Programming homework 1

Task 1.

Subtask 1.

In this task I just printed the versions of my Python as well as pandas and numpy versions (3.11.0 , 1.26.0, 2.1.1)

Subtask 2.

In this task I constructed a class 'DAO' which accepts 4 arguments - name of the input document, input document type, name of the output document and the type of output document. I also implemented a function 'load' which reads the input file depending on the type, but I choose to store the data in numpy arrays because I already have some experiance with handling them. If the given document type is not 'txt' or 'csv' I raise a 'ValueError'. In store function, I implement writing the loaded data into the given output file. I first check if the loaded data is just a string so I convert it into numpy array so I can standardize the writing into files. For both of this functions I implemented exception handling, in case some errors occur.

Subtask 3.

In subtask 3 I implement an abstract class called 'SubTaskABC' with only one method which I override in the subclasses of this one. The 'SubTask13' class takes a DAO object as a argument, and also has 2 more functions. The first one is process, which I override, and in that function I change the string from the input document in a way which was needed, using the second method of this class which reverts the string. After reverting it, I store it into the output file. I also implement the reverting of list of strings, in case the input file contained one.

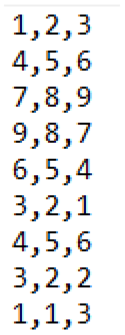
Subtask 4.

Subtask 4 requested that we implement 3 unit tests for the first two classes, so I did so. For the first class I implemented a test which checks the validity of store function. The second test does the same for store function, while the third checks if an exception rises if the input document type is invalid. These three tests seemed most logical to me since this class is not very complex. For the second class, I tested the process function, by checking if the reversed string was valid. I the same thing for the list of strings, and at the end I checked if the exception rises when the input is invalid. I implemented the unit tests using library called 'unittest' and ran them with the function 'TextTestRunner'.

Task 2.

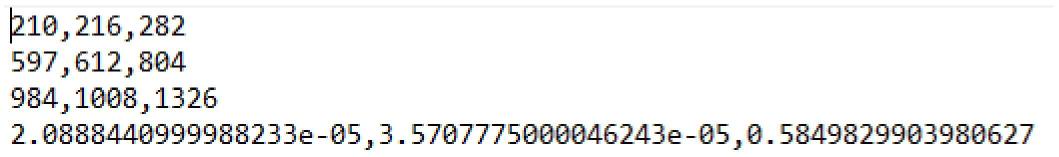
Subtask 1.

In this subtask I implemented the class 'Subtask21' which takes DAO object as an argument and consists of 6 functions. Function 'process' is the main one and it reads the input file which consists of n-number 3x3 matrices and multiplies them all. After that, it calculates the speedup of a numpy matrix multiplication over my matrix multiplication. I used 'time' library for calculating the speedup. The input of 'input21.csv' is the following:



```
1,2,3
4,5,6
7,8,9
9,8,7
6,5,4
3,2,1
4,5,6
3,2,2
1,1,3
```

And the output of the 'output21.csv' file is the following:

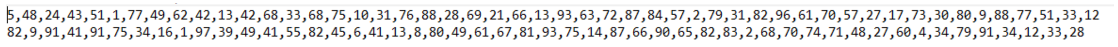


```
210,216,282
597,612,804
984,1008,1326
2.0888440999988233e-05,3.5707775000046243e-05,0.5849829903980627
```

Where the last row represents the speed of the numpy algorithm over 100000 repetitions, the speed of my implementation over 100000 repetitions and the last element is the speedup. First three rows are for the multiplied 3x3 matrix from the input file. The speedup of the numpy algorithm was 0.6, because I use 3 nested for loops for matrix multiplication which have big time complexity.

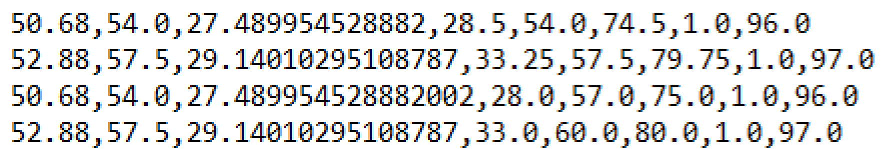
Subtask 2.

In this subtask I did the same thing as in the last with the process method in my new class 'Subtask22', but this time i measured calculating statistic formulas for random 100 numbers over 2 rows in 100000 repetitions. Numpy implementation achieved a speedup of 22 times, with the speed of the numpy algorithm of 0.0008395s, and speed of my calculations in 3.7820567s. The content of 'input22.csv' file was:



```
5,48,24,43,51,1,77,49,62,42,13,42,68,33,68,75,10,31,76,88,28,69,21,66,13,93,63,72,87,84,57,2,79,31,82,96,61,70,57,27,17,73,30,80,9,88,77,51,33,12
82,9,91,41,91,75,34,16,1,97,39,49,41,55,82,45,6,41,13,8,80,49,61,67,81,93,75,14,87,66,90,65,82,83,2,68,70,74,71,48,27,60,4,34,79,91,34,12,33,28
```

While te output of the 'output22.csv' file is:



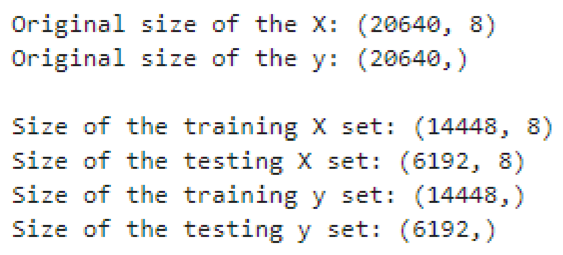
```
50.68,54.0,27.489954528882,28.5,54.0,74.5,1.0,96.0
52.88,57.5,29.14010295108787,33.25,57.5,79.75,1.0,97.0
50.68,54.0,27.489954528882002,28.0,57.0,75.0,1.0,96.0
52.88,57.5,29.14010295108787,33.0,60.0,80.0,1.0,97.0
```

where the first 2 rows are calculations from numpy, and the last 2 rows are my calculations.

Task 3.

Subtask 1.

In the following picture you can see the size of the original dataset as well as sizes of training and test datasets:



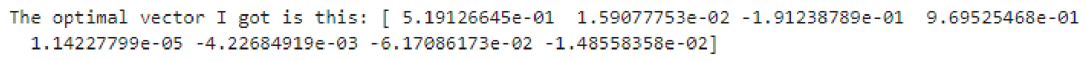
```
Original size of the X: (20640, 8)
Original size of the y: (20640,)

Size of the training X set: (14448, 8)
Size of the testing X set: (6192, 8)
Size of the training y set: (14448,)
Size of the testing y set: (6192,)
```

Subtask 2.

The MSE that I achieved is: 0.5305677824766754.

Subtask 3.

The optimal weight vector I computed was:

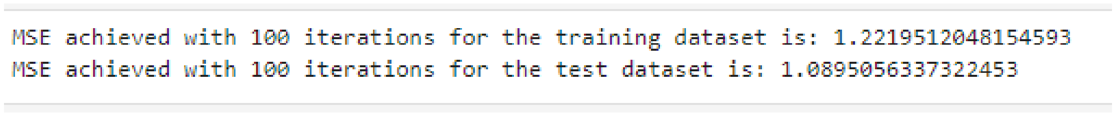


```
The optimal vector I got is this: [ 5.19126645e-01  1.59077753e-02 -1.91238789e-01  9.69525468e-01
  1.14227799e-05 -4.22684919e-03 -6.17086173e-02 -1.48558358e-02]
```

And the MSE that I achieved with closed form solution is: 0.6054030599879396.

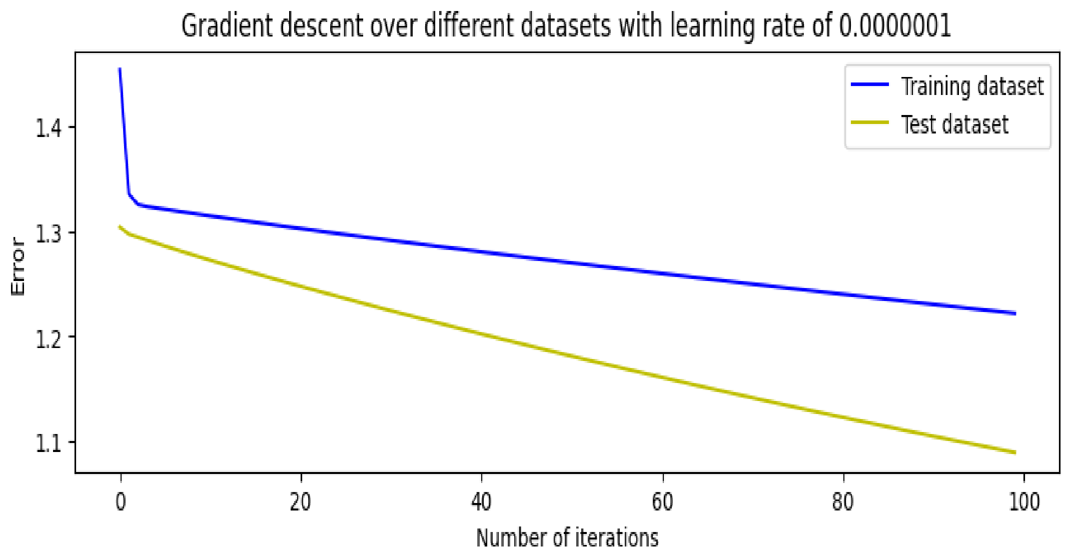
Subtask 4.

For the whole dataset, only training data and only test data, the MSE's I achieved after 100000 iterations are :



```
MSE achieved with 100 iterations for the training dataset is: 1.2219512048154593
MSE achieved with 100 iterations for the test dataset is: 1.0895056337322453
```

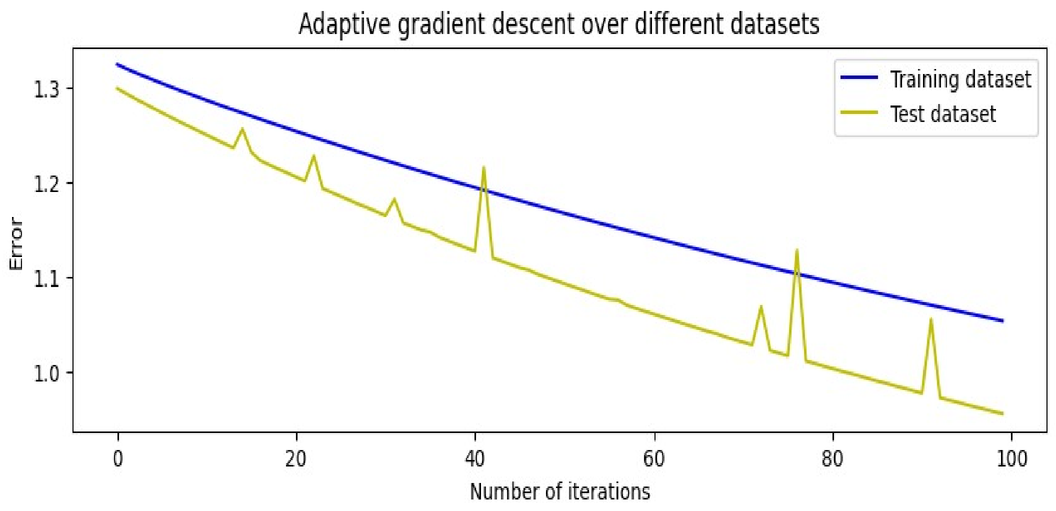
Which are plotted like this:



And from this plot we can see that the training set has a big decrease in the first few iterations up until they reach MSE of around 1.3 where its slows its decreasing drastically, on the other hand, test dataset decreases rather equally over the span of 100k iterations. The MSE's I achieved were a bit higher than the closed form one. The learning rate I used is 0.0000001.

Subtask 5.

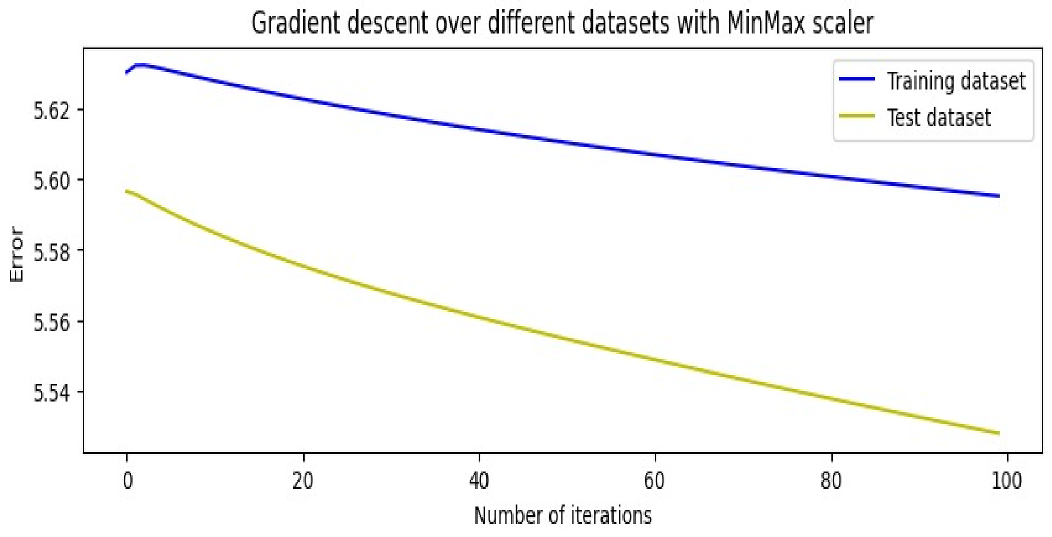
For the adaptive rule I choose bold driver as explained in the lectures (when the MSE of the current point is larger than the last one, I multiply it by 0.6, and if its not I multiply it by 1.1), and the results I got are better than in the last subtask as you can see from the plot:



The starting MSE is lower, and the decreasing of both datasets is constant and continuous.

Subtask 6.

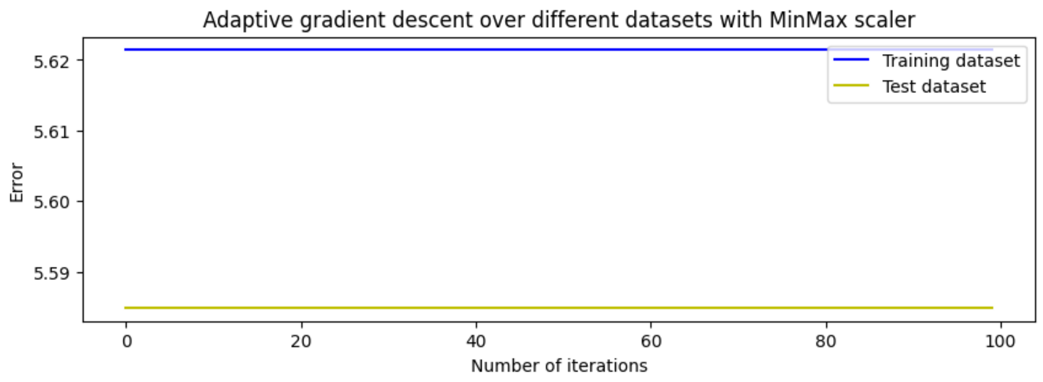
First I scaled the data and fitted it into the gradient descent function and I got a plot like this:



Which seems to be decreasing (faster for test dataset, slower for training dataset) but at a very slow pace and a high start MSE in comparison to other methods.

After that I fitted the scaled data into my adaptive learning rule algorithm:

```
<matplotlib.legend.Legend at 0x2b26c3b7550>
```



From which we can see I got the same constant MSE which is not changing for either of datasets.

MinMax scaler scaled the data by converting it to a number between 0 and 1 and it does so by dividing the difference between the point value and the maximum value, and difference between maximum and minimum value.

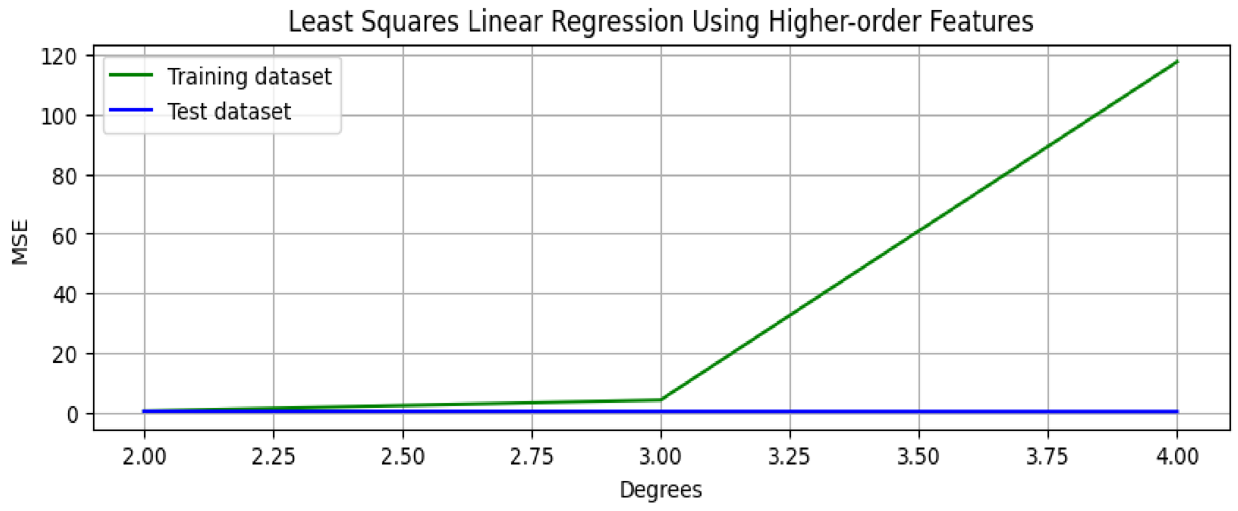
Subtask 7.

At the beggining I scale both X_train and X_test datasets and fit them into the model, after which I predict the values for the y dataset. After that I compare the MSE's for different degrees and I get the following results:

```
Degree 2 Polynomial - Mean Squared Error for this degree: 0.6745163730301048
Degree 2 Polynomial - Mean Squared Error for this degree: 0.6508463314463686
Degree 3 Polynomial - Mean Squared Error for this degree: 4.394988921779183
Degree 3 Polynomial - Mean Squared Error for this degree: 0.5860754537391314
Degree 4 Polynomial - Mean Squared Error for this degree: 117.52628282568787
Degree 4 Polynomial - Mean Squared Error for this degree: 0.5307584954511115
```

and the following plot:

```
<matplotlib.legend.Legend at 0x26da1a8edd0>
```



From which we can say that the model is overfitted and preforms really well for one dataset and not for the other.