Machine Learning
Assignment - 1

Ashwin Sheoran
2020288

Section-A

Ans1)
(a)

Ans 1)

We have $y_i = w^T x_i + \varepsilon$, for linear regression

We can take summation of i from 0 to n.

$$\sum_{i=0}^{n} y_i = \sum_{i=0}^{n} w^T x_i + \sum_{i=0}^{n} \varepsilon$$

ε is constant

$$\sum_{i=0}^{n} y_i = \sum_{i=0}^{n} w^T x_i + n\varepsilon$$

Divide the eqn by n.

$$\sum_{i=0}^{n} \frac{y_i}{n} = \sum_{i=0}^{n} \frac{w^T x_i}{n} + \frac{n\varepsilon}{n}$$

$$\bar{y} = w^T \bar{x} + \varepsilon \qquad \left[\bar{y} = \sum_{i=0}^{n} \frac{y_i}{n},\ \bar{x} = \sum_{i=0}^{n} \frac{x_i}{n}\right]$$

∴ Therefore least square fit line pass through $(\bar{x}, \bar{y})$

→ Reference Taken from lecture Notes.

Spiral

(b)
If two variables have a high correlation with a third variable, Then necessarily, they will not be correlated.
Let us say that A and B are both correlated with C, and then A and B necessarily won't be correlated.

Let us take an example.
Let A be the time for which the fruits are kept in the refrigerator
Let B be the temperature inside the refrigerator
Let C be the number of fruits spoiled after being kept in the refrigerator for time A.

Now, if we change the time A for which the fruits are kept in the refrigerator, then C will change (Less time means fewer fruits will be spoiled and vice versa)
Also, if we change the temperature inside the refrigerator, C will change (More temperature means more fruits spoiled)

So A and C are related, and B and C are.

But the temperature inside the refrigerator and the time for which fruits are kept in the refrigerator are not related.
Therefore A and B are not related.

Thus if two variables have a high correlation with a third
Variable, they will not necessarily be highly correlated.

(c)

Date .....

Ans 2) Weak Law of Large Numbers states that for Sequence of i.i.d random Variables for which $\mu = E[X]$ and $S_n = X_1 + X_2 \cdots X_n$

$$\lim_{n \to \infty} P\left[\left|\frac{S_n}{n} - \mu\right| \geq \varepsilon\right] = 0$$

should be true.

Basically it says the sample increases, the sample average $\bar{X}_n = \frac{S_n}{n}$ should converge to μ.

Let $Var(X) = \sigma^2$

Using Chebyshev Inequality.

$$P\left[|\bar{X}_n - \mu| \geq \varepsilon\right] \leq \frac{Var(\bar{X}_n)}{\varepsilon^2}$$

$$Var(\bar{X}_n) = Var\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) = \frac{Var}{n^2}\left(\sum_{i=1}^{n} X_i\right)$$

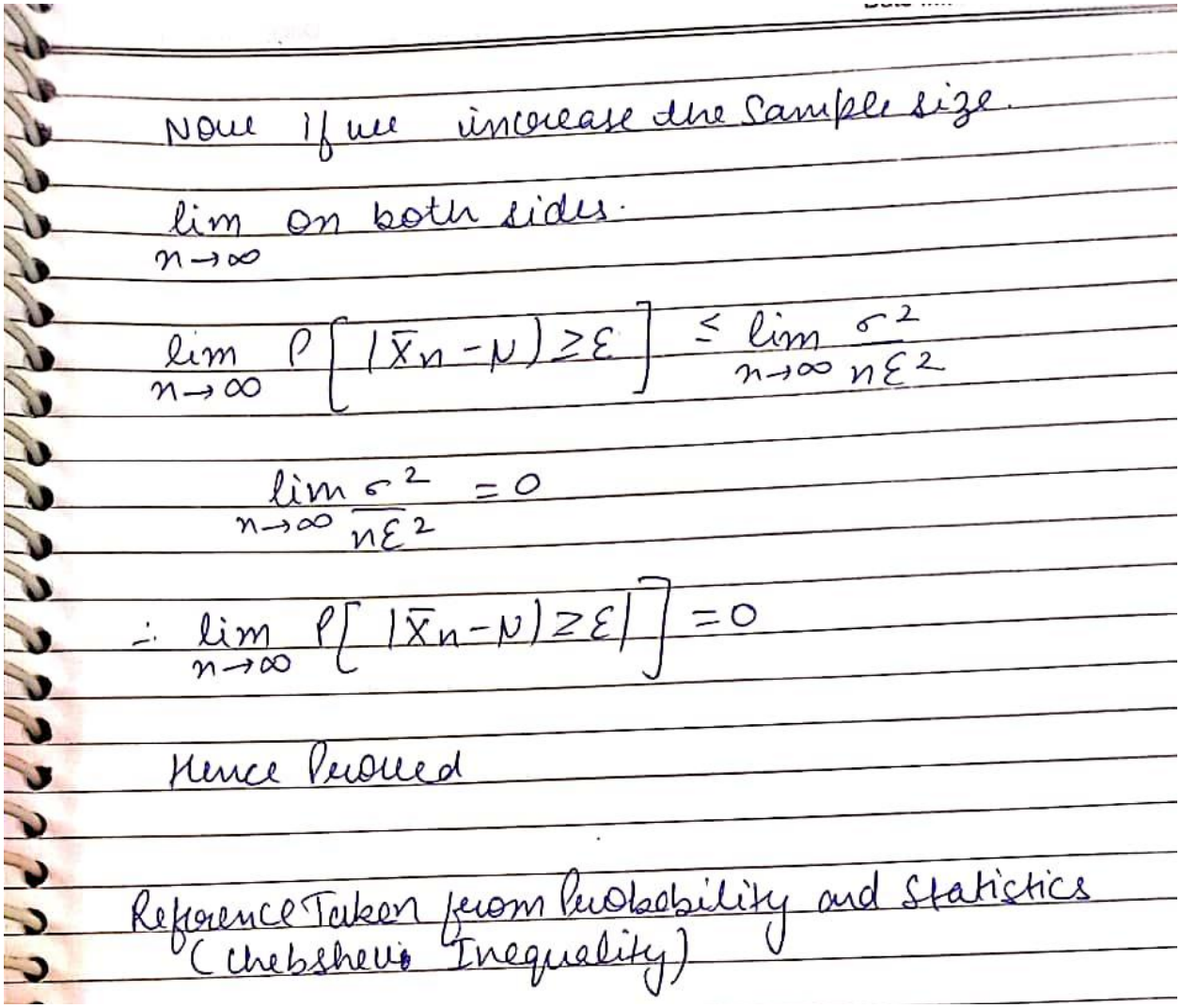
Since X_i are i.i.d.

$$Var(\bar{X}_n) = \frac{1}{n^2}\sum_{i=1}^{n} Var(X)$$

$$= \frac{1}{n^2} \times n\sigma^2 = \frac{\sigma^2}{n}$$

$$\therefore P\left[|\bar{X}_n - \mu) \geq \varepsilon\right] \leq \frac{\sigma^2}{n\varepsilon^2}$$

Spiral




```
### Weak law of Large Numbers Pseudo Code for a discrete random variable

err_lst=[]

for i in range (1000 , 100000 , 500):  ## Increasing the Sample size from 1000 to 100000 by 500 in every iteration
    sum = 0
    for j in range (i):
        sum = sum+ j
    mean = sum / i    ## finding mean of sample size
    pred = predicted_value    ## Finding Predicted value
    err = (predicted_value - mean)  ## Finding the Error
    err_lst.append(err)
print(err)
```

We will Observe that with increasing Sample size the error decreases

(d)

Date .....

Ans 4)

We have to derive MAP solution.

we have

$$P(w|D) = P(D|w) * P(w) / P(D)$$

$$P(w|D) = \text{argmax} \prod_{i=1}^{n} \left[P(y_i|x_i;w) * \frac{P(y_i)}{P(x_i|w)} \right]$$

$$= \text{argmax} \sum_{i=1}^{n} \log \left[P(y_i|x_i;w) * \frac{P(y_i)}{P(x_i;w)} \right]$$

$$= \text{argmax} \sum_{i=1}^{n} \log \left[P(y_i|x_i;w) \right] + \log \left[P(y_i) \right] - \log \left[P(x_i;w) \right]$$

Now $\log(P(y_i))$ is independent of w, so we can ignore it.

$$= \text{argmax} \sum_{i=1}^{n} \log \left[P(y_i|x_i;w) \right] - \sum_{i=1}^{n} \log \left[P(x_i;w) \right]$$

$$= \text{argmax} \sum_{i=1}^{n} \left(\log(1/\sqrt{2\pi\sigma^2}) - \left(\frac{(w^T x_i - y_i)^2}{2\sigma^2} \right) \right) - \sum_{i=1}^{n} \log(P(x_i;w))$$

Spiral

Date .....

Here $\left(\log\left(1/\sqrt{2\pi\sigma^2}\right)\right)$ is independent of w, so we can ignore it

$$P(w|D) = \text{argmax}\left\{\sum_{i=1}^{n}\left(-\frac{(w^T x_i - y_i)^2}{2\sigma^2}\right) - \sum_{i=1}^{n}\log(P(x_i;w))\right\}$$

$$= \text{argmin}\left[\sum_{i=1}^{n}\left(\frac{w^T x_i - y_i)^2}{2\sigma^2}\right) + \sum_{i=1}^{n}\log(P(x_i;w))\right]$$

Now $1/2\sigma^2$ is a constant

$$= \text{argmin}\left\{\sum_{i=1}^{n}(w^T x_i - y_i)^2 + \sum_{i=1}^{n}\log(P(x_i;w))\right\}$$

Now let us assume that $P(x_i;w)$ is I.I.D

∴ we get

$$= \text{argmin}\left\{\sum_{i=1}^{n}(w^T x_i - y_i)^2 + n\log(P(x;w))\right\}$$

$$= \text{argmin}\left(\sum_{i=1}^{n}(w^T x_i - y_i)^2 + n\log\frac{w^T \cdot w}{\sigma_0^2}\right.$$

Reference Taken from Lecture slides

Spiral

Section - B

Ans 2)

a)

I ran the model for K = 2,3,4,5.

```
The MEAN RMSE error for K-fold when K is 2 is 9.292982238230703
The MEAN RMSE error for K-fold when K is 3 is 9.245301562507313
The MEAN RMSE error for K-fold when K is 4 is 9.160991205002354
The MEAN RMSE error for K-fold when K is 5 is 9.120223543331463
```

I am getting the lowest RMSE for K = 5.

5 will be the optimal value for K.

I have taken 100000 epochs and 0.0000003 as the learning rate as these parameters were given the lowest RMSE by hit and trial.

<u>Concepts Used</u>

Gradient descent

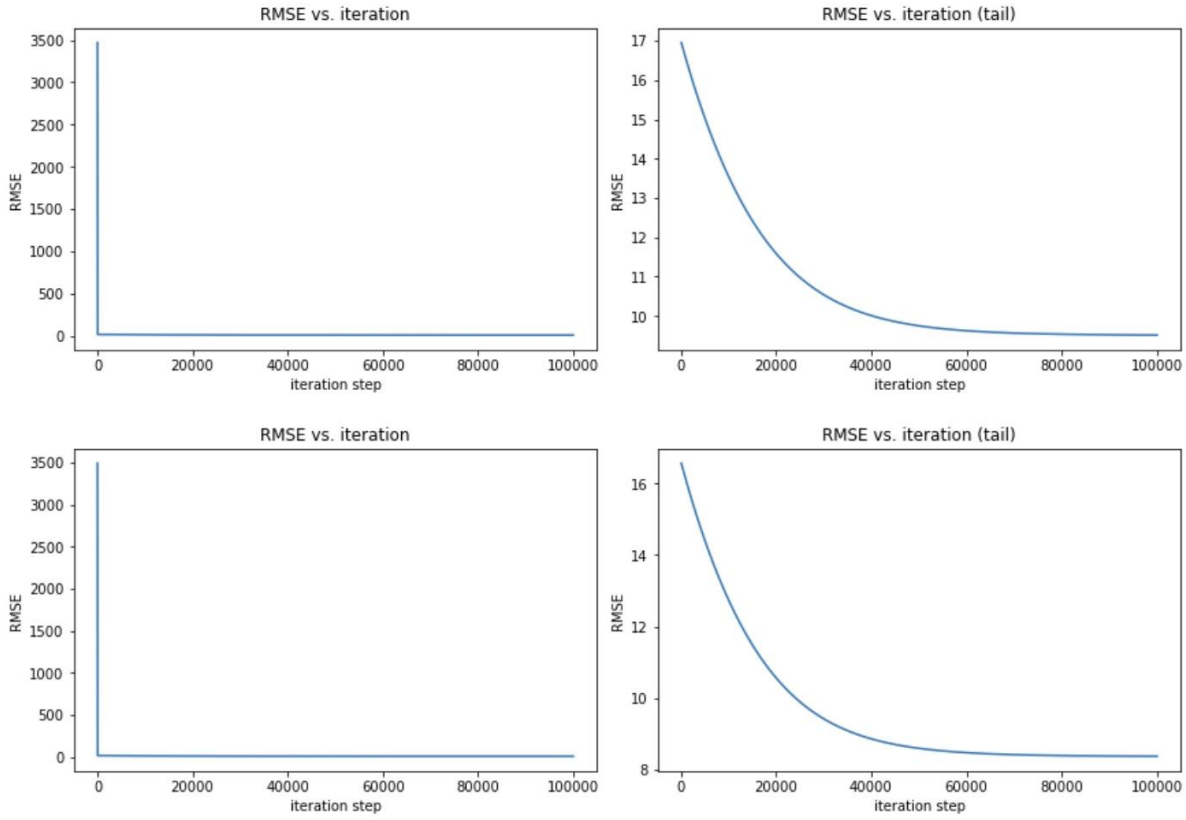
$$w_i = w_i - \eta \frac{\partial J(w)}{\partial w_i}$$

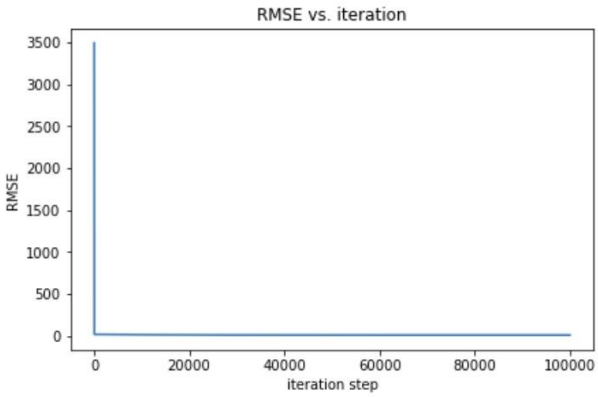
```
def model(X, Y, learning_rate, epochs):

    theta = np.ones((X.shape[1], 1))


    for i in range(epochs):
        y_pred = np.dot(X, theta)
        del_theta = (1/Y.size)*np.dot(X.T, y_pred - Y)   # d_theta for gradient descent
        theta = theta - learning_rate*del_theta   # Lecture-3 , 11th Slide

    return theta

epochs = 100000
learning_rate = 0.0000003

def check_error(X_test , Y_test , theta):

    y_pred = np.dot(X_test, theta)

    print("RMSE Error (validation loss) is ")
    print((np.sum(((abs(Y_test - y_pred))**2))/Y_test.shape[0])**0.5)
    return ((np.sum(((abs(Y_test - y_pred))**2))/Y_test.shape[0])**0.5)
```

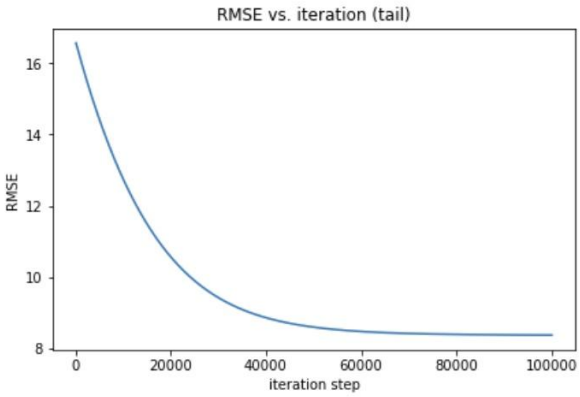
We are using del_theta by differentiating theta and then changing it according to gradient descent and then using the learning rate, and then using the theta; we finally got after all the epochs.
Then we computer RMSE and print it.

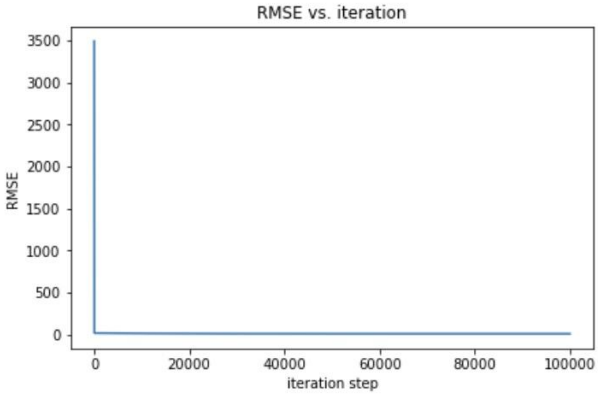
Reference Taken from Lecture Slides

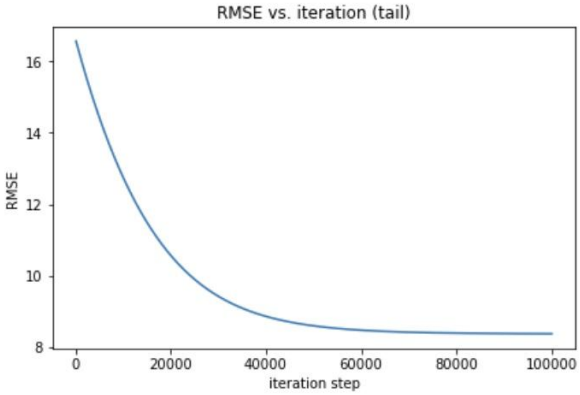
b)
Different RMSE vs Iterations Graph for K=5, on the train and value set

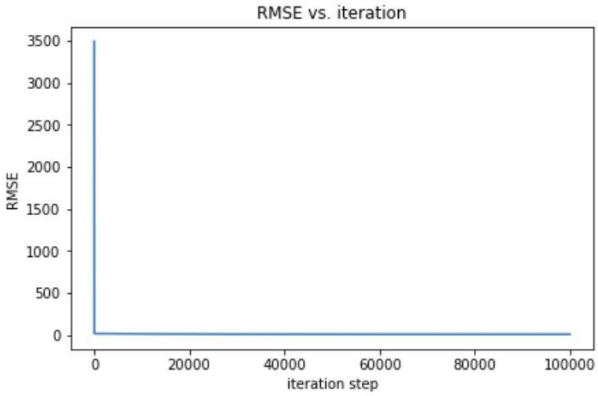


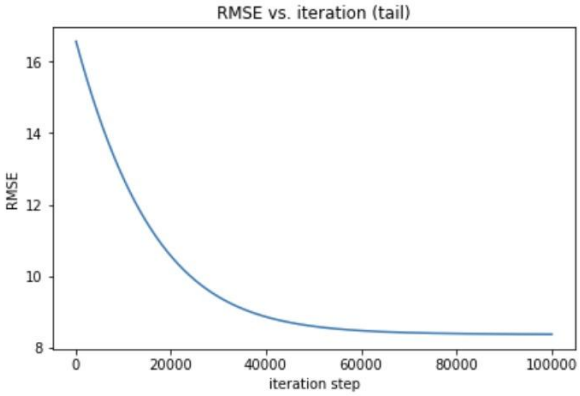

RMSE vs. iteration
RMSE
iteration step
0
500
1000
1500
2000
2500
3000
3500
0
20000
40000
60000
80000
100000


RMSE vs. iteration (tail)
RMSE
iteration step
8
10
12
14
16
0
20000
40000
60000
80000
100000


RMSE vs. iteration
RMSE
iteration step
0
500
1000
1500
2000
2500
3000
3500
0
20000
40000
60000
80000
100000


RMSE vs. iteration (tail)
RMSE
iteration step
8
10
12
14
16
0
20000
40000
60000
80000
100000


RMSE vs. iteration
RMSE
iteration step
0
500
1000
1500
2000
2500
3000
3500
0
20000
40000
60000
80000
100000


RMSE vs. iteration (tail)
RMSE
iteration step
8
10
12
14
16
0
20000
40000
60000
80000
100000

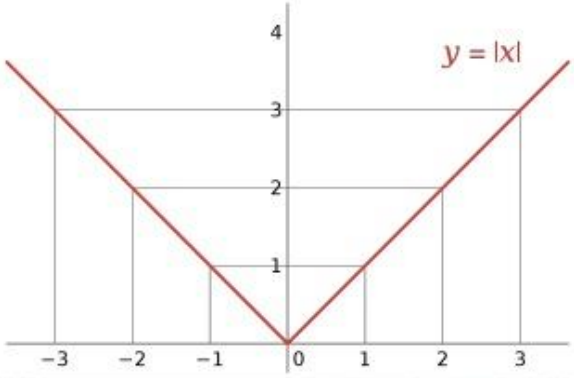
C) Modifying regression using L1(LASSO) and L2(Ridge Regression)
L1(Lasso Regression)

The RMSE is coming 9.654644061943909 for Lasso regularization

Concepts Used

Minimization objective = LS Obj + λ* (sum of absolute value of slope)

$$J(\Theta) = \frac{1}{M}\sum_{i=1}^{M}(\Theta^T\Phi(x_i)-y_i)^2+\lambda\sum_{j=1}^{p}|\Theta_j|$$

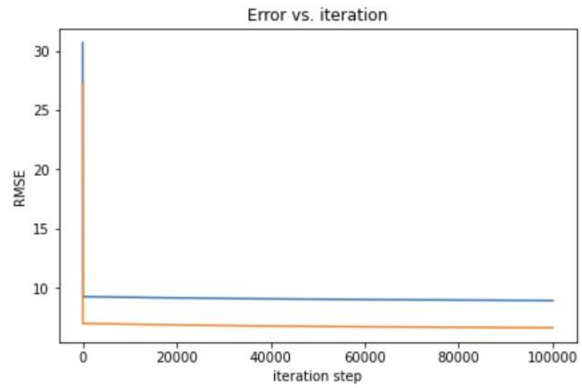


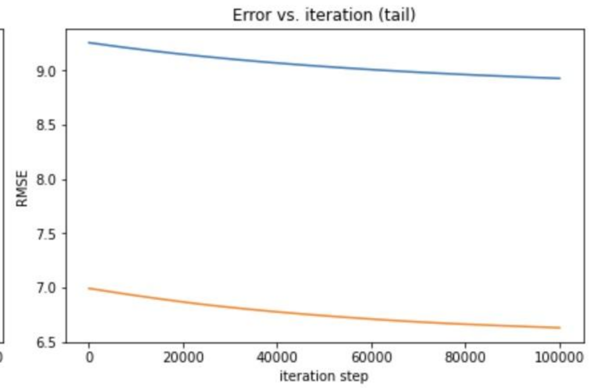
Reference taken from Lecture Notes

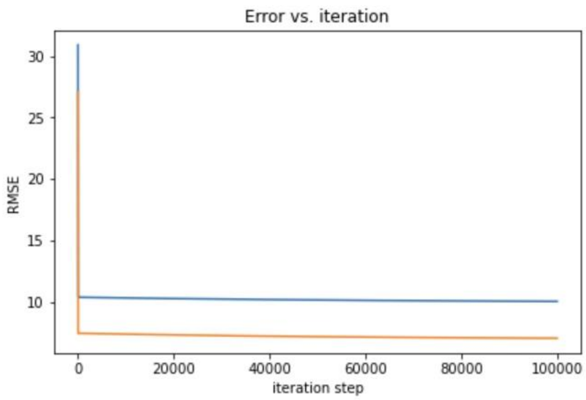
```
def Lasso(X, Y, learning_rate, epochs):

    W = np.zeros((X.shape[1], 1))
    b= 3
    alpha = 0.00005

    for i in range(epochs):
        y_pred = np.dot(X, W)
        LW = (-(2*(X.T).dot(Y - y_pred)) + (alpha))  /X.shape[0]  # Lecture-6 Slide 20
        W -= learning_rate*LW

    return W

epochs = 100000
learning_rate = 0.00000003

def check_errorl1(X_test , Y_test , theta ):

    y_pred = np.dot(X_test, theta)


    print("Training Error is ")
    print( (np.sum(((abs(Y_test - y_pred))))/Y_test.shape[0]) )


    print("RMSE Error (validation loss) is ")
    print((np.sum(((abs(Y_test - y_pred))**2))/Y_test.shape[0])**0.5)

    return ((np.sum(((abs(Y_test - y_pred))**2))/Y_test.shape[0])**0.5)
```

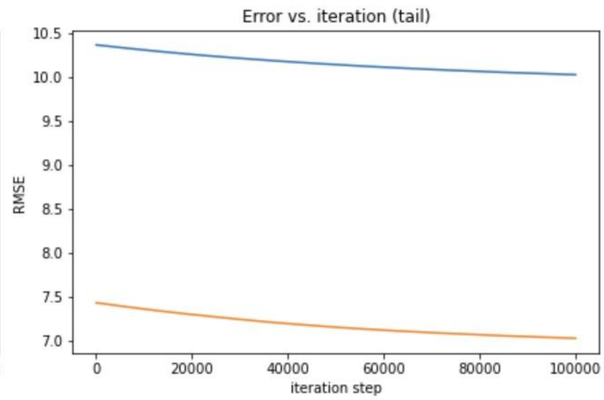
Here we are changing the theta (W) using the parameters according to the Lasso regularisation and then changing it using the learning rate, and then finally using it after running all the epochs. Then we calculate the training and the validation error.

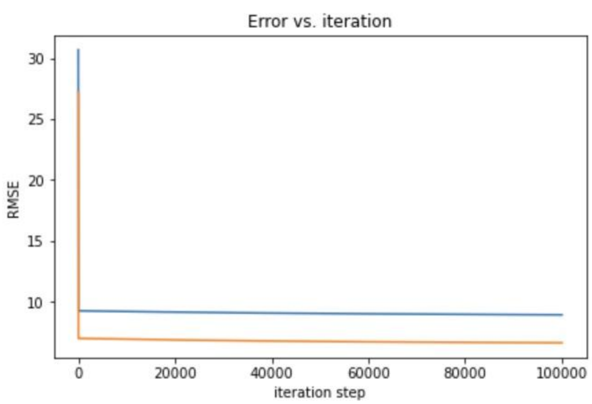
<u>Graphs</u>

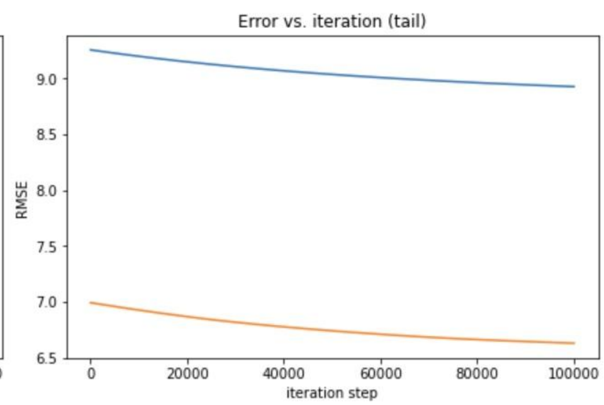

Error vs. iteration
RMSE
30
25
20
15
10
0
20000
40000
60000
80000
100000
iteration step

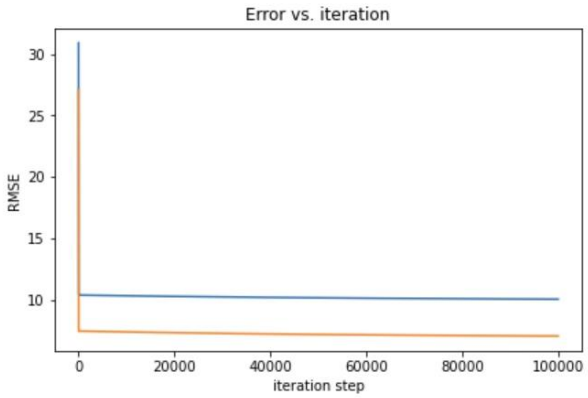

Error vs. iteration (tail)
RMSE
9.0
8.5
8.0
7.5
7.0
6.5
0
20000
40000
60000
80000
100000
iteration step

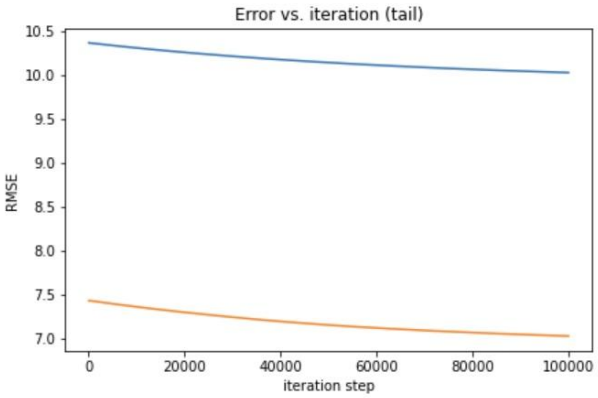

Error vs. iteration
RMSE
30
25
20
15
10
0
20000
40000
60000
80000
100000
iteration step

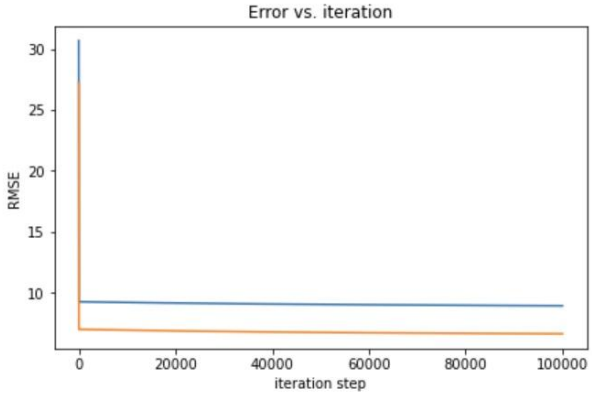

Error vs. iteration (tail)
RMSE
10.5
10.0
9.5
9.0
8.5
8.0
7.5
7.0
0
20000
40000
60000
80000
100000
iteration step

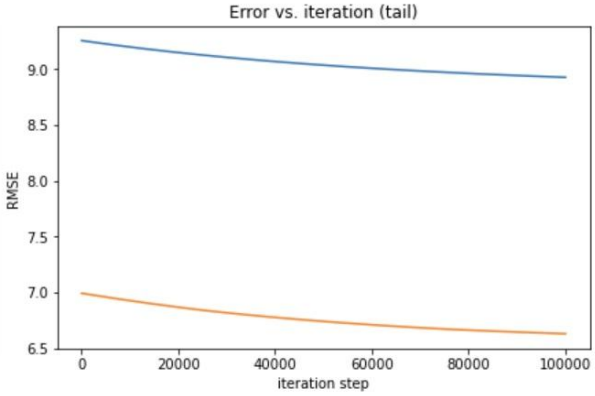

Error vs. iteration
RMSE
30
25
20
15
10
0
20000
40000
60000
80000
100000
iteration step


Error vs. iteration (tail)
RMSE
9.0
8.5
8.0
7.5
7.0
6.5
0
20000
40000
60000
80000
100000
iteration step


Error vs. iteration
RMSE
iteration step
0
20000
40000
60000
80000
100000
10
15
20
25
30


Error vs. iteration (tail)
RMSE
iteration step
0
20000
40000
60000
80000
100000
7.0
7.5
8.0
8.5
9.0
9.5
10.0
10.5


Error vs. iteration
RMSE
iteration step
0
20000
40000
60000
80000
100000
10
15
20
25
30


Error vs. iteration (tail)
RMSE
iteration step
0
20000
40000
60000
80000
100000
6.5
7.0
7.5
8.0
8.5
9.0

L2(Ridge Regression)

```
The RMSE error for Ridge Regularization when K is 5 is 9.654644061943909%
```

Concepts Used

Ridge Regularization

- Minimization objective = LS Obj + λ * (sum of square of slope)

$$J(\Theta) = \frac{1}{M}\sum_{i=1}^{M}(\Theta^T\Phi(x_i) - y_i)^2 + \lambda\sum_{j=1}^{p}\Theta_j^2$$

$$\frac{\partial J}{\partial \Theta} = \frac{2}{M}\sum_{i=1}^{M}(\Theta_T\Phi(x_i) - y_i)\Phi(x_i) + 2\lambda\Theta_j$$

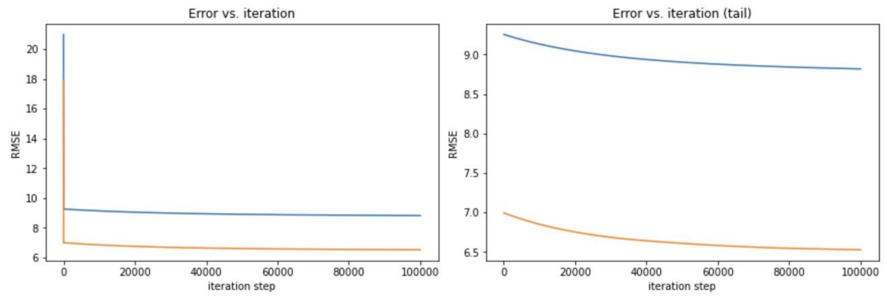
$$\Theta_(j+1) = (1 - 2\lambda\alpha)\Theta_j - \frac{2\alpha}{M}\sum_{i=1}^{M}(\Theta^T\Phi(x_i) - y_i)\Phi(x_i)$$

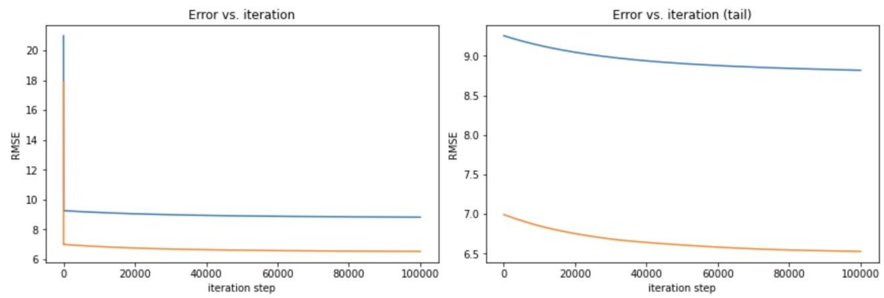
```
: def Ridge(X, Y, learning_rate, epochs):

      W = np.zeros((X.shape[1], 1))
      b= 3
      alpha = 0.00005
      print(W.shape)

      for i in range(epochs):
          y_pred = np.dot(X, W)

          LW = (-(2*(X.T).dot(Y - y_pred)) + (2*alpha*W))  /X.shape[0]

          W -= learning_rate*LW

      return W

  epochs = 100000
  learning_rate = 0.00000003

  def check_errorl1(X_test , Y_test , theta ):

      y_pred = np.dot(X_test, theta)


      print("Training Error is ")
      print( (np.sum(((abs(Y_test - y_pred))))/Y_test.shape[0]) )


      print("RMSE Error (validation loss) is ")
      print((np.sum(((abs(Y_test - y_pred))**2))/Y_test.shape[0])**0.5)
      return ((np.sum(((abs(Y_test - y_pred))**2))/Y_test.shape[0])**0.5)

  error_5 = 0
```

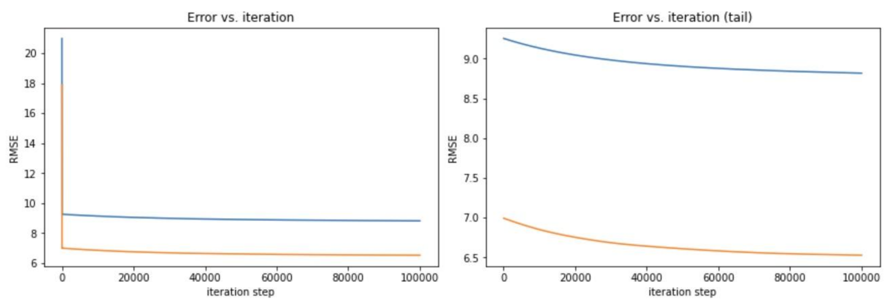
Here we are changing the theta (W) using the parameters according to the Ridge regression and then changing it using the learning rate, and then finally using it after running all the epochs. Then we calculate the training and the validation error.

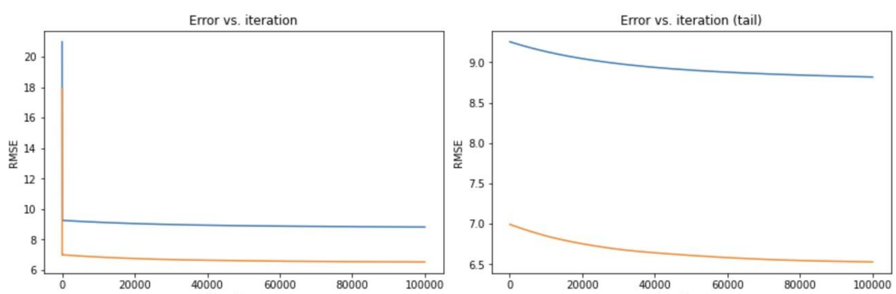
Reference taken from Lecture Notes

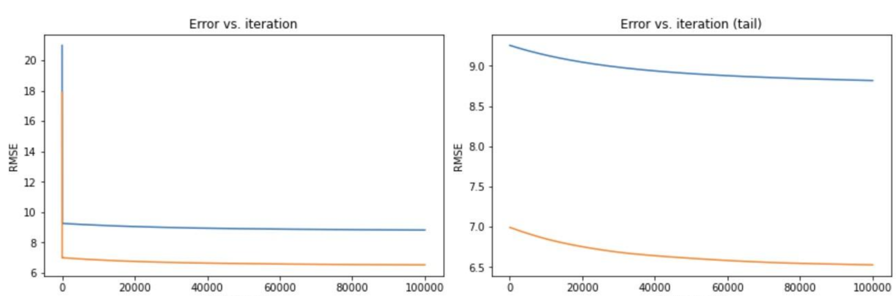
Graphs











Reference Taken from Lecture Notes

(d) Implement the normal equation (closed form) for linear regression

```
The MEAN RMSE error for Normal Equation for K = 5 is 8.746947008177O1%
```

Concepts Used

$W = (X^T.X)^{-1}.X^T.Y$
Where W is our Theta.

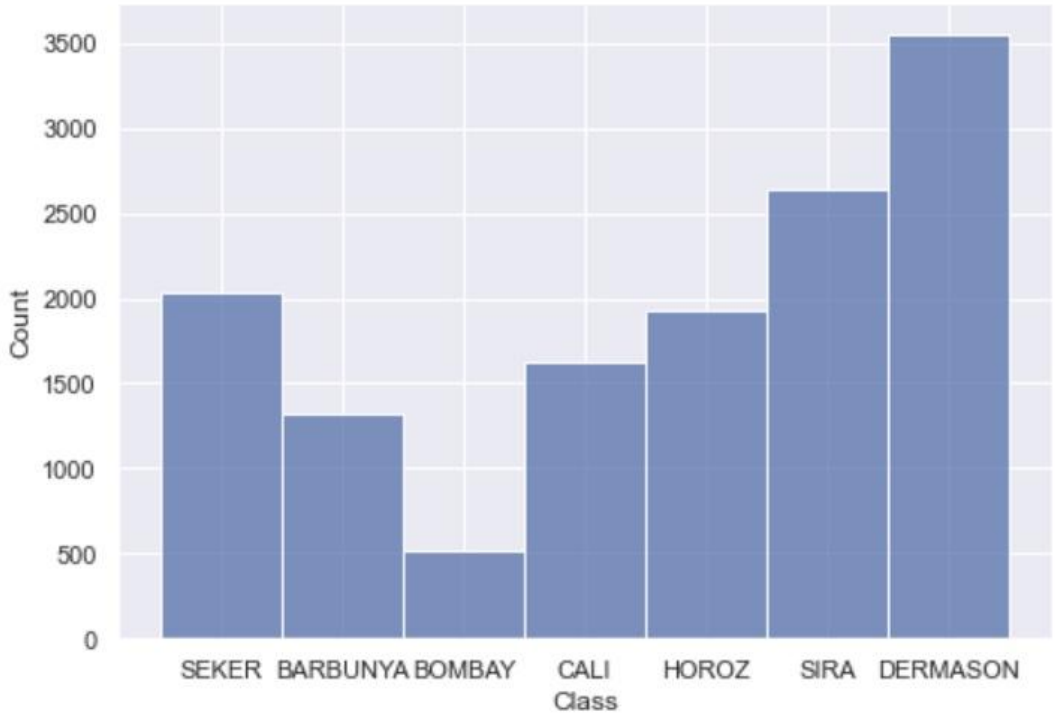
```
def Normal(X, Y, learning_rate, epochs):

    theta = np.ones((X.shape[1], 1))

    theta = ( np.linalg.inv((X.T.dot(X))) ).dot(X.T).dot(Y)

    return theta

def check_error(X_test , Y_test , theta):

    y_pred = np.dot(X_test, theta)

    print("RMSE Error (validation loss) is ")
    print((np.sum(((abs(Y_test - y_pred))**2))/Y_test.shape[0])**0.5)
    return ((np.sum(((abs(Y_test - y_pred))**2))/Y_test.shape[0])**0.5)

error_5_1=0
```

Here we have used the Normal equation to find the value of theta and used it to find the predicted values.

SECTION-C

C)

(a)The class distribution and analysis



We can observe that we have the least number of dry beans in the Bombay Variety and the most number of beans in the Dermason variety.

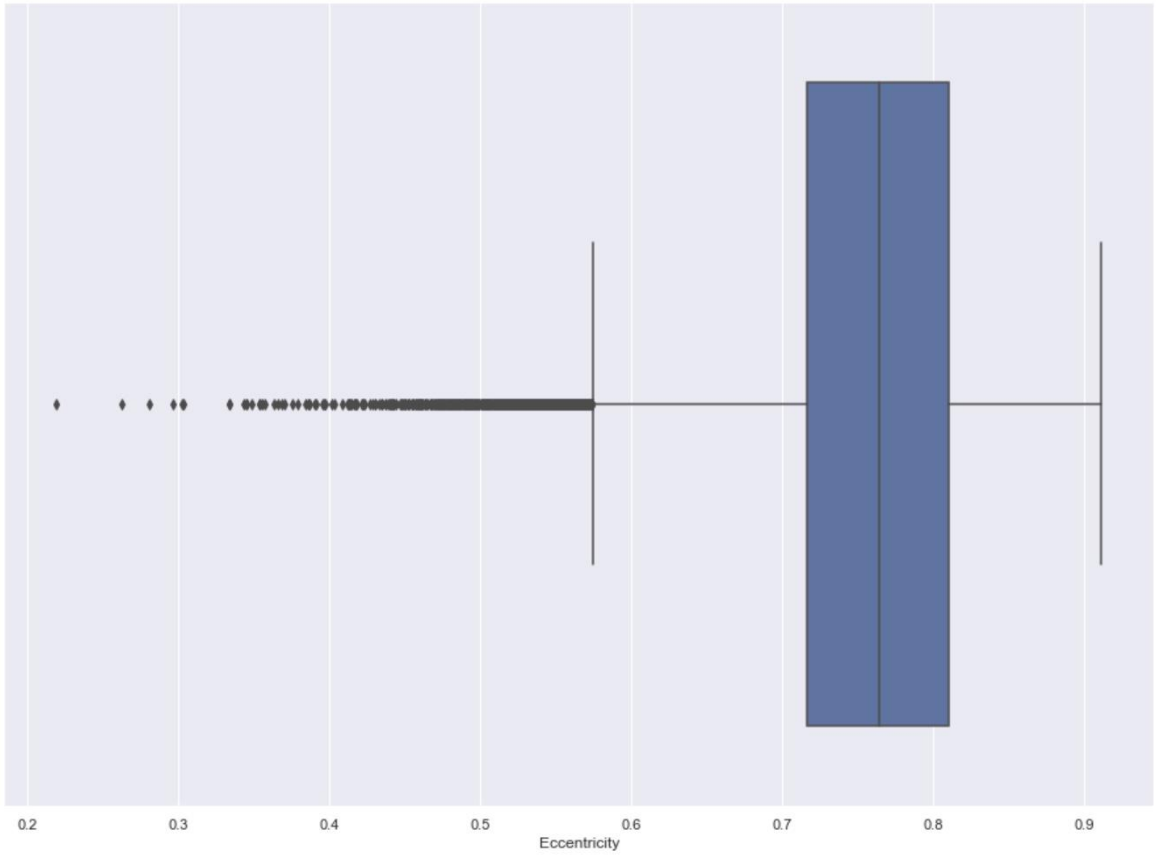
```
graph  = sns.displot(df, x="Class")
graph.fig.set_figwidth(7.27)
graph.fig.set_figheight(4.7)
```

(b) Performing EDA

(i)

(i) BoxPlot for Eccentrvity - As we can see below , that many values less that 0.6 are outliers and cause major deviation to the data

```
graph = sns.boxplot(x='Eccentricity', data=df)
```

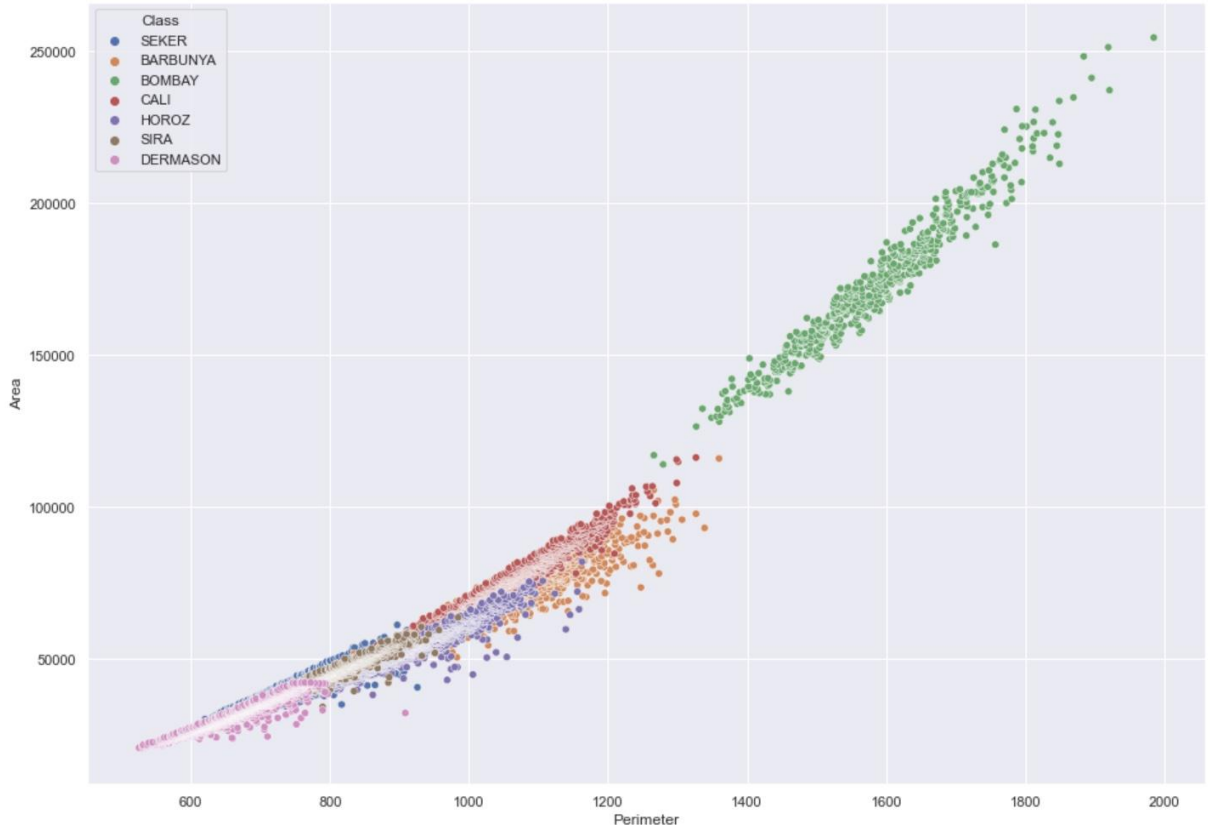


We are using the boxplot to find the outliers for eccentricity.

(ii)

(ii) Scatter Plot for Area/Perimeter - Here we can see that the graph is non linear as the rate with which Area increases as permeter increases (m - slope) is increasing We can Also see that the BOMBAY beans are the the biggest with largest area and perimeter And the DERMASON are the smallest

```
# df.plot.scatter(x = 'Perimeter', y = 'Area', s = 1)
graph = sns.scatterplot(x = 'Perimeter' , y='Area' , hue='Class' , data=df)
```



We are using Scatterplot to find the relation between Area and Perimeter

(iii)

(iii) Scatter Plot for Roundness/Class - Here we can see that SEKER are the most round and compact beans, whereas the HOROZ are least round and compact The compactness and roundness of BOMBAY beans are somewhat between SEKER and HOROZ

```
graph = sns.scatterplot(x = 'roundness' , y='Compactness' , hue='Class' , data=df)
```

We are using the Scatterplot to find the relationship between roundness and Compactness.

(iv)

iv. Graph between area and extent : Here we can see how all beans lie in same range of except HOROZ which has larger extent range. We can also see that BOMBAY is very much different from others in Area

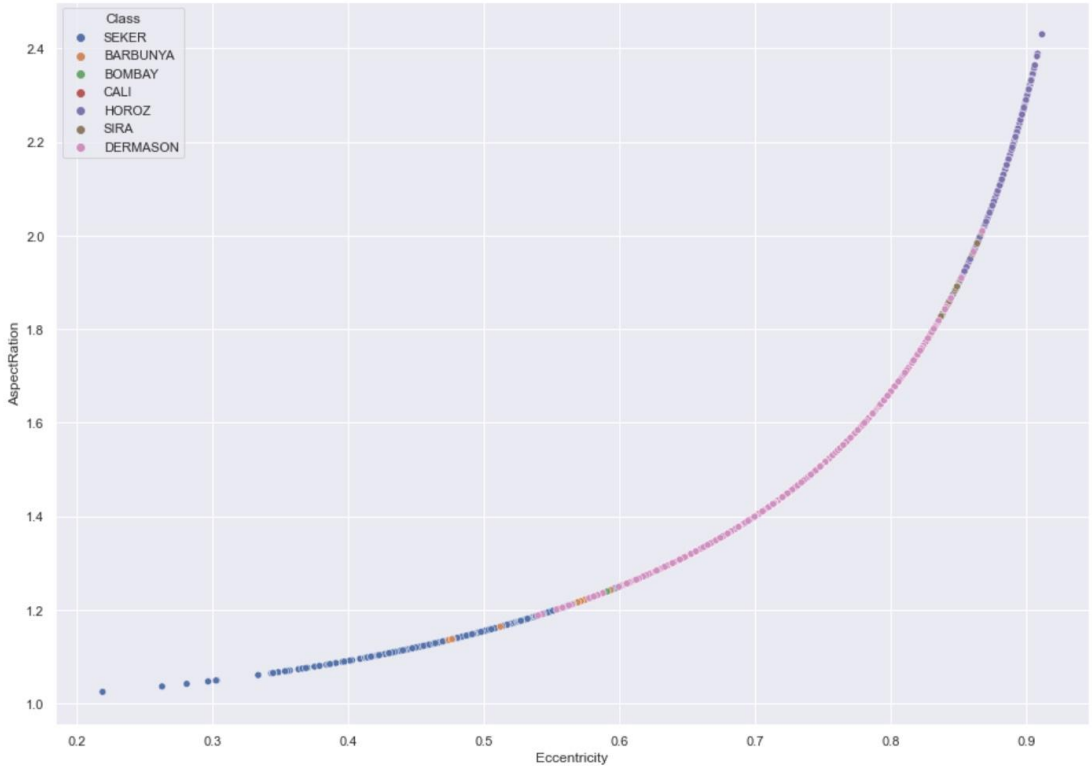
```
graph = sns.scatterplot(x = 'Area' , y='Extent' , hue='Class' , data=df)
```

We are using the Scatterplot to find the relationship between Extent and Area.

(v)

v. Graph between Eccentricity and Aspect Ratio : We can see that this is an exponential curve SEKER has the least eccentricity and least Aspect Ratio While HOROZ has the highest Eccentricity and hightest Aspect Ration

```
graph = sns.scatterplot(x = 'Eccentricity' , y='AspectRation', hue='Class'  , data=df)
```



We are using the Scatterplot to find the relationship between Aspect Ratio and eccentricity.

Check Missing Values

```
df.isnull().sum()  ## there are no null values in the data set.
```

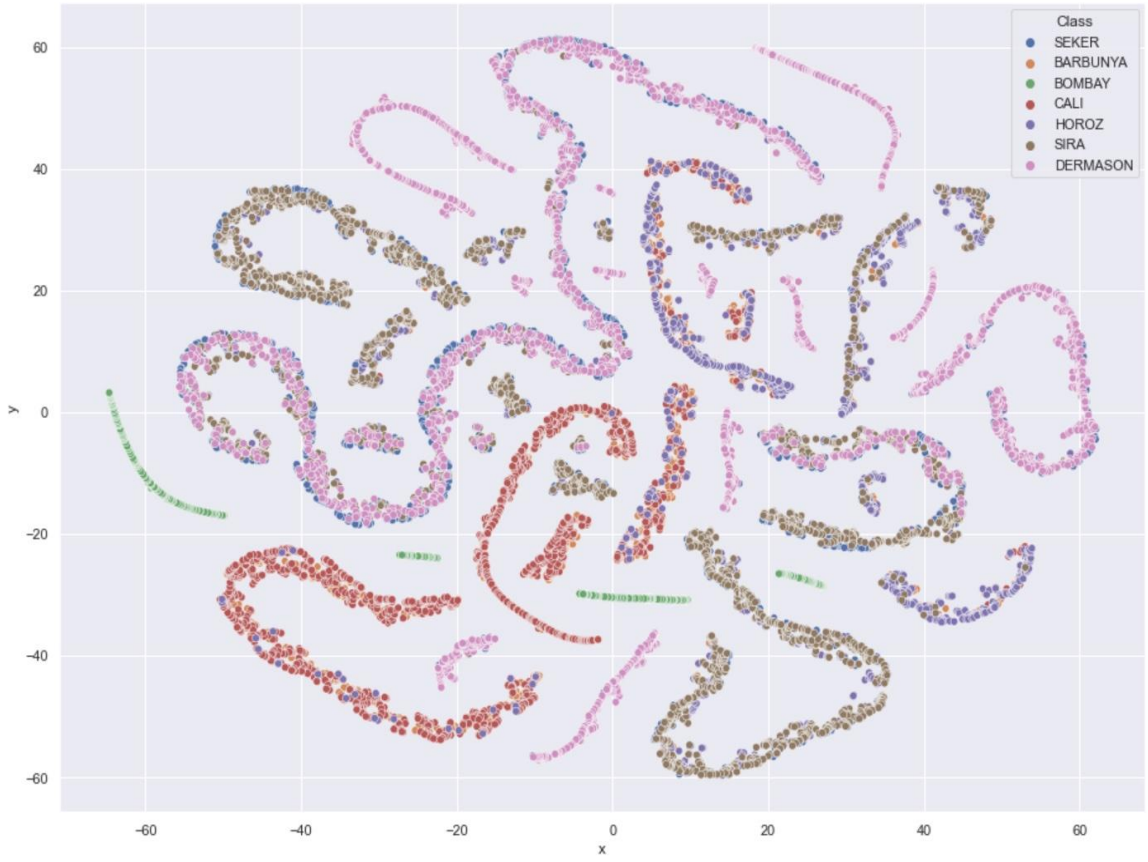
```
Area               0
Perimeter          0
MajorAxisLength    0
MinorAxisLength    0
AspectRation       0
Eccentricity       0
ConvexArea         0
EquivDiameter      0
Extent             0
Solidity           0
roundness          0
Compactness        0
ShapeFactor1       0
ShapeFactor2       0
ShapeFactor3       0
ShapeFactor4       0
Class              0
dtype: int64
```

No Missing Values reported

(C)
TSNE (t-distributed stochastic neighbor embedding) algorithm to reduce data dimensions to 2.
We have 17 Dimension data, and we have to reduce it to 2 dimensions.
We will use the TSNE algorithm for this reduction.
We have named the columns x and y and added them to the data frame

ScatterPlot

```
graph = sns.scatterplot(x = 'x' , y='y' , hue='Class' , data=df)
```



We can infer that HOROZ and CALI are similiar to each other Also are HOROZ and SIRA SEKER and SIRA are also similiar SEKER and DERAMASON are also similiar SEKER, CALI and BARBUNIYA are also simliar CALI and BARBUNIYA are most similiar to each other However DERAMASON is different from others in some aspects However BOMBAY is completely different from all others

(d)

Note- For Naive Bayes, we have added an extra column for Classes but in integer format, where the classes have been assigned a label integer.

(i)Multinomial Naive Bayes

Accuracy Score 0.786265148733015

	precision	recall	f1-score	support
0	0.59	0.59	0.59	255
1	1.00	1.00	1.00	92
2	0.77	0.72	0.74	350
3	0.87	0.87	0.87	735
4	0.82	0.77	0.80	390
5	0.77	0.76	0.77	376
6	0.73	0.81	0.77	525
accuracy			0.79	2723
macro avg	0.79	0.79	0.79	2723
weighted avg	0.79	0.79	0.79	2723

So we are getting Accuracy as 0.786 , Precision as 0.79 ,recall as 0.79 for Multinomial Naive Bayes

Multinomial Naive Bayes is used for multinomial models, suitable for classification with discrete (integer) features.

i. Multinomial Naive Bayes

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split

df['Class'] = pd.Categorical(df['Class'])
df['ClassCode'] = df['Class'].cat.codes
## Pre processing added a columns which assign specific numbers to the Class do that we can classify using integers
y_MNV = df[df.columns[19:20]]
X_MNV = df[df.columns[0:16]]
X_train_MNV, X_test_MNV, y_train_MNV, y_test_MNV = train_test_split(X_MNV, y_MNV, test_size=0.2, random_state=0)

clf = MultinomialNB()
# df

## Reference Taken from scikit-learn.org
```

We have added one more column ClassCode which represents Class but in integer form

```
clf.fit(X_MNV, y_MNV)
MultinomialNB()
y_pred_MNV = clf.predict(X_test_MNV)
y_pred_MNV = y_pred_MNV.reshape(y_pred_MNV.shape[0] , 1)
```

(ii)Categorial Naive Bayes

```
Accuracy Score 0.9063532868160118
              precision    recall  f1-score   support

           0       0.95      0.82      0.88       255
           1       1.00      1.00      1.00        92
           2       0.91      0.93      0.92       350
           3       0.92      0.95      0.93       735
           4       0.92      0.89      0.91       390
           5       0.96      0.86      0.91       376
           6       0.82      0.90      0.85       525

    accuracy                           0.91      2723
   macro avg       0.92      0.91      0.92      2723
weighted avg       0.91      0.91      0.91      2723
```

So we are getting Accuracy as 0.906 , Precision as 0.91 ,recall as 0.91 for Categorial Naive Bayes

We Chose Multinomial Naive Bayes and Categorial Naive Bayes as they provided most accuracy among all other Naive Bayes. From what all I could infer after reading the documentation and the Dataset Values Since we have multiple features , Multinomial Naive Bayes provided high (79%) accuracy Since we had categorised data (Class values) , Categorial naive Bayes provided high (90%) accuracy

ii. Categorial naive Bayes

```
from sklearn.naive_bayes import CategoricalNB
from sklearn.model_selection import train_test_split

clf = CategoricalNB()
df['Class'] = pd.Categorical(df['Class'])
df['ClassCode'] = df['Class'].cat.codes
## Pre processing added a columns which assign specific numbers to the Class do that we can classify using integers
y_CNB = df[df.columns[19:20]]
X_CNB = df[df.columns[0:16]]


## Reference Taken from scikit-learn.org
```

```
X_train_CNB, X_test_CNB, y_train_CNB, y_test_CNB = train_test_split(X_CNB, y_CNB, test_size=0.2, random_state=0)

clf.fit(X_CNB, y_CNB)
CategoricalNB()
y_pred_CNB = clf.predict(X_test_CNB)
y_pred_CNB = y_pred_CNB.reshape(y_pred_CNB.shape[0] , 1)
```

Categorial Naive Bayes is used for models, suitable for classification where features are distributed categorically.

Reference taken from scikit-learn.org.

(e)

Principal Component Analysis (PCA) to reduce the number of features and use the reduced data set for model training.

We trained the Categorial Naive Bayes models for 4 , 6 , 8 , 10 , and 12 features

```
PCA results with 4 features
Variance:  0.9999999749408085
Accuracy Score 0.9063532868160118
              precision    recall  f1-score   support

           0       0.95      0.82      0.88       255
           1       1.00      1.00      1.00        92
           2       0.91      0.93      0.92       350
           3       0.92      0.95      0.93       735
           4       0.92      0.89      0.91       390
           5       0.96      0.86      0.91       376
           6       0.82      0.90      0.85       525

    accuracy                           0.91      2723
   macro avg       0.92      0.91      0.92      2723
weighted avg       0.91      0.91      0.91      2723


PCA results with 6 features
Variance :  0.999999999997927
Accuracy Score 0.9063532868160118
              precision    recall  f1-score   support

           0       0.95      0.82      0.88       255
           1       1.00      1.00      1.00        92
           2       0.91      0.93      0.92       350
           3       0.92      0.95      0.93       735
           4       0.92      0.89      0.91       390
           5       0.96      0.86      0.91       376
           6       0.82      0.90      0.85       525

    accuracy                           0.91      2723
   macro avg       0.92      0.91      0.92      2723
weighted avg       0.91      0.91      0.91      2723
```

PCA results with 8 features
Variance : 0.9999999999998419
Accuracy Score 0.9063532868160118

	precision	recall	f1-score	support
0	0.95	0.82	0.88	255
1	1.00	1.00	1.00	92
2	0.91	0.93	0.92	350
3	0.92	0.95	0.93	735
4	0.92	0.89	0.91	390
5	0.96	0.86	0.91	376
6	0.82	0.90	0.85	525
accuracy			0.91	2723
macro avg	0.92	0.91	0.92	2723
weighted avg	0.91	0.91	0.91	2723

PCA results with 10 features
Variance : 0.9999999999999926
Accuracy Score 0.9063532868160118

	precision	recall	f1-score	support
0	0.95	0.82	0.88	255
1	1.00	1.00	1.00	92
2	0.91	0.93	0.92	350
3	0.92	0.95	0.93	735
4	0.92	0.89	0.91	390
5	0.96	0.86	0.91	376
6	0.82	0.90	0.85	525
accuracy			0.91	2723
macro avg	0.92	0.91	0.92	2723
weighted avg	0.91	0.91	0.91	2723

```
PCA results with 12 features
Variance :  1.0000000000000018
Accuracy Score 0.9063532868160118
              precision    recall  f1-score   support

           0       0.95      0.82      0.88       255
           1       1.00      1.00      1.00        92
           2       0.91      0.93      0.92       350
           3       0.92      0.95      0.93       735
           4       0.92      0.89      0.91       390
           5       0.96      0.86      0.91       376
           6       0.82      0.90      0.85       525

    accuracy                           0.91      2723
   macro avg       0.92      0.91      0.92      2723
weighted avg       0.91      0.91      0.91      2723
```

We are getting similar results from all the values (4,6,8,10,12)

Accuracy : 0.9063532868160118
Precision : 0.91
Recall: 0.91
F1-score : 0.91

Code for PCA when we take 4 features

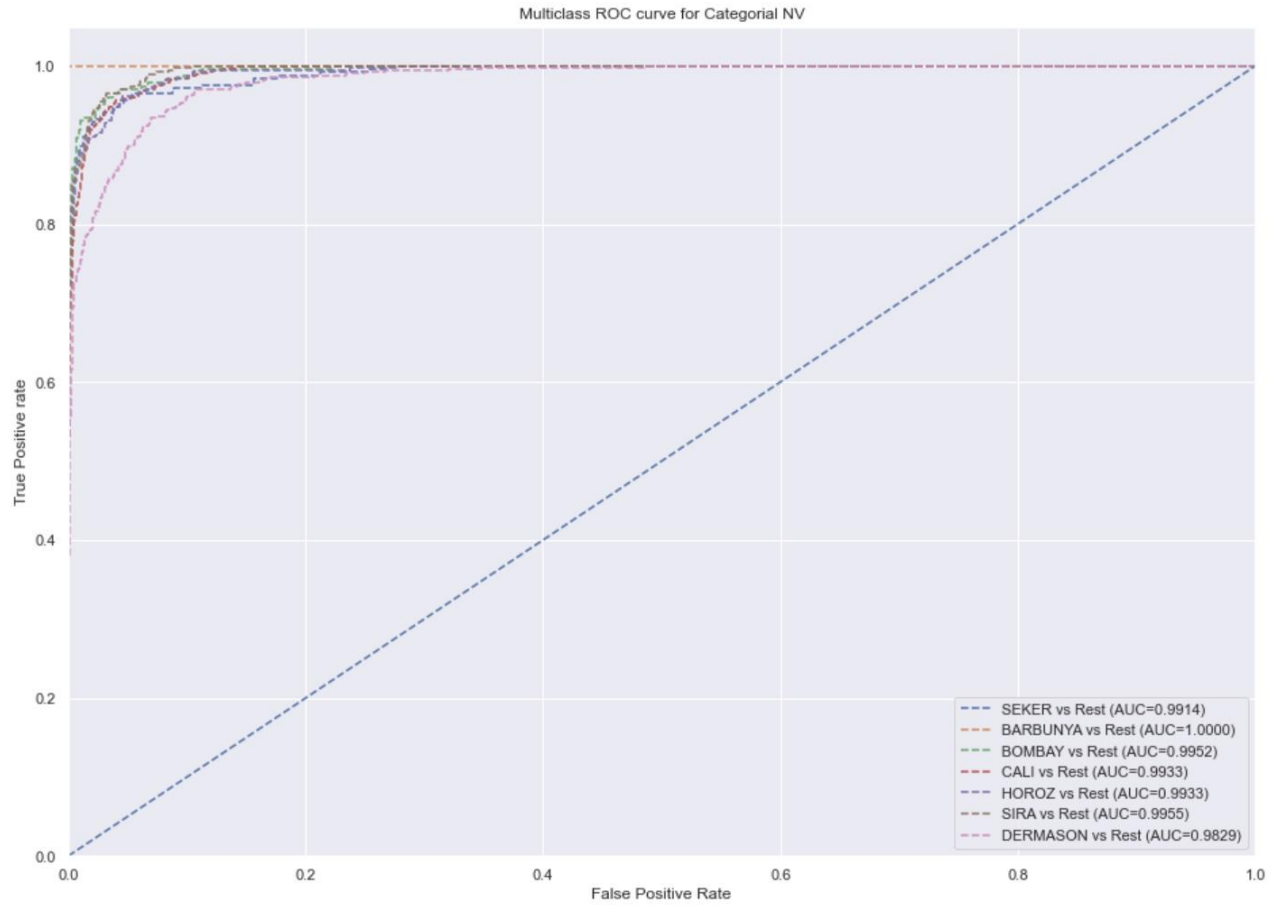
```
from sklearn.preprocessing import scale # Data scaling
from sklearn import decomposition #PCA
from sklearn.decomposition import PCA


df['Class'] = pd.Categorical(df['Class'])
df['ClassCode'] = df['Class'].cat.codes
## Pre processing added a columns which assign specific numbers to the Class do that we can classify using integers
y_PCA = df[df.columns[19:20]]
X_PCA = df[df.columns[0:16]]

pca = decomposition.PCA(n_components=4)   ## we are taking in 4 components
pca.fit(X_PCA)
scores = pca.transform(X_PCA)

pca_2 = pca

scores_df = pd.DataFrame(scores, columns=['PC1', 'PC2' , 'PC3' , 'PC4' ])  ## Creating a new dataframe
scores_df

# ##
print("PCA results with 4 features")
print("Variance: " , pca.explained_variance_ratio_.sum())

X_train_PCA, X_test_PCA, y_train_PCA, y_test_PCA = train_test_split(X_PCA, y_PCA, test_size=0.2, random_state=0)

clf.fit(X_PCA, y_PCA)
CategoricalNB()    ##Training the model using Categorial Naive bayes
y_pred_PCA = clf.predict(X_test_PCA)
y_pred_PCA = y_pred_PCA.reshape(y_pred_PCA.shape[0] , 1)
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
print("Accuracy Score" , accuracy_score(y_test_PCA , y_pred_PCA))

print(classification_report(y_test_PCA , y_pred_PCA))
```

Reference is taken from scikit-learn.org.

(f)

This is ROC Curve and AOC values for the Categorial Naive Bayes model for different Classes. (After PCA)

The ROC curve plots the true positive rates and false positive rates showing the performance of our model.



Even though our model has high AUC The model is easily able to distinguish the BARBUNIYA beans from the other beans as it has highest AUC = 1. Where as the model not easily able to distinguish DERMASON beans from other beans as it has lowest AUC = 0.9829.

Reference is taken from scikit-learn.org

Code for Producing the ROC curve

```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve,auc
from sklearn.preprocessing import label_binarize

y_test_binarized=label_binarize(y_test_PCA,classes=np.unique(y_test_PCA))
# print(y_test_binarized)
fpr = {}   ## false positive rate
tpr = {}  ## true positive rate
thresh ={}  ## threshold
roc_auc = dict()

pred_prob = clf.predict_proba(X_test_PCA)   ## plotting the ROC curve of Categorial Naive Bayes after PCA

n_class = 7
classes = df['Class'].unique()

for i in range(n_class):
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test_binarized[:,i], pred_prob[:,i])
    roc_auc[i] = auc(fpr[i], tpr[i])

    plt.plot(fpr[i], tpr[i], linestyle='--', label='%s vs Rest (AUC=%0.4f)'%(classes[i],roc_auc[i]))
    ## For Plotting the printing the AOC values on the curve

plt.plot([0,1],[0,1],'b--')
plt.xlim([0,1])
plt.ylim([0,1.05])
plt.title('Multiclass ROC curve for Categorial NV')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='lower right')
plt.show()
```

(g)
Sklearn’s implementation of Logistic Regression

```
Accuracy Score 0.9048843187660668
              precision    recall  f1-score   support

    BARBUNYA       0.93      0.89      0.91       270
      BOMBAY       1.00      1.00      1.00       116
        CALI       0.93      0.93      0.93       340
    DERMASON       0.89      0.90      0.90       702
       HOROZ       0.94      0.94      0.94       396
       SEKER       0.93      0.94      0.93       383
        SIRA       0.82      0.83      0.83       516

    accuracy                           0.90      2723
   macro avg       0.92      0.92      0.92      2723
weighted avg       0.91      0.90      0.90      2723
```

We have taken 10000 iterations for logistic regression We are getting Accuracy : 0.9210429673154609 Precision : 0.92 Recall : 0.92 F-1 Score : 0.92

These results are better than the results obtained from The Naive Bayes models

Reference Taken from scikit-learn.org

Code for Sklearn’s implementation of Logistic Regression

g. Using Sklearn’s implementation of Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

model = LogisticRegression(max_iter=10000)  ## Setting the number of epochs

y_train_LG = df[df.columns[16:17]]  ## Selecting the X and Y columns
X_train_LG = df[df.columns[0:16]]


# df[df.columns[16:17]]
X_train_LG, X_test_LG, y_train_LG, y_test_LG = train_test_split(X_train_LG,y_train_LG, test_size=0.2)
model.fit(X_train_LG, y_train_LG)   ## SK learn model
```

```
LogisticRegression(max_iter=10000)
```

```
y_pred_LG = model.predict(X_test_LG)
```