Partitional clustering
R Code

Yang Li

Section 1.1

Doing K-Means (Lloyd's algorithm)

Use the following code in R:

```
k <- kmeans(comp, x, algorithm='Lloyd', iter.max=1000)
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

x – Replace this with the number of clusters to find e.g. 2, 3. You can find out how many clusters is the optimal number by following the guide in Chapter 3.

Other parameters:

k – This is the name of the object that the results of the clustering are going to be stored in. It should be left as k because the follow up code (for plotting, comparing etc.) assumes it is called k.

algorithm='Lloyd' – Lloyd is the name of one algorithm that does K-Means. Refer to section 2 of Chapter 2 to learn about other algorithms for K-Means.

iter.max=1000 – This is the maximum number of iterations the algorithm is allowed to use before it is forcefully stopped, whether an optimum is reached or not. As such, it should be set to a large number like 1000 to make sure the algorithm finishes, though only a few dozen should usually be enough. If the algorithm does not converge before it reaches its maximum number of iterations, it will give a warning in R.

Plotting the results in a scatterplot

You can use the package **ggplot2** to plot the results of the clustering with colours distinguishing the clusters. This package will need to be installed.

If you have **2 dimensions**:

```
library(ggplot2)
ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=k$clust, size=3,
pch=16)
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data.

x=PC1, y=PC2 – This will be the name of your two dimensions if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of the dimensions in your data e.g. x=height, y=weight.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you can try fine-tuning this number if the result looks bad.

color=k$clust – This makes it so that your points are coloured based on what cluster they're in.

size=3 – This determines the size of your points. If the points are too big, it covers too much area and makes it impossible to tell where the point's position actually is. If it is too small, it will be hard to see. 3 is a good balance, though if you have a lot of points, you may wish to try a smaller number.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to k$clust. However, this may be excessively distracting if your points are already differently coloured.

If you have **more than 2 dimensions:**

You will need to do multiple graphs because a 2D graph can only plot 2 dimensions at once (the **rgl** package can plot 3D graphs but this is not useful for papers). The number of graphs you need will be equal to the number of ways your dimensions can be paired. You can find out how many dimensions you need can by typing choose(x,2) into R, where x is the number of dimensions.

The **gridExtra** package can be used to display multiple graphs on the same page. The code below assumes you have 3 dimensions. Each line which begins with pc is a new graph, the last lines joins all the graphs together.

```
library(gridExtra)
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, ncol=2)
```

If you have 4 dimensions:

```
pc12  <-  ggplot(comp,  aes(x=PC1,  y=PC2))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc13  <-  ggplot(comp,  aes(x=PC1,  y=PC3))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc23  <-  ggplot(comp,  aes(x=PC2,  y=PC3))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc14  <-  ggplot(comp,  aes(x=PC1,  y=PC4))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc24  <-  ggplot(comp,  aes(x=PC2,  y=PC4))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc34  <-  ggplot(comp,  aes(x=PC3,  y=PC4))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, ncol=3)
```

If you have 5 dimensions:

```
pc12  <-  ggplot(comp,  aes(x=PC1,  y=PC2))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc13  <-  ggplot(comp,  aes(x=PC1,  y=PC3))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc23  <-  ggplot(comp,  aes(x=PC2,  y=PC3))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc14  <-  ggplot(comp,  aes(x=PC1,  y=PC4))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc24  <-  ggplot(comp,  aes(x=PC2,  y=PC4))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc34  <-  ggplot(comp,  aes(x=PC3,  y=PC4))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc15  <-  ggplot(comp,  aes(x=PC1,  y=PC5))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc25  <-  ggplot(comp,  aes(x=PC2,  y=PC5))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc35  <-  ggplot(comp,  aes(x=PC3,  y=PC5))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc45  <-  ggplot(comp,  aes(x=PC4,  y=PC5))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, pc15, pc25, pc35, pc45, ncol=3)
```

If you need to add even more dimensions, study what was added to the code (new parts in **bold**) and follow the pattern. Do not forget to add more objects to the grid.arrange list in the last line.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1’s guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

x=PC1, y=PC2 – This is present in a slightly altered form in each line. It should be the name of the two dimensions to be plotted for that graph. Replace it with the name of the dimensions in your data e.g. x=height, y=weight for one line, then x=height, y=age for the next line etc.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you may try fine-tuning this number if the result looks bad.

color=k$clust – This makes it so that your points are coloured based on what cluster they’re in.

size=3 – This determines the size of your points. If the points are too big, it covers too much area and makes it impossible to tell where the point’s position actually is. If it is too small, it will be hard to see. 3 is a good balance, though if you have a lot of points, you may wish to try a smaller number.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to k$clust. However, this may be excessively distracting if your points are already differently coloured.

ncol=3 – This is the number of columns to arrange the graphs in. If you have more graphs you will need to increase this number, though too many graphs do not fit well on one page. You can also use nrow=3 to set the number of rows.

Seeing what’s in each cluster

You can list the objects in each cluster with the following code:

```r
clust <- names(sort(table(k$clust)))
clustnumb <- length(clust)
for(i in 1:clustnumb){
  print(c("Cluster number:", clust[i]))
  print(row.names(comp[k$clust==clust[i],]))
}
sort(table(k$clust))
```

It lists the clusters by order of size. Because of this, you must be careful when comparing two different clustering algorithms on the same data. The order of clusters may not be the same between the 2 algorithms.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

Comparing clusters by each original variable

You can see how the individuals in your cluster compare by plotting boxplots for all of your original variables (before doing a PCA). The package **ggplot2** will be used again.

First, you need to add a new column to your data which can be used to separate the individuals into clusters for the boxplots.

```
NAMEOFDATA$Cluster <- k$cluster
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning). Keep the $Cluster part on the end.

Now you can compare the clusters across one variable using the following code:

```
qplot(factor(Cluster),   VARIABLE,   geom   =   "boxplot",   data   =   NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE')  +  geom_boxplot(aes(fill  =  factor(Cluster))  +
theme(legend.position="NULL")
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE – Replace this with the name of the variable. This is the column heading from your original data you would like to compare the clusters by e.g. height. Note that you need to replace it in 2 places in the code.

If you want to display multiple boxplots on one page, you can use **gridExtra** again.

```
box1 <- qplot(factor(Cluster), VARIABLE1, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE1') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

box2 <- qplot(factor(Cluster), VARIABLE2, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE2') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

box3 <- qplot(factor(Cluster), VARIABLE3, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE3') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

grid.arrange(box1, box2, box3, ncol=2)
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE1, VARIABLE2, VARIABLE3 etc. – Replace these with the names of the variables i.e. column headings from your original data you would like to compare the clusters by. Note that for each one, there are 2 places in the code to replace them.

Adding extra graphs is easy, just don’t forget to add box4, box5 etc. to the grid.arrange list in the last line.

Section 2.1

Doing K-Means (MacQueen's algorithm)

Use the following code in R:

```
k <- kmeans(comp, x, algorithm='MacQueen', iter.max=1000)
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

x – Replace this with the number of clusters to find e.g. 2, 3. You can find out how many clusters is the optimal number by following the guide in Chapter 3.

Other parameters:

k – This is the name of the object that the results of the clustering are going to be stored in. It should be left as k because the follow up code (for plotting, comparing etc.) assumes it is called k.

algorithm='MacQueen' – MacQueen is the name of one algorithm that does K-Means. Refer to section 2 of Chapter 2 to learn about other algorithms for K-Means.

iter.max=1000 – This is the maximum number of iterations the algorithm is allowed to use before it is forcefully stopped, whether an optimum is reached or not. As such, it should be set to a large number like 1000 to make sure the algorithm finishes, though only a few dozen should usually be enough. If the algorithm does not converge before it reaches its maximum number of iterations, it will give a warning in R.

Plotting the results in a scatterplot

You can use the package **ggplot2** to plot the results of the clustering with colours distinguishing the clusters. This package will need to be installed.

If you have **2 dimensions**:

```
library(ggplot2)
ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=k$clust, size=3, pch=16)
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data.

x=PC1, y=PC2 – This will be the name of your two dimensions if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of the dimensions in your data e.g. x=height, y=weight.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you can try fine-tuning this number if the result looks bad.

color=k$clust – This makes it so that your points are coloured based on what cluster they're in.

size=3 – This determines the size of your points. If the points are too big, it covers too much area and makes it impossible to tell where the point's position actually is. If it is too small, it will be hard to see. 3 is a good balance, though if you have a lot of points, you may wish to try a smaller number.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to k$clust. However, this may be excessively distracting if your points are already differently coloured.

If you have **more than 2 dimensions:**

You will need to do multiple graphs because a 2D graph can only plot 2 dimensions at once (the **rgl** package can plot 3D graphs but this is not useful for papers). The number of graphs you need will be equal to the number of ways your dimensions can be paired. You can find out how many dimensions you need can by typing choose(x,2) into R, where x is the number of dimensions.

The **gridExtra** package can be used to display multiple graphs on the same page. The code below assumes you have 3 dimensions. Each line which begins with pc is a new graph, the last lines joins all the graphs together.

```
library(gridExtra)
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=k$clust, size=3, pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=k$clust, size=3, pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=k$clust, size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, ncol=2)
```

If you have 4 dimensions:

```
pc12  <-  ggplot(comp,  aes(x=PC1,  y=PC2))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc13  <-  ggplot(comp,  aes(x=PC1,  y=PC3))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc23  <-  ggplot(comp,  aes(x=PC2,  y=PC3))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc14  <-  ggplot(comp,  aes(x=PC1,  y=PC4))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc24  <-  ggplot(comp,  aes(x=PC2,  y=PC4))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc34  <-  ggplot(comp,  aes(x=PC3,  y=PC4))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, ncol=3)
```

If you have 5 dimensions:

```
pc12  <-  ggplot(comp,  aes(x=PC1,  y=PC2))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc13  <-  ggplot(comp,  aes(x=PC1,  y=PC3))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc23  <-  ggplot(comp,  aes(x=PC2,  y=PC3))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc14  <-  ggplot(comp,  aes(x=PC1,  y=PC4))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc24  <-  ggplot(comp,  aes(x=PC2,  y=PC4))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc34  <-  ggplot(comp,  aes(x=PC3,  y=PC4))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc15  <-  ggplot(comp,  aes(x=PC1,  y=PC5))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc25  <-  ggplot(comp,  aes(x=PC2,  y=PC5))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc35  <-  ggplot(comp,  aes(x=PC3,  y=PC5))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc45  <-  ggplot(comp,  aes(x=PC4,  y=PC5))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, pc15, pc25, pc35, pc45, ncol=3)
```

If you need to add even more dimensions, study what was added to the code (new parts in bold) and follow the pattern. Do not forget to add more objects to the grid.arrange list in the last line.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

x=PC1, y=PC2 – This is present in a slightly altered form in each line. It should be the name of the two dimensions to be plotted for that graph. Replace it with the name of the dimensions in your data e.g. x=height, y=weight for one line, then x=height, y=age for the next line etc.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you may try fine-tuning this number if the result looks bad.

color=k$clust – This makes it so that your points are coloured based on what cluster they're in.

size=3 – This determines the size of your points. If the points are too big, it covers too much area and makes it impossible to tell where the point's position actually is. If it is too small, it will be hard to see. 3 is a good balance, though if you have a lot of points, you may wish to try a smaller number.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to k$clust. However, this may be excessively distracting if your points are already differently coloured.

ncol=3 – This is the number of columns to arrange the graphs in. If you have more graphs you will need to increase this number, though too many graphs do not fit well on one page. You can also use nrow=3 to set the number of rows.

Seeing what's in each cluster

You can list the objects in each cluster with the following code:

```
clust <- names(sort(table(k$clust)))
clustnumb <- length(clust)
for(i in 1:clustnumb){
  print(c("Cluster number:", clust[i]))
  print(row.names(comp[k$clust==clust[i],]))
}
sort(table(k$clust))
```

It lists the clusters by order of size. Because of this, you must be careful when comparing two different clustering algorithms on the same data. The order of clusters may not be the same between the 2 algorithms.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1’s guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

Comparing clusters by each original variable

You can see how the individuals in your cluster compare by plotting boxplots for all of your original variables (before doing a PCA). The package **ggplot2** will be used again.

First, you need to add a new column to your data which can be used to separate the individuals into clusters for the boxplots.

```
NAMEOFDATA$Cluster <- k$cluster
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning). Keep the $Cluster part on the end.

Now you can compare the clusters across one variable using the following code:

```
qplot(factor(Cluster),   VARIABLE,   geom   =   "boxplot",   data   =   NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE')  +  geom_boxplot(aes(fill  =  factor(Cluster))  +
theme(legend.position="NULL")
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE – Replace this with the name of the variable. This is the column heading from your original data you would like to compare the clusters by e.g. height. Note that you need to replace it in 2 places in the code.

If you want to display multiple boxplots on one page, you can use **gridExtra** again.

```
box1 <- qplot(factor(Cluster), VARIABLE1, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE1') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

box2 <- qplot(factor(Cluster), VARIABLE2, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE2') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

box3 <- qplot(factor(Cluster), VARIABLE3, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE3') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

grid.arrange(box1, box2, box3, ncol=2)
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE1, VARIABLE2, VARIABLE3 etc. – Replace these with the names of the variables i.e. column headings from your original data you would like to compare the clusters by. Note that for each one, there are 2 places in the code to replace them.

Adding extra graphs is easy, just don't forget to add box4, box5 etc. to the grid.arrange list in the last line.

Section 2.2

Doing K-Means (Hartigan's algorithm)

Use the following code in R:

```
k <- kmeans(comp, x, algorithm='Hartigan-Wong', iter.max=1000)
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

x – Replace this with the number of clusters to find e.g. 2, 3. You can find out how many clusters is the optimal number by following the guide in Chapter 3.

Other parameters:

k – This is the name of the object that the results of the clustering are going to be stored in. It should be left as k because the follow up code (for plotting, comparing etc.) assumes it is called k.

algorithm='Hartigan' – Hartigan is the name of one algorithm that does K-Means. Refer to section 2 of Chapter 2 to learn about other algorithms for K-Means.

iter.max=1000 – This is the maximum number of iterations the algorithm is allowed to use before it is forcefully stopped, whether an optimum is reached or not. As such, it should be set to a large number like 1000 to make sure the algorithm finishes, though only a few dozen should usually be enough. If the algorithm does not converge before it reaches its maximum number of iterations, it will give a warning in R.

Plotting the results in a scatterplot

You can use the package **ggplot2** to plot the results of the clustering with colours distinguishing the clusters. This package will need to be installed.

If you have **2 dimensions**:

```
library(ggplot2)
ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=k$clust, size=3,
pch=16)
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data.

x=PC1, y=PC2 – This will be the name of your two dimensions if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of the dimensions in your data e.g. x=height, y=weight.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you can try fine-tuning this number if the result looks bad.

color=k$clust – This makes it so that your points are coloured based on what cluster they're in.

size=3 – This determines the size of your points. If the points are too big, it covers too much area and makes it impossible to tell where the point's position actually is. If it is too small, it will be hard to see. 3 is a good balance, though if you have a lot of points, you may wish to try a smaller number.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to k$clust. However, this may be excessively distracting if your points are already differently coloured.

If you have **more than 2 dimensions:**

You will need to do multiple graphs because a 2D graph can only plot 2 dimensions at once (the **rgl** package can plot 3D graphs but this is not useful for papers). The number of graphs you need will be equal to the number of ways your dimensions can be paired. You can find out how many dimensions you need can by typing choose(x,2) into R, where x is the number of dimensions.

The **gridExtra** package can be used to display multiple graphs on the same page. The code below assumes you have 3 dimensions. Each line which begins with pc is a new graph, the last lines joins all the graphs together.

```
library(gridExtra)
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, ncol=2)
```

If you have 4 dimensions:

```
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)
pc14 <- ggplot(comp, aes(x=PC1, y=PC4)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)
pc24 <- ggplot(comp, aes(x=PC2, y=PC4)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)
pc34 <- ggplot(comp, aes(x=PC3, y=PC4)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, ncol=3)
```

If you have 5 dimensions:

```
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)
pc14 <- ggplot(comp, aes(x=PC1, y=PC4)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)
pc24 <- ggplot(comp, aes(x=PC2, y=PC4)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)
pc34 <- ggplot(comp, aes(x=PC3, y=PC4)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)
pc15 <- ggplot(comp, aes(x=PC1, y=PC5)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)
pc25 <- ggplot(comp, aes(x=PC2, y=PC5)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)
pc35 <- ggplot(comp, aes(x=PC3, y=PC5)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)
pc45 <- ggplot(comp, aes(x=PC4, y=PC5)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, pc15, pc25, pc35, pc45, ncol=3)
```

If you need to add even more dimensions, study what was added to the code (new parts in bold) and follow the pattern. Do not forget to add more objects to the grid.arrange list in the last line.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

x=PC1, y=PC2 – This is present in a slightly altered form in each line. It should be the name of the two dimensions to be plotted for that graph. Replace it with the name of the dimensions in your data e.g. x=height, y=weight for one line, then x=height, y=age for the next line etc.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you may try fine-tuning this number if the result looks bad.

color=k$clust – This makes it so that your points are coloured based on what cluster they're in.

size=3 – This determines the size of your points. If the points are too big, it covers too much area and makes it impossible to tell where the point's position actually is. If it is too small, it will be hard to see. 3 is a good balance, though if you have a lot of points, you may wish to try a smaller number.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to k$clust. However, this may be excessively distracting if your points are already differently coloured.

ncol=3 – This is the number of columns to arrange the graphs in. If you have more graphs you will need to increase this number, though too many graphs do not fit well on one page. You can also use nrow=3 to set the number of rows.

Seeing what's in each cluster

You can list the objects in each cluster with the following code:

```
clust <- names(sort(table(k$clust)))
clustnumb <- length(clust)
for(i in 1:clustnumb){
  print(c("Cluster number:", clust[i]))
  print(row.names(comp[k$clust==clust[i],]))
}
sort(table(k$clust))
```

It lists the clusters by order of size. Because of this, you must be careful when comparing two different clustering algorithms on the same data. The order of clusters may not be the same between the 2 algorithms.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

Comparing clusters by each original variable

You can see how the individuals in your cluster compare by plotting boxplots for all of your original variables (before doing a PCA). The package **ggplot2** will be used again.

First, you need to add a new column to your data which can be used to separate the individuals into clusters for the boxplots.

```
NAMEOFDATA$Cluster <- k$cluster
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning). Keep the $Cluster part on the end.

Now you can compare the clusters across one variable using the following code:

```
qplot(factor(Cluster),   VARIABLE,   geom   =   "boxplot",   data   =   NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE')  +  geom_boxplot(aes(fill  =  factor(Cluster))  +
theme(legend.position="NULL")
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE – Replace this with the name of the variable. This is the column heading from your original data you would like to compare the clusters by e.g. height. Note that you need to replace it in 2 places in the code.

If you want to display multiple boxplots on one page, you can use **gridExtra** again.

```
box1  <-  qplot(factor(Cluster),  VARIABLE1,  geom  =  "boxplot",  data  =  NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE1')  +  geom_boxplot(aes(fill  =  factor(Cluster)))  +
theme(legend.position="NULL")

box2  <-  qplot(factor(Cluster),  VARIABLE2,  geom  =  "boxplot",  data  =  NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE2')  +  geom_boxplot(aes(fill  =  factor(Cluster)))  +
theme(legend.position="NULL")

box3  <-  qplot(factor(Cluster),  VARIABLE3,  geom  =  "boxplot",  data  =  NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE3')  +  geom_boxplot(aes(fill  =  factor(Cluster)))  +
theme(legend.position="NULL")

grid.arrange(box1, box2, box3, ncol=2)
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE1, VARIABLE2, VARIABLE3 etc. – Replace these with the names of the variables i.e. column headings from your original data you would like to compare the clusters by. Note that for each one, there are 2 places in the code to replace them.

Adding extra graphs is easy, just don’t forget to add box4, box5 etc. to the grid.arrange list in the last line.

Section 3.1

Doing K-Means (Multiple starts)

Use the following code in R:

```
k <- kmeans(comp, x, algorithm='Hartigan-Wong', iter.max=1000, nstart=25)
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

x – Replace this with the number of clusters to find e.g. 2, 3. You can find out how many clusters is the optimal number by following the guide in Chapter 3.

nstart=25 – This repeats the algorithm 25 times and chooses the best result. If the dataset is exceptionally large, you may wish to decrease this number slightly.

Other parameters:

k – This is the name of the object that the results of the clustering are going to be stored in. It should be left as k because the follow up code (for plotting, comparing etc.) assumes it is called k.

algorithm='Hartigan' – Hartigan is the name of one algorithm that does K-Means. Refer to section 2 of Chapter 2 to learn about other algorithms for K-Means.

iter.max=1000 – This is the maximum number of iterations the algorithm is allowed to use before it is forcefully stopped, whether an optimum is reached or not. As such, it should be set to a large number like 1000 to make sure the algorithm finishes, though only a few dozen should usually be enough. If the algorithm does not converge before it reaches its maximum number of iterations, it will give a warning in R.

Plotting the results in a scatterplot

You can use the package **ggplot2** to plot the results of the clustering with colours distinguishing the clusters. This package will need to be installed.

If you have **2 dimensions**:

```
library(ggplot2)
ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=k$clust, size=3,
pch=16)
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data.

x=PC1, y=PC2 – This will be the name of your two dimensions if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of the dimensions in your data e.g. x=height, y=weight.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you can try fine-tuning this number if the result looks bad.

color=k$clust – This makes it so that your points are coloured based on what cluster they're in.

size=3 – This determines the size of your points. If the points are too big, it covers too much area and makes it impossible to tell where the point's position actually is. If it is too small, it will be hard to see. 3 is a good balance, though if you have a lot of points, you may wish to try a smaller number.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to k$clust. However, this may be excessively distracting if your points are already differently coloured.

If you have **more than 2 dimensions:**

You will need to do multiple graphs because a 2D graph can only plot 2 dimensions at once (the **rgl** package can plot 3D graphs but this is not useful for papers). The number of graphs you need will be equal to the number of ways your dimensions can be paired. You can find out how many dimensions you need can by typing choose(x,2) into R, where x is the number of dimensions.

The **gridExtra** package can be used to display multiple graphs on the same page. The code below assumes you have 3 dimensions. Each line which begins with pc is a new graph, the last lines joins all the graphs together.

```
library(gridExtra)
pc12  <-  ggplot(comp,  aes(x=PC1,  y=PC2))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc13  <-  ggplot(comp,  aes(x=PC1,  y=PC3))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc23  <-  ggplot(comp,  aes(x=PC2,  y=PC3))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, ncol=2)
```

If you have 4 dimensions:

```
pc12  <-  ggplot(comp,  aes(x=PC1,  y=PC2))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc13  <-  ggplot(comp,  aes(x=PC1,  y=PC3))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc23  <-  ggplot(comp,  aes(x=PC2,  y=PC3))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc14  <-  ggplot(comp,  aes(x=PC1,  y=PC4))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc24  <-  ggplot(comp,  aes(x=PC2,  y=PC4))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc34  <-  ggplot(comp,  aes(x=PC3,  y=PC4))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, ncol=3)
```

If you have 5 dimensions:

```
pc12  <-  ggplot(comp,  aes(x=PC1,  y=PC2))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc13  <-  ggplot(comp,  aes(x=PC1,  y=PC3))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc23  <-  ggplot(comp,  aes(x=PC2,  y=PC3))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc14  <-  ggplot(comp,  aes(x=PC1,  y=PC4))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc24  <-  ggplot(comp,  aes(x=PC2,  y=PC4))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc34  <-  ggplot(comp,  aes(x=PC3,  y=PC4))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc15  <-  ggplot(comp,  aes(x=PC1,  y=PC5))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc25  <-  ggplot(comp,  aes(x=PC2,  y=PC5))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc35  <-  ggplot(comp,  aes(x=PC3,  y=PC5))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc45  <-  ggplot(comp,  aes(x=PC4,  y=PC5))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, pc15, pc25, pc35, pc45, ncol=3)
```

If you need to add even more dimensions, study what was added to the code (new parts in bold) and follow the pattern. Do not forget to add more objects to the grid.arrange list in the last line.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

x=PC1, y=PC2 – This is present in a slightly altered form in each line. It should be the name of the two dimensions to be plotted for that graph. Replace it with the name of the dimensions in your data e.g. x=height, y=weight for one line, then x=height, y=age for the next line etc.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you may try fine-tuning this number if the result looks bad.

color=k$clust – This makes it so that your points are coloured based on what cluster they're in.

size=3 – This determines the size of your points. If the points are too big, it covers too much area and makes it impossible to tell where the point's position actually is. If it is too small, it will be hard to see. 3 is a good balance, though if you have a lot of points, you may wish to try a smaller number.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to k$clust. However, this may be excessively distracting if your points are already differently coloured.

ncol=3 – This is the number of columns to arrange the graphs in. If you have more graphs you will need to increase this number, though too many graphs do not fit well on one page. You can also use nrow=3 to set the number of rows.

Seeing what's in each cluster

You can list the objects in each cluster with the following code:

```
clust <- names(sort(table(k$clust)))
clustnumb <- length(clust)
for(i in 1:clustnumb){
  print(c("Cluster number:", clust[i]))
  print(row.names(comp[k$clust==clust[i],]))
}
sort(table(k$clust))
```

It lists the clusters by order of size. Because of this, you must be careful when comparing two different clustering algorithms on the same data. The order of clusters may not be the same between the 2 algorithms.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

Comparing clusters by each original variable

You can see how the individuals in your cluster compare by plotting boxplots for all of your original variables (before doing a PCA). The package **ggplot2** will be used again.

First, you need to add a new column to your data which can be used to separate the individuals into clusters for the boxplots.

```
NAMEOFDATA$Cluster <- k$cluster
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning). Keep the $Cluster part on the end.

Now you can compare the clusters across one variable using the following code:

```
qplot(factor(Cluster), VARIABLE, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE – Replace this with the name of the variable. This is the column heading from your original data you would like to compare the clusters by e.g. height. Note that you need to replace it in 2 places in the code.

If you want to display multiple boxplots on one page, you can use **gridExtra** again.

```
box1 <- qplot(factor(Cluster), VARIABLE1, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE1') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

box2 <- qplot(factor(Cluster), VARIABLE2, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE2') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

box3 <- qplot(factor(Cluster), VARIABLE3, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE3') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

grid.arrange(box1, box2, box3, ncol=2)
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE1, VARIABLE2, VARIABLE3 etc. – Replace these with the names of the variables i.e. column headings from your original data you would like to compare the clusters by. Note that for each one, there are 2 places in the code to replace them.

Adding extra graphs is easy, just don't forget to add box4, box5 etc. to the grid.arrange list in the last line.

Section 3.2

Doing K-Means++

The package **LICORS** needs to be installed. Then use the following code in R:

```
library(LICORS)
k <- kmeanspp(comp, k = x, start = "random", iter.max = 1000, nstart = 25)
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1’s guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

k = x – Replace x with the number of clusters to find e.g. 2, 3. You can find out how many clusters is the optimal number by following the guide in Chapter 3.

nstart=25 – This repeats the algorithm 25 times and chooses the best result. If the dataset is exceptionally large, you may wish to decrease this number slightly.

Other parameters:

k (first one) – This is the name of the object that the results of the clustering are going to be stored in. It should be left as k because the follow up code (for plotting, comparing etc.) assumes it is called k.

start=”random” – K-Means++ has “non-random” initialisation, but the first centroid is still randomly chosen. You can, however, set the first centroid with the start parameter (this is unnecessary).

iter.max=1000 – This is the maximum number of iterations the algorithm is allowed to use before it is forcefully stopped, whether an optimum is reached or not. As such, it should be set to a large number like 1000 to make sure the algorithm finishes, though only a few dozen should be enough. If the algorithm does not converge before it reaches its maximum number of iterations, it will give a warning in R.

Plotting the results in a scatterplot

You can use the package **ggplot2** to plot the results of the clustering with colours distinguishing the clusters. This package will need to be installed.

If you have **2 dimensions**:

```
library(ggplot2)
ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=k$clust, size=3, pch=16)
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data.

x=PC1, y=PC2 – This will be the name of your two dimensions if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of the dimensions in your data e.g. x=height, y=weight.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you can try fine-tuning this number if the result looks bad.

color=k$clust – This makes it so that your points are coloured based on what cluster they're in.

size=3 – This determines the size of your points. If the points are too big, it covers too much area and makes it impossible to tell where the point's position actually is. If it is too small, it will be hard to see. 3 is a good balance, though if you have a lot of points, you may wish to try a smaller number.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to k$clust. However, this may be excessively distracting if your points are already differently coloured.

If you have **more than 2 dimensions:**

You will need to do multiple graphs because a 2D graph can only plot 2 dimensions at once (the **rgl** package can plot 3D graphs but this is not useful for papers). The number of graphs you need will be equal to the number of ways your dimensions can be paired. You can find out how many dimensions you need can by typing choose(x,2) into R, where x is the number of dimensions.

The **gridExtra** package can be used to display multiple graphs on the same page. The code below assumes you have 3 dimensions. Each line which begins with pc is a new graph, the last lines joins all the graphs together.

```
library(gridExtra)
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, ncol=2)
```

If you have 4 dimensions:

```
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=k$clust, size=3, pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=k$clust, size=3, pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=k$clust, size=3, pch = 16)
pc14 <- ggplot(comp, aes(x=PC1, y=PC4)) + geom_point(alpha=.7, color=k$clust, size=3, pch = 16)
pc24 <- ggplot(comp, aes(x=PC2, y=PC4)) + geom_point(alpha=.7, color=k$clust, size=3, pch = 16)
pc34 <- ggplot(comp, aes(x=PC3, y=PC4)) + geom_point(alpha=.7, color=k$clust, size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, ncol=3)
```

If you have 5 dimensions:

```
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=k$clust, size=3, pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=k$clust, size=3, pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=k$clust, size=3, pch = 16)
pc14 <- ggplot(comp, aes(x=PC1, y=PC4)) + geom_point(alpha=.7, color=k$clust, size=3, pch = 16)
pc24 <- ggplot(comp, aes(x=PC2, y=PC4)) + geom_point(alpha=.7, color=k$clust, size=3, pch = 16)
pc34 <- ggplot(comp, aes(x=PC3, y=PC4)) + geom_point(alpha=.7, color=k$clust, size=3, pch = 16)
pc15 <- ggplot(comp, aes(x=PC1, y=PC5)) + geom_point(alpha=.7, color=k$clust, size=3, pch = 16)
pc25 <- ggplot(comp, aes(x=PC2, y=PC5)) + geom_point(alpha=.7, color=k$clust, size=3, pch = 16)
pc35 <- ggplot(comp, aes(x=PC3, y=PC5)) + geom_point(alpha=.7, color=k$clust, size=3, pch = 16)
pc45 <- ggplot(comp, aes(x=PC4, y=PC5)) + geom_point(alpha=.7, color=k$clust, size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, pc15, pc25, pc35, pc45, ncol=3)
```

If you need to add even more dimensions, study what was added to the code (new parts in bold) and follow the pattern. Do not forget to add more objects to the grid.arrange list in the last line.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

x=PC1, y=PC2 – This is present in a slightly altered form in each line. It should be the name of the two dimensions to be plotted for that graph. Replace it with the name of the dimensions in your data e.g. x=height, y=weight for one line, then x=height, y=age for the next line etc.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you may try fine-tuning this number if the result looks bad.

color=k$clust – This makes it so that your points are coloured based on what cluster they're in.

size=3 – This determines the size of your points. If the points are too big, it covers too much area and makes it impossible to tell where the point's position actually is. If it is too small, it will be hard to see. 3 is a good balance, though if you have a lot of points, you may wish to try a smaller number.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to k$clust. However, this may be excessively distracting if your points are already differently coloured.

ncol=3 – This is the number of columns to arrange the graphs in. If you have more graphs you will need to increase this number, though too many graphs do not fit well on one page. You can also use nrow=3 to set the number of rows.

Seeing what's in each cluster

You can list the objects in each cluster with the following code:

```
clust <- names(sort(table(k$clust)))
clustnumb <- length(clust)
for(i in 1:clustnumb){
  print(c("Cluster number:", clust[i]))
  print(row.names(comp[k$clust==clust[i],]))
}
sort(table(k$clust))
```

It lists the clusters by order of size. Because of this, you must be careful when comparing two different clustering algorithms on the same data. The order of clusters may not be the same between the 2 algorithms.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

Comparing clusters by each original variable

You can see how the individuals in your cluster compare by plotting boxplots for all of your original variables (before doing a PCA). The package **ggplot2** will be used again.

First, you need to add a new column to your data which can be used to separate the individuals into clusters for the boxplots.

```
NAMEOFDATA$Cluster <- k$cluster
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning). Keep the $Cluster part on the end.

Now you can compare the clusters across one variable using the following code:

```
qplot(factor(Cluster),   VARIABLE,   geom   =   "boxplot",   data   =   NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE')  +  geom_boxplot(aes(fill  =  factor(Cluster)))  +
theme(legend.position="NULL")
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE – Replace this with the name of the variable. This is the column heading from your original data you would like to compare the clusters by e.g. height. Note that you need to replace it in 2 places in the code.

If you want to display multiple boxplots on one page, you can use **gridExtra** again.

```
box1 <- qplot(factor(Cluster), VARIABLE1, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE1') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

box2 <- qplot(factor(Cluster), VARIABLE2, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE2') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

box3 <- qplot(factor(Cluster), VARIABLE3, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE3') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

grid.arrange(box1, box2, box3, ncol=2)
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE1, VARIABLE2, VARIABLE3 etc. – Replace these with the names of the variables i.e. column headings from your original data you would like to compare the clusters by. Note that for each one, there are 2 places in the code to replace them.

Adding extra graphs is easy, just don’t forget to add box4, box5 etc. to the grid.arrange list in the last line.

Section 4.1

Doing K-Medians

The package **flexclust** needs to be installed. Then use the following code in R:

```
library(flexclust)
flexclust::kcca -> kcca
k <- kcca(comp, k = x, family=kccaFamily("kmedians"), save.data = TRUE)
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

k = x – Replace x with the number of clusters to find e.g. 2, 3. You can find out how many clusters is the optimal number by following the guide in Chapter 3.

Other parameters:

k (first one) – This is the name of the object that the results of the clustering are going to be stored in. It should be left as k because the follow up code (for plotting, comparing etc.) assumes it is called k.

family=kccaFamily("kmedians") – This selects medians as the centres.

Unfortunately you cannot set the nstart or iter.max with this package.

Plotting the results in a scatterplot

You can use the package **ggplot2** to plot the results of the clustering with colours distinguishing the clusters. This package will need to be installed.

If you have **2 dimensions**:

```
library(ggplot2)
ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=clusters(k), size=3,
pch=16)
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data.

x=PC1, y=PC2 – This will be the name of your two dimensions if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of the dimensions in your data e.g. x=height, y=weight.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you can try fine-tuning this number if the result looks bad.

color=clusters(k) – This makes it so that your points are coloured based on what cluster they're in.

size=3 – This determines the size of your points. If the points are too big, it covers too much area and makes it impossible to tell where the point's position actually is. If it is too small, it will be hard to see. 3 is a good balance, though if you have a lot of points, you may wish to try a smaller number.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to clusters(k). However, this may be excessively distracting if your points are already differently coloured.

If you have **more than 2 dimensions:**

You will need to do multiple graphs because a 2D graph can only plot 2 dimensions at once (the **rgl** package can plot 3D graphs but this is not useful for papers). The number of graphs you need will be equal to the number of ways your dimensions can be paired. You can find out how many dimensions you need can by typing choose(x,2) into R, where x is the number of dimensions.

The **gridExtra** package can be used to display multiple graphs on the same page. The code below assumes you have 3 dimensions. Each line which begins with pc is a new graph, the last lines joins all the graphs together.

```
library(gridExtra)
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color= clusters(k),
size=3, pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color= clusters(k),
size=3, pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color= clusters(k),
size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, ncol=2)
```

If you have 4 dimensions:

```r
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc14 <- ggplot(comp, aes(x=PC1, y=PC4)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc24 <- ggplot(comp, aes(x=PC2, y=PC4)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc34 <- ggplot(comp, aes(x=PC3, y=PC4)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, ncol=3)
```

If you have 5 dimensions:

```r
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc14 <- ggplot(comp, aes(x=PC1, y=PC4)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc24 <- ggplot(comp, aes(x=PC2, y=PC4)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc34 <- ggplot(comp, aes(x=PC3, y=PC4)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc15 <- ggplot(comp, aes(x=PC1, y=PC5)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc25 <- ggplot(comp, aes(x=PC2, y=PC5)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc35 <- ggplot(comp, aes(x=PC3, y=PC5)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc45 <- ggplot(comp, aes(x=PC4, y=PC5)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, pc15, pc25, pc35, pc45, ncol=3)
```

If you need to add even more dimensions, study what was added to the code (new parts in bold) and follow the pattern. Do not forget to add more objects to the grid.arrange list in the last line.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

x=PC1, y=PC2 – This is present in a slightly altered form in each line. It should be the name of the two dimensions to be plotted for that graph. Replace it with the name of the dimensions in your data e.g. x=height, y=weight for one line, then x=height, y=age for the next line etc.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you may try fine-tuning this number if the result looks bad.

color= clusters(k) – This makes it so that your points are coloured based on what cluster they're in.

size=3 – This determines the size of your points. If the points are too big, it covers too much area and makes it impossible to tell where the point's position actually is. If it is too small, it will be hard to see. 3 is a good balance, though if you have a lot of points, you may wish to try a smaller number.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to clusters(k). However, this may be excessively distracting if your points are already differently coloured.

ncol=3 – This is the number of columns to arrange the graphs in. If you have more graphs you will need to increase this number, though too many graphs do not fit well on one page. You can also use nrow=3 to set the number of rows.

Seeing what's in each cluster

You can list the objects in each cluster with the following code:

```
clust <- names(sort(table(clusters(k))))
clustnumb <- length(clust)
for(i in 1:clustnumb){
  print(c("Cluster number:", clust[i]))
  print(row.names(comp[(clusters(k))==clust[i],]))
}
sort(table(clusters(k)))
```

It lists the clusters by order of size. Because of this, you must be careful when comparing two different clustering algorithms on the same data. The order of clusters may not be the same between the 2 algorithms.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

Comparing clusters by each original variable

You can see how the individuals in your cluster compare by plotting boxplots for all of your original variables (before doing a PCA). The package **ggplot2** will be used again.

First, you need to add a new column to your data which can be used to separate the individuals into clusters for the boxplots.

```
NAMEOFDATA$Cluster <- clusters(k)
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning). Keep the $Cluster part on the end.

Now you can compare the clusters across one variable using the following code:

```
qplot(factor(Cluster),   VARIABLE,   geom   =   "boxplot",   data   =   NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE')  +  geom_boxplot(aes(fill  =  factor(Cluster))  +
theme(legend.position="NULL")
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE – Replace this with the name of the variable. This is the column heading from your original data you would like to compare the clusters by e.g. height. Note that you need to replace it in 2 places in the code.

If you want to display multiple boxplots on one page, you can use **gridExtra** again.

```
box1  <-  qplot(factor(Cluster),  VARIABLE1,  geom  =  "boxplot",  data  =  NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE1')  +  geom_boxplot(aes(fill  =  factor(Cluster)))  +
theme(legend.position="NULL")

box2  <-  qplot(factor(Cluster),  VARIABLE2,  geom  =  "boxplot",  data  =  NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE2')  +  geom_boxplot(aes(fill  =  factor(Cluster)))  +
theme(legend.position="NULL")

box3  <-  qplot(factor(Cluster),  VARIABLE3,  geom  =  "boxplot",  data  =  NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE3')  +  geom_boxplot(aes(fill  =  factor(Cluster)))  +
theme(legend.position="NULL")

grid.arrange(box1, box2, box3, ncol=2)
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE1, VARIABLE2, VARIABLE3 etc. – Replace these with the names of the variables i.e. column headings from your original data you would like to compare the clusters by. Note that for each one, there are 2 places in the code to replace them.

Adding extra graphs is easy, just don't forget to add box4, box5 etc. to the grid.arrange list in the last line.

Section 4.2

Doing K-Medians++

The package **flexclust** needs to be installed. Then use the following code in R:

```
library(flexclust)
flexclust::kcca -> kcca
k  <-  kcca(comp,  k  =  x,  family=kccaFamily("kmedians"),  save.data  =  TRUE,
control=list(initcent="kmeanspp"))
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

k = x – Replace x with the number of clusters to find e.g. 2, 3. You can find out how many clusters is the optimal number by following the guide in Chapter 3.

Other parameters:

k (first one) – This is the name of the object that the results of the clustering are going to be stored in. It should be left as k because the follow up code (for plotting, comparing etc.) assumes it is called k.

family=kccaFamily("kmedians") – This selects medians as the centres.

initcent="kmeanspp" – This activates ++ initialisation

Unfortunately you cannot set the nstart or iter.max with this package.

Plotting the results in a scatterplot

You can use the package **ggplot2** to plot the results of the clustering with colours distinguishing the clusters. This package will need to be installed.

If you have **2 dimensions**:

```
library(ggplot2)
ggplot(comp,  aes(x=PC1,  y=PC2))  +  geom_point(alpha=.7,  color=clusters(k),  size=3,
pch=16)
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data.

x=PC1, y=PC2 – This will be the name of your two dimensions if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of the dimensions in your data e.g. x=height, y=weight.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you can try fine-tuning this number if the result looks bad.

color=clusters(k) – This makes it so that your points are coloured based on what cluster they're in.

size=3 – This determines the size of your points. If the points are too big, it covers too much area and makes it impossible to tell where the point's position actually is. If it is too small, it will be hard to see. 3 is a good balance, though if you have a lot of points, you may wish to try a smaller number.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to clusters(k). However, this may be excessively distracting if your points are already differently coloured.

If you have **more than 2 dimensions:**

You will need to do multiple graphs because a 2D graph can only plot 2 dimensions at once (the **rgl** package can plot 3D graphs but this is not useful for papers). The number of graphs you need will be equal to the number of ways your dimensions can be paired. You can find out how many dimensions you need can by typing choose(x,2) into R, where x is the number of dimensions.

The **gridExtra** package can be used to display multiple graphs on the same page. The code below assumes you have 3 dimensions. Each line which begins with pc is a new graph, the last lines joins all the graphs together.

```
library(gridExtra)
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color= clusters(k),
size=3, pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color= clusters(k),
size=3, pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color= clusters(k),
size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, ncol=2)
```

If you have 4 dimensions:

```
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc14 <- ggplot(comp, aes(x=PC1, y=PC4)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc24 <- ggplot(comp, aes(x=PC2, y=PC4)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc34 <- ggplot(comp, aes(x=PC3, y=PC4)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, ncol=3)
```

If you have 5 dimensions:

```
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc14 <- ggplot(comp, aes(x=PC1, y=PC4)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc24 <- ggplot(comp, aes(x=PC2, y=PC4)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc34 <- ggplot(comp, aes(x=PC3, y=PC4)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc15 <- ggplot(comp, aes(x=PC1, y=PC5)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc25 <- ggplot(comp, aes(x=PC2, y=PC5)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc35 <- ggplot(comp, aes(x=PC3, y=PC5)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)
pc45 <- ggplot(comp, aes(x=PC4, y=PC5)) + geom_point(alpha=.7, color=clusters(k),
size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, pc15, pc25, pc35, pc45, ncol=3)
```

If you need to add even more dimensions, study what was added to the code (new parts in bold) and follow the pattern. Do not forget to add more objects to the grid.arrange list in the last line.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

x=PC1, y=PC2 – This is present in a slightly altered form in each line. It should be the name of the two dimensions to be plotted for that graph. Replace it with the name of the dimensions in your data e.g. x=height, y=weight for one line, then x=height, y=age for the next line etc.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you may try fine-tuning this number if the result looks bad.

color= clusters(k) – This makes it so that your points are coloured based on what cluster they're in.

size=3 – This determines the size of your points. If the points are too big, it covers too much area and makes it impossible to tell where the point's position actually is. If it is too small, it will be hard to see. 3 is a good balance, though if you have a lot of points, you may wish to try a smaller number.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to clusters(k). However, this may be excessively distracting if your points are already differently coloured.

ncol=3 – This is the number of columns to arrange the graphs in. If you have more graphs you will need to increase this number, though too many graphs do not fit well on one page. You can also use nrow=3 to set the number of rows.

Seeing what's in each cluster

You can list the objects in each cluster with the following code:

```
clust <- names(sort(table(clusters(k))))
clustnumb <- length(clust)
for(i in 1:clustnumb){
  print(c("Cluster number:", clust[i]))
  print(row.names(comp[(clusters(k))==clust[i],]))
}
sort(table(clusters(k)))
```

It lists the clusters by order of size. Because of this, you must be careful when comparing two different clustering algorithms on the same data. The order of clusters may not be the same between the 2 algorithms.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

Comparing clusters by each original variable

You can see how the individuals in your cluster compare by plotting boxplots for all of your original variables (before doing a PCA). The package **ggplot2** will be used again.

First, you need to add a new column to your data which can be used to separate the individuals into clusters for the boxplots.

```
NAMEOFDATA$Cluster <- clusters(k)
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning). Keep the $Cluster part on the end.

Now you can compare the clusters across one variable using the following code:

```
qplot(factor(Cluster),   VARIABLE,   geom   =   "boxplot",   data   =   NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE')  +  geom_boxplot(aes(fill  =  factor(Cluster))  +
theme(legend.position="NULL")
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE – Replace this with the name of the variable. This is the column heading from your original data you would like to compare the clusters by e.g. height. Note that you need to replace it in 2 places in the code.

If you want to display multiple boxplots on one page, you can use **gridExtra** again.

```
box1  <-  qplot(factor(Cluster),  VARIABLE1,  geom  =  "boxplot",  data  =  NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE1')  +  geom_boxplot(aes(fill  =  factor(Cluster)))  +
theme(legend.position="NULL")

box2  <-  qplot(factor(Cluster),  VARIABLE2,  geom  =  "boxplot",  data  =  NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE2')  +  geom_boxplot(aes(fill  =  factor(Cluster)))  +
theme(legend.position="NULL")

box3  <-  qplot(factor(Cluster),  VARIABLE3,  geom  =  "boxplot",  data  =  NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE3')  +  geom_boxplot(aes(fill  =  factor(Cluster)))  +
theme(legend.position="NULL")

grid.arrange(box1, box2, box3, ncol=2)
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE1, VARIABLE2, VARIABLE3 etc. – Replace these with the names of the variables i.e. column headings from your original data you would like to compare the clusters by. Note that for each one, there are 2 places in the code to replace them.

Adding extra graphs is easy, just don't forget to add box4, box5 etc. to the grid.arrange list in the last line.

Section 4.3

Doing K-Medoids

The package **cluster** needs to be installed. Then use the following code in R:

```
library(cluster)
k <- pam(comp, x, metric = "manhattan", stand = FALSE, medoids=NULL)
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

x – Replace with the number of clusters to find e.g. 2, 3. You can find out how many clusters is the optimal number by following the guide in Chapter 3.

Other parameters:

k (first one) – This is the name of the object that the results of the clustering are going to be stored in. It should be left as k because the follow up code (for plotting, comparing etc.) assumes it is called k.

metric="manhattan" – K-Medoids was designed with manhattan distances as the distance metric, however it can be changed to euclidean. If your data is a dissimilarity matrix then this parameter is ignored.

medoids=NULL – You can specify the initial medoids with a vector instead of NULL, though this is unnecessary.

Unfortunately you cannot set the nstart or iter.max with this package.

Plotting the results in a scatterplot

You can use the package **ggplot2** to plot the results of the clustering with colours distinguishing the clusters. This package will need to be installed.

If you have **2 dimensions**:

```
library(ggplot2)
ggplot(comp,  aes(x=PC1,  y=PC2))  +  geom_point(alpha=.7,  color=k$clust,  size=3,
pch=16)
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data.

x=PC1, y=PC2 – This will be the name of your two dimensions if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of the dimensions in your data e.g. x=height, y=weight.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you can try fine-tuning this number if the result looks bad.

color=k$clust – This makes it so that your points are coloured based on what cluster they're in.

size=3 – This determines the size of your points. If the points are too big, it covers too much area and makes it impossible to tell where the point's position actually is. If it is too small, it will be hard to see. 3 is a good balance, though if you have a lot of points, you may wish to try a smaller number.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to k$clust. However, this may be excessively distracting if your points are already differently coloured.

If you have **more than 2 dimensions:**

You will need to do multiple graphs because a 2D graph can only plot 2 dimensions at once (the **rgl** package can plot 3D graphs but this is not useful for papers). The number of graphs you need will be equal to the number of ways your dimensions can be paired. You can find out how many dimensions you need can by typing choose(x,2) into R, where x is the number of dimensions.

The **gridExtra** package can be used to display multiple graphs on the same page. The code below assumes you have 3 dimensions. Each line which begins with pc is a new graph, the last lines joins all the graphs together.

```
library(gridExtra)
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=k$clust,
size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, ncol=2)
```

If you have 4 dimensions:

```
pc12  <-  ggplot(comp,  aes(x=PC1,  y=PC2))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc13  <-  ggplot(comp,  aes(x=PC1,  y=PC3))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc23  <-  ggplot(comp,  aes(x=PC2,  y=PC3))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc14  <-  ggplot(comp,  aes(x=PC1,  y=PC4))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc24  <-  ggplot(comp,  aes(x=PC2,  y=PC4))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc34  <-  ggplot(comp,  aes(x=PC3,  y=PC4))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, ncol=3)
```

If you have 5 dimensions:

```
pc12  <-  ggplot(comp,  aes(x=PC1,  y=PC2))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc13  <-  ggplot(comp,  aes(x=PC1,  y=PC3))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc23  <-  ggplot(comp,  aes(x=PC2,  y=PC3))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc14  <-  ggplot(comp,  aes(x=PC1,  y=PC4))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc24  <-  ggplot(comp,  aes(x=PC2,  y=PC4))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc34  <-  ggplot(comp,  aes(x=PC3,  y=PC4))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc15  <-  ggplot(comp,  aes(x=PC1,  y=PC5))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc25  <-  ggplot(comp,  aes(x=PC2,  y=PC5))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc35  <-  ggplot(comp,  aes(x=PC3,  y=PC5))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)
pc45  <-  ggplot(comp,  aes(x=PC4,  y=PC5))  +  geom_point(alpha=.7,  color=k$clust,
size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, pc15, pc25, pc35, pc45, ncol=3)
```

If you need to add even more dimensions, study what was added to the code (new parts in bold) and follow the pattern. Do not forget to add more objects to the grid.arrange list in the last line.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

x=PC1, y=PC2 – This is present in a slightly altered form in each line. It should be the name of the two dimensions to be plotted for that graph. Replace it with the name of the dimensions in your data e.g. x=height, y=weight for one line, then x=height, y=age for the next line etc.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you may try fine-tuning this number if the result looks bad.

color=k$clust – This makes it so that your points are coloured based on what cluster they're in.

size=3 – This determines the size of your points. If the points are too big, it covers too much area and makes it impossible to tell where the point's position actually is. If it is too small, it will be hard to see. 3 is a good balance, though if you have a lot of points, you may wish to try a smaller number.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to k$clust. However, this may be excessively distracting if your points are already differently coloured.

ncol=3 – This is the number of columns to arrange the graphs in. If you have more graphs you will need to increase this number, though too many graphs do not fit well on one page. You can also use nrow=3 to set the number of rows.

Seeing what's in each cluster

You can list the objects in each cluster with the following code:

```
clust <- names(sort(table(k$clust)))
clustnumb <- length(clust)
for(i in 1:clustnumb){
  print(c("Cluster number:", clust[i]))
  print(row.names(comp[k$clust==clust[i],]))
}
sort(table(k$clust))
```

It lists the clusters by order of size. Because of this, you must be careful when comparing two different clustering algorithms on the same data. The order of clusters may not be the same between the 2 algorithms.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

Comparing clusters by each original variable

You can see how the individuals in your cluster compare by plotting boxplots for all of your original variables (before doing a PCA). The package **ggplot2** will be used again.

First, you need to add a new column to your data which can be used to separate the individuals into clusters for the boxplots.

```
NAMEOFDATA$Cluster <- k$cluster
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning). Keep the $Cluster part on the end.

Now you can compare the clusters across one variable using the following code:

```
qplot(factor(Cluster),  VARIABLE,  geom  =  "boxplot",  data  =  NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE')  +  geom_boxplot(aes(fill  =  factor(Cluster)))  +
theme(legend.position="NULL")
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE – Replace this with the name of the variable. This is the column heading from your original data you would like to compare the clusters by e.g. height. Note that you need to replace it in 2 places in the code.

If you want to display multiple boxplots on one page, you can use **gridExtra** again.

```
box1 <- qplot(factor(Cluster), VARIABLE1, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE1') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

box2 <- qplot(factor(Cluster), VARIABLE2, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE2') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

box3 <- qplot(factor(Cluster), VARIABLE3, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE3') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

grid.arrange(box1, box2, box3, ncol=2)
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE1, VARIABLE2, VARIABLE3 etc. – Replace these with the names of the variables i.e. column headings from your original data you would like to compare the clusters by. Note that for each one, there are 2 places in the code to replace them.

Adding extra graphs is easy, just don't forget to add box4, box5 etc. to the grid.arrange list in the last line.

Section 5.1

Doing Fuzzy C-Means

The package **vegclust** needs to be installed. Then use the following code in R:

```
library(vegclust)
k <- vegclust(x=comp, mobileCenters=x, method="FCM", nstart=25, iter.max=1000)
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

mobileCenters=x – Replace x with the number of clusters to find e.g. 2, 3. You can find out how many clusters is the optimal number by following the guide in Chapter 3.

nstart=25 – This repeats the algorithm 25 times and chooses the best result. If the dataset is exceptionally large, you may wish to decrease this number slightly.

Other parameters:

k (first one) – This is the name of the object that the results of the clustering are going to be stored in. It should be left as k because the follow up code (for plotting, comparing etc.) assumes it is called k.

method='FCM' – Sets the algorithm to Fuzzy C-Means

iter.max=1000 – This is the maximum number of iterations the algorithm is allowed to use before it is forcefully stopped, whether an optimum is reached or not. As such, it should be set to a large number like 1000 to make sure the algorithm finishes, though only a few dozen should usually be enough. If the algorithm does not converge before it reaches its maximum number of iterations, it will give a warning in R.

Plotting the results in a scatterplot

You can use the package **ggplot2** to plot the results of the clustering with colours distinguishing the clusters. This package will need to be installed.

However, if you want to show membership in the graph, you can do this by having the points be differently sized (depending on how strongly they are associated to their nearest cluster centre). You will need to assign sizes to points before plotting. First, retrieve the sizes with the following code:

```
nr <- nrow(k$memb)
nc <- ncol(k$memb)
B <- matrix(0,nrow=nr, ncol=nc)
for(i in 1:nr){
  x <- which(k$memb[i,]==max(k$memb[i,]))
  B[i,x] <- 1
}
```

Next, assign the sizes to the points. If you have 2 clusters, use the following code.

```
B[,1] + B[,2]*2 -> kmemb
kmax <- apply(k$memb[, 1:2], 1, max)
```

If you have 3 clusters, use this code instead (new parts in bold):

```
B[,1] + B[,2]*2 + B[,3]*3 -> kmemb
kmax <- apply(k$memb[, 1:3], 1, max)
```

If you have 4 clusters, use this code instead (new parts in bold):

```
B[,1] + B[,2]*2 + B[,3]*3 + B[,4]*4 -> kmemb
kmax <- apply(k$memb[, 1:4], 1, max)
```

Follow this pattern if you have more than 4 clusters.

Now that the sizes have been assigned, you can plot the results.

If you have **2 dimensions**:

```
library(ggplot2)
ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=kmemb, size=(2*kmax)^3,
pch=16)
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data.

x=PC1, y=PC2 – This will be the name of your two dimensions if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of the dimensions in your data e.g. x=height, y=weight.

size=(2*kmax)^3 – This sizes the points based on their degree of association to their nearest cluster centre. You can fine-tune the numbers. The larger the number that the 2 is changed to, the bigger all points will be. The larger the number that the 3 is changed to, the more exaggerated differences between points will be. Try different numbers to get the best look for your data.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you can try fine-tuning this number if the result looks bad.

color=kmemb – This makes it so that your points are coloured based on what cluster they're in.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to kmemb. However, this may be excessively distracting if your points are already differently coloured.

If you have **more than 2 dimensions:**

You will need to do multiple graphs because a 2D graph can only plot 2 dimensions at once (the **rgl** package can plot 3D graphs but this is not useful for papers). The number of graphs you need will be equal to the number of ways your dimensions can be paired. You can find out how many dimensions you need can by typing choose(x,2) into R, where x is the number of dimensions.

The **gridExtra** package can be used to display multiple graphs on the same page. The code below assumes you have 3 dimensions. Each line which begins with pc is a new graph, the last lines joins all the graphs together.

```
library(gridExtra)
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)

grid.arrange(pc12, pc13, pc23, ncol=2)
```

If you have 4 dimensions:

```
pc12  <-  ggplot(comp,  aes(x=PC1,  y=PC2))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc13  <-  ggplot(comp,  aes(x=PC1,  y=PC3))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc23  <-  ggplot(comp,  aes(x=PC2,  y=PC3))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc14  <-  ggplot(comp,  aes(x=PC1,  y=PC4))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc24  <-  ggplot(comp,  aes(x=PC2,  y=PC4))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc34  <-  ggplot(comp,  aes(x=PC3,  y=PC4))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, ncol=3)
```

If you have 5 dimensions:

```
pc12  <-  ggplot(comp,  aes(x=PC1,  y=PC2))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc13  <-  ggplot(comp,  aes(x=PC1,  y=PC3))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc23  <-  ggplot(comp,  aes(x=PC2,  y=PC3))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc14  <-  ggplot(comp,  aes(x=PC1,  y=PC4))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc24  <-  ggplot(comp,  aes(x=PC2,  y=PC4))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc34  <-  ggplot(comp,  aes(x=PC3,  y=PC4))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc15  <-  ggplot(comp,  aes(x=PC1,  y=PC5))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc25  <-  ggplot(comp,  aes(x=PC2,  y=PC5))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc35  <-  ggplot(comp,  aes(x=PC3,  y=PC5))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc45  <-  ggplot(comp,  aes(x=PC4,  y=PC5))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, pc15, pc25, pc35, pc45, ncol=3)
```

If you need to add even more dimensions, study what was added to the code (new parts in bold) and follow the pattern. Do not forget to add more objects to the grid.arrange list in the last line.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

x=PC1, y=PC2 – This is present in a slightly altered form in each line. It should be the name of the two dimensions to be plotted for that graph. Replace it with the name of the dimensions in your data e.g. x=height, y=weight for one line, then x=height, y=age for the next line etc.

size=(2*kmax)^3 – This sizes the points based on their degree of association to their nearest cluster centre. You can fine-tune the numbers. The larger the number that the 2 is changed to, the bigger all points will be. The larger the number that the 3 is changed to, the more exaggerated differences between points will be. Try different numbers to get the best look for your data.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you may try fine-tuning this number if the result looks bad.

color=kmemb – This makes it so that your points are coloured based on what cluster they're in.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to kmemb. However, this may be excessively distracting if your points are already differently coloured.

ncol=3 – This is the number of columns to arrange the graphs in. If you have more graphs you will need to increase this number, though too many graphs do not fit well on one page. You can also use nrow=3 to set the number of rows.

Seeing what's in each cluster

You can list the objects in each cluster with the following code:

```
clust <- names(sort(table(kmemb)))
clustnumb <- length(clust)
for(i in 1:clustnumb){
  print(c("Cluster number:", clust[i]))
  print(row.names(comp[kmemb==clust[i],]))
}
sort(table(kmemb))
k$memb
```

It lists the clusters by order of size. Because of this, you must be careful when comparing two different clustering algorithms on the same data. The order of clusters may not be the same between the 2 algorithms. The last line creates a table of memberships.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

Comparing clusters by each original variable

You can see how the individuals in your cluster compare by plotting boxplots for all of your original variables (before doing a PCA). The package **ggplot2** will be used again.

First, you need to add a new column to your data which can be used to separate the individuals into clusters for the boxplots.

```
NAMEOFDATA$Cluster <- kmemb
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning). Keep the $Cluster part on the end.

Now you can compare the clusters across one variable using the following code:

```
qplot(factor(Cluster),   VARIABLE,   geom   =   "boxplot",   data   =   NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE')  +  geom_boxplot(aes(fill  =  factor(Cluster))  +
theme(legend.position="NULL")
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE – Replace this with the name of the variable. This is the column heading from your original data you would like to compare the clusters by e.g. height. Note that you need to replace it in 2 places in the code.

If you want to display multiple boxplots on one page, you can use **gridExtra** again.

```
box1 <- qplot(factor(Cluster), VARIABLE1, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE1') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

box2 <- qplot(factor(Cluster), VARIABLE2, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE2') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

box3 <- qplot(factor(Cluster), VARIABLE3, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE3') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

grid.arrange(box1, box2, box3, ncol=2)
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE1, VARIABLE2, VARIABLE3 etc. – Replace these with the names of the variables i.e. column headings from your original data you would like to compare the clusters by. Note that for each one, there are 2 places in the code to replace them.

Adding extra graphs is easy, just don't forget to add box4, box5 etc. to the grid.arrange list in the last line.

Note that the medians shown on these boxplots are not weighted. There is an option to use weighted means instead for fuzzy clustering with the following limited workaround:

You can use the degrees of membership as weights. Once you find the weighed mean for each cluster, create a data frame with the clusters in one column and the weighted means in the other.

```
category <- c(1:max(kmemb))
wmean <- c(weightedmeanforcluster1, weightedmeanforcluster2, etc.)
df <- data.frame(category, wmean)
```

In the above code, you will need to fill in the wmean vector with your weighted means.

Then add the following code *to the same line* as each of your boxplots:

```
+ geom_point(data=df, aes(x=category, y=wmean), shape = 23, size = 3, fill =
"white", inherit.aes=FALSE)
```

Section 5.2

Doing Fuzzy C-Medoids

The package **vegclust** needs to be installed. Then use the following code in R:

```
library(vegclust)
k <- vegclust(x=comp, mobileCenters=x, method="FCMdd", nstart=25, iter.max=1000)
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

mobileCenters=x – Replace x with the number of clusters to find e.g. 2, 3. You can find out how many clusters is the optimal number by following the guide in Chapter 3.

nstart=25 – This repeats the algorithm 25 times and chooses the best result. If the dataset is exceptionally large, you may wish to decrease this number slightly.

Other parameters:

k (first one) – This is the name of the object that the results of the clustering are going to be stored in. It should be left as k because the follow up code (for plotting, comparing etc.) assumes it is called k.

method='FCMdd' – Sets the algorithm to Fuzzy C-Means

iter.max=1000 – This is the maximum number of iterations the algorithm is allowed to use before it is forcefully stopped, whether an optimum is reached or not. As such, it should be set to a large number like 1000 to make sure the algorithm finishes, though only a few dozen should usually be enough. If the algorithm does not converge before it reaches its maximum number of iterations, it will give a warning in R.

Plotting the results in a scatterplot

You can use the package **ggplot2** to plot the results of the clustering with colours distinguishing the clusters. This package will need to be installed.

However, if you want to show membership in the graph, you can do this by having the points be differently sized (depending on how strongly they are associated to their nearest cluster centre). You will need to assign sizes to points before plotting. First, retrieve the sizes with the following code:

```
nr <- nrow(k$memb)
nc <- ncol(k$memb)
B <- matrix(0,nrow=nr, ncol=nc)
for(i in 1:nr){
  x <- which(k$memb[i,]==max(k$memb[i,]))
  B[i,x] <- 1
}
```

Next, assign the sizes to the points. If you have 2 clusters, use the following code.

```
B[,1] + B[,2]*2 -> kmemb
kmax <- apply(k$memb[, 1:2], 1, max)
```

If you have 3 clusters, use this code instead (new parts in bold):

```
B[,1] + B[,2]*2 + B[,3]*3 -> kmemb
kmax <- apply(k$memb[, 1:3], 1, max)
```

If you have 4 clusters, use this code instead (new parts in bold):

```
B[,1] + B[,2]*2 + B[,3]*3 + B[,4]*4 -> kmemb
kmax <- apply(k$memb[, 1:4], 1, max)
```

Follow this pattern if you have more than 4 clusters.

Now that the sizes have been assigned, you can plot the results.

If you have **2 dimensions**:

```
library(ggplot2)
ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=kmemb, size=(2*kmax)^3,
pch=16)
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data.

x=PC1, y=PC2 – This will be the name of your two dimensions if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of the dimensions in your data e.g. x=height, y=weight.

size=(2*kmax)^3 – This sizes the points based on their degree of association to their nearest cluster centre. You can fine-tune the numbers. The larger the number that the 2 is changed to, the bigger all points will be. The larger the number that the 3 is changed to, the more exaggerated differences between points will be. Try different numbers to get the best look for your data.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you can try fine-tuning this number if the result looks bad.

color=kmemb – This makes it so that your points are coloured based on what cluster they're in.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to kmemb. However, this may be excessively distracting if your points are already differently coloured.

If you have **more than 2 dimensions:**

You will need to do multiple graphs because a 2D graph can only plot 2 dimensions at once (the **rgl** package can plot 3D graphs but this is not useful for papers). The number of graphs you need will be equal to the number of ways your dimensions can be paired. You can find out how many dimensions you need can by typing choose(x,2) into R, where x is the number of dimensions.

The **gridExtra** package can be used to display multiple graphs on the same page. The code below assumes you have 3 dimensions. Each line which begins with pc is a new graph, the last lines joins all the graphs together.

```
library(gridExtra)
pc12  <-  ggplot(comp,  aes(x=PC1,  y=PC2))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc13  <-  ggplot(comp,  aes(x=PC1,  y=PC3))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc23  <-  ggplot(comp,  aes(x=PC2,  y=PC3))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)

grid.arrange(pc12, pc13, pc23, ncol=2)
```

If you have 4 dimensions:

```
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc14 <- ggplot(comp, aes(x=PC1, y=PC4)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc24 <- ggplot(comp, aes(x=PC2, y=PC4)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc34 <- ggplot(comp, aes(x=PC3, y=PC4)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, ncol=3)
```

If you have 5 dimensions:

```
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc14 <- ggplot(comp, aes(x=PC1, y=PC4)) + geom_point(alpha=.7, color=kmemb,
size(2*kmax)^3, pch = 16)
pc24 <- ggplot(comp, aes(x=PC2, y=PC4)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc34 <- ggplot(comp, aes(x=PC3, y=PC4)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc15 <- ggplot(comp, aes(x=PC1, y=PC5)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc25 <- ggplot(comp, aes(x=PC2, y=PC5)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc35 <- ggplot(comp, aes(x=PC3, y=PC5)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc45 <- ggplot(comp, aes(x=PC4, y=PC5)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, pc15, pc25, pc35, pc45, ncol=3)
```

If you need to add even more dimensions, study what was added to the code (new parts in bold) and follow the pattern. Do not forget to add more objects to the grid.arrange list in the last line.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

x=PC1, y=PC2 – This is present in a slightly altered form in each line. It should be the name of the two dimensions to be plotted for that graph. Replace it with the name of the dimensions in your data e.g. x=height, y=weight for one line, then x=height, y=age for the next line etc.

size=(2*kmax)^3 – This sizes the points based on their degree of association to their nearest cluster centre. You can fine-tune the numbers. The larger the number that the 2 is changed to, the bigger all points will be. The larger the number that the 3 is changed to, the more exaggerated differences between points will be. Try different numbers to get the best look for your data.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you may try fine-tuning this number if the result looks bad.

color=kmemb – This makes it so that your points are coloured based on what cluster they're in.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to kmemb. However, this may be excessively distracting if your points are already differently coloured.

ncol=3 – This is the number of columns to arrange the graphs in. If you have more graphs you will need to increase this number, though too many graphs do not fit well on one page. You can also use nrow=3 to set the number of rows.

Seeing what's in each cluster

You can list the objects in each cluster with the following code:

```
clust <- names(sort(table(kmemb)))
clustnumb <- length(clust)
for(i in 1:clustnumb){
  print(c("Cluster number:", clust[i]))
  print(row.names(comp[kmemb==clust[i],]))
}
sort(table(kmemb))
k$memb
```

It lists the clusters by order of size. Because of this, you must be careful when comparing two different clustering algorithms on the same data. The order of clusters may not be the same between the 2 algorithms. The last line creates a table of memberships.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

Comparing clusters by each original variable

You can see how the individuals in your cluster compare by plotting boxplots for all of your original variables (before doing a PCA). The package **ggplot2** will be used again.

First, you need to add a new column to your data which can be used to separate the individuals into clusters for the boxplots.

```
NAMEOFDATA$Cluster <- kmemb
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning). Keep the $Cluster part on the end.

Now you can compare the clusters across one variable using the following code:

```
qplot(factor(Cluster), VARIABLE, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE – Replace this with the name of the variable. This is the column heading from your original data you would like to compare the clusters by e.g. height. Note that you need to replace it in 2 places in the code.

If you want to display multiple boxplots on one page, you can use **gridExtra** again.

```
box1  <-  qplot(factor(Cluster),  VARIABLE1,  geom  =  "boxplot",  data  =  NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE1')  +  geom_boxplot(aes(fill  =  factor(Cluster)))  +
theme(legend.position="NULL")

box2  <-  qplot(factor(Cluster),  VARIABLE2,  geom  =  "boxplot",  data  =  NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE2')  +  geom_boxplot(aes(fill  =  factor(Cluster)))  +
theme(legend.position="NULL")

box3  <-  qplot(factor(Cluster),  VARIABLE3,  geom  =  "boxplot",  data  =  NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE3')  +  geom_boxplot(aes(fill  =  factor(Cluster)))  +
theme(legend.position="NULL")

grid.arrange(box1, box2, box3, ncol=2)
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE1, VARIABLE2, VARIABLE3 etc. – Replace these with the names of the variables i.e. column headings from your original data you would like to compare the clusters by. Note that for each one, there are 2 places in the code to replace them.

Adding extra graphs is easy, just don't forget to add box4, box5 etc. to the grid.arrange list in the last line.

Note that the medians shown on these boxplots are not weighted. There is an option to use weighted means instead for fuzzy clustering with the following limited workaround:

You can use the degrees of membership as weights. Once you find the weighed mean for each cluster, create a data frame with the clusters in one column and the weighted means in the other.

```
category <- c(1:max(kmemb))
wmean <- c(weightedmeanforcluster1, weightedmeanforcluster2, etc.)
df <- data.frame(category, wmean)
```

In the above code, you will need to fill in the wmean vector with your weighted means.

Then add the following code *to the same line* as each of your boxplots:

```
+  geom_point(data=df,  aes(x=category,  y=wmean),  shape  =  23,  size  =  3,  fill  =
"white", inherit.aes=FALSE)
```

Section 6.1

Doing K-Means with Noise

The package **vegclust** needs to be installed. Then use the following code in R:

```
library(vegclust)
k  <-  vegclust(x=comp,  mobileCenters=x,  method="HNC",  nstart=25,  iter.max=1000,
dnoise=3)
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

mobileCenters=x – Replace x with the number of clusters to find e.g. 2, 3. You can find out how many clusters is the optimal number by following the guide in Chapter 3.

nstart=25 – This repeats the algorithm 25 times and chooses the best result. If the dataset is exceptionally large, you may wish to decrease this number slightly.

dnoise=3 – This is the noise threshold. You should try replacing 3 with different values. The larger the threshold is, the lower the number of points filtered as noise will be.

Other parameters:

k (first one) – This is the name of the object that the results of the clustering are going to be stored in. It should be left as k because the follow up code (for plotting, comparing etc.) assumes it is called k.

method='HNC' – Sets the algorithm to K-Means with noise.

iter.max=1000 – This is the maximum number of iterations the algorithm is allowed to use before it is forcefully stopped, whether an optimum is reached or not. As such, it should be set to a large number like 1000 to make sure the algorithm finishes, though only a few dozen should usually be enough. If the algorithm does not converge before it reaches its maximum number of iterations, it will give a warning in R.

Plotting the results in a scatterplot

You can use the package **ggplot2** to plot the results of the clustering with colours distinguishing the clusters. This package will need to be installed.

First though, you need to assign memberships to the points (including the noise group). If you have 2 clusters, you will have to assign *3* groups because of the noise group. Use the following code:

```
k$memb[,1] + k$memb[,2]*2 + k$memb[,3]*3 -> kmemb
```

If you have 3 clusters, you will need to assign 4 groups (new parts in **bold**):

```
k$memb[,1] + k$memb[,2]*2 + k$memb[,3]*3 + k$memb[,4]*4 -> kmemb
```

Follow this pattern if you have even more clusters.

Now you are ready to plot. If you have **2 dimensions**, use this code:

```
library(ggplot2)
ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=kmemb, size=3, pch=16)
```

You can choose to not plot the points filtered at all too. You can remove the noise points like this:

```
kmemb -> kmemb2
kmemb2[kmemb2==max(kmemb2)] <- NA
```

Then, replace kmemb with kmemb2 in the code for plotting.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data.

x=PC1, y=PC2 – This will be the name of your two dimensions if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of the dimensions in your data e.g. x=height, y=weight.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you can try fine-tuning this number if the result looks bad.

color=kmemb – This makes it so that your points are coloured based on what cluster they're in.

size=3 – This determines the size of your points. If the points are too big, it covers too much area and makes it impossible to tell where the point's position actually is. If it is too small, it will be hard to see. 3 is a good balance, though if you have a lot of points, you may wish to try a smaller number.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to kmemb. However, this may be excessively distracting if your points are already differently coloured.

If you have **more than 2 dimensions:**

You will need to do multiple graphs because a 2D graph can only plot 2 dimensions at once (the **rgl** package can plot 3D graphs but this is not useful for papers). The number of graphs you need will be equal to the number of ways your dimensions can be paired. You can find out how many dimensions you need can by typing choose(x,2) into R, where x is the number of dimensions.

The **gridExtra** package can be used to display multiple graphs on the same page. The code below assumes you have 3 dimensions. Each line which begins with pc is a new graph, the last lines joins all the graphs together.

```
library(gridExtra)
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)

grid.arrange(pc12, pc13, pc23, ncol=2)
```

If you have 4 dimensions:

```
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc14 <- ggplot(comp, aes(x=PC1, y=PC4)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc24 <- ggplot(comp, aes(x=PC2, y=PC4)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc34 <- ggplot(comp, aes(x=PC3, y=PC4)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, ncol=3)
```

If you have 5 dimensions:

```
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc14 <- ggplot(comp, aes(x=PC1, y=PC4)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc24 <- ggplot(comp, aes(x=PC2, y=PC4)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc34 <- ggplot(comp, aes(x=PC3, y=PC4)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc15 <- ggplot(comp, aes(x=PC1, y=PC5)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc25 <- ggplot(comp, aes(x=PC2, y=PC5)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc35 <- ggplot(comp, aes(x=PC3, y=PC5)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc45 <- ggplot(comp, aes(x=PC4, y=PC5)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, pc15, pc25, pc35, pc45, ncol=3)
```

If you need to add even more dimensions, study what was added to the code (new parts in bold) and follow the pattern. Do not forget to add more objects to the grid.arrange list in the last line.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1’s guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

x=PC1, y=PC2 – This is present in a slightly altered form in each line. It should be the name of the two dimensions to be plotted for that graph. Replace it with the name of the dimensions in your data e.g. x=height, y=weight for one line, then x=height, y=age for the next line etc.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you may try fine-tuning this number if the result looks bad.

color=kmemb – This makes it so that your points are coloured based on what cluster they're in.

size=3 – This determines the size of your points. If the points are too big, it covers too much area and makes it impossible to tell where the point's position actually is. If it is too small, it will be hard to see. 3 is a good balance, though if you have a lot of points, you may wish to try a smaller number.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to kmemb. However, this may be excessively distracting if your points are already differently coloured.

ncol=3 – This is the number of columns to arrange the graphs in. If you have more graphs you will need to increase this number, though too many graphs do not fit well on one page. You can also use nrow=3 to set the number of rows.

Seeing what's in each cluster

You can list the objects in each cluster with the following code:

```
clust <- names(sort(table(kmemb)))
clustnumb <- length(clust)
for(i in 1:clustnumb){
  print(c("Cluster number:", clust[i]))
  print(row.names(comp[kmemb==clust[i],]))
}
sort(table(kmemb))
```

It lists the clusters by order of size. Because of this, you must be careful when comparing two different clustering algorithms on the same data. The order of clusters may not be the same between the 2 algorithms. Also note that one of the clusters is the noise cluster. You can check the non-noise points only by replacing kmemb with kmemb2 (3 times) in the above code.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

Comparing clusters by each original variable

You can see how the individuals in your cluster compare by plotting boxplots for all of your original variables (before doing a PCA). The package **ggplot2** will be used again.

First, you need to add a new column to your data which can be used to separate the individuals into clusters for the boxplots.

```
kmemb -> kmemb2
kmemb2[kmemb2==max(kmemb2)] <- NA
NAMEOFDATA$Cluster <- kmemb2
NAMEOFDATA <- NAMEOFDATA[complete.cases(NAMEOFDATA),]
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning). Keep the $Cluster part on the end.

Now you can compare the clusters across one variable using the following code:

```
qplot(factor(Cluster),   VARIABLE,   geom   =   "boxplot",   data   =   NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE')  +  geom_boxplot(aes(fill  =  factor(Cluster)))  +
theme(legend.position="NULL")
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE – Replace this with the name of the variable. This is the column heading from your original data you would like to compare the clusters by e.g. height. Note that you need to replace it in 2 places in the code.

If you want to display multiple boxplots on one page, you can use **gridExtra** again.

```
box1  <-  qplot(factor(Cluster),  VARIABLE1,  geom  =  "boxplot",  data  =  NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE1')  +  geom_boxplot(aes(fill  =  factor(Cluster)))  +
theme(legend.position="NULL")

box2  <-  qplot(factor(Cluster),  VARIABLE2,  geom  =  "boxplot",  data  =  NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE2')  +  geom_boxplot(aes(fill  =  factor(Cluster)))  +
theme(legend.position="NULL")

box3  <-  qplot(factor(Cluster),  VARIABLE3,  geom  =  "boxplot",  data  =  NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE3')  +  geom_boxplot(aes(fill  =  factor(Cluster)))  +
theme(legend.position="NULL")

grid.arrange(box1, box2, box3, ncol=2)
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE1, VARIABLE2, VARIABLE3 etc. – Replace these with the names of the variables i.e. column headings from your original data you would like to compare the clusters by. Note that for each one, there are 2 places in the code to replace them.

Adding extra graphs is easy, just don’t forget to add box4, box5 etc. to the grid.arrange list in the last line.

Section 6.2

Doing K-Medoids with Noise

The package **vegclust** needs to be installed. Then use the following code in R:

```
library(vegclust)
k <- vegclust(x=comp, mobileCenters=x, method="HNCdd", nstart=25, iter.max=1000,
dnoise=3)
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1’s guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

mobileCenters=x – Replace x with the number of clusters to find e.g. 2, 3. You can find out how many clusters is the optimal number by following the guide in Chapter 3.

nstart=25 – This repeats the algorithm 25 times and chooses the best result. If the dataset is exceptionally large, you may wish to decrease this number slightly.

dnoise=3 – This is the noise threshold. You should try replacing 3 with different values. The larger the threshold is, the lower the number of points filtered as noise will be.

Other parameters:

k (first one) – This is the name of the object that the results of the clustering are going to be stored in. It should be left as k because the follow up code (for plotting, comparing etc.) assumes it is called k.

method=‘HNCdd’ – Sets the algorithm to K-Medoids with noise.

iter.max=1000 – This is the maximum number of iterations the algorithm is allowed to use before it is forcefully stopped, whether an optimum is reached or not. As such, it should be set to a large number like 1000 to make sure the algorithm finishes, though only a few dozen should usually be enough. If the algorithm does not converge before it reaches its maximum number of iterations, it will give a warning in R.

Plotting the results in a scatterplot

You can use the package **ggplot2** to plot the results of the clustering with colours distinguishing the clusters. This package will need to be installed.

First though, you need to assign memberships to the points (including the noise group). If you have 2 clusters, you will have to assign *3* groups because of the noise group:

```
k$memb[,1] + k$memb[,2]*2 + k$memb[,3]*3 -> kmemb
```

If you have 3 clusters, you will need to assign 4 groups (new parts in **bold**):

```
k$memb[,1] + k$memb[,2]*2 + k$memb[,3]*3 + k$memb[,4]*4 -> kmemb
```

Follow this pattern if you have even more clusters.

Now you are ready to plot. If you have **2 dimensions**:

```
library(ggplot2)
ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=kmemb, size=3, pch=16)
```

You can choose to not plot the points filtered at all too. You can remove the noise points like this:

```
kmemb -> kmemb2
kmemb2[kmemb2==max(kmemb2)] <- NA
```

Then, replace kmemb with kmemb2 in the code for plotting.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data.

x=PC1, y=PC2 – This will be the name of your two dimensions if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of the dimensions in your data e.g. x=height, y=weight.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you can try fine-tuning this number if the result looks bad.

color=kmemb – This makes it so that your points are coloured based on what cluster they're in.

size=3 – This determines the size of your points. If the points are too big, it covers too much area and makes it impossible to tell where the point's position actually is. If it is too small, it will be hard to see. 3 is a good balance, though if you have a lot of points, you may wish to try a smaller number.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to kmemb. However, this may be excessively distracting if your points are already differently coloured.

If you have **more than 2 dimensions:**

You will need to do multiple graphs because a 2D graph can only plot 2 dimensions at once (the **rgl** package can plot 3D graphs but this is not useful for papers). The number of graphs you need will be equal to the number of ways your dimensions can be paired. You can find out how many dimensions you need can by typing choose(x,2) into R, where x is the number of dimensions.

The **gridExtra** package can be used to display multiple graphs on the same page. The code below assumes you have 3 dimensions. Each line which begins with pc is a new graph, the last lines joins all the graphs together.

```
library(gridExtra)
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)

grid.arrange(pc12, pc13, pc23, ncol=2)
```

If you have 4 dimensions:

```
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc14 <- ggplot(comp, aes(x=PC1, y=PC4)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc24 <- ggplot(comp, aes(x=PC2, y=PC4)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc34 <- ggplot(comp, aes(x=PC3, y=PC4)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, ncol=3)
```

If you have 5 dimensions:

```
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc14 <- ggplot(comp, aes(x=PC1, y=PC4)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc24 <- ggplot(comp, aes(x=PC2, y=PC4)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc34 <- ggplot(comp, aes(x=PC3, y=PC4)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc15 <- ggplot(comp, aes(x=PC1, y=PC5)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc25 <- ggplot(comp, aes(x=PC2, y=PC5)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc35 <- ggplot(comp, aes(x=PC3, y=PC5)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)
pc45 <- ggplot(comp, aes(x=PC4, y=PC5)) + geom_point(alpha=.7, color=kmemb, size=3,
pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, pc15, pc25, pc35, pc45, ncol=3)
```

If you need to add even more dimensions, study what was added to the code (new parts in bold) and follow the pattern. Do not forget to add more objects to the grid.arrange list in the last line.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

x=PC1, y=PC2 – This is present in a slightly altered form in each line. It should be the name of the two dimensions to be plotted for that graph. Replace it with the name of the dimensions in your data e.g. x=height, y=weight for one line, then x=height, y=age for the next line etc.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you may try fine-tuning this number if the result looks bad.

color=kmemb – This makes it so that your points are coloured based on what cluster they're in.

size=3 – This determines the size of your points. If the points are too big, it covers too much area and makes it impossible to tell where the point's position actually is. If it is too small, it will be hard to see. 3 is a good balance, though if you have a lot of points, you may wish to try a smaller number.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to kmemb. However, this may be excessively distracting if your points are already differently coloured.

ncol=3 – This is the number of columns to arrange the graphs in. If you have more graphs you will need to increase this number, though too many graphs do not fit well on one page. You can also use nrow=3 to set the number of rows.

Seeing what's in each cluster

You can list the objects in each cluster with the following code:

```
clust <- names(sort(table(kmemb)))
clustnumb <- length(clust)
for(i in 1:clustnumb){
  print(c("Cluster number:", clust[i]))
  print(row.names(comp[kmemb==clust[i],]))
}
sort(table(kmemb))
```

It lists the clusters by order of size. Because of this, you must be careful when comparing two different clustering algorithms on the same data. The order of clusters may not be the same between the 2 algorithms. Also note that one of the clusters is the noise cluster. You can check the non-noise points only by replacing kmemb with kmemb2 (3 times) in the above code.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

Comparing clusters by each original variable

You can see how the individuals in your cluster compare by plotting boxplots for all of your original variables (before doing a PCA). The package **ggplot2** will be used again.

First, you need to add a new column to your data which can be used to separate the individuals into clusters for the boxplots.

```
kmemb -> kmemb2
kmemb2[kmemb2==max(kmemb2)] <- NA
NAMEOFDATA$Cluster <- kmemb2
NAMEOFDATA <- NAMEOFDATA[complete.cases(NAMEOFDATA),]
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning). Keep the $Cluster part on the end.

Now you can compare the clusters across one variable using the following code:

```
qplot(factor(Cluster),   VARIABLE,   geom   =   "boxplot",   data   =   NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE')  +  geom_boxplot(aes(fill  =  factor(Cluster)))  +
theme(legend.position="NULL")
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE – Replace this with the name of the variable. This is the column heading from your original data you would like to compare the clusters by e.g. height. Note that you need to replace it in 2 places in the code.

If you want to display multiple boxplots on one page, you can use **gridExtra** again.

```
box1  <-  qplot(factor(Cluster),  VARIABLE1,  geom  =  "boxplot",  data  =  NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE1')  +  geom_boxplot(aes(fill  =  factor(Cluster)))  +
theme(legend.position="NULL")

box2  <-  qplot(factor(Cluster),  VARIABLE2,  geom  =  "boxplot",  data  =  NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE2')  +  geom_boxplot(aes(fill  =  factor(Cluster)))  +
theme(legend.position="NULL")

box3  <-  qplot(factor(Cluster),  VARIABLE3,  geom  =  "boxplot",  data  =  NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE3')  +  geom_boxplot(aes(fill  =  factor(Cluster)))  +
theme(legend.position="NULL")

grid.arrange(box1, box2, box3, ncol=2)
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE1, VARIABLE2, VARIABLE3 etc. – Replace these with the names of the variables i.e. column headings from your original data you would like to compare the clusters by. Note that for each one, there are 2 places in the code to replace them.

Adding extra graphs is easy, just don’t forget to add box4, box5 etc. to the grid.arrange list in the last line.

Section 6.3

Doing Fuzzy C-Means with Noise

The package **vegclust** needs to be installed. Then use the following code in R:

```
library(vegclust)
k  <-  vegclust(x=comp,  mobileCenters=x,  method="NC",  nstart=25,  iter.max=1000,
dnoise=3)
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1’s guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

mobileCenters=x – Replace x with the number of clusters to find e.g. 2, 3. You can find out how many clusters is the optimal number by following the guide in Chapter 3.

nstart=25 – This repeats the algorithm 25 times and chooses the best result. If the dataset is exceptionally large, you may wish to decrease this number slightly.

dnoise=3 – This is the noise threshold. You should try replacing 3 with different values. The larger the threshold is, the lower the number of points filtered as noise will be.

Other parameters:

k (first one) – This is the name of the object that the results of the clustering are going to be stored in. It should be left as k because the follow up code (for plotting, comparing etc.) assumes it is called k.

method=‘NC’ – Sets the algorithm to Fuzzy C-Means with noise.

iter.max=1000 – This is the maximum number of iterations the algorithm is allowed to use before it is forcefully stopped, whether an optimum is reached or not. As such, it should be set to a large number like 1000 to make sure the algorithm finishes, though only a few dozen should usually be enough. If the algorithm does not converge before it reaches its maximum number of iterations, it will give a warning in R.

Plotting the results in a scatterplot

You can use the package **ggplot2** to plot the results of the clustering with colours distinguishing the clusters. This package will need to be installed.

However, if you want to show membership in the graph, you can do this by having the points be differently sized (depending on how strongly they are associated to their nearest cluster centre). You will need to assign sizes to points before plotting. First, retrieve the sizes with the following code:

```
nr <- nrow(k$memb)
nc <- ncol(k$memb)
B <- matrix(0,nrow=nr, ncol=nc)
for(i in 1:nr){
  x <- which(k$memb[i,]==max(k$memb[i,]))
  B[i,x] <- 1
}
```

Next, assign the sizes to the points. If you have 2 clusters, you need to assign *3* groups (1 for noise).

```
B[,1] + B[,2]*2 + B[,3]*3 -> kmemb
kmax <- apply(k$memb[, 1:3], 1, max)
```

If you have 3 clusters, you need to assign *4* groups (new parts in bold):

```
B[,1] + B[,2]*2 + B[,3]*3 + B[,4]*4 -> kmemb
kmax <- apply(k$memb[, 1:4], 1, max)
```

Follow this pattern if you have more than 3 clusters.

Now that the sizes have been assigned, you can plot the results.

If you have **2 dimensions**:

```
library(ggplot2)
ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=kmemb, size=(2*kmax)^3,
pch=16)
```

You can choose to not plot the points filtered at all too. You can remove the noise points like this:

```
kmemb -> kmemb2
kmemb2[kmemb2==max(kmemb2)] <- NA
```

Then, replace kmemb with kmemb2 in the code for plotting.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data.

x=PC1, y=PC2 – This will be the name of your two dimensions if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of the dimensions in your data e.g. x=height, y=weight.

size=(2*kmax)^3 – This sizes the points based on their degree of association to their nearest cluster centre. You can fine-tune the numbers. The larger the number that the 2 is changed to, the bigger all points will be. The larger the number that the 3 is changed to, the more exaggerated differences between points will be. Try different numbers to get the best look for your data.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you can try fine-tuning this number if the result looks bad.

color=kmemb – This makes it so that your points are coloured based on what cluster they're in.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to kmemb. However, this may be excessively distracting if your points are already differently coloured.

If you have **more than 2 dimensions:**

You will need to do multiple graphs because a 2D graph can only plot 2 dimensions at once (the **rgl** package can plot 3D graphs but this is not useful for papers). The number of graphs you need will be equal to the number of ways your dimensions can be paired. You can find out how many dimensions you need can by typing choose(x,2) into R, where x is the number of dimensions.

The **gridExtra** package can be used to display multiple graphs on the same page. The code below assumes you have 3 dimensions. Each line which begins with pc is a new graph, the last lines joins all the graphs together.

```
library(gridExtra)
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=kmemb, size=(2*kmax)^3, pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=kmemb, size=(2*kmax)^3, pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=kmemb, size=(2*kmax)^3, pch = 16)

grid.arrange(pc12, pc13, pc23, ncol=2)
```

If you have 4 dimensions:

```
pc12  <-  ggplot(comp,  aes(x=PC1,  y=PC2))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc13  <-  ggplot(comp,  aes(x=PC1,  y=PC3))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc23  <-  ggplot(comp,  aes(x=PC2,  y=PC3))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc14  <-  ggplot(comp,  aes(x=PC1,  y=PC4))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc24  <-  ggplot(comp,  aes(x=PC2,  y=PC4))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc34  <-  ggplot(comp,  aes(x=PC3,  y=PC4))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, ncol=3)
```

If you have 5 dimensions:

```
pc12  <-  ggplot(comp,  aes(x=PC1,  y=PC2))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc13  <-  ggplot(comp,  aes(x=PC1,  y=PC3))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc23  <-  ggplot(comp,  aes(x=PC2,  y=PC3))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc14  <-  ggplot(comp,  aes(x=PC1,  y=PC4))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc24  <-  ggplot(comp,  aes(x=PC2,  y=PC4))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc34  <-  ggplot(comp,  aes(x=PC3,  y=PC4))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc15  <-  ggplot(comp,  aes(x=PC1,  y=PC5))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc25  <-  ggplot(comp,  aes(x=PC2,  y=PC5))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc35  <-  ggplot(comp,  aes(x=PC3,  y=PC5))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)
pc45  <-  ggplot(comp,  aes(x=PC4,  y=PC5))  +  geom_point(alpha=.7,  color=kmemb,
size=(2*kmax)^3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, pc15, pc25, pc35, pc45, ncol=3)
```

If you need to add even more dimensions, study what was added to the code (new parts in bold) and follow the pattern. Do not forget to add more objects to the grid.arrange list in the last line.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

x=PC1, y=PC2 – This is present in a slightly altered form in each line. It should be the name of the two dimensions to be plotted for that graph. Replace it with the name of the dimensions in your data e.g. x=height, y=weight for one line, then x=height, y=age for the next line etc.

size=(2*kmax)^3 – This sizes the points based on their degree of association to their nearest cluster centre. You can fine-tune the numbers. The larger the number that the 2 is changed to, the bigger all points will be. The larger the number that the 3 is changed to, the more exaggerated differences between points will be. Try different numbers to get the best look for your data.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you may try fine-tuning this number if the result looks bad.

color=kmemb – This makes it so that your points are coloured based on what cluster they're in.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to kmemb. However, this may be excessively distracting if your points are already differently coloured.

ncol=3 – This is the number of columns to arrange the graphs in. If you have more graphs you will need to increase this number, though too many graphs do not fit well on one page. You can also use nrow=3 to set the number of rows.

Seeing what's in each cluster

You can list the objects in each cluster with the following code:

```
clust <- names(sort(table(kmemb)))
clustnumb <- length(clust)
for(i in 1:clustnumb){
  print(c("Cluster number:", clust[i]))
  print(row.names(comp[kmemb==clust[i],]))
}
sort(table(kmemb))
k$memb
```

It lists the clusters by order of size. Because of this, you must be careful when comparing two different clustering algorithms on the same data. The order of clusters may not be the same between the 2 algorithms. The last line creates a table of memberships. Also note that one of the clusters is the noise cluster. You can check the non-noise points only by replacing kmemb with kmemb2 (3 times) in the above code.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

Comparing clusters by each original variable

You can see how the individuals in your cluster compare by plotting boxplots for all of your original variables (before doing a PCA). The package **ggplot2** will be used again.

First, you need to add a new column to your data which can be used to separate the individuals into clusters for the boxplots.

```
kmemb -> kmemb2
kmemb2[kmemb2==max(kmemb2)] <- NA
NAMEOFDATA$Cluster <- kmemb2
NAMEOFDATA <- NAMEOFDATA[complete.cases(NAMEOFDATA),]
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning). Keep the $Cluster part on the end.

Now you can compare the clusters across one variable using the following code:

```
qplot(factor(Cluster),   VARIABLE,   geom   =   "boxplot",   data   =   NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE')  +  geom_boxplot(aes(fill  =  factor(Cluster))  +
theme(legend.position="NULL")
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE – Replace this with the name of the variable. This is the column heading from your original data you would like to compare the clusters by e.g. height. Note that you need to replace it in 2 places in the code.

If you want to display multiple boxplots on one page, you can use **gridExtra** again.

```
box1 <- qplot(factor(Cluster), VARIABLE1, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE1') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

box2 <- qplot(factor(Cluster), VARIABLE2, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE2') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

box3 <- qplot(factor(Cluster), VARIABLE3, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE3') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

grid.arrange(box1, box2, box3, ncol=2)
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE1, VARIABLE2, VARIABLE3 etc. – Replace these with the names of the variables i.e. column headings from your original data you would like to compare the clusters by. Note that for each one, there are 2 places in the code to replace them.

Adding extra graphs is easy, just don't forget to add box4, box5 etc. to the grid.arrange list in the last line.

Note that the medians shown on these boxplots are not weighted. There is an option to use weighted means instead for fuzzy clustering with the following limited workaround:

You can use the degrees of membership as weights. Once you find the weighed mean for each cluster, create a data frame with the clusters in one column and the weighted means in the other.

```
category <- c(1:max(kmemb))
wmean <- c(weightedmeanforcluster1, weightedmeanforcluster2, etc., 1)
df <- data.frame(category, wmean)
```

In the above code, you will need to fill in the wmean vector with your weighted means.

Then add the following code *to the same line* as each of your boxplots:

```
+ geom_point(data=df, aes(x=category, y=wmean), shape = 23, size = 3, fill =
"white", inherit.aes=FALSE)
```

Section 6.4

Doing Fuzzy C-Medoids with Noise

The package **vegclust** needs to be installed. Then use the following code in R:

```
library(vegclust)
k  <-  vegclust(x=comp,  mobileCenters=x,  method="NCdd",  nstart=25,  iter.max=1000,
dnoise=3)
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

mobileCenters=x – Replace x with the number of clusters to find e.g. 2, 3. You can find out how many clusters is the optimal number by following the guide in Chapter 3.

nstart=25 – This repeats the algorithm 25 times and chooses the best result. If the dataset is exceptionally large, you may wish to decrease this number slightly.

dnoise=3 – This is the noise threshold. You should try replacing 3 with different values. The larger the threshold is, the lower the number of points filtered as noise will be.

Other parameters:

k (first one) – This is the name of the object that the results of the clustering are going to be stored in. It should be left as k because the follow up code (for plotting, comparing etc.) assumes it is called k.

method='NCdd' – Sets the algorithm to Fuzzy C-Medoids with noise.

iter.max=1000 – This is the maximum number of iterations the algorithm is allowed to use before it is forcefully stopped, whether an optimum is reached or not. As such, it should be set to a large number like 1000 to make sure the algorithm finishes, though only a few dozen should usually be enough. If the algorithm does not converge before it reaches its maximum number of iterations, it will give a warning in R.

Plotting the results in a scatterplot

You can use the package **ggplot2** to plot the results of the clustering with colours distinguishing the clusters. This package will need to be installed.

However, if you want to show membership in the graph, you can do this by having the points be differently sized (depending on how strongly they are associated to their nearest cluster centre). You will need to assign sizes to points before plotting. First, retrieve the sizes with the following code:

```
nr <- nrow(k$memb)
nc <- ncol(k$memb)
B <- matrix(0,nrow=nr, ncol=nc)
for(i in 1:nr){
  x <- which(k$memb[i,]==max(k$memb[i,]))
  B[i,x] <- 1
}
```

Next, assign the sizes to the points. If you have 2 clusters, you need to assign *3* groups (1 for noise).

```
B[,1] + B[,2]*2 + B[,3]*3 -> kmemb
kmax <- apply(k$memb[, 1:3], 1, max)
```

If you have 3 clusters, you need to assign *4* groups (new parts in bold):

```
B[,1] + B[,2]*2 + B[,3]*3 + B[,4]*4 -> kmemb
kmax <- apply(k$memb[, 1:4], 1, max)
```

Follow this pattern if you have more than 3 clusters.

Now that the sizes have been assigned, you can plot the results.

If you have **2 dimensions**:

```
library(ggplot2)
ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=kmemb, size=(2*kmax)^3,
pch=16)
```

You can choose to not plot the points filtered at all too. You can remove the noise points like this:

```
kmemb -> kmemb2
kmemb2[kmemb2==max(kmemb2)] <- NA
```

Then, replace kmemb with kmemb2 in the code for plotting.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data.

x=PC1, y=PC2 – This will be the name of your two dimensions if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of the dimensions in your data e.g. x=height, y=weight.

size=(2*kmax)^3 – This sizes the points based on their degree of association to their nearest cluster centre. You can fine-tune the numbers. The larger the number that the 2 is changed to, the bigger all points will be. The larger the number that the 3 is changed to, the more exaggerated differences between points will be. Try different numbers to get the best look for your data.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you can try fine-tuning this number if the result looks bad.

color=kmemb – This makes it so that your points are coloured based on what cluster they're in.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to kmemb. However, this may be excessively distracting if your points are already differently coloured.

If you have **more than 2 dimensions:**

You will need to do multiple graphs because a 2D graph can only plot 2 dimensions at once (the **rgl** package can plot 3D graphs but this is not useful for papers). The number of graphs you need will be equal to the number of ways your dimensions can be paired. You can find out how many dimensions you need can by typing choose(x,2) into R, where x is the number of dimensions.

The **gridExtra** package can be used to display multiple graphs on the same page. The code below assumes you have 3 dimensions. Each line which begins with pc is a new graph, the last lines joins all the graphs together.

```
library(gridExtra)
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=kmemb, size=(2*kmax)^3, pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=kmemb, size=(2*kmax)^3, pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=kmemb, size=(2*kmax)^3, pch = 16)

grid.arrange(pc12, pc13, pc23, ncol=2)
```

If you have 4 dimensions:

```
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc14 <- ggplot(comp, aes(x=PC1, y=PC4)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc24 <- ggplot(comp, aes(x=PC2, y=PC4)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc34 <- ggplot(comp, aes(x=PC3, y=PC4)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, ncol=3)
```

If you have 5 dimensions:

```
pc12 <- ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc13 <- ggplot(comp, aes(x=PC1, y=PC3)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc23 <- ggplot(comp, aes(x=PC2, y=PC3)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc14 <- ggplot(comp, aes(x=PC1, y=PC4)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc24 <- ggplot(comp, aes(x=PC2, y=PC4)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc34 <- ggplot(comp, aes(x=PC3, y=PC4)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc15 <- ggplot(comp, aes(x=PC1, y=PC5)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc25 <- ggplot(comp, aes(x=PC2, y=PC5)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc35 <- ggplot(comp, aes(x=PC3, y=PC5)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)
pc45 <- ggplot(comp, aes(x=PC4, y=PC5)) + geom_point(alpha=.7, color=kmemb,
size=(2*kmax)^3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, pc15, pc25, pc35, pc45, ncol=3)
```

If you need to add even more dimensions, study what was added to the code (new parts in bold) and follow the pattern. Do not forget to add more objects to the grid.arrange list in the last line.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

x=PC1, y=PC2 – This is present in a slightly altered form in each line. It should be the name of the two dimensions to be plotted for that graph. Replace it with the name of the dimensions in your data e.g. x=height, y=weight for one line, then x=height, y=age for the next line etc.

size=(2*kmax)^3 – This sizes the points based on their degree of association to their nearest cluster centre. You can fine-tune the numbers. The larger the number that the 2 is changed to, the bigger all points will be. The larger the number that the 3 is changed to, the more exaggerated differences between points will be. Try different numbers to get the best look for your data.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you may try fine-tuning this number if the result looks bad.

color=kmemb – This makes it so that your points are coloured based on what cluster they're in.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to kmemb. However, this may be excessively distracting if your points are already differently coloured.

ncol=3 – This is the number of columns to arrange the graphs in. If you have more graphs you will need to increase this number, though too many graphs do not fit well on one page. You can also use nrow=3 to set the number of rows.

Seeing what's in each cluster

You can list the objects in each cluster with the following code:

```
clust <- names(sort(table(kmemb)))
clustnumb <- length(clust)
for(i in 1:clustnumb){
  print(c("Cluster number:", clust[i]))
  print(row.names(comp[kmemb==clust[i],]))
}
sort(table(kmemb))
k$memb
```

It lists the clusters by order of size. Because of this, you must be careful when comparing two different clustering algorithms on the same data. The order of clusters may not be the same between the 2 algorithms. The last line creates a table of memberships. Also note that one of the clusters is the noise cluster. You can check the non-noise points only by replacing kmemb with kmemb2 (3 times) in the above code.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

Comparing clusters by each original variable

You can see how the individuals in your cluster compare by plotting boxplots for all of your original variables (before doing a PCA). The package **ggplot2** will be used again.

First, you need to add a new column to your data which can be used to separate the individuals into clusters for the boxplots.

```
kmemb -> kmemb2
kmemb2[kmemb2==max(kmemb2)] <- NA
NAMEOFDATA$Cluster <- kmemb2
NAMEOFDATA <- NAMEOFDATA[complete.cases(NAMEOFDATA),]
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning). Keep the $Cluster part on the end.

Now you can compare the clusters across one variable using the following code:

```
qplot(factor(Cluster),   VARIABLE,   geom   =   "boxplot",   data   =   NAMEOFDATA,
xlab='Cluster',  ylab='VARIABLE')  +  geom_boxplot(aes(fill  =  factor(Cluster))  +
theme(legend.position="NULL")
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE – Replace this with the name of the variable. This is the column heading from your original data you would like to compare the clusters by e.g. height. Note that you need to replace it in 2 places in the code.

If you want to display multiple boxplots on one page, you can use **gridExtra** again.

```
box1 <- qplot(factor(Cluster), VARIABLE1, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE1') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

box2 <- qplot(factor(Cluster), VARIABLE2, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE2') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

box3 <- qplot(factor(Cluster), VARIABLE3, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE3') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

grid.arrange(box1, box2, box3, ncol=2)
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE1, VARIABLE2, VARIABLE3 etc. – Replace these with the names of the variables i.e. column headings from your original data you would like to compare the clusters by. Note that for each one, there are 2 places in the code to replace them.

Adding extra graphs is easy, just don't forget to add box4, box5 etc. to the grid.arrange list in the last line.

Note that the medians shown on these boxplots are not weighted. There is an option to use weighted means instead for fuzzy clustering with the following limited workaround:

You can use the degrees of membership as weights. Once you find the weighed mean for each cluster, create a data frame with the clusters in one column and the weighted means in the other.

```
category <- c(1:max(kmemb))
wmean <- c(weightedmeanforcluster1, weightedmeanforcluster2, etc., 1)
df <- data.frame(category, wmean)
```

In the above code, you will need to fill in the wmean vector with your weighted means.

Then add the following code *to the same line* as each of your boxplots:

```
+ geom_point(data=df, aes(x=category, y=wmean), shape = 23, size = 3, fill =
"white", inherit.aes=FALSE)
```

Section 7.1

Doing Kernal K-Means

The package **kernlab** needs to be installed. First, make sure your data is a matrix with this code:

```
compmatrix <- as.matrix(comp)
```

Then do a Kernel K-Means

```
k <- kkmeans(compmatrix, centers=x, kernel="rbfdot", alg="kkmeans")
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

centres=x – Replace this with the number of clusters to find e.g. 2, 3. You can find out how many clusters is the optimal number by following the guide in Chapter 3.

kernel="rbfdot" – This makes the algorithm use the Gaussian kernel, which is the best general use kernel. You may also try the following, though they are not as versatile as rbfdot.

- `polydot` Polynomial kernel
- `vanilladot` Linear kernel
- `tanhdot` Hyperbolic tangent kernel
- `laplacedot` Laplacian kernel
- `besseldot` Bessel kernel
- `anovadot` ANOVA RBF kernel
- `splinedot` Spline kernel
- `stringdot` String kernel

Other parameters:

k – This is the name of the object that the results of the clustering are going to be stored in. It should be left as k because the follow up code (for plotting, comparing etc.) assumes it is called k.

Unfortunately you cannot set the nstart or iter.max with this package.

Plotting the results in a scatterplot

You can use the package **ggplot2** to plot the results of the clustering with colours distinguishing the clusters. This package will need to be installed.

If you have **2 dimensions**:

```
library(ggplot2)
ggplot(comp, aes(x=PC1, y=PC2)) + geom_point(alpha=.7, color=k@.Data, size=3,
pch=16)
```

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data.

x=PC1, y=PC2 – This will be the name of your two dimensions if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of the dimensions in your data e.g. x=height, y=weight.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you can try fine-tuning this number if the result looks bad.

color=k@.Data – This makes it so that your points are coloured based on what cluster they're in.

size=3 – This determines the size of your points. If the points are too big, it covers too much area and makes it impossible to tell where the point's position actually is. If it is too small, it will be hard to see. 3 is a good balance, though if you have a lot of points, you may wish to try a smaller number.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to k@.Data. However, this may be excessively distracting if your points are already differently coloured.

If you have **more than 2 dimensions:**

You will need to do multiple graphs because a 2D graph can only plot 2 dimensions at once (the **rgl** package can plot 3D graphs but this is not useful for papers). The number of graphs you need will be equal to the number of ways your dimensions can be paired. You can find out how many dimensions you need can by typing choose(x,2) into R, where x is the number of dimensions.

The **gridExtra** package can be used to display multiple graphs on the same page. The code below assumes you have 3 dimensions. Each line which begins with pc is a new graph, the last lines joins all the graphs together.

```
library(gridExtra)
pc12  <-  ggplot(comp,  aes(x=PC1,  y=PC2))  +  geom_point(alpha=.7,  color=k@.Data,
size=3, pch = 16)
pc13  <-  ggplot(comp,  aes(x=PC1,  y=PC3))  +  geom_point(alpha=.7,  color=k@.Data,
size=3, pch = 16)
pc23  <-  ggplot(comp,  aes(x=PC2,  y=PC3))  +  geom_point(alpha=.7,  color=k@.Data,
size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, ncol=2)
```

If you have 4 dimensions:

```
pc12  <-  ggplot(comp,  aes(x=PC1,  y=PC2))  +  geom_point(alpha=.7,  color=k@.Data,
size=3, pch = 16)
pc13  <-  ggplot(comp,  aes(x=PC1,  y=PC3))  +  geom_point(alpha=.7,  color=k@.Data,
size=3, pch = 16)
pc23  <-  ggplot(comp,  aes(x=PC2,  y=PC3))  +  geom_point(alpha=.7,  color=k@.Data,
size=3, pch = 16)
pc14  <-  ggplot(comp,  aes(x=PC1,  y=PC4))  +  geom_point(alpha=.7,  color=k@.Data,
size=3, pch = 16)
pc24  <-  ggplot(comp,  aes(x=PC2,  y=PC4))  +  geom_point(alpha=.7,  color=k@.Data,
size=3, pch = 16)
pc34  <-  ggplot(comp,  aes(x=PC3,  y=PC4))  +  geom_point(alpha=.7,  color=k@.Data,
size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, ncol=3)
```

If you have 5 dimensions:

```
pc12  <-  ggplot(comp,  aes(x=PC1,  y=PC2))  +  geom_point(alpha=.7,  color=k@.Data,
size=3, pch = 16)
pc13  <-  ggplot(comp,  aes(x=PC1,  y=PC3))  +  geom_point(alpha=.7,  color=k@.Data,
size=3, pch = 16)
pc23  <-  ggplot(comp,  aes(x=PC2,  y=PC3))  +  geom_point(alpha=.7,  color=k@.Data,
size=3, pch = 16)
pc14  <-  ggplot(comp,  aes(x=PC1,  y=PC4))  +  geom_point(alpha=.7,  color=k@.Data,
size=3, pch = 16)
pc24  <-  ggplot(comp,  aes(x=PC2,  y=PC4))  +  geom_point(alpha=.7,  color=k@.Data,
size=3, pch = 16)
pc34  <-  ggplot(comp,  aes(x=PC3,  y=PC4))  +  geom_point(alpha=.7,  color=k@.Data,
size=3, pch = 16)
pc15  <-  ggplot(comp,  aes(x=PC1,  y=PC5))  +  geom_point(alpha=.7,  color=k@.Data,
size=3, pch = 16)
pc25  <-  ggplot(comp,  aes(x=PC2,  y=PC5))  +  geom_point(alpha=.7,  color=k@.Data,
size=3, pch = 16)
pc35  <-  ggplot(comp,  aes(x=PC3,  y=PC5))  +  geom_point(alpha=.7,  color=k@.Data,
size=3, pch = 16)
pc45  <-  ggplot(comp,  aes(x=PC3,  y=PC5))  +  geom_point(alpha=.7,  color=k@.Data,
size=3, pch = 16)

grid.arrange(pc12, pc13, pc23, pc14, pc24, pc34, pc15, pc25, pc35, pc45, ncol=3)
```

If you need to add even more dimensions, study what was added to the code (new parts in bold) and follow the pattern. Do not forget to add more objects to the grid.arrange list in the last line.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

x=PC1, y=PC2 – This is present in a slightly altered form in each line. It should be the name of the two dimensions to be plotted for that graph. Replace it with the name of the dimensions in your data e.g. x=height, y=weight for one line, then x=height, y=age for the next line etc.

Other parameters:

alpha=.7 – This determines how transparent the points in your scatterplot are. 1=Opaque, 0=Invisible, with everything in between possible. If you set it as opaque, it will be impossible to tell that there are 2 points at a position if they are on top of each other. If you set it too low, the points will be hard to see. 0.7 is a good balance, though you may try fine-tuning this number if the result looks bad.

color=k@.Data – This makes it so that your points are coloured based on what cluster they're in.

size=3 – This determines the size of your points. If the points are too big, it covers too much area and makes it impossible to tell where the point's position actually is. If it is too small, it will be hard to see. 3 is a good balance, though if you have a lot of points, you may wish to try a smaller number.

pch=16 – This determines what shape the points are. 16 is a basic full circle, and is the simplest looking shape to use. You may wish to have points in different clusters show up as different shapes, in which case you can change the 16 to k@.Data. However, this may be excessively distracting if your points are already differently coloured.

ncol=3 – This is the number of columns to arrange the graphs in. If you have more graphs you will need to increase this number, though too many graphs do not fit well on one page. You can also use nrow=3 to set the number of rows.

Seeing what's in each cluster

You can list the objects in each cluster with the following code:

```
clust <- names(sort(table(k@.Data)))
clustnumb <- length(clust)
for(i in 1:clustnumb){
  print(c("Cluster number:", clust[i]))
  print(row.names(comp[k@.Data ==clust[i],]))
}
sort(table(k@.Data))
```

It lists the clusters by order of size. Because of this, you must be careful when comparing two different clustering algorithms on the same data. The order of clusters may not be the same between the 2 algorithms.

Parameters you may need to replace:

comp – This will be the name of your data if you followed Chapter 1's guide on pre-processing, otherwise, replace it with the name of your data to be clustered.

Comparing clusters by each original variable

You can see how the individuals in your cluster compare by plotting boxplots for all of your original variables (before doing a PCA). The package **ggplot2** will be used again.

First, you need to add a new column to your data which can be used to separate the individuals into clusters for the boxplots.

```
NAMEOFDATA$Cluster <- k@.Data
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning). Keep the $Cluster part on the end.

Now you can compare the clusters across one variable using the following code:

```
qplot(factor(Cluster), VARIABLE, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE – Replace this with the name of the variable. This is the column heading from your original data you would like to compare the clusters by e.g. height. Note that you need to replace it in 2 places in the code.

If you want to display multiple boxplots on one page, you can use **gridExtra** again.

```
box1 <- qplot(factor(Cluster), VARIABLE1, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE1') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

box2 <- qplot(factor(Cluster), VARIABLE2, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE2') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

box3 <- qplot(factor(Cluster), VARIABLE3, geom = "boxplot", data = NAMEOFDATA,
xlab='Cluster', ylab='VARIABLE3') + geom_boxplot(aes(fill = factor(Cluster))) +
theme(legend.position="NULL")

grid.arrange(box1, box2, box3, ncol=2)
```

Parameters you may need to replace:

NAMEOFDATA – Replace this with the name of your original data (before all pre-processing steps except data cleaning).

VARIABLE1, VARIABLE2, VARIABLE3 etc. – Replace these with the names of the variables i.e. column headings from your original data you would like to compare the clusters by. Note that for each one, there are 2 places in the code to replace them.

Adding extra graphs is easy, just don't forget to add box4, box5 etc. to the grid.arrange list in the last line.