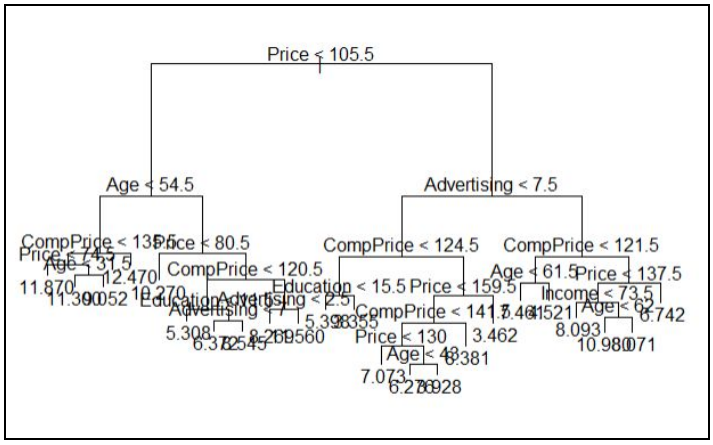
Rhys Jennings - STAT318 - Assignment 3

16/05/2020

1.

```r
library(tree)
#Train the ree
training_tree = tree(Sales~., training)

summary(training_tree)
plot(training_tree)
text(training_tree, pretty=0)

#Find MSE for the training data
training_prediction=predict(training_tree, training)
mean((training_prediction - training[,'Sales'])^2)

#Find MSE for the testing data
testing_prediction=predict(training_tree, testing)
mean((testing_prediction - testing[,'Sales'])^2)
```

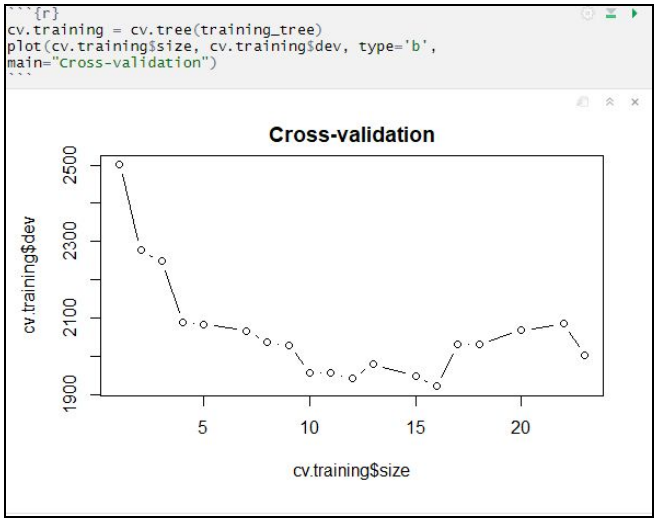
```
Regression tree:
tree(formula = Sales ~ ., data = training)
Variables actually used in tree construction:
[1] "Price"       "Age"         "CompPrice"   "Education"   "Advertising"
[6] "Income"
Number of terminal nodes:  23
Residual mean deviance:  3.296 = 913 / 277
Distribution of residuals:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 -4.994  -1.178   0.037   0.000   1.390   4.627
[1] 3.043394
[1] 6.03393
```



a) We can see (on the console output), that fitting the training dataset resulted in a MSE of 3.04 on the training data and predicting the testing data resulted in a 6.03 MSE.

b) It appears a size of 16 gives a good result. For the tree size. So when we prune we get train_mse = 3.71, test_mse=5.79.

```r
cv.training = cv.tree(training_tree)
plot(cv.training$size, cv.training$dev, type='b',
main="Cross-validation")
```




```r
prune.training=prune.tree(training_tree,best=16)
#plot(prune.training)
#text(prune.training, pretty=0)

training_prediction=predict(prune.training, training)

mean((training_prediction - training[,'Sales'])^2)

testing_prediction=predict(prune.training, testing)

mean((testing_prediction - testing[,'Sales'])^2)
```

```
[1] 3.70938
[1] 5.793296
```

This means the pruning performs better on the testing set, while not quite as good on the training set. This is expected as we are making the tree less accurate on the training set so we are not over fitting so much, which decreases our testing error.

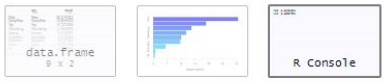
c) Our bagged regression tree performs much better than before, with a mse for the training error = 0.85, and for the testing error 4.353. It appears decorrelating the trees was certainly an effective strategy as it decreases the variance of our problem.

```r
library(randomForest)

bag.training=randomForest(Sales~., data= training, mtry=9,
importance=TRUE)

training_prediction=predict(bag.training, training)

mean((training_prediction - training[,'Sales'])^2)

testing_prediction=predict(bag.training, testing)

mean((testing_prediction - testing[,'Sales'])^2)
```

```
[1] 0.8492177
[1] 4.353752
```

d) After trying different variations, using different loops using different variables, the best MSE I discovered was using the amount of trees at 1000, a depth of 5 and a shrinkage of 0.01. This produced a MSE for the training at 1.66 and a MSE for the testing at 3.89, which is the lowest of all the models. This seemed to be a good balance of where the MSE test was at its lowest and the MSE of the training was also reasonably low

```r
boost.training=gbm(Sales~., data=training, distribution="gaussian",n.trees=1000
                   interaction.depth=5, shrinkage=0.01)

training_prediction=predict(boost.training, training, n.trees=1000)

MSEtrain <- mean((training_prediction - training[,'Sales'])^2)

testing_prediction=predict(boost.training, testing, n.trees=1000)

MSEtest <- mean((testing_prediction - testing[,'Sales'])^2)
summary(boost.training)

MSEtrain
MSEtest
```




```
[1] 1.655161
[1] 3.892795
```

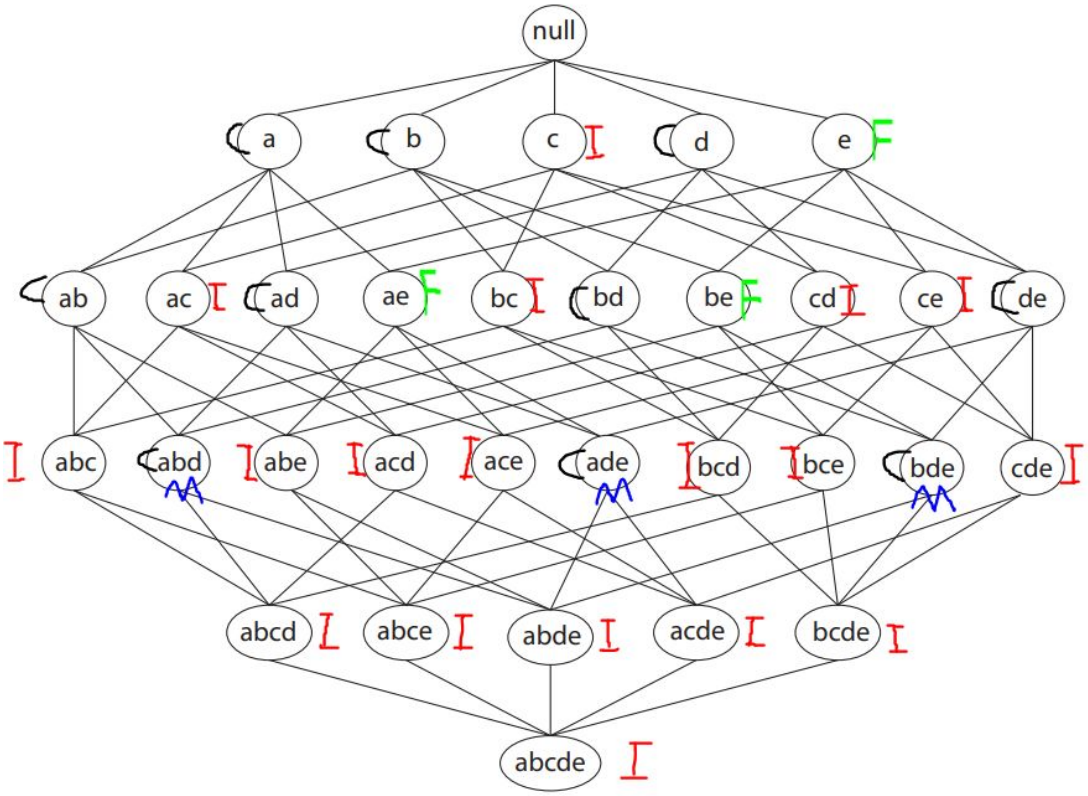
e) The boosted tree produced the best testing MSE and was reasonably close to the MSE training for the bagged tree. We can see from the output of the summary for the boosted tree that Price, CompPrice, and Age are the three predictors most important to this model.

	var <fctr>	rel.inf <dbl>
Price	Price	27.2951671
CompPrice	CompPrice	17.5410767
Age	Age	14.2597144
Population	Population	11.6019235
Income	Income	11.5807694
Advertising	Advertising	11.0033950
Education	Education	5.3049973
Urban	Urban	0.7428077
US	US	0.6701490

9 rows

Rhys Jennings

2)a)



b)

- Confidence = supp(X U Y) / supp(X)

  supp({d, e} U {b}) = 3/10 = 0.3
  supp({d,e}) = 5/10 = 0.5

  Confidence = 0.3/0.5 = 0.6

- Lift = supp(X U Y) / (supp(X) * supp(Y))

  supp({b}) = 8/10 = 0.8

  Lift = 0.3 / (0.5 * 0.8) = 0.75

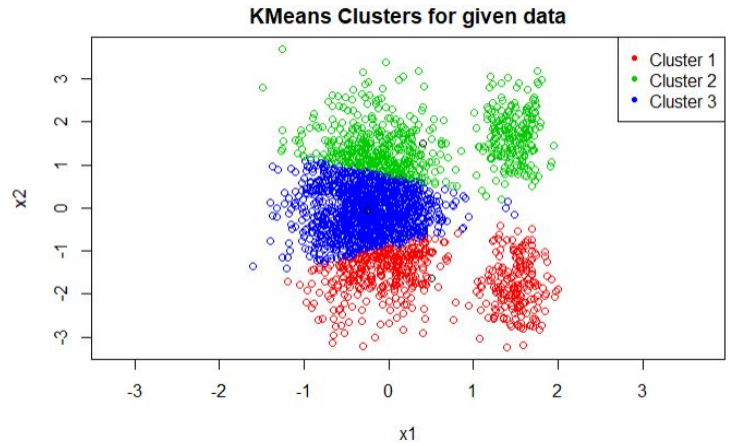
- The confidence of 60% is reasonably low, which means for all transactions including {d,e}, {b} is in 60% of them. The lift is 0.75, which is less than one. This implies that the presence of one item has a negative effect on the presence of the other, meaning the occurrence of {d,e} reduces the likelihood of {b}.

Rhys Jennings

3)

a)

```r
km.out=kmeans(clusterData[,1:2], 3, nstart=50)

all_points = c(clusterData$x1, clusterData$x2)
range = c(min(all_points), max(all_points))

plot(clusterData[,1:2,],col=(km.out$cluster+1), |
     xlim=range, ylim=range, main="KMeans Clusters for given data")
points(km.out$centers)
legend("topright", legend=c("Cluster 1", "Cluster 2", "Cluster 3"), col=c(2,3,4), pch=20,)
```

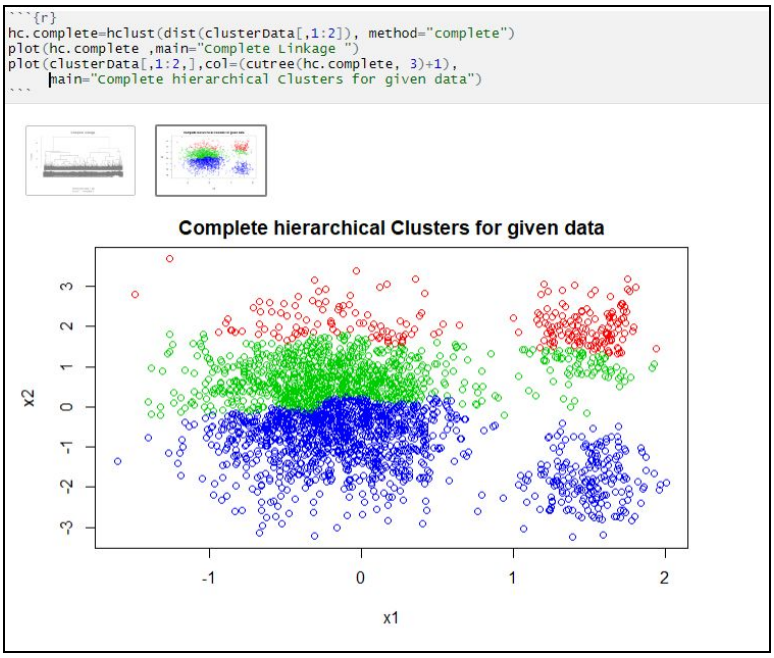


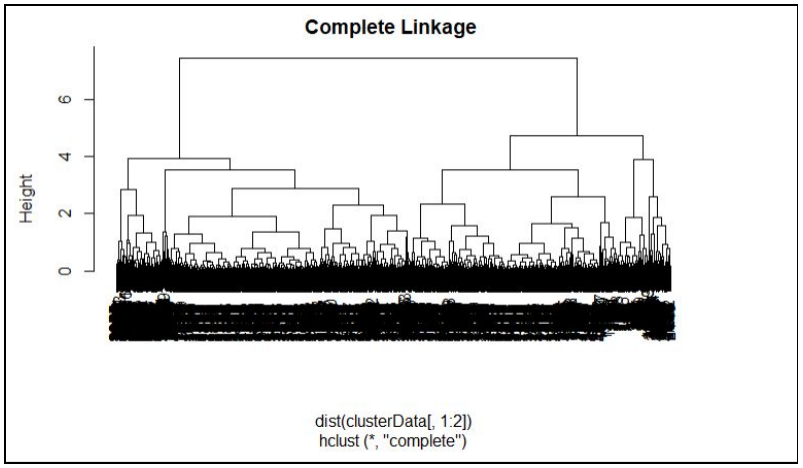
Note: Means are displayed on the graph as black points.

b)

Complete

```r
hc.complete=hclust(dist(clusterData[,1:2]), method="complete")
plot(hc.complete ,main="Complete Linkage ")
plot(clusterData[,1:2,],col=(cutree(hc.complete, 3)+1),
     main="Complete hierarchical Clusters for given data")
```

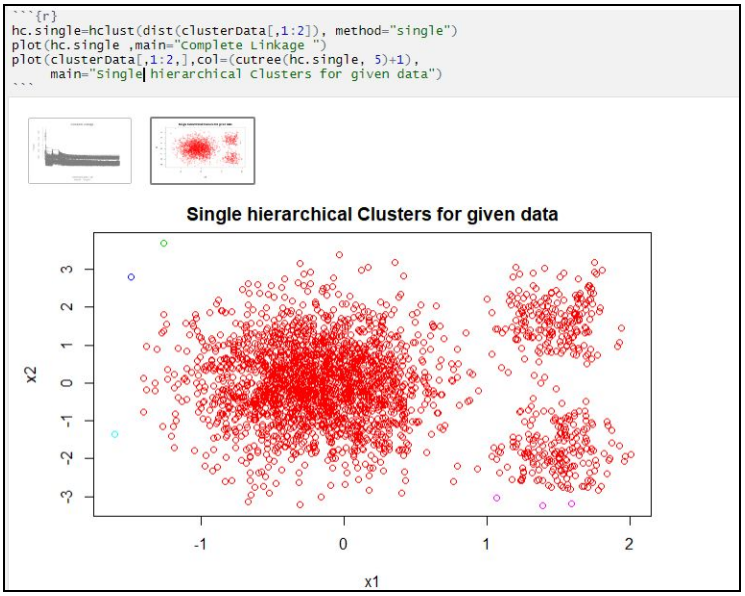


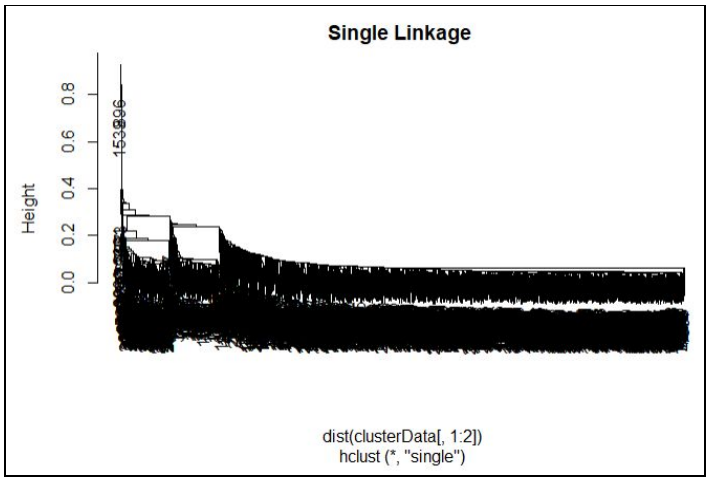


Rhys Jennings

Single

```r
hc.single=hclust(dist(clusterData[,1:2]), method="single")
plot(hc.single ,main="Complete Linkage ")
plot(clusterData[,1:2,],col=(cutree(hc.single, 5)+1),
     main="Single hierarchical Clusters for given data")
```





c)

- The Kmeans clustering has trouble because one of the clusters is so much larger than the others. The incorrect clustering shown is because it is trying to minimise the within cluster variation. If the left cluster was just one cluster, the within variation would be larger than the model it shows, because it is a large cluster with lots of observations, which would mean the within cluster variation would be large. Kmeans works better when the clusters are of equal size.

- The complete hierarchical clustering performs ok, but still not very accurate. Complete hierarchical clustering allows for points to be closer to points in other clusters than points in their own cluster.

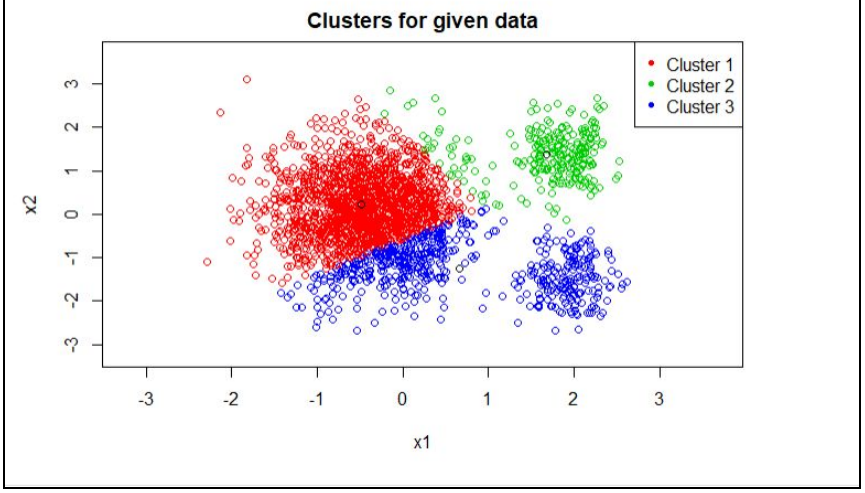
- The single hierarchical does not perform well and this is because there are two large outliers that are added to do the hierarchy last, meaning when we cut the tree at 3, we are left with each point (as they are an individual cluster that has not been added), and the large cluster of all the other points that have been added together.

Rhys Jennings

d)

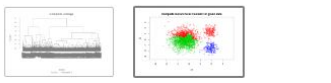
Kmeans

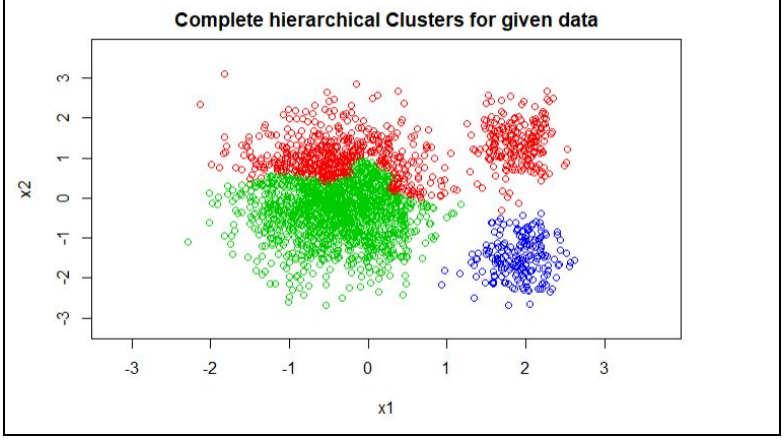
```r
x = scale(clusterData[,1:2], center=TRUE, scale=TRUE)

km.out=kmeans(x, 3, nstart=50)

all_points = c(clusterData$x1, clusterData$x2)
range = c(min(all_points), max(all_points))

plot(x,col=(km.out$cluster+1),
     xlim=range, ylim=range, main="KMeans Clusters for given data")
points(km.out$centers)
legend("topright", legend=c("Cluster 1", "Cluster 2", "Cluster 3"), col=c(2,3,4), pch=20,)
```

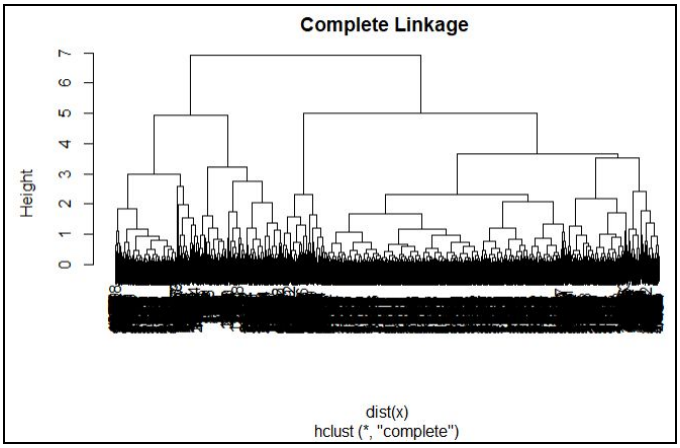


Complete

```r
hc.complete=hclust(dist(x), method="complete")
plot(hc.complete ,main="Complete Linkage ")
plot(x,col=(cutree(hc.complete, 3)+1),
     main="Complete hierarchical Clusters for given data",xlim=range, ylim=range )
```



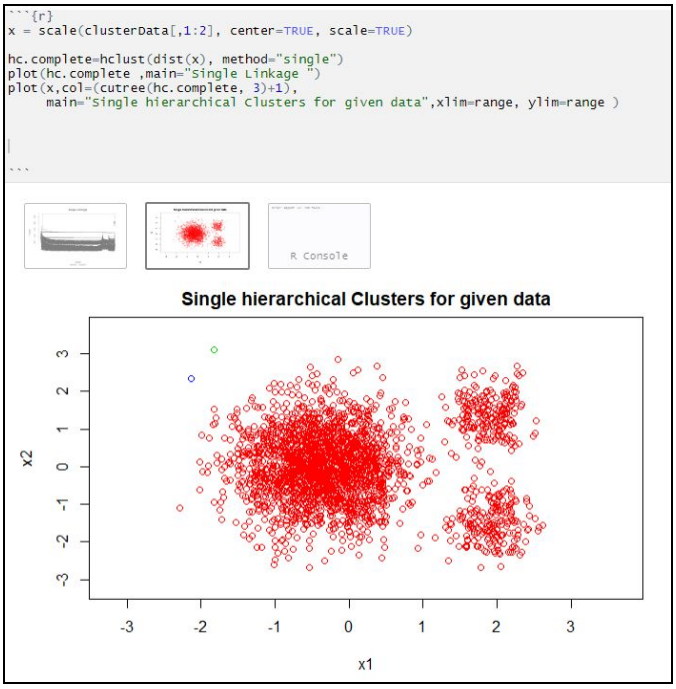


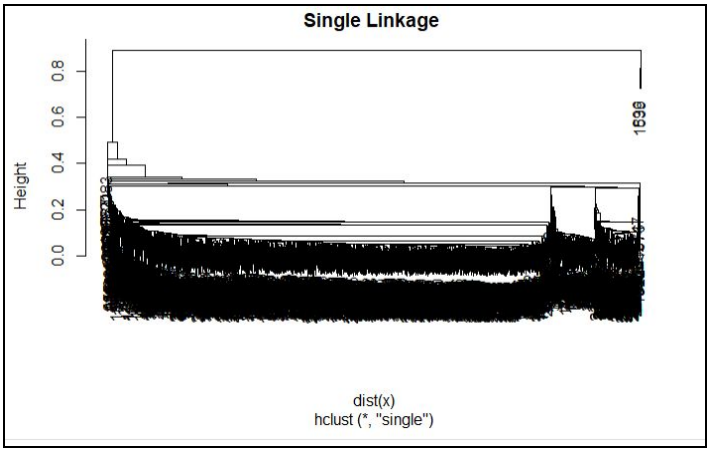


Rhys Jennings

Single

```r
x = scale(clusterData[,1:2], center=TRUE, scale=TRUE)

hc.complete=hclust(dist(x), method="single")
plot(hc.complete ,main="Single Linkage ")
plot(x,col=(cutree(hc.complete, 3)+1),
     main="Single hierarchical Clusters for given data",xlim=range, ylim=range )
```





- **KMeans:** This certainly improves our kmeans clustering. Scaling gives the same importance to all the variables so it improves our results
- **Complete:** Scaling appears to improve our complete clustering. This is because complete hierarchical clusters require calculation of distance. If we don't scale, attributes that have larger magnitudes are given more importance which affect the clustering. Scaling reduces this and provides a better model than not scaling,
- **Single:** Produces the same clustering. This is because even though our values are scaled, the two outliers still form their own cluster, meaning when we look at 3 clusters, there will be one cluster on each point, and every other point in the same cluster.