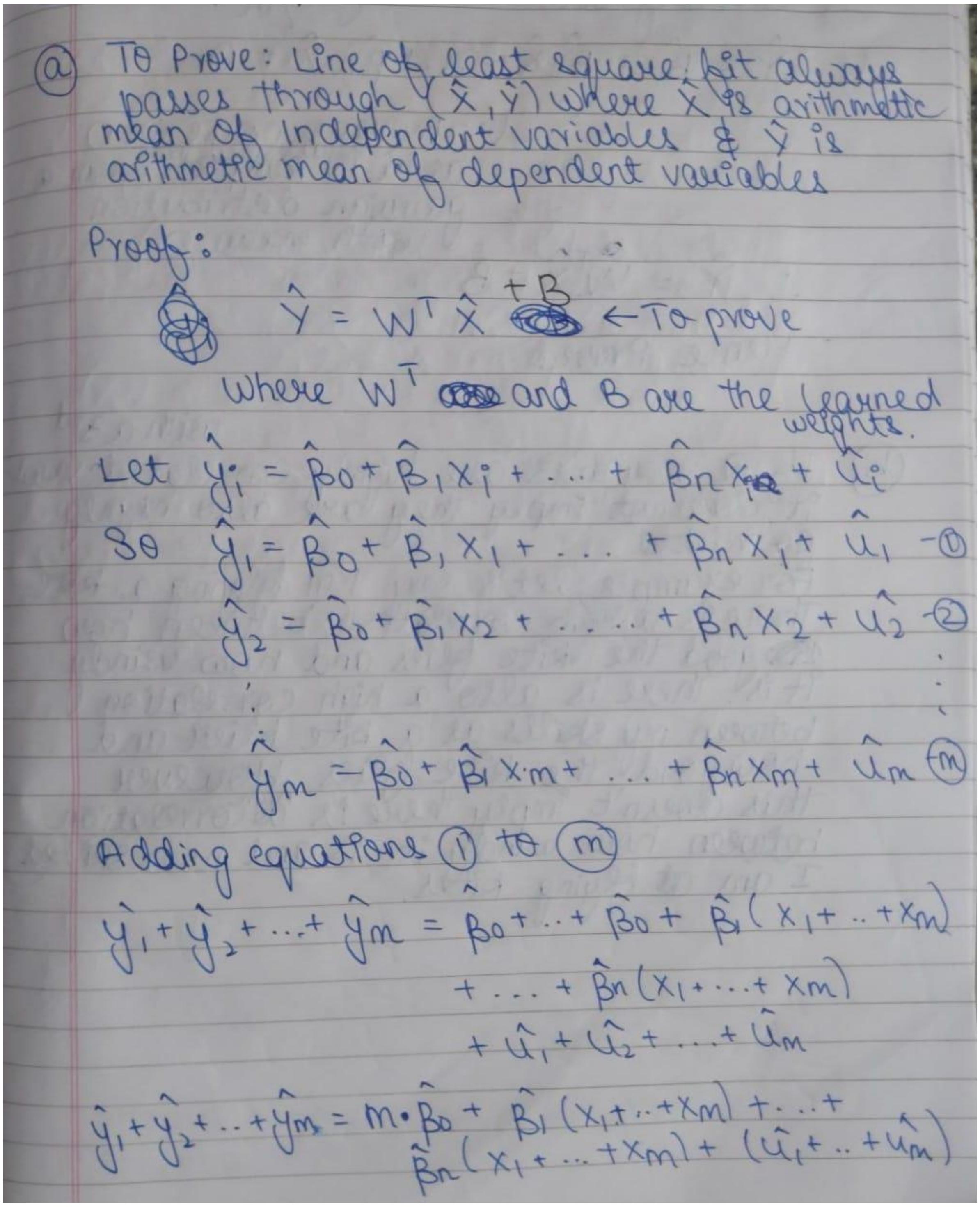
ML Assignment 1

Mohammad Aflah Khan, 2020082

Section A

1)

a)



(a) To Prove: Line of least square fit always passes through $(\hat{x}, \hat{y})$ where $\hat{x}$ is arithmetic mean of Independent variables & $\hat{y}$ is arithmetic mean of dependent variables

Proof:

$$\hat{Y} = W^T \hat{X} + B \leftarrow \text{To prove}$$

Where W^T and B are the learned weights.

Let $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i + \ldots + \beta_n x_i + \hat{u}_i$

So $\hat{y}_1 = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \ldots + \hat{\beta}_n x_1 + \hat{u}_1$ — (1)

$\hat{y}_2 = \hat{\beta}_0 + \hat{\beta}_1 x_2 + \ldots + \hat{\beta}_n x_2 + \hat{u}_2$ — (2)

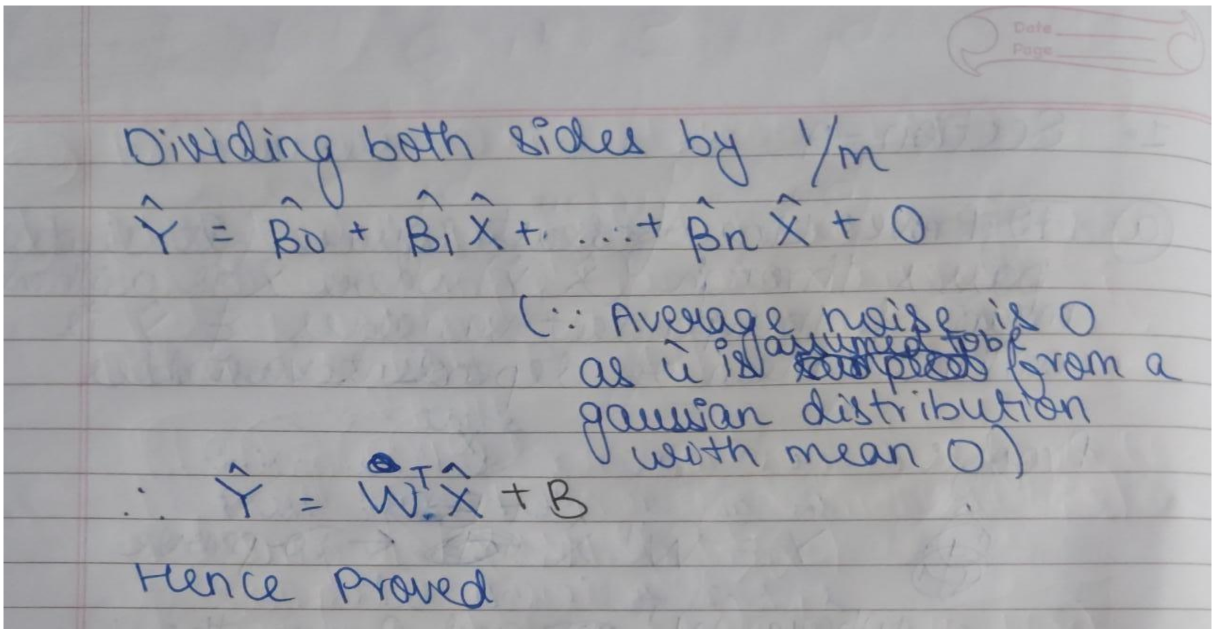
$\vdots$

$\hat{y}_m = \hat{\beta}_0 + \hat{\beta}_1 x_m + \ldots + \hat{\beta}_n x_m + \hat{u}_m$ (m)

Adding equations (1) to (m)

$$\hat{y}_1 + \hat{y}_2 + \ldots + \hat{y}_m = \hat{\beta}_0 + \ldots + \hat{\beta}_0 + \hat{\beta}_1(x_1 + \ldots + x_m) + \ldots + \hat{\beta}_n(x_1 + \ldots + x_m) + \hat{u}_1 + \hat{u}_2 + \ldots + \hat{u}_m$$

$$\hat{y}_1 + \hat{y}_2 + \ldots + \hat{y}_m = m \cdot \hat{\beta}_0 + \hat{\beta}_1(x_1 + \ldots + x_m) + \ldots + \hat{\beta}_n(x_1 + \ldots + x_m) + (\hat{u}_1 + \ldots + \hat{u}_m)$$



Dividing both sides by 1/m

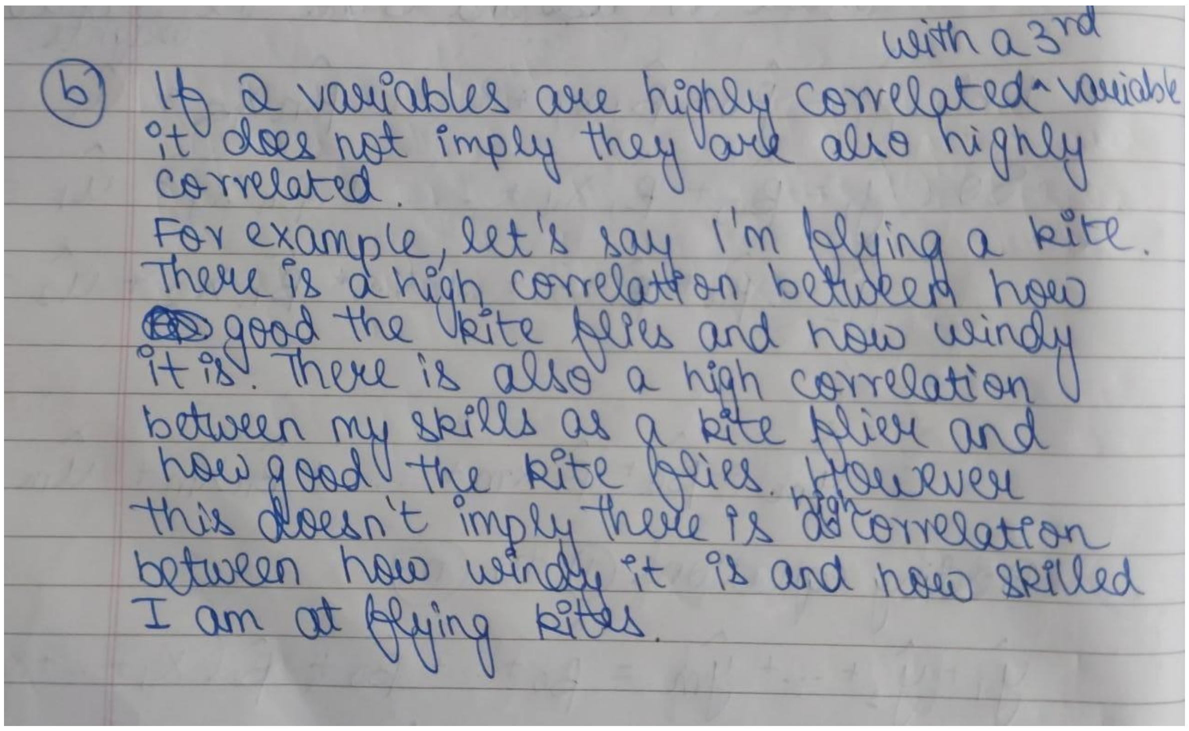
$\hat{Y} = \hat{B_0} + \hat{B_1}\hat{X} + \dots + \hat{\beta_n}\hat{X} + 0$

(∵ Average noise is 0 as $\hat{u}$ is ~~[illegible]~~ assumed to be from a gaussian distribution with mean 0)

∴ $\hat{Y} = W^T\hat{X} + B$

Hence Proved

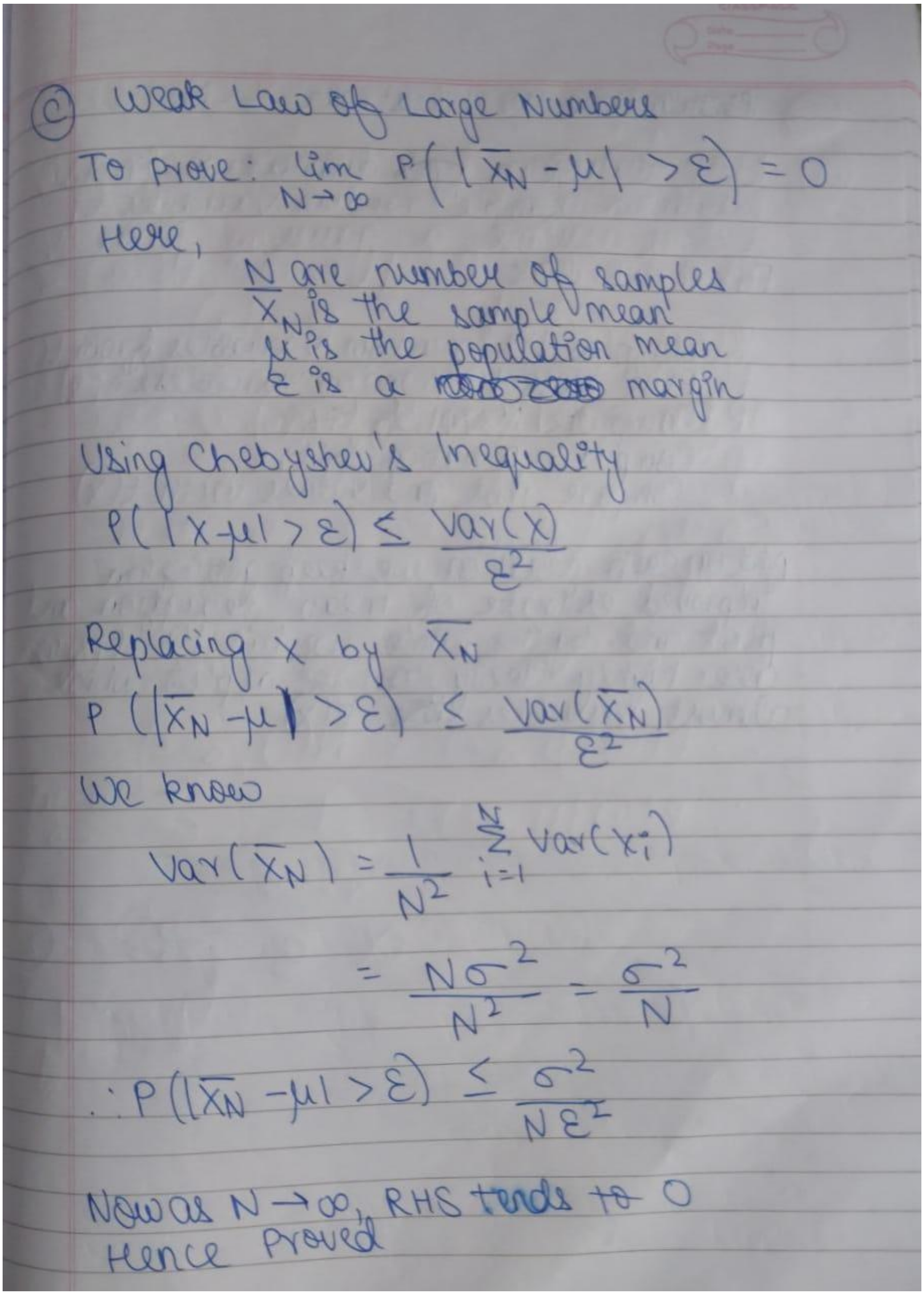
b)



(b) If 2 variables are highly correlated with a 3rd variable it does not imply they are also highly correlated.

For example, let's say I'm flying a kite. There is a high correlation between how [illegible] good the kite flies and how windy it is. There is also a high correlation between my skills as a kite flier and how good the kite flies. However this doesn't imply there is high ~~as~~ correlation between how windy it is and how skilled I am at flying kites.

c)



(c) Weak Law of Large Numbers

To prove: $\lim_{N \to \infty} P\left(|\overline{X_N} - \mu| > \varepsilon \right) = 0$

Here,

N are number of samples

$\overline{X_N}$ is the sample mean

μ is the population mean

ε is a [illegible] margin

Using Chebyshev's Inequality

$$P(|X - \mu| > \varepsilon) \leq \frac{Var(X)}{\varepsilon^2}$$

Replacing X by $\overline{X_N}$

$$P(|\overline{X_N} - \mu| > \varepsilon) \leq \frac{Var(\overline{X_N})}{\varepsilon^2}$$

We know

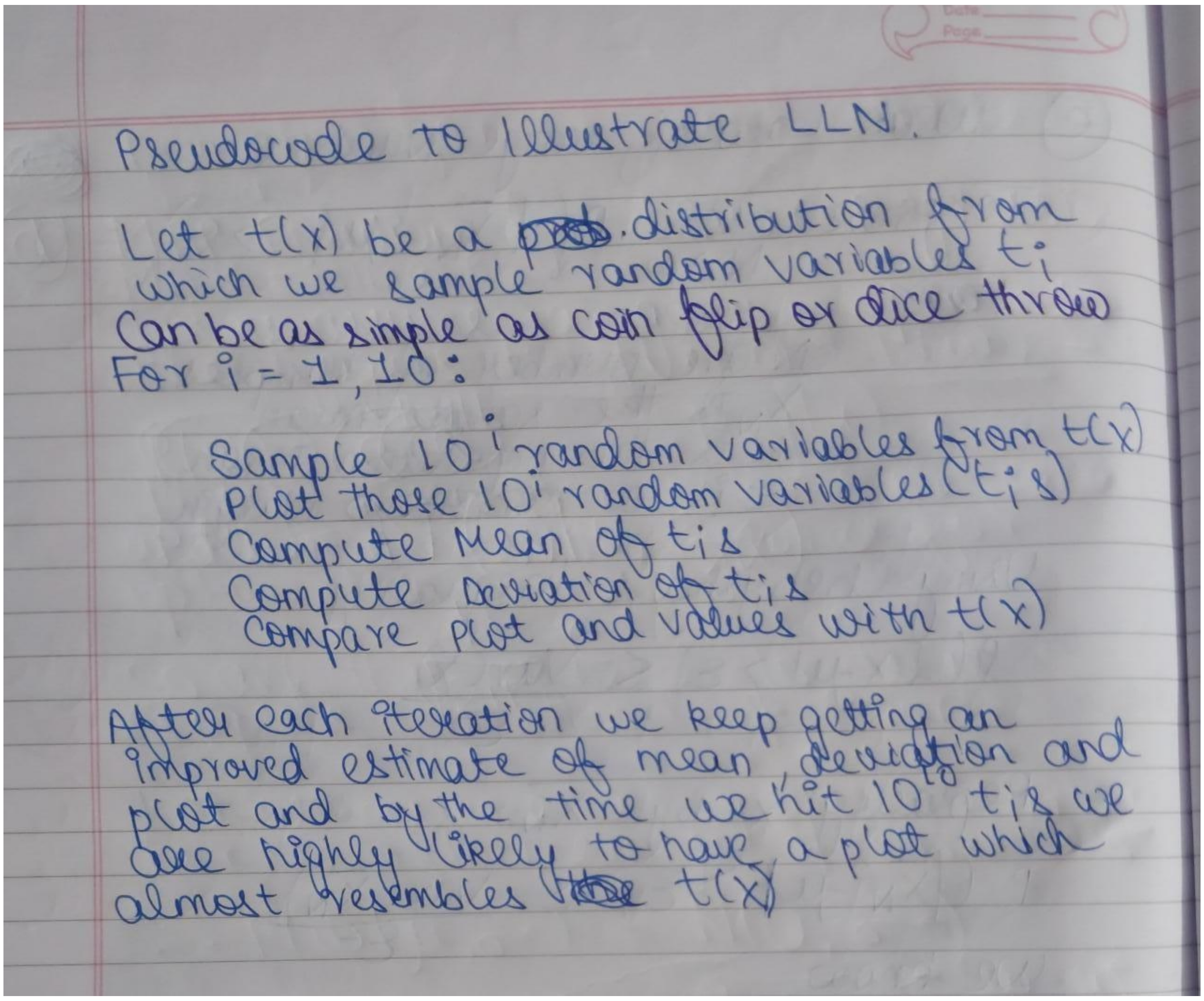
$$Var(\overline{X_N}) = \frac{1}{N^2} \sum_{i=1}^{N} Var(X_i)$$

$$= \frac{N\sigma^2}{N^2} = \frac{\sigma^2}{N}$$

$$\therefore P(|\overline{X_N} - \mu| > \varepsilon) \leq \frac{\sigma^2}{N\varepsilon^2}$$

Now as $N \to \infty$, RHS tends to 0

Hence Proved

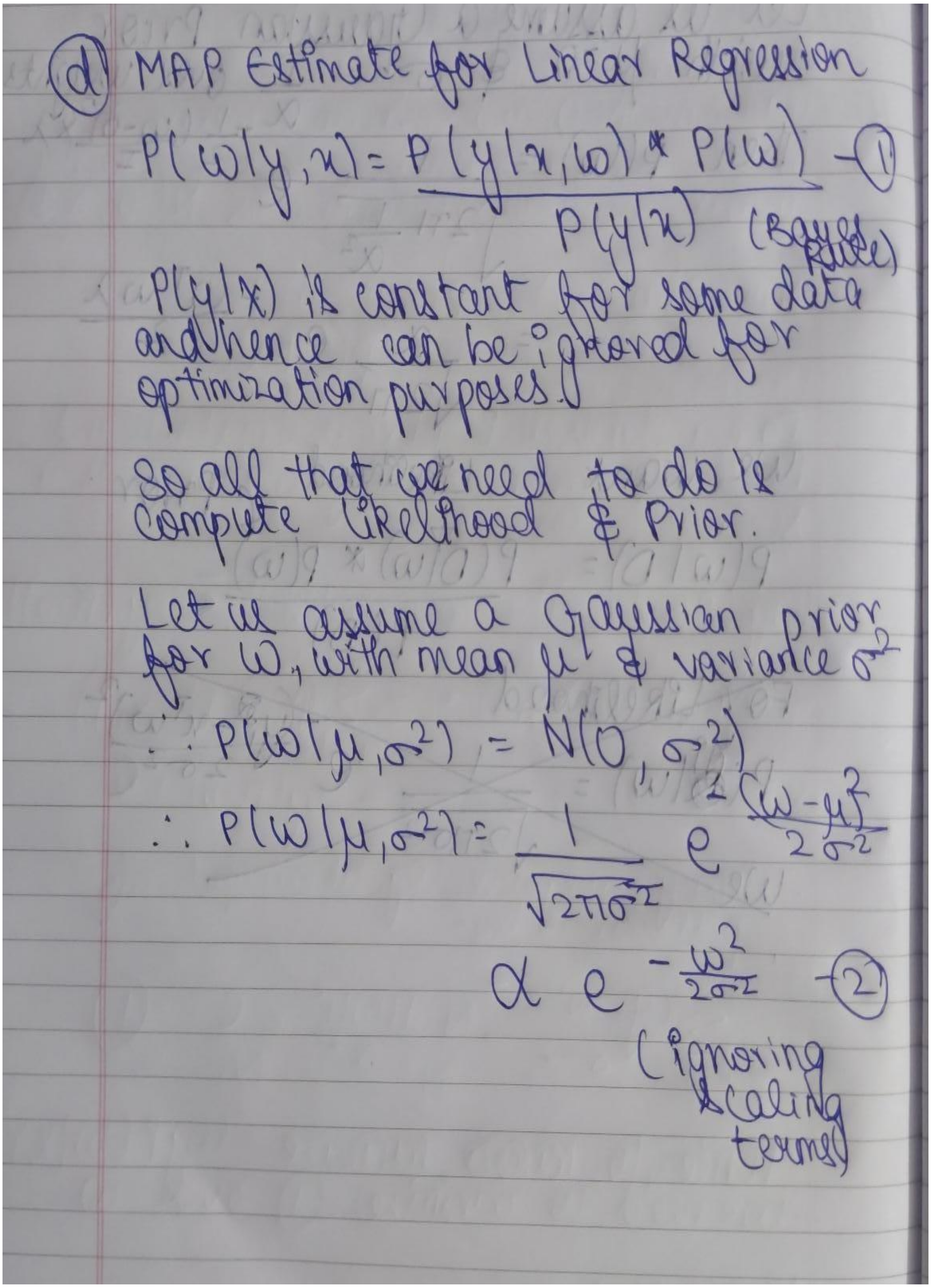


Pseudocode to Illustrate LLN.

Let t(x) be a .distribution from which we sample random variables t_i
Can be as simple as coin flip or dice throw

```
For i = 1, 10:

    Sample $10^i$ random variables from t(x)
    Plot those $10^i$ random variables ($t_i$s)
    Compute Mean of $t_i$s
    Compute Deviation of $t_i$s
    Compare plot and values with t(x)
```

After each iteration we keep getting an improved estimate of mean deviation and plot and by the time we hit 10^{10} t_is we are highly likely to have a plot which almost resembles t(x) d)



ⓓ MAP Estimate for Linear Regression

$$P(\omega|y,x) = \frac{P(y|x,\omega) * P(\omega)}{P(y|x)} \quad (1)$$

(Bayes Rule)

$P(y|x)$ is constant for some data and hence can be ignored for optimization purposes.

So all that we need to do is compute likelihood & Prior.

Let us assume a Gaussian prior for ω, with mean μ & variance σ^2

$$\therefore P(\omega|\mu,\sigma^2) = N(0,\sigma^2)$$

$$\therefore P(\omega|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\omega-\mu)^2}{2\sigma^2}}$$

$$\propto e^{-\frac{\omega^2}{2\sigma^2}} \quad (2)$$

(ignoring scaling terms)

For Likelihood

$P(y|x,w) = N(y|w^T x, \sigma^2)$

We model our target as a gaussian random variable with $\mu = w^T x$ and variance σ^2

$$w_{MLE} = \underset{w}{\operatorname{argmax}} N(y|w^T x, \sigma^2)$$

The PDF gives us

$$w_{MLE} = \underset{w}{\operatorname{argmax}}\ e^{-\frac{(y-w^T x)^3}{2\sigma^2}}$$

Taking log as it is easier to estimate

$$W_{MLE} = \underset{w}{\operatorname{argmax}} \frac{-1}{2\sigma^2}(y-w^T x)^2$$

$$= \underset{w}{\operatorname{argmin}} \frac{1}{2\sigma^2}(y-w^T x)^2 \quad \text{(3)}$$

Taking log on both sides in (1)

$$\log(P(w|y,x)) = \log(P(y|x,w)) + \log(P(w)) + C$$

$\therefore$ Using (2) & (3) and removing negative signs to get argmin.

$$\log(P(w|y,x)) \propto \frac{+1}{2\sigma^2}||y-w^T x||^2 + \frac{1}{2\sigma^2}||w||^2 \quad \text{(4)}$$

Hence for maximizing w we minimize R.H.S. of (4) to get MAP estimate

Section B

Pre-processing:

- Index Column Removed
- Column named 'X1 Transaction Date' was not very useful directly hence traced the upload year of dataset which turns out to be 2018 hence added a column which tells how many years ago was the house last sold as that seems to be a more useful feature
- A Standard Scaler was implemented for data scaling as without it the gradient descent was failing to converge and quickly reached extremely large weights. It is first fit on the train split and the value of mean and deviation obtained from the training set are used to transform both the train and validation set to avoid data leakage which might happen if we use values from validation set to compute mean and deviation.
- Instead of using a bias term separately a column of ones was appended to the data which essentially acts in the same way as now the parameter associated with that acts like the bias.

a)

The following part contains data for all RMSE and MSE for all validation folds for choices of K between 2 and 5 and the average validation and training RMSE alongside their variance, with the plot for the best choice of learning rate amongst 1e-5 to 100 increasing in jumps of 10.

2 Folds

Learning Rate: 0.0001

RMSE: 36.214587773310285, MSE: 1311.496367590795

RMSE: 35.816254490787934, MSE: 1282.804085748887

For Validation –

Average RMSE: 36.01542113204911, RMSE Variance: 0.039667350991257844

For Training –

Average RMSE: 36.006943388218325, RMSE Variance: 0.008231790478827305

Learning Rate: 0.001

RMSE: 16.640112872492434, MSE: 276.89335640928846

RMSE: 16.462243939582923, MSE: 271.0054755263347

For Validation –

Average RMSE: 16.55117840603768, RMSE Variance: 0.007909339323592082

For Training –

Average RMSE: 16.485028188659257, RMSE Variance: 0.09449232445420529

Learning Rate: 0.01

```
RMSE: 8.165105652285218, MSE: 66.66895031298003

RMSE: 9.629049634769604, MSE: 92.71859686885664

For Validation –

Average RMSE: 8.89707764352741, RMSE Variance: 0.5357829959630609

For Training –

Average RMSE: 8.730389008224652, RMSE Variance: 0.5452806822784588
```

Learning Rate: 0.1

```
RMSE: 8.168424487404579, MSE: 66.72315860643076

RMSE: 9.625700471314298, MSE: 92.6541095634603

For Validation –

Average RMSE: 8.897062479359438, RMSE Variance: 0.5309133233200101

For Training –

Average RMSE: 8.72207890775826, RMSE Variance: 0.5513249722704459
```

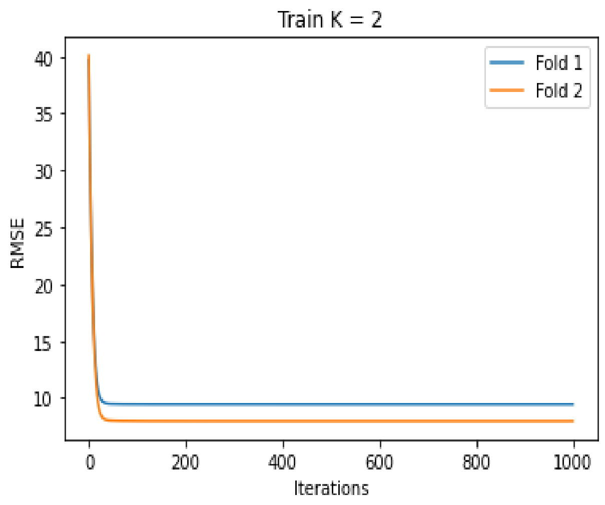
Learning Rate: 1

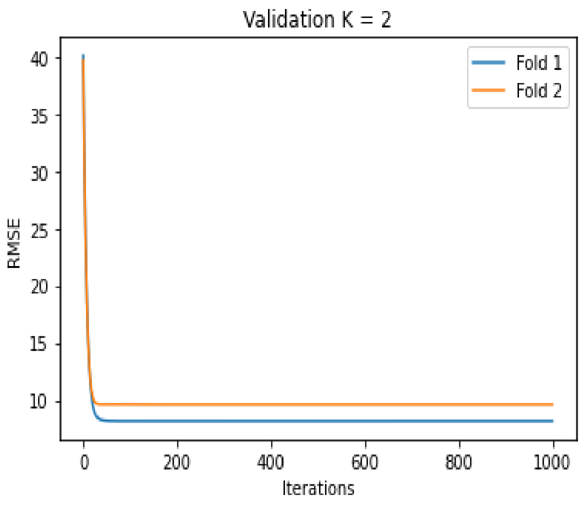
```
RMSE: 84.49480716852257, MSE: 7139.372438445812

RMSE: 177.42715001441903, MSE: 31480.39356223916

For Validation –

Average RMSE: 130.9609785914708, RMSE Variance: 2159.1050867068107

For Training –

Average RMSE: 136.48227609371898, RMSE Variance: 3632.430171015308
```

Learning Rate: 10

```
RMSE: 84579413.46265303, MSE: 7153677181686411.0

RMSE: 132325038.06850511, MSE: 1.7509915699831328e+16

For Validation –

Average RMSE: 108452225.76557907, RMSE Variance: 569911167250737.0

For Training –

Average RMSE: 111524875.97735351, RMSE Variance: 1212886059753292.5
```

As we can see a LR of 0.1 performs the best so we'll plot that and have a look





3 Folds

Learning Rate: 0.0001

```
RMSE: 36.73947356534616, MSE: 1349.7889178587693

RMSE: 36.01914827001337, MSE: 1297.379042097207

RMSE: 35.2821101665008, MSE: 1244.8272978010991

For Validation –

Average RMSE: 36.01357733395344, RMSE Variance: 0.35400019704664154

For Training –

Average RMSE: 36.00837256368507, RMSE Variance: 0.07546264074810426
```

Learning Rate: 0.001

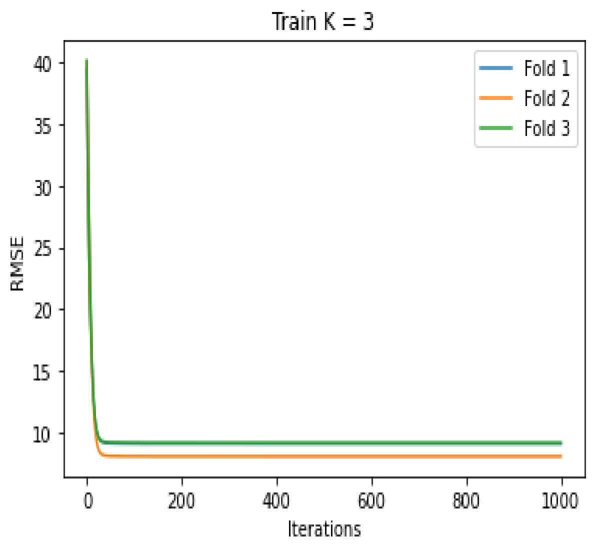
```
RMSE: 16.860427062435036, MSE: 284.27400072769177

RMSE: 17.319894362286966, MSE: 299.9787407207798

RMSE: 15.41354709832028, MSE: 237.57743415213753

For Validation –

Average RMSE: 16.53128950768076, RMSE Variance: 0.6598590801136908

For Training –

Average RMSE: 16.494837105934213, RMSE Variance: 0.04413064494311388
```

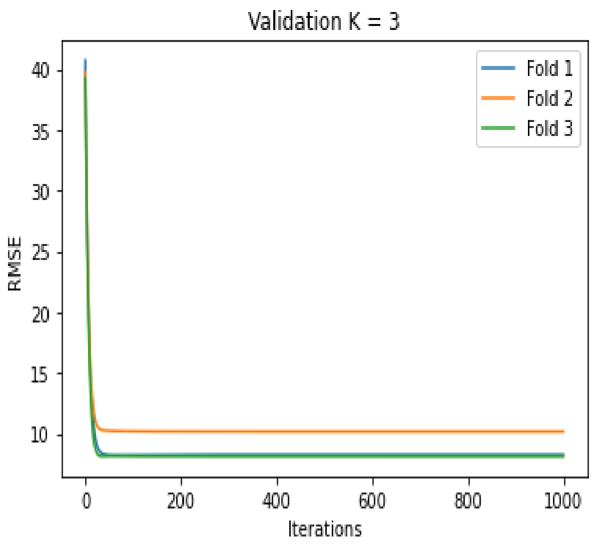
Learning Rate: 0.01

```
RMSE: 8.242068626301526, MSE: 67.93169524066393

RMSE: 10.204615502512498, MSE: 104.13417755411841

RMSE: 8.161705523576071, MSE: 66.61343705357214

For Validation –

Average RMSE: 8.869463217463364, RMSE Variance: 0.8923921835159042
```

```
For Training –

Average RMSE: 8.76597999848935, RMSE Variance: 0.2532398878078531

Learning Rate: 0.1

RMSE: 8.271321570215122, MSE: 68.41476051790593

RMSE: 10.183123066303592, MSE: 103.69599538348426

RMSE: 8.141517410338379, MSE: 66.28430574284293

For Validation –

Average RMSE: 8.865320682285699, RMSE Variance: 0.8711097483151744

For Training –

Average RMSE: 8.759110006687623, RMSE Variance: 0.2525193207292788

Learning Rate: 1

RMSE: 103.17415587379234, MSE: 10644.9064402696

RMSE: 127.9426682042143, MSE: 16369.32634721367

RMSE: 146.22767154381745, MSE: 21382.531925126557

For Validation –

Average RMSE: 125.78149854060803, RMSE Variance: 311.2695290822956

For Training –

Average RMSE: 127.44464405039685, RMSE Variance: 645.4994190573913

Learning Rate: 10

RMSE: 94557103.07437143, MSE: 8941045741817303.0

RMSE: 110768660.16529907, MSE: 1.2269696074815514e+16

RMSE: 116061654.17822073, MSE: 1.3470307570584902e+16

For Validation –

Average RMSE: 107129139.13929708, RMSE Variance: 83697343012366.38

For Training –

Average RMSE: 108118050.53500135, RMSE Variance: 224593803509508.03
```

As we can see a LR of 0.1 performs the best so we'll plot that and have a look





Note: All 3 curves are present in the plot however due to similarity between values it gets covered up by other curves above it but in certain places the blue line is visible

4 Folds

Learning Rate: 0.0001

RMSE: 36.247718644869835, MSE: 1313.8971069576442

RMSE: 36.09479003905462, MSE: 1302.8338679634367

RMSE: 36.129290180599426, MSE: 1305.3256089539582

RMSE: 35.57462579986026, MSE: 1265.554000800083

For Validation –

Average RMSE: 36.01160616609603, RMSE Variance: 0.06686750677457182

For Training –

Average RMSE: 36.00972583774011, RMSE Variance: 0.0033083854204209273

Learning Rate: 0.001

RMSE: 16.288022785777667, MSE: 265.2996862700125

RMSE: 16.66312980076215, MSE: 277.65989475704765

RMSE: 17.331599052044304, MSE: 300.38432570082296

RMSE: 15.840916748486539, MSE: 250.93464343248132

For Validation –

Average RMSE: 16.530917096767666, RMSE Variance: 0.2984174798856341

For Training –

Average RMSE: 16.496843437592865, RMSE Variance: 0.043591155809291134

Learning Rate: 0.01

```
RMSE: 7.008145318250513, MSE: 49.11410080171659
RMSE: 9.174003996217724, MSE: 84.16234932261877
RMSE: 10.81667607230247, MSE: 117.00048125312081
RMSE: 8.315327349782656, MSE: 69.14466893404345
For Validation –
Average RMSE: 8.828538184138342, RMSE Variance: 1.9123136090861883
For Training –
Average RMSE: 8.77155235020286, RMSE Variance: 0.22165884619557813
```

Learning Rate: 0.1

```
RMSE: 7.0322209679475005, MSE: 49.452131742040486
RMSE: 9.163139481159131, MSE: 83.96312515117724
RMSE: 10.82724997440253, MSE: 117.22934200819961
RMSE: 8.295665929025, MSE: 68.81807320598622
For Validation –
Average RMSE: 8.82956908813354, RMSE Variance: 1.9043777447275185
For Training –
Average RMSE: 8.764326266014095, RMSE Variance: 0.22346855468821888
```

Learning Rate: 1

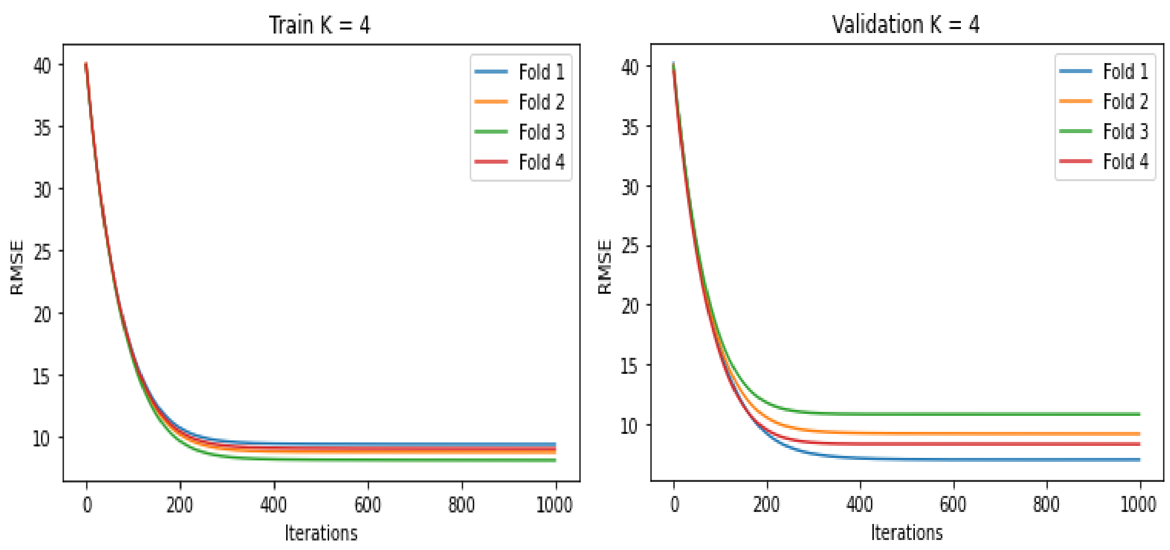
```
RMSE: 114.074140386032, MSE: 13012.909504812138
RMSE: 113.98316916898824, MSE: 12992.16285380619
RMSE: 137.04459222730262, MSE: 18781.22025874765
RMSE: 133.73107637480547, MSE: 17884.00078836405
For Validation –
Average RMSE: 124.70824453928209, RMSE Variance: 115.42709536312954
For Training –
Average RMSE: 126.59480569216929, RMSE Variance: 409.68870563037166
```

Learning Rate: 10

```
RMSE: 100834937.85492922, MSE: 1.016768469220744e+16
RMSE: 103833056.71478492, MSE: 1.0781303666735742e+16
RMSE: 110200759.45970513, MSE: 1.214420738549579e+16
RMSE: 111299088.80920602, MSE: 1.238748716975953e+16
For Validation –
Average RMSE: 106541960.70965631, RMSE Variance: 18981336691672.938
```

```
For Training -

Average RMSE: 107728389.05303583, RMSE Variance: 1359579935407785.97
```

As we can see a LR of 0.01 performs the best so we'll plot that and have a look



5 Folds

Learning Rate: 0.0001

```
RMSE: 36.002555641050144, MSE: 1296.1840126869115

RMSE: 37.18158163913651, MSE: 1382.4700131877732

RMSE: 35.32325273552617, MSE: 1247.7321838178568

RMSE: 36.32494644686623, MSE: 1319.5017343676993

RMSE: 35.23164782644089, MSE: 1241.2690085663571

For Validation -

Average RMSE: 36.01279685780399, RMSE Variance: 0.5098530038628459

For Training -

Average RMSE: 36.01050563258034, RMSE Variance: 0.033575947123506984
```

Learning Rate: 0.001

```
RMSE: 15.950620949442124, MSE: 254.42230867278198

RMSE: 17.688283351885264, MSE: 312.8753679365814

RMSE: 15.439037097855426, MSE: 238.36386650895608

RMSE: 17.745379984223877, MSE: 314.89851078449345

RMSE: 15.659314606027692, MSE: 245.2141339305522

For Validation -
```

```
Average RMSE: 16.496527197886877, RMSE Variance: 1.0194279760515688

For Training -

Average RMSE: 16.502311660962423, RMSE Variance: 0.06453214850134735
```

Learning Rate: 0.01

```
RMSE: 7.123179491114825, MSE: 50.73968606263885

RMSE: 9.260444732600645, MSE: 85.75583664555103

RMSE: 7.6541923493368405, MSE: 58.586660520646625

RMSE: 11.779289135520894, MSE: 138.75165253820057

RMSE: 7.709911545610658, MSE: 59.44273604114053

For Validation -

Average RMSE: 8.705403450836773, RMSE Variance: 2.8712651197947343

For Training -

Average RMSE: 8.785781139555151, RMSE Variance: 0.21633424581739652
```

Learning Rate: 0.1

```
RMSE: 7.138223284649192, MSE: 50.954231661507905

RMSE: 9.26195541669074, MSE: 85.78381814076691

RMSE: 7.6810808405147775, MSE: 58.9990028785232

RMSE: 11.749649897468515, MSE: 138.0542727130819

RMSE: 7.68764414330422, MSE: 59.09987247407967

For Validation -

Average RMSE: 8.703710716525489, RMSE Variance: 2.823659336631281

For Training -

Average RMSE: 8.778941029351845, RMSE Variance: 0.21652764713063033
```

Learning Rate: 1

```
RMSE: 117.71841991951976, MSE: 13857.626388348386

RMSE: 111.74338539535451, MSE: 12486.584179614727

RMSE: 130.16518411101674, MSE: 16942.975154654883

RMSE: 128.58805937700998, MSE: 16534.889014345445

RMSE: 134.6723241294322, MSE: 18136.63488642285

For Validation -

Average RMSE: 124.57747458646664, RMSE Variance: 72.19475033551696

For Training -

Average RMSE: 126.2045120198442, RMSE Variance: 230.68650614413235
```

Learning Rate: 10

RMSE: 104184210.5142754, MSE: 1.0854349720482852e+16

RMSE: 100176367.54662058, MSE: 1.0035304614835616e+16

RMSE: 111278854.62858762, MSE: 1.2382983487450336e+16

RMSE: 103608006.5948362, MSE: 1.0734619030555622e+16

RMSE: 113182584.1159316, MSE: 1.2810297347159932e+16

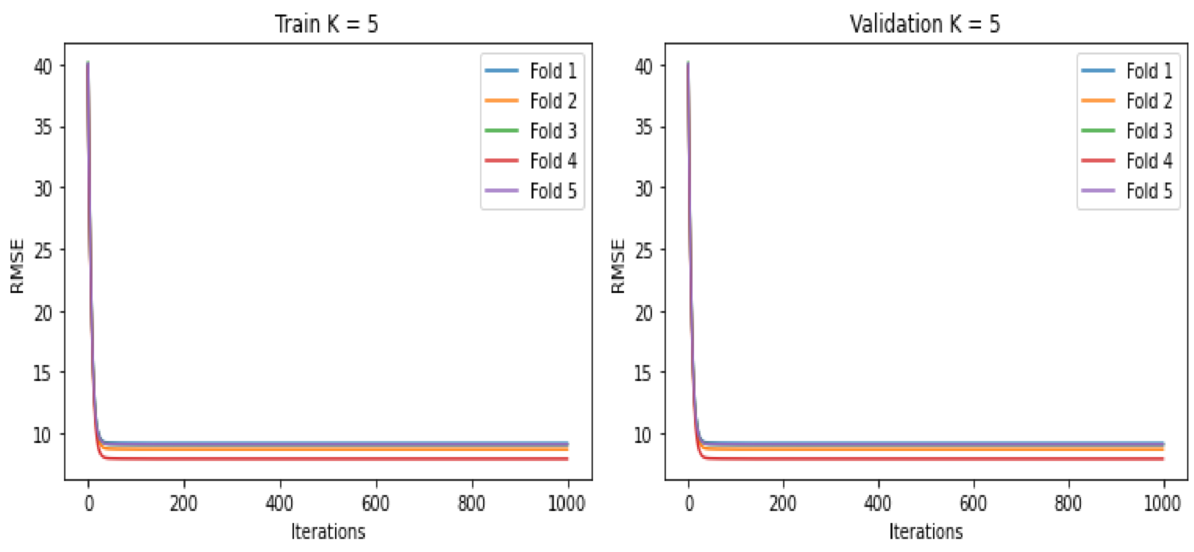
For Validation –

Average RMSE: 106486004.68005028, RMSE Variance: 24241647377181.82

For Training –

Average RMSE: 107607305.64109787, RMSE Variance: 77049656122183.4

<u>As we can see a LR of 0.1 performs the best so we'll plot that and have a look</u>



	K = 2	K = 3	K = 4	K = 5
Average Train RMSE	8.897062479359438	8.865320682285699	8.759110006687623	8.703710716525489
Average Val RMSE	8.72207890775826	8.759110006687623	8.764326266014095	8.778941029351845
Best Learning Rate	0.1	0.1	0.01	0.1

In general, we see that in most cases a LR of 0.1 is very good. Also, K = 5 gives us the best results in terms of average values, hence we stick with K = 5 for all subsequent parts.

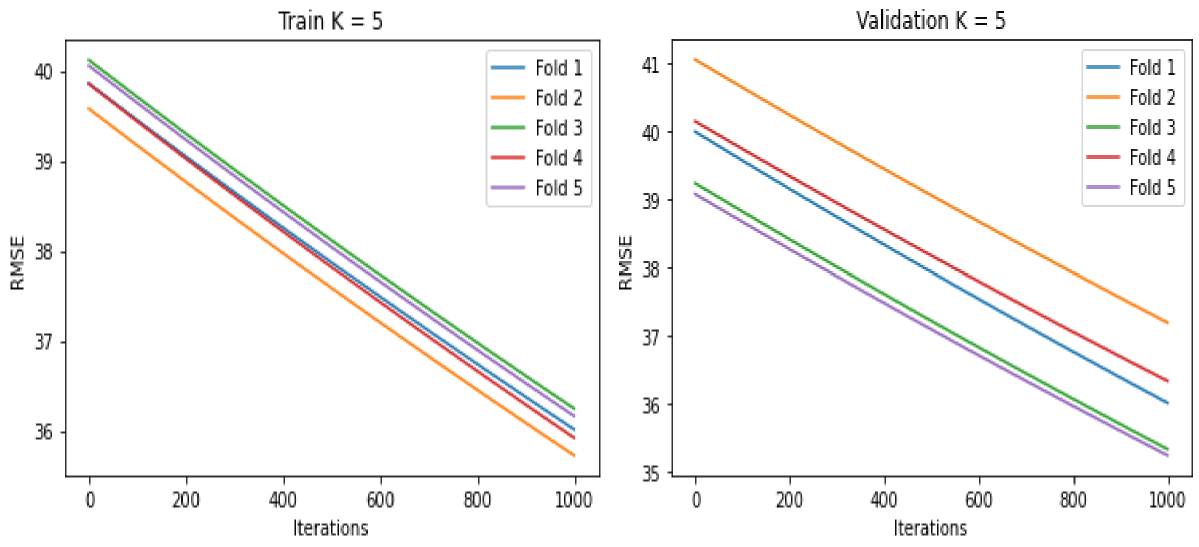
b)

All Plots with K = 5

<u>**Learning Rate = 0.0001**</u>

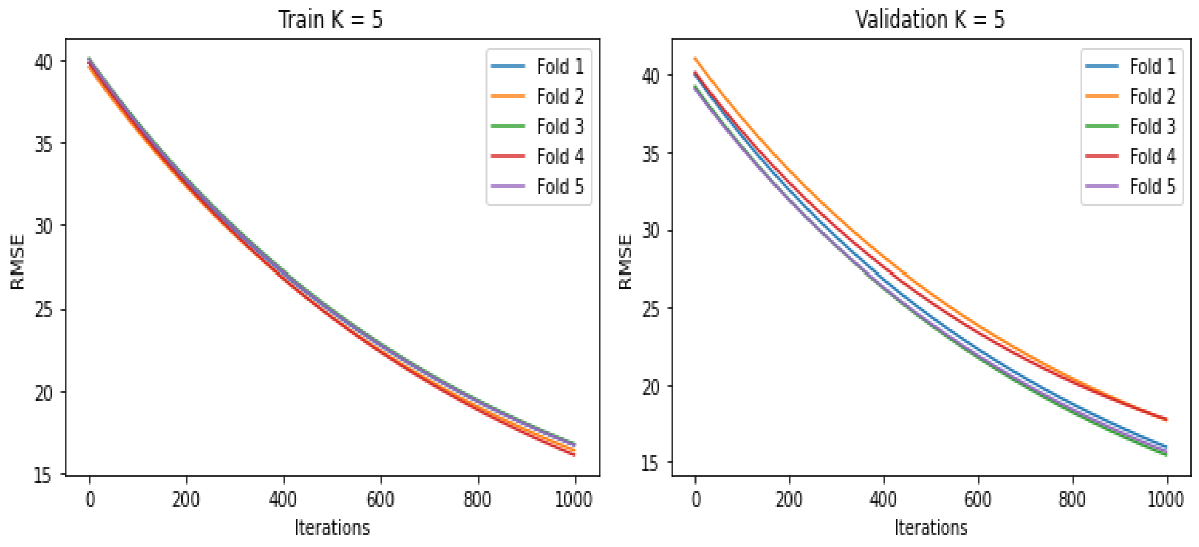
Average Val RMSE: 36.01279685780399, RMSE Variance: 0.5098530038628459

```
Average Train RMSE: 36.01050563258034, RMSE Variance: 0.033575947123506984
```



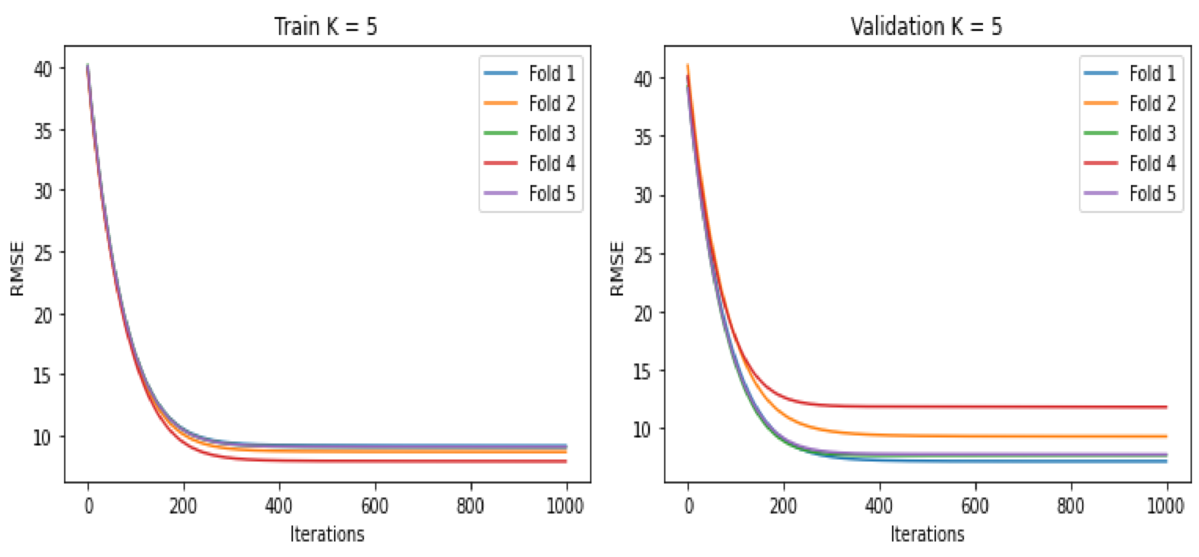
Learning Rate = 0.001

```
Average Val RMSE: 16.496527197886877, RMSE Variance: 1.0194279760515688
Average Train RMSE: 16.502311660962423, RMSE Variance: 0.06453214850134735
```



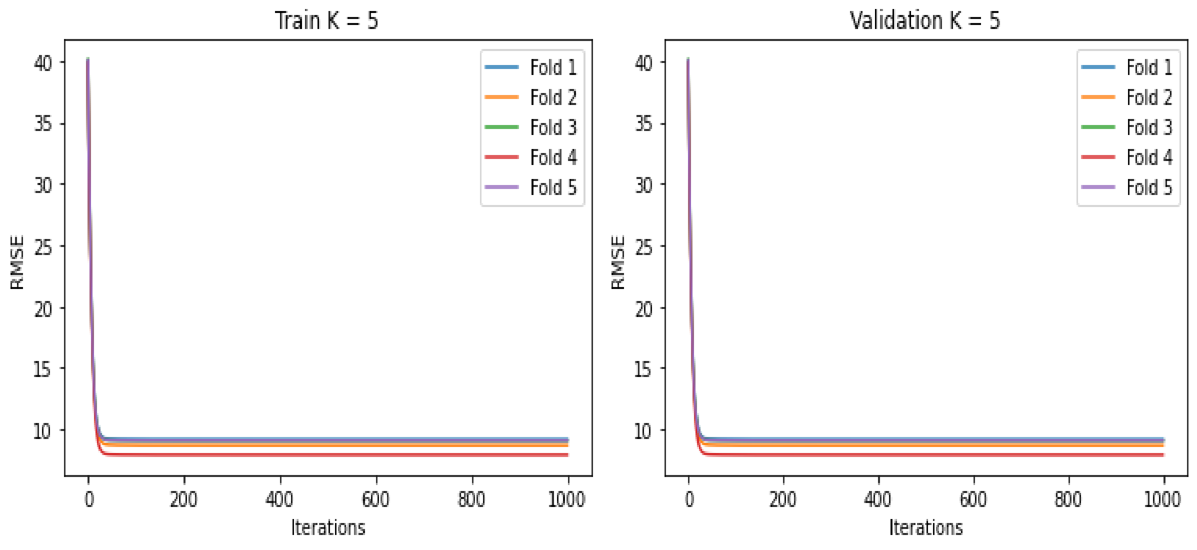
Learning Rate = 0.01

```
Average Val RMSE: 8.705403450836773, RMSE Variance: 2.8712651197947343

Average Train RMSE: 8.785781139555151, RMSE Variance: 0.21633424581739652
```



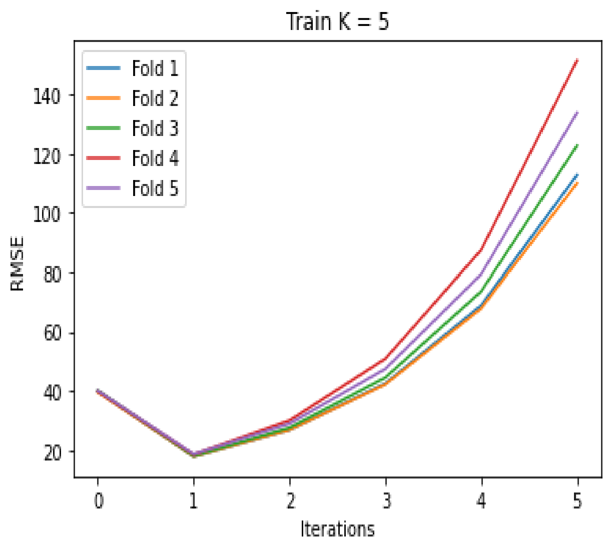
Learning Rate = 0.1

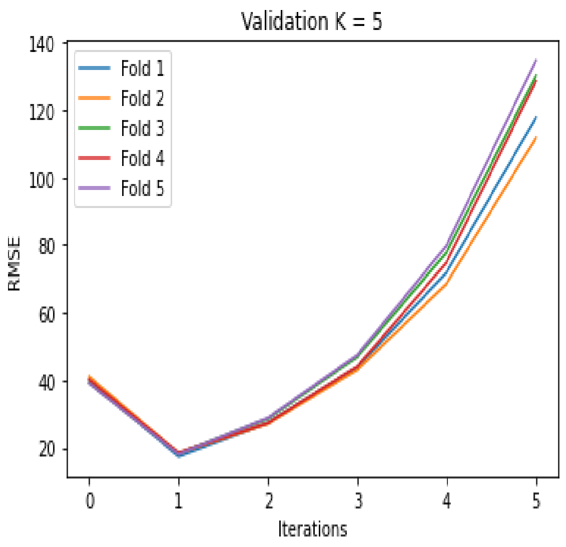
```
Average Val RMSE: 8.703710716525489, RMSE Variance: 2.823659336631281
Average Train RMSE: 8.778941029351845, RMSE Variance: 0.21652764713063033
```



Learning Rate = 1

```
Average Val RMSE: 124.57747458646664, RMSE Variance: 72.19475033551696
Average Train RMSE: 126.2045120198442, RMSE Variance: 230.68650614413235
```

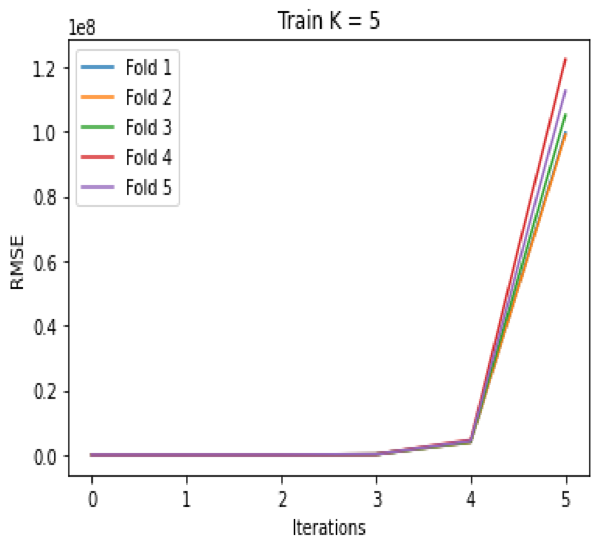


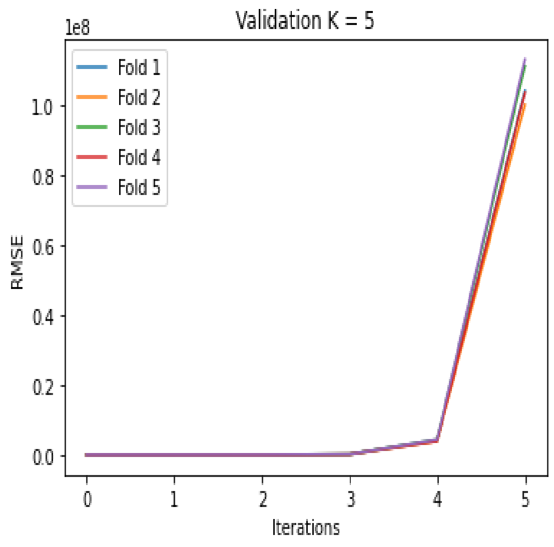


Learning Rate = 10

Average Val RMSE: 106486004.68005028, RMSE Variance: 24241647377181.82

Average Train RMSE: 107607305.64109787, RMSE Variance: 77049656122183.4





As we can see Learning rates 1 and 10 do not converge at all. Smaller learning rate like 0.0001 also do not converge well quickly. The sweet spot seems to be at learning rates 0.1 and 0.01.

c)

All analysis henceforth uses K = 5 and Learning Rate = 0.1 as shown before these 2 values tend to work the best both in isolation (in most cases) and even together

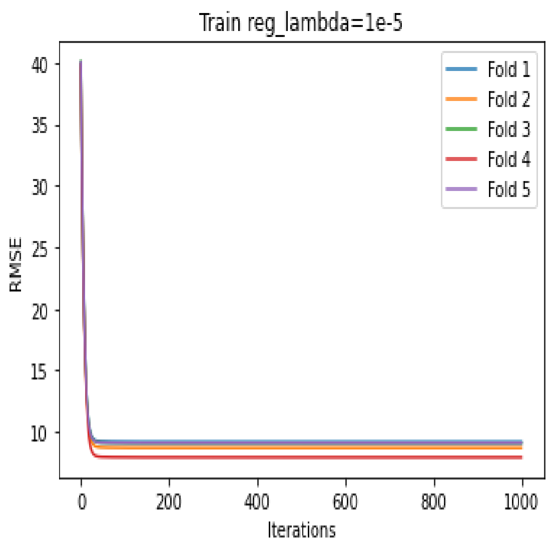
<u>L1 (Lasso) Regularization</u>

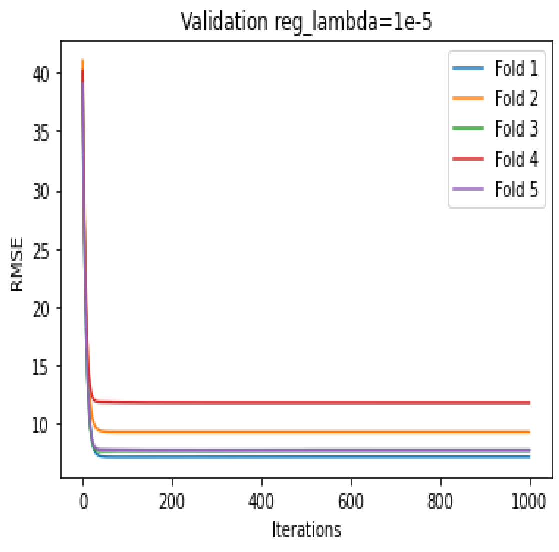
Regularization Parameter: 1e-05

For Validation –

Average RMSE: 8.703709672191906, RMSE Variance: 2.823658648322029

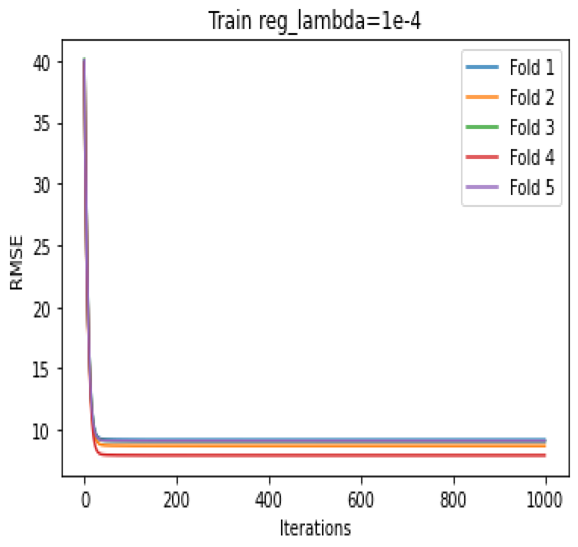
```
For Training -

Average RMSE: 8.778941029421645, RMSE Variance: 0.21652764714807127
```

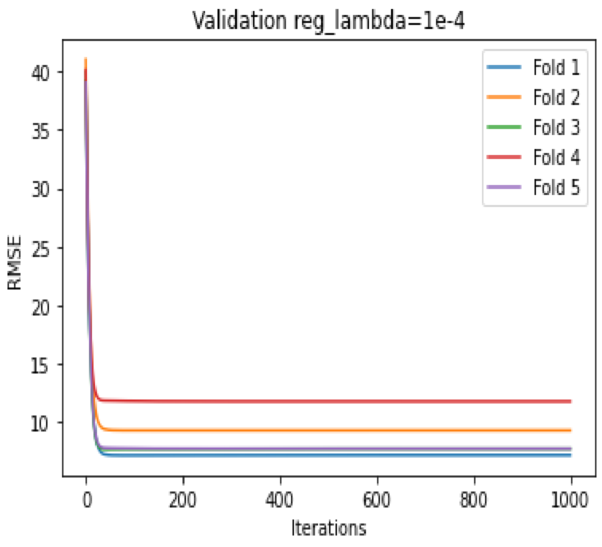




Regularization Parameter: 0.0001

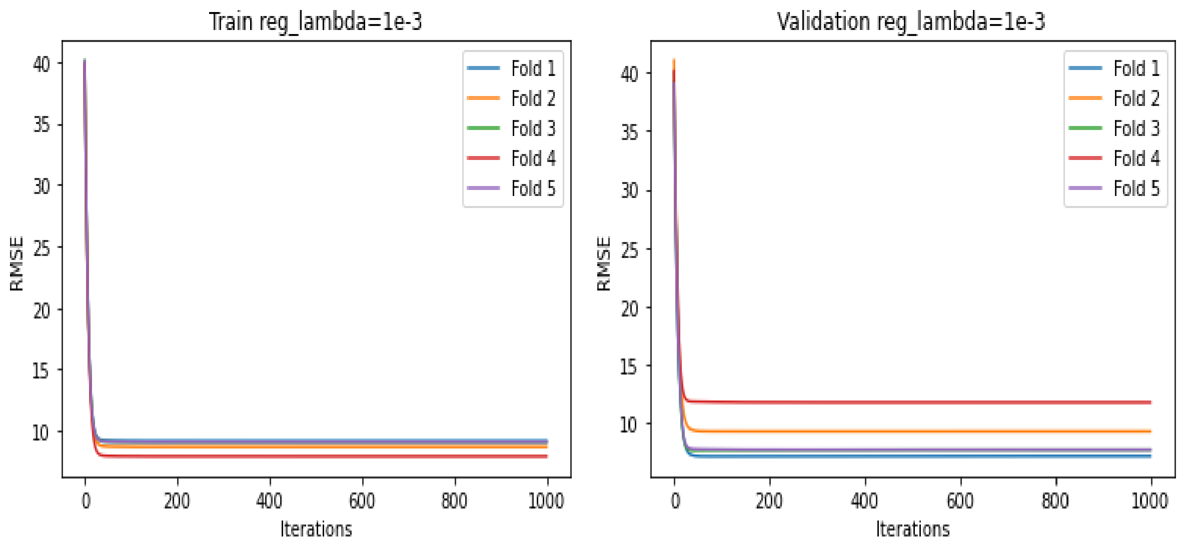
```
For Validation -

Average RMSE: 8.703700280194044, RMSE Variance: 2.8236524441959454

For Training -

Average RMSE: 8.778941036234235, RMSE Variance: 0.2165276488039587
```





Regularization Parameter: 0.001

```
For Validation -

Average RMSE: 8.7036070536675, RMSE Variance: 2.8235894778201622

For Training -

Average RMSE: 8.77894171661394, RMSE Variance: 0.21652781375555175
```



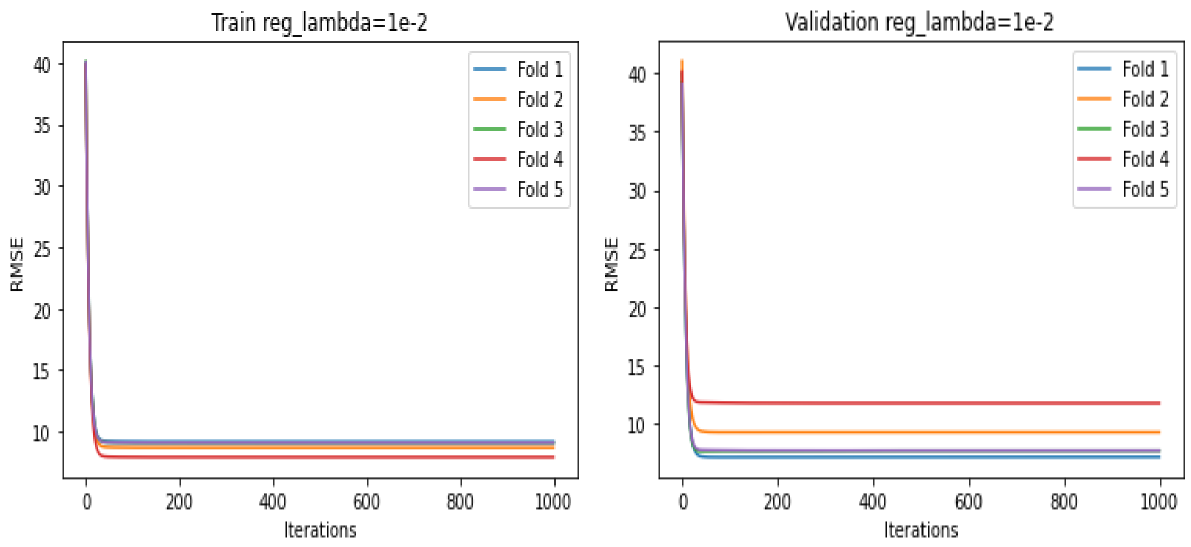
Regularization Parameter: 0.01

For Validation –

Average RMSE: 8.702744151366518, RMSE Variance: 2.822867132079941

For Training –

Average RMSE: 8.77900974551358, RMSE Variance: 0.21654430284622145



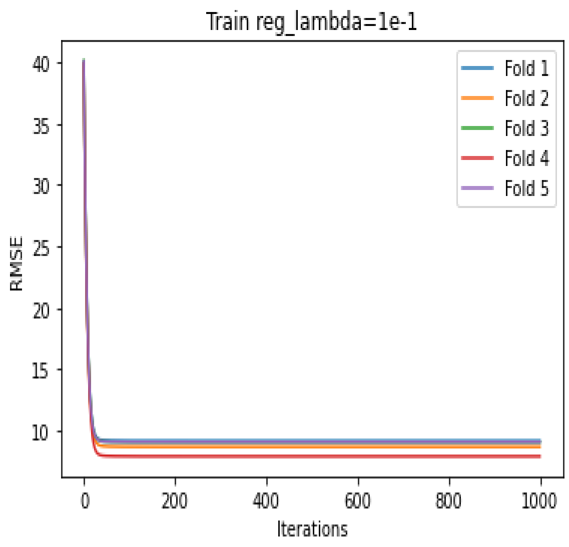
Regularization Parameter: 0.1

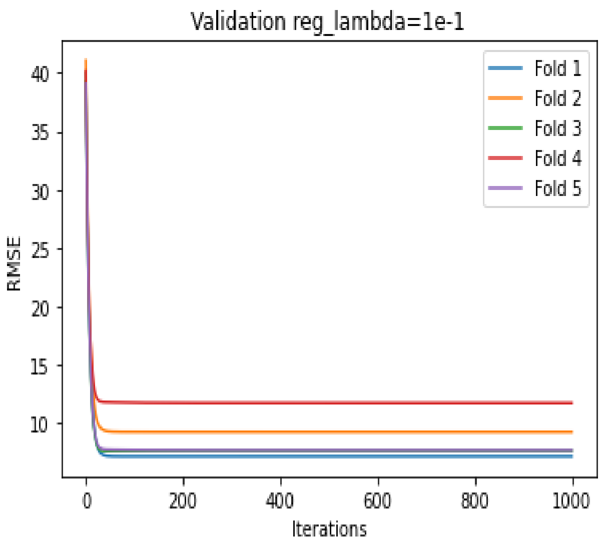
For Validation –

Average RMSE: 8.701422313685669, RMSE Variance: 2.8049388848522137

For Training –

Average RMSE: 8.783382032860445, RMSE Variance: 0.21721293490135185





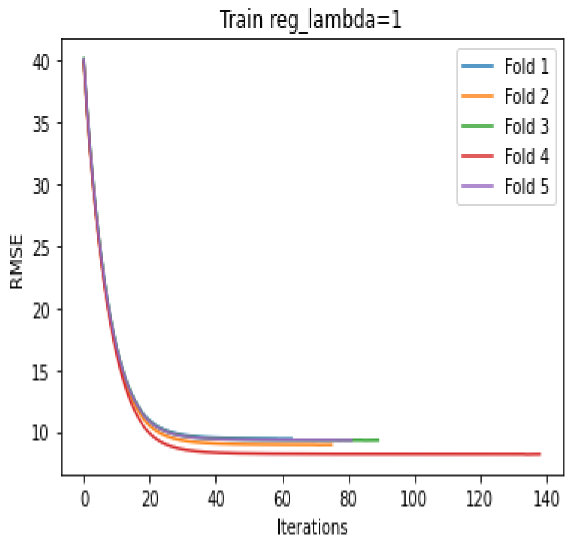
Regularization Parameter: 1

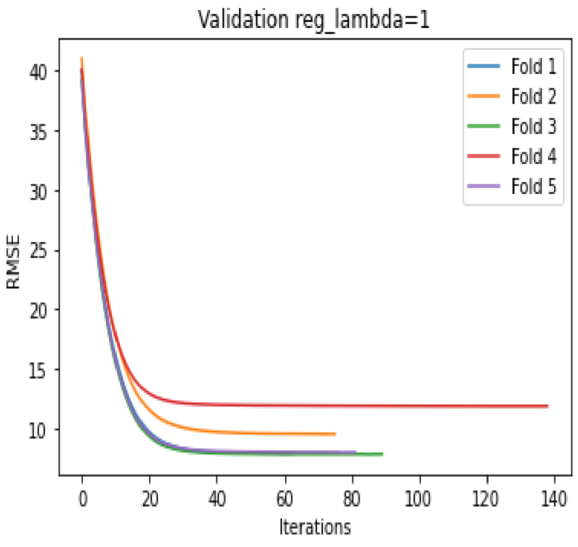
For Validation –

Average RMSE: 8.98601974318469, RMSE Variance: 2.428268730240807

For Training –

Average RMSE: 9.045598602103137, RMSE Variance: 0.20721821748496586





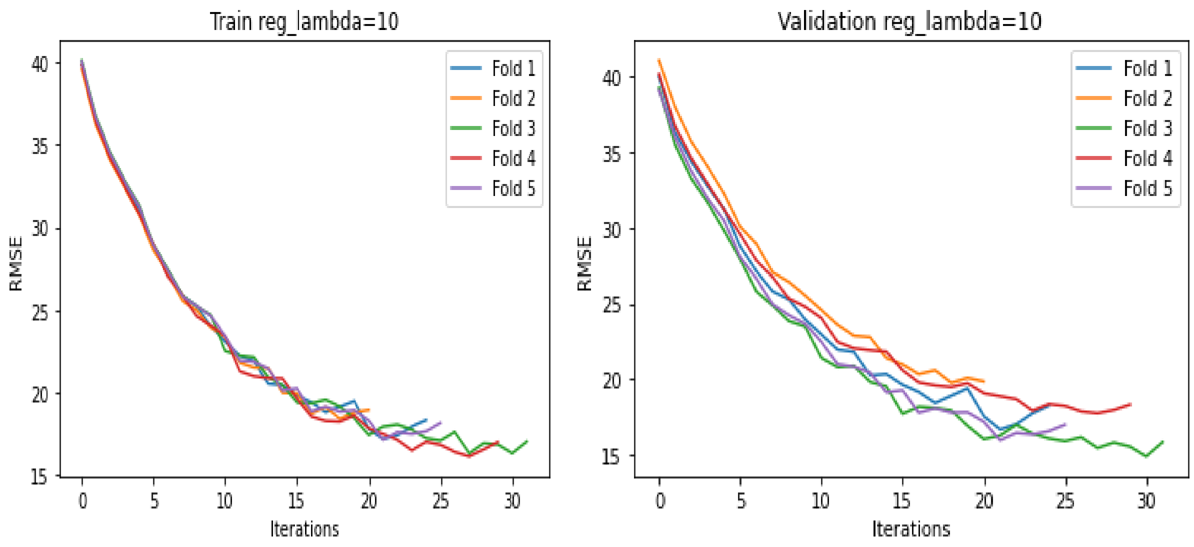
Regularization Parameter: 10

For Validation –

Average RMSE: 17.855742613020617, RMSE Variance: 1.8362954125422426

For Training –

Average RMSE: 17.8863240921771, RMSE Variance: 0.585577421229748



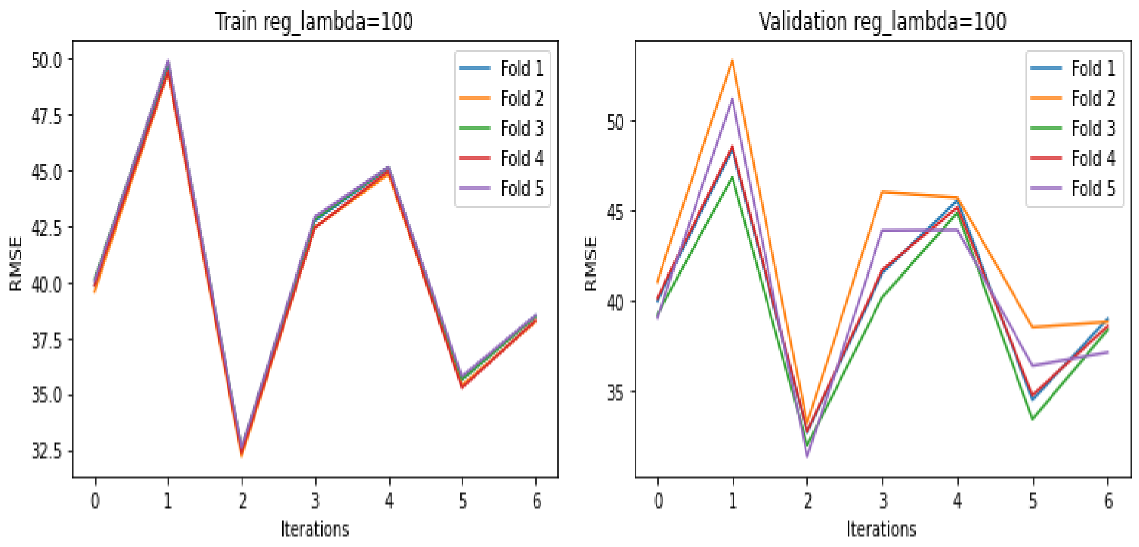
Regularization Parameter: 100

For Validation –

Average RMSE: 38.39220548030218, RMSE Variance: 0.43090682260927016

For Training –

Average RMSE: 38.387989172845806, RMSE Variance: 0.013262159706640225



Note: In some cases the curves stop at different points due to early stopping as different models converge at different paces

<u>L2 (Ridge) Regularization</u>

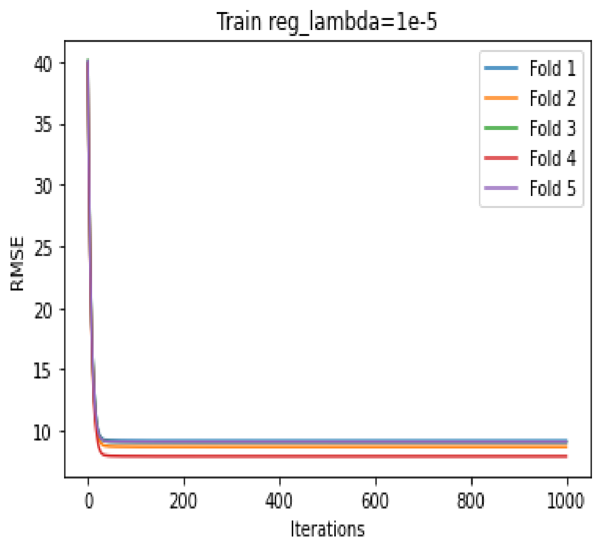
Regularization Parameter: 1e-05

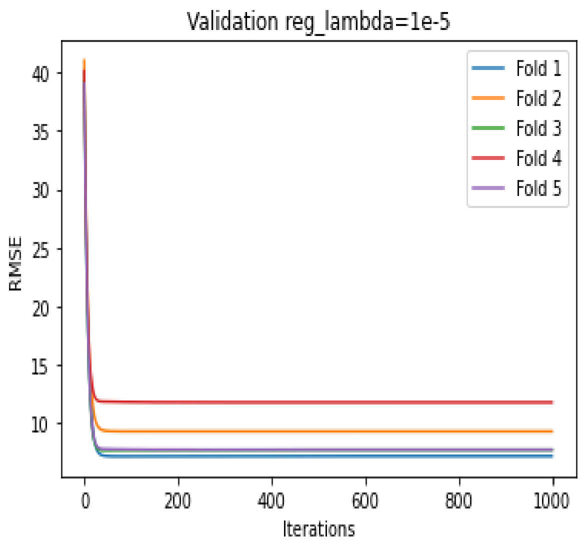
For Validation –

Average RMSE: 8.70370886792957, RMSE Variance: 2.8236234615944715

For Training –

Average RMSE: 8.778941064870889, RMSE Variance: 0.21652764538583974





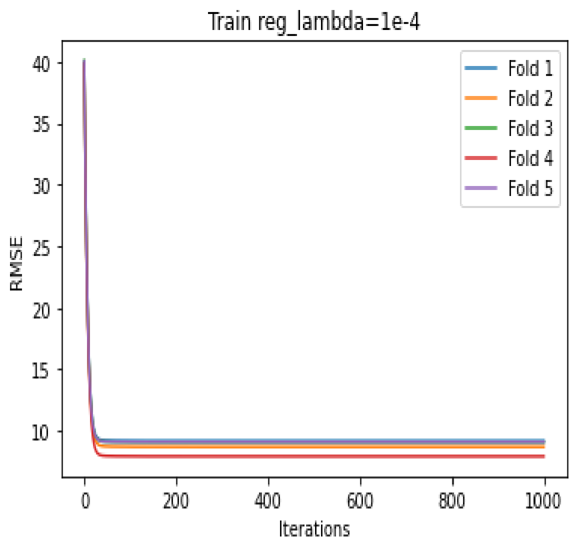
Regularization Parameter: 0.0001

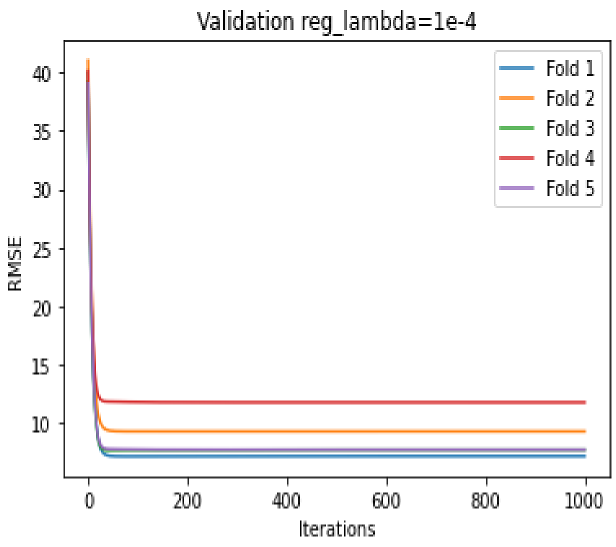
For Validation –

Average RMSE: 8.703695553671981, RMSE Variance: 2.823299404492735

For Training –

Average RMSE: 8.778944578429293, RMSE Variance: 0.21652747314345916





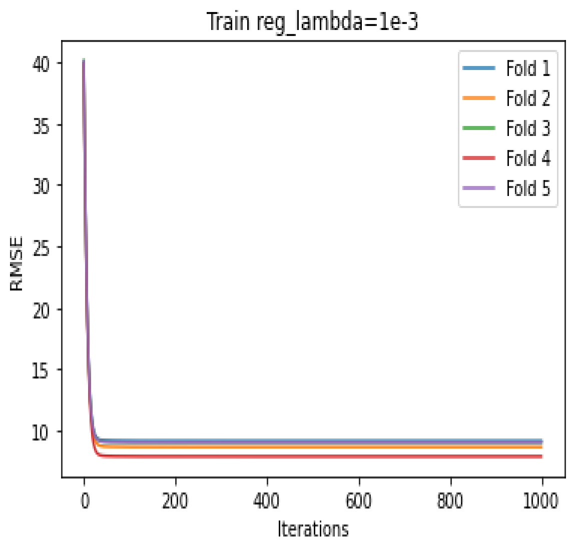
Regularization Parameter: 0.001

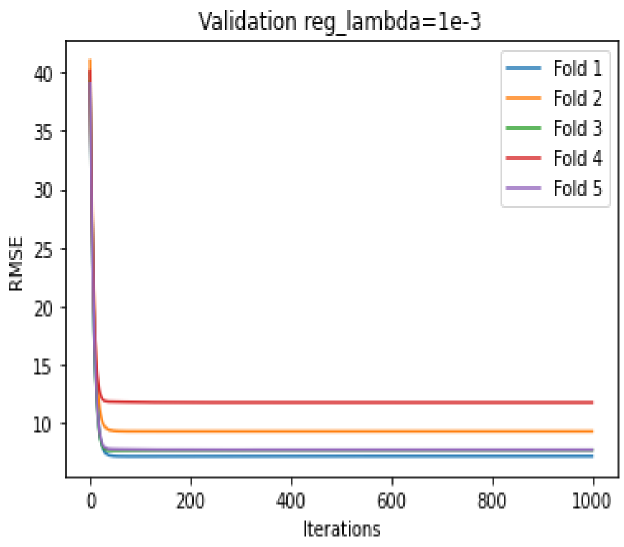
For Validation –

Average RMSE: 8.703889672477846, RMSE Variance: 2.81994233619207

For Training –

Average RMSE: 8.779294290608846, RMSE Variance: 0.21651031524722947





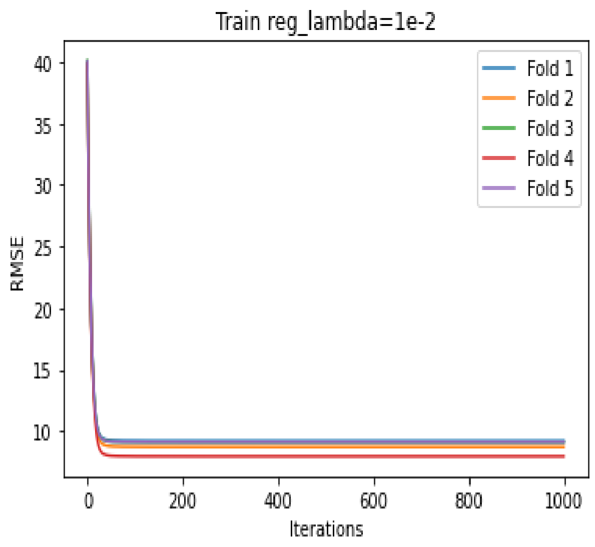
Regularization Parameter: 0.01

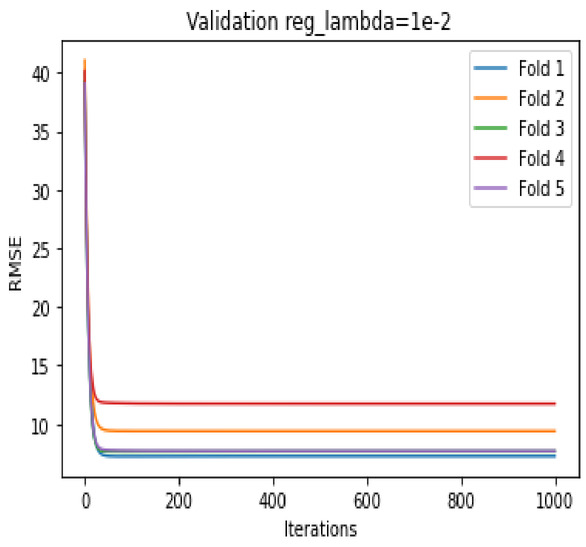
For Validation –

Average RMSE: 8.736885225993767, RMSE Variance: 2.7752938787261368

For Training –

Average RMSE: 8.812680651250712, RMSE Variance: 0.21486662361581602





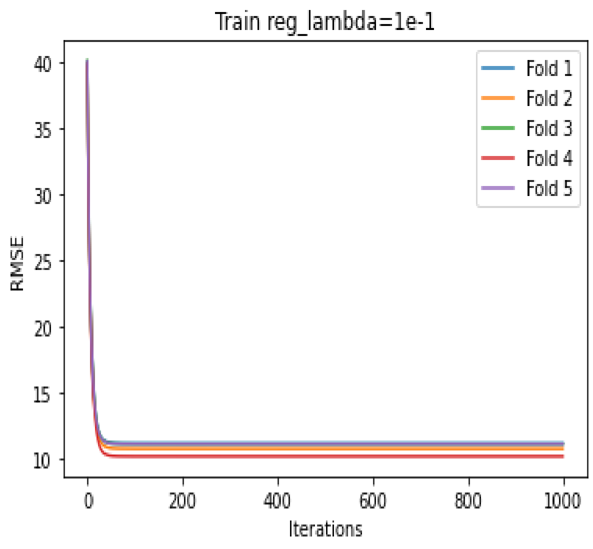
Regularization Parameter: 0.1

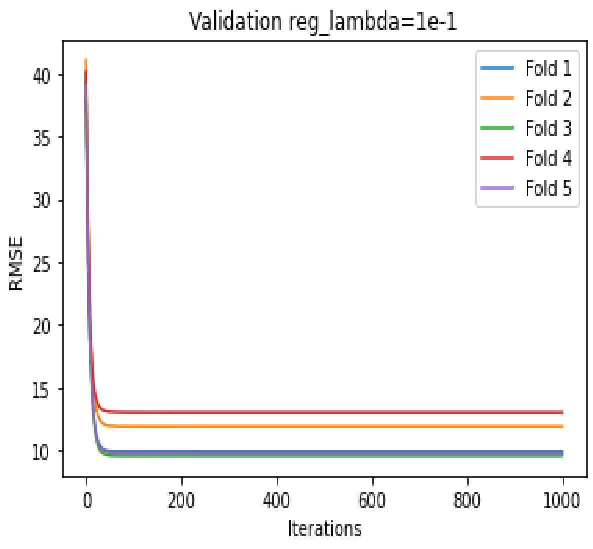
For Validation –

Average RMSE: 10.859364207752135, RMSE Variance: 1.9071935836059317

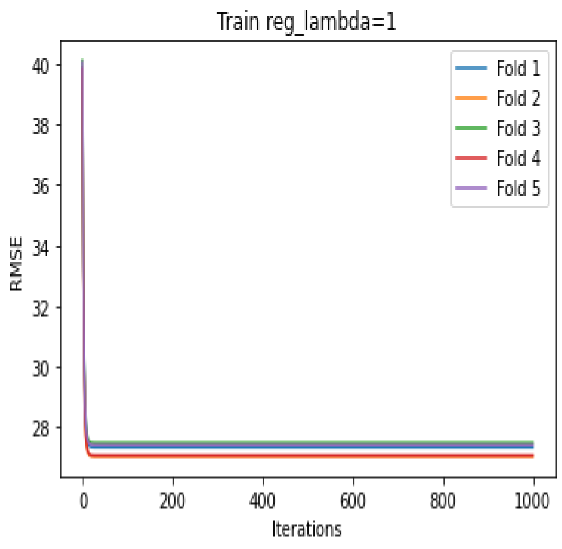
For Training –

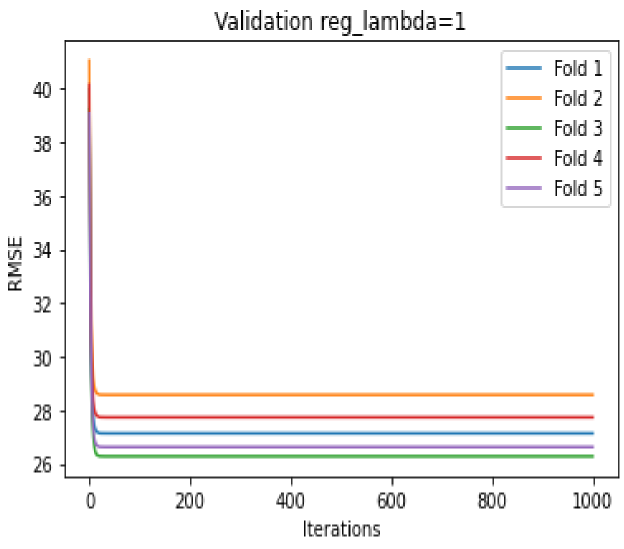
Average RMSE: 10.897660996629977, RMSE Variance: 0.1392299859524072



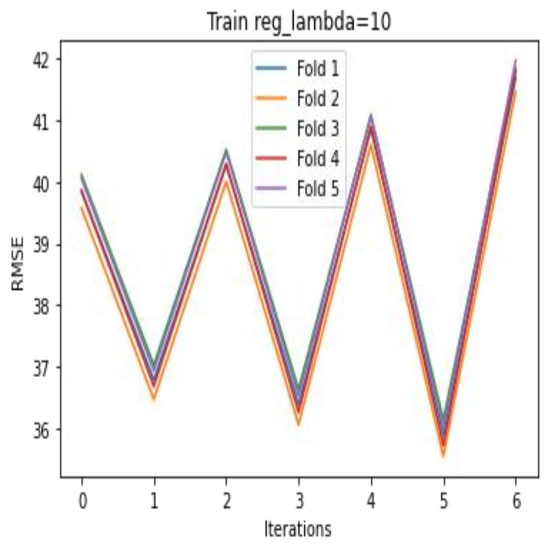


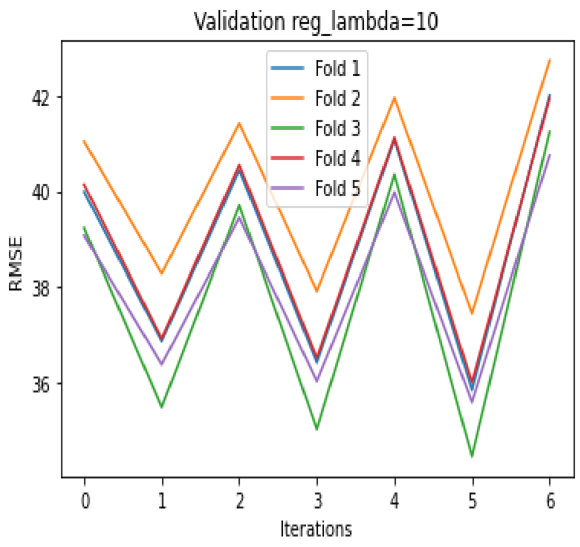
```
Regularization Parameter: 1

For Validation –

Average RMSE: 27.26579580232262, RMSE Variance: 0.6681218566017744

For Training –

Average RMSE: 27.26360070220097, RMSE Variance: 0.033574976267000047
```





```
Regularization Parameter: 10

For Validation –

Average RMSE: 41.74617907655433, RMSE Variance: 0.47248764227836026

For Training –

Average RMSE: 41.75601152011689, RMSE Variance: 0.02942145171798437
```





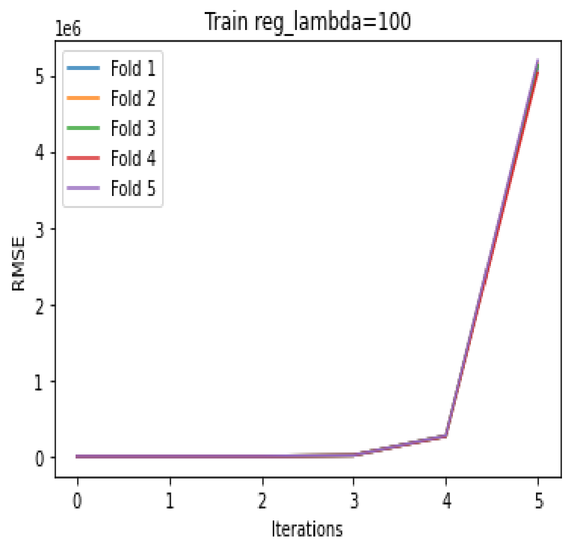
Regularization Parameter: 100

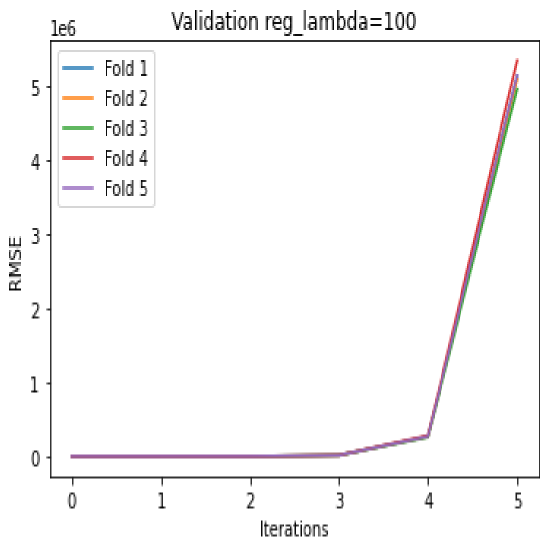
For Validation –

Average RMSE: 5127228.701350766, RMSE Variance: 15371391278.132935

For Training –

Average RMSE: 5118261.509663762, RMSE Variance: 2870607127.502921





Hence **Regularization Parameter: 0.01** works the best for Ridge while 0.1 works best for Lasso Regression. Overall, we see what we would expect that is as we begin to weigh the regularization loss more our model starts to fail to converge as it might greedily try to just minimize weights to minimize overall loss instead of minimizing the MSE. In intermediate values of the regularization parameter, we see a balance between the two. Since the model already converged well even without regularization, regularization does not help a lot and only offers decimal level improvement. Between Lasso and Ridge, Ridge uses square of weights which increases much faster and hence as evident from the values the model totally overshoots the minima by a large amount while in the case of Lasso it still overshoots but by a smaller amount

Since we’ve used early stopping everywhere, for parameter values 1 and 10 we see the model is unstable during training and hence stops after few iterations

d)

For K = 5

Using Analytical Solution

Fold 1

RMSE: 7.15465365030286, MSE: 51.18906885579204

Fold 2

RMSE: 9.266839756448137, MSE: 85.87431907168776

Fold 3

RMSE: 7.695705437572973, MSE: 59.22388218189022

Fold 4

RMSE: 11.724728270732339, MSE: 137.46925302251015

Fold 5

RMSE: 7.693396778376939, MSE: 59.18835398954066

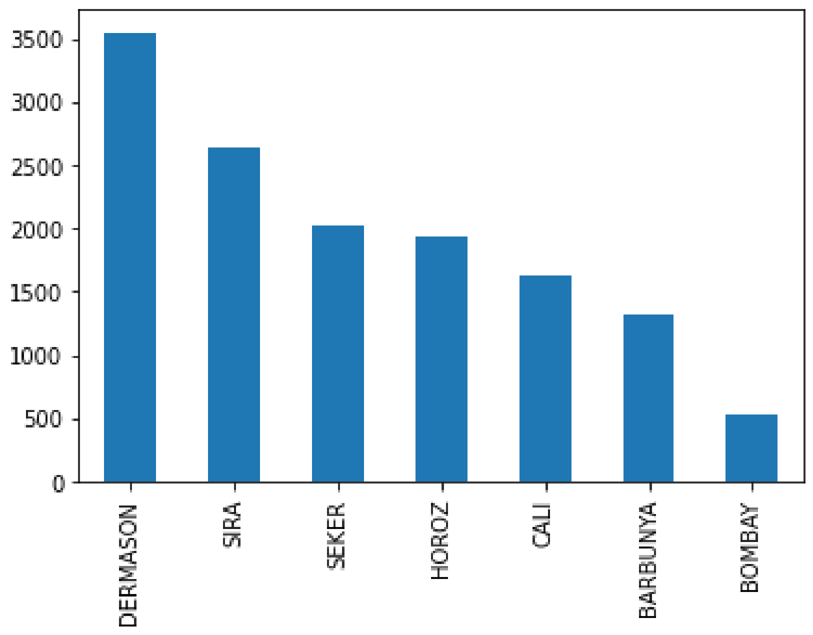
Average

RMSE: 8.707064778686648, Average MSE: 78.58897542428416

Section C

a)

Class Distribution:



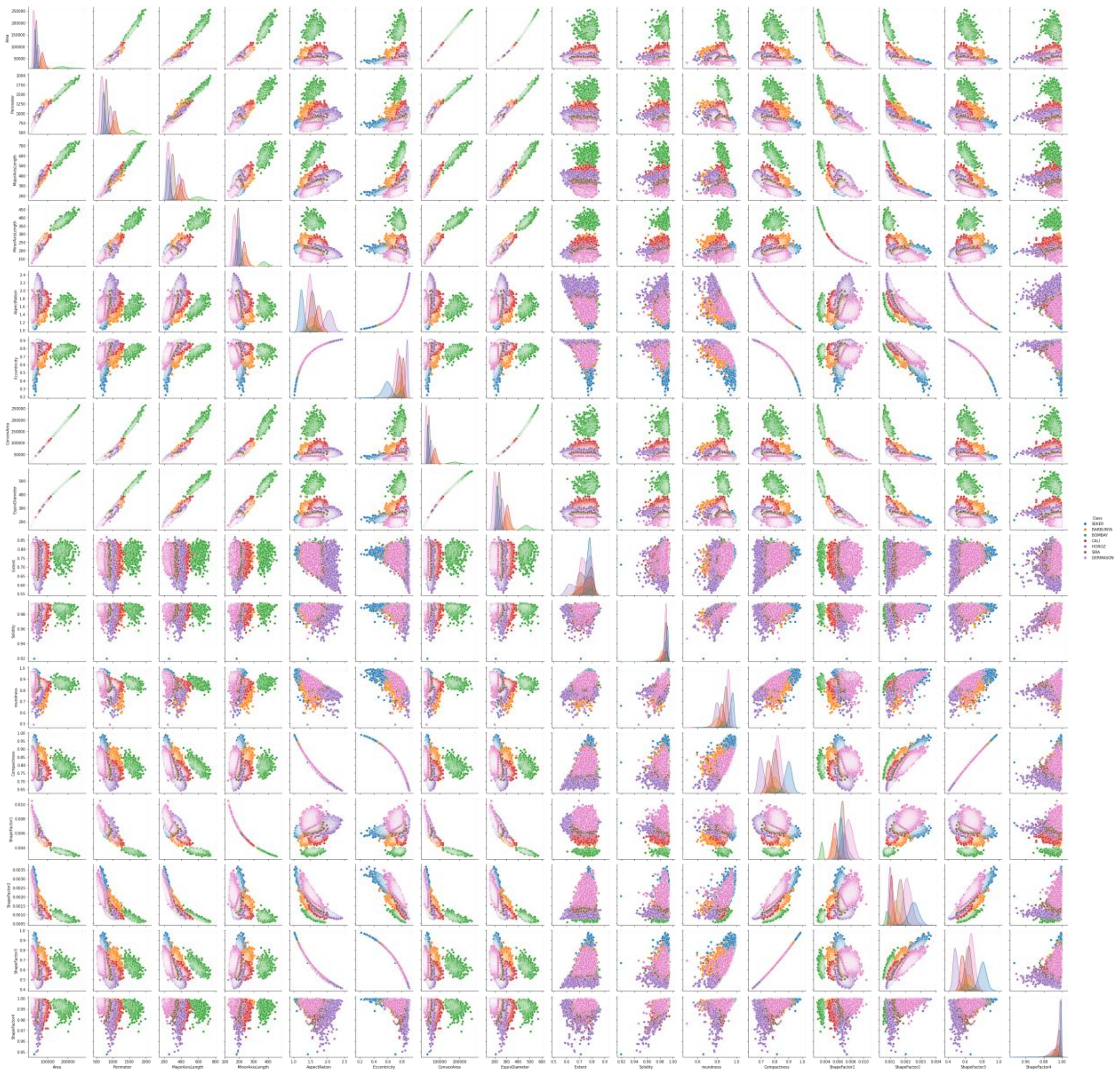
This shows that the classes are significantly imbalanced with occurrences of BOMBAY (least frequent class) being approximately 1/7th the occurrences of DERMASON (most frequent class).

This also indicates that any technique which uses this information of class distribution such as naïve bayes to obtain a prior will suffer due to skewed priors

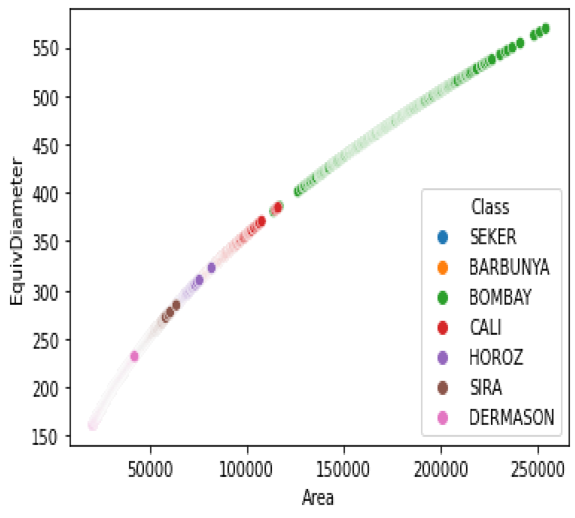
b)

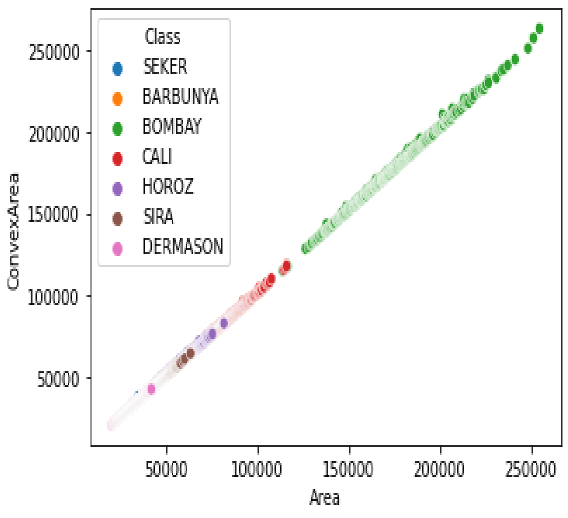
Exploratory Data Analysis

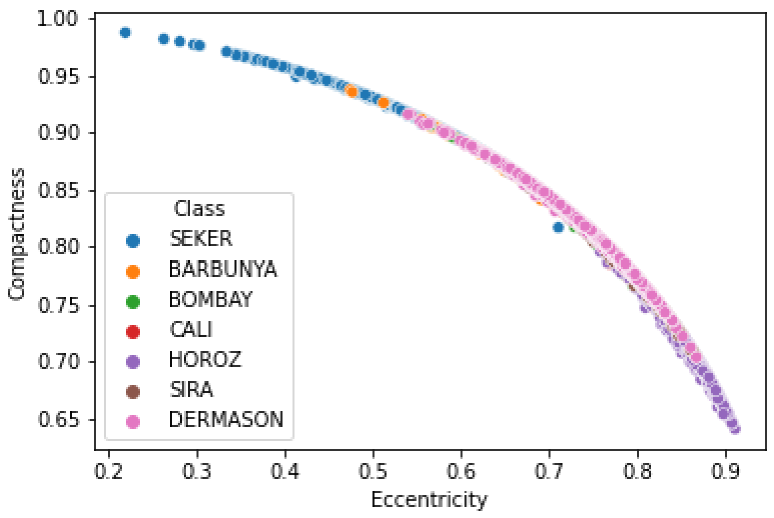
First step was to plot a pair plot over the entire dataset to visualize relations between all possible pair of features in the dataset and then choose some interesting ones to analyse. The pair plot is added below due to it's large size it's blurred (renders well if zoomed in) however the main analysis are done on individual plots which are plotted clearly, this simply acts as a starting point.



Let's look at some plots where we can see some sort of a relation clearly







This is to be expected in the plots between ConvexArea and Area or EquivDiameter and Area or Eccentricity and Compactness as if we look at the definitions of these features listed here:

- Area (A): The area of a bean zone and the number of pixels within its boundaries.

- Convex area (C): Number of pixels in the smallest convex polygon that can contain the area of a bean seed.

- Equivalent diameter (Ed): The diameter of a circle having the same area as a bean seed area.

- Eccentricity (Ec): Eccentricity of the ellipse having the same moments as the region.

- Compactness (CO): Measures the roundness of an object: Ed/L

We observe that Equivalent Diameter and Area are of course related as Equivalent Diameter is computed based on the area and if we peek at the curve it seems like the positive y-axis half of a parabola of the form $y^2 = 4ax$ and that is to be expected as Area and Diameter are related as -

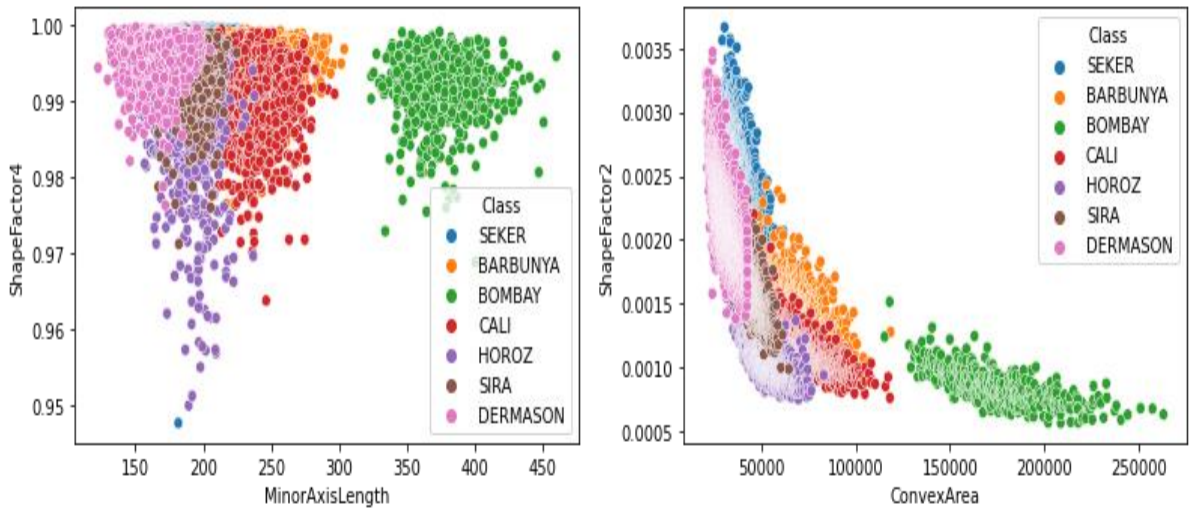
$$Area = \frac{\pi * Diameter^2}{4}$$

Similarly, we note that ConvexArea and Area also share a relation and their definitions also point towards the same conclusion as when there are large number of datapoints we could potentially have a quite good fitting Convex Polygon

And also, Eccentricity and Compactness are also related as Eccentricity of an ellipse gives information about the shape and so does the compactness

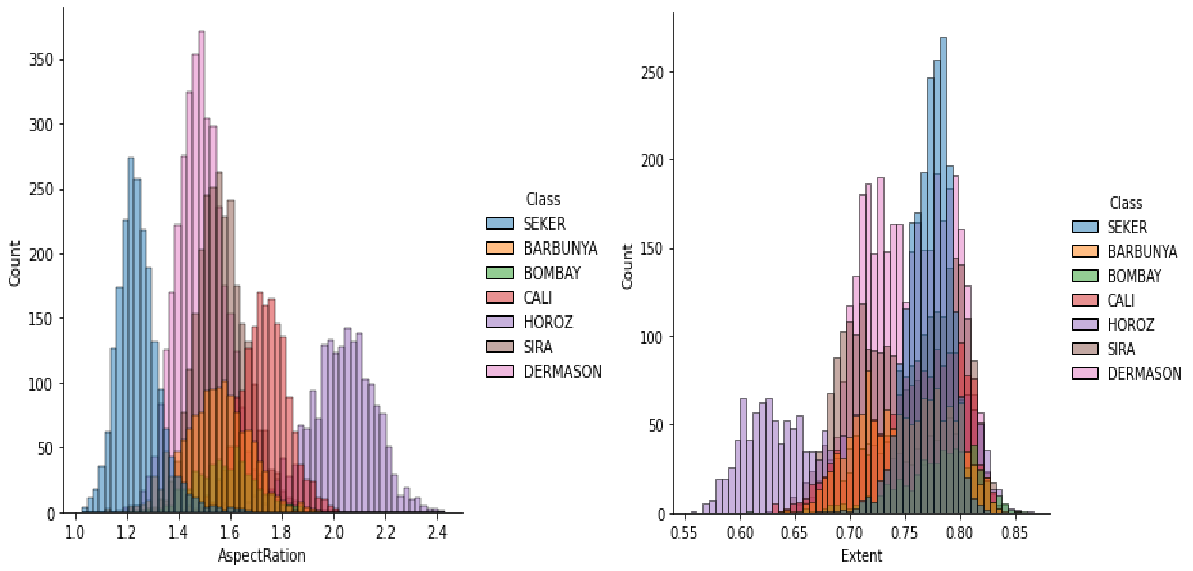
Also, what is evident is that for Bombay Beans the sizes are generally larger while Dermason Beans seem to be smaller in size. A simple google search to look up images also validates our observations.

Now let's look at some plots where there seem to be clear distinct clusters which are separated

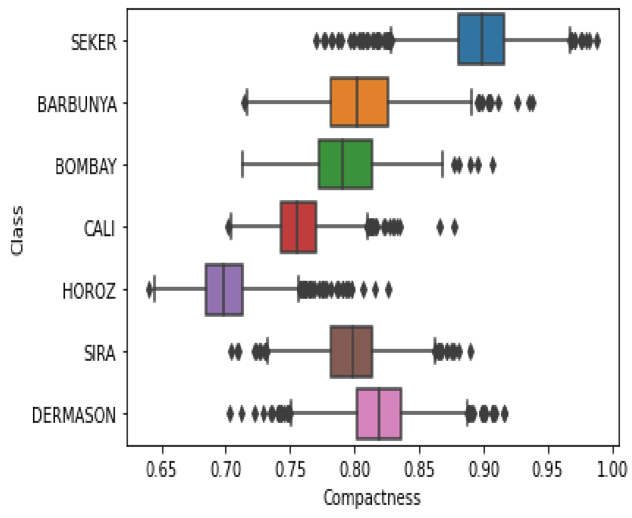


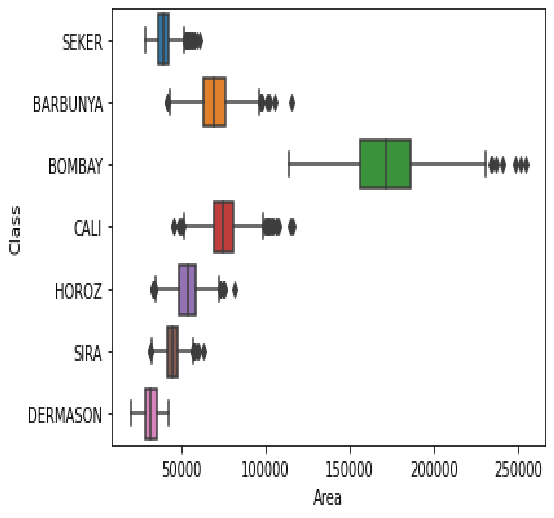
These plots clearly show BOMBAY as a distinct cluster and even amongst the remaining barring some outliers some sort of boundaries can be sketched which can provide reasonable accuracy in classifying amongst classes. This just again proves the point of BOMBAY being the largest and also shows that ShapeFactor2 is related somehow with ConvexArea where in the larger the Area the smaller the Factor while ShapeFactor4 is also related to MinorAxisLength but in the opposite way

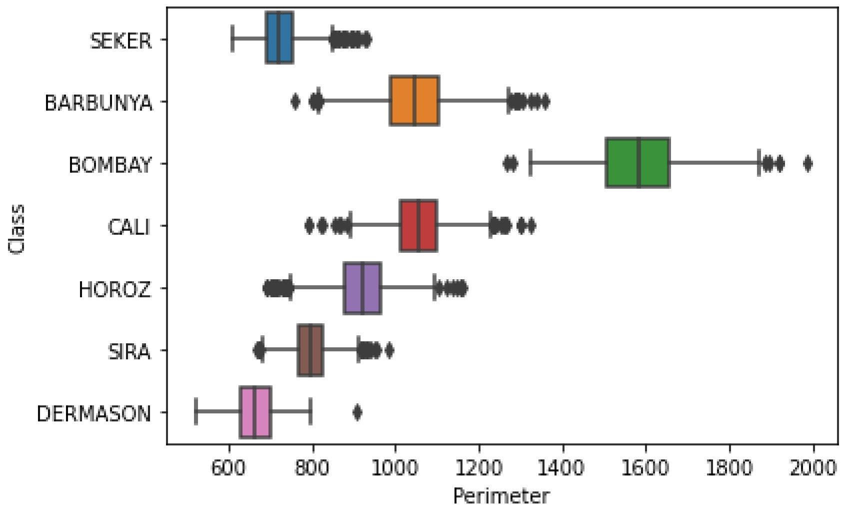
Let's now examine the distribution of some variables to see if their distributions are also somehow related



It seems that 1 class is always on the extremes in both the plots that is HOROZ and arguably SEKER is also towards the extremes but not so much in the second plot. Secondly the 2 distributions plotted albeit not very clearly but seem to show some sort of inverse relation as the plots kind of look like mirror images as classes change sides of their extremes and this can be investigated with more techniques



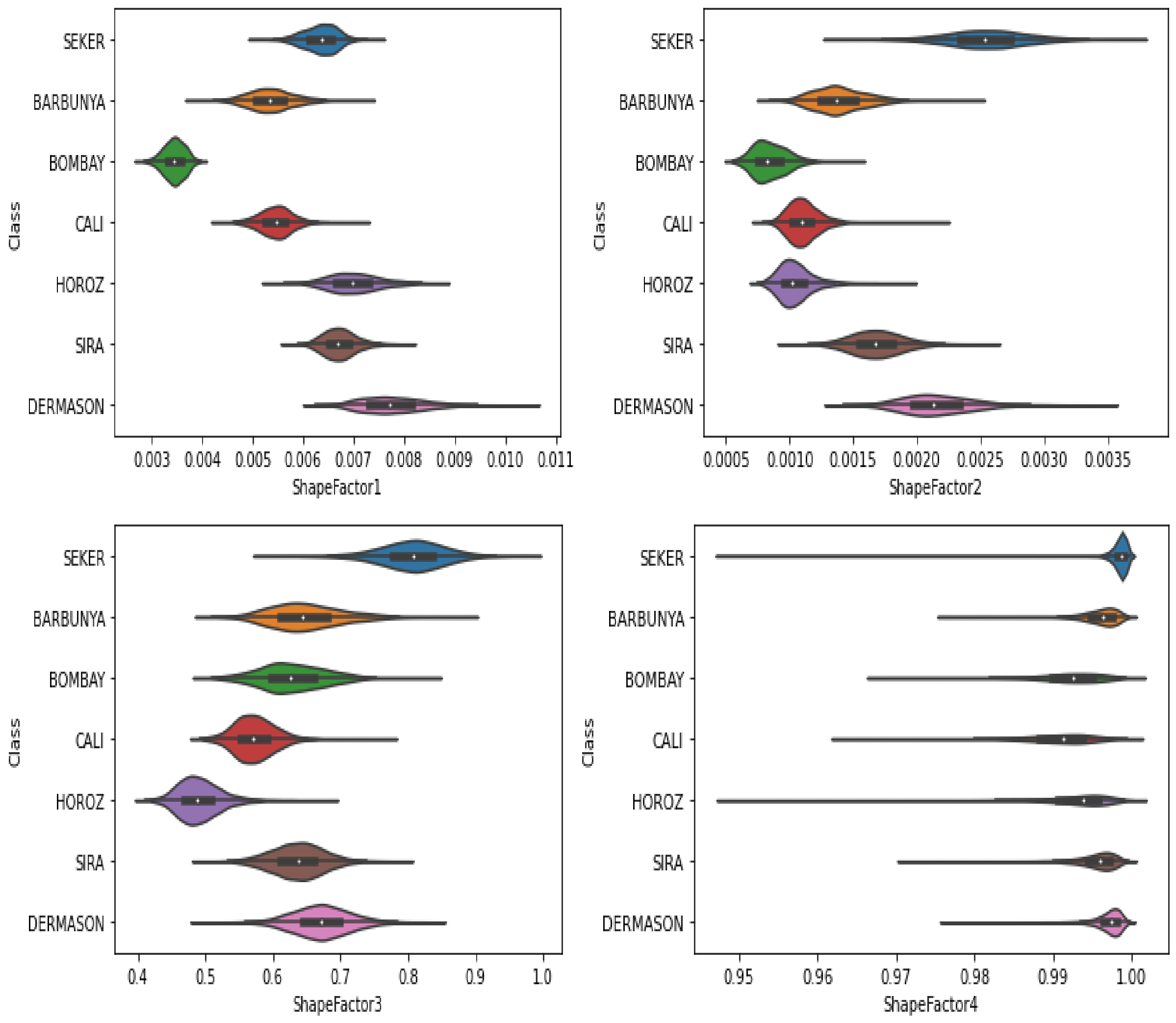




Using these boxplots we can reach to 2 clear conclusions-

- Compactness is highest for SEKER and lowest for HOROZ however both of these extremes also seem to share the greatest number of outliers

- The boxplots for Area and Perimeter look very alike as they are ofcourse correlated which again brings us to the point that there might be redundancy in features



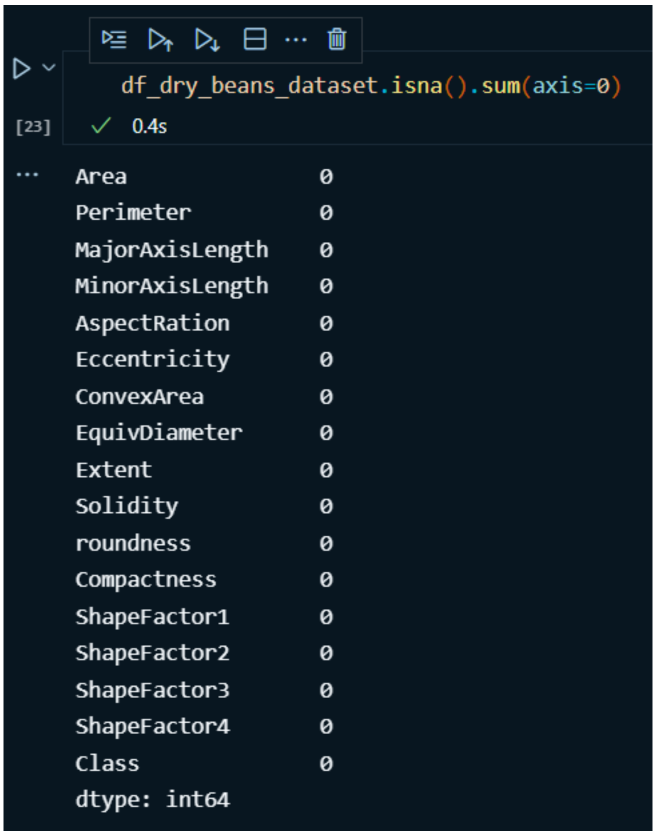
With the help of these Violin plots we can study the mysterious Shape Factors more effectively. As it's evident all of them seem to have been scaled down to be between 0 and 1 or the measurements themselves were like that

- ShapeFactor1 & ShapeFactor2 seem to be related as their violin plots have similar relative arrangements however the skewness is more for ShapeFactor2

- ShapeFactor3 and ShapeFactor4 also seem to follow the same general trends of relative ordering however ShapeFactor3 seems to be more `normally` distributed while ShapeFactor4 is filled with outliers and is highly skewed

Missing Value Analysis

There are no missing values in the dataset



```
df_dry_beans_dataset.isna().sum(axis=0)
```

[23] ✓ 0.4s

```
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
```

c)

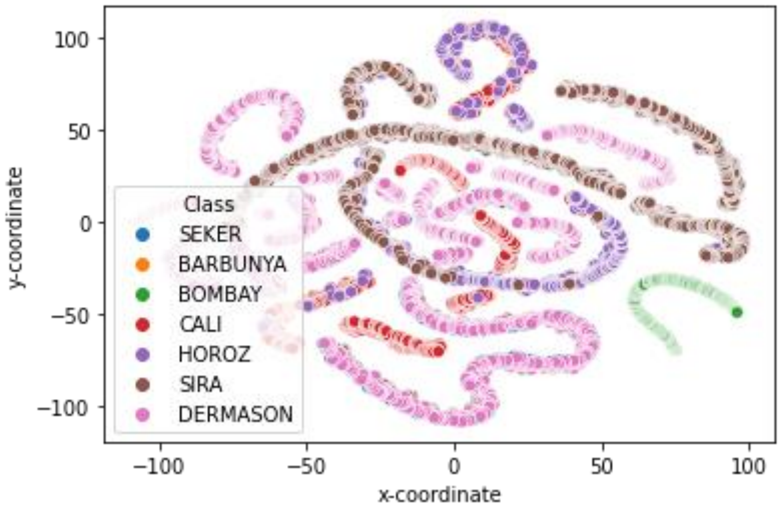
1)

TSNE Plots

Learning Rate – Auto

Init – Random

Perplexity – 30

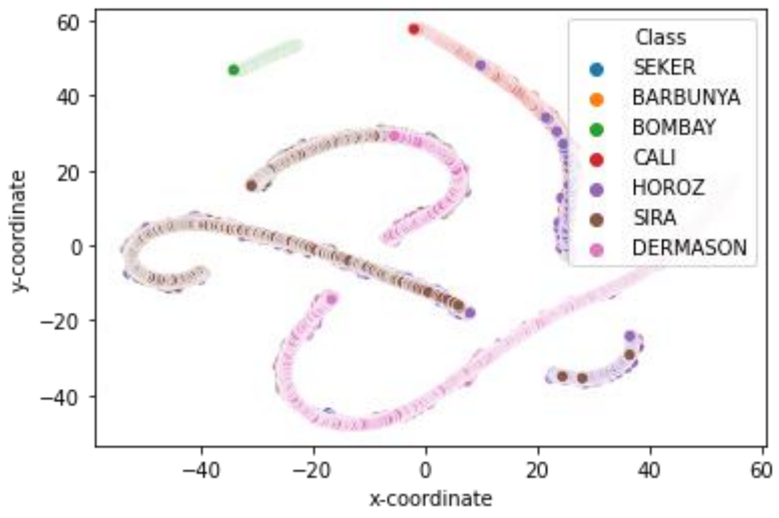


2)

Learning Rate - Auto

Init - Random

Perplexity - 200



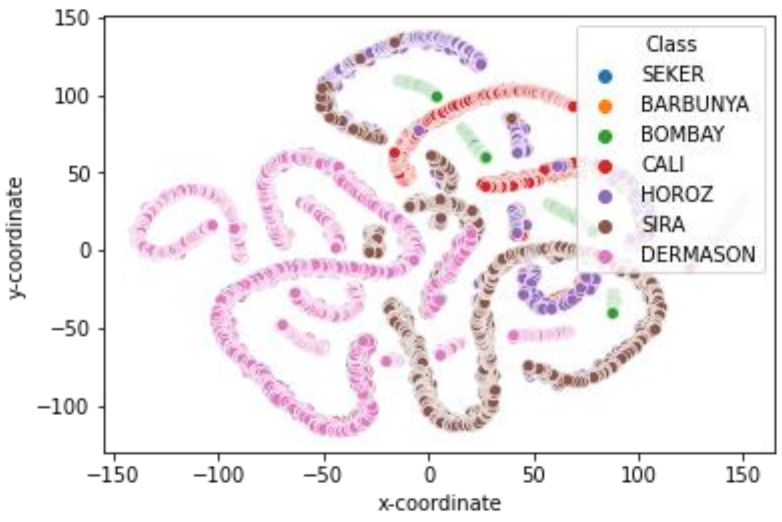
3)

Learning Rate - Auto

Init - Random

Perplexity - 30

Number of Iterations - 2000

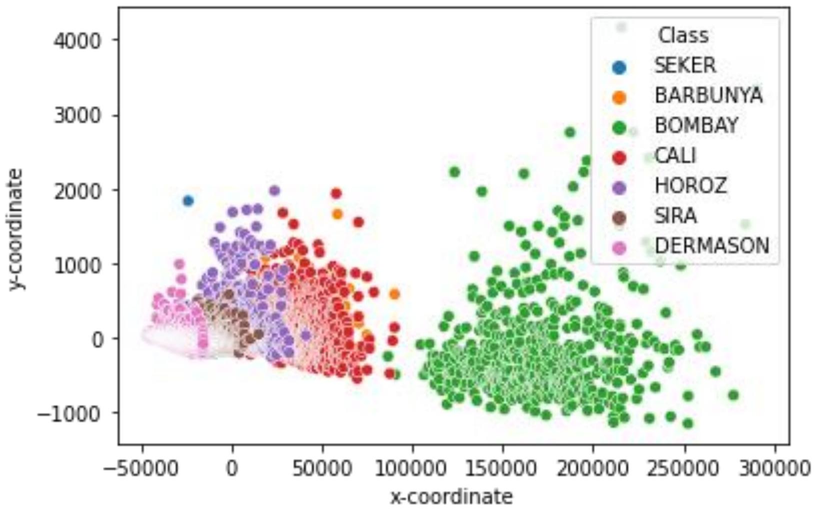


4)

Learning Rate - Auto

Init - PCA

Perplexity - 30



These 4 TNSE plots with varying hyper parameters show different insights. The default parameters work well but the reduces TSNE plot does not have clear clusters which can be obtained using KMeans type algorithms. The second plot shows that increasing the perplexity increases sparsity in the TSNE plot and points move in together. This was tried as the documentation listed larger datasets often require larger perplexity. The third plot retains the default perplexity but has higher iterations. These seem to create better chain of points as the pink and brown points are more joined as compared to the default number of iterations which are a 1000. Finally, init PCA simply initializes the embeddings before hand using PCA over the data. It sems to create much better clusters in the reduces dimension however the values range over a larger subset of the coordinate plane

d)

I tried Gaussian Naïve Bayes and Multinomial Naïve Bayes

For Gaussian Naïve Bayes

Accuracy 0.7579875137715755

Micro-Recall 0.7579875137715755

Macro-Recall 0.7601284951266063

Micro-Precision 0.7579875137715755

Macro-Precision 0.7630125373132769

For Gaussian Naïve Bayes with Different Smoothing Values Ranging From 0.1 to 1e-18 in jumps of 0.1

var_smoothing	Accuracy	Micro-Recall	Macro-Recall	Micro-Precision	Macro-Precision
0.1	0.613661	0.613661	0.596698	0.613661	0.545877
0.01	0.627249	0.627249	0.62902	0.627249	0.628388
0.001	0.625413	0.625413	0.630311	0.625413	0.629818
0.0001	0.627249	0.627249	0.632102	0.627249	0.631462
0.00001	0.640103	0.640103	0.64364	0.640103	0.643642
0.000001	0.683437	0.683437	0.687697	0.683437	0.693002
1E-07	0.733382	0.733382	0.737226	0.733382	0.74188
1E-08	0.753948	0.753948	0.756052	0.753948	0.758926
1E-09	0.757988	0.757988	0.760128	0.757988	0.763013
1E-10	0.789938	0.789938	0.791853	0.789938	0.798146
1E-11	0.856408	0.856408	0.85954	0.856408	0.866133
1E-12	0.88946	0.88946	0.89378	0.88946	0.897884
1E-13	0.896805	0.896805	0.903882	0.896805	0.904334
1E-14	0.904517	0.904517	0.912802	0.904517	0.91211
1E-15	0.905986	0.905986	0.914691	0.905986	0.913934
1E-16	0.905619	0.905619	0.914769	0.905619	0.914504
1E-17	0.904884	0.904884	0.913199	0.904884	0.912926
1E-18	0.903783	0.903783	0.912132	0.903783	0.911635

Since decreasing Var Smoothing improves results it led me to believe that the class imbalance is significantly affecting the results and hence, I tested again with equal class balance

For Multinomial Naïve Bayes

Accuracy 0.7844289386705839

Micro-Recall 0.7844289386705839

Macro-Recall 0.7900888714079898

Micro-Precision 0.7844289386705839

Macro-Precision 0.7944780576133089

Without Prior Fitting

Accuracy 0.7825927286081528

Micro-Recall 0.7825927286081528

Macro-Recall 0.7893324585654612

Micro-Precision 0.7825927286081528

Macro-Precision 0.7906709374967739

Since removing prior fitting improves our performance it also hints towards class balancing

<u>Redoing Experiments with class wise balanced dataset</u>

Multinomial Naïve Bayes

Accuracy 0.9097127222982216

Micro-Recall 0.9097127222982216

Macro-Recall 0.9062097022875084

Micro-Precision 0.9097127222982216

Macro-Precision 0.905838868011614

Gaussian Naïve Bayes

Accuracy 0.8987688098495212

Micro-Recall 0.8987688098495212

Macro-Recall 0.8942932598863722

Micro-Precision 0.8987688098495212

Macro-Precision 0.8939195994323036

Hence the class imbalances are a significant part of the reason why are models didn’t work well as they skewed the priors. Fixing them show huge gains in performance

The difference between the two Naïve Bayes implementations are how they try to model the distribution $P(x_i, y)$ by making certain assumptions.

Gaussian Naïve Bayes assumes that the likelihood of features follows a Gaussian Distribution and the mean and deviation of that Gaussian Distribution are estimated using Maximum Likelihood Estimation Method

On the other hand, Multinomial Naïve Bayes assumes a multinomially distributed data. The distribution is parametrized using $t_y = (t_{y1}, t_{y2}, \dots, t_{yn})$ for each class y and n being the number of features and $t_{yi} = P(x_i, y)$ that is the probability of feature i appearing in sample procured from some class y. Here the aim is to estimate t_y by using Maximum Likelihood Estimation Method.

Without any balancing the results were significantly better for Multinomial naïve bayes as it seems to capture the features very well and does not assume a Gaussian distribution for all features which is not true in general and also adapts better to Multiple Skewed Classes. After balancing the difference is much smaller as now the adaption part is not needed.

d)

PCA was performed using n_components 4, 6, 8, 10 and 12

In all cases the first principal component explains almost all variance that is it explains more than 99.99 % of the variance.

The experiments were performed using both class balanced and imbalanced datasets and the PCA reduced data was used to train a Gaussian Naïve Bayes classifier.

For Class Balanced We Have:

Number of Components	Accuracy	Micro-Recall	Macro-Recall	Micro-Precision	Macro-Precision	Micro-F1	Macro-F1
4	0.935705	0.935705	0.934226	0.935705	0.933444	0.935705	0.933621
6	0.946648	0.946648	0.945342	0.946648	0.945387	0.946648	0.945194
8	0.946648	0.946648	0.945342	0.946648	0.945387	0.946648	0.945194
10	0.946648	0.946648	0.945342	0.946648	0.945387	0.946648	0.945194
12	0.946648	0.946648	0.945342	0.946648	0.945387	0.946648	0.945194

For Class Imbalanced We Have:

Number of Components	Accuracy	Micro-Recall	Macro-Recall	Micro-Precision	Macro-Precision	Micro-F1	Macro-F1
4	0.894969	0.894969	0.909483	0.894969	0.903589	0.894969	0.905921
6	0.894602	0.894602	0.911263	0.894602	0.90533	0.894602	0.907219
8	0.894602	0.894602	0.911263	0.894602	0.90533	0.894602	0.907219
10	0.894602	0.894602	0.911263	0.894602	0.90533	0.894602	0.907219
12	0.894602	0.894602	0.911263	0.894602	0.90533	0.894602	0.907219

In both cases as alluded above since the first component which is common in all explains almost all variance there is no significant difference in performance. Infact beyond 6 components there is no change observed at all as even the second and third components explain variances in the order of 1E-3 % which brings practically no information.

Overall, we see PCA + Balanced Dataset gives the best results amongst all configurations tested!

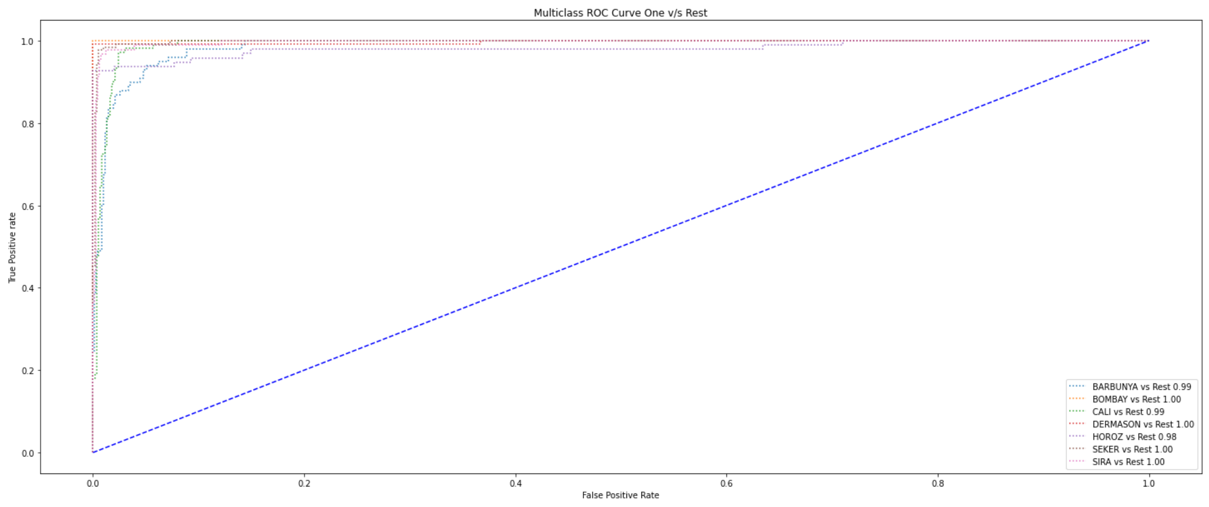
Since the top component explains almost all variance using more components gives very miniscule gains, infact beyond 6 components accuracy remains constant hence the PCA model with n_components = 6 is chosen for further analysis. Also, since the balanced dataset performs much better consistently it is used for the subsequent models too.

e)

Since Traditionally ROC-AUC Curves and Score are computed for Binary Classification we cannot directly use it for Multi Class Classification and hence

ROC-AUC Score with One v/s Rest 0.995230236474565

ROC-AUC Score with One v/s One 0.9950547714092179



Hence this shows a very score if we employ One v/s Rest strategy. In this strategy one classifier is fit per class. In each classifier one class if fit against all other classes. So in these cases we see the model achieves very high scores and is able to distinguish one class against the rest very well!

f)

Logistic Regression was Trained in the following configurations

The balanced dataset was used for these experiments

Configuration	**Accuracy**	**Micro-Recall**	**Macro-Recall**	**Micro-Precision**	**Macro-Precision**
100 Iterations	0.8426812	0.842681	0.8392474	0.8426812	0.83750133
1000 Iterations	0.945280	0.945280	0.9432624	0.9452804	0.9437263
10000 Iterations	0.95212038	0.95212038	0.9508718	0.9521203	0.9500840
100 Iterations Weight Balanced	0.835841	0.835841	0.831315	0.835841	0.829168
1000 Iterations Weight Balanced	0.94664842	0.9466484	0.9443982	0.9466484	0.94484974
10000 Iterations Weight Balanced	0.9534883	0.953488	0.9521705	0.953488	0.951199
100 Iterations Newton-cg Solver	0.95348837	0.95348837	0.952232	0.953488	0.9514837
100 Iterations liblinear Solver	0.9097127	0.9097127	0.9048246	0.9097127	0.906352
100 Iterations sag Solver	0.1409028	0.140902	0.1428571	0.1409028	0.0242409
100 Iterations saga Solver	0.140902	0.140902	0.1428571	0.140902	0.02015655

Newton-cg and liblinear solvers are best for giving results in lower number of iterations. Some solvers like sag and saga perform very poorly. Increasing

iterations till convergence also increases accuracy and adding weight
balancing gives a minor boost too.