

# 1. Targeting treatment for heart disease patients

Clustering algorithms are used to group together items that are similar to one another. There are many industries where it would be beneficial and insightful to use an unsupervised learning algorithm - retailers want to group similar customers for targeted ad campaigns, biologists want to find plants that share similar characteristics, and more. We are going to explore if it would be appropriate to use some clustering algorithms to group medical patients.



We are going to look at anonymized patients who have been diagnosed with heart disease. Patients with similar characteristics might respond to the same treatments, and doctors would benefit from learning about the outcomes of patients similar to those they are treating. The data we are analyzing comes from the V.A. Medical Center in Long Beach, CA. For more information, see [here](http://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/heart-disease.names) (<http://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/heart-disease.names>), and to download the data visit [here](https://archive.ics.uci.edu/ml/datasets/heart+Disease) (<https://archive.ics.uci.edu/ml/datasets/heart+Disease>).

Before beginning a project, it is important to get an idea of what the patient data looks like. In addition, the clustering algorithms used below require that the data be numeric, so it is necessary to ensure the patient data doesn't need any transformations. You will also be brushing up on your base R skills for some analysis.

In [175]:

```
# Loading the data
heart_disease = read.csv("datasets/heart_disease_patients.csv")

# Print the first ten rows of the data set
head(heart_disease, n = 10)

# Check that only numeric variables
lapply(heart_disease, class)
```

A data.frame: 10 x 12

id	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope
<int>	<int>	<int>	<int>	<int>	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>
1	63	1	1	145	233	1	2	150	0	2.3	3
2	67	1	4	160	286	0	2	108	1	1.5	2
3	67	1	4	120	229	0	2	129	1	2.6	2
4	37	1	3	130	250	0	0	187	0	3.5	3
5	41	0	2	130	204	0	2	172	0	1.4	1
6	56	1	2	120	236	0	0	178	0	0.8	1
7	62	0	4	140	268	0	2	160	0	3.6	3
8	57	0	4	120	354	0	0	163	1	0.6	1
9	63	1	4	130	254	0	2	147	0	1.4	2
10	53	1	4	140	203	1	2	155	1	3.1	3

**\$id**  
'integer'  
**\$age**  
'integer'  
**\$sex**  
'integer'  
**\$cp**  
'integer'  
**\$trestbps**  
'integer'  
**\$chol**  
'integer'  
**\$fbs**  
'integer'  
**\$restecg**  
'integer'  
**\$thalach**  
'integer'  
**\$exang**  
'integer'  
**\$oldpeak**  
'numeric'  
**\$slope**  
'integer'

In [176]:

```
# These packages need to be loaded in the first @tests cell.
library(testthat)
library(IRkernel.testthat)

soln_heart_disease <- read.csv('datasets/heart_disease_patients.csv')

run_tests({
  test_that("heart disease data loaded correctly", {
    expect_equal(heart_disease, soln_heart_disease,
                  info="heart_disease does not have right data, check the csv file name"
    )
    expect_identical(lapply(heart_disease, class)[1][[1]], 'integer',
                      info="variables not all type numeric")
  })
})
```

1/1 tests passed

## 2. Quantifying patient differences

It is important to conduct some exploratory data analysis to familiarize ourselves with the data before clustering. This will help us learn more about the variables and make an informed decision about whether we should scale the data. Because k-means and hierarchical clustering measures similarity between points using a distance formula, it can place extra emphasis on certain variables that have a larger scale and thus larger differences between points.

Exploratory data analysis helps us to understand the characteristics of the patients in the data. We need to get an idea of the value ranges of the variables and their distributions. This will also be helpful when we evaluate the clusters of patients from the algorithms. Are there more patients of one gender? What might an outlier look like?

In [177]:

```
# Evidence that the data should be scaled?
```

```
summary(heart_disease)
```

```
# Remove id
```

```
heart_disease = heart_disease[ , !(names(heart_disease) %in% c("id"))]
```

```
# Scaling data and saving as a data frame
```

```
scaled = scale(heart_disease)
```

```
# What does data look like now?
```

```
summary(scaled)
```

id	age	sex	cp
Min. : 1.0	Min. :29.00	Min. :0.0000	Min. :1.000
1st Qu.: 76.5	1st Qu.:48.00	1st Qu.:0.0000	1st Qu.:3.000
Median :152.0	Median :56.00	Median :1.0000	Median :3.000
Mean :152.0	Mean :54.44	Mean :0.6799	Mean :3.158
3rd Qu.:227.5	3rd Qu.:61.00	3rd Qu.:1.0000	3rd Qu.:4.000
Max. :303.0	Max. :77.00	Max. :1.0000	Max. :4.000
trestbps	chol	fbs	restecg
Min. : 94.0	Min. :126.0	Min. :0.0000	Min. :0.0000
1st Qu.:120.0	1st Qu.:211.0	1st Qu.:0.0000	1st Qu.:0.0000
Median :130.0	Median :241.0	Median :0.0000	Median :1.0000
Mean :131.7	Mean :246.7	Mean :0.1485	Mean :0.9901
3rd Qu.:140.0	3rd Qu.:275.0	3rd Qu.:0.0000	3rd Qu.:2.0000
Max. :200.0	Max. :564.0	Max. :1.0000	Max. :2.0000
thalach	exang	oldpeak	slope
Min. : 71.0	Min. :0.0000	Min. :0.00	Min. :1.000
1st Qu.:133.5	1st Qu.:0.0000	1st Qu.:0.00	1st Qu.:1.000
Median :153.0	Median :0.0000	Median :0.80	Median :2.000
Mean :149.6	Mean :0.3267	Mean :1.04	Mean :1.601
3rd Qu.:166.0	3rd Qu.:1.0000	3rd Qu.:1.60	3rd Qu.:2.000
Max. :202.0	Max. :1.0000	Max. :6.20	Max. :3.000

age	sex	cp	trestbps
Min. :-2.8145	Min. :-1.4549	Min. :-2.2481	Min. :-2.14149
1st Qu.: -0.7124	1st Qu.: -1.4549	1st Qu.: -0.1650	1st Qu.: -0.66420
Median : 0.1727	Median : 0.6851	Median : -0.1650	Median : -0.09601
Mean : 0.0000	Mean : 0.0000	Mean : 0.0000	Mean : 0.00000
3rd Qu.: 0.7259	3rd Qu.: 0.6851	3rd Qu.: 0.8765	3rd Qu.: 0.47218
Max. : 2.4961	Max. : 0.6851	Max. : 0.8765	Max. : 3.88132
chol	fbs	restecg	thalach
Min. :-2.3310	Min. :-0.4169	Min. :-0.995103	Min. :-3.4364
1st Qu.: -0.6894	1st Qu.: -0.4169	1st Qu.: -0.995103	1st Qu.: -0.7041
Median : -0.1100	Median : -0.4169	Median : 0.009951	Median : 0.1483
Mean : 0.0000	Mean : 0.0000	Mean : 0.000000	Mean : 0.0000
3rd Qu.: 0.5467	3rd Qu.: -0.4169	3rd Qu.: 1.015005	3rd Qu.: 0.7166
Max. : 6.1283	Max. : 2.3905	Max. : 1.015005	Max. : 2.2904
exang	oldpeak	slope	
Min. :-0.6955	Min. :-0.8954	Min. :-0.9747	
1st Qu.: -0.6955	1st Qu.: -0.8954	1st Qu.: -0.9747	
Median : -0.6955	Median : -0.2064	Median : 0.6480	
Mean : 0.0000	Mean : 0.0000	Mean : 0.0000	
3rd Qu.: 1.4331	3rd Qu.: 0.4827	3rd Qu.: 0.6480	
Max. : 1.4331	Max. : 4.4445	Max. : 2.2708	

In [178]:

```

soln_heart_disease = soln_heart_disease[ , !(names(soln_heart_disease) %in% c("id"))]
soln_scaled = scale(soln_heart_disease)

run_tests({
  test_that("remove correct column", {
    expect_identical(colnames(soln_heart_disease), colnames(heart_disease), info =
"Did you remove the id column?")
  })

  test_that("scaled data properly", {
    expect_identical(soln_scaled, scaled, info = "Did you scale the proper data se
t?")
  })
})

```

2/2 tests passed

### 3. Let's start grouping patients

Once we've figured out if we need to modify the data and made any necessary changes, we can now start the clustering process. For the k-means algorithm, it is necessary to select the number of clusters in advance.

It is also important to make sure that your results are reproducible when conducting a statistical analysis. This means that when someone runs your code on the same data, they will get the same results as you reported. Therefore, if you're conducting an analysis that has a random aspect, it is necessary to set a seed to ensure reproducibility.

Reproducibility is especially important since doctors will potentially be using our results to treat patients. It is vital that another analyst can see where the groups come from and be able to verify the results.

In [179]:

```

# Set the seed so that results are reproducible
seed_val = 10
set.seed(seed_val, kind = "Mersenne-Twister", normal.kind = "Inversion")

# Select a number of clusters
k = 5

# Run the k-means algorithms
first_clust = kmeans(scaled, centers = k, nstart = 1)

# How many patients are in each group?
first_clust$size

```

24 53 87 72 67

In [180]:

```

soln_seed_val = 10
set.seed(soln_seed_val, kind = "Mersenne-Twister", normal.kind = "Inversion")
soln_k = 5
soln_first_clust = kmeans(soln_scaled, centers = soln_k, nstart = 1)

run_tests({
  test_that("correct seed", {
    expect_equal(soln_seed_val, seed_val, info = "Is the seed set to 10?")
  })

  test_that("correct number of clusters", {
    expect_equal(soln_k, k, info = "Are you using five clusters?")
  })

  test_that("correct implementation of algorithm", {
    expect_equal(soln_first_clust$size, first_clust$size, info = "What is your nstart value?")
  })
})

```

3/3 tests passed

## 4. Another round of k-means

Because the k-means algorithm initially selects the cluster centers by randomly selecting points, different iterations of the algorithm can result in different clusters being created. If the algorithm is truly grouping together similar observations (as opposed to clustering noise), then cluster assignments will be somewhat robust between different iterations of the algorithm.

With regards to the heart disease data, this would mean that the same patients would be grouped together even when the algorithm is initialized at different random points. If patients are not in similar clusters with various algorithm runs, then the clustering method isn't picking up on meaningful relationships between patients.

We're going to explore how the patients are grouped together with another iteration of the k-means algorithm. We will then be able to compare the resulting groups of patients.

In [181]:

```

# Set the seed
seed_val = 38
set.seed(seed_val, kind = "Mersenne-Twister", normal.kind = "Inversion")

# Run the k-means algorithms
k = 5
second_clust = kmeans(scaled, centers = k, nstart = 1)

# How many patients are in each group?
second_clust$size

```

51 48 42 67 95

In [182]:

```
seed_val_2 = 38
set.seed(seed_val_2, kind = "Mersenne-Twister", normal.kind = "Inversion")
k_2 = 5
soln_second_clust = kmeans(soln_scaled, centers = k_2, nstart = 1)

run_tests({
  test_that("correct seed", {
    expect_equal(seed_val_2, seed_val, info = "Is the seed set to 10?")
  })

  test_that("correct number of clusters", {
    expect_equal(k_2, k, info = "Are you using five clusters?")
  })

  test_that("correct implmentation of algorithm", {
    expect_equal(soln_second_clust$size, second_clust$size, info = "What is your ns
tart value?")
  })
})
```

3/3 tests passed

## 5. Comparing patient clusters

It is important that the clusters resulting from the k-means algorithm are stable. Even though the algorithm begins by randomly initializing the cluster centers, if the k-means algorithm is the right choice for the data, then different initializations of the algorithm will result in similar clusters.

The clusters from different iterations may not be exactly the same, but the clusters should be roughly the same size and have similar distributions of variables. If there is a lot of change in clusters between different iterations of the algorithm, then k-means clustering is not a good choice for the data.

It is not possible to validate that the clusters obtained from an algorithm are ground truth are accurate since there is no true labeling for patients. Thus, it is necessary to examine how the clusters change between different iterations of the algorithm. We're going to use some visualizations to get an idea of the cluster stabilities. That way we can see how certain patient characteristics may have been used to group patients together.



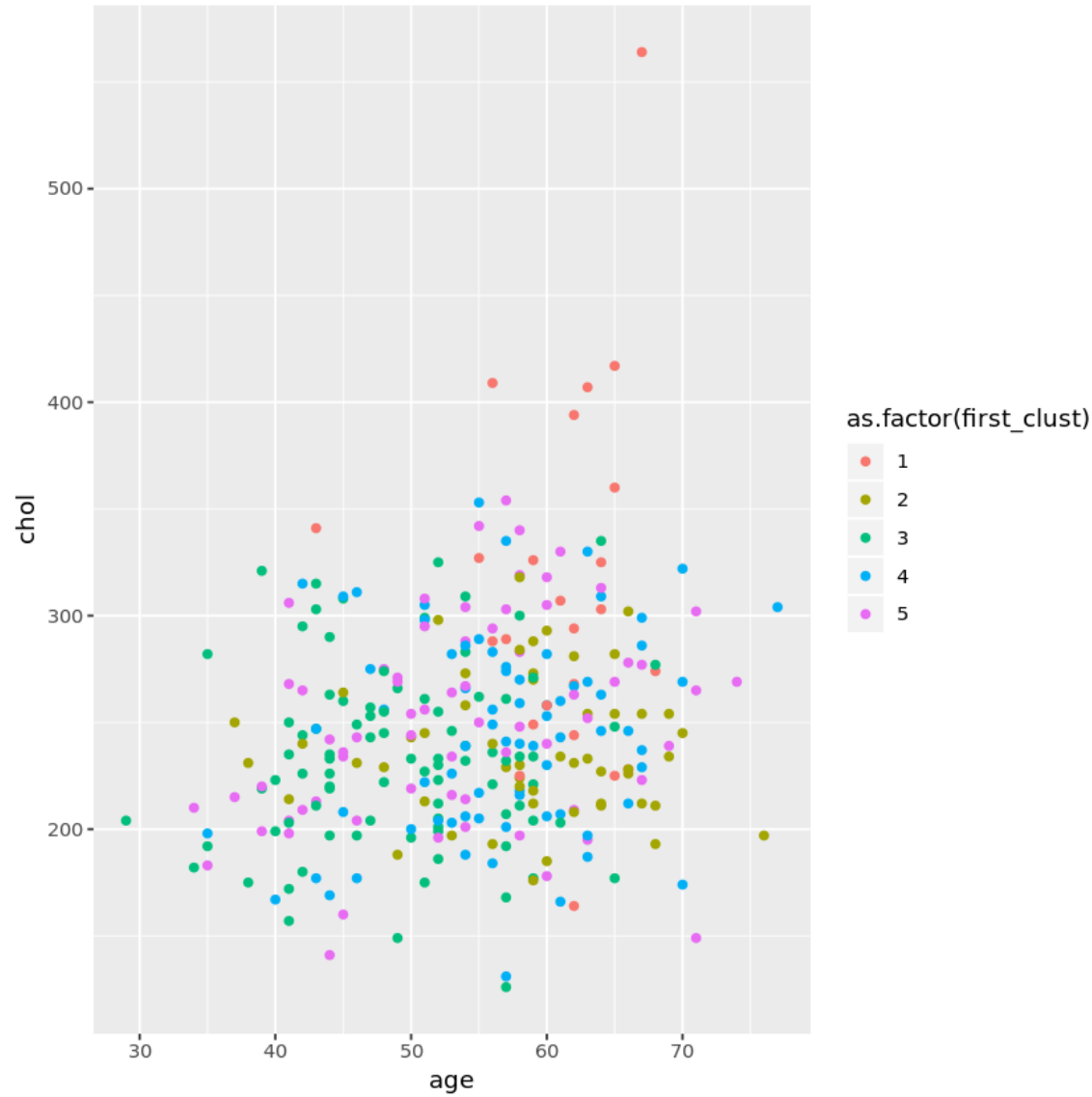
In [183]:

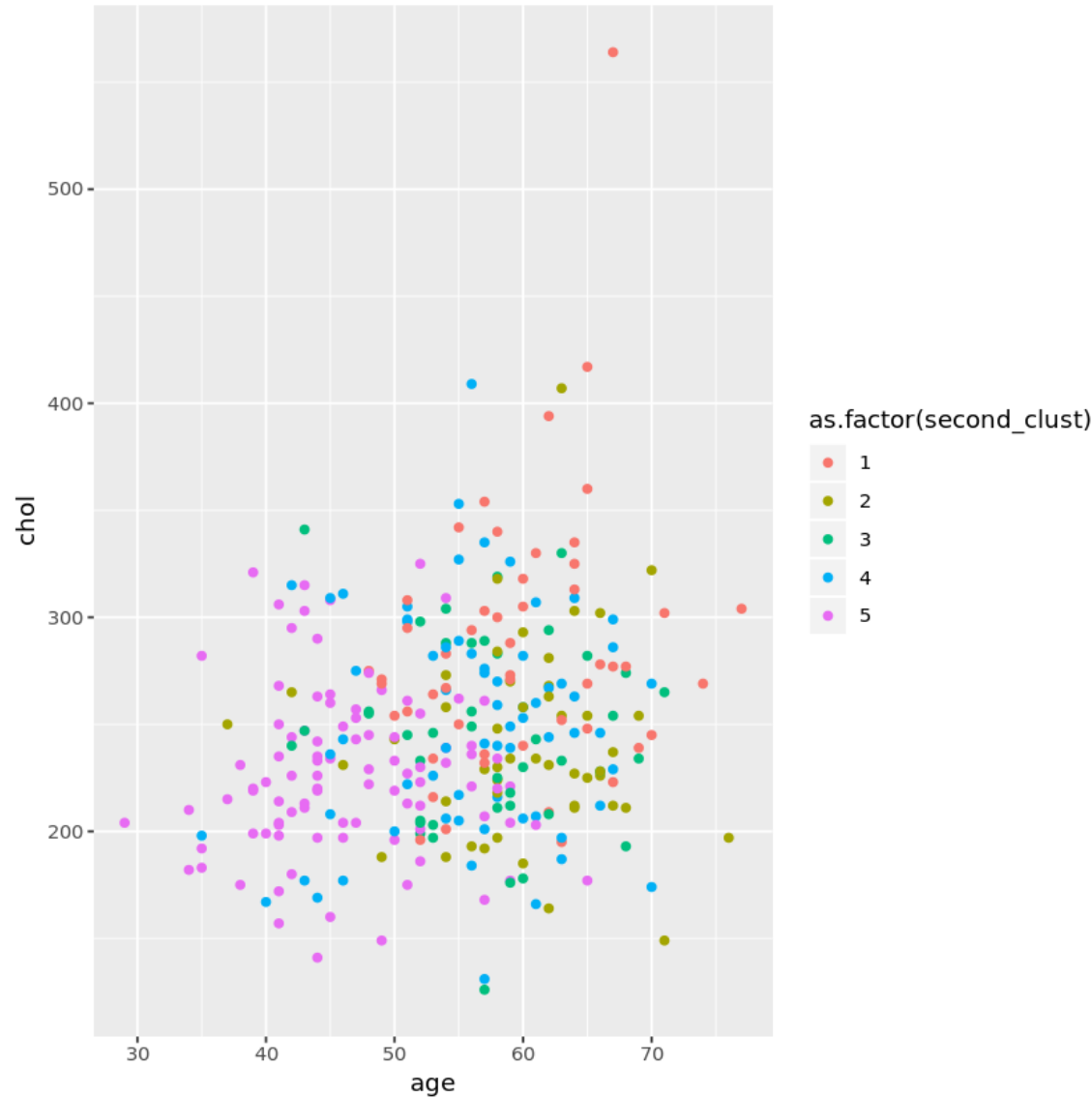
```
# Adding cluster assignments to the data
heart_disease[,c("first_clust")] = first_clust$cluster
heart_disease[,c("second_clust")] = second_clust$cluster

# Load ggplot2
library(ggplot2)

# Creating the plots of age and chol for the first clustering algorithm
plot_one = ggplot(heart_disease, aes(x = age, y = chol, color = as.factor(first_clust))) +
  geom_point()
plot_one

# Creating the plots of age and chol for the second clustering algorithm
plot_two = ggplot(heart_disease, aes(x = age, y = chol, color = as.factor(second_clust))) +
  geom_point()
plot_two
```





In [184]:

```
soln_heart_disease['first_clust'] = soln_first_clust$cluster
soln_heart_disease['second_clust'] = soln_second_clust$cluster

# creating the correct graphs and getting fingerprints
soln_plot_one = ggplot(soln_heart_disease, aes(x=age, y=chol, color=as.factor(first_clust))) + geom_point()
soln_plot_two = ggplot(soln_heart_disease, aes(x=age, y=chol, color=as.factor(second_clust))) + geom_point()

run_tests({
  test_that("cluster assignments added", {
    expect_identical(soln_heart_disease, heart_disease,
      info = "Did you add a column for both the first and second iteration?")
  })

  test_that("ggplot2 loaded", {
    expect_true('ggplot2' %in% .packages(),
      info = "Did you load ggplot2?")
  })

  test_that("first plot is correct", {
    expect_equal(soln_plot_one$labels, plot_one$labels,
      info = "Do you have the correct variables on the axes and used to color code?")
  })

  test_that("second plot is correct", {
    expect_equal(soln_plot_two$labels, plot_two$labels,
      info = "Do you have the correct variables on the axes and used to color code?")
  })
})
```

4/4 tests passed

## 6. Hierarchical clustering: another clustering approach

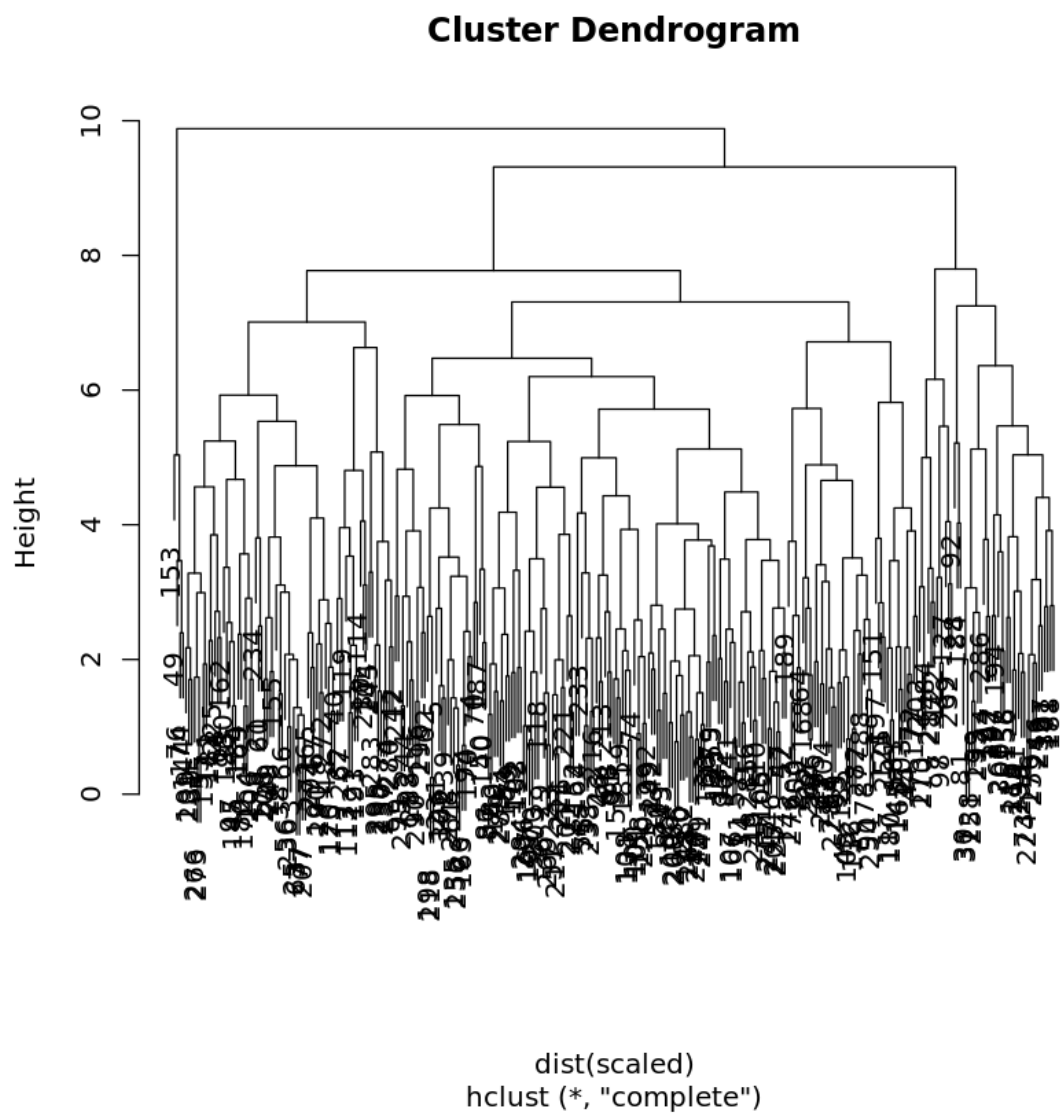
An alternative to k-means clustering is hierarchical clustering. This method works well when the data has a nested structure. It is possible that the data from heart disease patients follows this type of structure. For example, if men are more likely to exhibit certain characteristics, those characteristics might be nested inside the gender variable. Hierarchical clustering also does not require the number of clusters to be selected prior to running the algorithm.

Clusters can be selected by using the dendrogram. The dendrogram allows one to see how similar observations are to one another and are useful in selecting the number of clusters to group the data. It is now time for us to see how hierarchical clustering groups the data.

```
# Executing hierarchical clustering with complete linkage
hier_clust_1 = hclust(dist(scaled), method= "complete")

# Printing the dendrogram
plot(hier_clust_1)

# Getting cluster assignments based on number of selected clusters
hc_1_assign <- cutree(hier_clust_1, k = 5)
```



In [186]:

```
soln_hier_clust_1 = hclust(dist(soln_scaled), method='complete')
soln_hc_1_assign = cutree(soln_hier_clust_1, 5)

run_tests({
  test_that("correctly implemented clustering algorithm", {
    expect_identical(soln_hier_clust_1$merge, hier_clust_1$merge,
      info = "Did you make the distance matrix?")
    expect_identical(soln_hier_clust_1$labels, hier_clust_1$labels,
      info = "Did you make the distance matrix?")
    expect_identical(soln_hier_clust_1$method, hier_clust_1$method,
      info = "Did you use complete linkage?")

  })

  test_that("correct cutoff for cluster assignments", {
    expect_identical(soln_hc_1_assign, hc_1_assign,
      info = "Did you select five clusters?")
  })
})
```

2/2 tests passed

## 7. Hierarchical clustering round two

In hierarchical clustering, there are multiple ways to measure the dissimilarity between clusters of observations. Complete linkage records the largest dissimilarity between any two points in the two clusters being compared. On the other hand, single linkage is the smallest dissimilarity between any two points in the clusters. Different linkages will result in different clusters being formed.

We want to explore different algorithms to group our heart disease patients. The best way to measure dissimilarity between patients could be to look at the smallest difference between patients and minimize that difference when grouping together clusters. It is always a good idea to explore different dissimilarity measures. Let's implement hierarchical clustering using a new linkage function.

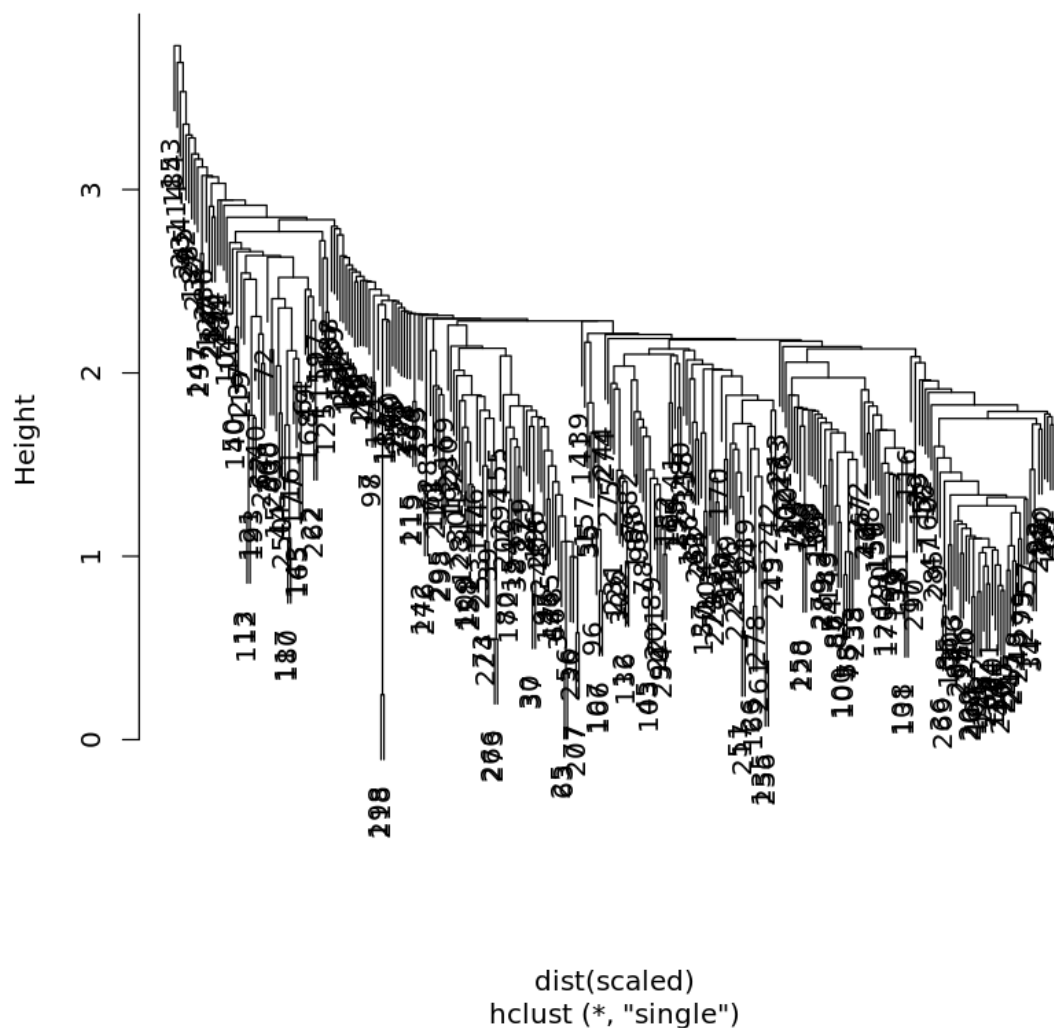
In [187]:

```
# Executing hierarchical clustering with single linkage
hier_clust_2 = hclust(dist(scaled), method = "single")

# Printing the dendrogram
plot(hier_clust_2)

# Getting cluster assignments based on number of selected clusters
hc_2_assign <- cutree(hier_clust_2, k = 5)
```

### Cluster Dendrogram



In [188]:

```
soln_hier_clust_2 = hclust(dist(soln_scaled), method='single')
soln_hc_2_assign = cutree(soln_hier_clust_2, 5)

run_tests({
  test_that("correctly implemented clustering algorithm", {
    expect_identical(soln_hier_clust_2$merge, hier_clust_2$merge,
      info = "Did you make the distance matrix?")
    expect_identical(soln_hier_clust_2$labels, hier_clust_2$labels,
      info = "Did you make the distance matrix?")
    expect_identical(soln_hier_clust_2$method, hier_clust_2$method,
      info = "Did you use single linkage?") })

  test_that("correct cutoff for cluster assignments", {
    expect_identical(soln_hc_2_assign, hc_2_assign, info = "Did you select five clusters?")
  })
})
```

2/2 tests passed

## 8. Comparing clustering results

The doctors are interested in grouping similar patients together in order to determine appropriate treatments. Therefore, they want to have clusters with more than a few patients to see different treatment options. While it is possible for a patient to be in a cluster by themselves, this means that the treatment they received might not be recommended for someone else in the group.

As with the k-means algorithm, the way to evaluate the clusters is to investigate which patients are being grouped together. Are there patterns evident in the cluster assignments or do they seem to be groups of noise? We're going to examine the clusters resulting from the two hierarchical algorithms.



In [189]:

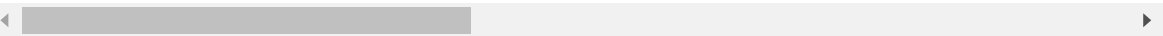
```
# Adding assignments of chosen hierarchical linkage
heart_disease['hc_clust'] = hc_1_assign

# Remove 'sex', 'first_clust', and 'second_clust' variables
hd_simple = heart_disease[, !(names(heart_disease) %in% c("sex", "first_clust", "second_clust"))]

# Getting mean and standard deviation summary statistics
clust_summary = do.call(data.frame, aggregate(. ~ hc_clust, data = hd_simple, function(x) c(avg = mean(x), sd = sd(x))))
clust_summary
```

A data.frame: 5 x 21

hc_clust	age.avg	age.sd	cp.avg	cp.sd	trestbps.avg	trestbps.sd	chol.avg	chol
<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<d
1	51.41667	8.540979	2.783333	0.9470625	129.1389	15.93800	239.8722	42.29
2	58.11111	7.754246	3.763889	0.6165112	130.0417	13.90657	253.2222	49.74
3	61.00000	3.908034	3.916667	0.2886751	168.5000	17.45904	284.9167	53.00
4	59.00000	9.203580	3.571429	0.8501112	134.7714	18.64070	233.8571	49.67
5	64.75000	2.061553	3.250000	0.5000000	138.7500	18.42779	433.7500	89.93



In [190]:

```

soln_heart_disease["hc_clust"] = soln_hc_1_assign

soln_hd_simple = soln_heart_disease[, !(names(soln_heart_disease) %in% c("sex", "first_
clust", "second_clust"))]

soln_clust_summary = do.call(data.frame, aggregate(. ~hc_clust, data = soln_hd_simple,
function(x) c(avg = mean(x), sd = sd(x))))

run_tests({
  test_that("selected first cluster assignments", {
    expect_identical(heart_disease['hc_clust'], soln_heart_disease['hc_clust'],
info = "You chose the incorrect hierarchical clustering assign
ments.")
  })

  test_that("removed columns properly", {
    expect_identical(soln_hd_simple, hd_simple,
info = "Did you remove three columns?")
  })

  test_that("proper summary analysis", {
    expect_identical(soln_clust_summary, clust_summary,
info = "Did you find the mean and standard deviation using mea
n(x) and sd(x)?")
  })
})

```

3/3 tests passed

## 9. Visualizing the cluster contents

In addition to looking at the distributions of variables in each of the hierarchical clustering runs, we will make visualizations to evaluate the algorithms. Even though the data has more than two dimensions, we can get an idea of how the data clusters by looking at a scatter plot of two variables. We want to look for patterns that appear in the data and see what patients get clustered together.

In [ ]:

```

# Plotting age and chol
plot_one = ggplot(heart_disease, aes(x = age, y = chol, color = as.factor(hc_clust))) +
  geom_point()
plot_one

# Plotting oldpeak and trestbps
plot_two = ggplot(heart_disease, aes(x = oldpeak, y = trestbps, color = as.factor(hc_cl
ust))) +
  geom_point()
plot_two

```

In [ ]:

```
soln_plot_one = ggplot(soln_heart_disease, aes(x=age, y=chol, color=as.factor(hc_clust))) + geom_point()
soln_plot_two = ggplot(soln_heart_disease, aes(x=oldpeak, y=trestbps, color=as.factor(hc_clust))) + geom_point()
run_tests({
  test_that("plot one is correct", {
    expect_identical(soln_plot_one$labels, plot_one$labels,
      info = "Check that you are using the correct variables for the
first plot")
  })

  test_that("plot two is correct", {
    expect_identical(soln_plot_two$labels, plot_two$labels,
      info = "Check that you are using the correct variables for the
second plot")
  })
})
```

## 10. Conclusion

Now that we've tried out multiple clustering algorithms, it is necessary to determine if we think any of them will work for clustering our patients. For the k-means algorithm, it is imperative that similar clusters are produced for each iteration of the algorithm. We want to make sure that the algorithm is clustering signal as opposed to noise.

For the sake of the doctors, we also want to have multiple patients in each group so they can compare treatments. We only did some preliminary work to explore the performance of the algorithms, and it is necessary to explore further before making a recommendation. Based on the above analysis are there any algorithms that you would want to investigate further to group patients?

Remember that it is important the k-mean algorithm seems stable when running multiple iterations. This means that we would see similar groups of patients showing up in the plots from the different iterations of the algorithm. For the hierarchical clustering, we need a method that puts a balanced number of patients in each group.

In [ ]:

```
# Add TRUE if the algorithm shows promise, add FALSE if it does not
explore_kmeans = FALSE
explore_hierarch_complete = TRUE
explore_hierarch_single = FALSE
```

In [ ]:

```
soln_1 = FALSE
soln_2 = TRUE
soln_3 = FALSE

run_tests({
  test_that("correct kmeans results", {
    expect_identical(soln_1, explore_kmeans, info = "Are the clusters stable between kmeans iterations?")
  })

  test_that("correct hierarchical with complete linkage results", {
    expect_identical(soln_2, explore_hierarch_complete, info = "Would you want to explore this method further?")
  })

  test_that("correct hierarchical with single linkage results", {
    expect_identical(soln_3, explore_hierarch_single, info = "Is the number of patients in each cluster balanced?")
  })
})
```