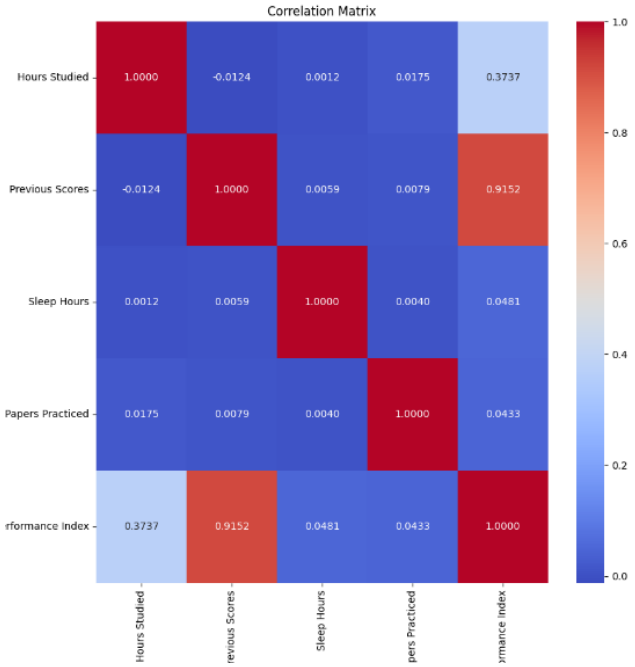
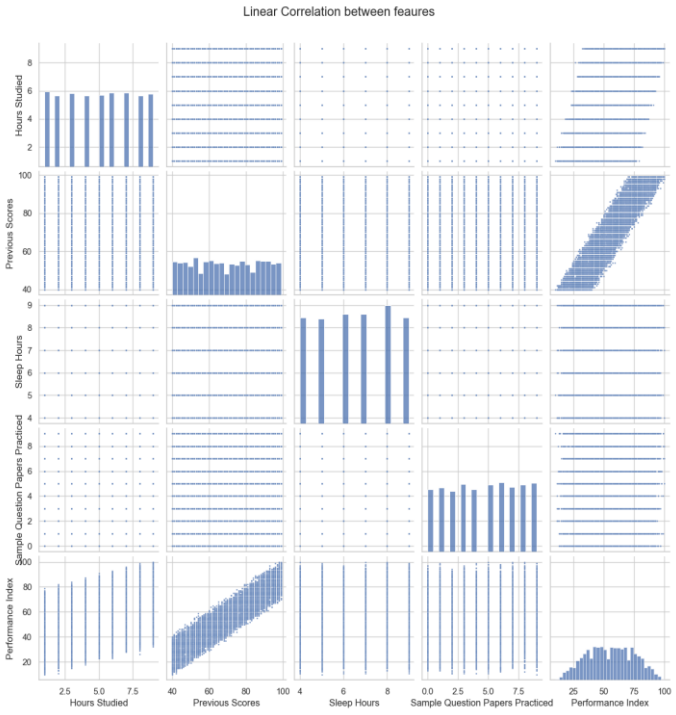
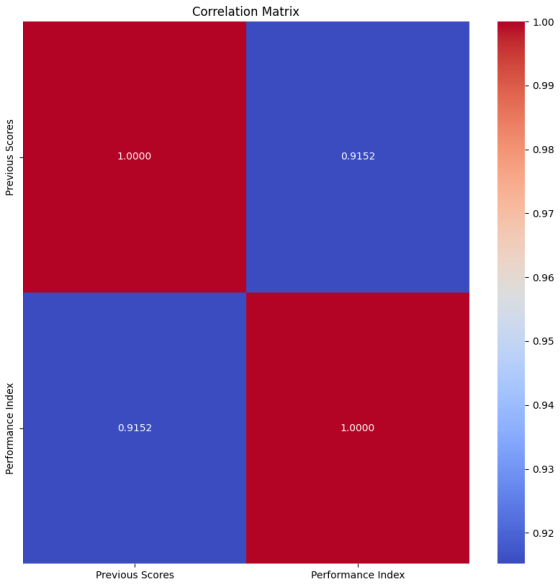
Preprocessing steps:

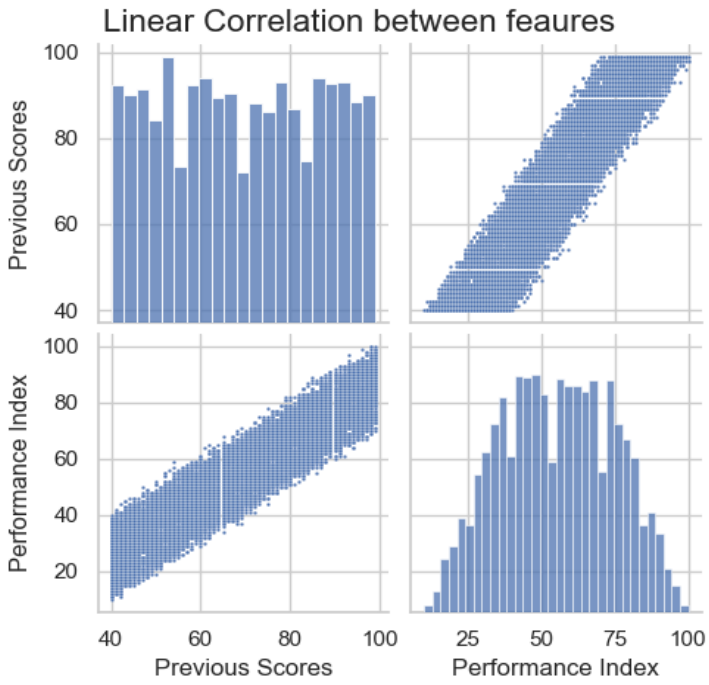
i. Identify my dependent “y” and “x” independent features and find any **linear correlation**.(in this case I chose Performance index(***Student scores***) as my dependent “y” variable, and Previous scores***(Student previous scores)***as my “x” independent variable.





ii. After separating them from the original dataset , my new dataset is this one***(Note that I did not need any further preprocessing like normalization , since these 2 features already have a range from 0 to 100 , since both represent students’ scores)***





iii. Because the original dataset has an equal of 1000 samples , I've decided to reduce the dataset to only 500 samples ***(500 samples should be enough for this),*** after that the data should be prepared as input for the Linear regression model.

Before

```
"x" train dataset type and shape : int64 , (10000, 1)  len 10000
cols = ['Previous Scores']:

[[99]
 [82]
 [51]
 ...
 [83]
 [97]
 [74]]


"y" train dataset type and shape : float64 , (10000,)  len 10000
cols = Performance Index:

[91. 65. 45. ... 74. 95. 64.]
```

After

```
"x" train dataset type and shape : int64 , (500, 1)  len 500
cols = ['Previous Scores']:

[[99]
 [82]
```

```
"y" train dataset type and shape : float64 , (500,)  len 500
cols = Performance Index:

[ 91.  65.  45.  36.  66.  61.  63.  42.  61.  69.  84.  73.  27.  33.
  68.  43.  67.  70.  30.  63.  71.  85.  73.  57.  35.  49.  66.  83.
  74.  74.  39.  36.  58.  47.  60.  74.  42.  68.  32.  64.  45.  39.
```

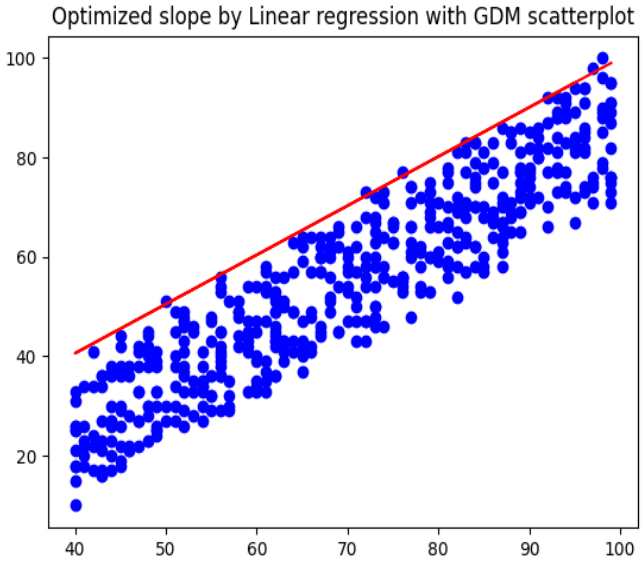
iv. For the first section "a", I was asked to use a steepest descent method , I could use a simple "***Gradient descent"*** optimizer , however because I wanted to learn and try more, I used for

this section a "***Gradient descent with momentum",*** then for section 2, I should use a steepest descent method with momentum, so I used **"ADAM"**

v. **Hyperparameters:**

- Learning rate = 0.001
- Momentum = 0.9
- Tolerance = 1e-6
- Epochs = 1000

vi. **Results :**I plotted a **scatter plot** to visualize the data and the best line optimized by "(GDM)***Gradient descent with momentum***" ,then I **create a table** with the ***actual, predicted and error values ,*** to analyze the results , we can see in the scatter plot that the line in fact does not fit well with the data points , it means that the GDM optimizer ***could not find the best optimal*** for **weights and bias** coefficients, ***(the problem is not in the hyperparameters since I set the same hyperparameters with ADAM and the results were much better).***



Actual	Predicted	Error
91	99	8
65	82.17	17.17
45	51.48	6.48
36	52.47	16.47
66	75.24	9.24
61	78.21	17.21
63	73.26	10.26
42	45.54	3.54
61	77.22	16.22
69	89.1	20.1
84	91.08	7.08
73	79.2	6.2
27	47.52	20.52
33	47.52	14.52
68	79.2	11.2
43	72.27	29.27
67	73.26	6.26
70	83.16	13.16
30	54.45	24.45
63	75.24	12.24
71	99	28
85	96.03	11.03
73	74.25	1.25
57	85.14	28.14
35	61.38	26.38
49	62.37	13.37
66	79.2	13.2
83	84.15	1.15
74	94.05	20.05

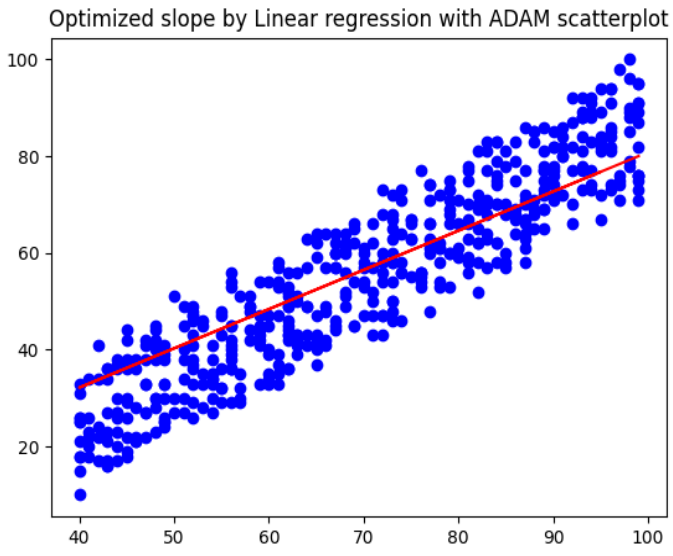
b) Use Fletcher-Reeves method, or the method combining gradient descent with momentum term.

I. For this section since I can use a gradient descent with momentum , I decided to use "ADAM" (***Adaptative Moment Estimation)*** optimizer ,which is an iterative optimization algorithm , it is a result as a combination of **RMSprop + Stochastic Gradient Descent with Momentum .**

II. **Hyperparameters:**

- Learning rate = 0.001
- Momentum = 0.9
- Tolerance = 1e-6
- Epochs = 1000

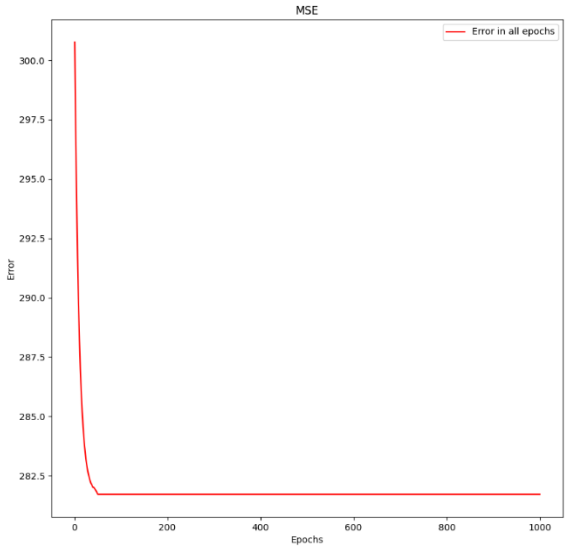
III. Results : We can see that the results using "ADAM", are much better even though when I set the same hyperparameter as with "(GDM)***Gradient descent with momentum***", however ADAM optimize better the **weights and bias** coefficients, and obtains lower error in most of the predictions.



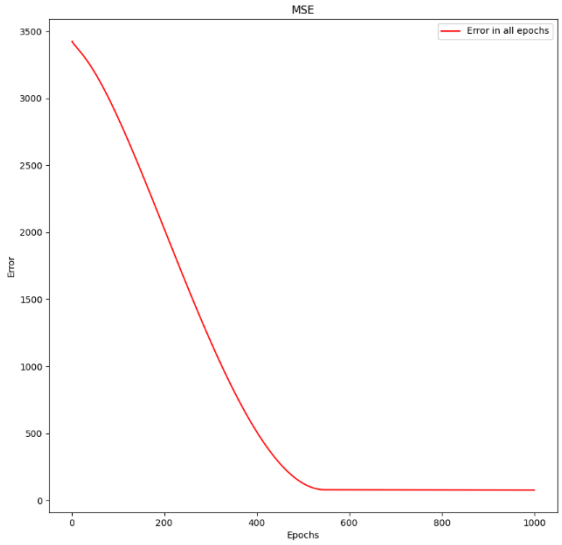
Actual	Predicted	Error
91	80.01	10.99
65	66.22	1.22
45	41.08	3.92
36	41.89	5.89
66	60.54	5.46
61	62.98	1.98
63	58.92	4.08
42	36.21	5.79
61	62.17	1.17
69	71.9	2.9
84	73.52	10.48
73	63.79	9.21
27	37.84	10.84
33	37.84	4.84
68	63.79	4.21
43	58.11	15.11
67	58.92	8.08
70	67.03	2.97
30	43.51	13.51
63	60.54	2.46
71	80.01	9.01
85	77.57	7.43
73	59.73	13.27
57	68.65	11.65
35	49.19	14.19
49	50	1
66	63.79	2.21
83	67.84	15.16
74	75.95	1.95

IV. Comparison:

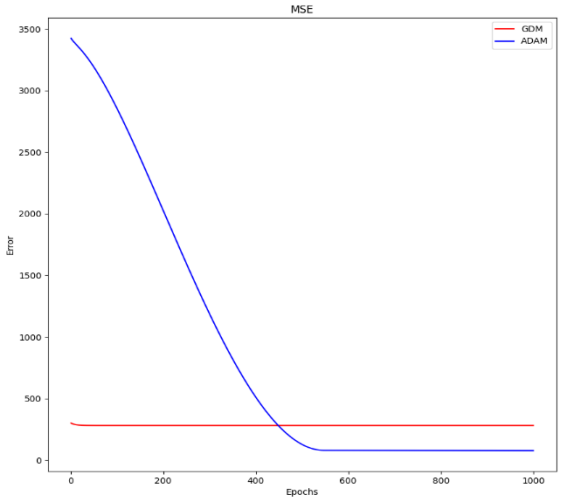
Gradient Descent with Momentum(GDM)



Adaptative Moment Estimation (ADAM)



GDM vs ADAM



We can notice that how GDM started with a low error , however it gets trapped in a local minimum , while ADAM in the beginning started with a high error but its gradients could continue toward an apparently global minimum ***(Note that this does mean that I did not say that ADAM is better than GDM ,but for this specific case ADAM is the winner).***

2. You are given the Titanic survival data. The **X** data contain the passengers' information ***(gender, age, …),*** while the **Y** data indicate whether the passengers survived (1) or not (0). The data were divided into training and testing data sets. Please use any method (including modules of Python or functions of MATLAB, or any online resource you can find) to train the logistic regression coefficients, and then test your prediction accuracy.

a. First at all revised the datasets in order to identify problems or see if they need to be preprocessed before the training and test , what I could identify in the test_X datasets , there were some Nulls values , so I deleted those rows with nulls values, but before I merged test_X and test_y datasets in order to match the rows that were going to be eliminated , otherwise it would be a discrepancy in the number of rows in both datasets.

```
DataFrame shape before cleaning: (418, 8)
Total null elements found: 87
Total rows removed: 87

cleaned_independent shape:
 (331, 7)

cleaned_dependent shape:
 (331, 1)
```

b. I decided to set the column "Id" as index so I created a function to do it , it could be deleted as well since there was an index already .

```
#Set column Id as index
X_train_df = self.set_col_index(X_train_df,'Id',verbose=False)
y_train_df = self.set_col_index(y_train_df,'Id',verbose=False)
X_test_df = self.set_col_index(X_test_df,'Id',verbose=False)
y_test_df = self.set_col_index(y_test_df,'Id',verbose=False)
```

c. Then in the **train_X and test_X datasets** I observed that the features **'Age'** and **'Fare'** , could be Normalized in order to get better results and convergence during the training.

Before Normalization

	Id	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	0	34.5	0	0	7.8292	2
1	1	3	1	47.0	1	0	7.0000	1
2	2	2	0	62.0	0	0	9.6875	2
3	3	3	0	27.0	0	0	8.6625	1
4	4	3	1	22.0	1	1	12.2875	1
5	5	3	0	14.0	0	0	9.2250	1
6	6	3	1	30.0	0	0	7.6292	2
7	7	2	0	26.0	1	1	29.0000	1
8	8	3	1	18.0	0	0	7.2292	0
9	9	3	0	21.0	2	0	24.1500	1

After Normalization (scaled from 0 to 1)

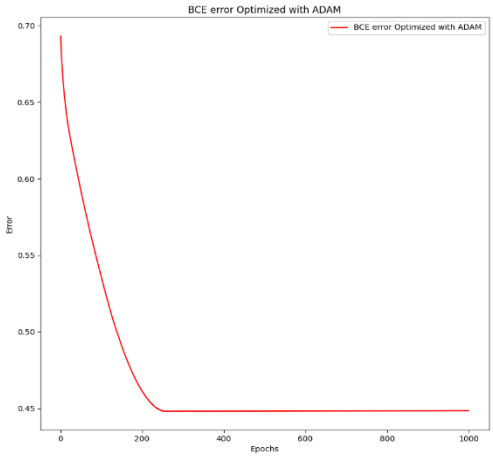
	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
count	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000
mean	2.308642	0.352413	0.365809	0.523008	0.381594	0.062858	0.895623
std	0.836071	0.477990	0.167014	1.102743	0.806057	0.096995	0.516354
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	2.000000	0.000000	0.258608	0.000000	0.000000	0.015440	1.000000
50%	3.000000	0.000000	0.346569	0.000000	0.000000	0.028213	1.000000
75%	3.000000	1.000000	0.447097	1.000000	0.000000	0.060508	1.000000
max	3.000000	1.000000	1.000000	8.000000	6.000000	1.000000	2.000000

d. After the preprocessing I started with the train , since this is just a binary classification problem , then the most logic choice as a loss function is **BCE*(Binary cross entropy loss function)*** , and to optimize the coefficients for this function I used the code of the optimizer ADAM that I wrote in the Linear Regression model *(To understand more please revise the code).*

1) Training Hyperparameters:

- Learning rate = 0.001
- Tolerance = 1e-6
- Epochs = 1000

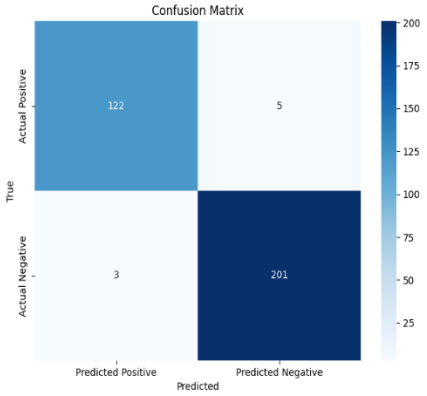
2) Results: As can be observed in the results the error of BCE was minimized below 0.45 , which is a good result for error in binary classifications models.

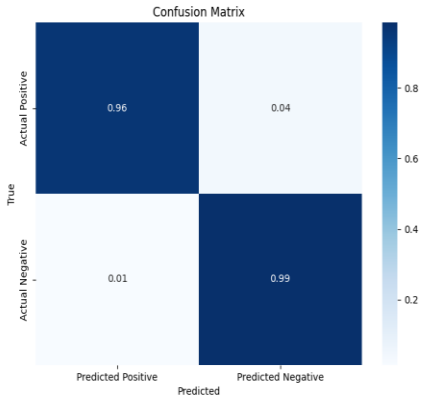



```
Epoch 1000
BCE: 0.4486
```

3) Confusion Matrix :A confusion matrix is a good way to visualize how well is classifying the model. As we can see in both confusion matrix our model is classifying correctly in more of the cases.

Confusion Matrix ***Normalized Confusion matrix***





4) **Predictions results :**in order to visualize better the predicted results compared to the actual values I created a dataset, and I also printed the results in the terminal and the Accuracy , as it can be observed the model obtained a 98% of accuracy***(you can find this dataset in the statistics file).***

Actual	Predicted	Classification
0	0	TN
1	0	FN
0	0	TN
0	0	TN
1	1	TP
0	0	TN
1	1	TP
0	0	TN
1	1	TP
0	0	TN
0	0	TN
1	1	TP
0	0	TN
1	1	TP
1	1	TP
0	0	TN
0	0	TN
1	1	TP
1	1	TP
0	0	TN
0	0	TN
0	0	TN
1	1	TP
0	0	TN
1	1	TP
0	0	TN
0	0	TN
0	0	TN
0	0	TN
1	0	FN
0	0	TN
0	0	TN
1	1	TP
0	0	TN
0	0	TN

```
Predictions: [0 0 0 0 1 0 1 0 1 0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 1
 1 0 0 1 1 0 0 1 1 0 0 0 1 0 0 0 0 1 1 1 0 0 1 1 0 1 0 1 1 1 1 0 1 0 0 0 1 1 0
 1 1 0 0 0 1 0 1 0 1 1 0 0 1 0 0 0 0 0 1 1 1 0 1 0 1 1 1 1 0 1 0 0 0 0 0 0 0 0 1
 0 0 1 0 0 0 0 0 0 1 1 0 0 1 1 0 1 0 1 0 1 0 0 1 0 0 0 0 1 1 0 1 1 0 0 1 1 0
 1 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 0 1 1 0 1 0 1 0 0 0 0 0 1 0
 0 0 0 0 1 1 1 1 0 0 1 0 1 1 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 1 0 0 0 0 1 0 1
 1 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0
 1 0 1 0 0 0 0 0 0 0 0 1 1 0 1 0 1 1 0 0 0 1 0 1 1 0 1 1 0 1 1 1 1 0 0 1 0 0 1
 1 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 1 0 1 1 0 0 1 0 1 0 1 0 0 0 0 0 1 1 1 1 0]
Accuracy: 0.98
```