Course = BIL 470 / HOMEWORK 2

Name = MERT CAN GÖNEN

ID = 181101039

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn import metrics
from LR import LineerRegressionModel
```

Exploratory Data Analysis (EDA)

Read Dataset

In [2]:
```
data = pd.read_csv("data.csv");
```

In [3]:
```
display(data);
```

	Gender	Height	Weight	Index
0	Male	174	96	4
1	Male	189	87	2
2	Female	185	110	4
3	Female	195	104	3
4	Male	149	61	3
...	...	...	...	...
495	Female	150	153	5
496	Female	184	121	4
497	Female	141	136	5
498	Male	150	95	5
499	Male	173	131	5

500 rows × 4 columns

Improve dataset: Remove Gender column which is not affecting the result.

In [4]:
```
data = data.drop(columns="Gender");
display(data);
```

	Height	Weight	Index
0	174	96	4
1	189	87	2
2	185	110	4
3	195	104	3
4	149	61	3
...	...	...	...
495	150	153	5
496	184	121	4
497	141	136	5
498	150	95	5
499	173	131	5

500 rows × 3 columns

Summary of each Features data:

In [5]:

```
h = data["Height"].describe();
w = data["Weight"].describe();
i = data["Index"].describe();
print(h);
print("\n");
print(w);
print("\n");
print(i);
```

```
count    500.000000
mean     169.944000
std       16.375261
min      140.000000
25%      156.000000
50%      170.500000
75%      184.000000
max      199.000000
Name: Height, dtype: float64


count    500.000000
mean     106.000000
std       32.382607
min       50.000000
25%       80.000000
50%      106.000000
75%      136.000000
max      160.000000
Name: Weight, dtype: float64


count    500.000000
mean       3.748000
std        1.355053
min        0.000000
25%        3.000000
50%        4.000000
75%        5.000000
max        5.000000
Name: Index, dtype: float64
```

Duplicit data in dataset:

In [6]:
```
display(data[data.duplicated()]);
```

	Height	Weight	Index
20	157	110	5
162	192	101	3
187	182	84	3
197	177	117	4
260	159	104	5
310	171	147	5
321	181	111	4
327	167	85	4
334	157	56	2
347	162	58	2
354	190	50	0
355	174	90	3
365	141	80	5
381	191	62	1
382	177	117	4
395	164	71	3
398	149	61	3
400	195	104	3
419	177	61	2
421	140	146	5
462	179	56	1
466	188	99	3
482	142	86	5
492	198	50	0

Drop duplicate values:

In [7]:
```
data.drop_duplicates(subset=None, inplace=True);
display(data[data.duplicated()]);
```

Height	Weight	Index

There is no more duplicate values.

In [8]:
```
h = data["Height"].describe();
w = data["Weight"].describe();
i = data["Index"].describe();
print(h);
```

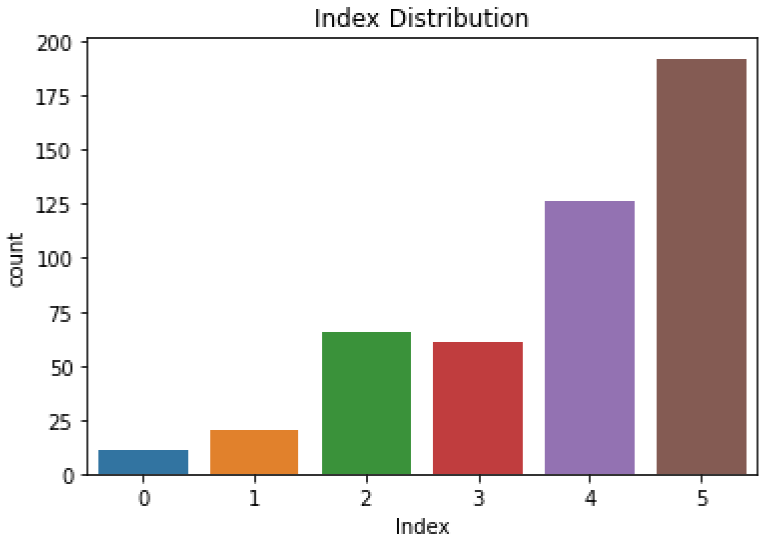
```
print("\n");
print(w);
print("\n");
print(i);
```

```
count    476.000000
mean     169.878151
std       16.332011
min      140.000000
25%      156.000000
50%      170.000000
75%      184.000000
max      199.000000
Name: Height, dtype: float64


count    476.000000
mean     106.920168
std       32.319945
min       50.000000
25%       80.000000
50%      107.000000
75%      137.000000
max      160.000000
Name: Weight, dtype: float64


count    476.000000
mean       3.779412
std        1.337585
min        0.000000
25%        3.000000
50%        4.000000
75%        5.000000
max        5.000000
Name: Index, dtype: float64
```

Index Distribution

In [9]:
```
plt.title("Index Distribution");
sns.countplot(data["Index"]);
```

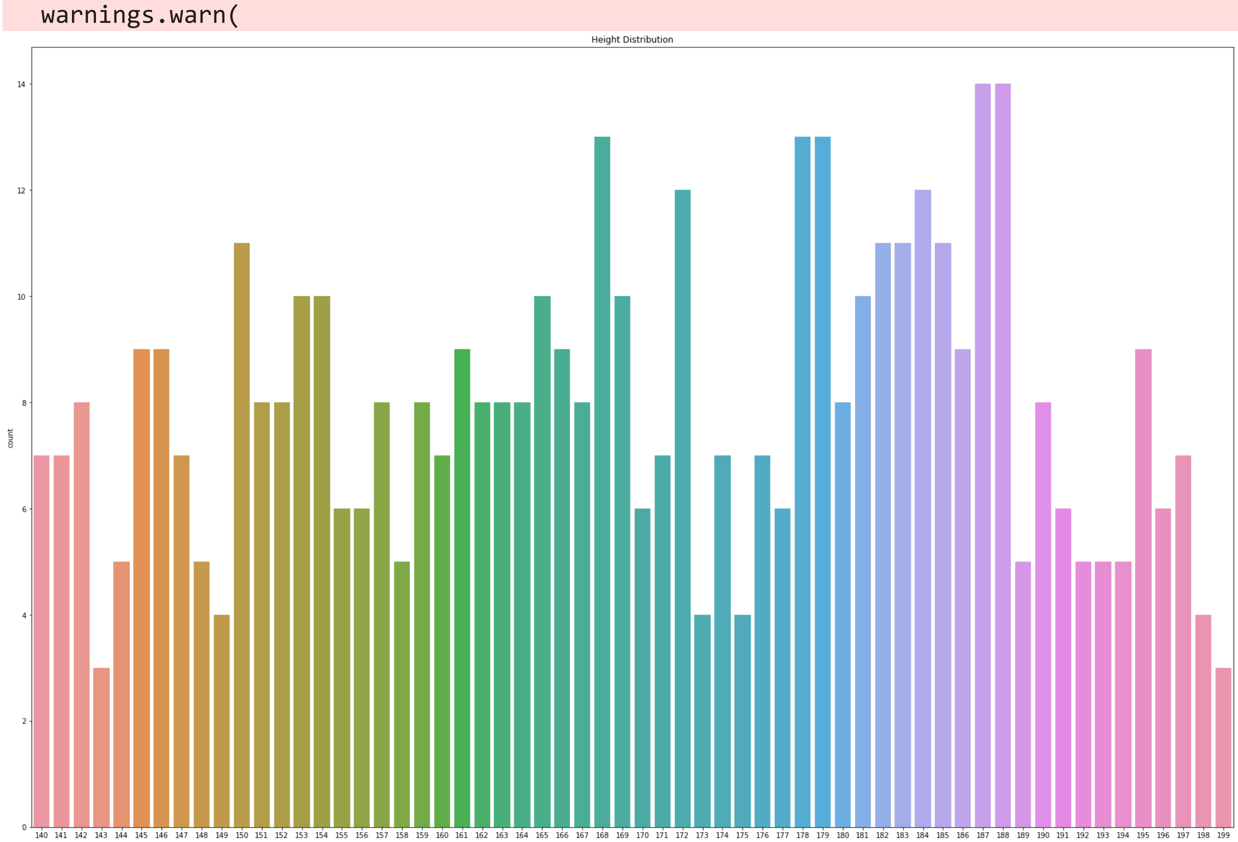
```
C:\Users\mgone\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\_dec
orators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From v
ersion 0.12, the only valid positional argument will be `data`, and passing other arg
uments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



Height Distribution

In [10]:
```
plt.figure(figsize=(30, 20));
plt.title("Height Distribution");
sns.countplot(data["Height"]);
```

```
C:\Users\mgone\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\_dec
orators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From v
ersion 0.12, the only valid positional argument will be `data`, and passing other arg
uments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

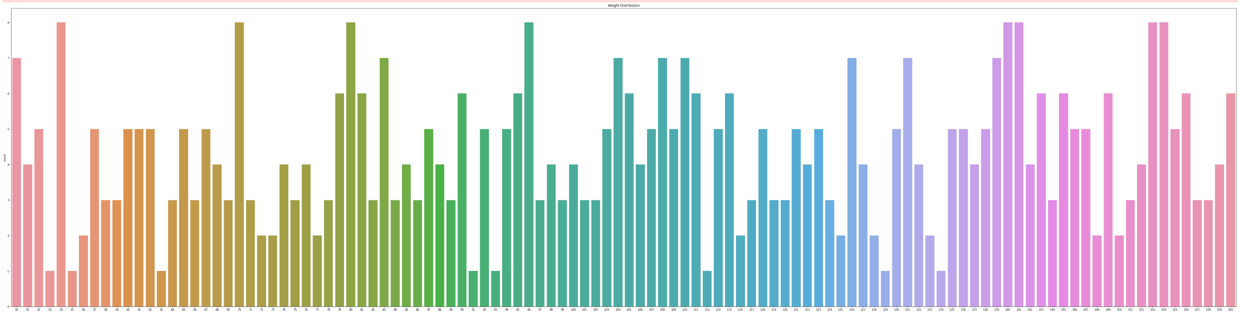


Weight Distribution

In [11]:
```
plt.figure(figsize=(80, 20));
```

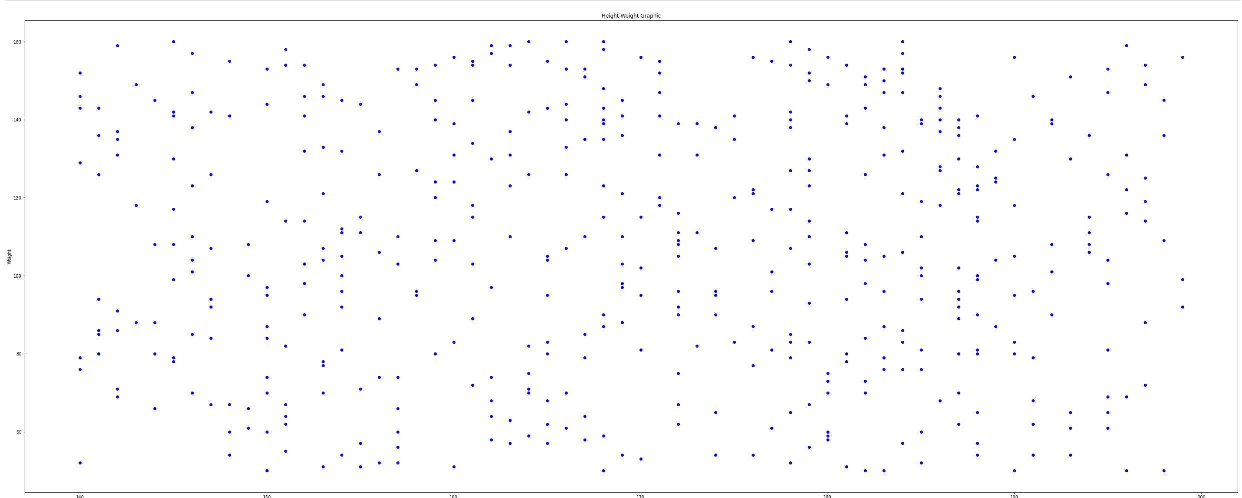
```
plt.title("Weight Distribution");
sns.countplot(data["Weight"]);
```

```
C:\Users\mgone\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\_dec
orators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From v
ersion 0.12, the only valid positional argument will be `data`, and passing other arg
uments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



Data on the 2D data space.

In [12]:
```
x = data["Height"].values.tolist();
y = data["Weight"].values.tolist();

plt.figure(figsize=(50, 20));
plt.title("Height-Weight Graphic")
plt.scatter(x, y, label= "dot", color= "red",
            marker= ".", s=20)

plt.plot(x, y, "bo")

plt.xlabel("Height")
plt.ylabel("Weight")

plt.show()
```



Lineer Regression Model

Training

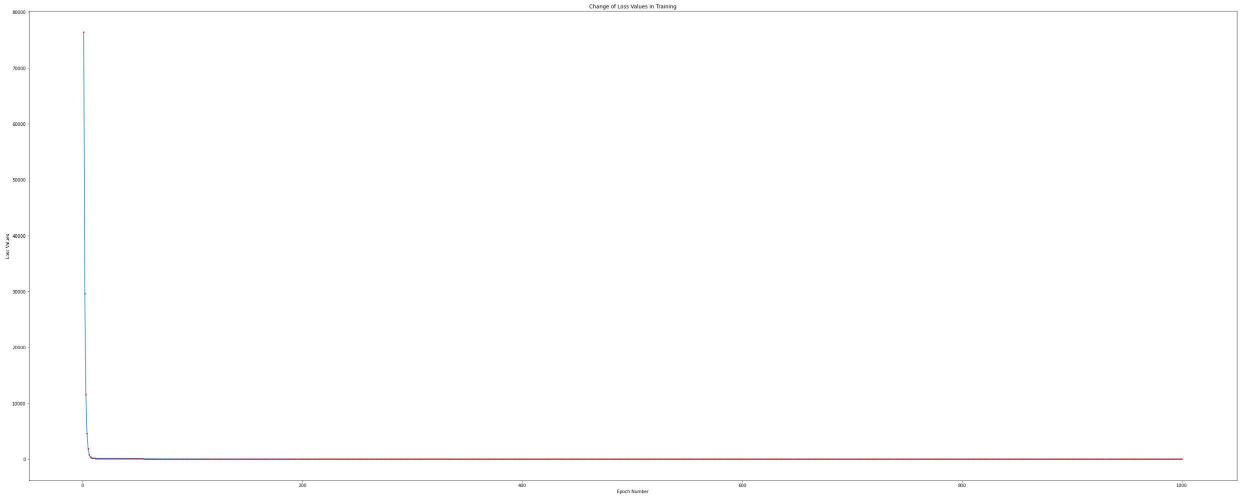
We split height, weight and index values to equal two parts. We will use first half of the data to traing our model. Data is already mixed. We can take first half of it as training data and the second half as test data.

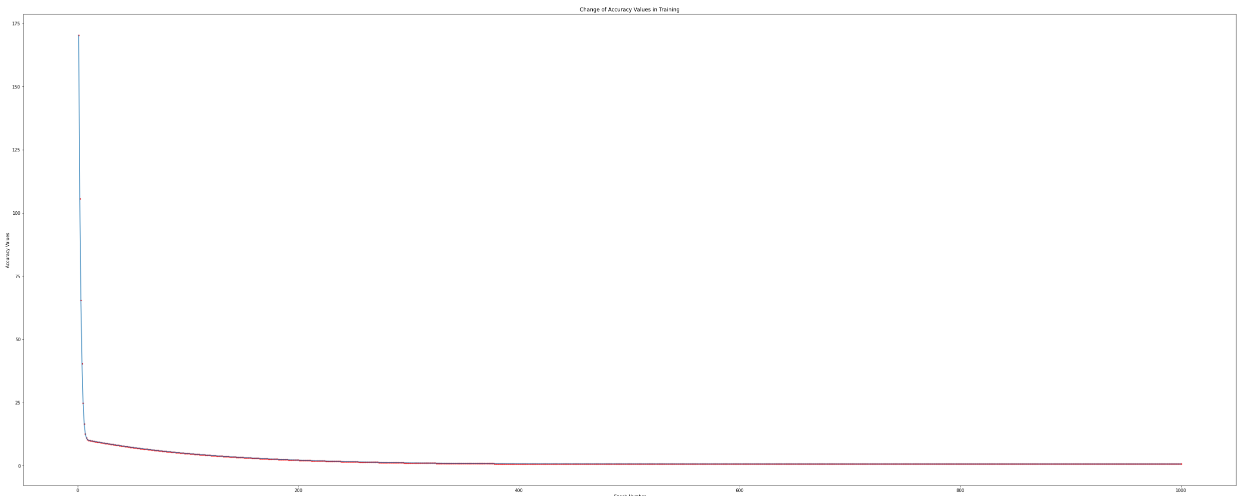
In [13]:
```
D = data.values.tolist();

X = data["Height"].values.tolist();
Y = data["Weight"].values.tolist();
Z = data["Index"].values.tolist();

x_train = X[:len(X)//2];
y_train = Y[:len(Y)//2];
z_train = Z[:len(Z)//2];

x_test = X[len(X)//2:];
y_test = Y[len(Y)//2:];
z_test = Z[len(Z)//2:];
```

Create Lineer Regression Decider and calculate Loss, m1, m2 and b values using epoch and learning rate.

In [14]:
```
lrm = LineerRegressionModel(learning_rate=0.000005, epoch=1000);
return_lists = lrm.fit(x_train, y_train, z_train, x_test, y_test, z_test);
loss_values = return_lists[0];
accuracy_values = return_lists[1];

y=np.arange(1,1001);

plt.figure(figsize=(50, 20));
plt.plot(y, loss_values);

plt.title("Change of Loss Values in Training");
plt.xlabel("Epoch Number");
plt.ylabel("Loss Values");

plt.scatter(y, loss_values, label= "dot", color= "red", marker= ".", s=25);

plt.show();

plt.figure(figsize=(50, 20));
plt.plot(y, accuracy_values);

plt.title("Change of Accuracy Values in Training");
plt.xlabel("Epoch Number");
plt.ylabel("Accuracy Values");

plt.scatter(y, accuracy_values, label= "dot", color= "red", marker= ".", s=25);

plt.show();

print("Description of Loss Values in Training");
df = pd.DataFrame(loss_values);
x = df.describe();
print(x);
print();
print("Description of Accuracy Values in Training");
df2 = pd.DataFrame(accuracy_values);
```

```
y = df2.describe();
print(y);
```

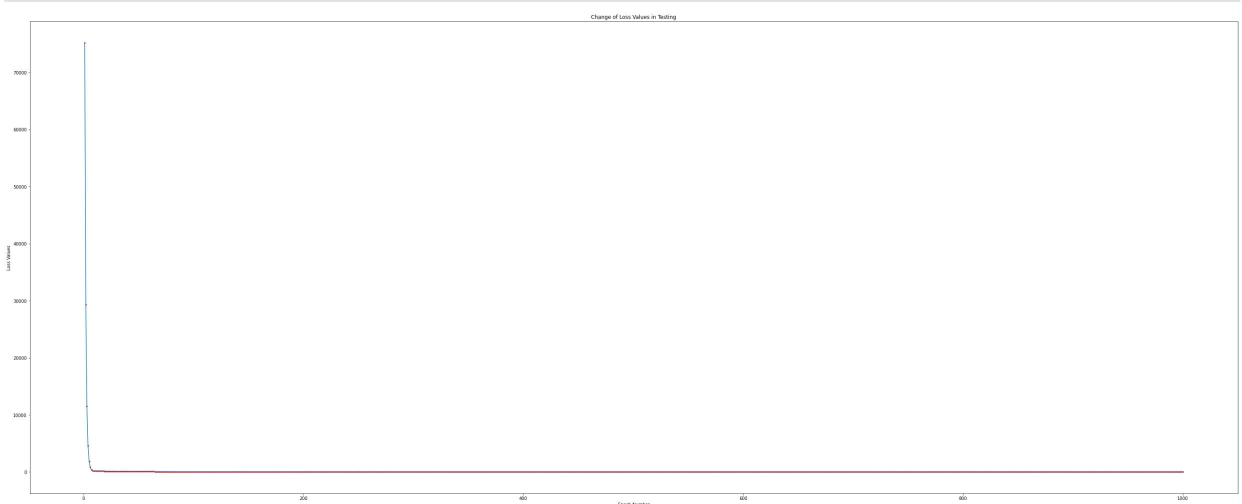


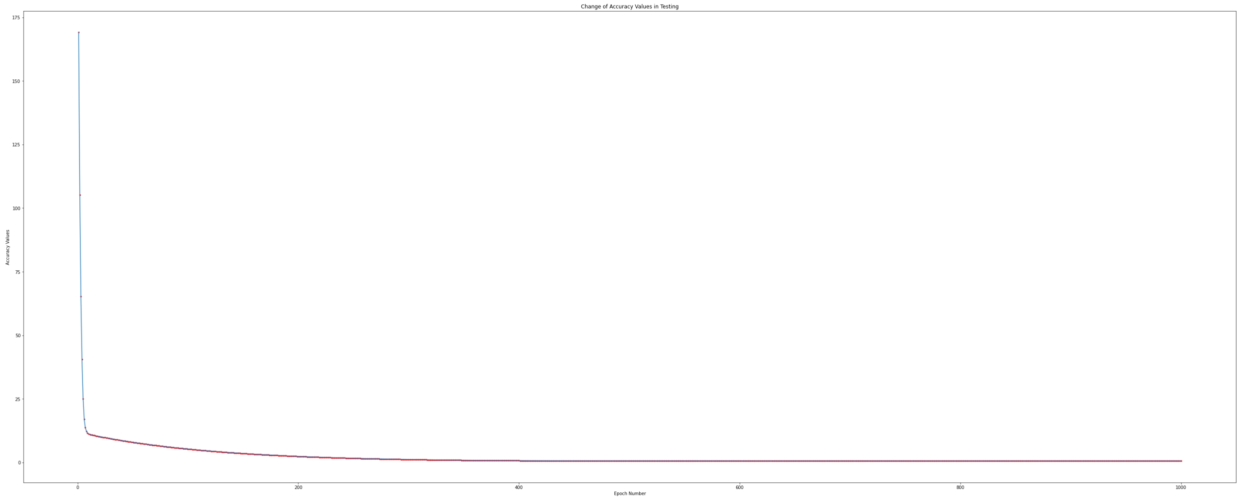



```
Description of Loss Values in Training
                  0
count   1000.000000
mean     135.167716
std     2620.847244
min        0.683892
25%        0.684526
50%        0.724378
75%        3.296848
max    76424.911765

Description of Accuracy Values in Training
                 0
count  1000.000000
mean      2.092425
std       7.040328
min       0.668142
25%       0.670595
50%       0.701266
75%       1.539085
max     170.266689
```

Testing

In [15]:
```
loss_valuesT = return_lists[2];
accuracy_valuesT = return_lists[3];

y=np.arange(1,1001);

plt.figure(figsize=(50, 20));
plt.plot(y, loss_valuesT);

plt.title("Change of Loss Values in Testing");
plt.xlabel("Epoch Number");
plt.ylabel("Loss Values");

plt.scatter(y, loss_valuesT, label= "dot", color= "red", marker= ".", s=25);

plt.show();

plt.figure(figsize=(50, 20));
plt.plot(y, accuracy_valuesT);

plt.title("Change of Accuracy Values in Testing");
plt.xlabel("Epoch Number");
plt.ylabel("Accuracy Values");

plt.scatter(y, accuracy_valuesT, label= "dot", color= "red", marker= ".", s=25);

plt.show();

print("Description of Loss Values in Testing");
df = pd.DataFrame(loss_valuesT);
x = df.describe();
print(x);
print();
print("Description of Accuracy Values in Testing");
df2 = pd.DataFrame(accuracy_valuesT);
y = df2.describe();
print(y);
```






```
Description of Loss Values in Testing
                  0
count   1000.000000
mean     134.763438
std     2580.324732
min        0.555005
25%        0.556023
50%        0.603803
75%        3.550252
max    75193.000000

Description of Accuracy Values in Testing
                 0
count  1000.000000
mean      2.157303
std       7.084272
min       0.580831
25%       0.583156
50%       0.622557
75%       1.652211
max     169.131924
```

In [16]:
```
prediction = lrm.predict(x_test, y_test);

prediction_result = [round(num) for num in prediction]
expected_result = z_test;

print("Test Features Expected Results")
print(expected_result)
print()
print("Prediction Results")
print(prediction_result)
print()
print("Actual Prediction Results")
print(prediction)
```

```
Test Features Expected Results
[1, 5, 0, 5, 3, 5, 3, 4, 3, 5, 5, 5, 5, 2, 4, 5, 5, 4, 5, 5, 2, 4, 5, 5, 5, 5, 1, 5,
5, 4, 0, 3, 3, 4, 2, 3, 1, 1, 5, 5, 4, 4, 4, 4, 5, 2, 5, 4, 3, 3, 4, 5, 5, 2, 4, 3,
4, 5, 4, 2, 4, 5, 4, 5, 5, 1, 5, 5, 5, 2, 2, 5, 3, 5, 4, 5, 4, 5, 5, 4, 2, 2, 3, 3,
5, 4, 2, 2, 2, 5, 5, 4, 5, 3, 4, 4, 3, 4, 4, 2, 5, 2, 2, 2, 2, 5, 4, 5, 1, 4, 1, 4,
5, 4, 5, 3, 4, 5, 4, 3, 5, 1, 2, 4, 5, 5, 5, 5, 3, 5, 5, 5, 2, 5, 4, 3, 2, 2, 2, 2,
3, 5, 3, 5, 5, 4, 2, 4, 4, 5, 2, 5, 5, 5, 1, 4, 5, 5, 5, 4, 5, 2, 5, 1, 5, 4, 1, 1,
4, 4, 4, 4, 2, 5, 5, 4, 2, 5, 5, 5, 1, 5, 4, 2, 5, 5, 4, 5, 4, 4, 5, 5, 5, 4, 5, 0,
2, 2, 4, 2, 4, 5, 4, 5, 5, 2, 5, 5, 5, 3, 5, 5, 2, 5, 3, 4, 5, 2, 2, 5, 5, 4, 4, 4,
4, 5, 2, 4, 5, 2, 5, 1, 5, 5, 4, 5, 5, 5]

Prediction Results
[2, 6, 1, 4, 3, 6, 3, 3, 2, 5, 5, 6, 5, 2, 4, 4, 5, 5, 5, 5, 2, 3, 3, 5, 5, 5, 2, 5,
4, 4, 2, 3, 3, 4, 3, 2, 2, 2, 5, 3, 4, 5, 5, 3, 6, 2, 4, 2, 4, 2, 3, 5, 6, 3, 3, 3,
3, 5, 4, 3, 4, 5, 4, 5, 3, 2, 5, 6, 5, 2, 2, 5, 3, 5, 3, 5, 4, 4, 5, 2, 2, 3, 2, 4,
6, 2, 2, 2, 2, 6, 4, 4, 5, 2, 4, 3, 3, 4, 4, 2, 5, 2, 3, 3, 2, 5, 4, 5, 2, 5, 2, 4,
6, 5, 5, 3, 4, 5, 2, 2, 4, 2, 2, 3, 5, 3, 5, 3, 3, 4, 6, 5, 2, 5, 3, 2, 2, 2, 2, 3,
3, 6, 3, 5, 5, 4, 2, 4, 5, 5, 2, 5, 4, 5, 2, 3, 6, 6, 4, 3, 4, 2, 5, 2, 5, 4, 2, 2,
5, 5, 3, 3, 3, 6, 3, 4, 2, 6, 5, 6, 2, 6, 4, 2, 5, 5, 5, 5, 2, 4, 4, 5, 6, 3, 4, 2,
3, 3, 5, 3, 3, 5, 4, 5, 5, 3, 5, 5, 4, 4, 5, 4, 2, 5, 3, 5, 5, 2, 2, 4, 4, 4, 3, 4,
4, 4, 3, 5, 5, 2, 5, 2, 3, 6, 4, 5, 3, 5]

Actual Prediction Results
[1.943214824741879, 5.574833703823112, 1.4977339083628736, 4.049631067999335, 3.35744
28192177117, 5.812112521639207, 3.033741416969892, 3.0586890616329194, 2.342344343681
884, 5.197709518196399, 5.051963129852098, 5.764969948706591, 5.251132301575292, 1.85
67922403101547, 3.8066572224400677, 4.245462340625988, 5.206742445036115, 4.668352740
988596, 4.67915761377184, 4.927826487074281, 1.6364089107672828, 2.898214718705729,
2.989747148177522, 4.965144957673566, 5.242099374735576, 5.325184060070758, 1.5330908
380623354, 4.682891100615189, 3.621630822596657, 4.2356382382926565, 1.81574028286751
95, 2.78644890186417, 3.1908778608005406, 3.983631465893673, 2.6420788717165875, 2.49
10330433759205, 1.7568120667017495, 2.05458505383663, 4.52339752813791, 3.28516300666
5772, 3.7434103367278437, 4.920359513387581, 4.94628792850052, 2.5041950448057935, 5.
765950719156503, 1.6387660394139136, 3.512617722148536, 2.490052272926009, 3.62594949
21431106, 2.2437405282701977, 2.8170915742703704, 5.131709916090737, 5.65359971961183
9, 3.001137203663869, 2.60515598886411, 3.113488203208532, 2.8113965465271975, 4.9731
9711406337, 3.744391107177755, 2.6854879578058535, 3.828852150709656, 4.5021833703182
335, 3.9624173080739955, 5.4833012743513185, 3.0600654198296384, 2.202292983080755,
5.047248872558836, 5.606457146679223, 5.193580443606242, 2.1649745124814 7, 1.89175358
22628093, 4.73906660038752, 3.381994876133931, 4.883436630535104, 3.269643876589268,
5.092034316844821, 4.112482365964752, 3.657968522746031, 4.970839985416739, 2.2087791
863175434, 1.7350127261789687, 3.0852026594489623, 2.3551107573649497, 3.505150748461
837, 5.798950520209334, 2.406967587590827, 2.2366691423303053, 1.9880002690278642, 1.
8345973120405659, 5.521015332697411, 4.483326341145187, 4.385703296183412, 5.22559947
4209161, 2.3763249151846266, 4.157267810250737, 3.1912734485473484, 2.784091773217539
4, 4.043540452509355, 3.6622871922924847, 2.1753837975179047, 4.9146644856444075, 1.9
459675411353174, 2.7723061299843854, 2.9351376015582065, 1.5906426960313862, 5.480944
1457046875, 4.3470084673874085, 5.089281600451383, 1.8143639246708003, 5.282755744431
404, 1.8998057386526133, 3.60041666477698, 5.752203535023525, 5.222846757815723, 5.44
2249316908683, 3.195592118093802, 4.470164339715313, 4.728657315351087, 2.24040262917
36554, 2.037499970607111, 4.4153651981397015, 1.8799679390296553, 2.0479092556435456,
3.260215362002745, 5.212833060526096, 3.1808641635109134, 5.0694438008284255, 3.34605
27637313655, 2.945942474341449, 4.2629430116023155, 5.597424219839508, 5.330879087813
932, 2.3160203408221376, 4.990092602336593, 2.8288772175035244, 2.178136513911343, 1.
5755191537016897, 1.8450065970770007, 2.2787018702228523, 2.5037994570589857, 2.92944
2573815033, 5.5950670911928775, 3.268267518392549, 4.858488985872076, 4.9113265865478
65, 3.7245533075547974, 2.273987612929591, 3.852423437175964, 4.569748925576911, 5.10
1462831431344, 2.1871694407510587, 5.13878130203063, 4.409670170396528, 4.74947588542
3956, 1.7393313957254226, 3.485312948838878, 5.5267103604405845, 5.5738529333732, 3.6
09845179363503, 3.388085491623912, 4.047273939352705, 2.0889612130861805, 4.642819913
```

```
622466, 1.8950914813593516, 5.205366086839396, 3.715124792968274, 1.673727381366568,
1.853058753466805, 5.007177685566113, 4.5838916974566954, 2.613208145253914, 3.281429
519822422, 2.6316695866801525, 5.569138676079938, 3.3941761071138927, 4.4866642402417
29, 2.0045001695542797, 5.712132348030802, 4.780118557830156, 5.635723460888705, 2.06
6370697069784, 5.523372461344042, 4.077521024012097, 2.390072099317604, 5.42280710503
2533, 5.285508460824843, 4.508859168511318, 5.036839587522402, 2.3588442442082993, 4.
261566653405596, 3.558383936884433, 5.230313731502423, 5.687580291114583, 3.071455475
315985, 4.278066553932011, 1.5000910370095042, 2.6198839434469985, 2.558994186381405
6, 4.504144911218057, 2.7935202878040624, 2.765630331791301, 5.248775172928661, 4.331
884925057712, 4.7437808576807825, 5.108534217371236, 2.663293029536265, 4.53616394182
0975, 5.147229046167241, 4.0647546103290315, 3.7453718776276665, 5.14251478887398, 3.
965170024467434, 2.089941983536092, 5.455015730591749, 2.657598001793091, 4.632015040
8392236, 5.102443601881256, 2.0771755698530265, 1.7666361690350805, 3.788195781013828
7, 4.078897382208816, 3.767962393644063, 2.939266676148364, 4.001112136870001, 3.8816
89751385445, 3.834942766199637, 2.6491502576564803, 4.790132255119783, 5.376645302549
829, 1.8907728118128977, 5.338346061500632, 1.6444610671570867, 3.4281566786166353,
5.561086519690135, 4.242709624232549, 4.924488587977738, 3.316786449521884, 4.6555863
27305532]
```

NOTE => The actual prediction values are used to calculate accuracy with Mean Error. The accuracy values are rounded to just show properly.

NOTE => In Training part, we have 1000 epoch. That's why, I did not print the output of the expected output and our output after each epoch turn.

Results

NOTE => Loss and Accuracy graphics are in the Trainig and Testing section of the report!

In [20]:

```
asd = lrm.predict(x_test, y_test)
err = 0
for i in range (len(y_test)):
    err = err + abs(z_test[i] - asd[i])
err = err / len(y_test)
print("After we train our model, this is the last Mean Absolute Error Value for test d
print(err)
```

```
After we train our model, this is the last Mean Absolute Error Value for test data:
0.5808307740977067
```

m1, m2 and b Values

In [17]:

```
r = lrm.values();
m1 = r[0]
m2 = r[1]
b = r[2]
print("m1: ", m1);
print("m2: ", m2);
print("b: ", b);
```

```
m1:  -0.002357128646630796
m2:  0.03869482879600432
b:  -0.005652989103906824
```

"Loss Values" and "Accuracy Values" charts for test and training data are available below the relevant sections in the report.

When we start to train our lineer regression model, during the first epochs accuracy values are like -170. This is because we randomly selected the initial assigned values for m1, m2, and b. After we train our model, accuracy values narrow down to 0.5 and 0.7 range. Because in every epoch, we calculate again loss value, m1, m2 and b according to result of last epoch.

Same goes for loss values. During the first epochs it is too high. This is also because we randomly selected the initial assigned values for m1, m2 and b. After the model is trained, accuracy values narrow down to 0.5 and 0.6 range as similar as accuracy values in training. But there is no overfitting. The accuracy value is lower than training as we expected. If we think about the overfitting graphic that we talked in our lectures, accuracy of training is usually more than testing.