MACHINE LEARNING WITH MATLAB

Data science assignment

Name:Anish.D

SRN:PES1201801334

ROLL NO:29

SECTION:A

DEPARTMENT:ECE

CAMPUS:RR CAMPUS

SEM:5

Import Data

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code creates a string array of player positions.

```
positions = ["G","G-F","F-G","F","F-C","C-F","C"]
```

```
positions = 1×7 string
"G"          "G-F"        "F-G"        "F"          "F-C"        "C-F"        "C"
```

Task 1

```
playerInfo=readtable("bballPlayers.txt")
```

```
playerInfo = 1250×5 table
```

Task 2

```
playerInfo.pos=categorical(playerInfo.pos)
```

```
playerInfo = 1250×5 table
```

Task 3

```
playerInfo.pos=categorical(playerInfo.pos,positions)
```

```
playerInfo = 1250×5 table
```

Task 4

```
playerInfo=rmmissing(playerInfo)
```

```
playerInfo = 1167×5 table
```

Task 5

```
allStats=readtable("bballStats.txt");
```

Task 6

```
allStats(:,19:end)=[]
```

```
allStats = 4587×18 table
```

Further Practice

```
idx=playerInfo.name == "LeBron James";
ID=playerInfo.playerID(idx)
```

```
ID = 1×1 cell array
    {'jamesle01'}
```

```
LJstats=allStats(allStats.playerID == string(ID),:)
```

```
LJstats = 8×18 table
```

Group and Merge Data

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code imports and formats the tables `playerInfo` and `allStats`.

```
playerInfo = readtable("bballPlayers.txt");
positions = ["G","G-F","F-G","F","F-C","C-F","C"];
playerInfo.pos = categorical(playerInfo.pos,positions);
playerInfo = rmmissing(playerInfo)
```

```
playerInfo = 1167×5 table
```

```
allStats = readtable("bballStats.txt");
allStats(:,19:end) = []
```

```
allStats = 4587×18 table
```

Task 1

```
playerStats=groupsummary(allStats,"playerID","sum")
```

```
playerStats = 609×19 table
```

Task 2

```
playerStats.GroupCount=[]
```

```
playerStats = 609×18 table
```

```
playerStats.Properties.VariableNames=allStats.Properties.VariableNames
```

```
playerStats = 609×18 table
```

Task 3

```
data=innerjoin(playerInfo,playerStats)
```

```
data = 609×22 table
```

Further Practice

```
joinedData = 5145×23 table
```

Explore Data

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads the table `data` which contains the basketball player data.

```
load bballData.mat
```

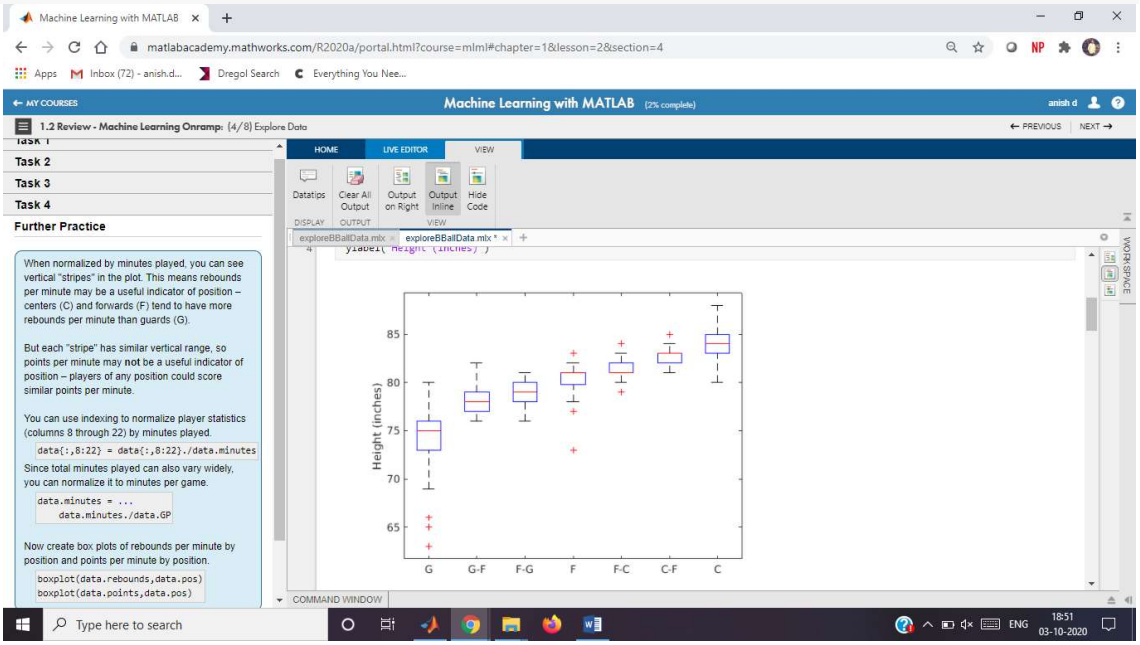
```
data
```

```
data = 609×22 table
```

Task 1

```
boxplot(data.height,data.pos)
```

```
ylabel("Height (inches)")
```

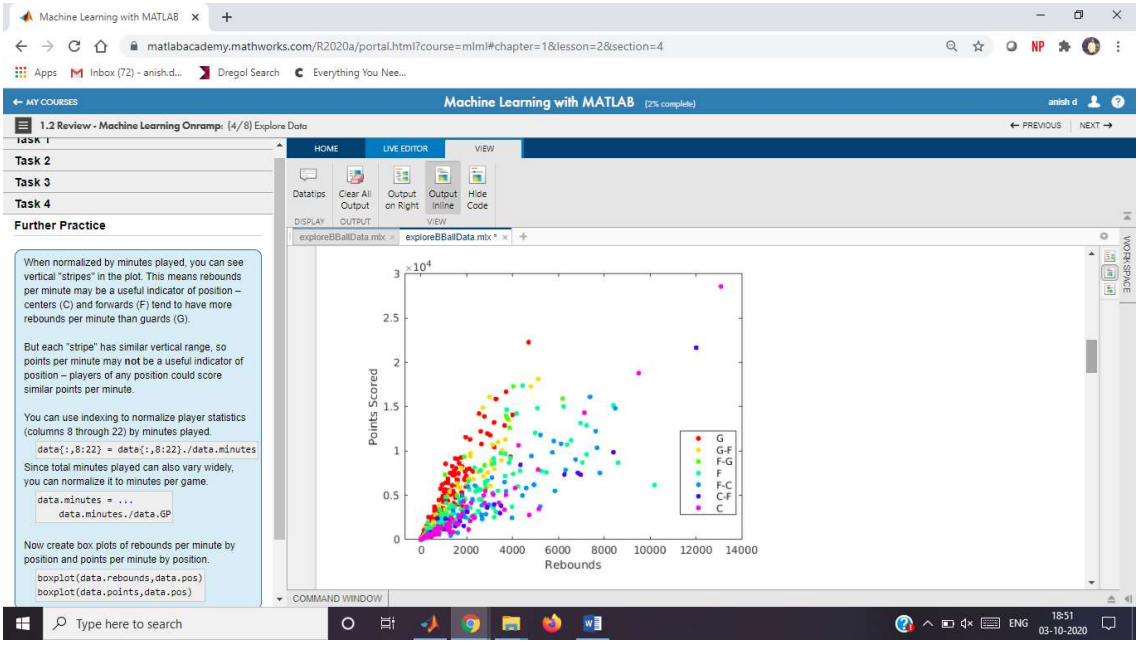


Task 2

```
gscatter(data.rebounds,data.points,data.pos)
```

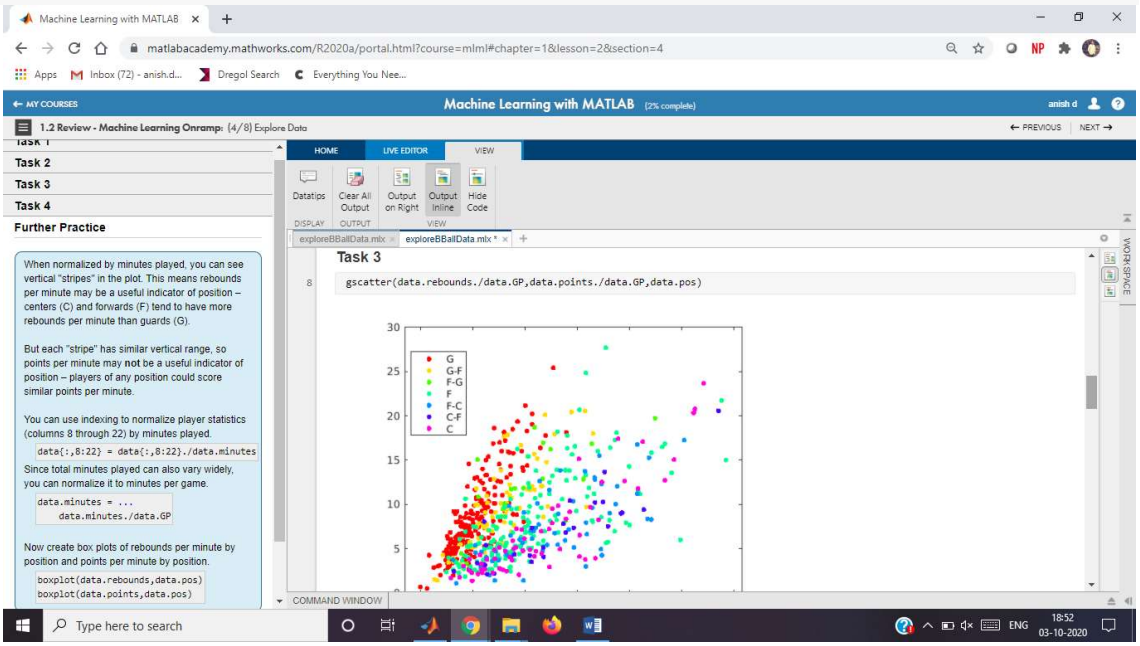
```
xlabel("Rebounds")
```

```
ylabel("Points Scored")
```



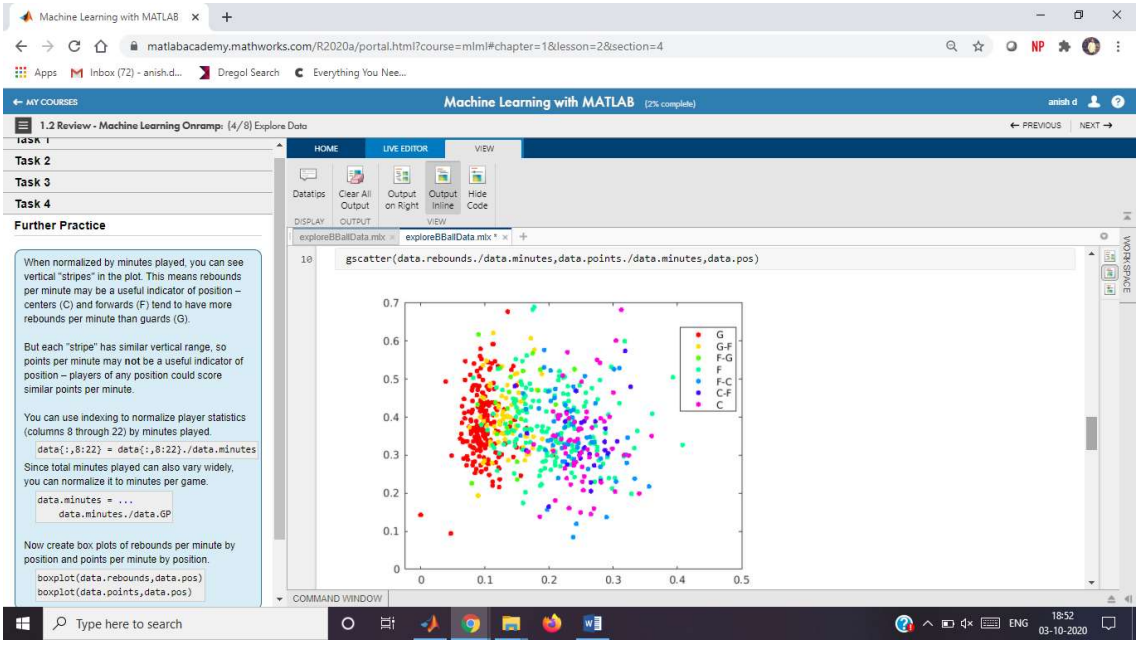
Task 3

```
gscatter(data.rebounds./data.GP,data.points./data.GP,data.pos)
```



Task 4

```
gscatter(data.rebounds./data.minutes,data.points./data.minutes,data.pos)
```



Further Practice

```
data{:,8:22}=data{:,8:22}./data.minutes
```

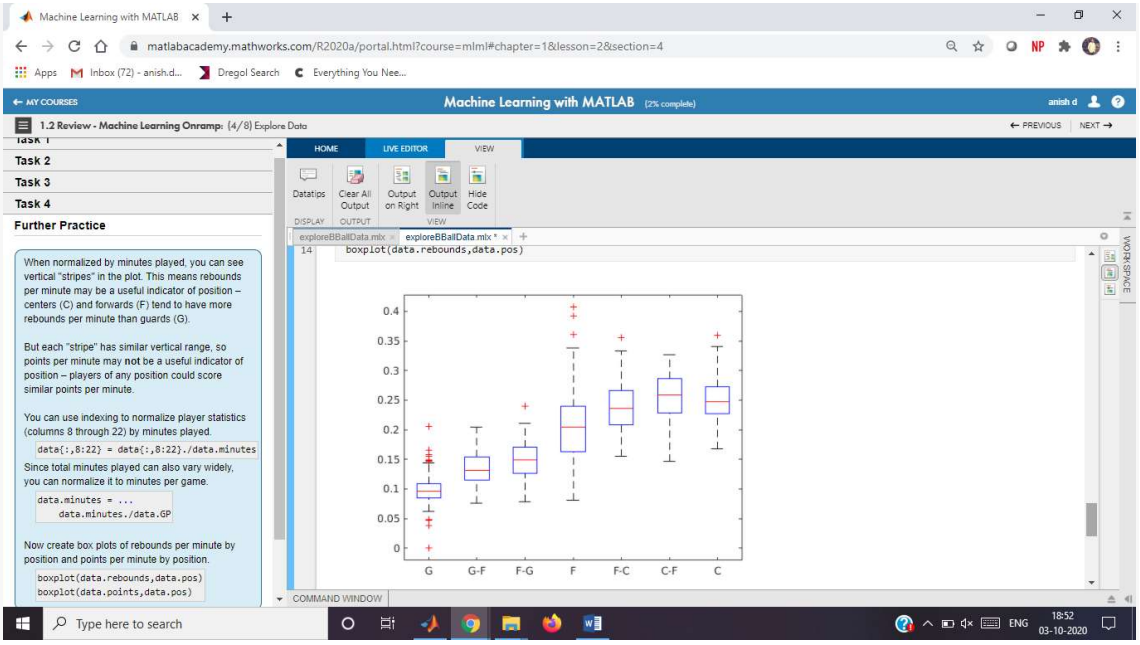
```
data = 609×22 table
```

```
data.minutes=data.minutes./data.GP
```

```
data = 609×22 table
```

```
boxplot(data.rebounds,data.pos)
```

```
boxplot(data.points,data.pos)
```



Train a Model and Make Predictions

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads and displays the normalized (per minute) data set.

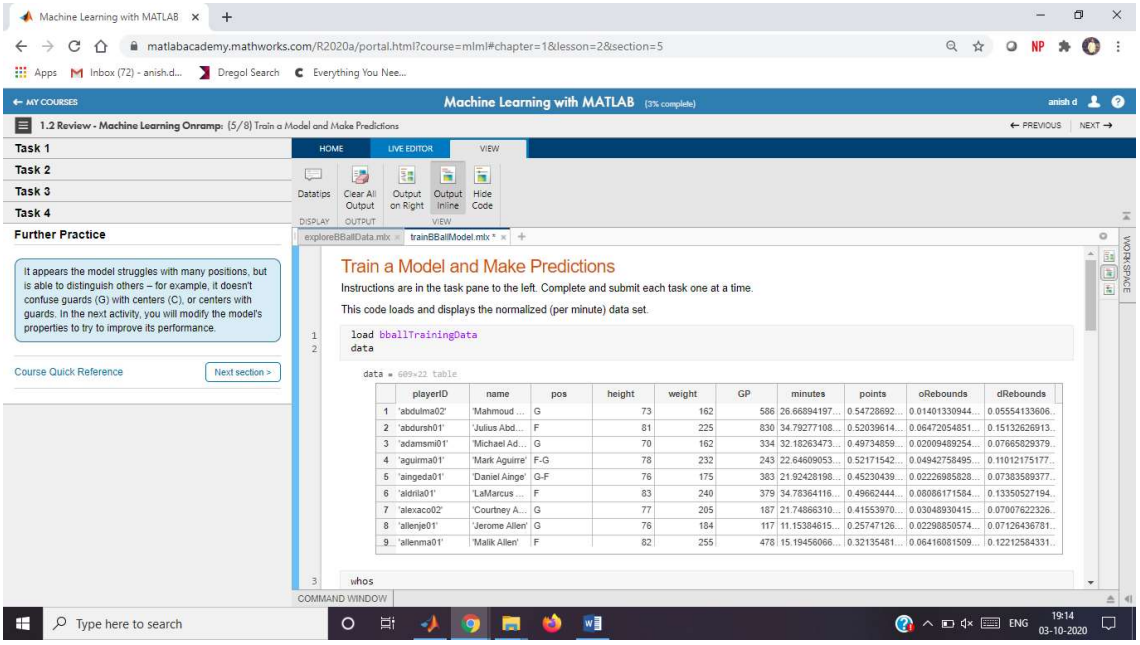
```
load bballTrainingData
```

```
data
```

```
data = 609×22 table
```

```
whos
```

```
  Name             Size              Bytes  Class                Attributes

  data           609x22             253417  table
  dataTest       182x20              33820  table
  dataTrain      427x20              71305  table
  knnmodel         1x1               79756  ClassificationKNN
  mdlLoss          1x1                   8  double
  predPos        182x1                 942  categorical
```



Task 1

```
knnmodel=fitcknn(dataTrain,"pos")
```

```
knnmodel =

  ClassificationKNN
           PredictorNames: {'height'  'weight'  'GP'  'minutes'  'points'  'oRebounds'
'dRebounds'  'rebounds'  'assists'  'steals'  'blocks'  'turnovers'  'PF'  'fgAttempted'
'fgMade'  'ftAttempted'  'ftMade'  'threeAttempted'  'threeMade'}
             ResponseName: 'pos'
    CategoricalPredictors: []
               ClassNames: [G    G-F    F-G    F    F-C    C-F    C]
           ScoreTransform: 'none'
          NumObservations: 427
                 Distance: 'euclidean'
             NumNeighbors: 1


  Properties, Methods
```

Task 2

```
predPos=predict(knnmodel,dataTest)
```

```
predPos = 182×1 categorical

G-F
C
G
F-C
F
G
F
G
G
G
```

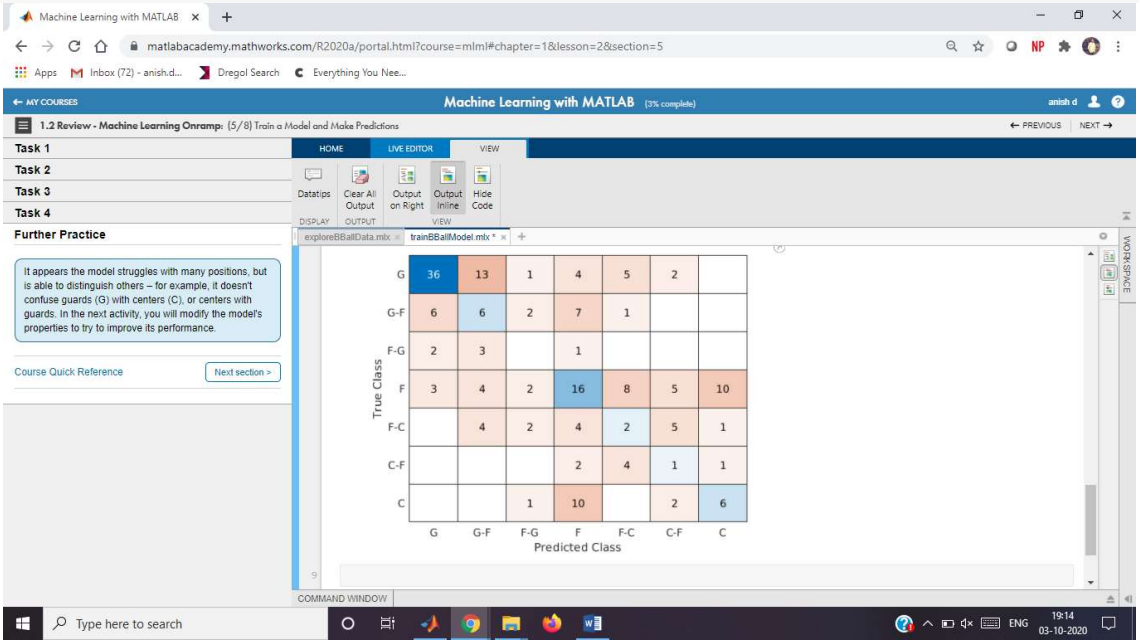
Task 3

```
mdlLoss=loss(knnmodel,dataTest)
```

```
mdlLoss =

   0.632248263562002
```

Task 4

```
confusionchart(dataTest.pos,predPos)
```



Further Practice

Predicting Player Position

Instructions are in the task pane to the left. Complete and submit each task one at a time.

Load data

Load the basketball player statistics (per minute) data set.

```
load bballTrainingData.mat

whos
```

```
  Name             Size             Bytes  Class    Attributes

  data           609x22            253417  table
  dataTest       182x20             33820  table
  dataTrain      427x20             71305  table
```

Train kNN model

Fit a kNN classification model to the training data.

```
knnmodel = fitcknn(dataTrain,"pos","NumNeighbors",5,"Standardize",true);
```

Evaluate model

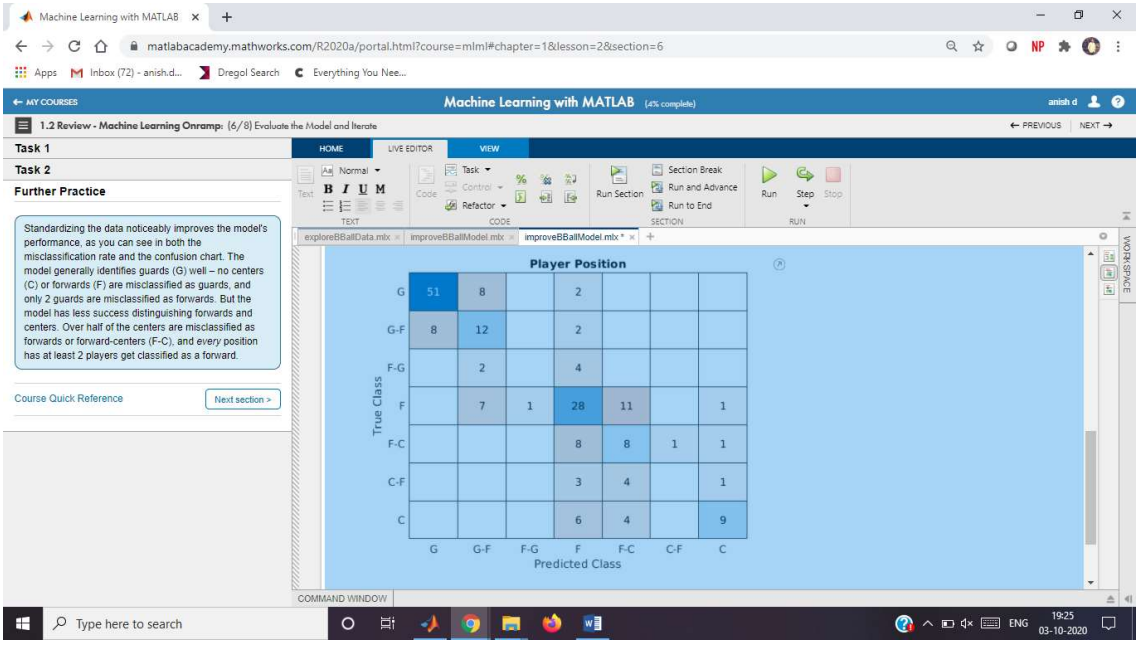
Calculate misclassification rate.

```
mdlLoss = loss(knnmodel,ataTest)
```

```
mdlLoss =

   0.408488539314294
```

Display a confusion chart.

```
predPos = predict(knnmodel,dataTest);

confusionchart(dataTest.pos,predPos);

title("Player Position")
```



Further Practice

Classical Multidimensional Scaling

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads and displays the data.

```
load data;
```

```
whos X;
```

```
  Name        Size             Bytes  Class     Attributes

  X         124x4               3968  double
```

Task 1

```
D=pdist(X)
```

```
D = 1×7626

    2.0250    3.2150    2.8857    0.8190    4.0122    2.7162    3.4925    2.9274    1.9251
2.0929    2.0482    2.4460    1.5394    1.4666    4.2891    1.7114    2.1354    2.7378
0.9318    1.2971    2.2531    1.4692    1.4127    2.1062    0.8100    2.5849    1.0226
2.8476    3.1070    2.4005    5.0555    2.0674    1.2046    2.1360    1.4733    3.5514
```

```
1.7065    1.6416    2.7010    1.0674    2.3241    1.1654    2.7345    1.6539    2.1314
1.0888    2.4780    1.0482    3.1955    2.6307
```

Task 2

```
[Y e]=cmdscale(D)
```

Y = 124×4

```
    5.1696   -1.2944    1.1102   -0.0717
    7.0363   -1.5951    0.4156    0.1360
    5.0959   -2.2218   -1.9668   -0.0208
    6.5495   -1.1599   -1.4205   -0.0591
    5.1581   -1.4514    0.3076   -0.0293
    5.4756   -0.0992    4.9281   -0.0875
    4.0896    0.6602    2.6558   -0.1166
    6.0890   -2.1255   -2.1387    0.2543
    5.4512    0.2453    3.5730    0.1621
    6.8605   -0.7875    0.3488    0.0298
```

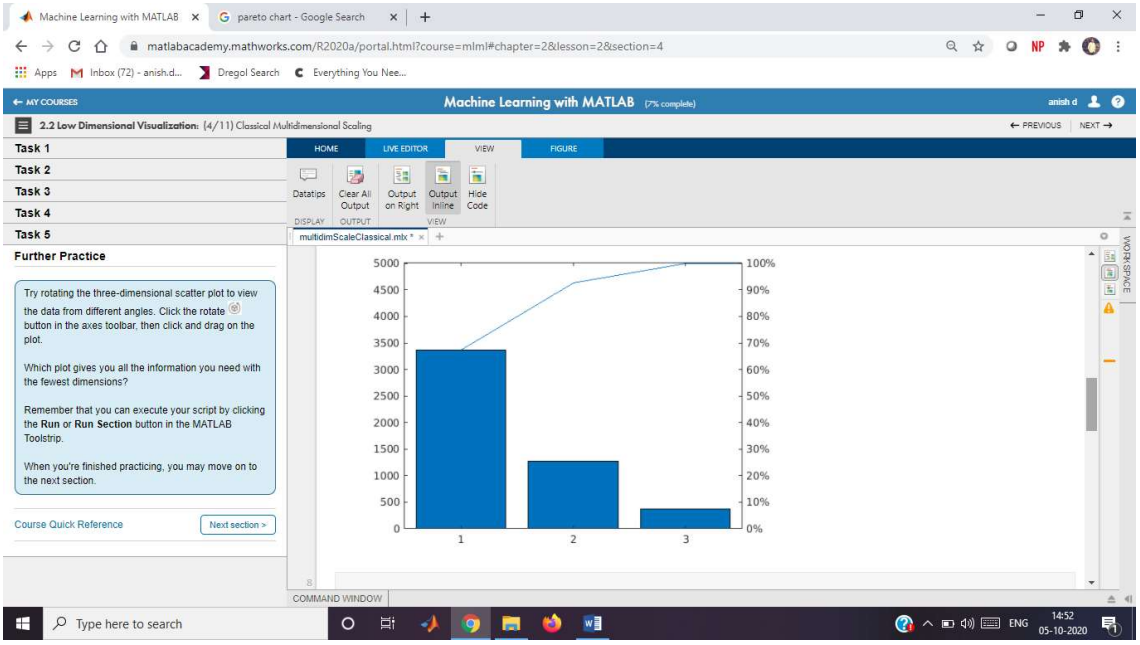
e = 124×1

10^3 ×

```
    3.3604
    1.2747
    0.3685
    0.0010
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
```

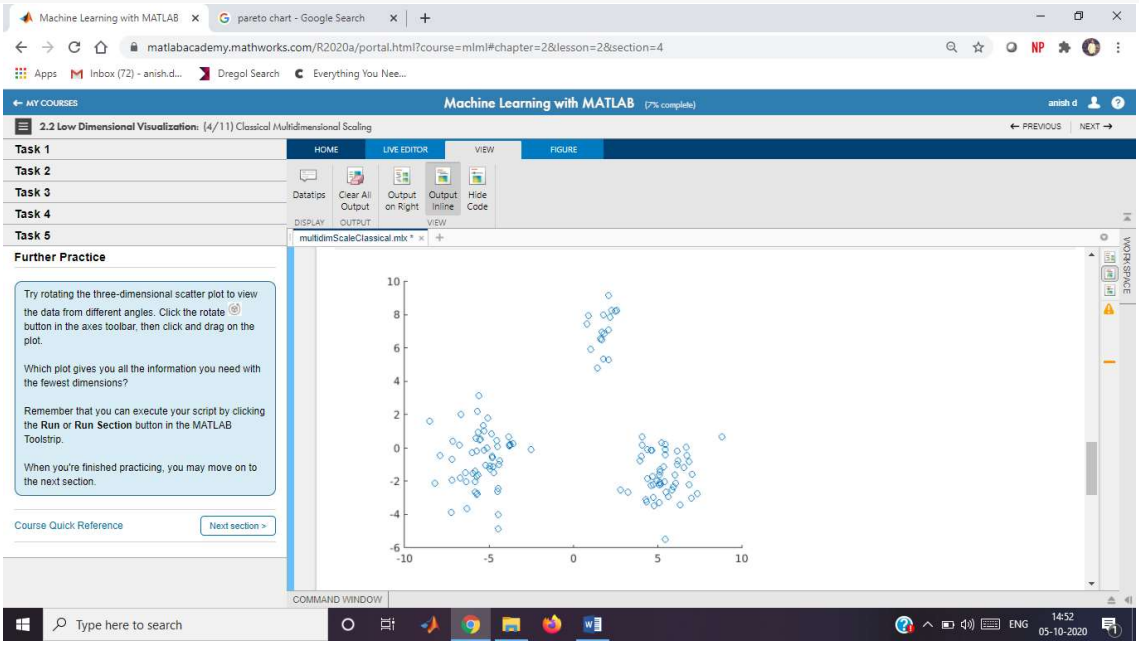
Task 3

```
pareto(e)
```



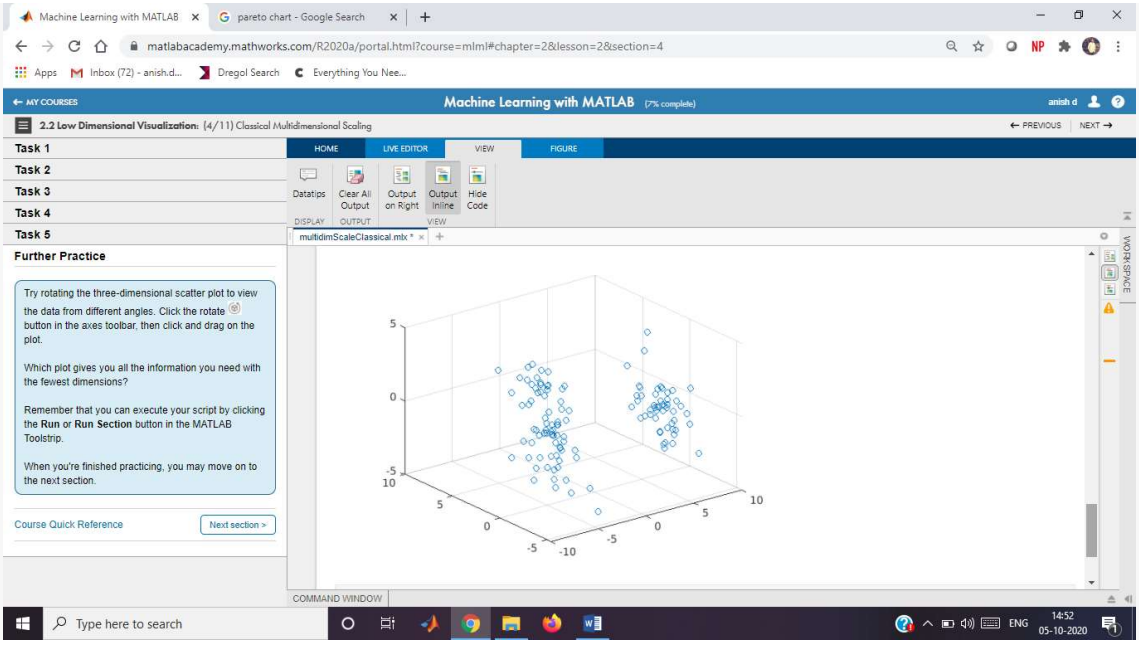
Task 4

```
scatter(Y(:,1),Y(:,2))
```



Task 5

```
scatter3(Y(:,1),Y(:,2),Y(:,3))
```



Further Practice

Nonclassical Multidimensional Scaling

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads and displays the data.

```
load data;
whos X;
```

```
  Name        Size             Bytes  Class     Attributes

  X         124x4               3968  double
```

Task 1

```
D=pdist(X)
```

```
D = 1×7626
    2.0250    3.2150    2.8857    0.8190    4.0122    2.7162    3.4925    2.9274    1.9251
2.0929    2.0482    2.4460    1.5394    1.4666    4.2891    1.7114    2.1354    2.7378
0.9318    1.2971    2.2531    1.4692    1.4127    2.1062    0.8100    2.5849    1.0226
2.8476    3.1070    2.4005    5.0555    2.0674    1.2046    2.1360    1.4733    3.5514
```

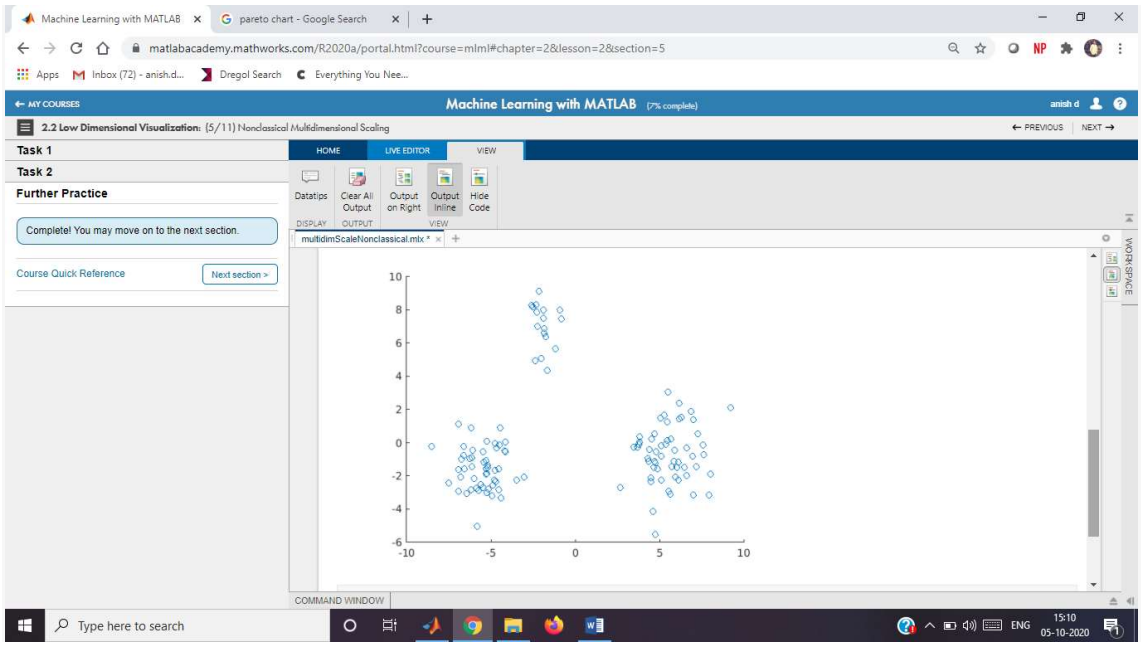
```
1.7065    1.6416    2.7010    1.0674    2.3241    1.1654    2.7345    1.6539    2.1314
1.0888    2.4780    1.0482    3.1955    2.6307
```

```
Y=mdscale(D,2)
```

```
Y = 124×2
   -5.2761   -1.0903
   -6.9296   -1.6278
   -5.1333   -2.8046
   -6.5811   -1.5753
   -5.2105   -1.3571
   -6.9345    1.1404
   -4.4601    0.9212
   -6.1572   -2.8471
   -6.1576    0.8986
   -6.7452   -1.0029
```

Task 2

```
scatter(Y(:,1),Y(:,2))
```



Further Practice

Principal Component Analysis

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads and displays the data.

```
load data;
whos X;
```

```
  Name        Size            Bytes  Class     Attributes

  X         124x4              3968  double
```

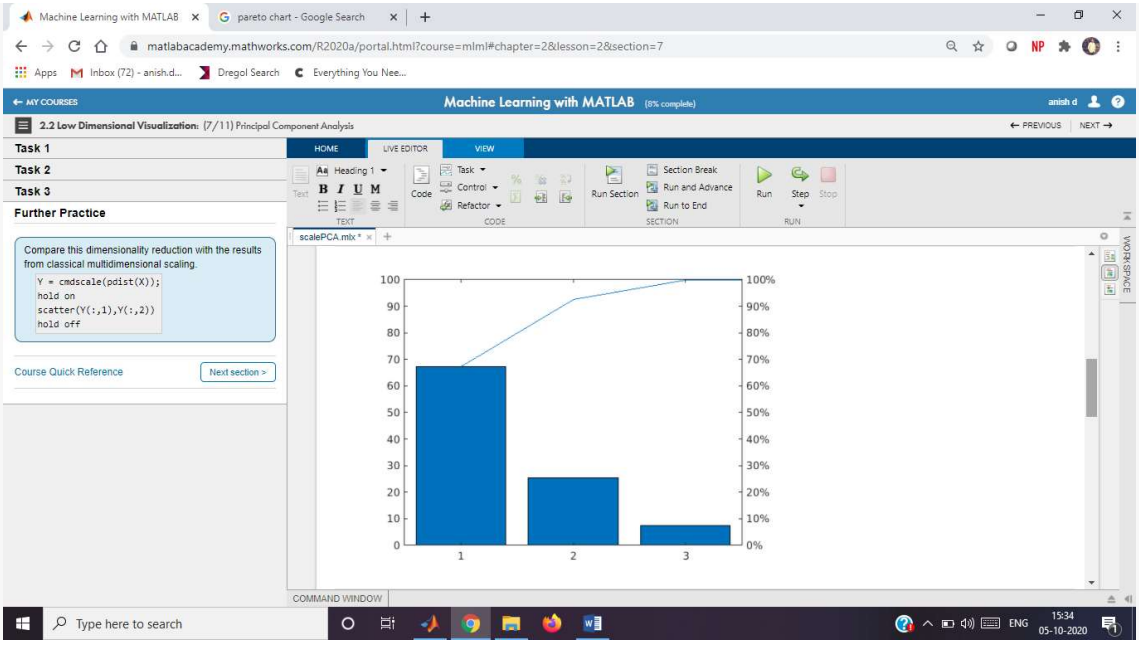
Task 1

```
[pcs,scrs,~,~,pexp]=pca(X)
```

```
pcs = 4×4
   -0.0023    0.7120    0.7022   -0.0015
    0.8936   -0.3138    0.3211    0.0009
    0.4489    0.6282   -0.6355   -0.0013
   -0.0002    0.0021   -0.0000    1.0000
scrs = 124×4
    5.1696   -1.2944    1.1102   -0.0717
    7.0363   -1.5951    0.4156    0.1360
    5.0959   -2.2218   -1.9668   -0.0208
    6.5495   -1.1599   -1.4205   -0.0591
    5.1581   -1.4514    0.3076   -0.0293
    5.4756   -0.0992    4.9281   -0.0875
    4.0896    0.6602    2.6558   -0.1166
    6.0890   -2.1255   -2.1387    0.2543
    5.4512    0.2453    3.5730    0.1621
    6.8605   -0.7875    0.3488    0.0298
pexp = 4×1
   67.1458
   25.4710
    7.3635
    0.0197
```

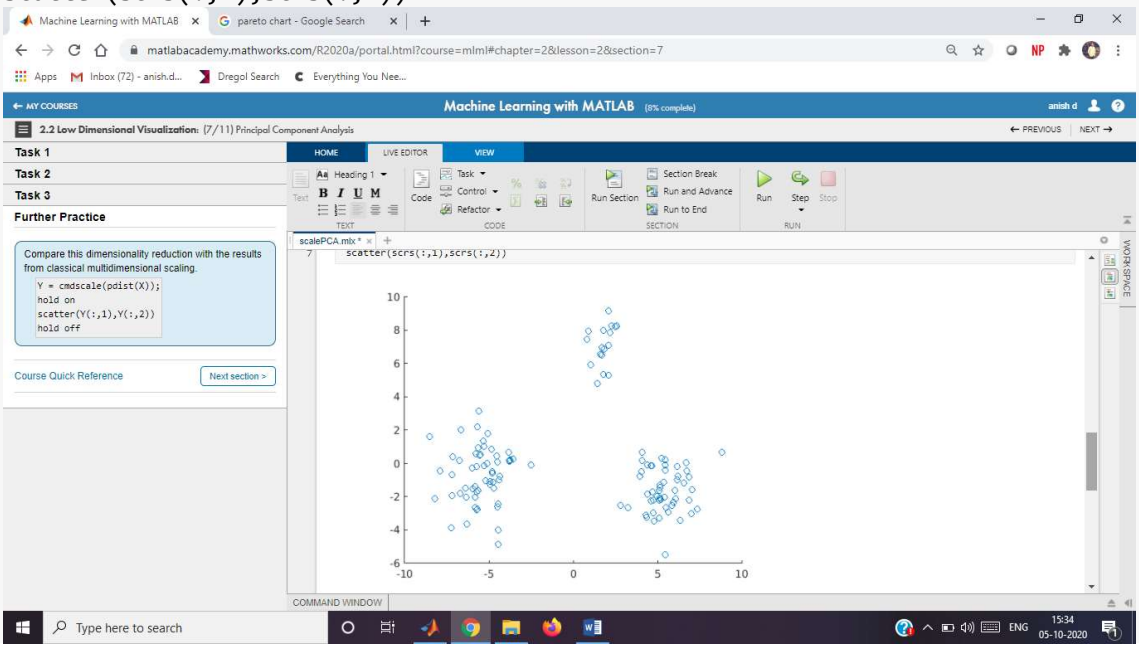
Task 2

```
pareto(pexp)
```



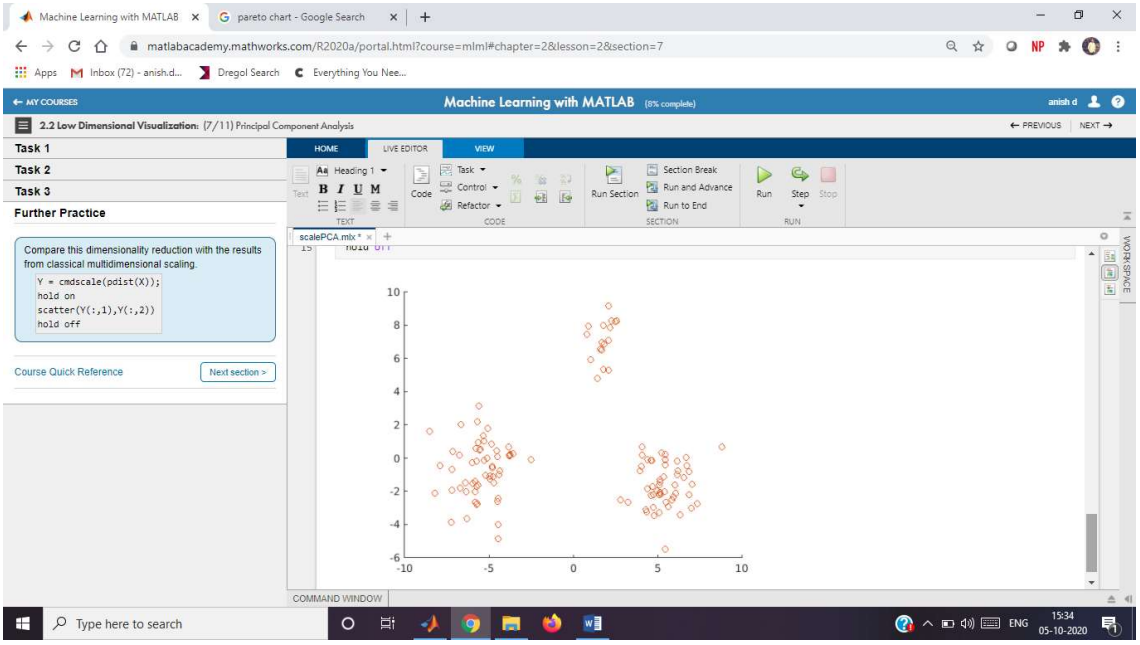
Task 3

```
scatter(scrs(:,1),scrs(:,2))
```



Further Practice

```
Y = cmdscale(pdist(X));
hold on
scatter(Y(:,1),Y(:,2))
hold off
```



Basketball Players

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads and formats the data.

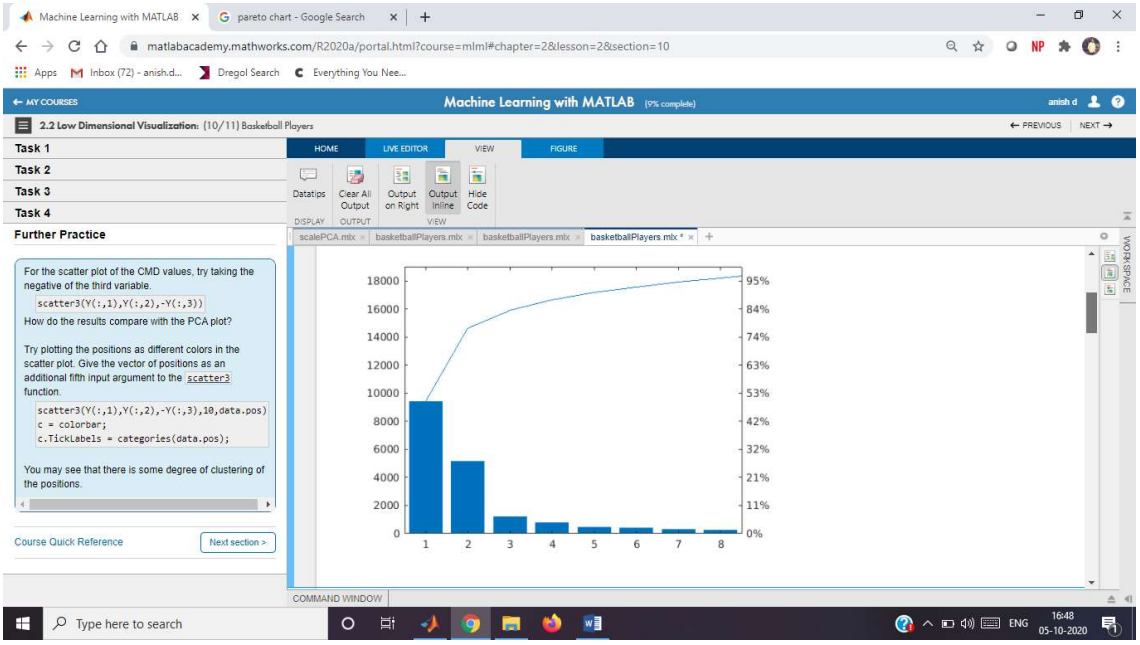
```
data = readtable("bball.txt");
data.pos = categorical(data.pos);
```

This code extracts and normalizes the columns of interest.

```
stats = data{:,[5 6 11:end]};
statsNorm = normalize(stats);
```

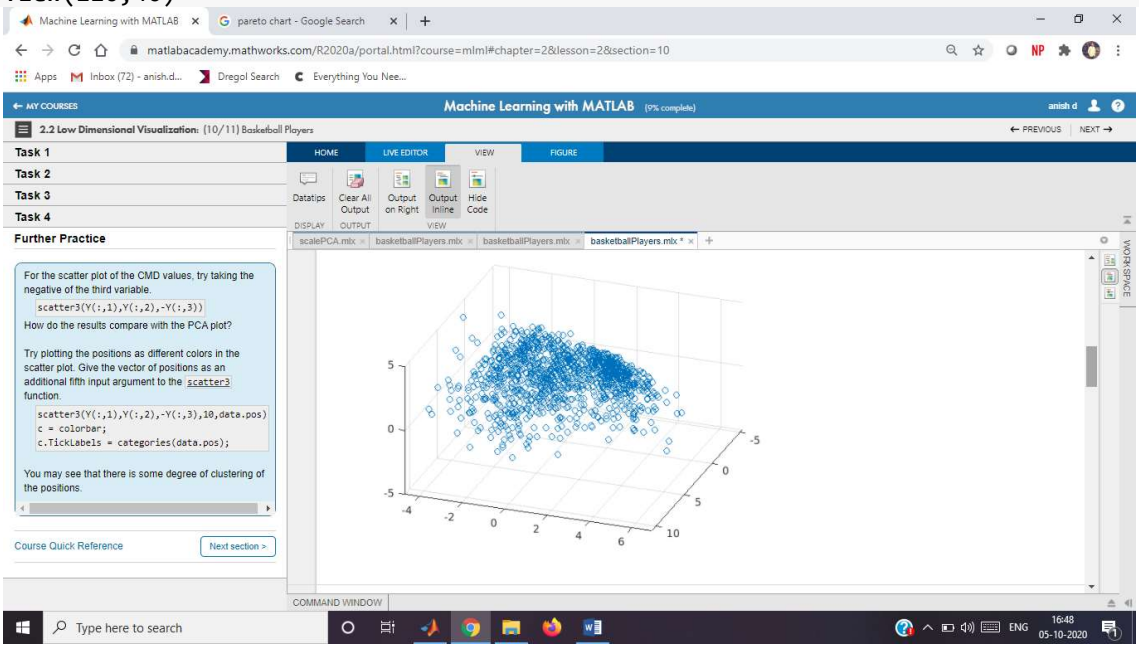
Task 1

```
pd = pdist(statsNorm);
[Y,e] = cmdscale(pd);
pareto(e)
```



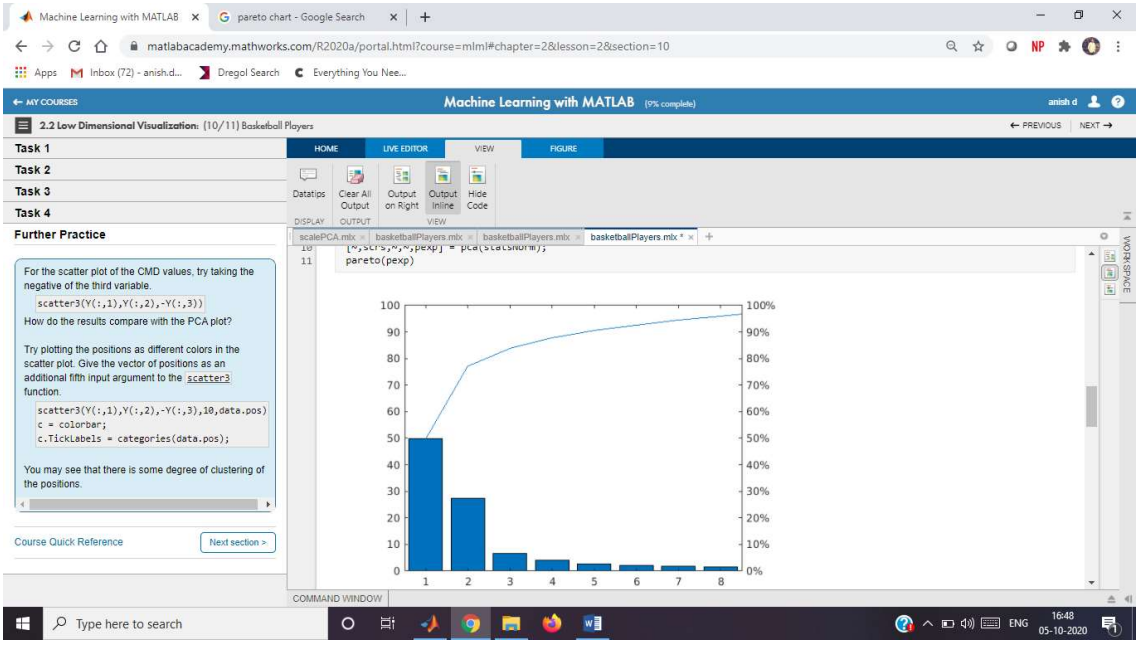
Task 2

```
scatter3(Y(:,1),Y(:,2),Y(:,3))
view(110,40)
```



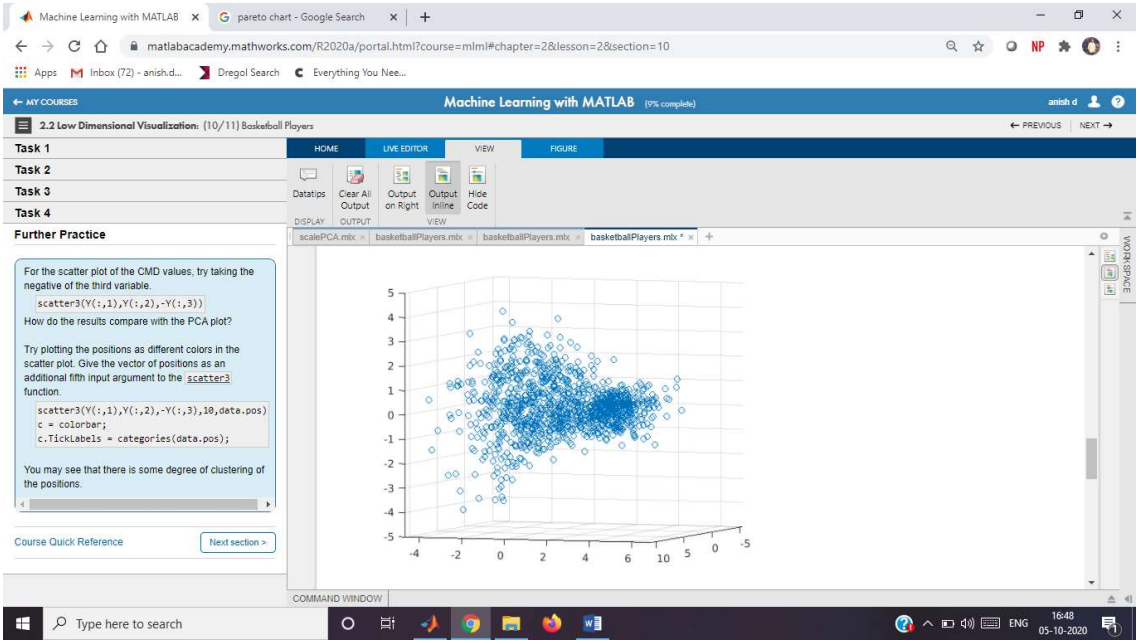
Task 3

```
[~,scrs,~,~,pexp] = pca(statsNorm);
pareto(pexp)
```



Task 4

```
scatter3(scrs(:,1),scrs(:,2),scrs(:,3))
view(110,4)
```



Further Practice

```
scatter3(Y(:,1),Y(:,2),-Y(:,3))
```

```
scatter3(Y(:,1),Y(:,2),-Y(:,3),10,data.pos)
c = colorbar;
c.TickLabels = categories(data.pos);
```

k-Means Clustering

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads and displays the data.

```
load data;
whos X;
```

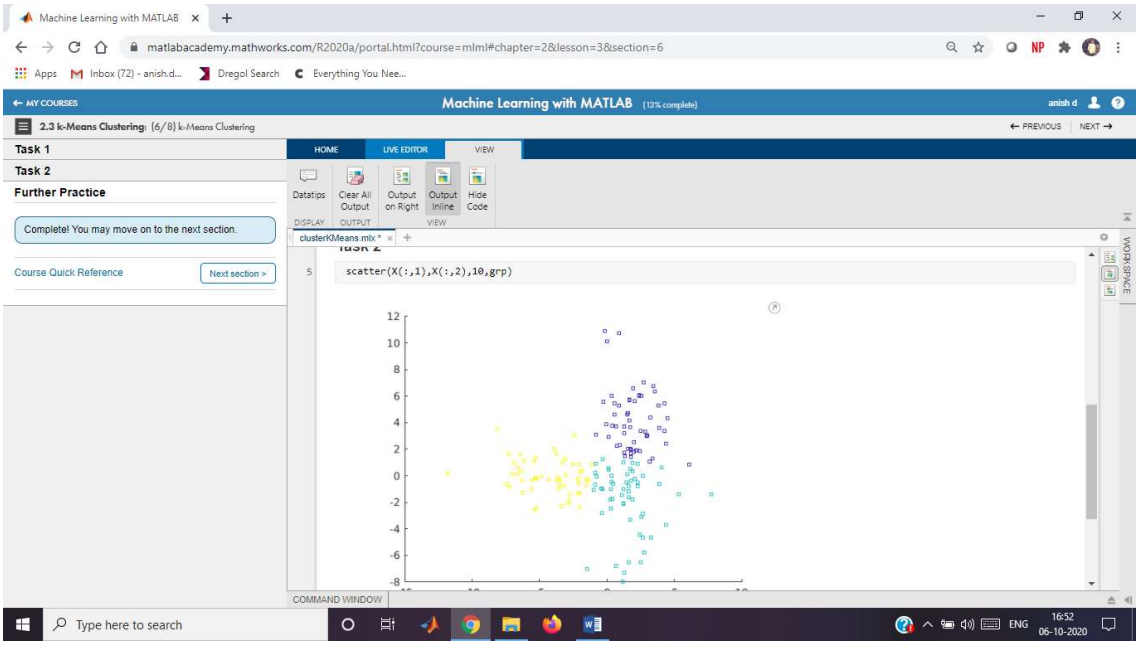
Task 1

```
grp=kmeans(X,3)
```

Task 2

```
scatter(X(:,1),X(:,2),10,grp)
```

Further Practice



Options for k-Means Clustering

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads and displays the data.

```
load rData

whos X
```

```
  Name        Size              Bytes  Class     Attributes

  X         217x2                3472  double
```

Task 1

```
grp=kmeans(X,3)
```

```
grp = 217×1

     3
     3
     3
     3
     3
     1
     1
     3
     1
     1
```

Task 2

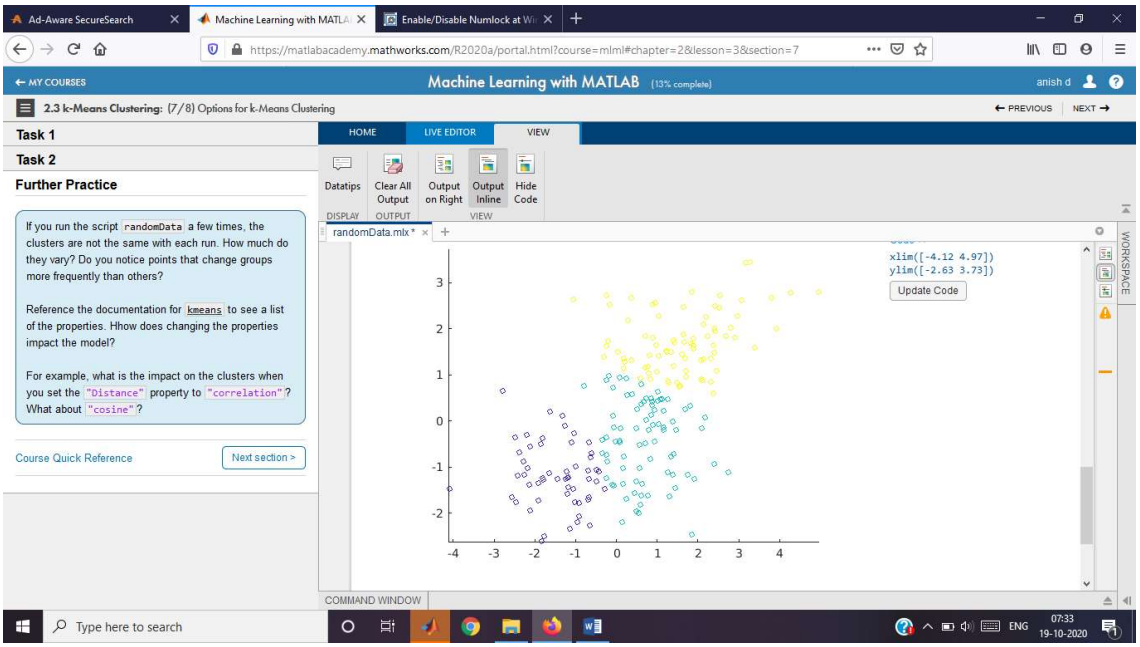
```
grp=kmeans(X,3,"Start","cluster","Replicates",5)
```

```
grp = 217×1

     2
     1
     1
     1
     1
     2
     2
     1
     2
     2
```

This code plots the data by group.

```
scatter(X(:,1),X(:,2),15,grp)
```

-



Basketball Players

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads and formats the data.

```
data = readtable("bball.txt");

data.pos = categorical(data.pos);
```

This code extracts and normalizes the columns of interest.

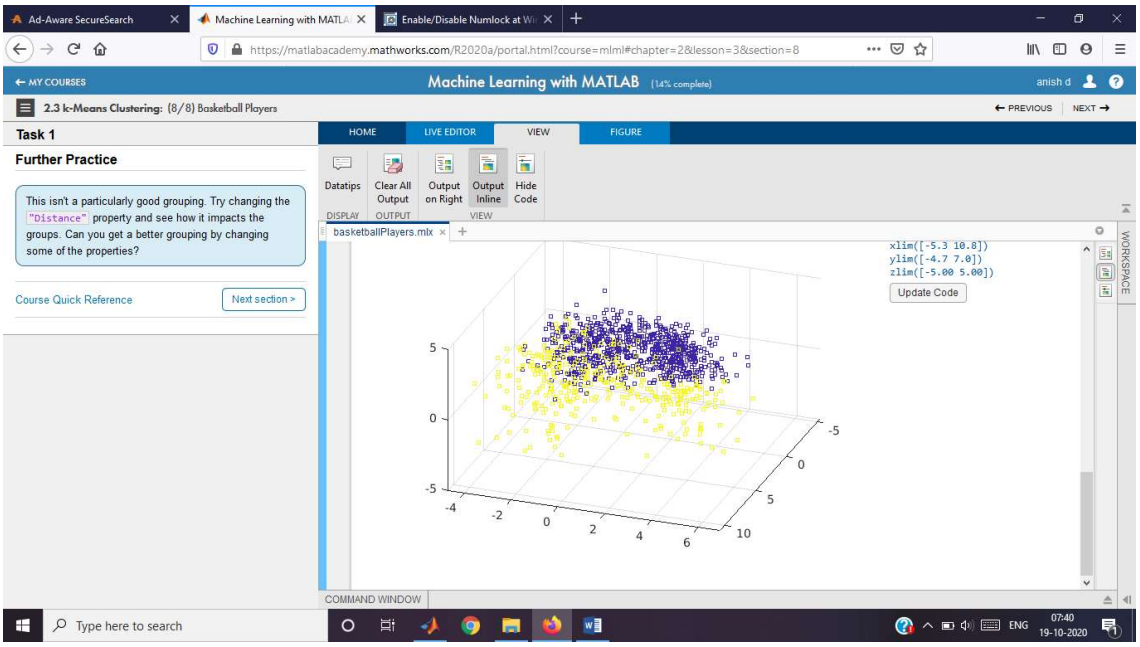
```
stats = data{:,[5 6 11:end]};

statsNorm = normalize(stats);
```

Task 1

```
grp=kmeans(statsNorm,2,"Replicates",5)
```

This code performs PCA and plots the transformed data by group.

```
[pcs,scrs] = pca(statsNorm);

scatter3(scrs(:,1),scrs(:,2),scrs(:,3),10,grp)

view(110,40)
```



Gaussian Mixture Models

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads and displays the data.

```
load data;
whos X;
```

```
  Name        Size             Bytes  Class     Attributes

  X         154x2              2464  double
```

Tasks 1 & 4

```
mdl=fitgmdist(X,3)
```

```
mdl =

Gaussian mixture distribution with 3 components in 2 dimensions
Component 1:
Mixing proportion: 0.442779
Mean:   -2.5596    5.0668

Component 2:
Mixing proportion: 0.384481
Mean:   -2.2042   -4.1388

Component 3:
Mixing proportion: 0.172740
Mean:    3.0404   -0.1552
```

```
mdl=fitgmdist(X,3,"CovarianceType","diagonal")
```

```
mdl =

Gaussian mixture distribution with 3 components in 2 dimensions
Component 1:
Mixing proportion: 0.170351
Mean:    3.0528   -0.1856
```

```
Component 2:
Mixing proportion: 0.451237
Mean:   -2.5086    4.9429

Component 3:
Mixing proportion: 0.378411
Mean:   -2.2295   -4.1579
```

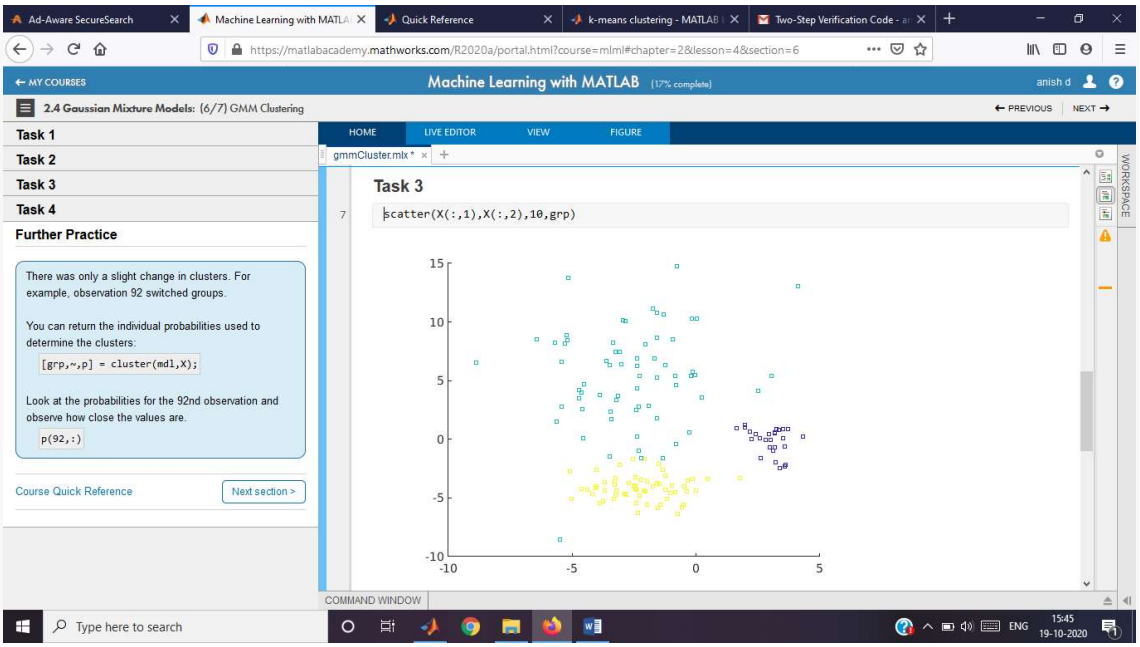
Task 2

```
grp=cluster(mdl,X)
```

```
grp = 154×1
     2
     2
     2
     2
     2
     2
     2
     2
     2
     2
```

Task 3

```
scatter(X(:,1),X(:,2),10,grp)
```



Further Practice

```
[grp,~,p]=cluster(mdl,X)
```

```
grp = 154×1
```

```
     2
     2
     2
     2
     2
     2
     2
     2
     2
     2
p = 154×3
    0.0000    1.0000    0.0000
    0.0000    1.0000    0.0000
    0.0000    1.0000    0.0000
    0.0000    1.0000    0.0000
    0.0000    1.0000    0.0000
    0.0000    1.0000    0.0000
    0.0230    0.9770    0.0000
    0.0000    1.0000    0.0000
    0.0004    0.9996    0.0000
    0.0000    1.0000    0.0000
```

```
p(92,:)
```

```
ans = 1×3
    0.0000    0.5182    0.4818
```

Basketball Players

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads and formats the data.

```
data = readtable("bball.txt");
data.pos = categorical(data.pos);
```

This code extracts and normalizes the columns of interest.

```
stats = data{:,[5 6 11:end]};
statsNorm = normalize(stats);
```

Task 1

```
gmModel=fitgmdist(statsNorm,2,"Replicates",5,"RegularizationValue",0.02)
```

```
Warning: Failed to converge in 100 iterations during replicate 1 for gmdistribution with 2
components
gmModel =

Gaussian mixture distribution with 2 components in 17 dimensions
```

```
[grp,~,gprob]=cluster(gmModel,statsNorm)
```

```
grp = 1117×1
     1
     2
     1
     1
     1
     2
     1
```

```
     2
     2
     2
gprob = 1117×2
    0.9963    0.0037
    0.0000    1.0000
    0.8087    0.1913
    0.9976    0.0024
    0.9998    0.0002
    0.0000    1.0000
    0.7642    0.2358
    0.0000    1.0000
    0.1327    0.8673
    0.0000    1.0000
```

This code performs PCA and plots the transformed data by group.

```
[pcs,scrs] = pca(statsNorm);
scatter3(scrs(:,1),scrs(:,2),scrs(:,3),10,grp)
view(110,40)
```

This code visualizes group separation.

```
gpsort = sortrows(gprob,1);
plot(gpsort)
xlabel("Point Ranking")
ylabel("Cluster Membership Probability")
legend("Cluster 1","Cluster 2")
```

Parallel Coordinates

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads the data.

```
load data

whos X
```

```
  Name        Size            Bytes  Class     Attributes

  X         124x4              3968  double
```

Task 1

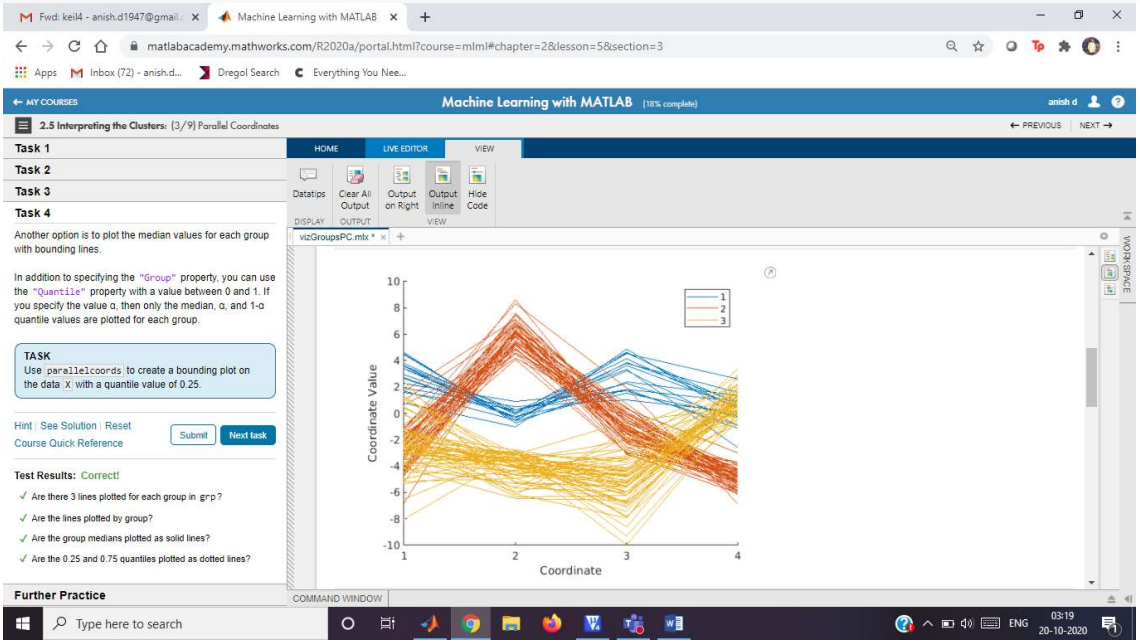
```
[grp,C]=kmeans(X,3)
```

```
grp = 124×1

     2
     2
     2
     2
     2
     2
     2
```

```
     2
     2
     2

C = 3×4

    3.0351   -0.1498    3.1239    0.1736
   -2.7202    6.0202   -1.1364   -4.9812
   -2.2648   -4.0838   -5.2336    1.0345
```

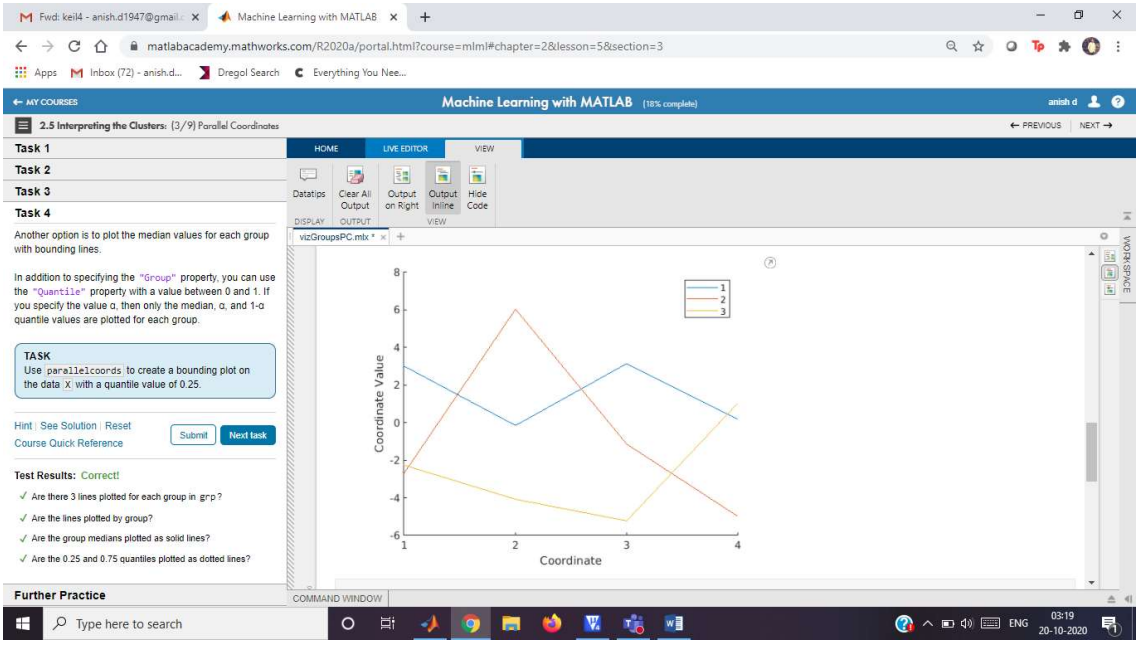
Task 2

```
parallelcoords(X,"Group",grp)
```



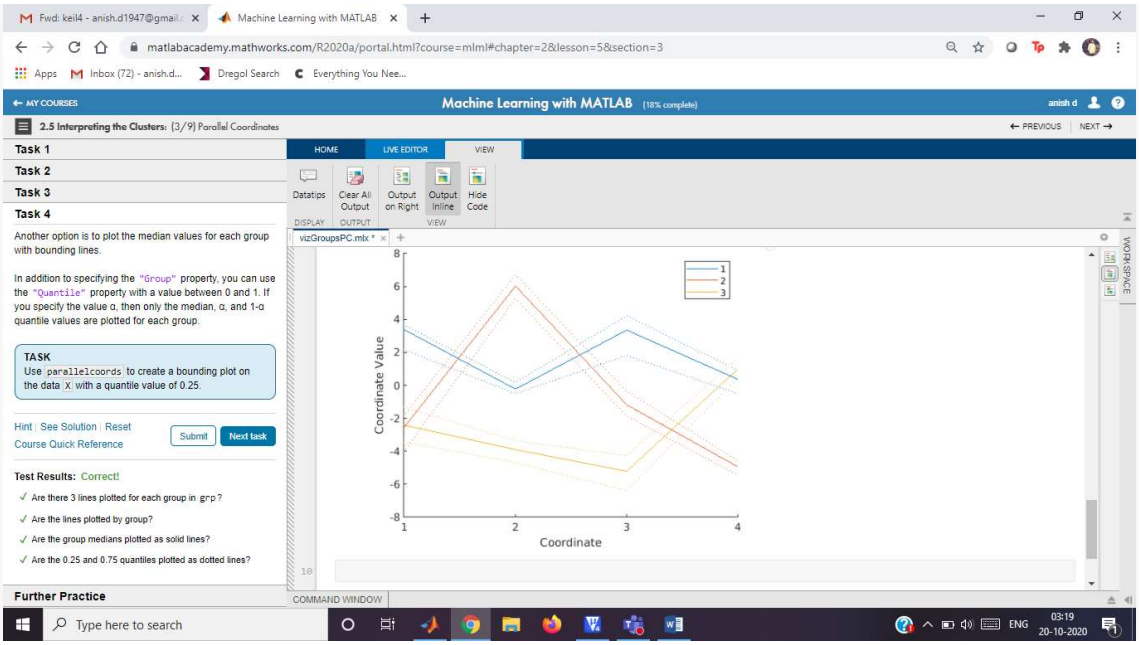
Task 3

```
parallelcoords(C,"Group",1:3)
```



Task 4

```
parallelcoords(X,"Group",grp,"Quantile",0.25)
```



Further Practice

Cross-Tabulation

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads the data.

```
load data2
```

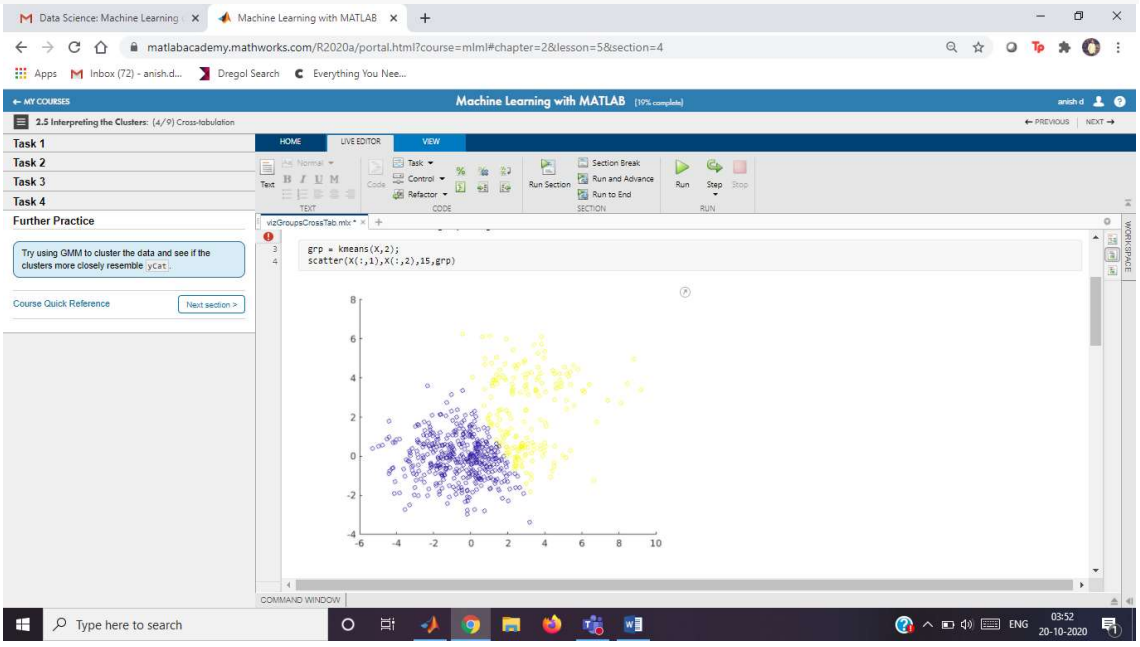
```
whos X yCat
```

```
  Name        Size              Bytes  Class          Attributes

  X         600x2                9600  double
  yCat      600x1                 818  categorical
```

This code clusters the data into 2 groups using `kmeans` and visualizes the clusters.

```
grp = kmeans(X,2);
```

```
scatter(X(:,1),X(:,2),15,grp)
```



Task 1

```
cts=crosstab(yCat,grp)
```

```
cts = 2×2
```

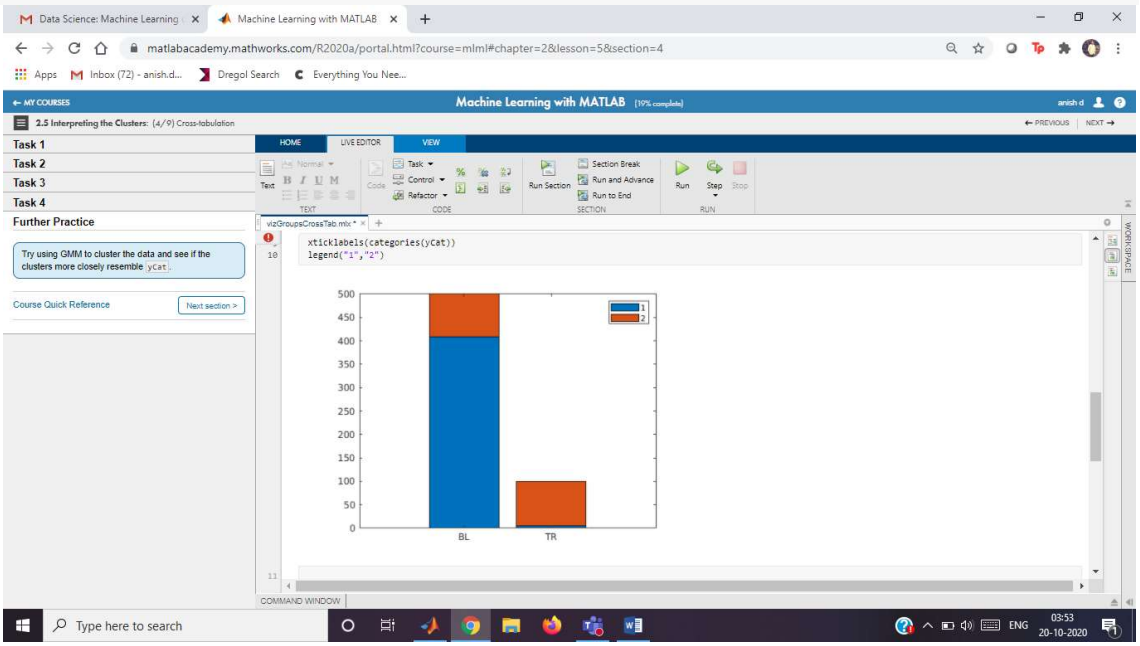
```
409    91
  5    95
```

Task 2

```
bar(cts,"stacked")
```

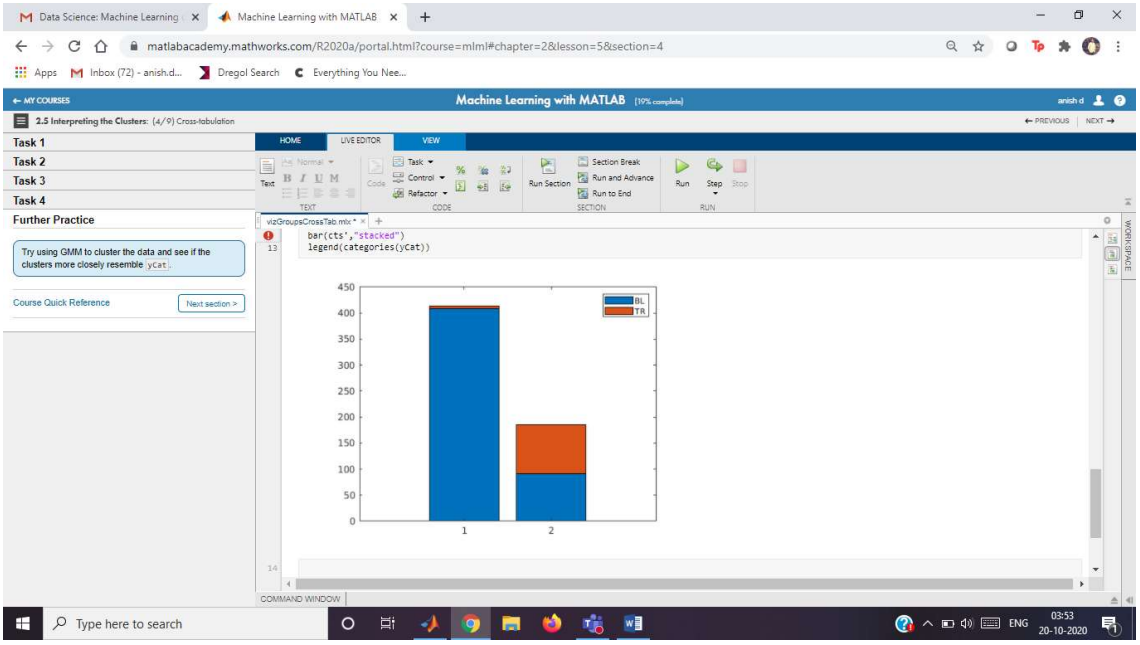
Task 3

```
xticklabels(categories(yCat))

legend("1","2")
```



Task 4

```
bar(cts','stacked")

legend(categories(yCat))
```



Further Practice

```
cluster=fitgmmodel(yCat,3)
```

```
Unrecognized function or variable 'fitgmmodel'.
```

Silhouette Plots

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads the data.

```
load data2
whos X yCat
```

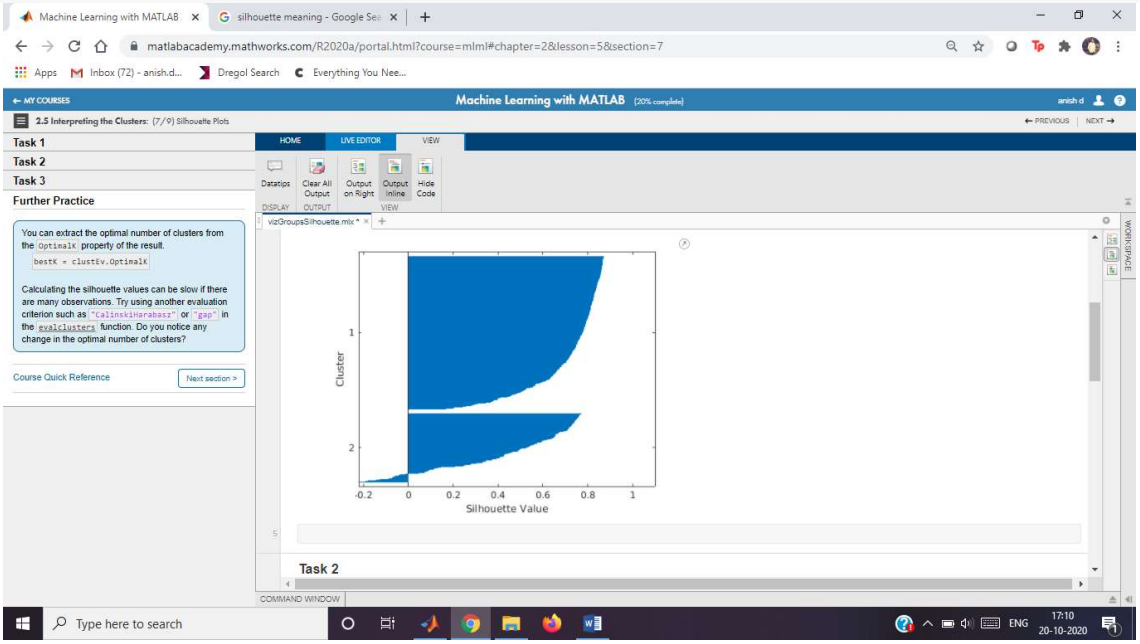
```
  Name        Size              Bytes  Class          Attributes

  X         600x2               9600  double
  yCat      600x1                818  categorical
```

This code clusters the data into 2 groups using `kmeans`.

```
grp = kmeans(X,2);
```

Task 1

```
silhouette(X,grp)
```

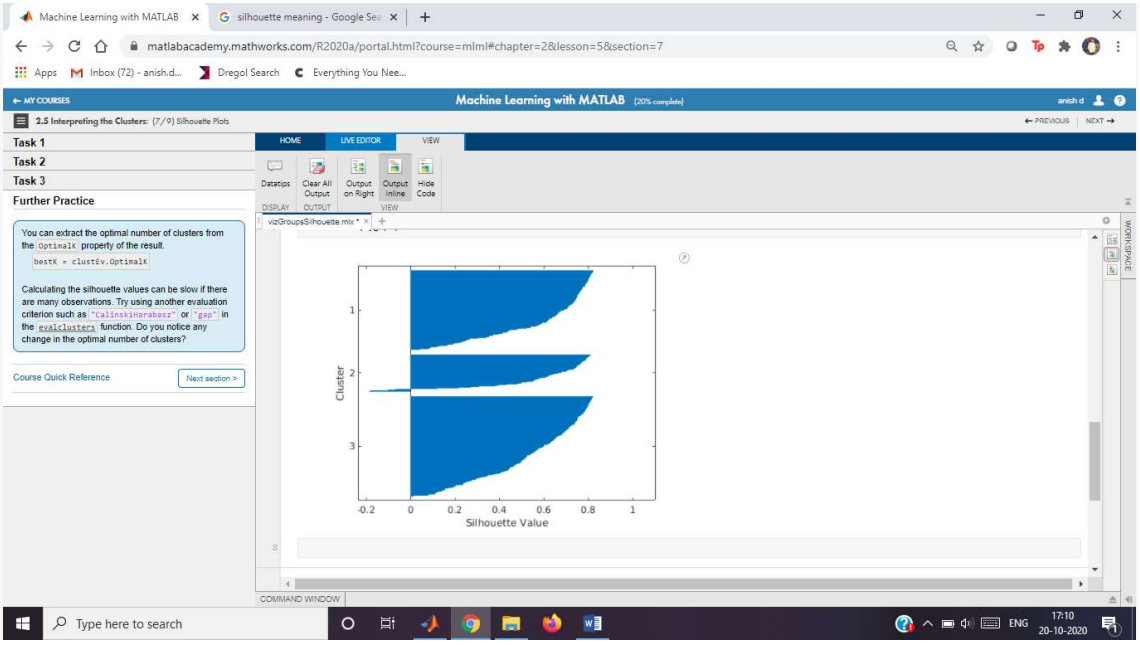


Task 2

```
grp3=kmeans(X,3)
```

```
grp3 = 600×1

     3
     1
     1
     3
     1
     3
     3
     1
     3
     3
```

```
silhouette(X,grp3)
```



Task 3

```
clustEv=evalclusters(X,"kmeans","silhouette","Klist",2:5)
```

```
clustEv =

  SilhouetteEvaluation with properties:

    NumObservations: 600
         InspectedK: [2 3 4 5]
    CriterionValues: [0.6354 0.5768 0.5148 0.5283]
           OptimalK: 2
```

Further Practice

```
bestK=clustEv.OptimalK
```

```
bestK = 2
```

Basketball Players

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads, formats, and normalizes the data.

```
data = readtable("bball.txt");
data.pos = categorical(data.pos);
stats = data{:,[5 6 11:end]};
labels = data.Properties.VariableNames([5 6 11:end]);
statsNorm = normalize(stats);
```

This code groups the data using a Gaussian mixture model.

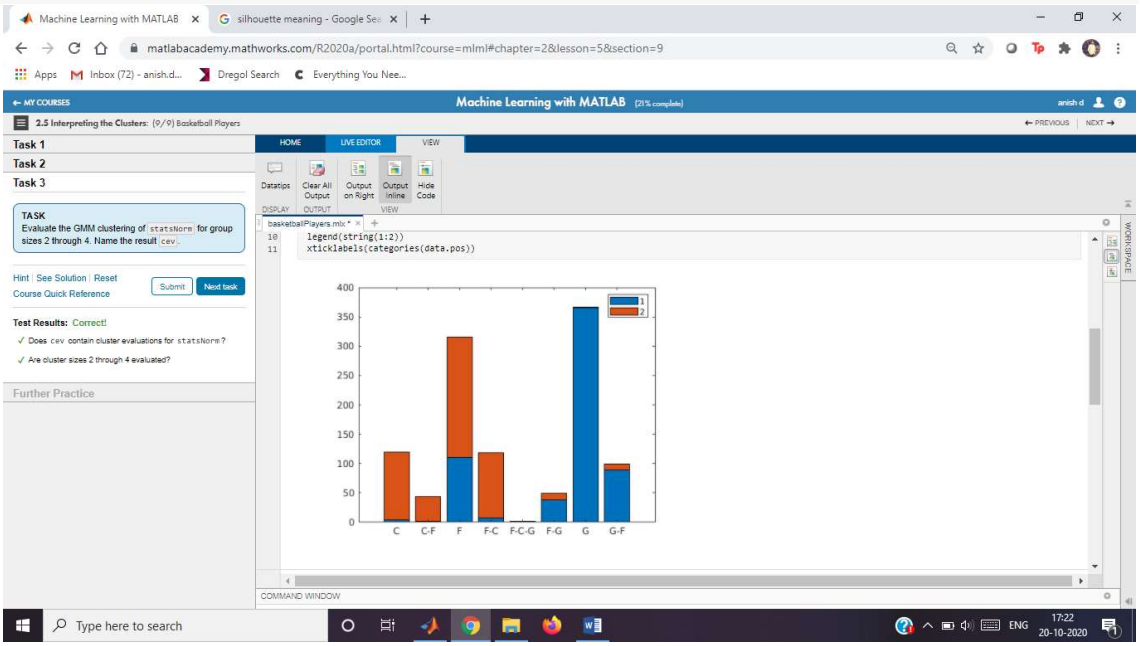
```
gmModel = fitgmdist(statsNorm,2,"Replicates",5,"RegularizationValue",0.02);
grp = cluster(gmModel,statsNorm);
```

Task 1

```
counts=crosstab(data.pos,grp)
```

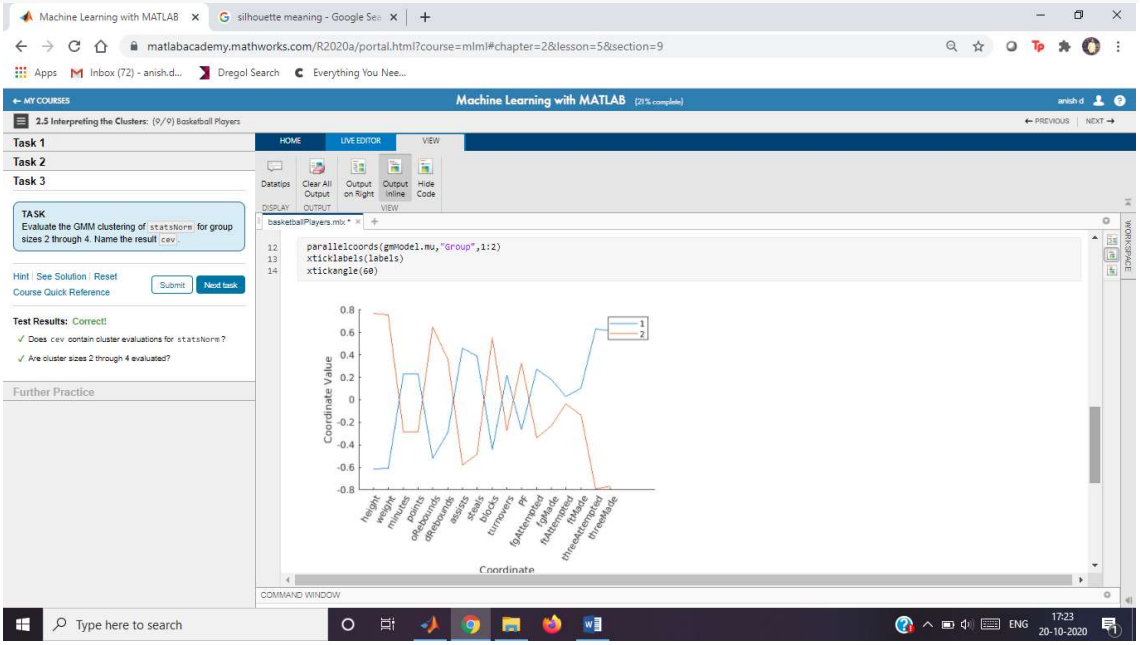
```
counts = 8×2

     4   115
     2    42
   110   206
     7   111
     2     0
    38    11
   366     1
    89    10
```

```
bar(counts,"stacked")
legend(string(1:2))
xticklabels(categories(data.pos))
```



Task 2

```
parallelcoords(gmModel.mu,"Group",1:2)

xticklabels(labels)

xtickangle(60)
```



Task 3

```
cev=evalclusters(statsNorm,"gmdistribution","DaviesBouldin","Klist",2:4)
```

```
Warning: Failed to converge in 100 iterations during replicate 5 for gmdistribution with 3 components

Warning: Failed to converge in 100 iterations during replicate 4 for gmdistribution with 4 components

cev =

  DaviesBouldinEvaluation with properties:

    NumObservations: 1117
        InspectedK: [2 3 4]
    CriterionValues: [1.0052 1.4527 1.3572]
           OptimalK: 2
```

```
optK=cev.OptimalK
```

```
optK = 2
```

Further Practice

Hierarchical Tree

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads the data.

```
load data
whos X
```

```
  Name        Size             Bytes  Class     Attributes

  X         124x4              3968  double
```

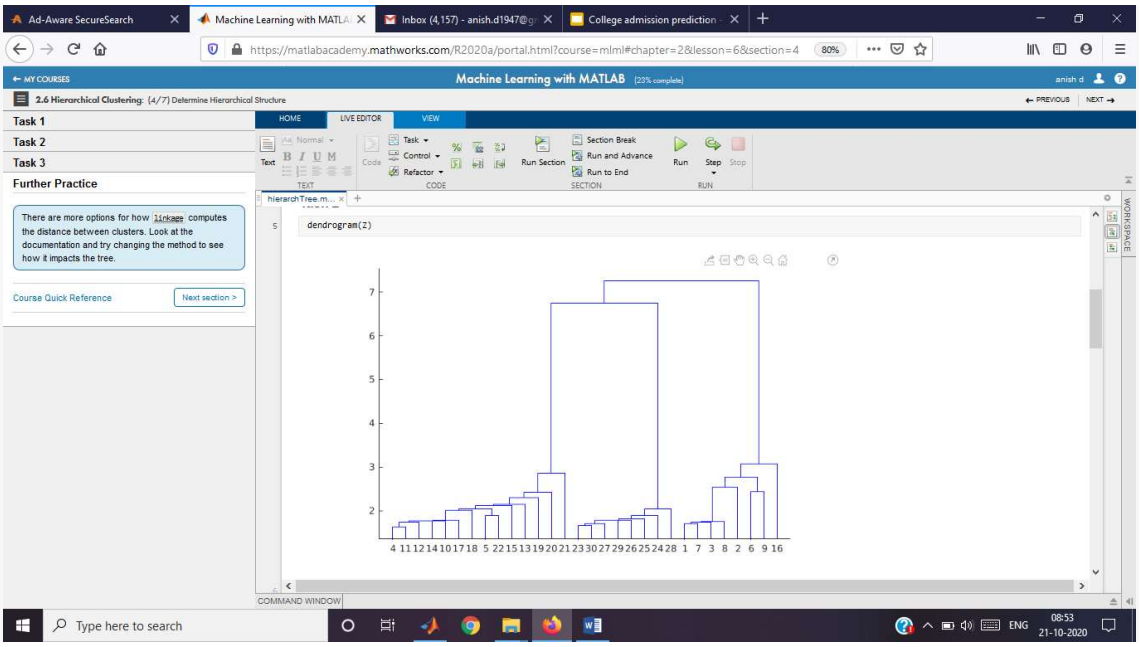
Task 1

```
Z=linkage(X)
```

```
Z = 123×3
   43.0000   49.0000    0.4461
   75.0000  104.0000    0.4839
   58.0000  107.0000    0.4987
   60.0000   77.0000    0.5282
   81.0000   97.0000    0.5393
   10.0000   18.0000    0.5517
    5.0000   34.0000    0.5743
   19.0000   42.0000    0.5966
   21.0000   36.0000    0.6059
  125.0000  131.0000    0.6123
```

Task 2

```
dendrogram(Z)
```

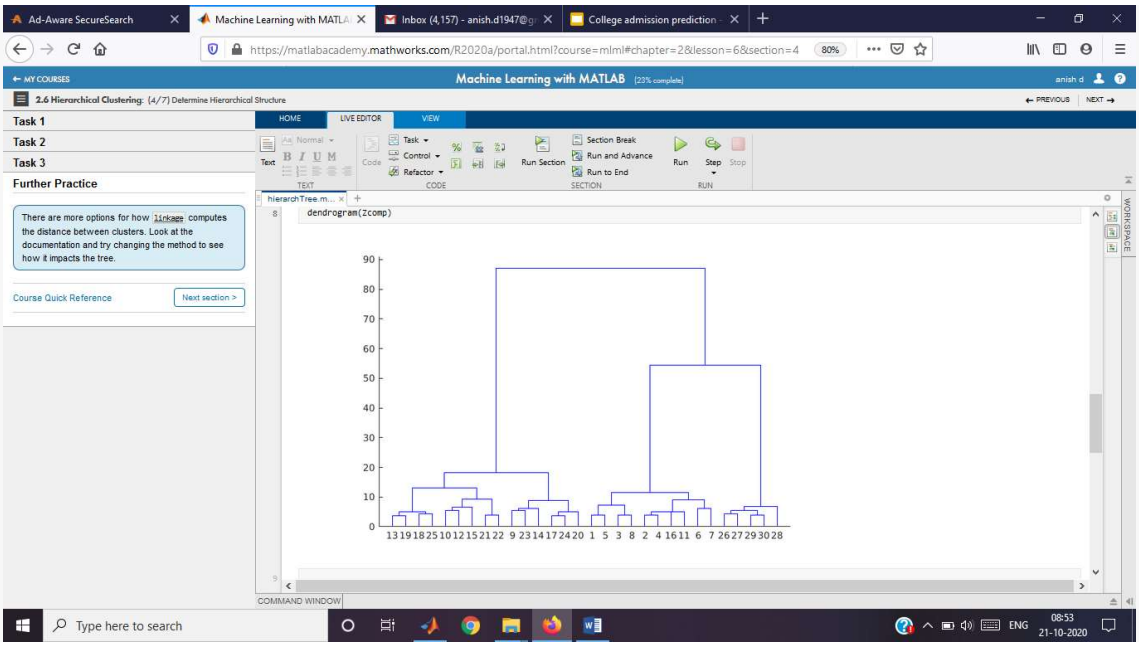


Task 3

```
Zcomp=linkage(X,"ward")
```

```
Zcomp = 123×3
   43.0000   49.0000    0.4461
   75.0000  104.0000    0.4839
   58.0000  107.0000    0.4987
   60.0000   77.0000    0.5282
   81.0000   97.0000    0.5393
   10.0000   18.0000    0.5517
    5.0000   34.0000    0.5743
   19.0000   42.0000    0.5966
   21.0000   36.0000    0.6059
   35.0000   51.0000    0.6213
```

```
dendrogram(Zcomp)
```



Further Practice

```
Zcomp2=linkage(X,"median")
```

```
Warning: Non-monotonic cluster tree -- the median linkage is probably not appropriate.
```

```
Zcomp2 = 123×3
   43.0000   49.0000    0.4461
   75.0000  104.0000    0.4839
   58.0000  107.0000    0.4987
   60.0000   77.0000    0.5282
   81.0000   97.0000    0.5393
   10.0000   18.0000    0.5517
    5.0000   34.0000    0.5743
   19.0000   42.0000    0.5966
   21.0000   36.0000    0.6059
   35.0000   51.0000    0.6213
```

```
dendrogram(Zcomp2)
```

Hierarchical Clustering

Instructions are in the task pane to the left. Complete and submit each task one at a time.

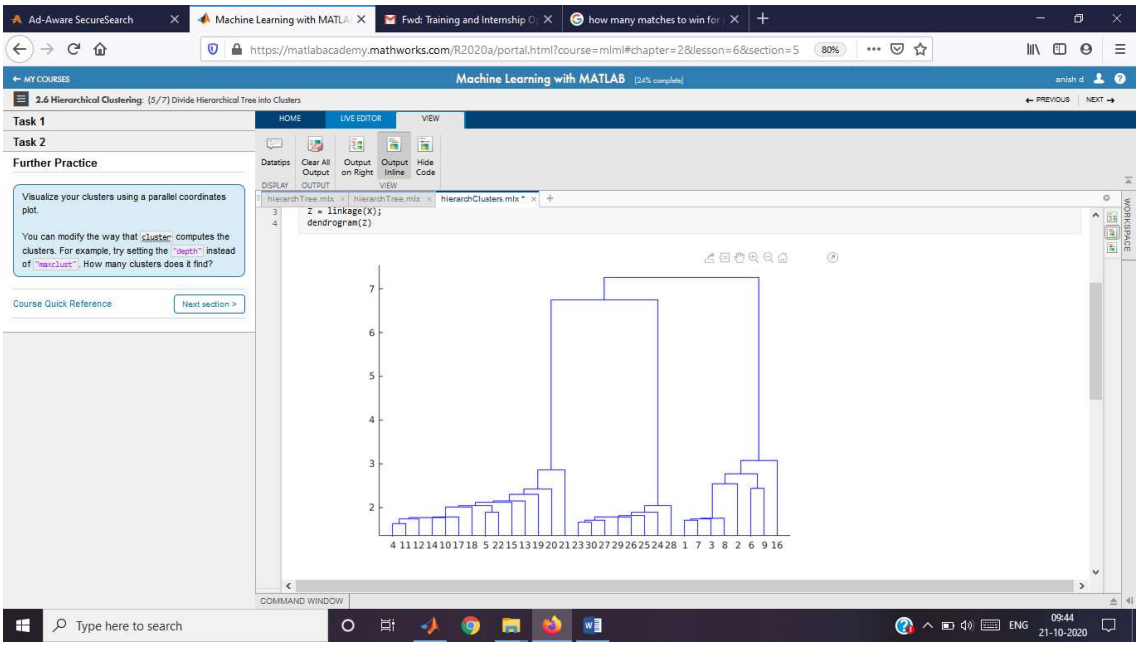
This code loads the data.

```
load data
whos X
```

```
  Name        Size             Bytes  Class     Attributes

  X         124x4               3968  double
```

This code creates a tree of hierarchical clusters.

```
Z = linkage(X);
dendrogram(Z)
```



Task 1

```
grp=cluster(Z,"maxclust",3)
```

```
grp = 124×1
     3
     3
     3
     3
     3
     3
     3
     3
     3
     3
     3
```

Task 2

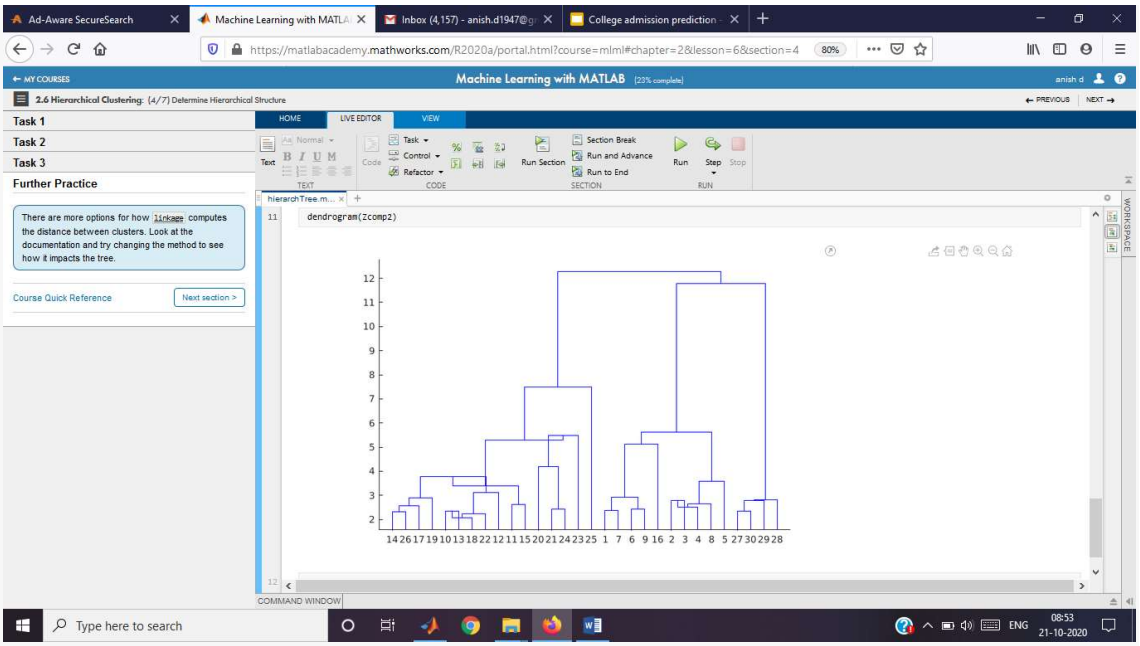
```
Y=pdist(X)
```

```
Y = 1×7626
    2.6056    3.2384    2.8874    0.8845    4.0136    2.7352    4.3374    3.4797    2.0891
2.1367    2.0517    2.4830    1.9490    1.5463    4.4767    1.8080    2.2917    2.8750
1.3134    1.3775    2.3632    1.4813    1.8872    2.1680    1.2320    2.5871    1.0661
3.0201    3.4046    2.4217    5.2930    2.3377    1.3094    2.1754    1.5251    3.5664
1.8041    1.7505    2.9649    2.0748    2.5062    1.2323    2.9339    1.8009    2.1460
1.7115    2.6073    1.2464    3.3412    2.7424
```

```
c=cophenet(Z,Y)
```

```
c = 0.9389
```

Further Practice



Basketball Players

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads, formats, and normalizes the data.

```
data = readtable("bball.txt");
data.pos = categorical(data.pos);
stats = data{:,[5 6 11:end]};
labels = data.Properties.VariableNames([5 6 11:end]);
statsNorm = normalize(stats);
```

This code extracts the data for the guard position (`G`).

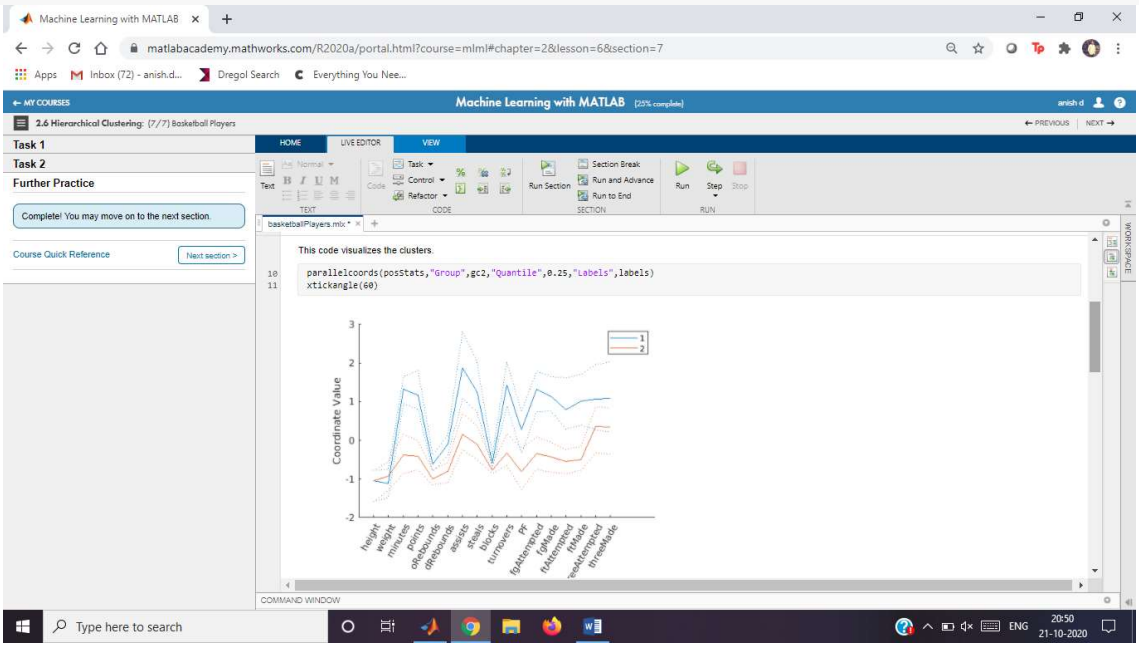
```
posStats = statsNorm(data.pos == "G",:);
```

Task 1

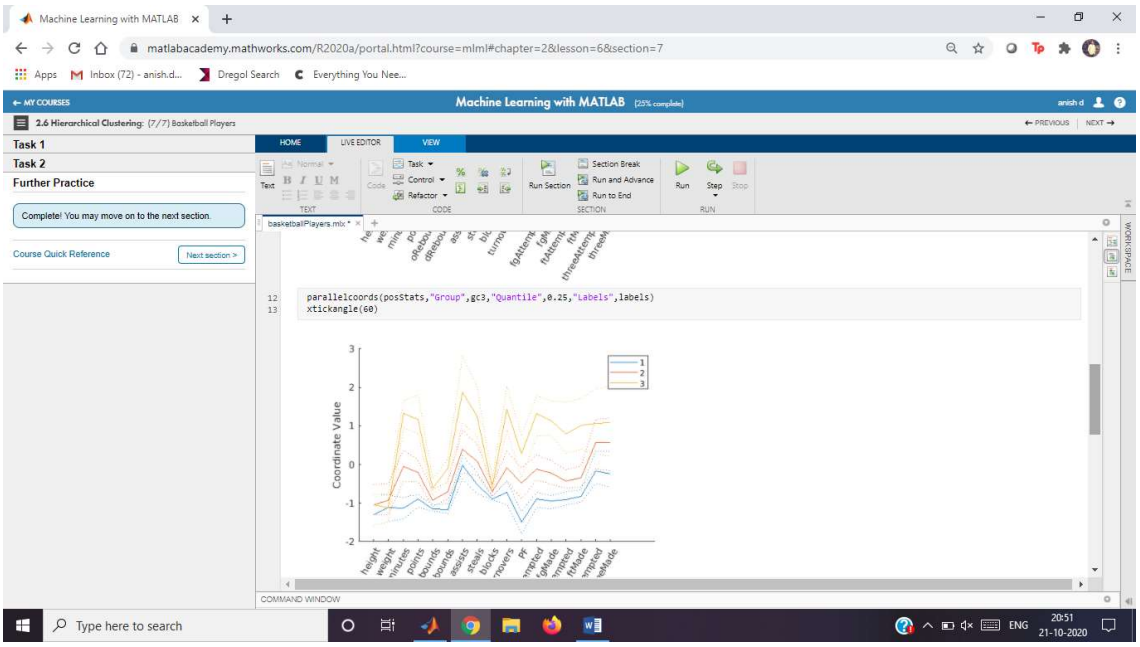
```
Z=linkage(posStats,"ward");
gc2=cluster(Z,"maxclust",2);
gc3=cluster(Z,"maxclust",3);
```

This code visualizes the clusters.

```
parallelcoords(posStats,"Group",gc2,"Quantile",0.25,"Labels",labels)

xtickangle(60)
```




```
parallelcoords(posStats,"Group",gc3,"Quantile",0.25,"Labels",labels)

xtickangle(60)
```

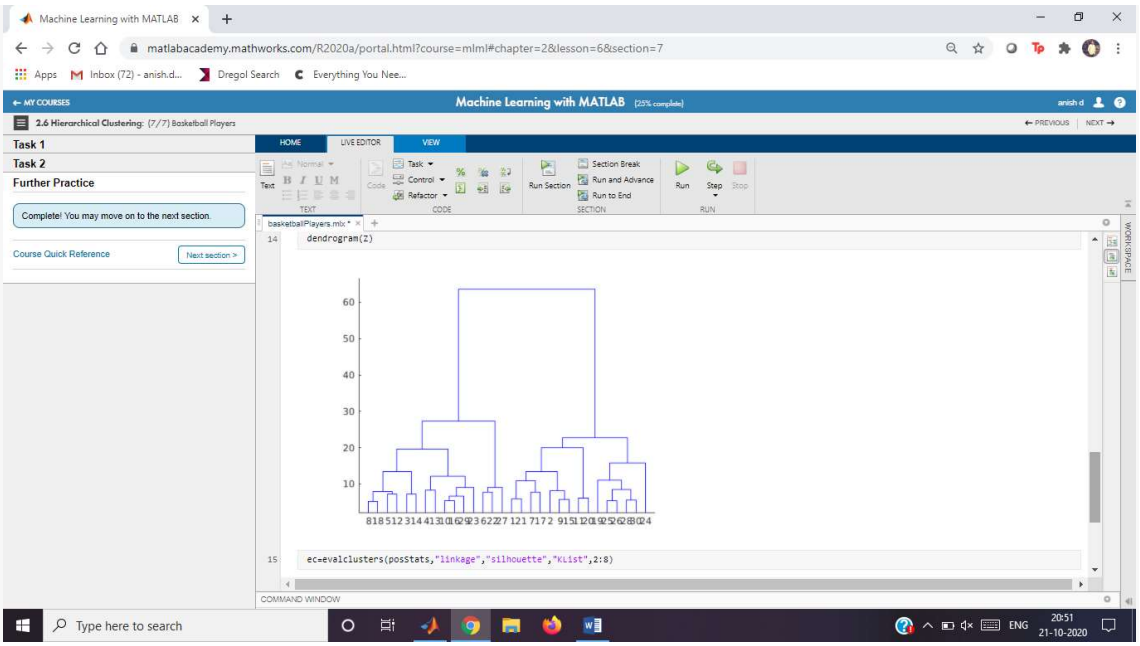


Task 2

```
dendrogram(Z)
```

```
ec=evalclusters(posStats,"linkage","silhouette","KList",2:8)
```

```
ec =

  SilhouetteEvaluation with properties:

    NumObservations: 367
         InspectedK: [2 3 4 5 6 7 8]
    CriterionValues: [0.5960 0.2794 0.2641 0.2348 0.2423 0.2516 0.2523]
           OptimalK: 2
```



Wine Color

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads and formats the data.

```
wineData = readtable("wineData.txt");

wineData.Color = categorical(wineData.Color);
```

This code extracts the numeric data.

```
numData = wineData{:,1:end-1};
```

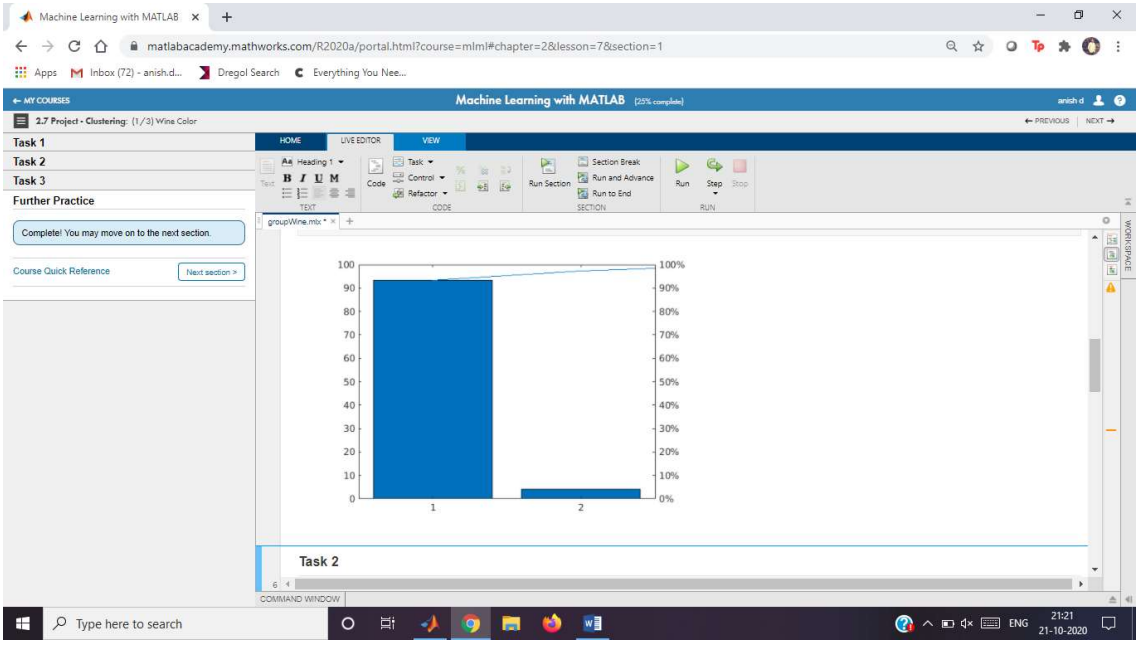
Task 1

```
[pcs,scrs,~,~,pexp]=pca(numData)
```

```
pcs = 12×12

   -0.0072   -0.0074   -0.0104    0.0229    0.9321    0.3532    0.0367   -0.0137   -0.0532
-0.0367   -0.0022   -0.0010
   -0.0012   -0.0015   -0.0042    0.0001    0.0086   -0.0043    0.6219   -0.4474    0.2669
0.5821   -0.0535   -0.0019
    0.0005    0.0000    0.0014    0.0021    0.0441    0.0332   -0.4042    0.4823    0.0597
```

```
0.7726   -0.0229    0.0000
    0.0415    0.0244   -0.0295    0.9949   -0.0500    0.0668    0.0063    0.0040   -0.0022
-0.0017    0.0005   -0.0004
   -0.0001   -0.0000   -0.0007    0.0001    0.0069   -0.0063    0.0353    0.0044    0.0317
0.0427    0.9978   -0.0112
    0.2265    0.9434   -0.2390   -0.0397    0.0060    0.0040    0.0002   -0.0007    0.0001
0.0003   -0.0001    0.0000
    0.9731   -0.2194    0.0618   -0.0334    0.0057    0.0040    0.0017   -0.0002    0.0000
-0.0001    0.0001    0.0000
    0.0000   -0.0000   -0.0001    0.0004    0.0013   -0.0009    0.0051    0.0029   -0.0004
0.0012    0.0109    0.9999
   -0.0007    0.0005    0.0001   -0.0072   -0.0399   -0.0267    0.5855    0.4972   -0.6366
0.0484   -0.0047   -0.0047
   -0.0007    0.0004    0.0005   -0.0017    0.0273   -0.0007    0.3225    0.5655    0.7184
-0.2421   -0.0266   -0.0025

scrs = 6448×12

   -3.0678    9.3734   32.5977   -2.4215    0.2119    0.3934    0.0380    0.0127   -0.1646
0.1659   -0.0024   -0.0003
    7.3196    3.5528   30.6864    5.8485    2.4961   -0.4909    0.0600   -0.0574   -0.1567
0.0507   -0.0176   -0.0008
    8.7630   31.6443   22.8187   -1.7619    0.0640    0.5840    0.0944   -0.1783   -0.1257
0.1246   -0.0192   -0.0010
   21.3467   -1.5987   29.3397   -3.4561    0.5879    0.4801    0.0748   -0.0397   -0.0934
0.0124   -0.0010   -0.0003
  -31.6879   -4.8063   28.8603   -1.8581   -1.6234   -0.0601    0.7002   -0.0855    0.2121
0.1241   -0.0200    0.0004
 -102.0967    5.1035   25.7605    2.3293    1.8636   -0.0173   -0.0522    0.2299    0.2191
0.0331    0.0074    0.0008
  -19.7925   10.4066   24.9020   -1.9422    0.3044   -1.5507    0.0337    0.0748    0.0164
-0.0199   -0.0070   -0.0005
  -11.5836    9.7293   25.0847    2.6223   -0.1091    0.2298    0.0545    0.0268   -0.1396
0.0269   -0.0078   -0.0010
  -31.8849    7.1653   25.7654   -1.9709   -0.4641   -0.1079    0.1669    0.0354   -0.0144
0.0388   -0.0250   -0.0016
   75.3687   12.9894   24.7293   -1.0703    0.6779   -0.3474    0.0733   -0.1096   -0.1476
0.0036   -0.0115    0.0006

pexp = 12×1

   93.2478
    3.9680
    2.2542
    0.4611
    0.0420
    0.0248
    0.0009
    0.0005
    0.0004
    0.0003
```

```
pareto(pexp)
```



Task 2

```
g=kmeans(numData,2,"Replicates",5);

gmModel=fitgmdist(numData,2,"Replicates",5);

g=cluster(gmModel,numData)
```

```
g = 6448×1

     2
     2
     2
     2
     1
     1
     2
     2
     2
     2
```

```
scatter(scrs(:,1),scrs(:,2),10,g)
```

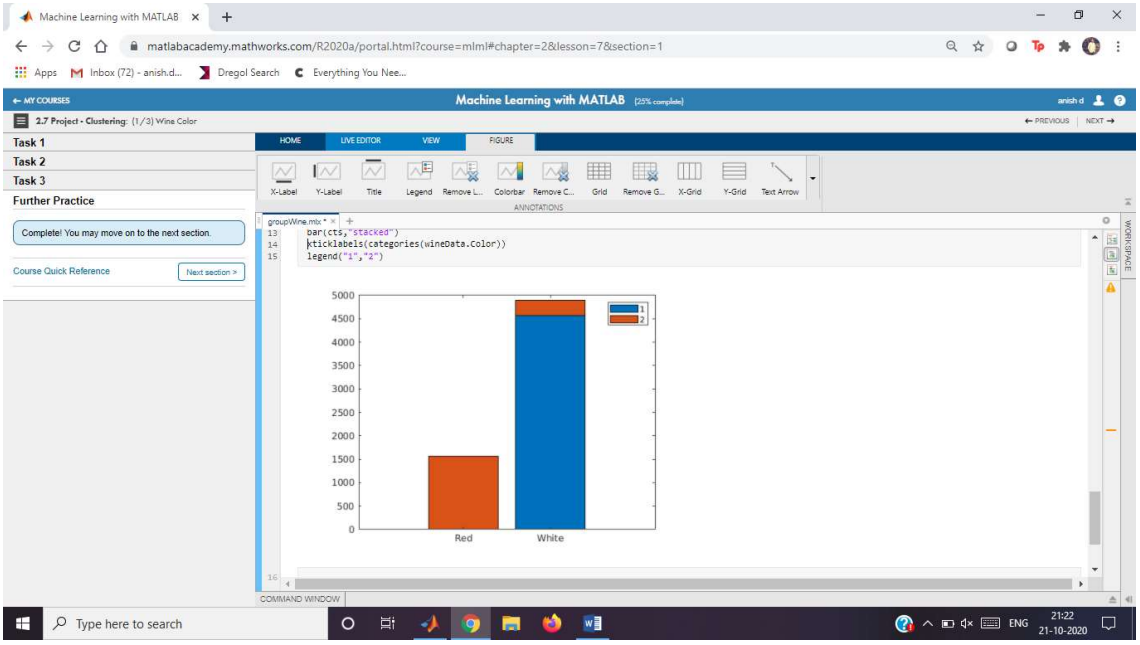
Task 3

```
cts = crosstab(wineData.Color,g)
```

```
cts = 2×2

        1546          14
         313        4575
```

```
bar(cts,"stacked")

xticklabels(categories(wineData.Color))
```

```
legend("1","2")
```



Corporate Bonds

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads the data and extracts the numeric data into a matrix.

```
bondData = readtable("bondData.txt");
bonds = bondData.Variables
```

```
bonds = 4082×4
    5.6500    4.0240    4.5634    9.0000
    6.2000    3.8030    5.0559   11.0000
    1.6400    1.5540    1.6116   11.0000
   10.2700   10.1930   25.3518    5.0000
    5.8300    4.4790    5.1488    8.0000
    6.4200    3.6750    4.7328    9.0000
    6.7800    5.7730    6.4308    7.0000
    6.5400    4.2420    5.7652    8.0000
    3.3600    2.8020    3.2064   10.0000
    7.2000    6.8750    7.1635    7.0000
```

Task 1

```
kGrp=kmeans(bonds,3,"Distance","cosine")
```

```
kGrp = 4082×1
     3
     3
     2
     1
     3
     3
     1
```

```
     3
     2
     1
```

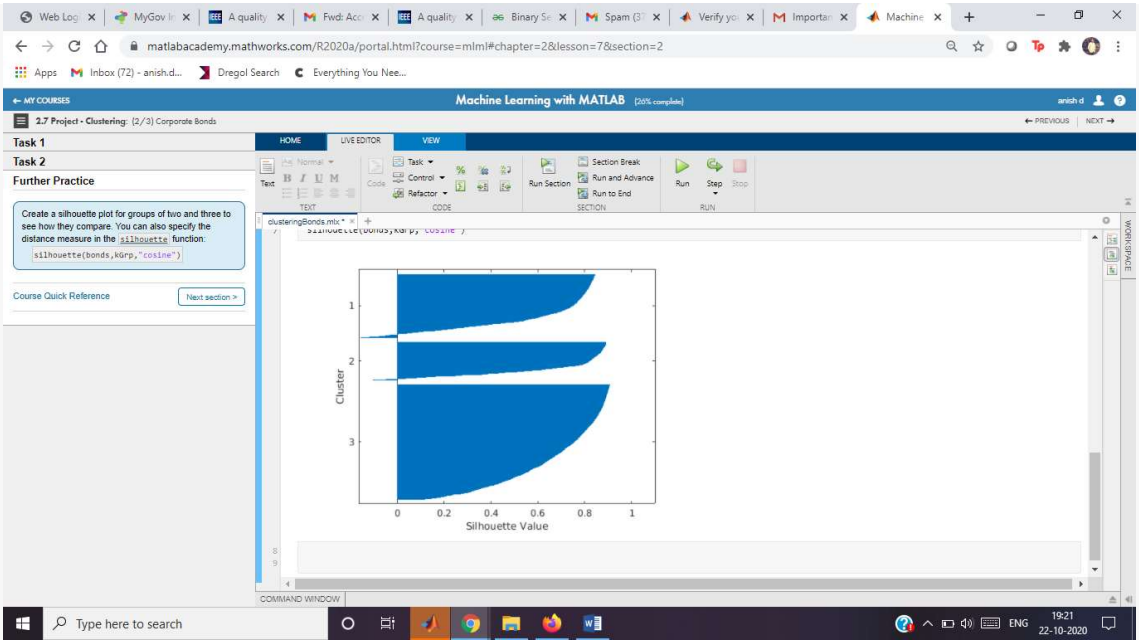
Task 2

```
eva=evalclusters(kGrp,"kmeans","silhouette","Klist",2:7,"Distance","cosine")
```

```
eva =
  SilhouetteEvaluation with properties:

    NumObservations: 4082
         InspectedK: [2 3 4 5 6 7]
    CriterionValues: [NaN NaN NaN NaN NaN NaN]
           OptimalK: NaN
```

Further Practice

```
silhouette(bonds,kGrp,"cosine")
```



Wheat Seed Kernels

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads the data, then extracts the numeric data into a matrix and the existing group numbers into a vector.

```
data = readtable("seeds.txt");
numData = data{:,1:end-1}
```

```
numData = 210×7
   15.2600   14.8400    0.8710    5.7630    3.3120    2.2210    5.2200
   14.8800   14.5700    0.8811    5.5540    3.3330    1.0180    4.9560
```

```
   14.2900   14.0900    0.9050    5.2910    3.3370    2.6990    4.8250
   13.8400   13.9400    0.8955    5.3240    3.3790    2.2590    4.8050
   16.1400   14.9900    0.9034    5.6580    3.5620    1.3550    5.1750
   14.3800   14.2100    0.8951    5.3860    3.3120    2.4620    4.9560
   14.6900   14.4900    0.8799    5.5630    3.2590    3.5860    5.2190
   14.1100   14.1000    0.8911    5.4200    3.3020    2.7000    5.0000
   16.6300   15.4600    0.8747    6.0530    3.4650    2.0400    5.8770
   16.4400   15.2500    0.8880    5.8840    3.5050    1.9690    5.5330
```

```
variety = data.variety
```

```
variety = 210×1
     1
     1
     1
     1
     1
     1
     1
     1
     1
     1
```

Task 1

```
grpK=kmeans(numData,3)
```

```
grpK = 210×1
     3
     3
     3
     3
     3
     3
     3
     3
     3
     3
```

```
gmmodel=fitgmdist(numData,3)
```

```
gmmodel =

Gaussian mixture distribution with 3 components in 7 dimensions
```

```
grpGMM=cluster(gmmodel,numData)
```

```
grpGMM = 210×1
     3
     3
     3
     3
     2
     3
     3
     3
     2
     2
```

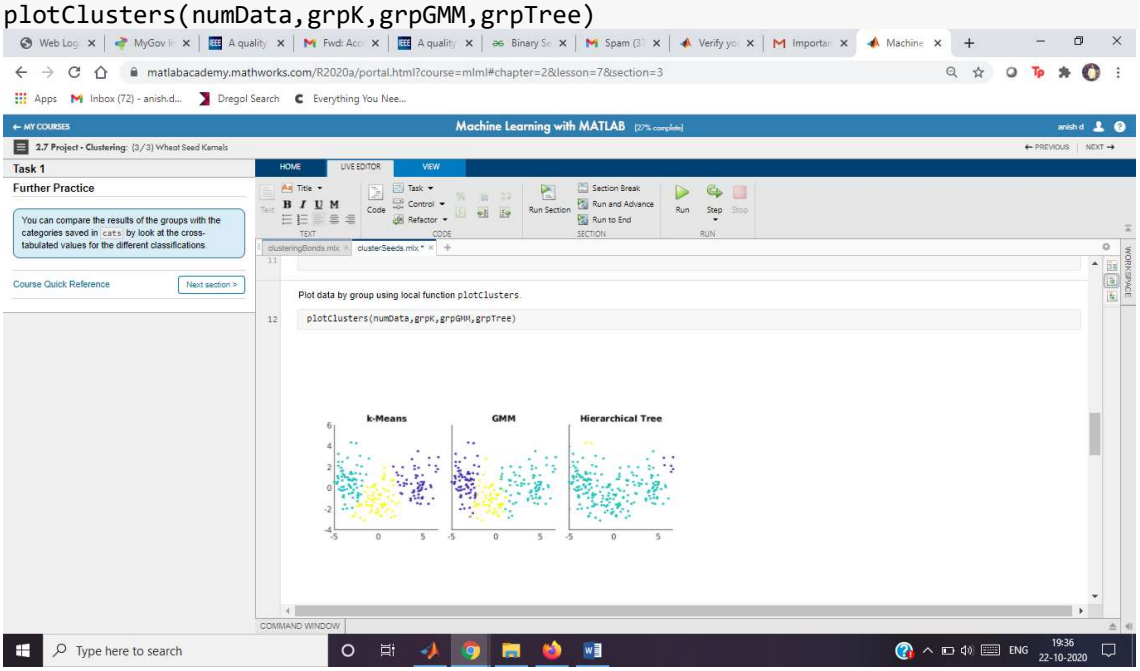
```
Z=linkage(numData)
```

```
Z = 209×3
  173.0000  207.0000    0.1174
  149.0000  199.0000    0.1339
  123.0000  134.0000    0.1358
    8.0000   29.0000    0.1790
  138.0000  139.0000    0.1914
   35.0000   50.0000    0.2060
   34.0000   68.0000    0.2076
  201.0000  210.0000    0.2117
   92.0000   93.0000    0.2120
   22.0000  214.0000    0.2165
```

```
grpTree=cluster(Z,"maxclust",3)
grpTree = 210×1
     2
     2
     2
     2
     2
     2
     2
     2
     2
     2
```

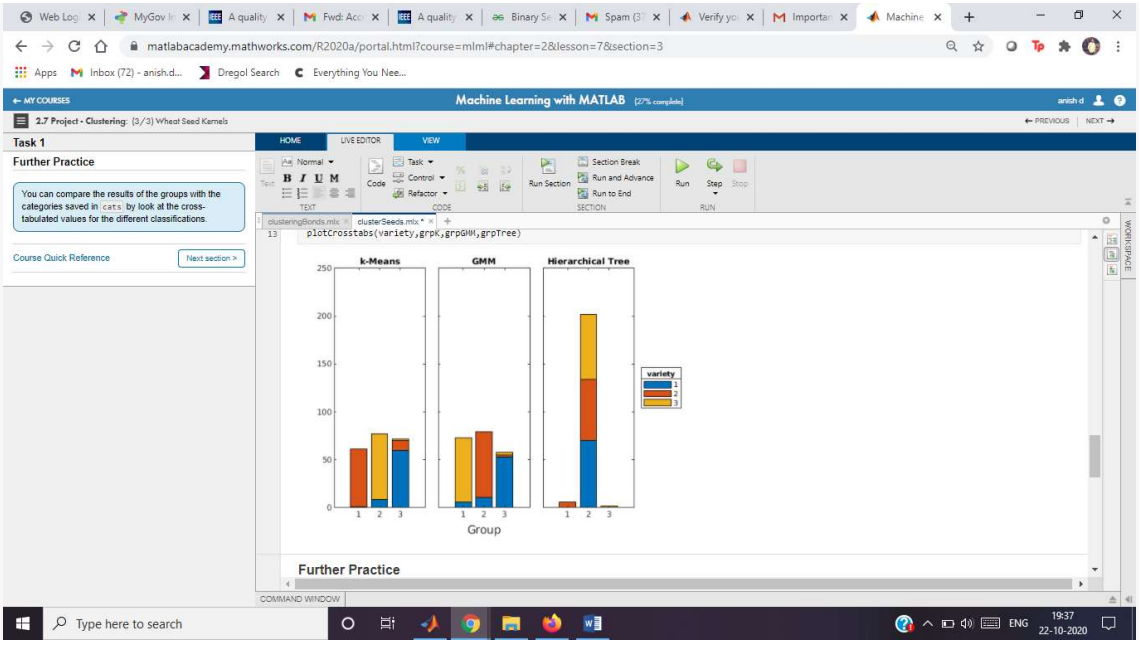
Plot data by group using local function `plotClusters`.

```
plotClusters(numData,grpK,grpGMM,grpTree)
```



Plot cross-tabulation values using local function `plotCrosstabs`.

```
plotCrosstabs(variety,grpK,grpGMM,grpTree)
```



Corporate Bonds

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads the data and extracts the numeric data into a matrix.

```
bondData = readtable("bondData.txt");
bonds = bondData.Variables;
```

Task 1

```
kGrp = kmeans(bonds,3,"Distance","cosine");
silhouette(bonds,kGrp,"cosine")
```

Task 2

```
eva = evalclusters(bonds,"kmeans","silhouette",...
    "KList",2:7,"Distance","cosine")


numClust = eva.OptimalK
kGrp = kmeans(bonds,numClust,"Distance","cosine");
silhouette(bonds,kGrp,"cosine")
```

Further Practice

```
classificationLearner()
```

Further Practice

This function creates side-by-side grouped scatter plots of the transformed data.

```
function plotClusters(data,gk,gm,gt)
[~,scrs] = pca(data);
sz = 8;


clf
tiledlayout(1,3,"TileSpacing","compact","Padding","compact");


ax(1) = nexttile;
scatter(scrs(:,1),scrs(:,2),sz,gk,"filled")
title("k-Means")


ax(2) = nexttile;
scatter(scrs(:,1),scrs(:,2),sz,gm,"filled")
title("GMM")


ax(3) = nexttile;
scatter(scrs(:,1),scrs(:,2),sz,gt,"filled")
title("Hierarchical Tree")


axis(ax,"square")
yticklabels(ax(2:end),[])
figure
end
```

This function creates side-by-side bar plots of the cross-tabluation values.

```
function plotCrosstabs(cats,gk,gm,gt)
clf
t = tiledlayout(1,3,"TileSpacing","compact","Padding","compact");


ax(1) = nexttile;
bar(crosstab(gk,cats),"stacked")
title("k-Means")
```

```
ax(2) = nexttile;
bar(crosstab(gm,cats),"stacked")
title("GMM")


ax(3) = nexttile;
bar(crosstab(gt,cats),"stacked")
title("Hierarchical Tree")


linkaxes(ax)
yticklabels(ax(2:end),[])
xlabel(t,"Group")
lgd = legend(string(1:3),"Location","eastoutside");
title(lgd,"variety")
figure
end
```

Nearest Neighbor Classification

Instructions are in the task pane to the left. Complete and submit each task one at a time.

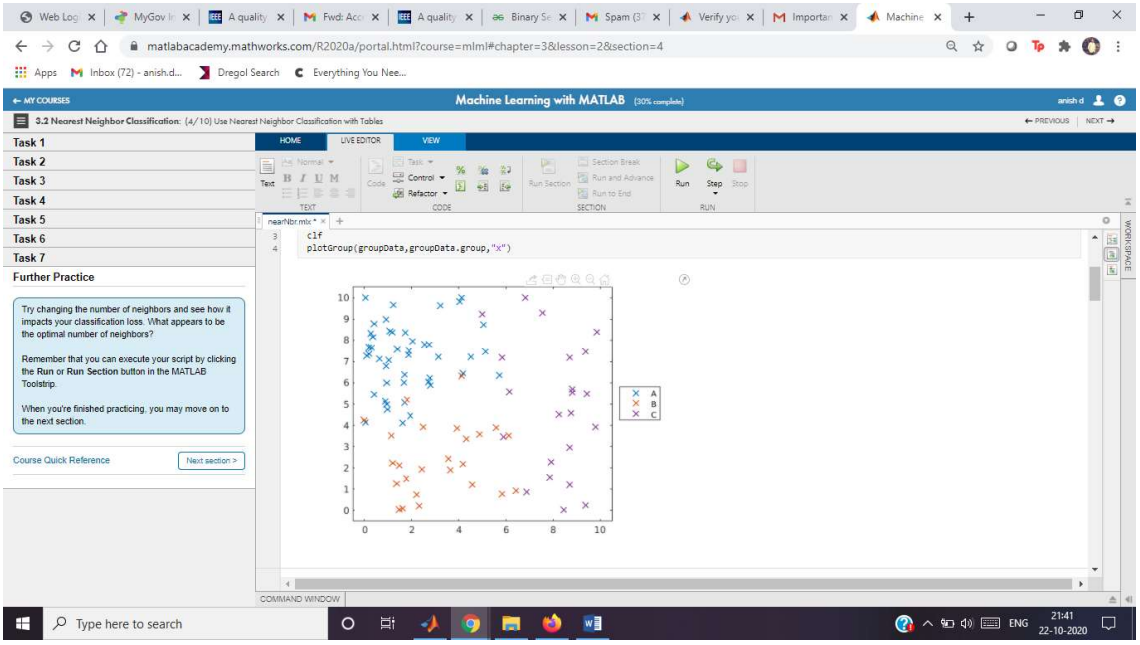
This code loads and displays the data.

```
load groups
whos
```

```
  Name            Size            Bytes  Class                Attributes

  dataTest       32x3              2270  table
  dataTrain      60x3              2746  table
  err10           1x1                 8  double
  err3            1x1                 8  double
  groupData      92x3              3290  table
  mdl             1x1              7030  ClassificationKNN
  pg10           32x1               352  categorical
  pg3            32x1               352  categorical
  pg7            32x1               352  categorical
```

This code uses the local function `plotGroup` to plot the data by group.

```
clf
plotGroup(groupData,groupData.group,"x")
```



Task 1

```
mdl=fitcknn(dataTrain,"group","NumNeighbors",3)
```

```
mdl =
  ClassificationKNN
            PredictorNames: {'x'  'y'}
              ResponseName: 'group'
     CategoricalPredictors: []
                ClassNames: [A    B    C]
            ScoreTransform: 'none'
           NumObservations: 60
                  Distance: 'euclidean'
              NumNeighbors: 3


  Properties, Methods
```

Task 2

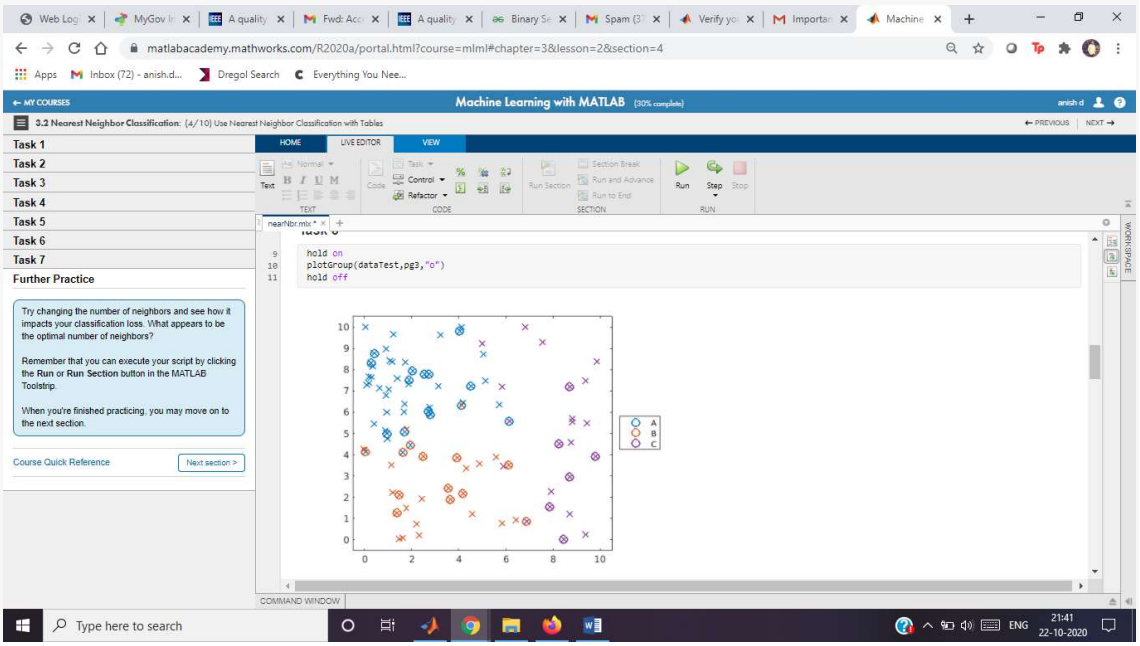
```
pg3=predict(mdl,dataTest)
```

```
pg3 = 32×1 categorical
A
A
A
A
A
A
A
A
A
B
```

Task 3

```
hold on
plotGroup(dataTest,pg3,"o")
```

```
hold off
```



Task 4

```
err3 = loss(mdl,dataTest)
err3 = 0.1880
```

Task 5

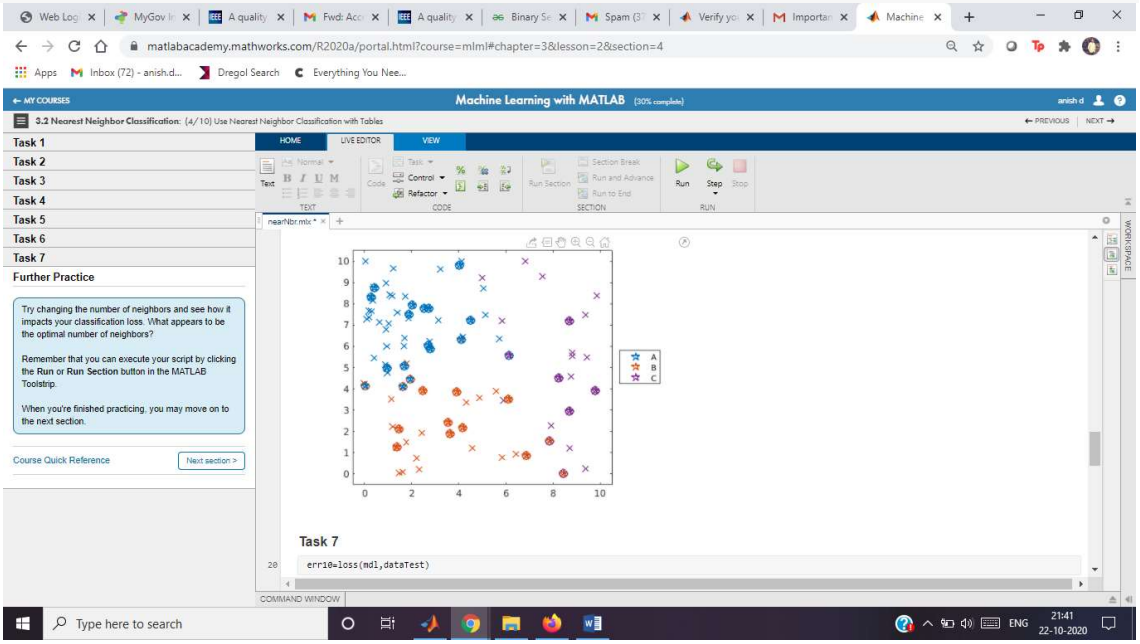
```
mdl.NumNeighbors=10
mdl =
  ClassificationKNN
             PredictorNames: {'x'  'y'}
               ResponseName: 'group'
      CategoricalPredictors: []
                 ClassNames: [A    B    C]
            ScoreTransform: 'none'
           NumObservations: 60
                  Distance: 'euclidean'
              NumNeighbors: 10


  Properties, Methods
```

Task 6

```
pg10=predict(mdl,dataTest)
pg10 = 32×1 categorical
A
A
A
A
A
A
```

```
A
A
A
A
```

```
hold on
plotGroup(dataTest,pg10,"p")
hold off
```



Task 7

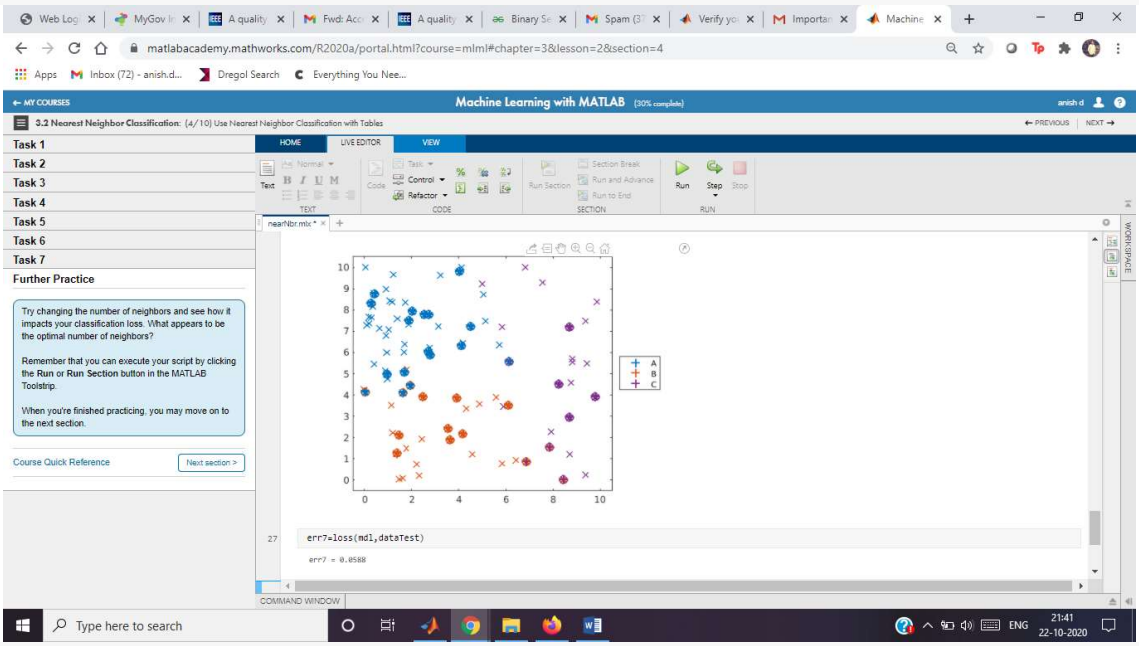
```
err10=loss(mdl,dataTest)
err10 = 0.1171
```

Further Practice

```
mdl.NumNeighbors=7
mdl =
  ClassificationKNN
             PredictorNames: {'x'  'y'}
               ResponseName: 'group'
      CategoricalPredictors: []
                 ClassNames: [A    B    C]
            ScoreTransform: 'none'
           NumObservations: 60
                  Distance: 'euclidean'
              NumNeighbors: 7


  Properties, Methods
pg7=predict(mdl,dataTest)
pg7 = 32×1 categorical
A
A
A
A
A
A
```

A
A
A
A

```
hold on
plotGroup(dataTest,pg7,"+")
hold off
```




```
err7=loss(mdl,dataTest)
err7 = 0.0588
```

This function uses `gscatter` to create a formatted plot of grouped data.

```
function plotGroup(data,grp,mkr)
    validateattributes(data,"table",{'nonempty','ncols',3})

    % Plot data by group
    colors = colororder;
    p = gscatter(data.x,data.y,grp,colors([1 2 4],:),mkr,9);

    % Format plot
    [p.LineWidth] = deal(1.5);
    legend("Location","eastoutside")
    xlim([-0.5 10.5])
    ylim([-0.5 10.5])
end
```

Heart Disease Analysis

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads the data.

```
heartData = readtable("heartDataNum.txt");
heartData.HeartDisease = categorical(heartData.HeartDisease);
```

This code partitions the data into training and test sets.

```
pt = cvpartition(heartData.HeartDisease,"HoldOut",0.3);
hdTrain = heartData(training(pt),:);
hdTest = heartData(test(pt),:);
```

Task 1

```
mdl = fitcknn(hdTrain,"HeartDisease");
```

Task 2

```
mdl.NumNeighbors=5
```

```
mdl =
  ClassificationKNN
           PredictorNames: {'Age'  'Cholesterol'  'ExerciseDuration'  'METs'
'RestingHeartRate'  'RestingSystolic'  'RestingDiastolic'  'MaxHeartRate'  'PeakExSystolic'
'PeakExDiastolic'  'InducedSTDep'}
             ResponseName: 'HeartDisease'
    CategoricalPredictors: []
               ClassNames: [false    true]
           ScoreTransform: 'none'
          NumObservations: 299
                 Distance: 'euclidean'
            NumNeighbors: 5


  Properties, Methods
```

Task 3

```
mdl.DistanceWeight="squaredinverse"
```

```
mdl =
  ClassificationKNN
           PredictorNames: {'Age'  'Cholesterol'  'ExerciseDuration'  'METs'
'RestingHeartRate'  'RestingSystolic'  'RestingDiastolic'  'MaxHeartRate'  'PeakExSystolic'
'PeakExDiastolic'  'InducedSTDep'}
             ResponseName: 'HeartDisease'
    CategoricalPredictors: []
               ClassNames: [false    true]
           ScoreTransform: 'none'
          NumObservations: 299
                 Distance: 'euclidean'
            NumNeighbors: 5


  Properties, Methods
```

Calculate the loss for the training and test sets

```
errTrain = resubLoss(mdl);
```

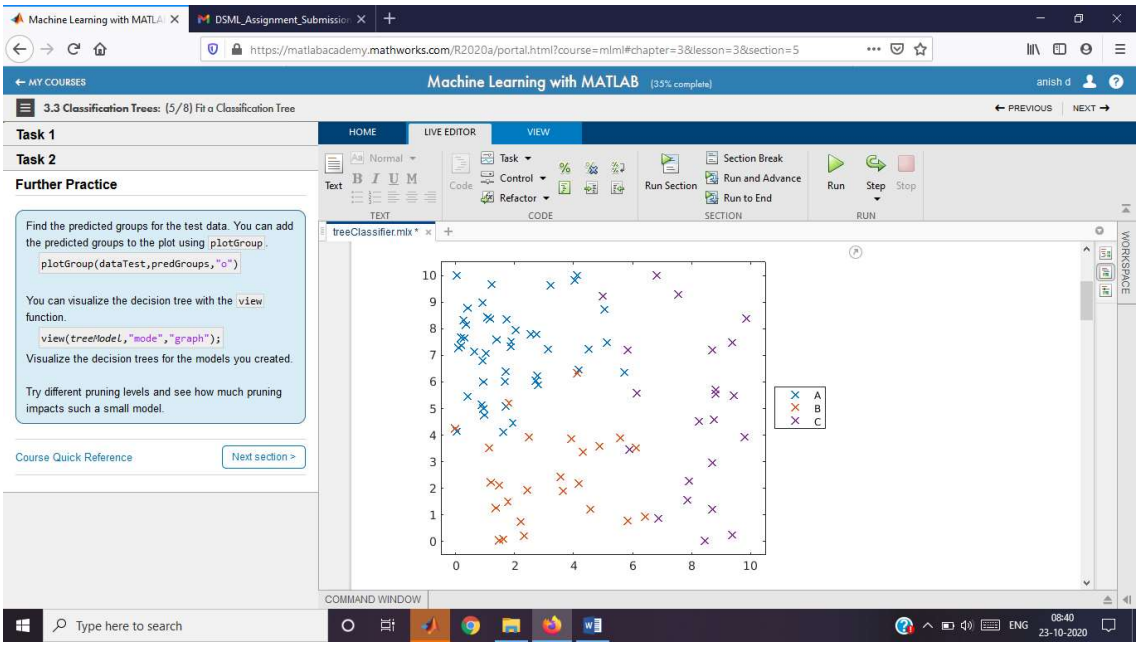
```
errTest = loss(mdl,hdTest);
disp("Training Error: " + errTrain)
```
```
Training Error: 0
```
```
disp("Test Error: " + errTest)
```
```
Test Error: 0.39873
```

Classification Trees

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads, partitions, and plots the data.

```
load groups

rng(0)

cvpt = cvpartition(groupData.group,"Holdout",0.35);

dataTrain = groupData(training(cvpt),:);

dataTest = groupData(test(cvpt),:);

plotGroup(groupData,groupData.group,"x")
```



Task 1

```
mdl=fitctree(dataTrain,"group")
```

Calculate loss.

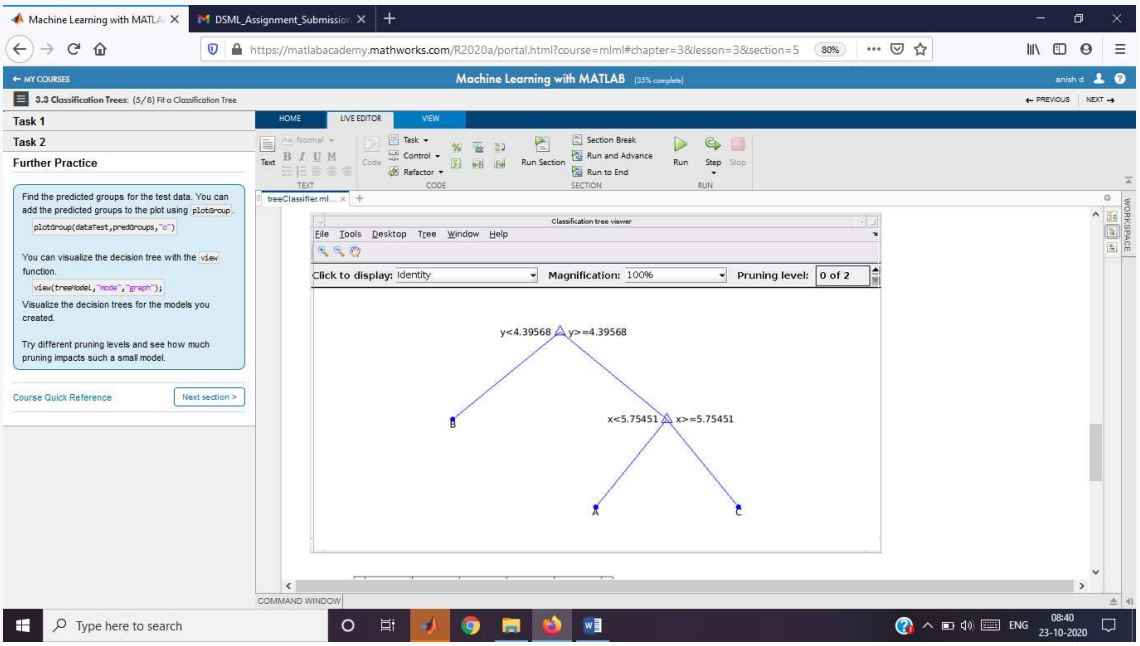
```
errTree = loss(mdl,dataTest);

disp("Classification Tree Loss: " + errTree)
```

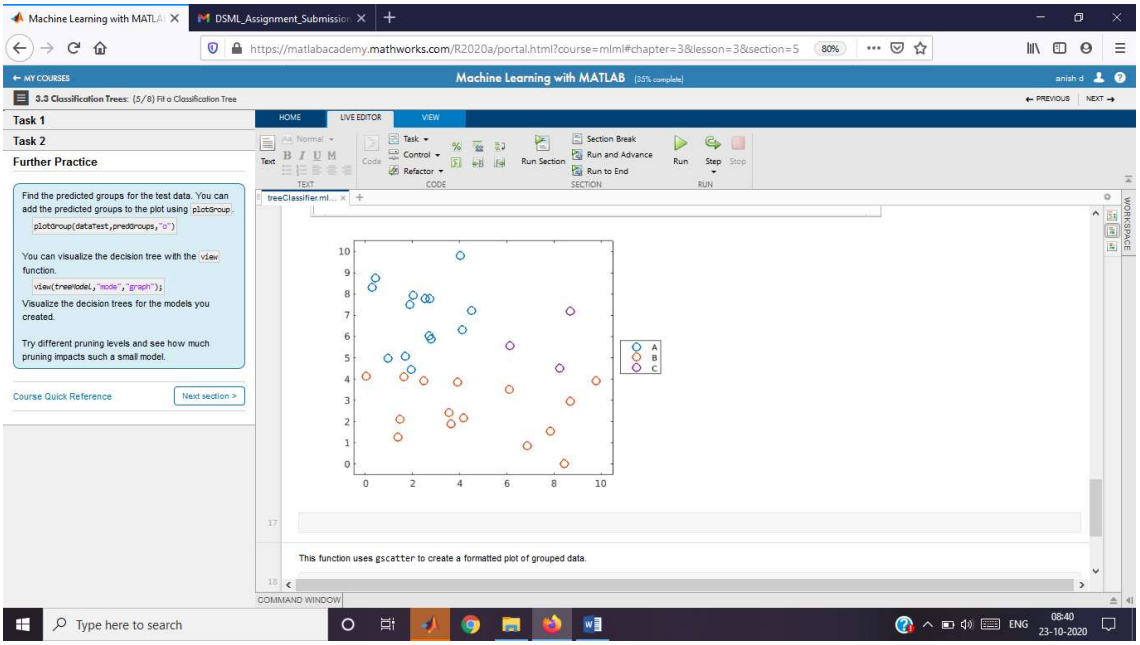
Task 2

```
mdlPruned=prune(mdl,"level",1)

errPruned=loss(mdlPruned,dataTest)
```

Further Practice

```
predGroups=predict(mdlPruned,dataTest)

plotGroup(dataTest,predGroups,"o")

view(mdlPruned,"mode","graph")
```





This function uses gscatter to create a formatted plot of grouped data.

```
function plotGroup(data,grp,mkr)
    validateattributes(data,"table",{'nonempty','ncols',3})

    % Plot data by group
    colors = colororder;
    p = gscatter(data.x,data.y,grp,colors([1 2 4],:),mkr,9);

    % Format plot
    [p.LineWidth] = deal(1.5);
    legend("Location","eastoutside")
    xlim([-0.5 10.5])
    ylim([-0.5 10.5])
end
```

Heart Disease Analysis (numeric data)

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads and formats the data.

```
heartData = readtable("heartDataNum.txt");
heartData.HeartDisease = categorical(heartData.HeartDisease);
```

This code partitions the data into training and test sets.

```
pt = cvpartition(heartData.HeartDisease,"HoldOut",0.3);
hdTrain = heartData(training(pt),:);
hdTest = heartData(test(pt),:);
```

Task 1

```
mdl=fitctree(hdTrain,"HeartDisease")
```

```
mdl =

  ClassificationTree
           PredictorNames: {'Age'  'Cholesterol'  'ExerciseDuration'  'METs'
'RestingHeartRate'  'RestingSystolic'  'RestingDiastolic'  'MaxHeartRate'  'PeakExSystolic'
'PeakExDiastolic'  'InducedSTDep'}
             ResponseName: 'HeartDisease'
    CategoricalPredictors: []
               ClassNames: [false    true]
           ScoreTransform: 'none'
          NumObservations: 299


  Properties, Methods
```

Task 2

```
mdl=prune(mdl,"level",3)
```

```
mdl =

  ClassificationTree
           PredictorNames: {'Age'  'Cholesterol'  'ExerciseDuration'  'METs'
'RestingHeartRate'  'RestingSystolic'  'RestingDiastolic'  'MaxHeartRate'  'PeakExSystolic'
'PeakExDiastolic'  'InducedSTDep'}
              ResponseName: 'HeartDisease'
    CategoricalPredictors: []
               ClassNames: [false    true]
           ScoreTransform: 'none'
          NumObservations: 299


  Properties, Methods
```

Calculate loss for training and test sets

```
errTrain = resubLoss(mdl);

errTest = loss(mdl,hdTest);

disp("Training Error: " + errTrain)
```

```
Training Error: 0.12709
```

```
disp("Test Error: " + errTest)
```

```
Test Error: 0.33554
```

Heart Disease Analysis

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads and formats the data.

```
heartData = readtable("heartDataAll.txt");

heartData = convertvars(heartData,12:22,"categorical");
```

This code partitions the data into training and test sets.

```
pt = cvpartition(heartData.HeartDisease,"HoldOut",0.3);

hdTrain = heartData(training(pt),:);

hdTest = heartData(test(pt),:);
```

Task 1

```
mdl=fitctree(hdTrain,"HeartDisease")
```

```
mdl =

  ClassificationTree
           PredictorNames: {1×21 cell}
             ResponseName: 'HeartDisease'
    CategoricalPredictors: [12 13 14 15 16 17 18 19 20 21]
               ClassNames: [false    true]
           ScoreTransform: 'none'
          NumObservations: 299


  Properties, Methods
```

```
mdlpruned=prune(mdl,"level",3)
```

```
mdlpruned =

  ClassificationTree
           PredictorNames: {1×21 cell}
             ResponseName: 'HeartDisease'
    CategoricalPredictors: [12 13 14 15 16 17 18 19 20 21]
               ClassNames: [false    true]
           ScoreTransform: 'none'
          NumObservations: 299


  Properties, Methods
```

```
mdlloss=loss(mdl,hdTrain)
```

```
mdlloss = 0.0702
```

```
mdlprunedLoss=loss(mdl,hdTrain)
```

```
mdlprunedLoss = 0.0702
```

Calculate loss for training and test sets

```
errTrain = resubLoss(mdl);

errTest = loss(mdl,hdTest);

disp("Training Error: " + errTrain)
```

```
Training Error: 0.070234
```

```
disp("Test Error: " + errTest)
```

```
Test Error: 0.36005
```

Naive Bayes Classification

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads, partitions, and plots the data.

```
load groups

rng(0)

pt = cvpartition(groupData.group,"Holdout",0.35);

dataTrain = groupData(training(pt),:);
```

```
dataTest = groupData(test(pt),:);

plotGroup(groupData,groupData.group,"x")
```

Tasks 1 & 2

```
mdlNB=fitcnb(dataTrain,"group")
```

```
mdlNB =

  ClassificationNaiveBayes
            PredictorNames: {'x'  'y'}
              ResponseName: 'group'
     CategoricalPredictors: []
                ClassNames: [A    B    C]
           ScoreTransform: 'none'
          NumObservations: 60
         DistributionNames: {'normal'  'normal'}
    DistributionParameters: {3×2 cell}


  Properties, Methods
```

```
mdlNB=fitcnb(dataTrain,"group","DistributionNames","kernel")
```

```
mdlNB =

  ClassificationNaiveBayes
            PredictorNames: {'x'  'y'}
              ResponseName: 'group'
     CategoricalPredictors: []
                ClassNames: [A    B    C]
           ScoreTransform: 'none'
          NumObservations: 60
         DistributionNames: {'kernel'  'kernel'}
    DistributionParameters: {3×2 cell}
                    Kernel: {'normal'  'normal'}
                   Support: {'unbounded'  'unbounded'}
                     Width: [3×2 double]


  Properties, Methods
```

Predict response and calculate loss.

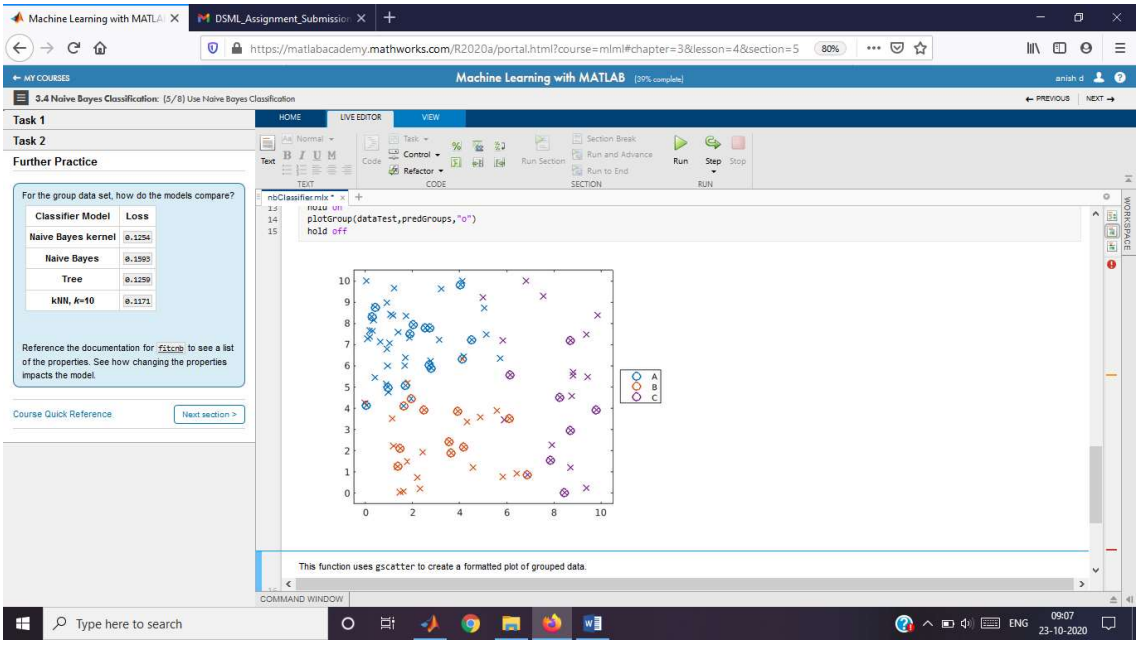
```
predGroups = predict(mdlNB,dataTest);

errSVM = loss(mdlNB,dataTest);

disp("Naive Bayes Loss: " + errSVM)
```

```
Naive Bayes Loss: 0.12546
```

Plot predictions.

```
hold on

plotGroup(dataTest,predGroups,"o")
```

```
hold off
```



This function uses `gscatter` to create a formatted plot of grouped data.

```
function plotGroup(data,grp,mkr)
    validateattributes(data,"table",{'nonempty','ncols',3})

    % Plot data by group
    colors = colororder;
    p = gscatter(data.x,data.y,grp,colors([1 2 4],:),mkr,9);

    % Format plot
    [p.LineWidth] = deal(1.5);
    legend("Location","eastoutside")
    xlim([-0.5 10.5])
    ylim([-0.5 10.5])
end
```

Heart Disease Analysis (numeric data)

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads and formats the data.

```
heartData = readtable("heartDataNum.txt");
heartData.HeartDisease = categorical(heartData.HeartDisease);
```

This code partitions the data into training and test sets.

```
pt = cvpartition(heartData.HeartDisease,"HoldOut",0.3);

hdTrain = heartData(training(pt),:);

hdTest = heartData(test(pt),:);
```

Tasks 1 & 2

```
mdl=fitcnb(hdTrain,"HeartDisease")
```

```
mdl =

  ClassificationNaiveBayes
           PredictorNames: {'Age'  'Cholesterol'  'ExerciseDuration'  'METs'
'RestingHeartRate'  'RestingSystolic'  'RestingDiastolic'  'MaxHeartRate'  'PeakExSystolic'
'PeakExDiastolic'  'InducedSTDep'}
              ResponseName: 'HeartDisease'
     CategoricalPredictors: []
                ClassNames: [false    true]
           ScoreTransform: 'none'
          NumObservations: 299
         DistributionNames: {'normal'  'normal'  'normal'  'normal'  'normal'  'normal'
'normal'  'normal'  'normal'  'normal'  'normal'}
    DistributionParameters: {2×11 cell}


  Properties, Methods
```

```
mdl=fitcnb(hdTrain,"HeartDisease","DistributionName","Kernel")
```

```
mdl =

  ClassificationNaiveBayes
           PredictorNames: {'Age'  'Cholesterol'  'ExerciseDuration'  'METs'
'RestingHeartRate'  'RestingSystolic'  'RestingDiastolic'  'MaxHeartRate'  'PeakExSystolic'
'PeakExDiastolic'  'InducedSTDep'}
              ResponseName: 'HeartDisease'
     CategoricalPredictors: []
                ClassNames: [false    true]
           ScoreTransform: 'none'
          NumObservations: 299
         DistributionNames: {'kernel'  'kernel'  'kernel'  'kernel'  'kernel'  'kernel'
'kernel'  'kernel'  'kernel'  'kernel'  'kernel'}
    DistributionParameters: {2×11 cell}
                    Kernel: {'normal'  'normal'  'normal'  'normal'  'normal'  'normal'
'normal'  'normal'  'normal'  'normal'  'normal'}
                   Support: {'unbounded'  'unbounded'  'unbounded'  'unbounded'
'unbounded'  'unbounded'  'unbounded'  'unbounded'  'unbounded'  'unbounded'  'unbounded'}
                     Width: [2×11 double]


  Properties, Methods
```

Calculate the loss for the training and test sets

```
errTrain = resubLoss(mdl);

errTest = loss(mdl,hdTest);
```

```
disp("Training Error: " + errTrain)
```

```
Training Error: 0.20736
```

```
disp("Test Error: " + errTest)
```

```
Test Error: 0.35996
```

Heart Disease Analysis

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads and formats the data.

```
heartData = readtable("heartDataAll.txt");

heartData = convertvars(heartData,12:22,"categorical");
```

This code partitions the data into training and test sets.

```
pt = cvpartition(heartData.HeartDisease,"HoldOut",0.3);

hdTrain = heartData(training(pt),:);

hdTest = heartData(test(pt),:);
```

Tasks 1 & 2

```
dists = [repmat("kernel",1,11) repmat("mvmn",1,10)]
```

```
dists = 1×21 string

"kernel"    "kernel"    "kernel"    "kernel"    "kernel"    "kernel"    "kernel"
"kernel"    "kernel"    "kernel"    "kernel"    "mvmn"      "mvmn"      "mvmn"      "mvmn"
"mvmn"      "mvmn"      "mvmn"      "mvmn"      "mvmn"      "mvmn"
```

```
mdl = fitcnb(hdTrain,"HeartDisease","DistributionNames",dists)
```

```
mdl =

  ClassificationNaiveBayes
           PredictorNames: {1×21 cell}
             ResponseName: 'HeartDisease'
    CategoricalPredictors: [12 13 14 15 16 17 18 19 20 21]
               ClassNames: [false    true]
           ScoreTransform: 'none'
          NumObservations: 299
        DistributionNames: {'kernel'  'kernel'  'kernel'  'kernel'  'kernel'  'kernel'
'kernel'  'kernel'  'kernel'  'kernel'  'kernel'  'kernel'  'mvmn'  'mvmn'  'mvmn'  'mvmn'  'mvmn'
'mvmn'  'mvmn'  'mvmn'  'mvmn'  'mvmn'}
    DistributionParameters: {2×21 cell}
        CategoricalLevels: {[]  []  []  []  []  []  []  []  []  []  []  [2×1 double]  [4×1
double]  [2×1 double]  [3×1 double]  [2×1 double]  [2×1 double]  [2×1 double]  [2×1 double]
[2×1 double]  [3×1 double]}
                   Kernel: {'normal'  'normal'  'normal'  'normal'  'normal'  'normal'
'normal'  'normal'  'normal'  'normal'  'normal'  []  []  []  []  []  []  []  []  []  []}
                  Support: {'unbounded'  'unbounded'  'unbounded'  'unbounded'
'unbounded'  'unbounded'  'unbounded'  'unbounded'  'unbounded'  'unbounded'  'unbounded'
[]  []  []  []  []  []  []  []  []  []}
                    Width: [2×21 double]


  Properties, Methods
```

Calculate the loss for the training and test sets

```
errTrain = resubLoss(mdl);

errTest = loss(mdl,hdTest);

disp("Training Error: " + errTrain)
```

```
Training Error: 0.20401
```

```
disp("Test Error: " + errTest)
```

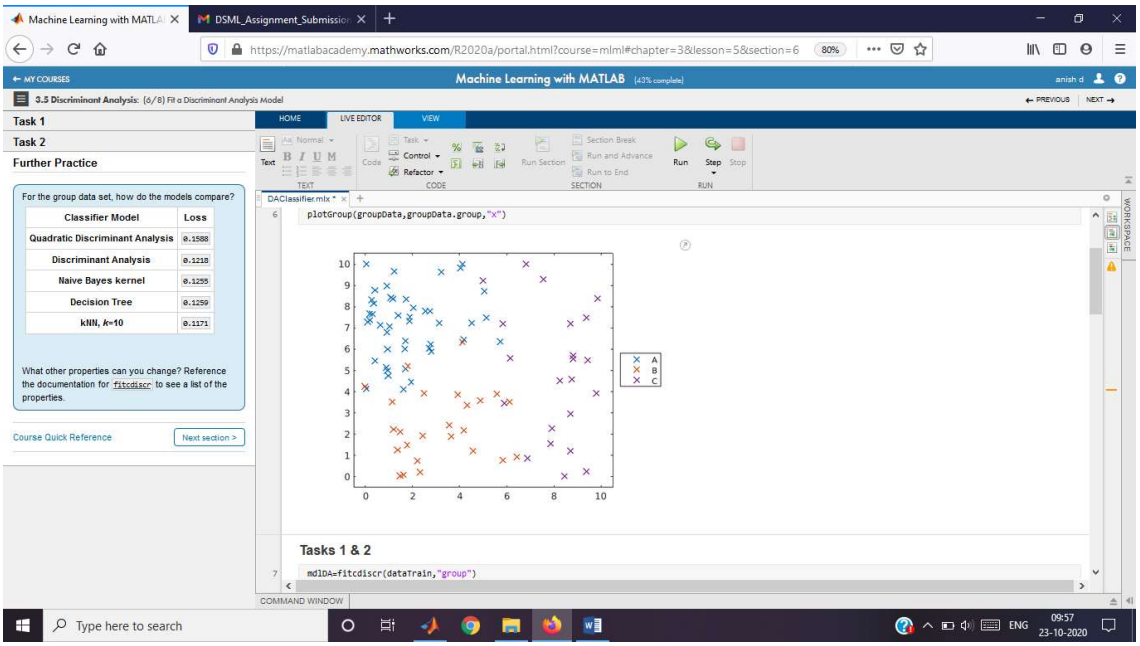
```
Test Error: 0.17958
```

Discriminant Analysis

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads and partitions the data.

```
load groups

rng(0)

cvpt = cvpartition(groupData.group,"Holdout",0.35);

dataTrain = groupData(training(cvpt),:);

dataTest = groupData(test(cvpt),:);

plotGroup(groupData,groupData.group,"x")
```



Tasks 1 & 2

```
mdlDA=fitcdiscr(dataTrain,"group")
```

```
mdlDA =
```

```
  ClassificationDiscriminant
           PredictorNames: {'x'  'y'}
             ResponseName: 'group'
    CategoricalPredictors: []
               ClassNames: [A    B    C]
           ScoreTransform: 'none'
          NumObservations: 60
              DiscrimType: 'linear'
                       Mu: [3×2 double]
                   Coeffs: [3×3 struct]


  Properties, Methods
```

```
predGroups=predict(mdlDA,dataTest)
```

```
predGroups = 32×1 categorical

A
A
A
A
A
A
A
A
A
A
```

```
mdlDA=fitcdiscr(dataTrain,"group","DiscrimType","quadratic")
```

```
mdlDA =

  ClassificationDiscriminant
           PredictorNames: {'x'  'y'}
             ResponseName: 'group'
    CategoricalPredictors: []
               ClassNames: [A    B    C]
           ScoreTransform: 'none'
          NumObservations: 60
              DiscrimType: 'quadratic'
                       Mu: [3×2 double]
                   Coeffs: [3×3 struct]


  Properties, Methods
```

```
predGroups=predict(mdlDA,dataTest)
```

```
predGroups = 32×1 categorical

A
A
A
A
A
A
A
A
A
B
```

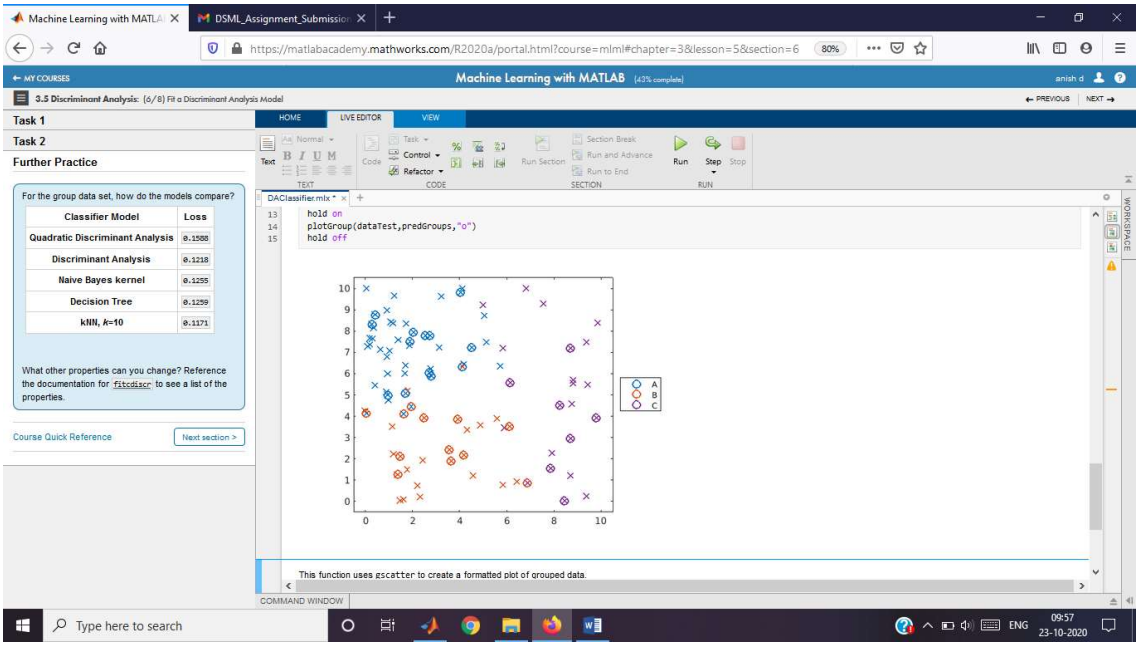
Calculate loss and plot predictions.

```
errDA = loss(mdlDA,dataTest);
```

```
disp("Discriminant Aanalysis Loss: " + errDA)
```

```
Discriminant Aanalysis Loss: 0.1588
```

```
hold on
plotGroup(dataTest,predGroups,"o")
hold off
```



This function uses `gscatter` to create a formatted plot of grouped data.

```
function plotGroup(data,grp,mkr)
    validateattributes(data,"table",{'nonempty','ncols',3})

    % Plot data by group
    colors = colororder;
    p = gscatter(data.x,data.y,grp,colors([1 2 4],:),mkr,9);

    % Format plot
    [p.LineWidth] = deal(1.5);
    legend("Location","eastoutside")
    xlim([-0.5 10.5])
    ylim([-0.5 10.5])
end
```

Heart Disease Analysis (numeric data)

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads and formats the data.

```
heartData = readtable("heartDataNum.txt");

heartData.HeartDisease = categorical(heartData.HeartDisease);
```

This code partitions the data into training and test sets.

```
pt = cvpartition(heartData.HeartDisease,"HoldOut",0.3);

hdTrain = heartData(training(pt),:);

hdTest = heartData(test(pt),:);
```

Tasks 1 & 2

```
mdl=fitcdiscr(hdTrain,"HeartDisease")
```

```
mdl =

  ClassificationDiscriminant
           PredictorNames: {'Age'  'Cholesterol'  'ExerciseDuration'  'METs'
'RestingHeartRate'  'RestingSystolic'  'RestingDiastolic'  'MaxHeartRate'  'PeakExSystolic'
'PeakExDiastolic'  'InducedSTDep'}
              ResponseName: 'HeartDisease'
     CategoricalPredictors: []
                ClassNames: [false    true]
           ScoreTransform: 'none'
          NumObservations: 299
              DiscrimType: 'linear'
                       Mu: [2×11 double]
                   Coeffs: [2×2 struct]


  Properties, Methods
```

```
mdl=fitcdiscr(hdTrain,"HeartDisease","DiscrimType","quadratic")
```

```
mdl =

  ClassificationDiscriminant
           PredictorNames: {'Age'  'Cholesterol'  'ExerciseDuration'  'METs'
'RestingHeartRate'  'RestingSystolic'  'RestingDiastolic'  'MaxHeartRate'  'PeakExSystolic'
'PeakExDiastolic'  'InducedSTDep'}
              ResponseName: 'HeartDisease'
     CategoricalPredictors: []
                ClassNames: [false    true]
           ScoreTransform: 'none'
          NumObservations: 299
              DiscrimType: 'quadratic'
                       Mu: [2×11 double]
                   Coeffs: [2×2 struct]


  Properties, Methods
```

Calculate the loss for the training and test sets

```
errTrain = resubLoss(mdl);
```

```
errTest = loss(mdl,hdTest);

disp("Training Error: " + errTrain)
```

```
Training Error: 0.17726
```

```
disp("Test Error: " + errTest)
```

```
Test Error: 0.42232
```

Support Vector Machines

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads, partitions, and plots the data.

```
load groups
cvpt = cvpartition(groupData.group,"Holdout",0.35);
dataTrain = groupData(training(cvpt),:);
dataTest = groupData(test(cvpt),:);
```

Tasks 1 & 2

```
mdlSVM=fitcsvm(dataTrain,"group")
```

```
mdlSVM =
  ClassificationSVM
             PredictorNames: {'x'  'y'}
               ResponseName: 'group'
    CategoricalPredictors: []
                ClassNames: [A    B]
            ScoreTransform: 'none'
           NumObservations: 111
                     Alpha: [65×1 double]
                      Bias: 1.7563
          KernelParameters: [1×1 struct]
            BoxConstraints: [111×1 double]
           ConvergenceInfo: [1×1 struct]
           IsSupportVector: [111×1 logical]
                    Solver: 'SMO'


  Properties, Methods
```

```
errSVM=loss(mdlSVM,dataTest)
```

```
errSVM = 0.2209
```

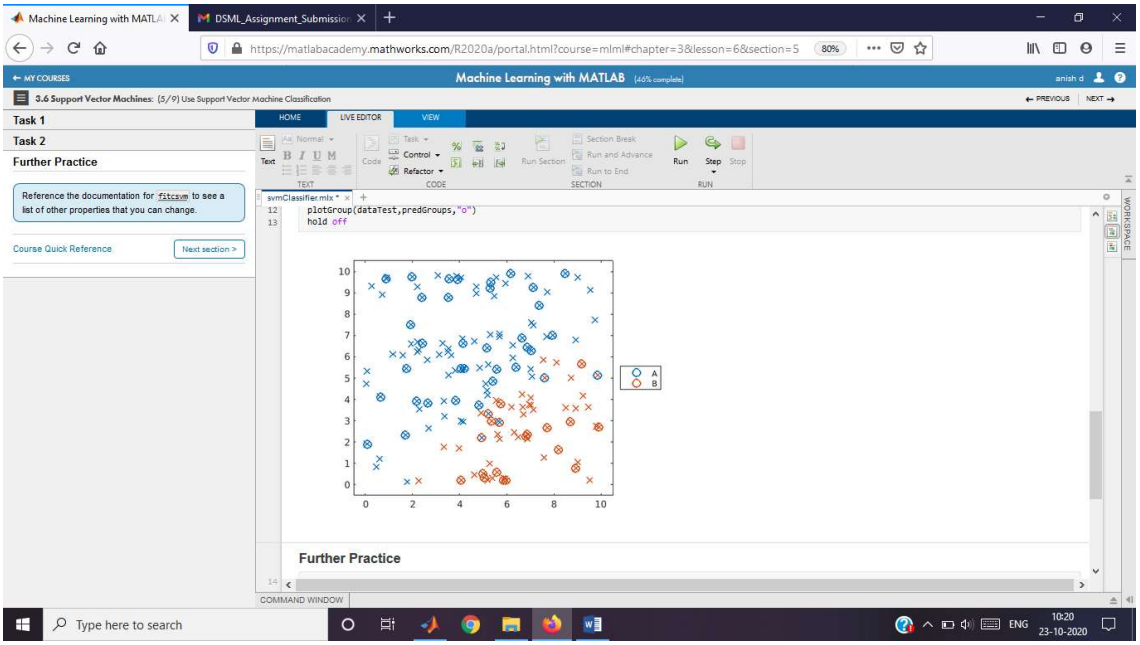
```
mdlSVM=fitcsvm(dataTrain,"group","KernelFunction","gaussian")
```

```
mdlSVM =
  ClassificationSVM
             PredictorNames: {'x'  'y'}
               ResponseName: 'group'
    CategoricalPredictors: []
                ClassNames: [A    B]
            ScoreTransform: 'none'
           NumObservations: 111
                     Alpha: [78×1 double]
                      Bias: -0.4228
          KernelParameters: [1×1 struct]
            BoxConstraints: [111×1 double]
           ConvergenceInfo: [1×1 struct]
           IsSupportVector: [111×1 logical]
                    Solver: 'SMO'


  Properties, Methods
```

```
errSVM_gaussian=loss(mdlSVM,dataTest)
errSVM_gaussian = 0.0848
```

Plot predicted groups.

```
predGroups = predict(mdlSVM,dataTest);
plotGroup(groupData,groupData.group,"x")
hold on
plotGroup(dataTest,predGroups,"o")
hold off
```



Further Practice

This function uses `gscatter` to create a formatted plot of grouped data.

```
function plotGroup(data,grp,mkr)
    validateattributes(data,"table",{'nonempty','ncols',3})

    % Plot data by group
    colors = colororder;
    p = gscatter(data.x,data.y,grp,colors([1 2 4],:),mkr,9);

    % Format plot
    [p.LineWidth] = deal(1.5);
    legend("Location","eastoutside")
```

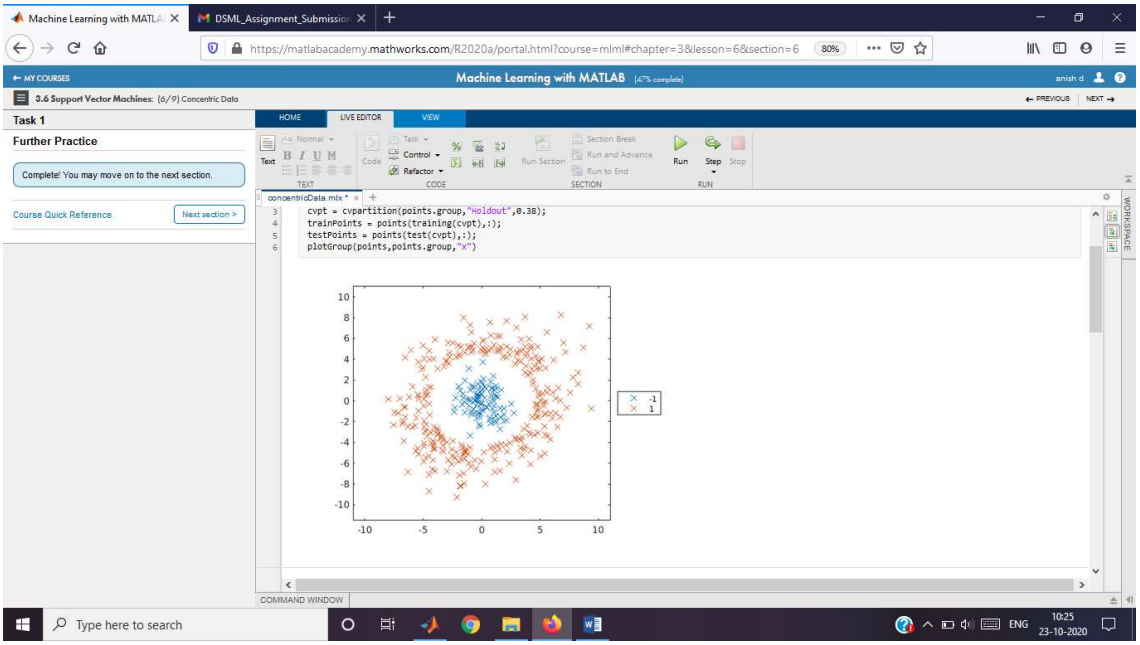
```
    xlim([-0.5 10.5])
    ylim([-0.5 10.5])
end
```

SVM with Concentric Data

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads, partitions, and plots the data.

```
load points
rng(123)
cvpt = cvpartition(points.group,"Holdout",0.38);
trainPoints = points(training(cvpt),:);
testPoints = points(test(cvpt),:);
plotGroup(points,points.group,"x")
```



Task 1

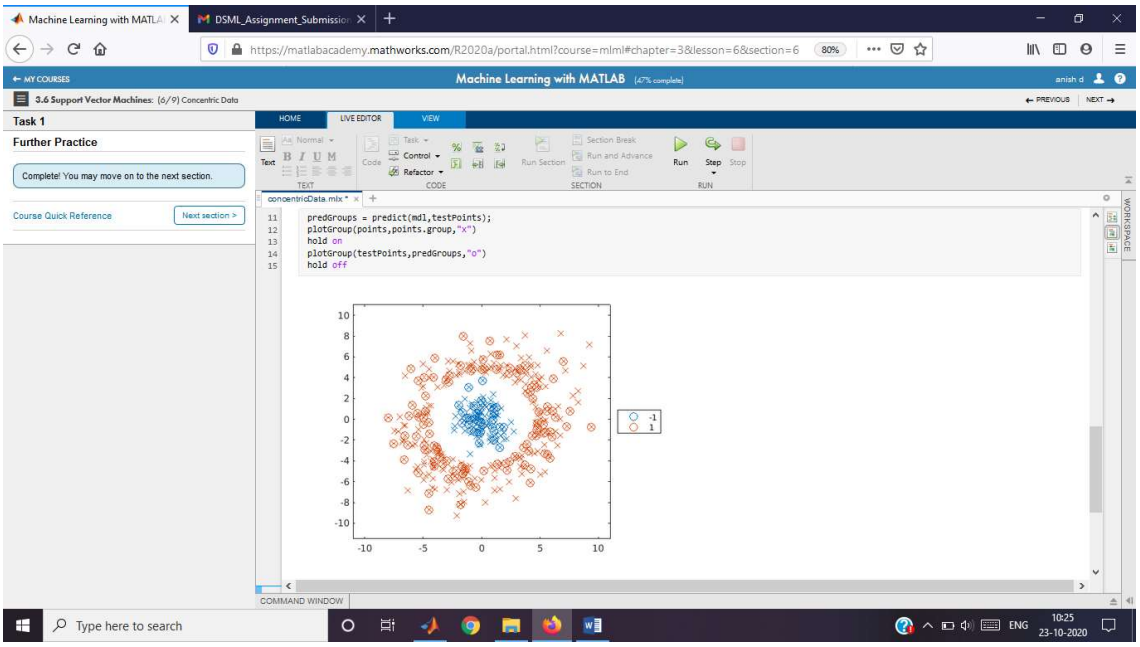
```
mdl=fitcsvm(trainPoints,"group","KernelFunction","polynomial")
```

```
mdl =
  ClassificationSVM
             PredictorNames: {'x'  'y'}
               ResponseName: 'group'
      CategoricalPredictors: []
                 ClassNames: [-1 1]
             ScoreTransform: 'none'
            NumObservations: 237
                      Alpha: [8×1 double]
                       Bias: -3.4542
           KernelParameters: [1×1 struct]
             BoxConstraints: [237×1 double]
            ConvergenceInfo: [1×1 struct]
            IsSupportVector: [237×1 logical]
                     Solver: 'SMO'


  Properties, Methods
```

```
err=loss(mdl,testPoints)
err = 0
```

Plot predicted groups.

```
predGroups = predict(mdl,testPoints);
plotGroup(points,points.group,"x")
hold on
plotGroup(testPoints,predGroups,"o")
hold off
```



This function uses `gscatter` to create a formatted plot of grouped data.

```
function plotGroup(data,grp,mkr)
    validateattributes(data,"table",{'nonempty','ncols',3})

    % Plot data by group
    colors = colororder;
    p = gscatter(data.x,data.y,grp,colors([1 2 4],:),mkr,9);

    % Format plot
    [p.LineWidth] = deal(1.25);
    legend("Location","eastoutside")
    xlim([-11 11])
    ylim([-11.5 11])
end
```

Heart Disease Analysis (numeric data)

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads and formats the data.

```
heartData = readtable("heartDataNum.txt");
heartData.HeartDisease = categorical(heartData.HeartDisease);
```

This code partitions the data into training and test sets.

```
pt = cvpartition(heartData.HeartDisease,"HoldOut",0.3);
hdTrain = heartData(training(pt),:);
hdTest = heartData(test(pt),:);
```

Tasks 1 & 2

```
mdl=fitcsvm(hdTrain,"HeartDisease")
```

```
mdl =
  ClassificationSVM
           PredictorNames: {'Age'  'Cholesterol'  'ExerciseDuration'  'METs'
'RestingHeartRate'  'RestingSystolic'  'RestingDiastolic'  'MaxHeartRate'  'PeakExSystolic'
'PeakExDiastolic'  'InducedSTDep'}
             ResponseName: 'HeartDisease'
    CategoricalPredictors: []
               ClassNames: [false    true]
           ScoreTransform: 'none'
          NumObservations: 299
                    Alpha: [171×1 double]
                     Bias: 0.3335
         KernelParameters: [1×1 struct]
           BoxConstraints: [299×1 double]
          ConvergenceInfo: [1×1 struct]
          IsSupportVector: [299×1 logical]
                   Solver: 'SMO'


  Properties, Methods
```

```
mdl=fitcsvm(hdTrain,"HeartDisease","kernelFunction","gaussian")
```

```
mdl =
  ClassificationSVM
           PredictorNames: {'Age'  'Cholesterol'  'ExerciseDuration'  'METs'
'RestingHeartRate'  'RestingSystolic'  'RestingDiastolic'  'MaxHeartRate'  'PeakExSystolic'
'PeakExDiastolic'  'InducedSTDep'}
             ResponseName: 'HeartDisease'
    CategoricalPredictors: []
               ClassNames: [false    true]
           ScoreTransform: 'none'
          NumObservations: 299
                    Alpha: [169×1 double]
                     Bias: 0.4863
         KernelParameters: [1×1 struct]
           BoxConstraints: [299×1 double]
          ConvergenceInfo: [1×1 struct]
          IsSupportVector: [299×1 logical]
                   Solver: 'SMO'


  Properties, Methods
```

Calculate the loss for the training and test sets

```
errTrain = resubLoss(mdl);
```

```
errTest = loss(mdl,hdTest);
disp("Training Error: " + errTrain)
```
```
Training Error: 0.19064
```
```
disp("Test Error: " + errTest)
```
```
Test Error: 0.39105
```

Heart Disease Analysis

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads and formats the data.

```
heartData = readtable("heartDataAll.txt");
heartData = convertvars(heartData,12:22,"categorical");
```

This code partitions the data into training and test sets.

```
pt = cvpartition(heartData.HeartDisease,"HoldOut",0.3);
hdTrain = heartData(training(pt),:);
hdTest = heartData(test(pt),:);
```

Task 1

```
mdl=fitcsvm(hdTrain,"HeartDisease","KernelFunction","gaussian")
```
```
mdl =
  ClassificationSVM
             PredictorNames: {1×21 cell}
               ResponseName: 'HeartDisease'
    CategoricalPredictors: [12 13 14 15 16 17 18 19 20 21]
                ClassNames: [false    true]
            ScoreTransform: 'none'
           NumObservations: 299
                     Alpha: [268×1 double]
                      Bias: 0.0278
          KernelParameters: [1×1 struct]
            BoxConstraints: [299×1 double]
           ConvergenceInfo: [1×1 struct]
           IsSupportVector: [299×1 logical]
                    Solver: 'SMO'


  Properties, Methods
```

Calculate the loss for the training and test sets

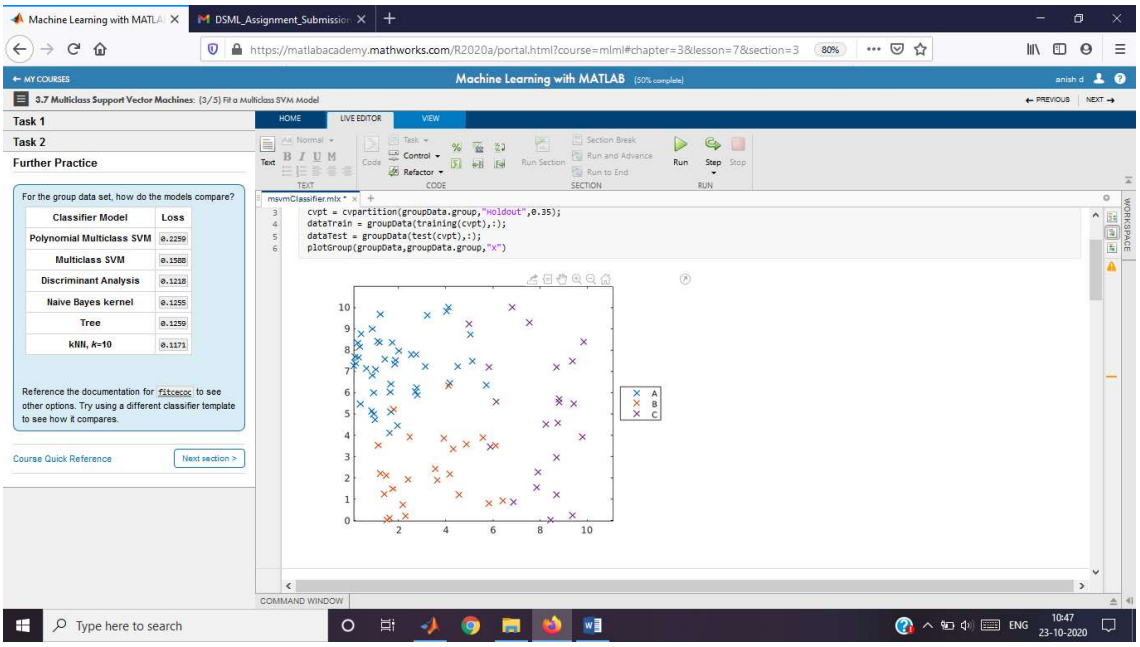
```
errTrain = resubLoss(mdl);
errTest = loss(mdl,hdTest);
disp("Training Error: " + errTrain)
```
```
Training Error: 0.036789
```
```
disp("Test Error: " + errTest)
```
```
Test Error: 0.26416
```

Multiclass SVM

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads, partitions, and plots the data.

```
load groups
rng(0)
cvpt = cvpartition(groupData.group,"Holdout",0.35);
dataTrain = groupData(training(cvpt),:);
dataTest = groupData(test(cvpt),:);
plotGroup(groupData,groupData.group,"x")
```



Task 1

```
mdlMSVM=fitcecoc(dataTrain,"group")
```

```
mdlMSVM =
  ClassificationECOC
           PredictorNames: {'x'  'y'}
             ResponseName: 'group'
    CategoricalPredictors: []
               ClassNames: [A    B    C]
           ScoreTransform: 'none'
           BinaryLearners: {3×1 cell}
               CodingName: 'onevsone'


  Properties, Methods
```

```
errMSVM=loss(mdlMSVM,dataTest)
```

```
errMSVM = 0.1588
```

Task 2

```
t=templateSVM("KernelFunction","polynomial")
```

```
t =
Fit template for classification SVM.

                    Alpha: [0×1 double]
            BoxConstraint: []
```

```
                CacheSize: []
            CachingMethod: ''
               ClipAlphas: []
    DeltaGradientTolerance: []
                   Epsilon: []
              GapTolerance: []
              KKTTolerance: []
            IterationLimit: []
            KernelFunction: 'polynomial'
               KernelScale: []
              KernelOffset: []
     KernelPolynomialOrder: []
                 NumPrint: []
                       Nu: []
          OutlierFraction: []
        RemoveDuplicates: []
          ShrinkagePeriod: []
                   Solver: ''
          StandardizeData: []
       SaveSupportVectors: []
           VerbosityLevel: []
                  Version: 2
                   Method: 'SVM'
                     Type: 'classification'
```

```
mdlMSVM=fitcecoc(dataTrain,"group","Learners",t)
```

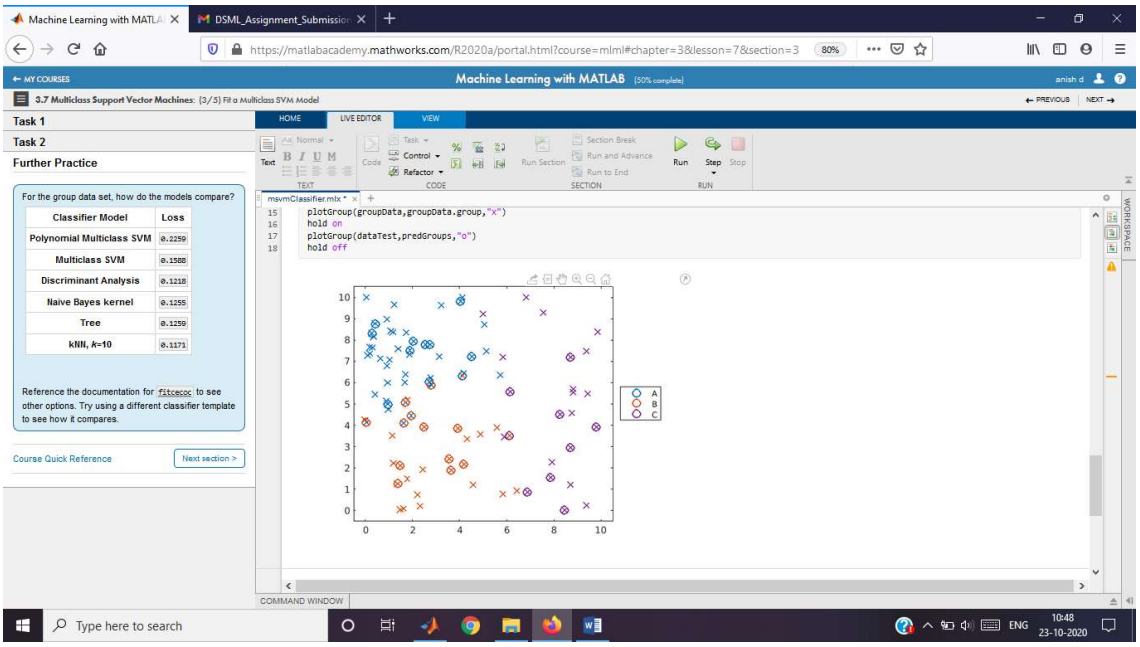
```
mdlMSVM =
  ClassificationECOC
           PredictorNames: {'x'  'y'}
             ResponseName: 'group'
    CategoricalPredictors: []
               ClassNames: [A    B    C]
           ScoreTransform: 'none'
           BinaryLearners: {3×1 cell}
               CodingName: 'onevsone'


  Properties, Methods
```

```
errMSVM=loss(mdlMSVM,dataTest)
```

```
errMSVM = 0.2259
```

Plot predicted groups.

```
predGroups = predict(mdlMSVM,dataTest);
plotGroup(groupData,groupData.group,"x")
hold on
plotGroup(dataTest,predGroups,"o")
hold off
```



Further Practice

This function uses `gscatter` to create a formatted plot of grouped data.

```
function plotGroup(data,grp,mkr)
    validateattributes(data,"table",{'nonempty','ncols',3})

    % Plot data by group
    colors = colororder;
    p = gscatter(data.x,data.y,grp,colors([1 2 4],:),mkr,9);

    % Format plot
    [p.LineWidth] = deal(1.5);
    legend("Location","eastoutside")
    xlim([-0.5 10.5])
    ylim([-0.5 10.5])
end
```

Heart Disease Analysis (numeric data)

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads and formats the data.

```
heartData = readtable("heartDataNumMulti.txt");
heartData.HeartDisease = categorical(heartData.HeartDisease);
```

This code partitions the data into training and test sets.

```
pt = cvpartition(heartData.HeartDisease,"HoldOut",0.3);
hdTrain = heartData(training(pt),:);
hdTest = heartData(test(pt),:);
```

Tasks 1 & 2

```
mdl = fitcecoc(hdTrain,"HeartDisease");
t=templateSVM("KernelFunction","gaussian")
```

```
t =
Fit template for classification SVM.

                    Alpha: [0×1 double]
              BoxConstraint: []
                  CacheSize: []
              CachingMethod: ''
                 ClipAlphas: []
    DeltaGradientTolerance: []
                  Epsilon: []
               GapTolerance: []
               KKTTolerance: []
            IterationLimit: []
            KernelFunction: 'gaussian'
               KernelScale: []
              KernelOffset: []
     KernelPolynomialOrder: []
                  NumPrint: []
                        Nu: []
           OutlierFraction: []
          RemoveDuplicates: []
           ShrinkagePeriod: []
                    Solver: ''
           StandardizeData: []
        SaveSupportVectors: []
            VerbosityLevel: []
                   Version: 2
                    Method: 'SVM'
                      Type: 'classification'
```

```
mdl = fitcecoc(hdTrain,"HeartDisease","Learners",t);
```

Calculate the loss for the training and test sets

```
errTrain = resubLoss(mdl);
errTest = loss(mdl,hdTest);
disp("Training Error: " + errTrain)
```

```
Training Error: 0.41806
```

```
disp("Test Error: " + errTest)
```

```
Test Error: 0.56643
```

Heart Disease Analysis

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads and formats the data.

```
heartData = readtable("heartDataAllMulti.txt");
heartData = convertvars(heartData,12:22,"categorical");
```

This code partitions the data into training and test sets.

```
pt = cvpartition(heartData.HeartDisease,"HoldOut",0.3);
hdTrain = heartData(training(pt),:);
hdTest = heartData(test(pt),:);
```

Task 1

```
mdl=fitcecoc(hdTrain,"HeartDisease")
```

```
mdl =
  ClassificationECOC
           PredictorNames: {1×21 cell}
             ResponseName: 'HeartDisease'
    CategoricalPredictors: [12 13 14 15 16 17 18 19 20 21]
               ClassNames: [0    1    2    3    4]
           ScoreTransform: 'none'
           BinaryLearners: {10×1 cell}
               CodingName: 'onevsone'


  Properties, Methods
```

Calculate the loss for the training and test sets

```
errTrain = resubLoss(mdl);
errTest = loss(mdl,hdTest);
disp("Training Error: " + errTrain)
```

```
Training Error: 0.35452
```

```
disp("Test Error: " + errTest)
```

```
Test Error: 0.56014
```

User Knowledge Assessment

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads and formats the data.

```
knowledge = readtable("userKnowledge.txt");
knowledge.knowledgeScore = categorical(knowledge.knowledgeScore);
```

Partition the data into training and test sets.

```
rng(0)
pt = cvpartition(knowledge.knowledgeScore,"HoldOut",0.3);
kTrain = knowledge(training(pt),:)
```

```
kTrain = 181×6 table
```

	studyTimeGoal	repetitionGoal	studyTimeSupporting	examSupporting	examGoal	knowledgeScore
1	0	0	0	0	0	veryLow
2	0.0800	0.0800	0.1000	0.2400	0.9000	high
3	0.0600	0.0600	0.0500	0.2500	0.3300	low
4	0.1000	0.1000	0.1500	0.6500	0.3000	middle

	studyTimeGoal	repetitionGoal	studyTimeSupporting	examSupporting	examGoal	knowledgeScore
5	0.0800	0.0800	0.0800	0.9800	0.2400	low
6	0.0900	0.1500	0.4000	0.1000	0.6600	middle
7	0.1000	0.1000	0.4300	0.2900	0.5600	middle
8	0.2000	0.1400	0.3500	0.7200	0.2500	low
9	0	0	0.5000	0.2000	0.8500	high
10	0.1800	0.1800	0.5500	0.3000	0.8100	high
11	0.0600	0.0600	0.5100	0.4100	0.3000	low
12	0.1000	0.1000	0.5200	0.7800	0.3400	middle
13	0.1000	0.1000	0.7000	0.1500	0.9000	high
14	0.2000	0.2000	0.7000	0.3000	0.6000	middle
15	0.1200	0.1200	0.7500	0.3500	0.8000	high
16	0.0500	0.0700	0.7000	0.0100	0.0500	veryLow
17	0.1500	0.3200	0.0500	0.2700	0.2900	low
18	0.2000	0.2900	0.2500	0.4900	0.5600	middle
19	0.1200	0.2800	0.2000	0.7800	0.2000	low
20	0.1000	0.2700	0.3100	0.2900	0.6500	middle
21	0.0600	0.2900	0.3500	0.7600	0.2500	low
22	0.0900	0.3000	0.6800	0.1800	0.8500	high
23	0.0400	0.2800	0.5500	0.2500	0.1000	veryLow
24	0.1500	0.2750	0.8000	0.2100	0.8100	high
25	0.1500	0.2950	0.7500	0.6500	0.2400	low
26	0.1800	0.3200	0.0400	0.1900	0.8200	high
27	0.0600	0.3500	0.1200	0.4300	0.2900	low
28	0.0900	0.3300	0.3100	0.2600	0	veryLow
29	0.1900	0.3800	0.3800	0.4900	0.4500	middle
30	0.0200	0.3300	0.3600	0.7600	0.1000	low
31	0.2000	0.4900	0.6000	0.2000	0.7800	high
32	0.1400	0.4900	0.5500	0.2900	0.6000	middle
33	0.1150	0.3500	0.6500	0.2700	0.0400	veryLow
34	0.1700	0.3600	0.8000	0.1400	0.6600	middle
35	0.1300	0.3900	0.8500	0.3800	0.7700	high
36	0.0900	0.5100	0.0200	0.1800	0.6700	middle
37	0.0600	0.5000	0.0900	0.2800	0.2500	low
38	0.0900	0.5500	0.1200	0.7800	0.0500	low
39	0.1800	0.7200	0.3700	0.2900	0.5500	middle
40	0.1000	0.6000	0.3300	0.4200	0.2600	low
41	0.2000	0.6800	0.7300	0.4800	0.2800	low
42	0.2400	0.5800	0.7600	0.8000	0.2800	middle
43	0.2900	0.0600	0.1900	0.5500	0.5100	middle
44	0.2800	0.1000	0.1200	0.2800	0.3200	low
45	0.3100	0.1000	0.4100	0.4200	0.7500	high
46	0.2900	0.1500	0.3300	0.6600	0.0800	veryLow

	studyTimeGoal	repetitionGoal	studyTimeSupporting	examSupporting	examGoal	knowledgeScore
47	0.3000	0.2000	0.5200	0.3000	0.5300	middle
48	0.2800	0.1600	0.6900	0.3300	0.7800	high
49	0.2550	0.1800	0.5000	0.4000	0.1000	veryLow
50	0.2450	0.1000	0.7100	0.2600	0.2000	veryLow
51	0.2950	0.2000	0.8600	0.4400	0.2800	low
52	0.3200	0.1200	0.7900	0.7600	0.2400	low
53	0.2500	0.2900	0.1500	0.4800	0.2600	low
54	0.2700	0.1000	0.1000	0.7000	0.2500	low
55	0.2480	0.3000	0.3100	0.2000	0.0300	veryLow
56	0.3250	0.2500	0.3800	0.3100	0.7900	high
57	0.2900	0.2900	0.4000	0.7800	0.1800	low
58	0.2900	0.3000	0.5200	0.0900	0.6700	middle
59	0.2510	0.2650	0.5700	0.6000	0.0900	veryLow
60	0.2550	0.3050	0.8600	0.6200	0.1500	low
61	0.2950	0.2500	0.7300	0.7700	0.1900	low
62	0.2580	0.2500	0.2950	0.3300	0.7700	high
63	0.2900	0.2500	0.2900	0.2900	0.5700	middle
64	0.2430	0.2700	0.0800	0.4200	0.2900	low
65	0.2700	0.2800	0.1800	0.4800	0.2600	low
66	0.2990	0.3200	0.3100	0.3300	0.8700	high
67	0.3000	0.2700	0.3100	0.3100	0.5400	middle
68	0.2950	0.2900	0.3100	0.7600	0.1000	low
69	0.2900	0.3000	0.5600	0.2500	0.6700	middle
70	0.2600	0.2800	0.6000	0.2900	0.5900	middle
71	0.3050	0.2550	0.6300	0.4000	0.5400	middle
72	0.3200	0.2700	0.5200	0.8100	0.3000	middle
73	0.2990	0.2950	0.8000	0.3700	0.8400	high
74	0.2760	0.2550	0.8100	0.2700	0.3300	low
75	0.3200	0.2800	0.7200	0.8900	0.5800	high
76	0.3290	0.5500	0.0200	0.4000	0.7900	high
77	0.2950	0.5900	0.2900	0.3100	0.5500	middle
78	0.2850	0.6400	0.1800	0.6100	0.4500	middle
79	0.2650	0.6000	0.2800	0.6600	0.0700	veryLow
80	0.3150	0.6900	0.2800	0.8000	0.7000	high
81	0.3250	0.6100	0.4600	0.3200	0.8100	high
82	0.3050	0.5500	0.5000	0.1100	0.3330	low
83	0.3000	0.8500	0.5400	0.2500	0.8300	middle
84	0.2990	0.7000	0.9500	0.2200	0.6600	high
85	0.2650	0.7600	0.8000	0.2800	0.2800	low
86	0.2550	0.7200	0.7200	0.6300	0.1400	low
87	0.3900	0.0500	0.0200	0.0600	0.3400	low
88	0.4500	0.0400	0.1800	0.5500	0.0700	veryLow

	studyTimeGoal	repetitionGoal	studyTimeSupporting	examSupporting	examGoal	knowledgeScore
89	0.4000	0.1200	0.4100	0.1000	0.6500	middle
90	0.3700	0.0600	0.3200	0.7800	0.1000	low
91	0.4100	0.0900	0.5800	0.1800	0.5800	middle
92	0.3800	0.0100	0.5300	0.2700	0.3000	low
93	0.3300	0.0400	0.5000	0.5500	0.1000	veryLow
94	0.4200	0.1500	0.6600	0.7800	0.4000	middle
95	0.3900	0.1500	0.8100	0.2200	0.2900	low
96	0.4200	0.2100	0.8700	0.5600	0.4800	middle
97	0.3650	0.2430	0.1900	0.2400	0.3500	low
98	0.4800	0.3000	0.1500	0.6500	0.7700	high
99	0.4900	0.2450	0.3800	0.1400	0.8600	high
100	0.3340	0.2950	0.3300	0.3200	0.3000	low
⋮						

```
kTest = knowledge(test(pt),:);
```

Fit model

Fit classifier and calculate loss.

```
mdl = fitctree(kTrain,"knowledgeScore");
errRate = loss(mdl,kTest)
```

```
errRate = 0.1030
```

Further Practice

Breast Cancer Detection

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads and formats the data.

```
bcDiag = readtable("breastCancerData.txt");
bcDiag.diagnosis = categorical(bcDiag.diagnosis);
```

Partition the data into training and test sets.

```
pt = cvpartition(bcDiag.diagnosis,"HoldOut",0.3);
bcTrain = bcDiag(training(pt),:);
bcTest = bcDiag(test(pt),:);
```

Task 1

Fit classifier and calculate loss.

```
mdl = fitcknn(bcTrain,"diagnosis","NumNeighbors",5);
errRate = loss(mdl,bcTest)
```

```
errRate =
   0.076627185825415
```

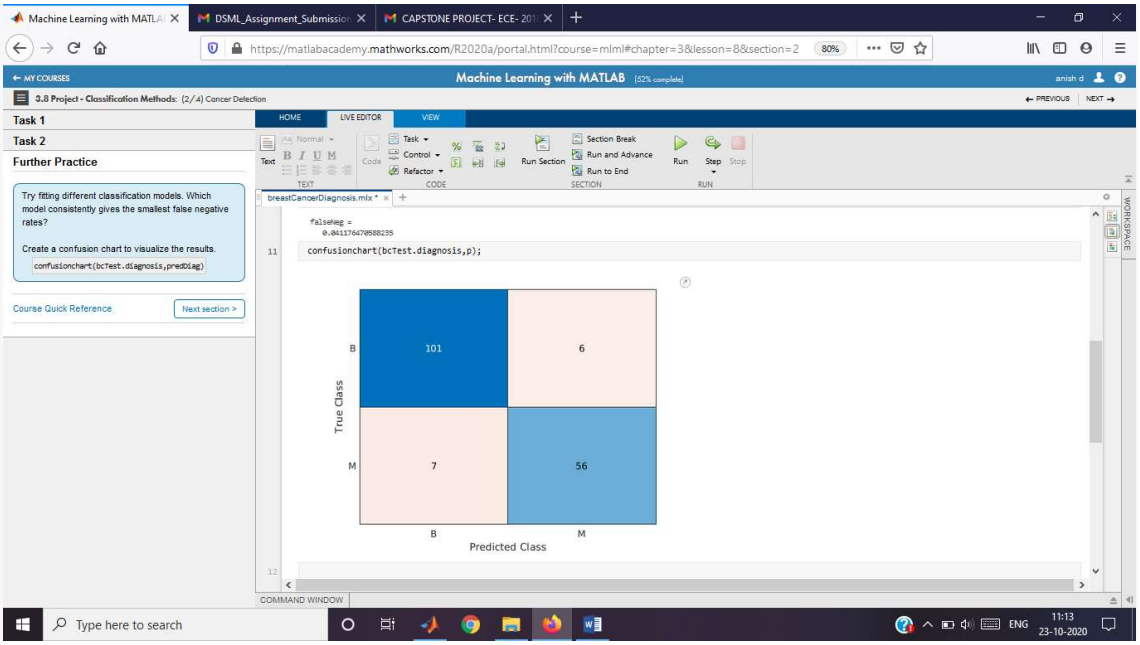
Task 2

Calculate false negative rate.

```
p = predict(mdl,bcTest);
falseNeg = mean((bcTest.diagnosis == "M") & (p == "B"))
```

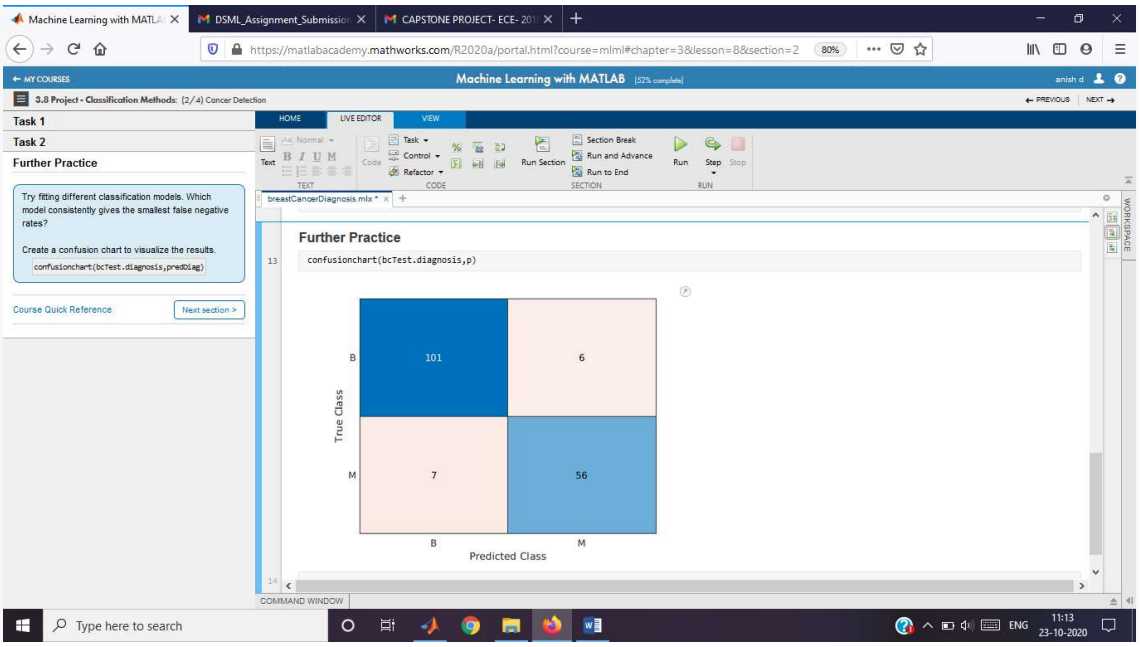
```
falseNeg =
   0.041176470588235
```

```
confusionchart(bcTest.diagnosis,p);
```



Further Practice

```
confusionchart(bcTest.diagnosis,p)
```



Mushroom Edibility

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads and formats the data.

```
mushrooms = readtable("mushrooms.txt");
mushrooms = convertvars(mushrooms,vartype("cellstr"),"categorical");
```

Partition the data into training and test sets.

```
pt = cvpartition(mushrooms.edibility,"HoldOut",0.3);
mTrain = mushrooms(training(pt),:);
mTest = mushrooms(test(pt),:);
```

Task 1

Use Naive Bayes classification to predict edibility.

```
mdlNB = fitcnb(mTrain,"edibility","DistributionNames","mvmn");
errRateNB = loss(mdlNB,mTest)
```

```
errRateNB =
   0.166666666666667
```

Task 2

Use *k*-NN classification to predict edibility.

```
mdlKNN = fitcknn(mTrain,"edibility","NumNeighbors",2);
errRateKNN = loss(mdlKNN,mTest)
```

```
errRateKNN =
   0.112603174603175
```

Wine Quality

Instructions are in the task pane to the left. Complete and submit each task one at a time.

This code loads and formats the data.

```
redData = readtable("redWine.txt");
redData.QCLabel = categorical(redData.QCLabel);
```

Partition the data into training and test sets.

```
pt = cvpartition(redData.QCLabel,"HoldOut",0.3);
redWineTrain = redData(training(pt),:);
redWineTest = redData(test(pt),:);
```

Task 1

Fit model and calculate loss.

```
mdl = fitcdiscr(redWineTrain,"QCLabel");
errRate = loss(mdl,redWineTest)
```

```
errRate =
   0.425631850607930
```