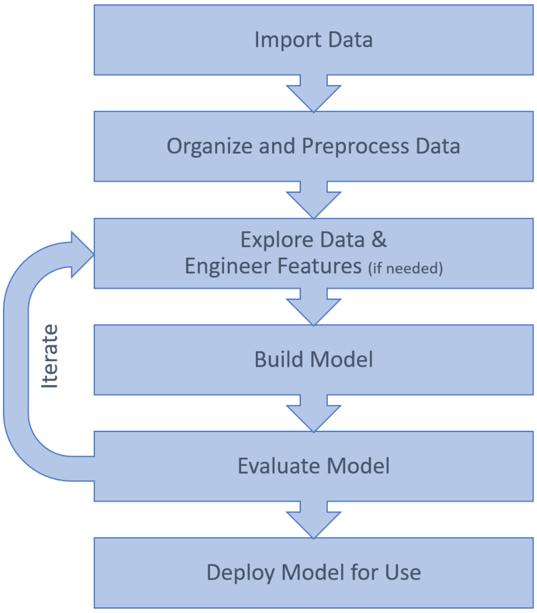
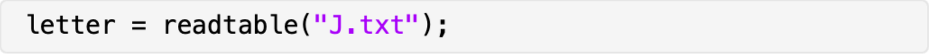
Report on modelling a Classifier using kNN algorithm to identify 14 handwritten letters

Pipeline

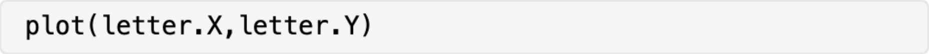


#importing a hand written txt file for letter ‘J’

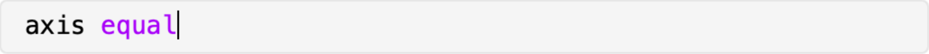
Input:



```
letter = readtable("J.txt");
```

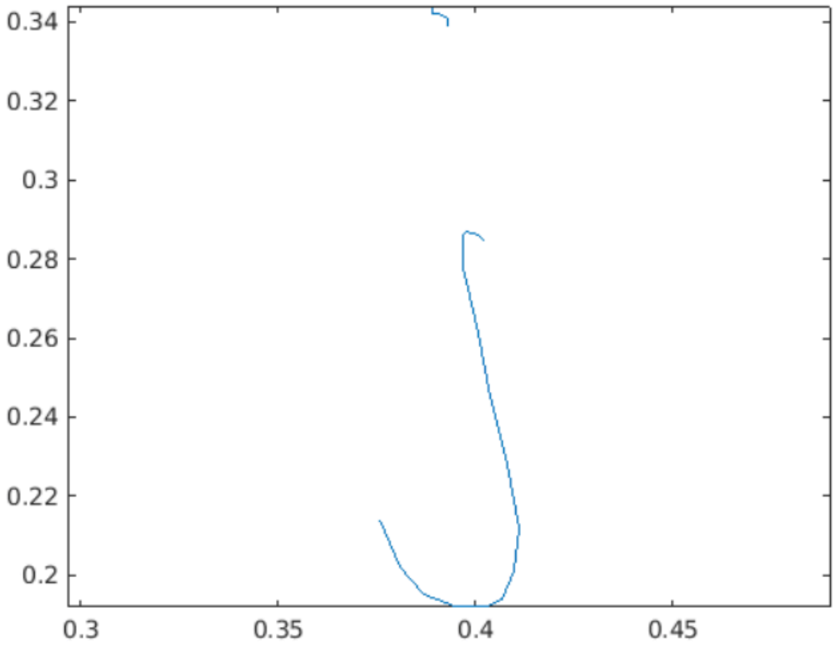


```
plot(letter.X,letter.Y)
```



```
axis equal
```

Output:

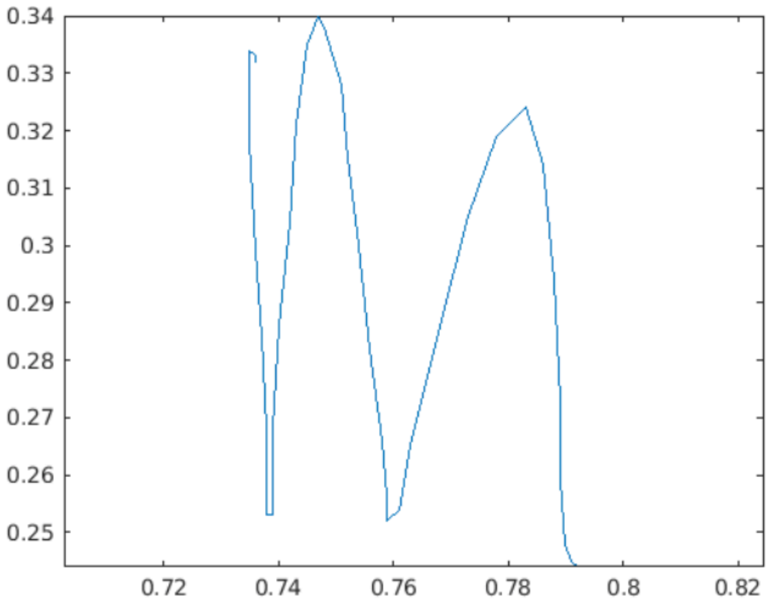


#importing a hand written txt file for letter ‘M’ ; X is horizontal and Y is vertical position of pen.

Input:

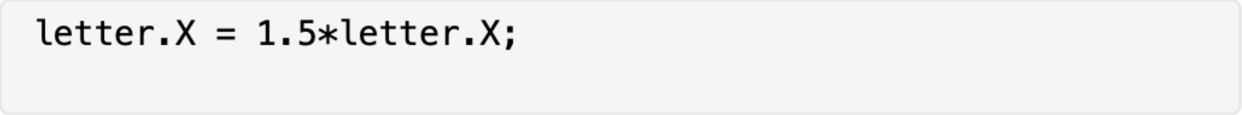
```
letter = readtable("M.txt");
plot(letter.X,letter.Y)
axis equal
```

Output:



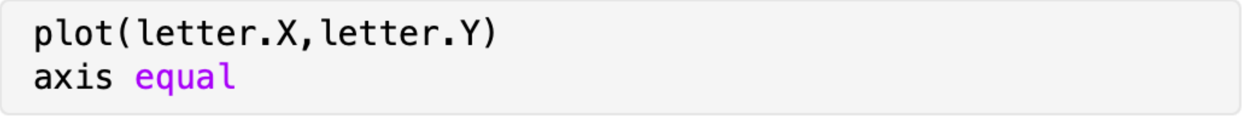
#Correcting Distance Units: the tablet used for data collection is 15*10 inches, so adjusting the horizontal distance to the range [0,1.5]

Input:



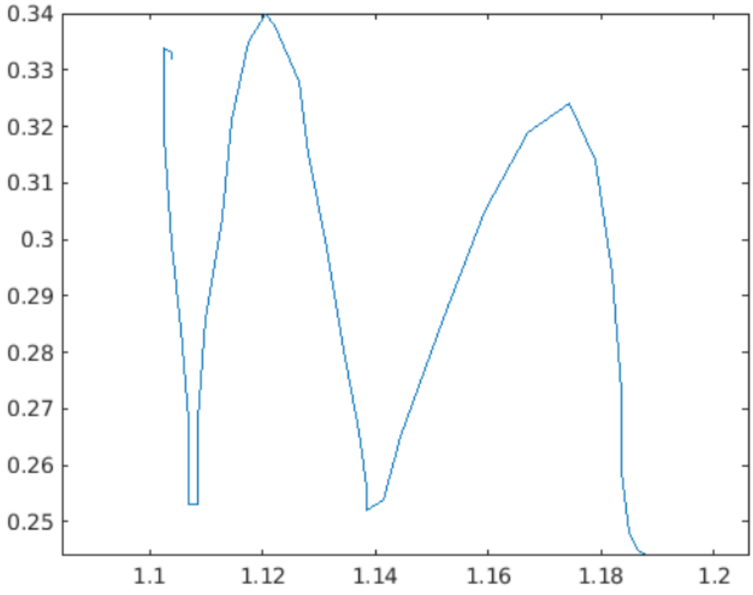
```
letter.X = 1.5*letter.X;
```

#View the result



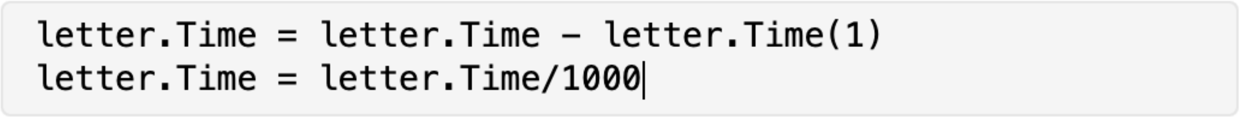
```
plot(letter.X,letter.Y)
axis equal
```

Output :



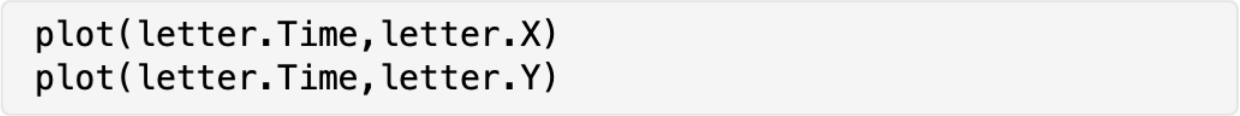
#Normalising time as the time here is from the start of data of the data collecting session, by subtracting time at which it started.

Input:



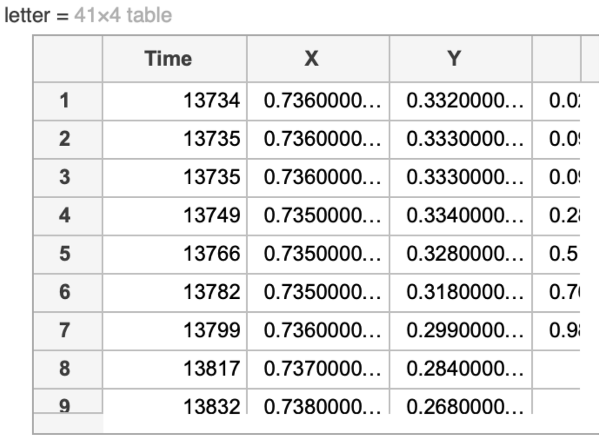
```
letter.Time = letter.Time - letter.Time(1)
letter.Time = letter.Time/1000
```

#View the result



```
plot(letter.Time,letter.X)
plot(letter.Time,letter.Y)
```

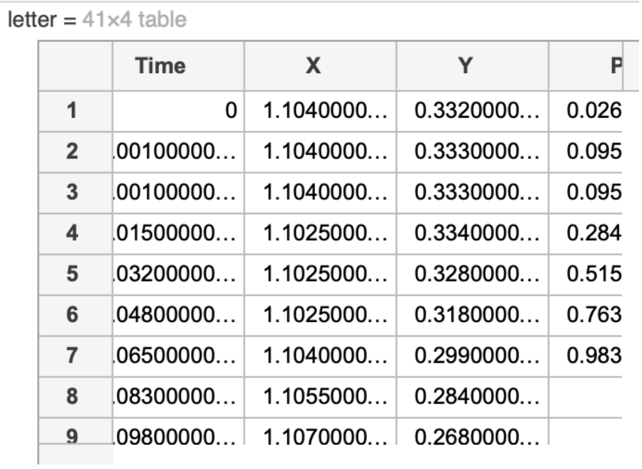
Output:



letter = 41x4 table

	Time	X	Y	
1	13734	0.7360000…	0.3320000…	0.0
2	13735	0.7360000…	0.3330000…	0.0
3	13735	0.7360000…	0.3330000…	0.0
4	13749	0.7350000…	0.3340000…	0.2
5	13766	0.7350000…	0.3280000…	0.5
6	13782	0.7350000…	0.3180000…	0.7
7	13799	0.7360000…	0.2990000…	0.9
8	13817	0.7370000…	0.2840000…	
9	13832	0.7380000…	0.2680000…	

→



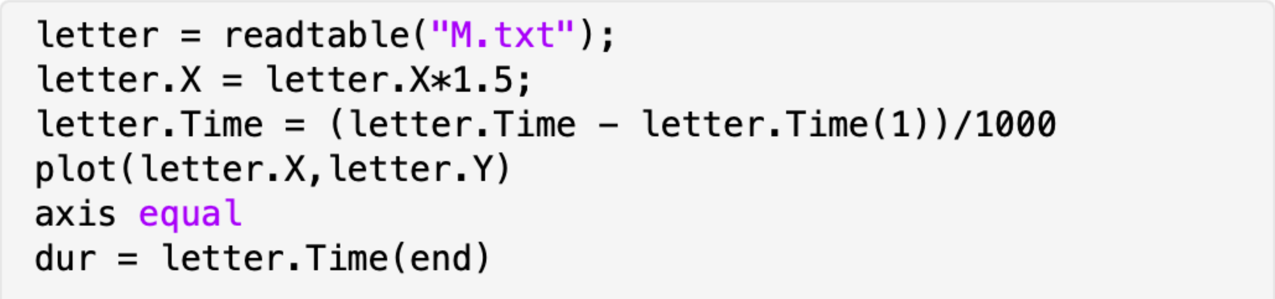
letter = 41x4 table

	Time	X	Y	P
1	0	1.1040000…	0.3320000…	0.026
2	.00100000…	1.1040000…	0.3330000…	0.095
3	.00100000…	1.1040000…	0.3330000…	0.095
4	.01500000…	1.1025000…	0.3340000…	0.284
5	.03200000…	1.1025000…	0.3280000…	0.515
6	.04800000…	1.1025000…	0.3180000…	0.763
7	.06500000…	1.1040000…	0.2990000…	0.983
8	.08300000…	1.1055000…	0.2840000…	
9	.09800000…	1.1070000…	0.2680000…	

#Calculating Features looking at the change in horizontal and vertical position of pen with time and the aspect ratio of each letter

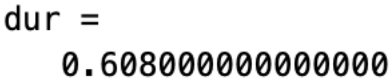
#Extraction of duration to write the letter, by extracting last value of letter.time

Input:



```
letter = readtable("M.txt");
letter.X = letter.X*1.5;
letter.Time = (letter.Time - letter.Time(1))/1000
plot(letter.X,letter.Y)
axis equal
dur = letter.Time(end)
```

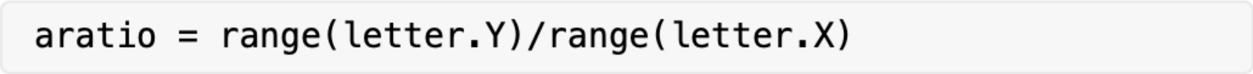
Output:



```
dur =
   0.608000000000000
```

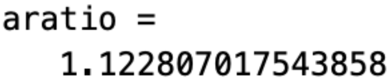
#calculating the aspect ratio

Input:



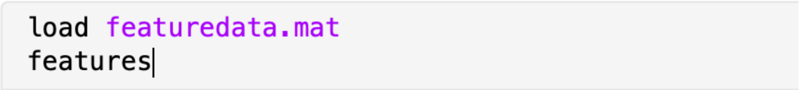
```
aratio = range(letter.Y)/range(letter.X)
```

Output:

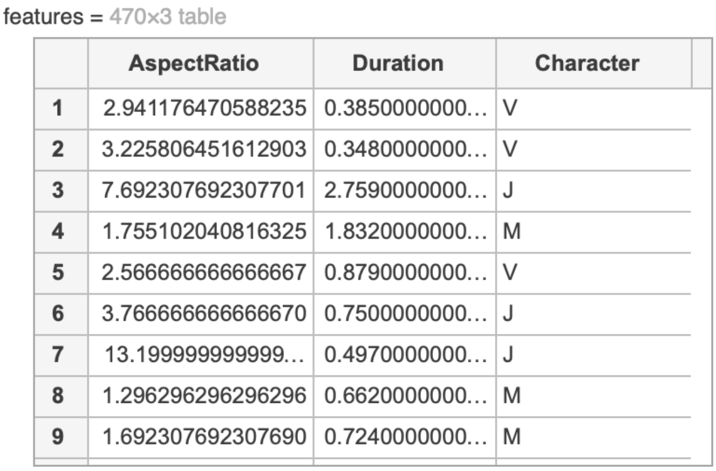


```
aratio =
   1.122807017543858
```

#Feature data table contains duration, aspect ratio and the character, importing the data and plotting the scatter plot of aspect ratio on x-axis and duration on y-axis



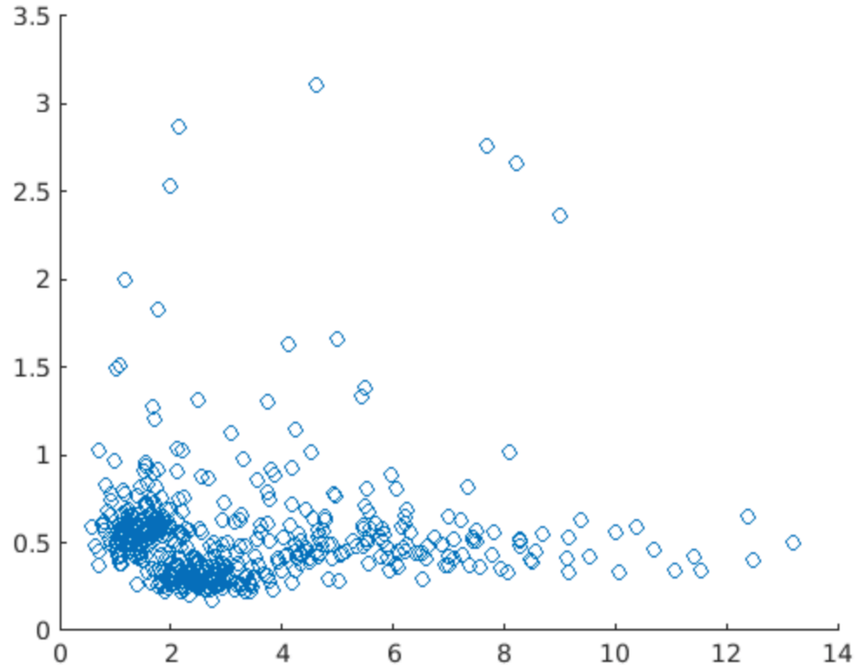
```
load featuredata.mat
features
```



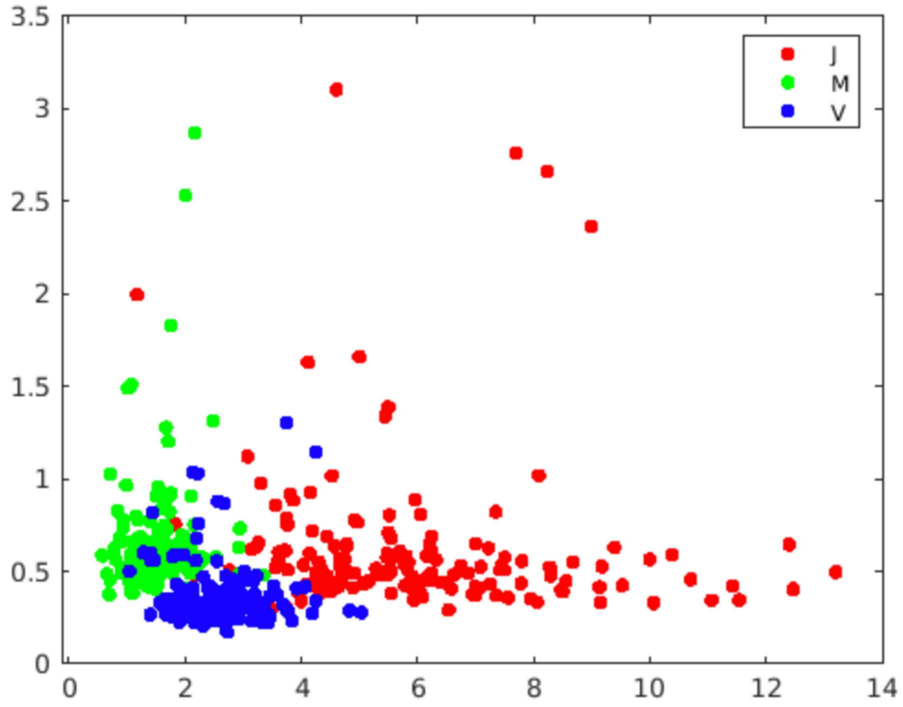
features = 470×3 table

	AspectRatio	Duration	Character
1	2.941176470588235	0.3850000000...	V
2	3.225806451612903	0.3480000000...	V
3	7.692307692307701	2.7590000000...	J
4	1.755102040816325	1.8320000000...	M
5	2.566666666666667	0.8790000000...	V
6	3.766666666666670	0.7500000000...	J
7	13.199999999999...	0.4970000000...	J
8	1.296296296296296	0.6620000000...	M
9	1.692307692307690	0.7240000000...	M

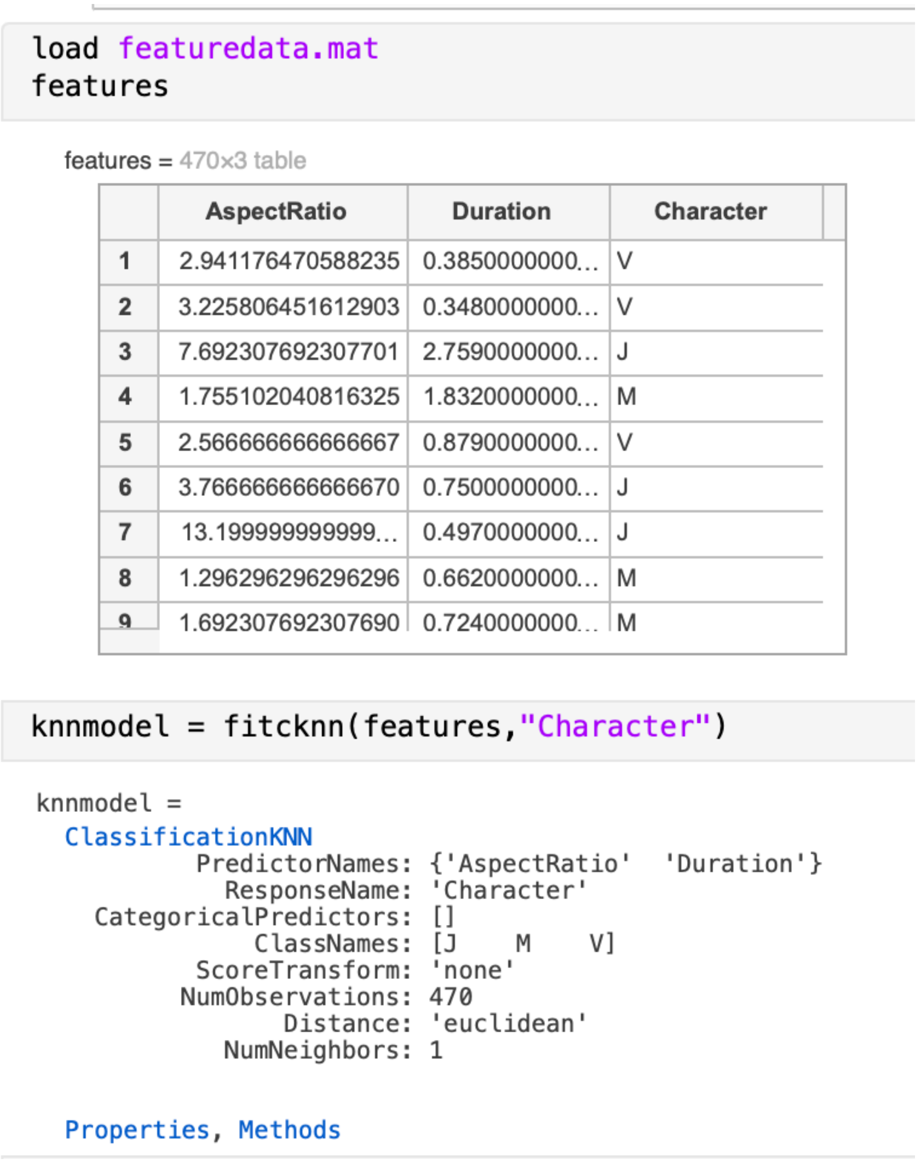
```
scatter(features.AspectRatio,features.Duration)
```



```
gscatter(features.AspectRatio,features.Duration,features.Character)
```



#Implementing k- nearest neighbour(kNN) model to fit the ‘features’ data to classify ‘Character’ variable



```
load featuredata.mat
features
```

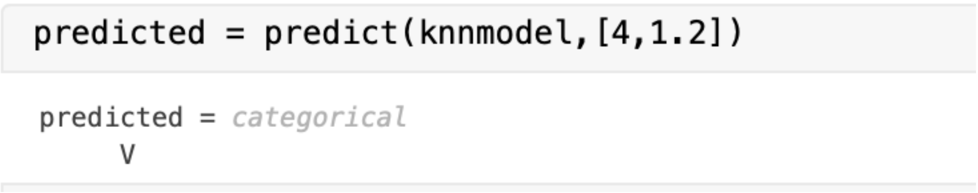
features = 470×3 table

	AspectRatio	Duration	Character
1	2.941176470588235	0.3850000000…	V
2	3.225806451612903	0.3480000000…	V
3	7.692307692307701	2.7590000000…	J
4	1.755102040816325	1.8320000000…	M
5	2.566666666666667	0.8790000000…	V
6	3.766666666666670	0.7500000000…	J
7	13.199999999999…	0.4970000000…	J
8	1.296296296296296	0.6620000000…	M
9	1.692307692307690	0.7240000000…	M

```
knnmodel = fitcknn(features,"Character")
```

```
knnmodel =
  ClassificationKNN
           PredictorNames: {'AspectRatio'  'Duration'}
             ResponseName: 'Character'
    CategoricalPredictors: []
               ClassNames: [J     M     V]
           ScoreTransform: 'none'
          NumObservations: 470
                 Distance: 'euclidean'
             NumNeighbors: 1

  Properties, Methods
```

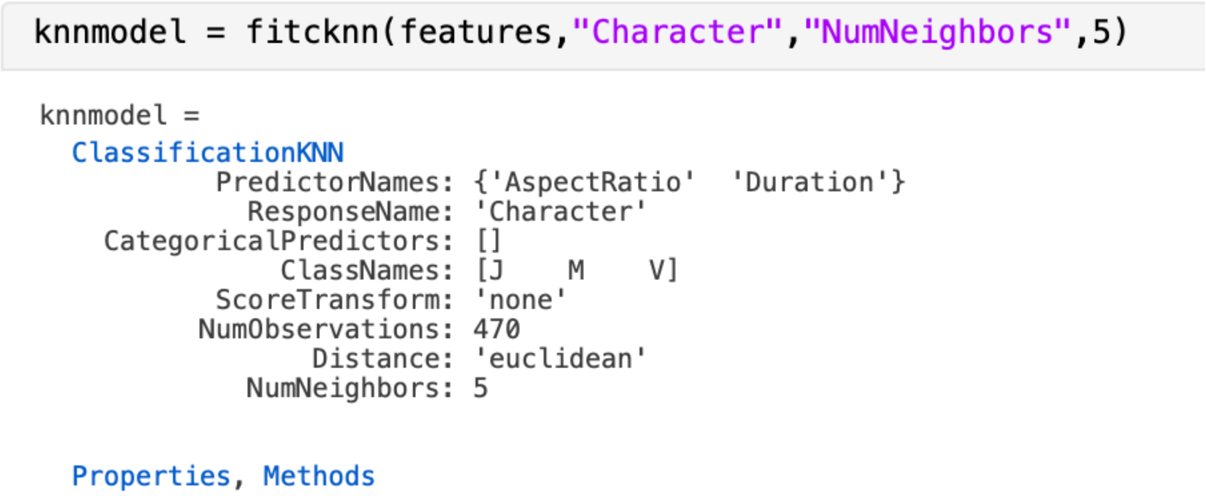
#Making predictions



```
predicted = predict(knnmodel,[4,1.2])
```

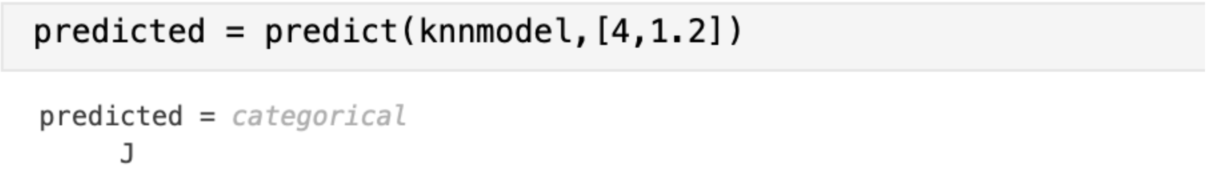
```
predicted = categorical
     V
```

#the model sensitive to any outliers in the training data, increasing k value to 5



```
knnmodel = fitcknn(features,"Character","NumNeighbors",5)
```

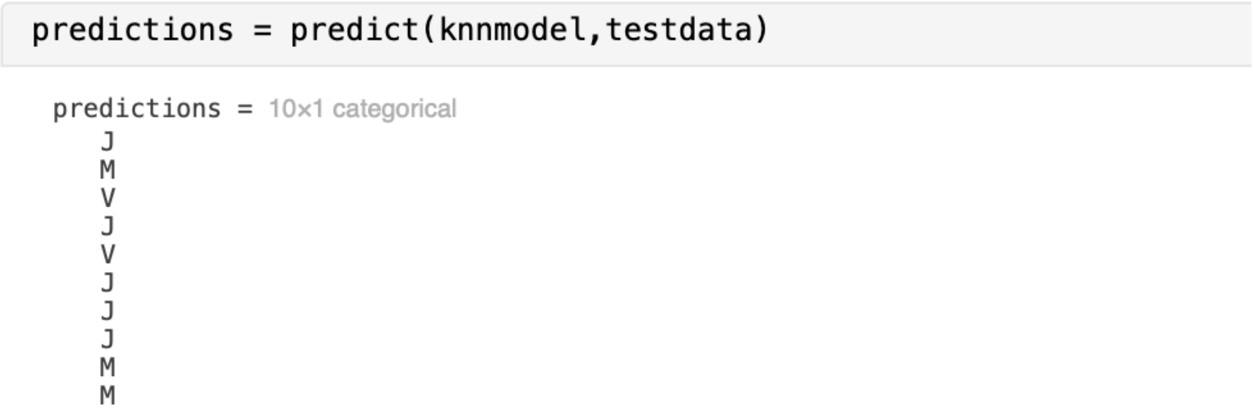
```
knnmodel = 
  ClassificationKNN
           PredictorNames: {'AspectRatio'  'Duration'}
             ResponseName: 'Character'
    CategoricalPredictors: []
               ClassNames: [J    M    V]
           ScoreTransform: 'none'
          NumObservations: 470
                 Distance: 'euclidean'
             NumNeighbors: 5


  Properties, Methods
```



```
predicted = predict(knnmodel,[4,1.2])
```

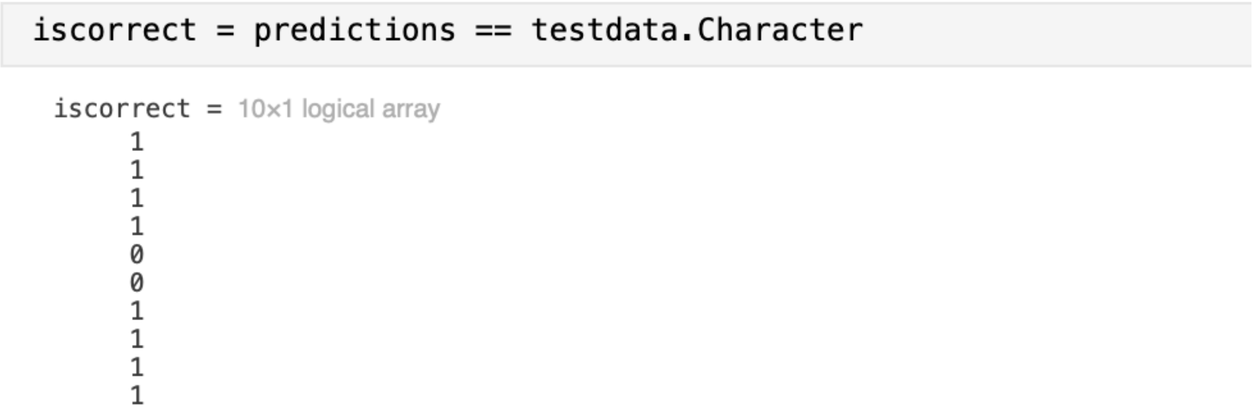
```
predicted = categorical
     J
```

#Evaluating the model by making predictions and creating a vector of 0s and 1s if wrong and right prediction respectively



```
predictions = predict(knnmodel,testdata)
```

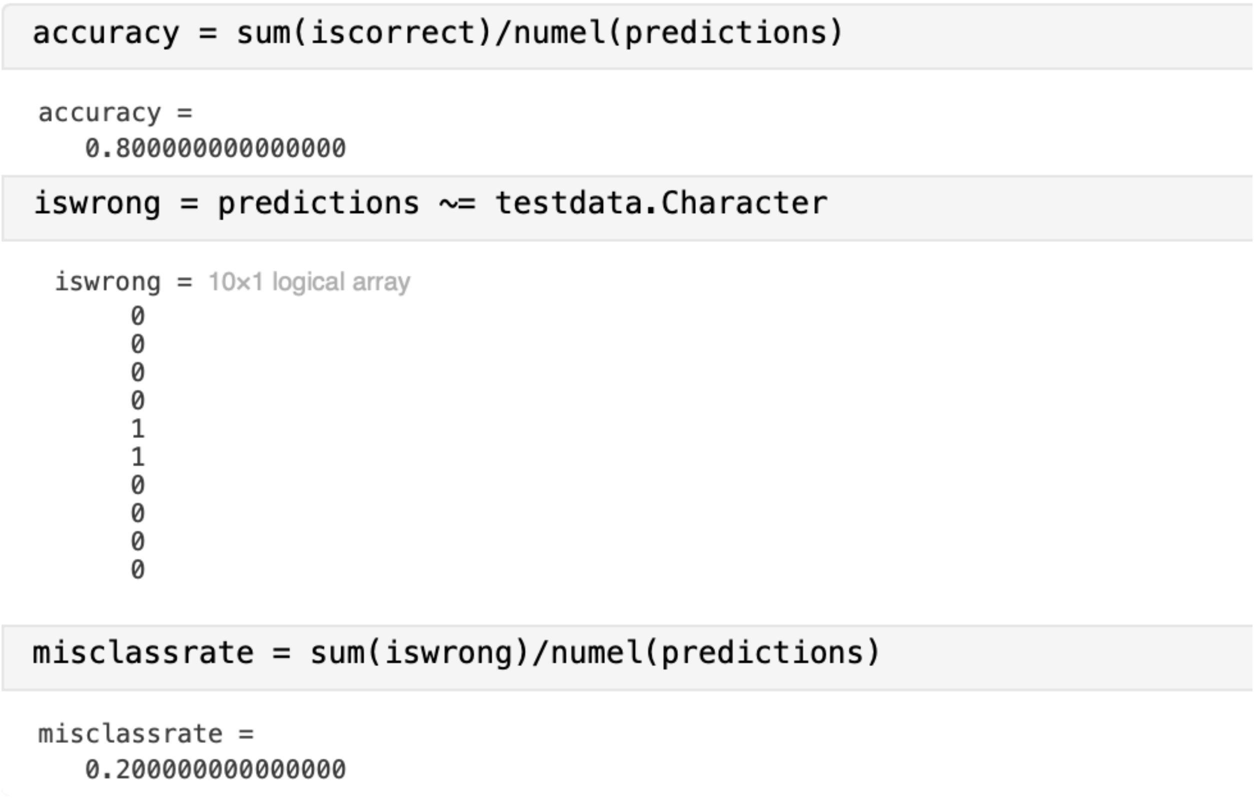
```
predictions = 10×1 categorical
   J
   M
   V
   J
   V
   J
   J
   J
   M
   M
```



```
iscorrect = predictions == testdata.Character
```

```
iscorrect = 10×1 logical array
     1
     1
     1
     1
     0
     0
     1
     1
     1
     1
```

#Calculating the accuracy and misclassification rate



```
accuracy = sum(iscorrect)/numel(predictions)
```

```
accuracy =
   0.800000000000000
```

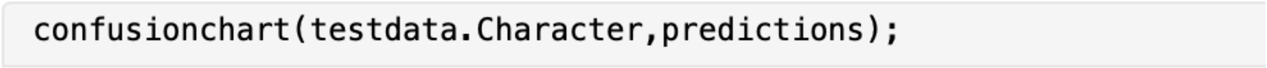
```
iswrong = predictions ~= testdata.Character
```

```
iswrong = 10×1 logical array
     0
     0
     0
     0
     1
     1
     0
     0
     0
     0
```

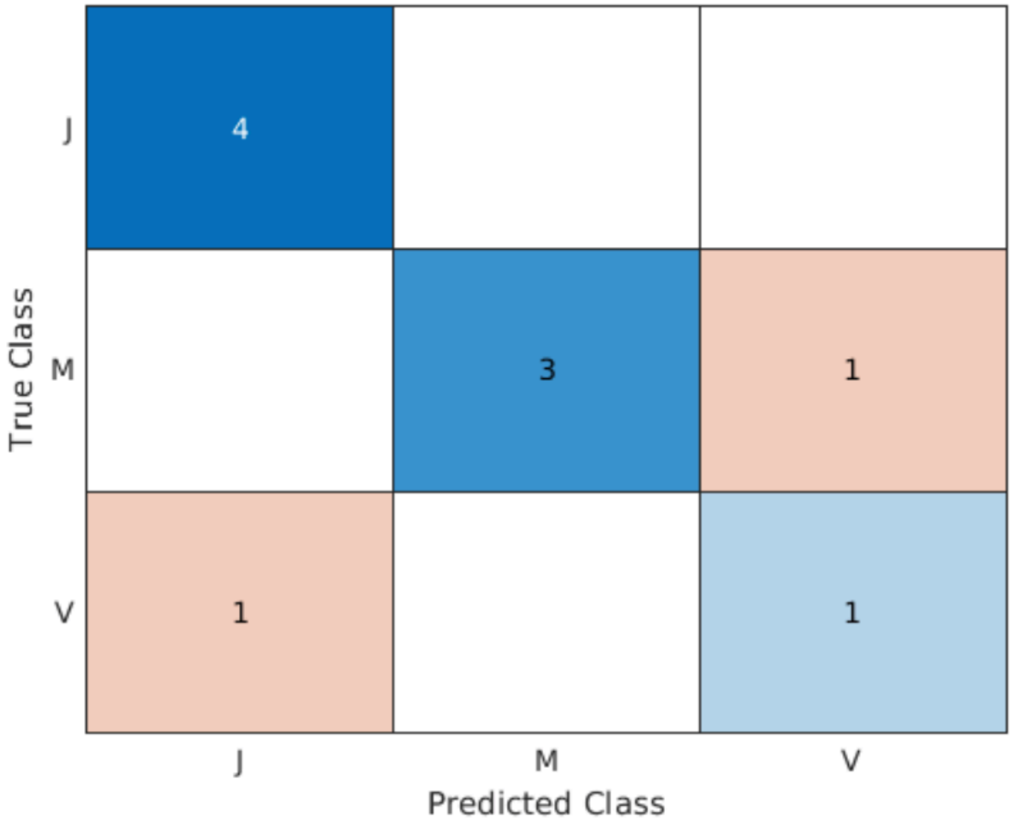
```
misclassrate = sum(iswrong)/numel(predictions)
```

```
misclassrate =
   0.200000000000000
```

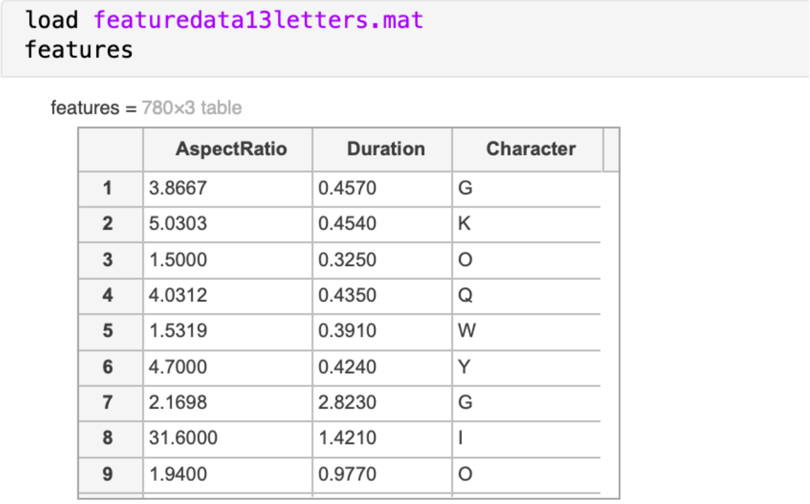
#Plotting the confusion chart



```
confusionchart(testdata.Character,predictions);
```



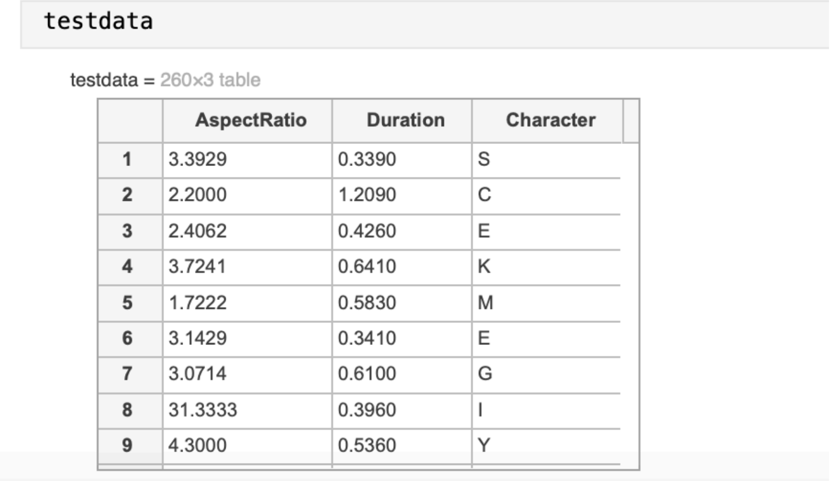
#Making a model for 13 letters



```
load featuredata13letters.mat
features
```

features = 780×3 table

	AspectRatio	Duration	Character
1	3.8667	0.4570	G
2	5.0303	0.4540	K
3	1.5000	0.3250	O
4	4.0312	0.4350	Q
5	1.5319	0.3910	W
6	4.7000	0.4240	Y
7	2.1698	2.8230	G
8	31.6000	1.4210	I
9	1.9400	0.9770	O



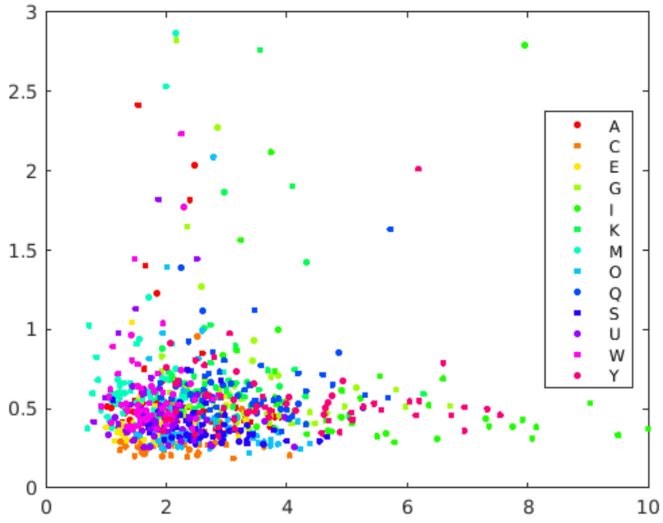
```
testdata
```

testdata = 260×3 table

	AspectRatio	Duration	Character
1	3.3929	0.3390	S
2	2.2000	1.2090	C
3	2.4062	0.4260	E
4	3.7241	0.6410	K
5	1.7222	0.5830	M
6	3.1429	0.3410	E
7	3.0714	0.6100	G
8	31.3333	0.3960	I
9	4.3000	0.5360	Y

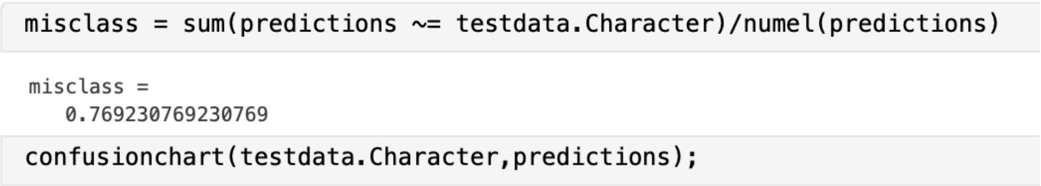
#plotting the scatter plot and Training the kNN model

```
gscatter(features.AspectRatio,features.Duration,features.Character)
xlim([0 10])
```



```
knnmodel = fitcknn(features,"Character","NumNeighbors",5);
predictions = predict(knnmodel,testdata);
```

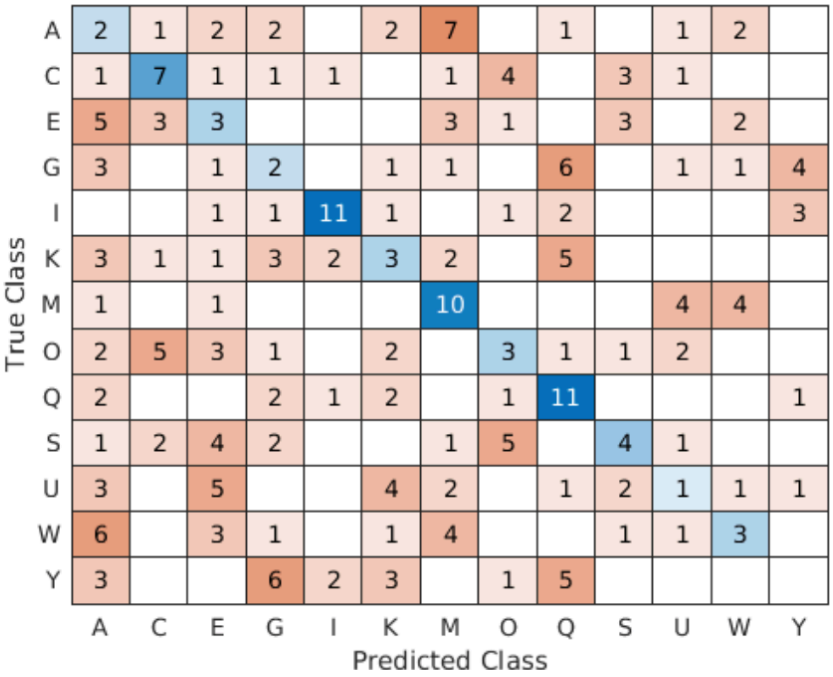
#Calculating misclassification rate and plating confusion chart



```
misclass = sum(predictions ~= testdata.Character)/numel(predictions)
```

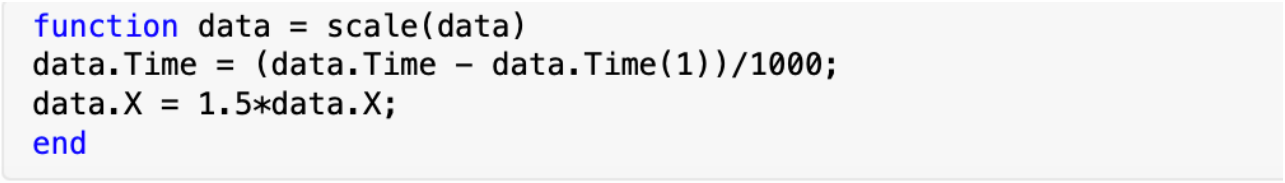
```
misclass =
   0.769230769230769
```

```
confusionchart(testdata.Character,predictions);
```



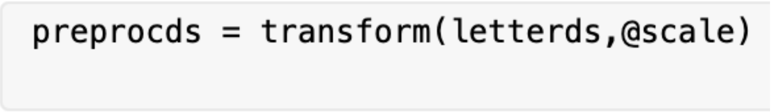
	A	C	E	G	I	K	M	O	Q	S	U	W	Y
A	2	1	2	2		2	7		1		1	2	
C	1	7	1	1	1		1	4		3	1		
E	5	3	3				3	1		3		2	
G	3		1	2		1	1		6		1	1	4
I			1	1	11	1		1	2				3
K	3	1	1	3	2	3	2		5				
M	1		1				10				4	4	
O	2	5	3	1		2		3	1	1	2		
Q	2			2	1	2		1	11				1
S	1	2	4	2			1	5		4	1		
U	3		5			4	2		1	2	1	1	1
W	6		3	1		1	4			1	1	3	
Y	3			6	2	3		1	5				

#adding a preprocessing function to scale the x-position to range [0,1.5]



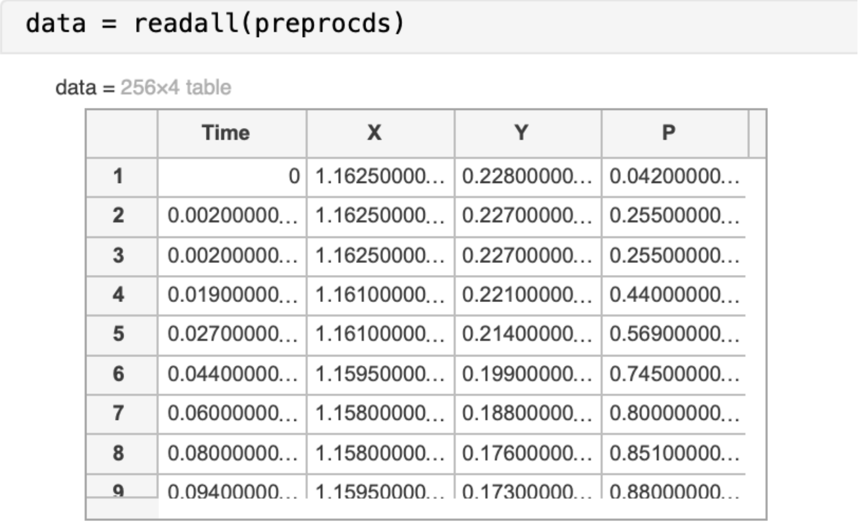
```
function data = scale(data)
data.Time = (data.Time - data.Time(1))/1000;
data.X = 1.5*data.X;
end
```

#using a function handle to call the function



```
preprocds = transform(letterds,@scale)
```

#scaling the data automatically whenever data Is read from preprocds datastore

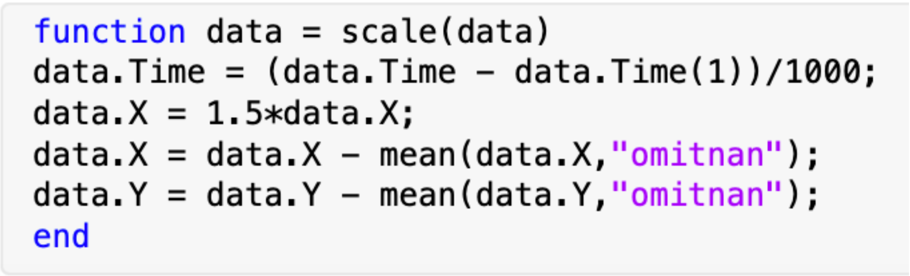


```
data = readall(preprocds)
```

data = 256×4 table

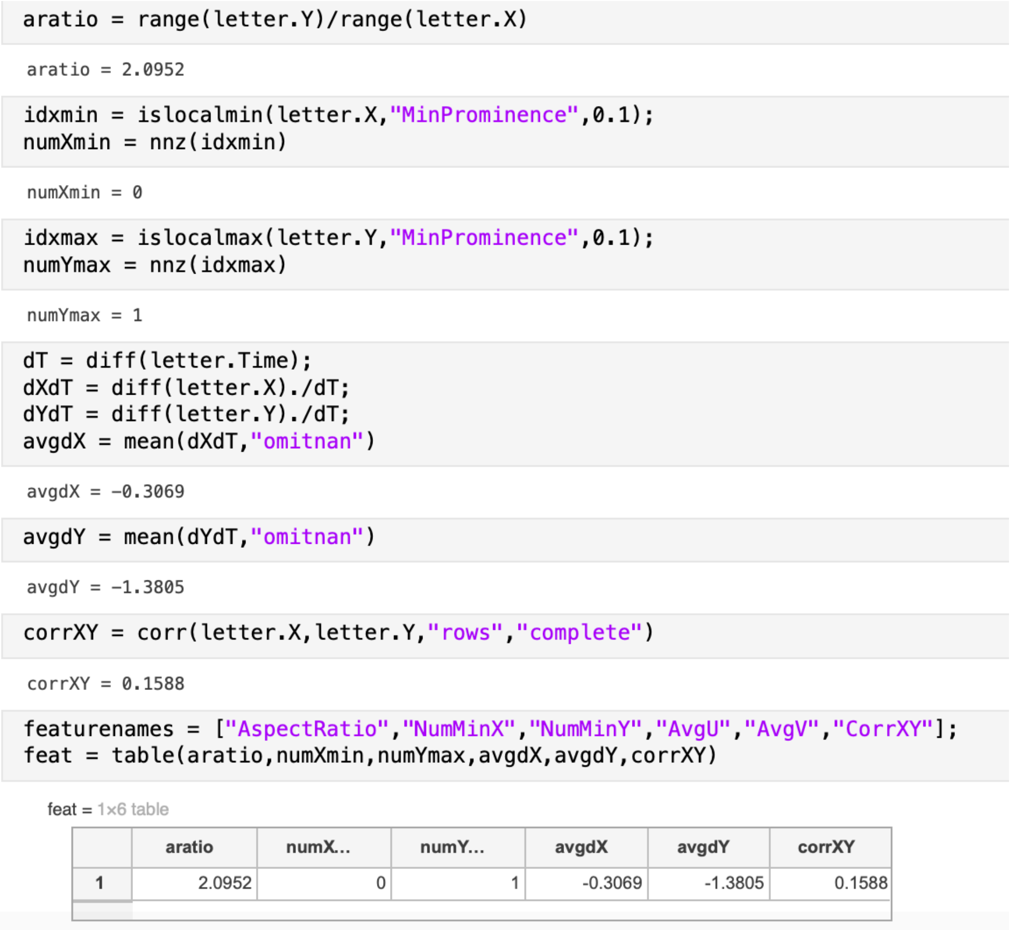
	Time	X	Y	P
1	0	1.16250000...	0.22800000...	0.04200000...
2	0.00200000...	1.16250000...	0.22700000...	0.25500000...
3	0.00200000...	1.16250000...	0.22700000...	0.25500000...
4	0.01900000...	1.16100000...	0.22100000...	0.44000000...
5	0.02700000...	1.16100000...	0.21400000...	0.56900000...
6	0.04400000...	1.15950000...	0.19900000...	0.74500000...
7	0.06000000...	1.15800000...	0.18800000...	0.80000000...
8	0.08000000...	1.15800000...	0.17600000...	0.85100000...
9	0.09400000...	1.15950000...	0.17300000...	0.88000000...

#normalising the data to for classification in the preprocessing by modifying the scaling function written earlier and omitting NaN in the data preprocessing step



```
function data = scale(data)
data.Time = (data.Time - data.Time(1))/1000;
data.X = 1.5*data.X;
data.X = data.X - mean(data.X,"omitnan");
data.Y = data.Y - mean(data.Y,"omitnan");
end
```

Creating a feature table for aspect ratio, numXmin, numYmax , avgdX, avgdY, corrXY



```
aratio = range(letter.Y)/range(letter.X)
```

```
aratio = 2.0952
```

```
idxmin = islocalmin(letter.X,"MinProminence",0.1);
numXmin = nnz(idxmin)
```

```
numXmin = 0
```

```
idxmax = islocalmax(letter.Y,"MinProminence",0.1);
numYmax = nnz(idxmax)
```

```
numYmax = 1
```

```
dT = diff(letter.Time);
dXdT = diff(letter.X)./dT;
dYdT = diff(letter.Y)./dT;
avgdX = mean(dXdT,"omitnan")
```

```
avgdX = -0.3069
```

```
avgdY = mean(dYdT,"omitnan")
```

```
avgdY = -1.3805
```

```
corrXY = corr(letter.X,letter.Y,"rows","complete")
```

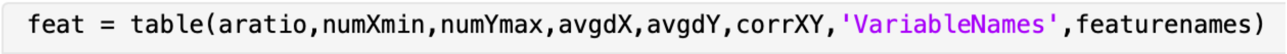
```
corrXY = 0.1588
```

```
featurenames = ["AspectRatio","NumMinX","NumMinY","AvgU","AvgV","CorrXY"];
feat = table(aratio,numXmin,numYmax,avgdX,avgdY,corrXY)
```

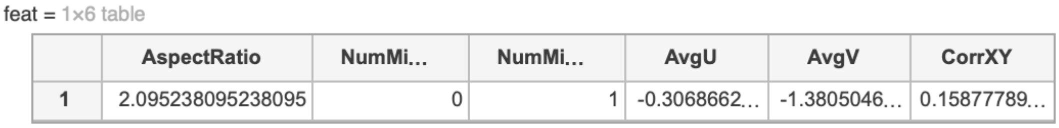
feat = 1×6 table

	aratio	numX...	numY...	avgdX	avgdY	corrXY
1	2.0952	0	1	-0.3069	-1.3805	0.1588

#Table variable names are stored in the array ‘featurenames’



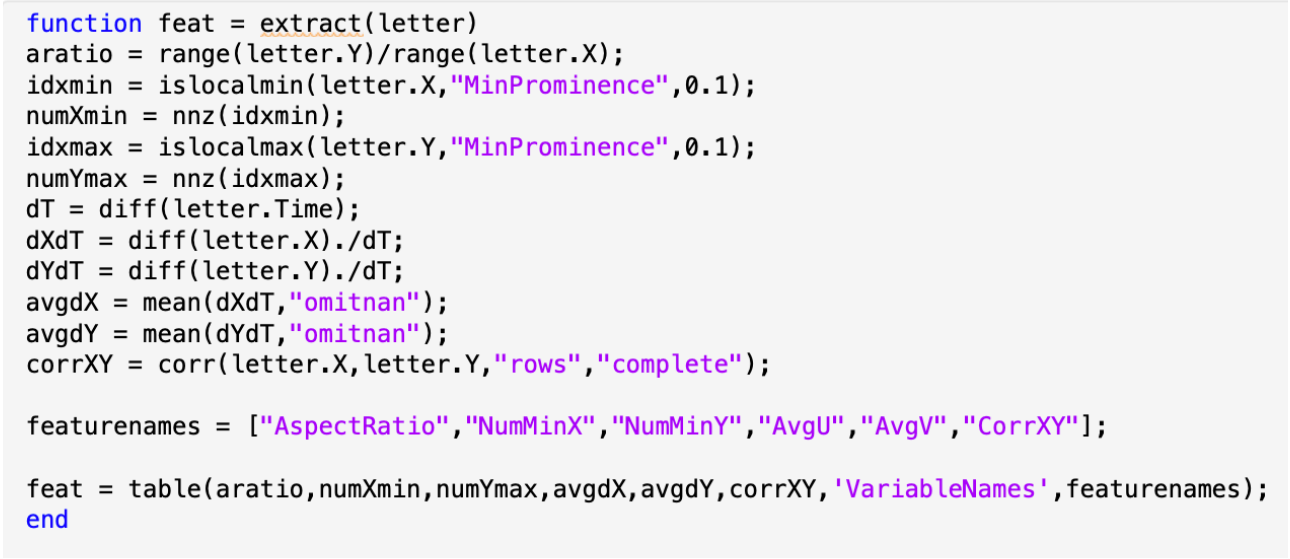
```
feat = table(aratio,numXmin,numYmax,avgdX,avgdY,corrXY,'VariableNames',featurenames)
```



feat = 1×6 table

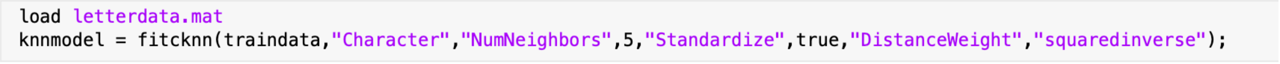
	AspectRatio	NumMi...	NumMi...	AvgU	AvgV	CorrXY
1	2.095238095238095	0	1	-0.3068662...	-1.3805046...	0.15877789...

# Creating a function to calculate ratio, numXmin, numYmax , avgdX, avgdY, corrXY automatically when the data is read



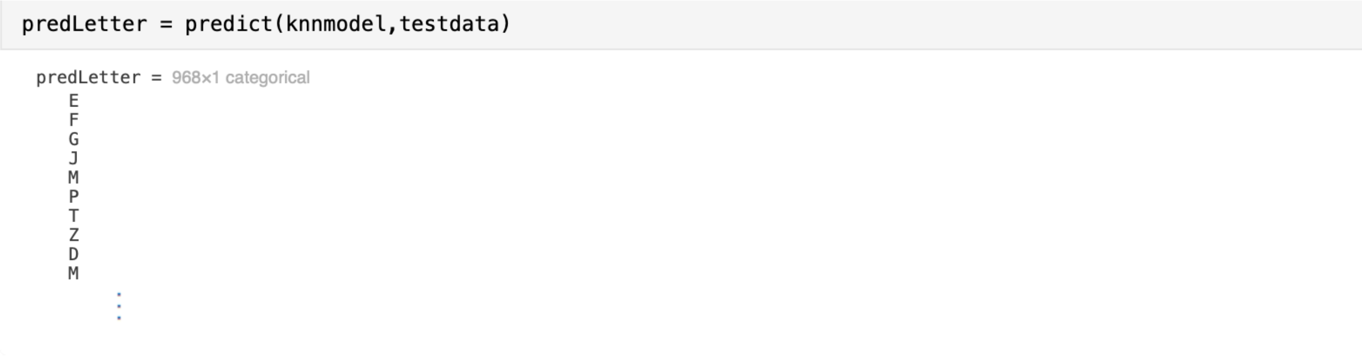
```
function feat = extract(letter)
aratio = range(letter.Y)/range(letter.X);
idxmin = islocalmin(letter.X,"MinProminence",0.1);
numXmin = nnz(idxmin);
idxmax = islocalmax(letter.Y,"MinProminence",0.1);
numYmax = nnz(idxmax);
dT = diff(letter.Time);
dXdT = diff(letter.X)./dT;
dYdT = diff(letter.Y)./dT;
avgdX = mean(dXdT,"omitnan");
avgdY = mean(dYdT,"omitnan");
corrXY = corr(letter.X,letter.Y,"rows","complete");

featurenames = ["AspectRatio","NumMinX","NumMinY","AvgU","AvgV","CorrXY"];

feat = table(aratio,numXmin,numYmax,avgdX,avgdY,corrXY,'VariableNames',featurenames);
end
```

Loading the ‘letterdata.mat’ file which contains both train data and test data for the 14 hand written letters and implementing kNN on it



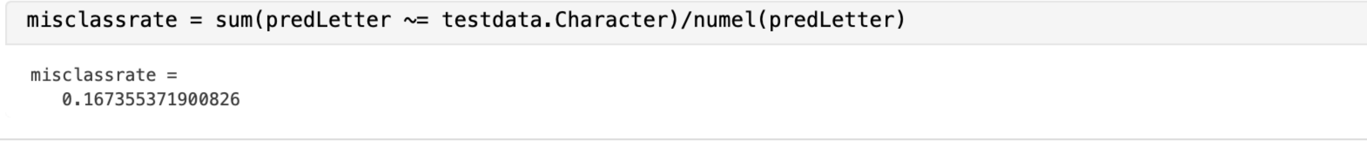
```
load letterdata.mat
knnmodel = fitcknn(traindata,"Character","NumNeighbors",5,"Standardize",true,"DistanceWeight","squaredinverse");
```

#Predicting the characters and calculating the misclassification rate and loss



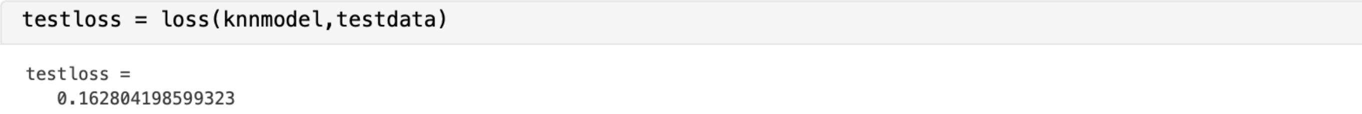
```
predLetter = predict(knnmodel,testdata)
```

```
predLetter = 968×1 categorical
    E
    F
    G
    J
    M
    P
    T
    Z
    D
    M
        ⋮
```



```
misclassrate = sum(predLetter ~= testdata.Character)/numel(predLetter)
```

```
misclassrate =
   0.167355371900826
```



```
testloss = loss(knnmodel,testdata)
```

```
testloss =
   0.162804198599323
```

Improving the model :
The model can be improved using the following techniques

- Using more or better data to train the model
- Building different models or trying out different k values
- Using cross-validation data set to select a k value.