Tuan Tran
A20357888
CS584
Summer 2020

A. Implementation details:

The program is divided into 4 total relevant files. There are three notebooks, each notebook represents one section in the implementation assignment. The content of each notebook has been organized such that it follows each of the instructions (part a, b, c, …) as stated in the prompt. Each cell in each notebook also has been fully documented and should run fine on any machine with the appropriate python modules. To use the program and check for correctness of the results stated, simply go into each notebook and run each of the cells. However, for notebook “Part 3” specifically, please download the data following the link from the data folder, and place the data file as it is in the same directory as “Part 3” notebook.

The model.py file contains the main model implementations. Inside are the function to calculate the Mean Squared Error given two result vectors, as well as three different models: Linear Regression with Gradient Descent, Linear Regression with Explicit, closed form result and Dual Linear Regression. Each of the model has 2 main functions which are fit() and predict(). For all 3, the 0th feature (with value 1) is included into the input feature matrix.

Throughout implementing the three models, I learned two lessons. Firstly, it’s much easier to implement the model in matrix form. Thus, for increased efficiency in implementation, we should work out the matrix form of the problem and the solution, before going into implementing. Second lesson is numpy specific. In order to ensure correctness of training, we **should not** reshape any vector from numpy shape (shape,) into (shape,1). This will mess up the computation and thus the result

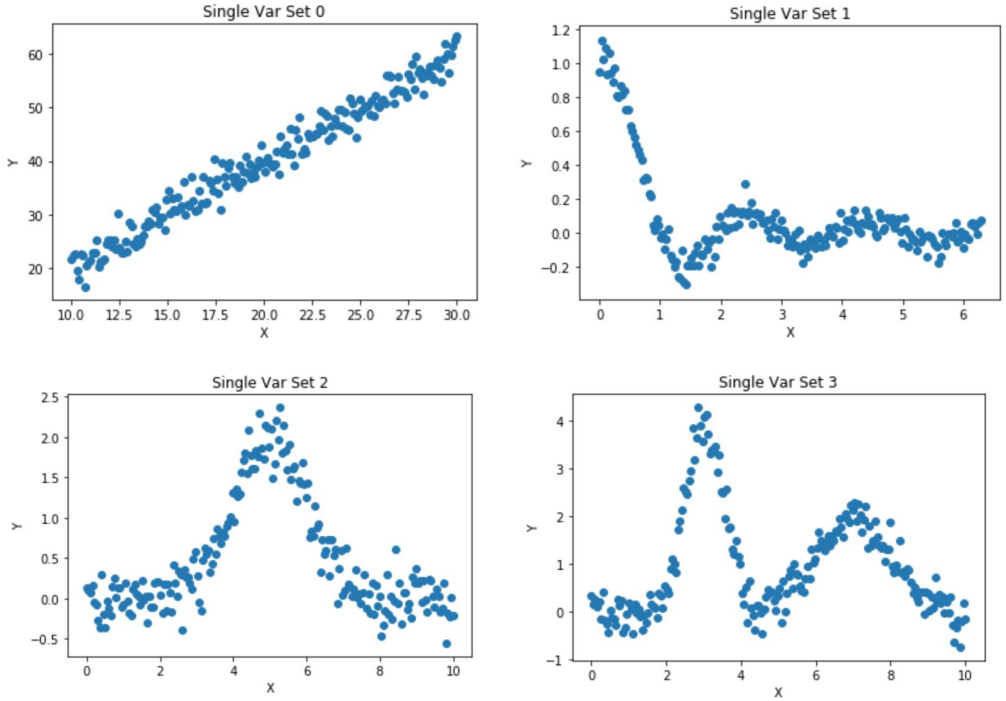
B. Result and Discussion:

1. Correctness of Implemented Algorithms:
   Using the datasets provided in part 1, the custom implemented Linear Regression was trained in order to compare with its counterpart from the module sklearn. Using the same exact train/test split, we obtain the following result:

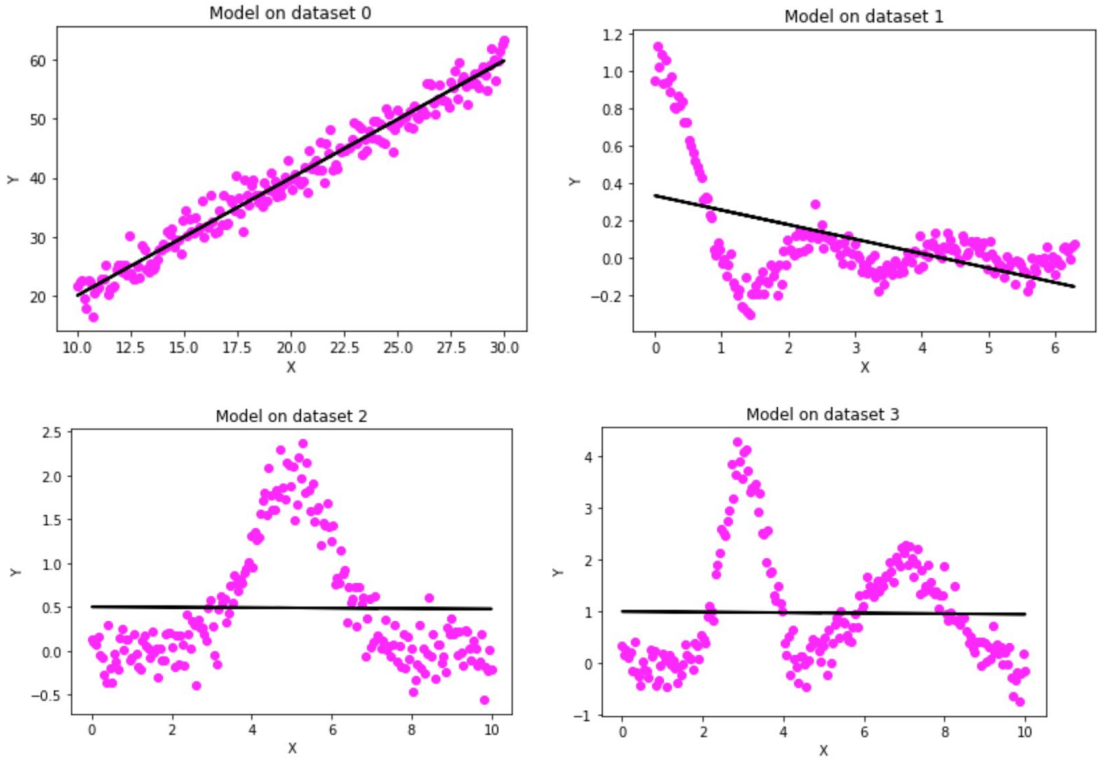
```
========================================
Dataset 0:
        My model train error: 4.392593732873428
        My model test error: 3.6294593219914404
        Sklearn model train error: 4.392593732873428
        Sklearn model test error: 3.6294593219912503
========================================
Dataset 1:
        My model train error: 0.05067358151251983
        My model test error: 0.09949817947230524
        Sklearn model train error: 0.05067358151251982
        Sklearn model test error: 0.0994981794723053
========================================
Dataset 2:
        My model train error: 0.48544381649386037
        My model test error: 0.5610902070544385
        Sklearn model train error: 0.4854438164938604
        Sklearn model test error: 0.5610902070544379
========================================
Dataset 3:
        My model train error: 1.0400517206280033
        My model test error: 1.935301596403096
        Sklearn model train error: 1.0400517206280033
        Sklearn model test error: 1.9353015964031002
```

The implementation result and sklearn matches exactly, as both sklearn and our implementation uses the closed form formula to find the solution. The correctness of the iterative version (Gradient Descent) will be evaluated in Part 2, and will be shown to also be correct.

2. Part 1:
   In part 1, we are given 4 different single-variable datasets. After plotting, we can observe that these datasets are very different, both in terms of shape as well as scale.

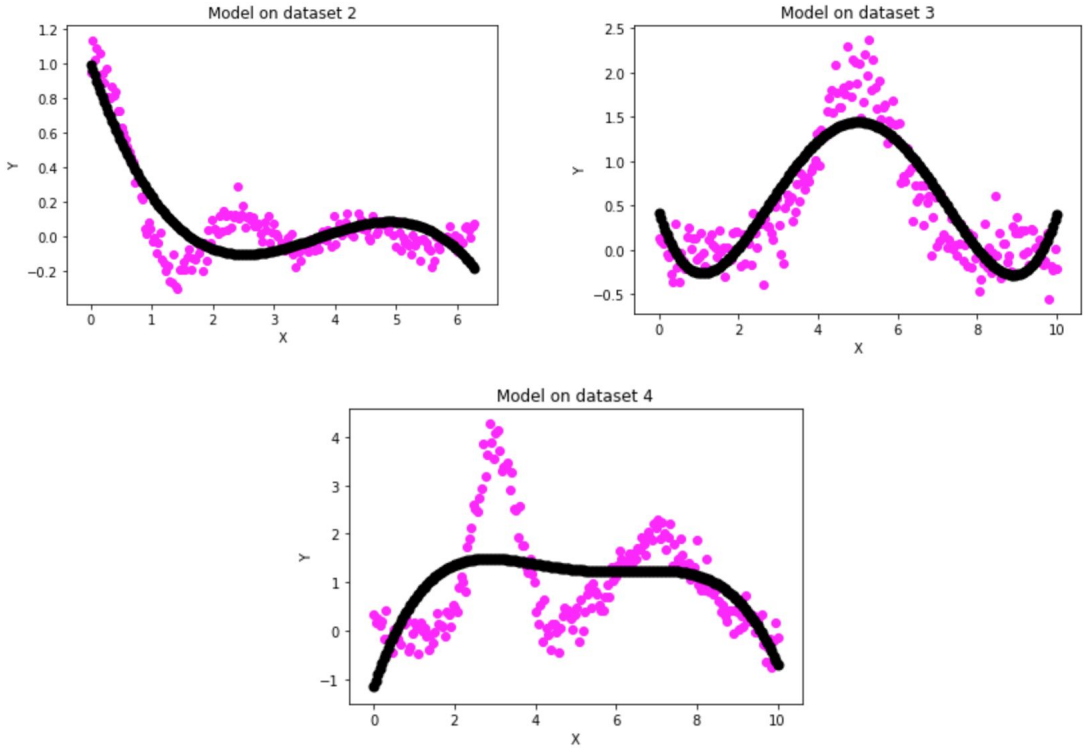


It was clear from the plot that Linear Regression would be a very suitable model for the first dataset, but then we would need something more complex in order to fit the data from the second dataset onwards. As expected, after splitting each of the data into training, testing set and fitting a Linear Regression model, the model performs the best on the first dataset with respect to the scale of the target variable. More concretely, we can plot the trained model on top of the data points.



We can see that the model fitted the first and second dataset pretty well, however it did bad for the latter two datasets. Since Linear Regression is unsuitable for the more complex problems, we turn to more complex models by evaluating the Polynomial Regression model with degree 2, 3, 4 and 5. Since there are relatively few data in each dataset, we will use 5-fold cross validation for each of the Polynomial Regression models, to determine which one will be the most suitable.

After fitting on the latter 3 datasets, we can observe that degree 5 will always give the best training error due to its complexity. However, we opt to choose degree 3 for dataset 2 and degree 4 for the last 2 datasets, based on the testing error results and also because a higher degree may reduce generalizability. We can indeed see a much better fit than simple Linear Regression.



A simple experiment was performed where we dropped the training size to only 20% of total data. In doing this, we observe that for Linear Regression, the change increased the testing error for the first dataset, but not the latter 3; however the change is indeed more problematic for Polynomial Regression since the number of features are now closer to the number of data available

3. Part 2:

The datasets in part 2 are more complex since each dataset has a lot more data **and** more features than those in part 1. We perform 5-fold cross validation on the training set, across 4 different polynomial mappings (degree 2, 3, 4 and 5). Similar to results in part 1, we observe that higher degree leads to lower training error, but not necessarily testing error. In the end, we choose polynomial mapping of degree 2 for dataset 1, 3 and 4 since it consistently gave the best average **testing** MSE result. For dataset 2, we chose degree 5 since it gave a major boost in average **testing** MSE.

Additionally, to check for correctness of our implementation of the iterative version Linear Regression, we benchmarked our implementation of Gradient Descent Linear Regression with the Explicit Linear Regression on each of the dataset using 5-fold cross validation (and mapping of degree 2). We use a learning rate of 1e-4 for first 2 datasets, and 1e-6 for latter 2

```
Dataset 1:
        Iterative:
                Average train MSE: 0.2903976393701412
                Average test MSE: 0.292034172422746
        Explicit:
                Average train MSE: 0.25893288314771007
                Average test MSE: 0.25893288314771007
Dataset 2:
        Iterative:
                Average train MSE: 0.01991017797793271
                Average test MSE: 0.019979081072201088
        Explicit:
                Average train MSE: 0.019943633719617975
                Average test MSE: 0.019943633719617975
Dataset 3:
        Iterative:
                Average train MSE: 0.9897733390127728
                Average test MSE: 0.9904899848411304
        Explicit:
                Average train MSE: 0.25078420598776296
                Average test MSE: 0.25078420598776296
Dataset 4:
        Iterative:
                Average train MSE: 0.004144630842162844
                Average test MSE: 0.004145749526527948
        Explicit:
                Average train MSE: 0.0038875023603957394
                Average test MSE: 0.0038875023603957394
```

We can observe that the result was pretty close. However, one thing to note is that Gradient Descent Linear Regression with polynomial mapping is very sensitive to learning rate. Changing the learning rate by a small amount can lead to bad results.

4. Part 3:

In this section, we chose the Airfoil Self-Noise dataset (link in data folder) which aims to predict the scaled sound pressure level. The data consists of 1503 instances with 5 total attributes. Since the frequency feature range is pretty big compared to the rest of the features, we first normalize the data. We then split the dataset into train and **held-out** test set. After splitting there are 1202 instances left in training set. Acknowledging that we have relatively few data, we then perform 5-fold cross validation using the training set **only**, to evaluate 4 different models: simple Linear Regression, and Polynomial Regression with degree 2, 4 and 8.

```
Linear Regression Model:
        Average training MSE: 36.570072184033386
        Average testing MSE: 37.09543505449684
Polynomial Model with Degree 2:
        Average training MSE: 27.41044943694407
        Average testing MSE: 28.67781185993119
Polynomial Model with Degree 4:
        Average training MSE: 24.529072930700657
        Average testing MSE: 28.144484707152678
Polynomial Model with Degree 8:
        Average training MSE: 22.519528963256285
        Average testing MSE: 32.710681513385396
```

We can observe that, for this dataset, it seems like degree 4 is the most suitable since degree 1 (Linear Regression) is too simple for such a task and degree 8 may be overkill. Indeed, degree 4 yields the best result also on the **held-out** test set, with MSE of 30.8