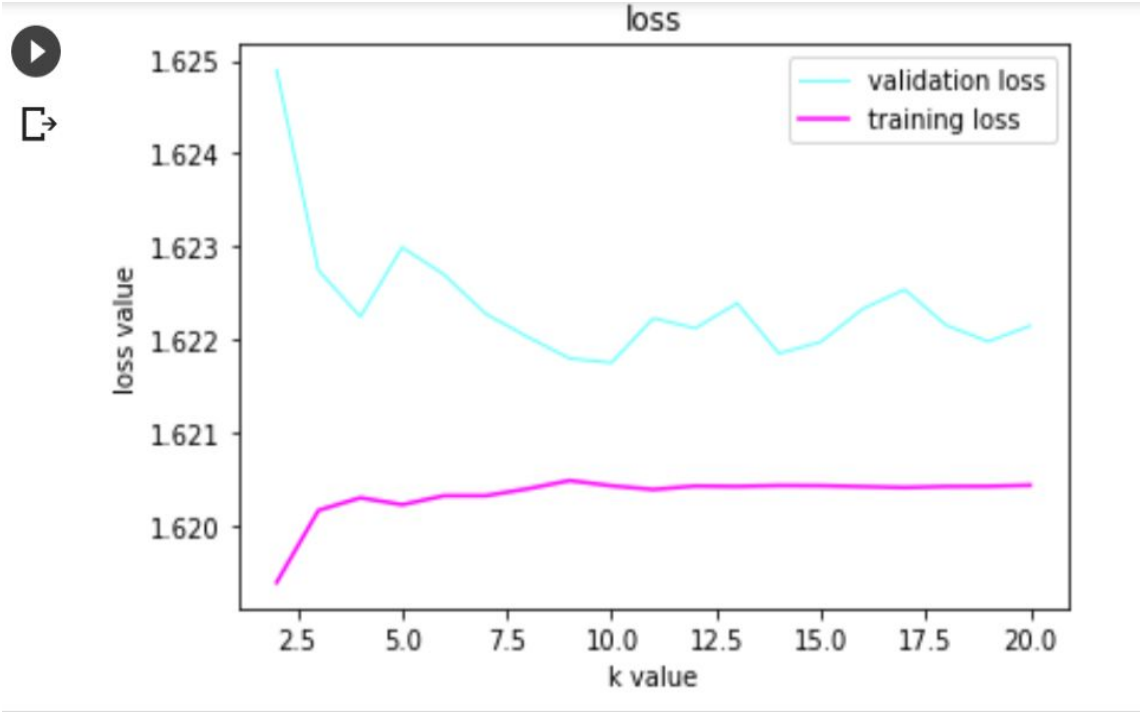
ML ASSIGNMENT 1 REPORT

• Choose an appropriate value of K and justify it in your report along with the preprocessing strategy.

I chose k=10 as the optimal value of 'k' sue to the following reasons:

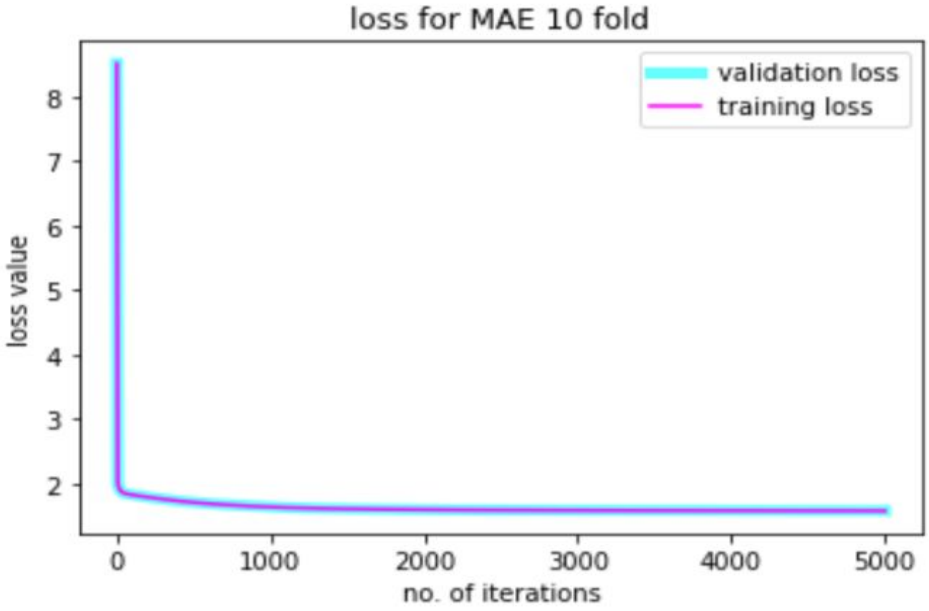
1) As k increases, the computation cost for splitting the parts, and running validation and training requires high computational cost. As both datasets are less than 10^4 and greater than 10^3, in the worst case scenario, it should not take more than 1 sec or 10^9 iterations thus larger k (K>15 or so) will increase our computation cost which is not feasible.
2) As number of folds increases, bias is low but variance is high thus taking a larger k, can lead to overfit of the model because redundancy in the training set increases. And vice versa in the smaller k, which has high bias and low variance can result into an underfit thus any k>15 and k<5 is eliminated.
3) There is a research which shows that k=10 is optimal value for choosing k for the datasets because the best training (ignoring overfit for a moment), can be achieved through LOOCV which is basically creating n-folds where n is size of sample data but that graph shows that the change in training loss and testing loss reduces by lot after k=10 thus k=10 is the best one can get.comparing LOOCV and k=10 with 15 of datasets and classifiers, it was seen in most cases that k=10 reaches the ideal ans as LOOCV.
4) I experimented for many values of k to see in which k difference between training loss and testing loss is minimum and thus i found out that every time k=10 gives the best ans for datasets.
   You can see below that k=10 reports min validation loss and testing loss reaches an asymptotic value after k=10.



Include plots between training loss v/s iterations and validation loss v/s itera-Tions.

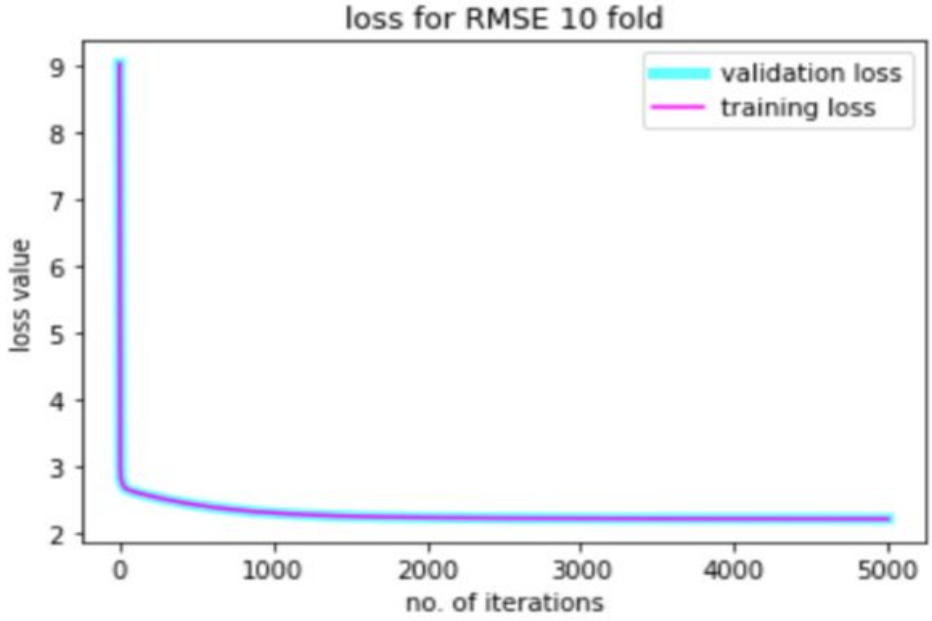
1) **DATASET 1 - MAE (with k=10)**

```
(4176, 8)
1.5755585988318344 1.5808338371223463
```



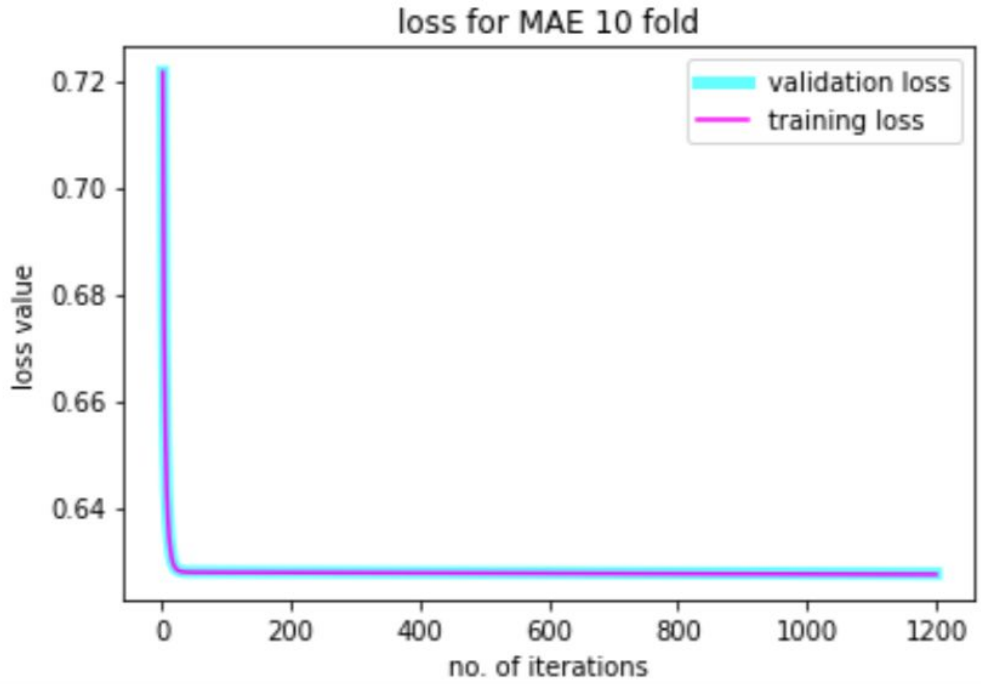
2) DATASET 1- RMSE (with k =10)

```
(4176, 8)
2.2239548827231954 2.2253609381711366
```



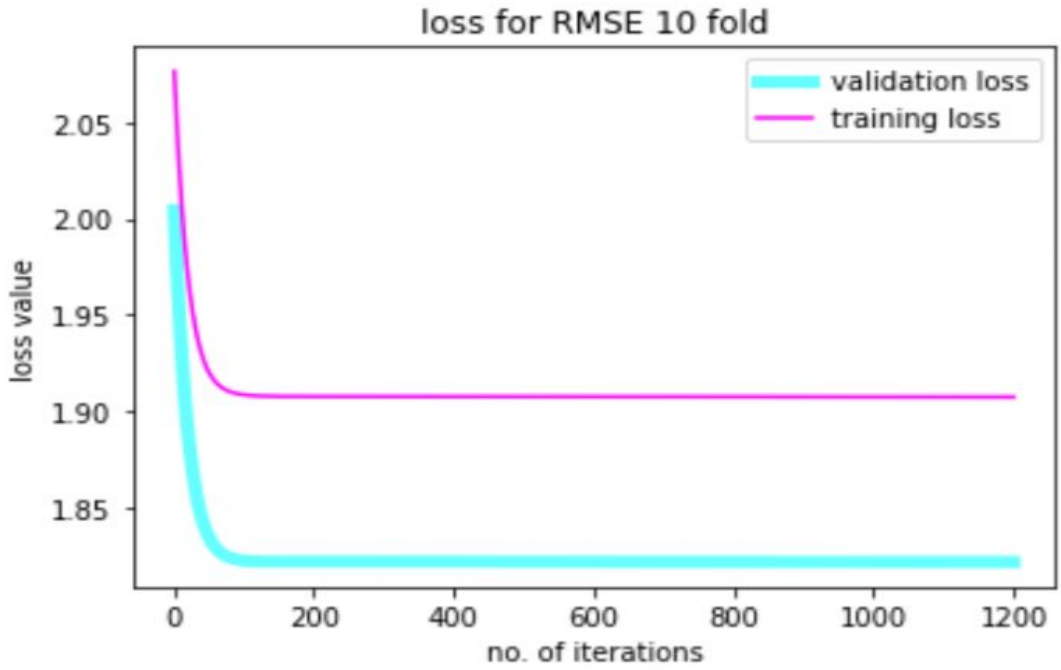
3) DATASET-2 MAE(with k=10)

```
0.6274828074382677 0.6275993111142142
```



4) **DATASET-2 RMSE(with k=10)**

```
(7017, 3)
1.9071805962720614 1.8213188450970985
```



(b) Include the best RMSE and MAE value achieved (as well as which fold achieves this) in your report.

The best MAE value for dataset 1

Training loss:- 1.5755
Validation loss:- 1.5808

The best MAE value for dataset 2

Training loss:- 0.6274
Validation loss:- 0.6275

The best RMSE value for dataset 1

Training loss:- 2.223
Validation loss:- 2.225

The best RMSE value for dataset 2

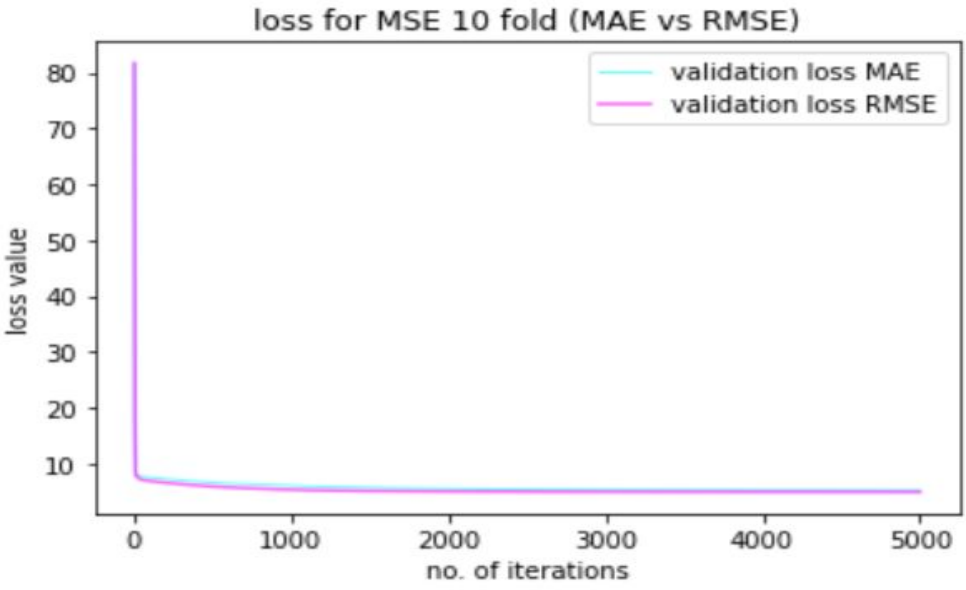
Training loss:- 1.9071
Validation loss:- 1.8213

(c) For each dataset, analyze and describe which of the losses leads to better performance.
(Hint: Compare the values of RMSE and MAE).

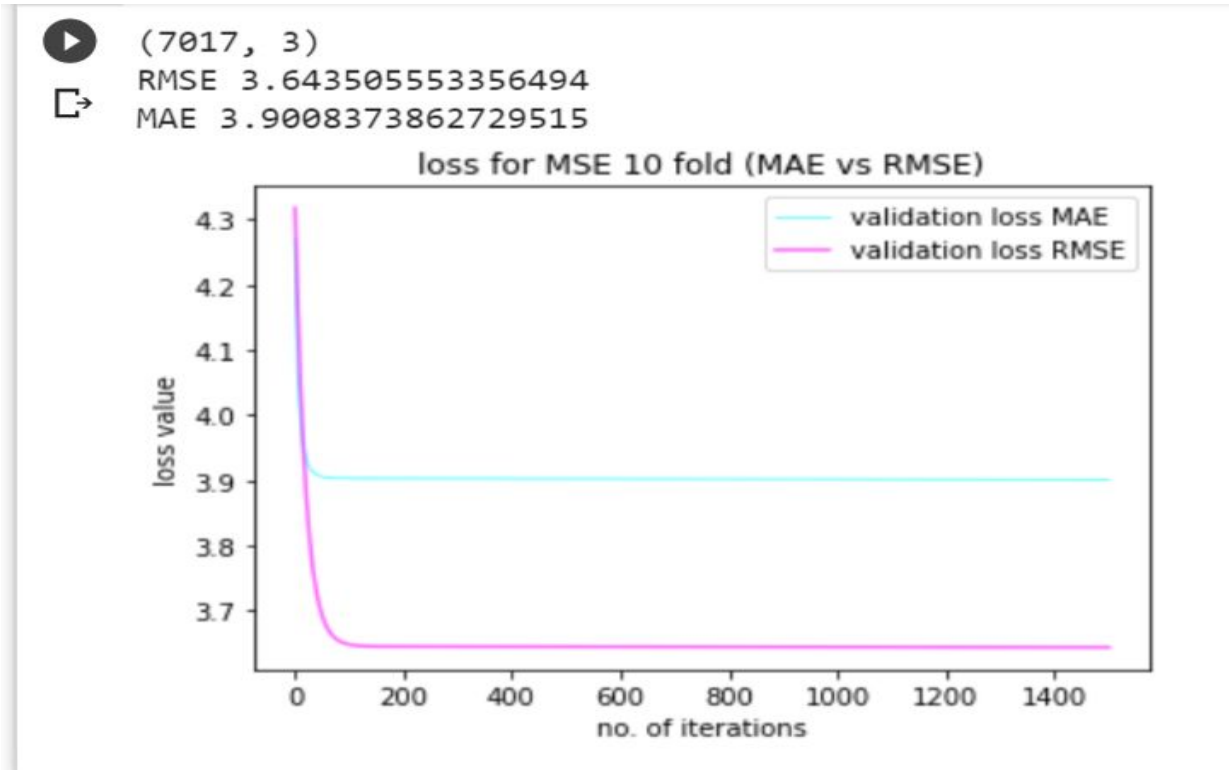
Ans: As can be seen from the graph below, and values present for validation losses (MSE used to compare the performance on two datasets), RMSE gives better performance or low testing error. It is due to the fact that RMSE is more sensitive to outliers than MAE thus RMSE keeps head to head with RMSE for these datasets because outliers are present, thus MAE converges faster than RMSE.

DATASET 1 (10 fold)

```
(4176, 8)
RMSE 4.946317282232901
MAE 5.275723272999478
```



DATASET 2 (10 fold)



(d) What is the relationship between MAE and RMSE?

When the user want to penalize outliers or higher magnitude error then we use RMSE.
Basically for a dataset RMSE is always greater than MAE.
[MAE] ≤ [RMSE]. The RMSE result will always be larger or equal to the MAE. If all of the errors have the same magnitude, then RMSE=MAE.
[RMSE] ≤ [MAE * sqrt(n)],

Under what conditions are RMSE and MAE expected to give similar values?
If the absolute error of all the rows are same, then RMSE cant penalize more thus at that time RMSE=MAE

Which loss will you prefer in such a case and why?

If there are outliers in the data that I want to penalize more, then RMSE is preferred more. RMSE is differentiable at every point rather than MAE and when the absolute value of errors are equal,it does make sense to use RMSE over MAE because function is easily calculated.

(e) Implement the normal equation form (closed form) of linear regression and get the optimal parameters directly. Consider the Dataset 1 and the most appropriate loss function you've described for this dataset in part(c). Compute the training and validation loss for the best fold for this loss described in part(b) using these optimal parameters.

k=10

```
training loss at fold:  1 5.004346915939697
validation loss at fold:  1 3.979457307051531
training loss at fold:  2 4.926985306316293
validation loss at fold:  2 4.67477893837322
training loss at fold:  3 4.9278407536668505
validation loss at fold:  3 4.675247321180135
training loss at fold:  4 4.8603676536940466
validation loss at fold:  4 5.282625168123754
training loss at fold:  5 4.9288122582991685
validation loss at fold:  5 4.663977945726463
training loss at fold:  6 4.865808372977023
validation loss at fold:  6 5.23685962214638
training loss at fold:  7 4.841207529580388
validation loss at fold:  7 5.464545071399419
training loss at fold:  8 4.869844962590583
validation loss at fold:  8 5.204598885976717
training loss at fold:  9 4.93086109913125
validation loss at fold:  9 4.6433860245125
training loss at fold:  10 4.802550362277975
validation loss at fold:  10 6.576749065854745
average: training:  4.895862521447327
validation loss 4.895862521447327

FOR RMSE
training loss at fold:  1 2.2370397662848323
validation loss at fold:  1 1.9948577159916772
training loss at fold:  2 2.219681352427932
validation loss at fold:  2 2.1621237102379736
training loss at fold:  3 2.219874040045257
validation loss at fold:  3 2.1622320229753638
training loss at fold:  4 2.2046241524790675
```

```
validation loss at fold:  4 2.2983962165222414
training loss at fold:  5 2.220092849026628
validation loss at fold:  5 2.159624491833352
training loss at fold:  6 2.2058577408747424
validation loss at fold:  6 2.288418585431079
training loss at fold:  7 2.200274421425743
validation loss at fold:  7 2.337636642294824
training loss at fold:  8 2.2067725217136864
validation loss at fold:  8 2.2813589998018102
training loss at fold:  9 2.220554232422899
validation loss at fold:  9 2.1548517407266097
training loss at fold:  10 2.1914721906239136
validation loss at fold:  10 2.5645173163491695
average: training:  2.21262432673247 validation loss 2.21262432673247
```

• Analyze the class distributions and comment on the feature values for each of the given features. (5 points)

d.info():
It gave us a number of features, their types and null counts with memory usage.

We can see from here that there are no null counts thus don't have to remove any data from the existing one. All types are floats thus type conversion and calculations can easy take place. No need to look for assigning any value or removing string features.

Memory is 53.7kb so time to load is bare minimum.

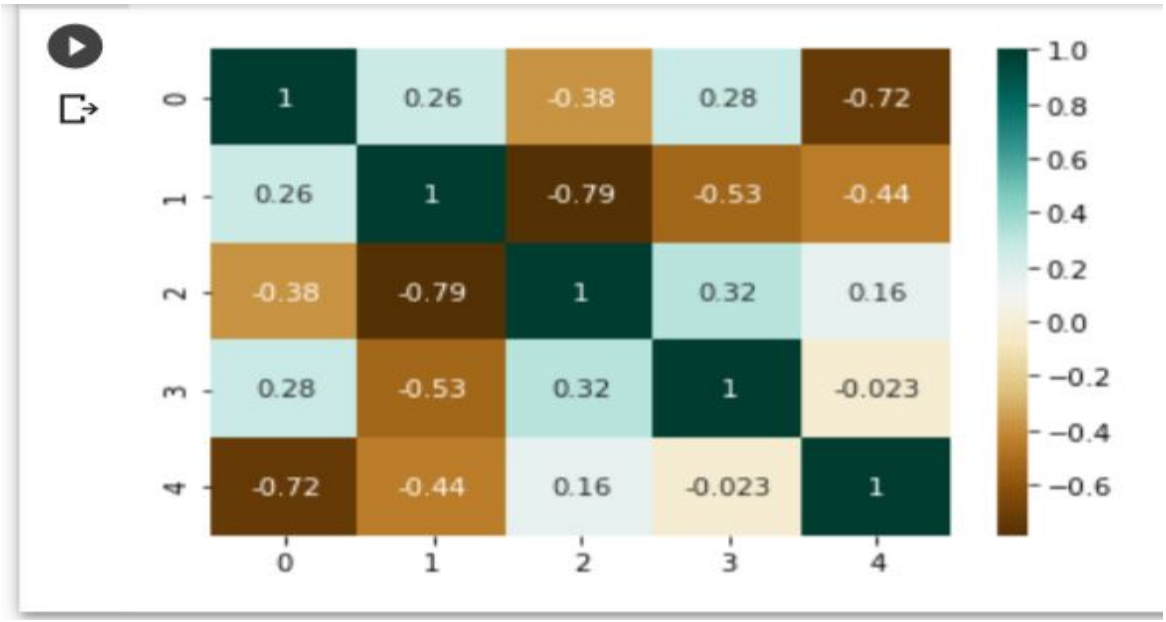
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1372 entries, 0 to 1371
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   0       1372 non-null   float64
 1   1       1372 non-null   float64
 2   2       1372 non-null   float64
 3   3       1372 non-null   float64
 4   4       1372 non-null   int64  
dtypes: float64(4), int64(1)
memory usage: 53.7 KB
```

d.describe(): it shows mean, max variance, standard deviation in order to show the functuations between data by dividing data into 4 parts and seeing if data is biased or not. We

can see that mean fluctuates a lot so large chunks of data are together thus shuffling will be required to get it balanced. Deviation of whole data is less so it is a good data pile.

	0	1	2	3	4
count	1372.000000	1372.000000	1372.000000	1372.000000	1372.000000
mean	0.433735	1.922353	1.397627	-1.191657	0.444606
std	2.842763	5.869047	4.310030	2.101013	0.497103
min	-7.042100	-13.773100	-5.286100	-8.548200	0.000000
25%	-1.773000	-1.708200	-1.574975	-2.413450	0.000000
50%	0.496180	2.319650	0.616630	-0.586650	0.000000
75%	2.821475	6.814625	3.179250	0.394810	1.000000
max	6.824800	12.951600	17.927400	2.449500	1.000000

d.corr(): finds correlation between attributes thus higher correlation means data is not good.



Perform a train:val:test split in the ratio 7:1:2 and implement Logistic Regression based on the given template functions. (5x4 = 20 points)

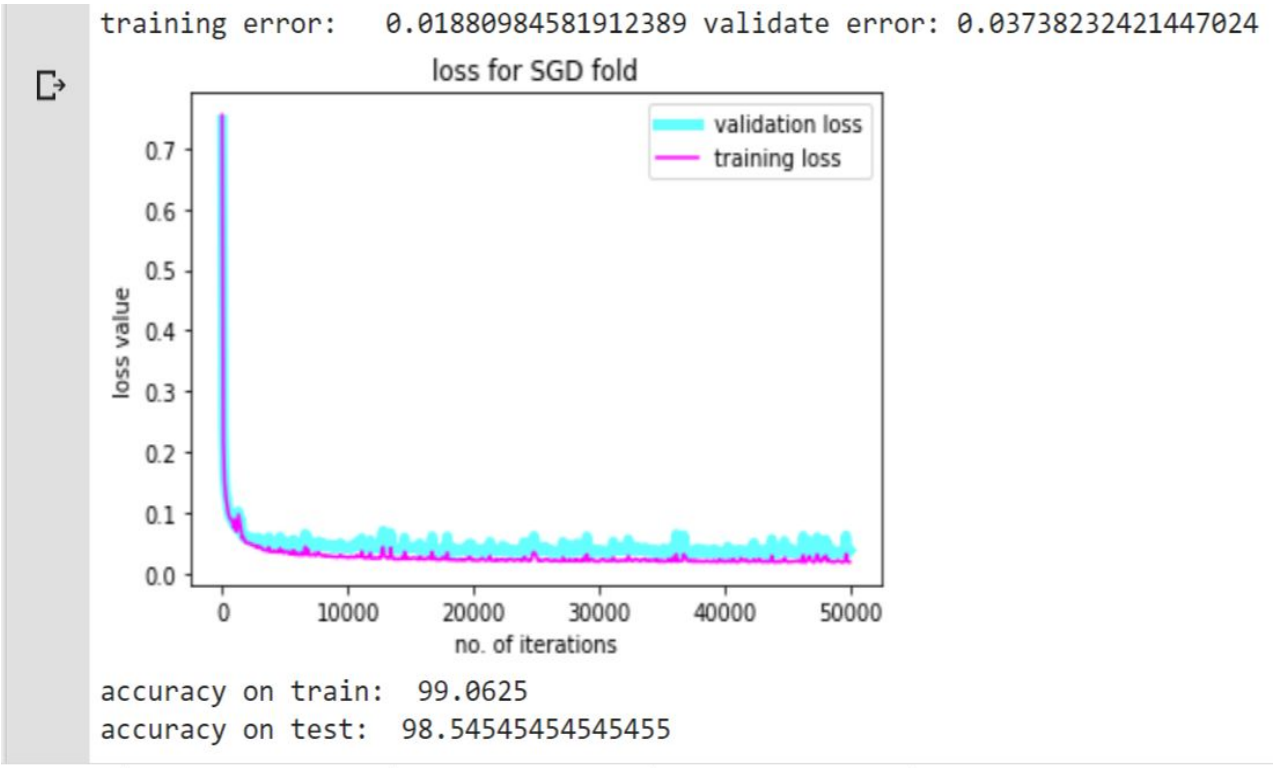
(a) Using Stochastic Gradient Descent (SGD), choose an appropriate learning rate and the number of epochs (iterations). Report the accuracy obtained on both the training and test set.

Learning rate: 0.02
Iterations: 50000

Accuracy on training: 99.0625

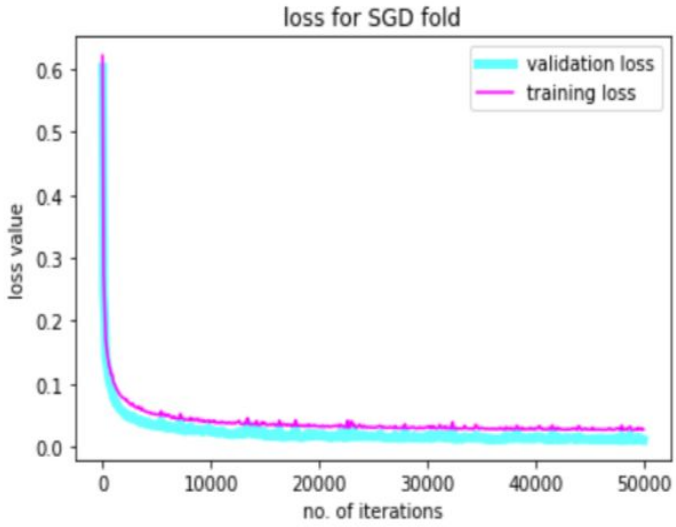
Accuracy on testing: 98.54

(b) Include plots between training loss v/s iterations and validation loss vs iterations.



(c) Re-run your implementation for 3 variations in learning rates - 0.0001, 0.01, 10
Alpha 0.01

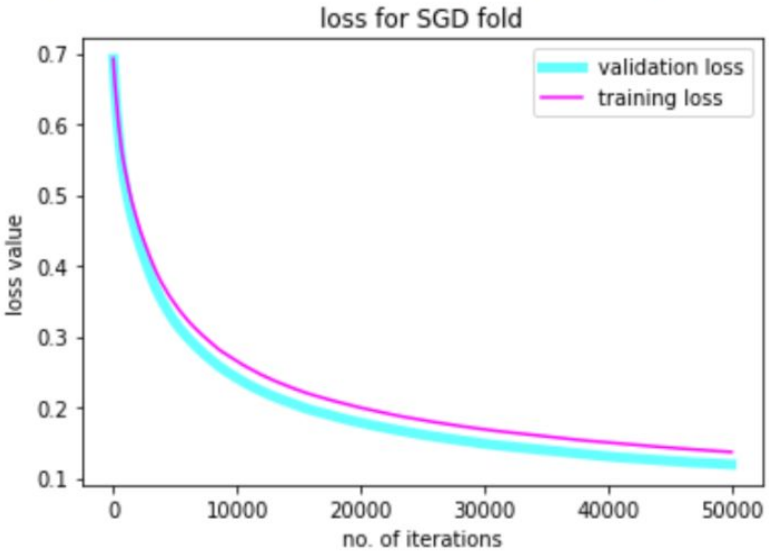
```
training error:   0.02721960674324763 validate error: 0.01029814989639379
```




```
accuracy on train:  98.64583333333333
accuracy on test:  99.27272727272727
```

Alpha 0.0001

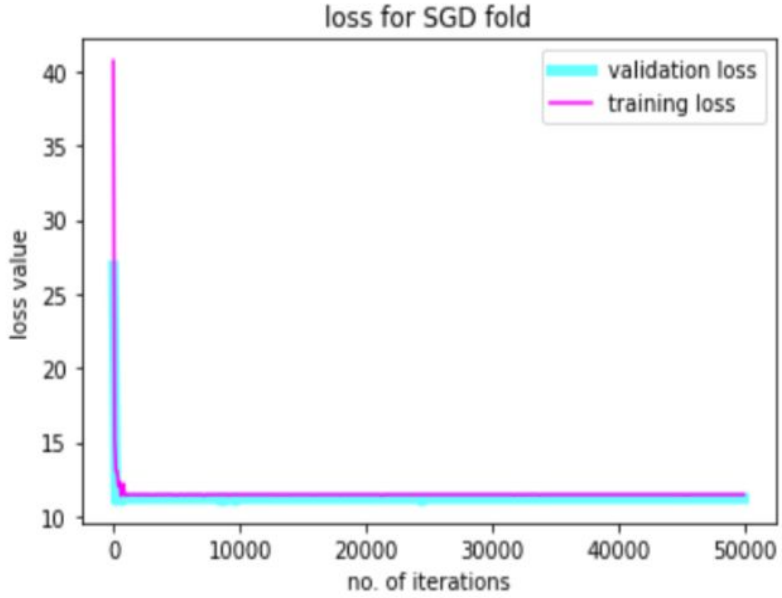
```
training error:   0.1370706138845528 validate error: 0.11947480863902707
```




```
accuracy on train:  96.66666666666667
accuracy on test:  94.9090909090909
```

Alpha 10

```
training error:   11.462556432151409 validate error: 11.193588861970037
```



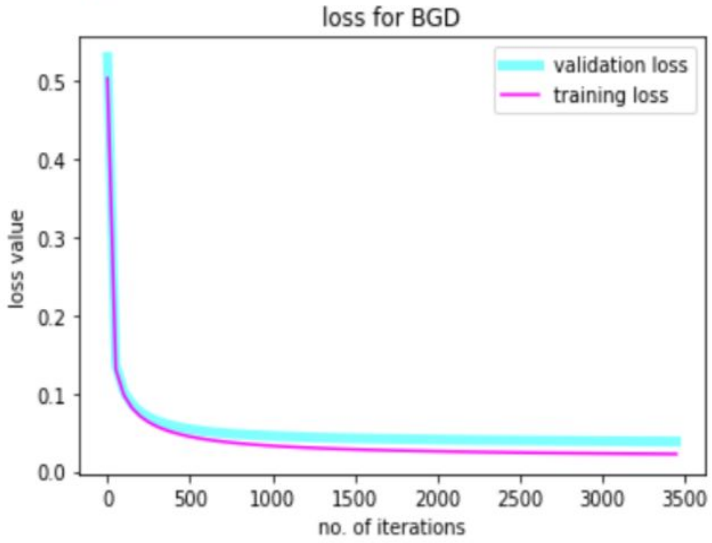

```
accuracy on train:  44.6875
accuracy on test:  42.90909090909091
```

(d) Now, implement Batch Gradient Descent (BGD) and re-run (a), (b), (c) with BGD.

iterations : 3500
Alpha: 0.1

Accuracy on training: 0.0223
Accuracy on testing: 98.909

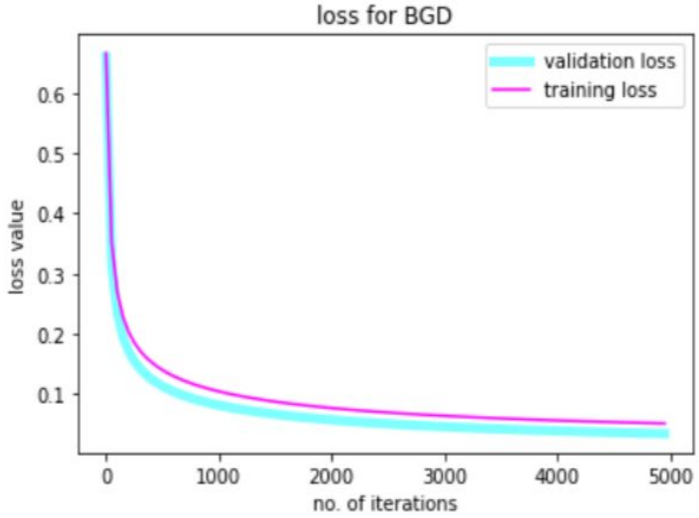
```
training error:   0.022382664960099535 validate error: 0.03848263390765887
```




```
accuracy on train:  99.16666666666667
accuracy on test:  98.9090909090909
```

Alpha 0.01

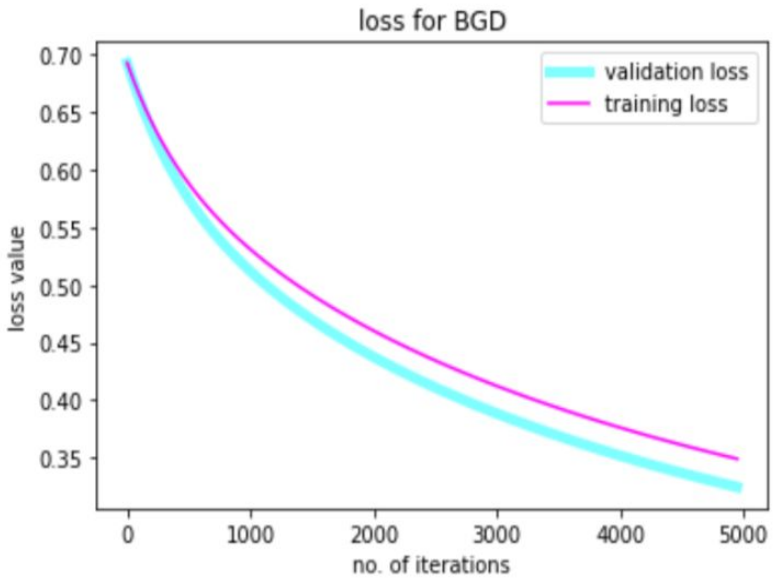
```
training error:   0.05100215471945246 validate error: 0.03404688905055328
```




```
accuracy on train:  98.54166666666667
accuracy on test:  98.9090909090909
```

Alpha 0.0001

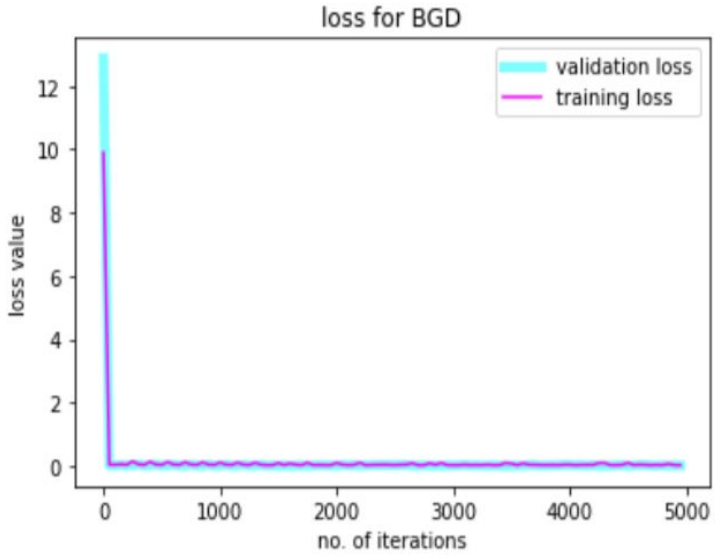
```
training error:   0.3489905403239952 validate error: 0.32454512286550335
```




```
accuracy on train:  90.41666666666667
accuracy on test:  88.0
```

Alpha 10

```
training error:   0.020170117935079175 validate error: 0.006966964864271474
```




```
accuracy on train:  99.16666666666667
accuracy on test:  98.9090909090909
```

Compare the performance of BGD and SGD, w.r.t the following observations: (2.5x4 = 10 points)
(a) Loss plots

The loss for SGD and BGD are nearly the same (0.021) value, but you can see that the SGD converges more quickly than BGD due to the slope. But SGD's graph fluctuates a lot (loss increases and decreases while BGD curve is smooth) so SGD can give us incorrect answers for some iterations, and some different loss value for another but if BGD is close to converge. Change is very minute for different thetas in different iterations.

(b) Number of epochs taken to converge.

Epocs taken by SGD seems to be more than BGD but in reality if we see the backend of both, we see that BGD's each epoc do n (no. of samples) time more work than SGD. thus to compare them fully, iterations taken by BGD = (n)*3500 < 50000
Thus SGD is faster.

(c) Use sklearn's LogisticRegression implementation on the same dataset above.

```
X=X_train[:,1:]
from sklearn.linear_model import LogisticRegression
model=LogisticRegression(solver='liblinear')
model.fit(X,Y.ravel())
t0=model.intercept_
tall=model.coef_
print(t0,tall)
b= np.concatenate([t0,tall[0]])
print("accuracy on train: " ,accuracy_logtsic(b,X_train,Y_train))
print("accuracy on test: " ,accuracy_logtsic(b,X_test,Y_test))
```

```
[2.95088487] [[-2.81380062 -1.53516039 -1.89757225 -0.13047016]]
accuracy on train:  98.85416666666667
accuracy on test:  98.9090909090909
```

(d) Report the accuracy obtained on both the training and test set. Use the same hyper-parameters as in the SGD implementation in 2(a), and compare sklearn performance with that.

```
[2.95088487] [[-2.81380062 -1.53516039 -1.89757225 -0.13047016]]
accuracy on train:  98.85416666666667
accuracy on test:  98.9090909090909
```

Same as SGD.

3. (10 points) You have a logistic regression model and you are using mean squared error
loss along with it. Assume you have a datapoint for which the model produces really wrong results (eg : y true = 1 and y pred is approaching 0 or vice versa). For this datapoint, show that the gradient calculated during gradient descent would approach 0. What are the implications of this, would the model be able to learn effectively? What would happen if we use cross entropy loss instead?

Q3. We are using MSE loss

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x) - y^i)^2$$

11 WED

gradient for logistic

$$\frac{\delta J(\theta)}{\delta \theta_j} = \frac{1}{2m} \sum_{i=1}^{n} 2 (h_\theta(x^i) - y^i) \cdot (h_\theta x_i - y_i)'$$

$$= \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y^{(i)}) \left(\frac{d\, h_\theta(x_i)}{d\theta_j} \right)$$

$$= \frac{dJ(\theta)}{d\theta_j} = \frac{1}{m} \sum_{i=1}^{n} (h_\theta(x_i) - y_i) \cdot \frac{e^{-\theta_i x_i)}}{(1 + e^{-\theta i x_i})^2}$$

12 THU

Case 1 ~~y-true~~ y_actual = 1 & y_pred = 0.

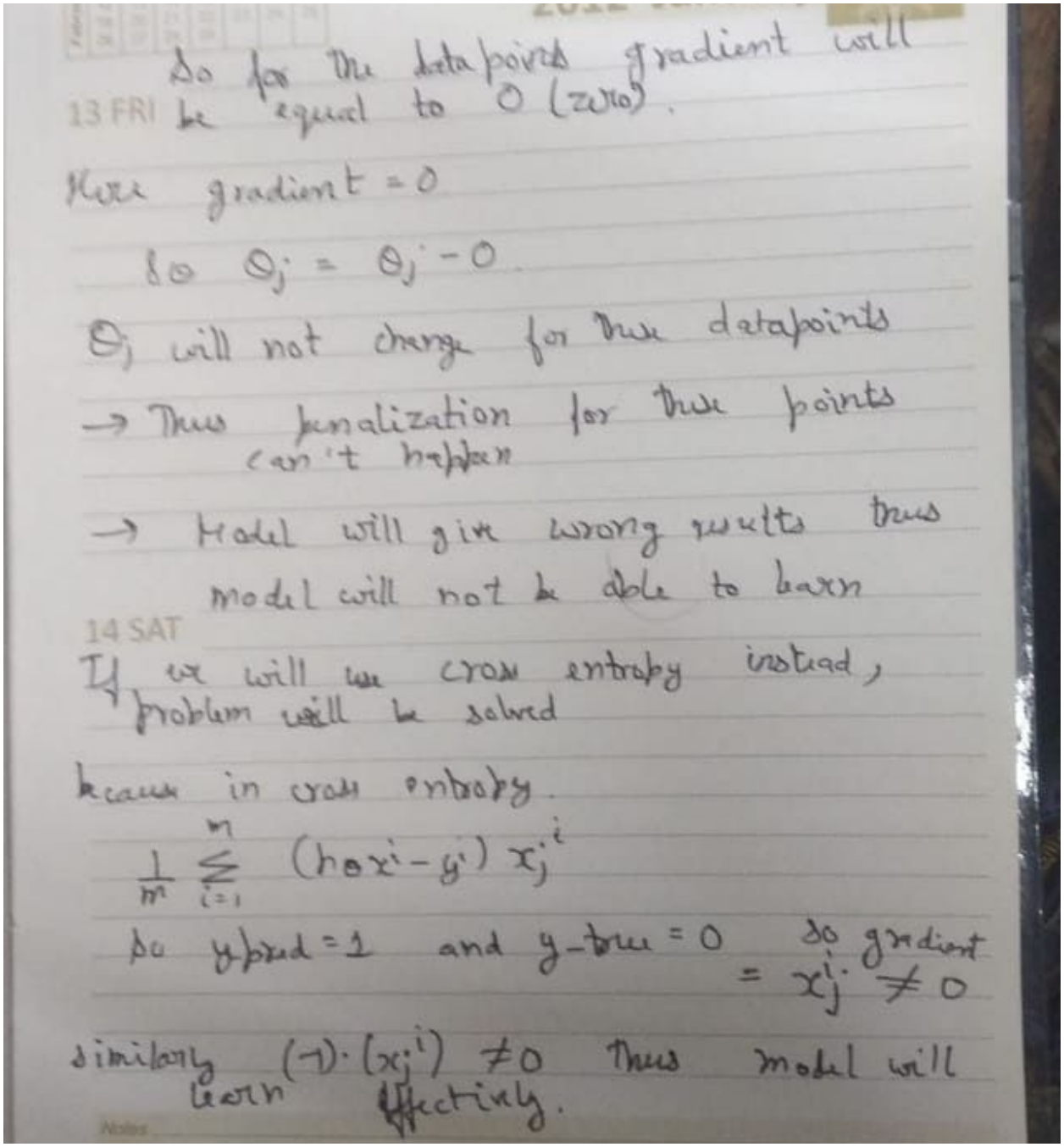
$y\_pred = 0 \quad h_\theta(x^i) = 0 \quad \frac{1}{1 + e^{-\theta^T x}} = 0.$

$gradient = (0-1) \cdot (0) \cdot (1-0)$

$= 0$

Case 2 $y\_true = 0 \quad y\_pred = 1.$

$(1-0)(1)(1-1) = 0.$



4. (2.5x4 = 10 points) A cancer research institute was conducting a study on the recurrence of neuroblastoma in children. They were examining the feasibility of using logistic regression to find out the likelihood that a patient will have disease recurrence in the next 5 years. A random sample of 33 patients was selected. The data collected from the patient consists of the extent to which the spread of the disease has occurred (X1, in percentage) and the age of the patient (X2, in years). The pilot study was followed by a revaluation of the patient condition after 5 years which was used to determine whether the disease recurred (Y = 1) or didn't recur (Y = 0) in 5 years. The data pertaining to the patients download:

- **Estimate β0, β1, and β2 using MLE.**
- **State the fitted response function.**
- **Obtain exp(β1) and exp(β2) and interpret these numbers.**
- **What is the estimated probability that a patient with 75% of disease spread and an age of 2 years will have a recurrence of disease in the next 5 years?**

```
In [22]: 1 logistic.theta_vector

Out[22]: array([-4.28312333,  0.06246534,  0.52165424])
```

Q4 1) Calculating Theta or 15 SUN
coefficients from logistic regression

So getting :-

$\beta_0 = -4.2831$

$\beta_1 = 0.062$

$\beta_2 = 0.5216$

16 MON

2) Fitted response function

$$= h(\theta)_x = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2)}}$$

$$h(\beta x) = \frac{1}{1 + e^{-(-4.2831 + 0.062 x_1 + 0.5216 x_2)}}$$

17 TUE 3) $\exp(\beta_1) = e^{0.0624} = 1.0643$

exp(B) is the effect of the independent variable on the odds ratio, Thus

$= \exp(\beta_2) \Rightarrow e^{0.521} = 1.6837$

if change in x_1 or years is 1, then will be a chance of 51% to occur

if change in x_2 is 1, then there will be 0.627 chance for that event to occur

18 WED 4) patient with 75% disease

so $x_1 = 75.0$

years 2

$x_2 = 2.0$

probability $h\theta(x) = \frac{1}{1 + e^{-(-4.283 + 0.0624 \times 75 + 2 \times 0.521)}}$

$= 0.809$

0.81 is probability for occurring of disease

Q5.Write the matrix expression for the sum of squared errors loss function. Derive an expression to find the β
that minimizes this loss for the above linear regression problem.
In order to derive the least squares solution, you would need to differentiate the matrix form directly. Once you have the solution expression, write the conditions under which the solution (in the matrix form) will exist.