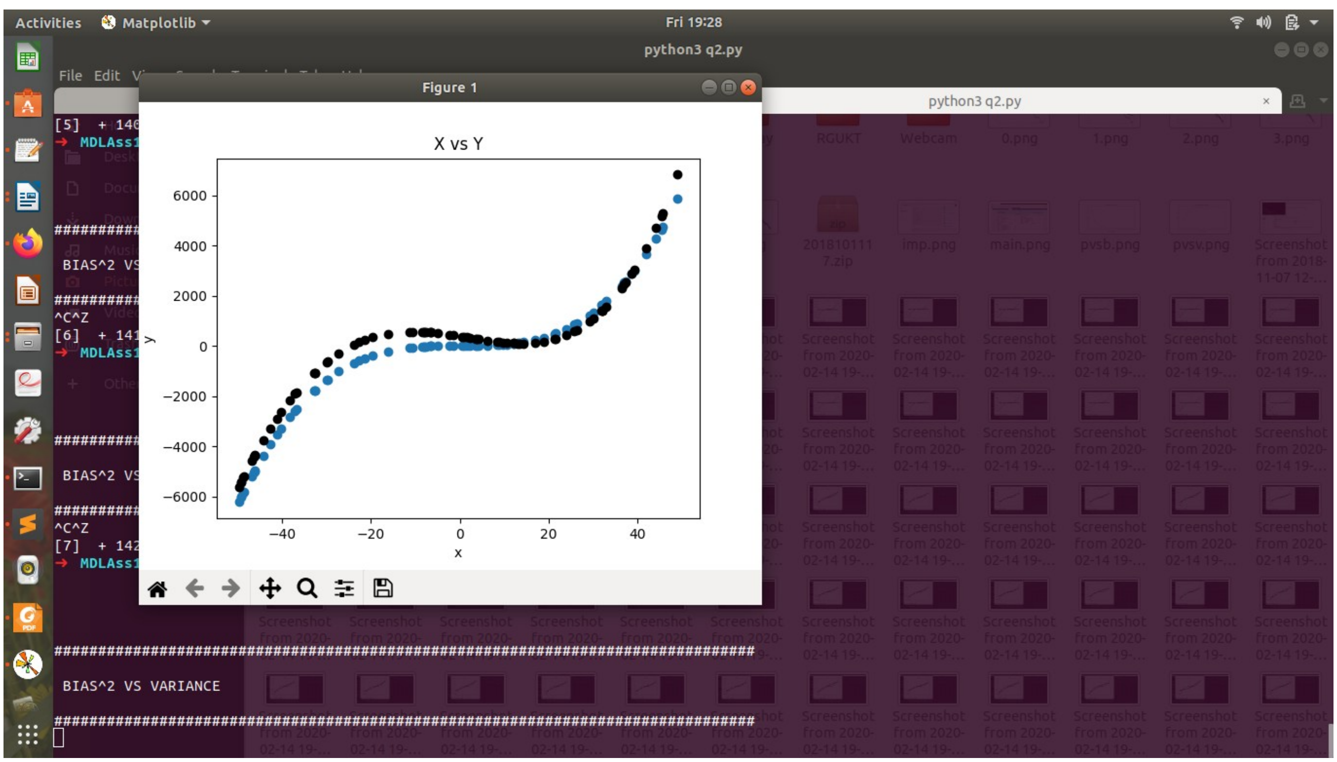
Figure 1
X vs Y
x
y
BIAS^2 VS VARIANCE


Activities
Matplotlib
Fri 19:26
python3 q2.py
Figure 1
X vs Y
6000
4000
2000
0
−2000
−4000
−6000
y
−40
−20
0
20
40
x
BIAS^2 VS VARIANCE

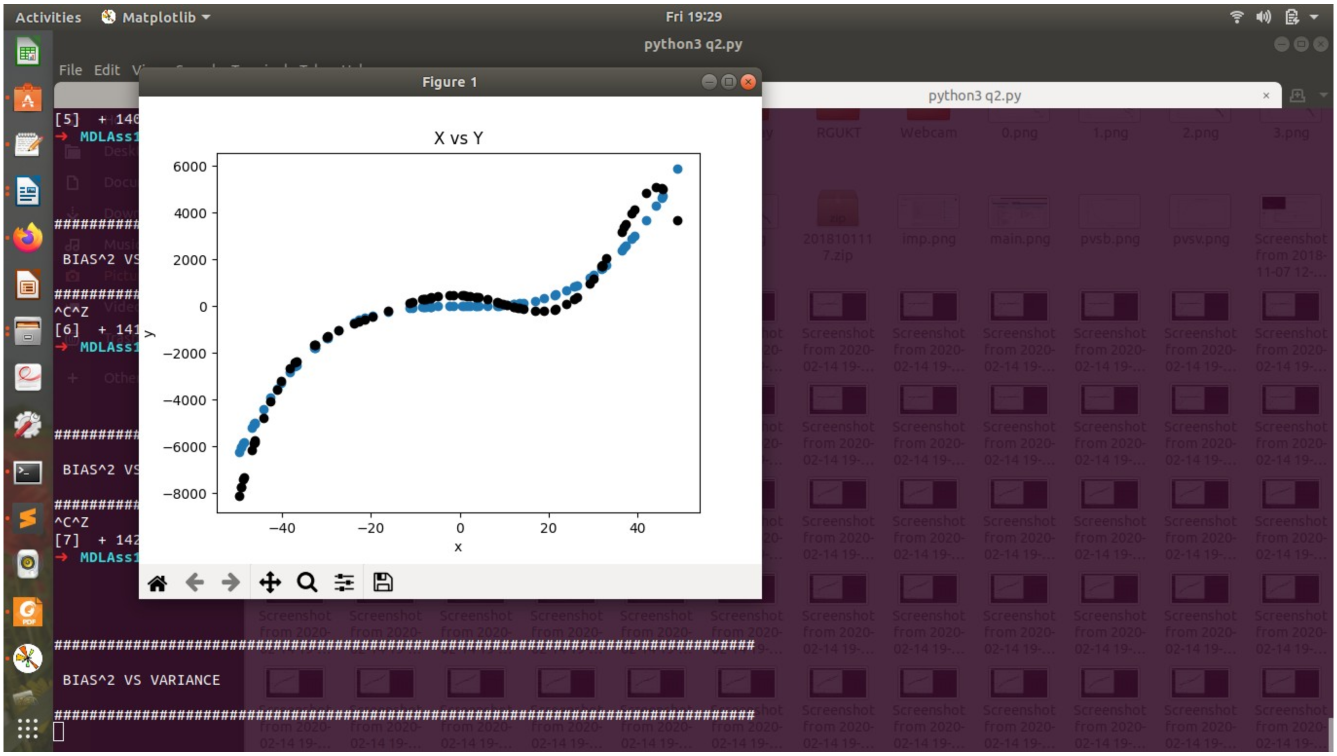

Activities
Matplotlib
Fri 19:27
python3 q2.py
Figure 1
X vs Y
6000
4000
2000
0
−2000
−4000
−6000
y
−40
−20
0
20
40
x
BIAS^2 VS VARIANCE

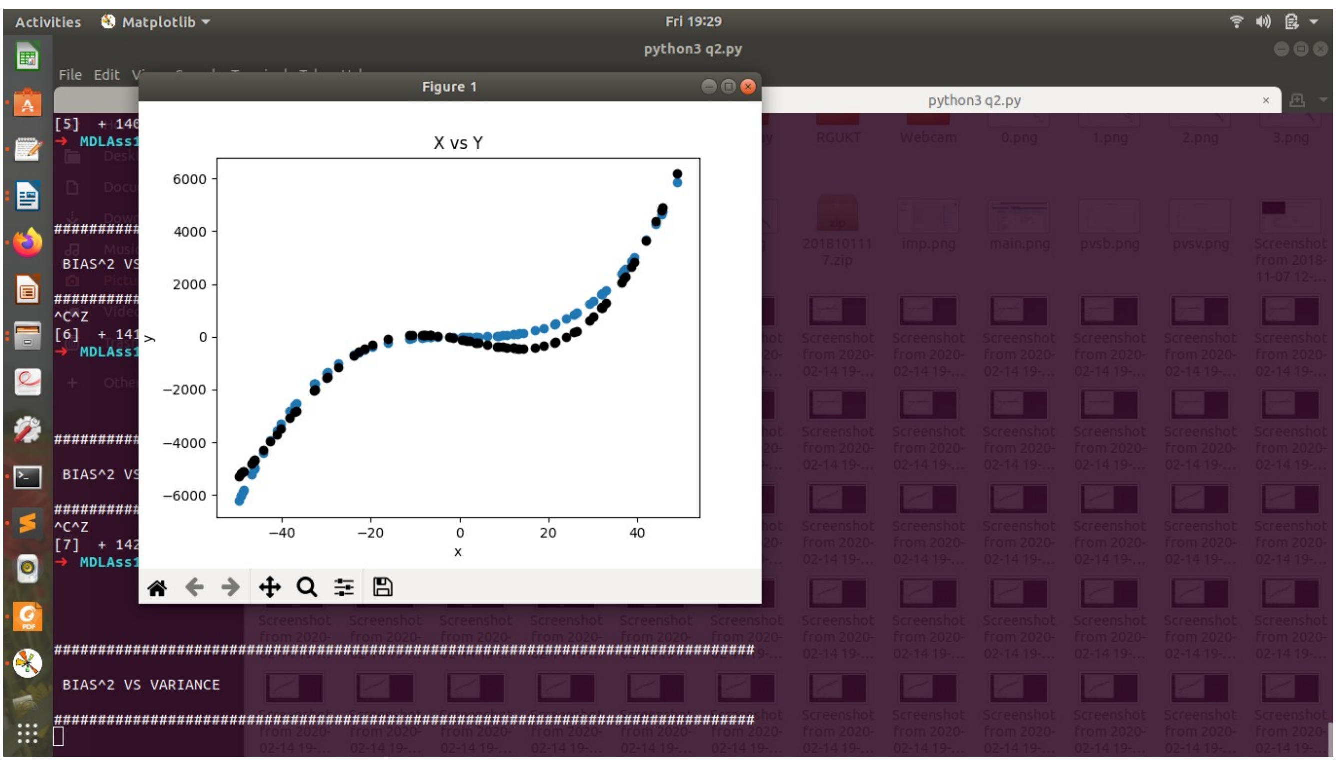


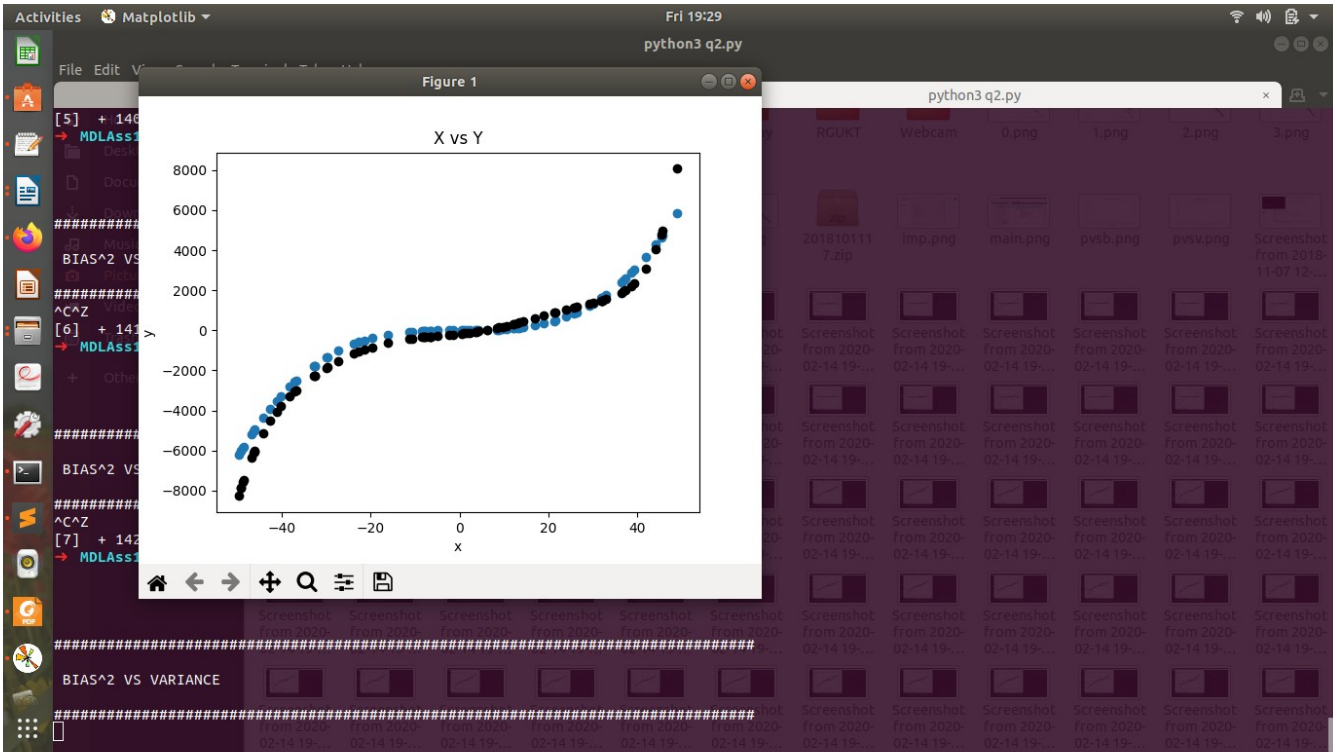
Activities
Matplotlib
Fri 19:30
python3 q2.py
Figure 1
X vs Y
BIAS^2 VS VARIANCE



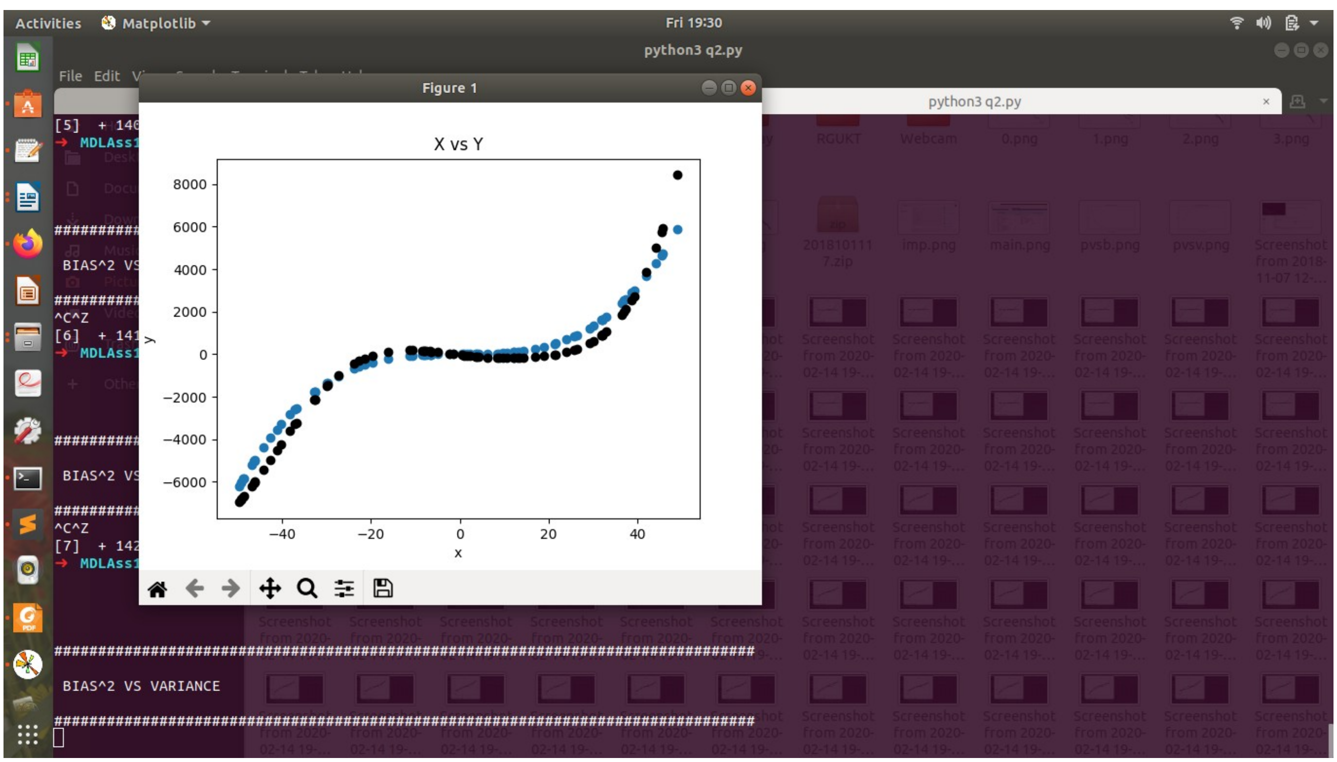
Activities
Matplotlib
Fri 19:28
python3 q2.py
Figure 1
X vs Y
BIAS^2 VS VARIANCE


Activities
Matplotlib
Fri 19:29
python3 q2.py
Figure 1
X vs Y
BIAS^2 VS VARIANCE

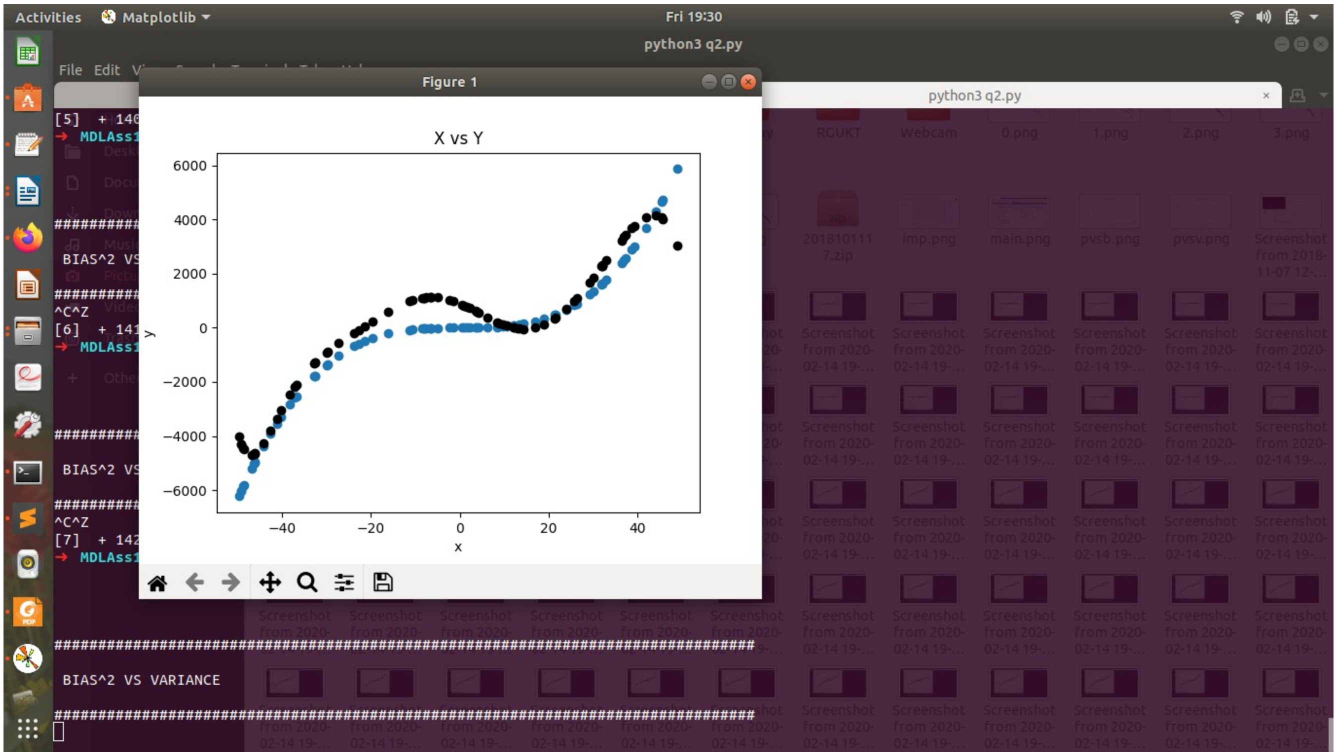

Activities
Matplotlib
Fri 19:29
python3 q2.py
Figure 1
X vs Y
BIAS^2 VS VARIANCE

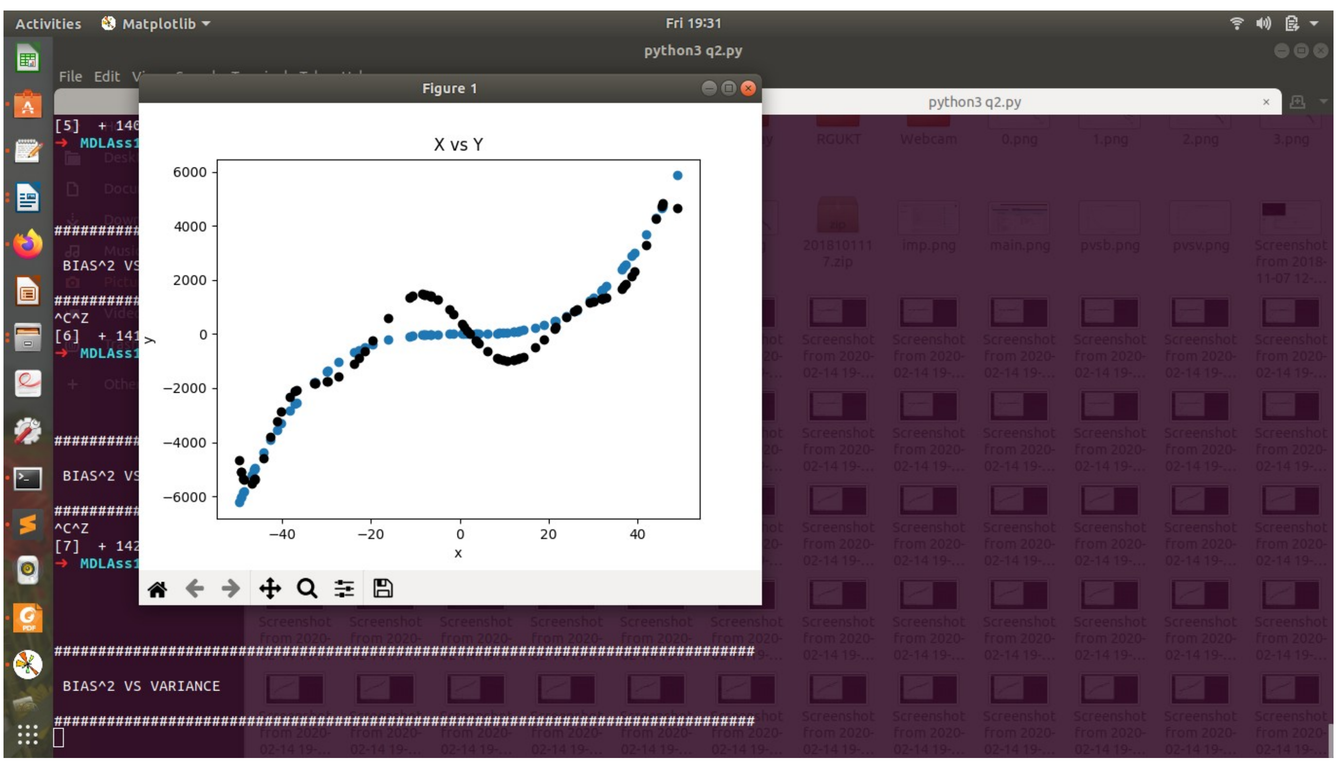


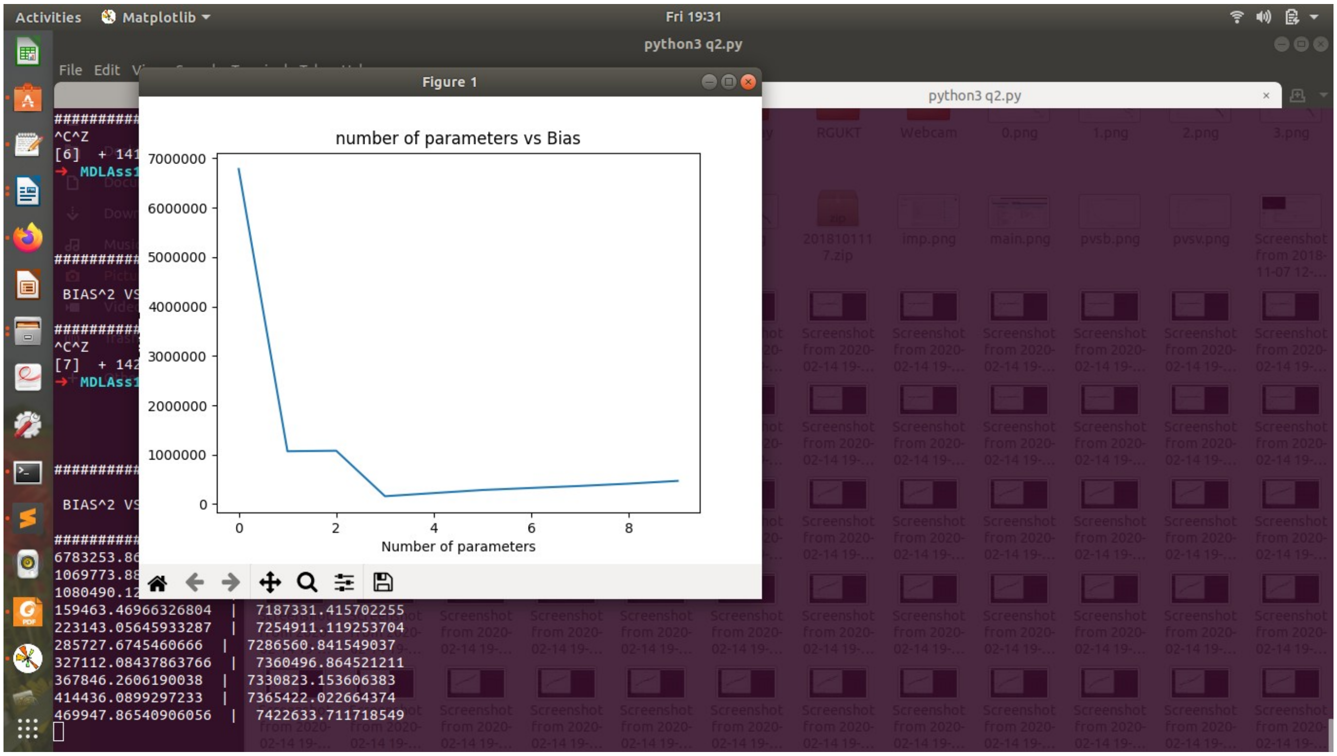
Activities
Matplotlib
Fri 19:29
python3 q2.py
Figure 1
X vs Y
8000
6000
4000
2000
0
−2000
−4000
−6000
−8000
−40
−20
0
20
40
x
y
BIAS^2 VS VARIANCE



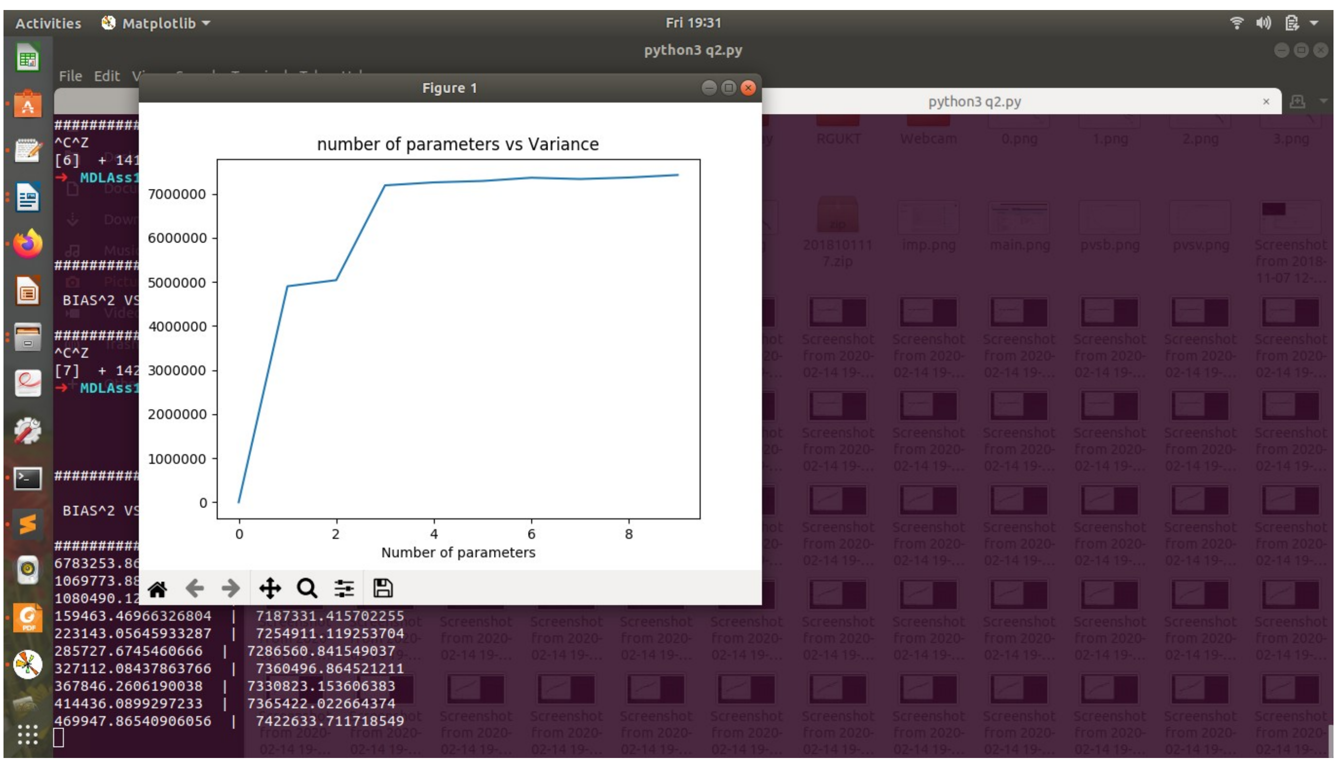
Activities
Matplotlib
Fri 19:30
python3 q2.py
Figure 1
X vs Y
8000
6000
4000
2000
0
−2000
−4000
−6000
−40
−20
0
20
40
x
y
BIAS^2 VS VARIANCE


Activities
Matplotlib
Fri 19:30
python3 q2.py
Figure 1
X vs Y
6000
4000
2000
0
−2000
−4000
−6000
−40
−20
0
20
40
x
y
BIAS^2 VS VARIANCE


Activities
Matplotlib
Fri 19:31
python3 q2.py
Figure 1
X vs Y
6000
4000
2000
0
−2000
−4000
−6000
−40
−20
0
20
40
x
y
BIAS^2 VS VARIANCE



__**Number of parameters versus Bias**__



__Number of parameters versus Variance__

<u>outputs:</u>

##
####

BIAS^2 VS VARIANCE

##
####
6783253.864969173 | 0.0
1069773.886017674 | 4898787.1462623775
1080490.1293431984 | 5037544.555105042
159463.46966326804 | 7187331.415702255
223143.05645933287 | 7254911.119253704
285727.6745460666 | 7286560.841549037
327112.08437863766 | 7360496.864521211
367846.2606190038 | 7330823.153606383
414436.0899297233 | 7365422.022664374
469947.86540906056 | 7422633.711718549