

All the slides

```
list.files(".", pattern = "*_slides.pdf")
```

```
## [1] "all_slides.pdf"          "bAnova_slides.pdf"
## [3] "bCluster_slides.pdf"     "bFactor_slides.pdf"
## [5] "bootstrap_R_slides.pdf"  "bootstrap_slides.pdf"
## [7] "bPrincomp_slides.pdf"    "choosing_R_slides.pdf"
## [9] "inference_2_R_slides.pdf" "inference_3_R_slides.pdf"
## [11] "inference_5_R_slides.pdf" "running_R_slides.pdf"
## [13] "windmill_slides.pdf"
```

C32/C33

Section 1

Running R

Running R online

Go to <https://jupyter.utoronto.ca/hub/login>.

You want to see this (if you don't, see next two slides):



After logging in, open: ☒ Jupyter Notebook ☐ RStudio ☐ JupyterLab

Log in to start

Use JupyterHelp to open tickets for support questions.

Welcome to the new
University of Toronto
JupyterHub for Teaching
site.

A proof of concept service, developed as a
partnership between the Office of the CIO

jupyter

R Studio

If you do, find the line that says “after logging in, open” and click the button next to R Studio to make it blue. Then click the orange Log In to Start, and log in with UTorID and password.

Problems 1

You might instead see this:



Log in to continue

Use [JupyterHelp](#) to open tickets for support questions.

Welcome to the new
University of Toronto
JupyterHub for Teaching
site.



A proof of concept service, developed as a
partnership between the [Office of the CIO](#)
(Information Technology Services), the

It looks exactly the same, but has no option to start R Studio. Log in anyway, which will bring you to the screen shown in Problems 2.

Problems 2

You might see something that looks like this:

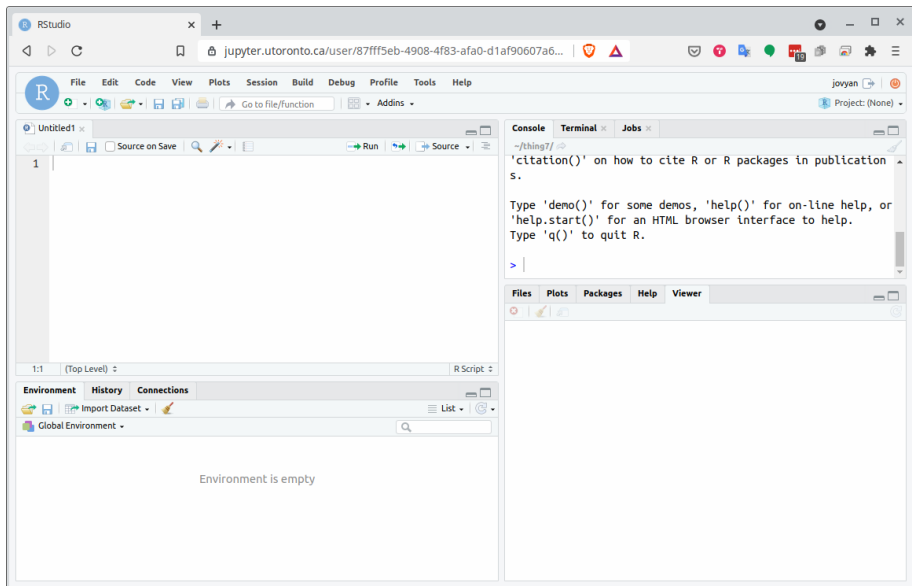


The screenshot shows the JupyterHub web interface. At the top left is the JupyterHub logo. At the top right are buttons for 'Logout' and 'Control Panel'. Below the logo is a navigation bar with 'Files', 'Running' (selected), and 'Clusters' tabs. Under the 'Running' tab, there is a prompt 'Select items to perform actions on them.' followed by buttons for 'Download Directory', 'Upload', 'New', and a refresh icon. Below this is a table of running clusters.

<input type="checkbox"/> 0 ▾	/	Name ▾	Last Modified	File size
<input type="checkbox"/>	cmdstanr		2 months ago	
<input type="checkbox"/>	rustle		3 days ago	
<input type="checkbox"/>	thing7		3 hours ago	

Click Logout (top right), even if you just logged in. This should bring you back to where you should be, from where you can select R Studio and log in.

How R Studio looks



Projects

- Each user has a “workspace”, a place where all your work is stored.
- Within that workspace, you can have as many Projects as you like.
- To create a new Project, click on the blue New Project button.
- I recommend having one project per *course*.
- R Studio restarts where you left off.

Make a new project

- Call it what you like. Mine is called `thing7`:
- Select:
 - File,
 - New Project,
 - New Directory,
 - New Project (again),
 - give it a name and click Create Project.
- You see the name of your new project top right.

R Notebooks

- At top right of previous view is Console, where you can enter R commands and see output.
- A better way to work is via “R Notebooks”. These allow you to combine narrative, code and output in one document.
- Data analysis is always a story: not only what you did, but why you did it, with the “why” being more important.
- To create a new notebook, select File, New File, R Notebook. This brings up an example notebook as over.
- The next example was done in another online environment, so looks (slightly) different, but will still work for you.

The template R Notebook

The screenshot displays the RStudio IDE with a new R Notebook titled 'Untitled1'. The interface includes a menu bar (File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Help) and a toolbar with icons for saving, running, and previewing. The main editor area contains R Markdown code:

```

1 ---
2 title: "R Notebook"
3 output: html_notebook
4 |---
5
6 This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute
7 code within the notebook, the results appear beneath the code.
8
9 Try executing this chunk by clicking the "Run" button within the chunk or by
10 placing your cursor inside it and pressing "Ctrl+Shift+Enter".
11
12 ```{r}
13 plot(cars)
14 ```
15
16 Add a new chunk by clicking the "Insert Chunk" button on the toolbar or by
17 pressing "Ctrl+Alt+I".
18
19 When you save the notebook, an HTML file containing the code and output will be
20 saved alongside it (click the "Preview" button or press "Ctrl+Shift+K" to preview
21 the HTML file).
22
23 The preview shows you a rendered HTML copy of the contents of the editor.
24 Consequently, unlike "Knit", "Preview" does not run any R code chunks. Instead,
25 the output of the chunk when it was last run in the editor is displayed.

```

The right-hand pane shows the 'Environment' tab, which is currently empty, displaying the message 'Environment is empty'. Below it, the 'Files' pane shows the project structure:

	Name	Size	Modified
<input type="checkbox"/>	..		
<input type="checkbox"/>	.Rhistory	0 B	Jun 29, 2018, 3:31 PM
<input type="checkbox"/>	project.Rproj	205 B	Jun 29, 2018, 4:33 PM

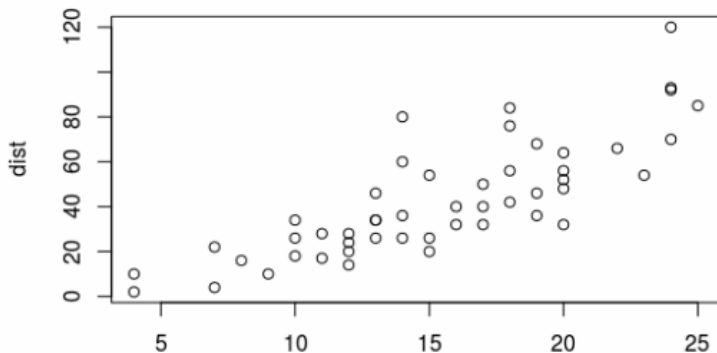
About this notebook

- The notebook begins with a title (that you can change).
- Most of this notebook is text (narrative). The stuff with *asterisks* around it will come out in italics in the final document.
- Pieces beginning with three “backticks” and `{r}`, in grey, are called code chunks. They contain R code. Three more backticks marks the end of a code chunk.
- Run code chunks by clicking on the green “play button” at the top right of the chunk. This one makes a scatterplot. If you click the play button, the plot is made and placed under the code, as over.

After running the code chunk

8 Try executing this chunk by clicking the **Run** button within the chunk or by
9 placing your cursor inside it and pressing **Ctrl+Shift+Enter**.

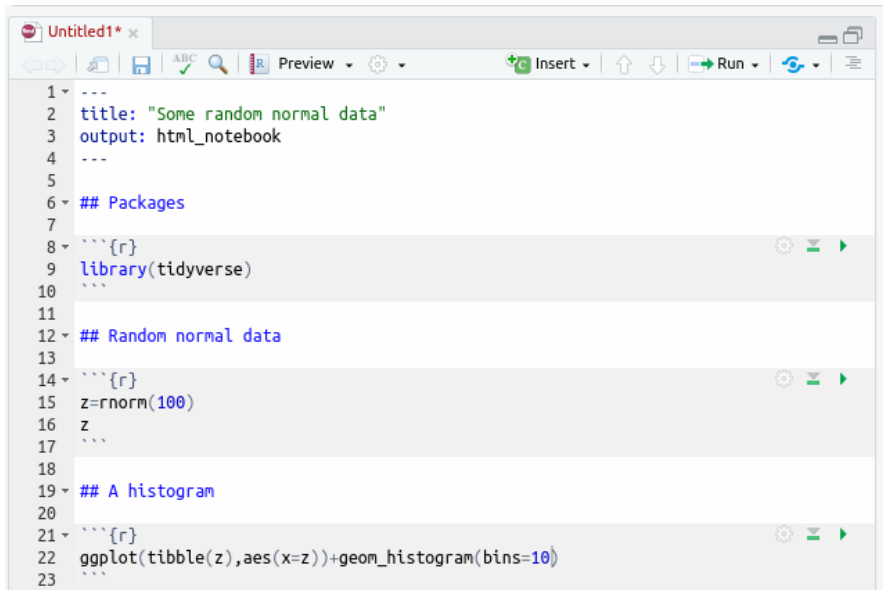
```
10 ```{r}  
11 plot(cars)  
12 ```
```



Making our own notebook

- Create another new notebook. Delete the template text and change the title to “Some random normal data”.
- Type `## Packages` and go down a couple of lines.
- Make a new code chunk by clicking Insert (at the top of the notebook window) and selecting R. Inside that chunk, type `library(tidyverse)`.
- Below that, type `## Random normal data`.
- Make another new code chunk below that, and insert two lines of code: `z=rnorm(100)` and then `z`.
- Below that, type text `## A histogram` and a code chunk containing `ggplot(tibble(z),aes(x=z))+geom_histogram(bins=10)`.

My R notebook



The screenshot shows an R notebook window titled "Untitled1* x". The interface includes a toolbar with icons for navigation, saving, searching, and running code. The code is organized into sections with comments and code blocks. The first section is a title and output format. The second section loads the tidyverse package. The third section generates random normal data. The fourth section creates a histogram using ggplot2.

```
1 ---  
2 title: "Some random normal data"  
3 output: html_notebook  
4 ---  
5  
6 ## Packages  
7  
8 ```{r}  
9 library(tidyverse)  
10 ```  
11  
12 ## Random normal data  
13  
14 ```{r}  
15 z=rnorm(100)  
16 z  
17 ```  
18  
19 ## A histogram  
20  
21 ```{r}  
22 ggplot(tibble(z),aes(x=z))+geom_histogram(bins=10)  
23 ```
```

Run the chunks

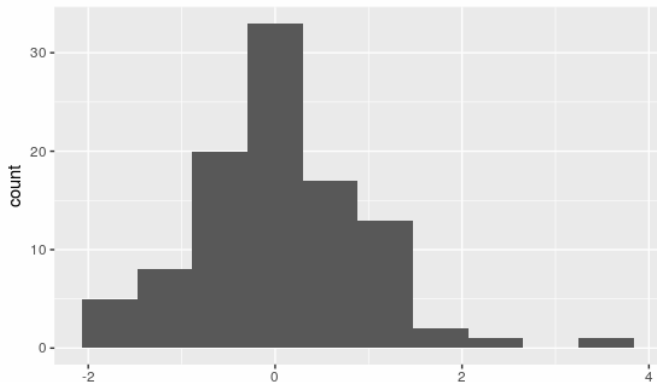
- Now run each of the three chunks in order. You'll see output below each one, including a histogram below the last one.
- When it works, add some narrative text before the code chunks explaining what is going to be done, and some text after describing what you see.
- Save the notebook (File, Save As). You don't need a file extension.
- Click Preview. This makes an HTML-formatted report. (The first may be gibberish: ignore that). Note what happened to the text.
- If you want to edit anything, go back to the R Notebook, change it, save it, and run Preview again.

The end of my (formatted) report

A histogram

To see whether we got a rough bell shape, we can draw a histogram:

```
ggplot(tibble(z),aes(x=z))+geom_histogram(bins=10)
```



Installing R on your own computer

- Free, open-source. Download and run on own computer.
- Two things: R itself (install first) and R Studio (front end).
- Go to <https://www.r-project.org/>:

The R Project for Statistical Computing

Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To **download R**, please choose your preferred **CRAN mirror**.

Click on Download

- R is stored on numerous “mirrors”, sites around the world. The top one, “0-Cloud”, picks one for you. Or you can choose one close to you (might be faster), eg. U of T:

CRAN Mirrors

The Comprehensive R Archive Network is available at the following URLs, please choose a location close to you. Some statistics on the status of the mirrors can be found here: [main page](#), [windows release](#), [windows old release](#).

If you want to host a new mirror at your institution, please have a look at the [CRAN Mirror HOWTO](#).

0-Cloud

<https://cloud.r-project.org/>

<http://cloud.r-project.org/>

Automatic redirection to servers worldwide, currently sponsored by Rstudio

Automatic redirection to servers worldwide, currently sponsored by Rstudio

Algeria

...

Bulgaria

<http://ftp.uni-sofia.bg/CRAN/>

Sofia University

Canada

<http://cran.stat.sfu.ca/>

Simon Fraser University, Burnaby

<http://mirror.its.dal.ca/cran/>

Dalhousie University, Halifax

<http://cran.utstat.utoronto.ca/>

University of Toronto

Chile

<https://dirichlet.mat.nuc.cl/>

Pontificia Universidad Catolica de Chile, Santiago

Click your mirror

- Click 0-Cloud or U of T (or other mirror), get:

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows**

- [Download R for Linux](#)
 - [Download R for \(Mac\) OS X](#)
 - [Download R for Windows](#)
- Click on your operating system, eg. Windows.

Click on Base

R for Windows

Subdirectories:

<u>base</u>	Binaries for base distribution (managed by Duncan Murdoch). This is what you want to <u>install R for the first time</u> .
<u>contrib</u>	Binaries of contributed CRAN packages (for R \geq 2.11.x; managed by Uwe Ligges). There is also information on <u>third party software</u> available for CRAN Windows services and corresponding environment and make variables.
<u>old contrib</u>	Binaries of contributed CRAN packages for outdated versions of R (for R $<$ 2.11.x; managed by Uwe Ligges).
<u>Rtools</u>	Tools to build R and R packages (managed by Duncan Murdoch). This is what you want to build your own packages on Windows, or to build R itself.

- Click on “base” here.

The actual download

- The version number, here 3.5.3, will be different for you.
- Click something like the top link below:

[Download R 3.5.3 for Windows](#) (79 megabytes, 32/64 bit)

[Installation and other instructions](#)

[New features in this version](#)

If you want to double-check that the package you have downloaded matches the package distributed b windows: both [graphical](#) and [command line versions](#) are available.

- Then install usual way.
- Or, for Mac, download and install R-3-5-1.pkg.
- Or, for Linux, click your distribution (eg. Ubuntu), then one of the cran35 links according to your version, then eg. r-base-core_3.5.1-1bionic_amd64.deb.


Now, R Studio

- Go [here](#). (The word “here” is clickable link.)
- Find this, and click Download:



RStudio

RStudio makes R easier to use. It includes a code editor, debugging & visualization tools.

 Download  Learn More

Scroll down...

- to this:

RStudio Desktop Open Source License	RStudio Desktop Commercial License	RStudio Server Open Source License	RStudio Server Pro Commercial License	RStudio Server Pro + RStudio Connect Commercial License
FREE	\$995 per year	FREE	\$9,995 per year	\$29,995 per year
DOWNLOAD	BUY	DOWNLOAD	DOWNLOAD	TALK

- Click left-side Download.

Find the one for you

- Scroll down, and click the installer for your machine (Windows, Mac, several flavours of Linux). Install as usual.

RStudio requires R 2.11.1 (or higher). If you don't already have R, you can download it [here](#).

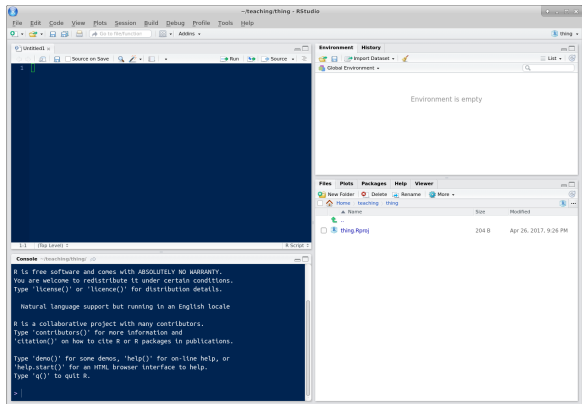
Installers for Supported Platforms

Installers	Size	Date	
RStudio 0.99.902 - Windows Vista/7/8/10	77.1 MB	2016-05-14	⌵
RStudio 0.99.902 - Mac OS X 10.6+ (64-bit)	60 MB	2016-05-14	⌵
RStudio 0.99.902 - Ubuntu 12.04+/Debian 8+ (32-bit)	81.6 MB	2016-05-14	⌵
RStudio 0.99.902 - Ubuntu 12.04+/Debian 8+ (64-bit)	88.3 MB	2016-05-14	⌵
RStudio 0.99.902 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (32-bit)	81 MB	2016-05-14	⌵
RStudio 0.99.902 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (64-bit)	81.9 MB	2016-05-14	⌵

Running R

- All of above only done once.
- To run R, run R Studio, which itself runs R.

How R Studio looks when you run it



- First time you run R Studio, click on Console window, and, next to the `>`, type `install.packages("tidyverse")`. Let it do what it needs to.

Projects

- A project is a “container” for code and data that belong together.
- Goes with a folder on some computer.
- File, New Project. You have option to create the new project in a new folder, or in a folder that already exists.
- Use a project for a collection of work that belongs together, eg. data files and notebooks for assignments. Putting everything in a project folder makes it easier to find.
- Example: use a project for (all) assignments, a different notebook within that project for each one.

Section 2

Choosing things in a dataframe

Doing things with data frames

Let's go back to our Australian athletes:

```
##
## -- Column specification -----
## cols(
##   Sex = col_character(),
##   Sport = col_character(),
##   RCC = col_double(),
##   WCC = col_double(),
##   Hc = col_double(),
##   Hg = col_double(),
##   Ferr = col_double(),
##   BMI = col_double(),
##   SSF = col_double(),
##   `Bfat` = col_double(),
##   LBM = col_double(),
##   Ht = col_double()
```

Choosing a column

```
athletes %>% select(Sport)
```

```
## # A tibble: 202 x 1
##   Sport
##   <chr>
## 1 Netball
## 2 Netball
## 3 Netball
## 4 Netball
## 5 Netball
## 6 Netball
## 7 Netball
## 8 Netball
## 9 Netball
## 10 Netball
## # ... with 192 more rows
```

Choosing several columns

```
athletes %>% select(Sport, Hg, BMI)
```

```
## # A tibble: 202 x 3
##   Sport      Hg    BMI
##   <chr>    <dbl> <dbl>
## 1 Netball  13.6   19.2
## 2 Netball  12.7   21.2
## 3 Netball  12.3   21.4
## 4 Netball  12.3   21.0
## 5 Netball  12.8   21.8
## 6 Netball  11.8   21.4
## 7 Netball  12.7   21.5
## 8 Netball  12.4   24.4
## 9 Netball  12.4   22.6
## 10 Netball 14.1   22.8
## # ... with 192 more rows
```


Choosing consecutive columns

```
athletes %>% select(Sex:WCC)
```

```
## # A tibble: 202 x 4
##   Sex      Sport      RCC      WCC
##   <chr>   <chr>   <dbl> <dbl>
## 1 female Netball  4.56  13.3
## 2 female Netball  4.15    6
## 3 female Netball  4.16   7.6
## 4 female Netball  4.32   6.4
## 5 female Netball  4.06   5.8
## 6 female Netball  4.12   6.1
## 7 female Netball  4.17    5
## 8 female Netball  3.8    6.6
## 9 female Netball  3.96   5.5
## 10 female Netball 4.44   9.7
## # ... with 192 more rows
```

Choosing all-but some columns

```
athletes %>% select(-(RCC:LBM))
```

```
## # A tibble: 202 x 4
##   Sex      Sport      Ht      Wt
##   <chr>   <chr>   <dbl> <dbl>
## 1 female Netball  177.   59.9
## 2 female Netball  173.    63
## 3 female Netball  176    66.3
## 4 female Netball  170.   60.7
## 5 female Netball  183    72.9
## 6 female Netball  178.   67.9
## 7 female Netball  177.   67.5
## 8 female Netball  174.   74.1
## 9 female Netball  174.   68.2
## 10 female Netball 174.   68.8
## # ... with 192 more rows
```

Select-helpers

Other ways to select columns: those whose name:

- `starts_with` something
- `ends_with` something
- `contains` something
- matches a “regular expression”
- `everything()` select all the columns

Columns whose names begin with S

```
athletes %>% select(starts_with("S"))
```

```
## # A tibble: 202 x 3
##   Sex      Sport      SSF
##   <chr>   <chr>   <dbl>
## 1 female Netball    49
## 2 female Netball  110.
## 3 female Netball    89
## 4 female Netball  98.3
## 5 female Netball  122.
## 6 female Netball  90.4
## 7 female Netball  107.
## 8 female Netball  157.
## 9 female Netball  101.
## 10 female Netball  126.
## # ... with 192 more rows
```

Columns whose names end with C

either uppercase or lowercase:

```
athletes %>% select(ends_with("c"))
```

```
## # A tibble: 202 x 3
##       RCC      WCC      Hc
##   <dbl> <dbl> <dbl>
## 1  4.56  13.3  42.2
## 2  4.15    6   38
## 3  4.16   7.6  37.5
## 4  4.32   6.4  37.7
## 5  4.06   5.8  38.7
## 6  4.12   6.1  36.6
## 7  4.17    5   37.4
## 8  3.8    6.6  36.5
## 9  3.96   5.5  36.3
## 10 4.44   9.7  41.4
```

Case-sensitive

This works with any of the select-helpers:

```
athletes %>% select(ends_with("C", ignore.case=F))
```

```
## # A tibble: 202 x 2
```

```
##       RCC      WCC
```

```
##    <dbl> <dbl>
```

```
##  1  4.56  13.3
```

```
##  2  4.15    6
```

```
##  3  4.16   7.6
```

```
##  4  4.32   6.4
```

```
##  5  4.06   5.8
```

```
##  6  4.12   6.1
```

```
##  7  4.17    5
```

```
##  8  3.8    6.6
```

```
##  9  3.96   5.5
```

```
## 10  4.44   9.7
```

Column names containing letter R

```
athletes %>% select(contains("r"))
```

```
## # A tibble: 202 x 3
##   Sport      RCC  Ferr
##   <chr>    <dbl> <dbl>
## 1 Netball  4.56    20
## 2 Netball  4.15    59
## 3 Netball  4.16    22
## 4 Netball  4.32    30
## 5 Netball  4.06    78
## 6 Netball  4.12    21
## 7 Netball  4.17   109
## 8 Netball  3.8     102
## 9 Netball  3.96    71
## 10 Netball 4.44    64
## # ... with 192 more rows
```

Exactly two characters, ending with T

In regular expression terms, this is `^.t$`:

- `^` means “start of text”
- `.` means “exactly one character, but could be anything”
- `$` means “end of text”.

```
athletes %>% select(matches("^.t$"))
```

```
## # A tibble: 202 x 2
```

```
##       Ht      Wt
```

```
##    <dbl> <dbl>
```

```
##  1  177.   59.9
```

```
##  2  173.    63
```

```
##  3  176   66.3
```

```
##  4  170.   60.7
```

```
##  5  183   72.9
```

```
##  6  178.   67.9
```

```
##  7  177   67.5
```


Choosing columns by property

- Use where as with summarizing several columns
- eg, to choose text columns:

```
athletes %>% select(where(is.character))
```

```
## # A tibble: 202 x 2
```

```
##   Sex      Sport
```

```
##   <chr>   <chr>
```

```
## 1 female Netball
```

```
## 2 female Netball
```

```
## 3 female Netball
```

```
## 4 female Netball
```

```
## 5 female Netball
```

```
## 6 female Netball
```

```
## 7 female Netball
```

```
## 8 female Netball
```

```
## 9 female Netball
```

Choosing rows by number

```
athletes %>% slice(16:25)
```

```
## # A tibble: 10 x 13
```

	Sex	Sport	RCC	WCC	Hc	Hg	Ferr	BMI
	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
## 1	female	Netba~	4.25	10.7	39.5	13.2	127	24.5
## 2	female	Netba~	4.46	10.9	39.7	13.7	102	24.0
## 3	female	Netba~	4.4	9.3	40.4	13.6	86	26.2
## 4	female	Netba~	4.83	8.4	41.8	13.4	40	20.0
## 5	female	Netba~	4.23	6.9	38.3	12.6	50	25.7
## 6	female	Netba~	4.24	8.4	37.6	12.5	58	25.6
## 7	female	Netba~	3.95	6.6	38.4	12.8	33	19.9
## 8	female	Netba~	4.03	8.5	37.7	13	51	23.4
## 9	female	BBall	3.96	7.5	37.5	12.3	60	20.6
## 10	female	BBall	4.41	8.3	38.2	12.7	68	20.7

```
## # ... with 5 more variables: SSF <dbl>,  
## # %Defat <dbl>, TDM <dbl>, U+ <dbl>, U+ <dbl>
```

Non-consecutive rows

```
athletes %>%
  slice(10,13,17,42)
```

```
## # A tibble: 4 x 13
##   Sex      Sport      RCC      WCC      Hc      Hg      Ferr      BMI
##   <chr>   <chr>    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 female Netball  4.44   9.7   41.4  14.1    64  22.8
## 2 female Netball  4.02   9.1   37.7  12.7   107  23.0
## 3 female Netball  4.46  10.9   39.7  13.7   102  24.0
## 4 female Row      4.37   8.1   41.8  14.3    53  23.5
## # ... with 5 more variables: SSF <dbl>,
## #   %Bfat <dbl>, LBM <dbl>, Ht <dbl>, Wt <dbl>
```

A random sample of rows

```
athletes %>% slice_sample(n=8)
```

```
## # A tibble: 8 x 13
##   Sex      Sport      RCC      WCC      Hc      Hg      Ferr      BMI
##   <chr>   <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 female Field      4.75    7.5   43.8   15.2     90   31.9
## 2 male   WPolo      5.03    7.5   43.6   14.4    102   23.2
## 3 male   Row       4.95    5.9   45.4   15.5    125   23.8
## 4 male   Field      5.01    8.9   46     15.9    212   30.2
## 5 male   WPolo      5.04    6     45.9   15      52   21.3
## 6 male   WPolo      5.16   12.9   47.6   15.6    156   21.9
## 7 female Netball   4.32    6.4   37.7   12.3     30   21.0
## 8 male   WPolo      4.95    7.5   44.5   15      50   24.3
## # ... with 5 more variables: SSF <dbl>,
## #   %Bfat <dbl>, LBM <dbl>, Ht <dbl>, Wt <dbl>
```

Rows for which something is true

```
athletes %>% filter(Sport == "Tennis")
```

```
## # A tibble: 11 x 13
```

```
##   Sex      Sport    RCC    WCC    Hc    Hg  Ferr  BMI
##   <chr> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 female Tennis    4      4.2  36.6  12      57  25.4
## 2 female Tennis  4.4      4    40.8  13.9     73  22.1
## 3 female Tennis  4.38    7.9  39.8  13.5     88  21.2
## 4 female Tennis  4.08    6.6  37.8  12.1    182  20.5
## 5 female Tennis  4.98    6.4  44.8  14.8     80  17.1
## 6 female Tennis  5.16    7.2  44.3  14.5     88  18.3
## 7 female Tennis  4.66    6.4  40.9  13.9    109  18.4
## 8 male   Tennis  5.66    8.3  50.2  17.7     38  23.8
## 9 male   Tennis  5.03    6.4  42.7  14.3    122  22.0
## 10 male   Tennis  4.97    8.8  43     14.9   233  22.3
## 11 male   Tennis  5.38    6.3  46     15.7    32  21.1
## # ... with 5 more variables: SSF <dbl>,
## #   %Bfat <dbl>, LBM <dbl>, Ht <dbl>, Wt <dbl>
```

More complicated selections

```
athletes %>% filter(Sport == "Tennis", RCC < 5)
```

```
## # A tibble: 7 x 13
```

```
##   Sex      Sport    RCC    WCC    Hc    Hg    Ferr    BMI
##   <chr> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 female Tennis    4     4.2  36.6  12     57   25.4
## 2 female Tennis  4.4     4   40.8  13.9   73   22.1
## 3 female Tennis  4.38   7.9  39.8  13.5   88   21.2
## 4 female Tennis  4.08   6.6  37.8  12.1  182   20.5
## 5 female Tennis  4.98   6.4  44.8  14.8   80   17.1
## 6 female Tennis  4.66   6.4  40.9  13.9  109   18.4
## 7 male    Tennis  4.97   8.8  43    14.9  233   22.3
## # ... with 5 more variables: SSF <dbl>,
## #   %Bfat <dbl>, LBM <dbl>, Ht <dbl>, Wt <dbl>
```

Another way to do “and”

```
athletes %>% filter(Sport == "Tennis") %>%
  filter(RCC < 5)
```

```
## # A tibble: 7 x 13
```

```
##   Sex      Sport    RCC    WCC    Hc    Hg    Ferr    BMI
##   <chr>   <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 female Tennis    4      4.2  36.6  12      57  25.4
## 2 female Tennis  4.4      4    40.8  13.9    73  22.1
## 3 female Tennis  4.38    7.9  39.8  13.5    88  21.2
## 4 female Tennis  4.08    6.6  37.8  12.1   182  20.5
## 5 female Tennis  4.98    6.4  44.8  14.8    80  17.1
## 6 female Tennis  4.66    6.4  40.9  13.9   109  18.4
## 7 male    Tennis  4.97    8.8  43    14.9   233  22.3
## # ... with 5 more variables: SSF <dbl>,
## #   %Bfat <dbl>, LBM <dbl>, Ht <dbl>, Wt <dbl>
```

Either/Or

```
athletes %>% filter(Sport == "Tennis" | RCC > 5)
```

```
## # A tibble: 66 x 13
```

	Sex	Sport	RCC	WCC	Hc	Hg	Ferr	BMI
	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
## 1	female	Row	5.02	6.4	44.8	15.2	48	19.8
## 2	female	T400m	5.31	9.5	47.1	15.9	29	21.4
## 3	female	Field	5.33	9.3	47	15	62	25.3
## 4	female	TSprnt	5.16	8.2	45.3	14.7	34	20.3
## 5	female	Tennis	4	4.2	36.6	12	57	25.4
## 6	female	Tennis	4.4	4	40.8	13.9	73	22.1
## 7	female	Tennis	4.38	7.9	39.8	13.5	88	21.2
## 8	female	Tennis	4.08	6.6	37.8	12.1	182	20.5
## 9	female	Tennis	4.98	6.4	44.8	14.8	80	17.1
## 10	female	Tennis	5.16	7.2	44.3	14.5	88	18.3

```
## # ... with 56 more rows, and 5 more variables:
```


Sorting into order

```
athletes %>% arrange(RCC)
```

```
## # A tibble: 202 x 13
```

	Sex	Sport	RCC	WCC	Hc	Hg	Ferr	BMI
	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
## 1	female	Netba~	3.8	6.6	36.5	12.4	102	24.4
## 2	female	Netba~	3.9	6.3	35.9	12.1	78	20.1
## 3	female	T400m	3.9	6	38.9	13.5	16	19.4
## 4	female	Row	3.91	7.3	37.6	12.9	43	22.3
## 5	female	Netba~	3.95	6.6	38.4	12.8	33	19.9
## 6	female	Row	3.95	3.3	36.9	12.5	40	24.5
## 7	female	Netba~	3.96	5.5	36.3	12.4	71	22.6
## 8	female	BBall	3.96	7.5	37.5	12.3	60	20.6
## 9	female	Tennis	4	4.2	36.6	12	57	25.4
## 10	female	Netba~	4.02	9.1	37.7	12.7	107	23.0

```
## # ... with 192 more rows, and 5 more variables:
```

```
## " " RCC <dbl> WCC <dbl> Hc <dbl> Hg <dbl> Ferr <dbl> BMI <dbl>
```

Breaking ties by another variable

```
athletes %>% arrange(RCC, BMI)
```

```
## # A tibble: 202 x 13
```

	Sex	Sport	RCC	WCC	Hc	Hg	Ferr	BMI
	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
## 1	female	Netba~	3.8	6.6	36.5	12.4	102	24.4
## 2	female	T400m	3.9	6	38.9	13.5	16	19.4
## 3	female	Netba~	3.9	6.3	35.9	12.1	78	20.1
## 4	female	Row	3.91	7.3	37.6	12.9	43	22.3
## 5	female	Netba~	3.95	6.6	38.4	12.8	33	19.9
## 6	female	Row	3.95	3.3	36.9	12.5	40	24.5
## 7	female	BBall	3.96	7.5	37.5	12.3	60	20.6
## 8	female	Netba~	3.96	5.5	36.3	12.4	71	22.6
## 9	female	Tennis	4	4.2	36.6	12	57	25.4
## 10	female	Netba~	4.02	9.1	37.7	12.7	107	23.0

```
## # ... with 192 more rows, and 5 more variables:
```

```
athletes %>% arrange(desc(BMI))
```

```
## # A tibble: 202 x 13
##   Sex      Sport   RCC    WCC    Hc    Hg   Ferr   BMI
##   <chr>   <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 male    Field  5.48   6.2  48.2  16.3    94  34.4
## 2 male    Field  4.96   8.3  45.3  15.7   141  33.7
## 3 male    Field  5.48   4.6  49.4   18    132  32.5
## 4 female  Field  4.75   7.5  43.8  15.2    90  31.9
## 5 male    Field  5.01   8.9  46    15.9   212  30.2
## 6 male    Field  5.01   8.9  46    15.9   212  30.2
## 7 male    Field  5.09   8.9  46.3  15.4    44  30.0
## 8 female  Field  4.58   5.8  42.1  14.7   164  28.6
## 9 female  Field  4.51    9   39.7  14.3    36  28.1
## 10 male   WPolo  5.34   6.2  49.8  17.2   143  27.8
## # ... with 192 more rows, and 5 more variables:
```

“The top ones”

```
athletes %>%  
  arrange(desc(Wt)) %>%  
  slice(1:7) %>%  
  select(Sport, Wt)
```

```
## # A tibble: 7 x 2  
##   Sport      Wt  
##   <chr> <dbl>  
## 1 Field   123.  
## 2 BBall   114.  
## 3 Field   111.  
## 4 Field   108.  
## 5 Field   103.  
## 6 WPolo   101  
## 7 BBall   100.
```

Another way

```
athletes %>%
  slice_max(order_by = Wt, n=7) %>%
  select(Sport, Wt)
```

```
## # A tibble: 7 x 2
##   Sport      Wt
##   <chr> <dbl>
## 1 Field   123.
## 2 BBall   114.
## 3 Field   111.
## 4 Field   108.
## 5 Field   103.
## 6 WPolo    101
## 7 BBall    100.
```

Create new variables from old ones

```
athletes %>%
  mutate(wt_lb = Wt * 2.2) %>%
  select(Sport, Sex, Wt, wt_lb) %>%
  arrange(Wt)
```

```
## # A tibble: 202 x 4
##   Sport      Sex      Wt wt_lb
##   <chr>    <chr> <dbl> <dbl>
## 1 Gym      female  37.8  83.2
## 2 Gym      female  43.8  96.4
## 3 Gym      female  45.1  99.2
## 4 Tennis   female  45.8  101.
## 5 Tennis   female  47.4  104.
## 6 Gym      female  47.8  105.
## 7 T400m    female  49.2  108.
## 8 Row      female  49.8  110.
## 9 T400m    female  50.0  110.
```

Turning the result into a number

Output is always data frame unless you explicitly turn it into something else, eg. the weight of the heaviest athlete, as a number:

```
athletes %>% arrange(desc(Wt)) %>% pluck("Wt", 1)
```

```
## [1] 123.2
```

Or the 20 heaviest weights in descending order:

```
athletes %>%  
  arrange(desc(Wt)) %>%  
  slice(1:20) %>%  
  pluck("Wt")
```

```
## [1] 123.20 113.70 111.30 108.20 102.70 101.00  
## [7] 100.20 98.00 97.90 97.90 97.00 96.90  
## [13] 96.30 94.80 94.80 94.70 94.70 94.60  
## [19] 94.25 94.20
```

Another way to do the last one

```
athletes %>%
  arrange(desc(Wt)) %>%
  slice(1:20) %>%
  pull("Wt")
```

```
## [1] 123.20 113.70 111.30 108.20 102.70 101.00
## [7] 100.20  98.00  97.90  97.90  97.00  96.90
## [13]  96.30  94.80  94.80  94.70  94.70  94.60
## [19]  94.25  94.20
```

`pull` grabs the column you name *as a vector* (of whatever it contains).

To find the mean height of the women athletes

Two ways:

```
athletes %>% group_by(Sex) %>% summarize(m = mean(Ht))
```

```
## # A tibble: 2 x 2
##   Sex      m
##   <chr>  <dbl>
## 1 female 175.
## 2 male   186.
```

```
athletes %>%
  filter(Sex == "female") %>%
  summarize(m = mean(Ht))
```

```
## # A tibble: 1 x 1
##       m
##   <dbl>
## 1 175.
```

Summary of data selection/arrangement “verbs”

Verb	Purpose
<code>select</code>	Choose columns
<code>print</code>	Display non-default # of rows/columns
<code>slice</code>	Choose rows by number
<code>sample_n</code>	Choose random rows
<code>filter</code>	Choose rows satisfying conditions
<code>arrange</code>	Sort in order by column(s)
<code>mutate</code>	Create new variables
<code>group_by</code>	Create groups to summarize by
<code>summarize</code>	Calculate summary statistics (by groups if defined)
<code>pluck</code>	Extract items from data frame
<code>pull</code>	Extract a single column from a data frame as a vector

Looking things up in another data frame

- Suppose you are working in the nails department of a hardware store and you find that you have sold these items:

```
my_url <- "http://ritsokiguess.site/datafiles/nail_sales.csv"
sales <- read_csv(my_url)
sales
```

```
## # A tibble: 6 x 2
##   product_code sales
##   <chr>         <dbl>
## 1 061-5344-6      10
## 2 161-0090-0       6
## 3 061-5388-2       2
## 4 161-0199-4       8
## 5 061-5375-2       5
## 6 061-4525-2       3
```

Product descriptions and prices

- but you don't remember what these product codes are, and you would like to know the total revenue from these sales.
- Fortunately you found a list of product descriptions and prices:

```
my_url <- "http://ritsokiguess.site/datafiles/nail_desc.csv"
desc <- read_csv(my_url)
desc
```

```
## # A tibble: 7 x 5
```

	product_code	description	size	qty	price
	<chr>	<chr>	<chr>	<dbl>	<dbl>
## 1	061-4525-2	spike nail	"10\""	1	1.49
## 2	061-5329-4	masonry nail	"1.5\""	112	8.19
## 3	061-5344-6	finishing nail	"1\""	1298	6.99
## 4	061-5375-2	roofing nail	"1.25\""	192	6.99
## 5	061-5388-2	framing nail	"4\""	25	8.19
## 6	161-0090-0	wood nail	"1\""	25	2.39
## 7	161-0199-4	panel nail	"1-5/8\""	20	4.69

The lookup

- How do you “look up” the product codes to find the product descriptions and prices?
- `left_join`.

```
sales %>% left_join(desc)
```

```
## Joining, by = "product_code"
```

```
## # A tibble: 6 x 6
```

```
##   product_code sales description  size      qty price
##   <chr>         <dbl> <chr>      <chr>  <dbl> <dbl>
## 1 061-5344-6      10 finishing n~ "1\" 1298  6.99
## 2 161-0090-0       6 wood nail   "1\"  25  2.39
## 3 061-5388-2       2 framing nail "4\"  25  8.19
## 4 161-0199-4       8 panel nail  "1-5/~ 20  4.69
## 5 061-5375-2       5 roofing nail "1.25~ 192  6.99
## 6 061-4525-2       3 spike nail  "10\"  1  1.49
```

What we have

- this looks up all the rows in the *first* dataframe that are also in the *second*.
- by default matches all columns with same name in two dataframes (product_code here)
- get *all* columns in *both* dataframes. The rows are the ones for that product_code.

So now can work out how much the total revenue was:

```
sales %>% left_join(desc) %>%
  mutate(product_revenue = sales*price) %>%
  summarize(total_revenue = sum(product_revenue))
```

```
## Joining, by = "product_code"
```

```
## # A tibble: 1 x 1
```

```
##   total_revenue
```

```
##           <dbl>
```

More comments

- if any product codes are not matched, you get NA in the added columns
- anything in the *second* dataframe that was not in the first does not appear (here, any products that were not sold)
- other variations (examples follow):
 - if there are two columns with the same name in the two dataframes, and you only want to match on one, use `by` with one column name
 - if the columns you want to look up have different names in the two dataframes, use `by` with a “named list”

Matching on only some matching names

- Suppose the sales dataframe *also* had a column qty (which was the quantity sold):

```
sales %>% rename("qty"="sales") -> sales1
sales1
```

```
## # A tibble: 6 x 2
##   product_code  qty
##   <chr>        <dbl>
## 1 061-5344-6      10
## 2 161-0090-0       6
## 3 061-5388-2       2
## 4 161-0199-4       8
## 5 061-5375-2       5
## 6 061-4525-2       3
```

- The qty in sales1 is the quantity sold, but the qty in desc is the number of nails in a package. These should *not* be matched: they are different things.

Matching only on product code

```
sales1 %>%
  left_join(desc, by = "product_code")
```

```
## # A tibble: 6 x 6
```

```
##   product_code qty.x description  size  qty.y price
##   <chr>         <dbl> <chr>      <chr>  <dbl> <dbl>
## 1 061-5344-6      10 finishing n~ "1\" 1298  6.99
## 2 161-0090-0       6 wood nail    "1\"  25  2.39
## 3 061-5388-2       2 framing nail "4\"  25  8.19
## 4 161-0199-4       8 panel nail   "1-5/~ 20  4.69
## 5 061-5375-2       5 roofing nail "1.25~ 192  6.99
## 6 061-4525-2       3 spike nail   "10\"  1  1.49
```

- Get qty.x (from sales1) and qty.y (from desc).

Matching on different names 1/2

- Suppose the product code in sales was just code:

```
sales %>% rename("code" = "product_code") -> sales2
sales2
```

```
## # A tibble: 6 x 2
##   code      sales
##   <chr>    <dbl>
## 1 061-5344-6    10
## 2 161-0090-0     6
## 3 061-5388-2     2
## 4 161-0199-4     8
## 5 061-5375-2     5
## 6 061-4525-2     3
```

- How to match the two product codes that have different names?

Matching on different names 2/2

- Use `by`, but like this:

```
sales2 %>%
  left_join(desc, by = c("code"="product_code"))
```

```
## # A tibble: 6 x 6
```

	code	sales	description	size	qty	price
	<chr>	<dbl>	<chr>	<chr>	<dbl>	<dbl>
## 1	061-5344~	10	finishing na~	"1\""	1298	6.99
## 2	161-0090~	6	wood nail	"1\""	25	2.39
## 3	061-5388~	2	framing nail	"4\""	25	8.19
## 4	161-0199~	8	panel nail	"1-5/8\""	20	4.69
## 5	061-5375~	5	roofing nail	"1.25\""	192	6.99
## 6	061-4525~	3	spike nail	"10\""	1	1.49

Other types of join

- `right_join`: interchanges roles, looking up keys from second dataframe in first.
- `anti_join`: give me all the rows in the first dataframe that are *not* in the second. (Use this eg. to see whether the product descriptions are incomplete.)
- `full_join`: give me all the rows in both dataframes, with missings as needed.

Full join here

```
sales %>% full_join(desc)
```

```
## Joining, by = "product_code"
```

```
## # A tibble: 7 x 6
```

```
##   product_code sales description  size    qty price
##   <chr>         <dbl> <chr>      <chr> <dbl> <dbl>
## 1 061-5344-6      10 finishing n~ "1\" 1298  6.99
## 2 161-0090-0       6 wood nail    "1\"  25  2.39
## 3 061-5388-2       2 framing nail "4\"  25  8.19
## 4 161-0199-4       8 panel nail  "1-5/~ 20  4.69
## 5 061-5375-2       5 roofing nail "1.25~ 192  6.99
## 6 061-4525-2       3 spike nail  "10\"  1  1.49
## 7 061-5329-4      NA masonry nail "1.5\" 112  8.19
```

- The missing sales for “masonry nail” says that it was in the lookup table desc, but we didn’t sell any.

Section 3

Inference part 2

Errors in testing

What can happen:

Truth	Decision	
	Do not reject	Reject null
Null true	Correct	Type I error
Null false	Type II error	Correct

Tension between truth and decision about truth (imperfect).

- Prob. of type I error denoted α . Usually fix α , eg. $\alpha = 0.05$.
- Prob. of type II error denoted β . Determined by the planned experiment. Low β good.
- Prob. of not making type II error called **power** ($= 1 - \beta$). *High power* good.

Power

- Suppose $H_0 : \theta = 10$, $H_a : \theta \neq 10$ for some parameter θ .
- Suppose H_0 wrong. What does that say about θ ?
- Not much. Could have $\theta = 11$ or $\theta = 8$ or $\theta = 496$. In each case, H_0 wrong.
- How likely a type II error is depends on what θ is:
 - If $\theta = 496$, should be able to reject $H_0 : \theta = 10$ even for small sample, so β should be small (power large).
 - If $\theta = 11$, might have hard time rejecting H_0 even with large sample, so β would be larger (power smaller).
- Power depends on true parameter value, and on sample size.
- So we play “what if”: “if θ were 11 (or 8 or 496), what would power be?”.

Figuring out power

- Time to figure out power is before you collect any data, as part of planning process.
- Need to have idea of what kind of departure from null hypothesis of interest to you, eg. average improvement of 5 points on reading test scores. (Subject-matter decision, not statistical one.)
- Then, either:
 - “I have this big a sample and this big a departure I want to detect. What is my power for detecting it?”
 - “I want to detect this big a departure with this much power. How big a sample size do I need?”

How to understand/estimate power?

- Suppose we test $H_0 : \mu = 10$ against $H_a : \mu \neq 10$, where μ is population mean.
- Suppose in actual fact, $\mu = 8$, so H_0 is wrong. We want to reject it. How likely is that to happen?
- Need population SD (take $\sigma = 4$) and sample size (take $n = 15$). In practice, get σ from pilot/previous study, and take the n we plan to use.
- Idea: draw a random sample from the true distribution, test whether its mean is 10 or not.
- Repeat previous step “many” times.
- “Simulation”.

Making it go

- Random sample of 15 normal observations with mean 8 and SD 4:

```
x = rnorm(15, 8, 4)
x
```

```
## [1] 14.487469  5.014611  6.924277  5.201860
## [5]  8.852952 10.835874  3.686684 11.165242
## [9]  8.016188 12.383518  1.378099  3.172503
## [13] 13.074996 11.353573  5.015575
```

- Test whether x from population with mean 10 or not (over):

...continued

```
t.test(x, mu = 10)
```

```
##
```

```
## One Sample t-test
```

```
##
```

```
## data: x
```

```
## t = -1.8767, df = 14, p-value = 0.08157
```

```
## alternative hypothesis: true mean is not equal to 10
```

```
## 95 percent confidence interval:
```

```
## 5.794735 10.280387
```

```
## sample estimates:
```

```
## mean of x
```

```
## 8.037561
```

- Fail to reject the mean being 10 (a Type II error).

or get just P-value

```
t.test(x, mu = 10)$p.value
```

```
## [1] 0.0815652
```

Run this lots of times

- without a loop!
- use `rowwise` to work one random sample at a time
- draw random samples from the truth
- test that $\mu = 10$
- get P-value
- Count up how many of the P-values are 0.05 or less.

In code

```
tibble(sim = 1:1000) %>%
  rowwise() %>%
  mutate(my_sample = list(rnorm(15, 8, 4))) %>%
  mutate(t_test = list(t.test(my_sample, mu = 10))) %>%
  mutate(p_val = t_test$p.value) %>%
  count(p_val <= 0.05)
```

```
## # A tibble: 2 x 2
##   `p_val <= 0.05`      n
##   <lg1>             <int>
## 1 FALSE             578
## 2 TRUE              422
```

We correctly rejected 422 times out of 1000, so the estimated power is 0.422.

Calculating power

- Simulation approach very flexible: will work for any test. But answer different each time because of randomness.
- In some cases, for example 1-sample and 2-sample t-tests, power can be calculated.
- `power.t.test`. delta difference between null and true mean:

```
power.t.test(n = 15, delta = 10-8, sd = 4, type = "one.sample")
```

```
##  
##      One-sample t test power calculation  
##  
##              n = 15  
##            delta = 2  
##             sd = 4  
##      sig.level = 0.05  
##            power = 0.4378466  
## alternative = two.sided
```


Comparison of results

Method	Power
Simulation	0.422
power.t.test	0.4378

- Simulation power is similar to calculated power; to get more accurate value, repeat more times (eg. 10,000 instead of 1,000), which takes longer.
- CI for power based on simulation approx. 0.42 ± 0.03 .
- With this small a sample size, the power is not great. With a bigger sample, the sample mean should be closer to 8 most of the time, so would reject $H_0 : \mu = 10$ more often.

Calculating required sample size

- Often, when planning a study, we do not have a particular sample size in mind. Rather, we want to know how big a sample to take. This can be done by asking how big a sample is needed to achieve a certain power.
- The simulation approach does not work naturally with this, since you have to supply a sample size.
- For the power-calculation method, you supply a value for the power, but leave the sample size missing.
- Re-use the same problem: $H_0 : \mu = 10$ against 2-sided alternative, true $\mu = 8$, $\sigma = 4$, but now aim for power 0.80.

Using power.t.test

- No n=, replaced by a power=:

```
power.t.test(power=0.80, delta=10-8, sd=4, type="one.sample")
```

```
##  
##      One-sample t test power calculation  
##  
##              n = 33.3672  
##            delta = 2  
##             sd = 4  
##      sig.level = 0.05  
##             power = 0.8  
##      alternative = two.sided
```

- Sample size must be a whole number, so round up to 34 (to get at least as much power as you want).

Power curves

- Rather than calculating power for one sample size, or sample size for one power, might want a picture of relationship between sample size and power.
- Or, likewise, picture of relationship between difference between true and null-hypothesis means and power.
- Called power curve.
- Build and plot it yourself.

Building it

- If you feed `power.t.test` a collection (“vector”) of values, it will do calculation for each one.
- Do power for variety of sample sizes, from 10 to 100 in steps of 10:

```
ns=seq(10,100,10)
ns
```

```
## [1] 10 20 30 40 50 60 70 80 90 100
```

- Calculate powers:

```
ans=power.t.test(n=ns, delta=10-8, sd=4, type="one.sample")
ans$power
```

```
## [1] 0.2928286 0.5644829 0.7539627 0.8693979
## [5] 0.9338976 0.9677886 0.9847848 0.9929987
## [9] 0.9968496 0.9986097
```

Building a plot (1/2)

- Make a data frame out of the values to plot:

```
d=tibble(n=ns, power=ans$power)
d
```

```
## # A tibble: 10 x 2
```

```
##       n power
```

```
##    <dbl> <dbl>
```

```
##  1     10 0.293
```

```
##  2     20 0.564
```

```
##  3     30 0.754
```

```
##  4     40 0.869
```

```
##  5     50 0.934
```

```
##  6     60 0.968
```

```
##  7     70 0.985
```

```
##  8     80 0.993
```

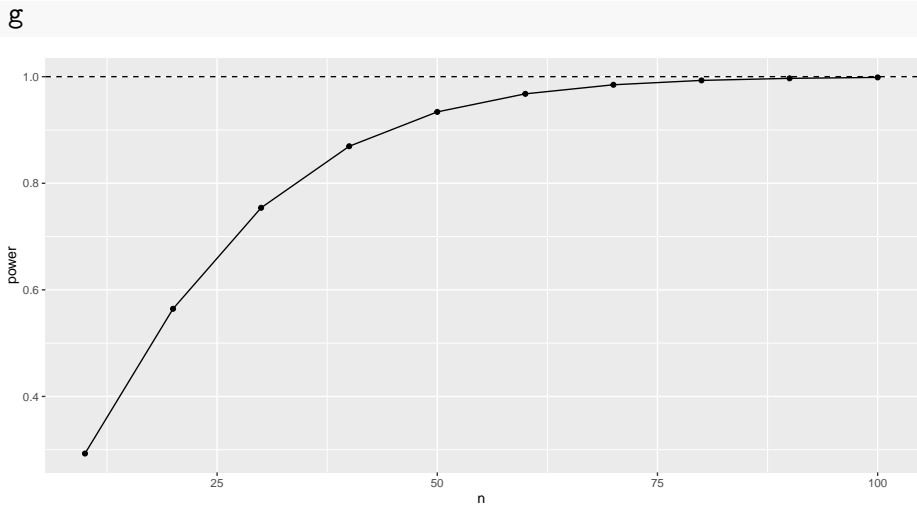
```
##  9     90 0.997
```

Building a plot (2/2)

- Plot these as points joined by lines, and add horizontal line at 1 (maximum power):

```
g = ggplot(d, aes(x = n, y = power)) + geom_point() +  
  geom_line() +  
  geom_hline(yintercept = 1, linetype = "dashed")
```

The power curve

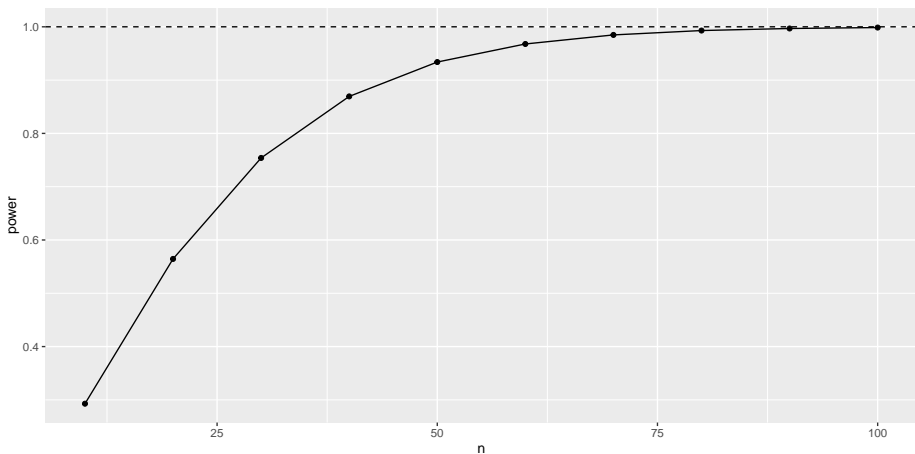


Another way to do it:

```
tibble(n=ns) %>% rowwise() %>%  
  mutate(power_output =  
    list(power.t.test(n = n, delta = 10-8, sd = 4,  
                      type = "one.sample")) %>%  
mutate(power = power_output$power) %>%  
ggplot(aes(x=n, y=power)) + geom_point() + geom_line() +  
  geom_hline(yintercept=1, linetype="dashed") -> g2
```

The power curve done the other way

g2



Power curves for means

- Can also investigate power as it depends on what the true mean is (the farther from null mean 10, the higher the power will be).
- Investigate for two different sample sizes, 15 and 30.
- First make all combos of mean and sample size:

```
means=seq(6,10,0.5)
```

```
means
```

```
## [1] 6.0 6.5 7.0 7.5 8.0 8.5 9.0 9.5 10.0
```

```
ns=c(15,30)
```

```
ns
```

```
## [1] 15 30
```

```
combos=crossing(mean=means, n=ns)
```

The combos

```
combos
```

```
## # A tibble: 18 x 2
```

```
##       mean      n
```

```
##    <dbl> <dbl>
```

```
##  1      6     15
```

```
##  2      6     30
```

```
##  3     6.5     15
```

```
##  4     6.5     30
```

```
##  5      7     15
```

```
##  6      7     30
```

```
##  7     7.5     15
```

```
##  8     7.5     30
```

```
##  9      8     15
```

```
## 10      8     30
```

```
## 11     8.5     15
```

```
## 12     8.5     30
```


More of what's in the output

```
names(ans)
```

```
## [1] "n"          "delta"      "sd"
## [4] "sig.level"  "power"      "alternative"
## [7] "note"      "method"
```

```
ans$power
```

```
## [1] 0.94908647 0.99956360 0.88277128 0.99619287
## [5] 0.77070660 0.97770385 0.61513033 0.91115700
## [9] 0.43784659 0.75396272 0.27216777 0.51028173
## [13] 0.14530058 0.26245348 0.06577280 0.09719303
## [17] 0.02500000 0.02500000
```

Make a data frame to plot, pulling things from the right places:

```
d=tibble(n=factor(combos$n), mean=combos$mean,
          power=ans$power)
```

```
d
```

```
## # A tibble: 18 x 3
```

```
##      n      mean  power
```

```
##    <fct> <dbl>  <dbl>
```

```
##  1 15      6    0.949
```

```
##  2 30      6    1.00
```

```
##  3 15     6.5  0.883
```

```
##  4 30     6.5  0.996
```

```
##  5 15      7    0.771
```

```
##  6 30      7    0.978
```

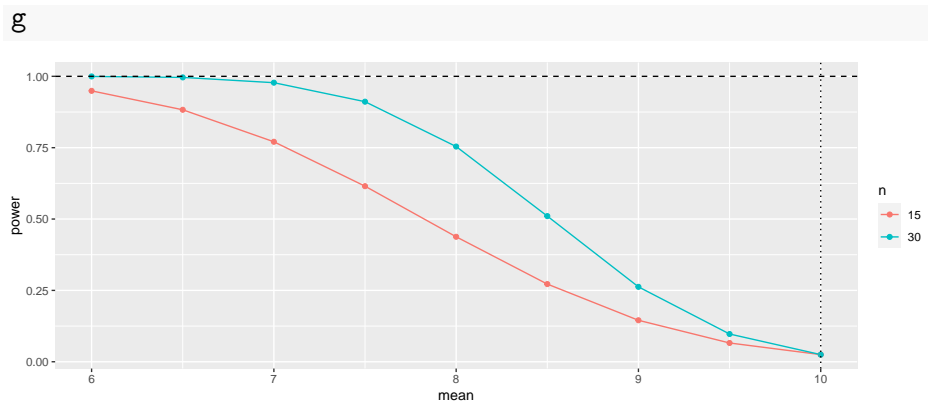
```
##  7 15     7.5  0.615
```

```
##  8 30     7.5  0.911
```

then make the plot:

```
g = ggplot(d, aes(x = mean, y = power, colour = n)) +  
  geom_point() + geom_line() +  
  geom_hline(yintercept = 1, linetype = "dashed") +  
  geom_vline(xintercept = 10, linetype = "dotted")
```


The power curves



Comments

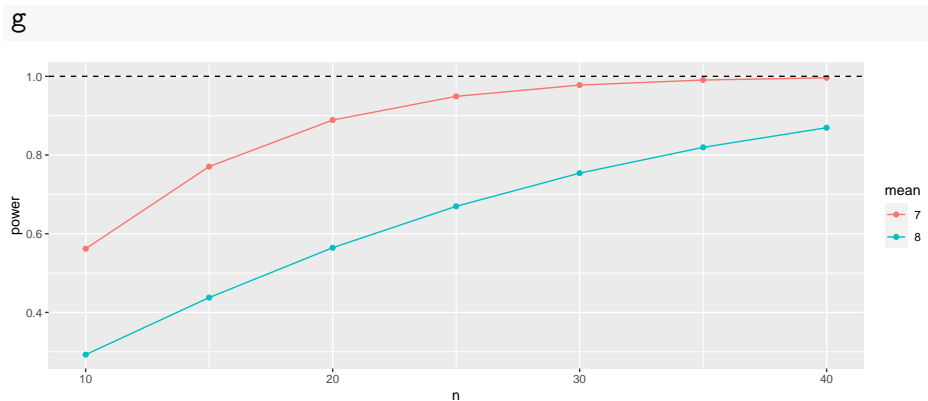
- When $\text{mean}=10$, that is, the true mean equals the null mean, H_0 is actually true, and the probability of rejecting it then is $\alpha = 0.05$.
- As the null gets more wrong (mean decreases), it becomes easier to correctly reject it.
- The blue power curve is above the red one for any mean < 10 , meaning that no matter how wrong H_0 is, you always have a greater chance of correctly rejecting it with a larger sample size.
- Previously, we had $H_0 : \mu = 10$ and a true $\mu = 8$, so a mean of 8 produces power 0.42 and 0.80 as shown on the graph.
- With $n = 30$, a true mean that is less than about 7 is almost certain to be correctly rejected. (With $n = 15$, the true mean needs to be less than 6.)

Power by sample size for means 7 and 8

Similar procedure to before:

```
means=c(7, 8)
ns=seq(10, 40, 5)
combos=crossing(mean=means, n=ns)
ans=with(combos, power.t.test(n=n, delta=10-mean, sd=4,
                             type="one.sample"))
d=tibble(mean=factor(combos$mean), n=combos$n,
          power=ans$power)
g=ggplot(d, aes(x=n, y=power, colour=mean)) +
  geom_point() + geom_line() +
  geom_hline(yintercept=1, linetype="dashed")
```

The power curves



Two-sample power

```
##
## -- Column specification -----
## cols(
##   group = col_character(),
##   score = col_double()
## )
```

- For kids learning to read, had sample sizes of 22 (approx) in each group
- and these group SDs:

```
kids
```

```
## # A tibble: 44 x 2
##   group score
##   <chr> <dbl>
## 1 t      24
## 2 t      61
## 3 +      59
```

Setting up

- suppose a 5-point improvement in reading score was considered important (on this scale)
- in a 2-sample test, null (difference of) mean is zero, so δ is true difference in means
- what is power for these sample sizes, and what sample size would be needed to get power up to 0.80?
- SD in both groups has to be same in power.t.test, so take as 14.

Calculating power for sample size 22 (per group)

```
power.t.test(n=22, delta=5, sd=14, type="two.sample",  
             alternative="one.sided")
```

```
##  
##       Two-sample t test power calculation  
##  
##               n = 22  
##             delta = 5  
##             sd = 14  
##       sig.level = 0.05  
##             power = 0.3158199  
##       alternative = one.sided  
##  
## NOTE: n is number in *each* group
```

sample size for power 0.8

```
power.t.test(power=0.80, delta=5, sd=14, type="two.sample",  
             alternative="one.sided")
```

```
##  
##      Two-sample t test power calculation  
##  
##              n = 97.62598  
##            delta = 5  
##             sd = 14  
##      sig.level = 0.05  
##             power = 0.8  
## alternative = one.sided  
##  
## NOTE: n is number in *each* group
```


Comments

- The power for the sample sizes we have is very small (to detect a 5-point increase).
- To get power 0.80, we need 98 kids in *each* group!

Section 4

The bootstrap for sampling distributions

Assessing assumptions

- Our t -tests assume normality of variable being tested
- but, Central Limit Theorem says that normality matters less if sample is “large”
- in practice “approximate normality” is enough, but how do we assess whether what we have is normal enough?
- so far, use histogram/boxplot and make a call, allowing for sample size.

What actually has to be normal

- is: **sampling distribution of sample mean**
- the distribution of sample mean over *all possible samples*
- but we only have *one* sample!
- Idea: assume our sample is representative of the population, and draw samples from our sample (!), with replacement.
- This gives an idea of what different samples from the population might look like.
- Called *bootstrap*, after expression “to pull yourself up by your own bootstraps”.

Blue Jays attendances

```
jays$attendance
```

```
## [1] 48414 17264 15086 14433 21397 34743 44794 14184
## [9] 15606 18581 19217 21519 21312 30430 42917 42419
## [17] 29306 15062 16402 19014 21195 33086 37929 15168
## [25] 17276
```

- A bootstrap sample:

```
s <- sample(jays$attendance, replace = TRUE)
s
```

```
## [1] 21195 34743 21312 44794 16402 19014 34743 21195
## [9] 17264 18581 19014 19217 34743 19217 14433 15062
## [17] 16402 15062 34743 15062 15086 15168 15086 48414
## [25] 30430
```

Getting mean of bootstrap sample

- A bootstrap sample is same size as original, but contains repeated values (eg. 15062) and missing ones (42917).
- We need the mean of our bootstrap sample:

```
mean(s)
```

```
## [1] 23055.28
```

- This is a little different from the mean of our actual sample:

```
mean(jays$attendance)
```

```
## [1] 25070.16
```

- Want a sense of how the sample mean might vary, if we were able to take repeated samples from our population.
- Idea: take lots of *bootstrap* samples, and see how *their* sample means vary.

Setting up bootstrap sampling

- Begin by setting up a dataframe that contains a row for each bootstrap sample. I usually call this column `sim`. Do just 4 to get the idea:

```
tibble(sim = 1:4)
```

```
## # A tibble: 4 x 1
##   sim
##   <int>
## 1     1
## 2     2
## 3     3
## 4     4
```

Drawing the bootstrap samples

- Then set up to work one row at a time, and draw a bootstrap sample of the attendances in each row:

```
tibble(sim = 1:4) %>%
  rowwise() %>%
  mutate(sample = list(sample(jays$attendance, replace = TRUE)))
```

```
## # A tibble: 4 x 2
##   sim sample
##   <int> <list>
## 1     1 <dbl [25]>
## 2     2 <dbl [25]>
## 3     3 <dbl [25]>
## 4     4 <dbl [25]>
```

- Each row of our dataframe contains *all* of a bootstrap sample of 25 observations drawn with replacement from the attendances.

Sample means

- Find the mean of each sample:

```
tibble(sim = 1:4) %>%
  rowwise() %>%
  mutate(sample = list(sample(jays$attendance, replace = TRUE))) %>%
  mutate(my_mean = mean(sample))
```

```
## # A tibble: 4 x 3
##   sim sample      my_mean
##   <int> <list>      <dbl>
## 1     1 <dbl [25]>  28472.
## 2     2 <dbl [25]>  28648.
## 3     3 <dbl [25]>  23329.
## 4     4 <dbl [25]>  24808.
```

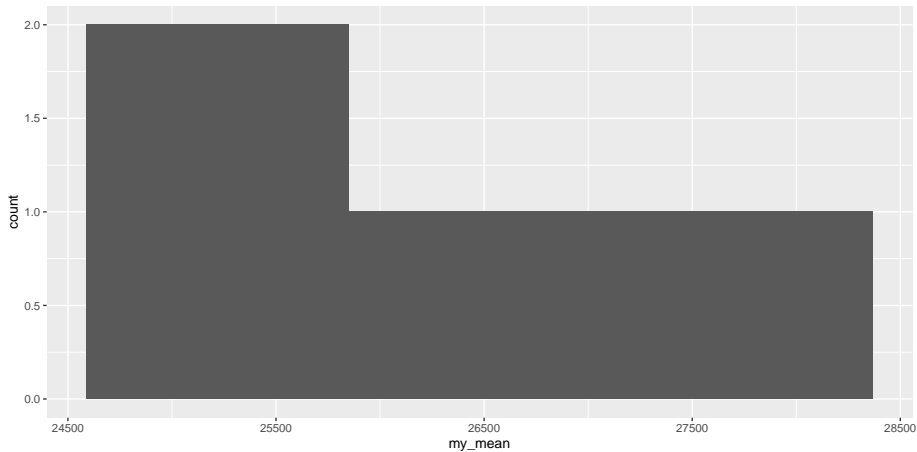
- These are (four simulated values of) the bootstrapped sampling distribution of the sample mean.

Make a histogram of them

- rather pointless here, but to get the idea:

```
tibble(sim = 1:4) %>%  
  rowwise() %>%  
  mutate(sample = list(sample(jays$attendance, replace = TRUE))) %>%  
  mutate(my_mean = mean(sample)) %>%  
  ggplot(aes(x = my_mean)) + geom_histogram(bins = 3) -> g
```

The (pointless) histogram

 σ 

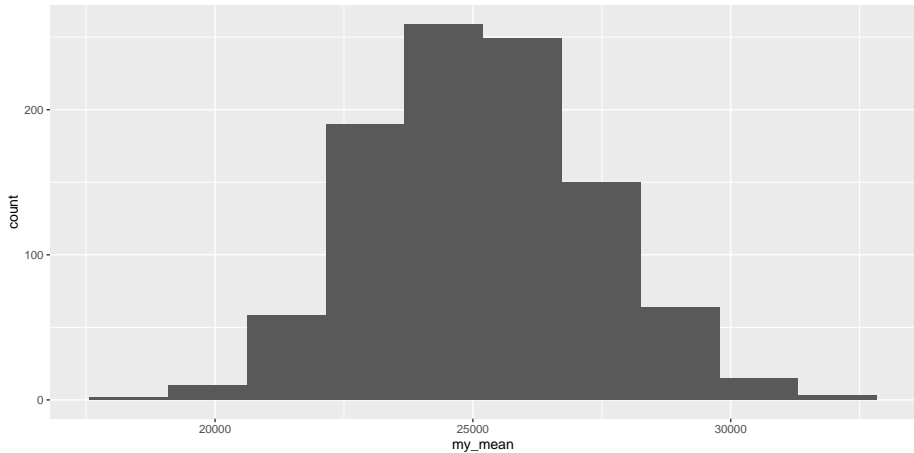
Now do again with a decent number of bootstrap samples

- say 1000, and put a decent number of bins on the histogram also:

```
tibble(sim = 1:1000) %>%  
  rowwise() %>%  
  mutate(sample = list(sample(jays$attendance, replace = TRUE))) %>%  
  mutate(my_mean = mean(sample)) %>%  
  ggplot(aes(x = my_mean)) + geom_histogram(bins = 10) -> g
```

The (better) histogram

a



Comments

- This is very close to normal
- The bootstrap says that the sampling distribution of the sample mean is close to normal, even though the distribution of the data is not
- A sample size of 25 is big enough to overcome the skewness that we saw
- This is the Central Limit Theorem in practice
- It is surprisingly powerful.
- Thus, the t -test is actually perfectly good here.

Comments on the code

- You might have been wondering about this:

```
tibble(sim = 1:4) %>%
  rowwise() %>%
  mutate(sample = list(sample(jays$attendance, replace = TRUE)))
```

```
## # A tibble: 4 x 2
##   sim sample
##   <int> <list>
## 1     1 <dbl [25]>
## 2     2 <dbl [25]>
## 3     3 <dbl [25]>
## 4     4 <dbl [25]>
```

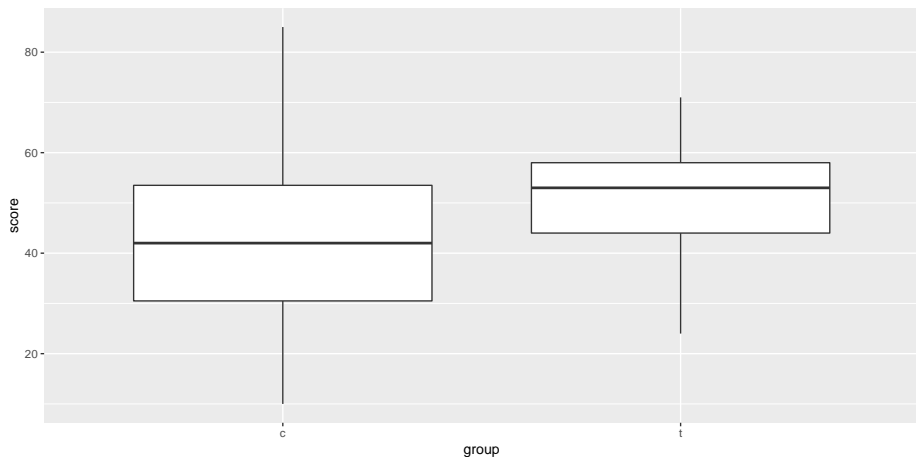
- how did we squeeze all 25 sample values into one cell?
 - sample is a so-called “list-column” that can contain anything.
- why did we have to put list() around the sample()?
 - because sample produces a collection of numbers, not just a single one
 - the list() signals this: “make a list-column of samples”.

Two samples

- Assumption: *both* samples are from a normal distribution.
- In practice, each sample is “normal enough” given its sample size, since Central Limit Theorem will help.
- Use bootstrap on each group independently, as above.

Kids learning to read

```
ggplot(kids, aes(x=group, y=score)) + geom_boxplot()
```



Getting just the control group

- Use filter to select rows where something is true:

```
kids %>% filter(group=="c") -> controls
controls
```

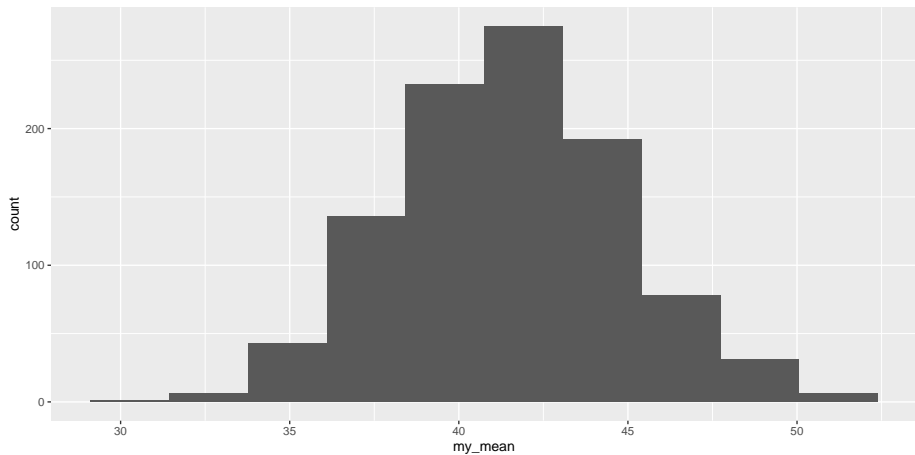
```
## # A tibble: 23 x 2
##   group score
##   <chr> <dbl>
## 1 c      42
## 2 c      33
## 3 c      46
## 4 c      37
## 5 c      43
## 6 c      41
## 7 c      10
## 8 c      42
## 9 c      55
```

Bootstrap these

```
tibble(sim = 1:1000) %>%  
  rowwise() %>%  
  mutate(sample = list(sample(controls$score, replace = TRUE))) %>%  
  mutate(my_mean = mean(sample)) %>%  
  ggplot(aes(x = my_mean)) + geom_histogram(bins = 10) -> g
```

Plot

g

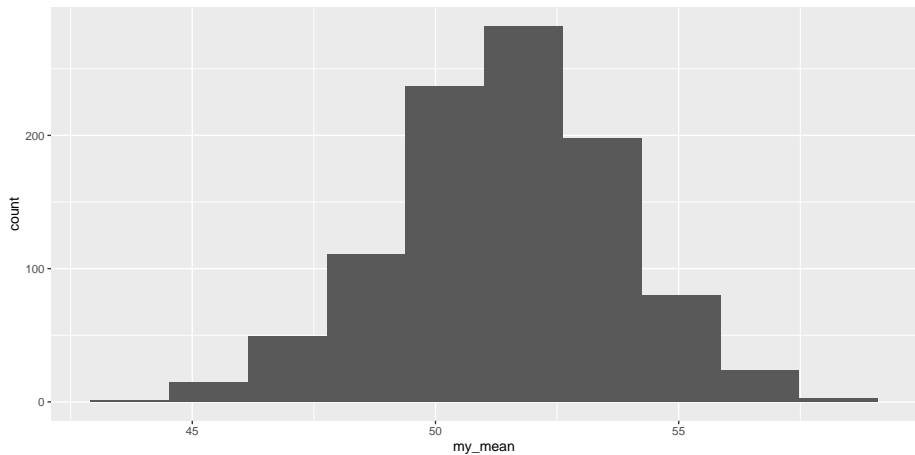


... and the treatment group:

```
kids %>% filter(group=="t") -> treats
tibble(sim = 1:1000) %>%
  rowwise() %>%
  mutate(sample = list(sample(treats$score, replace = TRUE))) %>%
  mutate(my_mean = mean(sample)) %>%
  ggplot(aes(x = my_mean)) + geom_histogram(bins = 10) -> g
```

Histogram

09



Comments

- sampling distributions of sample means both look pretty normal
- as we thought, no problems with our two-sample t at all.

Section 5

Inference part 3

Duality between confidence intervals and hypothesis tests

- Tests and CIs really do the same thing, if you look at them the right way. They are both telling you something about a parameter, and they use same things about data.
- To illustrate, some data (two groups):

```
my_url <- "http://ritsokiguess.site/datafiles/duality.txt"  
twogroups <- read_delim(my_url, " ")
```

```
##  
## -- Column specification -----  
## cols(  
##   y = col_double(),  
##   group = col_double()  
## )
```

The data

```
twogroups
```

```
## # A tibble: 15 x 2
```

```
##       y group
```

```
##   <dbl> <dbl>
```

```
## 1     10     1
```

```
## 2     11     1
```

```
## 3     11     1
```

```
## 4     13     1
```

```
## 5     13     1
```

```
## 6     14     1
```

```
## 7     14     1
```

```
## 8     15     1
```

```
## 9     16     1
```

```
## 10    13     2
```

```
## 11    13     2
```

```
## 12    14     2
```

```
## 13    17     2
```

```
## 14    18     2
```

```
## 15    19     2
```

95% CI (default)

```
t.test(y ~ group, data = twogroups)
```

```
##
```

```
## Welch Two Sample t-test
```

```
##
```

```
## data: y by group
```

```
## t = -2.0937, df = 8.7104, p-value = 0.0668
```

```
## alternative hypothesis: true difference in means is not equal to 0
```

```
## 95 percent confidence interval:
```

```
## -5.5625675 0.2292342
```

```
## sample estimates:
```

```
## mean in group 1 mean in group 2
```

```
## 13.00000 15.66667
```

90% CI

```
t.test(y ~ group, data = twogroups, conf.level = 0.90)

##
##  Welch Two Sample t-test
##
## data:  y by group
## t = -2.0937, df = 8.7104, p-value = 0.0668
## alternative hypothesis: true difference in means is not equal to 0
## 90 percent confidence interval:
##  -5.010308 -0.323025
## sample estimates:
## mean in group 1 mean in group 2
##           13.00000           15.66667
```

Hypothesis test

Null is that difference in means is zero:

```
t.test(y ~ group, mu=0, data = twogroups)
```

```
##  
## Welch Two Sample t-test  
##  
## data: y by group  
## t = -2.0937, df = 8.7104, p-value = 0.0668  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -5.5625675 0.2292342  
## sample estimates:  
## mean in group 1 mean in group 2  
## 13.00000 15.66667
```

Comparing results

Recall null here is $H_0 : \mu_1 - \mu_2 = 0$. P-value 0.0668.

- 95% CI from -5.6 to 0.2 , contains 0.
- 90% CI from -5.0 to -0.3 , does not contain 0.
- At $\alpha = 0.05$, would not reject H_0 since P-value > 0.05 .
- At $\alpha = 0.10$, *would* reject H_0 since P-value < 0.10 .

Not just coincidence. Let $C = 100(1 - \alpha)$, so C% gives corresponding CI to level- α test. Then following always true. (\iff means “if and only if”.)

Reject H_0 at level α	\iff	$C\%$ CI does not contain H_0 value
Do not reject H_0 at level α	\iff	$C\%$ CI contains H_0 value

Idea: “Plausible” parameter value inside CI, not rejected; “Implausible” parameter value outside CI, rejected.

The value of this

- If you have a test procedure but no corresponding CI:
- you make a CI by including all the parameter values that would not be rejected by your test.
- Use:
 - $\alpha = 0.01$ for a 99% CI,
 - $\alpha = 0.05$ for a 95% CI,
 - $\alpha = 0.10$ for a 90% CI, and so on.

Testing for non-normal data

- The IRS (“Internal Revenue Service”) is the US authority that deals with taxes (like Revenue Canada).
- One of their forms is supposed to take no more than 160 minutes to complete. A citizen’s organization claims that it takes people longer than that on average.
- Sample of 30 people; time to complete form recorded.
- Read in data, and do t -test of $H_0 : \mu = 160$ vs. $H_a : \mu > 160$.
- For reading in, there is only one column, so can pretend it is delimited by anything.

Read in data

```
my_url <- "http://ritsokiguess.site/datafiles/irs.txt"
irs <- read_csv(my_url)
irs
```

```
## # A tibble: 30 x 1
```

```
##       Time
```

```
##    <dbl>
```

```
##  1      91
```

```
##  2      64
```

```
##  3     243
```

```
##  4     167
```

```
##  5     123
```

```
##  6      65
```

```
##  7      71
```

```
##  8     204
```

```
##  9     110
```

```
## 10     172
```

Test whether mean is 160 or greater

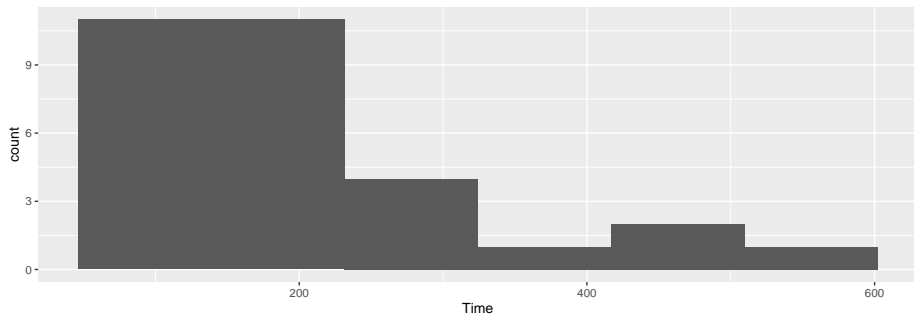
```
with(irs, t.test(Time, mu = 160,  
                  alternative = "greater"))
```

```
##  
## One Sample t-test  
##  
## data: Time  
## t = 1.8244, df = 29, p-value = 0.03921  
## alternative hypothesis: true mean is greater than 160  
## 95 percent confidence interval:  
## 162.8305 Inf  
## sample estimates:  
## mean of x  
## 201.2333
```

Reject null; mean (for all people to complete form) greater than 160.

But, look at a graph

```
ggplot(irs, aes(x = Time)) + geom_histogram(bins = 6)
```



Comments

- Skewed to right.
- Should look at *median*, not mean.

The sign test

- But how to test whether the median is greater than 160?
- Idea: if the median really is 160 (H_0 true), the sampled values from the population are equally likely to be above or below 160.
- If the population median is greater than 160, there will be a lot of sample values greater than 160, not so many less. Idea: test statistic is number of sample values greater than hypothesized median.

Getting a P-value for sign test 1/3

- How to decide whether “unusually many” sample values are greater than 160? Need a sampling distribution.
- If H_0 true, pop. median is 160, then each sample value independently equally likely to be above or below 160.
- So number of observed values above 160 has binomial distribution with $n = 30$ (number of data values) and $p = 0.5$ (160 is hypothesized to be *median*).

Getting P-value for sign test 2/3

- Count values above/below 160:

```
irs %>% count(Time > 160)
```

```
## # A tibble: 2 x 2
##   `Time > 160`      n
##   <lgl>          <int>
## 1 FALSE           13
## 2 TRUE            17
```

- 17 above, 13 below. How unusual is that? Need a *binomial table*.

Getting P-value for sign test 3/3

- R function `dbinom` gives the probability of eg. exactly 17 successes in a binomial with $n = 30$ and $p = 0.5$:

```
dbinom(17, 30, 0.5)
```

```
## [1] 0.1115351
```

- but we want probability of 17 *or more*, so get all of those, find probability of each, and add them up:

```
tibble(x=17:30) %>%  
  mutate(prob=dbinom(x, 30, 0.5)) %>%  
  summarize(total=sum(prob))
```

```
## # A tibble: 1 x 1  
##   total  
##   <dbl>  
## 1 0.292
```


Using my package smmr

- I wrote a package `smmr` to do the sign test (and some other things). Installation is a bit fiddly:
 - Install devtools with `install.packages("devtools")`
 - then install `smmr`:

```
library(devtools)
install_github("nxskok/smmr")
```

- Then load it:

```
library(smmr)
```

smmr for sign test

- smmr's function `sign_test` needs three inputs: a data frame, a column and a null median:

```
sign_test(irs, Time, 160)
```

```
## $above_below
## below above
##      13      17
##
## $p_values
##   alternative   p_value
## 1         lower 0.8192027
## 2         upper 0.2923324
## 3    two-sided 0.5846647
```

Comments (1/3)

- Testing whether population median *greater than* 160, so want *upper-tail* P-value 0.2923. Same as before.
- Also get table of values above and below; this too as we got.

Comments (2/3)

- P-values are:

Test	P-value
t	0.0392
Sign	0.2923

- These are very different: we reject a mean of 160 (in favour of the mean being bigger), but clearly *fail* to reject a median of 160 in favour of a bigger one.
- Why is that? Obtain mean and median:

```
irs %>% summarize(mean_time = mean(Time),
                  median_time = median(Time))
```

```
## # A tibble: 1 x 2
##   mean_time median_time
##   <dbl>         <dbl>
## 1      201.         172.
```

Comments (3/3)

- The mean is pulled a long way up by the right skew, and is a fair bit bigger than 160.
- The median is quite close to 160.
- We ought to be trusting the sign test and not the t-test here (median and not mean), and therefore there is no evidence that the “typical” time to complete the form is longer than 160 minutes.
- Having said that, there are clearly some people who take a lot longer than 160 minutes to complete the form, and the IRS could focus on simplifying its form for these people.
- In this example, looking at any kind of average is not really helpful; a better question might be “do an unacceptably large fraction of people take longer than (say) 300 minutes to complete the form?”: that is, thinking about worst-case rather than average-case.

Confidence interval for the median

- The sign test does not naturally come with a confidence interval for the median.
- So we use the “duality” between test and confidence interval to say: the (95%) confidence interval for the median contains exactly those values of the null median that would not be rejected by the two-sided sign test (at $\alpha = 0.05$).

For our data

- The procedure is to try some values for the null median and see which ones are inside and which outside our CI.
- `smmr` has `pval_sign` that gets just the 2-sided P-value:

```
pval_sign(160, irs, Time)
```

```
## [1] 0.5846647
```

- Try a couple of null medians:

```
pval_sign(200, irs, Time)
```

```
## [1] 0.3615946
```

```
pval_sign(300, irs, Time)
```

```
## [1] 0.001430906
```

- So 200 inside the 95% CI and 300 outside.

Doing a whole bunch

- Choose our null medians first:

```
(d <- tibble(null_median=seq(100,300,20)))
```

```
## # A tibble: 11 x 1
##   null_median
##   <dbl>
## 1      100
## 2      120
## 3      140
## 4      160
## 5      180
## 6      200
## 7      220
## 8      240
## 9      260
## 10     280
## 11     300
```


... and then

“for each null median, run the function `pval_sign` for that null median and get the P-value”:

```
d %>% rowwise() %>%  
  mutate(p_value = pval_sign(null_median, irs, Time))
```

```
## # A tibble: 11 x 2  
##   null_median p_value  
##   <dbl>      <dbl>  
## 1      100 0.000325  
## 2      120 0.0987  
## 3      140 0.200  
## 4      160 0.585  
## 5      180 0.856  
## 6      200 0.362  
## 7      220 0.0428  
## 8      240 0.0161
```

Make it easier for ourselves

```
d %>% rowwise() %>%
  mutate(p_value = pval_sign(null_median, irs, Time)) %>%
  mutate(in_out = ifelse(p_value > 0.05, "inside", "outside"))
```

```
## # A tibble: 11 x 3
##   null_median p_value in_out
##   <dbl>      <dbl> <chr>
## 1      100 0.000325 outside
## 2      120 0.0987   inside
## 3      140 0.200    inside
## 4      160 0.585    inside
## 5      180 0.856    inside
## 6      200 0.362    inside
## 7      220 0.0428   outside
## 8      240 0.0161   outside
## 9      260 0.00522  outside
## 10     280 0.00140  inside
```

confidence interval for median?

- 95% CI to this accuracy from 120 to 200.
- Can get it more accurately by looking more closely in intervals from 100 to 120, and from 200 to 220.

A more efficient way: bisection

- Know that top end of CI between 200 and 220:

```
lo=200  
hi=220
```

- Try the value halfway between: is it inside or outside?

```
(try = (lo + hi) / 2)
```

```
## [1] 210
```

```
pval_sign(try,irs,Time)
```

```
## [1] 0.09873715
```

- Inside, so upper end is between 210 and 220. Repeat (over):

... bisection continued

```
lo = try  
(try = (lo + hi) / 2)
```

```
## [1] 215
```

```
pval_sign(try, irs, Time)
```

```
## [1] 0.06142835
```

- 215 is inside too, so upper end between 215 and 220.
- Continue until have as accurate a result as you want.

Bisection automatically

- A loop, but not a for since we don't know how many times we're going around. Keep going while a condition is true:

```
lo = 200
hi = 220
while (hi - lo > 1) {
  try = (hi + lo) / 2
  ptry = pval_sign(try, irs, Time)
  print(c(try, ptry))
  if (ptry <= 0.05)
    hi = try
  else
    lo = try
}
```

The output from this loop

```
## [1] 210.00000000    0.09873715
## [1] 215.00000000    0.06142835
## [1] 217.50000000    0.04277395
## [1] 216.25000000    0.04277395
## [1] 215.62500000    0.04277395
```

- 215 inside, 215.625 outside. Upper end of interval to this accuracy is 215.

Using smmr

- smmr has function `ci_median` that does this (by default 95% CI):

```
ci_median(irs,Time)
```

```
## [1] 119.0065 214.9955
```

- Uses a more accurate bisection than we did.
- Or get, say, 90% CI for median:

```
ci_median(irs,Time,conf.level=0.90)
```

```
## [1] 123.0031 208.9960
```

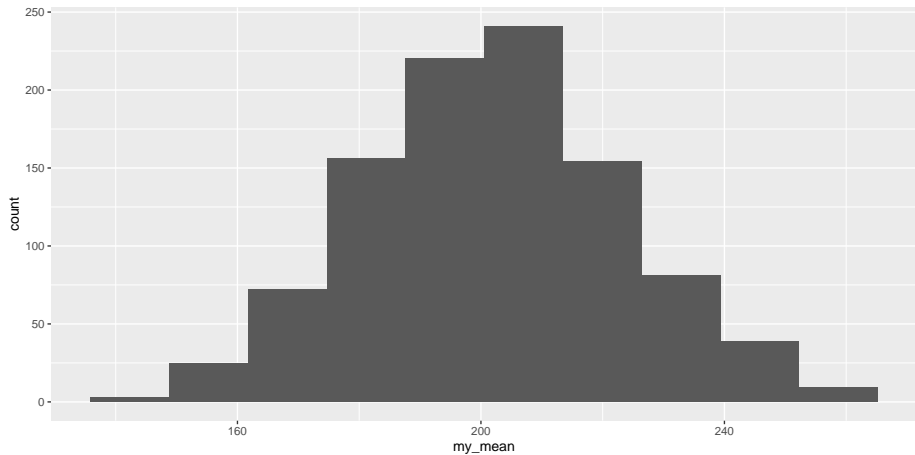
- 90% CI is shorter, as it should be.

Bootstrap (optional)

- but, was the sample size (30) big enough to overcome the skewness?
- Bootstrap, again:

```
tibble(sim = 1:1000) %>%  
  rowwise() %>%  
  mutate(my_sample = list(sample(irs$Time, replace = TRUE))) %>%  
  mutate(my_mean = mean(my_sample)) %>%  
  ggplot(aes(x=my_mean)) + geom_histogram(bins=10) -> g
```

The sampling distribution

 σ 

Comments

- A little skewed to right, but not nearly as much as I was expecting.
- The t -test for the mean might actually be OK for these data, *if the mean is what you want*.
- In actual data, mean and median very different; we chose to make inference about the median.
- Thus for us it was right to use the sign test.

Section 6

Inference part 5

Two-sample test: What to do if normality fails

- If normality fails (for one or both of the groups), what do we do then?
- Again, can compare medians: use the thought process of the sign test, which does not depend on normality and is not damaged by outliers.
- A suitable test called Mood's median test.
- Before we get to that, a diversion.

The chi-squared test for independence

Suppose we want to know whether people are in favour of having daylight savings time all year round. We ask 20 males and 20 females whether they each agree with having DST all year round ("yes") or not ("no"). Some of the data:

```
my_url <- "http://ritsokiguess.site/datafiles/dst.txt"
dst <- read_delim(my_url, " ")
dst %>% slice_sample(n=5) # randomly sample 5 rows
```

```
## # A tibble: 5 x 2
##   gender agree
##   <chr>   <chr>
## 1 male    yes
## 2 male    yes
## 3 male    yes
## 4 female yes
## 5 male    no
```

... continued

Count up individuals in each category combination, and arrange in contingency table:

```
tab=with(dst,table(gender,agree))
tab
```

```
##           agree
## gender    no  yes
##  female  11   9
##   male    3  17
```

- Most of the males say “yes”, but the females are about evenly split.
- Looks like males more likely to say “yes”, ie. an association between gender and agreement.
- Test an H_0 of “no association” (“independence”) vs. alternative that there is really some association.
- Done with `chisq.test`.

...And finally

```
chisq.test(tab,correct=F)
```

```
##  
##  Pearson's Chi-squared test  
##  
## data:  tab  
## X-squared = 7.033, df = 1, p-value = 0.008002
```

- Reject null hypothesis of no association
- therefore there is a difference in rates of agreement between (all) males and females (or that gender and agreement are associated).
- Without `correct=F` uses “Yates correction”; this way, should give same answers as calculated by hand (if you know how).

Mood's median test

- Before our diversion, we wanted to compare medians of two groups.
- Recall sign test: count number of values above and below something (there, hypothesized median).
- Idea of Mood's median test:
 - Work out the median of all the data, regardless of group ("grand median").
 - Count how many data values in each group are above/below this grand median.
 - Make contingency table of group vs. above/below.
 - Test for association.
- If group medians equal, each group should have about half its observations above/below grand median. If not, one group will be mostly above grand median and other below.

Mood's median test for reading data

- Find overall median score:

```
(kids %>% summarize(med=median(score)) %>% pull(med) -> m)
```

```
## [1] 47
```

- Make table of above/below vs. group:

```
tab=with(kids, table(group, score>m))
tab
```

```
##
```

```
## group FALSE TRUE
```

```
##      c      15      8
```

```
##      t       7     14
```

- Treatment group scores mostly above median, control group scores mostly below, as expected.

The test

- Do chi-squared test:

```
chisq.test(tab,correct=F)
```

```
##
```

```
## Pearson's Chi-squared test
```

```
##
```

```
## data:  tab
```

```
## X-squared = 4.4638, df = 1, p-value = 0.03462
```

- This test actually two-sided (tests for any association).
- Here want to test that new reading method *better* (one-sided).
- Most of treatment children above overall median, so do 1-sided test by halving P-value to get 0.017.
- This way too, children do better at learning to read using the new method.

Or by smmr

- `median_test` does the whole thing:

```
median_test(kids,score,group)
```

```
## $table
##      above
## group above below
##      c      8      15
##      t     14       7
##
## $test
##      what      value
## 1 statistic 4.46376812
## 2          df 1.00000000
## 3    P-value 0.03462105
```

- P-value again two-sided.

Comments

- P-value 0.013 for (1-sided) t -test, 0.017 for (1-sided) Mood median test.
- Like the sign test, Mood's median test doesn't use the data very efficiently (only, is each value above or below grand median).
- Thus, if we can justify doing t -test, we should do it. This is the case here.
- The t -test will usually give smaller P-value because it uses the data more efficiently.
- The time to use Mood's median test is if we are definitely unhappy with the normality assumption (and thus the t -test P-value is not to be trusted).

Jumping rats

- Link between exercise and healthy bones (many studies).
- Exercise stresses bones and causes them to get stronger.
- Study (Purdue): effect of jumping on bone density of growing rats.
- 30 rats, randomly assigned to 1 of 3 treatments:
 - No jumping (control)
 - Low-jump treatment (30 cm)
 - High-jump treatment (60 cm)
- 8 weeks, 10 jumps/day, 5 days/week.
- Bone density of rats (mg/cm^3) measured at end.
- See whether larger amount of exercise (jumping) went with higher bone density.
- Random assignment: rats in each group similar in all important ways.
- So entitled to draw conclusions about cause and effect.

Reading the data

Values separated by spaces:

```
my_url <- "http://ritsokiguess.site/datafiles/jumping.txt"
rats <- read_delim(my_url, " ")
```

```
##
## -- Column specification -----
## cols(
##   group = col_character(),
##   density = col_double()
## )
```

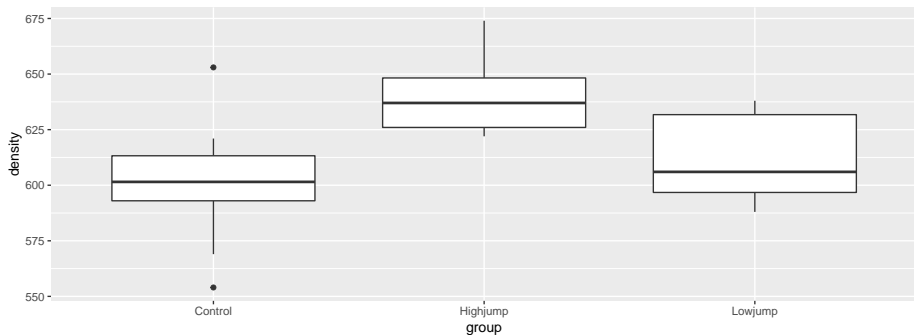
The data (some random rows)

```
rats %>% slice_sample(n=12)
```

```
## # A tibble: 12 x 2
##   group      density
##   <chr>      <dbl>
## 1 Lowjump      631
## 2 Control      614
## 3 Lowjump      596
## 4 Highjump     622
## 5 Control      569
## 6 Highjump     626
## 7 Lowjump      638
## 8 Lowjump      635
## 9 Highjump     650
## 10 Control     593
## 11 Lowjump     594
## 12 Lowjump     607
```

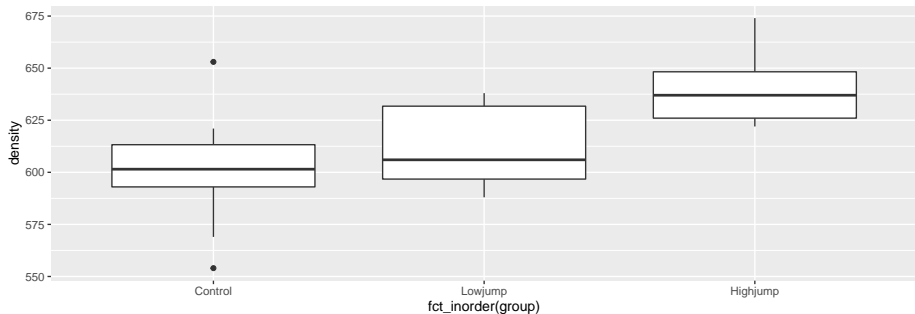

Boxplots

```
ggplot(rats, aes(y=density, x=group)) + geom_boxplot()
```



Or, arranging groups in data (logical) order

```
ggplot(rats, aes(y=density, x=fct_inorder(group))) +  
geom_boxplot()
```



Analysis of Variance

- Comparing > 2 groups of independent observations (each rat only does one amount of jumping).
- Standard procedure: analysis of variance (ANOVA).
- Null hypothesis: all groups have same mean.
- Alternative: “not all means the same”, at least one is different from others.

Testing: ANOVA in R

```
rats.aov=aov(density~group,data=rats)
summary(rats.aov)
```

```
##              Df Sum Sq Mean Sq F value Pr(>F)
## group         2    7434     3717   7.978 0.0019 **
## Residuals    27   12579       466
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

- Usual ANOVA table, small P-value: significant result.
- Conclude that the mean bone densities are not all equal.
- Reject null, but not very useful finding.

Which groups are different from which?

- ANOVA really only answers half our questions: it says “there are differences”, but doesn’t tell us which groups different.
- One possibility (not the best): compare all possible pairs of groups, via two-sample t.
- First pick out each group:

```
rats %>% filter(group=="Control") -> controls  
rats %>% filter(group=="Lowjump") -> lows  
rats %>% filter(group=="Highjump") -> highs
```

Control vs. low

```
t.test(controls$density, lows$density)
```

```
##  
## Welch Two Sample t-test  
##  
## data: controls$density and lows$density  
## t = -1.0761, df = 16.191, p-value = 0.2977  
## alternative hypothesis: true difference in means is not equ  
## 95 percent confidence interval:  
## -33.83725 11.03725  
## sample estimates:  
## mean of x mean of y  
## 601.1 612.5
```

No sig. difference here.

Control vs. high

```
t.test(controls$density, highs$density)
```

```
##  
## Welch Two Sample t-test  
##  
## data: controls$density and highs$density  
## t = -3.7155, df = 14.831, p-value = 0.002109  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -59.19139 -16.00861  
## sample estimates:  
## mean of x mean of y  
## 601.1 638.7
```

These are different.

Low vs. high

```
t.test( lows$density, highs$density )
```

```
##  
## Welch Two Sample t-test  
##  
## data:  lows$density and highs$density  
## t = -3.2523, df = 17.597, p-value = 0.004525  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -43.15242 -9.24758  
## sample estimates:  
## mean of x mean of y  
##      612.5      638.7
```

These are different too.

But...

- We just did 3 tests instead of 1.
- So we have given ourselves 3 chances to reject H_0 : all means equal, instead of 1.
- Thus α for this combined test is not 0.05.

John W. Tukey



- American statistician, 1915–2000
- Big fan of exploratory data analysis
- Invented boxplot
- Invented "honestly significant differences"
- Invented jackknife estimation
- Coined computing term "bit"
- Co-inventor of Fast Fourier Transform

Honestly Significant Differences

- Compare several groups with one test, telling you which groups differ from which.
- Idea: if all population means equal, find distribution of highest sample mean minus lowest sample mean.
- Any means unusually different compared to that declared significantly different.

Tukey on rat data

```
rats.aov=aov(density~group,data=rats)
TukeyHSD(rats.aov)
```

```
##    Tukey multiple comparisons of means
##      95% family-wise confidence level
##
## Fit: aov(formula = density ~ group, data = rats)
##
## $group
```

	diff	lwr	upr	p adj
Highjump-Control	37.6	13.66604	61.533957	0.0016388
Lowjump-Control	11.4	-12.53396	35.333957	0.4744032
Lowjump-Highjump	-26.2	-50.13396	-2.266043	0.0297843

- Again conclude that bone density for highjump group significantly higher than for other two groups.

Why Tukey's procedure better than all t-tests

Look at P-values for the two tests:

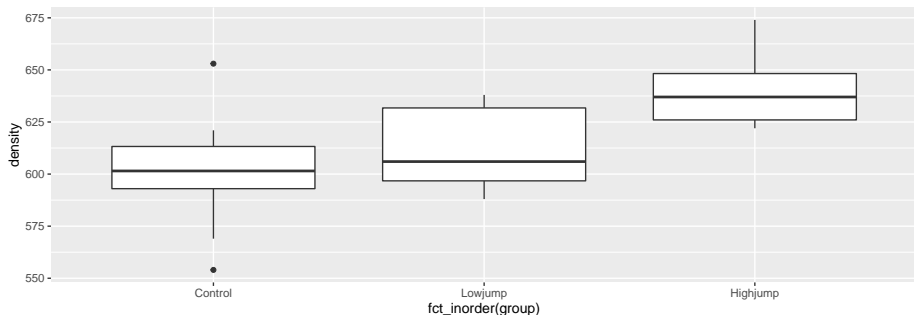
Comparison	Tukey	t-tests

Highjump-Control	0.0016	0.0021
Lowjump-Control	0.4744	0.2977
Lowjump-Highjump	0.0298	0.0045

- Tukey P-values (mostly) higher.
- Proper adjustment for doing three t-tests at once, not just one in isolation.
- lowjump-highjump comparison would no longer be significant at $\alpha = 0.01$.

Checking assumptions

```
ggplot(rats, aes(y=density, x=fct_inorder(group))) +  
geom_boxplot()
```

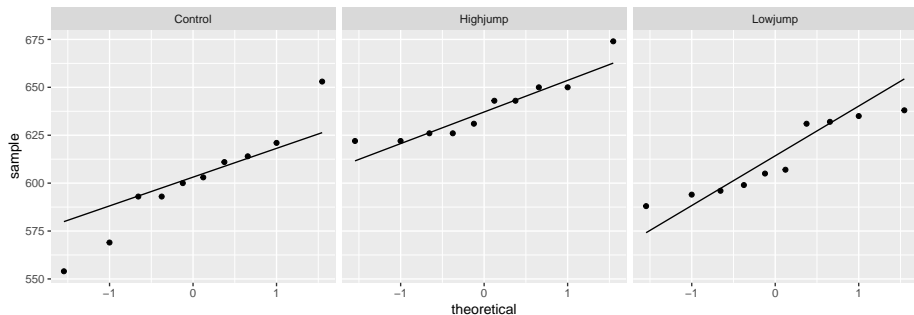


Assumptions:

- Normally distributed data within each group
- with equal group SDs.

Normal quantile plots by group

```
ggplot(rats, aes(sample = density)) + stat_qq() +  
  stat_qq_line() + facet_wrap( ~ group)
```



The assumptions

- Normally-distributed data within each group
- Equal group SDs. These are shaky here because:
 - control group has outliers
 - highjump group appears to have less spread than others. Possible remedies (in general):
- Transformation of response (usually works best when SD increases with mean)
- If normality OK but equal spreads not, can use Welch ANOVA. (Regular ANOVA like pooled t-test; Welch ANOVA like Welch-Satterthwaite t-test.)
- Can also use Mood's Median Test (see over). This works for any number of groups.

Mood's median test 1/4

- Find median of all bone densities, regardless of group:

```
(rats %>% summarize(med = median(density)) %>% pull(med) -> m)
```

```
## [1] 621.5
```

- Count up how many observations in each group above or below overall median:

```
tab = with(rats, table(group, density > m))
tab
```

```
##
```

```
## group      FALSE TRUE
```

```
## Control      9    1
```

```
## Highjump     0   10
```

```
## Lowjump      6    4
```

Mood's median test 2/4

```
tab
```

```
##
```

```
## group      FALSE TRUE
```

```
##   Control      9    1
```

```
##   Highjump     0   10
```

```
##   Lowjump      6    4
```

- All Highjump obs above overall median.
- Most Control obs below overall median.
- Suggests medians differ by group.

Mood's median test 3/4

- Test whether association between group and being above/below overall median significant using chi-squared test for association:

```
chisq.test(tab,correct=F)
```

```
##
```

```
## Pearson's Chi-squared test
```

```
##
```

```
## data:  tab
```

```
## X-squared = 16.8, df = 2, p-value = 0.0002249
```

- Very small P-value says that being above/below overall median depends on group.
- That is, groups do not all have same median.

Mood's median test 4/4

Or with `median_test` from `smmr`, same as before.

```
median_test(rats,density,group)
```

```
## $table
##           above
## group      above below
##   Control      1     9
##   Highjump     10     0
##   Lowjump      4     6
##
## $test
##           what           value
## 1 statistic 1.680000e+01
## 2          df 2.000000e+00
## 3    P-value 2.248673e-04
```

Comments

- No doubt that medians differ between groups (not all same).
- This test is equivalent of F -test, not of Tukey.
- To determine which groups differ from which, can compare all possible pairs of groups via (2-sample) Mood's median tests, then adjust P-values by multiplying by number of 2-sample Mood tests done (Bonferroni):

```
pairwise_median_test(rats,density,group)
```

```
## # A tibble: 3 x 4
##   g1      g2      p_value adj_p_value
##   <chr>   <chr>    <dbl>      <dbl>
## 1 Control Highjump 0.000148    0.000443
## 2 Control Lowjump  0.371      1
## 3 Highjump Lowjump  0.371      1
```

- Now, lowjump-highjump difference no longer significant.

Welch ANOVA

- For these data, Mood's median test probably best because we doubt both normality and equal spreads.
- When normality OK but spreads differ, Welch ANOVA way to go.
- Welch ANOVA done by `oneway.test` as shown (for illustration):

```
oneway.test(density~group,data=rats)
```

```
##
## One-way analysis of means (not assuming
## equal variances)
##
## data: density and group
## F = 8.8164, num df = 2.000, denom df =
## 17.405, p-value = 0.002268
```

- P-value very similar, as expected.
- Appropriate Tukey-equivalent here called Games-Howell.

Games-Howell

- Lives in package PMCMRplus (also userfriendlyscience). Install first.

```
library(PMCMRplus)
```

```
gamesHowellTest(density~factor(group),data=rats)
```

```
##
```

```
## Pairwise comparisons using Games-Howell test
```

```
## data: density by factor(group)
```

```
##           Control Highjump
```

```
## Highjump 0.0056  -
```

```
## Lowjump  0.5417  0.0120
```

```
##
```

```
## P value adjustment method: none
```

Deciding which test to do

For two or more samples:

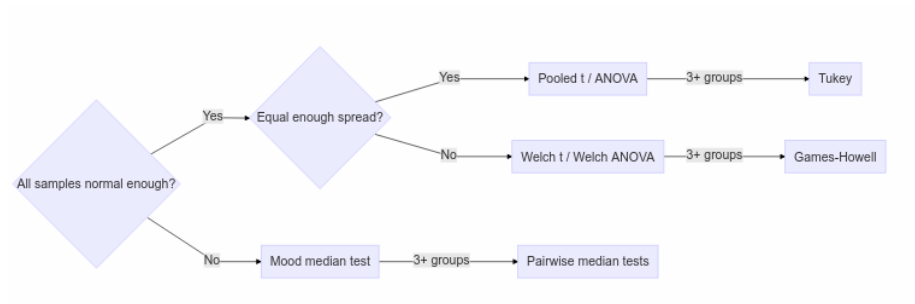


Figure 1: Test flow chart

Section 7

Case study: windmill

The windmill data

- Engineer: does amount of electricity generated by windmill depend on how strongly wind blowing?
- Measurements of wind speed and DC current generated at various times.
- Assume the “various times” to be randomly selected — aim to generalize to “this windmill at all times”.
- Research questions:
 - Relationship between wind speed and current generated?
 - If so, what kind of relationship?
 - Can we model relationship to do predictions?

Packages for this section

```
library(tidyverse)  
library(broom)
```

Reading in the data

```
my_url <-  
  "http://ritsokiguess.site/datafiles/windmill.csv"  
windmill <- read_csv(my_url)
```

```
##  
## -- Column specification -----  
## cols(  
##   wind_velocity = col_double(),  
##   DC_output = col_double()  
## )
```

The data

```
windmill
```

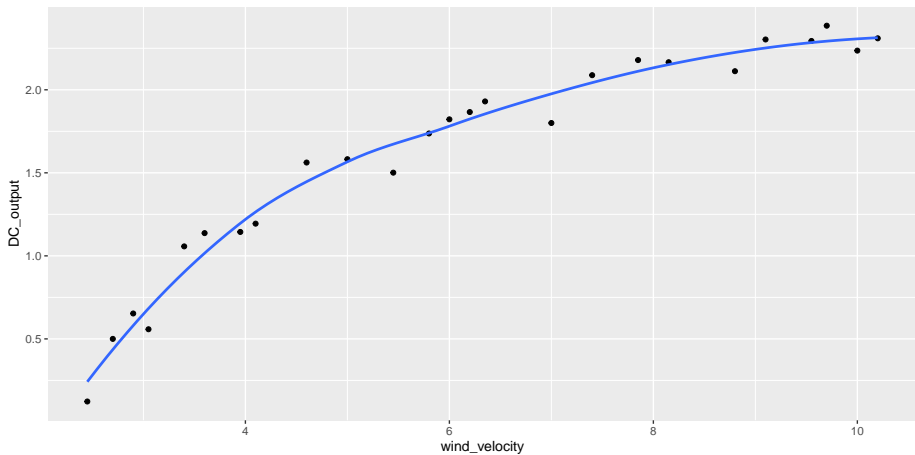
```
## # A tibble: 25 x 2
##   wind_velocity DC_output
##           <dbl>     <dbl>
## 1             5       1.58
## 2             6       1.82
## 3           3.4       1.06
## 4           2.7       0.5
## 5          10       2.24
## 6           9.7       2.39
## 7          9.55       2.29
## 8           3.05      0.558
## 9           8.15       2.17
## 10          6.2       1.87
## # ... with 15 more rows
```

Strategy

- Two quantitative variables, looking for relationship: regression methods.
- Start with picture (scatterplot).
- Fit models and do model checking, fixing up things as necessary.
- Scatterplot:
 - 2 variables, DC_output and wind_velocity.
 - First is output/response, other is input/explanatory.
 - Put DC_output on vertical scale.
- Add trend, but don't want to assume linear:

```
ggplot(windmill, aes(y = DC_output, x = wind_velocity)) +  
  geom_point() + geom_smooth(se = F)
```

Scatterplot



Comments

- Definitely a relationship: as wind velocity increases, so does DC output. (As you'd expect.)
- Is relationship linear? To help judge, `geom_smooth` smooths scatterplot trend. (Trend called “loess”, “Locally weighted least squares” which downweights outliers. Not constrained to be straight.)
- Trend more or less linear for while, then curves downwards (levelling off?). Straight line not so good here.

Fit a straight line (and see what happens)

```
DC.1 <- lm(DC_output ~ wind_velocity, data = windmill)
summary(DC.1)
```

```
##
## Call:
## lm(formula = DC_output ~ wind_velocity, data = windmill)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.59869 -0.14099  0.06059  0.17262  0.32184
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.13088    0.12599   1.039    0.31
## wind_velocity  0.24115    0.01905  12.659 7.55e-12 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2361 on 23 degrees of freedom
## Multiple R-squared:  0.8745, Adjusted R-squared:  0.869
## F-statistic: 160.3 on 1 and 23 DF,  p-value: 7.546e-12
```

Another way of looking at the output

- The standard output tends to go off the bottom of the page rather easily. Package broom has these:

```
glance(DC.1)
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic p.value
##   <dbl>      <dbl> <dbl>      <dbl>    <dbl>
## 1      0.874      0.869 0.236      160. 7.55e-12
## # ... with 7 more variables: df <dbl>,
## #   logLik <dbl>, AIC <dbl>, BIC <dbl>,
## #   deviance <dbl>, df.residual <int>, nobs <int>
```

showing that the R-squared is 87%, and

```
tidy(DC.1)
```

```
## # A tibble: 2 x 5
##   term          estimate std.error statistic p.value
##   <chr>          <dbl>      <dbl>      <dbl>    <dbl>
## 1 (Intercept)    0.131    0.126        1.04 3.10e- 1
## 2 wind_veloci~   0.241    0.0190       12.7 7.55e-12
```

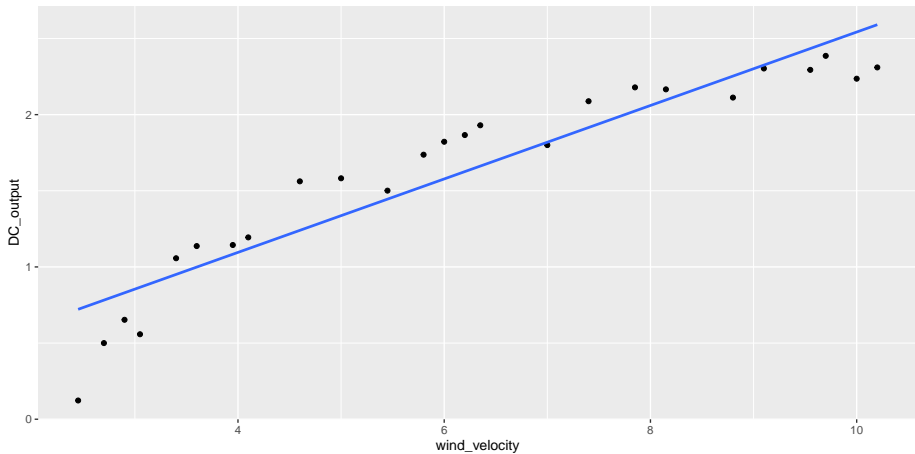
Comments

- Strategy: `lm` actually fits the regression. Store results in a variable. Then look at the results, eg. via `summary` or `glance/tidy`.
- My strategy for model names: base on response variable (or data frame name) and a number. Allows me to fit several models to same data and keep track of which is which.
- Results actually pretty good: `wind.velocity` strongly significant, R-squared (87%) high.
- How to check whether regression is appropriate? Look at the residuals, observed minus predicted, plotted against fitted (predicted).
- Plot using the regression object as “data frame” (see over).

back to scatterplot, but this time add line

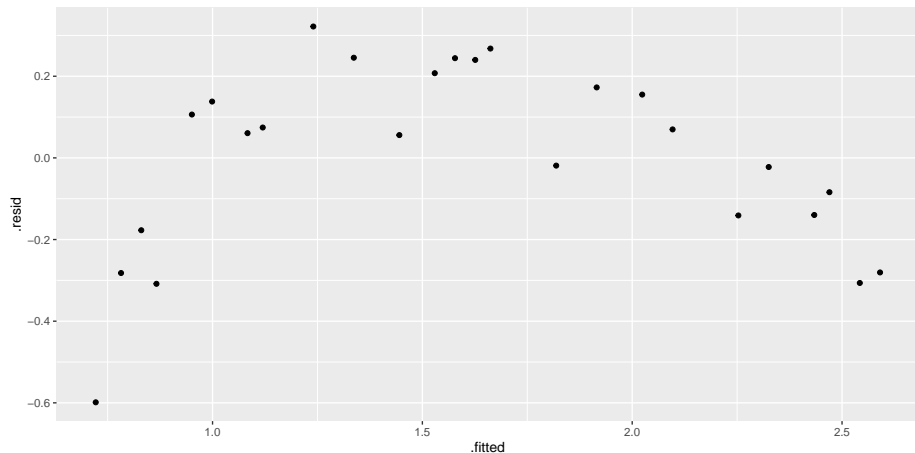
```
ggplot(windmill, aes(y = DC_output, x = wind_velocity)) +  
  geom_point() + geom_smooth(method="lm", se = F)
```

The plot



Plot of residuals against fitted values

```
ggplot(DC.1, aes(y = .resid, x = .fitted)) + geom_point()
```



Comments on residual plot

- Residual plot should be a random scatter of points.
- Should be no pattern “left over” after fitting the regression.
- Smooth trend should be more or less straight across at 0.
- Here, have a curved trend on residual plot.
- This means original relationship must have been a curve (as we saw on original scatterplot).
- Possible ways to fit a curve:
 - Add a squared term in explanatory variable.
 - Transform response variable (doesn't work well here).
 - See what science tells you about mathematical form of relationship, and try to apply.

Parabolas and fitting parabola model

- A parabola has equation

$$y = ax^2 + bx + c$$

with coefficients a, b, c . About the simplest function that is not a straight line.

- Fit one using `lm` by adding x^2 to right side of model formula with `+`:

```
DC.2 <- lm(DC_output ~ wind_velocity + I(wind_velocity^2),
  data = windmill
)
```

```
summary(DC.2)
```

```
##
```

```
## Call:
```

```
## lm(formula = DC_output ~ wind_velocity + I(wind_velocity^2)
##     data = windmill)
```

```
##
```


Parabola model output

```
tidy(DC.2)
```

```
## # A tibble: 3 x 5
```

	term	estimate	std.error	statistic	p.value
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>
## 1	(Intercept)	-1.16	0.175	-6.62	1.18e- 6
## 2	wind_veloci~	0.723	0.0614	11.8	5.77e-11
## 3	I(wind_velo~	-0.0381	0.00480	-7.95	6.59e- 8

```
glance(DC.2)
```

```
## # A tibble: 1 x 12
```

	r.squared	adj.r.squared	sigma	statistic	p.value
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
## 1	0.968	0.965	0.123	328.	4.16e-17

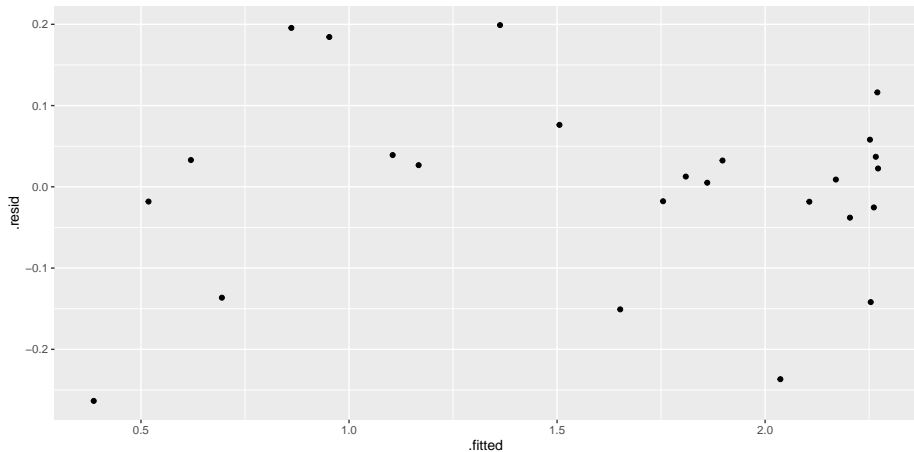
... with 7 more variables: df <dbl>,
 ## # logLik <dbl>, AIC <dbl>, BIC <dbl>,
 ## # deviance <dbl>, df.residual <int>, nobs <int>

Comments on output

- R-squared has gone up a lot, from 87% (line) to 97% (parabola).
- Coefficient of squared term strongly significant (P-value 6.59×10^{-8}).
- Adding squared term has definitely improved fit of model.
- Parabola model better than linear one.
- But...need to check residuals again.

Residual plot from parabola model

```
ggplot(DC.2, aes(y = .resid, x = .fitted)) +  
  geom_point()
```



Make scatterplot with fitted line and curve

- Residual plot basically random. Good.
- Scatterplot with fitted line and curve like this:

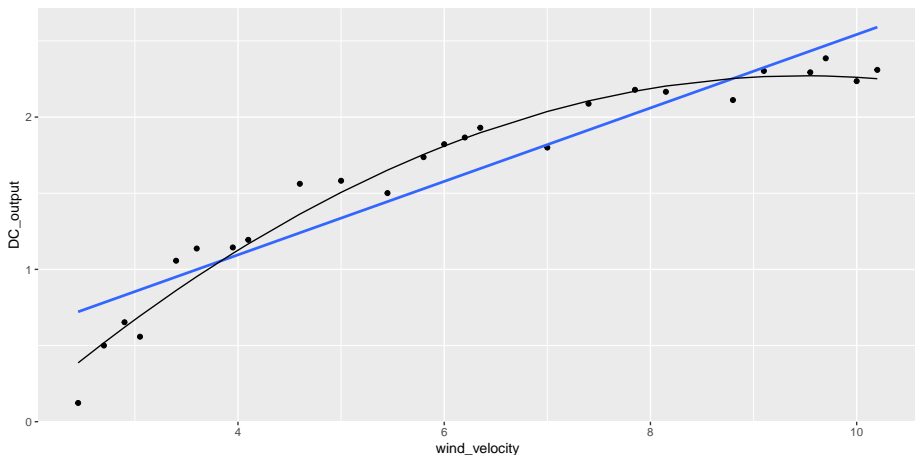
```
ggplot(windmill, aes(y = DC_output, x = wind_velocity)) +  
  geom_point() + geom_smooth(method = "lm", se = F) +  
  geom_line(data = DC.2, aes(y = .fitted))
```

Comments

- This plots:
 - scatterplot (`geom_point`);
 - straight line (via tweak to `geom_smooth`, which draws best-fitting line);
 - fitted curve, using the predicted `DC_output` values, joined by lines (with points not shown).
- Trick in the `geom_line` is use the predictions as the y-points to join by lines (from `DC.2`), instead of the original data points. Without the `data` and `aes` in the `geom_line`, original data points would be joined by lines.

Scatterplot with fitted line and curve

```
## `geom_smooth()` using formula 'y ~ x'
```



Curve clearly fits better than line.

Another approach to a curve

- There is a problem with parabolas, which we'll see later.
- Ask engineer, “what should happen as wind velocity increases?”:
 - Upper limit on electricity generated, but otherwise, the larger the wind velocity, the more electricity generated.
- Mathematically, *asymptote*. Straight lines and parabolas don't have them, but eg. $y = 1/x$ does: as x gets bigger, y approaches zero without reaching it.
- What happens to $y = a + b(1/x)$ as x gets large?
 - y gets closer and closer to a : that is, a is asymptote.
- Fit this, call it asymptote model.
- Fitting the model here because we have math to justify it.
 - Alternative, $y = a + be^{-x}$, approaches asymptote faster.

How to fit asymptote model?

- Define new explanatory variable to be $1/x$, and predict y from it.
- x is velocity, distance over time.
- So $1/x$ is time over distance. In walking world, if you walk 5 km/h, take 12 minutes to walk 1 km, called your pace. So 1 over wind_velocity we call wind_pace.
- Make a scatterplot first to check for straightness (next page).

```
windmill %>% mutate(wind_pace = 1 / wind_velocity) -> windmill
ggplot(windmill, aes(y = DC_output, x = wind_pace)) +
  geom_point() + geom_smooth(se = F)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

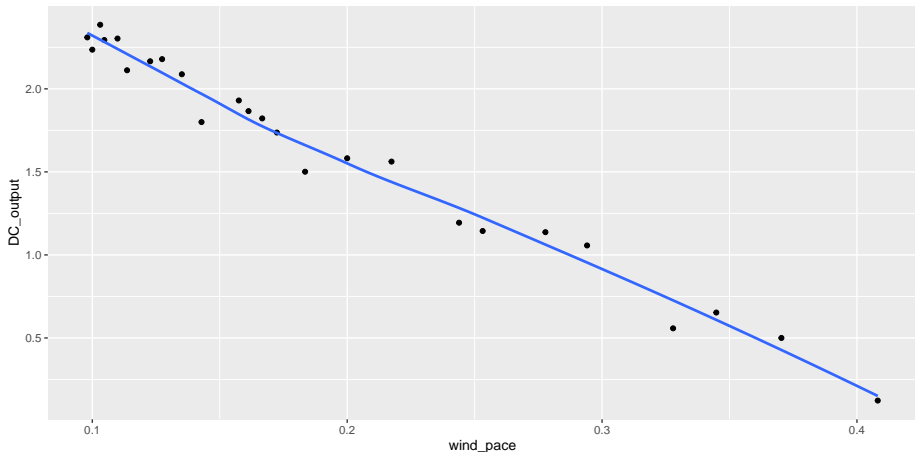
- and run regression like this (output page after):

```
DC.3 <- lm(DC_output ~ wind_pace, data = windmill)
summary(DC.3)
```


Scatterplot for wind_pace

Pretty straight. Blue actually smooth curve not line:

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Regression output

```
glance(DC.3)
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic p.value
##   <dbl>      <dbl> <dbl>      <dbl>    <dbl>
## 1    0.980      0.979 0.0942    1128. 4.74e-21
## # ... with 7 more variables: df <dbl>,
## #   logLik <dbl>, AIC <dbl>, BIC <dbl>,
## #   deviance <dbl>, df.residual <int>, nobs <int>
```

```
tidy(DC.3)
```

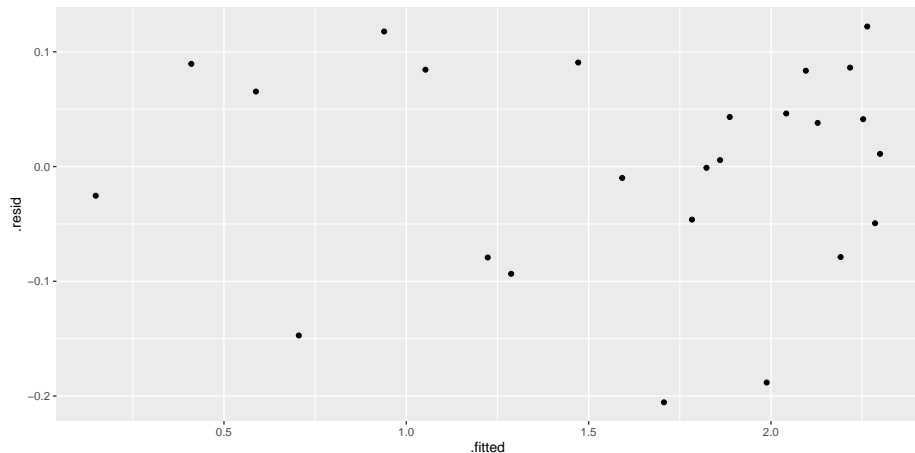
```
## # A tibble: 2 x 5
##   term          estimate std.error statistic p.value
##   <chr>          <dbl>      <dbl>      <dbl>    <dbl>
## 1 (Intercept)    2.98      0.0449      66.3 8.92e-28
## 2 wind_pace     -6.93      0.206     -33.6 4.74e-21
```

Comments

- R-squared, 98%, even higher than for parabola model (97%).
- Simpler model, only one explanatory variable (`wind.pace`) vs. 2 for parabola model (`wind.velocity` and its square).
- `wind.pace` (unsurprisingly) strongly significant.
- Looks good, but check residual plot (over).

Residual plot for asymptote model

```
ggplot(DC.3, aes(y = .resid, x = .fitted)) + geom_point()
```



Plotting trends on scatterplot

- Residual plot not bad. But residuals go up to 0.10 and down to -0.20 , suggesting possible skewness (not normal). I think it's not perfect, but OK overall.
- Next: plot scatterplot with all three fitted lines/curves on it (for comparison), with legend saying which is which.
- First make data frame containing what we need, taken from the right places:

```
w2 <- tibble(  
  wind_velocity = windmill$wind_velocity,  
  DC_output = windmill$DC_output,  
  linear = fitted(DC.1),  
  parabola = fitted(DC.2),  
  asymptote = fitted(DC.3)  
)
```

What's in w2

w2

```
## # A tibble: 25 x 5
##   wind_velocity DC_output linear parabola asymptote
##           <dbl>      <dbl> <dbl>      <dbl>      <dbl>
## 1             5        1.58   1.34        1.51        1.59
## 2             6        1.82   1.58        1.81        1.82
## 3            3.4        1.06   0.951       0.861       0.939
## 4            2.7         0.5   0.782       0.518       0.411
## 5            10        2.24   2.54        2.26        2.29
## 6            9.7        2.39   2.47        2.27        2.26
## 7            9.55       2.29   2.43        2.27        2.25
## 8            3.05       0.558  0.866       0.694       0.705
## 9            8.15       2.17   2.10        2.20        2.13
## 10           6.2        1.87   1.63        1.86        1.86
## # ... with 15 more rows
```

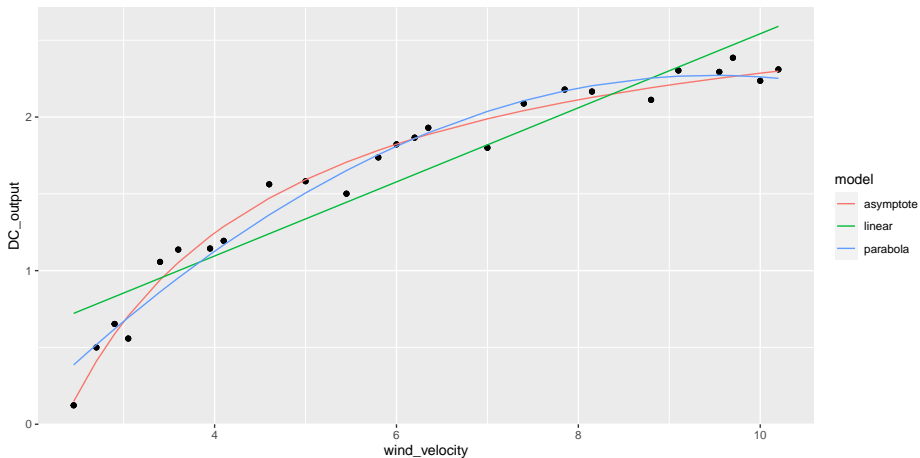
Making the plot

- ggplot likes to have one column of x 's to plot, and one column of y 's, with another column for distinguishing things.
- But we have three columns of fitted values, that need to be combined into one.
- `pivot_longer`, then plot:

w2 %>%

```
pivot_longer(linear:asymptote, names_to="model",  
              values_to="fit") %>%  
ggplot(aes(x = wind_velocity, y = DC_output)) +  
geom_point() +  
geom_line(aes(y = fit, colour = model))
```

Scatterplot with fitted curves



Comments

- Predictions from curves are very similar.
- Predictions from asymptote model as good, and from simpler model (one x not two), so prefer those.
- Go back to asymptote model summary.

Asymptote model summary

```
tidy(DC.3)
```

```
## # A tibble: 2 x 5
```

##	term	estimate	std.error	statistic	p.value
##	<chr>	<dbl>	<dbl>	<dbl>	<dbl>
## 1	(Intercept)	2.98	0.0449	66.3	8.92e-28
## 2	wind_pace	-6.93	0.206	-33.6	4.74e-21

Comments

- Intercept in this model about 3.
- Intercept of asymptote model is the asymptote (upper limit of DC.output).
- Not close to asymptote yet.
- Therefore, from this model, wind could get stronger and would generate appreciably more electricity.
- This is extrapolation! Would like more data from times when wind.velocity higher.
- Slope -7 . Why negative?
 - As wind.velocity increases,
 - wind.pace goes down,
 - and DC.output goes up. Check.
- Actual slope number hard to interpret.

Checking back in with research questions

- Is there a relationship between wind speed and current generated?
 - Yes.
- If so, what kind of relationship is it?
 - One with an asymptote.
- Can we model the relationship, in such a way that we can do predictions?
 - Yes, see model DC.3 and plot of fitted curve.
- Good. Job done.

Job done, kinda

- Just because the parabola model and asymptote model agree over the range of the data, doesn't necessarily mean they agree everywhere.
- Extend range of wind.velocity to 1 to 16 (steps of 0.5), and predict DC.output according to the two models:

```
wv <- seq(1, 16, 0.5)
wv
```

```
## [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
## [10] 5.5 6.0 6.5 7.0 7.5 8.0 8.5 9.0 9.5
## [19] 10.0 10.5 11.0 11.5 12.0 12.5 13.0 13.5 14.0
## [28] 14.5 15.0 15.5 16.0
```

- R has `predict`, which requires what to predict for, as data frame. The data frame has to contain values, with matching names, for all explanatory variables in `regression(s)`.

Setting up data frame to predict from

- Linear model had just `wind_velocity`.
- Parabola model had that as well (squared one will be calculated)
- Asymptote model had just `wind_pace` (reciprocal of velocity).
- So create data frame called `wv_new` with those in:

```
wv_new <- tibble(wind_velocity = wv, wind_pace = 1 / wv)
```

wv_new

```
wv_new
```

```
## # A tibble: 31 x 2
##   wind_velocity wind_pace
##         <dbl>     <dbl>
## 1             1         1
## 2             1.5     0.667
## 3             2         0.5
## 4             2.5     0.4
## 5             3         0.333
## 6             3.5     0.286
## 7             4         0.25
## 8             4.5     0.222
## 9             5         0.2
## 10            5.5     0.182
## # ... with 21 more rows
```

Doing predictions, one for each model

- Use same names as before:

```
linear <- predict(DC.1, wv_new)
parabola <- predict(DC.2, wv_new)
asymptote <- predict(DC.3, wv_new)
```

- Put it all into a data frame for plotting, along with original data:

```
my_fits <- tibble(
  wind_velocity = wv_new$wind_velocity,
  linear, parabola, asymptote
)
```


my_fits

my_fits

```
## # A tibble: 31 x 4
##   wind_velocity linear parabola asymptote
##         <dbl>   <dbl>     <dbl>     <dbl>
## 1           1    0.372   -0.471    -3.96
## 2          1.5    0.493   -0.157    -1.64
## 3           2    0.613    0.137    -0.488
## 4          2.5    0.734    0.413     0.205
## 5           3    0.854    0.670     0.667
## 6          3.5    0.975    0.907     0.998
## 7           4    1.10     1.13     1.25
## 8          4.5    1.22     1.33     1.44
## 9           5    1.34     1.51     1.59
## 10          5.5    1.46     1.67     1.72
## # ... with 21 more rows
```

Making a plot 1/2

- To make a plot, we use the same trick as last time to get all three predictions on a plot with a legend (saving result to add to later):

```
my_fits %>%  
  pivot_longer(  
    linear:asymptote,  
    names_to="model",  
    values_to="fit"  
  ) %>%  
  ggplot(aes(  
    y = fit, x = wind_velocity,  
    colour = model  
  )) + geom_line() -> g
```

Making a plot 2/2

- The observed wind velocities were in this range:

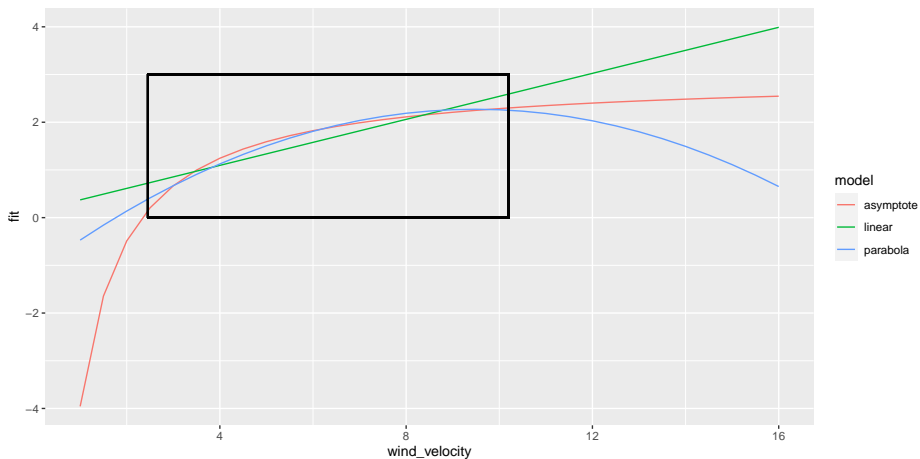
```
(vels <- range(windmill$wind_velocity))
```

```
## [1] 2.45 10.20
```

- DC.output between 0 and 3 from asymptote model. Add rectangle to graph around where the data were:

```
g + geom_rect(  
  xmin = vels[1], xmax = vels[2], ymin = 0, ymax = 3,  
  alpha=0, colour = "black"  
)
```

The plot



Comments (1)

- Over range of data, two models agree with each other well.
- Outside range of data, they disagree violently!
- For larger `wind.velocity`, asymptote model behaves reasonably, parabola model does not.
- What happens as `wind.velocity` goes to zero? Should find `DC.output` goes to zero as well. Does it?

Comments (2)

- For parabola model:

```
tidy(DC.2)
```

```
## # A tibble: 3 x 5
```

##	term	estimate	std.error	statistic	p.value
##	<chr>	<dbl>	<dbl>	<dbl>	<dbl>
## 1	(Intercept)	-1.16	0.175	-6.62	1.18e- 6
## 2	wind_veloci~	0.723	0.0614	11.8	5.77e-11
## 3	I(wind_velo~	-0.0381	0.00480	-7.95	6.59e- 8

- Nope, goes to -1.16 (intercept), actually significantly different from zero.

Comments (3): asymptote model

```
tidy(DC.3)
```

```
## # A tibble: 2 x 5
##   term          estimate std.error statistic  p.value
##   <chr>          <dbl>    <dbl>    <dbl>    <dbl>
## 1 (Intercept)    2.98    0.0449    66.3 8.92e-28
## 2 wind_pace     -6.93    0.206   -33.6 4.74e-21
```

- As `wind.velocity` heads to 0, `wind.pace` heads to $+\infty$, so `DC.output` heads to $-\infty$!
- Also need more data for small `wind.velocity` to understand relationship. (Is there a lower asymptote?)
- Best we can do now is to predict `DC.output` to be zero for small `wind.velocity`.
- Assumes a “threshold” wind velocity below which no electricity generated at all.

Summary

- Often, in data analysis, there is no completely satisfactory conclusion, as here.
- Have to settle for model that works OK, with restrictions.
- Always something else you can try.
- At some point you have to say “I stop.”

C33 only

Section 8

The bootstrap in more detail

Packages for this section

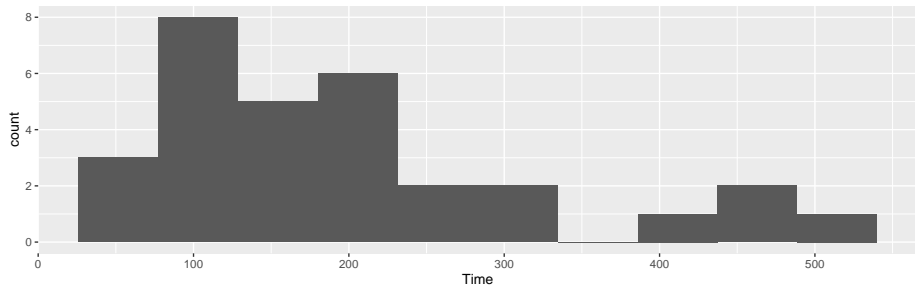
```
library(tidyverse)
library(bootstrap)
```

[My source for this](#)

Is my sampling distribution normal enough?

- Recall the IRS data that we used as a motivation for the sign test:

```
ggplot(irs, aes(x=Time))+geom_histogram(bins=10)
```



- We said that a t procedure for the mean would not be a good idea because the distribution is skewed.

What *actually* matters

- It's not the distribution of the *data* that has to be approx normal (for a *t* procedure).
- What matters is the *sampling distribution of the sample mean*.
- If the sample size is large enough, the sampling distribution will be normal enough even if the data distribution is not.
 - This is why we had to consider the sample size as well as the shape.
- But how do we know whether this is the case or not? We only have *one* sample.

The (nonparametric) bootstrap

- Typically, our sample will be reasonably representative of the population.
- Idea: pretend the sample *is* the population, and sample from it *with replacement*.
- Calculate test statistic, and repeat many times.
- This gives an idea of how our statistic might vary in repeated samples: that is, its sampling distribution.
- Called the **bootstrap distribution** of the test statistic.
- If the bootstrap distribution is approx normal, infer that the true sampling distribution also approx normal, therefore inference about the mean such as t is good enough.
- If not, we should be more careful.

Why it works

- We typically estimate population parameters by using the corresponding sample thing: eg. estimate population mean using sample mean.
- This called **plug-in principle**.
- The fraction of sample values less than a value x called the **empirical distribution function** (as a function of x).
- By plug-in principle, the empirical distribution function is an estimate of the population CDF.
- In this sense, the sample *is* an estimate of the population, and so sampling from it is an estimate of sampling from the population.

Bootstrapping the IRS data

- Sampling with replacement is done like this (the default sample size is as long as the original data):

```
boot <- sample(irs$Time, replace=T)  
mean(boot)
```

```
## [1] 248.0333
```

- That's one bootstrapped mean. We need a whole bunch.

A whole bunch

- Use the same idea as for simulating power:

```
tibble(sim = 1:1000) %>%
  rowwise() %>%
  mutate(boot_sample = list(sample(irs$Time, replace = TRUE)))
```

```
## # A tibble: 1,000 x 2
##       sim boot_sample
##   <int> <list>
## 1       1 <dbl [30]>
## 2       2 <dbl [30]>
## 3       3 <dbl [30]>
## 4       4 <dbl [30]>
## 5       5 <dbl [30]>
## 6       6 <dbl [30]>
## 7       7 <dbl [30]>
## 8       8 <dbl [30]>
## 9       9 <dbl [30]>
## 10      10 <dbl [30]>
```

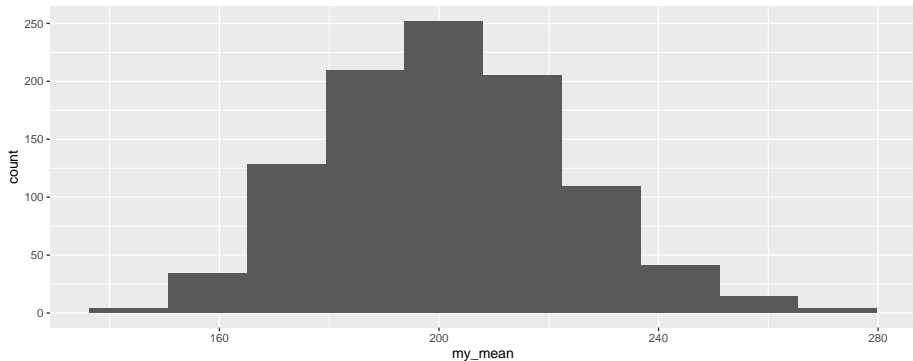

Get the mean of each of those

```
tibble(sim = 1:1000) %>%
  rowwise() %>%
  mutate(boot_sample = list(sample(irs$Time, replace = TRUE))) %>%
  mutate(my_mean = mean(boot_sample)) -> samples
samples
```

```
## # A tibble: 1,000 x 3
##       sim boot_sample my_mean
##   <int> <list>         <dbl>
## 1     1 1 <dbl [30]>         196
## 2     2 2 <dbl [30]>         202.
## 3     3 3 <dbl [30]>         263.
## 4     4 4 <dbl [30]>         173.
## 5     5 5 <dbl [30]>         204.
## 6     6 6 <dbl [30]>         197.
## 7     7 7 <dbl [30]>         210.
## 8     8 8 <dbl [30]>         160.
## 9     9 9 <dbl [30]>         198.
## 10    10 10 <dbl [30]>         178
```

Sampling distribution of sample mean

```
ggplot(samples, aes(x=my_mean)) + geom_histogram(bins=10)
```

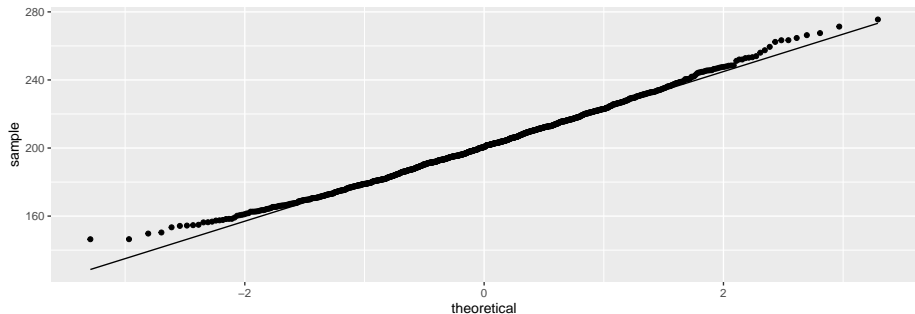


- Is that a slightly long right tail?

Normal quantile plot

might be better than a histogram:

```
ggplot(samples, aes(sample = my_mean)) +  
  stat_qq()+stat_qq_line()
```



- a very very slight right-skewness, but very close to normal.

Confidence interval from the bootstrap distribution

There are two ways (at least):

- percentile bootstrap interval: take the 2.5 and 97.5 percentiles (to get the middle 95%). This is easy, but not always the best:

```
(b_p=quantile(samples$my_mean, c(0.025, 0.975)))
```

```
##      2.5%      97.5%
## 162.5775 246.9092
```

- bootstrap t : use the SD of the bootstrapped sampling distribution as the SE of the estimator of the mean and make a t interval:

```
n <- length(irs$Time)
t_star <- qt(0.975, n-1)
b_t <- with(samples, mean(my_mean)+c(-1, 1)*t_star*sd(my_mean))
b_t
```

```
## [1] 156.5070 246.4032
```

Comparing

- get ordinary t interval:

```
my_names=c("LCL", "UCL")
o_t <- t.test(irs$Time)$conf.int
```

- Compare the 2 bootstrap intervals with the ordinary t -interval:

```
tibble(limit=my_names, o_t, b_t, b_p)
```

```
## # A tibble: 2 x 4
##   limit    o_t    b_t    b_p
##   <chr> <dbl> <dbl> <dbl>
## 1 LCL    155.  157.  163.
## 2 UCL    247.  246.  247.
```

- The bootstrap t and the ordinary t are very close
- The percentile bootstrap interval is noticeably shorter (common) and higher (skewness).

Which to prefer?

- If the intervals agree, then they are all good.
- If they disagree, they are all bad!
- In that case, use BCA interval (over).

Bias correction and acceleration

- this from “An introduction to the bootstrap”, by Brad Efron and Robert J. Tibshirani.
- there is way of correcting the CI for skewness in the bootstrap distribution, called the BCa method
- complicated (see the Efron and Tibshirani book), but implemented in bootstrap package.

Run this on the IRS data:

```
bca=bcanon(irs$Time, 1000, mean)
bca$confpoints
```

```
##      alpha bca point
## [1,] 0.025  161.8333
## [2,] 0.050  168.0667
## [3,] 0.100  174.8333
## [4,] 0.160  180.7667
## [5,] 0.840  224.1333
## [6,] 0.900  232.3000
## [7,] 0.950  241.9333
## [8,] 0.975  253.7333
```


use 2.5% and 97.5% points for CI

```
bca$confpoints %>% as_tibble() %>%  
  filter(alpha %in% c(0.025, 0.975)) %>%  
  pull(`bca point`) -> b_bca  
b_bca
```

```
## [1] 161.8333 253.7333
```

Comparing

```
tibble(limit=my_names, o_t, b_t, b_p, b_bca)
```

```
## # A tibble: 2 x 5
##   limit    o_t    b_t    b_p b_bca
##   <chr> <dbl> <dbl> <dbl> <dbl>
## 1 LCL    155.  157.  163.  162.
## 2 UCL    247.  246.  247.  254.
```

- The BCA interval says that the mean should be estimated even higher than the bootstrap percentile interval does.
- The BCA interval is the one to trust.

Bootstrapping the correlation

Recall the soap data:

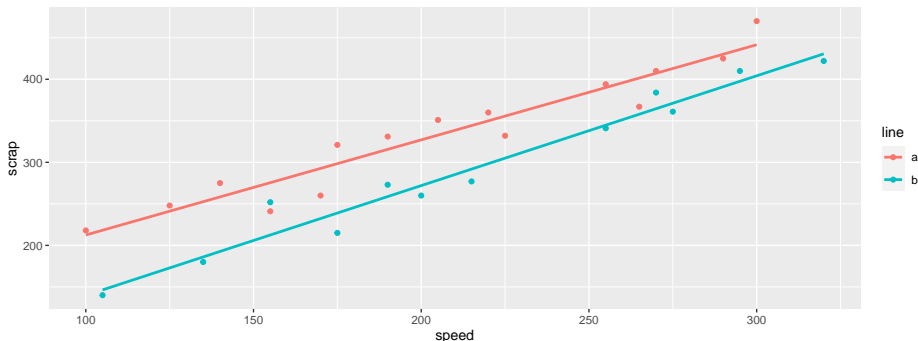
```
url <- "http://ritsokiguess.site/datafiles/soap.txt"
soap <- read_delim(url, " ")
soap
```

```
## # A tibble: 27 x 4
##       case scrap speed line
##   <dbl> <dbl> <dbl> <chr>
## 1       1    218   100 a
## 2       2    248   125 a
## 3       3    360   220 a
## 4       4    351   205 a
## 5       5    470   300 a
## 6       6    394   255 a
## 7       7    332   225 a
## 8       8    321   175 a
```

Scatterplot

```
ggplot(soap, aes(x=speed, y=scrap, colour=line))+  
  geom_point()+geom_smooth(method="lm", se=F)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Comments

- Line B produces less scrap for any given speed.
- For line B, estimate the correlation between speed and scrap (with a confidence interval.)

Extract the line B data; standard correlation test

```
soap %>% filter(line=="b") -> line_b
with(line_b, cor.test(speed, scrap))

##
## Pearson's product-moment correlation
##
## data: speed and scrap
## t = 15.829, df = 10, p-value = 2.083e-08
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
## 0.9302445 0.9947166
## sample estimates:
## cor
## 0.9806224
```

Bootstrapping a correlation 1/2

- This illustrates a different technique: we need to keep the x and y values *together*.
- Sample *rows* of the data frame rather than individual values of speed and scrap:

```
line_b %>% sample_frac(replace=T)
```

```
## # A tibble: 12 x 4
##   case scrap speed line
##   <dbl> <dbl> <dbl> <chr>
## 1     19   341   255 b
## 2     19   341   255 b
## 3     19   341   255 b
## 4     17   277   215 b
## 5     16   140   105 b
## 6     20   215   175 b
## 7     18   384   270 b
## 8     19   341   255 b
## 9     21   180   135 b
## 10    26   273   190 b
## 11    17   277   215 b
## 12    27   410   295 b
```

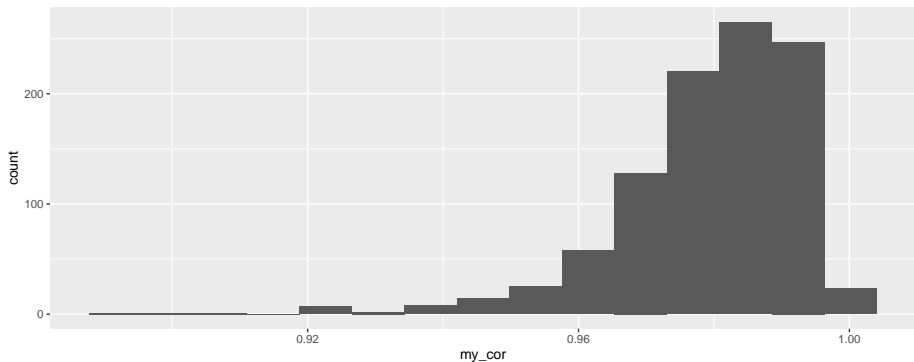
Bootstrapping a correlation 2/2

1000 times:

```
tibble(sim = 1:1000) %>%  
  rowwise() %>%  
  mutate(boot_df = list(sample_frac(line_b, replace = TRUE)))  
  mutate(my_cor = with(boot_df, cor(speed, scrap))) -> cors
```


A picture of this

```
ggplot(cors, aes(x=my_cor))+geom_histogram(bins=15)
```



Comments and next steps

- This is very left-skewed.
- Bootstrap percentile interval is:

```
(b_p=quantile(cors$my_cor, c(0.025, 0.975)))
```

```
##          2.5%        97.5%  
## 0.9450798 0.9962227
```

- We probably need the BCA interval instead.

Getting the BCA interval 1/2

- To use `bcanon`, write a function that takes a vector of row numbers and returns the correlation between speed and scrap for those rows:

```
theta=function(rows, d) {
  d %>% slice(rows) %>% with(., cor(speed, scrap))
}
theta(1:3, line_b)
```

```
## [1] 0.9928971
```

```
line_b %>% slice(1:3)
```

```
## # A tibble: 3 x 4
```

```
##   case scrap speed line
```

```
##   <dbl> <dbl> <dbl> <chr>
```

```
## 1    16   140   105 b
```

```
## 2    17   277   215 b
```

```
## 3    18   384   270 b
```

- That looks about right.

Getting the BCA interval 2/2

- Inputs to `bcanon` are now:
 - row numbers (1 through 12 in our case: 12 rows in `line_b`)
 - number of bootstrap samples
 - the function we just wrote
 - the data frame:

```
points=bcanon(1:12, 1000, theta, line_b)$confpoints
points %>% as_tibble() %>%
  filter(alpha %in% c(0.025, 0.975)) %>%
  pull(`bca point`) -> b_bca
b_bca
```

```
## [1] 0.9314334 0.9947634
```

Comparing the results

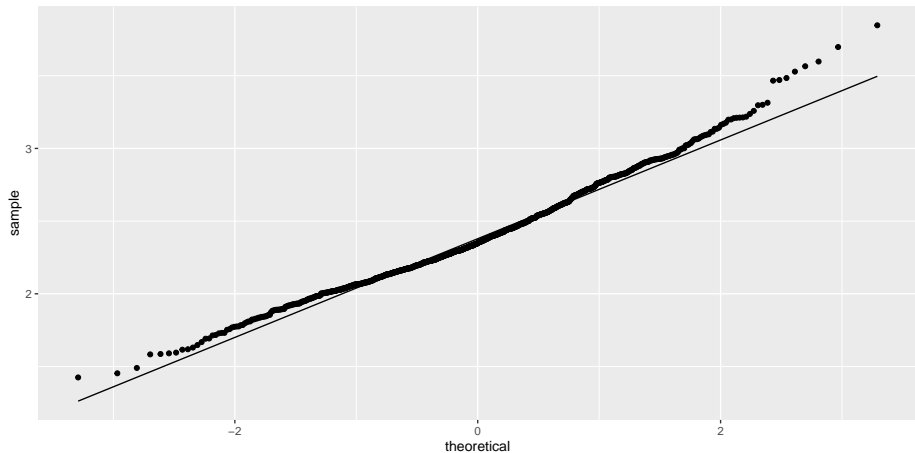
```
tibble(limit=my_names, o_c, b_p, b_bca)
```

```
## # A tibble: 2 x 4
##   limit    o_c    b_p b_bca
##   <chr> <dbl> <dbl> <dbl>
## 1 LCL    0.930 0.945 0.931
## 2 UCL    0.995 0.996 0.995
```

- The bootstrap percentile interval doesn't go down far enough.
- The BCA interval seems to do a better job in capturing the skewness of the distribution.
- The ordinary confidence interval for the correlation is very similar to the BCA one, and thus seems to be trustworthy here even though the correlation has a very skewed distribution. (`cor.test` uses the Fisher z transformation which “spreads out” correlations close to 1).

The z -transformed bootstrapped correlations

```
cors %>%  
  mutate(z = 0.5 * log((1+my_cor)/(1-my_cor))) %>%  
  ggplot(aes(sample=z)) + stat_qq() + stat_qq_line()
```



Section 9

ANOVA revisited

Analysis of variance

- Analysis of variance used with:
 - counted/measured response
 - categorical explanatory variable(s)
 - that is, data divided into groups, and see if response significantly different among groups
 - or, see whether knowing group membership helps to predict response.
- Typically two stages:
 - F -test to detect *any* differences among/due to groups
 - if F -test significant, do *multiple comparisons* to see which groups significantly different from which.
- Need special multiple comparisons method because just doing (say) two-sample t -tests on each pair of groups gives too big a chance of finding “significant” differences by accident.

Packages

These:

```
library(tidyverse)
library(broom)
library(car) # for Levene's test
```

Example: Pain threshold and hair colour

- Do people with different hair colour have different abilities to deal with pain?
- Men and women of various ages divided into 4 groups by hair colour: light and dark blond, light and dark brown.
- Each subject given a pain sensitivity test resulting in pain threshold score: higher score is higher pain tolerance.
- 19 subjects altogether.

The data

In hairpain.txt:

hair	pain	
lightblond	62	darkblond 43
lightblond	60	lightbrown 42
lightblond	71	lightbrown 50
lightblond	55	lightbrown 41
lightblond	48	lightbrown 37
darkblond	63	darkbrown 32
darkblond	57	darkbrown 39
darkblond	52	darkbrown 51
darkblond	41	darkbrown 30
		darkbrown 35

Summarizing the groups

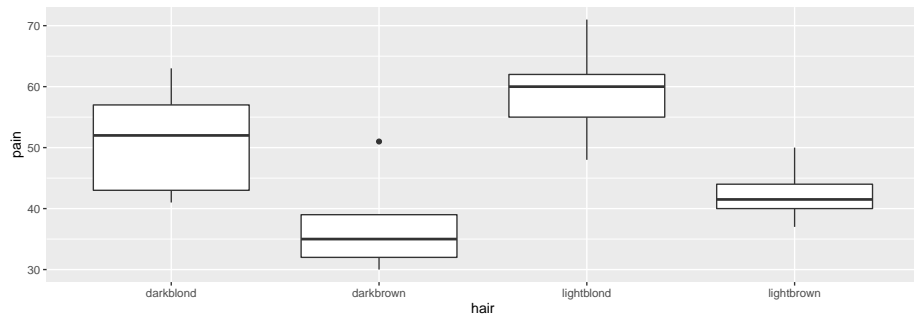
```
my_url <- "http://ritsokiguess.site/datafiles/hairpain.txt"
hairpain <- read_delim(my_url, " ")
hairpain %>%
  group_by(hair) %>%
  summarize(
    n = n(),
    xbar = mean(pain),
    s = sd(pain)
  )
```

```
## # A tibble: 4 x 4
##   hair      n  xbar    s
##   <chr>  <int> <dbl> <dbl>
## 1 darkblond    5  51.2  9.28
## 2 darkbrown    5  37.4  8.32
## 3 lightblond   5  59.2  8.53
## 4 lightbrown   4  42.5  5.45
```

Brown-haired people seem to have lower pain tolerance.

Boxplot

```
ggplot(hairpain, aes(x = hair, y = pain)) + geom_boxplot()
```



Assumptions

- Data should be:
 - normally distributed within each group
 - same spread for each group
- darkbrown group has upper outlier (suggests not normal)
- darkblond group has smaller IQR than other groups.
- But, groups *small*.
- Shrug shoulders and continue for moment.

Testing equality of SDs

- via **Levene's test** in package `car`:

```
leveneTest(pain ~ hair, data = hairpain)
```

```
## Warning in leveneTest.default(y = y, group =
## group, ...): group coerced to factor.
```

```
## # A tibble: 2 x 3
##       Df `F value` `Pr(>F)`
##   <int>    <dbl>    <dbl>
## 1      3    0.393    0.760
## 2     15     NA      NA
```

- No evidence (at all) of difference among group SDs.
- Possibly because groups *small*.

Analysis of variance

```
hairpain.1 <- aov(pain ~ hair, data = hairpain)
summary(hairpain.1)
```

```
##              Df Sum Sq Mean Sq F value    Pr(>F)
## hair           3   1361    453.6     6.791 0.00411 **
## Residuals     15   1002     66.8
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

- P-value small: the mean pain tolerances for the four groups are *not* all the same.
- Which groups differ from which, and how?

Multiple comparisons

- Which groups differ from which? Multiple comparisons method. Lots.
- Problem: by comparing all the groups with each other, doing many tests, have large chance to (possibly incorrectly) reject H_0 : groups have equal means.
- 4 groups: 6 comparisons (1 vs 2, 1 vs 3, ..., 3 vs 4). 5 groups: 10 comparisons. Thus 6 (or 10) chances to make mistake.
- Get “familywise error rate” of 0.05 (whatever), no matter how many comparisons you’re doing.
- My favourite: Tukey, or “honestly significant differences’’: how far apart might largest, smallest group means be (if actually no differences). Group means more different: significantly different.

Tukey

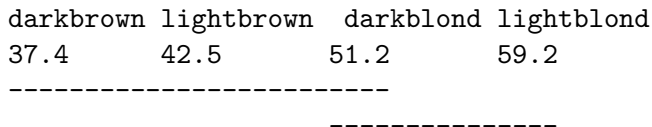
- TukeyHSD:

```
TukeyHSD(hairpain.1)
```

```
##      Tukey multiple comparisons of means
##      95% family-wise confidence level
##
## Fit: aov(formula = pain ~ hair, data = hairpain)
##
## $hair
##              diff            lwr            upr
## darkbrown-darkblond  -13.8 -28.696741  1.0967407
## lightblond-darkblond   8.0  -6.896741 22.8967407
## lightbrown-darkblond  -8.7 -24.500380  7.1003795
## lightblond-darkbrown  21.8   6.903259 36.6967407
## lightbrown-darkbrown   5.1 -10.700380 20.9003795
## lightbrown-lightblond -16.7 -32.500380 -0.8996205
##              p adj
## darkbrown-darkblond  0.0740679
## lightblond-darkblond 0.4355768
## lightbrown-darkblond 0.4447000
```

The old-fashioned way

- List group means in order
- Draw lines connecting groups that are *not* significantly different:



- lightblond significantly higher than everything except darkblond (at $\alpha = 0.05$).
- darkblond in middle ground: not significantly less than lightblond, not significantly greater than darkbrown and lightbrown.
- More data might resolve this.
- Looks as if blond-haired people do have higher pain tolerance, but not completely clear.

Some other multiple-comparison methods

- Work any time you do k tests at once (not just ANOVA).
 - **Bonferroni**: multiply all P-values by k .
 - **Holm**: multiply smallest P-value by k , next-smallest by $k - 1$, etc.
 - **False discovery rate**: multiply smallest P-value by $k/1$, 2nd-smallest by $k/2$, ..., i -th smallest by k/i .
- Stop after non-rejection.

Example

- P-values 0.005, 0.015, 0.03, 0.06 (4 tests all done at once) Use $\alpha = 0.05$.
- Bonferroni:
 - Multiply all P-values by 4 (4 tests).
 - Reject only 1st null.
- Holm:
 - Times smallest P-value by 4: $0.005 * 4 = 0.020 < 0.05$, reject.
 - Times next smallest by 3: $0.015 * 3 = 0.045 < 0.05$, reject.
 - Times next smallest by 2: $0.03 * 2 = 0.06 > 0.05$, do not reject. Stop.

...Continued

- With P-values 0.005, 0.015, 0.03, 0.06:
- False discovery rate:
 - Times smallest P-value by 4: $0.005 * 4 = 0.02 < 0.05$: reject.
 - Times second smallest by $4/2$: $0.015 * 4/2 = 0.03 < 0.05$, reject.
 - Times third smallest by $4/3$: $0.03 * 4/3 = 0.04 < 0.05$, reject.
 - Times fourth smallest by $4/4$: $0.06 * 4/4 = 0.06 > 0.05$, do not reject.
Stop.

pairwise.t.test

```
with(hairpain, pairwise.t.test(pain, hair, p.adj = "none"))
```

```
##
## Pairwise comparisons using t tests with pooled SD
##
## data:  pain and hair
##
##           darkblond darkbrown lightblond
## darkbrown  0.01748    -              -
## lightblond 0.14251    0.00075    -
## lightbrown 0.13337    0.36695    0.00817
##
## P value adjustment method: none
```

```
with(hairpain, pairwise.t.test(pain, hair, p.adj = "holm"))
```

```
##
## Pairwise comparisons using t tests with pooled SD
##
## data:  pain and hair
##
##           darkblond darkbrown lightblond
## darkbrown  0.0699    -              -
## lightblond 0.4001    0.0045    -
## lightbrown 0.4001    0.4001    0.0408
##
## P value adjustment method: holm
```

pairwise.t.test part 2

```
with(hairpain, pairwise.t.test(pain, hair, p.adj = "fdr"))
```

```
##
## Pairwise comparisons using t tests with pooled SD
##
## data:  pain and hair
##
##           darkblond darkbrown lightblond
## darkbrown  0.0350    -            -
## lightblond 0.1710    0.0045    -
## lightbrown 0.1710    0.3670    0.0245
##
## P value adjustment method: fdr
```

```
with(hairpain, pairwise.t.test(pain, hair, p.adj = "bon"))
```

```
##
## Pairwise comparisons using t tests with pooled SD
##
## data:  pain and hair
##
##           darkblond darkbrown lightblond
## darkbrown  0.1049    -            -
## lightblond 0.8550    0.0045    -
## lightbrown 0.8002    1.0000    0.0490
##
## P value adjustment method: bonferroni
```


Comments

- P-values all adjusted upwards from “none”.
- Required because 6 tests at once.
- Highest P-values for Bonferroni: most “conservative”.
- Prefer Tukey or FDR or Holm.
- Tukey only applies to ANOVA, not to other cases of multiple testing.

Rats and vitamin B

- What is the effect of dietary vitamin B on the kidney?
- A number of rats were randomized to receive either a B-supplemented diet or a regular diet.
- Desired to control for initial size of rats, so classified into size classes lean and obese.
- After 20 weeks, rats' kidneys weighed.
- Variables:
 - Response: kidneyweight (grams).
 - Explanatory: diet, ratsize.
- Read in data:

```
my_url <- "http://ritsokiguess.site/datafiles/vitaminb.txt"  
vitaminb <- read_delim(my_url, " ")
```

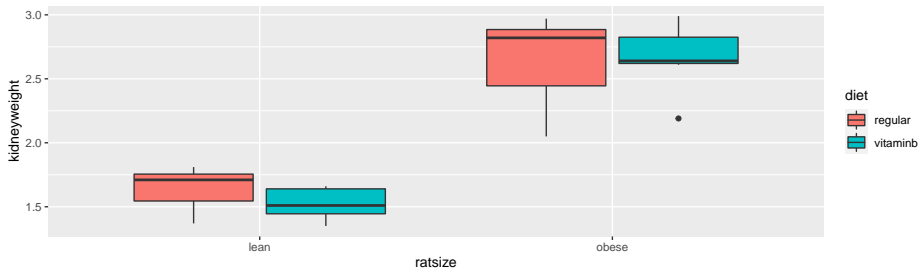
The data

```
vitaminb
```

```
## # A tibble: 28 x 3
##   ratsize diet      kidneyweight
##   <chr>   <chr>         <dbl>
## 1 lean    regular          1.62
## 2 lean    regular          1.8
## 3 lean    regular          1.71
## 4 lean    regular          1.81
## 5 lean    regular          1.47
## 6 lean    regular          1.37
## 7 lean    regular          1.71
## 8 lean    vitaminb         1.51
## 9 lean    vitaminb         1.65
## 10 lean   vitaminb         1.45
## # ... with 18 more rows
```

Grouped boxplot

```
ggplot(vitaminb, aes(  
  x = ratsize, y = kidneyweight,  
  fill = diet  
)) + geom_boxplot()
```



What's going on?

- Calculate group means:

```
summary <- vitaminb %>%
  group_by(ratsize, diet) %>%
  summarize(mean = mean(kidneyweight))
```

```
## `summarise()` has grouped output by 'ratsize'. You can override using the
summary
```

```
## # A tibble: 4 x 3
##   ratsize diet      mean
##   <chr>   <chr>   <dbl>
## 1 lean    regular    1.64
## 2 lean    vitaminb   1.53
## 3 obese   regular    2.64
## 4 obese   vitaminb   2.67
```

- Rat size: a large and consistent effect.
- Diet: small/no effect (compare same rat size, different diet).

ANOVA with interaction

```
vitaminb.1 <- aov(kidneyweight ~ ratsize * diet,
  data = vitaminb
)
summary(vitaminb.1)
```

```
##              Df Sum Sq Mean Sq F value    Pr(>F)
## ratsize        1  8.068    8.068 141.179 1.53e-11 ***
## diet           1   0.012    0.012   0.218   0.645
## ratsize:diet    1   0.036    0.036   0.638   0.432
## Residuals      24   1.372    0.057
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

- Significance/nonsignificance as we expected.
- Note no significant interaction (can be removed).

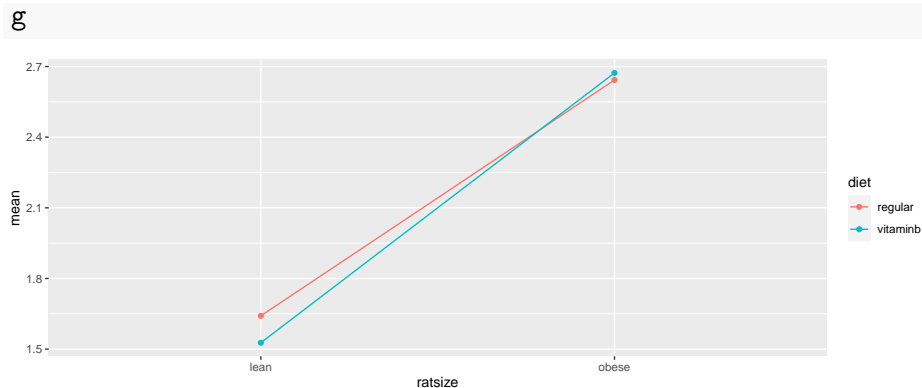
Interaction plot

- Plot mean of response variable against one of the explanatory, using other one as groups. Start from summary:

```
g <- ggplot(summary, aes(  
  x = ratsize, y = mean,  
  colour = diet, group = diet  
)) +  
  geom_point() + geom_line()
```

- For this, have to give *both* group and colour.

The interaction plot



Lines basically parallel, indicating no interaction.

Take out interaction

```
vitaminb.2 <- update(vitaminb.1, . ~ . - ratsize:diet)
summary(vitaminb.2)
```

```
##              Df Sum Sq Mean Sq F value    Pr(>F)
## ratsize         1  8.068    8.068 143.256 7.59e-12 ***
## diet            1  0.012    0.012   0.221   0.643
## Residuals      25  1.408    0.056
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

- No Tukey for diet: not significant.
- No Tukey for ratsize: only two sizes, and already know that obese rats have larger kidneys than lean ones.
- Bottom line: diet has no effect on kidney size once you control for size of rat.

Assessing assumptions: residuals

- In two-way ANOVA, not many observations per treatment group.
- Difficult to check for normality / equal spreads.
- *But*, any regular ANOVA also a regression.
- Use regression residual ideas.
- In ANOVA, one fitted value per treatment group (based on means).
- Residual: observation minus fitted value.

Previous ANOVA as regression

```
vitaminb.3 <- lm(kidneyweight ~ ratsize + diet, data = vitaminb)
summary(vitaminb.3)
```

```
##
## Call:
## lm(formula = kidneyweight ~ ratsize + diet, data = vitaminb)
##
## Residuals:
```

	Min	1Q	Median	3Q	Max
	-0.62893	-0.12625	0.04071	0.14607	0.35321

```
##
## Coefficients:
```

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	1.60536	0.07768	20.67	< 2e-16
ratsizeobese	1.07357	0.08970	11.97	7.59e-12
dietvitaminb	-0.04214	0.08970	-0.47	0.643

```
##
## (Intercept) ***
## ratsizeobese ***
## dietvitaminb
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

Reproduce ANOVA

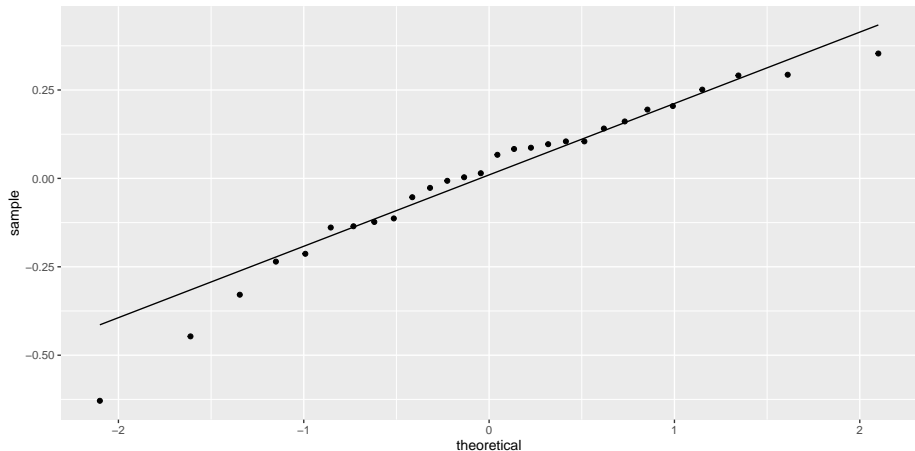
```
tidy(drop1(vitaminb.3, test = "F"))
```

```
## # A tibble: 3 x 7
##   term      df  sumsq    rss    AIC statistic
##   <chr>   <dbl>  <dbl> <dbl> <dbl>      <dbl>
## 1 <none>    NA  NA      1.41 -77.7      NA
## 2 ratsize     1  8.07    9.48 -26.3    143.
## 3 diet        1  0.0124  1.42 -79.5     0.221
## # ... with 1 more variable: p.value <dbl>
```

- ANOVA and regression drop1 output always the same.
- this time, ANOVA and regression summary output have same P-values, but only because categorical variables both have two levels.

Are the residuals normal?

```
ggplot(vitaminb.3, aes(sample=.resid)) +  
  stat_qq() + stat_qq_line()
```



Residuals against fitted

```
ggplot(vitaminb.3, aes(x=.fitted, y=.resid)) + geom_point()
```

Comments

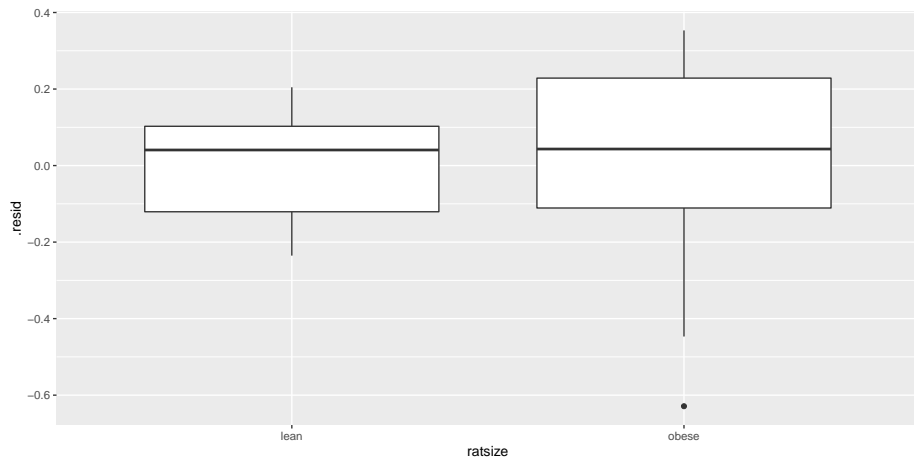
- 2 rat sizes, 2 diets: only $2 \times 2 = 4$ different fitted values
- larger fitted values have greater spread (fan-out, transformation?)
- add residuals to data to plot residuals against size, diet (augment from broom):

```
vitaminb.3 %>% augment(vitaminb) -> vitaminb.3a
```

- explanatory ratsize, diet categorical, so plot resid vs. them with *boxplots*.

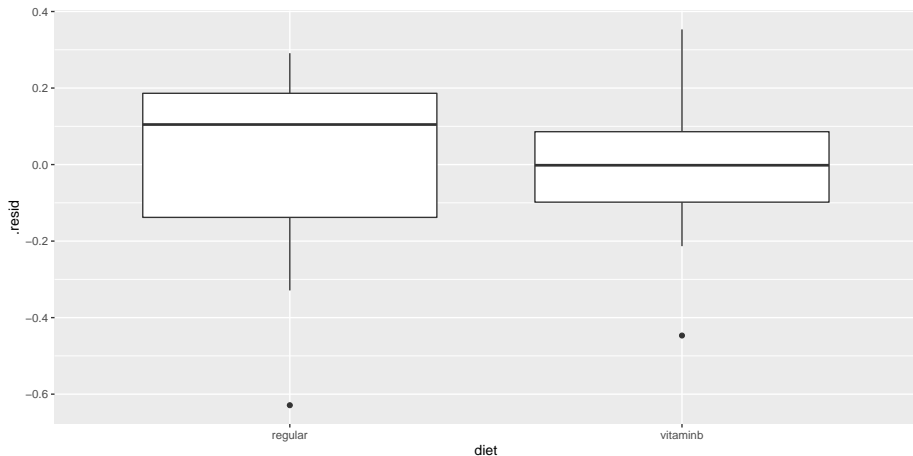
Residuals vs rat size

```
ggplot(vitaminb.3a, aes(x = ratsize, y = .resid)) +  
  geom_boxplot()
```



Residuals vs diet

```
ggplot(vitaminb.3a, aes(x = diet, y = .resid)) +  
  geom_boxplot()
```



Comments

- there are low outliers on the plot against diet
- residuals for obese rats seem more spread out than for lean rats
- case for transformation of rat weights
- however, story from our analysis very clear:
 - rat size strongly significant
 - diet nowhere near significant
- and so expect transformation to make no difference to conclusions.

The auto noise data

In 1973, the President of Texaco cited an automobile filter developed by Associated Octel Company as effective in reducing pollution. However, questions had been raised about the effects of filter silencing. He referred to the data included in the report (and below) as evidence that the silencing properties of the Octel filter were at least equal to those of standard silencers.

```
u <- "http://ritsokiguess.site/datafiles/autonoise.txt"
autonoise <- read_table(u)
```

```
##
## -- Column specification -----
## cols(
##   noise = col_double(),
##   size = col_character(),
##   type = col_character(),
##   side = col_character())
```

The data

```
autonoise
```

```
## # A tibble: 36 x 4
##   noise size  type  side
##   <dbl> <chr> <chr> <chr>
## 1   840 M      Std    R
## 2   770 L      Octel  L
## 3   820 M      Octel  R
## 4   775 L      Octel  R
## 5   825 M      Octel  L
## 6   840 M      Std    R
## 7   845 M      Std    L
## 8   825 M      Octel  L
## 9   815 M      Octel  L
## 10  845 M      Std    R
## # ... with 26 more rows
```

Making boxplot

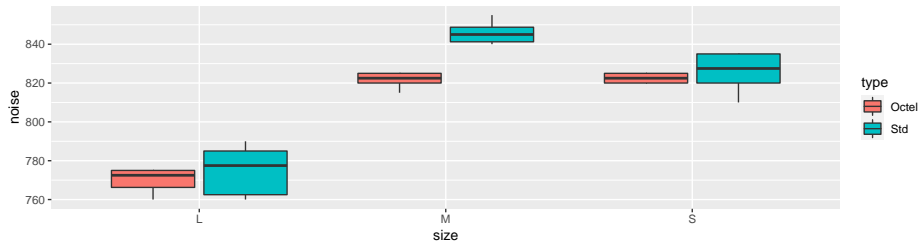
- Make a boxplot, but have combinations of filter type and engine size.
- Use grouped boxplot again, thus:

```
g <- autonoise %>%  
  ggplot(aes(x = size, y = noise, fill = type)) +  
  geom_boxplot()
```

The boxplot

- See difference in engine noise between Octel and standard is larger for medium engine size than for large or small.
- Some evidence of differences in spreads (ignore for now):

gg



ANOVA

```
autonoise.1 <- aov(noise ~ size * type, data = autonoise)
summary(autonoise.1)
```

```
##              Df Sum Sq Mean Sq F value    Pr(>F)
## size          2  26051   13026  199.119 < 2e-16 ***
## type          1   1056    1056   16.146 0.000363 ***
## size:type      2    804     402    6.146 0.005792 **
## Residuals     30   1962         65
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

- The interaction is significant, as we suspected from the boxplots.
- The within-group spreads don't look very equal, but only based on 6 obs each.

Tukey: ouch!

```
autonoise.2 <- TukeyHSD(autonoise.1)
autonoise.2$`size:type`
```

##		diff	lwr	upr
##	M:Octel-L:Octel	51.6666667	37.463511	65.869823
##	S:Octel-L:Octel	52.5000000	38.296844	66.703156
##	L:Std-L:Octel	5.0000000	-9.203156	19.203156
##	M:Std-L:Octel	75.8333333	61.630177	90.036489
##	S:Std-L:Octel	55.8333333	41.630177	70.036489
##	S:Octel-M:Octel	0.8333333	-13.369823	15.036489
##	L:Std-M:Octel	-46.6666667	-60.869823	-32.463511
##	M:Std-M:Octel	24.1666667	9.963511	38.369823
##	S:Std-M:Octel	4.1666667	-10.036489	18.369823
##	L:Std-S:Octel	-47.5000000	-61.703156	-33.296844
##	M:Std-S:Octel	23.3333333	9.130177	37.536489
##	S:Std-S:Octel	3.3333333	-10.869823	17.536489
##	M:Std-L:Std	70.8333333	56.630177	85.036489
##	S:Std-L:Std	50.8333333	36.630177	65.036489
##	S:Std-M:Std	-20.0000000	-34.203156	-5.796844
##		p adj		
##	M:Octel-L:Octel	6.033496e-11		
##	S:Octel-L:Octel	4.089762e-11		
##	L:Std-L:Octel	8.890358e-01		
##	M:Std-L:Octel	4.000000e-14		

Interaction plot

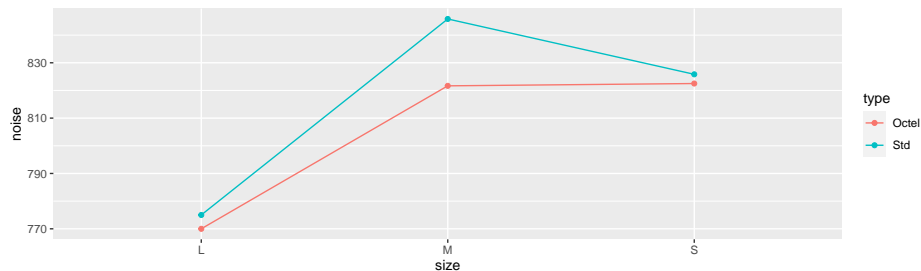
- This time, don't have summary of mean noise for each size-type combination.
- One way is to compute summaries (means) first, and feed into `ggplot` as in vitamin B example.
- Or, have `ggplot` compute them for us, thus:

```
g <- ggplot(autonoise, aes(  
  x = size, y = noise,  
  colour = type, group = type  
)) +  
  stat_summary(fun = mean, geom = "point") +  
  stat_summary(fun = mean, geom = "line")
```

Interaction plot

The lines are definitely *not* parallel, showing that the effect of type is different for medium-sized engines than for others:

g

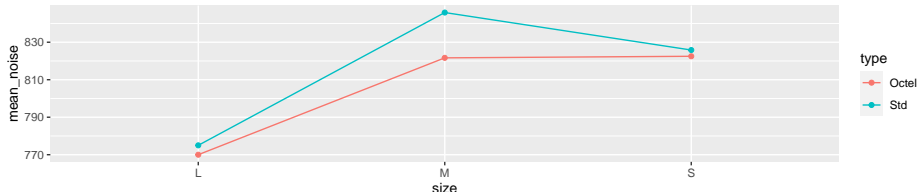


If you don't like that...

...then compute the means first, in a pipeline:

```
autonoise %>%
  group_by(size, type) %>%
  summarize(mean_noise = mean(noise)) %>%
  ggplot(aes(
    x = size, y = mean_noise, group = type,
    colour = type
  )) + geom_point() + geom_line()
```

`summarise()` has grouped output by 'size'. You can override using the `



Simple effects for auto noise example

- In auto noise example, weren't interested in all comparisons between car size and filter type combinations.
- Wanted to demonstrate (lack of) difference between filter types *for each car type*.
- These are called **simple effects** of one variable (filter type) conditional on other variable (car type).
- To do this, pull out just the data for small cars, compare noise for the two filter types. Then repeat for medium and large cars. (Three one-way ANOVAs.)

Do it using dplyr tools

- Small cars:

```
autonoise %>%
  filter(size == "S") %>%
  aov(noise ~ type, data = .) %>%
  summary()
```

```
##              Df Sum Sq Mean Sq F value Pr(>F)
## type           1   33.3   33.33    0.548  0.476
## Residuals     10  608.3   60.83
```

- No filter difference for small cars.
- For Medium, change S to M and repeat.

Simple effect of filter type for medium cars

```
autonoise %>%
  filter(size == "M") %>%
  aov(noise ~ type, data = .) %>%
  summary()
```

```
##              Df Sum Sq Mean Sq F value    Pr(>F)
## type           1 1752.1  1752.1    68.93 8.49e-06 ***
## Residuals     10   254.2    25.4
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

- There *is* an effect of filter type for medium cars. Look at means to investigate (over).

Mean noise for each filter type

...for medium engine size:

```
autonoise %>%
  filter(size == "M") %>%
  group_by(type) %>%
  summarize(m = mean(noise))
```

```
## # A tibble: 2 x 2
##   type      m
##   <chr> <dbl>
## 1 Octel  822.
## 2 Std    846.
```

- Octel filters produce *less* noise for medium cars.

Large cars

- Large cars:

```
autonoise %>%
  filter(size == "L") %>%
  aov(noise ~ type, data = .) %>%
  summary()
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
## type	1	75	75	0.682	0.428
## Residuals	10	1100	110		

- No significant difference again.

All at once, using split/apply/combine

The “split” part:

```
autonoise %>%  
  group_by(size) %>%  
  nest()
```

```
## # A tibble: 3 x 2  
##   size data  
##   <chr> <list>  
## 1 M     <tibble [12 x 3]>  
## 2 L     <tibble [12 x 3]>  
## 3 S     <tibble [12 x 3]>
```

Now have *three* rows, with the data frame for each size encoded as *one element* of this data frame.

Apply

- Write function to do aov on a data frame with columns noise and type, returning P-value:

```
aov_pval <- function(x) {
  noise.1 <- aov(noise ~ type, data = x)
  gg <- tidy(noise.1)
  gg$p.value[1]
}
```

- Test it:

```
autonoise %>%
  filter(size == "L") %>%
  aov_pval()
```

```
## [1] 0.428221
```

- Check.

Combine

- Apply this function to each of the nested data frames (one per engine size):

```
autonoise %>%
  nest_by(size) %>%
  rowwise() %>%
  mutate(p_val = aov_pval(data))
```

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

##	size	data	p_val
##	<chr>	<list<tibble[,3]>>	<dbl>
## 1	L	[12 x 3]	0.428
## 2	M	[12 x 3]	0.00000849
## 3	S	[12 x 3]	0.476

Tidy up

- The data column was stepping-stone to getting answer. Don't need it any more:

```
autonoise %>%
  nest_by(size) %>%
  rowwise() %>%
  mutate(p_val = aov_pval(data)) %>%
  select(-data) -> simple_effects
simple_effects
```

```
## # A tibble: 3 x 2
##   size      p_val
##   <chr>    <dbl>
## 1 L      0.428
## 2 M      0.00000849
## 3 S      0.476
```

Simultaneous tests

- When testing simple effects, doing several tests at once. (In this case, 3.) Have to adjust P-values for this. Eg. Holm:

```
simple_effects %>% ungroup() %>% arrange(p_val) %>%
  mutate(multiplier = 4 - row_number()) %>%
  mutate(p_val_adj = p_val * multiplier)
```

```
## # A tibble: 3 x 4
##   size      p_val multiplier p_val_adj
##   <chr>    <dbl>      <dbl>    <dbl>
## 1 M      0.00000849          3 0.0000255
## 2 L      0.428              2 0.856
## 3 S      0.476              1 0.476
```

- No change in rejection decisions.
- Octel filters sig. better in terms of noise for medium cars, and not sig. different for other sizes.
- Octel filters never significantly worse than standard ones.

Confidence intervals

- Perhaps better way of assessing simple effects: look at *confidence intervals* rather than tests.
- Gives us sense of accuracy of estimation, and thus whether non-significance might be lack of power: “absence of evidence is not evidence of absence”.
- Works here because *two* filter types, using *t*.test for each engine type.
- Want to show that the Octel filter is equivalent to or better than the standard filter, in terms of engine noise.

Equivalence and noninferiority

- Known as “equivalence testing” in medical world. A good read: [link](#). Basic idea: decide on size of difference δ that would be considered “equivalent”, and if CI entirely inside $\pm\delta$, have evidence in favour of equivalence.
- We really want to show that the Octel filters are “no worse” than the standard one: that is, equivalent *or better* than standard filters.
- Such a “noninferiority test” done by checking that upper limit of CI, new minus old, is *less* than δ . (This requires careful thinking about (i) which way around the difference is and (ii) whether a higher or lower value is better.)

CI for small cars

Same idea as for simple effect test:

```
autonoise %>%  
  filter(size == "S") %>%  
  t.test(noise ~ type, data = .) %>%  
  pluck("conf.int")
```

```
## [1] -14.517462    7.850795  
## attr(,"conf.level")  
## [1] 0.95
```


CI for medium cars

```
autonoise %>%  
  filter(size == "M") %>%  
  t.test(noise ~ type, data = .) %>%  
  pluck("conf.int")
```

```
## [1] -30.75784 -17.57549  
## attr(,"conf.level")  
## [1] 0.95
```

CI for large cars

```
autonoise %>%  
  filter(size == "L") %>%  
  t.test(noise ~ type, data = .) %>%  
  pluck("conf.int")  
  
## [1] -19.270673    9.270673  
## attr(,"conf.level")  
## [1] 0.95
```

Or, all at once: split/apply/combine

```
ci_func <- function(x) {
  tt <- t.test(noise ~ type, data = x)
  tt$conf.int
}
```

```
autonoise %>% nest_by(size) %>%
  rowwise() %>%
  mutate(ci = list(ci_func(data))) %>%
  unnest_wider(ci) -> cis
```

```
## New names:
## * `` -> ...1
## * `` -> ...2
## New names:
## * `` -> ...1
## * `` -> ...2
## New names:
```

Results

```
cis
```

```
## # A tibble: 3 x 4
##   size          data ...1    ...2
##   <chr> <list<tibble[,3]>> <dbl> <dbl>
## 1 L      [12 x 3] -19.3    9.27
## 2 M      [12 x 3] -30.8  -17.6
## 3 S      [12 x 3] -14.5    7.85
```

Procedure

- Function to get CI of difference in noise means for types of filter on input data frame
- Nest by size (mini-df data per size)
- Calculate CI for each thing in data: CI is two numbers long
- `unnest ci` column (wider) to see two numbers in each CI.

CIs and noninferiority test

- Suppose we decide that a 20 dB difference would be considered equivalent. (I have no idea whether that is reasonable.)
- Intervals:

```
cis
```

```
## # A tibble: 3 x 4
##   size          data ...1    ...2
##   <chr> <list<tibble[,3]>> <dbl>  <dbl>
## 1 L      [12 x 3] -19.3    9.27
## 2 M      [12 x 3] -30.8  -17.6
## 3 S      [12 x 3] -14.5    7.85
```

Comments

- In all cases, upper limit of CI is less than 20 dB. The Octel filters are “noninferior” to the standard ones.
- Caution: we did 3 procedures at once again. The true confidence level is not 95%. (Won't worry about that here.)

Contrasts in ANOVA

- Sometimes, don't want to compare *all* groups, only *some* of them.
- Might be able to specify these comparisons ahead of time; other comparisons of no interest.
- Wasteful to do ANOVA and Tukey.

Example: chainsaw kickback

- From [link](#).
- Forest manager concerned about safety of chainsaws issued to field crew. 4 models of chainsaws, measure “kickback” (degrees of deflection) for 5 of each:

A	B	C	D

42	28	57	29
17	50	45	29
24	44	48	22
39	32	41	34
43	61	54	30

- So far, standard 1-way ANOVA: what differences are there among models?

chainsaw kickback (2)

- But: models A and D are designed to be used at home, while models B and C are industrial models.
- Suggests these comparisons of interest:
- home vs. industrial
- the two home models A vs. D
- the two industrial models B vs. C.
- Don't need to compare *all* the pairs of models.

What is a contrast?

- Contrast is a linear combination of group means.
- Notation: μ_A for (population) mean of group A , and so on.
- In example, compare two home models: $H_0 : \mu_A - \mu_D = 0$.
- Compare two industrial models: $H_0 : \mu_B - \mu_C = 0$.
- Compare average of two home models vs. average of two industrial models: $H_0 : \frac{1}{2}(\mu_A + \mu_D) - \frac{1}{2}(\mu_B + \mu_C) = 0$ or $H_0 : 0.5\mu_A - 0.5\mu_B - 0.5\mu_C + 0.5\mu_D = 0$.
- Note that coefficients of contrasts add to 0, and right-hand side is 0.

Contrasts in R

- Comparing two home models A and D ($\mu_A - \mu_D = 0$):

```
c.home <- c(1, 0, 0, -1)
```

- Comparing two industrial models B and C ($\mu_B - \mu_C = 0$):

```
c.industrial <- c(0, 1, -1, 0)
```

- Comparing home average vs. industrial average
($0.5\mu_A - 0.5\mu_B - 0.5\mu_C + 0.5\mu_D = 0$):

```
c.home.ind <- c(0.5, -0.5, -0.5, 0.5)
```

Orthogonal contrasts

- What happens if we multiply the contrast coefficients one by one?

```
c.home * c.industrial
```

```
## [1] 0 0 0 0
```

```
c.home * c.home.ind
```

```
## [1] 0.5 0.0 0.0 -0.5
```

```
c.industrial * c.home.ind
```

```
## [1] 0.0 -0.5 0.5 0.0
```

- in each case, the results **add up to zero**. Such contrasts are called **orthogonal**.

Orthogonal contrasts (2)

- Compare these:

```
c1 <- c(1, -1, 0)
c2 <- c(0, 1, -1)
sum(c1 * c2)
```

```
## [1] -1
```

Not zero, so $c1$ and $c2$ are *not* orthogonal.

- Orthogonal contrasts are much easier to deal with.
- Can use non-orthogonal contrasts, but more trouble (beyond us).

Read in data

```
url <- "http://ritsokiguess.site/datafiles/chainsaw.txt"
chain.wide <- read_table(url)
chain.wide
```

```
## # A tibble: 5 x 4
##       A      B      C      D
##   <dbl> <dbl> <dbl> <dbl>
## 1    42    28    57    29
## 2    17    50    45    29
## 3    24    44    48    22
## 4    39    32    41    34
## 5    43    61    54    30
```

Tidying

Need all the kickbacks in *one* column:

```
chain.wide %>%  
  pivot_longer(A:D, names_to = "model",  
               names_ptypes = list(model=factor()),  
               values_to = "kickback") -> chain
```


Starting the analysis (2)

The proper data frame (tiny):

```
chain
```

```
## # A tibble: 20 x 2
##   model kickback
##   <fct>      <dbl>
## 1 A          42
## 2 B          28
## 3 C          57
## 4 D          29
## 5 A          17
## 6 B          50
## 7 C          45
## 8 D          29
## 9 A          24
## 10 B         44
## 11 C         48
## 12 D         22
## 13 A         39
## 14 B         32
## 15 C         41
## 16 D         34
## 17 A         43
## 18 B         61
## 19 C         54
## 20 D         30
```

Setting up contrasts

```
m <- cbind(c.home, c.industrial, c.home.ind)
```

```
m
```

```
##      c.home c.industrial c.home.ind
## [1,]      1           0         0.5
## [2,]      0           1        -0.5
## [3,]      0          -1        -0.5
## [4,]     -1           0         0.5
```

```
contrasts(chain$model) <- m
```

ANOVA *as if* regression

```
chain.1 <- lm(kickback ~ model, data = chain)
summary(chain.1)
```

```
##
## Call:
## lm(formula = kickback ~ model, data = chain)
##
## Residuals:
```

	Min	1Q	Median	3Q	Max
##	-16.00	-7.10	0.60	6.25	18.00

```
##
## Coefficients:
```

	Estimate	Std. Error	t value
## (Intercept)	38.450	2.179	17.649
## modelc.home	2.100	3.081	0.682
## modelc.industrial	-3.000	3.081	-0.974
## modelc.home.ind	-15.100	4.357	-3.466

```
##
## Pr(>|t|)
```

	Pr(> t)
## (Intercept)	6.52e-12 ***
## modelc.home	0.50524
## modelc.industrial	0.34469
## modelc.home.ind	0.00319 **

```
## ---
## Signif. codes:  0.001 '***' 0.01 '**' 0.05 '*' 0.1 '.' 0.2 ' ' 1
```

Conclusions

```
tidy(chain.1) %>% select(term, p.value)
```

```
## # A tibble: 4 x 2
##   term                p.value
##   <chr>              <dbl>
## 1 (Intercept)        6.52e-12
## 2 modelc.home         5.05e- 1
## 3 modelc.industrial  3.45e- 1
## 4 modelc.home.ind     3.19e- 3
```

- Two home models not sig. diff. (P-value 0.51)
- Two industrial models not sig. diff. (P-value 0.34)
- Home, industrial models *are* sig. diff. (P-value 0.0032).

Means by model

- The means:

```
chain %>%
  group_by(model) %>%
  summarize(mean.kick = mean(kickback)) %>%
  arrange(desc(mean.kick))
```

```
## # A tibble: 4 x 2
##   model mean.kick
##   <fct>     <dbl>
## 1 C         49
## 2 B         43
## 3 A         33
## 4 D        28.8
```

- Home models A & D have less kickback than industrial ones B & C.
- Makes sense because industrial users should get training to cope with additional kickback.

Section 10

Cluster analysis

Cluster Analysis

- One side-effect of discriminant analysis: could draw picture of data (if 1st 2s LDs told most of story) and see which individuals “close” to each other.
- Discriminant analysis requires knowledge of groups.
- Without knowledge of groups, use *cluster analysis*: see which individuals close together, which groups suggested by data.
- Idea: see how individuals group into “clusters” of nearby individuals.
- Base on “dissimilarities” between individuals.
- Or base on standard deviations and correlations between variables (assesses dissimilarity behind scenes).

Packages

```
library(MASS) # for lda later  
library(tidyverse)  
library(spatstat) # for crossdist later  
library(ggrepel)
```


One to ten in 11 languages

	English	Norwegian	Danish	Dutch	German
1	one	en	en	een	eins
2	two	to	to	twee	zwei
3	three	tre	tre	drie	drei
4	four	fire	fire	vier	vier
5	five	fem	fem	vijf	funf
6	six	seks	seks	zes	sechs
7	seven	sju	syv	zeven	sieben
8	eight	atte	otte	acht	acht
9	nine	ni	ni	negen	neun
10	ten	ti	ti	tien	zehn

One to ten

	French	Spanish	Italian	Polish	Hungarian	Finnish
1	un	uno	uno	jeden	egy	yksi
2	deux	dos	due	dwa	ketto	kaksi
3	trois	tres	tre	trzy	harom	kolme
4	quatre	cuatro	quattro	cztery	negy	nelja
5	cinq	cinco	cinque	piec	ot	viisi
6	six	seis	sei	szesc	hat	kuusi
7	sept	siete	sette	siedem	het	seitseman
8	huit	ocho	otto	osiem	nyolc	kahdeksan
9	neuf	nueve	nove	dziewiec	kilenc	yhdeksan
10	dix	diez	dieci	dziesiec	tiz	kymmenen

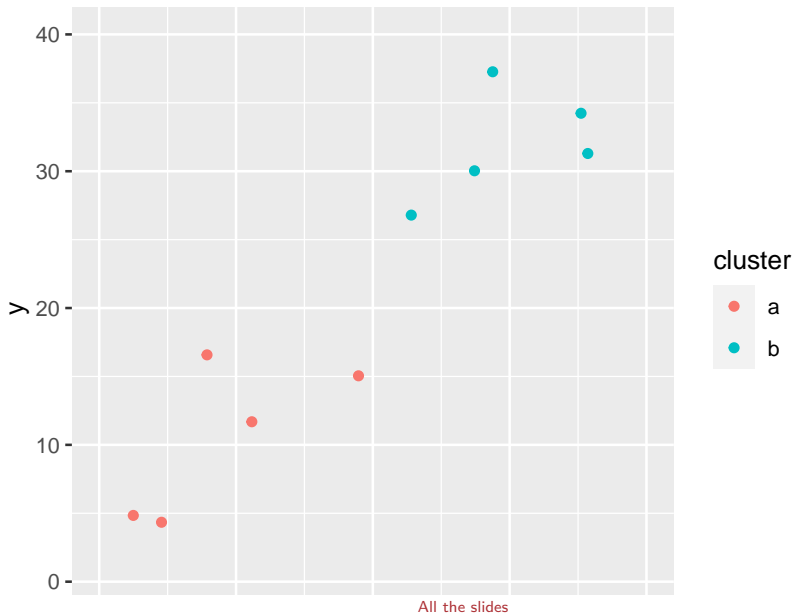
Dissimilarities and languages example

- Can define dissimilarities how you like (whatever makes sense in application).
- Sometimes defining “similarity” makes more sense; can turn this into dissimilarity by subtracting from some maximum.
- Example: numbers 1–10 in various European languages. Define similarity between two languages by counting how often the same number has a name starting with the same letter (and dissimilarity by how often number has names starting with different letter).
- Crude (doesn't even look at most of the words), but see how effective.

Two kinds of cluster analysis

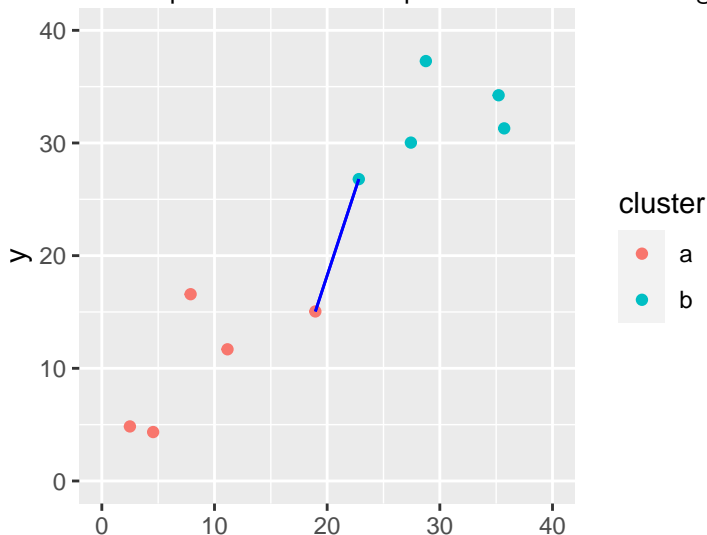
- Looking at process of forming clusters (of similar languages): **hierarchical cluster analysis** (`hclust`).
- Start with each individual in cluster by itself.
- Join “closest” clusters one by one until all individuals in one cluster.
- How to define closeness of two *clusters*? Not obvious, investigate in a moment.
- Know how many clusters: which division into that many clusters is “best” for individuals? **K-means clustering** (`kmeans`).

Two made-up clusters



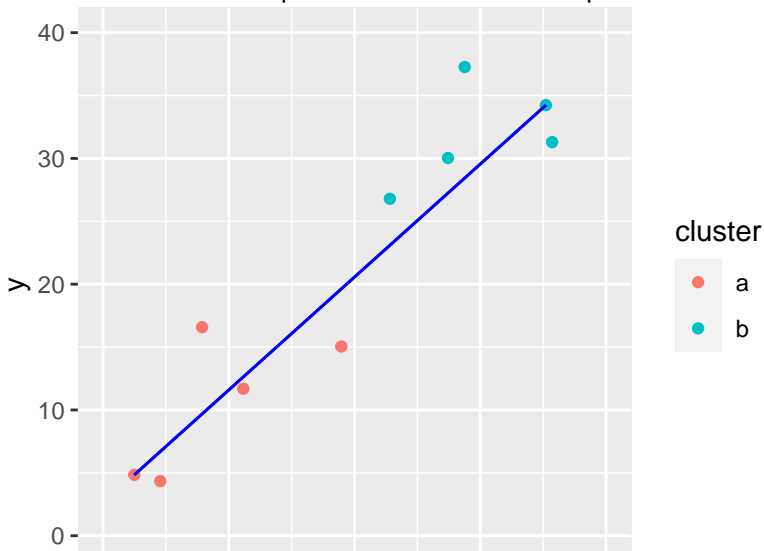
Single-linkage distance

Find the red point and the blue point that are closest together:



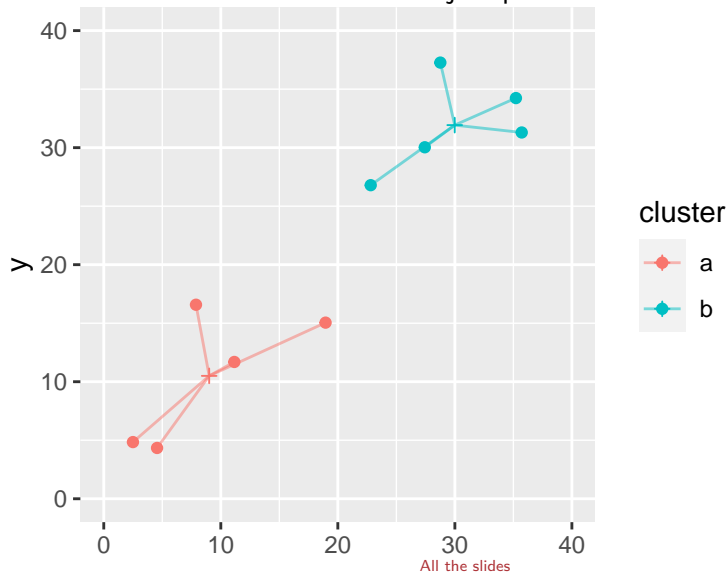
Complete linkage

Find the red and blue points that are farthest apart:



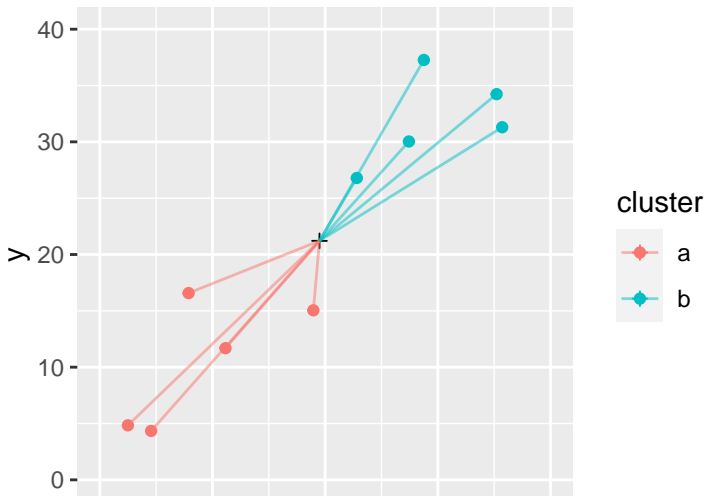
Ward's method

Work out mean of each cluster and join point to its mean:



Ward's method part 2 xxx

Now imagine combining the two clusters and working out overall mean.
Join each point to this mean:



Ward's method part 3

- Sum of squares (ii) will be bigger than (i) (points closer to own cluster mean than combined mean).
- Ward's distance is (ii) minus (i).
- Think of as “cost” of combining clusters:
- if clusters close together, (ii) only a little larger than (i)
- if clusters far apart, (ii) a lot larger than (i) (as in example).

Hierarchical clustering revisited

- Single linkage, complete linkage, Ward are ways of measuring closeness of clusters.
- Use them, starting with each observation in own cluster, to repeatedly combine two closest clusters until all points in one cluster.
- They will give different answers (clustering stories).
- Single linkage tends to make “stringy” clusters because clusters can be very different apart from two closest points.
- Complete linkage insists on whole clusters being similar.
- Ward tends to form many small clusters first.

Dissimilarity data in R

Dissimilarities for language data were how many number names had *different* first letter:

```
my_url <- "http://ritsokiguess.site/datafiles/languages.txt"
(number.d <- read_table(my_url))
```

```
## # A tibble: 11 x 12
```

##	la	en	no	dk	nl	de	fr	es	it
##	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
## 1	en	0	2	2	7	6	6	6	6
## 2	no	2	0	1	5	4	6	6	6
## 3	dk	2	1	0	6	5	6	5	5
## 4	nl	7	5	6	0	5	9	9	9
## 5	de	6	4	5	5	0	7	7	7
## 6	fr	6	6	6	9	7	0	2	1
## 7	es	6	6	5	9	7	2	0	1
## 8	it	6	6	5	9	7	1	1	0
## 9	pl	7	7	6	10	8	5	3	4
## 10	hu	9	8	8	8	9	10	10	10
## 11	fi	9	9	9	9	9	9	9	8

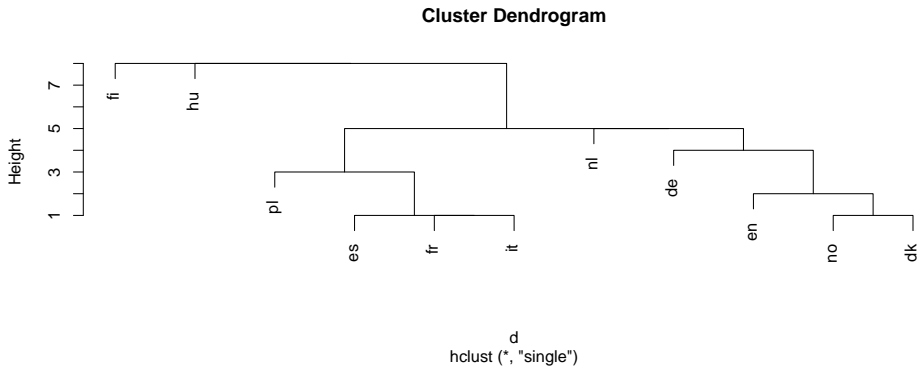
Making a distance object

```
d <- number.d %>%
  select(-la) %>%
  as.dist()
d
```

```
##      en no dk nl de fr es it pl hu
## no   2
## dk   2  1
## nl   7  5  6
## de   6  4  5  5
## fr   6  6  6  9  7
## es   6  6  5  9  7  2
## it   6  6  5  9  7  1  1
## pl   7  7  6 10  8  5  3  4
## hu   9  8  8  8  9 10 10 10 10
## fi   9  9  9  9  9  9  9  8  9  8
```

Cluster analysis and dendrogram

```
d.hc <- hclust(d, method = "single")  
plot(d.hc)
```



Comments

- Tree shows how languages combined into clusters.
- First (bottom), Spanish, French, Italian joined into one cluster, Norwegian and Danish into another.
- Later, English joined to Norse languages, Polish to Romance group.
- Then German, Dutch make a Germanic group.
- Finally, Hungarian and Finnish joined to each other and everything else.

Clustering process

```
d.hc$labels
```

```
## [1] "en" "no" "dk" "nl" "de" "fr" "es" "it" "pl" "hu" "fi"
```

```
d.hc$merge
```

```
##      [,1] [,2]
```

```
## [1,]   -2   -3
```

```
## [2,]   -6   -8
```

```
## [3,]   -7    2
```

```
## [4,]   -1    1
```

```
## [5,]   -9    3
```

```
## [6,]   -5    4
```

```
## [7,]   -4    6
```

```
## [8,]    5    7
```

```
## [9,]  -10    8
```

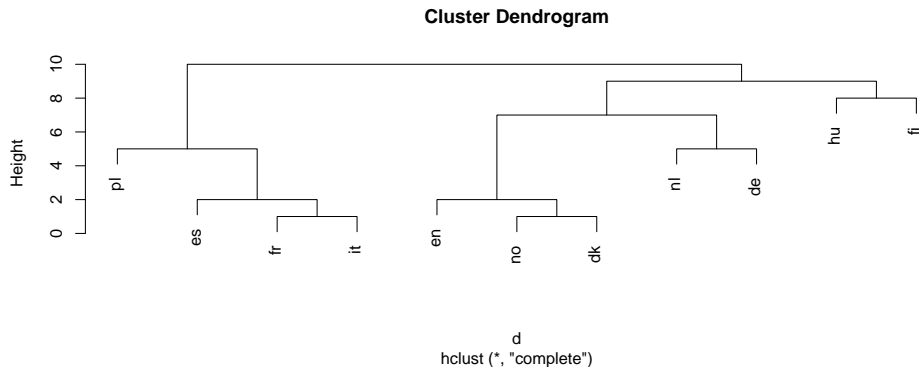
```
## [10,] -11    9
```


Comments

- Lines of merge show what was combined
 - First, languages 2 and 3 (no and dk)
 - Then languages 6 and 8 (fr and it)
 - Then #7 combined with cluster formed at step 2 (es joined to fr and it).
 - Then en joined to no and dk ...
 - Finally fi joined to all others.

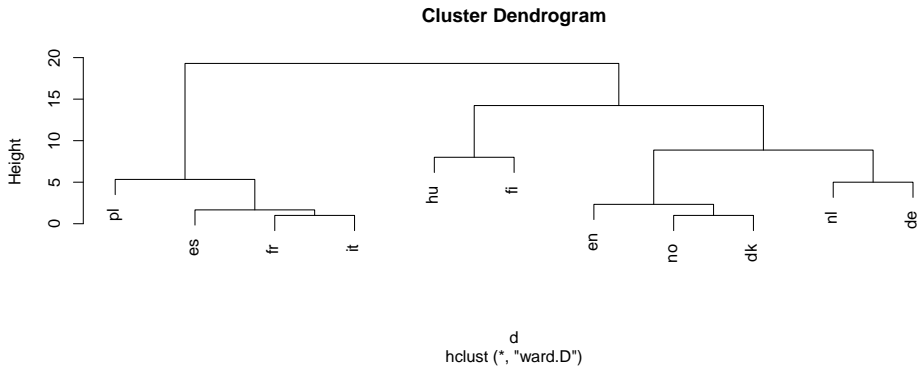
Complete linkage

```
d.hc <- hclust(d, method = "complete")  
plot(d.hc)
```



Ward

```
d.hc <- hclust(d, method = "ward.D")  
plot(d.hc)
```



Chopping the tree

- Three clusters (from Ward) looks good:

```
cutree(d.hc, 3)
```

```
## en no dk nl de fr es it pl hu fi  
##  1  1  1  1  1  2  2  2  2  3  3
```

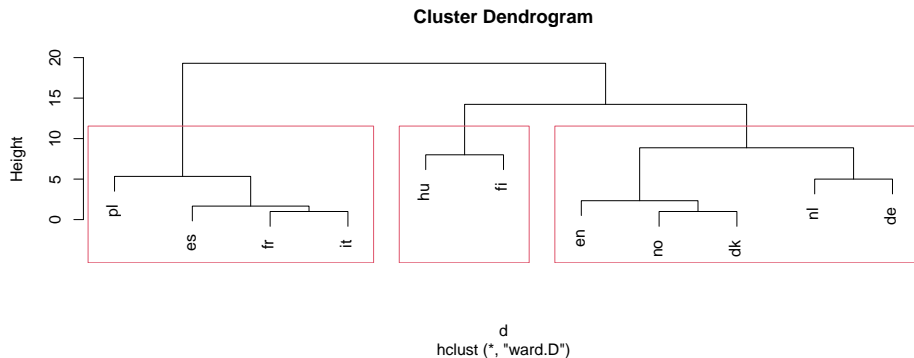
Turning the “named vector” into a data frame

```
cutree(d.hc, 3) %>% enframe(name="country", value="cluster")
```

```
## # A tibble: 11 x 2
##   country cluster
##   <chr>      <int>
## 1 en         1
## 2 no         1
## 3 dk         1
## 4 nl         1
## 5 de         1
## 6 fr         2
## 7 es         2
## 8 it         2
## 9 pl         2
## 10 hu        3
## 11 fi        3
```

Drawing those clusters on the tree

```
plot(d.hc)  
rect.hclust(d.hc, 3)
```



Comparing single-linkage and Ward

- In Ward, Dutch and German get joined earlier (before joining to Germanic cluster).
- Also Hungarian and Finnish get combined earlier.

Making those dissimilarities

Original data:

```
my_url <- "http://ritsokiguess.site/datafiles/one-ten.txt"
lang <- read_delim(my_url, " ")
lang
```

```
## # A tibble: 10 x 11
##   en    no    dk    nl    de    fr    es    it    pl
##   <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
## 1 one   en    en    een  eins  un    uno   uno   jeden
## 2 two   to    to    twee zwei  deux  dos   due   dwa
## 3 three tre  tre  drie  drei  trois tres  tre   trzy
## 4 four  fire  fire  vier  vier  quatre quat~ quatt~ cztery
## 5 five  fem   fem   vijf  funf  cinq  cinco cinco piec
## 6 six   seks  seks  zes   sechs six   seis  sei   szesc
## 7 seven sju   syv   zeven sieben sept  siete sette siedem
## 8 eight atte  otte  acht  acht  huit  ocho  otto  osiem
## 9 nine  ni    ni    negen neun  neuf  nueve nove  dziew~
## 10 ten  ti    ti    tien  zehn  dix   diez  dieci dzies~
## # ... with 2 more variables: hu <chr>, fi <chr>
```

It would be a lot easier to extract the first letter if the number names were all in one column.

Tidy, and extract first letter

```
lang %>% mutate(number=row_number()) %>%
  pivot_longer(-number, names_to="language", values_to="name") %>%
  mutate(first=str_sub(name,1,1)) -> lang.long
lang.long %>% print(n=12)
```

```
## # A tibble: 110 x 4
##   number language name  first
##   <int> <chr>    <chr> <chr>
## 1       1 en      one    o
## 2       1 no      en     e
## 3       1 dk      en     e
## 4       1 nl      een    e
## 5       1 de      eins   e
## 6       1 fr      un     u
## 7       1 es      uno    u
## 8       1 it      uno    u
## 9       1 pl      jeden  j
## 10      1 hu      egy    e
## 11      1 fi      yksi   y
## 12      2 en      two    t
```

Calculating dissimilarity

- Suppose we wanted dissimilarity between English and Norwegian. It's the number of first letters that are different.
- First get the lines for English:

```
english <- lang.long %>% filter(language == "en")
english
```

```
## # A tibble: 10 x 4
##   number language name first
##   <int> <chr>    <chr> <chr>
## 1      1 en      one   o
## 2      2 en      two   t
## 3      3 en     three t
## 4      4 en     four  f
## 5      5 en     five  f
## 6      6 en     six   s
## 7      7 en    seven s
## 8      8 en    eight e
## 9      9 en    nine  n
## 10     10 en    ten   t
```

And then the lines for Norwegian

```
norwegian <- lang.long %>% filter(language == "no")
norwegian
```

```
## # A tibble: 10 x 4
##   number language name first
##   <int> <chr>    <chr> <chr>
## 1      1 no      en     e
## 2      2 no      to     t
## 3      3 no      tre    t
## 4      4 no      fire   f
## 5      5 no      fem    f
## 6      6 no      seks   s
## 7      7 no      sju    s
## 8      8 no      atte   a
## 9      9 no      ni     n
## 10    10 no      ti     t
```

And now we want to put them side by side, matched by number. This is what `left_join` does. (A “join” is a lookup of values in one table using another.)

The join

```
english %>% left_join(norwegian, by = "number")
```

```
## # A tibble: 10 x 7
##   number language.x name.x first.x language.y name.y first.y
##   <int> <chr>      <chr> <chr> <chr>      <chr> <chr>
## 1     1     en      one   o     no        en     e
## 2     2     en      two   t     no        to     t
## 3     3     en      three t     no        tre     t
## 4     4     en      four  f     no        fire    f
## 5     5     en      five  f     no        fem     f
## 6     6     en      six   s     no        seks    s
## 7     7     en      seven s     no        sju     s
## 8     8     en      eight e     no        atte    a
## 9     9     en      nine  n     no        ni      n
## 10    10    en      ten   t     no        ti      t
```

first.x is 1st letter of English word, first.y 1st letter of Norwegian word.

Counting the different ones

```
english %>%
  left_join(norwegian, by = "number") %>%
  count(different=(first.x != first.y))
```

```
## # A tibble: 2 x 2
##   different     n
##   <lgl>       <int>
## 1 FALSE         8
## 2 TRUE          2
```

or

```
english %>%
  left_join(norwegian, by = "number") %>%
  count(different=(first.x != first.y)) %>%
  filter(different) %>% pull(n) -> ans

ans
```

```
## [1] 2
```

Words for 1 and 8 start with different letter; rest are same.

A language with itself

The answer should be zero:

```
english %>%  
  left_join(english, by = "number") %>%  
  count(different=(first.x != first.y)) %>%  
  filter(different) %>% pull(n) -> ans  
ans
```

```
## integer(0)
```

- but this is “an integer vector of length zero”.
- so we have to allow for this possibility when we write a function to do it.

Function to do this for any two languages

```
countdiff <- function(lang.1, lang.2, d) {  
  d %>% filter(language == lang.1) -> lang1d  
  d %>% filter(language == lang.2) -> lang2d  
  lang1d %>%  
    left_join(lang2d, by = "number") %>%  
    count(different = (first.x != first.y)) %>%  
    filter(different) %>% pull(n) -> ans  
  # if ans has length zero, set answer to (integer) zero.  
  ifelse(length(ans)==0, 0L, ans)  
}
```

Testing

```
countdiff("en", "no", lang.long)
```

```
## [1] 2
```

```
countdiff("en", "en", lang.long)
```

```
## [1] 0
```

English and Norwegian have two different; English and English have none different.

Check.

For all pairs of languages?

- First need all the languages:

```
languages <- names(lang)
languages
```

```
## [1] "en" "no" "dk" "nl" "de" "fr" "es" "it" "pl"
## [10] "hu" "fi"
```

- and then all *pairs* of languages:

```
pairs <- crossing(lang = languages, lang2 = languages)
```

Some of these

```
pairs %>% slice(1:12)
```

```
## # A tibble: 12 x 2
##   lang lang2
##   <chr> <chr>
## 1 de    de
## 2 de    dk
## 3 de    en
## 4 de    es
## 5 de    fi
## 6 de    fr
## 7 de    hu
## 8 de    it
## 9 de    nl
## 10 de   no
## 11 de   pl
## 12 dk    de
```

Run countdiff for all those language pairs

```
pairs %>% rowwise() %>%
  mutate(diff = countdiff(lang, lang2, lang.long)) -> thediff
thediff
```

```
## # A tibble: 121 x 3
##   lang lang2 diff
##   <chr> <chr> <int>
## 1 de    de      0
## 2 de    dk      5
## 3 de    en      6
## 4 de    es      7
## 5 de    fi      9
## 6 de    fr      7
## 7 de    hu      9
## 8 de    it      7
## 9 de    nl      5
## 10 de   no      4
## # ... with 111 more rows
```

Make square table of these

```
thediff %>% pivot_wider(names_from=lang2, values_from=diff)
```

```
## # A tibble: 11 x 12
##   lang    de    dk    en    es    fi    fr    hu    it
##   <chr> <int> <int> <int> <int> <int> <int> <int> <int>
## 1 de      0     5     6     7     9     7     9     7
## 2 dk      5     0     2     5     9     6     8     5
## 3 en      6     2     0     6     9     6     9     6
## 4 es      7     5     6     0     9     2    10     1
## 5 fi      9     9     9     9     0     9     8     9
## 6 fr      7     6     6     2     9     0    10     1
## 7 hu      9     8     9    10     8    10     0    10
## 8 it      7     5     6     1     9     1    10     0
## 9 nl      5     6     7     9     9     9     8     9
## 10 no     4     1     2     6     9     6     8     6
## 11 pl      8     6     7     3     9     5    10     4
## # ... with 3 more variables: nl <int>, no <int>, pl <int>
```

and that was where we began.

Another example

Birth, death and infant mortality rates for 97 countries (variables not dissimilarities):

24.7	5.7	30.8	Albania	12.5	11.9	14.4	Bulgaria
13.4	11.7	11.3	Czechoslovakia	12	12.4	7.6	Former_E._Germany
11.6	13.4	14.8	Hungary	14.3	10.2	16	Poland
13.6	10.7	26.9	Romania	14	9	20.2	Yugoslavia
17.7	10	23	USSR	15.2	9.5	13.1	Byelorussia_SSR
13.4	11.6	13	Ukrainian_SSR	20.7	8.4	25.7	Argentina
46.6	18	111	Bolivia	28.6	7.9	63	Brazil
23.4	5.8	17.1	Chile	27.4	6.1	40	Columbia
32.9	7.4	63	Ecuador	28.3	7.3	56	Guyana
...							

- Want to find groups of similar countries (and how many groups, which countries in each group).
- Tree would be unwieldy with 97 countries.
- More automatic way of finding given number of clusters?

Reading in

```
url <- "http://ritsokiguess.site/datafiles/birthrate.txt"
vital <- read_table(url)
```

```
##
```

```
## -- Column specification -----
```

```
## cols(
```

```
##   birth = col_double(),
```

```
##   death = col_double(),
```

```
##   infant = col_double(),
```

```
##   country = col_character()
```

```
## )
```

The data

```
vital
```

```
## # A tibble: 97 x 4
##   birth death infant country
##   <dbl> <dbl>   <dbl> <chr>
## 1  24.7    5.7   30.8 Albania
## 2  13.4   11.7   11.3 Czechoslovakia
## 3  11.6   13.4   14.8 Hungary
## 4  13.6   10.7   26.9 Romania
## 5  17.7    10    23    USSR
## 6  13.4   11.6    13    Ukrainian_SSR
## 7  46.6    18   111    Bolivia
## 8  23.4    5.8   17.1 Chile
## 9  32.9    7.4    63    Ecuador
## 10 34.8    6.6    42    Paraguay
## # ... with 87 more rows
```

Standardizing

- Infant mortality rate numbers bigger than others, consequence of measurement scale (arbitrary).
- Standardize (numerical) columns of data frame to have mean 0, SD 1, done by scale.

```
vital %>%  
  mutate(across(where(is.numeric), ~scale(.))) -> vital.s
```


Three clusters

Pretend we know 3 clusters is good. Take off the column of countries, and run `kmeans` on the resulting data frame, asking for 3 clusters:

```
vital.s %>% select(-country) %>%  
  kmeans(3) -> vital.km3  
names(vital.km3)
```

```
## [1] "cluster"      "centers"      "totss"  
## [4] "withinss"     "tot.withinss" "betweenss"  
## [7] "size"         "iter"         "ifault"
```

A lot of output, so look at these individually.

What's in the output?

- Cluster sizes:

```
vital.km3$size
```

```
## [1] 40 25 32
```

- Cluster centres:

```
vital.km3$centers
```

```
##          birth      death    infant
## 1 -1.0376994 -0.3289046 -0.90669032
## 2  1.1780071  1.3323130  1.32732200
## 3  0.3768062 -0.6297388  0.09639258
```

- Cluster 2 has lower than average rates on everything; cluster 3 has much higher than average.

Cluster sums of squares and membership

```
vital.km3$withinss
```

```
## [1] 17.21617 28.32560 21.53020
```

Cluster 1 compact relative to others (countries in cluster 1 more similar).

```
vital.km3$cluster
```

```
## [1] 3 1 1 1 1 1 2 1 3 3 1 2 1 1 1 1 1 1 1 1 2 2 1 3 3 3
## [29] 1 3 1 3 3 1 1 3 3 3 2 2 3 3 2 2 3 2 2 2 3 1 1 1 1 1 1
## [57] 3 3 3 3 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 1 2 1 3 3 2 3 1
## [85] 2 2 2 2 3 2 2 2 2 2 3 2 2
```

The cluster membership for each of the 97 countries.

Store countries and clusters to which they belong

```
vital.3 <- tibble(  
  country = vital.s$country,  
  cluster = vital.km3$cluster  
)
```

Next, which countries in which cluster?

Write function to extract them:

```
get_countries <- function(i, d) {  
  d %>% filter(cluster == i) %>% pull(country)  
}
```

Cluster membership: cluster 2

```
get_countries(2, vital.3)
```

##	[1]	"Bolivia"	"Mexico"	"Afghanistan"	"Iran"	"Bangladesh"
##	[6]	"Gabon"	"Ghana"	"Namibia"	"Sierra_Leone"	"Swaziland"
##	[11]	"Uganda"	"Zaire"	"Cambodia"	"Nepal"	"Angola"
##	[16]	"Congo"	"Ethiopia"	"Gambia"	"Malawi"	"Mozambique"
##	[21]	"Nigeria"	"Somalia"	"Sudan"	"Tanzania"	"Zambia"

Cluster 3

```
get_countries(3, vital.3)
```

##	[1]	"Albania"	"Ecuador"	"Paraguay"
##	[4]	"Kuwait"	"Oman"	"Turkey"
##	[7]	"India"	"Mongolia"	"Pakistan"
##	[10]	"Algeria"	"Botswana"	"Egypt"
##	[13]	"Libya"	"Morocco"	"South_Africa"
##	[16]	"Zimbabwe"	"Brazil"	"Columbia"
##	[19]	"Guyana"	"Peru"	"Venezuela"
##	[22]	"Bahrain"	"Iraq"	"Jordan"
##	[25]	"Lebanon"	"Saudi_Arabia"	"Indonesia"
##	[28]	"Malaysia"	"Philippines"	"Vietnam"
##	[31]	"Kenya"	"Tunisia"	

Cluster 1

```
get_countries(1, vital.3)
```

```
## [1] "Czechoslovakia" "Hungary"
## [3] "Romania" "USSR"
## [5] "Ukrainian_SSR" "Chile"
## [7] "Uruguay" "Finland"
## [9] "France" "Greece"
## [11] "Italy" "Norway"
## [13] "Spain" "Switzerland"
## [15] "Austria" "Canada"
## [17] "Israel" "China"
## [19] "Korea" "Singapore"
## [21] "Thailand" "Bulgaria"
## [23] "Former_E._Germany" "Poland"
## [25] "Yugoslavia" "Byelorussia_SSR"
## [27] "Argentina" "Belgium"
```

Problem!

- `kmeans` uses randomization. So result of one run might be different from another run.
- Example: just run again on 3 clusters, table of results:

```
vital.s %>%
  select(-country) %>% kmeans(3) -> vital.km3a
table(
  first = vital.km3$cluster,
  second = vital.km3a$cluster
)
```

```
##      second
## first  1  2  3
##      1 40  0  0
##      2  0 24  1
##      3  4  0 28
```

- Clusters are similar but *not same*.

Solution to this

- `nstart` option on `kmeans` runs that many times, takes best. Should be same every time:

```
vital.s %>%  
  select(-country) %>%  
  kmeans(3, nstart = 20) -> vital.km3b
```

How many clusters?

- Three was just a guess.
- Idea: try a whole bunch of `#clusters` (say 2–20), obtain measure of goodness of fit for each, make plot.
- Appropriate measure is `tot.withinss`.
- Run `kmeans` for each `#clusters`, get `tot.withinss` each time.

Function to get tot.withinss

...for an input number of clusters, taking only numeric columns of input data frame:

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

Note: writing function to be as general as possible, so that we can re-use it later.

Constructing within-cluster SS

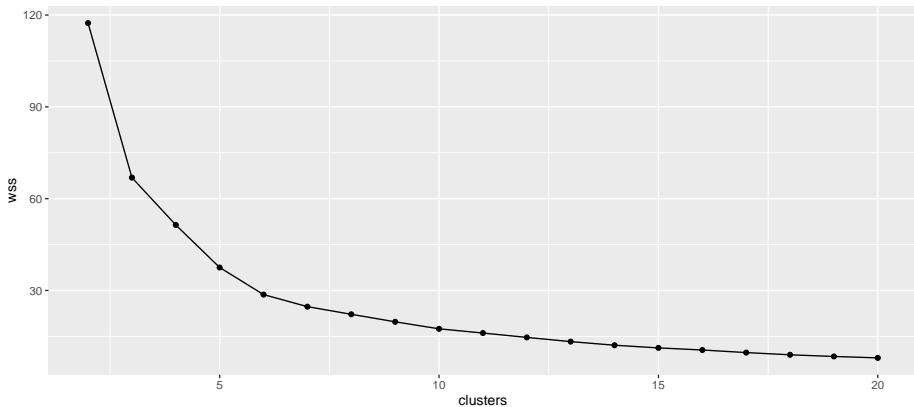
Make a data frame with desired numbers of clusters, and fill it with the total within-group sums of squares. `ss` expects a single number of clusters, not a vector of several, so run `rowwise`:

```
tibble(clusters = 2:20) %>%
  rowwise() %>%
  mutate(wss = ss(clusters, vital.s)) -> ssd
ssd
```

```
## # A tibble: 19 x 2
##   clusters    wss
##   <int>    <dbl>
## 1         2 117.
## 2         3  66.9
## 3         4  51.4
## 4         5  37.5
## 5         6  28.7
```

Scree plot

```
ggplot(ssd, aes(x = clusters, y = wss)) + geom_point() +  
  geom_line()
```



Interpreting scree plot

- Lower wss better.
- But lower for larger #clusters, harder to explain.
- Compromise: low-ish wss and low-ish #clusters.
- Look for “elbow” in plot.
- Idea: this is where wss decreases fast then slow.
- On our plot, small elbow at 6 clusters. Try this many clusters.

Six clusters, using nstart

```
vital.s %>%
  select(-country) %>%
  kmeans(6, nstart = 20) -> vital.km6
vital.km6$size
```

```
## [1] 17 24 13 20 13 10
```

```
vital.km6$centers
```

```
##          birth          death          infant
## 1  1.2049466  0.6972333  1.0165097
## 2  0.4160993 -0.5169988  0.2648754
## 3 -1.1458296  0.2636810 -0.9301055
## 4 -1.1331101 -0.4617719 -0.9428918
## 5 -0.3548334 -1.1812663 -0.7096686
## 6  1.1700347  2.1719052  1.6537224
```

Make a data frame of countries and clusters

```
vital.6 <- tibble(
  country = vital.s$country,
  cluster = vital.km6$cluster
)
vital.6 %>% sample_n(10)
```

```
## # A tibble: 10 x 2
##   country      cluster
##   <chr>         <int>
## 1 Swaziland         1
## 2 Switzerland       4
## 3 Philippines       2
## 4 Guyana            2
## 5 Finland           4
## 6 Vietnam           2
## 7 Paraguay          2
## 8 ...              ...
```


Cluster 1

Below-average death rate, though other rates a little higher than average:

```
get_countries(1, vital.6)
```

```
## [1] "Iran"           "Bangladesh" "Botswana"    "Gabon"
## [5] "Ghana"          "Namibia"     "Swaziland"   "Uganda"
## [9] "Zaire"          "Cambodia"    "Nepal"       "Congo"
## [13] "Kenya"          "Nigeria"     "Sudan"       "Tanzania"
## [17] "Zambia"
```

Cluster 2

High on everything:

```
get_countries(2, vital.6)
```

##	[1]	"Ecuador"	"Paraguay"	"Oman"
##	[4]	"Turkey"	"India"	"Mongolia"
##	[7]	"Pakistan"	"Algeria"	"Egypt"
##	[10]	"Libya"	"Morocco"	"South_Africa"
##	[13]	"Zimbabwe"	"Brazil"	"Guyana"
##	[16]	"Peru"	"Iraq"	"Jordan"
##	[19]	"Lebanon"	"Saudi_Arabia"	"Indonesia"
##	[22]	"Philippines"	"Vietnam"	"Tunisia"

Cluster 3

Low on everything, though death rate close to average:

```
get_countries(3, vital.6)
```

```
## [1] "Czechoslovakia" "Hungary"  
## [3] "Romania"        "Ukrainian_SSR"  
## [5] "Norway"         "Korea"  
## [7] "Bulgaria"       "Former_E._Germany"  
## [9] "Belgium"        "Denmark"  
## [11] "Germany"        "Sweden"  
## [13] "U.K."
```

Cluster 4

Low on everything, especially death rate:

```
get_countries(4, vital.6)
```

```
## [1] "USSR" "Uruguay"
## [3] "Finland" "France"
## [5] "Greece" "Italy"
## [7] "Spain" "Switzerland"
## [9] "Austria" "Canada"
## [11] "Poland" "Yugoslavia"
## [13] "Byelorussia_SSR" "Argentina"
## [15] "Ireland" "Netherlands"
## [17] "Portugal" "Japan"
## [19] "U.S.A." "Hong_Kong"
```

Cluster 5

Higher than average on everything, though not the highest:

```
get_countries(5, vital.6)
```

```
## [1] "Albania"           "Chile"
## [3] "Israel"            "Kuwait"
## [5] "China"             "Singapore"
## [7] "Thailand"           "Columbia"
## [9] "Venezuela"         "Bahrain"
## [11] "United_Arab_Emirates" "Malaysia"
## [13] "Sri_Lanka"
```

Cluster 6

Very high death rate, just below average on all else:

```
get_countries(6, vital.6)
```

```
## [1] "Bolivia"      "Mexico"      "Afghanistan"  
## [4] "Sierra_Leone" "Angola"      "Ethiopia"  
## [7] "Gambia"       "Malawi"      "Mozambique"  
## [10] "Somalia"
```

Comparing our 3 and 6-cluster solutions

```
table(three = vital.km3$cluster, six = vital.km6$cluster)
```

```
##          six
## three  1  2  3  4  5  6
##      1  0  0 13 20  7  0
##      2 15  0  0  0  0 10
##      3  2 24  0  0  6  0
```

Compared to 3-cluster solution:

- most of (old) cluster 3 gone to (new) cluster 2
- cluster 1 split into clusters 3 and 4 (two types of “richer” countries)
- cluster 3 split into clusters 2 and 5 (two types of “poor” countries, divided by death rate).

Getting a picture from kmeans

- Use discriminant analysis on clusters found, treating them as “known” groups.

Discriminant analysis

- So what makes the groups different?
- Uses package MASS (loaded):

```
vital.lda <- lda(vital.km6$cluster ~ birth + death + infant,
                 data = vital.s)
vital.lda$svd
```

```
## [1] 17.407851  8.743023  1.000331
```

```
vital.lda$scaling
```

```
##           LD1           LD2           LD3
## birth  -2.088306  1.6066337 -1.7791031
## death  -1.359398 -2.5075513 -0.6581161
## infant -1.184993  0.4780262  2.2687506
```

- LD1 is some of everything (high=poor, low=rich).
- LD2 mainly death rate, high or low.

A data frame to make plot from

- Get predictions first:

```
vital.pred <- predict(vital.lda)
d <- data.frame(
  country = vital.s$country,
  cluster = vital.km6$cluster,
  vital.pred$x
)
glimpse(d)
```

```
## Rows: 97
## Columns: 5
## $ country <chr> "Albania", "Czechoslovakia", "Hu~
## $ cluster <int> 5, 3, 3, 3, 4, 3, 6, 5, 2, 2, 4,~
## $ LD1      <dbl> 2.8215814, 3.3109528, 3.0010047,~
## $ LD2      <dbl> 1.983429, -2.796716, -3.891051, ~
## $ LD3      <dbl> 0.13334944, -0.19415639, -0.0258~
```

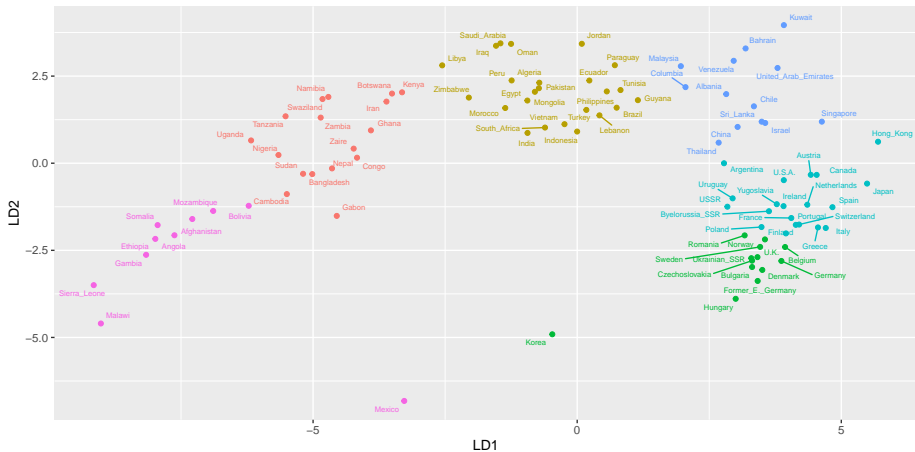
What's in there; making a plot

- d contains country names, cluster memberships and discriminant scores.
- Plot LD1 against LD2, colouring points by cluster and labelling by country:

```
g <- ggplot(d, aes(  
  x = LD1, y = LD2, colour = factor(cluster),  
  label = country  
)) + geom_point() +  
  geom_text_repel(size = 2, max.overlaps = Inf) + guides(colour = "none")
```

The plot

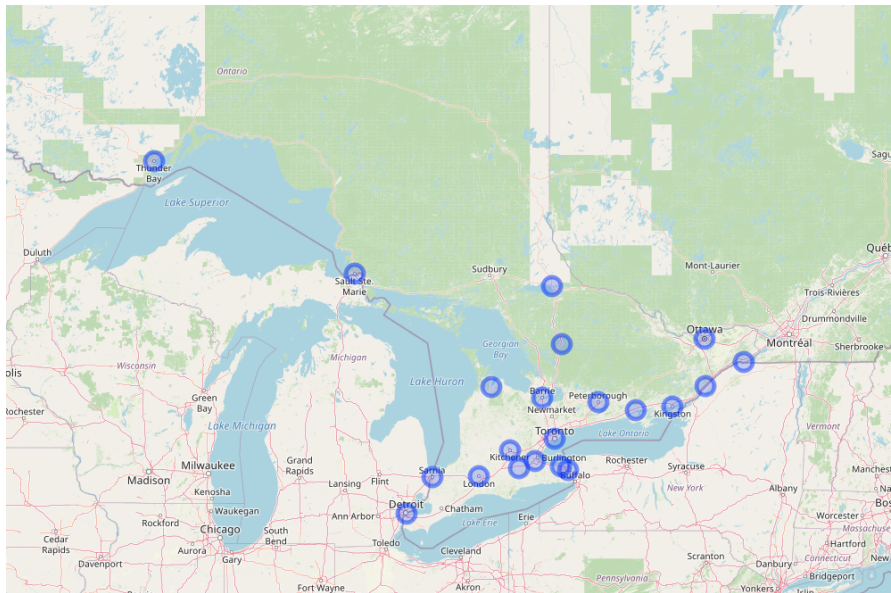
g



Final example: a hockey league

- An Ontario hockey league has teams in 21 cities. How can we arrange those teams into 4 geographical divisions?
- Distance data in spreadsheet.
- Take out spaces in team names.
- Save as “text/csv”.
- Distances, so back to `hclust`.

A map

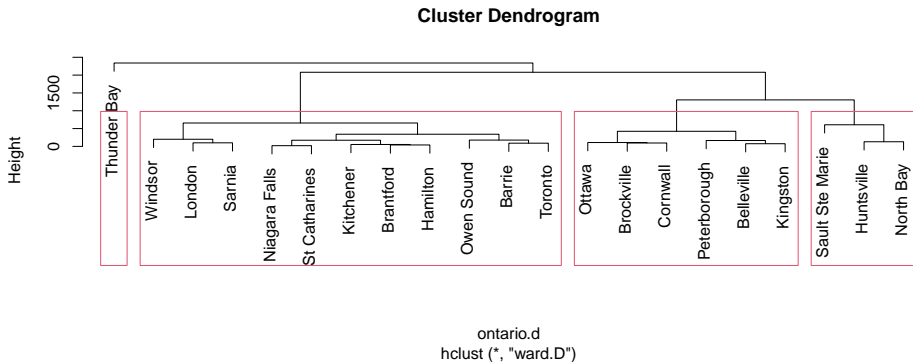


Attempt 1

```
my_url <-  
  "http://ritsokiguess.site/datafiles/ontario-road-distances.csv"  
ontario <- read_csv(my_url)  
ontario.d <- ontario %>% select(-1) %>% as.dist()  
ontario.hc <- hclust(ontario.d, method = "ward.D")
```

Plot, with 4 clusters

```
plot(ontario.hc)  
rect.hclust(ontario.hc, 4)
```

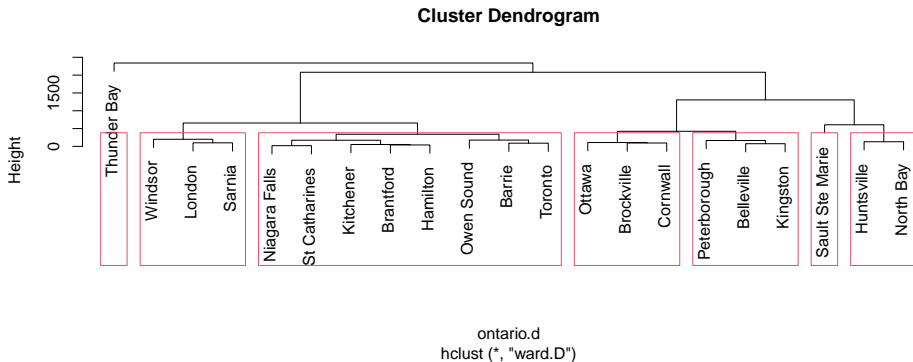


Comments

- Can't have divisions of 1 team!
- "Southern" divisions way too big!
- Try splitting into more. I found 7 to be good:

Seven clusters

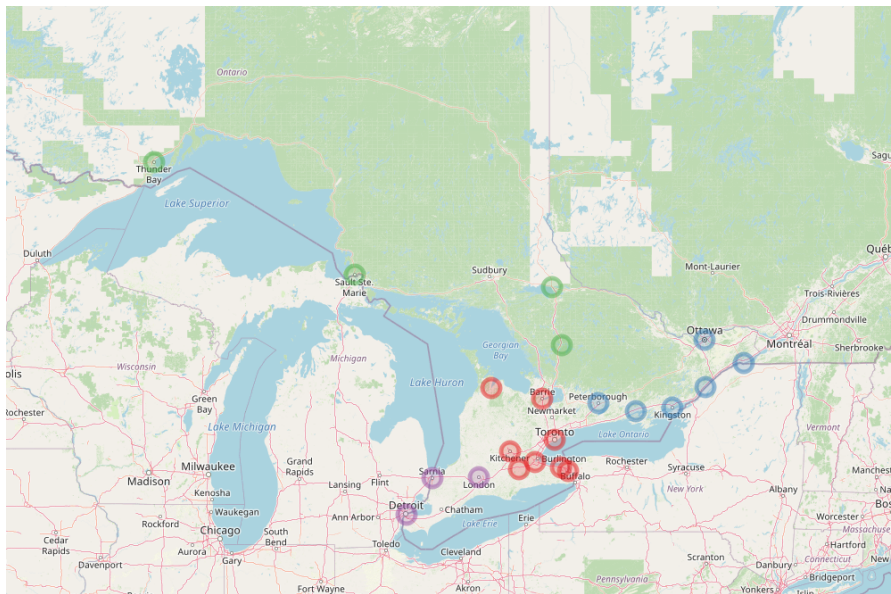
```
plot(ontario.hc)
rect.hclust(ontario.hc, 7)
```



Divisions now

- I want to put Huntsville and North Bay together with northern teams.
- I'll put the Eastern teams together. Gives:
- North: Sault Ste Marie, Sudbury, Huntsville, North Bay
- East: Brockville, Cornwall, Ottawa, Peterborough, Belleville, Kingston
- West: Windsor, London, Sarnia
- Central: Owen Sound, Barrie, Toronto, Niagara Falls, St Catharines, Brantford, Hamilton, Kitchener
- Getting them same size beyond us!

Another map



Section 11

Principal components

Principal Components

- Have measurements on (possibly large) number of variables on some individuals.
- Question: can we describe data using fewer variables (because original variables correlated in some way)?
- Look for direction (linear combination of original variables) in which values *most spread out*. This is *first principal component*.
- Second principal component then direction uncorrelated with this in which values then most spread out. And so on.

Principal components

- See whether small number of principal components captures most of variation in data.
- Might try to interpret principal components.
- If 2 components good, can make plot of data.
- (Like discriminant analysis, but no groups.)
- “What are important ways that these data vary?”

Packages

You might not have installed the first of these. See over for instructions.

```
library(ggbiplot) # see over  
library(tidyverse)  
library(ggrepel)
```


Installing ggbiplot

- ggbiplot not on CRAN, so usual `install.packages` will not work. This is same procedure you used for `smmr` in C32:
- Install package `devtools` first (once):

```
install.packages("devtools")
```

- Then install `ggbiplot` (once):

```
library(devtools)  
install_github("vqv/ggbiplot")
```

Small example: 2 test scores for 8 people

```
my_url <- "http://ritsokiguess.site/datafiles/test12.txt"
test12 <- read_table2(my_url)
test12
```

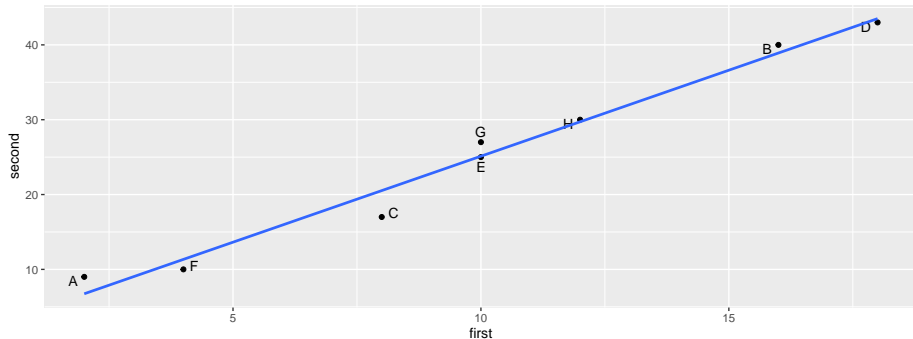
```
## # A tibble: 8 x 3
##   first second id
##   <dbl>  <dbl> <chr>
## 1      2      9 A
## 2     16     40 B
## 3      8     17 C
## 4     18     43 D
## 5     10     25 E
## 6      4     10 F
## 7     10     27 G
## 8     12     30 H
```

```
g <- ggplot(test12, aes(x = first, y = second, label = id)) +
  geom_point() + geom_text_repel()
```

The plot

```
g + geom_smooth(method = "lm", se = F)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Principal component analysis

- Grab just the numeric columns:

```
test12 %>% select(where(is.numeric)) -> test12_numbers
```

- Strongly correlated, so data nearly 1-dimensional:

```
cor(test12_numbers)
```

```
##           first    second
## first  1.000000  0.989078
## second 0.989078  1.000000
```

Finding principal components

- Make a score summarizing this one dimension. Like this:

```
test12.pc <- princomp(test12_numbers, cor = T)
summary(test12.pc)
```

```
## Importance of components:
```

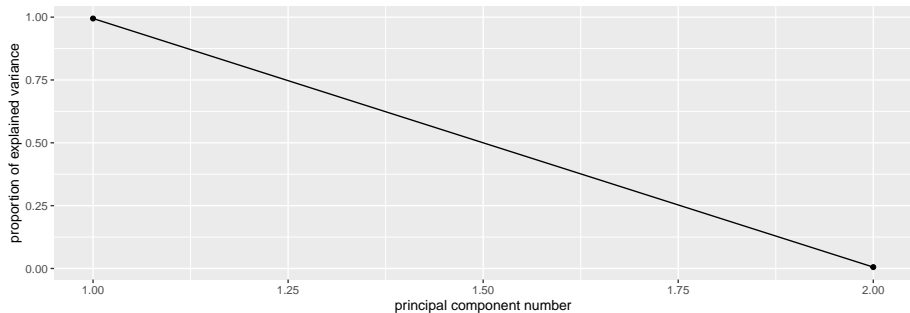
##	Comp.1	Comp.2
## Standard deviation	1.410347	0.104508582
## Proportion of Variance	0.994539	0.005461022
## Cumulative Proportion	0.994539	1.000000000

Comments

- “Standard deviation” shows relative importance of components (as for LDs in discriminant analysis)
- Here, first one explains almost all (99.4%) of variability.
- That is, look only at first component and ignore second.
- cor=T standardizes all variables first. Usually wanted, because variables measured on different scales. (Only omit if variables measured on same scale and expect similar variability.)

Scree plot

```
ggscreeplot(test12.pc)
```



Imagine scree plot continues at zero, so 2 components is a *big* elbow (take one component).

Component loadings

explain how each principal component depends on (standardized) original variables (test scores):

```
test12.pc$loadings
```

```
##
## Loadings:
##          Comp.1 Comp.2
## first    0.707  0.707
## second   0.707 -0.707
##
##          Comp.1 Comp.2
## SS loadings      1.0    1.0
## Proportion Var   0.5    0.5
## Cumulative Var   0.5    1.0
```

First component basically sum of (standardized) test scores. That is, person tends to score similarly on two tests, and a composite score would summarize performance.

Component scores

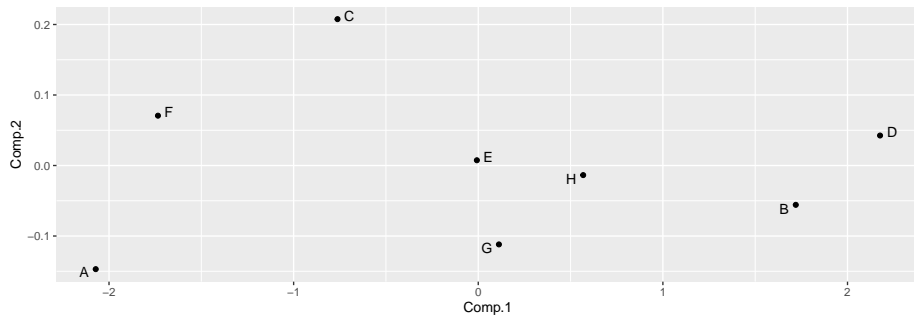
```
d <- data.frame(test12, test12.pc$scores)
d
```

```
## # A tibble: 8 x 5
##   first second id      Comp.1   Comp.2
##   <dbl>  <dbl> <chr>    <dbl>    <dbl>
## 1      2      9 A     -2.07    -0.147
## 2     16     40 B      1.72    -0.0558
## 3      8     17 C     -0.762    0.208
## 4     18     43 D      2.18     0.0425
## 5     10     25 E     -0.00746  0.00746
## 6      4     10 F     -1.73     0.0707
## 7     10     27 G      0.112   -0.112
## 8     12     30 H      0.568   -0.0136
```

- Person A is a low scorer, very negative comp.1 score.
- Person D is high scorer, high positive comp.1 score.
- Person F average scorer, near-zero comp.1 score

Plot of scores

```
ggplot(d, aes(x = Comp.1, y = Comp.2, label = id)) +  
  geom_point() + geom_text_repel()
```



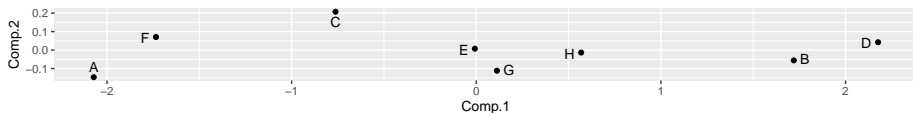
Comments

- Vertical scale exaggerates importance of comp.2.
- Fix up to get axes on same scale:

```
g <- ggplot(d, aes(x = Comp.1, y = Comp.2, label = id)) +  
  geom_point() + geom_text_repel() +  
  coord_fixed()
```

- Shows how exam scores really spread out along one dimension:

gg

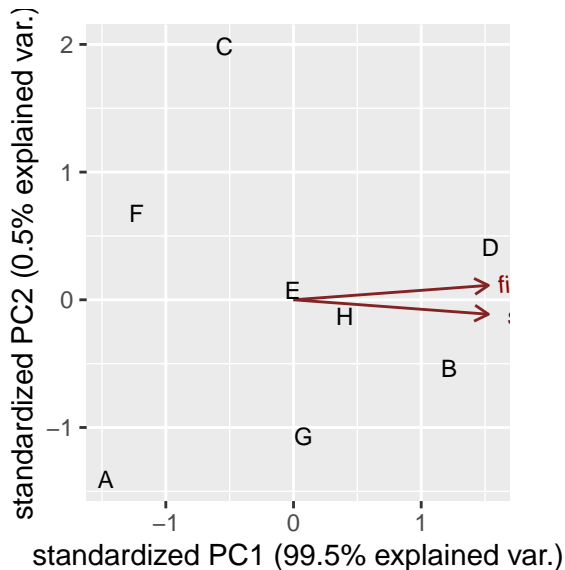


The biplot

- Plotting variables and individuals on one plot.
- Shows how components and original variables related.
- Shows how individuals score on each component, and therefore suggests how they score on each variable.
- Add `labels` option to identify individuals:

```
g <- ggbiplot(test12.pc, labels = test12$id)
```

The biplot



Comments

- Variables point almost same direction (left). Thus very negative value on comp.1 goes with high scores on both tests, and test scores highly correlated.
- Position of individuals on plot according to scores on principal components, implies values on original variables. Eg.:
- D very negative on comp.1, high scorer on both tests.
- A and F very positive on comp.1, poor scorers on both tests.
- C positive on comp.2, high score on first test relative to second.
- A negative on comp.2, high score on second test relative to first.

Places rated

Every year, a new edition of the Places Rated Almanac is produced. This rates a large number (in our data 329) of American cities on a number of different criteria, to help people find the ideal place for them to live (based on what are important criteria for them).

The data for one year are in <http://ritsokiguess.site/datafiles/places.txt>.
The data columns are aligned but the column headings are not.

The criteria

There are nine of them:

- climate: a higher value means that the weather is better
- housing: a higher value means that there is more good housing or a greater choice of different types of housing
- health: higher means better healthcare facilities
- crime: higher means more crime (bad)
- trans: higher means better transportation (this being the US, probably more roads)
- educate: higher means better educational facilities, schools, colleges etc.
- arts: higher means better access to the arts (theatre, music etc)
- recreate: higher means better access to recreational facilities
- econ: higher means a better economy (more jobs, spending power etc)

Each city also has a numbered id.

Read in the data

```
my_url <- "http://ritsokiguess.site/datafiles/places.txt"
places0 <- read_table2(my_url)
```

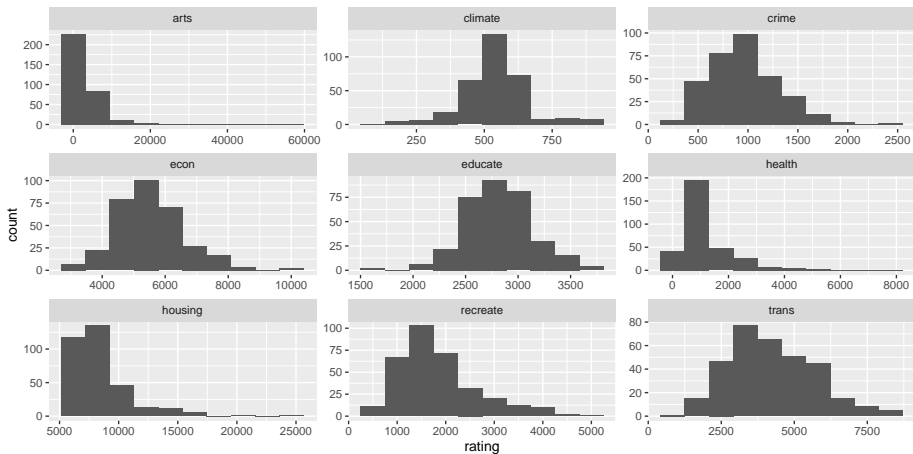
```
##
## -- Column specification -----
## cols(
##   climate = col_double(),
##   housing = col_double(),
##   health = col_double(),
##   crime = col_double(),
##   trans = col_double(),
##   educate = col_double(),
##   arts = col_double(),
##   recreate = col_double(),
##   econ = col_double(),
##   id = col_double()
## )
```

Look at distributions of everything

```
places0 %>%  
  pivot_longer(-id, names_to = "criterion",  
               values_to = "rating") %>%  
  ggplot(aes(x = rating)) + geom_histogram(bins = 10) +  
  facet_wrap(~criterion, scales = "free") -> g
```

The histograms

9



Transformations

- Several of these variables have long right tails
- Take logs of everything but id:

```
places0 %>%  
  mutate(across(-id, ~log(.))) -> places
```

Just the numerical columns

- get rid of the id column

```
places %>% select(-id) -> places_numeric
```

Principal components

```
places.1 <- princomp(places_numeric, cor = TRUE)
summary(places.1)
```

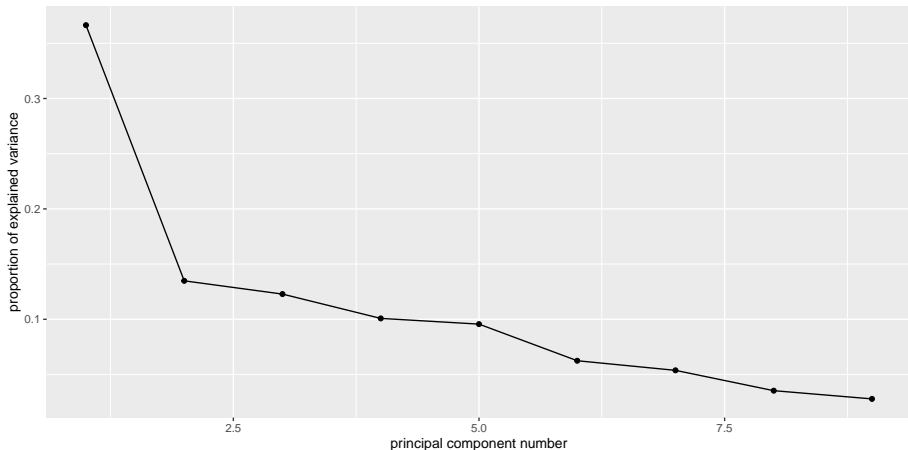
```
## Importance of components:
```

	Comp.1	Comp.2	Comp.3	Comp.4	Comp.5
## Standard deviation	1.8159827	1.1016178	1.0514418	0.9525124	0.92770076
## Proportion of Variance	0.3664214	0.1348402	0.1228367	0.1008089	0.09562541
## Cumulative Proportion	0.3664214	0.5012617	0.6240983	0.7249072	0.82053259

	Comp.6	Comp.7	Comp.8	Comp.9
## Standard deviation	0.74979050	0.69557215	0.56397886	0.50112689
## Proportion of Variance	0.06246509	0.05375785	0.03534135	0.02790313
## Cumulative Proportion	0.88299767	0.93675552	0.97209687	1.00000000

scree plot

```
ggscreeplot(places.1)
```



- big elbow at 2 (1 component); smaller elbow at 6 (5) and maybe 4 (3).

What is in each component?

```
places.1$loadings
```

```
##
## Loadings:
##          Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7
## climate    0.158          0.800  0.377          0.217  0.151
## housing    0.384  0.139          0.197 -0.580          0.275
## health     0.410 -0.372          0.113          -0.535 -
0.135
## crime      0.259  0.474  0.128          0.692 -0.140 -
0.110
## trans      0.375 -0.141 -0.141 -0.430  0.191  0.324  0.679
## educate    0.274 -0.452 -0.241  0.457  0.225  0.527 -
0.262
## arts       0.474 -0.104          -0.147          -0.321 -
0.120
##          0.259  0.200  0.404  0.200  0.204
```


Assessing the components

Look at component loadings and make a call about “large” (in absolute value) vs “small”. Large loadings are a part of the component and small ones are not. Thus, if we use 0.4 as cutoff:

- component #1 depends on health and arts
- #2 depends on economy and crime, and negatively on education.
- #3 depends on climate, and negatively on economy.
- #4 depends on education and the economy, negatively on transportation and recreation opportunities.
- #5 depends on crime and negatively on housing.

Comments

- The use of 0.4 is arbitrary; you can use whatever you like. It can be difficult to decide whether a variable is “in” or “out”.
- The large (far from zero) loadings indicate what distinguishes the cities as places to live, for example:
 - places that are rated high for health also tend to be rated high for arts
 - places that have a good economy tend to have a bad climate (and vice versa)
 - places that have a lot of crime tend to have bad housing.

Making a plot 1/3

How can we make a visual showing the cities? We need a “score” for each city on each component, and we need to identify the cities (we have a numerical id in the original dataset):

```
cbind(city_id = places$id, places.1$scores) %>%  
  as_tibble() -> places_score
```

The `as_tibble` is needed at the end because the scores are a matrix.

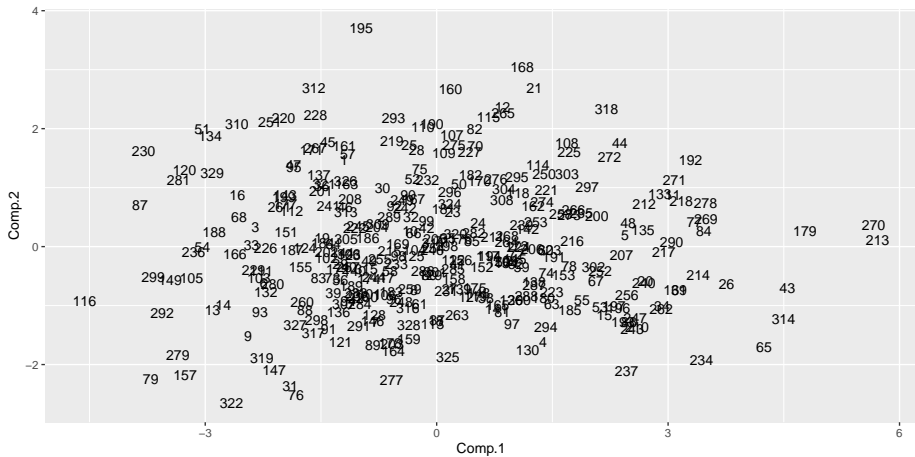
Making a plot 2/3

- Plot the first two scores against each other, labelling each point by the id of the city it belongs to:

```
ggplot(places_score, aes(x = Comp.1, y = Comp.2,  
                        label = city_id)) +  
  geom_text() -> g
```

Making a plot 3/3

၁၁၀



Comments

- Cities 213 and 270 are high on component 1, and city 116 is low. City 195 is high on component 2, and city 322 is low.
- This suggests that cities 213 and 270 are high on health and arts, and city 116 is low. City 195 should be high on economy and crime and low on education, and city 322 should be the other way around.

Checking this 1/2

- The obvious way of checking this is in two steps: first, work out what high or low means for each variable:

```
summary(places)
```

```
##      climate      housing      health      crime      trans      educate
## Min.   :4.654   Min.    : 8.548   Min.    :3.761   Min.    :5.730   Min.    :7.043   Min.    :7.439
## 1st Qu.:6.174   1st Qu.: 8.819   1st Qu.:6.368   1st Qu.:6.561   1st Qu.:8.052   1st Qu.:7.871
## Median :6.295   Median : 8.972   Median :6.725   Median :6.853   Median :8.314   Median :7.935
## Mean   :6.260   Mean    : 8.997   Mean    :6.805   Mean    :6.796   Mean    :8.283   Mean    :7.936
## 3rd Qu.:6.384   3rd Qu.: 9.107   3rd Qu.:7.276   3rd Qu.:7.053   3rd Qu.:8.557   3rd Qu.:8.010
## Max.   :6.813   Max.    :10.071   Max.    :8.968   Max.    :7.823   Max.    :9.062   Max.    :8.238
##
##      arts      recreate      econ      id
## Min.   : 3.951   Min.    :5.704   Min.    :8.021   Min.    : 1
## 1st Qu.: 6.657   1st Qu.:7.182   1st Qu.:8.485   1st Qu.: 83
## Median : 7.534   Median :7.421   Median :8.591   Median :165
## Mean   : 7.383   Mean    :7.429   Mean    :8.598   Mean    :165
## 3rd Qu.: 8.254   3rd Qu.:7.685   3rd Qu.:8.718   3rd Qu.:247
## Max.   :10.946   Max.    :8.476   Max.    :9.208   Max.    :329
```

Checking this 2/2

- and then find the values on the variables of interest for our cities of interest, and see where they sit on here.
- Cities 270, 213, and 116 were extreme on component 1, which depended mainly on health and arts:

```
places %>% select(id, health, arts) %>%
  filter(id %in% c(270, 213, 166))
```

```
## # A tibble: 3 x 3
##       id health  arts
##   <dbl> <dbl> <dbl>
## 1   166    6.14  5.01
## 2   213    8.97 10.9
## 3   270    8.22  9.56
```

City 166 is near or below Q1 on both variables. City 213 is the highest of all on both health and arts, while city 270 is well above Q3 on both.

Checking component 2

- Component 2 depended positively on economy and crime and negatively on education. City 195 was high and 322 was low:

```
places %>% select(id, econ, crime, educate) %>%
  filter(id %in% c(195, 322))
```

```
## # A tibble: 2 x 4
##       id  econ crime educate
##   <dbl> <dbl> <dbl>   <dbl>
## 1   195  9.21  7.06    7.79
## 2   322  8.10  6.14    7.97
```

- City 195 is the highest on economy, just above Q3 on crime, and below Q1 on education. City 322 should be the other way around: nearly the lowest on economy, below Q1 on crime, and between the median and Q3 on education. This is as we'd expect.

A better way: percentile ranks

- It is a lot of work to find the value of each city on each variable in the data summary.
- A better way is to work out the percentile ranks of each city on each variable and then look at those:

```
places %>%  
  mutate(across(-id, ~percent_rank(.))) -> places_pr
```

Look up cities and variables again

```
places_pr %>% select(id, health, arts) %>%  
  filter(id %in% c(270, 213, 166))
```

```
## # A tibble: 3 x 3  
##       id health  arts  
##   <dbl> <dbl> <dbl>  
## 1   166  0.152 0.0488  
## 2   213    1    1  
## 3   270  0.970 0.982
```

This shows that city 270 was also really high on these two variables: in the 97th percentile for health and the 98th for arts.

Component 2

- What about the extreme cities on component 2?

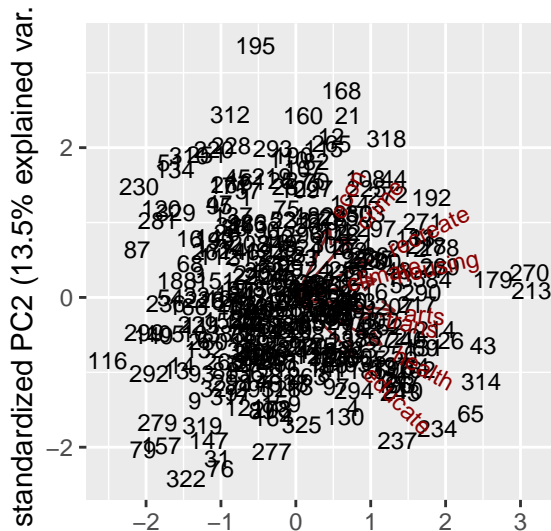
```
places_pr %>% select(id, econ, crime, educate) %>%
  filter(id %in% c(195, 322))
```

```
## # A tibble: 2 x 4
##       id      econ  crime educate
##   <dbl>   <dbl>   <dbl>   <dbl>
## 1   195 1      0.762  0.0884
## 2   322 0.00610 0.0732  0.631
```

- City 322 was really low on economy and crime, but only just above average on education. City 195 was the highest on economy and really low on education, but only somewhat high on crime (76th percentile).
- This, as you see, is much easier once you have set it up.

The biplot

```
ggbiplot(places.1, labels = places$id)
```



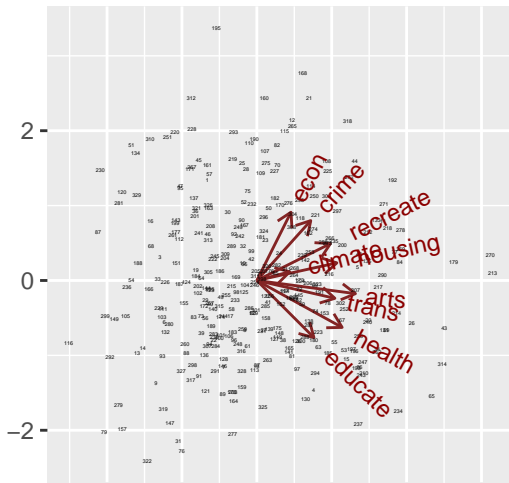
Comments

- This is hard to read!
- There are a lot of cities that overshadow the red arrows for the variables.
- reduce the size of the city labels

Biplot, attempt 2

```
ggbiplot(places.1, labels = places$id,
          labels.size = 0.8)
```

andardized PC2 (13.5% explained var.



All the slides

Comments on attempt #2

- Now at least can see the variables
- All of them point somewhat right (all belong partly to component 1)
- Some of them (economy, crime, education) point up/down, belong to component 2 as well.
- In this case, cannot really see both observations (cities) and variables (criteria) together, which defeats the purpose of the biplot.
- Have to try it and see.

Principal components from correlation matrix

Create data file like this:

```
1          0.9705 -0.9600
0.9705     1          -0.9980
-0.9600  -0.9980     1
```

and read in like this:

```
my_url <- "http://ritsokiguess.site/datafiles/cov.txt"
mat <- read_table(my_url, col_names = F)
mat
```

```
## # A tibble: 3 x 3
##       X1      X2      X3
##   <dbl> <dbl> <dbl>
## 1  1      0.970 -0.96
## 2  0.970  1      -0.998
## 3 -0.96  -0.998  1
```

Pre-processing

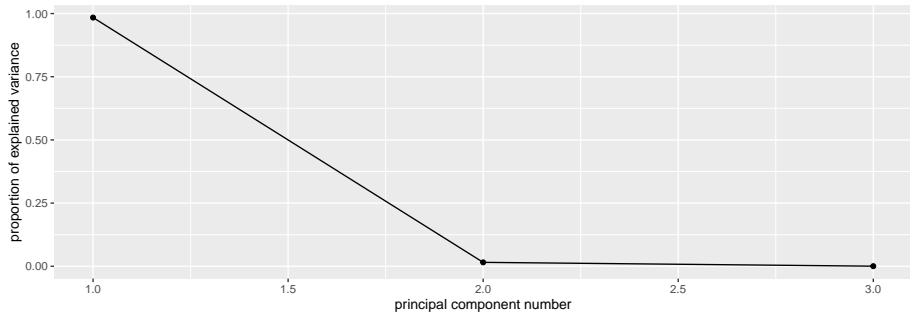
A little pre-processing required:

- Turn into matrix (from data frame)
- Feed into `princomp` as `covmat=`

```
mat.pc <- mat %>%  
  as.matrix() %>%  
  princomp(covmat = .)
```

Scree plot: one component fine

```
ggscreeplot(mat.pc)
```



Component loadings

Compare correlation matrix:

```
mat
```

```
## # A tibble: 3 x 3
##       X1      X2      X3
##   <dbl> <dbl> <dbl>
## 1  1      0.970 -0.96
## 2  0.970  1      -0.998
## 3 -0.96  -0.998  1
```

with component loadings

```
mat.pc$loadings
```

```
##
## Loadings:
##      Comp.1 Comp.2 Comp.3
## X1  0.573  0.812  0.112
## X2  0.581 -0.306 -0.755
## X3 -0.578  0.498 -0.646
##
##
##      Comp.1 Comp.2 Comp.3
## SS loadings  1.000  1.000  1.000
```

Comments

- When X_1 large, X_2 also large, X_3 small.
 - Then comp.1 *positive*.
- When X_1 small, X_2 small, X_3 large.
 - Then comp.1 *negative*.

No scores

- With correlation matrix rather than data, no component scores
 - So no principal component plot
 - and no biplot.

Section 12

Factor analysis

Principal components and factor analysis

- Principal components:
 - Purely mathematical.
 - Find eigenvalues, eigenvectors of correlation matrix.
 - No testing whether observed components reproducible, or even probability model behind it.
- Factor analysis:
 - some way towards fixing this (get test of appropriateness)
 - In factor analysis, each variable modelled as: “common factor” (eg. verbal ability) and “specific factor” (left over).
 - Choose the common factors to “best” reproduce pattern seen in correlation matrix.
 - Iterative procedure, different answer from principal components.

Packages

```
library(ggbiplot)  
library(tidyverse)
```

Example

- 145 children given 5 tests, called PARA, SENT, WORD, ADD and DOTS. 3 linguistic tasks (paragraph comprehension, sentence completion and word meaning), 2 mathematical ones (addition and counting dots).
- Correlation matrix of scores on the tests:

para	1	0.722	0.714	0.203	0.095
sent	0.722	1	0.685	0.246	0.181
word	0.714	0.685	1	0.170	0.113
add	0.203	0.246	0.170	1	0.585
dots	0.095	0.181	0.113	0.585	1

- Is there small number of underlying “constructs” (unobservable) that explains this pattern of correlations?

To start: principal components

Using correlation matrix. Read that first:

```
my_url <- "http://ritsokiguess.site/datafiles/rex2.txt"
kids <- read_delim(my_url, " ")
kids
```

```
## # A tibble: 5 x 6
##   test   para sent word  add dots
##   <chr> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 para   1      0.722 0.714 0.203 0.095
## 2 sent  0.722  1      0.685 0.246 0.181
## 3 word  0.714 0.685  1      0.17  0.113
## 4 add   0.203 0.246 0.17   1      0.585
## 5 dots  0.095 0.181 0.113 0.585  1
```

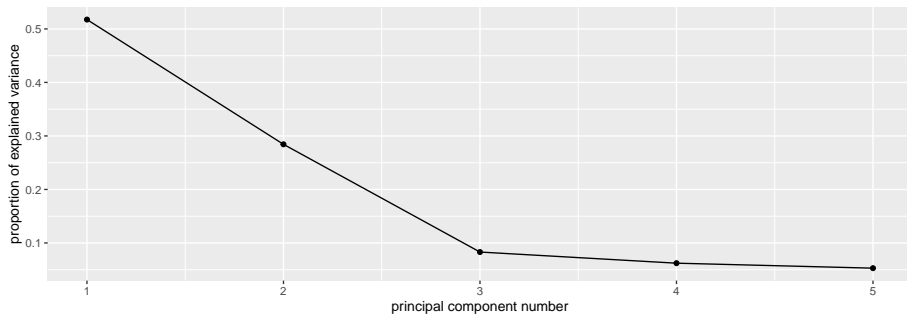
Principal components on correlation matrix

```
kids %>%  
  select(where(is.numeric)) %>%  
  as.matrix() -> m  
kids.0 <- princomp(covmat = m)
```

I used kids.0 here since I want kids.1 and kids.2 later.

Scree plot

```
ggscreeplot(kids.0)
```



Principal component results

- Need 2 components. Loadings:

```
kids.0$loadings
```

```
##
## Loadings:
##      Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
## para  0.534  0.245  0.114          0.795
## sent  0.542  0.164          0.660 -0.489
## word  0.523  0.247 -0.144 -0.738 -0.316
## add   0.297 -0.627  0.707
## dots  0.241 -0.678 -0.680          0.143
##
##              Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
## SS loadings      1.0   1.0   1.0   1.0   1.0
## Proportion Var   0.2   0.2   0.2   0.2   0.2
## Cumulative Var   0.2   0.4   0.6   0.8   1.0
```

Comments

- First component has a bit of everything, though especially the first three tests.
- Second component rather more clearly add and dots.
- No scores, plots since no actual data.
- See how factor analysis compares on these data.

Factor analysis

- Specify number of factors first, get solution with exactly that many factors.
- Includes hypothesis test, need to specify how many children wrote the tests.
- Works from correlation matrix via `covmat` or actual data, like `princomp`.
- Introduces extra feature, *rotation*, to make interpretation of loadings (factor-variable relation) easier.

Factor analysis for the kids data

- Create “covariance list” to include number of children who wrote the tests.
- Feed this into `factanal`, specifying how many factors (2).
- Start with the matrix we made before.

```
m
```

```
##      para  sent  word   add  dots
## [1,] 1.000 0.722 0.714 0.203 0.095
## [2,] 0.722 1.000 0.685 0.246 0.181
## [3,] 0.714 0.685 1.000 0.170 0.113
## [4,] 0.203 0.246 0.170 1.000 0.585
## [5,] 0.095 0.181 0.113 0.585 1.000
```

```
ml <- list(cov = m, n.obs = 145)
kids.2 <- factanal(factors = 2, covmat = ml)
```

Uniquenesses

```
kids.2$uniquenesses
```

```
##          para          sent          word          add          dots
## 0.2424457 0.2997349 0.3272312 0.5743568 0.1554076
```

- Uniquenesses say how “unique” a variable is (size of specific factor). Small uniqueness means that the variable is summarized by a factor (good).
- Very large uniquenesses are bad; add’s uniqueness is largest but not large enough to be worried about.
- Also see “communality” for this idea, where *large* is good and *small* is bad.

Loadings

```
kids.2$loadings
```

```
##
## Loadings:
##      Factor1 Factor2
## [1,] 0.867
## [2,] 0.820  0.166
## [3,] 0.816
## [4,] 0.167  0.631
## [5,]      0.918
##
##      Factor1 Factor2
## SS loadings    2.119  1.282
## Proportion Var  0.424  0.256
## Cumulative Var  0.424  0.680
```

- Loadings show how each factor depends on variables. Blanks indicate “small”, less than 0.1.

Comments

- Factor 1 clearly the “linguistic” tasks, factor 2 clearly the “mathematical” ones.
- Two factors together explain 68% of variability (like regression R-squared).
- Which variables belong to which factor is *much* clearer than with principal components.

Are 2 factors enough?

```
kids.2$STATISTIC
```

```
## objective
```

```
## 0.5810578
```

```
kids.2$dof
```

```
## [1] 1
```

```
kids.2$PVAL
```

```
## objective
```

```
## 0.445898
```

P-value not small, so 2 factors OK.

1 factor

```
kids.1 <- factanal(factors = 1, covmat = ml)  
kids.1$STATISTIC
```

```
## objective  
## 58.16534
```

```
kids.1$dof
```

```
## [1] 5
```

```
kids.1$PVAL
```

```
## objective  
## 2.907856e-11
```

1 factor rejected (P-value small). Definitely need more than 1.

Places rated, again

- Read data, transform, rerun principal components, get biplot:

```
my_url <- "http://ritsokiguess.site/datafiles/places.txt"
places0 <- read_table2(my_url)
places0 %>%
  mutate(across(-id, ~log(.))) -> places
places %>% select(-id) -> places_numeric
places.1 <- princomp(places_numeric, cor = TRUE)
g <- ggbiplot(places.1, labels = places$id,
              labels.size = 0.8)
```

- This is all exactly as for principal components (nothing new here).

The biplot

89

standardized PC2 (13.5% explained var.)



Comments

- Most of the criteria are part of components 1 *and* 2.
- If we can rotate the arrows counterclockwise:
 - economy and crime would point straight up
 - part of component 2 only
 - health and education would point to the right
 - part of component 1 only
- would be easier to see which variables belong to which component.
- Factor analysis includes a rotation to help with interpretation.

Factor analysis

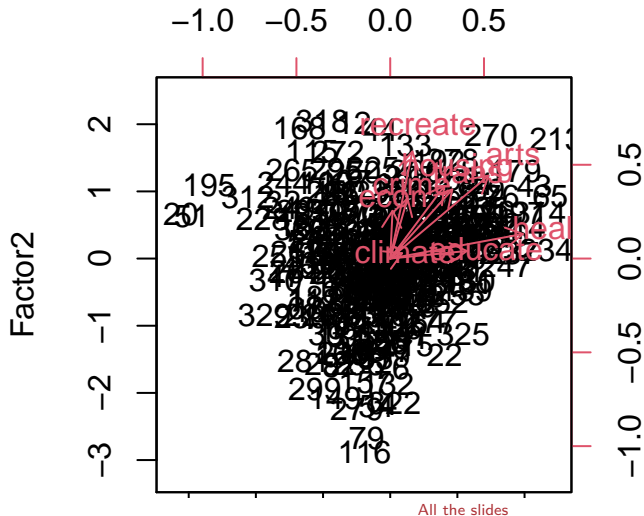
- Have to pick a number of factors *first*.
- Do this by running principal components and looking at scree plot.
- In this case, 3 factors seemed good (revisit later):

```
places.3 <- factanal(places_numeric, 3, scores = "r")
```

- There are different ways to get factor scores. These called “regression” scores.

A bad biplot

```
biplot(places.3$scores, places.3$loadings,
       xlab = places$id)
```



Comments

- I have to find a way to make a better biplot!
- Some of the variables now point straight up and some straight across (if you look carefully for the red arrows among the black points).
- This should make the factors more interpretable than the components were.

Factor loadings

```
places.3$loadings
```

```
##
## Loadings:
##          Factor1 Factor2 Factor3
## climate          0.994
## housing    0.360    0.482    0.229
## health     0.884    0.164
## crime      0.115    0.400    0.205
## trans      0.414    0.460
## educate    0.511
## arts       0.655    0.552    0.102
## recreate   0.148    0.714
## econ              0.318   -0.114
##
##          Factor1 Factor2 Factor3
## SS loadings    1.814    1.551    1.120
## Proportion Var  0.202    0.172    0.124
## Cumulative Var  0.202    0.374    0.498
```

Comments on loadings

- These are at least somewhat clearer than for the principal components:
 - Factor 1: health, education, arts: “well-being”
 - Factor 2: housing, transportation, arts (again), recreation: “places to be”
 - Factor 3: climate (only): “climate”
- In this analysis, economic factors don't seem to be important.

Factor scores

- Make a dataframe with the city IDs and factor scores:

```
cbind(id = places$id, places.3$scores) %>%  
  as_tibble() -> places_scores
```

- Make percentile ranks again (for checking):

```
places %>%  
  mutate(across(-id, ~percent_rank(.))) -> places_pr
```

Highest scores on factor 1, “well-being”:

- for the top 4 places:

```
places_scores %>%
  arrange(desc(Factor1))
```

```
## # A tibble: 329 x 4
##       id Factor1 Factor2 Factor3
##   <dbl>   <dbl>   <dbl>   <dbl>
## 1    213     2.47    1.78     0.506
## 2     65     2.39    0.925   -0.287
## 3    234     2.32    0.122    0.524
## 4    314     2.22    0.671    0.521
## 5     43     2.15    1.08     0.463
## 6    214     1.92    0.462    0.368
## 7    197     1.91    0.673   -2.39
## 8    179     1.86    1.31     1.82
## 9    237     1.82    0.0833  0.287
```


Check percentile ranks for factor 1

```
places_pr %>%
  select(id, health, educate, arts) %>%
  filter(id %in% c(213, 65, 234, 314))
```

```
## # A tibble: 4 x 4
##       id health educate  arts
##   <dbl> <dbl>   <dbl> <dbl>
## 1    65  0.997   0.963  0.997
## 2   213    1     0.723    1
## 3   234  0.991    1     0.985
## 4   314  0.985   0.994  0.991
```

- These are definitely high on the well-being variables.
- City #213 is not so high on education, but is highest of all on the others.

Highest scores on factor 2, “places to be”:

```
places_scores %>%
  arrange(desc(Factor2))
```

```
## # A tibble: 329 x 4
##       id Factor1 Factor2 Factor3
##   <dbl>   <dbl>   <dbl>   <dbl>
## 1    318   -1.01     2.05  -0.0957
## 2     12  -0.540     2.02  -3.80
## 3    168  -1.35     1.94   0.273
## 4     44  -0.149     1.92  -0.556
## 5    270   1.50     1.84   1.94
## 6    213   2.47     1.78   0.506
## 7    133   0.227     1.69   1.13
## 8    115  -1.17     1.59   0.292
## 9    272  -0.776     1.59   1.89
## 10   278   0.930     1.47   1.54
```

Check percentile ranks for factor 2

```
places_pr %>%
  select(id, housing, trans, arts, recreate) %>%
  filter(id %in% c(318, 12, 168, 44))
```

```
## # A tibble: 4 x 5
```

```
##       id housing trans  arts recreate
##   <dbl>   <dbl> <dbl> <dbl>    <dbl>
## 1    12    0.933 0.729 0.604    0.896
## 2    44    0.927 0.963 0.735    0.988
## 3   168    0.832 0.872 0.442    0.979
## 4   318    0.881 0.744 0.668    0.963
```

- These are definitely high on housing and recreation.
- Some are (very) high on transportation, but not so much on arts.
- Could look at more cities to see if #168 being low on arts is a fluke.

Highest scores on factor 3, “climate”:

```
places_scores %>%
  arrange(desc(Factor3))
```

```
## # A tibble: 329 x 4
##       id Factor1 Factor2 Factor3
##   <dbl>   <dbl>   <dbl>   <dbl>
## 1    227  -0.184    0.385    2.04
## 2    218   0.881    0.897    2.02
## 3    269   0.932    1.19     1.98
## 4    270   1.50     1.84     1.94
## 5     11   0.688    1.05     1.92
## 6    265  -1.46     1.32     1.90
## 7    272  -0.776    1.59     1.89
## 8    179   1.86     1.31     1.82
## 9    273   0.120    0.581    1.80
## 10    47  -1.57   -0.260    1.80
```

Check percentile ranks for factor 3

```
places_pr %>%  
  select(id, climate) %>%  
  filter(id %in% c(227, 218, 269, 270))
```

```
## # A tibble: 4 x 2  
##       id climate  
##   <dbl>   <dbl>  
## 1    218   0.997  
## 2    227   0.991  
## 3    269   0.994  
## 4    270   0.997
```

This is very clear.

Uniquenesses

- We said earlier that the economy was not part of any of our factors:

```
places.3$uniquenesses
```

```
##      climate      housing      health      crime      trans      educate
## 0.0050000 0.5859175 0.1854084 0.7842407 0.6165449 0.7351921
##      econ
## 0.8856382
```

- The higher the uniqueness, the less the variable concerned is part of any of our factors (and that maybe another factor is needed to accommodate it).
- This includes economy and maybe crime.

Test of significance

We can test whether the three factors that we have is enough, or whether we need more to describe our data:

```
places.3$PVAL
```

```
##      objective
```

```
## 1.453217e-14
```

- 3 factors are not enough.
- What would 5 factors look like?

Five factors

```
places.5 <- factanal(places_numeric, 5, scores = "r")
places.5$loadings
```

```
##
## Loadings:
##          Factor1 Factor2 Factor3 Factor4 Factor5
## climate                0.131   0.559
## housing    0.286    0.505   0.289  -0.113   0.475
## health     0.847    0.214                0.187
## crime              0.196   0.143   0.948   0.181
## trans     0.389    0.515                0.175
## educate    0.534
## arts       0.611    0.564                0.172   0.145
## recreate              0.705                0.115   0.136
## econ                      0.978   0.135
##
##
```


Comments

- On (new) 5 factors:
 - Factor 1 is health, education, arts: same as factor 1 before.
 - Factor 2 is housing, transportation, arts, recreation: as factor 2 before.
 - Factor 3 is economy.
 - Factor 4 is crime.
 - Factor 5 is climate and housing: like factor 3 before.
- The two added factors include the two “missing” variables.
- Is this now enough?

```
places.5$PVAL
```

```
##      objective
```

```
## 0.0009741394
```

- No. My guess is that the authors of Places Rated chose their 9 criteria to capture different aspects of what makes a city good or bad to live in, and so it was too much to hope that a small number of factors would come out of these.

A bigger example: BEM sex role inventory

- 369 women asked to rate themselves on 60 traits, like “self-reliant” or “shy”.
- Rating 1 “never or almost never true of me” to 7 “always or almost always true of me”.
- 60 personality traits is a lot. Can we find a smaller number of factors that capture aspects of personality?
- The whole BEM sex role inventory on next page.

The whole inventory

1. self reliant	21.reliable	41.warm
2. yielding	22.analytical	42.solemn
3. helpful	23.sympathetic	43.willing to take a stand
4. defends own beliefs	24.jealous	44.tender
5. cheerful	25.leadership ability	45.friendly
6. moody	26.sensitive to other's needs	46.aggressive
7. independent	27.truthful	47.gullible
8. shy	28.willing to take risks	48.inefficient
9. conscientious	29.understanding	49.acts as a leader
10.athletic	30.secretive	50.childlike
11.affectionate	31.makes decisions easily	51.adaptable
12.theatrical	32.compassionate	52.individualistic
13.assertive	33.sincere	53.does not use harsh language
14.flatterable	34.self-sufficient	54.unsystematic
15.happy	35.eager to soothe hurt feelings	55.competitive
16.strong personality	36.conceited	56.loves children
17.loyal	37.dominant	57.tactful
18.unpredictable	38.soft spoken	58.ambitious
19.forceful	39.likable	59.gentle
20.feminine	40.masculine	60.conventional

Some of the data

```
my_url <- "http://ritsokiguess.site/datafiles/factor.txt"
bem <- read_tsv(my_url)
bem
```

```
## # A tibble: 369 x 45
##   subno helpful reliant defbel yielding cheerful indpt athlet shy assert
##   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl> <dbl>   <dbl> <dbl>   <dbl>
## 1     1     1     7     7     5     5     7     7     7     1     7
## 2     2     2     5     6     6     6     2     3     3     3     4
## 3     3     3     7     6     4     4     5     5     2     3     4
## 4     4     4     6     6     7     4     6     6     3     4     4
## 5     5     5     6     6     7     4     7     7     7     2     7
## 6     6     7     5     6     7     4     6     6     2     4     4
## 7     7     8     6     4     6     6     6     3     1     3     3
## 8     8     9     7     6     7     5     6     7     5     2     5
## 9     9    10     7     6     6     4     4     5     2     2     5
## 10    10    11     7     4     7     4     7     5     2     1     5
## # ... with 359 more rows, and 35 more variables: strpers <dbl>, forceful <dbl>,
## # affect <dbl>, flatter <dbl>, loyal <dbl>, analyt <dbl>, feminine <dbl>,
## # sympathy <dbl>, moody <dbl>, sensitiv <dbl>, undstand <dbl>, compass <dbl>,
## # leaderab <dbl>, soothe <dbl>, risk <dbl>, decide <dbl>, selfsuff <dbl>,
## # conscien <dbl>, dominant <dbl>, masculin <dbl>, stand <dbl>, happy <dbl>,
## # softspok <dbl>, warm <dbl>, truthful <dbl>, tender <dbl>, gullible <dbl>,
## # leadact <dbl>, childlik <dbl>, individ <dbl>, foullang <dbl>
```

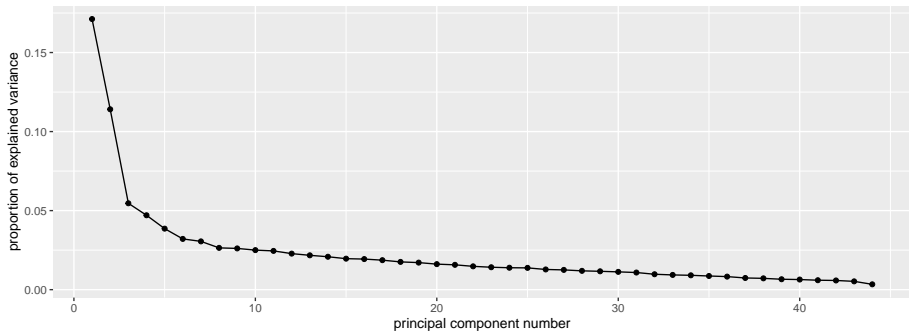
Principal components first

...to decide on number of factors:

```
bem.pc <- bem %>%  
  select(-subno) %>%  
  princomp(cor = T)
```

The scree plot

```
(g <- ggscreeplot(bem.pc))
```

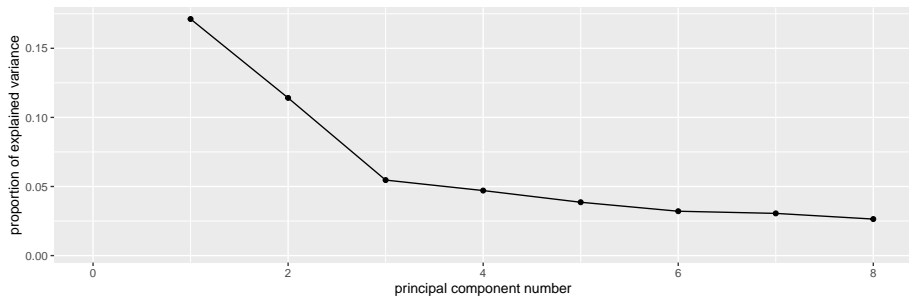


- No obvious elbow.

Zoom in to search for elbow

Possible elbows at 3 (2 factors) and 6 (5):

```
g + scale_x_continuous(limits = c(0, 8))
```



but is 2 really good?

```
summary(bem.pc)
```

```
## Importance of components:
```

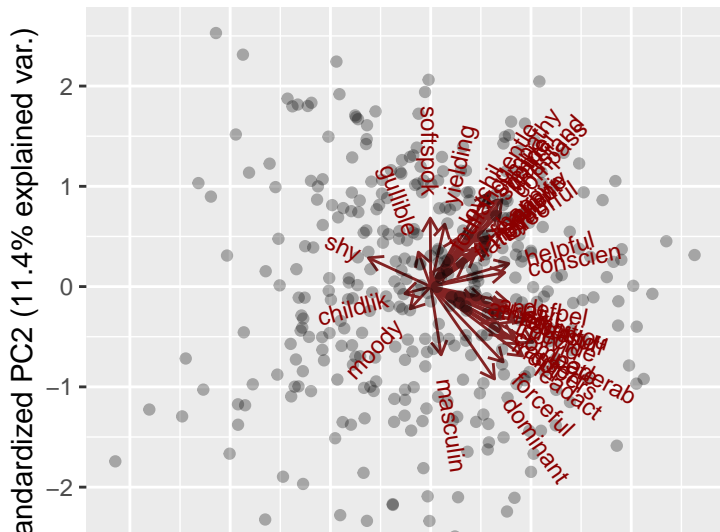
	Comp.1	Comp.2	Comp.3	Comp.4	Comp.5
## Standard deviation	2.7444993	2.2405789	1.55049106	1.43886350	1.30318840
## Proportion of Variance	0.1711881	0.1140953	0.05463688	0.04705291	0.03859773
## Cumulative Proportion	0.1711881	0.2852834	0.33992029	0.38697320	0.42557093
	Comp.6	Comp.7	Comp.8	Comp.9	Comp.10
## Standard deviation	1.18837867	1.15919129	1.07838912	1.07120568	1.04901318
## Proportion of Variance	0.03209645	0.03053919	0.02643007	0.02607913	0.02500974
## Cumulative Proportion	0.45766738	0.48820657	0.51463664	0.54071577	0.56572551
	Comp.11	Comp.12	Comp.13	Comp.14	Comp.15
## Standard deviation	1.03848656	1.00152287	0.97753974	0.95697572	0.9287543
## Proportion of Variance	0.02451033	0.02279655	0.02171782	0.02081369	0.0196042
## Cumulative Proportion	0.59023584	0.61303238	0.63475020	0.65556390	0.6751681
	Comp.16	Comp.17	Comp.18	Comp.19	Comp.20
## Standard deviation	0.92262649	0.90585705	0.8788668	0.86757525	0.84269120
## Proportion of Variance	0.01934636	0.01864948	0.0175547	0.01710652	0.01613928
## Cumulative Proportion	0.69451445	0.71316392	0.7307186	0.74782514	0.76396443
	Comp.21	Comp.22	Comp.23	Comp.24	Comp.25
## Standard deviation	0.83124925	0.80564654	0.78975423	0.78100835	0.77852606
## Proportion of Variance	0.01570398	0.01475151	0.01417527	0.01386305	0.01377506
## Cumulative Proportion	0.77966841	0.79441992	0.80859519	0.82245823	0.83623330

Comments

- Want overall fraction of variance explained (“cumulative proportion’ ’) to be reasonably high.
- 2 factors, 28.5%. Terrible!
- Even 56% (10 factors) not that good!
- Have to live with that.

Biplot

```
ggbiplot(bem.pc, alpha = 0.3)
```



Comments

- Ignore individuals for now.
- Most variables point to 1 o'clock or 4 o'clock.
- Suggests factor analysis with rotation will get interpretable factors (rotate to 12 o'clock and 3 o'clock, for example).
- Try for 2-factor solution (rough interpretation, will be bad):

```
bem %>%  
  select(-subno) %>%  
  factanal(factors = 2) -> bem.2
```

- Show output in pieces (just print `bem.2` to see all of it).

Uniquenesses, sorted

```
sort(bem.2$uniquenesses)
```

```
## leaderab leadact warm tender dominant gentle
## 0.4091894 0.4166153 0.4764762 0.4928919 0.4942909 0.5064551
## forceful strpers compass stand undstand assert
## 0.5631857 0.5679398 0.5937073 0.6024001 0.6194392 0.6329347
## soothe affect decide selfsuff sympathy indpt
## 0.6596103 0.6616625 0.6938578 0.7210246 0.7231450 0.7282742
## helpful defbel risk reliant individ compete
## 0.7598223 0.7748448 0.7789761 0.7808058 0.7941998 0.7942910
## conscien happy sensitiv loyal ambitiou shy
## 0.7974820 0.8008966 0.8018851 0.8035264 0.8101599 0.8239496
## softspok cheerful masculin yielding feminine truthful
## 0.8339058 0.8394916 0.8453368 0.8688473 0.8829927 0.8889983
## lovchil analyt athlet flatter gullible moody
## 0.8924392 0.8968744 0.9229702 0.9409500 0.9583435 0.9730607
## childlik foullang
## 0.9800360 0.9821662
```

Comments

- Mostly high or very high (bad).
- Some smaller, eg.: Leadership ability (0.409), Acts like leader (0.417), Warm (0.476), Tender (0.493).
- Smaller uniquenesses captured by one of our two factors.
- Larger uniquenesses are not: need more factors to capture them.

Factor loadings, some

```
bem.2$loadings
```

```
##
## Loadings:
##           Factor1 Factor2
## helpful    0.314   0.376
## reliant    0.453   0.117
## defbel     0.434   0.193
## yielding  -0.131   0.338
## cheerful   0.152   0.371
## indpt      0.521
## athlet     0.267
## shy        -0.414
## assert     0.605
## strpers    0.657
## forceful   0.649  -0.126
## affect     0.178   0.554
## flatter    0.223
## loyal      0.151   0.417
## analyt     0.295   0.127
## feminine   0.113   0.323
## sympathy   0.526
## moody      -0.162
## ...
```

Making a data frame

There are too many to read easily, so make a data frame. A bit tricky:

```
bem.2$loadings %>%
  unclass() %>%
  as_tibble() %>%
  mutate(trait = rownames(bem.2$loadings)) -> loadings
loadings %>% slice(1:8)
```

```
## # A tibble: 8 x 3
##   Factor1  Factor2 trait
##   <dbl>    <dbl> <chr>
## 1    0.314    0.376 helpful
## 2    0.453    0.117 reliant
## 3    0.434    0.193 defbel
## 4   -0.131    0.338 yielding
## 5    0.152    0.371 cheerful
## 6    0.521    0.00587 indpt
## 7    0.267    0.0755 athlet
## 8   -0.414   -0.0654 shy
```

Pick out the big ones on factor 1

Arbitrarily defining > 0.4 or < -0.4 as “big”:

```
loadings %>% filter(abs(Factor1) > 0.4)
```

```
## # A tibble: 17 x 3
##   Factor1 Factor2 trait
##   <dbl>   <dbl> <chr>
## 1  0.453  0.117  reliant
## 2  0.434  0.193  defbel
## 3  0.521  0.00587 indpt
## 4 -0.414 -0.0654 shy
## 5  0.605  0.0330 assert
## 6  0.657  0.0208 strpers
## 7  0.649 -0.126  forceful
## 8  0.765  0.0695 leaderab
## 9  0.442  0.161  risk
## 10 0.542  0.113  decide
## 11 0.511  0.134  selfsuff
## 12 0.668 -0.245  dominant
## 13 0.607  0.172  stand
## 14 0.763 -0.0407 leadact
## 15 0.445  0.0891 individ
## 16 0.450  0.0532 compete
```


Factor 2, the big ones

```
loadings %>% filter(abs(Factor2) > 0.4)
```

```
## # A tibble: 11 x 3
##   Factor1 Factor2 trait
##   <dbl>    <dbl> <chr>
## 1  0.178    0.554 affect
## 2  0.151    0.417 loyal
## 3  0.0230   0.526 sympathy
## 4  0.135    0.424 sensitiv
## 5  0.0911   0.610 undstand
## 6  0.114    0.627 compass
## 7  0.0606   0.580 soothe
## 8  0.119    0.430 happy
## 9  0.0796   0.719 warm
## 10 0.0511   0.710 tender
## 11 -0.0187   0.702 gentle
```

Plotting the two factors

- A bi-plot, this time with the variables reduced in size. Looking for unusual individuals.
- Have to run `factanal` again to get factor scores for plotting.

```
bem %>% select(-subno) %>%  
  factanal(factors = 2, scores = "r") -> bem.2a  
biplot(bem.2a$scores, bem.2a$loadings, cex = c(0.5, 0.5))
```

- Numbers on plot are row numbers of `bem` data frame.



Comments

- Variables mostly up (“feminine”) and right (“masculine”), accomplished by rotation.
- Some unusual individuals: 311, 214 (low on factor 2), 366 (high on factor 2), 359, 258 (low on factor 1), 230 (high on factor 1).

Individual 366

```
bem %>% slice(366) %>% glimpse()
```

```
## Rows: 1
## Columns: 45
## $ subno      <dbl> 755
## $ helpful    <dbl> 7
## $ reliant    <dbl> 7
## $ defbel     <dbl> 5
## $ yielding   <dbl> 7
## $ cheerful   <dbl> 7
## $ indpt      <dbl> 7
## $ athlet     <dbl> 7
## $ shy        <dbl> 2
## $ assert     <dbl> 1
## $ strpers    <dbl> 3
## $ forceful   <dbl> 1
## $ affect     <dbl> 7
## $ flatter    <dbl> 9
## $ loyal      <dbl> 7
## $ analyt     <dbl> 7
## $ feminine   <dbl> 7
## $ sympathy   <dbl> 7
## $ moody      <dbl> 1
## $ sensitiv   <dbl> 7
## $ undstand   <dbl> 7
## $ compass    <dbl> 6
## $ leaderab   <dbl> 3
## $ soothe     <dbl> 7
## $ risk       <dbl> 7
## $ decide     <dbl> 7
## $ selfsuff   <dbl> 7
## $ conscien   <dbl> 7
```

Comments

- Individual 366 high on factor 2, but hard to see which traits should have high scores (unless we remember).
- Idea 1: use percentile ranks as before.
- Idea 2: Rating scale is easy to interpret. So *tidy* original data frame to make easier to look things up.

Tidying original data

```
bem %>%
  mutate(row = row_number()) %>%
  pivot_longer(c(-subno, -row), names_to="trait",
               values_to="score") -> bem_tidy

bem_tidy
```

```
## # A tibble: 16,236 x 4
##   subno    row trait      score
##   <dbl> <int> <chr>    <dbl>
## 1      1      1 1 helpful      7
## 2      1      1 1 reliant      7
## 3      1      1 1 defbel       5
## 4      1      1 1 yielding     5
## 5      1      1 1 cheerful     7
## 6      1      1 1 indpt        7
## 7      1      1 1 athlet       7
## 8      1      1 1 shy          1
## 9      1      1 1 assert       7
## 10     1      1 1 strpers      7
## # ... with 16,226 more rows
```

Recall data frame of loadings

```
loadings %>% slice(1:10)
```

```
## # A tibble: 10 x 3
##   Factor1 Factor2 trait
##   <dbl>   <dbl> <chr>
## 1  0.314  0.376  helpful
## 2  0.453  0.117  reliant
## 3  0.434  0.193  defbel
## 4 -0.131  0.338  yielding
## 5  0.152  0.371  cheerful
## 6  0.521  0.00587 indpt
## 7  0.267  0.0755  athlet
## 8 -0.414 -0.0654  shy
## 9  0.605  0.0330  assert
## 10 0.657  0.0208  strpers
```

Want to add the factor scores for each trait to our tidy data frame `bem_tidy`. This is a left-join (over), matching on the column `trait` that is in both data frames (thus, the default):

Looking up loadings

```
bem_tidy %>% left_join(loadings) -> bem_tidy
```

```
## Joining, by = "trait"
```

```
bem_tidy %>% sample_n(12)
```

```
## # A tibble: 12 x 6
```

##	subno	row	trait	score	Factor1	Factor2
##	<dbl>	<int>	<chr>	<dbl>	<dbl>	<dbl>
## 1	555	324	indpt	5	0.521	0.00587
## 2	589	345	gentle	5	-0.0187	0.702
## 3	311	181	warm	4	0.0796	0.719
## 4	126	81	cheerful	7	0.152	0.371
## 5	401	227	individ	6	0.445	0.0891
## 6	264	152	gullible	5	-0.153	0.135
## 7	226	125	feminine	6	0.113	0.323
## 8	550	319	forceful	1	0.649	-0.126
## 9	342	201	affect	7	0.178	0.554
## 10	316	184	defbel	7	0.434	0.193
## 11	238	134	softspok	2	-0.230	0.336
## 12	244	138	strpers	7	0.657	0.0208

Individual 366, high on Factor 2

So now pick out the rows of the tidy data frame that belong to individual 366 (row=366) and for which the Factor2 score exceeds 0.4 in absolute value (our “big” from before):

```
bem_tidy %>% filter(row == 366, abs(Factor2) > 0.4)
```

```
## # A tibble: 11 x 6
##   subno   row trait      score Factor1 Factor2
##   <dbl> <int> <chr>    <dbl>   <dbl>   <dbl>
## 1    755   366 affect      7  0.178  0.554
## 2    755   366 loyal      7  0.151  0.417
## 3    755   366 sympathy  7  0.0230 0.526
## 4    755   366 sensitiv  7  0.135  0.424
## 5    755   366 undstand  7  0.0911 0.610
## 6    755   366 compass   6  0.114  0.627
## 7    755   366 soothe    7  0.0606 0.580
## 8    755   366 happy     7  0.119  0.430
## 9    755   366 warm      7  0.0796 0.719
## 10   755   366 tender    7  0.0511 0.710
## 11   755   366 gentle    7 -0.0187 0.702
```

As expected, high scorer on these.

Several individuals

Rows 311 and 214 were *low* on Factor 2, so their scores should be low. Can we do them all at once?

```
bem_tidy %>% filter(
  row %in% c(366, 311, 214),
  abs(Factor2) > 0.4
)
```

```
## # A tibble: 33 x 6
##   subno   row trait    score Factor1 Factor2
##   <dbl> <int> <chr>    <dbl>   <dbl>   <dbl>
## 1    369   214 affect      1  0.178  0.554
## 2    369   214 loyal       7  0.151  0.417
## 3    369   214 sympathy   4  0.0230 0.526
## 4    369   214 sensitiv   7  0.135  0.424
## 5    369   214 undstand   5  0.0911 0.610
## 6    369   214 compass    5  0.114  0.627
## 7    369   214 soothe     3  0.0606 0.580
## 8    369   214 happy      4  0.119  0.430
## 9    369   214 warm       1  0.0796 0.719
## 10   369   214 tender     3  0.0511 0.710
## # ... with 23 more rows
```

Individual by column

Un-tidy, that is, pivot_wider:

```
bem_tidy %>%
  filter(
    row %in% c(366, 311, 214),
    abs(Factor2) > 0.4
  ) %>%
  select(-subno, -Factor1, -Factor2) %>%
  pivot_wider(names_from=row, values_from=score)
```

```
## # A tibble: 11 x 4
##   trait      `214` `311` `366`
##   <chr>    <dbl> <dbl> <dbl>
## 1 affect      1     5     7
## 2 loyal       7     4     7
## 3 sympathy    4     4     7
## 4 sensitiv    7     4     7
## 5 undstand    5     3     7
## 6 compass     5     4     6
## 7 soothe      3     4     7
## 8 happy       4     3     7
## 9 warm        1     3     7
## 10 tender     3     4     7
## 11 gentle     2     3     7
```

366 high, 311 middling, 214 (sometimes) low.

Individuals 230, 258, 359

These were high, low, low on factor 1. Adapt code:

```
bem_tidy %>%
  filter(row %in% c(359, 258, 230), abs(Factor1) > 0.4) %>%
  select(-subno, -Factor1, -Factor2) %>%
  pivot_wider(names_from=row, values_from=score)
```

```
## # A tibble: 17 x 4
##   trait      `230` `258` `359`
##   <chr>    <dbl> <dbl> <dbl>
## 1 reliant      7     4     1
## 2 defbel      7     1     1
## 3 indpt      7     7     1
## 4 shy        2     7     5
## 5 assert      7     3     1
## 6 strpers     7     1     3
## 7 forceful    7     1     1
## 8 leaderab    7     1     1
## 9 risk       7     5     7
## 10 decide     7     1     2
## 11 selfsuff   7     4     1
## 12 dominant   7     1     1
## 13 stand      7     1     6
## 14 leadact    7     1     1
## 15 individ    7     3     3
## 16 compete    6     2     1
## 17 ambitiou   7     2     4
```

Is 2 factors enough?

Suspect not:

```
bem.2$PVAL
```

```
##      objective
```

```
## 1.458183e-150
```

2 factors resoundingly rejected. Need more. Have to go all the way to 15 factors to not reject:

```
bem %>%
```

```
  select(-subno) %>%
```

```
  factanal(factors = 15) -> bem.15
```

```
bem.15$PVAL
```

```
## objective
```

```
## 0.132617
```

Even then, only just over 50% of variability explained.

What's important in 15 factors?

- Let's take a look at the important things in those 15 factors.
- Get 15-factor loadings into a data frame, as before:

```
bem.15$loadings %>%  
  unclass() %>%  
  as_tibble() %>%  
  mutate(trait = rownames(bem.15$loadings)) -> loadings
```

- then show the highest few loadings on each factor.

Factor 1 (of 15)

```
loadings %>%  
  arrange(desc(abs(Factor1))) %>%  
  select(Factor1, trait) %>%  
  slice(1:10)
```

```
## # A tibble: 10 x 2  
##   Factor1 trait  
##   <dbl> <chr>  
## 1  0.813 compass  
## 2  0.676 undstand  
## 3  0.661 sympathy  
## 4  0.641 sensitiv  
## 5  0.597 soothe  
## 6  0.348 warm  
## 7  0.280 gentle  
## 8  0.279 tender  
## 9  0.250 helpful  
## 10 0.234 conscien
```

Compassionate, understanding, sympathetic, soothing: thoughtful of others.

Factor 2

```
loadings %>%  
  arrange(desc(abs(Factor2))) %>%  
  select(Factor2, trait) %>%  
  slice(1:10)
```

```
## # A tibble: 10 x 2  
##   Factor2 trait  
##   <dbl> <chr>  
## 1  0.762 strpers  
## 2  0.716 forceful  
## 3  0.698 assert  
## 4  0.504 dominant  
## 5  0.393 leaderab  
## 6  0.367 stand  
## 7  0.351 leadact  
## 8 -0.313 softspok  
## 9 -0.287 shy  
## 10 0.260 analyt
```

Strong personality, forceful, assertive, dominant: getting ahead.

Factor 3

```
loadings %>%  
  arrange(desc(abs(Factor3))) %>%  
  select(Factor3, trait) %>%  
  slice(1:10)
```

```
## # A tibble: 10 x 2  
##   Factor3 trait  
##   <dbl> <chr>  
## 1  0.670 reliant  
## 2  0.648 selfsuff  
## 3  0.620 indpt  
## 4  0.390 helpful  
## 5 -0.339 gullible  
## 6  0.333 individ  
## 7  0.332 decide  
## 8  0.329 conscien  
## 9  0.288 leaderab  
## 10 0.280 defbel
```

Self-reliant, self-sufficient, independent: going it alone.

Factor 4

```
loadings %>%  
  arrange(desc(abs(Factor4))) %>%  
  select(Factor4, trait) %>%  
  slice(1:10)
```

```
## # A tibble: 10 x 2  
##   Factor4 trait  
##   <dbl> <chr>  
## 1  0.696 gentle  
## 2  0.692 tender  
## 3  0.599 warm  
## 4  0.447 affect  
## 5  0.394 softspok  
## 6  0.278 lovchil  
## 7  0.244 undstand  
## 8  0.244 happy  
## 9  0.213 loyal  
## 10 0.202 soothe
```

Gentle, tender, warm (affectionate): caring for others.

Factor 5

```
loadings %>%
  arrange(desc(abs(Factor5))) %>%
  select(Factor5, trait) %>%
  slice(1:10)
```

```
## # A tibble: 10 x 2
##   Factor5 trait
##   <dbl> <chr>
## 1  0.696 compete
## 2  0.674 ambitiou
## 3  0.345 risk
## 4  0.342 individ
## 5  0.281 athlet
## 6  0.270 leaderab
## 7  0.245 decide
## 8  0.206 dominant
## 9  0.193 leadact
## 10 0.185 strpers
```

Ambitious, competitive (with a bit of risk-taking and individualism): Being the best.

Factor 6

```
loadings %>%
  arrange(desc(abs(Factor6))) %>%
  select(Factor6, trait) %>%
  slice(1:10)
```

```
## # A tibble: 10 x 2
##   Factor6 trait
##   <dbl> <chr>
## 1  0.868 leadact
## 2  0.608 leaderab
## 3  0.338 dominant
## 4  0.201 forceful
## 5 -0.192 shy
## 6  0.179 risk
## 7  0.170 masculin
## 8  0.164 decide
## 9  0.159 compete
## 10 0.147 athlet
```

Acts like a leader, leadership ability (with a bit of Dominant): Taking charge.

Factor 7

```
loadings %>%  
  arrange(desc(abs(Factor7))) %>%  
  select(Factor7, trait) %>%  
  slice(1:10)
```

```
## # A tibble: 10 x 2  
##   Factor7 trait  
##   <dbl> <chr>  
## 1  0.670 happy  
## 2  0.667 cheerful  
## 3 -0.522 moody  
## 4  0.219 athlet  
## 5  0.213 warm  
## 6  0.172 gentle  
## 7 -0.164 masculin  
## 8  0.160 reliant  
## 9  0.147 yielding  
## 10 0.141 lovchil
```

Happy and cheerful.

Factor 8

```
loadings %>%  
  arrange(desc(abs(Factor8))) %>%  
  select(Factor8, trait) %>%  
  slice(1:10)
```

```
## # A tibble: 10 x 2  
##   Factor8 trait  
##   <dbl> <chr>  
## 1  0.630 affect  
## 2  0.516 flatter  
## 3 -0.251 softspok  
## 4  0.221 warm  
## 5  0.188 tender  
## 6  0.185 strpers  
## 7 -0.180 shy  
## 8  0.180 compete  
## 9  0.166 loyal  
## 10 0.155 helpful
```

Affectionate, flattering: Making others feel good.

Factor 9

```
loadings %>%  
  arrange(desc(abs(Factor9))) %>%  
  select(Factor9, trait) %>%  
  slice(1:10)
```

```
## # A tibble: 10 x 2  
##   Factor9 trait  
##   <dbl> <chr>  
## 1  0.863 stand  
## 2  0.340 defbel  
## 3  0.245 individ  
## 4  0.194 risk  
## 5 -0.172 shy  
## 6  0.171 decide  
## 7  0.120 assert  
## 8  0.116 conscien  
## 9  0.112 analyt  
## 10 -0.112 gullible
```

Taking a stand.

Factor 10

```
loadings %>%  
  arrange(desc(abs(Factor10))) %>%  
  select(Factor10, trait) %>%  
  slice(1:10)
```

```
## # A tibble: 10 x 2  
##   Factor10 trait  
##   <dbl> <chr>  
## 1  0.808  feminine  
## 2 -0.264  masculin  
## 3  0.245  softspok  
## 4  0.232  conscien  
## 5  0.202  selfsuff  
## 6  0.176  yielding  
## 7  0.141  gentle  
## 8  0.113  flatter  
## 9  0.109  decide  
## 10 -0.0941 lovchil
```

Feminine. (A little bit of not-masculine!)

Factor 11

```
loadings %>%  
  arrange(desc(abs(Factor11))) %>%  
  select(Factor11, trait) %>%  
  slice(1:10)
```

```
## # A tibble: 10 x 2  
##   Factor11 trait  
##   <dbl> <chr>  
## 1  0.916 loyal  
## 2  0.189 affect  
## 3  0.159 truthful  
## 4  0.125 helpful  
## 5  0.104 analyt  
## 6  0.101 tender  
## 7  0.0972 lovchil  
## 8  0.0964 gullible  
## 9  0.0935 cheerful  
## 10 0.0821 conscien
```

Loyal.

Factor 12

```
loadings %>%  
  arrange(desc(abs(Factor12))) %>%  
  select(Factor12, trait) %>%  
  slice(1:10)
```

```
## # A tibble: 10 x 2  
##   Factor12 trait  
##   <dbl> <chr>  
## 1    0.611 childlik  
## 2   -0.285 selfsuff  
## 3   -0.279 conscien  
## 4    0.259 moody  
## 5    0.201 shy  
## 6   -0.167 decide  
## 7    0.154 masculin  
## 8    0.146 dominant  
## 9    0.138 compass  
## 10   -0.130 leaderab
```

Childlike. (With a bit of moody, shy, not-self-sufficient, not-conscientious.)

Factor 13

```
loadings %>%  
  arrange(desc(abs(Factor13))) %>%  
  select(Factor13, trait) %>%  
  slice(1:10)
```

```
## # A tibble: 10 x 2  
##   Factor13 trait  
##   <dbl> <chr>  
## 1    0.573 truthful  
## 2   -0.278 gullible  
## 3    0.263 happy  
## 4    0.189 warm  
## 5   -0.167 shy  
## 6    0.165 loyal  
## 7   -0.144 yielding  
## 8   -0.130 assert  
## 9    0.114 defbel  
## 10   -0.111 lovchil
```

Truthful. (With a bit of happy and not-gullible.)

Factor 14

```
loadings %>%  
  arrange(desc(abs(Factor14))) %>%  
  select(Factor14, trait) %>%  
  slice(1:10)
```

```
## # A tibble: 10 x 2  
##   Factor14 trait  
##   <dbl> <chr>  
## 1  0.443 decide  
## 2  0.237 selfsuff  
## 3  0.195 forceful  
## 4 -0.186 softspok  
## 5  0.160 risk  
## 6 -0.148 strpers  
## 7  0.146 dominant  
## 8  0.128 happy  
## 9  0.115 compass  
## 10 0.105 masculin
```

Decisive. (With a bit of self-sufficient and not-soft-spoken.)

Factor 15

```
loadings %>%  
  arrange(desc(abs(Factor15))) %>%  
  select(Factor15, trait) %>%  
  slice(1:10)
```

```
## # A tibble: 10 x 2  
##   Factor15 trait  
##   <dbl> <chr>  
## 1 -0.324 compass  
## 2  0.247 athlet  
## 3  0.229 sensitiv  
## 4  0.199 risk  
## 5 -0.164 affect  
## 6  0.163 moody  
## 7 -0.112 individ  
## 8  0.110 warm  
## 9  0.105 cheerful  
## 10 0.101 reliant
```

Not-compassionate, athletic, sensitive: A mixed bag. (“Cares about self”?)

Anything left out? Uniquenesses

```
enframe(bem.15$uniquenesses, name="quality", value="uniq") %>%
  arrange(desc(uniq)) %>%
  slice(1:10)
```

```
## # A tibble: 10 x 2
##   quality    uniq
##   <chr>     <dbl>
## 1 foullang  0.914
## 2 lovchil   0.824
## 3 analyt    0.812
## 4 yielding  0.791
## 5 masculin  0.723
## 6 athlet    0.722
## 7 shy       0.703
## 8 gullible  0.700
## 9 flatter   0.663
## 10 helpful  0.652
```

Uses foul language especially, also loves children and analytical. So could use even more factors.