assignment 2

Zahra Parham 2122841

1.1

in the first question, I generate 2000 samples from multivariate Gaussian distribution based on the mean and covariance that was given.

The steps I took:

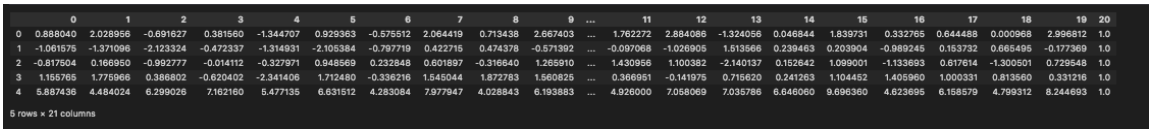
1. read the first mean, second mean and covariance matrix with pandas:

```
the first mean matrix[2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.]
```

```
the zero mean matrix[1.3 1.3 1.3 1.3 1.3 1.3 1.3 1.3 1.3 1.3 1.3 1.3 1.3 1.3 1.3 1.3 1.3
 1.3 1.3]
```

```
the covariance matrix[[7.92747613 5.48849335 6.18812365 5.17766549 5.84803095 6.15325514
  4.6623167  5.45500286 4.98574312 5.18758586 3.9436397  5.26132514
  7.09637106 5.99810181 6.0932167  5.99518462 5.87317499 5.70538764
  5.59907089 5.91786273]
 [5.48849335 6.81598145 5.34490996 4.31020106 5.41645238 5.53995027
  4.30811201 3.93658438 4.11429237 4.98428213 3.34393835 4.59050685
  5.85714703 5.07764986 5.41356597 5.19974264 5.60959615 5.09170926
  5.31814733 5.27567458]
 [6.18812365 5.34490996 7.322937   4.87361613 5.78994181 6.60647915
  4.61175555 4.83989875 4.86905732 5.05339559 3.26344459 4.68117364
  6.36913218 5.17240371 6.107158   5.99640412 6.24914837 4.99487173
  4.72323191 5.0810024 ]
 [5.17766549 4.31020106 4.87361613 5.77060116 5.26312518 4.50414113
  3.70666948 4.39831182 3.37569889 4.20589158 2.74695007 4.1714241
  5.86605885 4.78774458 4.78672203 5.08695389 4.67678137 4.57195323
  3.91537274 5.75972062]
 [5.84803095 5.41645238 5.78994181 5.26312518 7.01239219 5.39143047
  5.00679847 4.38069986 4.64759399 5.11560693 4.07101072 4.99248684
  6.2216833  5.83054834 5.95588404 6.17991278 5.89633465 5.15754486
  5.35690654 5.72019074]
 [6.15325514 5.53995027 6.60647915 4.50414113 5.39143047 6.73659871
  4.45649259 4.90819445 4.73391895 5.30643771 2.98349649 4.82631716
  6.37875695 4.98412722 5.80736823 6.00013022 5.93975039 4.99010068
  4.79687342 5.19390579]
 [4.6623167  4.30811201 4.61175555 3.70666948 5.00679847 4.45649259

show more (open the raw output data in a text editor) ...

  6.35142867 4.89494004]
 [5.91786273 5.27567458 5.0810024  5.75972062 5.72019074 5.19390579
  3.8718588  4.49208748 4.04194538 5.2901418  3.44846334 4.48212019
```

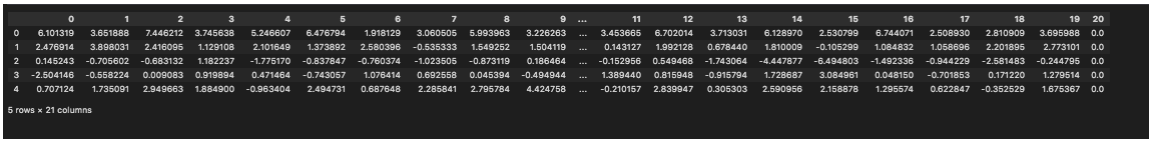
2. generate data from multivariate Gaussian distribution with Ds1_m1 as mean and Ds1_cov as covariance and label this data to be positive(Y = 1)



	0	1	2	3	4	5	6	7	8	9	...	11	12	13	14	15	16	17	18	19	20
0	0.888040	2.028956	-0.691627	0.381560	-1.344707	0.929363	-0.575512	2.064419	0.713438	2.667403	...	1.762272	2.884086	-1.324056	0.046844	1.839731	0.332765	0.644488	0.000968	2.996812	1.0
1	-1.061575	-1.371096	-2.123324	-0.472337	-1.314931	-2.105384	-0.797719	0.422715	0.474378	-0.571392	...	-0.097068	-1.026905	1.513566	0.239463	0.203904	-0.989245	0.153732	0.665495	-0.177369	1.0
2	-0.817504	0.166950	-0.992777	-0.014112	-0.327971	0.948569	0.232848	0.601897	-0.316640	1.265910	...	1.430956	1.100382	-2.140137	0.152642	1.099001	-1.133693	0.617614	-1.300501	0.729548	1.0
3	1.155765	1.775966	0.386802	-0.620402	-2.341406	1.712480	-0.336216	1.545044	1.872783	1.560825	...	0.366951	-0.141975	0.715620	0.241263	1.104452	1.405960	1.000331	0.813560	0.331216	1.0
4	5.887436	4.484024	6.299026	7.162160	5.477135	6.631512	4.283084	7.977947	4.028843	6.193883	...	4.926000	7.058069	7.035786	6.646060	9.696360	4.623695	6.158579	4.799312	8.244693	1.0

5 rows × 21 columns

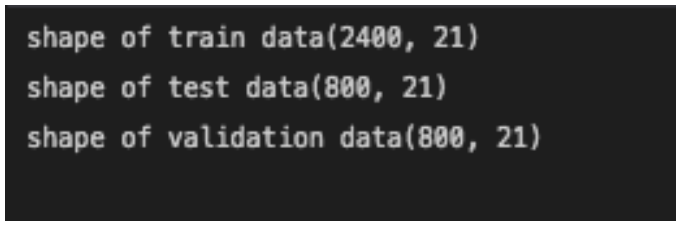
3. generate data from multivariate Gaussian distribution with Ds1_m0 as mean and Ds1_cov as covariance and label this data to be negative(Y = 0)



	0	1	2	3	4	5	6	7	8	9	...	11	12	13	14	15	16	17	18	19	20
0	6.101319	3.651888	7.446212	3.745638	5.246607	6.476794	1.918129	3.060505	5.993963	3.226263	...	3.453665	6.702014	3.713031	6.128970	2.530799	6.744071	2.508930	2.810909	3.695988	0.0
1	2.476914	3.898031	2.416095	1.129108	2.101649	1.373892	2.580396	-0.535333	1.549252	1.504119	...	0.143127	1.992128	0.678440	1.810009	-0.105299	1.084832	1.058696	2.201895	2.773101	0.0
2	0.145243	-0.705602	-0.683132	1.182237	-1.775170	-0.837847	-0.760374	-1.023505	-0.873119	0.186464	...	-0.152956	0.549468	-1.743064	-4.447877	-6.494803	-1.492336	-0.944229	-2.581483	-0.244795	0.0
3	-2.504146	-0.558224	0.009083	0.919894	0.471464	-0.743057	1.076414	0.692558	0.045394	-0.494944	...	1.389440	0.815948	-0.915794	1.728687	3.084961	0.048150	-0.701853	0.171220	1.279514	0.0
4	0.707124	1.735091	2.949663	1.884900	-0.963404	2.494731	0.687648	2.285841	2.795784	4.424758	...	-0.210157	2.839947	0.305303	2.590956	2.158878	1.295574	0.622847	-0.352529	1.675367	0.0

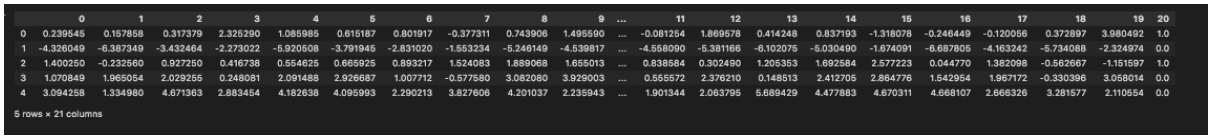
5 rows × 21 columns

4. pick the train and test and validation data based on the percentage in assignment and also pick the data randomly without replacement(by shuffling the data at the first step)



5. I saved the test dataset as DS1_test.csv, the train dataset as DS1_train, and the validation dataset as DS1_valid.csv. also I saved all this dataset together as DS1.csv

DS1 dataset:



	0	1	2	3	4	5	6	7	8	9	...	11	12	13	14	15	16	17	18	19	20
0	0.239545	0.157858	0.317379	2.325290	1.085985	0.615187	0.801917	-0.377311	0.743906	1.495590	...	-0.081254	1.869578	0.414248	0.837193	-1.318078	-0.246449	-0.120056	0.372897	3.980492	1.0
1	-4.326049	-6.387349	-3.432464	-2.273022	-5.920508	-3.791945	-2.831020	-1.553234	-5.246149	-4.539817	...	-4.558090	-5.381166	-6.102075	-5.030490	-1.674091	-6.687805	-4.163242	-5.734088	-2.324974	0.0
2	1.400250	-0.232560	0.927250	0.416738	0.554625	0.665925	0.893217	1.524083	1.889068	1.655013	...	0.838584	0.302490	1.205353	1.692584	2.577223	0.044770	1.382098	-0.562667	-1.151597	1.0
3	1.070849	1.965054	2.029255	0.248081	2.091488	2.926687	1.007712	-0.577580	3.082080	3.929003	...	0.555572	2.376210	0.148513	2.412705	2.864776	1.542954	1.967172	-0.330396	3.058014	0.0
4	3.094258	1.334980	4.671363	2.883454	4.182638	4.095993	2.290213	3.827606	4.201037	2.235943	...	1.901344	2.063795	5.689429	4.477883	4.670311	4.668107	2.666326	3.281577	2.110554	0.0

5 rows × 21 columns

6. Also I saved all these dataset in text file too :-)

1.2

1. for the second question at first I implement the GDA function

1.2.a

The best fit accuracy achieved by the classifier is 95.5 %:

```
the best accuracy which is obtained from test data : 95.5 %
```

1.2.b

The coefficient learnt is:

```
the coefficients:

 W:

 [-14.52871582   8.6679611    5.47148688   2.64368349   9.49078082
   4.31224404 -16.07282748  24.25975005  28.88107278  -9.53380856
  13.28764016  11.78327262 -15.16782655 -12.33807653   5.53784804
 -13.17590736 -28.97928019   6.88347419  -0.18497918   5.25040029]

 W0 : -27.174500816893772
```

1.3

for the third question I implemented the KNN function:

1.3.a

This classifier performs worse than GDA function and the best accuracy is for k=37 with the accuracy of 57.75 on validation dataset.since KNN is completely non-parametric approach, no assumptions made about the shape of the decision boundary K is the hyperparameter that refers to the number of nearest neighbours to include in the majority of the voting process so basically depending on different values of K, the accuracy might differ. Actually we want to tune our algorithm by finding the best value for K so it will change the prediction as well. Typically the k value is set to the square root of the number of samples in training set which here is 2400 so typically it is a good idea to set the K around 49.

```
k value = 1 =======> accuracy = 50.5
k value = 3 =======> accuracy = 52.375
k value = 5 =======> accuracy = 52.125
k value = 7 =======> accuracy = 52.625
k value = 9 =======> accuracy = 53.5
k value = 11 =======> accuracy = 54.0
k value = 13 =======> accuracy = 53.5
k value = 15 =======> accuracy = 51.375
k value = 17 =======> accuracy = 53.25
k value = 19 =======> accuracy = 54.875
k value = 21 =======> accuracy = 55.375
k value = 23 =======> accuracy = 56.5
k value = 25 =======> accuracy = 57.5
k value = 27 =======> accuracy = 57.125
k value = 29 =======> accuracy = 56.875
k value = 31 =======> accuracy = 57.0
k value = 33 =======> accuracy = 56.75
k value = 35 =======> accuracy = 56.75
k value = 37 =======> accuracy = 57.75
k value = 39 =======> accuracy = 55.875
k value = 41 =======> accuracy = 56.75
k value = 43 =======> accuracy = 56.875
k value = 45 =======> accuracy = 56.0
k value = 47 =======> accuracy = 56.625
k value = 49 =======> accuracy = 57.125

show more (open the raw output data in a text editor) ...

k value = 153 =======> accuracy = 55.0
k value = 155 =======> accuracy = 54.625
k value = 157 =======> accuracy = 54.625
k value = 159 =======> accuracy = 55.0
the best k is : 37 and the best accuracy is : 57.75
```

1.3.b

the best accuracy for test dataset is 54.5:

```
test accuracy for k = 37 is : 54.5
```

```
k value = 1 ======> accuracy = 49.125
k value = 3 ======> accuracy = 49.625
k value = 5 ======> accuracy = 51.125
k value = 7 ======> accuracy = 53.25
k value = 9 ======> accuracy = 54.0
k value = 11 ======> accuracy = 52.625
k value = 13 ======> accuracy = 53.25
k value = 15 ======> accuracy = 54.125
k value = 17 ======> accuracy = 55.25
k value = 19 ======> accuracy = 54.125
k value = 21 ======> accuracy = 56.375
k value = 23 ======> accuracy = 54.25
k value = 25 ======> accuracy = 54.125
k value = 27 ======> accuracy = 54.25
k value = 29 ======> accuracy = 53.5
k value = 31 ======> accuracy = 53.875
k value = 33 ======> accuracy = 54.25
k value = 35 ======> accuracy = 54.875
k value = 37 ======> accuracy = 54.5
k value = 39 ======> accuracy = 55.5
k value = 41 ======> accuracy = 56.5
k value = 43 ======> accuracy = 56.875
k value = 45 ======> accuracy = 56.375
k value = 47 ======> accuracy = 57.25
k value = 49 ======> accuracy = 57.125

show more (open the raw output data in a text editor) ...

k value = 153 ======> accuracy = 56.125
k value = 155 ======> accuracy = 56.5
k value = 157 ======> accuracy = 56.125
k value = 159 ======> accuracy = 55.625
the best k is : 93 and the best accuracy is : 58.5
```

1.4

for this question I repeated the exact step for 1.1 and this is DS2 dataset:

	0	1	2	3	4	5	6	7	8	9	...	11	12	13	14	15	16	17	18	19	20
0	1.135183	1.002957	2.204475	2.234593	1.718117	1.579917	3.185297	2.610832	1.552993	2.526851	...	-1.234818	0.808853	-0.266035	0.049918	0.539747	-0.164770	0.594459	3.441934	1.872095	1.0
1	4.134915	5.070674	5.743357	4.823102	4.003875	4.113020	4.716796	3.940880	4.916695	3.768333	...	3.673463	4.337357	5.376726	4.758728	4.070818	3.900327	3.104892	4.999454	4.574333	1.0
2	0.589231	-3.333033	0.373499	0.584210	-2.327226	-1.253408	0.858874	-0.933623	1.220143	-0.033987	...	1.242410	0.339465	-0.497557	-0.262457	-0.914020	-0.114639	0.816679	-1.872823	0.604071	1.0
3	1.615361	4.232913	3.421192	1.973626	2.802039	4.034054	4.015263	3.580293	2.536173	3.141107	...	4.452235	2.117597	2.821531	1.752671	3.122594	2.183314	4.915613	3.956169	4.886782	0.0
4	-0.339204	0.800596	1.563569	2.380164	2.127669	0.281610	-0.174989	-1.384316	-0.323583	2.554605	...	-0.659739	2.241526	-1.900959	0.387279	0.991670	0.930648	1.304879	1.235094	0.116769	1.0

5 rows × 21 columns

5.1.a

```
the best accuracy which is obtained from test data : 53.125 %
```

5.1.b

```
the coefficients:

 W:

 [-0.01477806  0.08812504  0.05462724 -0.03620929 -0.13729712  0.02569758
  0.03158517 -0.03190441 -0.09910052  0.02377519  0.06182258 -0.06146903
 -0.03351293 -0.0809642   0.06672574  0.03365209  0.02585048  0.02596038
  0.01378598 -0.01928984]

 W0 : 0.06408193955440583
```

5.2

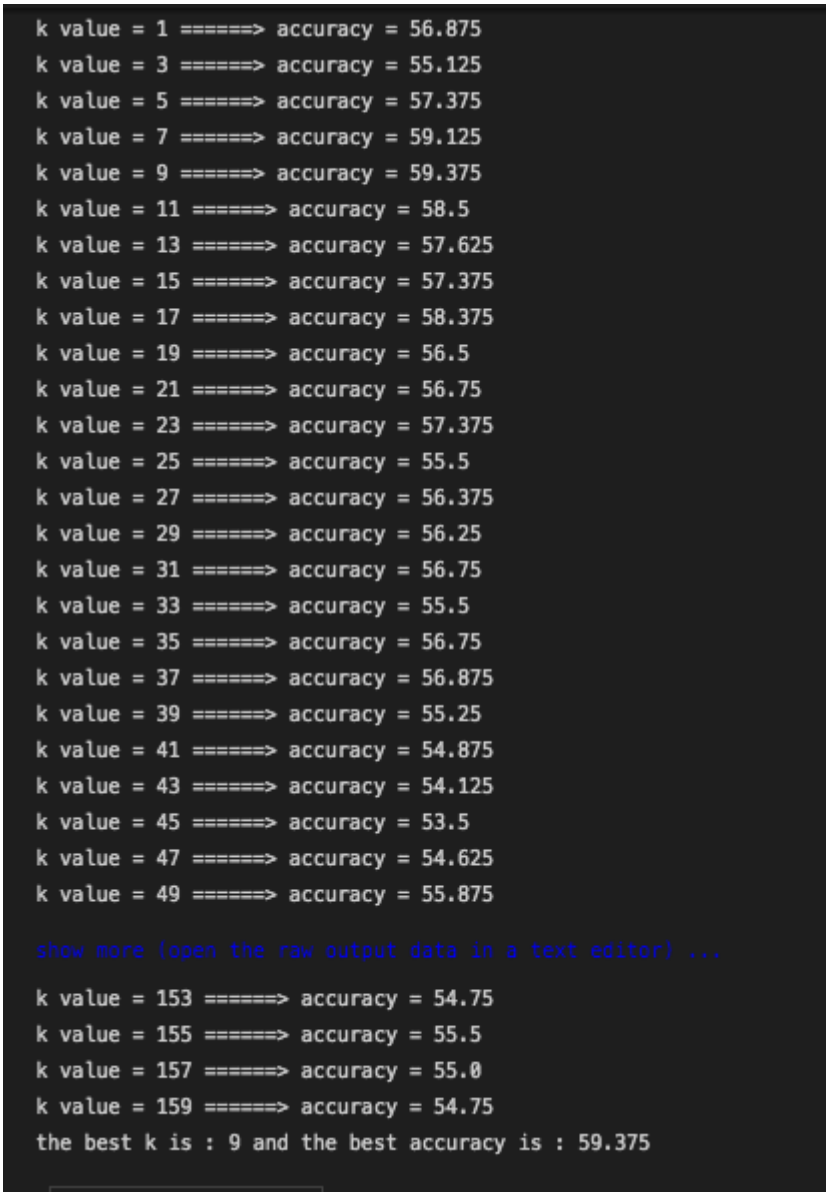
The best accuracy for validation data is 55.5 for k = 59. The performance of KNN is better tha GDA. Since the variance in DS2 is bigger than DS1 so GDA perform worse. K is the hyperparameter that refers to the number of nearest neighbours to include in the majority of the voting process so basically depending on different values of K, the accuracy might differ. Actually we want to tune our algorithm by finding the best value for K so it will change the prediction as well. Typically the k value is set to the square root of the number of samples in training set which here is 2400 so typically it is a good idea to set the K around 49.

```
k value = 1 ======> accuracy = 53.5
k value = 3 ======> accuracy = 53.125
k value = 5 ======> accuracy = 55.125
k value = 7 ======> accuracy = 54.5
k value = 9 ======> accuracy = 54.625
k value = 11 ======> accuracy = 55.125
k value = 13 ======> accuracy = 53.5
k value = 15 ======> accuracy = 53.5
k value = 17 ======> accuracy = 54.75
k value = 19 ======> accuracy = 54.25
k value = 21 ======> accuracy = 53.625
k value = 23 ======> accuracy = 53.75
k value = 25 ======> accuracy = 55.25
k value = 27 ======> accuracy = 54.25
k value = 29 ======> accuracy = 53.5
k value = 31 ======> accuracy = 53.25
k value = 33 ======> accuracy = 52.75
k value = 35 ======> accuracy = 52.75
k value = 37 ======> accuracy = 53.5
k value = 39 ======> accuracy = 53.375
k value = 41 ======> accuracy = 52.25
k value = 43 ======> accuracy = 54.0
k value = 45 ======> accuracy = 54.0
k value = 47 ======> accuracy = 53.875
k value = 49 ======> accuracy = 55.125

show more (open the raw output data in a text editor) ...

k value = 153 ======> accuracy = 50.75
k value = 155 ======> accuracy = 50.75
k value = 157 ======> accuracy = 51.125
k value = 159 ======> accuracy = 50.25
the best k is : 59 and the best accuracy is : 55.5
```

5.3

the best accuracy for test dataset for k=9 is 59.375 %. and for the previous K=59 the accuracy is 53.75.

```
for the best k which is : 59 the accuracy is : 53.75
```



6

GDA: GDA function perform better on the fist dataset since GDA is perform the best for the data that come from a normal distribution since the second dataset (DS-2) is come from three different normal distribution so it's normal that it performs better on DS-1 cause the covariance of the data are not the same. and also the variance within each class of DS-1 are smaller than DS-2 so it results minimization in class overlap. and also for having best performance for GDA it should be better that the feature of the data be mutually independent and since we just generate the data randomly we can see that they are independent from together.

it perform better than KNN for the first dataset

KNN: it actually performs the same in both classes cause KNN assume that the distribution locally smooth but it's not guaranteed in this dataset. but for the second dataset it perform better than GDA.

also GDA is so faster than KNN algorithm.

2.1.(a)

I just implement the GNB function and flatten each dataset(train, test, validation) and pass it to this function.

equation for prior probability:

$$p(C_k) = p(y = C_k) = \sum_{i=1}^{N} I(y_i = C_k)/N$$

The mean formula:

$$\mu(C_k) = \sum_{i=1}^{N} I(x_i = C_k) * x_i/N$$

The covariance formula:

$$\sigma^2(C_k) = \sum_{i=1}^{N} I(x_i = C_k)(x_i - \mu(C_k))^2/N$$

the prior probability for each class is:

```
The prior class probabilities is :
 [0.09864 0.11356 0.09936 0.10202 0.09718 0.09012 0.09902 0.1035  0.09684
 0.09976]
```

The first row for the mean for class = 0

```
mean for class conditional density for class 0 is :

[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 1.01776315e-04
2.01167247e-04 8.26932557e-05 0.00000000e+00 7.15614713e-05
1.95601355e-04 9.30299127e-05 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 2.30586963e-05
7.95127459e-06 0.00000000e+00 6.75858340e-05 1.59025492e-04
3.49856082e-05 0.00000000e+00 0.00000000e+00 7.31517262e-05
2.24225943e-04 2.32972345e-04 6.36101967e-05 1.37557050e-04
```

The first row for diagonal covariance matrices for class = 0 (cause this matrix is singular so I add some small constant(0.01) to make it non-singular

```
diagonal covariance matrices for class 0 is :

[0.01       0.01       0.01       0.01       0.01       0.01
0.01       0.01       0.01       0.01       0.01       0.01
0.01       0.01       0.01       0.01       0.01       0.01
0.01       0.01       0.01       0.01       0.01       0.01
0.01       0.01       0.01       0.01       0.01       0.01
0.01       0.01       0.01       0.01       0.01       0.01
0.01       0.01       0.01       0.01714684 0.02412618 0.01580681
0.01       0.01502512 0.02373534 0.01653266 0.01       0.01
0.01       0.01       0.01       0.01       0.01       0.01
0.01       0.01       0.01       0.01       0.01       0.01161921
0.01055835 0.01       0.01474595 0.01982281 0.01245673 0.01
0.01       0.01513679 0.02320786 0.02203611 0.01446678 0.01815784
0.0200382  0.01073437 0.01424344 0.01759352 0.01       0.01
0.01       0.01       0.01       0.01       0.01       0.01
0.01       0.01       0.01       0.01469012 0.01558317 0.02598387
0.02478144 0.0235433  0.03063566 0.0347804  0.04632776 0.06841333
0.08633699 0.10336719 0.10927435 0.12480232 0.1230133  0.11286937
```

2.1.(b)

the parameter for GNB is :

```
 parameters is  : [1168.1520533453372, -4924.758580405665, -873.8434358775482, -4074.5714353191356, 1370.2676645798588, 1743.9452141197787, -7879.872527563546, 1271.678580489655, 1895.4193869314272, 1056.5173745332647]
```

the best accuracy is 70.8

```
best fit accuracy achieved : 70.8
```

2.2.(a)

the best accuracy is for K=3 with the accuracy of 97.2 %.

K is the hyperparameter that refers to the number of nearest neighbours to include in the majority of the voting process so basically depending on different values of K, the accuracy might differ. Actually we want to tune our algorithm by finding the best value for K so it will change the prediction as well.

```
k value = 1 ======> accuracy = 97.11999999999999
k value = 3 ======> accuracy = 97.2
k value = 5 ======> accuracy = 97.18
k value = 7 ======> accuracy = 97.08
k value = 9 ======> accuracy = 97.05
k value = 11 ======> accuracy = 96.92
k value = 13 ======> accuracy = 96.78
k value = 15 ======> accuracy = 96.64
the best k is : 3 and the best accuracy is : 97.2
```

2.2.(b)

The best accuracy on test data with k=3 is 96.81

```
the best k is : 3 and the best accuracy for test data is : 96.81
```

2.3

The K-NN performs better than GNB on MNIST dataset since

for GNB there is an assumption and that is that the feature of the dataset should be mutually independent but in MNIST it will not happen since the our data is image and the pixel of this image is our feature and we can see for sure that those features are not mutually independent, actually we can see that some pixel of each data is even the same(for example the entire black part) and also it's important for GNB that the data comes from a distribution which here this assumption it's not valid too. also GNB is much faster than KNN so it could be used for prediction in real time.

KNN has best performance on finding the similarity between observations(data) because of it's locally optimization

KNN perform good when the data are locally smooth so here we can assume that our data are locally smooth and it performs well cause it compute distances between the data and it makes decision base on that, here our dataset is images and we can capture the differences between two images based on their pixels(features) very well. and also about some percentage error that we have it because some picture of the number can be really the same like 3 and 8. and it can make wrong predict for them

actually KNN has the large real time computation compare to GNB