Assignment 8

Instructions: Make an R Notebook and in it answer the question or questions below. When you are done, hand in on Quercus the *output* from Previewing (or Knitting) your Notebook, probably an `html` or `pdf` file. An `html` file is easier for the grader to deal with. Do *not* hand in the Notebook itself. You want to show that you can (i) write code that will answer the questions, (ii) run that code and get some sensible output, (iii) write some words that show you know what is going on and that reflect your conclusions about the data. Your goal is to convince the grader that you *understand* what you are doing: not only doing the right thing, but making it clear that you know *why* it's the right thing.

Do *not* expect to get help on this assignment. The purpose of the assignments is for you to see how much *you* have understood. You will find that you also learn something from grappling with the assignments. The time to get help is after you watch the lectures and work through the problems from PASIAS, via tutorial and the discussion board, that is *before* you start work on the assignment. The only reasons to contact the instructor while working on an assignment are to report (i) something missing like a data file that cannot possibly be read, (ii) something *beyond your control* that makes it impossible to finish the assignment in time after you have started it.

There is a time limit on this assignment (you will see Quercus counting down the time remaining).

1. The beetle *Araptus attenatus* lives in the Sonora desert. For each of 39 populations of these beetles, the average genetic distance (a kind of dissimilarity between the genetic material of living things) was calculated from each other population. We will cluster the different populations, with the aim of seeing whether populations that live in different locations are genetically distinct. The data are in http://ritsokiguess.site/STAD29/arapat.csv. The populations are labelled by numbers or letters.

   (a) Read in and display some of the data.

Solution:

The usual:

```
my_url <- "http://ritsokiguess.site/STAD29/arapat.csv"
arapat <- read_csv(my_url)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double(),
##   population = col_character()
## )
## i Use `spec()` for the full column specifications.
```

```
arapat
```

```
## # A tibble: 39 x 40
##    population `101` `102`  `12` `153` `156` `157` `159` `160` `161` `162` `163`
##    <chr>      <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 101         0     4.27  6.44  6.21  8.27  7.27  6.21  6.48  6.25  6.00  8.50
##  2 102         4.27  0     7.83  7.64  9.11  8.25  7.52  7.86  7.67  7.40  9.34
```

```
##  3 12          6.44  7.83  0     3.12  8.40  7.14  4.55  3.11  2.39  2.77  8.10
##  4 153         6.21  7.64  3.12  0     8.25  7.01  5.21  3.29  3.11  2.91  8.13
##  5 156         8.27  9.11  8.40  8.25  0     3.90  8.22  8.41  8.30  8.11  8.84
##  6 157         7.27  8.25  7.14  7.01  3.90  0     7.05  7.12  7.07  6.89  7.28
##  7 159         6.21  7.52  4.55  5.21  8.22  7.05  0     4.52  4.37  4.41  8.13
##  8 160         6.48  7.86  3.11  3.29  8.41  7.12  4.52  0     2.51  2.94  8.08
##  9 161         6.25  7.67  2.39  3.11  8.30  7.07  4.37  2.51  0     1.88  8.13
## 10 162         6.00  7.40  2.77  2.91  8.11  6.89  4.41  2.94  1.88  0     8.04
## # ... with 29 more rows, and 28 more variables: 164 <dbl>, 165 <dbl>,
## #   166 <dbl>, 168 <dbl>, 169 <dbl>, 171 <dbl>, 173 <dbl>, 175 <dbl>,
## #   177 <dbl>, 32 <dbl>, 48 <dbl>, 51 <dbl>, 58 <dbl>, 64 <dbl>, 73 <dbl>,
## #   75 <dbl>, 77 <dbl>, 84 <dbl>, 88 <dbl>, 89 <dbl>, 9 <dbl>, 93 <dbl>,
## #   98 <dbl>, Aqu <dbl>, Const <dbl>, ESan <dbl>, Mat <dbl>, SFr <dbl>
```

If you want to say so, you might note that the values you can see look like distances in that eg. the value for 12 (row) and 102 (column) is the same as for 102 (row) and 12 (column). Note also that the column names that are numbers have (at least on mine) backticks around them, because numbers are not legal column names.

There is a long story about how I got hold of these, but the short version is that they came from here. (The data actually came from a principal component analysis.)

(b) Carry out a hierarchical cluster analysis of the populations, using Ward's method, and display a dendrogram.

Solution:

These are already dissimilarities (distances), and so the issue is to get them into the form of a `dist` object, which is `as.dist` (as in the lecture notes):

```
arapat %>% select(-population) %>%
  as.dist() -> d
```

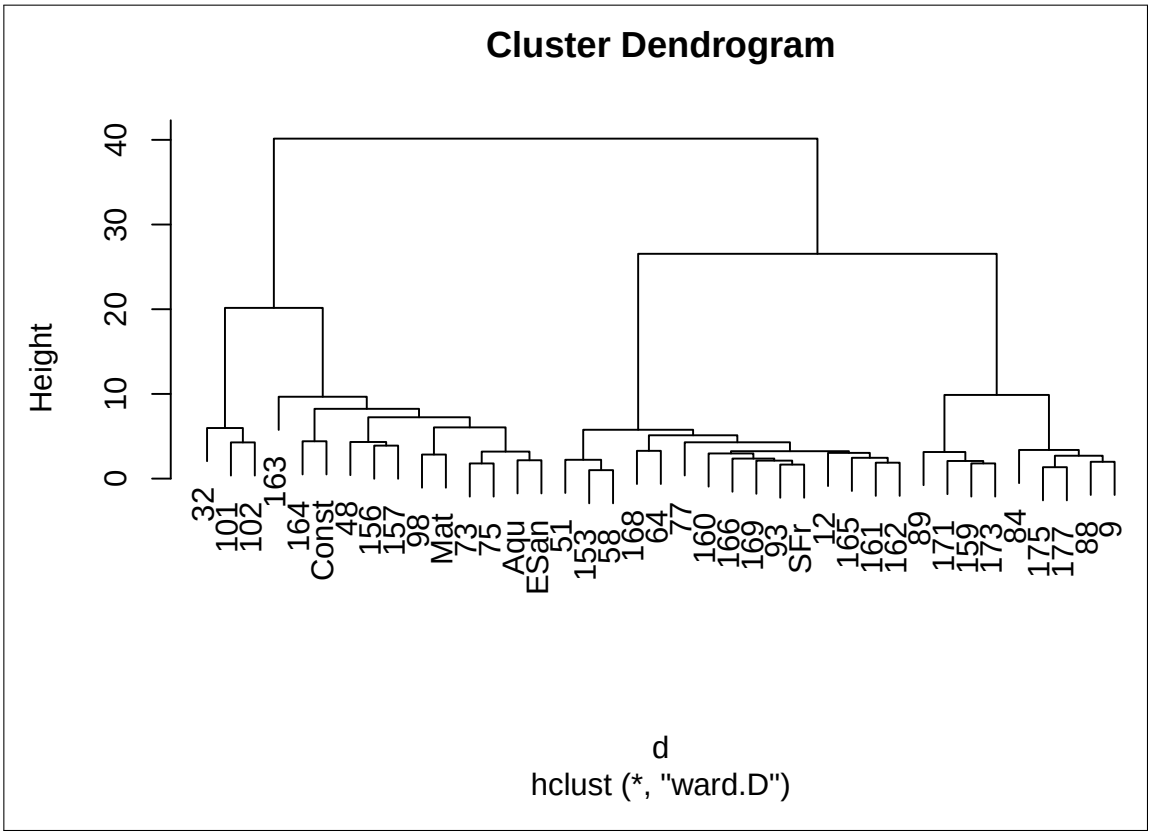
Don't display `d` as it is huge!

Then feed `d` into `hclust`:

```
arapat.1 <- hclust(d, method = "ward.D")
```

and `plot` the result to get a dendrogram:

```
plot(arapat.1)
```

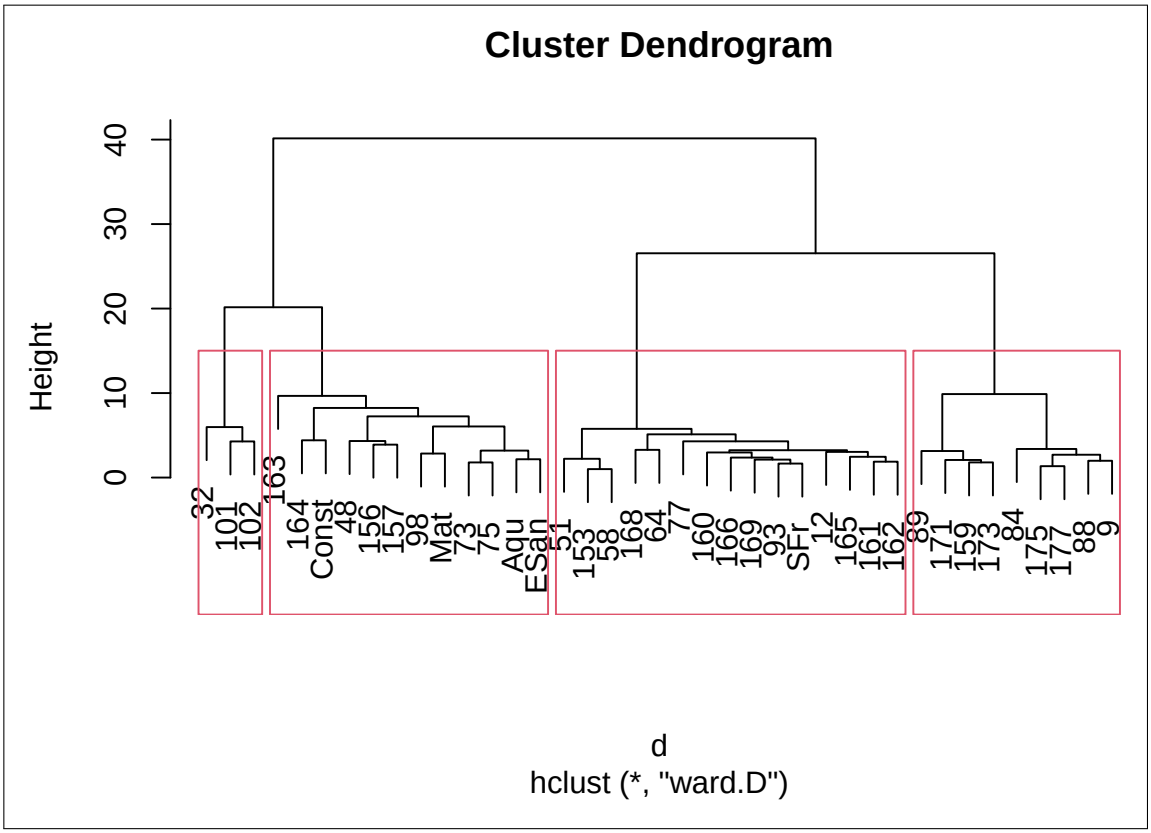


(c) Choose a suitable number of clusters and display them on your dendrogram.

Solution:

It looks to me as if four clusters will offer some insight. You'll probably need to draw the graph again (or else you will get some message about `plot.new` not having been called yet):

```
plot(arapat.1)
rect.hclust(arapat.1, 4)
```



(d) The populations labelled 32, 101, 102 are located on the mainland. The populations 98, Mat, 157, 73, 75, Aqu, ESan, 156, and 48 are located on the southern part of Baja California. Is there any evidence that these two sets of populations are genetically distinct from the rest, based on your dendrogram? Discuss briefly.

Solution:

Populations 32, 101, 102 make up the cluster on the left of my dendrogram, so they are definitely genetically distinct. The other populations mentioned are all in the second of my clusters, but there are some other populations there as well (163, 164, Const). So these populations are not quite as distinct as 32, 101, 102 are. Something like that. (Get the idea of "distinct but not completely distinct" across somehow.)

2. Canberra is the capital city of Australia. The data in http://ritsokiguess.site/STAD29/canberra.csv are of 17 quantitative environmental variables summarizing the weather on each of 354 days during 2007 and 2008. (The original data set has 366 rows, but some of them had missing values.) Each variable has been standardized, so you can read the values as if they were z-scores.

   Here are the variables, and their units before they were standardized:

   - `MinTemp` minimum temperature on that day (all temperatures in degrees Celsius)
   - `MaxTemp` maximum temperature
   - `Rainfall` in mm
   - `Evaporation` Class A pan evaporation, mm.
   - `Sunshine` hours of bright sunshine
   - `WindGustSpeed` speed of strongest wind gust, km/h

- `Temp9am` temperature at 9:00am
- `RelHumid9am` relative humidity at 9:00am (percent)
- `Cloud9am` fraction of sky covered by cloud at 9:00am
- `WindSpeed9am` average wind speed over 10 minutes before 9:00am
- `Pressure9am` atmospheric pressure at 9:00am (hectopascals)
- `Temp3pm`, `RelHumid3pm`, `Cloud3pm`, `WindSpeed3pm`, `Pressure3pm`: as above but for 3:00pm

(a) Read in and display (some of) the data.

Solution:

```
my_url <- "http://ritsokiguess.site/STAD29/canberra.csv"
weather <- read_csv(my_url)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   Date = col_date(format = ""),
##   MinTemp = col_double(),
##   MaxTemp = col_double(),
##   Rainfall = col_double(),
##   Evaporation = col_double(),
##   Sunshine = col_double(),
##   WindGustSpeed = col_double(),
##   WindSpeed9am = col_double(),
##   WindSpeed3pm = col_double(),
##   Humidity9am = col_double(),
##   Humidity3pm = col_double(),
##   Pressure9am = col_double(),
##   Pressure3pm = col_double(),
##   Cloud9am = col_double(),
##   Cloud3pm = col_double(),
##   Temp9am = col_double(),
##   Temp3pm = col_double()
## )
```

```
weather
```

```
## # A tibble: 354 x 17
##    Date       MinTemp MaxTemp Rainfall Evaporation Sunshine WindGustSpeed
##    <date>       <dbl>   <dbl>    <dbl>       <dbl>    <dbl>         <dbl>
##  1 2007-11-01  0.106    0.551   -0.335     -0.434   -0.463        -0.768
##  2 2007-11-02  1.10     0.939    0.515     -0.0593   0.506        -0.0776
##  3 2007-11-03  1.05     0.417    0.515      0.465   -1.32          3.45
##  4 2007-11-04  0.988   -0.760    9.06       0.990    0.335         1.07
##  5 2007-11-05  0.0395  -0.671    0.326      0.391    0.762         0.766
##  6 2007-11-06 -0.193   -0.552   -0.335      0.465    0.0782        0.306
##  7 2007-11-07 -0.210   -0.358   -0.288     -0.134    0.135         0.229
##  8 2007-11-08  0.156   -0.537   -0.335      0.391   -0.947         0.0759
##  9 2007-11-09  0.239   -0.164   -0.335     -0.209   -1.09          0.613
## 10 2007-11-10  0.173    0.328    3.49       0.316   -0.0642       -0.691
## # ... with 344 more rows, and 10 more variables: WindSpeed9am <dbl>,
## #   WindSpeed3pm <dbl>, Humidity9am <dbl>, Humidity3pm <dbl>,
## #   Pressure9am <dbl>, Pressure3pm <dbl>, Cloud9am <dbl>, Cloud3pm <dbl>,
```

```
## #   Temp9am <dbl>, Temp3pm <dbl>
```

These look like z-scores. Some of the distributions are actually skewed to the right (like `Rainfall`), so their values go more positive than they go negative.

Extra: there is a (shortish) story about arranging this data set for you. It comes originally from the `rattle` package, as data set `weather`. Using the two colons in the name means "the thing called `weather` that lives in the package `rattle`" and saves me having to say `library(rattle)`, since I also called *my* dataframe `weather`:

```
rattle::weather
```

```
## # A tibble: 366 x 24
##    Date       Location MinTemp MaxTemp Rainfall Evaporation Sunshine WindGustDir
##    <date>     <chr>       <dbl>   <dbl>    <dbl>       <dbl>    <dbl> <ord>
##  1 2007-11-01 Canberra     8      24.3      0           3.4      6.3 NW
##  2 2007-11-02 Canberra    14      26.9      3.6         4.4      9.7 ENE
##  3 2007-11-03 Canberra    13.7    23.4      3.6         5.8      3.3 NW
##  4 2007-11-04 Canberra    13.3    15.5     39.8         7.2      9.1 NW
##  5 2007-11-05 Canberra     7.6    16.1      2.8         5.6     10.6 SSE
##  6 2007-11-06 Canberra     6.2    16.9      0           5.8      8.2 SE
##  7 2007-11-07 Canberra     6.1    18.2      0.2         4.2      8.4 SE
##  8 2007-11-08 Canberra     8.3    17        0           5.6      4.6 E
##  9 2007-11-09 Canberra     8.8    19.5      0           4        4.1 S
## 10 2007-11-10 Canberra     8.4    22.8     16.2         5.4      7.7 E
## # ... with 356 more rows, and 16 more variables: WindGustSpeed <dbl>,
## #   WindDir9am <ord>, WindDir3pm <ord>, WindSpeed9am <dbl>, WindSpeed3pm <dbl>,
## #   Humidity9am <int>, Humidity3pm <int>, Pressure9am <dbl>, Pressure3pm <dbl>,
## #   Cloud9am <int>, Cloud3pm <int>, Temp9am <dbl>, Temp3pm <dbl>,
## #   RainToday <fct>, RISK_MM <dbl>, RainTomorrow <fct>
```

It goes from November 2007 to the end of October 2008, and has 366 rows because 2008 was a leap year. We only want the date and most of the quantitative columns for a cluster analysis (there are some others). Here's how I did that:

```
rattle::weather %>%
  select(Date, where(~is.numeric(.)), -RISK_MM) %>%
  drop_na() %>%
  mutate(across(-Date, ~scale(.))) -> weather0
weather0
```

```
## # A tibble: 354 x 17
##    Date       MinTemp[,1] MaxTemp[,1] Rainfall[,1] Evaporation[,1] Sunshine[,1]
##    <date>           <dbl>       <dbl>        <dbl>           <dbl>        <dbl>
##  1 2007-11-01      0.106        0.551       -0.335         -0.434       -
0.463
##  2 2007-11-02      1.10         0.939        0.515         -0.0593       0.506
##  3 2007-11-03      1.05         0.417        0.515          0.465       -
1.32
##  4 2007-11-04      0.988       -0.760        9.06           0.990        0.335
##  5 2007-11-05      0.0395      -0.671        0.326          0.391        0.762
##  6 2007-11-06     -0.193       -0.552       -0.335          0.465        0.0782
##  7 2007-11-07     -0.210       -0.358       -0.288         -0.134        0.135
##  8 2007-11-08      0.156       -0.537       -0.335          0.391       -
```

```
0.947
##  9 2007-11-09      0.239       -0.164       -0.335         -0.209       -
1.09
## 10 2007-11-10      0.173        0.328        3.49           0.316       -
0.0642
## # ... with 344 more rows, and 11 more variables: WindGustSpeed <dbl[,1]>,
## #   WindSpeed9am <dbl[,1]>, WindSpeed3pm <dbl[,1]>, Humidity9am <dbl[,1]>,
## #   Humidity3pm <dbl[,1]>, Pressure9am <dbl[,1]>, Pressure3pm <dbl[,1]>,
## #   Cloud9am <dbl[,1]>, Cloud3pm <dbl[,1]>, Temp9am <dbl[,1]>,
## #   Temp3pm <dbl[,1]>
```

```
write_csv(weather0, "canberra.csv")
```

On mine, this looks strange, but if you do it, it should look all right. There are 354 rows left, because some rows had missing values on the columns we wanted to keep, so I dropped those rows.

First, I grab the columns I want out of this data set. This is a select-helper you might not have seen before; `where` with something inside it that is true or false will select the columns where the thing is true. Read this as something like "for each column, select it if `is.numeric` of it is true" (that is, if it is quantitative). I also want you to have the dates, so I'm keeping them too. There is a quantitative column called `RISK_MM` that appears to be the *next* day's rain that I don't want, so I get rid of it here. Next, there are a few missing values, so get rid of them. Last, I wanted to standardize all the quantitative columns, so that you could focus on the cluster analysis. The logic of this thing with `across` is "for each column that is not `Date`, standardize it". `across` is a good way of doing something with several columns at once. You will sometimes see `mutate_at`, which is an older way to accomplish the same thing.[1]

(b) Obtain a scree plot for a K-means cluster analysis. You can re-use ideas from my lecture notes if they help you.

Solution:

For a scree plot, you need to plot the total within-group sum of squares against each number of clusters. So you need to obtain that first. The way I did it, which you can copy, is to write a function to get the total within-group sum of squares for each number of clusters from a dataframe. This is the one in the notes:

```
ss <- function(i, d) {
  d %>%
    select_if(is.numeric) %>%
    kmeans(i, nstart = 20) -> km
  km$tot.withinss
}
```

`select_if` is the old version of `across` with `where`. See extra 2. This means "grab all the columns that contain numbers". Then we had to work this out for each number of clusters. To start that off, make a little dataframe from 2 up to the maximum number of clusters you might be using. I'm going up to 30; there are lots of observations, more than in the example in class. The best answer is to *not* go up to 20, just because I did in lecture; we have more observations than we did in the lecture one, so we ought to be considering a larger number of clusters. If you do go only up to 20, have a reason for doing so that is better than "that's what was done in class".

So let's put that together. There's random number generation (in `kmeans`), so I set the random number seed first to get the same results each time I run this:

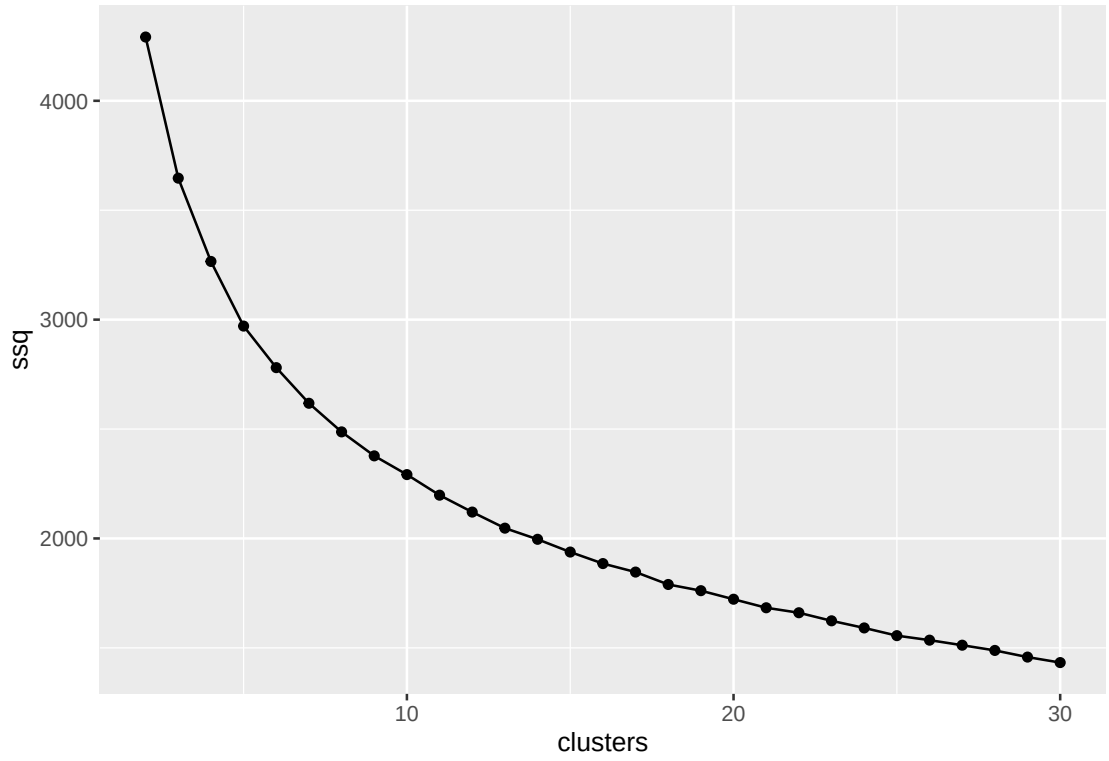
```
set.seed(457299)
tibble(clusters = 2:30) %>%
  mutate(ssq = map_dbl(clusters, ~ss(., weather))) -> ssqs
ssqs
```

```
## # A tibble: 29 x 2
##    clusters   ssq
##       <int> <dbl>
##  1        2 4291.
##  2        3 3647.
##  3        4 3266.
##  4        5 2971.
##  5        6 2781.
##  6        7 2618.
##  7        8 2487.
##  8        9 2377.
##  9       10 2292.
## 10       11 2198.
## # ... with 19 more rows
```

I've given this a name, since I'm about to use it in a plot. See Extra 2 for another way to do this.

Now the simple part: plot the total within-cluster sums of squares against the number of clusters, joining the points by lines:

```
ggplot(ssqs, aes(x = clusters, y = ssq)) + geom_point() + geom_line()
```



Extra 1: `select_if` is from last year's version of the notes, and I forgot to change it. It does the same thing as the right `where`. To demonstrate, let's make a small dataframe:

```
d1 <- tribble(
  ~x, ~g, ~y,
  10, "first", 21,
  11, "second", 24,
  12, "first", 26,
  13, "second", 29
)
d1
```

```
## # A tibble: 4 x 3
##       x g          y
##   <dbl> <chr>  <dbl>
## 1    10 first     21
## 2    11 second    24
## 3    12 first     26
## 4    13 second    29
```

To grab just the number columns, `select_if` works this way:

```
d1 %>% select_if(is.numeric)
```

```
## # A tibble: 4 x 2
##       x     y
##   <dbl> <dbl>
## 1    10    21
## 2    11    24
```

```
## 3    12    26
## 4    13    29
```

and to do the same thing using `where` goes this way:

```
d1 %>% select(where(is.numeric))
```

```
## # A tibble: 4 x 2
##       x     y
##   <dbl> <dbl>
## 1    10    21
## 2    11    24
## 3    12    26
## 4    13    29
```

The reason for the update is that you no longer have to use a special `select`; you use the ordinary one with `where` to choose columns by type, and a select-helper to select columns by name.

Extra 2: in between the last revision of the lecture notes (last summer) and now, I've learned about a thing called `rowwise` for doing things one row at a time. This avoids using `map`, and produces a `mutate` that looks less forbidding. It goes this way:

```
tibble(clusters = 2:30) %>%
  rowwise() %>%
  mutate(ssq = ss(clusters, weather))
```

```
## Warning: did not converge in 10 iterations
```

```
## # A tibble: 29 x 2
## # Rowwise:
##    clusters   ssq
##       <int> <dbl>
##  1        2 4291.
##  2        3 3647.
##  3        4 3266.
##  4        5 2971.
##  5        6 2778.
##  6        7 2615.
##  7        8 2487.
##  8        9 2377.
##  9       10 2288.
## 10       11 2203.
## # ... with 19 more rows
```

The same. The `mutate` looks like a very familiar thing now, because the `rowwise` makes `ss` run on one cluster number at a time. Without the `rowwise`, it tries to do all the numbers of clusters at once and fails, because `kmeans` expects a single number of clusters, not a collection of different numbers:

```
tibble(clusters = 2:15) %>%
  mutate(ssq = ss(clusters, weather))
```

```
## Error: Problem with `mutate()` input `ssq`.
## x must have same number of columns in 'x' and 'centers'
## i Input `ssq` is `ss(clusters, weather)`.
```

(c) From your scree plot, what is a sensible number of clusters to use? Explain briefly.

Solution:

Look for an elbow, if you can see one. I have a tiny one at 5 clusters, so that's my choice. You might reasonably consider this to be too far up the mountain, and you might prefer something like 25 clusters, at least on my plot. Go with whatever looks sensible from your scree plot.

(d) Fit a K-means analysis with your chosen number of clusters. Obtain a dataframe that contains the dates of observation and the cluster that each date belongs to. Display some of your dataframe.

Solution:

K-means with 5 clusters (for me) looks like this. Don't forget to get rid of the column of dates for the analysis:

```
weather %>% select(-Date) %>% kmeans(5) -> weather.1
```

The dates come from the original dataframe and the clusters from the one you just made, so take things from the right places:

```
cl <- tibble(Date = weather$Date, cluster = weather.1$cluster)
cl
```

```
## # A tibble: 354 x 2
##    Date       cluster
##    <date>       <int>
##  1 2007-11-01       5
##  2 2007-11-02       5
##  3 2007-11-03       3
##  4 2007-11-04       2
##  5 2007-11-05       2
##  6 2007-11-06       2
##  7 2007-11-07       1
##  8 2007-11-08       3
##  9 2007-11-09       3
## 10 2007-11-10       5
## # ... with 344 more rows
```

(e) Australia is in the southern hemisphere, so its seasons are opposite to Canada's. Specifically, March through May is autumn, June through August is winter, September through November is spring, and December through February is summer. (These months are all inclusive.) Work out the season for each of the dates in the dataframe you just made, and add them to your dataframe. Hint: find out how `between` works (citing your source).

Solution:

There are two steps here: get the month from each date, and convert the month to a season. The first part uses `month` from `lubridate`:

```
cl %>%
  mutate(month = month(Date))
```

```
## # A tibble: 354 x 3
```

```
##    Date       cluster month
##    <date>       <int> <dbl>
##  1 2007-11-01       5    11
##  2 2007-11-02       5    11
##  3 2007-11-03       3    11
##  4 2007-11-04       2    11
##  5 2007-11-05       2    11
##  6 2007-11-06       2    11
##  7 2007-11-07       1    11
##  8 2007-11-08       3    11
##  9 2007-11-09       3    11
## 10 2007-11-10       5    11
## # ... with 344 more rows
```

Having the month as a number makes the next step easier. The easiest way to make the seasons is to use `case_when`, and to take advantage of `between`. `between` takes three inputs: a column, a lower limit, and an upper limit, and it returns `TRUE` if the value from the column is between the lower and upper limits (inclusive). See, for example, https://dplyr.tidyverse.org/reference/between.html. That goes like this:

```
cl %>%
  mutate(month = month(Date)) %>%
  mutate(season = case_when(
    between(month, 9, 11) ~ "spring",
    between(month, 3, 5)  ~ "autumn",
    between(month, 6, 8)  ~ "winter",
    TRUE                  ~ "summer"
  )) -> seasons
seasons
```

```
## # A tibble: 354 x 4
##    Date       cluster month season
##    <date>       <int> <dbl> <chr>
##  1 2007-11-01       5    11 spring
##  2 2007-11-02       5    11 spring
##  3 2007-11-03       3    11 spring
##  4 2007-11-04       2    11 spring
##  5 2007-11-05       2    11 spring
##  6 2007-11-06       2    11 spring
##  7 2007-11-07       1    11 spring
##  8 2007-11-08       3    11 spring
##  9 2007-11-09       3    11 spring
## 10 2007-11-10       5    11 spring
## # ... with 344 more rows
```

Get rid of the column `month` now if you like, and maybe even the dates, since they have served their purpose. I left summer until the end, because the Australian summer is split over two years, and that is much easier to save for the "otherwise" case, rather than trying to figure it out explicitly.

I used the name "autumn" for the season because that's what they call it in Australia.

(f) Does there seem to be a correspondence between season and cluster? Count something appropriate, and describe any correspondence you see.

Solution:

The thing to count is the combination of cluster and season.

Another way to think about this is that you're looking for an "association" between two categorical variables (`cluster` is categorical even though its values are numbers, because the numbers are labels; you wouldn't want to take the mean of them). With this in mind, you might think of a chi-squared test, and to make *that*, you need a table of frequencies, which you'll need to get first.

The easiest way to do this is with `table`:

```
with(seasons, table(cluster, season))
```

```
##         season
## cluster autumn spring summer winter
##       1     46     23      0     44
##       2     11     11      9     20
##       3     10     12     13     22
##       4      4     17     19      0
##       5     18     27     48      0
```

Having season as columns is better in R terms, because the season names are legal column names and the cluster numbers are not. It also makes sense if you have more clusters than I did, because you can have (many) clusters going down the page and (few) seasons going across.

Since I said `count`, you might also think of this:

```
seasons %>% count(cluster, season)
```

```
## # A tibble: 17 x 3
##    cluster season     n
##      <int> <chr>  <int>
##  1       1 autumn    46
##  2       1 spring    23
##  3       1 winter    44
##  4       2 autumn    11
##  5       2 spring    11
##  6       2 summer     9
##  7       2 winter    20
##  8       3 autumn    10
##  9       3 spring    12
## 10       3 summer    13
## 11       3 winter    22
## 12       4 autumn     4
## 13       4 spring    17
## 14       4 summer    19
## 15       5 autumn    18
## 16       5 spring    27
## 17       5 summer    48
```

The problem with this is that it's not easy to make sense of. The best way to use this kind of thing is to make one of season or cluster (season is better as discussed above) into columns by `pivot_wider`:

```
seasons %>% count(cluster, season) %>%
  pivot_wider(names_from = season, values_from = n,
              values_fill = 0)
```

```
## # A tibble: 5 x 5
##   cluster autumn spring winter summer
##     <int>  <int>  <int>  <int>  <int>
## 1       1     46     23     44      0
## 2       2     11     11     20      9
## 3       3     10     12     22     13
## 4       4      4     17      0     19
## 5       5     18     27      0     48
```

`values_fill` supplies a value for when the cell would otherwise be missing. In this case, if `count` doesn't find any observations for a certain season-cluster combination, that combination is missing from the output to `count` (check it and see), and so the corresponding entry after you pivot wider is also missing. These are actually zero frequencies, so the correct value to fill in is zero.

To interpret this table, you can condition on season, or you can condition on cluster number. Either way is good.

If you condition on season, you would say something like this, for my results (yours will almost certainly be different, in terms of cluster numbers at least):

- most of the autumn days are in cluster 1
- the spring days are in clusters 1 and 5 (or, spring days could be in any cluster)
- most of the summer days are in cluster 5
- most of the winter days are in cluster 1 (or, in clusters 1 through 3).

and you might also note that the correspondence is far from perfect; there are quite a few days in the "wrong" cluster for their season.

If you condition on cluster, you might get this, or the equivalent for your cluster numbers:

- the cluster 1 days are never in the summer
- most of the cluster 2 days are in winter
- the cluster 3 days could be any season, but are most likely to be winter
- the cluster 4 days are mostly spring and summer
- most of the cluster 5 days are in summer.

You will almost certainly have something different from me. Talk intelligently about what you see.

Extra 1: I said chi-squared test, which I didn't do because there seems to be enough of a correspondence to be worth discussing even without it. I'm guessing that it will be strongly significant. R's `chisq.test` (that we saw in the discussion of Mood's median test in C32) requires the output from `table` as input:

```
tab <- with(seasons, table(cluster, season))
tab
```

```
##        season
## cluster autumn spring summer winter
##       1     46     23      0     44
##       2     11     11      9     20
```

```
##       3     10     12     13     22
##       4      4     17     19      0
##       5     18     27     48      0
```

```
season.1 <- chisq.test(tab)
season.1
```

```
##
##  Pearson's Chi-squared test
##
## data:  tab
## X-squared = 139.59, df = 12, p-value < 2.2e-16
```

Oh yes.

If you remember how to calculate these by hand, you'll remember that you get the test statistic by working out observed minus expected squared divided by expected for each cell, and then adding up the results. The test statistic will come out large if any of these are large. Taking the square roots of these and putting an appropriate plus or minus sign on the front gives you "residuals", which you can access here:

```
season.1$residuals
```

```
##        season
## cluster     autumn     spring     summer     winter
##       1  3.3002209 -1.0688231 -5.3300661  3.1583430
##       2 -0.5088362 -0.5460108 -1.0673727  2.1620297
##       3 -1.1439528 -0.6544979 -0.3514689  2.1908268
##       4 -1.9098451  2.1419208  2.8202284 -3.1172928
##       5 -1.1129019  0.6901636  5.0913070 -4.7532326
```

Where the residuals are far from zero, that means the observed and expected were a long way apart, so the frequencies of days in that cell were much larger or smaller than expected. You can then go along the rows or down the columns and pick out the unusual ones, for example:

- cluster 1 has more autumn and winter days, and fewer summer days, than expected.
- cluster 2 has more winter days than expected.
- cluster 3 also has more winter days than expected.
- cluster 4 has more spring and summer days (and fewer days of the other seasons) than expected.
- cluster 5 has a lot more summer days and a lot fewer winter days than expected.

Extra 2: if you have more clusters than my five, have them go down the page. In discussing them, you probably want to save yourself some work and condition on seasons. Here's 25 clusters for me:

```
weather %>% select(-Date) %>% kmeans(25) -> weather.2
cl <- tibble(Date = weather$Date, cluster = weather.2$cluster)
cl %>%
  mutate(month = month(Date)) %>%
  mutate(season = case_when(
    between(month, 9, 11) ~ "spring",
    between(month, 3, 5)  ~ "autumn",
    between(month, 6, 8)  ~ "winter",
    TRUE                  ~ "summer"
  )) -> seasons
seasons %>% count(cluster, season) %>%
  pivot_wider(names_from = season, values_from = n,
              values_fill = 0)
```

```
## # A tibble: 25 x 5
##    cluster autumn spring summer winter
##      <int>  <int>  <int>  <int>  <int>
##  1       1      2      5      1      9
##  2       2      4      5      8      0
##  3       3      9      8      0      1
##  4       4      3      0      0     13
##  5       5      0      8      3      0
##  6       6      7      8      3      2
##  7       7      8      5      3      0
##  8       8      4      2      0      3
##  9       9      1      2      1      1
## 10      10      8      1      0      3
## # ... with 15 more rows
```

A trick with so many clusters is to `View()` this dataframe, and then you can sort the columns to find out which clusters are commonest in each season:

- Autumn days are most commonly in clusters 3, 7, 10, 11, 18
- Spring days could be in almost any cluster, but most commonly in clusters 3, 5, 6, 19
- Summer days are most likely to be in cluster 22 (or possibly in 2, 20, 24, 25)
- Winter days are almost all in clusters 1, 4, 11, 14, 16, 17.

If you want to go the other way, you might pick out clusters dominated by one season, such as 18 for autumn, 20 or 22 for summer, 4, 11, 14, 17 for winter. There are really no clusters that are characteristically spring.

Extra 3: We could make a picture with a discriminant analysis, using the clusters as known groups. I'm going to use my 5-cluster solution for this. Let's start by adding the clusters to the original data:

```
weather %>% mutate(cluster = weather.1$cluster) %>%
  lda(cluster ~ . - Date, data = .) -> discrim
discrim
```

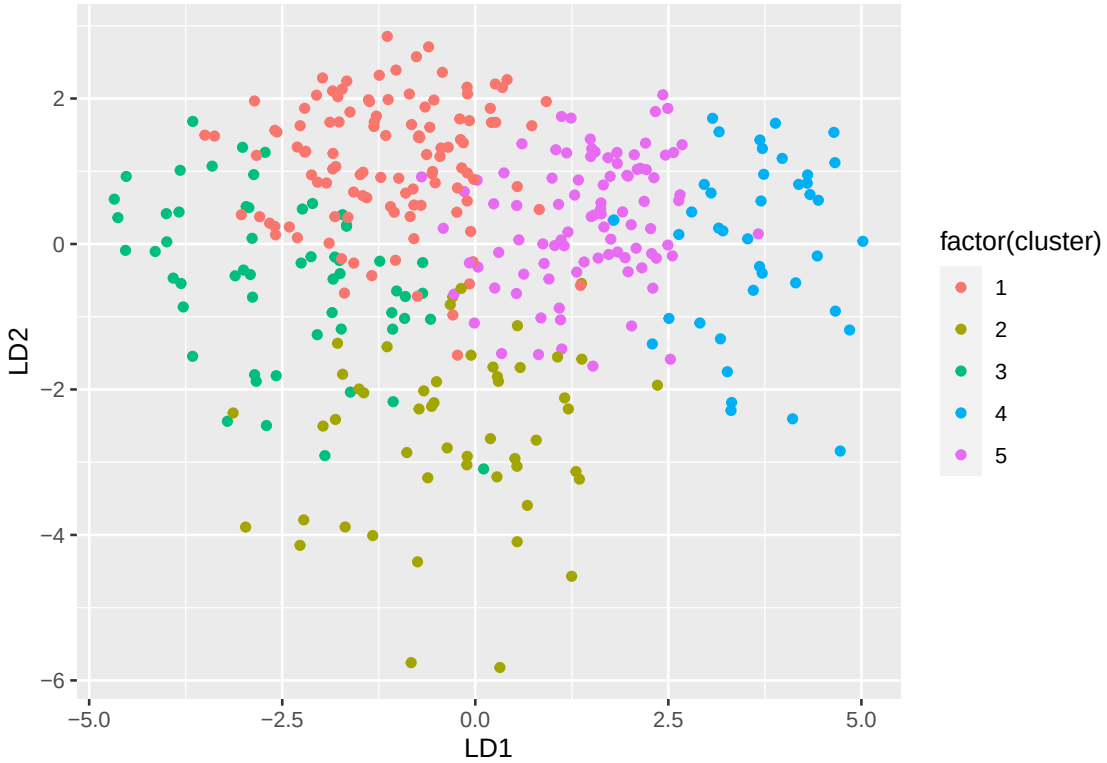
```
## Call:
## lda(cluster ~ . - Date, data = .)
##
## Prior probabilities of groups:
##         1         2         3         4         5
```

```
## 0.3192090 0.1440678 0.1610169 0.1129944 0.2627119
##
## Group means:
##      MinTemp    MaxTemp   Rainfall Evaporation    Sunshine WindGustSpeed
## 1 -0.9262696 -0.6210941 -0.2732206  -0.6530522  0.18058727   -0.59018161
## 2 -0.1242309 -0.6826453  0.7117203  -0.1416107 -0.08991918    1.34999880
## 3  0.1874992 -0.5626943  0.3396986  -0.4420162 -1.69713305   -0.39119439
## 4  0.9927870  1.5588971 -0.2835425   1.4119122  1.02479676    0.54959584
## 5  0.6516699  0.8034003 -0.1445691   0.5347901  0.42929280   -0.01984054
##   WindSpeed9am WindSpeed3pm Humidity9am Humidity3pm Pressure9am Pressure3pm
## 1 -0.393018452   -0.2307883   0.3254525  -0.1292065  0.79020872   0.7719715
## 2  1.462265264    1.2260345  -0.2304044   0.4140495 -1.05711451  -0.8706044
## 3 -0.114342547   -0.3275549   0.8651804   1.3688555  0.03442945   0.1055626
## 4  0.004177901    0.2855545  -1.2688637  -1.1673414 -0.64059500  -0.7436192
## 5 -0.256064885   -0.3139810  -0.2536155  -0.4069602 -0.12601531  -0.2054221
##      Cloud9am      Cloud3pm    Temp9am    Temp3pm
## 1 -0.37658928 -0.3981652723 -0.8638262 -0.5707547
## 2  0.17231803  0.1246737568 -0.2816544 -0.7445832
## 3  1.12231683  1.2005694968 -0.1383711 -0.5962505
## 4 -0.85941655 -0.7464600238  1.3918432  1.5469724
## 5  0.04484915  0.0006478903  0.6902168  0.8018968
##
## Coefficients of linear discriminants:
##                        LD1         LD2         LD3         LD4
## MinTemp       -0.10976321 -0.51912272 -0.37652801  1.29767489
## MaxTemp        1.11659668 -0.48595970  0.10157811 -1.37656925
## Rainfall      -0.12730823 -0.22051658  0.06370335 -0.09112648
## Evaporation    0.16995689  0.15086924 -0.07292771 -0.28339270
## Sunshine       0.67168599  0.18896916  1.00172180  1.70482046
## WindGustSpeed  0.17838343 -0.51838549  0.18517266  0.37753064
## WindSpeed9am   0.12475412 -0.55131103  0.25753766 -0.04344261
## WindSpeed3pm   0.09844632 -0.11981222 -0.03998844 -0.56347058
## Humidity9am   -0.51794560  0.34833966  0.34469305  0.10662696
## Humidity3pm    0.25788229 -0.25693953 -0.52755161 -0.07112335
## Pressure9am   -0.46537137  0.80346899  0.05700435  0.63082534
## Pressure3pm    0.22756298 -0.24986760 -0.20592759 -0.50519412
## Cloud9am       0.11377039 -0.02571451 -0.01611378  0.80110567
## Cloud3pm      -0.11977314 -0.19297893 -0.16234296  0.76154178
## Temp9am       -0.51415454  1.00600772  0.17895205 -0.66799783
## Temp3pm        0.76447057  0.01038316 -1.19387894  0.57194430
##
## Proportion of trace:
##    LD1    LD2    LD3    LD4
## 0.5625 0.2304 0.1791 0.0280
```

Dunno what you make of that. High on LD1 means high max temperature, high sunshine, low humidity in the morning, high temp in the afternoon but *low* temp in the morning. "Becoming hot and sunny", you might say. LD2 is a mixed bag: low minimum temp, low wind gust and morning wind, high pressure and temp in the morning. I dunno.

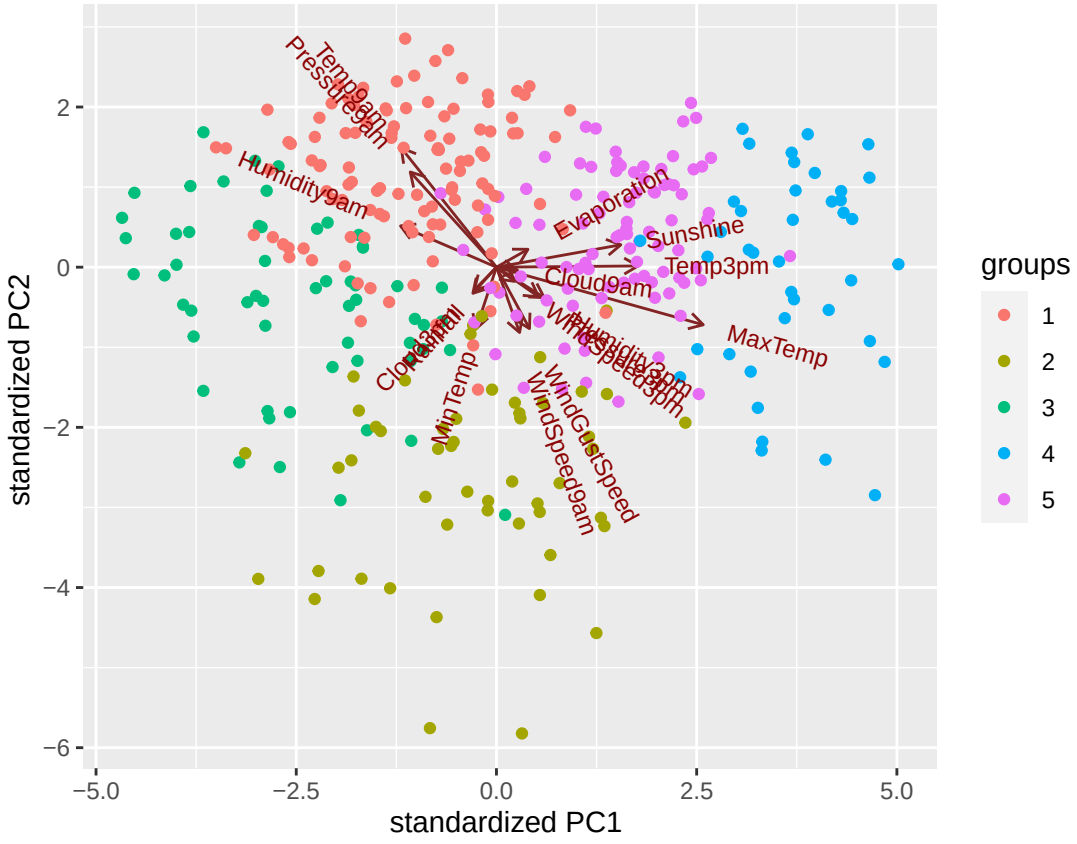
Now let's get the discriminant scores:

```
p <- predict(discrim)
scores <- data.frame(cluster = weather.1$cluster, p$x)
ggplot(scores, aes(x = LD1, y = LD2, colour = factor(cluster))) + geom_point()
```



Well, the clusters are separated some. A biplot might be illuminating:

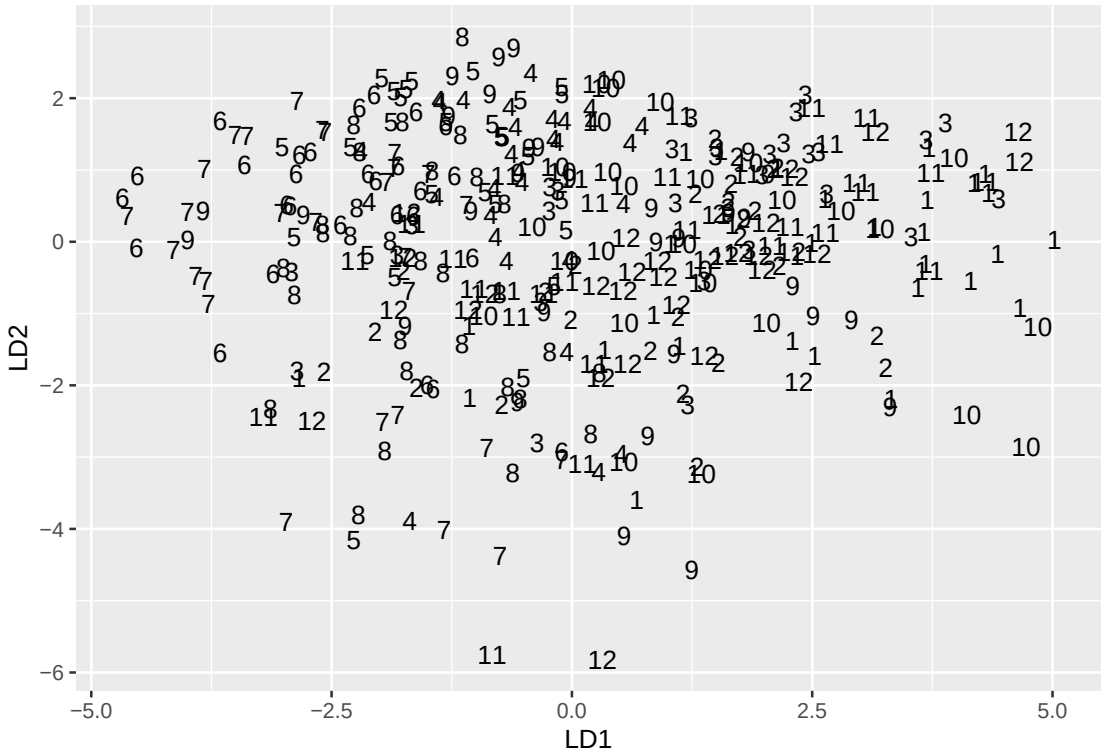
```
ggbiplot(discrim, groups = factor(weather.1$cluster))
```



The high temperature and sunshine stuff definitely points right, and it is perhaps clearer that humidity and the 9am temperature and pressure point the other way, so that these are high on the left and low on the right. There isn't much pointing up and down; the main things are morning wind speed and wind gust, and the minimum temperature. I found out how to add the clusters, so now you can get a sense of what kind of weather the clusters stand for and what distinguishes them.

Let me add one more thing – the month of the year, which I will attempt to plot instead of the points. This might be a bit fiddly:

```
data.frame(month = seasons$month, p$x) %>%
  ggplot(aes(x = LD1, y = LD2, label = month)) + geom_text()
```



In so far as there is anything to see, a lot of points towards the end/start of the year (summer) seem to be on the right, and a lot of points in the middle of the year (winter) seem to be on the left. But what are those two summer points doing at the bottom?[2] This is very unclear to me, and I didn't want to make you grapple with this.

As far as the plotting is concerned, `geom_text` is for plotting text *at* points, so that the text takes the place of the symbol that `geom_point` would put there. This is different from labelling points that are already there, which you would do with `geom_text_repel` or similar.

Notes

1. In May 2020, the 1.0.0 version of "dplyr" came out, which has a lot of new and better ways to do things, like "across". "where" is part of this, too.

2. The biplot suggests that these may be windy days.