

STAC32

Applications of Statistical Methods

Ken Butler

July 23, 2017

`jin intro, child="intro.Rnw" & i=`

Section 1

History, installation and connection

The men behind our software



Jim Goodnight, CEO SAS Inc



Ross Ihaka
Robert Gentleman
(Duncan Temple Lang)
originators of R

History

SAS

From late 1960s, North Carolina State.

Then: punched cards, “submit” job, get output later. Still SAS’s way of operating: run list of commands, get lot of output.

Commercialized, corporate ethos.

Strength: Submitting same commands again gets *exactly* same results. (Government, industry).

Long history: well-tested.

R

From 1993, New Zealand.

R style: enter commands one at a time, see output/graphics right away.

Open-source, free. Core group, anyone can contribute.

Grew out of commercial software S, which appeared when graphics terminals new (emphasis on graphics).

Concept of “function” lets you add onto R or do non-standard things.

Big user community makes sure everything works.

Installing R

- ▶ 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. Choose one close to you (faster), eg. U of T:

Bulgaria

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

Canada

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

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

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

Chile

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

<http://dirichlet.mat.puc.cl/>

China

<http://mirror.bjtu.edu.cn/cran/>

<https://mirrors.tuna.tsinghua.edu.cn/CRAN/>

Sofia University

Simon Fraser University, Burnaby

Dalhousie University, Halifax

University of Toronto

Pontificia Universidad Católica de Chile, Santiago

Pontificia Universidad Católica de Chile, Santiago

Beijing Jiaotong University, Beijing

TUNA Team, Tsinghua University

Click U of T

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

Download and Install R

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

- [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 >= 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

- ▶ Click the top link below:

[R-3.3.0 for Windows \(32/64 bit\)](#)

[Download R 3.3.0 for Windows](#) (62 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 exactly matches the package distributed by R, you can compare the [md5sum](#) of the .exe to the [true fingerprint](#). You will need a version of md5sum for windows: both [graphical](#) and [command line versions](#) are available.

[Frequently asked questions](#)

- ▶ Then install usual way.

Now, R Studio

- ▶ Go to <https://www.rstudio.com/>.
- ▶ Scroll down to this, and click Learn More (the left one):



Powerful IDE for R

RStudio IDE is a powerful and productive user interface for R. It's free and open source, and works great on Windows, Mac, and Linux.

[Learn More >](#)



R Packages

Our developers, expert trainers and authors of several popular R packages, including ggplot2, lubridate, and ot

[Learn More >](#)

Scroll down

- ▶ Scroll down to this:
-

Support	Community forums only	<ul style="list-style-type: none">• Priority Email Support• 8 hour response during business hours (ET)
License	AGPL v3	RStudio License Agreement
Pricing	Free	\$995/year

[DOWNLOAD RSTUDIO DESKTOP](#)[BUY NOW](#)

- ▶ Click “Download RStudio Desktop”.

Find the one for you

- ▶ Scroll down, and click the installer for your machine (Windows, Mac, 4 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](#).

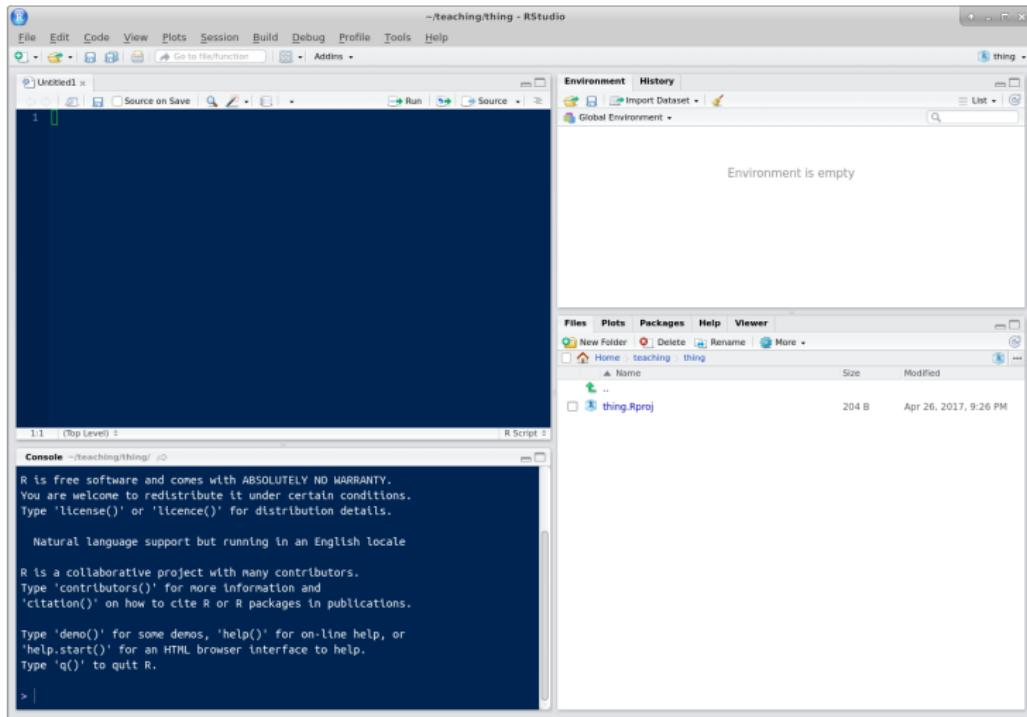
Installers for Supported Platforms

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.

Connecting to SAS

- ▶ SAS on your own computer big, expensive.
- ▶ U of T has “site licence” allows us to buy SAS for own computer (re-licensed every year, etc.)
- ▶ SAS offers “SAS Studio” that is free for the academic world. This runs through a web browser (accessible everywhere) with everything hosted on SAS’s servers, or on a “virtual machine” on own computer.
- ▶ The hard part is getting registered for it.

Getting registered for online version

- ▶ Go to <https://odamid.oda.sas.com>. Get to this:

The screenshot shows the SAS sign-in page with a dark blue header containing the "Sign In to SAS®" logo and the SAS slogan "THE POWER TO KNOW.". Below the header is a form with two input fields: "User ID:" and "Password:", each with a red asterisk indicating a required field. A "Sign In" button is located below the inputs. At the bottom of the page, there is a section with three links: "DON'T HAVE AN ACCOUNT? [REGISTER HERE](#)", "FORGOT YOUR USER ID? [CLICK HERE](#)", and "FORGOT YOUR PASSWORD? [CLICK HERE](#)". Copyright information at the bottom reads "Copyright © 2002 - 2013 by SAS Institute Inc., Cary, NC USA". A white arrow points from the text "Go down to ‘Don’t have an account?’ and click ‘Register Here’." in the accompanying text to the "REGISTER HERE" link.

- ▶ Bookmark this page.
- ▶ Go down to “Don’t have an account?” and click “Register Here”.

Enter your name and e-mail

... and select country (Canada):

Create an Account

Enter your name and email address.

Then check your email to complete the registration process.

First Name

Enter first name

Last Name

Enter last name

Email address

Enter email

Country Select a country ▼

Submit

Go check your e-mail

and look for something like this:



Dear Megan Butler,

Thank you for your interest in SAS® OnDemand for Academics.

A SAS Profile was created just for you. **Click on the link below to activate your profile and complete the registration process:**

<https://odamid.oda.sas.com/SASODAREgistration/activate.html?token=8BC03221-340E-98EF-9288-15EBBEE376CC>

Don't know what we're talking about?

If for some reason you did not register for SAS OnDemand for Academics (or if you think we sell shoes or airline tickets) just ignore this message.



Click on the link.

Choose a password

E-mail Verification

Thank you for verifying your email address.
Create a password to complete your registration.

E-mail address

Password

Confirm Password

I agree to the SAS OnDemand for Academics license. ([View license](#))

Create Account

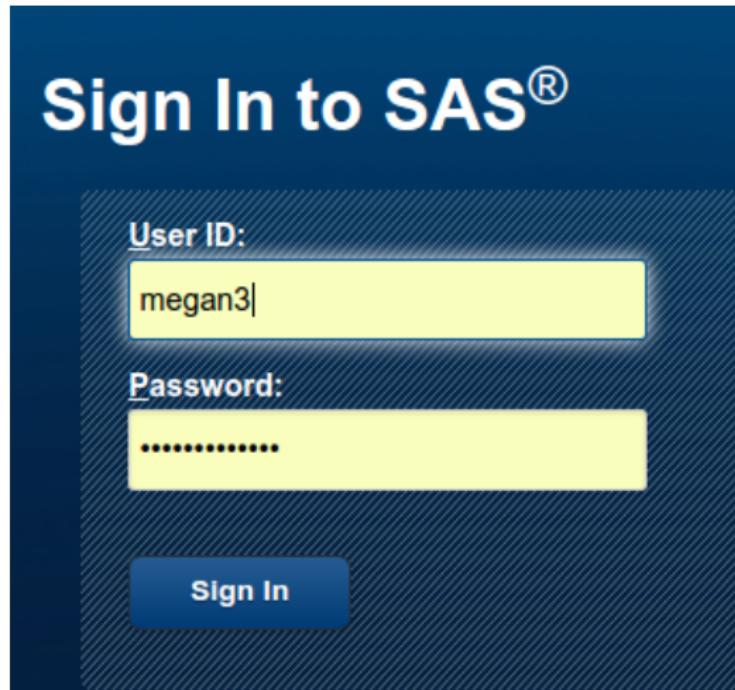
Password rules:

At least 8 characters long and contains characters from at least 3 of the following 4 categories:

- ▶ Click orange Create Account. You then get a user ID. Make a note of it.
- ▶ This completes the registration. You only do this once.

Log into SAS

Go back to the page you bookmarked earlier:



Type your user ID and password into the boxes, and click Sign In.

The dashboard

Dashboard Megan Butler ▾

Applications

[SAS® Studio](#)
Write and run SAS code with a Web-based SAS development environment.

Courses I teach (create a new course)

Name	Description	Institution
------	-------------	-------------

Courses I am enrolled in

Name	Description	Instructor	Institution
------	-------------	------------	-------------

To enroll in a course, you will need an 'enrollment link' sent by the course instructor

On the Dashboard, click SAS Studio. (Ignore the stuff about the courses.)

SAS, as you see it

Something like this:

The screenshot shows the SAS Web Editor interface. On the left, there is a sidebar with a search bar and a "Folders" section. Under "Folders", there are "Folder Shortcuts" and "My Folders". "My Folders" is expanded, showing "sasuserv93" which contains "Program 1.sas" and "Program 2.sas", and "UserProject.cws". On the right, there is a main workspace titled "Program 1". It has tabs for "CODE", "LOG", and "RESULTS". Below the tabs is a toolbar with various icons. A text input field contains the placeholder "Enter your code here".

Installing SAS on your own machine

- ▶ Pro: not dependent on SAS's servers.
- ▶ Con: fiendishly complicated!
- ▶ On your own computer, SAS runs in “virtual machine” (so doesn’t matter what OS you have, as long as the virtual machine runs on it).

Getting SAS for your own machine

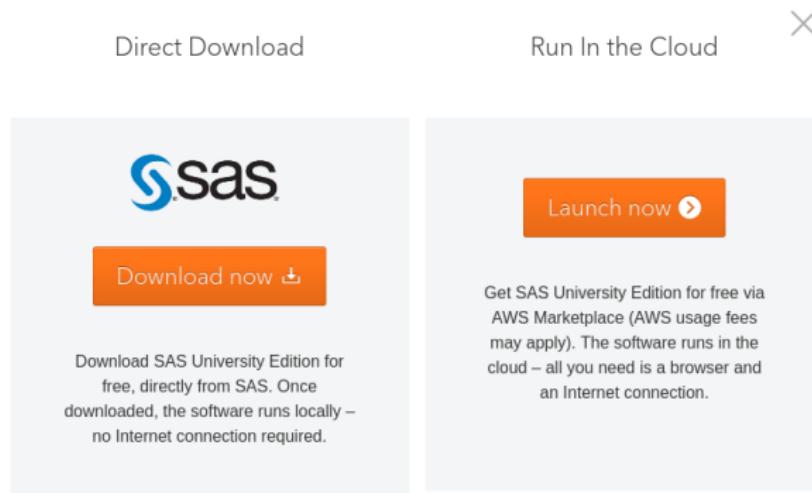
- ▶ Go to sas.com and navigate to Products and Solutions, then SAS University Edition, or go to http://www.sas.com/en_ca/software/university-edition.html.
- ▶ See this:

The screenshot shows the SAS University Edition landing page. At the top, there's a navigation bar with links for Welcome Megan, Edit Profile, Log Out, Canada (dropdown), En Français, Contact Us, Chat Now, and a search bar. Below the navigation is a secondary menu with links for Products & Solutions, Industries, Support & Training, Customer Stories, Partners, Community, and About SAS. The main content area features a large orange background graphic with geometric shapes and numbers (30, 20). The title "SAS® University Edition" is prominently displayed, followed by the tagline "Pay nothing. Gain everything." A blue button labeled "Get free software" with a right-pointing arrow is visible. The URL in the browser's address bar is "Home > Products & Solutions > SAS University Edition".

Free SAS® software. An interactive, online community. Superior training and documentation. And the analytical skills you need to secure your future.

And then

- ▶ Click Get Free Software. See this:



- ▶ Click Download Now on the *left*.

Select operating system

- ▶ by clicking appropriate tab, eg:

The screenshot shows a web browser displaying the SAS University Edition download page. The URL in the address bar is [http://www.sas.com/en_us/software/university-edition/download.html](#). The page has a light orange header with the text "Download SAS® University Edition". Below the header, there is a sub-header "Start by choosing your operating system below:". At the bottom of the page, there are three tabs: "Windows" (which is highlighted in blue), "OS X", and "LINUX".

Before You Begin

To ensure a smooth, trouble-free installation, make sure that your Windows desktop or laptop computer meets the minimum system requirements:

- Microsoft Windows 7, 8, 8.1 or 10
- 64-bit hardware,
minimum 1GB RAM
- One of these browsers:
 - Microsoft Internet Explorer 9,
10 or 11
 - Mozilla Firefox 21 or later
 - Google Chrome 27 or later

Starting setup

- ▶ Click tab for your operating system, and check that your system is good.
- ▶ Scroll down (4 steps):



Get the Quick Start Guide (PDF or Video).

And don't just download the PDF – actually read it. Or watch the video if that's more your thing. Or do both!



Quick Start PDF

[Download PDF](#) A small orange circular icon with a white arrow pointing to the right, indicating an external link.

Seriously. The Quick Start Guides give you step-by-step instructions for installing and configuring SAS University Edition on your laptop or desktop computer. You won't regret it.



Quick Start Video

[Watch video](#) A small orange circular icon with a white arrow pointing to the right, indicating an external link.

Download VirtualBox

- ▶ SAS runs on “virtual machine” (has own operating system regardless of what yours is). Download and install virtual machine:



Install Oracle VirtualBox virtualization software* on your machine.

Because SAS University Edition is a virtual application (or vApp), you need virtualization software to run it. You can download Oracle VirtualBox for Windows, a free virtualization software package, using the link below:

[Download VirtualBox for Windows](#) 

* Note: In addition to Oracle VirtualBox, SAS University Edition also works with VMware Workstation Player virtualization software. If you prefer to use VMware Workstation Player, you can download it here: [VMware Workstation Player download page](#). Charges may apply.

Scroll down some more

- ▶ You will be downloading a 1.7GB “app” (this may take a while). You may have to create a username/password first (next page):



Download the SAS® University Edition vApp.

Once you click the download button below, you'll be prompted to:

- ① Create a SAS profile if you don't already have one. If you already have a SAS profile, log in.
- ② Accept the user licensing agreement.
- ③ Begin the download. If your browser asks whether you want to save or open the file, click **Save** to save the file in your Downloads directory.

The screenshot shows a landing page for the SAS University Edition. At the top, it says "SAS® University Edition". Below that is a large blue button with the text "Get download" and a downward arrow icon. At the bottom of the page, there is a note in small text: "Note: The file is more than 1.7GB. Depending on your Internet connection, it might take awhile to download. Grab a snack, call a friend, read a book – it will be done before you know it. And remember – you're getting the world's most powerful analytics software. It's worth the wait!"

Creating a “profile”

- ▶ New User on the right (unless you already have a SAS profile):

The screenshot shows the SAS login interface. On the left, there's a "Already Have a Profile?" form with fields for Email (containing "nxskok@hotmail.com") and Password (redacted). There's also a "Keep me logged in" checkbox and a "Login" button. Below the login form are links for "Forgot Password?", "Need Help?", and "Chat Now". On the right, there's a "SAS Login" header and a "New User?" section. This section includes a "Create your profile to take full advantage of our site." message, a prominent orange "Create" button, and a "Why Create a Profile?" section listing benefits like joining communities, getting help, downloading software, and managing e-learning newsletters.

SAS Login

Already Have a Profile?

Email: nxskok@hotmail.com

Password:

Keep me logged in

Login

[Forgot Password?](#)

[Need Help?](#)

[Chat Now](#)

New User?

Create your profile to take full advantage of our site.

Create

Why Create a Profile?

Get social - join SAS communities.
Get help - follow your tech support questions.
Get software - download software/fixes.
Get informed - manage e-learning/newsletters.

Finally, step 4

- ▶ Follow the steps in the Quick Start Guide. Step 1 you probably already did:

Step 1: Install Oracle VirtualBox and download the SAS University Edition vApp.

- a. Install the latest release of Oracle VirtualBox using the link provided by your site administrator, or see <https://www.virtualbox.org/wiki/Downloads>.
- b. See the SAS University Edition download page (at http://www.sas.com/en_us/software/university-edition/download.html) to get the SAS University Edition vApp. When downloading the SAS University Edition vApp, you might be prompted by your browser to save or run the file. Click **Save** to save this file in your **Downloads** directory.

Quick Start step 2

- ▶ Follow the instructions. This attaches the “app” to your virtual machine so that it will run:

Step 2: Add the SAS University Edition vApp to VirtualBox.

- a. Launch VirtualBox, and then select **File > Import Appliance**.
- b. From the **Downloads** directory, select the file for the SAS University Edition vApp (an OVA file), and then click **Open**.
- c. Click **Next**, and then click **Import**.

Step 3: setting up file access

- ▶ This is kind of complicated, but follow the steps through, and then you can read in data files:

Step 3: Create a folder for your data and results.

- a. On your local computer (in a location that you will remember), create a folder called SASUniversityEdition and a subfolder called myfolders. You will save all of your SAS University Edition files to this location.
- b. In VirtualBox, select the SAS University Edition vApp, and then select **Machine > Settings**.
- c. In the navigation pane of the Settings dialog box, select **Shared Folders**, and then click .
- d. In the Add Share dialog box, select **Other** as the folder path.
- e. In the Browse for Folder window, open the SASUniversityEdition folder and select the myfolders subfolder. Click **OK** (or **Choose**, depending on your operating system).
- f. In the Add Share dialog box, confirm that **Read-only** is not selected, and then select the **Auto-mount** and **Make Permanent** (if available) options. Click **OK**.
- g. Click **OK** to close the Settings dialog box.

Start SAS

- ▶ All of the above you only do once (installation).
- ▶ To start SAS, do the below (every time):

Step 4: Start the SAS University Edition vApp.

In VirtualBox, select the SAS University Edition vApp, and then select **Machine > Start**. It might take a few minutes for the virtual machine to start.

Note: When the virtual machine is running, the screen with the SAS logo is replaced with a black console screen (called the Welcome window). You can minimize this window, but **do not close the Welcome window until you are ready to end your SAS session**.

Step 5: Open the SAS University Edition.

- a. In a web browser on your local computer, enter <http://localhost:10080>.
- b. From the SAS University Edition: Information Center, click **Start SAS Studio**.

SAS Studio online and on your machine

- ▶ SAS Studio runs identically whether it's online or on your machine.
- ▶ With one exception: accessing files (typically data files).
- ▶ Otherwise, any reference to SAS Studio applies equally well to either version.

Accessing data files in SAS Studio

- ▶ Depends on whether you're running SAS Studio online or on your computer.
- ▶ If you're running online, you have a username that you used for logging in, like ken or megan3.
- ▶ Online: access file as /home/ plus your username plus filename: eg. /home/megan3/mydata.txt.
- ▶ On your computer: /folders/myfolders/ plus filename, eg. /folders/myfolders/mydata.txt.
- ▶ Slashes in both cases are *forward* slashes, and you need one to start the filename.

Section 2

Reading data from files

Introduction

- ▶ First thing we need to do is to read in data, so that we can use our software to analyze.
- ▶ Both R and SAS.
- ▶ Consider these:
 - ▶ Spreadsheet data saved as .csv file.
 - ▶ “Delimited” data such as values separated by spaces.
 - ▶ Actual Excel spreadsheets.

A spreadsheet

test1.xlsx - LibreOffice Calc

File Edit View Insert Format Sheet Data Tools Window Help

Liberation Sans 10 A A A f(x) Σ =

	A	B	C	D	E
1	id	x	y	group	
2	p1	10	21	upper	
3	p2	11	20	lower	
4	p3	13	25	upper	
5	p4	15	27	lower	
6	p5	16	30	upper	
7	p6	17	31	lower	
8					
9					
10					
11					

Sheet1 | Default | Sum=0 | 200%

Save as .csv

- ▶ .csv or “comma-separated values” is a way of turning spreadsheet values into plain text.
- ▶ Easy to read into R or SAS
- ▶ but does *not* preserve formulas. (This is a reason for doing *all* your calculations in your statistical software, and *only* having data in your spreadsheet.)
- ▶ File, Save As Text CSV (or similar).

The .csv file

```
id,x,y,group
p1,10,21,upper
p2,11,20,lower
p3,13,25,upper
p4,15,27,lower
p5,16,30,upper
p6,17,31,lower
```

Reading file in R

- ▶ Start (as always) with

```
library(tidyverse)

## Loading tidyverse: ggplot2
## Loading tidyverse: tibble
## Loading tidyverse: tidyr
## Loading tidyverse: readr
## Loading tidyverse: purrr
## Loading tidyverse: dplyr

## Conflicts with tidy packages
-----
```



```
## filter(): dplyr, stats
## lag():   dplyr, stats
```

If you don't get this, do this (should only need it once):

```
install.packages("tidyverse")
```

Finding the file

- ▶ Run this:

```
f=file.choose()
```

This brings up a file selector. Use it to find your .csv file. Then type the name f to display what it contains:

```
f
```

```
## [1] "/home/ken/teaching/c32/notes/2017/test1.csv"
```

(This is Linux. Mac looks similar, Windows different.)

- ▶ Now R knows where our file is, and we can read it in (over).

Reading in the file

- ▶ Use `read_csv` with the name of the file. Save the read-in file in something, here called `mydata`:

```
mydata=read_csv(f)
```

```
## Parsed with column specification:  
## cols(  
##   id = col_character(),  
##   x = col_integer(),  
##   y = col_integer(),  
##   group = col_character()  
## )
```

- ▶ `read_csv` guesses what kind of thing is in each column. Here it correctly guesses that:
 - ▶ `id` and `group` are text (categorical variables). `id` is actually “identifier variable”: identifies individuals.
 - ▶ `x` and `y` are integers (quantitative variables that here have no decimal point). Decimal numbers would be labelled `num` or `double`.

Looking at what we read in

- ▶ Again, type the name of the thing to display it:

```
mydata
```

```
## # A tibble: 6 x 4
##       id     x     y group
##   <chr> <int> <int> <chr>
## 1 p1     10    21 upper
## 2 p2     11    20 lower
## 3 p3     13    25 upper
## 4 p4     15    27 lower
## 5 p5     16    30 upper
## 6 p6     17    31 lower
```

- ▶ This is a “tibble” or data frame, the standard way of storing a data set in R.
- ▶ Tibbles print as much as will display on the screen. If there are more rows or columns, it will say so.

View-ing your data frame

- ▶ Another way to examine your data frame is to View it, like this:

```
View(mydata)
```

- ▶ This pops up a “data frame viewer” top left:

The screenshot shows a data frame viewer window titled "mydata". The window has a toolbar with icons for file operations and a search bar. Below the toolbar is a table with the following data:

	id	x	y	group
1	p1	10	21	upper
2	p2	11	20	lower
3	p3	13	25	upper
4	p4	15	27	lower
5	p5	16	30	upper
6	p6	17	31	lower

At the bottom of the viewer, a message says "Showing 1 to 6 of 6 entries".

What you can and cannot do with this View

- ▶ Read-only: cannot edit data
- ▶ Can display data satisfying conditions: click on Filter, then:
 - ▶ for a categorical variable, type name of category you want
 - ▶ for a quantitative variable, use slider to describe values you want.
- ▶ Can sort a column into ascending or descending order (click little arrows next to column name).
- ▶ Clicking the symbol left of Filter “pops out” View into separate (bigger) window.
- ▶ Cannot include in output to hand in (except by taking screenshot and handing *that* in).

Summarizing what we read in

- ▶ It is *always* a good idea to look at your data after you have read it in, to make sure you have believable numbers (and the right number of individuals and variables).
- ▶ Sometimes the data set is too big, and you want to summarize it:

```
glimpse(mydata)
```

```
## Observations: 6
## Variables: 4
## $ id      <chr> "p1", "p2", "p3", "p4", "p5", "p6"
## $ x       <int> 10, 11, 13, 15, 16, 17
## $ y       <int> 21, 20, 25, 27, 30, 31
## $ group   <chr> "upper", "lower", "upper", "lower", "upper"
```

- ▶ This tells you how many observations and variables, and shows you the first few values (almost all of them here).

Five-number summary

- ▶ this way:

```
summary(mydata)
```

```
##      id              x              y      group
##  Length:6        Min.   :10.00   Min.   :20.00  Length:6
##  Class :character 1st Qu.:11.50  1st Qu.:22.00  Class :character
##  Mode   :character Median :14.00  Median :26.00  Mode   :character
##                                         Mean   :13.67  Mean   :25.67
##                                         3rd Qu.:15.75 3rd Qu.:29.25
##                                         Max.   :17.00  Max.   :31.00
```

- ▶ For quantitative variables, a five-number summary plus the mean.
- ▶ For categorical variables, count of how many rows.
- ▶ Quick check for errors: these often show up as values too high or too low, so the min and/or max will be unreasonable.

Reading from a URL

- ▶ Any data file on the Web can be read directly.

- ▶ Example data:

`http://www.utsc.utoronto.ca/~butler/c32/global.csv.`

- ▶ Use URL instead of filename:

```
url = "http://www.utsc.utoronto.ca/~butler/c32/global.csv"
global = read_csv(url)
```

```
## Parsed with column specification:
## cols(
##   warehouse = col_character(),
##   size = col_integer(),
##   cost = col_double()
## )
```

The data

```
global

## # A tibble: 10 x 3
##   warehouse size   cost
##       <chr>  <int>  <dbl>
## 1         A    225 11.95
## 2         B    350 14.13
## 3         A    150  8.93
## 4         A    200 10.98
## 5         A    175 10.03
## 6         A    180 10.13
## 7         B    325 13.75
## 8         B    290 13.30
## 9         B    400 15.00
## 10        A    125  7.97
```

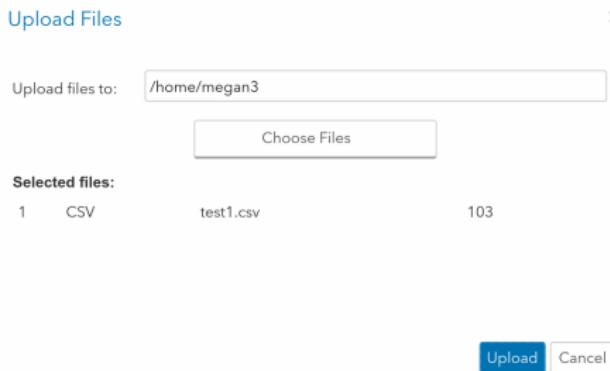
Now, in SAS

- ▶ Now, do whole thing again in SAS.
- ▶ In SAS Studio, click on Files (Home) and find the Upload button (4th one in top row) (should be not greyed out):



... Continued

- ▶ Click the Upload button, and then Choose Files in the box that pops up. This brings up a file selector as `file.choose` does in R. Find your .csv file, and click to “open” it. It should appear on your Upload Files box:



- ▶ Click Upload. When it's done, you should see your .csv file under Files (Home) on the left.

Reading in the data

- ▶ In SAS Studio, click New (leftmost button under Server Files and Folders) and select New SAS Program.
- ▶ On the right, in the Code tab, type code like this, only instead of ken put your username:

```
proc import  
    datafile='/home/ken/test1.csv'  
    dbms=csv  
    out=mydata  
    replace;  
    getnames=yes;
```

```
proc print;
```

- ▶ Make sure you get *all* the semicolons in the right places!
- ▶ This will read in the data that you uploaded, and list the whole data set.

Running the code

- ▶ Find the “running human” under the word Code. Click it.
- ▶ If all goes well, you should see the data set displayed in a Results tab:

The screenshot shows the SAS software interface. At the top, there is a title bar with a logo and the text "*Program 1". Below the title bar, there are four tabs: CODE (selected), LOG, RESULTS (highlighted with a dashed border), and OUTPUT DATA. Underneath the tabs are several icons: a magnifying glass, a document, a downward arrow, a computer monitor, a mail icon, and a refresh symbol. Below these icons is a link labeled "Table of Contents". To the right of the interface, there is a data table titled "RESULTS". The table has columns labeled Obs, id, x, y, and group. The data is as follows:

Obs	id	x	y	group
1	p1	10	21	upper
2	p2	11	20	lower
3	p3	13	25	upper
4	p4	15	27	lower
5	p5	16	30	upper
6	p6	17	31	lower

- ▶ If not, you'll get taken to the Log tab, which will show you where your error was. Fix it, and try again. (SAS can sometimes guess what you meant, even if it's not what you typed.)

That code

- ▶ proc print displays the whole data set.
- ▶ The proc import organizes reading in the data:
 - ▶ datafile says where to find the data file (on SAS Studio's server, where you uploaded it to).
 - ▶ dbms says what kind of file it is, a .csv in this case.
 - ▶ out gives the data set a name within SAS (that can be used to refer to this data set later)
 - ▶ getnames means to take the variable names from the top line of the data file (which is usually where they are).

Alternatively

- ▶ Click the New button, but then Import Data.
- ▶ Find your data file on the left, and drag it across to Select File on the right.
- ▶ Some code will appear. This is (basically) the `proc import` code we used above.
- ▶ Copy the text from FILENAME down to RUN; (inclusive).
- ▶ Open a New SAS Code window. Paste the copied code into it.
- ▶ Add anything else at the bottom, like a `proc print`, and run as before.

Summarizing a data set

- ▶ Replace the proc print with proc means:

```
proc means;
```

That gives the mean, SD, min, max and #observations for each variable (below).

- ▶ Note that you only get means for quantitative variables.
- ▶ proc print, proc means etc. work on *the most recently created data set* (usually what you want).

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
x	6	13.6666667	2.8047579	10.0000000	17.0000000
y	6	25.6666667	4.5460606	20.0000000	31.0000000

Reading from a URL

- ▶ A little extra setup:

```
filename myurl  
      url "http://www.utsc.utoronto.ca/~butler/c32/global.csv"  
  
proc import  
      datafile=myurl  
      dbms=csv  
      out=global  
      replace;  
      getnames=yes;  
  
proc print;
```

- ▶ The `filename` line says that the piece of text is actually a URL rather than a filename on this computer.

Did it work?

Obs	warehouse	size	cost
1	A	225	11.95
2	B	350	14.13
3	A	150	8.93
4	A	200	10.98
5	A	175	10.03
6	A	180	10.13
7	B	325	13.75
8	B	290	13.3
9	B	400	15
10	A	125	7.97

Space-delimited files

- Another common format for data is a text file with the values separated by spaces. Data below in two long columns with right side below left side:

cup tempdiff	SIGG 24.5
Starbucks 13	CUPPS 6
Starbucks 7	CUPPS 6
Starbucks 7	CUPPS 18.5
Starbucks 17.5	CUPPS 10
Starbucks 10	CUPPS 17.5
Starbucks 15.5	CUPPS 11
Starbucks 6	CUPPS 6.5
Starbucks 6	Nissan 2
SIGG 12	Nissan 1.5
SIGG 16	Nissan 2
SIGG 9	Nissan 3
SIGG 23	Nissan 0
SIGG 11	Nissan 7
SIGG 20.5	Nissan 0.5
SIGG 12.5	Nissan 6
SIGG 20.5	

Reading in these data

- ▶ Change the proc import:

```
proc import  
    datafile='/home/ken/coffee.txt'  
    dbms=dlm  
    out=coffee  
    replace;  
    delimiter=' ' ;  
    getnames=yes;
```

- ▶ On `dbms`, `dlm` means “delimited file”, that is, “values separated by something”. So we have to say what the values are separated by, namely exactly one space. (The values could be separated by anything.)

Did it work?

- ▶ The first 15 (of 32) lines. It seems to have worked:

```
proc print data=coffee(obs=15);
```

Obs	cup	tempdiff
1	Starbucks	13
2	Starbucks	7
3	Starbucks	7
4	Starbucks	17.5
5	Starbucks	10
6	Starbucks	15.5
7	Starbucks	6
8	Starbucks	6
9	SIGG	12
10	SIGG	16
11	SIGG	9
12	SIGG	23
13	SIGG	11
14	SIGG	20.5
15	SIGG	12.5

Reading the coffee data into R

- ▶ Save the text file somewhere and then find it:

```
f=file.choose()
```

```
f
```

```
## [1] "/home/ken/teaching/c32/notes/2017/coffee.txt"
```

- ▶ This time, `read_delim`, and again we have to say what the thing is separating the values:

```
coffee=read_delim(f, " ")
```

```
## Parsed with column specification:
```

```
## cols(
##   cup = col_character(),
##   tempdiff = col_double()
## )
```

- ▶ Name of the cup, text, and tempdiff, a decimal number.

Looking at the values

coffee

```
## # A tibble: 32 x 2
##       cup tempdiff
##   <chr>    <dbl>
## 1 Starbucks     13.0
## 2 Starbucks      7.0
## 3 Starbucks      7.0
## 4 Starbucks     17.5
## 5 Starbucks     10.0
## 6 Starbucks     15.5
## 7 Starbucks      6.0
## 8 Starbucks      6.0
## 9 SIGG          12.0
## 10 SIGG         16.0
## # ... with 22 more rows
```

These were four brands of travel mug (in cup), and for each, how much the temperature of the coffee in the mug decreased over 30 minutes.

Reading from the Web

- ▶ For R, use the URL in place of the filename (or in place of the `f` saved from `file.choose()`).
- ▶ for SAS, do the `filename` thing, which works for any type of file.

Reading soap data in R

```
url="http://www.utsc.utoronto.ca/~butler/c32/soap.txt"
soap=read_delim(url," ")

## Parsed with column specification:
## cols(
##   case = col_integer(),
##   scrap = col_integer(),
##   speed = col_integer(),
##   line = col_character()
## )
```

The soap data

```
soap
```

```
## # A tibble: 27 x 4
##       case scrap speed line
##   <int> <int> <int> <chr>
## 1     1     1    218    100    a
## 2     2     2    248    125    a
## 3     3     3    360    220    a
## 4     4     4    351    205    a
## 5     5     5    470    300    a
## 6     6     6    394    255    a
## 7     7     7    332    225    a
## 8     8     8    321    175    a
## 9     9     9    410    270    a
## 10   10    10    260    170    a
## # ... with 17 more rows
```

Reading soap data in SAS

```
filename myurl  
url "http://www.utsc.utoronto.ca/~butler/c32/soap.txt";  
  
proc import  
    datafile=myurl  
    dbms=dlm  
    out=soap  
    replace;  
    delimiter=' ' ;  
    getnames=yes;  
  
proc print data=soap(obs=10);
```

Ten rows of the soap data

Obs	case	scrap	speed	line
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
9	9	410	270	a
10	10	260	170	a

Data aligned in columns

- Sometimes you see data aligned in columns, thus:

DrugA	DrugB	DrugC
4	6	6
5	8	7
4	4	6
3	5	6
2	4	7
4	6	5
3	5	6
4	8	5
4	6	5

- `read_delim` will not work: values separated by *more than one* space.
- In R, `read_table` works for this.
- In SAS, not possible with `proc import`.

Reading in column-aligned data

```
drugs=read_table("migraine.txt")  
  
## Parsed with column  
specification:  
## cols(  
##   DrugA = col_integer(),  
##   DrugB = col_integer(),  
##   DrugC = col_integer()  
## )
```

```
drugs  
  
## # A tibble: 9 x 3  
##   DrugA DrugB DrugC  
##   <int> <int> <int>  
## 1     4     6     6  
## 2     5     8     7  
## 3     4     4     6  
## 4     3     5     6  
## 5     2     4     7  
## 6     4     6     5  
## 7     3     5     6  
## 8     4     8     5  
## 9     4     6     5
```

Reading an Excel sheet directly

- ▶ Here is my spreadsheet from before, but tarterd up a bit:

The screenshot shows the LibreOffice Calc application window titled "test2.xlsx - LibreOffice Calc". The menu bar includes File, Edit, View, Insert, Format, Sheet, Data, Tools, Window, and Help. The toolbar contains icons for file operations like Open, Save, Print, and Paste. The font and size dropdowns are set to "Liberation Sans" and "10". The formula bar shows "B10" and a formula input field with $f(x)$, Σ , and a result placeholder. The spreadsheet has columns A through E and rows 1 through 8. Row 1 contains headers: "id", "x", "y", "group". Rows 2 through 7 contain data: p1, 10, 21, upper; p2, 11, 20, lower; p3, 13, 25, upper; p4, 15, 27, lower; p5, 16, 30, upper; p6, 17, 31, lower. Row 8 is empty. The status bar at the bottom shows "Sheet 1 of 2", "Default", and "Sum=0".

	A	B	C	D	E
1	id	x	y	group	
2	p1	10	21	upper	
3	p2	11	20	lower	
4	p3	13	25	upper	
5	p4	15	27	lower	
6	p5	16	30	upper	
7	p6	17	31	lower	
8					

- ▶ It is now a workbook with a second sheet called “notes” (that we don’t want).

Reading it in

- ▶ Read into R, saying that we only want the sheet “data”. Start with `file.choose` as before (omitted):

```
library(readxl)
mydata2=read_excel(f,sheet="data")
mydata2

## # A tibble: 6 x 4
##       id     x     y group
##   <chr> <dbl> <dbl> <chr>
## 1 p1     10    21 upper
## 2 p2     11    20 lower
## 3 p3     13    25 upper
## 4 p4     15    27 lower
## 5 p5     16    30 upper
## 6 p6     17    31 lower
```

- ▶ That has worked properly.

Reading Excel spreadsheet into SAS

- ▶ Upload the spreadsheet file (as for uploading .csv file)
- ▶ Then, like this:

```
proc import  
    datafile='/home/ken/test2.xlsx'  
    dbms=xlsx  
    out=mydata  
    replace;  
    sheet=data;  
    getnames=yes;
```

- ▶ dbms is now xlsx for reading this type of file (or xls if you have old-style spreadsheet).
- ▶ Use sheet= to say which worksheet you want (no quotes).

The spreadsheet as data set

- Did it work? Yes:

```
proc print;
```

Obs	id	x	y	group
1	p1	10	21	upper
2	p2	11	20	lower
3	p3	13	25	upper
4	p4	15	27	lower
5	p5	16	30	upper
6	p6	17	31	lower

Reading Excel files from the Web

- ▶ No surprises here.
- ▶ In R, `read_excel` will take a URL rather than a filename, in the same way that `read_csv` and `read_delim` do.
- ▶ In SAS, define a `filename` as before, and use it in the appropriate `proc import`.

Section 3

Graphs

Our data

- ▶ To illustrate making graphs, we need some data.
- ▶ Data on 202 male and female athletes at the Australian Institute of Sport.
- ▶ Variables:
 - ▶ categorical: Sex of athlete, sport they play
 - ▶ quantitative: height (cm), weight (kg), lean body mass, red and white blood cell counts, haematocrit and haemoglobin (blood), ferritin concentration, body mass index, percent body fat.
- ▶ Values separated by *tabs* (which impacts reading in).

Reading data into R

- ▶ Use `read_tsv` ("tab-separated values"), like `read_csv`.
- ▶ Data in `ais.txt`:

```
athletes=read_tsv("ais.txt")  
  
## Parsed with column specification:  
## cols(  
##   Sex = col_character(),  
##   Sport = col_character(),  
##   RCC = col_double(),  
##   WCC = col_double(),  
##   Hc = col_double(),  
##   Hg = col_double(),  
##   Ferr = col_integer(),  
##   BMI = col_double(),  
##   SSF = col_double(),  
##   '%Bfat' = col_double(),  
##   LBM = col_double(),  
##   Ht = col_double(),  
##   Wt = col_double()  
## )
```

The data (some)

```
athletes
```

```
## # A tibble: 202 x 13
##       Sex   Sport    RCC    WCC     Hc     Hg   Ferr    BMI    SSF `^%Bfat`    LB
##       <chr> <chr> <dbl> <dbl> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <dbl>
## 1 female Netball  4.56  13.3  42.2  13.6    20 19.16  49.0  11.29 53.1
## 2 female Netball  4.15   6.0  38.0  12.7    59 21.15 110.2  25.26 47.0
## 3 female Netball  4.16   7.6  37.5  12.3    22 21.40  89.0  19.39 53.4
## 4 female Netball  4.32   6.4  37.7  12.3    30 21.03  98.3  19.63 48.7
## 5 female Netball  4.06   5.8  38.7  12.8    78 21.77 122.1  23.11 56.0
## 6 female Netball  4.12   6.1  36.6  11.8    21 21.38  90.4  16.86 56.4
## 7 female Netball  4.17   5.0  37.4  12.7   109 21.47 106.9  21.32 53.1
## 8 female Netball  3.80   6.6  36.5  12.4   102 24.45 156.6  26.57 54.4
## 9 female Netball  3.96   5.5  36.3  12.4    71 22.63 101.1  17.93 55.9
## 10 female Netball 4.44   9.7  41.4  14.1    64 22.80 126.4  24.97 51.6
## # ... with 192 more rows, and 2 more variables: Ht <dbl>, Wt <dbl>
```

Reading data into SAS

- ▶ Upload file to SAS Studio first.
- ▶ A bit trickier because we can't *type* tab: have to use special code '09'x (ASCII code 09 in hex):

```
proc import  
    datafile='/home/ken/ais.txt'  
    dbms=dlm  
    out=sports  
    replace;  
    delimiter='09'x;  
    getnames=yes;
```

Some of the data, tiny

```
proc print data=sports(obs=9);
```

Obs	Sex	Sport	RCC	WCC	Hc
1	female	Netball	4.56	13.3	42.2
2	female	Netball	4.15	6	38
3	female	Netball	4.16	7.6	37.5
4	female	Netball	4.32	6.4	37.7
5	female	Netball	4.06	5.8	38.7
6	female	Netball	4.12	6.1	36.6
7	female	Netball	4.17	5	37.4
8	female	Netball	3.8	6.6	36.5
9	female	Netball	3.96	5.5	36.3

Obs	Hg	Ferr	BMI	SSF
1	13.6	20	19.16	49
2	12.7	59	21.15	110.2
3	12.3	22	21.4	89
4	12.3	30	21.03	98.3
5	12.8	78	21.77	122.1
6	11.8	21	21.38	90.4
7	12.7	109	21.47	106.9
8	12.4	102	24.45	156.6
9	12.4	71	22.63	101.1

Or, summarized

```
proc means;
```

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
RCC	202	4.7186139	0.4579764	3.8000000	6.7200000
WCC	202	7.1086634	1.8005490	3.3000000	14.3000000
Hc	202	43.0915842	3.6629894	35.9000000	59.7000000
Hg	202	14.5663366	1.3624515	11.6000000	19.2000000
Ferr	202	76.8762376	47.5012388	8.0000000	234.0000000
BMI	202	22.9558911	2.8639328	16.7500000	34.4200000
SSF	202	69.0217822	32.5653330	28.0000000	200.8000000
_Bfat	202	13.5074257	6.1898260	5.6300000	35.5200000
LBM	202	64.8737129	13.0701972	34.3600000	106.0000000
Ht	202	180.1039604	9.7344945	148.9000000	209.4000000
Wt	202	75.0081683	13.9255740	37.8000000	123.2000000

Kinds of graph

Depends on number and type of variables you have:

Categorical	Quantitative	Graph
1	0	bar chart
0	1	histogram
2	0	grouped bar charts
1	1	side-by-side boxplots
0	2	scatterplot
2	1	grouped boxplots
1	2	scatterplot with points identified by group (eg. by colour)

With more variables, might want *separate plots by groups*. This is called facetting in R, paneling in SAS.

Workhorse graphing procedures

R and SAS have standard graphing procedures, that we use for all our graphs:

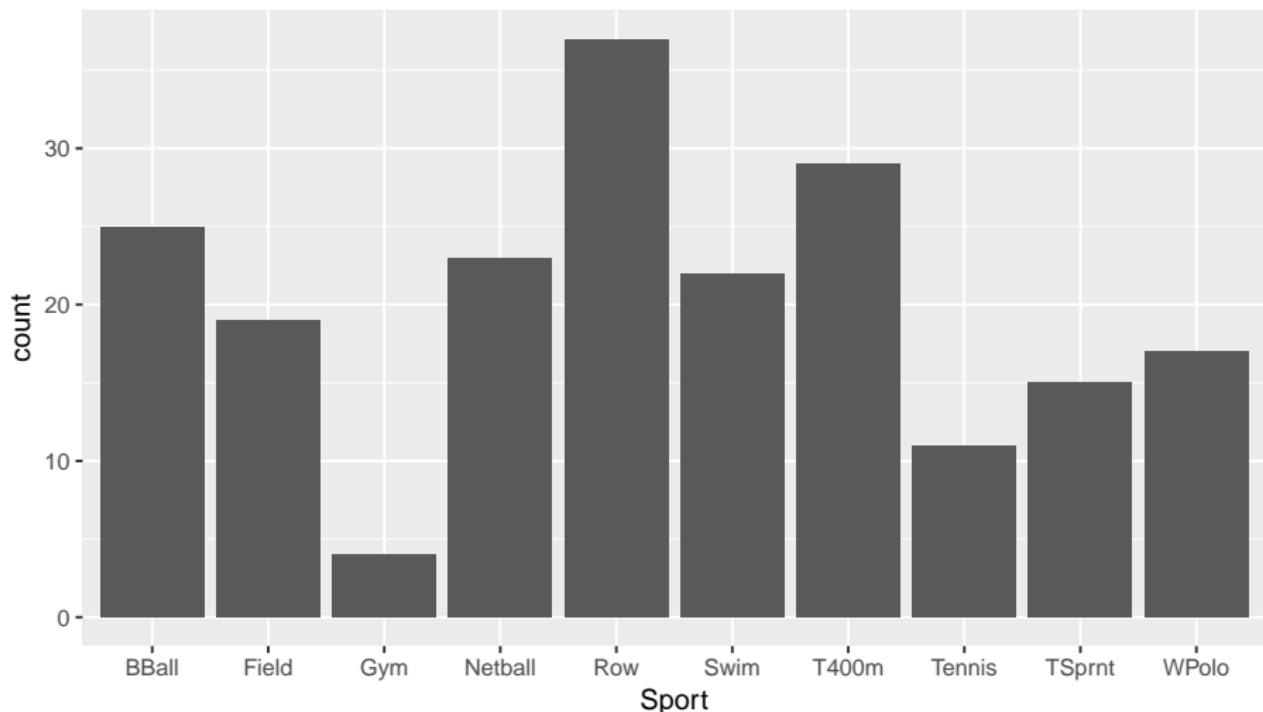
- ▶ in R, **ggplot**
- ▶ in SAS, **proc sgplot**

Use them in different ways to get precise graph we want.

Let's start with bar chart of the sports played by the athletes.

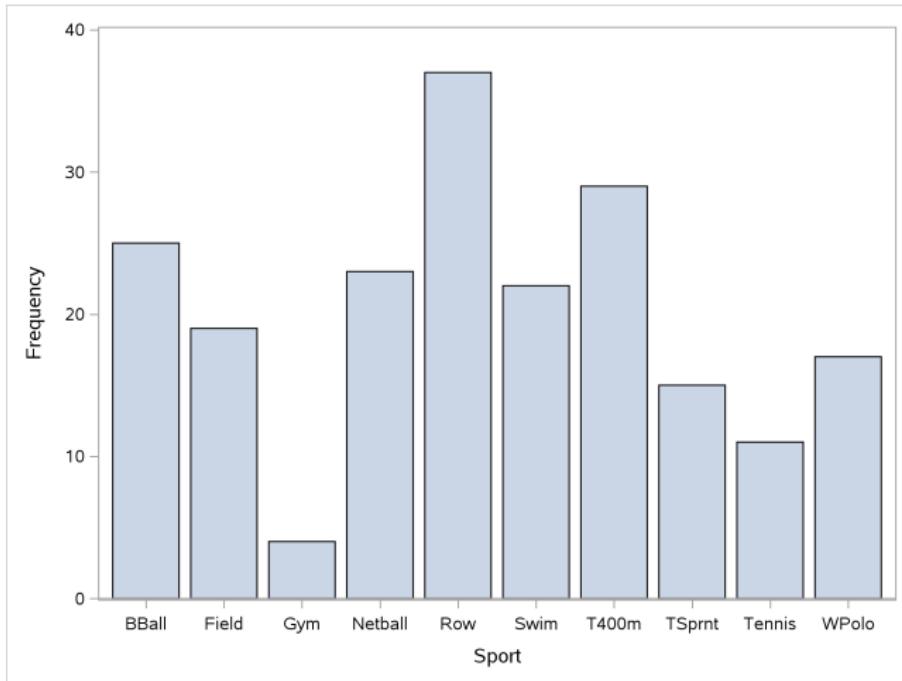
Bar chart in R

```
ggplot(athletes,aes(x=Sport))+geom_bar()
```



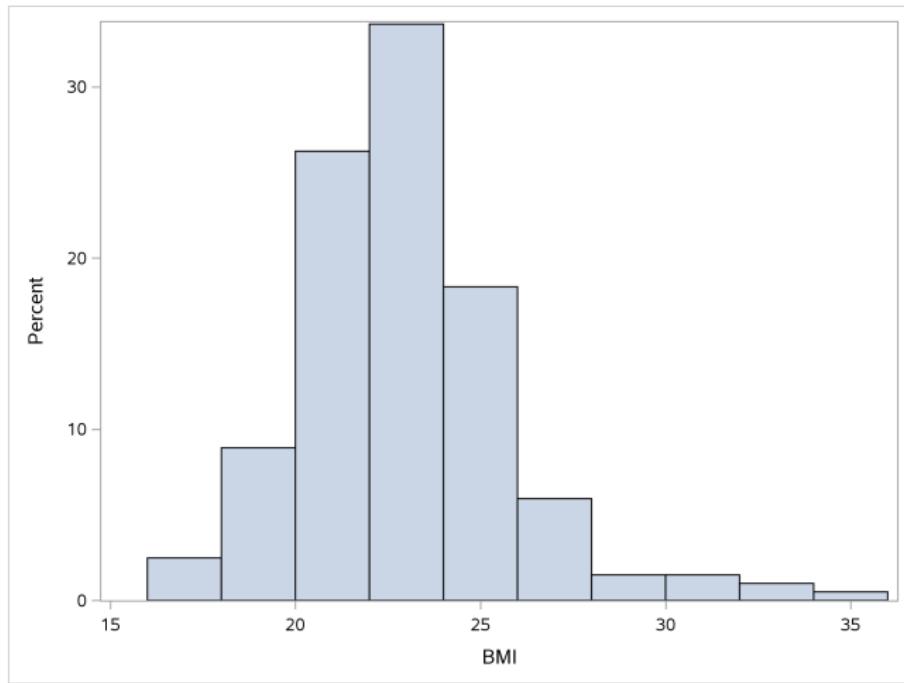
Bar chart in SAS

```
proc sgplot;  
    vbar Sport;
```



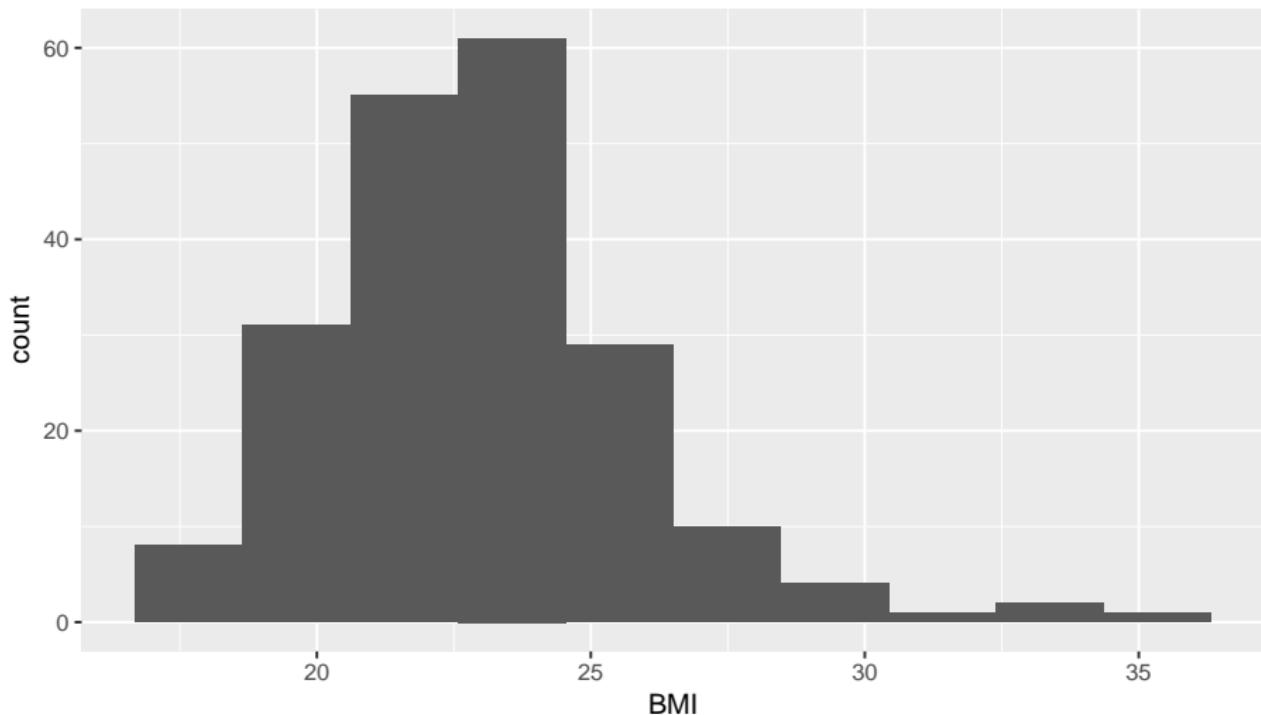
Histogram of body mass index, in SAS

```
proc sgplot;  
    histogram BMI;
```



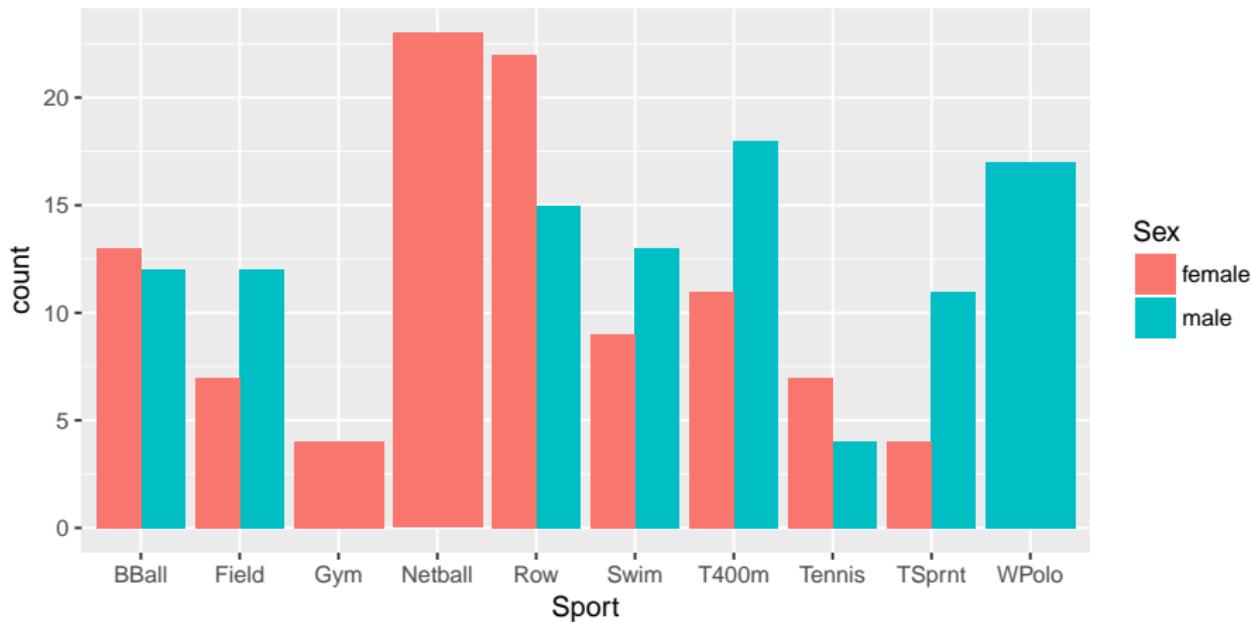
BMI histogram in R

```
ggplot(athletes,aes(x=BMI))+geom_histogram(bins=10)
```



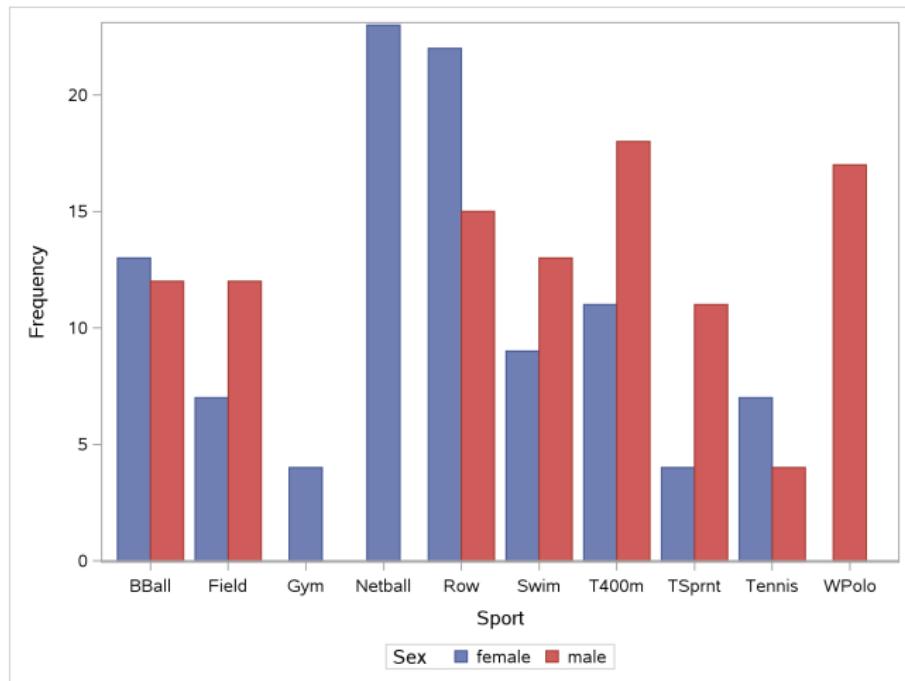
Which sports are played by males and females?

```
ggplot(athletes,aes(x=Sport,fill=Sex))+  
  geom_bar(position="dodge")
```



Grouped bar plot in SAS

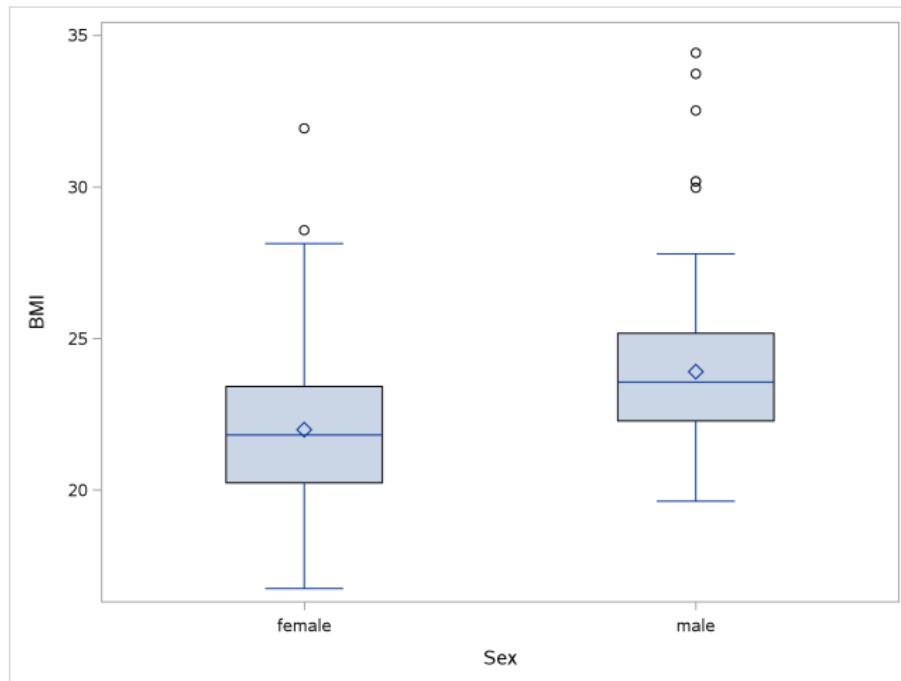
```
proc sgplot;  
    vbar Sport / group=Sex groupdisplay=cluster;
```



BMI by gender

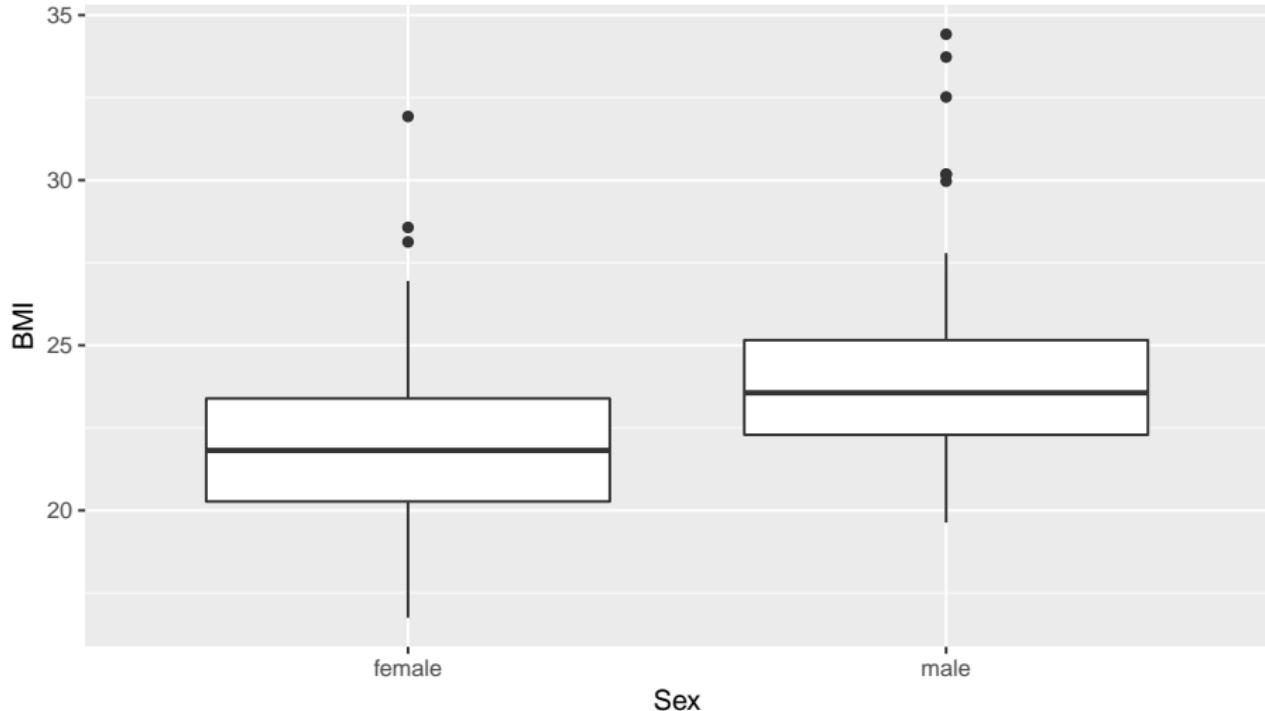
Side-by-side boxplots:

```
proc sgplot;  
    vbox BMI / category=Sex;
```



And in R

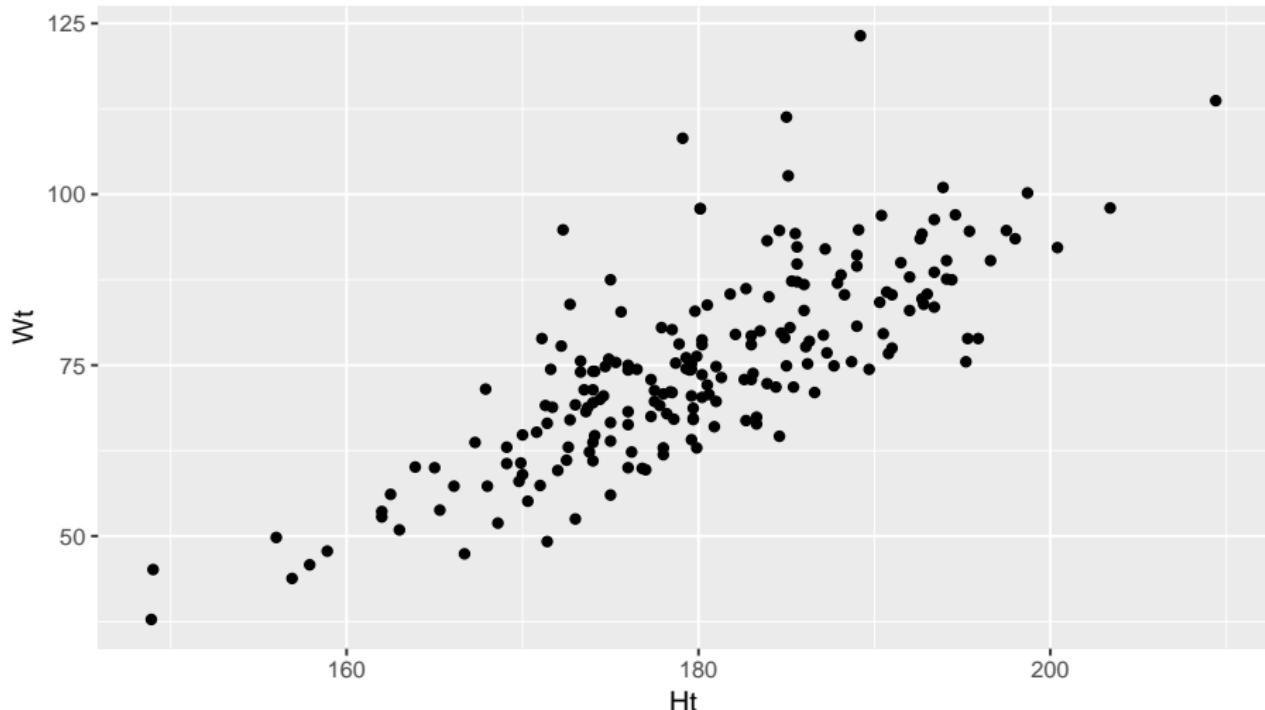
```
ggplot(athletes,aes(x=Sex,y=BMI))+geom_boxplot()
```



Height vs. weight

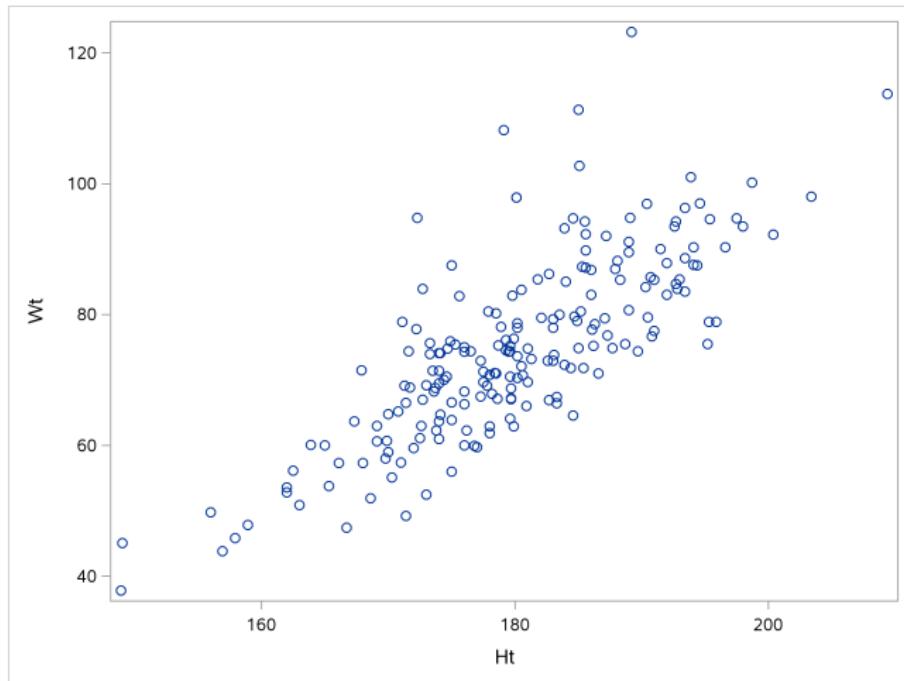
Scatterplot:

```
ggplot(athletes,aes(x=Ht,y=Wt))+geom_point()
```



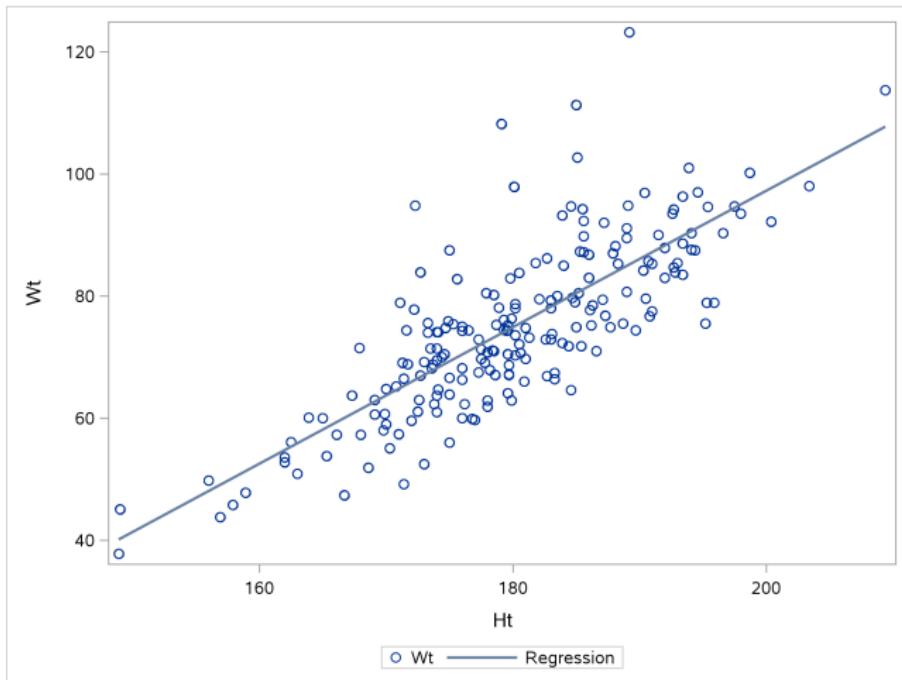
Height vs. weight again

```
proc sgplot;  
    scatter x=Ht y=Wt;
```



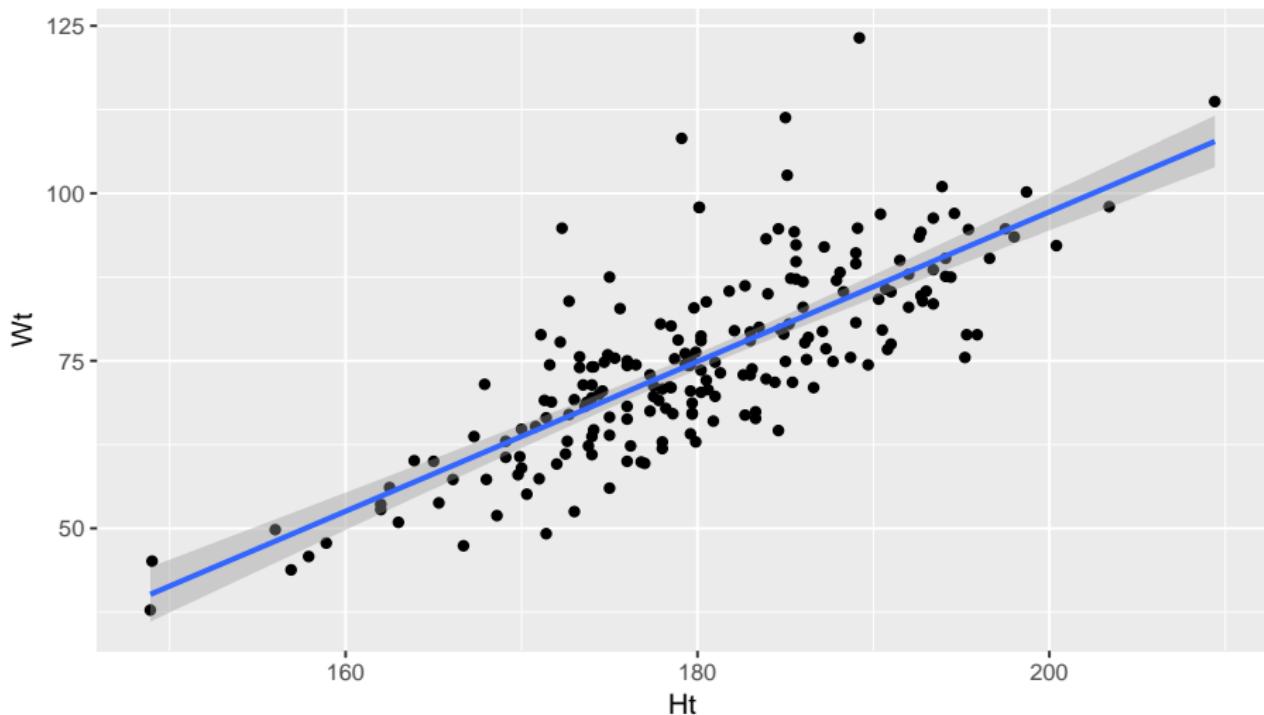
and again, with regression line

```
proc sgplot;  
    scatter x=Ht y=Wt;  
    reg x=Ht y=Wt;
```



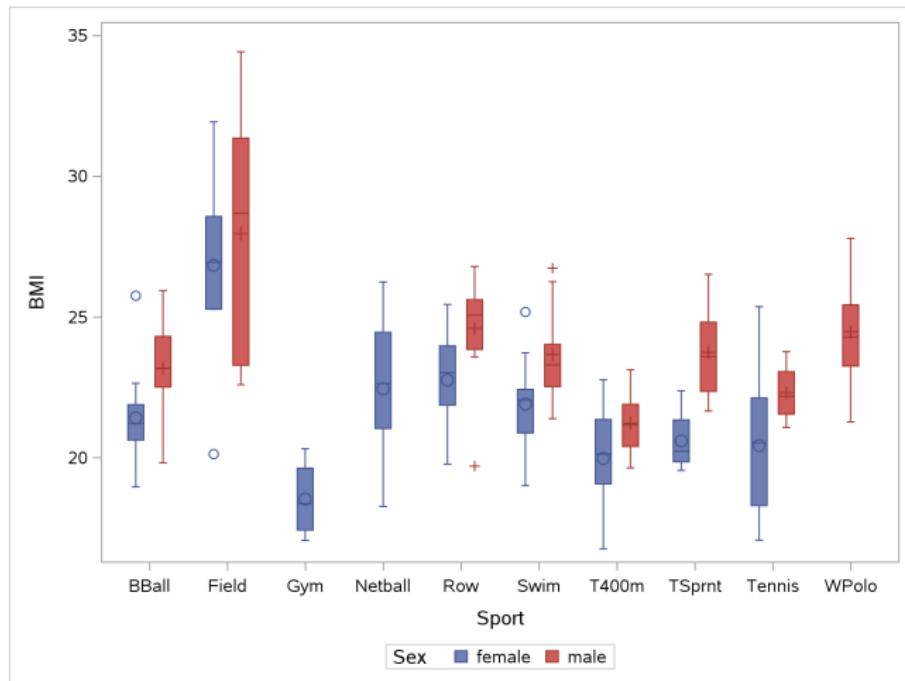
One more time

```
ggplot(athletes,aes(x=Ht,y=Wt))+  
  geom_point() + geom_smooth(method="lm")
```



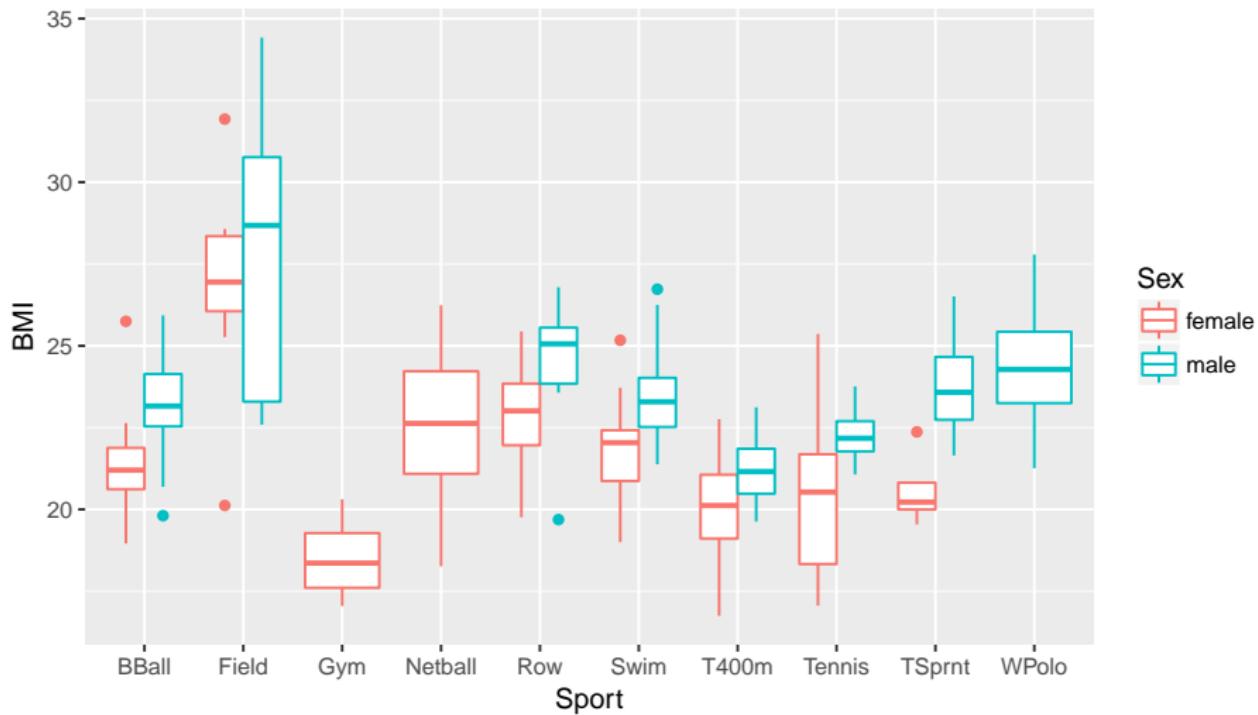
BMI by sport and gender

```
proc sgplot;  
    vbox BMI / group=Sex category=Sport;
```



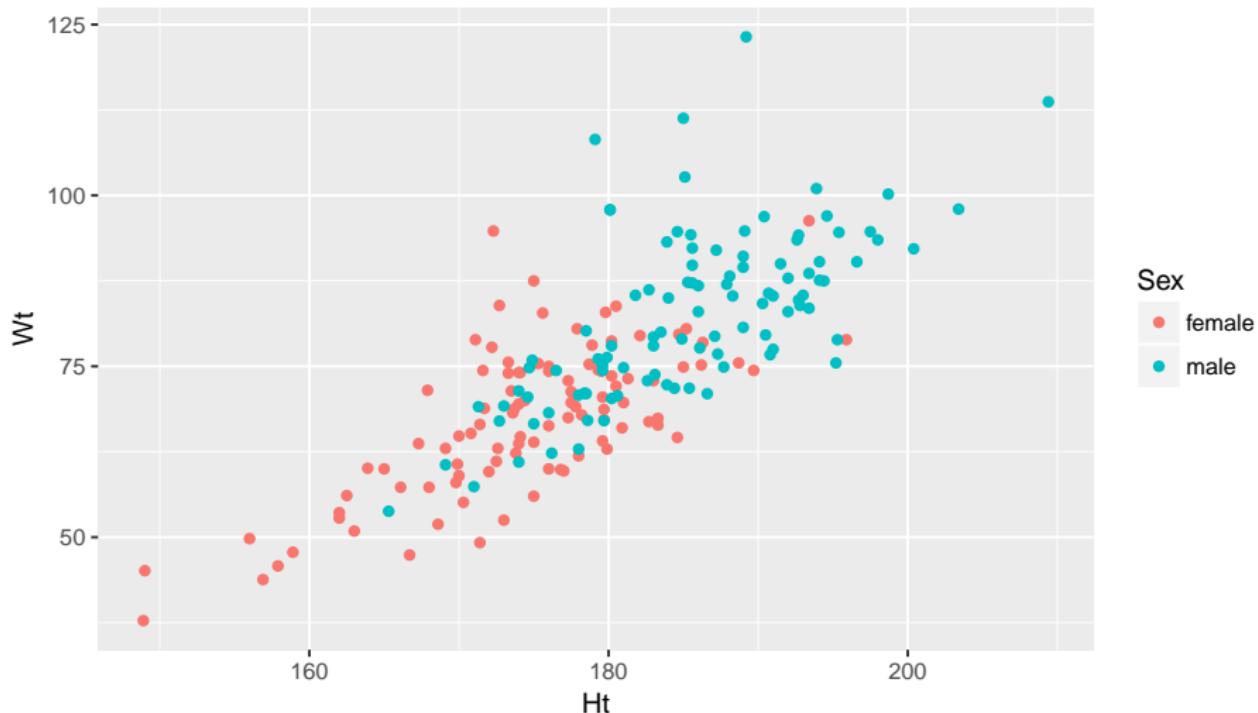
R

```
ggplot(athletes,aes(x=Sport,y=BMI,colour=Sex))+  
  geom_boxplot()
```



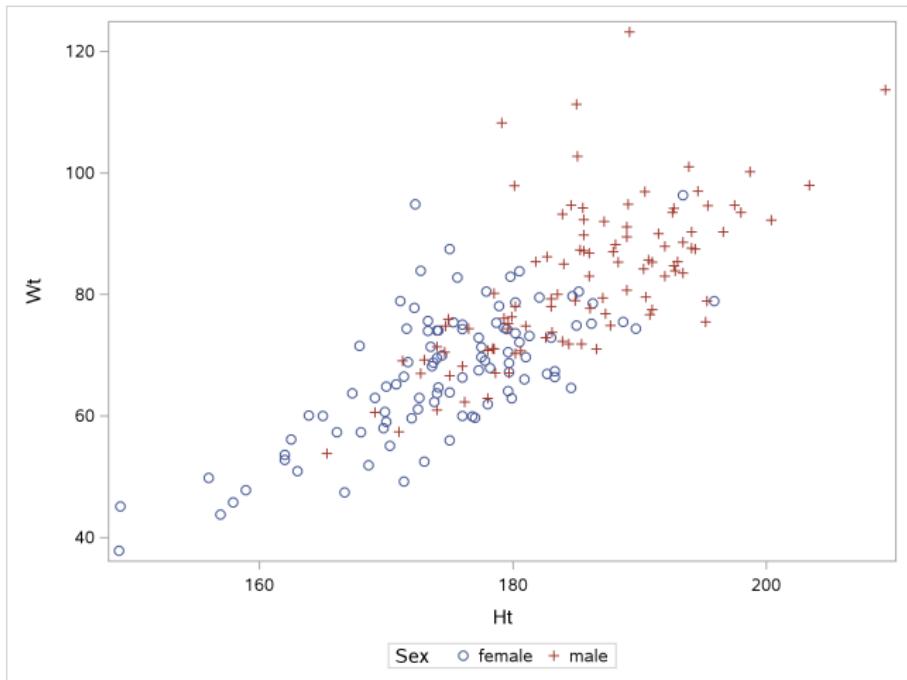
Height and weight by gender

```
ggplot(athletes,aes(x=Ht,y=Wt,colour=Sex))+  
  geom_point()
```



And in SAS

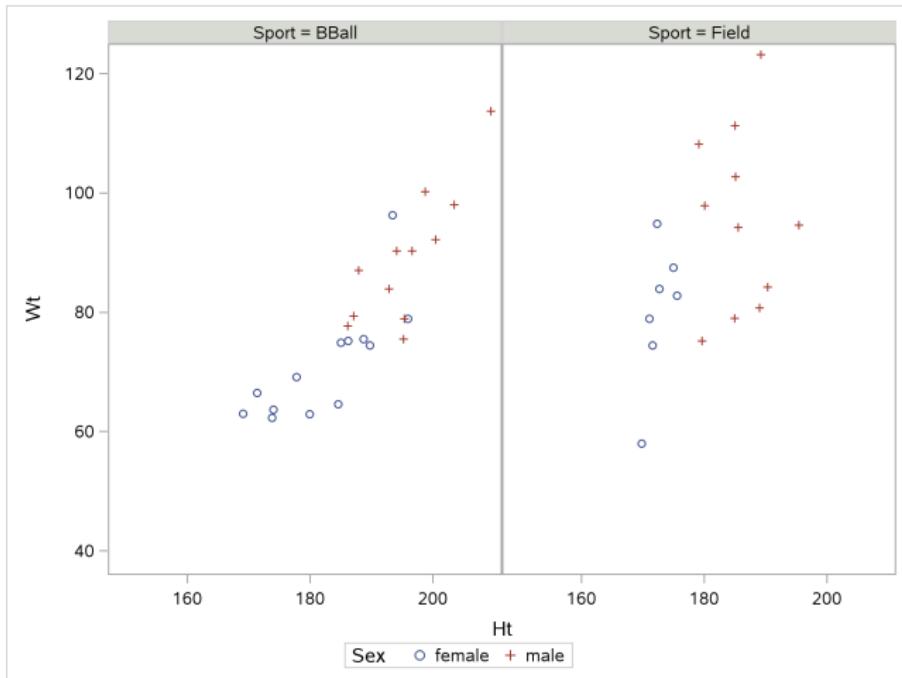
```
proc sgplot;  
  scatter x=Ht y=Wt / group=Sex;
```



Height by weight for each sport

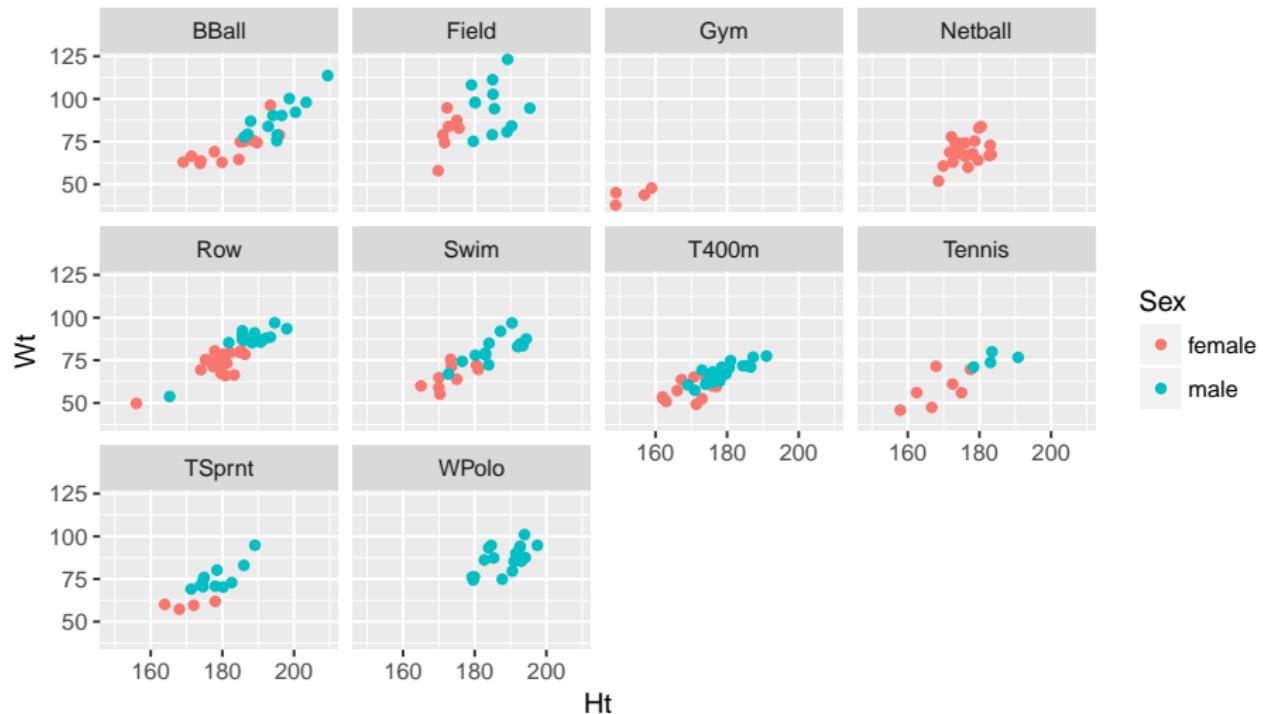
Separate plot for each sport, first two panels here:

```
proc sgpanel;  
  panelby Sport;  
  scatter x=Ht y=Wt / group=Sex;
```



same in R, with facets

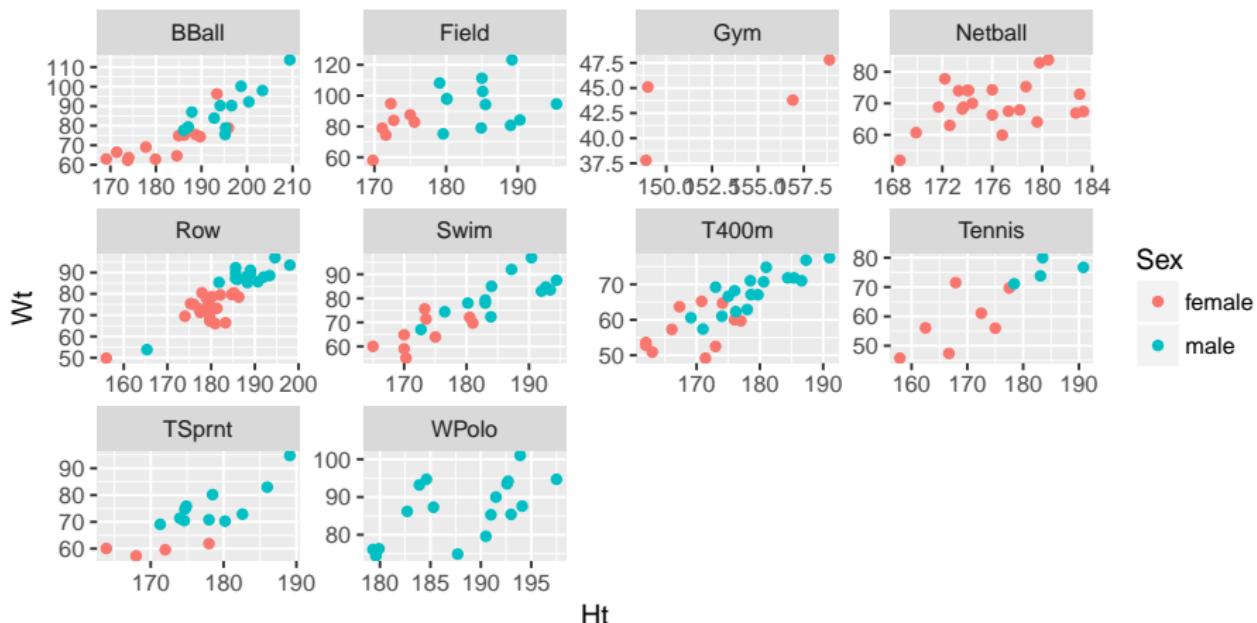
```
ggplot(athletes,aes(x=Ht,y=Wt,colour=Sex))+  
  geom_point() + facet_wrap(~Sport)
```



Filling each facet

Default uses same scale for each facet. To use different scales for each facet, this:

```
ggplot(athletes,aes(x=Ht,y=Wt,colour=Sex))+  
  geom_point() + facet_wrap(~Sport,scales="free")
```



Section 4

R Studio scripts and projects

The console and the script

- ▶ You can always type R commands at the console, next to the >.
- ▶ But if you want to use those commands again, or modify them, you have to find them first (up/down arrow keys)
- ▶ plus you have no record of what you did.
- ▶ Better: create a *script*. File - New File - R Script, opens a window top left, into which you can type commands.
- ▶ To run commands in a script:
 - ▶ move to or click on the line containing what you want to run, then click Run above script window. Runs that line, and moves cursor down to next line.
 - ▶ select one or more lines of code, then click Run: runs all those lines.

Advantages of a script

- ▶ A script can be *saved* and re-opened later.
- ▶ This gives you a record of what you did (“reproducible research”)
- ▶ If something in the script doesn’t work, you can fix just that part without worrying about the part of the script that works
- ▶ If you want to run the same code on different data, or slightly different code, you can use your script as a starting point. (For example, you can keep a bunch of scripts that make example plots that you use often.)

Projects

- ▶ A project is a “container” for code and data that belong together.
- ▶ Goes with a folder on your 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 code scripts for an assignment. Putting data files in a project folder makes it easier to find them.
- ▶ Later, when we learn about R Markdown, see how to write reports that include code. These can go in project folder too.

Comparison with SAS

- ▶ SAS code in SAS Studio *is* a script: some code that is saved in a file and can be reloaded and used again.
- ▶ You can create subfolders under your main folder on SAS Studio. These serve the same purpose as R Studio projects: you can use them as containers for code, data etc.
- ▶ Looking ahead, SAS does not have a nice report-writing mechanism as R Studio does. (It does have one, `statrep`, but that is based on \LaTeX .)

Section 5

More detailed summaries of data

Summarizing data in R

- ▶ Have seen `summary` (5-number summary of each column). But what if we want:
 - ▶ a summary or two of just one column
 - ▶ a count of observations in each category of a categorical variable?
 - ▶ summaries by group
 - ▶ a different summary of all columns (eg. SD)
- ▶ To do this, meet **pipe** operator `%>%`. This takes input data frame, does something do it, and outputs result.
- ▶ Output from a pipe can be used as input to something else, so can have a sequence of pipes.
- ▶ Summaries include: `mean`, `median`, `min`, `max`, `sd`, `IQR`, `quantile` (for obtaining quartiles or any percentile), `n` (for counting observations).
- ▶ Use our Australian athletes data again.

Summarizing one column

- ▶ Like this, for example the mean height:

```
athletes %>% summarize(m=mean(Ht))  
  
## # A tibble: 1 x 1  
##       m  
##   <dbl>  
## 1 180.104
```

or to get mean and SD of BMI:

```
athletes %>% summarize(m=mean(BMI), s=sd(BMI))  
  
## # A tibble: 1 x 2  
##       m           s  
##   <dbl>     <dbl>  
## 1 22.95589 2.863933
```

Quartiles

- ▶ `quantile` calculates percentiles, so we want the 25th and 75th percentiles:

```
athletes %>% summarize( Q1=quantile(Wt,0.25),  
                           Q3=quantile(Wt,0.75))  
  
## # A tibble: 1 x 2  
##       Q1     Q3  
##   <dbl> <dbl>  
## 1 66.525 84.125
```

Counting how many

for example, number of athletes in each sport:

```
athletes %>% count(Sport)
```

```
## # A tibble: 10 x 2
##       Sport     n
##   <chr> <int>
## 1 BBall    25
## 2 Field    19
## 3 Gym      4
## 4 Netball   23
## 5 Row      37
## 6 Swim     22
## 7 T400m    29
## 8 Tennis    11
## 9 TSprnt   15
## 10 WPolo   17
```

Another way (which will make sense in a moment):

```
athletes %>% group_by(Sport) %>%
  summarize(count=n())
```

```
## # A tibble: 10 x 2
##       Sport count
##   <chr> <int>
## 1 BBall    25
## 2 Field    19
## 3 Gym      4
## 4 Netball   23
## 5 Row      37
## 6 Swim     22
## 7 T400m    29
## 8 Tennis    11
## 9 TSprnt   15
## 10 WPolo   17
```

Summaries by group

- ▶ Might want separate summaries for each “group”, eg. mean and SD of height for males and females. Strategy is `group_by` (to define the groups) and then `summarize`:

```
athletes %>% group_by(Sex) %>%  
  summarize(m=mean(Ht), s=sd(Ht))  
  
## # A tibble: 2 x 3  
##       Sex     m     s  
##   <chr>  <dbl>  <dbl>  
## 1 female 174.5940 8.242203  
## 2 male   185.5059 7.903487
```

- ▶ This explains second variation on counting within group: “within each sport, how many athletes were there?”

Summarizing several columns

- ▶ Standard deviation of each (numeric) column:

```
athletes %>% summarize_if(is.numeric, funs(sd))

## # A tibble: 1 x 11
##       RCC      WCC      Hc      Hg     Ferr      BMI      SSF
##   <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 0.4579764 1.800549 3.662989 1.362451 47.50124 2.863933 32.56533
## # ... with 3 more variables: LBM <dbl>, Ht <dbl>, Wt <dbl>
```

- ▶ Median and IQR of all columns whose name starts with H:

```
athletes %>% summarize_at(vars(starts_with("H")),
  funs(median, IQR))

## # A tibble: 1 x 6
##   Hc_median Hg_median Ht_median Hc_IQR Hg_IQR Ht_IQR
##       <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1      43.5     14.7    179.7    4.975    2.075   12.175
```

Summarizing data in SAS

- ▶ Already saw proc means to find means, SDs and sample sizes.
- ▶ proc means will also calculate means of only some variables or by group.
- ▶ Also, proc means can calculate other statistics (by group if desired), despite its name.
- ▶ SAS names for other statistics: mean, median, stddev (SD), qrange (IQR), Q1, Q3 (quartiles).

Specifying summaries, variables and groups

- ▶ To specify which summaries to calculate, list them on the proc means line.
- ▶ To specify which variables to calculate summaries for, use a line starting with var.
- ▶ To specify which groups to calculate for, use a line starting with class and the name of the grouping variable.
- ▶ Examples over.

Quartiles of athlete weight

```
proc means Q1 Q3 Qrange;  
var Wt;
```

The MEANS Procedure

Analysis Variable : Wt

Lower Quartile	Upper Quartile	Quartile Range
66.5000000	84.2000000	17.7000000

Mean and SD of height by gender

- ▶ Thus:

```
proc means mean stddev;  
    var Ht;  
    class Sex;
```

The MEANS Procedure

Analysis Variable : Ht

Sex	N	Obs	Mean	Std Dev
<hr/>				
female	100	174.5940000	8.2422026	
male	102	185.5058824	7.9034874	

How many athletes from each sport?

- ▶ Have to pick a variable to count observations of (though it doesn't matter):

```
proc means n;  
  var BMI;  
  class Sport;
```

- ▶ Results over.

Results

The MEANS Procedure

Analysis Variable : BMI

Sport	Obs	N
<hr/>		
BBall	25	25
Field	19	19
Gym	4	4
Netball	23	23
Row	37	37
Swim	22	22
T400m	29	29
TSprnt	15	15
Tennis	11	11
WPolo	17	17

A perhaps better way to count

```
proc freq;  
    tables Sport;
```

The FREQ Procedure

Sport	Frequency	Percent	Cumulative Frequency	Cumulative Percent
BBall	25	12.38	25	12.38
Field	19	9.41	44	21.78
Gym	4	1.98	48	23.76
Netball	23	11.39	71	35.15
Row	37	18.32	108	53.47
Swim	22	10.89	130	64.36
T400m	29	14.36	159	78.71
TSprnt	15	7.43	174	86.14
Tennis	11	5.45	185	91.58
WPolo	17	8.42	202	100.00

SD of all the (numerical) columns

- ▶ Just don't specify a var or a class:

```
proc means stddev;
```

The MEANS Procedure

Variable	Std Dev

RCC	0.4579764
WCC	1.8005490
Hc	3.6629894
Hg	1.3624515
Ferr	47.5012388
BMI	2.8639328
SSF	32.5653330
_Bfat	6.1898260
LBM	13.0701972
Ht	9.7344945
Wt	13.9255740

Section 6

Inference

Statistical Inference and Science

- ▶ Previously: *descriptive statistics*. “Here are data; what do they say?”.
- ▶ May need to *take some action* based on information in data.
- ▶ Or want to *generalize* beyond data (sample) to larger world (population).
- ▶ Science: first guess about how world works.
- ▶ Then collect data, by sampling.
- ▶ Is guess correct (based on data) for whole world, or not?

Sample data are imperfect

- ▶ Sample data never entirely represent what you're observing.
- ▶ There is always random error present.
- ▶ Thus you can never be entirely certain about your conclusions.
- ▶ The Toronto Blue Jays' average home attendance in part of 2015 season was 25,070 (up to May 27 2015, from baseball-reference.com).
- ▶ Does that mean the attendance at every game was exactly 25,070? Certainly not. Actual attendance depends on many things, eg.:
 - ▶ how well the Jays are playing
 - ▶ the opposition
 - ▶ day of week
 - ▶ weather
 - ▶ random chance

Reading the attendances

...as a .csv file:

```
jays=read_csv("jays15-home.csv")  
  
## Parsed with column specification:  
## cols(  
##   .default = col_character(),  
##   row = col_integer(),  
##   game = col_integer(),  
##   runs = col_integer(),  
##   Oppruns = col_integer(),  
##   innings = col_integer(),  
##   position = col_integer(),  
##   'game time' = col_time(format = ""),  
##   attendance = col_integer()  
## )  
  
## See spec(...) for full column specifications.
```

Taking a look

jays

```
## # A tibble: 25 x 21
##       row   game           date      box team venue  opp result  runs
##   <int> <int>        <chr>     <chr> <chr> <chr> <chr> <chr> <int>
## 1     82     7 Monday, Apr 13 boxescore    TOR <NA>  TBR     L  1
## 2     83     8 Tuesday, Apr 14 boxescore    TOR <NA>  TBR     L  2
## 3     84     9 Wednesday, Apr 15 boxescore    TOR <NA>  TBR     W 12
## 4     85    10 Thursday, Apr 16 boxescore    TOR <NA>  TBR     L  2
## 5     86    11 Friday, Apr 17 boxescore    TOR <NA>  ATL     L  7
## 6     87    12 Saturday, Apr 18 boxescore    TOR <NA>  ATL  W-wo  6
## 7     88    13 Sunday, Apr 19 boxescore    TOR <NA>  ATL     L  2
## 8     89    14 Tuesday, Apr 21 boxescore    TOR <NA>  BAL     W 13
## 9     90    15 Wednesday, Apr 22 boxescore    TOR <NA>  BAL     W  4
## 10    91    16 Thursday, Apr 23 boxescore    TOR <NA>  BAL     W  7
## # ... with 15 more rows, and 12 more variables: Oppruns <int>,
## #   innings <int>, wl <chr>, position <int>, gb <chr>, winner <chr>,
## #   loser <chr>, save <chr>, `game time` <time>, Daynight <chr>,
## #   attendance <int>, streak <chr>
```

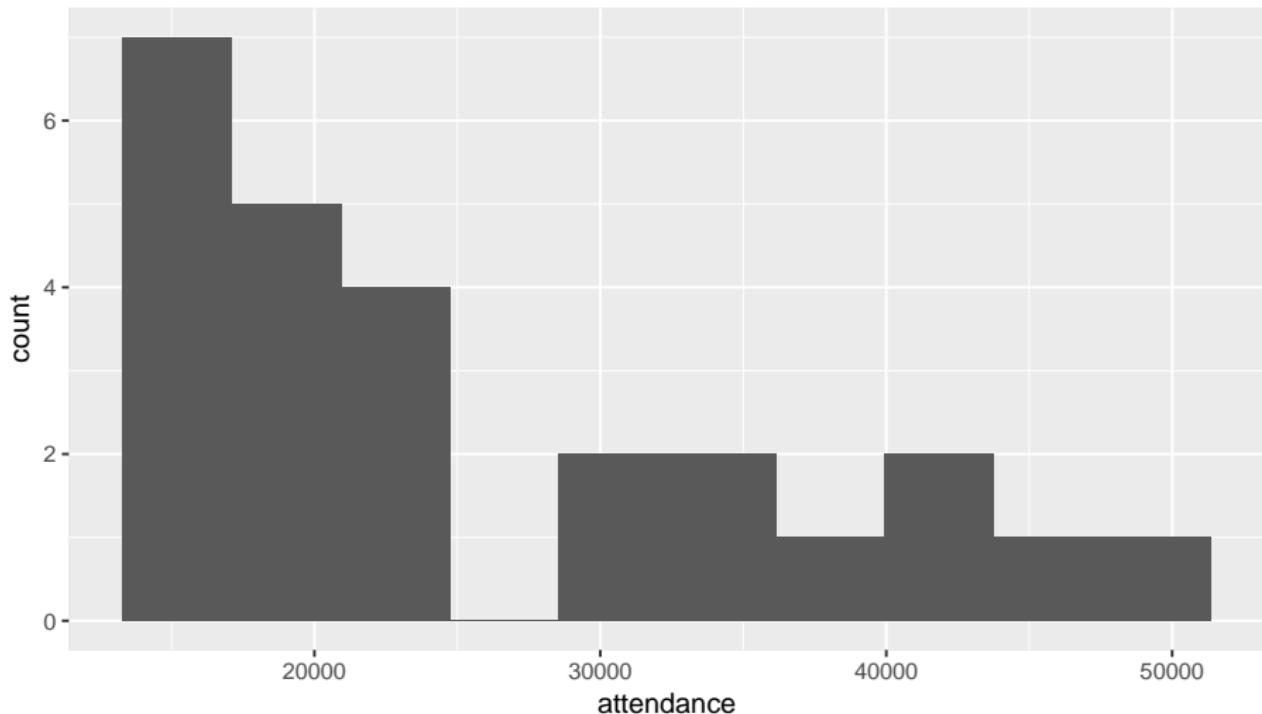
Another wav

glimpse(jays)

```
## Observations: 25
## Variables: 21
## $ row      <int> 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, ...
## $ game     <int> 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 27, 28, 29, 30...
## $ date     <chr> "Monday, Apr 13", "Tuesday, Apr 14", "Wednesday, Ap...
## $ box       <chr> "boxscore", "boxscore", "boxscore", "boxscore", "bo...
## $ team      <chr> "TOR", "TOR", "TOR", "TOR", "TOR", "TOR", "TOR", "T...
## $ venue     <chr> NA, ...
## $ opp       <chr> "TBR", "TBR", "TBR", "TBR", "ATL", "ATL", "ATL", ...
## $ result    <chr> "L", "L", "W", "L", "W-wo", "L", "W", "W", "W"...
## $ runs      <int> 1, 2, 12, 2, 7, 6, 2, 13, 4, 7, 3, 3, 5, 7, 7, 3, 1...
## $ Oppruns   <int> 2, 3, 7, 4, 8, 5, 5, 6, 2, 6, 1, 6, 1, 0, 1, 6, 6, ...
## $ innings   <int> NA, NA, NA, NA, NA, 10, NA, NA, NA, NA, NA, NA, ...
## $ wl        <chr> "4-3", "4-4", "5-4", "5-5", "5-6", "6-6", "6-7", "7...
## $ position  <int> 2, 3, 2, 4, 4, 3, 4, 2, 2, 1, 4, 5, 3, 3, 3, 5, ...
## $ gb        <chr> "1", "2", "1", "1.5", "2.5", "1.5", "1.5", "2", "1"...
## $ winner    <chr> "Odorizzi", "Geltz", "Buehrle", "Archer", "Martin", ...
## $ loser     <chr> "Dickey", "Castro", "Ramirez", "Sanchez", "Cecil", ...
## $ save      <chr> "Boxberger", "Jepsen", NA, "Boxberger", "Grilli", N...
## $ game time <time> 02:30:00, 03:06:00, 03:02:00, 03:00:00, 03:09:00, ...
## $ Daynight  <chr> "N", "N", "N", "N", "D", "D", "N", "N", "N", "N", ...
## $ attendance <int> 48414, 17264, 15086, 14433, 21397, 34743, 44794, 14...
## $ streak    <chr> "-", "--", "+", "-", "--", "+", "-", "+", "++", "++...
```

Attendance histogram

```
ggplot(jays,aes(x=attendance))+geom_histogram(bins=10)
```



Comments

- ▶ Attendances have substantial variability, ranging from just over 10,000 to around 50,000.
- ▶ Distribution somewhat skewed to right (but no outliers).
- ▶ These are a sample of “all possible games” (or maybe “all possible games played in April and May”). What can we say about mean attendance in all possible games based on this evidence?
- ▶ Think about:
 - ▶ Confidence interval
 - ▶ Hypothesis test.

Getting CI for mean attendance

- ▶ t.test function does CI and test. Look at CI first:

```
t.test(jays$attendance)

##
##  One Sample t-test
##
## data: jays$attendance
## t = 11.389, df = 24, p-value = 3.661e-11
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
## 20526.82 29613.50
## sample estimates:
## mean of x
## 25070.16
```

- ▶ From 20,500 to 29,600.

Or, 90% CI

- ▶ by including a value for `conf.level`:

```
t.test(jays$attendance, conf.level=0.90)

##
##  One Sample t-test
##
## data: jays$attendance
## t = 11.389, df = 24, p-value = 3.661e-11
## alternative hypothesis: true mean is not equal to 0
## 90 percent confidence interval:
## 21303.93 28836.39
## sample estimates:
## mean of x
## 25070.16
```

- ▶ From 21,300 to 28,800. (Shorter, as it should be.)

Comments

- ▶ Need to say “column attendance within data frame jays” using \$.
- ▶ 95% CI from about 20,000 to about 30,000.
- ▶ Not estimating mean attendance well at all!
- ▶ Generally want confidence interval to be *shorter*, which happens if:
 - ▶ SD smaller
 - ▶ sample size *bigger*
 - ▶ confidence level smaller
- ▶ Last one is a cheat, really, since reducing confidence level increases chance that interval won’t contain pop. mean at all!

Another way to access data frame columns

```
with(jays,t.test(attendance))

##
## One Sample t-test
##
## data: attendance
## t = 11.389, df = 24, p-value = 3.661e-11
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
## 20526.82 29613.50
## sample estimates:
## mean of x
## 25070.16
```

Hypothesis test

- ▶ CI answers question “what is the mean?”
- ▶ Might have a value μ in mind for the mean, and question “Is the mean equal to μ , or not?”
- ▶ For example, 2014 average attendance was 29,327.
- ▶ Second question answered by **hypothesis test**.
- ▶ Value being assessed goes in **null hypothesis**: here, $H_0 : \mu = 29,327$.
- ▶ **Alternative hypothesis** says how null might be wrong, eg.
 $H_a : \mu \neq 29,327$.
- ▶ Assess evidence *against null*. If that evidence strong enough, *reject null hypothesis*; if not, *fail to reject null hypothesis* (sometimes *retain null*).
- ▶ Note asymmetry between null and alternative, and utter absence of word “accept”.

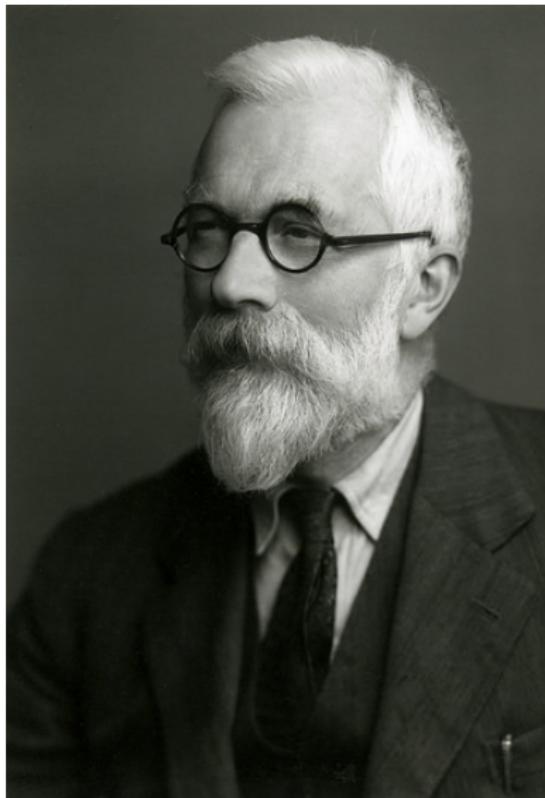
α and errors

- ▶ Hypothesis test ends with *decision*:
 - ▶ reject null hypothesis
 - ▶ do not reject null hypothesis.
- ▶ but decision may be *wrong*:

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

- ▶ Either type of error is bad, but for now focus on controlling Type I error: write $\alpha = P(\text{type I error})$, and devise test so that α *small*, typically 0.05.
- ▶ That is, **if null hypothesis true**, have only small chance to reject it (which would be a mistake).
- ▶ Worry about type II errors later (when we consider power of test).

Why 0.05? This man.



Responsible for:

- ▶ analysis of variance
- ▶ Fisher information
- ▶ Linear discriminant analysis
- ▶ Fisher's z-transformation
- ▶ Fisher-Yates shuffle
- ▶ Behrens-Fisher problem

Sir Ronald A. Fisher, 1890–1962.

Why 0.05? (2)

- ▶ From *The Arrangement of Field Experiments* (1926):

the line at about the level at which we can say: "Either there is something in the treatment, or a coincidence has occurred such as does not occur more than once in twenty trials." This level, which we may call the 5 per cent. point, would be indicated, though very roughly, by the greatest chance deviation observed in twenty successive trials. To

- ▶ and

If one in twenty does not seem high enough odds, we may, if we prefer it, draw the line at one in fifty (the 2 per cent. point), or one in a hundred (the 1 per cent. point). Personally, the writer prefers to set a low standard of significance at the 5 per cent. point, and ignore entirely all results which fail to reach this level. A scientific fact should be regarded as experimentally established only if a properly designed experiment rarely fails to give this level of significance. The very high

Test statistic: going from data to decision

- ▶ “How far away from null hypothesis is data?”
- ▶ In testing for mean, statistic is

$$t = \frac{\bar{x} - \mu}{s/\sqrt{n}}$$

- ▶ and for our baseball attendance data

```
t.stat=(25070-29327)/(11000/sqrt(25))
```

```
t.stat
```

```
## [1] -1.935
```

P-value

- ▶ The probability of observing a test statistic value *as extreme or more extreme than that observed, if the null hypothesis is true.*
- ▶ “More extreme” depends on H_a : count both sides if two-sided, count only the proper side if one-sided.
- ▶ Our H_a was $H_a : \mu \neq 29,327$, two-sided.

t-table

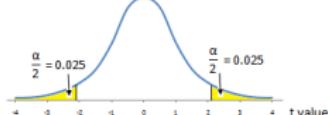
Student's t Distribution Table

For example, the *t* value for

18 degrees of freedom

is 2.101 for 95% confidence

interval (2-Tail $\alpha = 0.05$).



	90%	95%	97.5%	99%	99.5%	99.95%	1-Tail Confidence Level
	80%	90%	95%	98%	99%	99.9%	2-Tail Confidence Level
df	0.100	0.050	0.025	0.010	0.005	0.0005	1-Tail Alpha
0.20	0.10	0.05	0.02	0.01	0.001	0.0005	2-Tail Alpha

df	0.20	0.10	0.05	0.02	0.01	0.001	2-Tail Alpha
1	3.0777	6.3138	12.7062	31.8205	63.6567	636.6192	
2	1.8856	2.9200	4.3027	6.9546	9.9248	31.5991	
3	1.6377	2.3534	3.1824	4.5407	5.8409	12.9240	
4	1.5332	2.1318	2.7764	3.7469	4.6041	8.6103	
5	1.4759	2.0150	2.5706	3.3649	4.0321	6.8688	
6	1.4398	1.9432	2.4469	3.1427	3.7074	5.9588	
7	1.4149	1.8946	2.3646	2.9980	3.4995	5.4079	
8	1.3968	1.8595	2.3060	2.8965	3.3554	5.0413	
9	1.3830	1.8331	2.2622	2.8214	3.2498	4.7809	
10	1.3722	1.8125	2.2281	2.7638	3.1693	4.5869	
11	1.3634	1.7959	2.2010	2.7181	3.1058	4.4370	
12	1.3562	1.7823	2.1788	2.6810	3.0545	4.3178	
13	1.3502	1.7709	2.1604	2.6503	3.0123	4.2208	
14	1.3450	1.7613	2.1448	2.6245	2.9768	4.1405	
15	1.3406	1.7531	2.1314	2.6025	2.9467	4.0728	
16	1.3368	1.7459	2.1199	2.5835	2.9208	4.0150	
17	1.3334	1.7396	2.1098	2.5669	2.8982	3.9651	
18	1.3304	1.7341	2.1009	2.5524	2.8784	3.9216	
19	1.3277	1.7291	2.0930	2.5395	2.8609	3.8834	
20	1.3253	1.7247	2.0860	2.5280	2.8453	3.8495	
21	1.3232	1.7207	2.0796	2.5176	2.8314	3.8193	
22	1.3212	1.7171	2.0739	2.5083	2.8188	3.7921	
23	1.3195	1.7139	2.0687	2.4999	2.8073	3.7676	
24	1.3178	1.7109	2.0639	2.4922	2.7969	3.7454	
25	1.3163	1.7081	2.0595	2.4851	2.7874	3.7251	
26	1.3150	1.7056	2.0555	2.4786	2.7787	3.7066	
27	1.3137	1.7033	2.0518	2.4727	2.7707	3.6896	
28	1.3125	1.7011	2.0484	2.4671	2.7633	3.6739	
29	1.3114	1.6991	2.0452	2.4620	2.7564	3.6594	
30	1.3104	1.6972	2.0423	2.4573	2.7500	3.6460	

- ▶ $n = 25$ so df is $25 - 1 = 24$.
- ▶ Look up 1.935 not -1.935 .
- ▶ $1.71 < 1.935 < 2.06$
- ▶ P-value between 0.05 and 0.10
- ▶ P-value not less than 0.05: do not reject null.
- ▶ No evidence of change in mean attendance.

Or, again, using `t.test`:

```
t.test(jays$attendance, mu=29327)

##
##  One Sample t-test
##
## data: jays$attendance
## t = -1.9338, df = 24, p-value = 0.06502
## alternative hypothesis: true mean is not equal to 29327
## 95 percent confidence interval:
##  20526.82 29613.50
## sample estimates:
## mean of x
## 25070.16
```

- ▶ See test statistic -1.93 , P-value 0.065 .
- ▶ Again, do not reject null: conclusion same.

And in SAS

```
proc import  
  datafile='/home/ken/jays15-home.csv'  
  dbms=csv  
  out=jays  
  replace;  
getnames=yes;
```

Checking what I read in

- ▶ Especially important in SAS:

```
proc print data=jays(obs=6);
```

Obs	row	game date	box	team	venue	opp	result
1	82	7 Monday, Apr 13	boxscore	TOR		TBR	L
2	83	8 Tuesday, Apr 14	boxscore	TOR		TBR	L
3	84	9 Wednesday, Apr 15	boxscore	TOR		TBR	W
4	85	10 Thursday, Apr 16	boxscore	TOR		TBR	L
5	86	11 Friday, Apr 17	boxscore	TOR		ATL	L
6	87	12 Saturday, Apr 18	boxscore	TOR		ATL	W-wo

Obs	runs	Oppruns	innings	wl	position	gb	winner
1	1	2	.	4-3	2	1	Odorizzi
2	2	3	.	4-4	3	2	Geltz
3	12	7	.	5-4	2	1	Buehrle
4	2	4	.	5-5	4	1.5	Archer
5	7	8	.	5-6	4	2.5	Martin
6	6	5	10	6-6	3	1.5	Cecil

Obs	loser	save	game_time	Daynight	attendance	streak
1	Dickey	Boxberger	2:30:00.000	N	48414	-
2	Castro	Jepsen	3:06:00.000	N	17264	--
3	Ramirez		3:02:00.000	N	15086	+
4	Sanchez	Boxberger	3:00:00.000	N	14433	-
5	Cecil	Grilli	3:09:00.000	N	21397	--
6	Marimon		2:41:00.000	D	34743	+

Doing the *t*-test

```
proc ttest h0=29327;  
    var attendance;
```

N	Mean	Std Dev	Std Err	Minimum	Maximum
25	25070.2	11006.7	2201.3	14184.0	48414.0
	Mean	95% CL Mean	Std Dev	95% CL Std Dev	
	25070.2	20526.8 29613.5	11006.7	8594.3	15312.0
		DF	t Value	Pr > t	
		24	-1.93	0.0650	

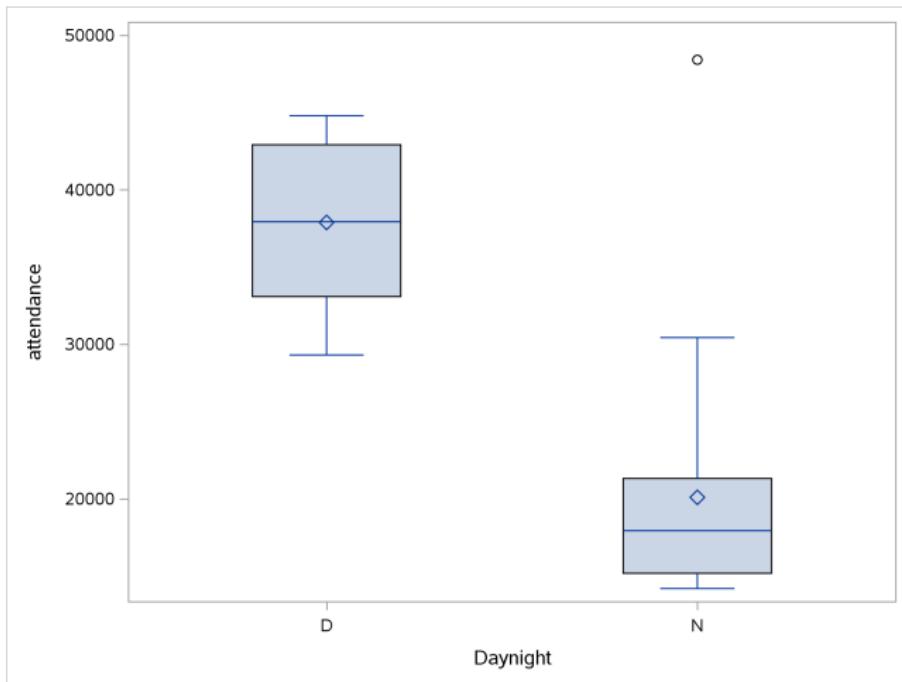
Same CI (20527 to 29614) as R, also same P-value 0.0650.

Day and night games

- ▶ `daynight` is D for a day game and N for a night game. How do attendances compare for these?

```
proc sgplot;  
    vbox attendance / category=daynight;
```

The boxplot



Comments

- ▶ Attendances on average *much* higher for day games than night ones.
- ▶ Why?
- ▶ What is that upper outlier in the night games?

Another example: learning to read

- ▶ You devised new method for teaching children to read.
- ▶ Guess it will be more effective than current methods.
- ▶ To support this guess, collect data.
- ▶ Want to generalize to “all children in Canada”.
- ▶ So take random sample of all children in Canada.
- ▶ Or, argue that sample you actually have is “typical” of all children in Canada.
- ▶ Randomization: whether or not a child in sample or not has nothing to do with anything else about that child.
- ▶ Aside: if your new method good for teaching *struggling* children to read, then “all kids” is “all kids having trouble learning to read”, and you take a sample of *those*.

The data (some), in SAS

- ▶ Data in file drp.txt with header line, group then reading test score, separated by *space*:

```
proc import  
  datafile='/home/ken/drp.txt'  
  dbms=dlm  
  out=reading  
  replace;  
  delimiter=' ' ;  
  getnames=yes;  
  
  proc print;
```

The data, some, tiny

Obs	group	score
-----	-------	-------

1	t	24
---	---	----

2	t	61
---	---	----

3	t	59
---	---	----

4	t	46
---	---	----

5	t	43
---	---	----

6	t	44
---	---	----

7	t	52
---	---	----

8	t	43
---	---	----

9	t	58
---	---	----

10	t	67
----	---	----

11	t	62
----	---	----

12	t	57
----	---	----

13	t	71
----	---	----

14	t	49
----	---	----

15	t	54
----	---	----

16	t	43
----	---	----

17	t	53
----	---	----

18	t	57
----	---	----

19	t	49
----	---	----

20	t	56
----	---	----

21	t	33
----	---	----

22	c	42
----	---	----

23	c	33
----	---	----

24	c	46
----	---	----

25	c	37
----	---	----

26	c	43
----	---	----

27	c	41
----	---	----

28	c	10
----	---	----

29	c	42
----	---	----

30	c	55
----	---	----

31	c	19
----	---	----

32	c	17
----	---	----

33	c	55
----	---	----

Analysis

- ▶ Groups labelled c for “control” and t for “treatment”.
- ▶ Start with summaries (group means) and plot (boxplot).
- ▶ No pairing, matching: might compare means with *two-sample t-test*.
- ▶ For test, need approx. normality, but don’t need equal variability.
- ▶ Use summaries to decide if test reasonable.

Comparing means

```
proc means;  
    class group;  
    var score;
```

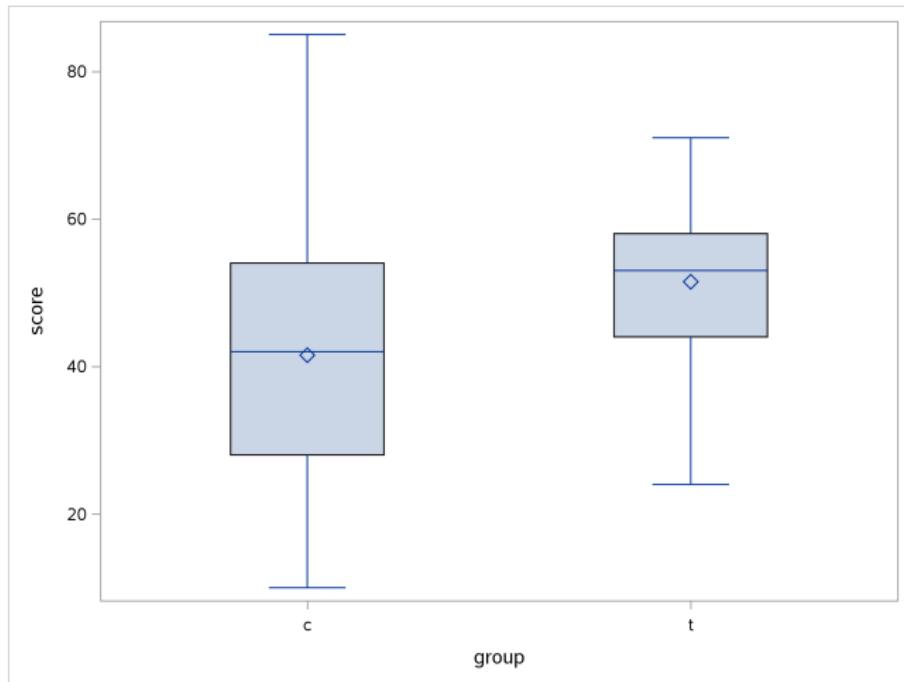
The MEANS Procedure

Analysis Variable : score

group	N	N	Mean	Std Dev	Minimum	Maximum
<hr/>						
c	23	23	41.5217391	17.1487332	10.0000000	85.0000000
t	21	21	51.4761905	11.0073568	24.0000000	71.0000000

Boxplots

```
proc sgplot;  
    vbox score / category=group;
```



Comments

- ▶ Groups not actually same size (maybe 2 kids had to drop out).
- ▶ Means a fair bit different, treatment mean higher.
- ▶ But a lot of variability, so groups do overlap.
- ▶ Standard deviations somewhat different too.
- ▶ Biggest threat to normality is outliers, none here.
- ▶ Both distributions not far off symmetric.
- ▶ t -test should be good enough.

The *t*-test

```
proc ttest side=L;  
  var score;  
  class group;
```

The TTEST Procedure

Variable: score

Method	Variances	DF	t Value	Pr < t
Pooled	Equal	42	-2.27	0.0143
Satterthwaite	Unequal	37.855	-2.31	0.0132

plus a lot more output.

Comments and Conclusions

- ▶ One-sided test (looking for *improvement*). `side` can be L (lower), U (upper) or 2 (two-sided, can be omitted.) This is L because control group first in alphabetical order.
- ▶ Right *t*-test is Satterthwaite (does not assume equal variability)
- ▶ P-value $0.0132 < 0.05$: there *is* increase in reading scores.
- ▶ Should not use pooled test, because SDs not close; even so, result very similar (P-value 0.0143).
- ▶ One-sided test doesn't give (regular) CI for difference in means. To get that, repeat analysis without `side=L`.

Doing it with R

- ▶ Proper reading-in function is `read_delim`.
- ▶ If you know that the file is in current folder, read it in by name (instead of searching for it):

```
kids=read_delim("drp.txt"," ")  
  
## Parsed with column specification:  
## cols(  
##   group = col_character(),  
##   score = col_integer()  
## )
```

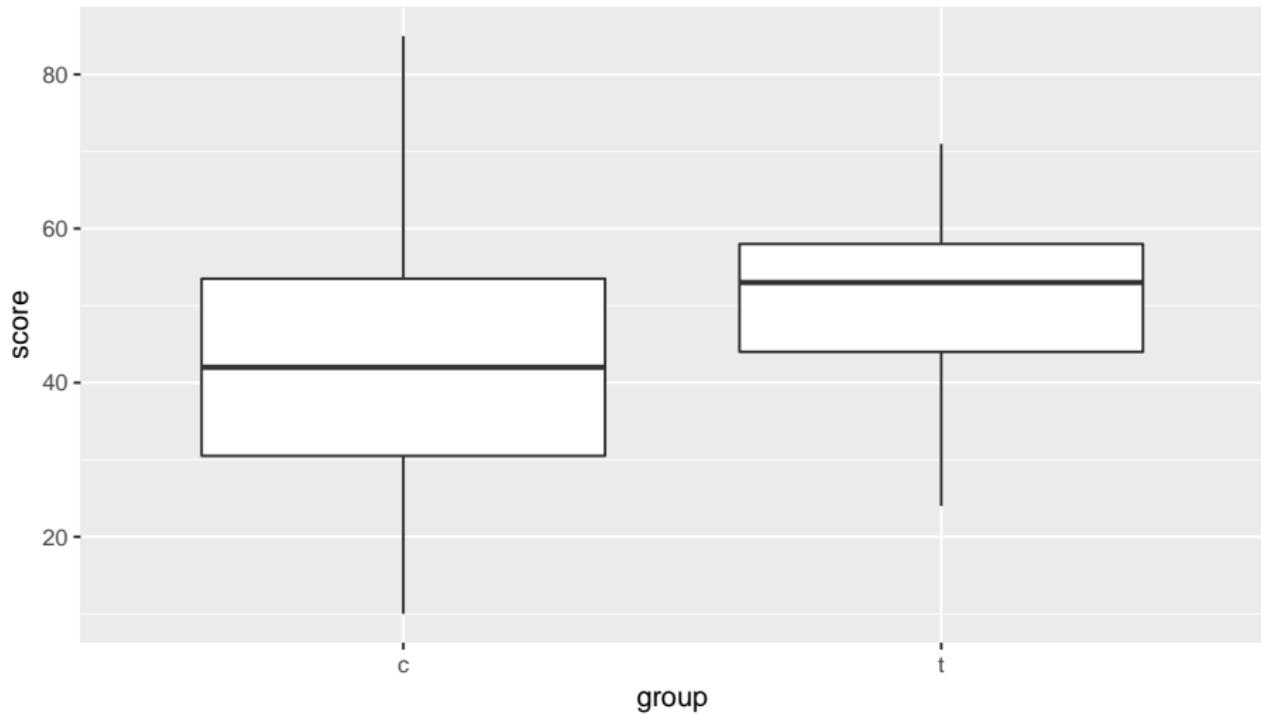
The data

```
kids

## # A tibble: 44 x 2
##       group score
##       <chr>  <int>
## 1       t     24
## 2       t     61
## 3       t     59
## 4       t     46
## 5       t     43
## 6       t     44
## 7       t     52
## 8       t     43
## 9       t     58
## 10      t     67
## # ... with 34 more rows
```

Boxplots

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



The (Satterthwaite-Welch) t -test

- ▶ c (control) before t (treatment) alphabetically, so proper alternative is “less”.
- ▶ R does Satterthwaite test by default (as before: new reading program really helps):

```
t.test(score~group,data=kids,alternative="less")  
  
##  
## Welch Two Sample t-test  
##  
## data: score by group  
## t = -2.3109, df = 37.855, p-value = 0.01319  
## alternative hypothesis: true difference in means is less than 0  
## 95 percent confidence interval:  
##       -Inf -2.691293  
## sample estimates:  
## mean in group c mean in group t  
##          41.52174      51.47619
```

The pooled *t*-test

- ▶ Pooled (equal variances) test this way:

```
t.test(score~group,data=kids,alternative="less",
       var.equal=T)

##
##  Two Sample t-test
##
## data: score by group
## t = -2.2666, df = 42, p-value = 0.01431
## alternative hypothesis: true difference in means is less than 0
## 95 percent confidence interval:
##        -Inf -2.567497
## sample estimates:
## mean in group c mean in group t
##          41.52174      51.47619
```

- ▶ Similar P-value to Satterthwaite test (and same results as SAS).

Two-sided test; CI

- ▶ To do 2-sided test, leave out alternative:

```
t.test(score~group,data=kids)

##
##  Welch Two Sample t-test
##
## data: score by group
## t = -2.3109, df = 37.855, p-value = 0.02638
## alternative hypothesis: true difference in means is not equal to zero
## 95 percent confidence interval:
## -18.67588 -1.23302
## sample estimates:
## mean in group c mean in group t
##           41.52174          51.47619
```

- ▶ Also gives CI: new reading program increases average scores by somewhere between about 1 and 19 points.
- ▶ Confidence intervals inherently two-sided, so do 2-sided test to get them.

Logic of testing

- ▶ Work out what *would* happen if null hypothesis were true.
- ▶ Compare to what actually *did* happen.
- ▶ If these are too far apart, conclude that null hypothesis *is not* true after all. (Be guided by P-value.)

As applied to our reading programs:

- ▶ If reading programs equally good, expect to see a difference in means close to 0.
 - ▶ Closeness quantified eg. by t .
- ▶ Mean reading score was 10 higher for new program.
- ▶ Difference of 10 was unusually big (P-value small from t -test). So conclude that new reading program is effective.

Nothing here about what happens if null hypothesis is *false*. This is power and type II error probability.

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?”
- ▶ R or SAS.

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”.
- ▶ Most easily in R.

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 8.852952 10.83  
## [8] 11.165242 8.016188 12.383518 1.378099 3.172503 13.07  
## [15] 5.015575
```

- ▶ Test whether x from population with mean 10 or not:

```
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
```

... continued

- ▶ Fail to reject the mean being 10 (a Type II error).
- ▶ Same again, but just get the P-value:

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

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

Write a function...

- ▶ ...to generate the random sample and return its P-value:

```
sim=function() {  
  x=rnorm(15,8,4)  
  t.test(x,mu=10)$p.value  
}
```

- ▶ Test it (different from before: random data):

```
sim()  
## [1] 0.1286652
```

- ▶ Once again fail to reject a null mean of 10 (incorrectly).

To run lots of times

- ▶ Like this — “as many times as the first thing, do the second thing”:

```
pvals=replicate(1000,sim())
head(pvals)
```

```
## [1] 0.189800019 0.000728556 0.250217788 0.277753581 0.0
```

- ▶ What fraction of those P-values are 0.05 or smaller? This is our best guess at how often our test will correctly reject:

```
table(pvals<=0.05)
```

```
##
```

```
## FALSE TRUE
```

```
## 578 422
```

- ▶ Test correctly rejects 422 times of 1000: 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*.
- ▶ In R, `power.t.test`. `delta` is *difference* between null and true mean:

```
power.t.test(n=15, delta=2, 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
```

Calculating power in SAS

- ▶ The magic proc is proc power. Here's how we do our example: a one-sample t -test with $n = 15$, $H_0 : \mu = 10$ vs. $H_a : \mu \neq 10$, and a true $\mu = 8$, so that the difference between the true μ and the H_0 value of μ is 2:

```
proc power;  
    onesamplemeans  
    test=t  
    mean=2  
    stddev=4  
    ntotal=15  
    power=..;
```

The results

The POWER Procedure
One-Sample t Test for Mean

Fixed Scenario Elements

Distribution	Normal
Method	Exact
Mean	2
Standard Deviation	4
Total Sample Size	15
Number of Sides	2
Null Mean	0
Alpha	0.05

Computed Power

Power

0.438

Comparison of results

Method	Power
Simulation	0.422
<code>power.t.test</code>	0.4378
<code>proc power</code>	0.438

- ▶ Two calculated power values are same to within rounding.
- ▶ Simulation power is similar; 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 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 methods, 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=2,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).

Using proc power

Explicitly leave ntotal missing, and supply value for power:

```
proc power;  
    onesamplemeans  
    test=t  
    mean=2  
    stddev=4  
    ntotal=.  
    power=0.80;
```

Results

The POWER Procedure
One-Sample t Test for Mean

Fixed Scenario Elements

Distribution	Normal
Method	Exact
Mean	2
Standard Deviation	4
Nominal Power	0.8
Number of Sides	2
Null Mean	0
Alpha	0.05

Computed N Total

Actual Power	N Total
0.808	34

SAS says that with $n = 34$, power actually 0.808.

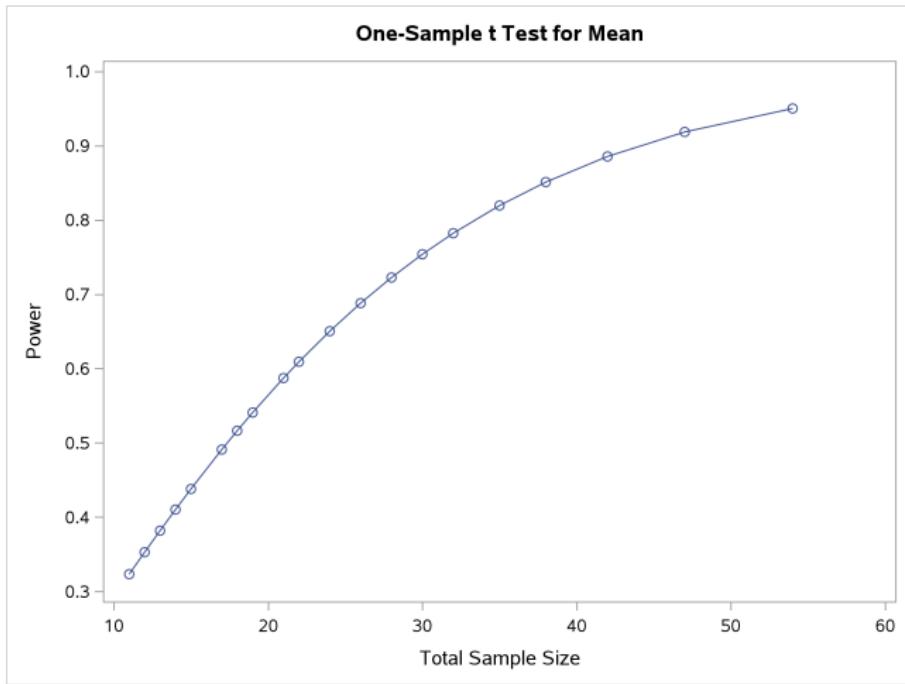
Power curves

- ▶ 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**.
- ▶ SAS makes these automatically (have to learn how); in R, build and plot it yourself.

In SAS

```
proc power plotonly;
  onesamplemeans
    test=t
    mean=2
    stddev=4
    ntotal=.
    power=0.80;
  plot y=power min=0.3 max=0.95;
```

The graph



“Diminishing returns”: increasing sample size increases power, but by decreasing amount.

Building the same thing in R

- ▶ 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=2,sd=4,type="one.sample")
ans$power
## [1] 0.2928286 0.5644829 0.7539627 0.8693979 0.9338976
## [8] 0.9929987 0.9968496 0.9986097
```

Building a plot

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

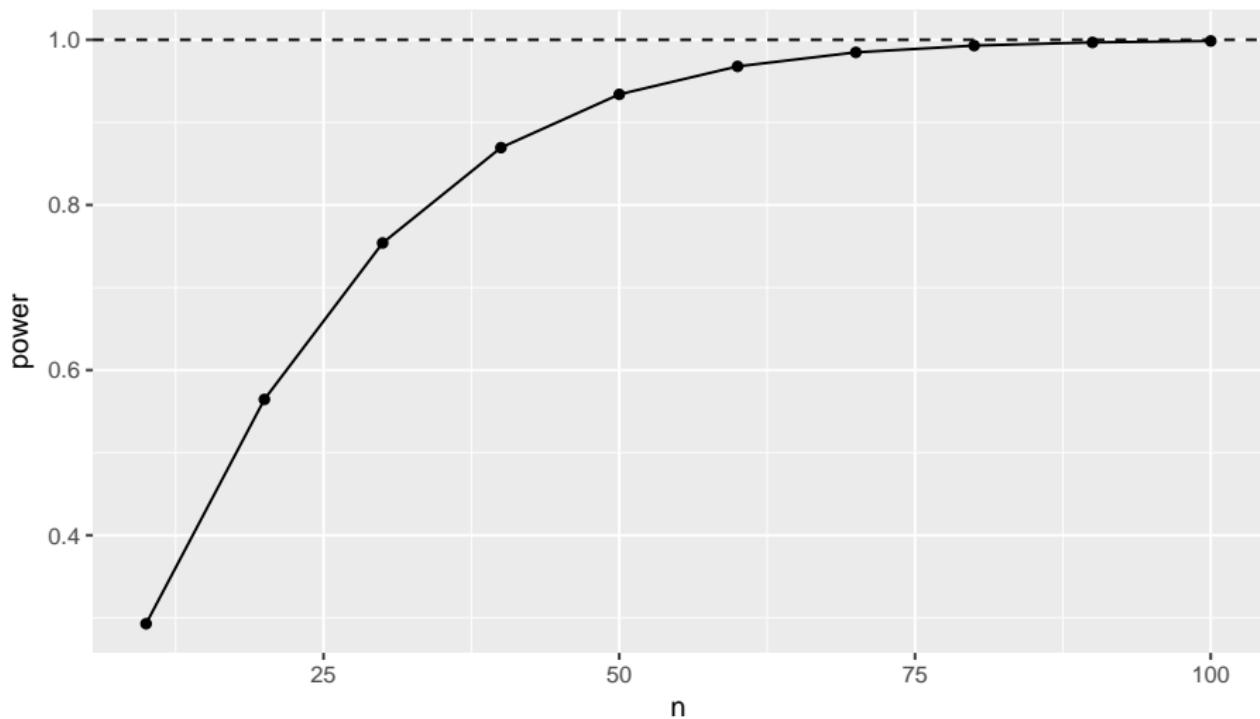
```
d=tibble(n=ns, power=ans$power)
```

- ▶ 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

g



Power curves for mean differences

- ▶ Can also investigate power as it depends on how far the true mean is from the null mean (the farther apart they are, the higher the power will be).
- ▶ Investigate for two different sample sizes, 15 and 34.
- ▶ First make all combos of mean diff and sample size:

```
diffs=seq(0,4,0.5)
diffs
## [1] 0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0

ns=c(15,34)
ns
## [1] 15 34

combos=expand.grid(diff=diffs,n=ns)
```

The combos

combos

```
##      diff  n
```

```
## 1    0.0 15
```

```
## 2    0.5 15
```

```
## 3    1.0 15
```

```
## 4    1.5 15
```

```
## 5    2.0 15
```

```
## 6    2.5 15
```

```
## 7    3.0 15
```

```
## 8    3.5 15
```

```
## 9    4.0 15
```

```
## 10   0.0 34
```

```
## 11   0.5 34
```

```
## 12   1.0 34
```

```
## 13   1.5 34
```

```
## 14   2.0 34
```

```
## 15   2.5 34
```

```
## 16   3.0 34
```

Calculate and plot

- ▶ Calculate the powers, carefully:

```
ans=power.t.test(n=combos$n,delta=combos$diff,SD=4,  
type="one.sample")
```

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

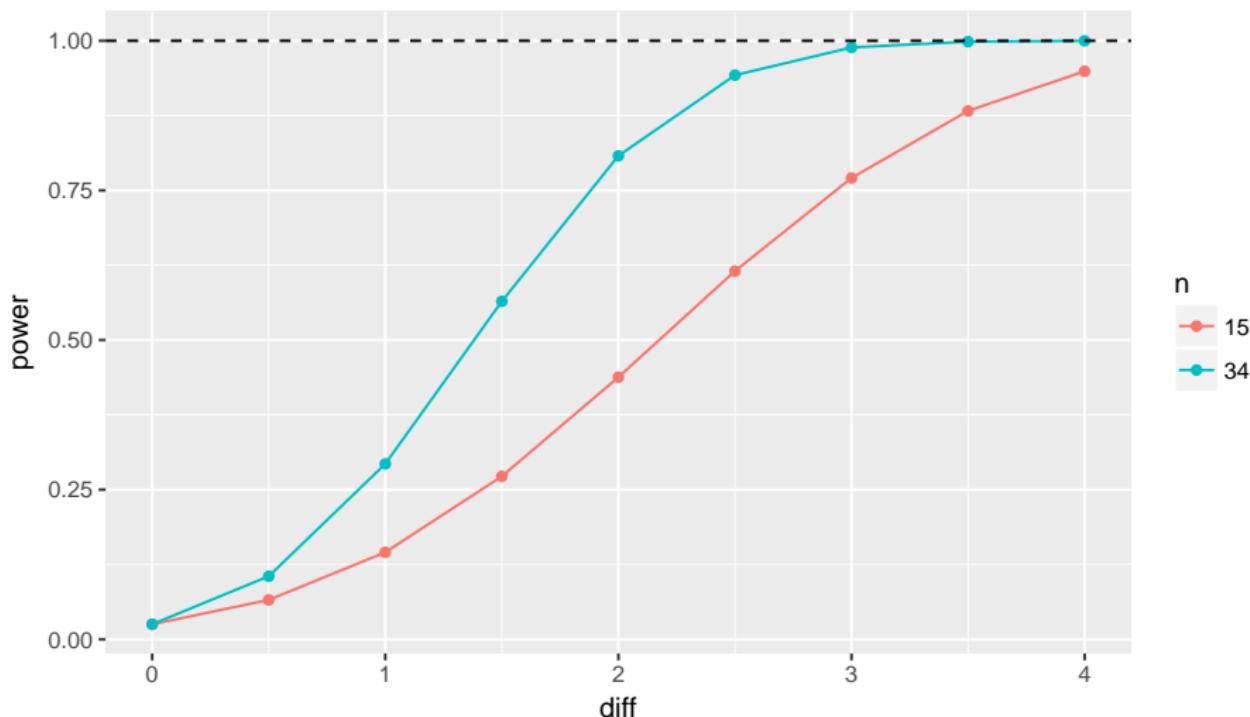
```
d=tibble(n=factor(combos$n),diff=combos$diff,  
power=ans$power)
```

- ▶ then make the plot:

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

The power curves

g



Comments

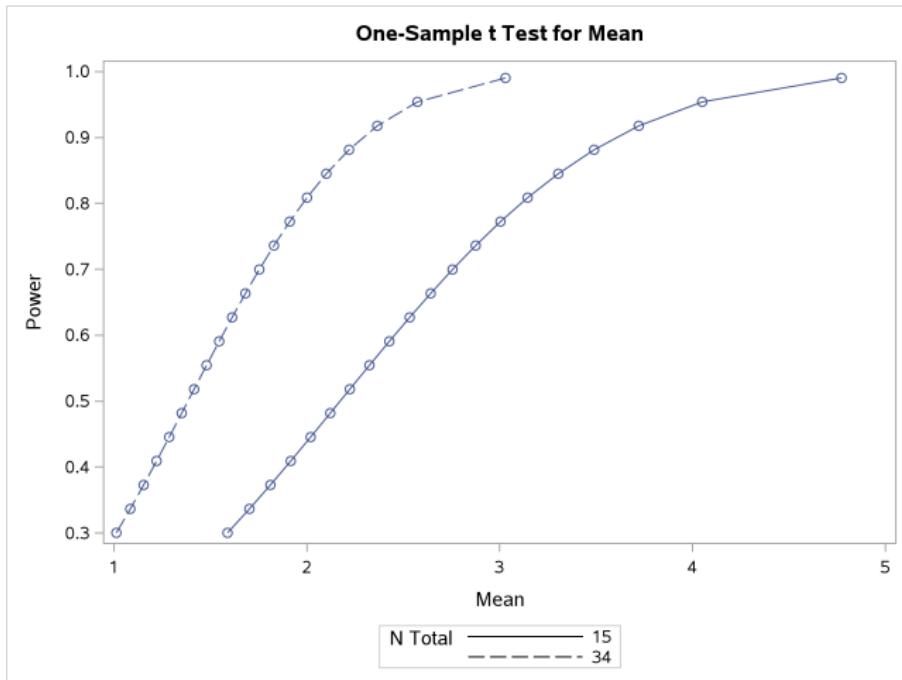
- ▶ When $\text{diff}=0$, that is, the true mean equals the null mean, H_0 is actually *true*, and the probability of rejecting it then is $\alpha = 0.05$.
- ▶ As the null gets more wrong (diff increases), it becomes easier to correctly reject it.
- ▶ The blue power curve is above the red one for any $\text{diff} > 0$, 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 $\text{diff}=2$, producing power 0.42 and 0.80 as shown on the graph.
- ▶ With $n = 34$, a mean that is away from the null by more than about 3 is almost certain to be correctly rejected. (With $n = 15$, the difference needs to be more than 4.)

The exact same thing in SAS

```
proc power plotonly;
  onesamplemeans
    test=t
    mean=.
    stddev=4
    ntotal=15 34
    power=0.80;
  plot y=power min=0.3 max=0.99;
```

I left the `mean` (difference) blank, and had SAS calculate for powers between 0.30 and 0.99. I specify two different sample sizes as shown.

The SAS power curves



The value labelled `mean` is actually the *difference* between the true mean and the null mean (10).

Calculating power for a *t*-test

Think about reading programs again. Suppose we treat the study that was done as a pilot study and wish to plan the real thing.

Recall sample statistics:

```
proc means;  
    var score;  
    class group;
```

The MEANS Procedure

Analysis Variable : score

group	N	Obs	N	Mean	Std Dev	Minimum	Maximum
c	23	23	41.5217391	17.1487332	10.0000000	85.0000000	
t	21	21	51.4761905	11.0073568	24.0000000	71.0000000	

What to consider

- ▶ What kind of t -test (here 2-sample, not 1-sample or paired)
- ▶ Given a 2-sample t , Satterthwaite not pooled
- ▶ What kind of H_a (here one-sided)
- ▶ What the population SD is (usually have to guess this). Here the sample SDs were 11 and 17, so 15 seems a fair guess (same for each group).
- ▶ What size departure from null of interest to us (here, if the new reading method increases mean test score by 5–10 points, that is of interest).
- ▶ Draw some pictures showing sample size-power relationship for these mean test score differences.

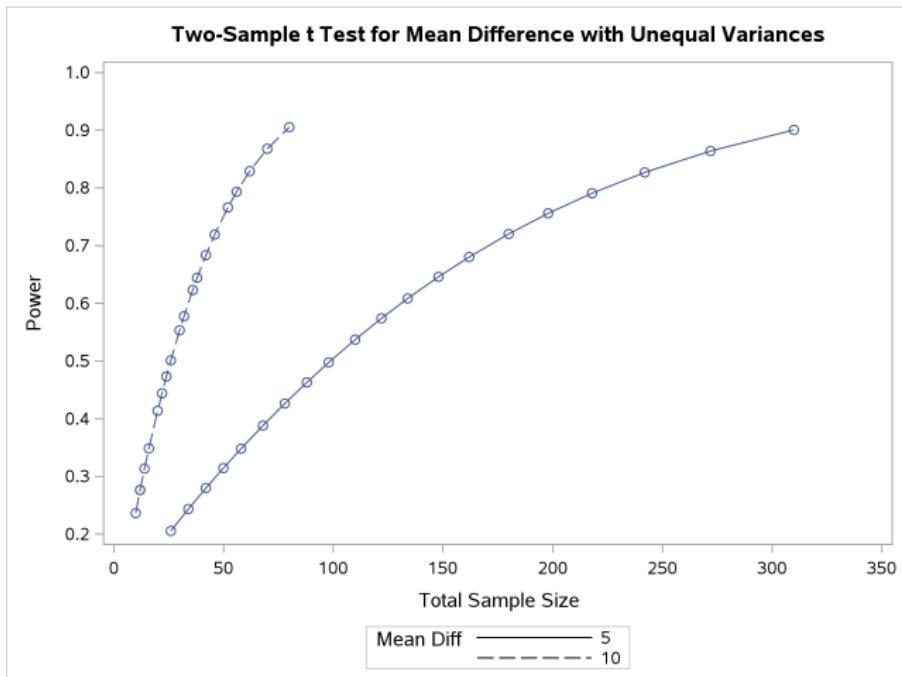
Code

```
proc power plotonly;
  twosamplemeans
    test=diff_satt
    sides=1
    meandiff=5 10
    stddev=15
    ntotal=.
    power=0.4;
  plot y=power min=0.2 max=0.9;
```

Code comments

- ▶ `twosamplemeans`
- ▶ `test=diff_satt` to specify Satterthwaite two-sample test
- ▶ `sides=1` (1-sided test)
- ▶ Population SDs taken to be 15 for both groups.
- ▶ Leave total sample size blank to plot power against sample size for each mean difference.
- ▶ Have to supply a value for power even though we want to let it vary.

The power curves



Comments

- ▶ If the new reading method actually leads to a 10-point increase in mean test scores (rather than 5), we will be much more easily able to prove that it works.
- ▶ Original total sample size was $23 + 21 = 44$:
 - ▶ if new program improves reading scores by 10, power is barely acceptable 0.6 or so
 - ▶ if new program improves reading scores by only 5, power is definitely unacceptable 0.3.
 - ▶ If we want to reach power 0.8, we need total of about 60 children (2×30) if the mean improvement is 10, and over 200 (2×100) (!) if the mean improvement is only 5.

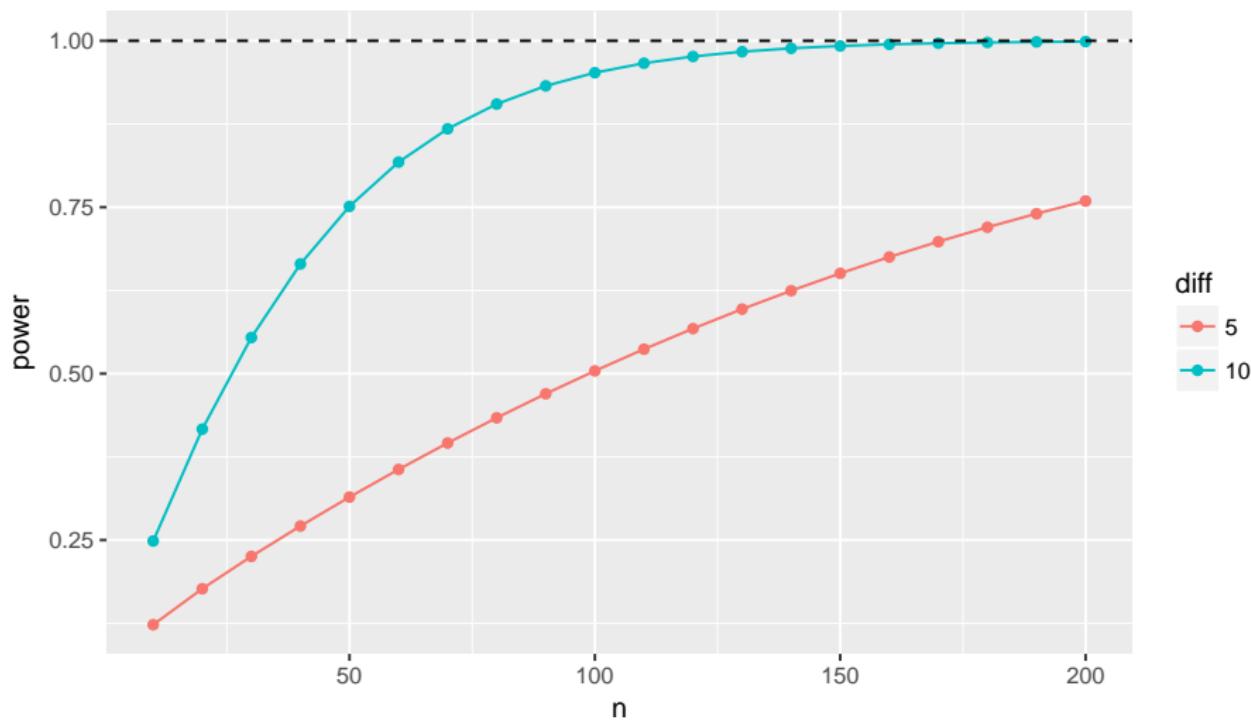
Same picture in R

Similar procedure to before:

```
diffs=c(5,10)
ns=seq(10,200,10) # total sample size
 combos=expand.grid(diff=diffs,n=ns)
ans=power.t.test(n=combos$n/2,delta=combos$diff, sd=15,
 type="two.sample",alternative="one.sided")
# divide by 2 above because R wants per group
d=tibble(diff=factor(combos$diff),n=combos$n,
 power=ans$power)
g=ggplot(d,aes(x=n,y=power,colour=diff))+geom_point()+
geom_line()+
geom_hline(yintercept=1,linetype="dashed")
```

The power curves

g



Or, more accurately...

- ▶ Didn't actually have same-size groups or equal population SDs. Assumed them for R.
- ▶ SAS will allow different values per group.
- ▶ We had sample sizes 21 and 23, sample SDs 11 and 17 (use as population SDs).
- ▶ Unequal sample sizes usually decrease power, but smaller sample size with smaller SD actually better. Overall effect unclear.
- ▶ SAS: use groupstddevs and groupns, and vertical bars.

```
proc power;
  twosamplemeans
    test=diff_satt
    sides=1
    meandiff=5
    groupstddevs=11|17
    groupns=21|23
    power=.;
```

Results

The POWER Procedure

Two-Sample t Test for Mean Difference with Unequal Variances

Fixed Scenario Elements

Distribution	Normal
Method	Exact
Number of Sides	1
Mean Difference	5
Group 1 Standard Deviation	11
Group 2 Standard Deviation	17
Group 1 Sample Size	21
Group 2 Sample Size	23
Null Difference	0
Nominal Alpha	0.05

Computed Power

Actual Alpha	Power
0.0499	0.309

Power actually went *up* a tiny bit.

Unequal sample sizes

To show effect of unequal sample sizes, go back to SDs both being 15, but have very unequal sample sizes. What effect does that have on power?

```
proc power;  
twosamplemeans  
    test=diff_satt  
    sides=1  
    meandiff=5  
    stddev=15  
    groupns=10|34  
    power=.;
```

Results

Power for 22 in each group was 29%:

```
The UNIVARIATE Procedure
Variable: Time

Tests for Location: Mu0=160

Test      -Statistic-      -----p Value-----
Student's t      t  1.824364      Pr > |t|      0.0784
Sign            M      2      Pr >= |M|      0.5847
Signed Rank     S      50      Pr >= |S|      0.3118

Computed Power

Actual
Alpha      Power
0.0505      0.225
```

Unequal sample sizes bring power down to 22.5%.

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.
- ▶ Illustrate with R and SAS, R first.

Some data, to illustrate

```
twogroups=read_delim("duality.txt", " ")  
  
## Parsed with column specification:  
## cols(  
##   y = col_integer(),  
##   group = col_integer()  
## )  
  
glimpse(twogroups)  
  
## Observations: 15  
## Variables: 2  
## $ y      <int> 10, 11, 11, 13, 13, 14, 14, 14, 15, 16, 13, 13,  
## $ group  <int> 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 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 zero
## 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 zero
## 90 percent confidence interval:
## -5.010308 -0.323025
## 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.

In SAS

```
proc import  
    datafile='/home/ken/duality.txt'  
    dbms=dlm  
    out=duality  
    replace;  
    delimiter=' '|;  
    getnames=yes;  
  
proc print;
```

Output over.

The data, small

Obs	y	group
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

Test and CI at default $\alpha = 0.05$

```
proc ttest;  
  var y;  
  class group;
```

The TTEST Procedure					
Variable: y					
group	Method	Mean	95% CL Mean		Std Dev
1		13.0000	11.4627	14.5373	2.0000
2		15.6667	12.8769	18.4564	2.6583
Diff (1-2)	Pooled	-2.6667	-5.2580	-0.0754	2.2758
Diff (1-2)	Satterthwaite	-2.6667	-5.5626	0.2292	
group					
Method					
1			1.3509	3.8315	
2			1.6593	6.5198	
Diff (1-2)	Pooled		1.6499	3.6665	
Diff (1-2)	Satterthwaite				
Method					
Variances					
DF					
Pooled	Equal	13	-2.22	0.0446	
Satterthwaite	Unequal	8.7104	-2.09	0.0668	

Getting 90% CI

```
proc ttest alpha=0.10;  
  var y;  
  class group;
```

The TTEST Procedure					
Variable: y					
group	Method	Mean	90% CL Mean		Std Dev
1		13.0000	11.7603	14.2397	2.0000
2		15.6667	13.4798	17.8535	2.6583
Diff (1-2)	Pooled	-2.6667	-4.7909	-0.5425	2.2758
Diff (1-2)	Satterthwaite	-2.6667	-5.0103	-0.3230	
group					
Method					
1			1.4365	3.4220	
2			1.7865	5.5539	
Diff (1-2)	Pooled		1.7352	3.3806	
Diff (1-2)	Satterthwaite				
Method					
Variances					
DF					
Pooled	Equal	13	-2.22	0.0446	
Satterthwaite	Unequal	8.7104	-2.09	0.0668	

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.

Reading in data

```
proc import  
  datafile='/home/ken/irs.txt'  
  dbms=csv  
  out=irs  
  replace;  
getnames=yes;
```

Checking: all looks good

```
proc print;
```

Obs	Time
1	91
2	64
3	243
4	167
5	123
6	65
7	71
8	204
9	110
10	178
11	264
12	119
13	112
14	142
15	451
16	474
17	209
18	104
19	84
20	302
21	527
22	303
23	228
24	391
25	215
26	188
27	150
28	102
29	162
30	194

t-test

n = 30 data values:

```
proc ttest h0=160 sides=U;  
  var Time;
```

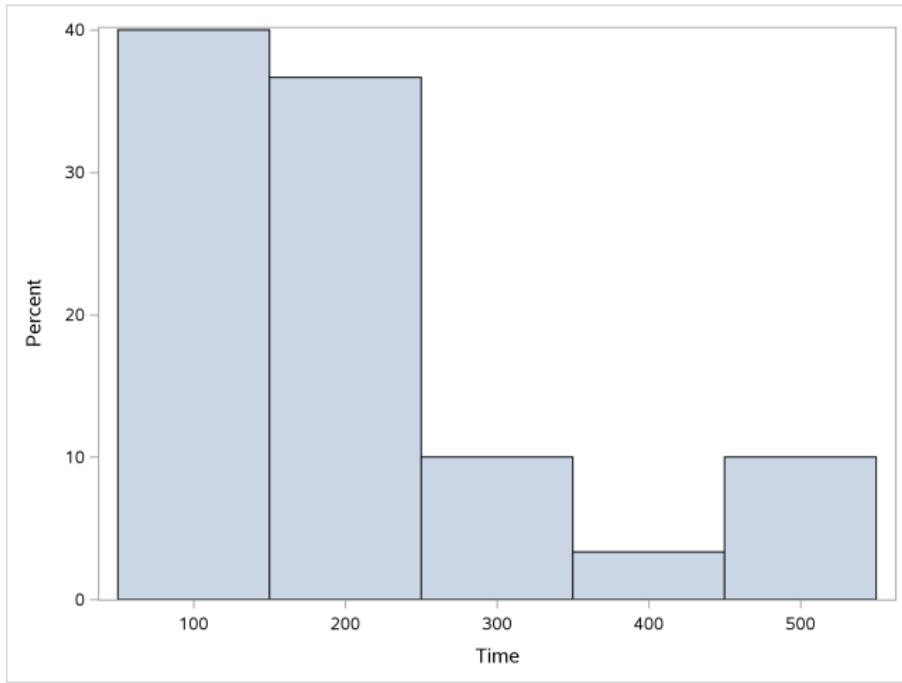
N	Mean	Std Dev	Std Err	Minimum	Maximum
30	201.2	123.8	22.6015	64.0000	527.0
Mean	95% CL Mean		Std Dev	95% CL	Std Dev
201.2	162.8	Infty	123.8	98.5899	166.4
DF	t Value		Pr > t		
29	1.82		0.0392		

Comments

- ▶ All looks good, and we have shown that the mean time to complete this form is greater than 160 minutes (P-value 0.0392).
- ▶ **But**, the *t*-test assumes approximately normally-distributed data. Do we have that? Histogram:

```
proc sgplot;  
    histogram Time;
```

The histogram



The times are *skewed to the right*. Maybe we should be looking not at the *mean* time but the *median*.

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.
- ▶ 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).

Obtaining P-value for sign test 1/2

- ▶ Do this in R. Read data in (I checked that this worked):

```
irs=read_csv("irs.txt")  
  
## Parsed with column specification:  
## cols(  
##   Time = col_integer()  
## )
```

- ▶ 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*.

Obtaining P-value for sign test 2/2

- ▶ 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 them:

```
p=dbinom(17:30,30,0.5)
```

```
p
```

```
## [1] 1.115351e-01 8.055309e-02 5.087564e-02 2.798160e-02
```

```
## [6] 5.450961e-03 1.895986e-03 5.529961e-04 1.327191e-04
```

```
## [11] 3.781170e-06 4.051253e-07 2.793968e-08 9.313226e-09
```

- ▶ and add them up to get our P-value:

```
sum(p)
```

```
## [1] 0.2923324
```

Doing the sign test in SAS

- ▶ SAS has proc univariate which obtains a whole bunch of information about a single variable, including these, *which are two-sided*:

```
proc univariate location=160;  
var Time;
```

The UNIVARIATE Procedure				
Variable: Time				
Tests for Location: Mu0=160				
Test	-Statistic-	-----	p Value-----	
Student's t	t 1.824364	Pr > t	0.0784	
Sign	M 2	Pr >= M	0.5847	
Signed Rank	S 50	Pr >= S	0.3118	
Computed Power				
Actual				
Alpha	Power			
0.0505	0.225			

Comments

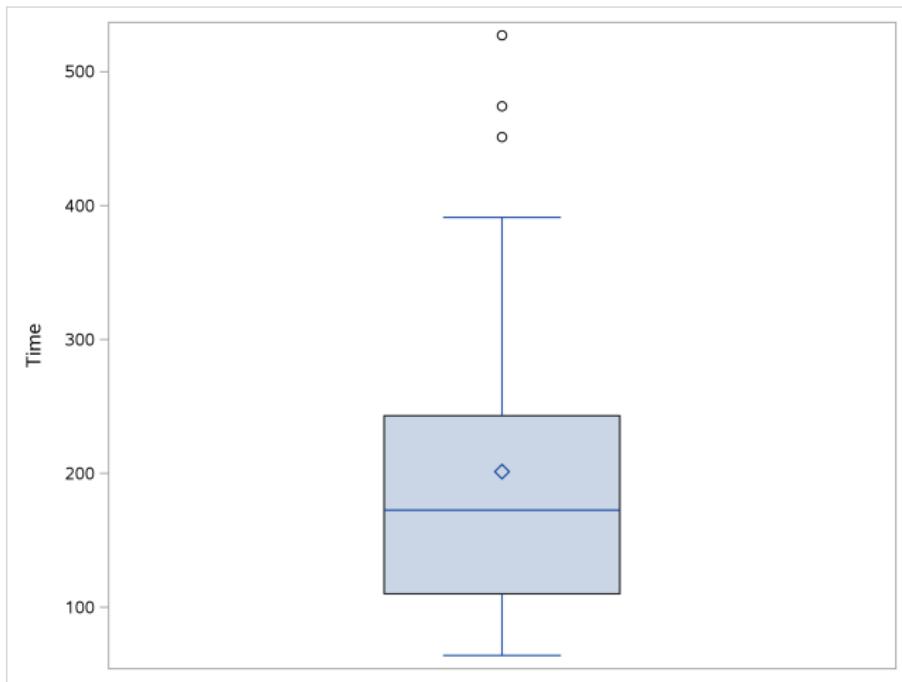
- ▶ P-values are (take half of the two-sided SAS ones):

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? Look at boxplot:

```
proc sgplot;  
vbox Time;
```

The boxplot



Concluding comments (about this)

- ▶ 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 $\alpha = 0.05$).
- ▶ The two-sided sign test goes like this:
 - ▶ Count the number S of data values above the hypothesized median.
 - ▶ Calculate the probability of getting *either* S or more successes, and *also* of getting S or fewer successes.
 - ▶ Find the smaller of these, and *multiply it by 2* (two-sided test).

Check procedure for null median 160

- ▶ How many observations do we have?

```
n_obs=length(irs$Time)
```

- ▶ This:

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

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

- ▶ but we need the second thing in the n column:

```
S=irs %>% count(Time>160) %>% select(n) %>% slice(2) %>%
  as.numeric()
```

S

```
## [1] 17
```

... continued

- ▶ then find the probability of S or more:

```
p1=sum(dbinom(S:n_obs,n_obs,0.5)) ; p1  
## [1] 0.2923324
```

- ▶ and of S or less:

```
p2=sum(dbinom(0:S,n_obs,0.5)) ; p2  
## [1] 0.8192027
```

- ▶ and the smaller of these, doubled (two-sided):

```
min_p=min(p1,p2) ; 2*min_p  
## [1] 0.5846647
```

- ▶ This is the same two-sided P-value that came from SAS.

Make a function of this

- ▶ To calculate the confidence interval, we'll be doing this a lot with different null medians.
- ▶ So make a function that takes the null median as input ("hard-code" the data frame and column for now). Most of the code above we can copy and slightly edit:

```
sign_test=function(null_median) {  
  n_obs=length(irs$Time)  
  S=irs %>% count(Time>null_median) %>% select(n) %>% slice(2)  
  as.numeric()  
  p1=sum(dbinom(S:n_obs,n_obs,0.5))  
  p2=sum(dbinom(0:S,n_obs,0.5))  
  min_p=min(p1,p2)  
  2*min_p  
}
```

- ▶ Test, where we know the answer: check.

```
sign_test(160)
```

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

Doing a whole bunch

- We could do it one at a time:

```
sign_test(190) ; sign_test(300)
```

```
## [1] 0.5846647  
## [1] 0.001430906
```

- 190 is inside the interval, and 300 is outside.
- but this is inefficient. Better to choose our null medians first:

```
null_medians=seq(100,300,20) ; null_medians
```

```
## [1] 100 120 140 160 180 200 220 240 260 280 300
```

- and then run the function for each of them, which goes like this, “for each null median, run the function `sign_test` for that null median and get the P-value”:

```
p=map_dbl(null_medians,sign_test)
```

The results

- ▶ Make a data frame of results:

```
d=tibble(median=null_medians,p_value=p)
d

## # A tibble: 11 x 2
##   median      p_value
##   <dbl>        <dbl>
## 1     100 0.0003249142
## 2     120 0.0987371467
## 3     140 0.2004884221
## 4     160 0.5846647117
## 5     180 0.8555355519
## 6     200 0.3615946081
## 7     220 0.0427739453
## 8     240 0.0161248017
## 9     260 0.0052228794
## 10    280 0.0014309064
## 11    300 0.0014309064
```

- ▶ 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.

Refining the bottom end of the interval

```
null_medians=seq(100,120,2)
tibble(median=null_medians,
       p_value=map_dbl(null_medians,sign_test))

## # A tibble: 11 x 2
##   median     p_value
##   <dbl>     <dbl>
## 1 100  0.0003249142
## 2 102  0.0014309064
## 3 104  0.0052228794
## 4 106  0.0052228794
## 5 108  0.0052228794
## 6 110  0.0161248017
## 7 112  0.0427739453
## 8 114  0.0427739453
## 9 116  0.0427739453
## 10 118  0.0427739453
## 11 120  0.0987371467
```

Generalizing that function

- ▶ I want to be able to use that `sign_test` function on *any* data frame and any column from that data frame, not just the ones we coded in above.
- ▶ Add the data frame and column name to the function inputs.
- ▶ To get out the column we want: its name is unknown to us as we write the function, so we use a different technique to `$` that we used before:

```
sign_test=function(null_median,mydata,mycol) {  
  n_obs=length(mydata[[mycol]])  
  S=mydata %>% count(. [[mycol]]>null_median) %>%  
    select(n) %>% slice(2) %>% as.numeric()  
  p1=sum(dbinom(S:n_obs,n_obs,0.5))  
  p2=sum(dbinom(0:S,n_obs,0.5))  
  min_p=min(p1,p2)  
  2*min_p  
}
```

Does it work?

```
sign_test(160,irs,"Time")  
## [1] 0.5846647  
  
null_medians=seq(100,140,20)  
tibble(medians=seq(100,140,20),  
       p_value=map_dbl(null_medians,sign_test,irs,"Time"))  
  
## # A tibble: 3 x 2  
##   medians     p_value  
##   <dbl>        <dbl>  
## 1     100 0.0003249142  
## 2     120 0.0987371467  
## 3     140 0.2004884221
```

- ▶ yes!
- ▶ To obtain a column whose name *is stored in a variable*, use . to refer to “the data frame that came from the previous step” and [[around the variable that contains the name.

Some different data, and a different test

Take a look at these data (12 rows of 3 columns):

Case	Drug A	Drug B		7	14.9	16.7
1	2.0	3.5		8	6.6	6.0
2	3.6	5.7		9	2.3	3.8
3	2.6	2.9		10	2.0	4.0
4	2.6	2.4		11	6.8	9.1
5	7.3	9.9		12	8.5	20.9
6	3.4	3.3				

Matched pairs

- ▶ Data are comparison of 2 drugs for effectiveness at reducing pain.
- ▶ 12 subjects (cases) were arthritis sufferers
- ▶ Response is #hours of pain relief from each drug.
- ▶ In reading example, each child tried only *one* reading method.
- ▶ But here, each subject tried out *both* drugs, giving us two measurements.
- ▶ Possible because, if you wait long enough, one drug has no influence over effect of other.
- ▶ Advantage: focused comparison of drugs. Compare one drug with another on *same* person, removes a lot of random variability.
- ▶ **Matched pairs**, requires different analysis.
- ▶ Design: randomly choose 6 of 12 subjects to get drug A first, other 6 get drug B first.

Reading data, in SAS

```
proc import  
    datafile='/home/ken/analgesic.txt'  
    dbms=dlm  
    out=pain  
    replace;  
    delimiter=' '|;  
    getnames=yes;
```

The data

```
proc print;
```

Obs	subject	drugA	drugB
1	1	2	3.5
2	2	3.6	5.7
3	3	2.6	2.9
4	4	2.6	2.4
5	5	7.3	9.9
6	6	3.4	3.3
7	7	14.9	16.7
8	8	6.6	6
9	9	2.3	3.8
10	10	2	4
11	11	6.8	9.1
12	12	8.5	20.9

Matched pairs t-test

```
proc ttest;  
    paired druga*drugb;
```

N	Mean	Std Dev	Std Err	Minimum	Maximum
12	-2.1333	3.4092	0.9841	-12.4000	0.6000
	Mean	95% CL Mean	Std Dev	95% CL Std Dev	
	-2.1333	-4.2994	0.0327	3.4092	2.4150
		DF	t Value	Pr > t	
		11	-2.17	0.0530	

Comments

- ▶ P-value 0.0530.
- ▶ At $\alpha = 0.05$, cannot quite reject null of no difference, though result is very close to significance.
- ▶ “Hand-calculation” way of doing this is to find the 12 differences, one for each subject, and do 1-sample t -test on those differences. Shown on next page.

Alternative way to do matched pairs

- ▶ Define a new variable to calculate and store differences.
- ▶ This is done by creating a *new data set* and then defining the new variable, as shown:

```
data pain2;  
    set pain;  
    diff=druga-drugb;
```

- ▶ `set` means “bring in everything from the old data set”. To that we add the new variable `diff`.

The new data set pain2

```
proc print;
```

Obs	subject	drugA	drugB	diff
1	1	2	3.5	-1.5
2	2	3.6	5.7	-2.1
3	3	2.6	2.9	-0.3
4	4	2.6	2.4	0.2
5	5	7.3	9.9	-2.6
6	6	3.4	3.3	0.1
7	7	14.9	16.7	-1.8
8	8	6.6	6	0.6
9	9	2.3	3.8	-1.5
10	10	2	4	-2.0
11	11	6.8	9.1	-2.3
12	12	8.5	20.9	-12.4

Now do *t*-test on differences

```
proc ttest h0=0;  
    var diff;
```

t-test is an ordinary 1-sample test on diff. Note that null-hypothesis mean has to be given with only one sample.

N	Mean	Std Dev	Std Err	Minimum	Maximum
12	-2.1333	3.4092	0.9841	-12.4000	0.6000
	Mean	95% CL Mean	Std Dev	95% CL Std Dev	
	-2.1333	-4.2994	0.0327	3.4092	2.4150
		DF	t Value	Pr > t	
		11	-2.17	0.0530	

Paired test in R: reading the data

Values separated by spaces:

```
pain=read_delim("analgesic.txt", " ")  
  
## Parsed with column specification:  
## cols(  
##   subject = col_integer(),  
##   druga = col_double(),  
##   drugb = col_double()  
## )
```

The data

```
pain
```

```
## # A tibble: 12 x 3
##   subject druga drugb
##   <int>  <dbl>  <dbl>
## 1      1    2.0    3.5
## 2      2    3.6    5.7
## 3      3    2.6    2.9
## 4      4    2.6    2.4
## 5      5    7.3    9.9
## 6      6    3.4    3.3
## 7      7   14.9   16.7
## 8      8    6.6    6.0
## 9      9    2.3    3.8
## 10     10   2.0    4.0
## 11     11   6.8    9.1
## 12     12   8.5   20.9
```

Paired test

```
with(pain, t.test(druga, drugb, paired=T))

##
##  Paired t-test
##
## data:  druga and drugb
## t = -2.1677, df = 11, p-value = 0.05299
## alternative hypothesis: true difference in means is not equal to zero
## 95 percent confidence interval:
## -4.29941513  0.03274847
## sample estimates:
## mean of the differences
##                      -2.133333
```

P-value as before. Likewise, you can calculate the differences yourself and do a 1-sample *t*-test on them, over:

T-testing the differences

- ▶ First calculate a column of differences (separate from the data frame, this way):

```
diff=with(pain,druga-drugb)
```

```
diff
```

```
## [1] -1.5 -2.1 -0.3  0.2 -2.6  0.1 -1.8  0.6 -1.5 -2.0 -  
## [12] -12.4
```

- ▶ then throw them into t.test, testing that the mean is zero, with same result as before:

```
t.test(diff,mu=0)
```

```
##
```

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

```
##
```

```
## data: diff
```

```
## t = -2.1677, df = 11, p-value = 0.05299
```

```
## alternative hypothesis: true mean is not equal to 0
```

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

```
## -4.29941513 0.03274847
```

```
## sample estimates:
```

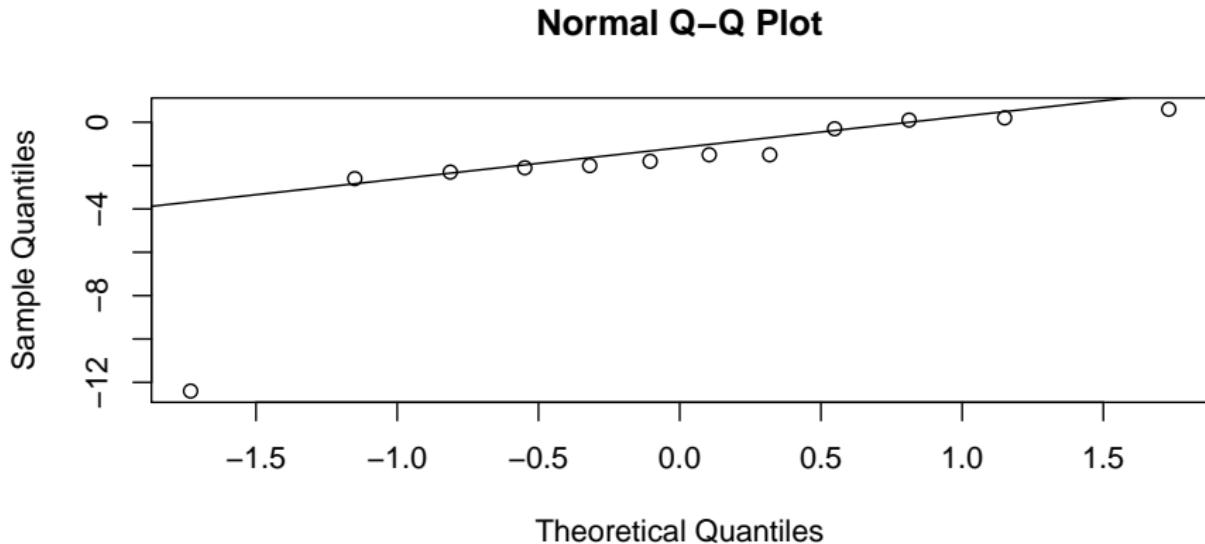
```
## mean of x
```

Assessing normality

- ▶ Matched pairs analyses assume (theoretically) that differences normally distributed.
- ▶ 1-sample and 2-sample t -tests assume (each) group normally distributed.
- ▶ Though we know that t -tests generally behave well even without normality.
- ▶ How to assess normality? A normal quantile plot.
- ▶ Idea: scatter of points should follow the straight line, without curving.
- ▶ Outliers show up at bottom left or top right of plot as points off the line.

The normal quantile plot

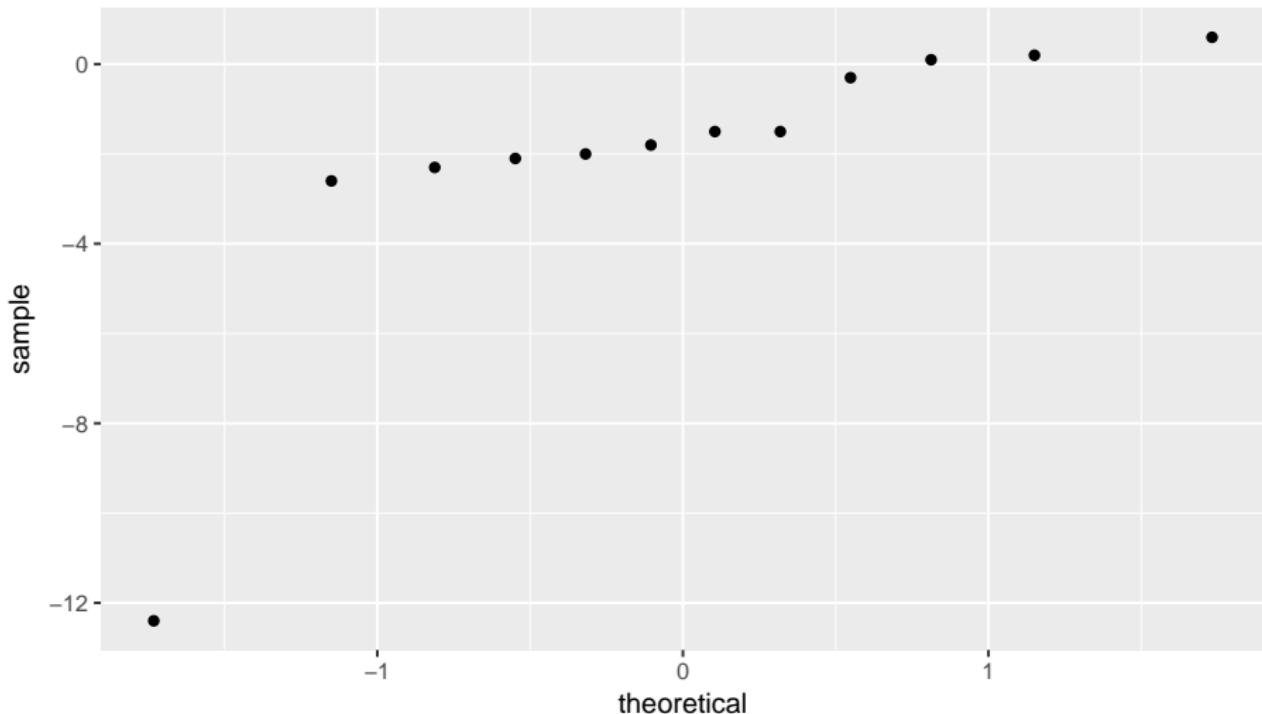
```
qqnorm(diff) ; qqline(diff)
```



Points should follow the straight line. Bottom left one way off, so normality questionable here: outlier.

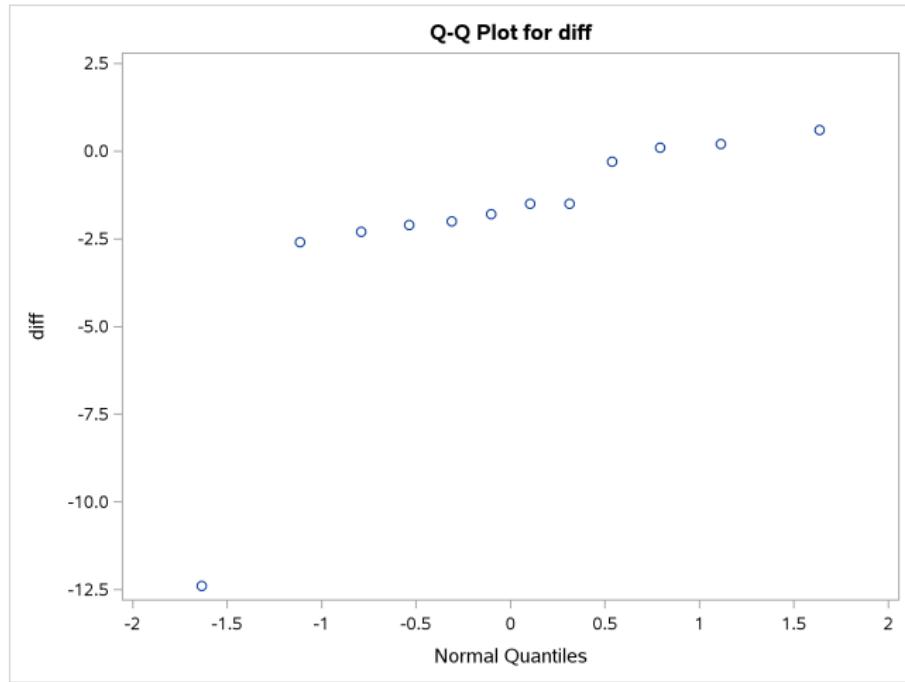
And using “ggplot”, but without line

```
ggplot(pain,aes(sample=diff))+stat_qq()
```



And in SAS...

```
proc univariate noprint;  
qqplot diff;
```

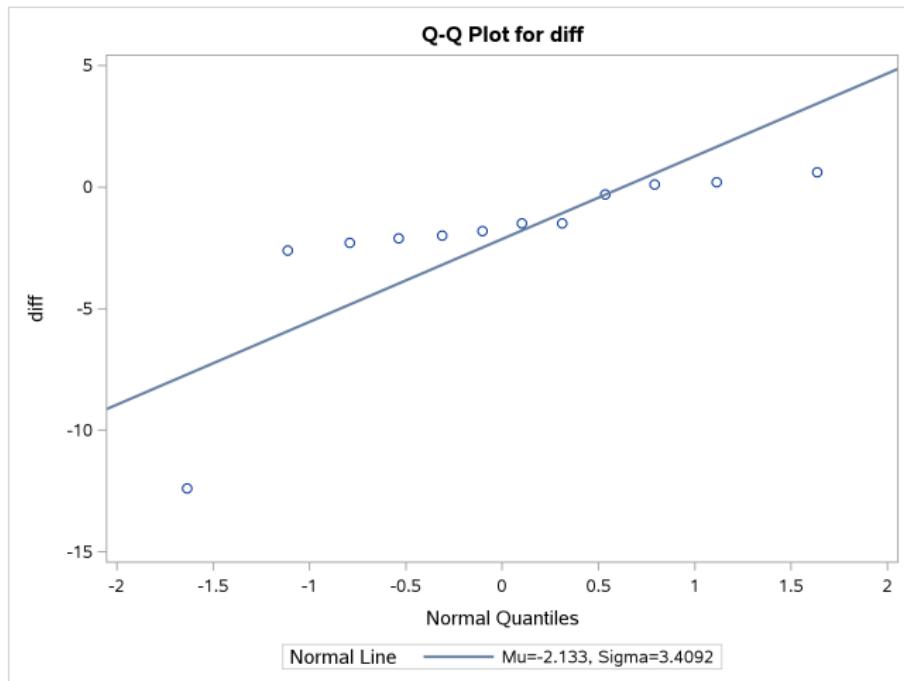


Getting a line on SAS normal quantile plot

- ▶ SAS doesn't automatically provide a line, but even without one, you see that these data are not normal because of the outlier bottom left.
- ▶ SAS can draw lines, but requires you to give a mean and SD to make the line with.
- ▶ Simplest way is to have SAS estimate them from the data, but the line is usually not very good.
- ▶ Or we can estimate them another way from IQR.

Having SAS estimate them

```
proc univariate noint;  
qqplot diff / normal(mu=est sigma=est);
```



Another way to estimate μ and σ

- ▶ Problem above is that SD was grossly inflated by outlier.
- ▶ On standard normal, quartiles about ± 0.675 :

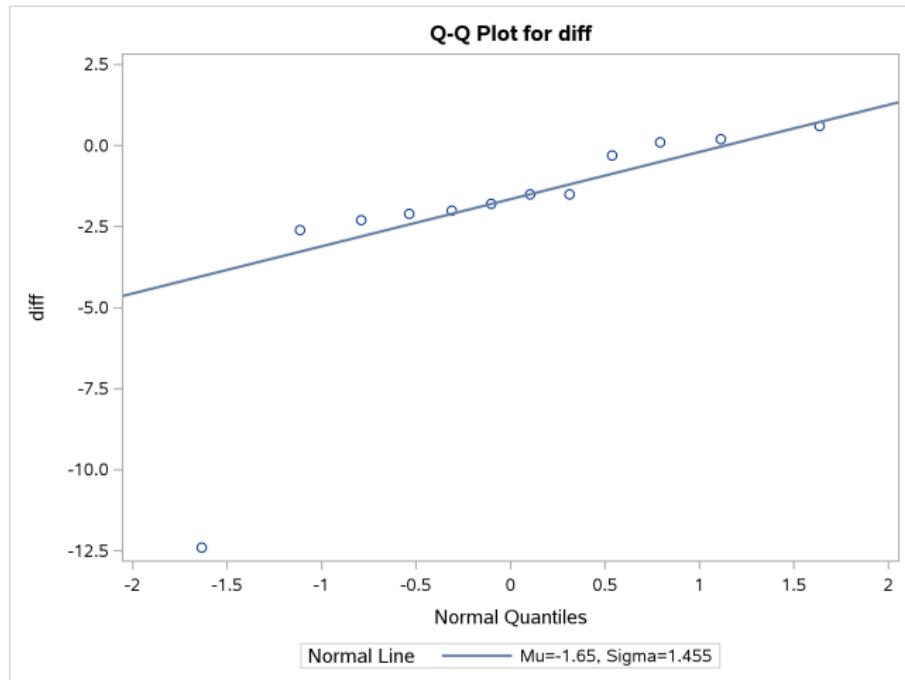
```
qnorm(0.25); qnorm(0.75)
```

```
## [1] -0.6744898  
## [1] 0.6744898
```

- ▶ So IQR of standard normal about $2(0.675) = 1.35$.
- ▶ Thus IQR of *any* normal about 1.35σ .
- ▶ Idea: estimate σ by taking sample IQR and dividing by 1.35. Not affected by outliers.
- ▶ Here, IQR is 1.95, so estimate of σ is 1.455.
- ▶ In similar spirit, estimate μ by median, -1.65 .

Using improved μ and σ

```
proc univariate noprint;  
qqplot diff / normal(mu=-1.65 sigma=1.455);
```

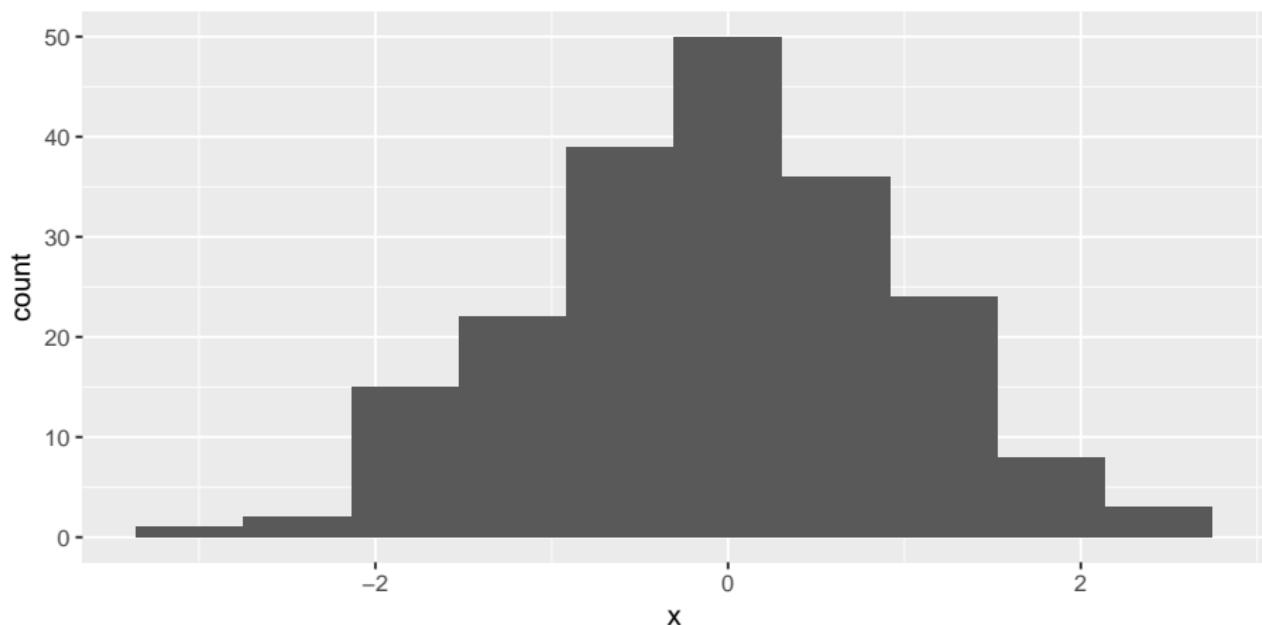


More normal quantile plots

- ▶ How straight does a normal quantile plot have to be?
- ▶ There is randomness in real data, so even a normal quantile plot from normal data won't look *perfectly* straight.
- ▶ With a small sample, can look not very straight even from normal data.
- ▶ Looking for *systematic* departure from a straight line; random wiggles ought not to concern us.
- ▶ Look at some examples where we know the answer, so that we can see what to expect.

Normal data, large sample

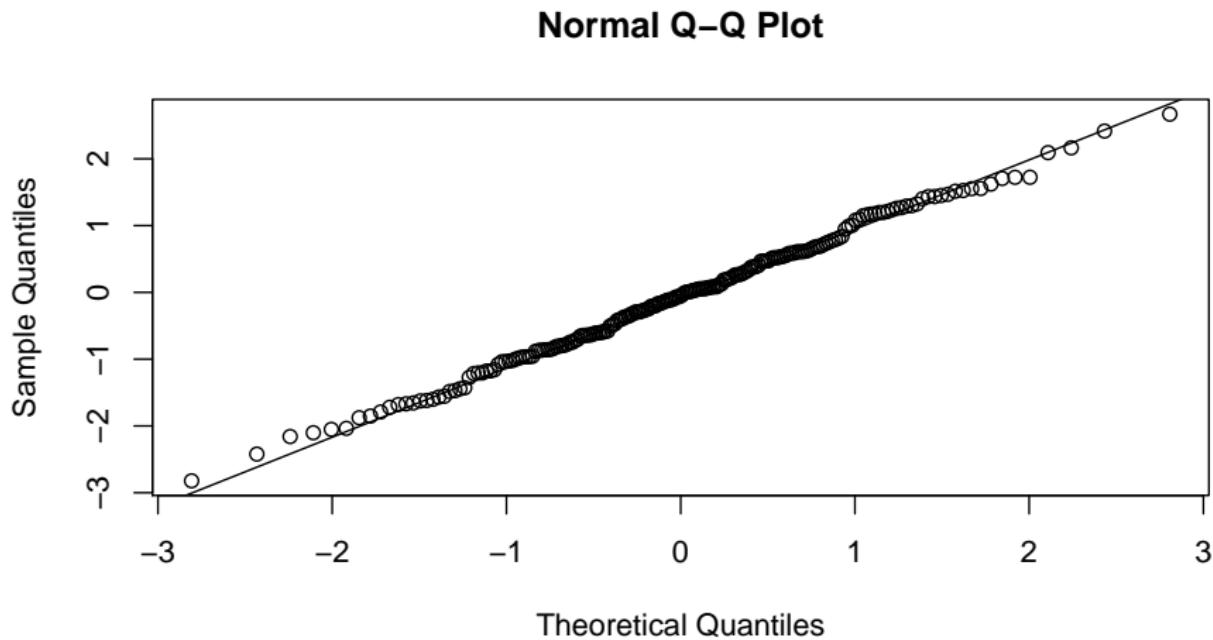
```
d=tibble(x=rnorm(200))  
ggplot(d,aes(x=x))+geom_histogram(bins=10)
```



As normal as you could wish for.

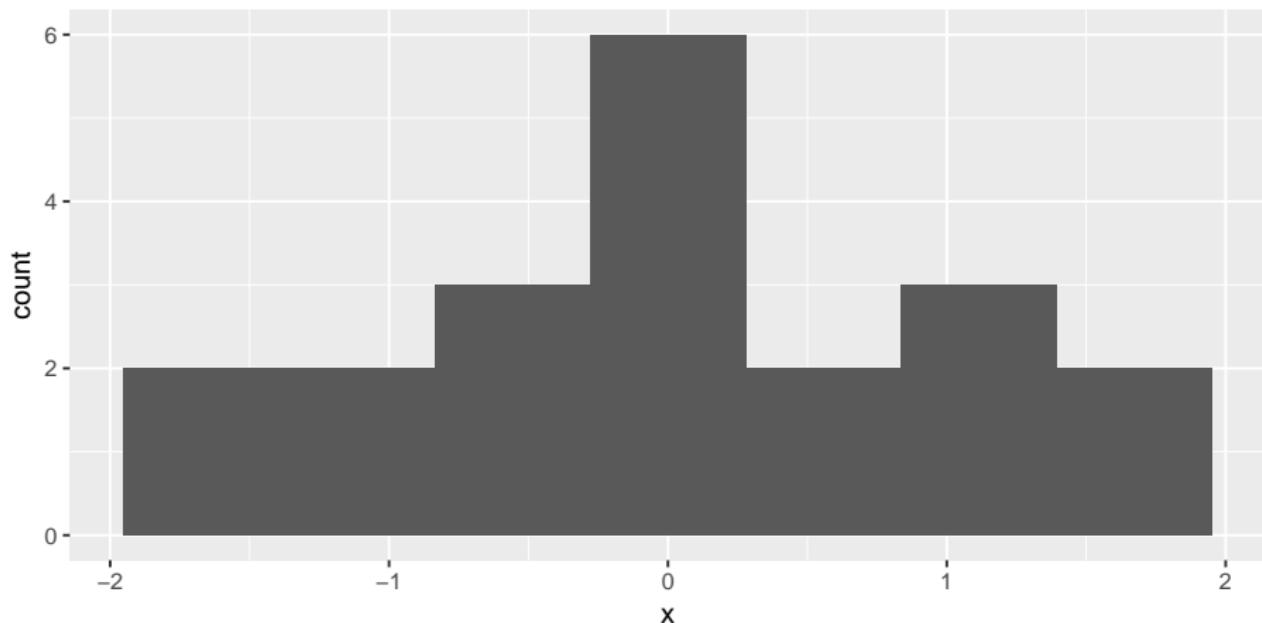
The normal quantile plot

```
qqnorm(d$x) ; qqline(d$x)
```



Normal data, small sample

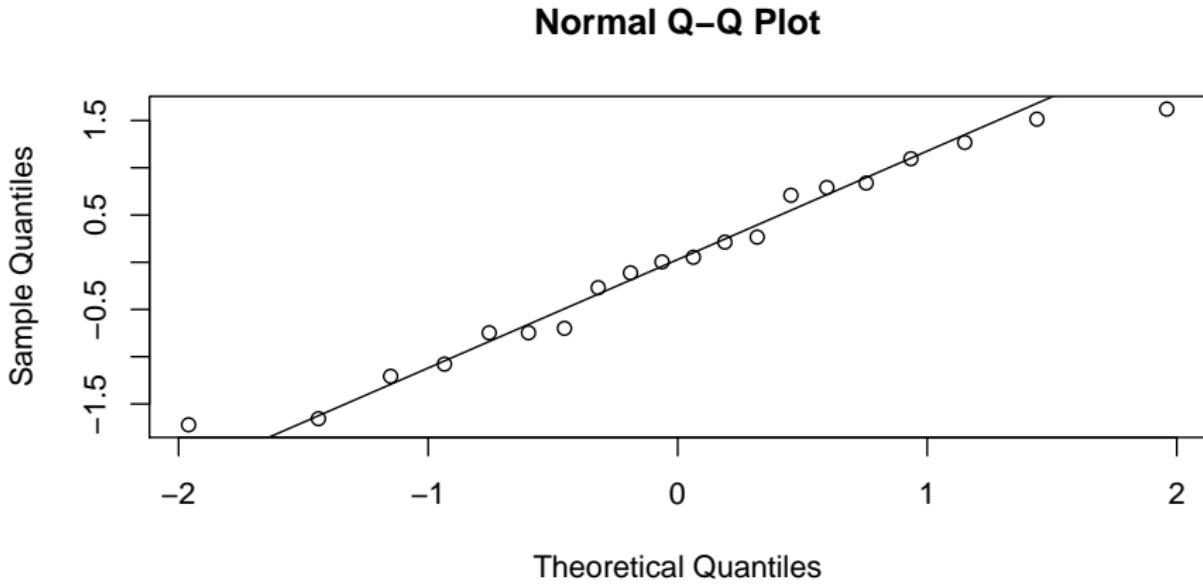
```
d=tibble(x=rnorm(20))  
ggplot(d,aes(x=x))+geom_histogram(bins=7)
```



Not so convincingly normal, but not obviously skewed.

The normal quantile plot

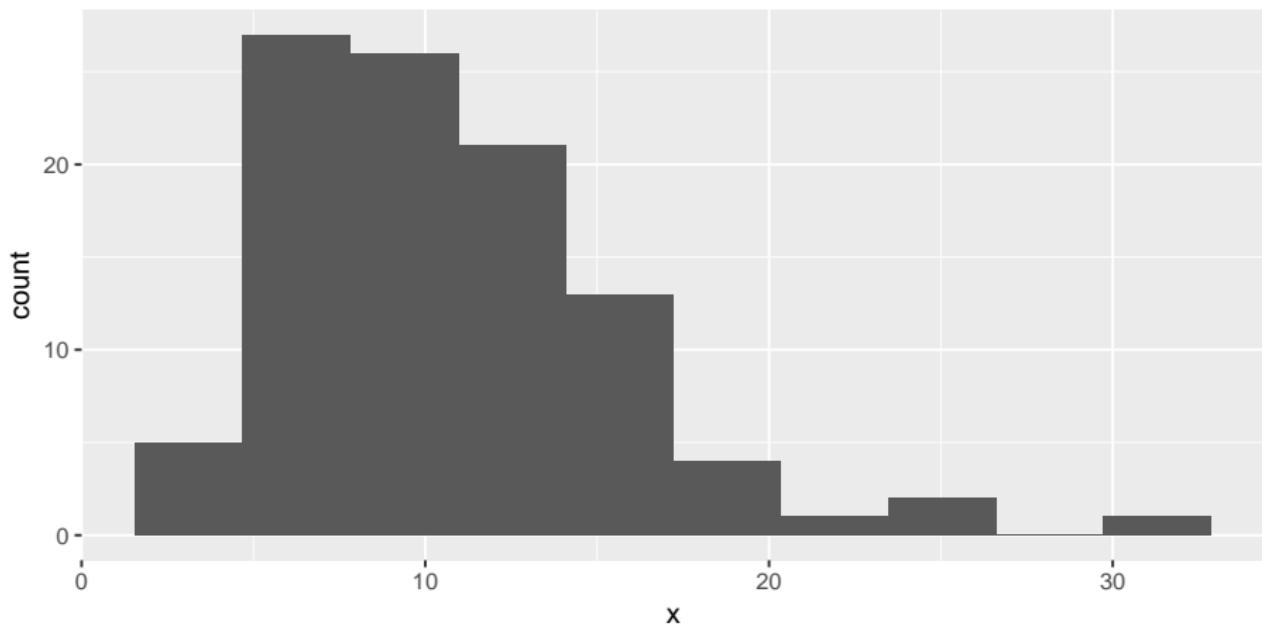
```
qqnorm(d$x) ; qqline(d$x)
```



Good, apart from the highest and lowest points being slightly off. I'd call this good.

Chi-squared data, $df = 10$

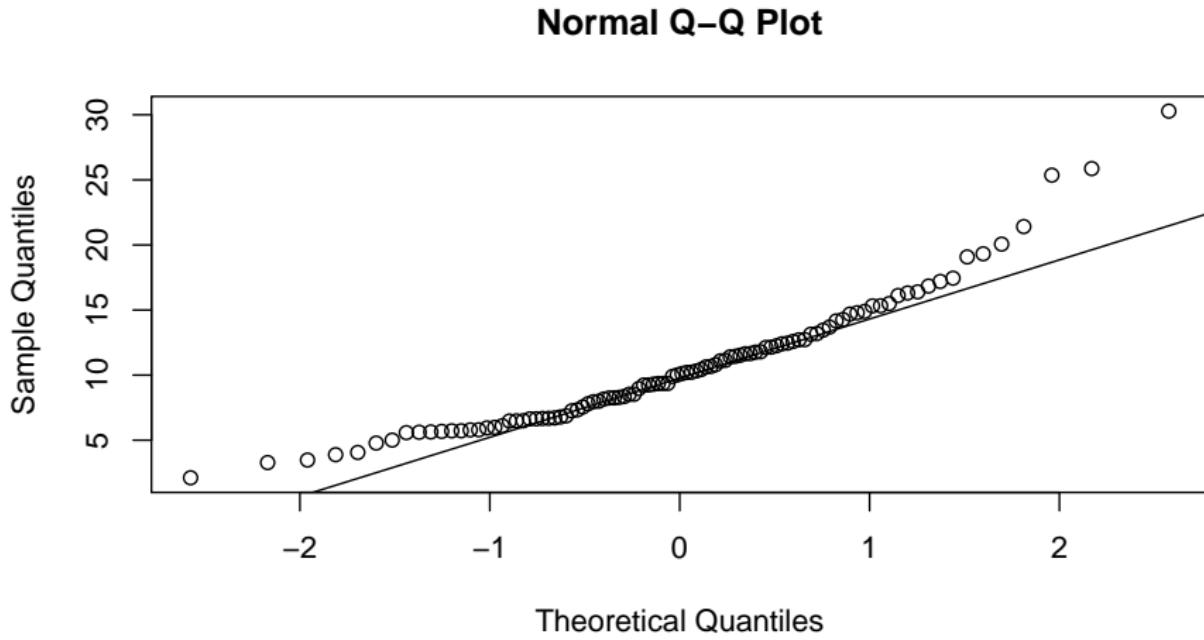
```
d=tibble(x=rchisq(100,10))  
ggplot(d,aes(x=x))+geom_histogram(bins=10)
```



Somewhat skewed to right.

The normal quantile plot

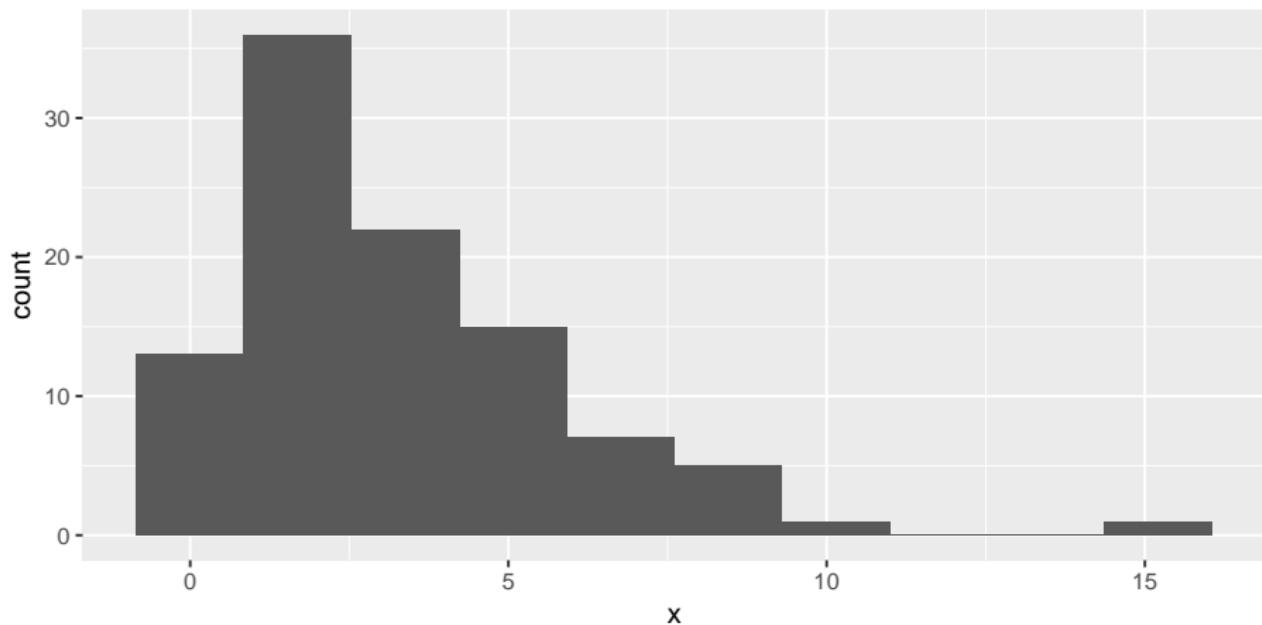
```
qqnorm(d$x) ; qqline(d$x)
```



Somewhat opening-up curve.

Chi-squared data, $df = 3$

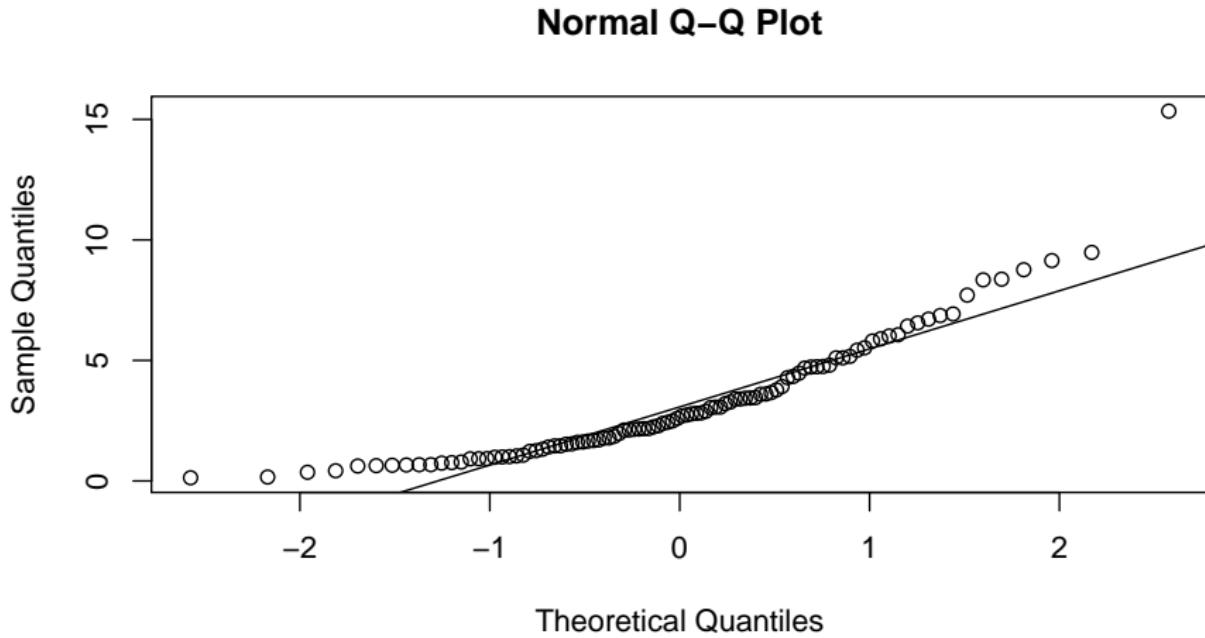
```
d=tibble(x=rchisq(100,3))  
ggplot(d,aes(x=x))+geom_histogram(bins=10)
```



Definitely skewed to right.

The normal quantile plot

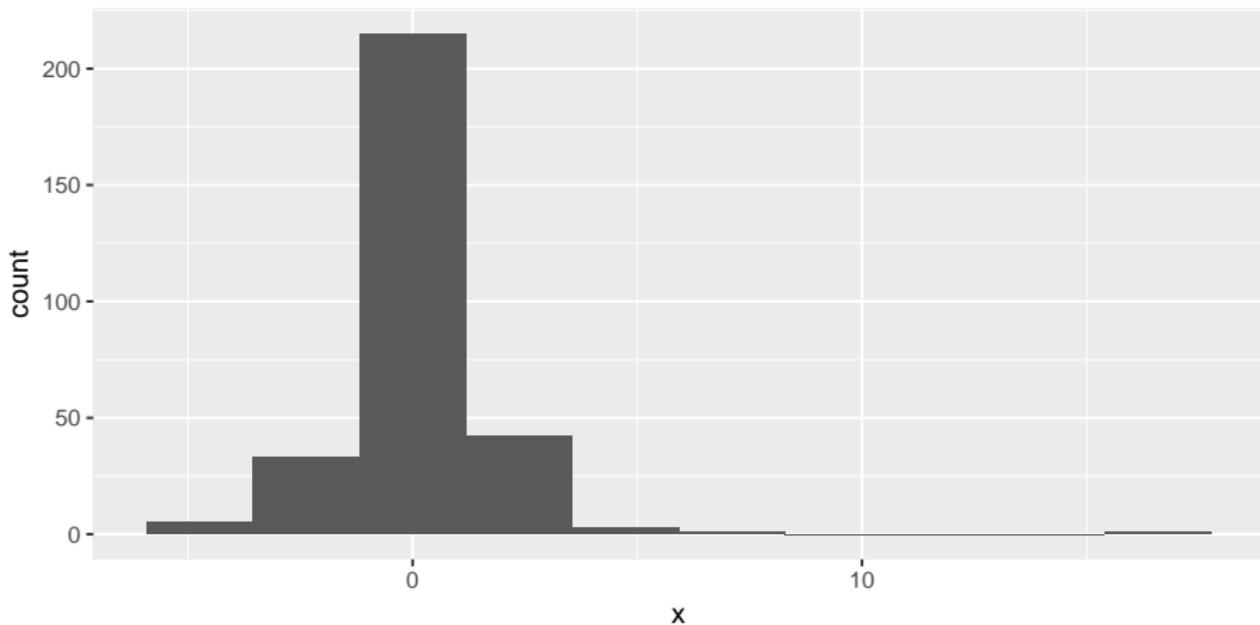
```
qqnorm(d$x) ; qqline(d$x)
```



Clear upward-opening curve.

t-distributed data, $df = 3$

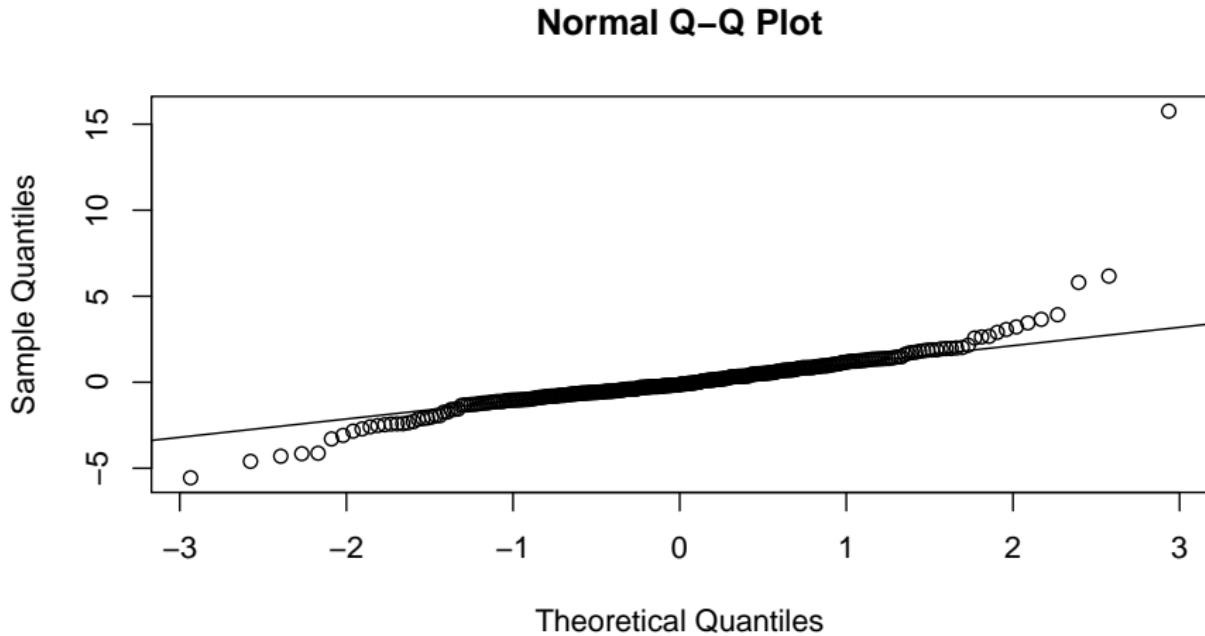
```
d=tibble(x=rt(300,3))  
ggplot(d,aes(x=x))+geom_histogram(bins=10)
```



Long tails (or a very sharp peak).

The normal quantile plot

```
qqnorm(d$x) ; qqline(d$x)
```



Low values too low and high values too high for normal.

Dealing with non-normality

- ▶ One approach: do nothing, since the t -tests are robust to at least some non-normality.
- ▶ Another: make a sign test, and test whether *median* difference is zero. In SAS, calculate differences and ask for it.

Matched-pairs sign test in SAS

- ▶ Already have differences in diff (if not, do data-and-set thing to get them), so:

```
proc univariate;  
    var diff;
```

```
The UNIVARIATE Procedure  
Variable: diff
```

```
Tests for Location: Mu0=0
```

Test	-Statistic-	-----	p Value-----
Student's t	t -2.16771	Pr > t	0.0530
Sign	M -3	Pr >= M	0.1460
Signed Rank	S -32	Pr >= S	0.0088

Results; Look back at the data

- ▶ P-value for t -test 0.0530, for sign test 0.1460.
- ▶ Sign test says “no evidence of difference between drugs A and B”, while t -test says marginal evidence of difference.
- ▶ Data (differences drug A minus drug B):

```
diff
```

```
## [1] -1.5 -2.1 -0.3  0.2 -2.6  0.1 -1.8
## [8]  0.6 -1.5 -2.0 -2.3 -12.4
```

- ▶ See the big outlier (at end of list).

Sign test in R

- ▶ In R, either use the `sign_test` function that we wrote before:

```
d=tibble(mydiff=diff)
sign_test(0,d,"mydiff")

## [1] 0.1459961
```

- ▶ Or, count the number of positive and negative differences:

```
d %>% count(mydiff>0)

## # A tibble: 2 x 2
##   `mydiff > 0`     n
##       <lgl> <int>
## 1 FALSE      9
## 2 TRUE       3
```

- ▶ and calculate the P-value as

```
sum(dbinom(0:3,12,0.5))*2

## [1] 0.1459961
```

The kids' reading data, again

```
kids %>% slice(1:15)
```

```
## # A tibble: 15 x 2
##   group score
##   <chr> <int>
## 1 t     24
## 2 t     61
## 3 t     59
## 4 t     46
## 5 t     43
## 6 t     44
## 7 t     52
## 8 t     43
## 9 t     58
## 10 t    67
## 11 t    62
## 12 t    57
## 13 t    71
## 14 t    49
## 15 t    54
```

```
kids %>% slice(16:30)
```

```
## # A tibble: 15 x 2
##   group score
##   <chr> <int>
## 1 t     43
## 2 t     53
## 3 t     57
## 4 t     49
## 5 t     56
## 6 t     33
## 7 c     42
## 8 c     33
## 9 c     46
## 10 c    37
## 11 c    43
## 12 c    41
## 13 c    10
## 14 c    42
## 15 c    55
```

```
kids %>% slice(31:44)
```

```
## # A tibble: 14 x 2
##   group score
##   <chr> <int>
## 1 c     19
## 2 c     17
## 3 c     55
## 4 c     26
## 5 c     54
## 6 c     60
## 7 c     28
## 8 c     62
## 9 c     20
## 10 c    53
## 11 c    48
## 12 c    37
## 13 c    85
## 14 c    42
```

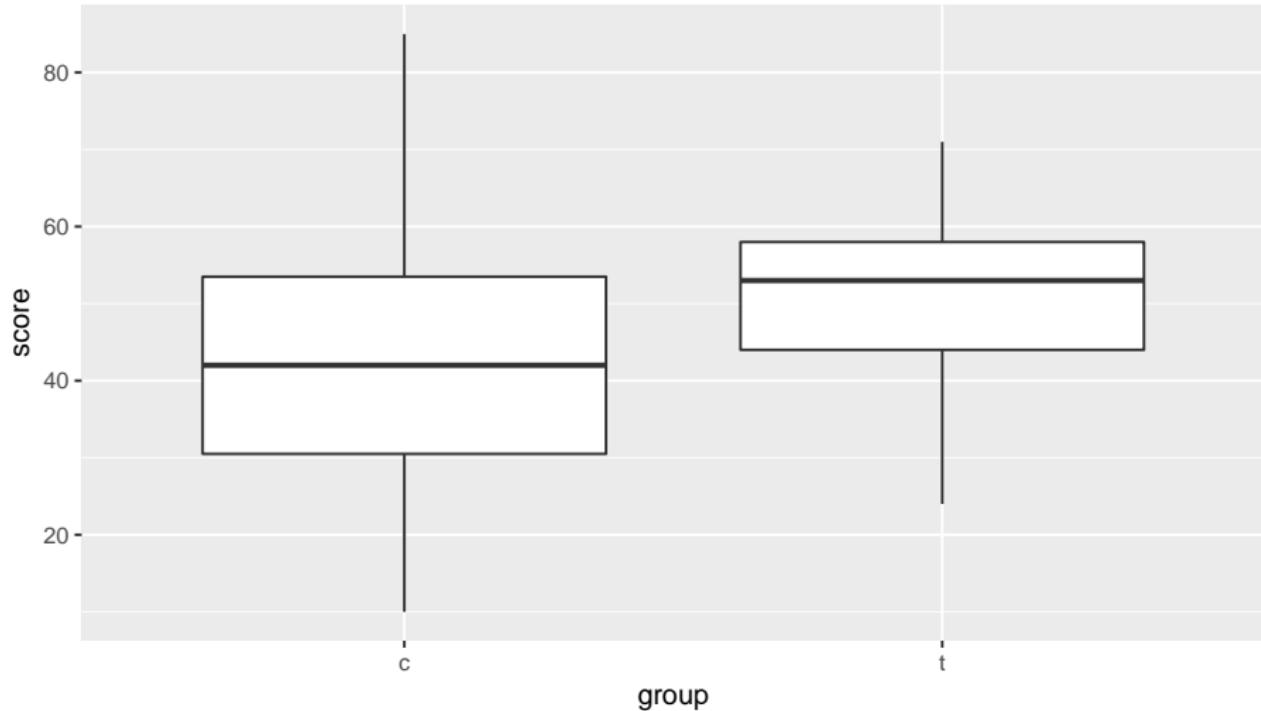
- ▶ 21 kids in “treatment”, new reading method; 23 in “control”, standard reading method.

Assessing assumptions

- ▶ We did two-sample t -test (Satterthwaite-Welch) before.
- ▶ Assumes approx. normal data *within each group*.
- ▶ Does *not* assume equal spread.
- ▶ (Pooled t -test *does* assume equal spread).
- ▶ Assess each group separately. I think boxplots good enough, since we are looking for *serious* problems with normality like outliers or clear skewness.

Boxplots for reading data

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



Comments

- ▶ These boxplots show no problems with normality. They are both more or less symmetric (equal whiskers) and there are no outliers.
- ▶ Equal spreads are questionable, but we don't need that.
- ▶ We ought be happy with the two-sample *t*-test, which was this:

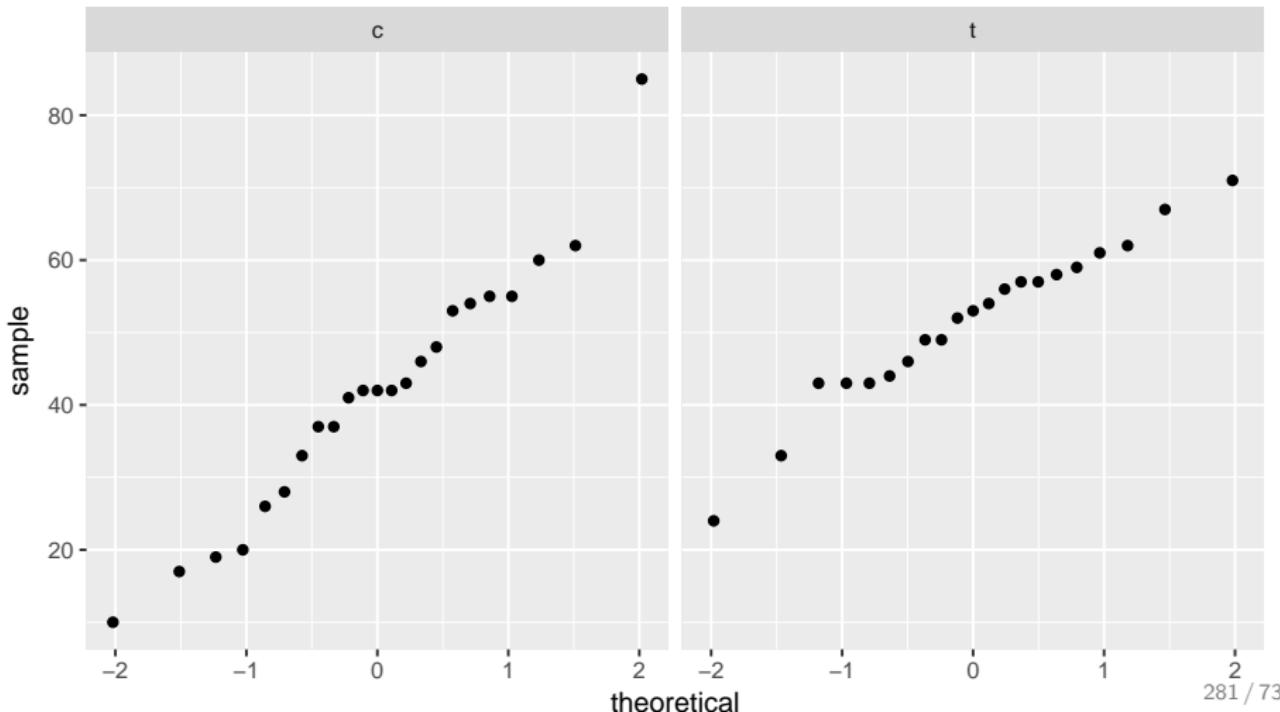
```
t.test(score~group,data=kids,alternative="less")  
  
##  
## Welch Two Sample t-test  
##  
## data: score by group  
## t = -2.3109, df = 37.855, p-value = 0.01319  
## alternative hypothesis: true difference in means is less than 0  
## 95 percent confidence interval:  
##       -Inf -2.691293  
## sample estimates:  
## mean in group c mean in group t  
##          41.52174      51.47619
```

from which we concluded that the new reading method really does help.

Facetted normal quantile plots

If you really want them, this:

```
ggplot(kids, aes(sample=score))+stat_qq()+
  facet_wrap(~group)
```



What to do if normality fails

- ▶ (On the previous page, the only indication of non-normality is the highest score in the control group, which is a little too high for normality.)
- ▶ 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:

```
dst %>% sample_n(5) # randomly sample 5 rows

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

... 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:

```
m=median(kids$score)
```

- ▶ 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), so entitled to do 1-sided test by halving P-value to get 0.017. (This step has to be justified situation-by-situation.)
- ▶ This way too, children do better at learning to read using the new method.

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).

And now in SAS

```
proc import  
  datafile='/home/ken/drpt.txt'  
  dbms=dlm  
  out=reading  
  replace;  
  delimiter=' '|;  
  getnames=yes;  
  
proc print;
```

The data (tiny)

Obs	group	score
-----	-------	-------

1	t	24
---	---	----

2	t	61
---	---	----

3	t	59
---	---	----

4	t	46
---	---	----

5	t	43
---	---	----

6	t	44
---	---	----

7	t	52
---	---	----

8	t	43
---	---	----

9	t	58
---	---	----

10	t	67
----	---	----

11	t	62
----	---	----

12	t	57
----	---	----

13	t	71
----	---	----

14	t	49
----	---	----

15	t	54
----	---	----

16	t	43
----	---	----

17	t	53
----	---	----

18	t	57
----	---	----

19	t	49
----	---	----

20	t	56
----	---	----

21	t	33
----	---	----

22	c	42
----	---	----

23	c	33
----	---	----

24	c	46
----	---	----

25	c	37
----	---	----

26	c	43
----	---	----

27	c	41
----	---	----

28	c	10
----	---	----

29	c	42
----	---	----

30	c	55
----	---	----

31	c	19
----	---	----

32	c	17
----	---	----

33	c	55
----	---	----

Doing Mood's median test

```
proc npar1way median;  
    var score;  
    class group;
```

The NPAR1WAY Procedure

Median Scores (Number of Points Above Median) for Variable score
Classified by Variable group

group	N	Sum of Scores	Expected Under H0	Std Dev Under H0	Mean Score
t	21	14.0	10.50	1.675750	0.666667
c	23	8.0	11.50	1.675750	0.347826

Average scores were used for ties.

"Sum of scores" is number of values above median in each group (checks with earlier calculation).

Results

Median Two-Sample Test

Statistic	14.0000
Z	2.0886
One-Sided Pr > Z	0.0184
Two-Sided Pr > Z	0.0367

Median One-Way Analysis

Chi-Square	4.3623
DF	1
Pr > Chi-Square	0.0367

- ▶ Same test statistic and (two-sided) P-value as R, more or less (in bottom table). Again can halve it if justified (it is justified here).
- ▶ Top table does as z-test, which gives 1-sided P-value as well.

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.

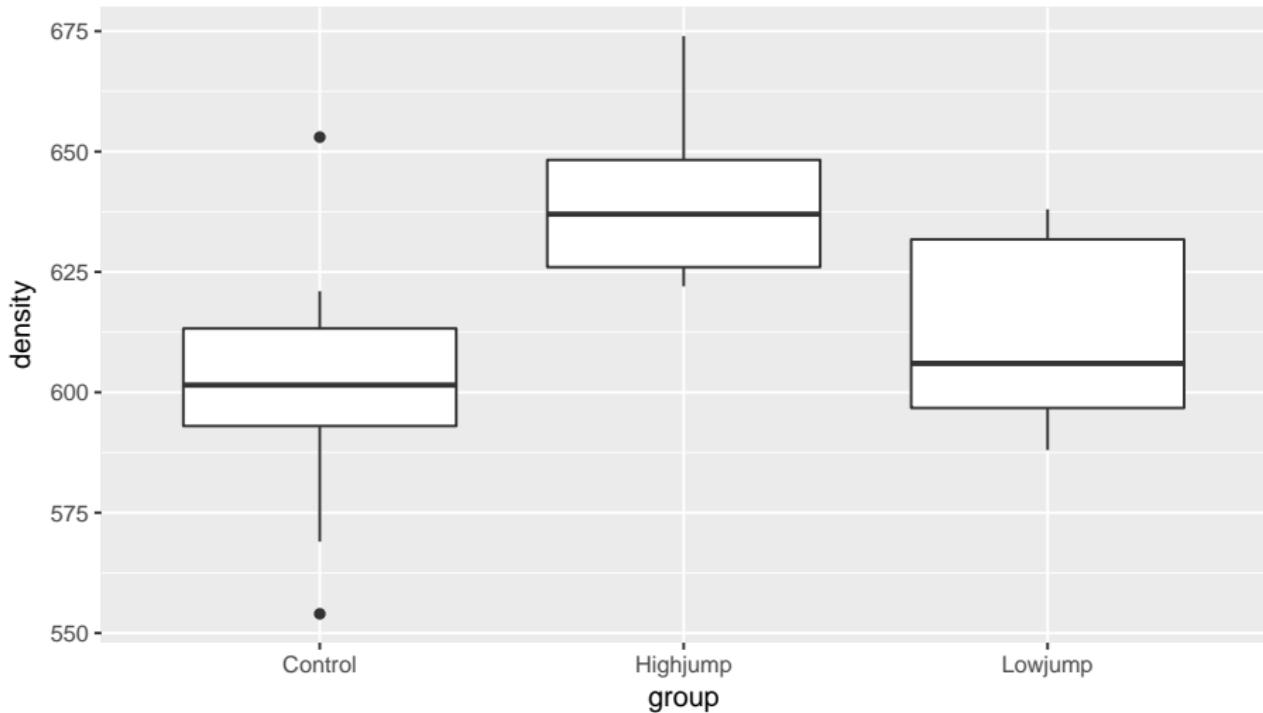
Reading the data

Values separated by spaces:

```
rats=read_delim("jumping.txt"," ")  
  
## Parsed with column specification:  
## cols(  
##   group = col_character(),  
##   density = col_integer()  
## )  
  
glimpse(rats)  
  
## Observations: 30  
## Variables: 2  
## $ group <chr> "Control", "Control", "Contro...  
## $ density <int> 611, 621, 614, 593, 593, 653,...
```

Boxplots

```
ggplot(rats,aes(y=density,x=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.

Same thing in SAS

Read in data and do ANOVA:

```
proc import  
  datafile='/home/ken/jumping.txt'  
    dbms=dlm  
    out=rats  
    replace;  
  delimiter=' '|;  
  getnames=yes;  
  
proc anova;  
  class group;  
  model density=group;
```

Results (some)

The ANOVA Procedure

Dependent Variable: density

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	7433.86667	3716.93333	7.98	0.0019
Error	27	12579.50000	465.90741		
Corrected Total	29	20013.36667			
Source	DF	Anova SS	Mean Square	F Value	Pr > F
group	2	7433.86667	3716.93333	7.98	0.0019

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:

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

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 equal to zero
## 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 eq
## 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 eq
## 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 no longer significant at $\alpha = 0.01$.

Same stuff in SAS

```
proc anova;  
  class group;  
  model density=group;  
  means group / tukey;
```

Tukey output (some)

The ANOVA Procedure

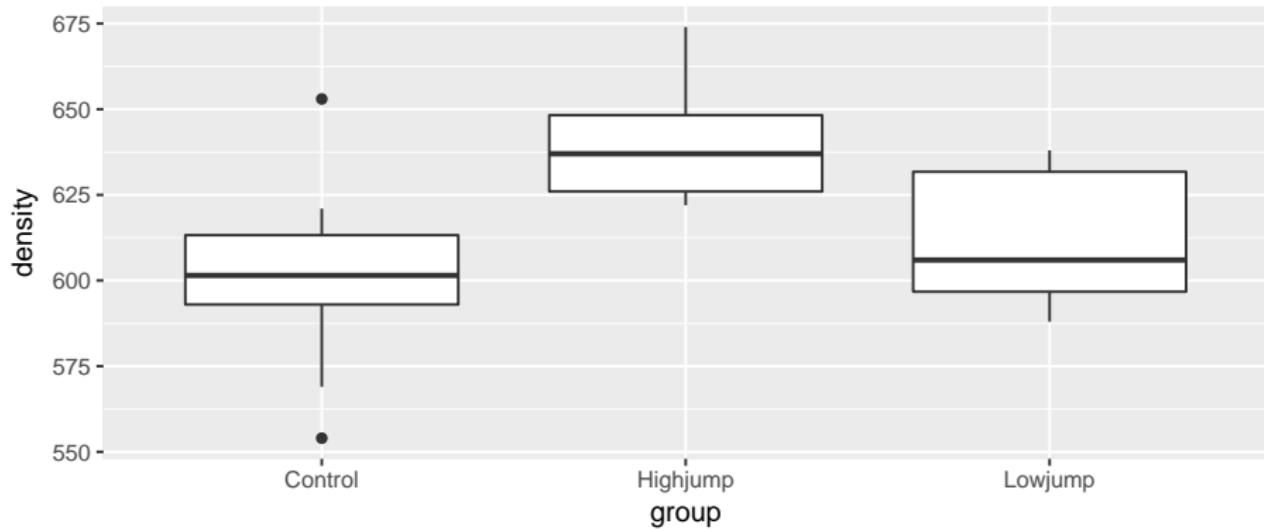
Tukey's Studentized Range (HSD) Test for density

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	group
A	638.700	10	Highjum
B	612.500	10	Lowjump
B			
B	601.100	10	Control

Checking assumptions

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



Assumptions:

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

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

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

```
m=median(rats$density) ; 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
```

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

Mood's median test 2/2

- ▶ 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.
- ▶ 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.

Welch ANOVA

- ▶ For these data, Mood's median test probably best because we doubt both normality and equal spreads.
- ▶ 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
```

- ▶ Appropriate Tukey-equivalent here called Games-Howell (not done here).

Back to SAS: Mood's median test

```
proc npar1way median;  
  var density;  
  class group;
```

Output part 1, confirming number of density values above grand median in each group:

The NPAR1WAY Procedure

Median Scores (Number of Points Above Median) for Variable density
Classified by Variable group

group	N	Sum of Scores	Expected Under H0	Std Dev Under H0	Mean Score
Control	10	1.0	5.0	1.313064	0.10
Lowjump	10	4.0	5.0	1.313064	0.40
Highjum	10	10.0	5.0	1.313064	1.00

Average scores were used for ties.

Rest of output

Median One-Way Analysis

Chi-Square	16.2400
DF	2
Pr > Chi-Square	0.0003

Because there are more than 2 groups, we only get the chi-squared test. This is (strongly) significant, so the median bone densities in the three groups are not all the same.

Welch ANOVA in SAS

That appears to go this way. The instruction to do the Welch ANOVA goes on the means line, where the Tukey would go if we were doing that:

```
proc anova;  
    class group;  
    model density=group;  
    means group / hovtest=levene welch;
```

Ignore the usual ANOVA in the output, and look right to the end:

Results

The ANOVA Procedure

Welch's ANOVA for density

Source	DF	F Value	Pr > F
group	2.0000	8.82	0.0023
Error	17.4054		
Level of	-----density-----		
group	N	Mean	Std Dev
Control	10	601.100000	27.3636011
Highjum	10	638.700000	16.5935061
Lowjump	10	612.500000	19.3290225

The Welch's ANOVA is the same as R's. Also note that the P-values for the regular ANOVA (0.0019) and the Welch ANOVA (0.0023) are almost identical here, so allowing for unequal spreads has made almost no difference, even though the group SDs look different.

Section 7

Reports and R Markdown

Communicating your results

Being a statistician means being able to do several things:

1. Obtain and process the data for analysis
2. Do a suitable analysis
3. Check that the analysis was reasonable
4. Communicate your findings to the world

The last part is perhaps the *most* important, because analyses do not exist in isolation; you do an analysis to answer a question, and the answer to the question is the most important thing.

This is true whether you are in the corporate world, answering to a boss, or in graduate school, where you will eventually have to convince your thesis committee (and, by extension, the academic world) that what you have done is interesting, statistically sound *and* important.

Reports

- ▶ Final step of your process is to write a report. This is a sales job, because you have to convince your readers that what you have done is worth their time reading.
- ▶ Writing a report requires good language skills. You cannot become a good statistician without that.
- ▶ This is why so many of my questions end “explain briefly”. You need to learn to provide a complete *and concise* explanation of what your results tell you and why.
- ▶ Reports are usually structured in a similar way, as shown on next page.

Report structure

Introduction: tell your readers about your problem and what you hope to find out. You need to provide enough explanation for the reader to know what you're trying to achieve. You might also want to refer to what other people have done.

Methods: Where the data came from, and how it was collected (describing the technology that was used, if any). Scientific people call this section "Methods". If you needed to do any work to get the data into the right form, this is also the place to describe that.

Analysis and results: It is not enough to *give* the analysis; you have to explain what you are doing and what made you do it. The results should be described in a matter-of-fact way (the opinions come in the next section).

Conclusions: What does the analysis tell you about your problem? Place the results in context. Offer (supported) opinions about what the results mean, to you and the world.

A typical journal article

<http://jap.physiology.org/content/100/3/839>

Effect of low-repetition jump training on bone mineral density in young women

Takeru Kato, Toru Terashima, Takenori Yamashita, Yasuhiko Hatanaka, Akiko Honda, Yoshihisa Umemura

Journal of Applied Physiology Published 1 March 2006 Vol. 100 no. 3, 839-843 DOI: 10.1152/japplphysiol.00666.2005

Title and authors, with journal and page numbers, so that you have everything you need to refer to it.

Abstract

Journal articles typically begin with Abstract that summarizes question and gives highlights of results and conclusion:

Abstract

The hypothesis of the present study was that low-repetition and high-impact training of 10 maximum vertical jumps/day, 3 times/wk would be effective for improving bone mineral density (BMD) in ordinary young women. Thirty-six female college students, with mean age, height, and weight of 20.7 ± 0.7 yr, 158.9 ± 4.6 cm, and 50.4 ± 5.5 kg, respectively, were randomly divided into two groups: jump training and a control group. After the 6 mo of maximum vertical jumping exercise intervention, BMD in the femoral neck region significantly increased in the jump group from the baseline (0.984 ± 0.081 vs. 1.010 ± 0.080 mg/cm²; $P < 0.01$), although there was no significant change in the control group (0.985 ± 0.0143 vs. 0.974 ± 0.134 mg/cm²). And also lumbar spine (L₂-4) BMD significantly increased in the jump training group from the baseline (0.991 ± 0.115 vs. 1.015 ± 0.113 mg/cm²; $P < 0.01$), whereas no significant change was observed in the control group (1.007 ± 0.113 vs. 1.013 ± 0.110 mg/cm²). No significant interactions were observed at other measurement sites, Ward's triangle, greater trochanter, and total hip BMD. Calcium intakes and accelerometry-determined physical daily activity showed no significant difference between the two groups. From the results of the present study, low-repetition and high-impact jumps enhanced BMD at the specific bone sites in young women who had almost reached the age of peak bone mass.

Abstract tells you whether paper is worth reading.

Introduction

PHYSICAL ACTIVITY MAY PLAY an important role in maximizing bone mass during childhood and may have long-lasting benefits on bone health. Because peak bone mass is thought to be attained by the end of the third decade, the early adult years may be the final opportunity for its augmentation (13). Skeletal unloading, such as long bed rest, immobilization, and microgravity environment, lead to bone loss, whereas the positive effects of physical exercise on bone mass is generally acknowledged. It has been shown that dynamic loading is more effective for increasing bone mineral density (BMD) than static loading (15). Furthermore, the strain rate is more important than the number of loading trials (22).

Prepubescent children (7.5–8.2 yr) who have not yet reached their peak bone mass have shown significant development in lumbar spine bone mass by 100 two-footed drop landings off of a 61-cm-high box 3 times/wk compared with a randomized control group (6). Bassey and Ramsdale (1) found a significant increase in femoral BMD after 6 mo of 50 jumps daily among premenopausal

Introduction begins with plain-English first sentence. The numbers in brackets are references to what other people have said.

Materials and methods

MATERIALS AND METHODS

Subjects and groups.

One hundred twenty-eight female college students with experience in weighted food records were asked to take part in this study, and 48 students volunteered to participate. The subjects completed the questionnaire containing information about menstrual cycle, pregnancy, past and current physical activity, smoking habit, as well as background information, including history of bone diseases, medication use, and bone fracture. The entry criteria for subjects were eumenorrheic, nonpregnant, no oral medication, nonsmoker, no regular high-impact training, with no medical or surgical problems likely to affect bone metabolism or providing contraindications to exercise. Six subjects were excluded because they had regularly engaged in high-impact sports such as volleyball, basketball, and tennis in the last 5 yr.

Forty-two subjects were randomly divided into two groups, jump training or a control group. In compliance with the university's Institutional Review Board policy, the purpose and all experimental procedures were explained, and written, informed consent was then obtained from each subject. The study was approved by the local health research review board. The subjects were permitted to withdraw at any time for any reason. Bone measurements were conducted at initial baseline and

The subjects. Experiments on humans require “ethical approval” and certain procedures need to be followed.

Taking measurements

BMD and deoxypyridinoline measurements.

Bone mineral density (g/cm^2) was assessed, using dual-energy X-ray absorptiometry (ALOKA, DCS-3000), in the lumbar spine (L2–4, anterior-posterior view) and the right proximal femur. The femoral neck, Ward's triangle, and greater trochanter of the proximal femur were selected for analysis according to the manufacturer's software. The same radiographer made the initial and final dual-energy X-ray absorptiometry measurements, and the groups (jumping or control) were blinded.

...

Maximum vertical jump and ground reaction force.

Maximum vertical jump height was measured by a jump height measuring device (Takei Scientific Instruments, Jump-MD) in both the pre- and postexercise program. At both visits for measuring jump height, subjects jumped vertically at least twice with maximum voluntary effort, and the best performance was recorded. The subjects stood at the center of the circular thin rubber mat (38 cm in diameter). The jumper attached the height-measuring device to her waist. The jump height measuring device and the circular mat were attached by a rope so that the traveling distance from the standing position to the maximum height reached at waist level could be measured. When the jumpers could not land stably within the circular rubber mat, the jumpers had to perform another trial.

Results (a)

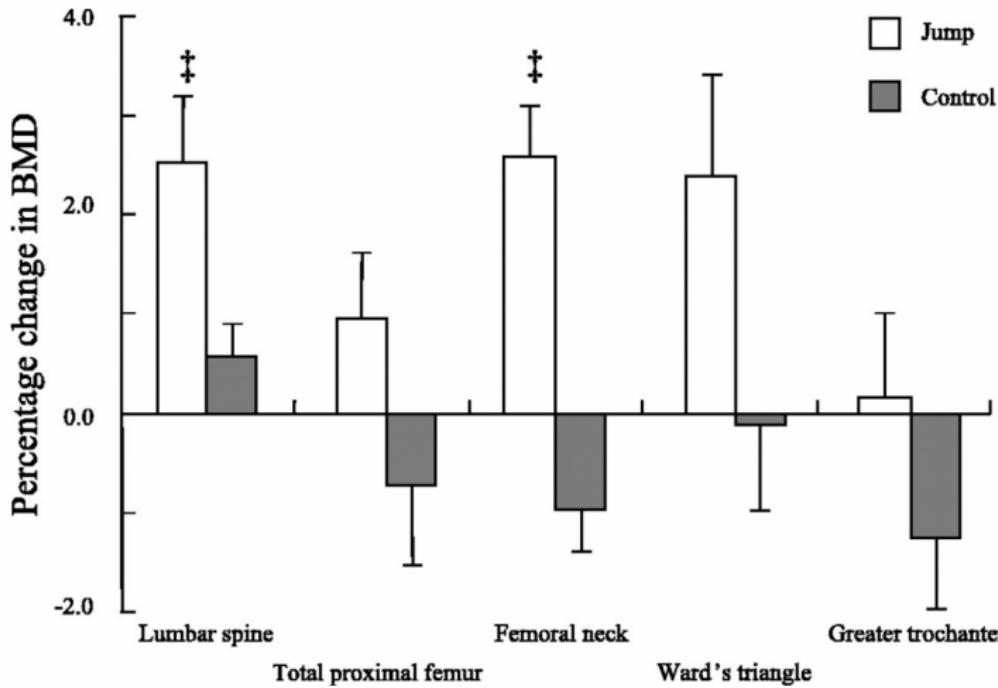
RESULTS

Descriptive statistics indicated that the initial height, age, and movement count were not significantly different between the jump and control groups (**Table 1**). In the jump group, compliance (82%) at jump training was averaged 2.5 times/wk. After the 6 mo of exercise intervention, body weight (BW) was significantly decreased within groups ($P < 0.05$) (**Table 1**). The maximum vertical jump height was significantly improved within groups ($P < 0.05$) (**Table 1**). The average vertical jump height with the same measuring method in the similar age group among Japanese women was 41.0–41.8 cm (**17**). Calcium intake and urinary DPD showed no significant differences between and within groups (**Table 1**).

... noting that the two groups were not significantly different before the study, but changed in important respects over time.

Results also shown in table.

Results (b)



Graph showing that bone mass density has changed greatly as a result of the jumping. (Graphs are always good.)

Conclusions (selected)

DISCUSSION

The most important observation made in the present study was that jump training of 10 jumps/day, 30 jumps/wk significantly increased BMD at the femoral neck ($P < 0.05$), whereas BMD in the control group remained unchanged after 6 mo of exercise intervention. Other investigators have shown that loading with many repetitions at one time had a relatively small additional effect on bones compared with loading of only 10–40 repetitions (23, 26). After many repetitions of mechanical loading on bones, the mechanosensor might show decreased sensitivity (19, 20). Thus its effectiveness as a bone stimulus would appear similar even with fewer repetitions. The loading interval may be another important factor associated with mechanosensor sensitivity (20). A high

...
Our findings are in agreement with those of Kohrt et al. (12), who observed a positive high-impact loading effect on femoral neck BMD in postmenopausal women. The training program involved

...
There are, however, some limitations of the present study. First, although an important issue, our study did not measure strain in the proximal femur during the maximum vertical jump. Bassey et al. (3) measured the compressive axial forces in an instrumented massive femoral implant and

Note use of (relatively) plain English, description of most important findings, comparisons to other work, and admission of limitations.

References to other work (some)

REFERENCES

1. ↗ Bassey EJ and Ramsdale SJ. Increase in femoral bone density in young women following high-impact exercise. *Osteoporos Int* 4: 72–75, 1994. [CrossRef](#) [PubMed](#) [Web of Science](#)
2. ↗ Bassey EJ, Rothwell MC, Littlewood JJ, and Pye DW. Pre- and postmenopausal women have different bone mineral density responses to the same high-impact exercise. *J Bone Miner Res* 13: 1805–1813, 1998. [CrossRef](#) [PubMed](#) [Web of Science](#)
3. ↗ Bassey EJ, Littlewood JJ, and Taylor SJ. Relations between compressive axial forces in an instrumented massive femoral implant, ground reaction forces, and integrated electromyographs from vastus lateralis during various 'osteogenic' exercises. *J Biomech* 30: 213–223, 1997. [CrossRef](#) [PubMed](#) [Web of Science](#)
4. ↗ Beverly MC, Rider TA, Evans MJ, and Smith R. Local bone mineral response to brief exercise that stresses the skeleton. *BMJ* 22: 233–235, 1989.
5. ↗ Chang S, Sipila S, Taffe DR, Puolakka J, and Suominen H. Change in bone mass distribution induced by hormone replacement therapy and high-impact physical exercise in post-menopausal women. *Bone* 31: 126–135, 2002. [PubMed](#)
6. ↗ Fuchs RK, Bauer JJ, and Snow CM. Jumping improves hip and lumbar spine bone mass in prepubescent children: a randomised controlled trial. *J Bone Miner Res* 16: 148–156 2001. [CrossRef](#) [PubMed](#) [Web of Science](#)
7. ↗ Heinonen A, Kannus P, Sievanen H, Pasanen M, Rinne M, Uusi-rasi K, and Vuori I. Randomised controlled trial of effect of high-impact exercise on selected risk factors for osteoporotic fractures. *Lancet* 348: 1343–1347, 1996. [CrossRef](#) [PubMed](#) [Web of Science](#)

Reproducibility

- ▶ The paper we just looked at contained a lot of information.
- ▶ Partly, this was to show that the researchers followed proper procedure (important with human subjects).
- ▶ Also allows anyone to do analysis on same data and get same results (**reproducible**).
- ▶ Allows anyone to follow same procedure on own data and see if results same (**replication**).
- ▶ As statisticians, we need our *own* reports to be reproducible, and to be able to replicate them on different data.
- ▶ Strategy for this: write reports so that they *include the code* and a way of running it.
- ▶ This can be done in R (R Markdown). Can also be done in SAS with more trouble (machinery called Statrep).

R Markdown introduction

- ▶ R Studio: File, New File, R Markdown. Give it a title (anything) and put your name in as Author. Click OK (accepting other defaults).
- ▶ A template document opens up. This is R Markdown code.
- ▶ R Markdown is a “markup language” like HTML: it contains a mixture of actual text and instructions to format text.
- ▶ To see what the final document looks like, click Knit. You’ll be invited to save the Markdown document somewhere. Do so.
- ▶ In R Studio, a “preview” window will open with the actual formatted report. It has text, a hyperlink, some boldface text, R code and output, and a graph. Go back to the R Markdown code, and see if you can figure out how those things appeared in the output.

R Markdown text formatting

- ▶ To make a section heading, put ## before the title:

```
## Section heading
```

Can use more or fewer #: more of them makes a lower level subheading (H1 through H6 in HTML).

- ▶ To make *italic* text, do this:

This text is in italics.

- ▶ To make **boldface** text:

Bold text.

- ▶ To start a new paragraph, leave a blank line or end a line with *two* spaces.
- ▶ More at <https://www.rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf>.

Code Chunks

- ▶ The “magic” of R Markdown is that you can include code, and in output you will see **code plus output the code produced**.
- ▶ This means that in your report, *the output must be up-to-date* because it was literally produced from that code when the document was knitted.
- ▶ To insert a code chunk, Click Insert and select R. (You can also use other languages if they are installed on your computer.) You see this in your document:

```
```{r}
```

```
...
```

- ▶ In between the two lines with “backticks”, insert any R code you like. It will be run and the code plus output inserted in your document.
- ▶ To not show the code, only the output, change the top line to read

```
```{r,echo=F}
```

Why this is better than copy-and-paste

- ▶ This seems like more trouble than copying-and-pasting the code and output into a Word document. Why should I do it?
- ▶ You are *guaranteed* to get code and output that matches up. If you copy-and-paste, how do you know you remembered to copy the most recent run of your code? (When you change your code, you have to remember to run it again, *and* to re-copy the output.)
- ▶ Anyone else, or you yourself later, can make the document again from the R Markdown file (and the data files), or run the same code on a new data file. This makes the analysis reproducible. Any procedure that depends on copy-pasting the right thing is not reproducible.
- ▶ Bosses have a habit of asking for small changes to a document. You make those small changes in the R Markdown file, knit again, and you have your results with minimal fuss.

Other output formats

- ▶ The basic (and fastest) form of output is HTML. This is best for while you're writing the report, or if you want to put it on a web site.
- ▶ Word .doc output: when you think you've finished writing (slow). If you want to make changes, *edit the R Markdown*, close the Word doc and re-knit.
- ▶ PDF, if you have \LaTeX installed (looks like \LaTeX without any \LaTeX coding)
- ▶ Presentations of various flavours (makes suitable HTML out of the R Markdown).

Markdown and L^AT_EX

- ▶ Presentation, or Knit to PDF: can use L^AT_EX formulas, like

```
 $$x = \{ -b \pm \sqrt{b^2 - 4ac} \} / (2a) $$
```

which comes out as

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- ▶ I encourage you to learn L^AT_EX! Much better than Word for mathematical, scientific (and for that matter academic) documents.
- ▶ Include SAS in L^AT_EX using statrep. More in a moment.

Sweave

- ▶ A more sophisticated R/ \LaTeX connection via Sweave: can embed R code chunks in \LaTeX document. In R Studio, File-New-R Sweave. This produces a skeleton \LaTeX document. Insert code chunk in usual way (looks different).
- ▶ File extension is .Rnw.
- ▶ Go to Tools, Global Options, Sweave, and make sure top option says “weave Rnw files using knitr” (should only need to do this once).
- ▶ Knit in R Studio as usual, or at command line via

```
Rscript -e 'knitr::knit("filename.Rnw")'
```

- ▶ This produces .tex file, which is then processed as usual (two steps altogether).
- ▶ Make any changes to the .Rnw file and run both steps again.
- ▶ Command-line version works with Makefile.

Statrep

- ▶ SAS's version of Sweave: include SAS code in L^AT_EX document, run when document produced.
- ▶ Uses SAS Studio run on virtual machine on your computer (not on online SAS Studio unless you want extra difficulty).
- ▶ Setup:
 - ▶ SAS Studio running on virtual machine on your computer (or full-version SAS on your computer).
 - ▶ Folder on your computer linked to virtual machine (the folder you refer to as /folders/myfolders on virtual machine), if using virtual-machine SAS.
 - ▶ Construct document on your computer in that linked folder.
 - ▶ Link: <http://support.sas.com/rnd/app/papers/statrep.html>.

Workflow (1/3)

- ▶ Get template \LaTeX document with right headers (over) in correct folder.
- ▶ Code: \LaTeX as usual. Code chunks and output separated (since SAS output usually rather long).
- ▶ Insert data step or proc import by enclosing SAS code in `\begin{Datastep}` and `\end{Datastep}`. (Can be used for anything that produces no output.)
- ▶ Insert proc step (running a procedure to do analysis, whose output you want) by enclosing SAS code in Sascode environment like this. Choose a label name for `store=` by which to grab output (use any label name you like):

```
\begin{Sascode}[store=labelx]
  proc means;
\end{Sascode}
```

Extra L^AT_EX headers

Put these above your \begin{document}:

```
\usepackage{statrep}
\usepackage{parskip,xspace}
\newcommand*\Statrep{\mbox{\textsf{StatRep}}\xspace}
\newcommand*\Code[1]{\texttt{\textbf{#1}}}
\newcommand*\cs[1]{\texttt{\textbf{\textbackslash#1}}}
\def\SRrootdir{/folders/myfolders}
\def\SRmacropath{/folders/myfolders/statrep_macros.sas}
```

You may not need all of these, but try them first. The last two tell Statrep where to put/find SAS code generated.

Workflow (2/3)

- ▶ Output of two kinds: text output (tables of numbers) and graphics output (graphs/plots).
- ▶ Specify *which* output you want by using the label you created thus, plus create an extra label as shown:

```
\Listing[store=labelx]{labelxx}  
\Graphic[store=labelx]{labelxy}
```

- ▶ Listing and Graphic are *both singular* and *both start with capital letter*.
- ▶ The store= is the same label as in the Sascode (how Statrep knows which output to grab). Label at the end *must be distinct* (names of files created in background).

Workflow (3/3)

- ▶ To produce .pdf, run \LaTeX twice, with a run of SAS in between:
- ▶ First time: if you look at .pdf, you see placeholders for SAS output (when it is produced). Also a file ending in _SR.sas, which is SAS code to produce output.
- ▶ Go to SAS Studio (on virtual machine); see this file in list of files. Open this file.
- ▶ “Submit” it. This won’t produce any visible output, but its output will be saved in files. If there are errors, fix them *in the \LaTeX document* and run \LaTeX again.
- ▶ Second time: run \LaTeX again. In the .pdf, the SAS output placeholders are replaced by actual output. Done. (If you don’t like anything, eg. there are SAS errors/no SAS output, *change the \LaTeX and repeat.*)

Grabbing only some of the output

- ▶ SAS output tends to be long. It is divided into sections (each containing one table or graph). Each one has a name.
- ▶ To find out what the names are, after you have run the _SR.sas file in SAS Studio, look in the Log tab. Look for the code whose output you want only part of, like this:

```
1864      proc ttest h0=0;
1865          var diff;
1866
1867      %endoutput(id)

NOTE: PROCEDURE TTEST used (Total process time):
      real time      0.14 seconds
      cpu time       0.14 seconds
```

... Continued

- ▶ Scroll down in Log tab until you see something like this:

```
1869      %write(id,store=id,type=listing)
NOTE: Processing document id.
Note: New page for \Ttest#1\diff#1\PT#1
Note: Writing Listing file : /folders/myfolders/
Note: Writing Listing file : /folders/myfolders/
Note: Writing Listing file : /folders/myfolders/
Objects          Type  Status  Group
Ttest.diff.PT    Note   Selected  1
Ttest.diff.Statistics  Table  Selected  1
Ttest.diff.ConfLimits  Table  Selected  2
Ttest.diff.TTests    Table  Selected  3
Ttest.diff.SummaryPanel Graph   . 
Ttest.diff.QQPlot    Graph   . 
----
```

- ▶ Note the `store=` at the top, and the table of “objects” at the bottom.
- ▶ In the “objects” table, each object you can select has a 3-part name (left column). You need only *last* part.

Grabbing what you want

- ▶ The “type” column says whether it’s a table of numbers (use Listing) or a graph (use Graphic).
- ▶ Note that I got this via `\begin{Sascode}[store=id]`.
- ▶ To get just the confidence intervals, do this:

```
\Listing[store=id,objects=conflimits]{idd}
```

- ▶ To get just the normal quantile plot, this:
- ```
\Graphic[store=id,objects=qqplot]{ide}
```

- ▶ To get more than one object (of same type), separate by **space**:

```
\Listing[store=id,objects=statistics ttests]{idf}
```

which gets both summary statistics and *t*-tests, but nothing else.

## Using the Log tab to find errors

- ▶ Sometimes SAS output unexpectedly missing in final document.  
Usually because of a SAS error somewhere.
- ▶ To find, go to Log tab in SAS Studio.
- ▶ Search for text **ERROR:**, the first occurrence. This is the first place that SAS got confused (there may be others). You might find that SAS Studio has opened this for you.
- ▶ Fix this, in .tex document, and run again.
- ▶ Repeat as necessary.

## Processing R and SAS in same document

- ▶ You might have a document that contains both R and SAS code to be run (and output inserted). How to handle *both*?
- ▶ Process (most easily with Makefile):
  - ▶ Create Sweave document with name like `test.Rnw` (with `.Rnw` extension), *in the folder attached to the SAS virtual machine*. Add R code chunks with `<>>=` format and SAS code chunks in Statrep style.
  - ▶ Use `knitr` to process the R. This makes file `test.tex` with R code run and output inserted.
  - ▶ Run  $\text{\LaTeX}$  once to produce `test_SR.sas`.
  - ▶ Run `test_SR.sas` in SAS Studio (on virtual machine).
  - ▶ Run  $\text{\LaTeX}$  again to grab SAS output.
  - ▶ Check final `test.pdf`: should have both R and SAS output.
  - ▶ Make any changes to `test.Rnw` and repeat whole process.

## Section 8

Tidying and organizing data

## Tidying data

- ▶ Data rarely come to us as we want to use them.
- ▶ Before we can do analysis, typically have organizing to do.
- ▶ This is typical of ANOVA-type data, “wide format”:

```
pig feed1 feed2 feed3 feed4
1 60.8 68.7 92.6 87.9
2 57.0 67.7 92.1 84.2
3 65.0 74.0 90.2 83.1
4 58.6 66.3 96.5 85.7
5 61.7 69.8 99.1 90.3
```

- ▶ 20 pigs are randomly allocated to one of four feeds. At the end of the study, the weight of each pig is recorded, and we want to know whether there are any differences in mean weights among the feeds.
- ▶ Problem: want the weights all in *one* column, with 2nd column labelling which feed each weight was from. Untidy!

## Tidy and untidy data (Wickham)

- ▶ Data set easier to deal with if:
  - ▶ each observation is one *row*
  - ▶ each variable is one *column*
  - ▶ each type of observation unit is one *table*
- ▶ Data arranged this way called “tidy”; otherwise called “untidy”.
- ▶ For the pig data, response variable is weight, but scattered over 4 columns, which are *levels* of a factor feed.
- ▶ Want all the weights in *one* column, with a second column feed saying which feed that weight goes with.
- ▶ Then we can run aov.

## Reading in data

```
pigs1=read_delim("pigs1.txt"," ")

Parsed with column specification:
cols(
pig = col_integer(),
feed1 = col_double(),
feed2 = col_double(),
feed3 = col_double(),
feed4 = col_double()
)

glimpse(pigs1)

Observations: 5
Variables: 5
$ pig <int> 1, 2, 3, 4, 5
$ feed1 <dbl> 60.8, 57.0, 65.0, 58.6, 61.7
$ feed2 <dbl> 68.7, 67.7, 74.0, 66.3, 69.8
$ feed3 <dbl> 92.6, 92.1, 90.2, 96.5, 99.1
$ feed4 <dbl> 87.9, 84.2, 83.1, 85.7, 90.3
```

## Gathering up the columns

- ▶ This is a very common reorganization, and the magic “verb” is `gather`:

```
pigs2=pigs1 %>% gather(feed,weight,feed1:feed4)
glimpse(pigs2)

Observations: 20
Variables: 3
$ pig <int> 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4
$ feed <chr> "feed1", "feed1", "feed1", "feed1", "feed1",
$ weight <dbl> 60.8, 57.0, 65.0, 58.6, 61.7, 68.7, 67.7
```

- ▶ `pigs2` is now in “long” format, ready for analysis.
- ▶ Anatomy of `gather`: what makes the columns different (different feeds), what makes them the same (all weights), which columns to combine.
- ▶ Column `pig` now is 1–5 4 times (number of pig within each feed group).

## ... and finally, the analysis

- ▶ which is just what we saw before:

```
weight.1=aov(weight~feed,data=pigs2)
summary(weight.1)

Df Sum Sq Mean Sq F value Pr(>F)
feed 3 3521 1173.5 119.1 3.72e-11 ***
Residuals 16 158 9.8

Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

- ▶ The mean weights of pigs on the different feeds are definitely not all equal.
- ▶ So we run Tukey to see which ones differ (over).

# Tukey

TukeyHSD(weight.1)

```
Tukey multiple comparisons of means
95% family-wise confidence level
##
Fit: aov(formula = weight ~ feed, data = pigs2)
##
$feed
diff lwr upr p adj
feed2-feed1 8.68 3.001038 14.358962 0.0024000
feed3-feed1 33.48 27.801038 39.158962 0.0000000
feed4-feed1 25.62 19.941038 31.298962 0.0000000
feed3-feed2 24.80 19.121038 30.478962 0.0000000
feed4-feed2 16.94 11.261038 22.618962 0.0000013
feed4-feed3 -7.86 -13.538962 -2.181038 0.0055599
```

All of the feeds differ! To find the best and worst, get mean weight by feed group (over).

## Mean weights by feed

I borrowed an idea from later to put the means in descending order:

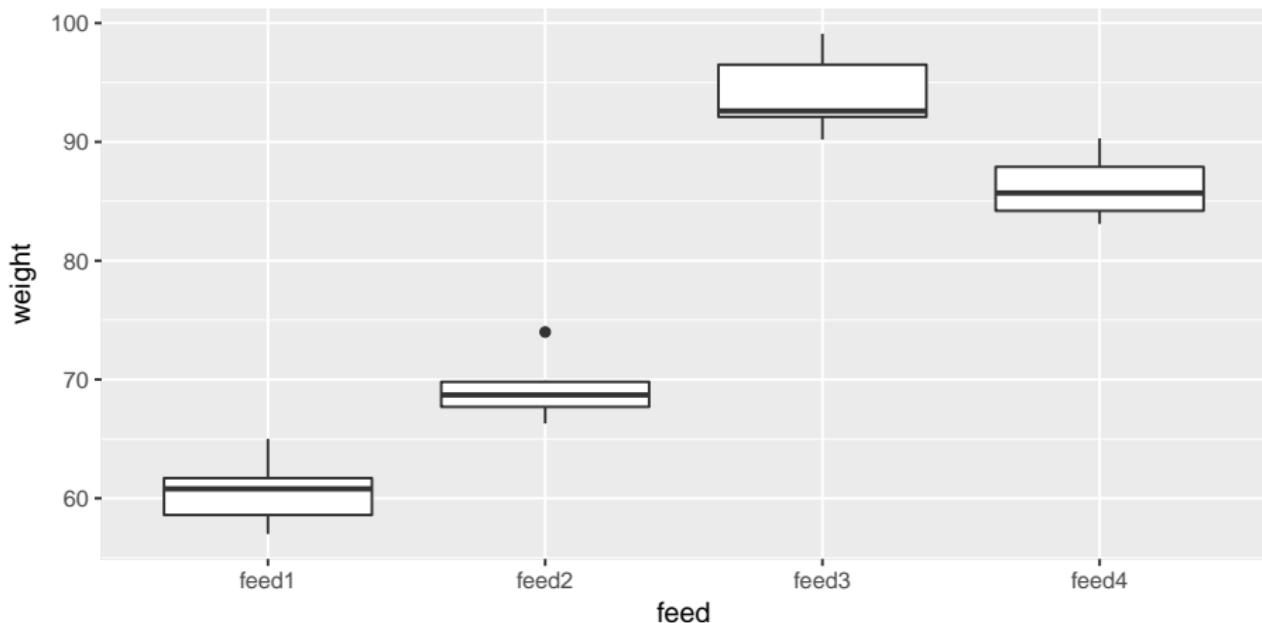
```
pigs2 %>% group_by(feed) %>%
 summarize(mean_weight=mean(weight)) %>%
 arrange(desc(mean_weight))

A tibble: 4 x 2
feed mean_weight
<chr> <dbl>
1 feed3 94.10
2 feed4 86.24
3 feed2 69.30
4 feed1 60.62
```

Feed 3 is best, feed 1 worst.

## Should we have any concerns about the ANOVA?

```
ggplot(pigs2, aes(x=feed, y=weight)) + geom_boxplot()
```



Feed 2 has an outlier, but there are only 5 pigs in each group, and the conclusion is so clear that I am OK with this.

## Tuberculosis

- ▶ The World Health Organization keeps track of number of cases of various diseases, eg. tuberculosis.
- ▶ Some data:

```
tb=read_csv("tb.csv")
```

*## Parsed with column specification:*

```
cols(
.default = col_integer(),
iso2 = col_character()
)
```

*## See spec(...) for full column specifications.*

- ▶ Variables (see over): country (abbreviated), year. Then number of cases for each gender and age group, eg. m1524 is males aged 15–24. Also mu and fu, where age is unknown.
- ▶ Lots of missings. Want to get rid of.

# The data

```
tb

A tibble: 5,769 x 22
iso2 year m04 m514 m014 m1524 m2534 m3544 m4554
<chr> <int> <int> <int> <int> <int> <int> <int> <int>
1 AD 1989 NA NA NA NA NA NA NA
2 AD 1990 NA NA NA NA NA NA NA
3 AD 1991 NA NA NA NA NA NA NA
4 AD 1992 NA NA NA NA NA NA NA
5 AD 1993 NA NA NA NA NA NA NA
6 AD 1994 NA NA NA NA NA NA NA
7 AD 1996 NA NA 0 0 0 4 1
8 AD 1997 NA NA 0 0 1 2 2
9 AD 1998 NA NA 0 0 0 1 0
10 AD 1999 NA NA 0 0 0 1 1
... with 5,759 more rows, and 13 more variables:
m5564 <int>, m65 <int>, mu <int>, f04 <int>,
f514 <int>, f014 <int>, f1524 <int>, f2534 <int>,
f3544 <int>, f4554 <int>, f5564 <int>, f65 <int>,
fu <int>
```

## Gather the gender-age group columns

```
tb2=tb %>% gather(genage,freq,m04:fu,na.rm=T)
```

- ▶ what makes the columns-to-be-gathered different, then
- ▶ what makes them the same, then
- ▶ the columns to gather, then (optionally)
- ▶ get rid of the missing values.

## Results

tb2

```
A tibble: 35,750 x 4
iso2 year genage freq
* <chr> <int> <chr> <int>
1 AD 2005 m04 0
2 AD 2006 m04 0
3 AD 2008 m04 0
4 AE 2006 m04 0
5 AE 2007 m04 0
6 AE 2008 m04 0
7 AG 2007 m04 0
8 AL 2005 m04 0
9 AL 2006 m04 1
10 AL 2007 m04 0
... with 35,740 more rows
```

## Separating

- ▶ 4 columns, but 5 variables, since genage contains both gender and age group. Split that up using `separate`.
- ▶ `separate` needs 3 things:
  - ▶ what to separate (no quotes needed),
  - ▶ what to separate into (here you *do* need quotes),
  - ▶ how to split.
- ▶ For “how to split”, here “after first character”:

```
tb3=tb2 %>% separate(genage, c("gender", "age"), 1)
```

## Tidied tuberculosis data

tb3

```
A tibble: 35,750 x 5
iso2 year gender age freq
* <chr> <int> <chr> <chr> <int>
1 AD 2005 m 04 0
2 AD 2006 m 04 0
3 AD 2008 m 04 0
4 AE 2006 m 04 0
5 AE 2007 m 04 0
6 AE 2008 m 04 0
7 AG 2007 m 04 0
8 AL 2005 m 04 0
9 AL 2006 m 04 1
10 AL 2007 m 04 0
... with 35,740 more rows
```

## In practice...

- ▶ instead of doing the pipe one step at a time, you *debug* it one step at a time, and when you have each step working, you use that step's output as input to the next step, thus:

```
tb3=tb %>% gather(genage,freq,m04:fu,na.rm=T) %>%
 separate(genage,c("gender","age"),1)
```

- ▶ You can split the R code over as many lines as you like, *as long as each line is incomplete*, so that R knows more is to come.
- ▶ I like to put the pipe symbol on the end of the line.

## Total tuberculosis cases by year

```
s=tb3 %>% group_by(year) %>% summarize(cases=sum(freq))
```

```
s %>% slice(1:10)
```

```
A tibble: 10 x 2
year cases
<int> <int>
1 1980 959
2 1981 805
3 1982 824
4 1983 786
5 1984 814
6 1985 799
7 1986 754
8 1987 670
9 1988 682
10 1989 654
```

```
s %>% slice(11:20)
```

```
A tibble: 10 x 2
year cases
<int> <int>
1 1990 549
2 1991 544
3 1992 512
4 1993 492
5 1994 750
6 1995 513971
7 1996 635705
8 1997 733204
9 1998 840389
10 1999 994517
```

```
s %>% slice(21:29)
```

```
A tibble: 9 x 2
year cases
<int> <int>
1 2000 1147819
2 2001 1234483
3 2002 1522702
4 2003 1859986
5 2004 2184264
6 2005 2360363
7 2006 2514638
8 2007 2584071
9 2008 2648589
```

Something very interesting happened between 1994 and 1995.

## Some weather data

```
weather=read_csv("weather.csv")

Parsed with column specification:
cols(
.default = col_double(),
id = col_character(),
year = col_integer(),
month = col_integer(),
element = col_character(),
d9 = col_character(),
d12 = col_character(),
d18 = col_character(),
d19 = col_character(),
d20 = col_character(),
d21 = col_character(),
d22 = col_character(),
d24 = col_character()
)

See spec(...) for full column specifications.
```

# The data

weather

```
A tibble: 22 x 35
id year month element d1 d2 d3 d4
<chr> <int> <int> <chr> <dbl> <dbl> <dbl> <dbl>
1 MX17004 2010 1 tmax NA NA NA NA
2 MX17004 2010 1 tmin NA NA NA NA
3 MX17004 2010 2 tmax NA 27.3 24.1 NA
4 MX17004 2010 2 tmin NA 14.4 14.4 NA
5 MX17004 2010 3 tmax NA NA NA NA
6 MX17004 2010 3 tmin NA NA NA NA
7 MX17004 2010 4 tmax NA NA NA NA
8 MX17004 2010 4 tmin NA NA NA NA
9 MX17004 2010 5 tmax NA NA NA NA
10 MX17004 2010 5 tmin NA NA NA NA
... with 12 more rows, and 27 more variables: d5 <dbl>,
d6 <dbl>, d7 <dbl>, d8 <dbl>, d9 <chr>, d10 <dbl>,
d11 <dbl>, d12 <chr>, d13 <dbl>, d14 <dbl>, d15 <dbl>,
d16 <dbl>, d17 <dbl>, d18 <chr>, d19 <chr>, d20 <chr>,
d21 <chr>, d22 <chr>, d23 <dbl>, d24 <chr>, d25 <dbl>,
d26 <dbl>, d27 <dbl>, d28 <dbl>, d29 <dbl>, d30 <dbl>,
d31 <dbl>
```

## The columns

These are daily weather records for a weather station in Mexico.

`id`: identifier for this weather station (always same here)

`year, month`: obvious

`element`: whether temperature given was daily max or daily min

`d1, d2, ...`: day of the month from 1st to 31st.

Numbers in data frame all temperatures (for different days of the month),  
so first step is

```
weather2 = weather %>% gather(day, temperature, d1:d31, na.rm=T)
```

## Results

```
weather2
```

```
A tibble: 66 x 6
id year month element day temperature
* <chr> <int> <int> <chr> <chr> <chr>
1 MX17004 2010 12 tmax d1 29.9
2 MX17004 2010 12 tmin d1 13.8
3 MX17004 2010 2 tmax d2 27.3
4 MX17004 2010 2 tmin d2 14.4
5 MX17004 2010 11 tmax d2 31.3
6 MX17004 2010 11 tmin d2 16.3
7 MX17004 2010 2 tmax d3 24.1
8 MX17004 2010 2 tmin d3 14.4
9 MX17004 2010 7 tmax d3 28.6
10 MX17004 2010 7 tmin d3 17.5
... with 56 more rows
```

## The days

- ▶ Column element contains *names of two different variables*, that should each be in separate column.
- ▶ Distinct from eg. `m1524` in tuberculosis data, that contained *levels of two different factors*, handled by `separate`.
- ▶ Untangling names of variables handled by `spread`:

```
weather3 = weather %>%
 gather(day, temperature, d1:d31, na.rm=T) %>%
 spread(element, temperature)
```

## Result

```
weather3
```

```
A tibble: 33 x 6
id year month day tmax tmin
* <chr> <int> <int> <chr> <chr> <chr>
1 MX17004 2010 1 d30 27.8 14.5
2 MX17004 2010 2 d11 29.7 13.4
3 MX17004 2010 2 d2 27.3 14.4
4 MX17004 2010 2 d23 29.9 10.7
5 MX17004 2010 2 d3 24.1 14.4
6 MX17004 2010 3 d10 34.5 16.8
7 MX17004 2010 3 d16 31.1 17.6
8 MX17004 2010 3 d5 32.1 14.2
9 MX17004 2010 4 d27 36.3 16.7
10 MX17004 2010 5 d27 33.2 18.2
... with 23 more rows
```

## Further improvements

- ▶ We have tidy data now, but can improve things further.
- ▶ `mutate` creates new columns from old (or assign back to change a variable).
- ▶ Would like the numerical dates. `separate` works, but also produces column named `d` whose value is always `d`. Instead pull out number as below.
- ▶ `select` keeps columns (or drops, with minus). Station id has no value to us:

```
weather4 = weather %>%
 gather(day, temperature, d1:d31, na.rm=T) %>%
 spread(element, temperature) %>%
 mutate(day=parse_number(day)) %>%
 select(-id)
```

## Results

```
weather4
```

```
A tibble: 33 x 5
year month day tmax tmin
<int> <int> <dbl> <chr> <chr>
1 2010 1 30 27.8 14.5
2 2010 2 11 29.7 13.4
3 2010 2 2 27.3 14.4
4 2010 2 23 29.9 10.7
5 2010 2 3 24.1 14.4
6 2010 3 10 34.5 16.8
7 2010 3 16 31.1 17.6
8 2010 3 5 32.1 14.2
9 2010 4 27 36.3 16.7
10 2010 5 27 33.2 18.2
... with 23 more rows
```

## Final step(s)

- ▶ Make year-month-day into proper date.
- ▶ Keep only date, tmax, tmin:

```
weather5 = weather %>%
 gather(day, temperature, d1:d31, na.rm=T) %>%
 spread(element, temperature) %>%
 mutate(day=parse_number(day)) %>%
 select(-id) %>%
 unite(datestr, c(year, month, day), sep="-") %>%
 mutate(date=as.Date(datestr)) %>%
 select(c(date, tmax, tmin))
```

## Final results

```
weather5
```

```
A tibble: 33 x 3
date tmax tmin
<date> <chr> <chr>
1 2010-01-30 27.8 14.5
2 2010-02-11 29.7 13.4
3 2010-02-02 27.3 14.4
4 2010-02-23 29.9 10.7
5 2010-02-03 24.1 14.4
6 2010-03-10 34.5 16.8
7 2010-03-16 31.1 17.6
8 2010-03-05 32.1 14.2
9 2010-04-27 36.3 16.7
10 2010-05-27 33.2 18.2
... with 23 more rows
```

## Plotting the temperatures

- ▶ Plot temperature against date joined by lines, but with separate lines for max and min.
- ▶ ggplot requires something like

```
ggplot(weather5, aes(x=date, y=temperature))
```

only we have *two* temperatures, one a max and one a min, that we want to keep separate.

- ▶ The trick: combine `tmax` and `tmin` together into *one* column, keeping track of what kind of temp they are. (This actually same format as `weather2`, which we said was “untidy”.) Are making `weather5` untidy *for purposes of drawing graph* only.
- ▶ Then can do something like

```
ggplot(..., aes(x=date, y=temperature, colour=maxmin))
```

to distinguish max and min on graph.

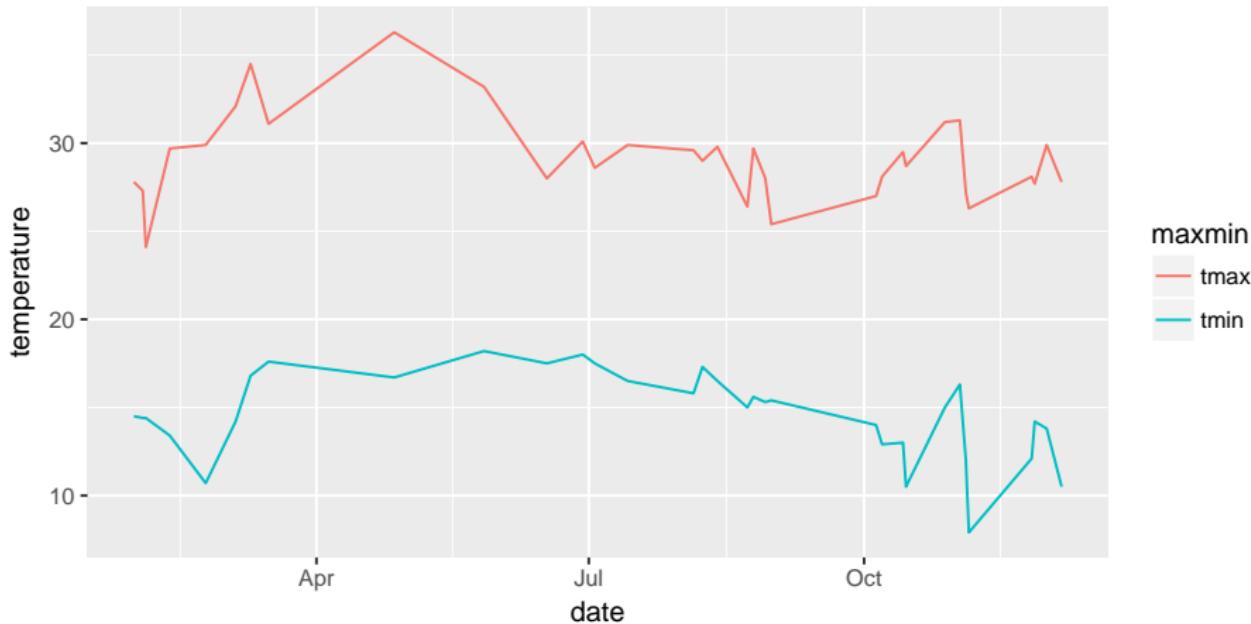
## Setting up plot

- ▶ Since we only need data frame for plot, we can do the column-creation and plot in a chain.
- ▶ The temperature columns are actually text (see printout of `weather5`), but for graph they need to be numbers.
- ▶ For a `ggplot` in a chain, the initial data frame is omitted, because it is *whatever came out of the previous step*.
- ▶ To make those “one column”s: `gather` (result like `weather2`):

```
g=weather5 %>%
 gather(maxmin,temperature,tmax:tmin) %>%
 mutate(temperature=as.numeric(temperature)) %>%
 ggplot(aes(x=date,y=temperature,colour=maxmin))+
 geom_line()
```

# The plot

gg



## Summary of tidying “verbs”

| Verb     | Purpose                                                                                      |
|----------|----------------------------------------------------------------------------------------------|
| gather   | Combine columns that measure same thing into one                                             |
| spread   | Take column that measures one thing under different conditions and put into multiple columns |
| separate | Turn a column that encodes several variables into several columns                            |
| unite    | Combine several (related) variables into one “combination” variable                          |

gather and spread are opposites; separate and unite are opposites.

# Doing things with data frames

Let's go back to our Australian athletes:

```
athletes
```

```
A tibble: 202 x 13
Sex Sport RCC WCC Hc Hg Ferr BMI SSF
<chr> <chr> <dbl> <dbl> <dbl> <dbl> <int> <dbl> <dbl>
1 female Netball 4.56 13.3 42.2 13.6 20 19.16 49.0
2 female Netball 4.15 6.0 38.0 12.7 59 21.15 110.2
3 female Netball 4.16 7.6 37.5 12.3 22 21.40 89.0
4 female Netball 4.32 6.4 37.7 12.3 30 21.03 98.3
5 female Netball 4.06 5.8 38.7 12.8 78 21.77 122.1
6 female Netball 4.12 6.1 36.6 11.8 21 21.38 90.4
7 female Netball 4.17 5.0 37.4 12.7 109 21.47 106.9
8 female Netball 3.80 6.6 36.5 12.4 102 24.45 156.6
9 female Netball 3.96 5.5 36.3 12.4 71 22.63 101.1
10 female Netball 4.44 9.7 41.4 14.1 64 22.80 126.4
... with 192 more rows, and 4 more variables: `%Bfat` <dbl>,
LBM <dbl>, Ht <dbl>, Wt <dbl>
```

Following are some tasks that we might want to perform on this data frame.

## 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 19.16
2 Netball 12.7 21.15
3 Netball 12.3 21.40
4 Netball 12.3 21.03
5 Netball 12.8 21.77
6 Netball 11.8 21.38
7 Netball 12.7 21.47
8 Netball 12.4 24.45
9 Netball 12.4 22.63
10 Netball 14.1 22.80
... 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 6.0
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 5.0
8 female Netball 3.80 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 176.8 59.9
2 female Netball 172.6 63.0
3 female Netball 176.0 66.3
4 female Netball 169.9 60.7
5 female Netball 183.0 72.9
6 female Netball 178.2 67.9
7 female Netball 177.3 67.5
8 female Netball 174.1 74.1
9 female Netball 173.6 68.2
10 female Netball 173.7 68.8
... with 192 more rows
```

## Select-helpers

Other ways to select columns:

- ▶ starts\_with something
- ▶ ends\_with something
- ▶ contains something
- ▶ matches a “regular expression”
- ▶ num\_range like x1 to x3

## Columns beginning with S

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

A tibble: 202 x 3
Sex Sport SSF
<chr> <chr> <dbl>
1 female Netball 49.0
2 female Netball 110.2
3 female Netball 89.0
4 female Netball 98.3
5 female Netball 122.1
6 female Netball 90.4
7 female Netball 106.9
8 female Netball 156.6
9 female Netball 101.1
10 female Netball 126.4
... with 192 more rows
```

## Columns ending 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 6.0 38.0
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 5.0 37.4
8 3.80 6.6 36.5
9 3.96 5.5 36.3
10 4.44 9.7 41.4
... with 192 more rows
```

## Column names containing letter R

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

A tibble: 202 x 3
Sport RCC Ferr
<chr> <dbl> <int>
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 3.80 102
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>
1 176.8 59.9
2 172.6 63.0
3 176.0 66.3
4 169.9 60.7
5 183.0 72.9
6 178.2 67.9
7 177.3 67.5
8 174.1 74.1
```

## Displaying some numbered columns 1/2

Make up a data frame to illustrate. This sample generates random values equally likely to be anything 0–9 (without replacement):

```
d=tibble(y=sample(0:9,5),
 x1=sample(0:9,5),
 x2=sample(0:9,5),
 x3=sample(0:9,5))

d

A tibble: 5 x 4
y x1 x2 x3
<int> <int> <int> <int>
1 6 2 1 9
2 4 1 5 1
3 2 6 4 8
4 5 0 2 6
5 3 5 6 0
```

## Displaying some numbered columns 2/2

Just display x2 and x3:

```
d %>% select(num_range("x",2:3))

A tibble: 5 x 2
x2 x3
<int> <int>
1 1 9
2 5 1
3 4 8
4 2 6
5 6 0
```

## Displaying more than 10 rows

```
athletes %>% print(n=12)

A tibble: 202 x 13
Sex Sport RCC WCC Hc Hg Ferr BMI SSF
<chr> <chr> <dbl> <dbl> <dbl> <dbl> <int> <dbl> <dbl>
1 female Netball 4.56 13.3 42.2 13.6 20 19.16 49.0
2 female Netball 4.15 6.0 38.0 12.7 59 21.15 110.2
3 female Netball 4.16 7.6 37.5 12.3 22 21.40 89.0
4 female Netball 4.32 6.4 37.7 12.3 30 21.03 98.3
5 female Netball 4.06 5.8 38.7 12.8 78 21.77 122.1
6 female Netball 4.12 6.1 36.6 11.8 21 21.38 90.4
7 female Netball 4.17 5.0 37.4 12.7 109 21.47 106.9
8 female Netball 3.80 6.6 36.5 12.4 102 24.45 156.6
9 female Netball 3.96 5.5 36.3 12.4 71 22.63 101.1
10 female Netball 4.44 9.7 41.4 14.1 64 22.80 126.4
11 female Netball 4.27 10.6 37.7 12.5 68 23.58 114.0
12 female Netball 3.90 6.3 35.9 12.1 78 20.06 70.0
... with 190 more rows, and 4 more variables: `%Bfat` <dbl>,
LBM <dbl>, Ht <dbl>, Wt <dbl>
```

# Displaying all the columns

Just for 5 rows here:

```
athletes %>% print(n=5, width=Inf)

A tibble: 202 x 13
Sex Sport RCC WCC Hc Hg Ferr BMI SSF `^%Bfat` LBM
<chr> <chr> <dbl> <dbl> <dbl> <dbl> <int> <dbl> <dbl> <dbl> <dbl>
1 female Netball 4.56 13.3 42.2 13.6 20 19.16 49.0 11.29 53.14
2 female Netball 4.15 6.0 38.0 12.7 59 21.15 110.2 25.26 47.09
3 female Netball 4.16 7.6 37.5 12.3 22 21.40 89.0 19.39 53.44
4 female Netball 4.32 6.4 37.7 12.3 30 21.03 98.3 19.63 48.78
5 female Netball 4.06 5.8 38.7 12.8 78 21.77 122.1 23.11 56.05
... with 197 more rows
```

## Choosing rows by number

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

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

```
Sex Sport RCC WCC Hc Hg Ferr BMI SSE
<chr> <chr> <dbl> <dbl> <dbl> <dbl> <int> <dbl> <dbl>
1 female Netball 4.25 10.7 39.5 13.2 127 24.47 156.6
2 female Netball 4.46 10.9 39.7 13.7 102 23.99 115.9
3 female Netball 4.40 9.3 40.4 13.6 86 26.24 181.7
4 female Netball 4.83 8.4 41.8 13.4 40 20.04 71.6
5 female Netball 4.23 6.9 38.3 12.6 50 25.72 143.5
6 female Netball 4.24 8.4 37.6 12.5 58 25.64 200.8
7 female Netball 3.95 6.6 38.4 12.8 33 19.87 68.9
8 female Netball 4.03 8.5 37.7 13.0 51 23.35 103.6
9 female BBall 3.96 7.5 37.5 12.3 60 20.56 109.1
10 female BBall 4.41 8.3 38.2 12.7 68 20.67 102.8
... with 4 more variables: `%Bfat` <dbl>, LBM <dbl>, Ht <
Wt <dbl>
```

## Non-consecutive rows

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

A tibble: 4 x 13
Sex Sport RCC WCC Hc Hg Ferr BMI SSF
<chr> <chr> <dbl> <dbl> <dbl> <dbl> <int> <dbl> <dbl>
1 female Netball 4.44 9.7 41.4 14.1 64 22.80 126.4
2 female Netball 4.02 9.1 37.7 12.7 107 23.01 77.0
3 female Netball 4.46 10.9 39.7 13.7 102 23.99 115.9
4 female Row 4.37 8.1 41.8 14.3 53 23.47 98.0
... with 4 more variables: `%Bfat` <dbl>, LBM <dbl>, Ht <
Wt <dbl>
```

## A random sample of rows

```
athletes %>% sample_n(8)

A tibble: 8 x 13
Sex Sport RCC WCC Hc Hg Ferr BMI SSF
<chr> <chr> <dbl> <dbl> <dbl> <dbl> <int> <dbl> <dbl>
1 female Netball 4.24 8.4 37.6 12.5 58 25.64 200.8
2 female BBall 4.42 5.7 39.9 13.2 44 20.62 97.9
3 female Tennis 4.40 4.0 40.8 13.9 73 22.12 98.1
4 male T400m 5.21 7.5 47.5 16.5 20 21.89 46.7
5 male Row 5.40 6.8 49.5 17.3 183 26.07 44.7
6 female Swim 4.51 5.1 40.9 14.0 115 19.00 52.5
7 male Swim 5.32 6.0 47.5 16.3 155 23.29 54.4
8 male Swim 5.34 6.6 48.6 16.5 35 22.81 57.0
... with 4 more variables: `~Bfat` <dbl>, LBM <dbl>, Ht <
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 SSF
<chr> <chr> <dbl> <dbl> <dbl> <dbl> <int> <dbl> <dbl>
1 female Tennis 4.00 4.2 36.6 12.0 57 25.36 109.0
2 female Tennis 4.40 4.0 40.8 13.9 73 22.12 98.1
3 female Tennis 4.38 7.9 39.8 13.5 88 21.25 80.6
4 female Tennis 4.08 6.6 37.8 12.1 182 20.53 68.3
5 female Tennis 4.98 6.4 44.8 14.8 80 17.06 47.6
6 female Tennis 5.16 7.2 44.3 14.5 88 18.29 61.9
7 female Tennis 4.66 6.4 40.9 13.9 109 18.37 38.2
8 male Tennis 5.66 8.3 50.2 17.7 38 23.76 56.5
9 male Tennis 5.03 6.4 42.7 14.3 122 22.01 47.6
10 male Tennis 4.97 8.8 43.0 14.9 233 22.34 60.4
11 male Tennis 5.38 6.3 46.0 15.7 32 21.07 34.9
... with 4 more variables: `~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 SSF
<chr> <chr> <dbl> <dbl> <dbl> <dbl> <int> <dbl> <dbl>
1 female Tennis 4.00 4.2 36.6 12.0 57 25.36 109.0
2 female Tennis 4.40 4.0 40.8 13.9 73 22.12 98.1
3 female Tennis 4.38 7.9 39.8 13.5 88 21.25 80.6
4 female Tennis 4.08 6.6 37.8 12.1 182 20.53 68.3
5 female Tennis 4.98 6.4 44.8 14.8 80 17.06 47.6
6 female Tennis 4.66 6.4 40.9 13.9 109 18.37 38.2
7 male Tennis 4.97 8.8 43.0 14.9 233 22.34 60.4
... with 4 more variables: `~Bfat` <dbl>, LBM <dbl>, Ht <
Wt <dbl>
```

## Either/Or

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

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

```
Sex Sport RCC WCC Hc Hg Ferr BMI SSF
<chr> <chr> <dbl> <dbl> <dbl> <dbl> <int> <dbl> <dbl>
1 female Row 5.02 6.4 44.8 15.2 48 19.76 91.0
2 female T400m 5.31 9.5 47.1 15.9 29 21.35 57.9
3 female Field 5.33 9.3 47.0 15.0 62 25.27 102.8
4 female TSprnt 5.16 8.2 45.3 14.7 34 20.30 46.1
5 female Tennis 4.00 4.2 36.6 12.0 57 25.36 109.0
6 female Tennis 4.40 4.0 40.8 13.9 73 22.12 98.1
7 female Tennis 4.38 7.9 39.8 13.5 88 21.25 80.6
8 female Tennis 4.08 6.6 37.8 12.1 182 20.53 68.3
9 female Tennis 4.98 6.4 44.8 14.8 80 17.06 47.6
10 female Tennis 5.16 7.2 44.3 14.5 88 18.29 61.9
... with 56 more rows, and 4 more variables: `%Bfat` <dbl>
LBM <dbl>, Ht <dbl>, Wt <dbl>
```

## Sorting into order

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

```
A tibble: 202 x 13
Sex Sport RCC WCC Hc Hg Ferr BMI SSE
<chr> <chr> <dbl> <dbl> <dbl> <dbl> <int> <dbl> <dbl>
1 female Netball 3.80 6.6 36.5 12.4 102 24.45 156.6
2 female Netball 3.90 6.3 35.9 12.1 78 20.06 70.0
3 female T400m 3.90 6.0 38.9 13.5 16 19.37 48.4
4 female Row 3.91 7.3 37.6 12.9 43 22.27 125.9
5 female Netball 3.95 6.6 38.4 12.8 33 19.87 68.9
6 female Row 3.95 3.3 36.9 12.5 40 24.54 74.9
7 female Netball 3.96 5.5 36.3 12.4 71 22.63 101.1
8 female BBall 3.96 7.5 37.5 12.3 60 20.56 109.1
9 female Tennis 4.00 4.2 36.6 12.0 57 25.36 109.0
10 female Netball 4.02 9.1 37.7 12.7 107 23.01 77.0
... with 192 more rows, and 4 more variables: `~Bfat` <dbl>
LBM <dbl>, Ht <dbl>, Wt <dbl>
```

## Breaking ties by another variable

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

```
A tibble: 202 x 13
Sex Sport RCC WCC Hc Hg Ferr BMI SSE
<chr> <chr> <dbl> <dbl> <dbl> <dbl> <int> <dbl> <dbl>
1 female Netball 3.80 6.6 36.5 12.4 102 24.45 156.6
2 female T400m 3.90 6.0 38.9 13.5 16 19.37 48.4
3 female Netball 3.90 6.3 35.9 12.1 78 20.06 70.0
4 female Row 3.91 7.3 37.6 12.9 43 22.27 125.9
5 female Netball 3.95 6.6 38.4 12.8 33 19.87 68.9
6 female Row 3.95 3.3 36.9 12.5 40 24.54 74.9
7 female BBall 3.96 7.5 37.5 12.3 60 20.56 109.1
8 female Netball 3.96 5.5 36.3 12.4 71 22.63 101.1
9 female Tennis 4.00 4.2 36.6 12.0 57 25.36 109.0
10 female Netball 4.02 9.1 37.7 12.7 107 23.01 77.0
... with 192 more rows, and 4 more variables: `~Bfat` <dbl>
LBM <dbl>, Ht <dbl>, Wt <dbl>
```

## Descending order

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

```
A tibble: 202 x 13
Sex Sport RCC WCC Hc Hg Ferr BMI SSF
<chr> <chr> <dbl> <dbl> <dbl> <dbl> <int> <dbl> <dbl>
1 male Field 5.48 6.2 48.2 16.3 94 34.42 82.7
2 male Field 4.96 8.3 45.3 15.7 141 33.73 113.5
3 male Field 5.48 4.6 49.4 18.0 132 32.52 55.7
4 female Field 4.75 7.5 43.8 15.2 90 31.93 131.9
5 male Field 5.01 8.9 46.0 15.9 212 30.18 112.5
6 male Field 5.01 8.9 46.0 15.9 212 30.18 96.9
7 male Field 5.09 8.9 46.3 15.4 44 29.97 71.1
8 female Field 4.58 5.8 42.1 14.7 164 28.57 109.6
9 female Field 4.51 9.0 39.7 14.3 36 28.13 136.3
10 male WPolo 5.34 6.2 49.8 17.2 143 27.79 75.7
... with 192 more rows, and 4 more variables: `~Bfat` <dbl>
LBM <dbl>, Ht <dbl>, Wt <dbl>
```

## “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
2 BBall 113.7
3 Field 111.3
4 Field 108.2
5 Field 102.7
6 WPolo 101.0
7 BBall 100.2
```

## Create new variables from old ones

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

A tibble: 202 x 3
Sport Wt wt_lb
<chr> <dbl> <dbl>
1 Netball 59.9 131.78
2 Netball 63.0 138.60
3 Netball 66.3 145.86
4 Netball 60.7 133.54
5 Netball 72.9 160.38
6 Netball 67.9 149.38
7 Netball 67.5 148.50
8 Netball 74.1 163.02
9 Netball 68.2 150.04
10 Netball 68.8 151.36
... with 192 more rows
```

## 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)) %>% slice(1) %>%
 pull(Wt)
[1] 123.2
```

All of these tools can be combined, as with this example.

## 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 174.5940
2 male 185.5059
```

or

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

A tibble: 1 x 1
m
<dbl>
1 174.594
```

## 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>as.numeric</code> | Turn result from data frame into number             |
| <code>group_by</code>   | Create groups to summarize by                       |
| <code>summarize</code>  | Calculate summary statistics (by groups if defined) |

## Looking things up in another data frame

Recall the tuberculosis data set, tidied:

```
tb3
```

```
A tibble: 35,750 x 5
iso2 year gender age freq
* <chr> <int> <chr> <chr> <int>
1 AD 2005 m 04 0
2 AD 2006 m 04 0
3 AD 2008 m 04 0
4 AE 2006 m 04 0
5 AE 2007 m 04 0
6 AE 2008 m 04 0
7 AG 2007 m 04 0
8 AL 2005 m 04 0
9 AL 2006 m 04 1
10 AL 2007 m 04 0
... with 35,740 more rows
```

What are actual names of those countries in iso2?

## Actual country names

Found actual country names to go with those abbreviations, in spreadsheet:

```
library(readxl)
country_names=read_excel("ISOCountryCodes081507.xlsx")
```

## The country names

```
country_names

A tibble: 252 x 3
Code Code_UC Country
<chr> <chr> <chr>
1 ad AD Andorra
2 ae AE United Arab Emirates
3 af AF Afghanistan
4 ag AG Antigua and Barbuda
5 ai AI Anguilla
6 al AL Albania
7 am AM Armenia
8 an AN Netherlands Antilles
9 ao AO Angola
10 aq AQ Antarctica
... with 242 more rows
```

## Looking up country codes

Matching a variable in one data frame to one in another is called a **join** (database terminology):

```
tb3 %>% left_join(country_names, by=c("iso2"="Code_UC"))

A tibble: 35,750 x 7
iso2 year gender age freq Code Country
<chr> <int> <chr> <chr> <int> <chr> <chr>
1 AD 2005 m 04 0 ad Andorra
2 AD 2006 m 04 0 ad Andorra
3 AD 2008 m 04 0 ad Andorra
4 AE 2006 m 04 0 ae United Arab Emirates
5 AE 2007 m 04 0 ae United Arab Emirates
6 AE 2008 m 04 0 ae United Arab Emirates
7 AG 2007 m 04 0 ag Antigua and Barbuda
8 AL 2005 m 04 0 al Albania
9 AL 2006 m 04 1 al Albania
10 AL 2007 m 04 0 al Albania
... with 35,740 more rows
```

## Total cases by country

```
tb3 %>% group_by(iso2) %>% summarize(cases=sum(freq)) %>%
 left_join(country_names, by=c("iso2"="Code_UC")) %>%
 select(Country, cases)

A tibble: 213 x 2
Country cases
<chr> <int>
1 Andorra 64
2 United Arab Emirates 487
3 Afghanistan 80005
4 Antigua and Barbuda 21
5 Anguilla 1
6 Albania 2467
7 Armenia 6757
8 Netherlands Antilles 81
9 Angola 195512
10 Argentina 64894
... with 203 more rows
```

## Some of the foregoing in SAS

- ▶ SAS is less flexible than R's tidyverse tools, but some of the previous can be done (with effort).
- ▶ Basic idea: create a new dataset using `data` and `set`, and then provide additional code to say what to do to the previous dataset.
- ▶ Read Australian athletes data again:

```
proc import
 datafile='/home/ken/ais.txt'
 dbms=dlm
 out=sports
 replace;
 delimiter='09'x;
 getnames=yes;
```

## Check the data

```
proc print data=sports(obs=10);
```

| Obs | Sex    | Sport   | RCC   | WCC   | Hc    |
|-----|--------|---------|-------|-------|-------|
| 1   | female | Netball | 4.56  | 13.3  | 42.2  |
| 2   | female | Netball | 4.15  | 6     | 38    |
| 3   | female | Netball | 4.16  | 7.6   | 37.5  |
| 4   | female | Netball | 4.32  | 6.4   | 37.7  |
| 5   | female | Netball | 4.06  | 5.8   | 38.7  |
| 6   | female | Netball | 4.12  | 6.1   | 36.6  |
| 7   | female | Netball | 4.17  | 5     | 37.4  |
| 8   | female | Netball | 3.8   | 6.6   | 36.5  |
| 9   | female | Netball | 3.96  | 5.5   | 36.3  |
| 10  | female | Netball | 4.44  | 9.7   | 41.4  |
| Obs |        | Hg      | Ferr  | BMI   | SSF   |
| 1   |        | 13.6    | 20    | 19.16 | 49    |
| 2   |        | 12.7    | 59    | 21.15 | 110.2 |
| 3   |        | 12.3    | 22    | 21.4  | 89    |
| 4   |        | 12.3    | 30    | 21.03 | 98.3  |
| 5   |        | 12.8    | 78    | 21.77 | 122.1 |
| 6   |        | 11.8    | 21    | 21.38 | 90.4  |
| 7   |        | 12.7    | 109   | 21.47 | 106.9 |
| 8   |        | 12.4    | 102   | 24.45 | 156.6 |
| 9   |        | 12.4    | 71    | 22.63 | 101.1 |
| 10  |        | 14.1    | 64    | 22.8  | 126.4 |
| Obs |        | _Bfat   | LBM   | Ht    | Wt    |
| 1   |        | 11.29   | 53.14 | 176.8 | 59.9  |
| 2   |        | 25.26   | 47.09 | 172.6 | 63    |
| 3   |        | 19.39   | 53.44 | 176   | 66.3  |
| 4   |        | 19.63   | 48.78 | 169.9 | 60.7  |
| 5   |        | 22.11   | 56.05 | 169   | 70.2  |

## Choosing variables

keep to say which ones you want:

```
data sports2;
 set sports;
 keep Sport Sex Ht Wt;
```

```
proc print data=sports2(obs=8);
```

| Obs | Sex    | Sport   | Ht    | Wt   |
|-----|--------|---------|-------|------|
| 1   | female | Netball | 176.8 | 59.9 |
| 2   | female | Netball | 172.6 | 63   |
| 3   | female | Netball | 176   | 66.3 |
| 4   | female | Netball | 169.9 | 60.7 |
| 5   | female | Netball | 183   | 72.9 |
| 6   | female | Netball | 178.2 | 67.9 |
| 7   | female | Netball | 177.3 | 67.5 |
| 8   | female | Netball | 174.1 | 74.1 |

## Un-choosing variables

drop to say which ones you don't want. Note the double-dash to denote "this through that":

```
data sports3;
 set sports;
 drop RCC--LBM;
```

```
proc print data=sports3(obs=8);
```

| Obs | Sex    | Sport   | Ht    | Wt   |
|-----|--------|---------|-------|------|
| 1   | female | Netball | 176.8 | 59.9 |
| 2   | female | Netball | 172.6 | 63   |
| 3   | female | Netball | 176   | 66.3 |
| 4   | female | Netball | 169.9 | 60.7 |
| 5   | female | Netball | 183   | 72.9 |
| 6   | female | Netball | 178.2 | 67.9 |
| 7   | female | Netball | 177.3 | 67.5 |
| 8   | female | Netball | 174.1 | 74.1 |

## Comments

- ▶ Normally don't worry about explicitly dropping variables you don't need; you just ignore them in your analysis.
- ▶ keep and drop mostly for final "tidy" version of datasets that you create.
- ▶ Can also feed proc print the columns to display, with var.
- ▶ For example, might want to discard intermediate steps of a calculation.

## Calculating a new variable

Put the calculation in the data step, as we have seen before:

```
data sports4;
 set sports;
 Wt_lb=Wt*2.2;
 keep Sport Wt Wt_lb;
```

```
proc print data=sports4(obs=8);
```

| Obs | Sport   | Wt   | Wt_lb  |
|-----|---------|------|--------|
| 1   | Netball | 59.9 | 131.78 |
| 2   | Netball | 63   | 138.60 |
| 3   | Netball | 66.3 | 145.86 |
| 4   | Netball | 60.7 | 133.54 |
| 5   | Netball | 72.9 | 160.38 |
| 6   | Netball | 67.9 | 149.38 |
| 7   | Netball | 67.5 | 148.50 |
| 8   | Netball | 74.1 | 163.02 |

## Choosing rows by row number

SAS has a special variable `_N_` that holds the row number:

```
data sports5;
 set sports;
 if _N_>=16 and _N_<=25;

proc print;
```

| Obs | Sex    | Sport   | RCC  | WCC  | Hc   |
|-----|--------|---------|------|------|------|
| 1   | female | Netball | 4.25 | 10.7 | 39.5 |
| 2   | female | Netball | 4.46 | 10.9 | 39.7 |
| 3   | female | Netball | 4.4  | 9.3  | 40.4 |
| 4   | female | Netball | 4.83 | 8.4  | 41.8 |
| 5   | female | Netball | 4.23 | 6.9  | 38.3 |
| 6   | female | Netball | 4.24 | 8.4  | 37.6 |
| 7   | female | Netball | 3.95 | 6.6  | 38.4 |
| 8   | female | Netball | 4.03 | 8.5  | 37.7 |
| 9   | female | BBall   | 3.96 | 7.5  | 37.5 |
| 10  | female | BBall   | 4.41 | 8.3  | 38.2 |

| Obs | Hg | Ferr | BMI | SSF       |
|-----|----|------|-----|-----------|
|     |    |      |     | 422 / 735 |

## Choosing each of a number of rows

```
data sports6;
 set sports;
 if _N_ in (10, 13, 17, 42);
```

```
proc print;
```

| Obs | Sex    | Sport   | RCC  | WCC  | Hc   |
|-----|--------|---------|------|------|------|
| 1   | female | Netball | 4.44 | 9.7  | 41.4 |
| 2   | female | Netball | 4.02 | 9.1  | 37.7 |
| 3   | female | Netball | 4.46 | 10.9 | 39.7 |
| 4   | female | Row     | 4.37 | 8.1  | 41.8 |

| Obs | Hg   | Ferr | BMI   | SSF   |
|-----|------|------|-------|-------|
| 1   | 14.1 | 64   | 22.8  | 126.4 |
| 2   | 12.7 | 107  | 23.01 | 77    |
| 3   | 13.7 | 102  | 23.99 | 115.9 |
| 4   | 14.3 | 53   | 23.47 | 98    |

| Obs | _Bfat | LBM | Ht | Wt | 423 / 735 |
|-----|-------|-----|----|----|-----------|
|     |       |     |    |    |           |

## Choosing rows where a condition is true

if like R's filter, but note that SAS uses *one* equals sign in testing for equality:

```
data sports7;
 set sports;
 if Sport="Tennis";
```

```
proc print;
```

| Obs | Sex    | Sport  | RCC  | WCC | Hc   |
|-----|--------|--------|------|-----|------|
| 1   | female | Tennis | 4    | 4.2 | 36.6 |
| 2   | female | Tennis | 4.4  | 4   | 40.8 |
| 3   | female | Tennis | 4.38 | 7.9 | 39.8 |
| 4   | female | Tennis | 4.08 | 6.6 | 37.8 |
| 5   | female | Tennis | 4.98 | 6.4 | 44.8 |
| 6   | female | Tennis | 5.16 | 7.2 | 44.3 |
| 7   | female | Tennis | 4.66 | 6.4 | 40.9 |
| 8   | male   | Tennis | 5.66 | 8.3 | 50.2 |
| 9   | male   | Tennis | 5.03 | 6.4 | 42.7 |
| 10  | male   | Tennis | 4.97 | 8.8 | 43   |
| 11  | male   | Tennis | 5.38 | 6.3 | 46   |

## Multiple conditions 1/2

Join them with actual words and, or:

```
data sports8;
 set sports;
 if Sport="Tennis" and RCC<5;
```

```
proc print;
 var Sex--RCC;
```

| Obs | Sex    | Sport  | RCC  |
|-----|--------|--------|------|
| 1   | female | Tennis | 4    |
| 2   | female | Tennis | 4.4  |
| 3   | female | Tennis | 4.38 |
| 4   | female | Tennis | 4.08 |
| 5   | female | Tennis | 4.98 |
| 6   | female | Tennis | 4.66 |
| 7   | male   | Tennis | 4.97 |

## Multiple conditions 2/2

```
data sports9;
 set sports;
 if Sport="Tennis" or RCC>5;
```

```
proc print;
 var Sex--RCC BMI;
```

| Obs | Sex    | Sport  | RCC  | BMI   |
|-----|--------|--------|------|-------|
| 1   | female | Row    | 5.02 | 19.76 |
| 2   | female | T400m  | 5.31 | 21.35 |
| 3   | female | Field  | 5.33 | 25.27 |
| 4   | female | TSprnt | 5.16 | 20.3  |
| 5   | female | Tennis | 4    | 25.36 |
| 6   | female | Tennis | 4.4  | 22.12 |
| 7   | female | Tennis | 4.38 | 21.25 |
| 8   | female | Tennis | 4.08 | 20.53 |
| 9   | female | Tennis | 4.98 | 17.06 |
| 10  | female | Tennis | 5.16 | 18.29 |
| 11  | female | Tennis | 4.66 | 18.37 |
| 12  | male   | Swim   | 5.13 | 22.46 |
| 13  | male   | Swim   | 5.09 | 23.68 |
| 14  | male   | Swim   | 5.17 | 23.15 |
| 15  | male   | Swim   | 5.11 | 22.32 |
| 16  | male   | Swim   | 5.03 | 24.02 |
| 17  | male   | Swim   | 5.32 | 23.29 |
| 18  | male   | Swim   | 5.34 | 22.81 |
| 19  | male   | Swim   | 5.33 | 21.38 |
| 20  | male   | Row    | 5.04 | 25.84 |

## Using data where a condition is true

- ▶ Rather than creating a new data set containing the values that satisfy a condition, we can tell SAS which data to use right in a proc.
- ▶ Key idea: put `where` and a logical condition as the *first* line of the proc.
- ▶ For example, mean BMI of tennis players:

```
proc means;
 where sport="Tennis";
 var BMI;
```

# Mean and SD of BMI for tennis players

The MEANS Procedure

Analysis Variable : BMI

| N  | Mean       | Std Dev   | Minimum    | Maximum    |
|----|------------|-----------|------------|------------|
| 11 | 21.1054545 | 2.4626789 | 17.0600000 | 25.3600000 |

## Arranging values in order

This is proc sort, which produces an output data set that is the “most recent” one:

```
proc sort data=sports;
 by RCC;
```

```
proc print;
 var Sex--RCC;
```

| Obs | Sex    | Sport   | RCC  |
|-----|--------|---------|------|
| 1   | female | Netball | 3.8  |
| 2   | female | Netball | 3.9  |
| 3   | female | T400m   | 3.9  |
| 4   | female | Row     | 3.91 |
| 5   | female | Netball | 3.95 |
| 6   | female | Row     | 3.95 |
| 7   | female | Netball | 3.96 |
| 8   | female | BBall   | 3.96 |
| 9   | female | Tennis  | 4    |
| 10  | female | Netball | 4.02 |
| 11  | female | Netball | 4.03 |

## Using a second variable as tiebreaker

```
proc sort data=sports;
 by RCC BMI;
```

```
proc print;
 var Sex--RCC BMI;
```

| Obs | Sex    | Sport   | RCC  | BMI   |
|-----|--------|---------|------|-------|
| 1   | female | Netball | 3.8  | 24.45 |
| 2   | female | T400m   | 3.9  | 19.37 |
| 3   | female | Netball | 3.9  | 20.06 |
| 4   | female | Row     | 3.91 | 22.27 |
| 5   | female | Netball | 3.95 | 19.87 |
| 6   | female | Row     | 3.95 | 24.54 |
| 7   | female | BBall   | 3.96 | 20.56 |
| 8   | female | Netball | 3.96 | 22.63 |
| 9   | female | Tennis  | 4    | 25.36 |
| 10  | female | Netball | 4.02 | 23.01 |
| 11  | female | Netball | 4.03 | 23.35 |
| 12  | female | Netball | 4.06 | 21.77 |
| 13  | female | Swim    | 4.07 | 20.42 |
| 14  | female | Tennis  | 4.08 | 20.53 |

## Descending order

```
proc sort data=sports;
 by descending BMI;
```

```
proc print;
 var Sex--RCC BMI;
```

| Obs | Sex    | Sport | RCC  | BMI   |
|-----|--------|-------|------|-------|
| 1   | male   | Field | 5.48 | 34.42 |
| 2   | male   | Field | 4.96 | 33.73 |
| 3   | male   | Field | 5.48 | 32.52 |
| 4   | female | Field | 4.75 | 31.93 |
| 5   | male   | Field | 5.01 | 30.18 |
| 6   | male   | Field | 5.01 | 30.18 |
| 7   | male   | Field | 5.09 | 29.97 |
| 8   | female | Field | 4.58 | 28.57 |
| 9   | female | Field | 4.51 | 28.13 |
| 10  | male   | WPolo | 5.34 | 27.79 |
| 11  | male   | WPolo | 4.9  | 27.56 |
| 12  | male   | Field | 5.11 | 27.39 |
| 13  | female | Field | 4.81 | 26.95 |
| 14  | male   | WPolo | 5.08 | 26.86 |

## Displaying the seven heaviest athletes

```
proc sort data=sports;
 by descending Wt;

data sports10;
 set sports;
 if _N_<=7;
 keep Sport Wt;

proc print;
```

| Obs | Sport | Wt    |
|-----|-------|-------|
| 1   | Field | 123.2 |
| 2   | BBall | 113.7 |
| 3   | Field | 111.3 |
| 4   | Field | 108.2 |
| 5   | Field | 102.7 |
| 6   | WPolo | 101   |
| 7   | BBall | 100.2 |

## Tidying data

- ▶ Tidying data in SAS is hard. Let's illustrate the SAS version of `gather` on the pigs data, that we have to read in first.
- ▶ Each line of this dataset has to produce *four* lines of the long data set.

```
proc import
 datafile='/home/ken/pigs1.txt'
 dbms=dlm out=pigs replace;
 delimiter=' '|;
 getnames=yes;
```

```
proc print;
```

| Obs | pig | feed1 | feed2 | feed3 | feed4 |
|-----|-----|-------|-------|-------|-------|
| 1   | 1   | 60.8  | 68.7  | 92.6  | 87.9  |
| 2   | 2   | 57    | 67.7  | 92.1  | 84.2  |
| 3   | 3   | 65    | 74    | 90.2  | 83.1  |
| 4   | 4   | 58.6  | 66.3  | 96.5  | 85.7  |
| 5   | 5   | 61.7  | 69.8  | 99.1  | 90.3  |

## Making the long data set, the tedious way

```
data pigs2;
 set pigs;
 feed='feed1';
 weight=feed1;
 output;
 feed='feed2';
 weight=feed2;
 output;
 feed='feed3';
 weight=feed3;
 output;
 feed='feed4';
 weight=feed4;
 output;
 keep pig feed weight;
```

## The long data set

```
proc print;
```

| Obs | pig | feed  | weight |
|-----|-----|-------|--------|
| 1   | 1   | feed1 | 60.8   |
| 2   | 1   | feed2 | 68.7   |
| 3   | 1   | feed3 | 92.6   |
| 4   | 1   | feed4 | 87.9   |
| 5   | 2   | feed1 | 57.0   |
| 6   | 2   | feed2 | 67.7   |
| 7   | 2   | feed3 | 92.1   |
| 8   | 2   | feed4 | 84.2   |
| 9   | 3   | feed1 | 65.0   |
| 10  | 3   | feed2 | 74.0   |
| 11  | 3   | feed3 | 90.2   |
| 12  | 3   | feed4 | 83.1   |
| 13  | 4   | feed1 | 58.6   |
| 14  | 4   | feed2 | 66.3   |
| 15  | 4   | feed3 | 96.5   |
| 16  | 4   | feed4 | 85.7   |
| 17  | 5   | feed1 | 61.7   |
| 18  | 5   | feed2 | 69.8   |
| 19  | 5   | feed3 | 99.1   |
| 20  | 5   | feed4 | 90.3   |

## Using a SAS array to reduce repetition

```
data pigs3;
 set pigs;
 array feed_array [4] feed1-feed4;
 do i=1 to 4;
 weight=feed_array[i];
 feed=vname(feed_array[i]);
 output;
 end;
 keep pig feed weight;
```

- ▶ In SAS, an array is a mechanism for referring to a group of variables together, here the four feed variables. The  $i$ -th element of the array refers to the  $i$ -th feed variable.
- ▶ In the loop (indented), weight is set to the *value* of the appropriate one of the feed variables, while feed is set to the *name* of that feed variable. Compare the coding without the loop.

## The long data set, again

```
proc print;
```

| Obs | pig | weight | feed  |
|-----|-----|--------|-------|
| 1   | 1   | 60.8   | feed1 |
| 2   | 1   | 68.7   | feed2 |
| 3   | 1   | 92.6   | feed3 |
| 4   | 1   | 87.9   | feed4 |
| 5   | 2   | 57.0   | feed1 |
| 6   | 2   | 67.7   | feed2 |
| 7   | 2   | 92.1   | feed3 |
| 8   | 2   | 84.2   | feed4 |
| 9   | 3   | 65.0   | feed1 |
| 10  | 3   | 74.0   | feed2 |
| 11  | 3   | 90.2   | feed3 |
| 12  | 3   | 83.1   | feed4 |
| 13  | 4   | 58.6   | feed1 |
| 14  | 4   | 66.3   | feed2 |
| 15  | 4   | 96.5   | feed3 |
| 16  | 4   | 85.7   | feed4 |
| 17  | 5   | 61.7   | feed1 |
| 18  | 5   | 69.8   | feed2 |
| 19  | 5   | 99.1   | feed3 |
| 20  | 5   | 90.3   | feed4 |

## The ANOVA, again, with output part 1

```
proc anova;
 class feed;
 model weight=feed;
 means feed / tukey;
```

The ANOVA Procedure  
Dependent Variable: weight

| Source          | DF | Sum of Squares | Mean Square | F Value | Pr > F |
|-----------------|----|----------------|-------------|---------|--------|
| Model           | 3  | 3520.525500    | 1173.508500 | 119.14  | <.0001 |
| Error           | 16 | 157.600000     | 9.850000    |         |        |
| Corrected Total | 19 | 3678.125500    |             |         |        |
| Source          | DF | Anova SS       | Mean Square | F Value | Pr > F |
| feed            | 3  | 3520.525500    | 1173.508500 | 119.14  | <.0001 |

The mean weights are not all the same for each feed.

## Tukey output

Means with the same letter are not significantly different.

| Tukey Grouping | Mean   | N | feed  |
|----------------|--------|---|-------|
| A              | 94.100 | 5 | feed3 |
| B              | 86.240 | 5 | feed4 |
| C              | 69.300 | 5 | feed2 |
| D              | 60.620 | 5 | feed1 |

All of the feeds have significantly different mean weight, with feed 3 being the best and feed 1 the worst.

## Section 9

Case study 1: the windmill data

## R packages used

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

##
Attaching package: 'MASS'

The following object is masked from 'package:dplyr':
select

library(leaps)
```

## 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?
- ▶ Do analysis in R.

## Reading in the data

```
windmill=read_csv("windmill.csv")

Parsed with 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 5.00 1.582
2 6.00 1.822
3 3.40 1.057
4 2.70 0.500
5 10.00 2.236
6 9.70 2.386
7 9.55 2.294
8 3.05 0.558
9 8.15 2.166
10 6.20 1.866
... with 15 more rows
```

## Strategy

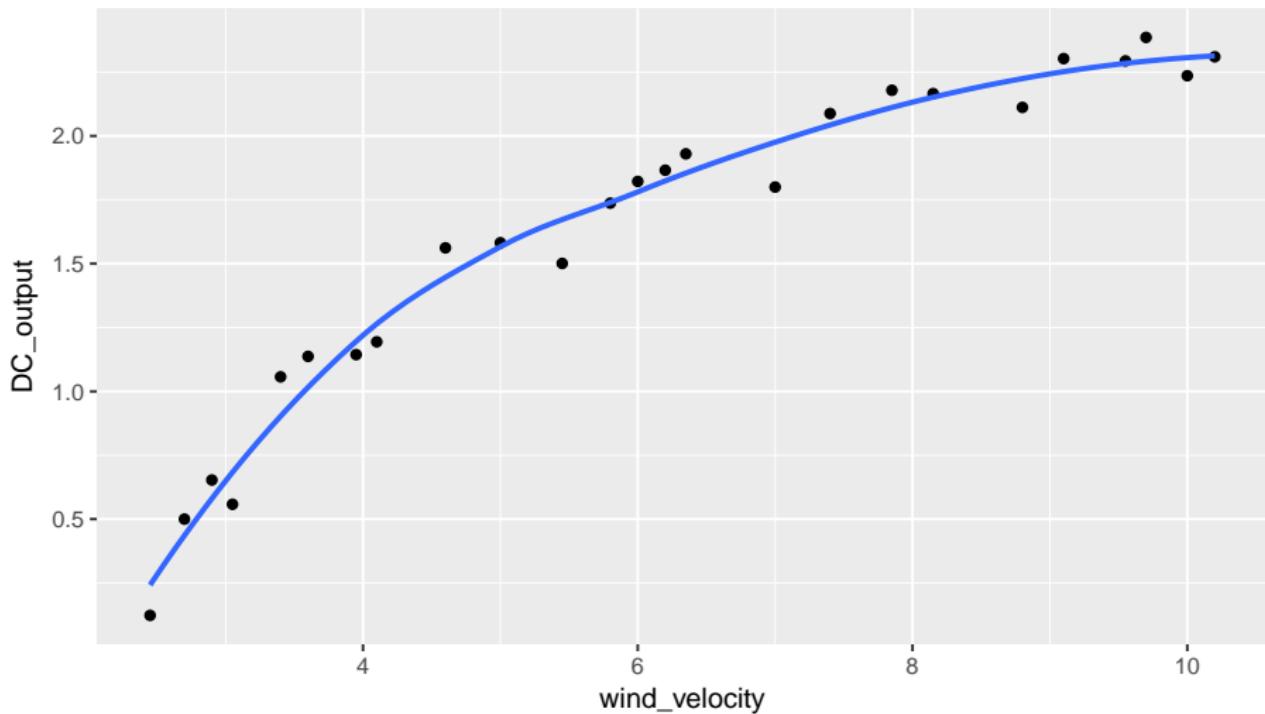
- ▶ Two quantitative variables, looking for relationship: regression methods.
- ▶ Start with picture (scatterplot).
- ▶ Fit models and do model checking, fixing up things as necessary.
- ▶ Scatterplot:
  - ▶ 2 variables, DC\_output and wind\_velocity.
  - ▶ First is output/response, other is input/explanatory.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

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

# Scatterplot

gg

```
`geom_smooth()` using method = 'loess'
```



## 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. Straight line not so good here.

## Fitting a straight line

- Let's try fitting a straight line anyway, 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
```

## Comments

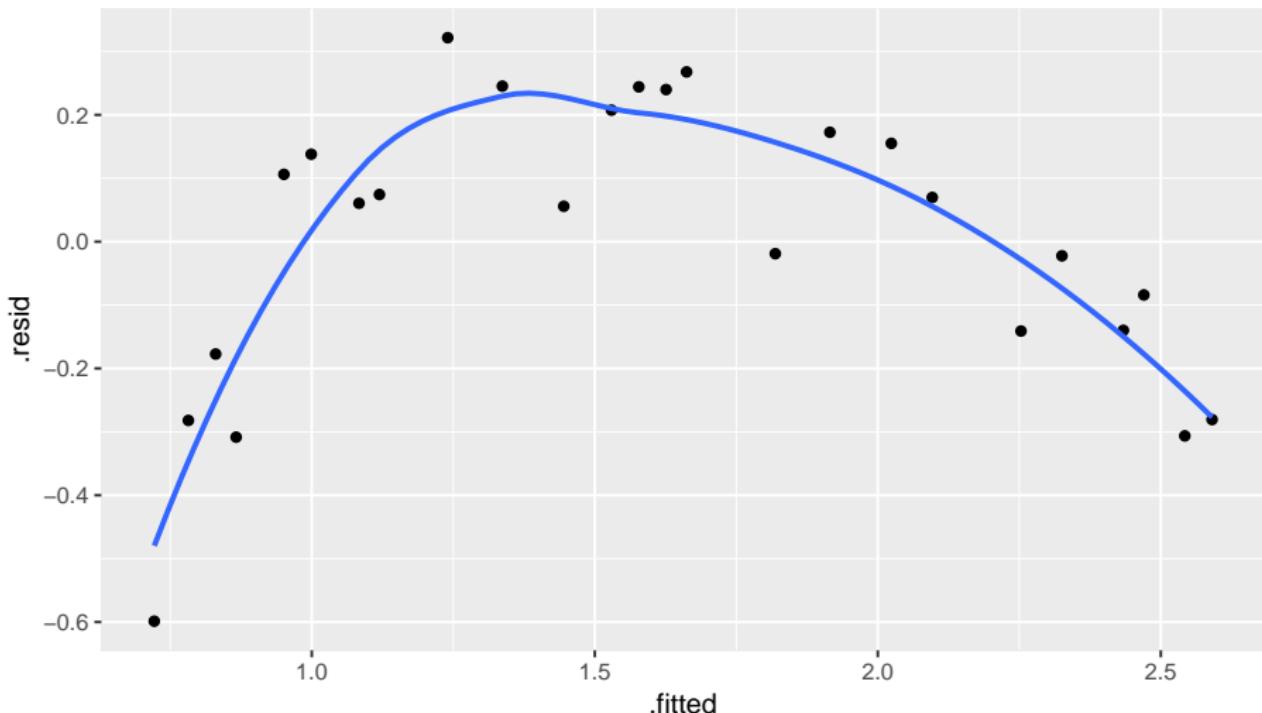
- ▶ Strategy: `lm` actually fits the regression. Store results in a variable.  
*Then* look at the results, eg. via `summary`.
- ▶ My strategy for model names: use response variable and a number.  
Allows me to fit several models to same data and keep track of which is which.
- ▶ Results actually pretty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

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

# Plot of residuals against fitted values

g

```
'geom_smooth()' using method = 'loess'
```



## 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

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

for suitable  $a, b, c$ . 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)
```
- ▶ The `I()` necessary because `^` in model formula otherwise means something different (to do with interactions in ANOVA).
- ▶ This actually *multiple regression*.
- ▶ Call it *parabola model*.

## Parabola model output

summary(DC.2)

```

Call:
lm(formula = DC_output ~ wind_velocity + I(wind_velocity^2),
data = windmill)

Residuals:
Min 1Q Median 3Q Max
-0.26347 -0.02537 0.01264 0.03908 0.19903

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.155898 0.174650 -6.618 1.18e-06 ***
wind_velocity 0.722936 0.061425 11.769 5.77e-11 ***
I(wind_velocity^2) -0.038121 0.004797 -7.947 6.59e-08 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1227 on 22 degrees of freedom
Multiple R-squared: 0.9676, Adjusted R-squared: 0.9646
F-statistic: 328.3 on 2 and 22 DF, p-value: < 2.2e-16
```

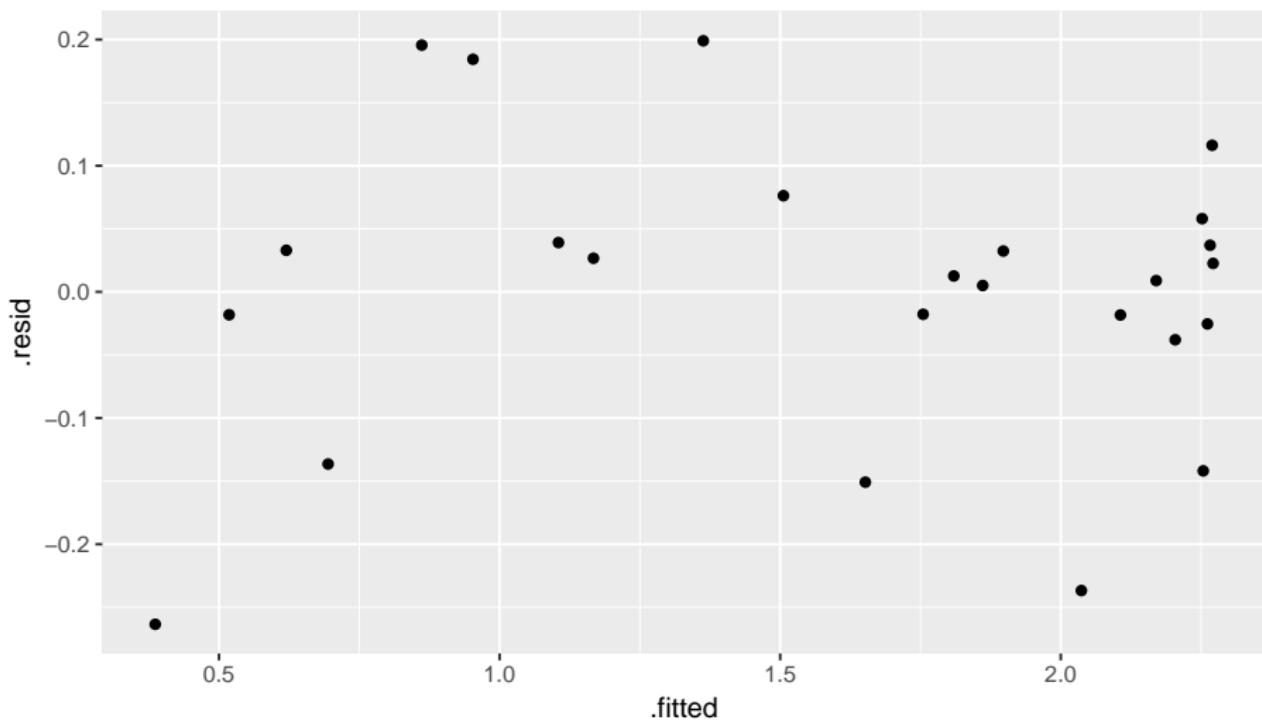
## Comments on output

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

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

## Residual plot from parabola model

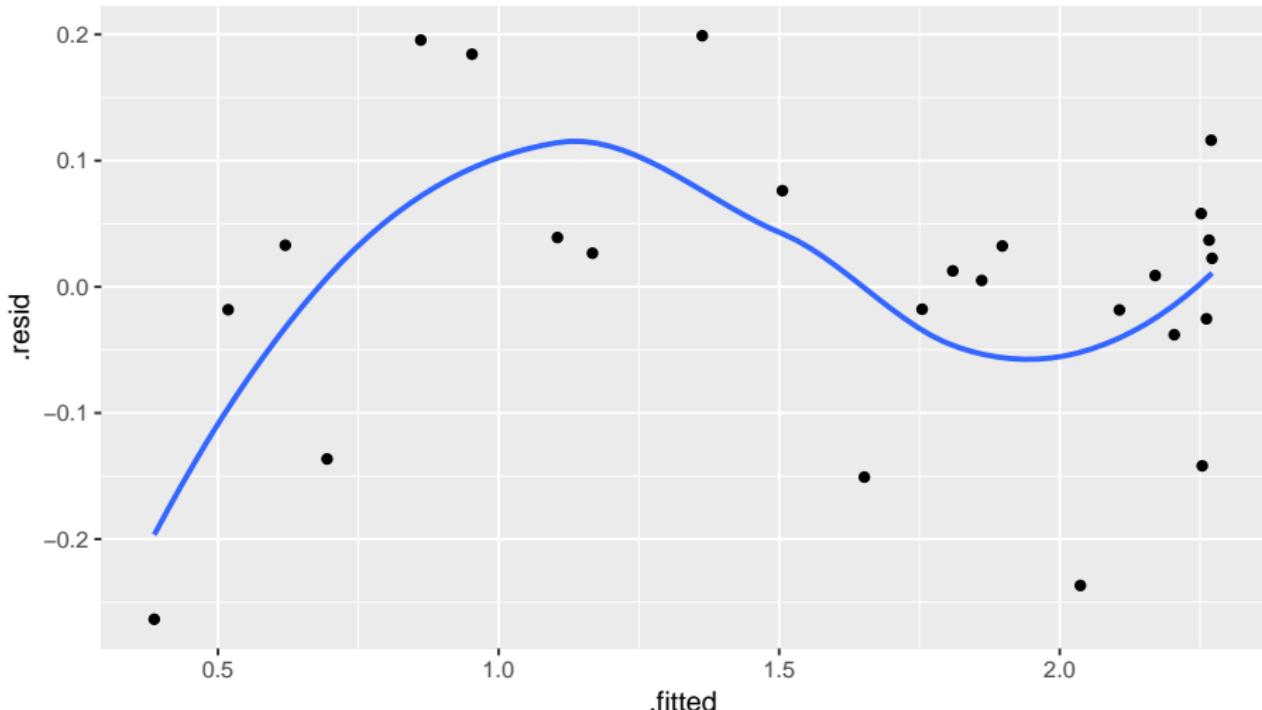
g



Or, with smooth trend

```
g+geom_smooth(se=F)
```

```
'geom_smooth()' using method = 'loess'
```



## Scatterplot with fitted line and curve

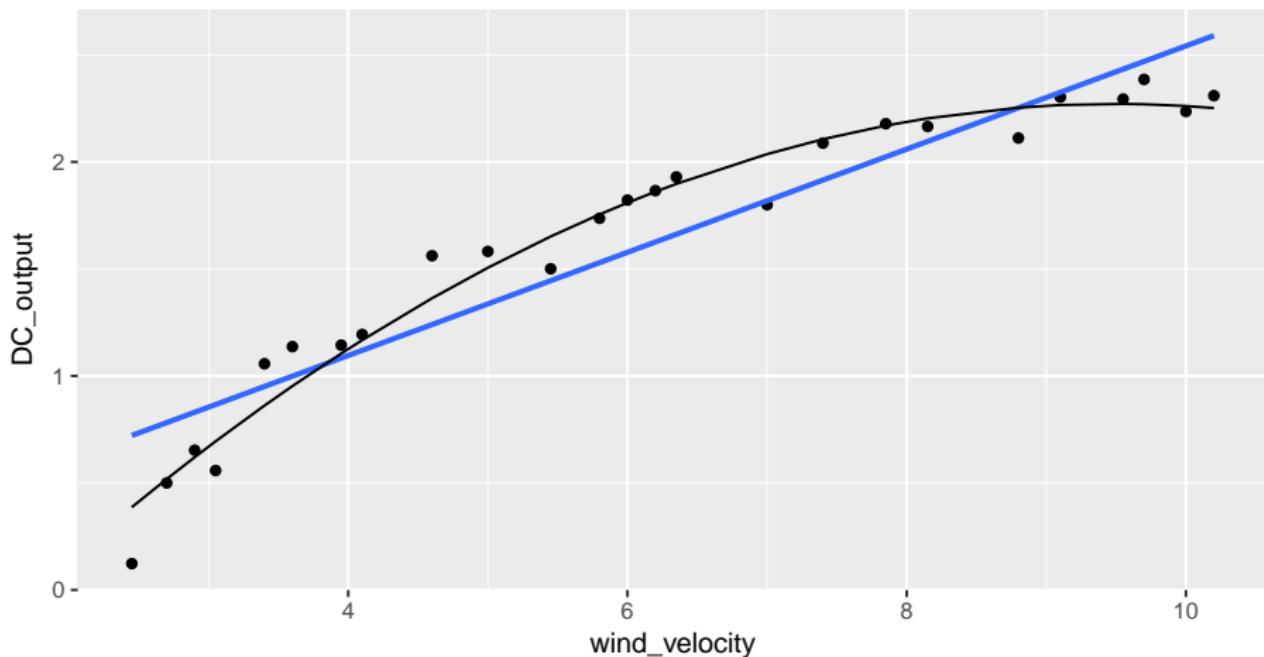
- ▶ Residual plot basically random. Good.
- ▶ Smooth trend *not* flat, but might be deceived by few residuals above 0 in middle.
- ▶ Scatterplot with fitted line and curve like this:

```
g=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))
```

- ▶ 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

g



Curve clearly fits better than line.

## Another approach to a curve

- ▶ There is a problem with parabolas, which we'll see later.
- ▶ Go back to engineer with findings so far. Ask, "what should happen as wind velocity increases?":

*Upper limit on electricity generated, but otherwise, the larger the wind velocity, the more electricity generated.*

- ▶ Mathematically, sounds like *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  $a$  —  $a$  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?

- ▶ Same idea as for parabola: 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 call 1 over wind\_velocity `wind_pace`.
- ▶ Make a scatterplot first to check for straightness (next page)

```
windmill$wind_pace=1/windmill$wind_velocity
g=ggplot(windmill,aes(y=DC_output,x=wind_pace))+
 geom_point() + geom_smooth(se=F)
```

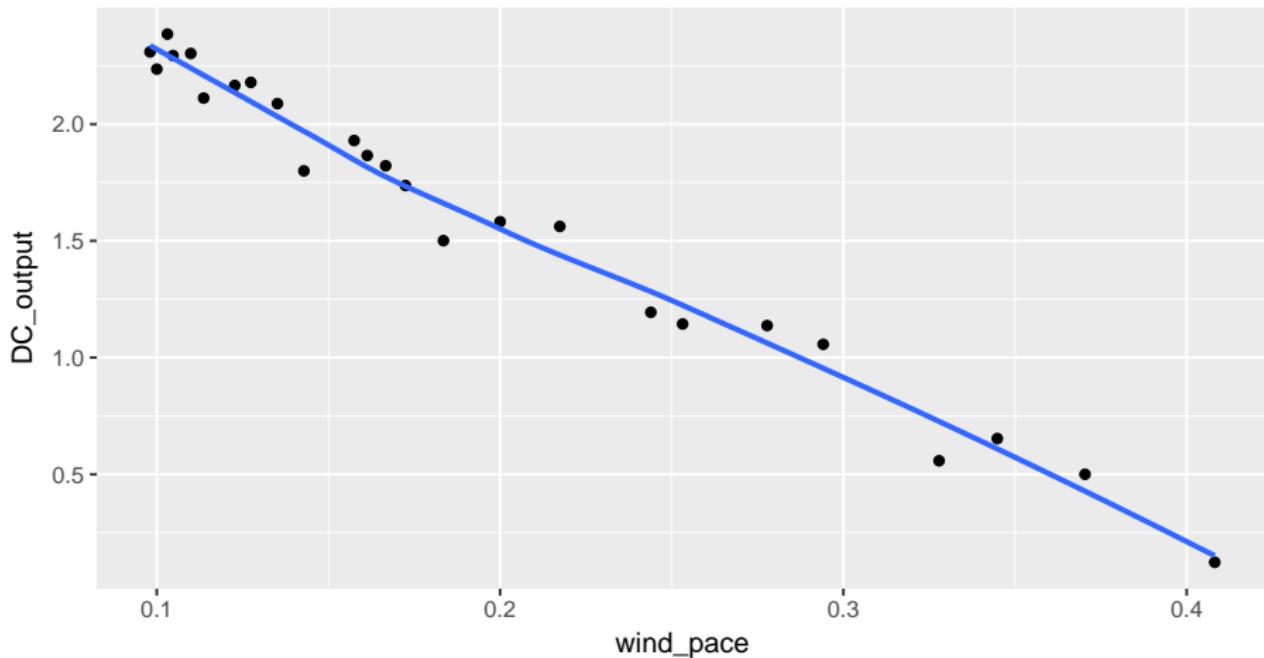
- ▶ and run regression like this (page after):

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

## Scatterplot for wind\_pace

g

```
'geom_smooth()' using method = 'loess'
```



That's pretty straight.

## Regression output

```

Call:
lm(formula = DC_output ~ wind_pace, data = windmill)

Residuals:
Min 1Q Median 3Q Max
-0.20547 -0.04940 0.01100 0.08352 0.12204

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 2.9789 0.0449 66.34 <2e-16 ***
wind_pace -6.9345 0.2064 -33.59 <2e-16 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.09417 on 23 degrees of freedom
Multiple R-squared: 0.98, Adjusted R-squared: 0.9792
F-statistic: 1128 on 1 and 23 DF, p-value: < 2.2e-16
```

## Comments

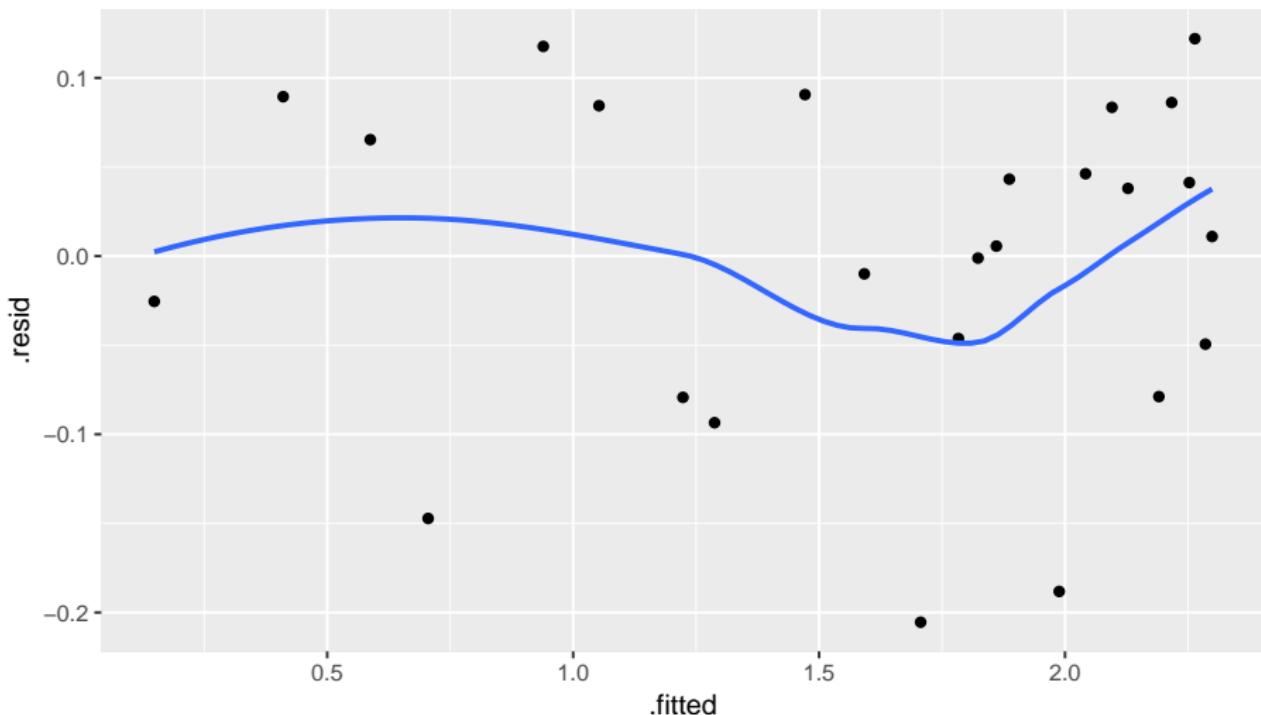
- ▶ R-squared, 98%, even higher than for parabola model (97%).
- ▶ Simpler model, only one explanatory variable (`wind.pace`) vs. 2 for parabola model (`wind.velocity` and `vel2`).
- ▶ `wind.pace` (unsurprisingly) strongly significant.
- ▶ Looks good, but check residual plot:

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

## Residual plot for asymptote model

g

```
'geom_smooth()' using method = 'loess'
```



## 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 5.00 1.582 1.3366195 1.5057592 1.5919507
2 6.00 1.822 1.5777683 1.8093653 1.8231023
3 3.40 1.057 0.9507813 0.8614064 0.9392874
4 2.70 0.500 0.7819771 0.5181275 0.4105093
5 10.00 2.236 2.5423638 2.2613723 2.2854054
6 9.70 2.386 2.4700192 2.2697860 2.2639584
7 9.55 2.294 2.4338468 2.2714197 2.2527296
8 3.05 0.558 0.8663792 0.6944367 0.7052381
9 8.15 2.166 2.0962384 2.2039449 2.1279955
10 6.20 1.866 1.6259981 1.8609376 1.8603848
... with 15 more rows
```

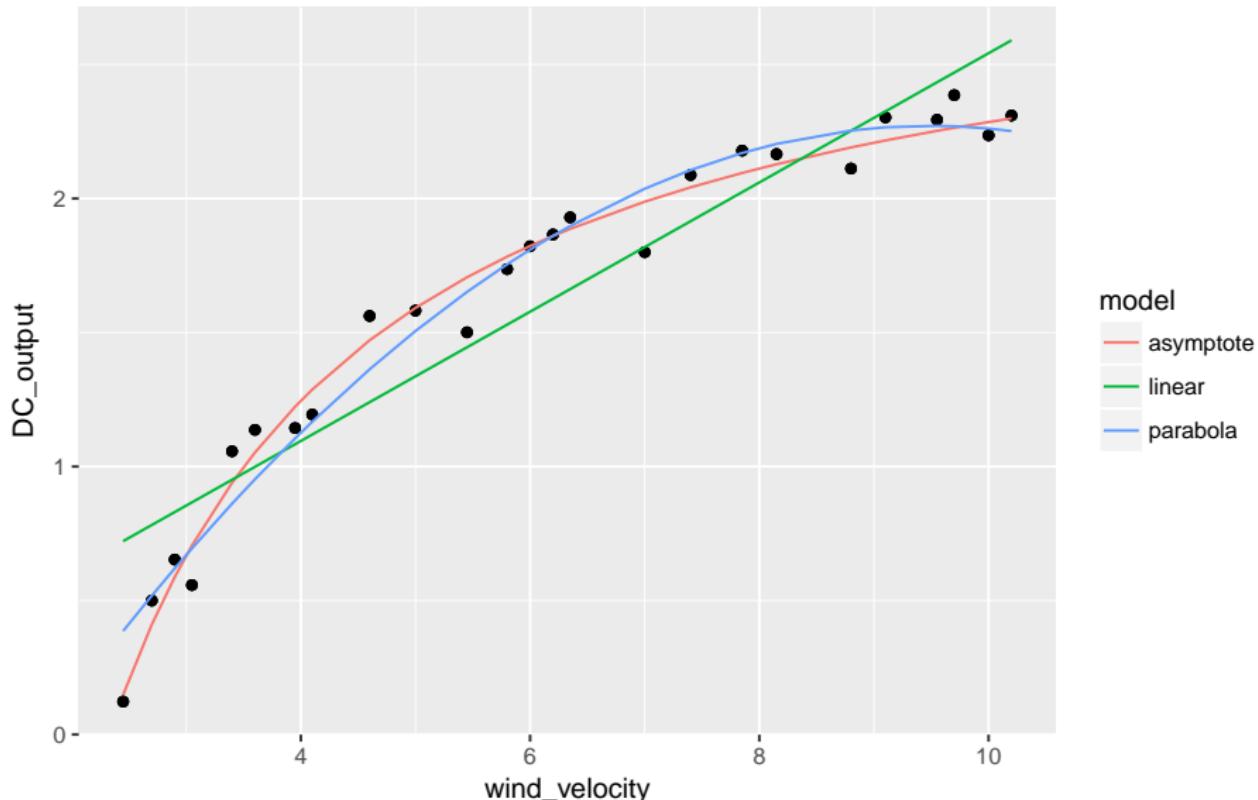
## Making the plot

- ▶ `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.
- ▶ `gather`, then plot:

```
g=w2 %>% gather(model,fit,linear:asymptote) %>%
 ggplot(aes(x=wind_velocity,y=DC_output))+
 geom_point()+
 geom_line(aes(y=fit,colour=model))
```

# Scatterplot with fitted curves

g



## 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

```
summary(DC.3)
```

```

Call:
lm(formula = DC_output ~ wind_pace, data = windmill)

Residuals:
Min 1Q Median 3Q Max
-0.20547 -0.04940 0.01100 0.08352 0.12204

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 2.9789 0.0449 66.34 <2e-16 ***
wind_pace -6.9345 0.2064 -33.59 <2e-16 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.09417 on 23 degrees of freedom
Multiple R-squared: 0.98, Adjusted R-squared: 0.9792
F-statistic: 1128 on 1 and 23 DF, p-value: < 2.2e-16
```

## 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 5.5
[11] 6.0 6.5 7.0 7.5 8.0 8.5 9.0 9.5 10.0 10.5
[21] 11.0 11.5 12.0 12.5 13.0 13.5 14.0 14.5 15.0 15.5
[31] 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 1.0 1.0000000
2 1.5 0.6666667
3 2.0 0.5000000
4 2.5 0.4000000
5 3.0 0.3333333
6 3.5 0.2857143
7 4.0 0.2500000
8 4.5 0.2222222
9 5.0 0.2000000
10 5.5 0.1818182
... 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 0.3720240 -0.4710832 -3.9556872
2 1.5 0.4925984 -0.1572664 -1.6441714
3 2.0 0.6131729 0.1374900 -0.4884135
4 2.5 0.7337473 0.4131860 0.2050412
5 3.0 0.8543217 0.6698215 0.6673444
6 3.5 0.9748962 0.9073966 0.9975609
7 4.0 1.0954706 1.1259112 1.2452233
8 4.5 1.2160450 1.3253654 1.4378497
9 5.0 1.3366195 1.5057592 1.5919507
10 5.5 1.4571939 1.6670925 1.7180334
... with 21 more rows
```

## Making a plot

- To make a plot, we use the same trick as last time to get all three predictions on a plot with a legend:

```
g=my_fits %>% gather(model,fit,
 linear:asymptote) %>%
 ggplot(aes(y=fit,x=wind_velocity,
 colour=model))+geom_line()
```

- The observed wind velocities were in this range:

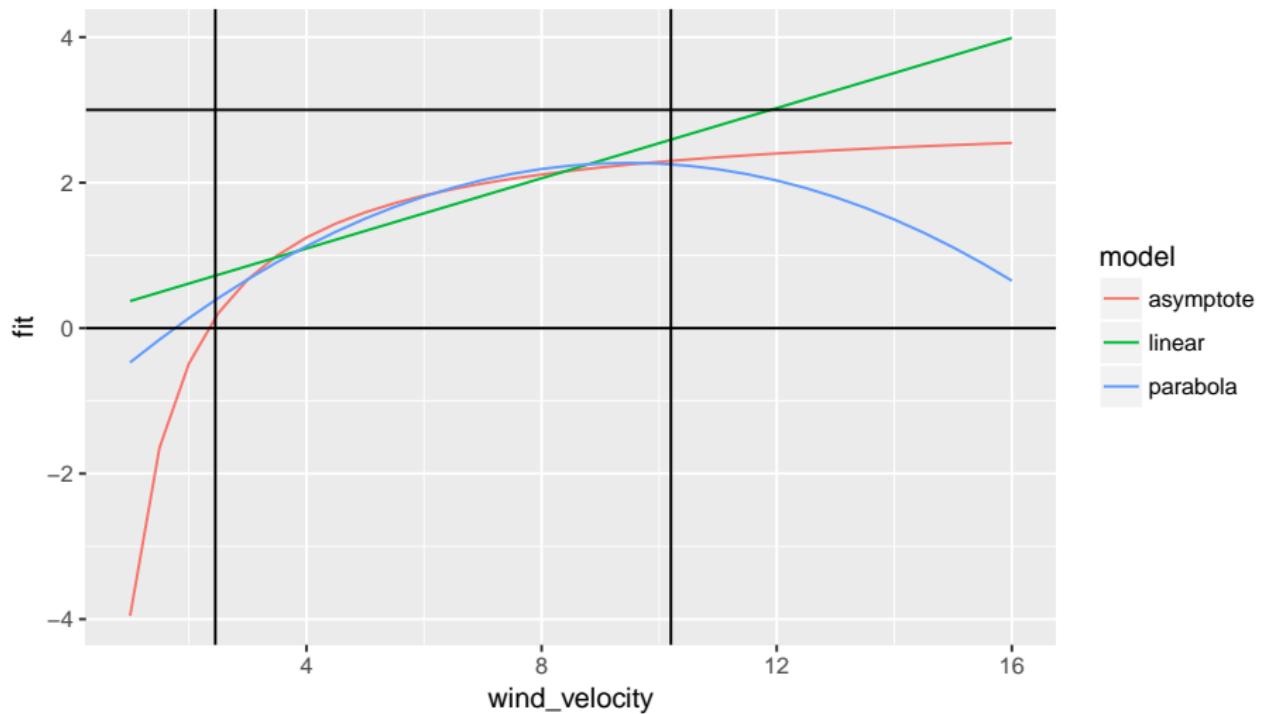
```
vels=range(windmill$wind_velocity) ; vels
[1] 2.45 10.20
```

- DC.output between 0 and 3 from asymptote model. Add lines to graph:

```
g2=g+geom_hline(yintercept=0)+geom_hline(yintercept=3)+
 geom_vline(xintercept=vels[1])+
 geom_vline(xintercept=vels[2])
```

# The plot

g2



## 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?
- ▶ For parabola model:

```
coef(DC.2)
```

```
(Intercept) wind_velocity
-1.15589824 0.72293590
I(wind_velocity^2)
-0.03812088
```

- ▶ Nope, goes to  $-1.15$  (intercept), actually significantly different from zero.

## Comments (2)

- ▶ What about asymptote model?

`coef(DC.3)`

```
(Intercept) wind_pace
2.978860 -6.934547
```

- ▶ As `wind.velocity` heads to 0, `wind.pace` heads to  $+\infty$ , so `DC.output` heads to  $-\infty$ !
- ▶ Predicted `DC.output` crosses 0 approx when ( $w$  is `wind.pace` and  $v$  `wind.velocity`)  $3 - 7w = 0$ , ie.  $w = 3/7$  and  $v = 7/3$ , and is negative below that — nonsense!
- ▶ 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.”

## Section 10

Case study 2: Electricity, peak hour demand and total energy usage

## Another regression example (SAS)

- ▶ Electric utility company wants to relate peak-hour demand (kW) to total energy usage (kWh) during a month.
- ▶ Important planning problem, because generation system must be large enough to meet maximum demand.
- ▶ Data from 53 residential customers from August.
- ▶ Read in data and draw scatterplot:

```
proc import
 datafile='/home/ken/utility.txt'
 dbms=dlm
 out=util
 replace;
 delimiter=' ' ;
 getnames=yes;
```

## Check data

The first few rows, which look reasonable:

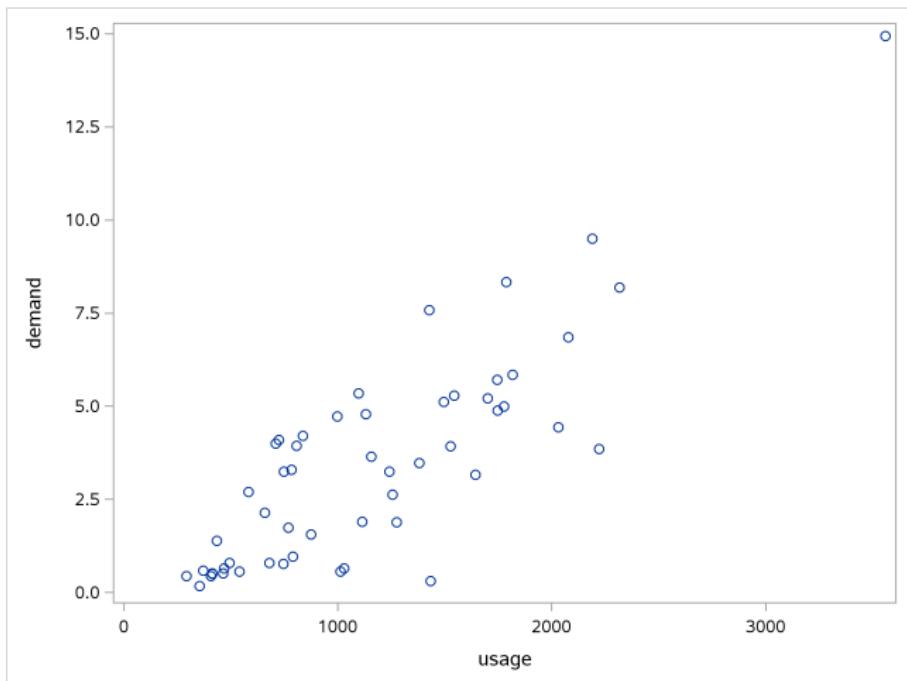
```
proc print data=util(obs=8);
```

| Obs | usage | demand |
|-----|-------|--------|
| 1   | 679   | 0.79   |
| 2   | 292   | 0.44   |
| 3   | 1012  | 0.56   |
| 4   | 493   | 0.79   |
| 5   | 582   | 2.7    |
| 6   | 1156  | 3.64   |
| 7   | 997   | 4.73   |
| 8   | 2189  | 9.5    |

Make a scatterplot:

```
proc sgplot;
 scatter x=usage y=demand;
```

# Scatterplot



## Fitting a regression

- ▶ Concern: outlier top right (though appears to be legit values)
- ▶ Trend basically straight, and outlier appears to be on it.
- ▶ So try fitting regression:

```
proc reg;
 model demand=usage;
```

# Regression output

The REG Procedure  
Model: MODEL1  
Dependent Variable: demand

Root MSE                    1.57720      R-Square        0.7046  
Dependent Mean            3.41321      Adj R-Sq        0.6988  
Coeff Var                46.20882

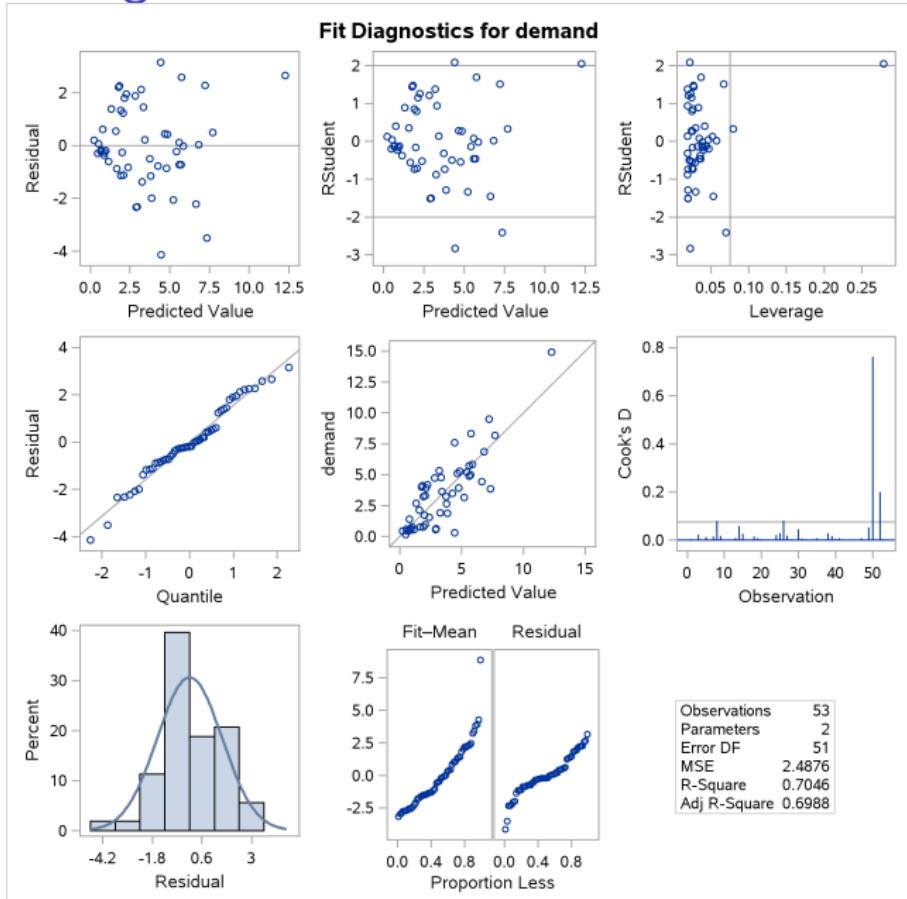
## Parameter Estimates

| Variable  | DF | Parameter Estimate | Standard Error | t Value | Pr >  t |
|-----------|----|--------------------|----------------|---------|---------|
| Intercept | 1  | -0.83130           | 0.44161        | -1.88   | 0.0655  |
| usage     | 1  | 0.00368            | 0.00033390     | 11.03   | <.0001  |

## Comments

- ▶ R-squared 70%: not bad!
- ▶ Statistically significant slope: demand really does depend on usage.
- ▶ But should look at residuals.
- ▶ Output from regression also includes array of “diagnostic plots”, over:

# Regression diagnostics



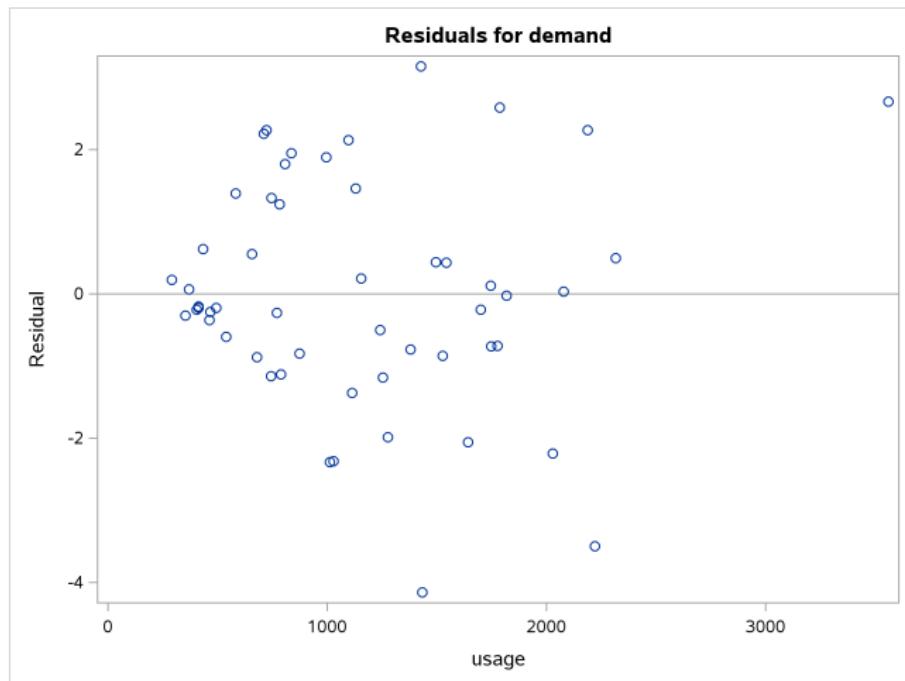
## What the diagnostic plots show

Counting from top left, along the rows:

1. Regular residual plot, against fitted values
2. Standardized residuals against fitted values, with the advantage that the standardized residuals behave like z-scores
3. Standardized residuals against leverages (high leverage means unusual  $x$ )
4. normal quantile plot of residuals
5. response against predicted
6. Cook's distance (overall influence) against observation number
7. histogram (with normal curve) of residuals
8. I never use this one!
9. summary of regression

Over, residuals against  $x$ 's (only one here, usage).

# Residual plot



## General comments on these plots

- ▶ I usually look at plots #1 and #4 of the diagnostic plots, and maybe the big plot of residuals against  $x$ .
- ▶ Plot of residuals against fitted values shows (if it has a pattern) any problems with the regression.
- ▶ Residuals should be approx. normal. Normal quantile plot shows if they are not.
- ▶ Plot of residuals against  $x$ 's show any problems with that particular  $x$  (eg. nonlinearity). With only one  $x$ , same conclusion as residual plot.
- ▶ Look at leverages/Cook's distances to see if any unusually large ones.

## Comments for these data

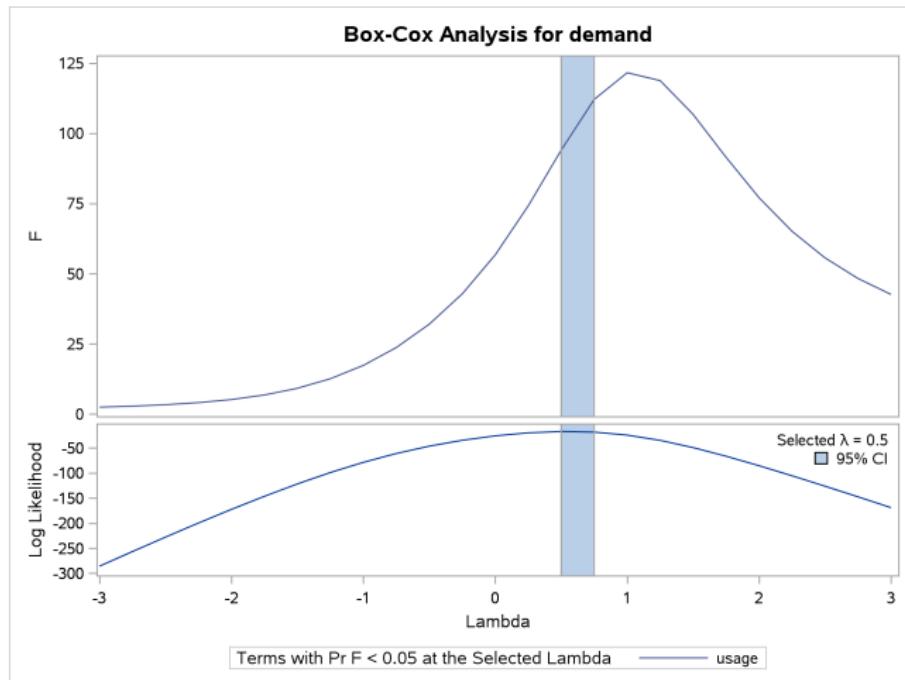
- ▶ No trend in residuals vs. fitted
- ▶ but: residuals for demand close to 0 are themselves close to zero
- ▶ and: residuals for larger demand tend to get farther from zero
- ▶ at least up to demand 5 or so.
- ▶ One of the assumptions hiding behind regression is that residuals should be of equal size, not “fanning out” as here.
- ▶ Remedy: transformation of response variable.
- ▶ Note: there is one point with large leverage, the observation with large usage and demand.

## But what transformation?

- ▶ Best way: consult with person who brought you the data.
- ▶ Can't do that here!
- ▶ No idea what transformation would be good.
- ▶ Let data choose: “Box-Cox transformation”.
- ▶ Scale is that of “ladder of powers”: power transformation, but 0 is log.
- ▶ SAS: proc transreg:

```
proc transreg;
 model boxcox(demand)=identity(usage);
```

# Output (graph)



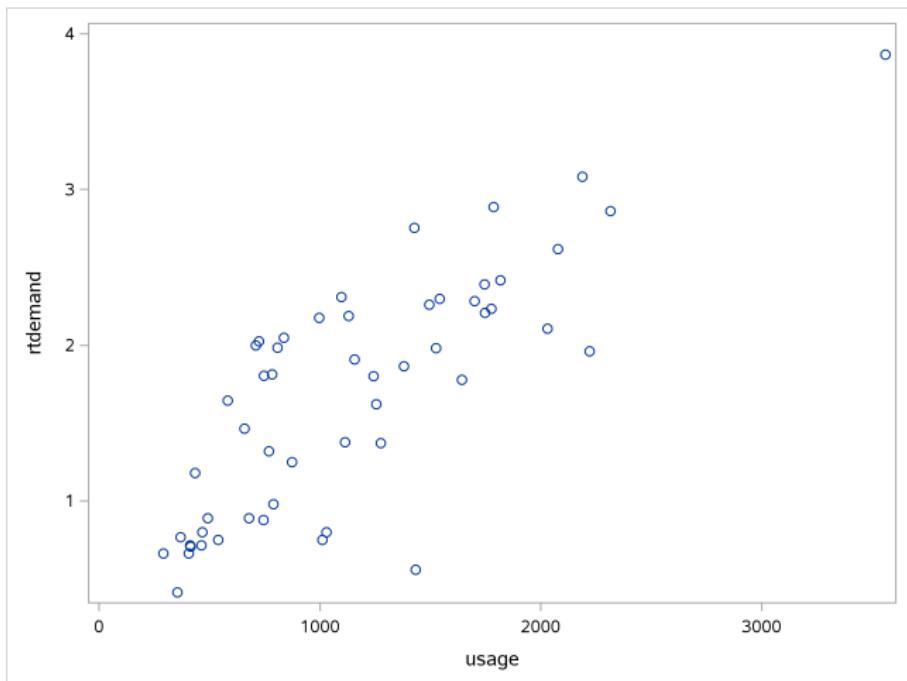
## Comments

- ▶ SAS finds best transformation, here power 0.50.
- ▶ Also gives you a CI for power, here 0.50 to 0.75.
- ▶ Ideal transformation should be defensible power, typically from set  $\{-1, -0.5, 0, 0.5, 1, 2\}$ . Here that would be power 0.5, which would be square root.
- ▶ Try that and see how it looks.
- ▶ Create another new data set by bringing in everything from old one and make a scatterplot:

```
data trans;
 set util;
 rtdemand=sqrt(demand);

proc sgplot;
 scatter x=usage y=rtdemand;
```

## New scatterplot



## Regression with new response variable

- ▶ Scatter plot still looks straight.
- ▶ Data set `trans` is most recently-created (default) one, so used in scatterplot above and `proc reg` below. We save residuals and fitted values in output data set, which then becomes default:

```
proc reg;
 model rtDemand=usage;
 output out=outRT r=res p=fit;
```

# Output

The REG Procedure  
Model: MODEL1  
Dependent Variable: rtDemand

Root MSE                    0.46404      R-Square            0.6485  
Dependent Mean            1.68040      Adj R-Sq           0.6416  
Coeff Var                27.61503

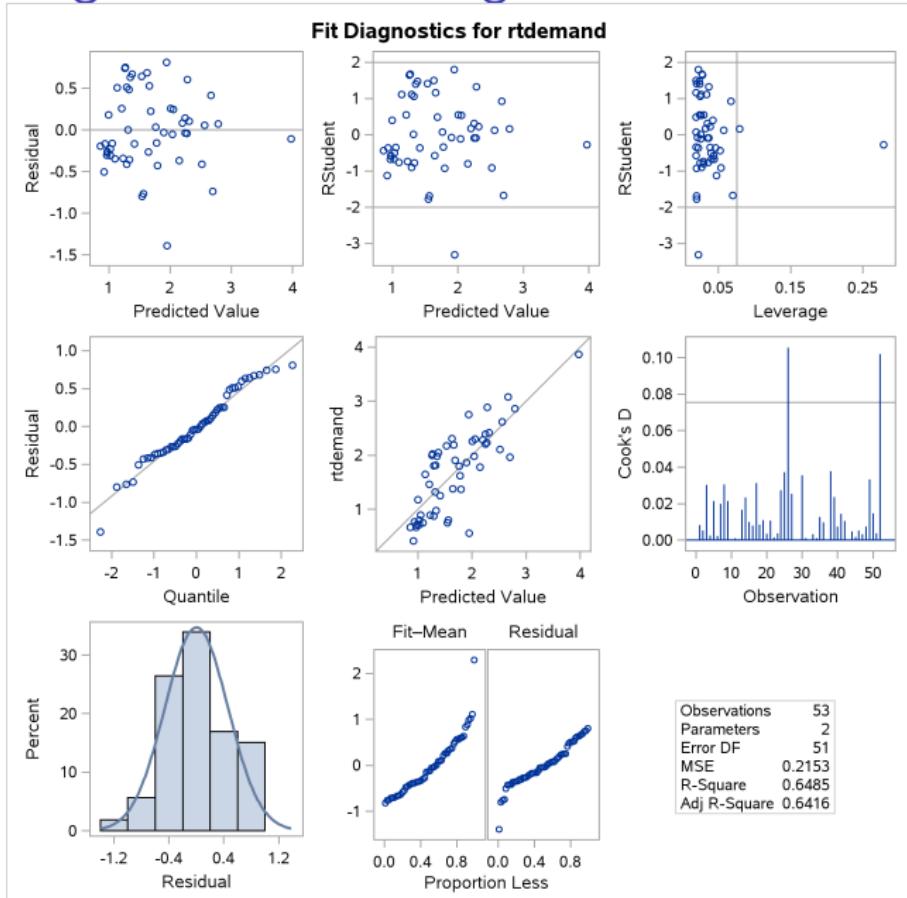
## Parameter Estimates

| Variable  | DF | Parameter Estimate | Standard Error | t Value | Pr >  t |
|-----------|----|--------------------|----------------|---------|---------|
| Intercept | 1  | 0.58223            | 0.12993        | 4.48    | <.0001  |
| usage     | 1  | 0.00095286         | 0.00009824     | 9.70    | <.0001  |

## Comments

- ▶ R-squared actually decreased (from 70% to 65%).
- ▶ Slope still strongly significant.
- ▶ Should take a look at residuals now (over):

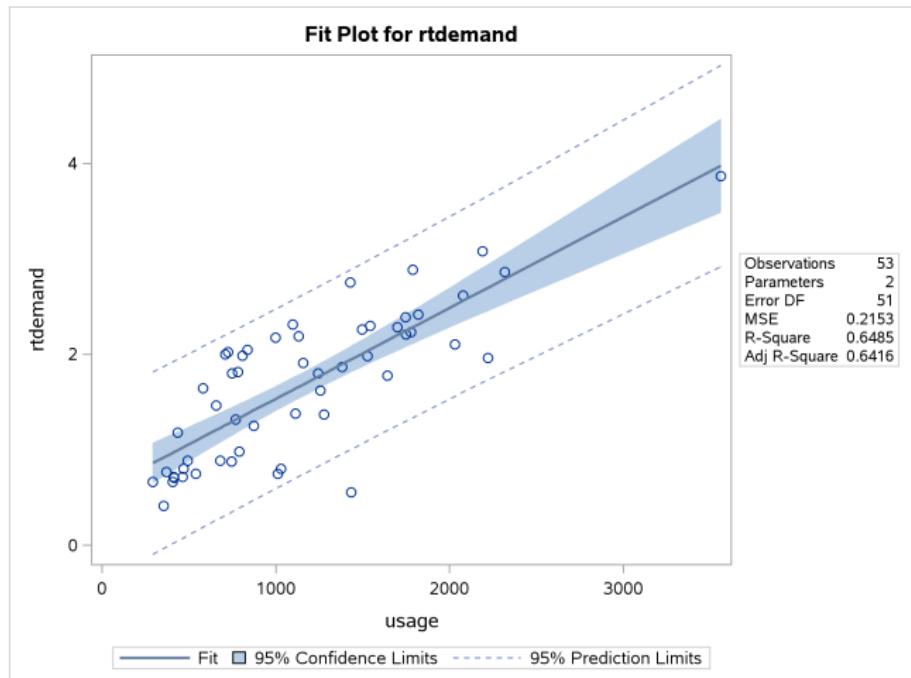
# Residual diagnostics for 2nd regression



## Comments

- ▶ Better. No trends, approx. constant variability.
- ▶ One mildly suspicious outlier at the bottom.
- ▶ Can trust this regression.
- ▶ Better a lower R-squared from a regression we can trust than a higher one from one we cannot.
- ▶ Look at scatterplot of `rtdemand` against `usage` with regression line on it (in graphics output from regression).

# Scatterplot with fitted line



## Predictions

- When we transformed the response variable, have to think carefully about predictions. Using `usage=1000`, and with R as calculator:

```
int=0.58223
slope=0.00095286
pred=int+slope*1000
pred
[1] 1.53509
```

- It's a prediction, but of the response variable in regression, which was `rtdemand`, square root of demand.
- To predict actual demand, need to undo the transformation.
- Undoing square root is *squaring*:

```
pred^2
[1] 2.356501
```

## More predictions

- ▶ For usage 1000, 2000, 3000 all at once:

```
usage=c(1000,2000,3000)
rt.demand=int+slope*usage
demand=rt.demand^2
demand
[1] 2.356501 6.189895 11.839173
```

- ▶ Transformations are non-linear changes.
- ▶ Here, though the usage values equally spaced, predicted demand values are not.
- ▶ Larger gap between 2nd and 3rd than 1st and 2nd.

## Section 11

### Case study 3: Asphalt

## The asphalt data

- ▶ 31 asphalt pavements prepared under different conditions. How does quality of pavement depend on these?

- ▶ Variables:

`pct.a.surf` The percentage of asphalt in the surface layer

`pct.a.base` The percentage of asphalt in the base layer

`fines` The percentage of fines in the surface layer

`voids` The percentage of voids in the surface layer

`rut.depth` The change in rut depth per million vehicle passes

`viscosity` The viscosity of the asphalt

`run` 2 data collection periods: `run 1` for run 1, 0 for run 2.

- ▶ `rut.depth` response. Depends on other variables, how? R this time.

## Getting set up

- ▶ Read in data:

```
asphalt=read_delim("asphalt.txt", " ")

Parsed with column specification:
cols(
pct.a.surf = col_double(),
pct.a.base = col_double(),
fines = col_double(),
voids = col_double(),
rut.depth = col_double(),
viscosity = col_double(),
run = col_integer()
)
```

- ▶ Quantitative variables with one response: multiple regression.
- ▶ Some issues here that don't come up in "simple" regression; handle as we go. (STAB27/STAC67 ideas.)

## The data

asphalt

```
A tibble: 31 x 7
pct.a.surf pct.a.base fines voids rut.depth
<dbl> <dbl> <dbl> <dbl> <dbl>
1 4.68 4.87 8.4 4.916 6.75
2 5.19 4.50 6.5 4.563 13.00
3 4.82 4.73 7.9 5.321 14.75
4 4.85 4.76 8.3 4.865 12.60
5 4.86 4.95 8.4 3.776 8.25
6 5.16 4.45 7.4 4.397 10.67
7 4.82 5.05 6.8 4.867 7.28
8 4.86 4.70 8.6 4.828 12.67
9 4.78 4.84 6.7 4.865 12.58
10 5.16 4.76 7.7 4.034 20.60
... with 21 more rows, and 2 more variables:
viscosity <dbl>, run <int>
```

## Plotting response “rut depth” against everything else

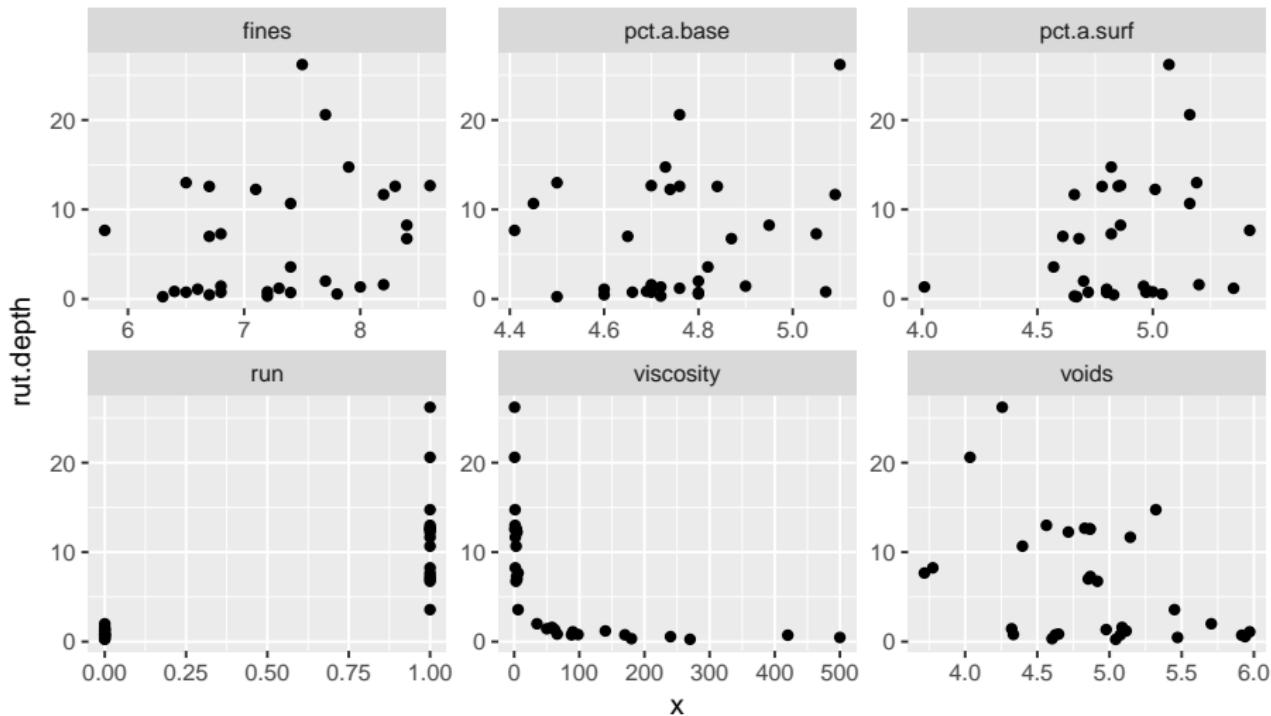
Same idea as for plotting separate predictions on one plot:

```
g=asphalt %>% gather(xname,x,
 c(pct.a.surf:voids,viscosity:run)) %>%
 ggplot(aes(x=x,y=rut.depth))+geom_point() +
 facet_wrap(~xname,scales="free")
```

“gather all the x-variables together into one column called *x*, with another column *xname* saying which *x* they were, then plot these *x*'s against *rut.depth*, a separate facet for each *x*-variable.”

# The plot

g



## Interpreting the plots

- ▶ One plot of rut depth against each of the six other variables.
- ▶ Get rough idea of what's going on.
- ▶ Trends mostly weak.
- ▶ Viscosity has strong but non-linear trend.
- ▶ Run has effect but variability bigger when run is 1.
- ▶ Weak but downward trend for voids.
- ▶ Non-linearity of rut.depth-viscosity relationship should concern us.
- ▶ Take this back to asphalt engineer: suggests *log* of viscosity.

## Log of “viscosity”: more nearly linear?

- ▶ Create new variable *in data frame* to hold log of viscosity:

```
asphalt_lv=asphalt %>% mutate(log.viscosity=log(viscosity))
```

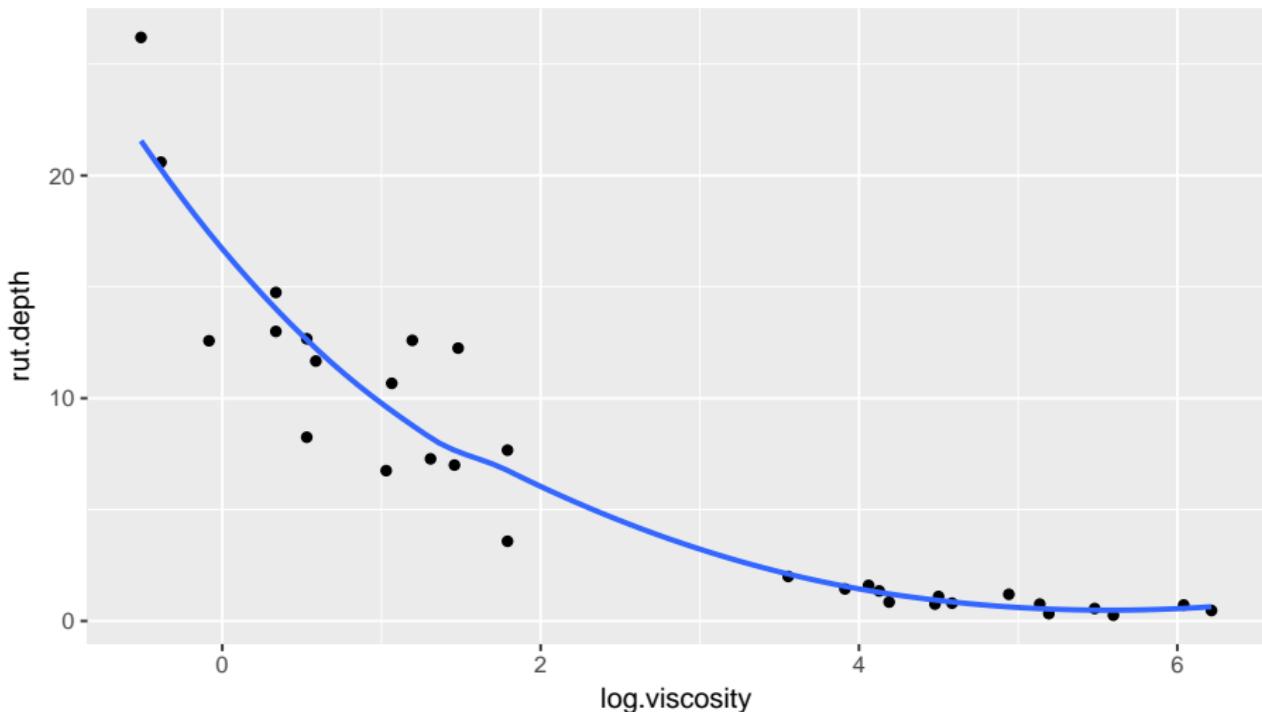
Now we have to remember to use `asphalt_lv` as our data frame from here on:

```
g=ggplot(asphalt_lv,aes(y=rut.depth,x=log.viscosity))+
 geom_point()+geom_smooth(se=F)
```

## Rut depth against log-viscosity

g

```
'geom_smooth()' using method = 'loess'
```



## Comments and next steps

- ▶ Not *very* linear, but better than before.
- ▶ In multiple regression, hard to guess which  $x$ 's affect response. So typically start by predicting from *everything* else.
- ▶ Model formula has response on left, squiggle, explanatorys on right joined by plusses:

```
rut.1=lm(rut.depth~pct.a.surf+pct.a.base+fines+
 voids+log.viscosity+run,data=asphalt_lv)
```

## Regression output

```
summary(rut.1)
```

```

Call:
lm(formula = rut.depth ~ pct.a.surf + pct.a.base + fines + voids +
log.viscosity + run, data = asphalt_lv)

Residuals:
Min 1Q Median 3Q Max
-4.1211 -1.9075 -0.7175 1.6382 9.5947

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) -12.9937 26.2188 -0.496 0.6247
pct.a.surf 3.9706 2.4966 1.590 0.1248
pct.a.base 1.2631 3.9703 0.318 0.7531
fines 0.1164 1.0124 0.115 0.9094
voids 0.5893 1.3244 0.445 0.6604
log.viscosity -3.1515 0.9194 -3.428 0.0022 **
run -1.9655 3.6472 -0.539 0.5949

Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

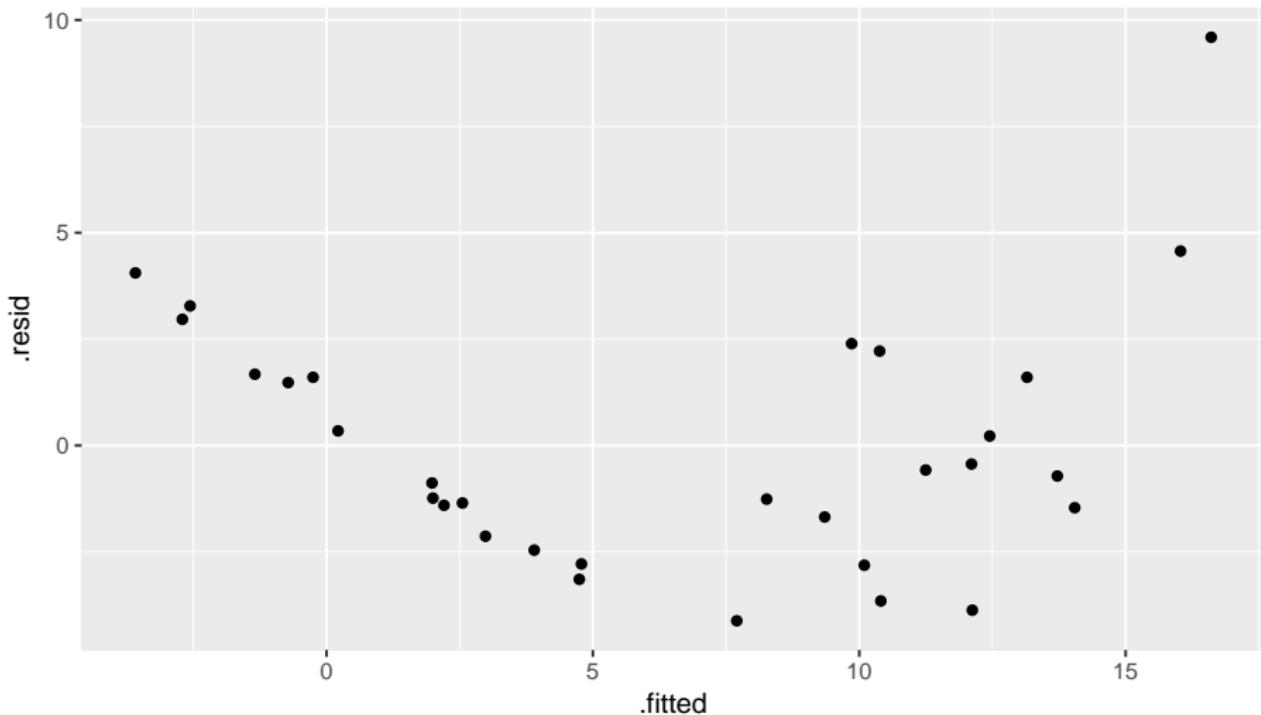
Residual standard error: 3.324 on 24 degrees of freedom
Multiple R-squared: 0.806, Adjusted R-squared: 0.7575
F-statistic: 16.62 on 6 and 24 DF, p-value: 1.743e-07
```

## Comments

- ▶ At bottom: R-squared 81%, not so bad.
- ▶  $F$ -statistic and P-value assert that *something* helping to predict `rut.depth`.
- ▶ Table of coefficients says `log.viscosity`.
- ▶ But confused by clearly non-significant variables: remove those to get clearer picture of what *is* helpful.
- ▶ Before we do anything, look at residual plots:
  - (a) of residuals against fitted values (as usual)
  - (b) of residuals against *each explanatory*.
- ▶ Problem with (a): fix response variable; problem with some plots in (b): fix those explanatory variables.

## Plot fitted values against residuals

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



## Plotting residuals against $x$ variables

- ▶ Problem here is that residuals are in the fitted model, and the observed  $x$ -values are in the original data frame `asphalt_lv`.
- ▶ Package `broom` contains a function `augment` that combines these two together so that they can later be plotted: a model first, and then a data frame:

```
rut.1a=augment(rut.1,asphalt_lv)

Warning: Deprecated: please use
'purrr::possibly()' instead

Warning: Deprecated: please use
'purrr::possibly()' instead
```

## What does rut.1a contain?

```
rut.1a %>% as_tibble()

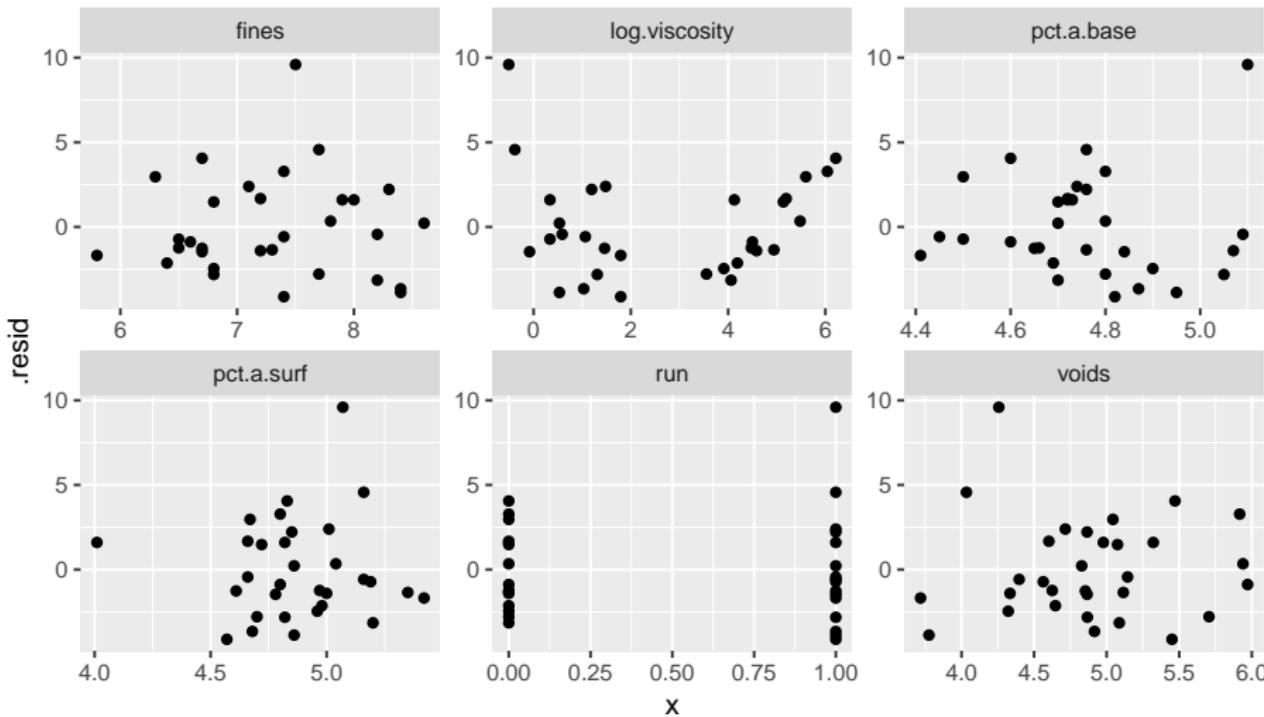
A tibble: 31 x 15
pct.a.surf pct.a.base fines voids rut.depth
<dbl> <dbl> <dbl> <dbl> <dbl>
1 4.68 4.87 8.4 4.916 6.75
2 5.19 4.50 6.5 4.563 13.00
3 4.82 4.73 7.9 5.321 14.75
4 4.85 4.76 8.3 4.865 12.60
5 4.86 4.95 8.4 3.776 8.25
6 5.16 4.45 7.4 4.397 10.67
7 4.82 5.05 6.8 4.867 7.28
8 4.86 4.70 8.6 4.828 12.67
9 4.78 4.84 6.7 4.865 12.58
10 5.16 4.76 7.7 4.034 20.60
... with 21 more rows, and 10 more variables:
viscosity <dbl>, run <int>, log.viscosity <dbl>,
.fitted <dbl>, .se.fit <dbl>, .resid <dbl>,
.hat <dbl>, .sigma <dbl>, .cooksdi <dbl>,
.std.resid <dbl>
```

## Plotting residuals against x-variables

```
g=rut.1a %>% gather(xname,x,
 c(pct.a.surf:voids,run,log.viscosity)) %>%
 ggplot(aes(x=x,y=.resid))+
 geom_point() + facet_wrap(~xname,scales="free")
```

# The plot

g



## Comments

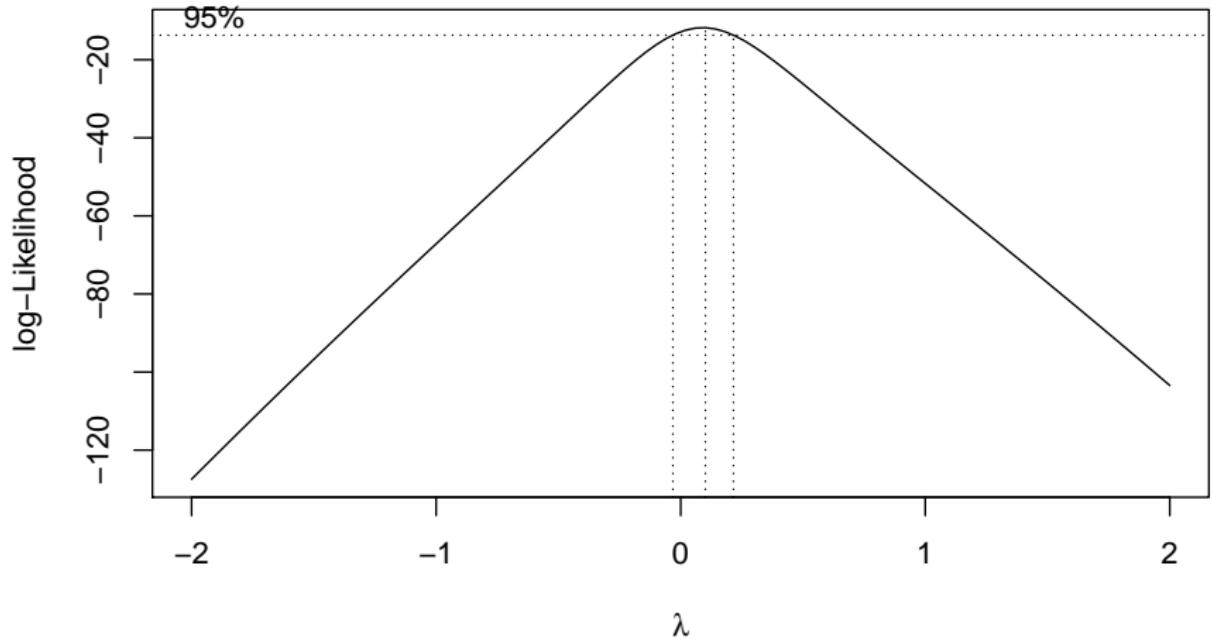
- ▶ There is serious curve in plot of residuals vs. fitted values. Suggests a transformation of  $y$ .
- ▶ The residuals-vs- $x$ 's plots don't know any serious trends. Worst probably that potential curve against log-viscosity.
- ▶ Also, large positive residual, 10, that shows up on all plots. Perhaps transformation of  $y$  will help with this too.
- ▶ If residual-fitted plot OK, but some residual- $x$  plots not, try transforming *those*  $x$ 's, eg. by adding  $x^2$  to help with curve.

## Which transformation?

- ▶ I have no idea what would be a good transformation.
- ▶ Box-Cox again?

```
boxcox(rut.depth~pct.a.surf+pct.a.base+fines+voids+
log.viscosity+run, data=asphalt_lv)
```

## Box-Cox plot



## Comments on Box-Cox plot

- ▶ Best single choice of transformation parameter  $\lambda$  is peak of curve, close to 0.
- ▶ Vertical dotted lines give CI for  $\lambda$ , about  $(-0.05, 0.2)$ .
- ▶  $\lambda = 0$  means “log”.
- ▶ Narrowness of confidence interval mean that these *not* supported by data:
  - ▶ No transformation ( $\lambda = 1$ )
  - ▶ Square root ( $\lambda = 0.5$ )
  - ▶ Reciprocal ( $\lambda = -1$ ).
- ▶ So make new data frame with new response variable:

```
asphalt_2=asphalt_lv %>%
 mutate(log.rut.depth=log(rut.depth))
```

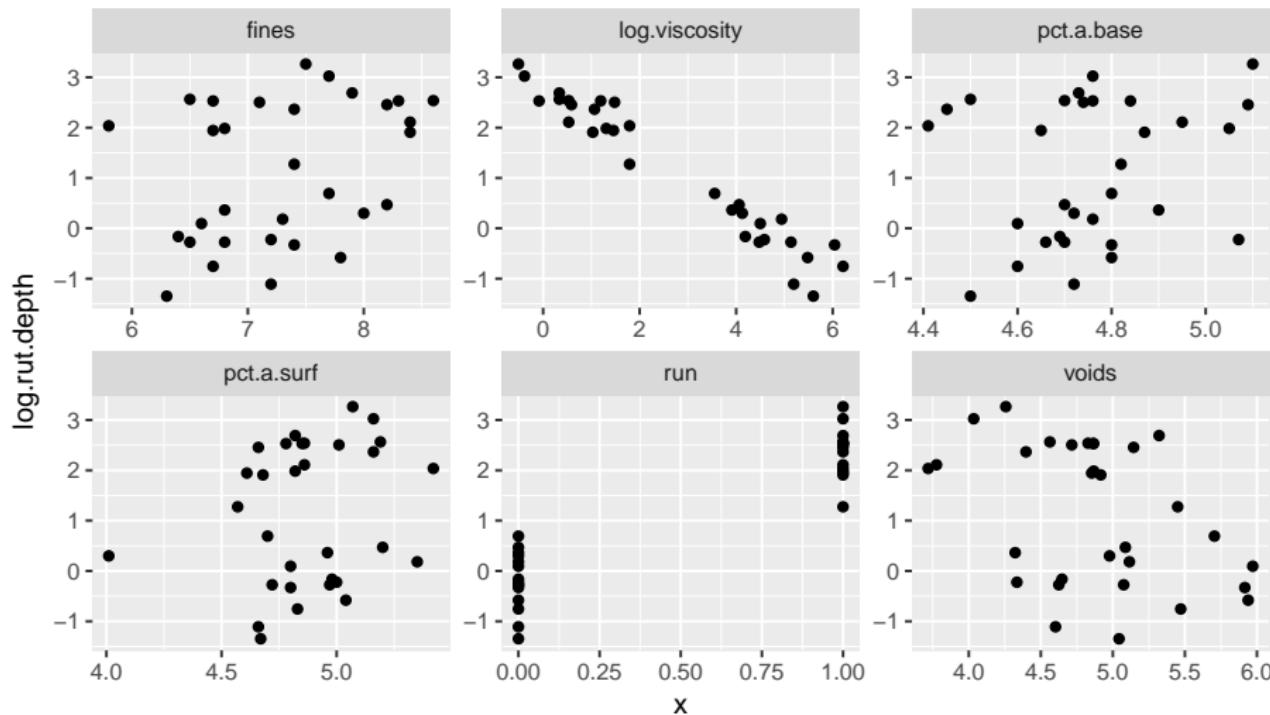
## Relationships with explanatory variables

- ▶ As before: plot response (now log.rut.depth) against other explanatory variables, all in one shot:

```
g3=asphalt_2 %>% gather(xname,x,
 c(pct.a.surf:voids,run:log.viscosity)) %>%
 ggplot(aes(y=log.rut.depth,x=x))+geom_point() +
 facet_wrap(~xname,scales="free")
```

# The new plots

g3



## Modelling with transformed response

- ▶ These trends look pretty straight, especially with log.viscosity.
- ▶ Values of log.rut.depth for each run have same spread.
- ▶ Other trends weak, but are straight if they exist.
- ▶ Start modelling from the beginning again.
- ▶ Model log.rut.depth in terms of everything else, see what can be removed.
- ▶ Off we go:

```
rut.2=lm(log.rut.depth~pct.a.surf+pct.a.base+
 fines+voids+log.viscosity+run, data=asphalt_2)
```

- ▶ broom has function tidy that displays just the coefficients, exactly what we need.

# Output

```
tidy(rut.2)

term estimate std.error statistic p.value
1 (Intercept) -1.57298922 2.43617401 -0.6456802 5.246124e-01
2 pct.a.surf 0.58357559 0.23198113 2.5156167 1.898184e-02
3 pct.a.base -0.10336733 0.36890801 -0.2801981 7.817265e-01
4 fines 0.09775203 0.09406823 1.0391609 3.090864e-01
5 voids 0.19885377 0.12305882 1.6159245 1.191815e-01
6 log.viscosity -0.55768728 0.08543236 -6.5278223 9.445184e-07
7 run 0.34004833 0.33888780 1.0034245 3.256663e-01
```

## Taking out everything non-significant

- ▶ Try: remove everything but pct.a.surf and log.viscosity:

```
rut.3=lm(log.rut.depth~pct.a.surf+log.viscosity,data=asphalt_2)
```

- ▶ Check that removing all those variables wasn't too much:

```
anova(rut.3,rut.2)
```

```
Analysis of Variance Table
##
Model 1: log.rut.depth ~ pct.a.surf + log.viscosity
Model 2: log.rut.depth ~ pct.a.surf + pct.a.base + fines + voids +
run
Res.Df RSS Df Sum of Sq F Pr(>F)
1 28 2.8809
2 24 2.2888 4 0.59216 1.5523 0.2191
```

- ▶  $H_0$ : two models equally good;  $H_a$ : bigger model better.
- ▶ Null not rejected here; small model as good as the big one, so prefer simpler smaller model rut.3.

## Take out least significant in sequence

```
tidy(rut.2)

term estimate std.error statistic p.value
1 (Intercept) -1.57298922 2.43617401 -0.6456802 5.246124e-01
2 pct.a.surf 0.58357559 0.23198113 2.5156167 1.898184e-02
3 pct.a.base -0.10336733 0.36890801 -0.2801981 7.817265e-01
4 fines 0.09775203 0.09406823 1.0391609 3.090864e-01
5 voids 0.19885377 0.12305882 1.6159245 1.191815e-01
6 log.viscosity -0.55768728 0.08543236 -6.5278223 9.445184e-07
7 run 0.34004833 0.33888780 1.0034245 3.256663e-01
```

Get used to reading scientific notation! Largest P-value is 0.78 for `pct.a.base`, not significant.

Or, as over.

## Put the P-values in order

```
tidy(rut.2) %>% arrange(p.value)

term estimate std.error statistic p.value
1 log.viscosity -0.55768728 0.08543236 -6.5278223 9.445184e-07
2 pct.a.surf 0.58357559 0.23198113 2.5156167 1.898184e-02
3 voids 0.19885377 0.12305882 1.6159245 1.191815e-01
4 fines 0.09775203 0.09406823 1.0391609 3.090864e-01
5 run 0.34004833 0.33888780 1.0034245 3.256663e-01
6 (Intercept) -1.57298922 2.43617401 -0.6456802 5.246124e-01
7 pct.a.base -0.10336733 0.36890801 -0.2801981 7.817265e-01
```

The largest one is at the bottom.

## Take out pct.a.base:

```
rut.4=lm(log.rut.depth~pct.a.surf+fines+voids+log.viscosity+run,
 data=asphalt_2)
tidy(rut.4) %>% arrange(p.value)

term estimate std.error statistic p.value
1 log.viscosity -0.55249041 0.08184342 -6.750578 4.483621e-07
2 pct.a.surf 0.59299473 0.22526263 2.632459 1.432182e-02
3 voids 0.20046739 0.12063727 1.661737 1.090561e-01
4 (Intercept) -2.07849871 1.60665073 -1.293684 2.075999e-01
5 run 0.35977353 0.32532859 1.105877 2.793085e-01
6 fines 0.08894579 0.08701332 1.022209 3.164725e-01
```

fines is next to go, P-value 0.32.

# “Update”

Another way to do the same thing:

```
rut.4=update(rut.2,.~.-pct.a.base)
tidy(rut.4) %>% arrange(p.value)

term estimate std.error statistic p.value
1 log.viscosity -0.55249041 0.08184342 -6.750578 4.483621e-07
2 pct.a.surf 0.59299473 0.22526263 2.632459 1.432182e-02
3 voids 0.20046739 0.12063727 1.661737 1.090561e-01
4 (Intercept) -2.07849871 1.60665073 -1.293684 2.075999e-01
5 run 0.35977353 0.32532859 1.105877 2.793085e-01
6 fines 0.08894579 0.08701332 1.022209 3.164725e-01
```

Again, `fines` is the one to go. (Output identical as it should be.)

## Take out fines:

```
rut.5=update(rut.4,.~.-fines)
tidy(rut.5) %>% arrange(p.value)

term estimate std.error statistic p.value
1 log.viscosity -0.5803887 0.07722544 -7.5155118 5.591602e-08
2 pct.a.surf 0.5483723 0.22118326 2.4792668 1.997130e-02
3 voids 0.2318797 0.11675845 1.9859780 5.767397e-02
4 run 0.2946793 0.31931076 0.9228604 3.645654e-01
5 (Intercept) -1.2553309 1.39146826 -0.9021628 3.752525e-01
```

Can't take out intercept, so run, with P-value 0.36, goes next.

## Take out run:

```
rut.6=update(rut.5,.~.-run)
tidy(rut.6) %>% arrange(p.value)

term estimate std.error statistic p.value
1 log.viscosity -0.6464911 0.02878643 -22.4581853 5.288267e-19
2 pct.a.surf 0.5554686 0.22044153 2.5198000 1.796333e-02
3 voids 0.2447934 0.11559805 2.1176259 4.356036e-02
4 (Intercept) -1.0207945 1.36430001 -0.7482185 4.607966e-01
```

Again, can't take out intercept, so largest P-value is for voids, 0.044. But this is significant, so we shouldn't remove voids.

## Comments

- ▶ Here we stop: pct.a.surf, voids and log.viscosity would all make fit significantly worse if removed. So they stay.
- ▶ Different final result from taking things out one at a time (top), than by taking out 4 at once (bottom):

```
coef(rut.6)

(Intercept) pct.a.surf voids log.viscosity
-1.0207945 0.5554686 0.2447934 -0.6464911

coef(rut.3)

(Intercept) pct.a.surf log.viscosity
0.9001389 0.3911481 -0.6185628
```

- ▶ I like one-at-a-time method better, as general strategy.
- ▶ Point: Can make difference which way we go.

## Comments on variable selection

- ▶ Best way to decide which  $x$ 's belong: *expert knowledge*: which of them *should* be important.
- ▶ Best automatic method: what we did, “backward selection” .
- ▶ Do *not* learn about “stepwise regression”! See notes for a reference.
- ▶ R has function `step` that does backward selection, like this:

```
step(rut.2,direction="backward")
```

Gets same answer as we did (by removing least significant  $x$ ).

- ▶ Removing non-significant  $x$ 's may remove interesting ones whose P-values happened not to reach 0.05. Consider using less stringent cutoff like 0.20 or even bigger.
- ▶ Can also fit all possible regressions, as over (may need to do `install.packages("leaps")` first.

# All possible regressions

```
leaps=regsubsets(log.rut.depth~pct.a.surf+pct.a.base+fines+voids+
 log.viscosity+run,data=asphalt_2,nbest=2)
s=summary(leaps)
with(s,data.frame(rsq,outmat))

rsq pct.a.surf pct.a.base fines voids log.viscosity run
1 (1) 0.9452562
1 (2) 0.8624107
2 (1) 0.9508647 *
2 (2) 0.9479541
3 (1) 0.9578631 *
3 (2) 0.9534561 *
4 (1) 0.9591996 *
4 (2) 0.9589206 *
5 (1) 0.9608365 *
5 (2) 0.9593265 * *
6 (1) 0.9609642 * * * *
```

“Best” model in the sense of highest R-squared has *everything* in it.

## Comments

- ▶ Problem: even adding a worthless  $x$  increases R-squared. So try for line where R-squared stops increasing “too much”, eg. top line (just `log.viscosity`), first 3-variable line (backwards-elimination model). Hard to judge.
- ▶ One solution (STAC67): *adjusted R-squared*, where adding worthless variable makes it go *down*.
- ▶ `data.frame` rather than `tibble` because there are several columns in `outmat`.

## All possible regressions, adjusted R-squared

```
with(s,data.frame(adjr2,outmat))

adjr2 pct.a.surf pct.a.base fines voids log.viscosity run
1 (1) 0.9433685
1 (2) 0.8576662
2 (1) 0.9473550 *
2 (2) 0.9442365
3 (1) 0.9531812 *
3 (2) 0.9482845 * *
4 (1) 0.9529226 *
4 (2) 0.9526007 * * *
5 (1) 0.9530038 *
5 (2) 0.9511918 * * * *
6 (1) 0.9512052 * * * * *
```

Backward-elimination model comes out best again.

## Revisiting the best model

- Best model was our rut.6:

```
summary(rut.6)

##
Call:
lm(formula = log.rut.depth ~ pct.a.surf + voids + log.viscosity,
data = asphalt_2)
##
Residuals:
Min 1Q Median 3Q Max
-0.53548 -0.20181 -0.01702 0.16748 0.54707
##
Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.02079 1.36430 -0.748 0.4608
pct.a.surf 0.55547 0.22044 2.520 0.0180 *
voids 0.24479 0.11560 2.118 0.0436 *
log.viscosity -0.64649 0.02879 -22.458 <2e-16 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
Residual standard error: 0.3025 on 27 degrees of freedom
Multiple R-squared: 0.9579, Adjusted R-squared: 0.9532
F-statistic: 204.6 on 3 and 27 DF, p-value: < 2.2e-16
```

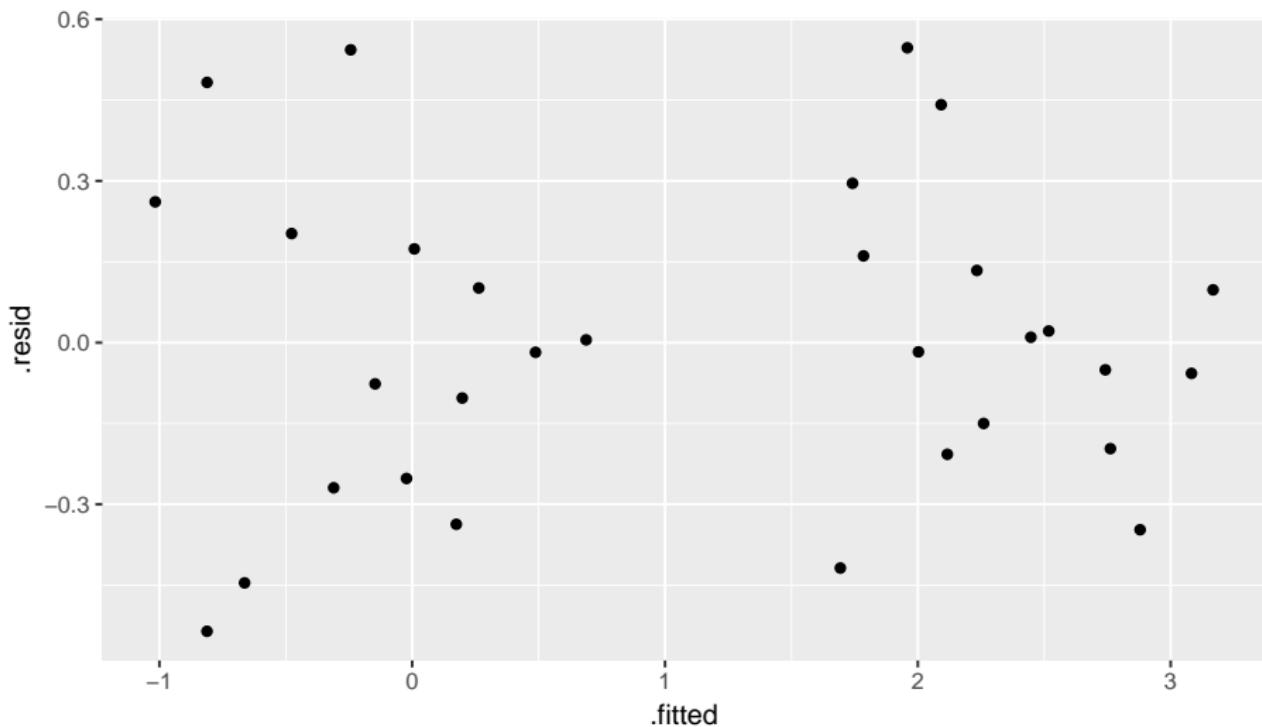
## Revisiting (2)

- ▶ Regression slopes say that rut depth increases as log-viscosity decreases, pct.a.surf increases and voids increases. This more or less checks out with out scatterplots against log.viscosity.
- ▶ We should check residual plots again, though previous scatterplots say it's unlikely that there will be a problem:

```
g=ggplot(rut.6,aes(y=.resid,x=.fitted))+geom_point()
```

## Residuals against fitted values

g



## Plotting residuals against x's

- Do our trick again to put them all on one plot:

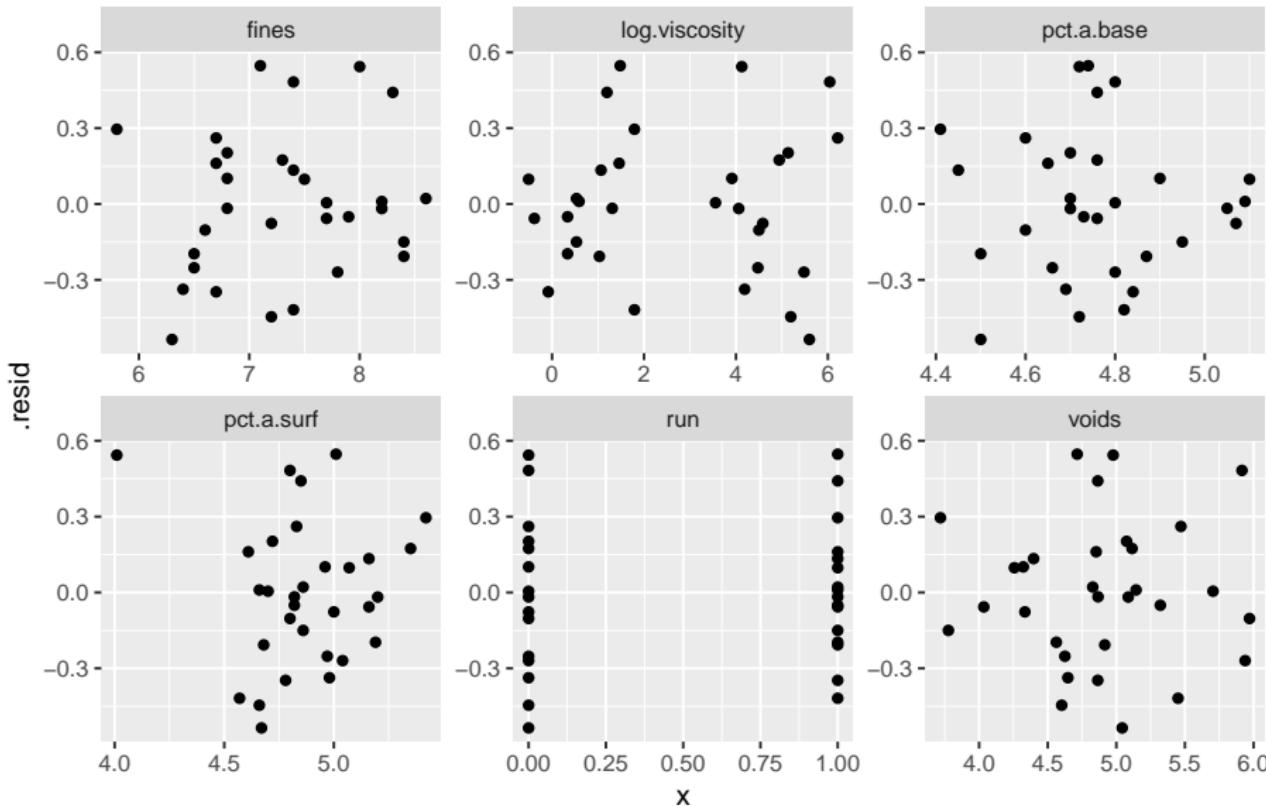
```
g2=augment(rut.6,asphalt_2) %>%
 gather(xname,x,
 c(pct.a.surf:voids,run:log.viscosity)) %>%
 ggplot(aes(y=.resid,x=x))+geom_point()+
 facet_wrap(~xname,scales="free")

Warning: Deprecated: please use
'purrr::possibly()' instead

Warning: Deprecated: please use
'purrr::possibly()' instead
```

# Residuals against the x's

g2



## Comments

- ▶ None of the plots show any sort of pattern. The points all look random on each plot.
- ▶ On the plot of fitted values (and on the one of `log.viscosity`), the points seem to form a “left half” and a “right half” with a gap in the middle. This is not a concern.
- ▶ One of the `pct.a.surf` values is low outlier (4), shows up top left of that plot.
- ▶ Only two possible values of `run`; the points in each group look randomly scattered around 0, with equal spreads.
- ▶ Residuals seem to go above zero further than below, suggesting a mild non-normality, but not enough to be a problem.

## Variable-selection strategies

- ▶ Expert knowledge.
- ▶ Backward elimination.
- ▶ All possible regressions.
- ▶ Taking a variety of models to experts and asking their opinion.
- ▶ Use a looser cutoff to eliminate variables in backward elimination (eg. only if P-value greater than 0.20).
- ▶ If goal is *prediction*, eliminating worthless variables less important.
- ▶ If goal is *understanding*, want to eliminate worthless variables where possible.
- ▶ Results of variable selection not always reproducible, so caution advised.

## Section 12

### Functions in R

## Don't repeat yourself

- ▶ See this:

```
a=50 ; b=11 ; d=3
as=sqrt(a-1)
as
[1] 7

bs=sqrt(b-1)
bs
[1] 3.162278

ds=sqrt(d-1)
ds
[1] 1.414214
```

- ▶ Same calculation done three different times, by copying, pasting and editing.
- ▶ *Dangerous:* what if you forget to change something after you pasted?

## Anatomy of function

- ▶ *Header line* with function name and input value(s).
- ▶ *Body* with calculation of values to return.
- ▶ *Return value*: the output from function.

In our case:

```
sqrt_minus_1=function(x) {
 ans=sqrt(x-1)
 return(ans)
}
```

or more simply

```
sqrt_minus_1=function(x) {
 sqrt(x-1)
}
```

If last line of function calculates value without saving it, *that* value is returned.

## About the input; testing

- ▶ The input to a function can be called *anything*. Here we called it x. This is the name used *inside* the function.
- ▶ The function is a “machine” for calculating square-root-minus-1. It doesn’t do anything until you *call* it:

```
sqrt_minus_1(50)
```

```
[1] 7
```

```
sqrt_minus_1(11)
```

```
[1] 3.162278
```

```
sqrt_minus_1(3)
```

```
[1] 1.414214
```

- ▶ It works!

# Vectorization

- We conceived our function to work on numbers:

```
sqrt_minus_1(3.25)
```

```
[1] 1.5
```

- but it actually works on vectors too, as a free bonus of R:

```
sqrt_minus_1(c(50,11,3))
```

```
[1] 7.000000 3.162278 1.414214
```

- or even data frames:

```
d=tibble(x=1:2,y=3:4)
```

```
sqrt_minus_1(d)
```

```
x y
```

```
1 0 1.414214
```

```
2 1 1.732051
```

## More than one input

- ▶ Allow the value to be subtracted, before taking square root, to be input to function as well, thus:

```
sqrt_minus_value=function(x,d) {
 sqrt(x-d)
}
```

- ▶ Call the function with the x and d inputs in the right order:

```
sqrt_minus_value(51,2)
[1] 7
```

- ▶ or give the inputs names, *in which case they can be in any order*:

```
sqrt_minus_value(d=2,x=51)
[1] 7
```

## Defaults 1/2

- ▶ Many R functions have values that you can change if you want to, but usually you don't want to, for example:

```
x=c(3,4,5,NA,6,7)
mean(x)
[1] NA

mean(x,na.rm=T)
[1] 5
```

- ▶ By default, the mean of data with a missing value is missing, but if you specify `na.rm=T`, the missing values are removed before the mean is calculated.
- ▶ That is, `na.rm` has a default value of `F`: that's what it will be unless you change it.

## Defaults 2/2

- In our function, set a default value for d like this:

```
sqrt_minus_value=function(x,d=1) {
 sqrt(x-d)
}
```

- If you don't specify a value for d, 1 will be used:

```
sqrt_minus_value(51,2)
[1] 7

sqrt_minus_value(51)
[1] 7.071068
```

## Catching errors before they happen

- ▶ What happened here?

```
sqrt_minus_value(6,8)
Warning in sqrt(x - d): NaNs produced
[1] NaN
```

- ▶ Message not helpful. *Actually*, function tried to take square root of negative number.
- ▶ In fact, not even error, just warning.
- ▶ *Check* that the square root will be OK first. Here's how:

```
sqrt_minus_value=function(x,d=1) {
 stopifnot(x-d>=0)
 sqrt(x-d)
}
```

## What happens with stopifnot

- ▶ This should be good, and is:

```
sqrt_minus_value(8,6)
```

```
[1] 1.414214
```

- ▶ This should fail, and see how it does:

```
sqrt_minus_value(6,8)
```

```
Error: x - d >= 0 is not TRUE
```

- ▶ Where the function fails, we get informative error, but if everything good, the stopifnot does nothing.
- ▶ stopifnot contains one or more logical conditions, and *all of them* have to be true for function to work. So put in everything that you want to be *true*.

## Using R's built-ins

- ▶ When you write a function, you can use *anything* built-in to R, or even any functions that you defined before.
- ▶ For example, if you will be calculating a lot of regression-line slopes, you don't have to do this from scratch: you can use R's regression calculations, like this:

```
xx=1:4
yy=c(10,11,10,14)
yy.1=lm(yy~xx)
coef(yy.1)

(Intercept) xx
8.5 1.1
```

- ▶ These are the intercept and the slope, in that order.

# Is this the right thing?

Check by looking at the `summary` output from the regression:

```
summary(yy.1)

##
Call:
lm(formula = yy ~ xx)
##
Residuals:
1 2 3 4
0.4 0.3 -1.8 1.1
##
Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 8.5000 1.8775 4.527 0.0455 *
xx 1.1000 0.6856 1.605 0.2498

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
Residual standard error: 1.533 on 2 degrees of freedom
Multiple R-squared: 0.5628, Adjusted R-squared: 0.3442
F-statistic: 2.574 on 1 and 2 DF, p-value: 0.2498
```

## Making this into a function

- ▶ First step: make sure you have it working without a function.
- ▶ We do: fit an `lm` and take the second thing out of `coef`.
- ▶ Two inputs, the `x` and the `y`, which I take in *that order*.
- ▶ Output: just the slope (we throw away intercept). Thus:

```
slope=function(x,y) {
 y.1=lm(y~x)
 ans=coef(y.1)
 ans[2]
}
```

- ▶ Check using our data from before: correct:

```
slope(xx,yy)
x
1.1
```

## Passing things on

- ▶ `lm` has a lot of options, with defaults, that we might want to change. Instead of intercepting all the possibilities and passing them on, we can do this:

```
slope=function(x,y,...) {
 y.1=lm(y~x,...)
 ans=coef(y.1)
 ans[2]
}
```

- ▶ The `...` in the header line means “accept any other input”, and the `...` in the `lm` line means “pass anything other than `x` and `y` straight on to `lm`”.

## Using ...

- ▶ One of the things `lm` will accept is a vector called `subset` containing the list of observations to include in the regression.
- ▶ So we should be able to do this:

```
xx
[1] 1 2 3 4

yy
[1] 10 11 10 14

slope(xx,yy,subset=1:2)

x
1
```

- ▶ Just uses the first two observations in `xx` and `yy`, so the slope should be  $(11 - 10)/(2 - 1) = 1$  and is.

## Running a function for each of several inputs

- ▶ Suppose we have a data frame containing several different x's to use in regressions:

```
d=tibble(x1=1:4,x2=c(8,7,6,5),x3=c(2,4,6,9))
d

A tibble: 4 x 3
x1 x2 x3
<int> <dbl> <dbl>
1 1 8 2
2 2 7 4
3 3 6 6
4 4 5 9
```

- ▶ Want to use these as different x's for a regression with our yy as the response, and collect together the three different slopes.
- ▶ Python-like way: a for loop.
- ▶ R-like way: `map_dbl`: less coding, but more thinking.

## The loop way

- ▶ “Pull out” column  $i$  of data frame  $d$  as  $d \%>\% \text{pull}(i)$ .
- ▶ Create empty vector `slopes` to store the slopes.
- ▶ Looping variable  $i$  goes from 1 to 3 (3 columns, thus 3 slopes):

```
slopes=numeric(3)
for (i in 1:3) {
 xx=d %>% pull(i)
 slopes[i]=slope(xx,yy)
}
slopes
[1] 1.1000000 -1.1000000 0.5140187
```

- ▶ Check this by doing the three `lm`'s, one at a time.

## The `map dbl` way

- ▶ “for each of these (columns of `d`), run this function (`slope`), with constant extra input (`yy`), and collect together the answers”.
- ▶ Since `slope` returns a decimal number (a `dbl`), appropriate function-running function is `map dbl`:

```
map_dbl(d, slope, yy)
```

```
x1 x2 x3
1.1000000 -1.1000000 0.5140187
```

- ▶ Same as loop, with a lot less coding.
- ▶ “Find the square roots of each of the numbers 1 through 10”:

```
map_dbl(1:10, sqrt)
```

```
[1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751
[8] 2.828427 3.000000 3.162278
```

## Summarizing all columns of a data frame

```
d
A tibble: 4 x 3
x1 x2 x3
<int> <dbl> <dbl>
1 1 8 2
2 2 7 4
3 3 6 6
4 4 5 9

map_dbl(d, mean)
x1 x2 x3
2.50 6.50 5.25
```

The mean of each column, with the columns labelled.

## What if summary returns more than one thing?

- ▶ For example:

```
quartiles=function(x) {
 quantile(x,c(0.25,0.75))
}
quartiles(1:5)
25% 75%
2 4
```

- ▶ When function returns more than one thing, `map` instead of `map_dbl`.

## map results

- ▶ Try:

```
map(d,quartiles)

$x1
25% 75%
1.75 3.25

$x2
25% 75%
5.75 7.25

$x3
25% 75%
3.50 6.75
```

- ▶ A list. Better:

```
map(d,quartiles) %>% bind_rows()

A tibble: 2 x 3
x1 x2 x3
<dbl> <dbl> <dbl>
1 1.75 5.75 3.50
2 3.25 7.25 6.75
```

Or even

```
d %>% map(quartiles) %>% bind_rows()

A tibble: 2 x 3
x1 x2 x3
<dbl> <dbl> <dbl>
1 1.75 5.75 3.50
2 3.25 7.25 6.75
```

This works because the implicit first thing in `map` is (the columns of) the data frame that came out of the previous step.

These are 1st and 3rd quartiles of each column of `d`, according to R's default definition (see help for `quantile`).

## Another example: the geometric distribution

- ▶ Recall *binomial distribution*, eg. toss coin 10 times and count how many heads ( $W$ ).
- ▶ In general, prob. of success =  $p$  on every independent trial. Fixed # trials,  $W$  is #successes.
- ▶ Another angle: *how many trials to get my first success?*
- ▶ Random variable now #trials (denote  $X$ ); #successes fixed (= 1).
- ▶ *Geometric distribution.*
- ▶  $P(X = 1) = p$  (success first time).
- ▶  $P(X = 2) = (1 - p)p$  (fail, then succeed).
- ▶  $P(X = 3) = (1 - p)^2 p$  (fail 2 times, then succeed).
- ▶  $P(X = n) = (1 - p)^{n-1} p$  (fail  $n - 1$  times, then succeed).
- ▶ Implement in R.

## Writing a geometric probability function

- ▶ Input: #trials whose prob. we want x, single-trial success prob. p.  
Output: probability of succeeding for 1st time after exactly x trials (number).
- ▶ This:

```
geometric=function(x,p)
{
 p*(1-p)^(x-1)
}
```

- ▶ Testing:

```
geometric(1,0.4)
```

```
[1] 0.4
```

```
geometric(2,0.4)
```

```
[1] 0.24
```

Prob. of succeeding first time same as  $p$ : good; chance of first success on second trial? Fail, then succeed:  $(0.6)(0.4) = 0.24$ .

## Errors

- ▶ What if user gives  $p$  outside of  $[0, 1]$ , or  $x$  less than 1?
- ▶ Function actually gives answer, but it is nonsense!

```
geometric(0, 0.5)
```

```
[1] 1
```

```
geometric(2, 1.1)
```

```
[1] -0.11
```

- ▶ Ugh!

## Catching errors: stopifnot

- ▶ 3 things to check:  $p \geq 0$  or bigger,  $p \leq 1$  or smaller,  $x \geq 1$  or bigger:

```
geometric=function(x,p)
{
 stopifnot(p>=0,p<=1,x>=1)
 p*(1-p)^(x-1)
}
```

# Testing

- ▶ Test:

```
geometric(2,0.5)
[1] 0.25

geometric(0,0.5)
Error: x >= 1 is not TRUE

geometric(2,1.1)
Error: p <= 1 is not TRUE
```

Last two fail, and `stopifnot` tells you why.

## Another way

- ▶  $P(X = 1) = p, P(X = k) = (1 - p)^{k-1}p.$
- ▶ After first trial, each probability is  $1 - p$  times the one before.
- ▶ Thus, this actually works, even though it appears to define a function in terms of itself:

```
geometric2=function(x,p) {
 stopifnot(p>=0,p<=1,x>=1)
 if (x==1) {
 p
 } else {
 (1-p)*geometric2(x-1,p)
 }
}
```

## Testing version 2

```
geometric2(1,0.4)
[1] 0.4
geometric2(2,0.4)
[1] 0.24
geometric2(2,0.5)
[1] 0.25
```

Same as before.

## Calling geometric with vector x

- ▶ What happens?
- ▶ Try it and see.

```
geometric(1:5,0.5)
```

```
[1] 0.50000 0.25000 0.12500 0.06250 0.03125
```

- ▶ Probabilities of first success taking 1, 2, 3, ... trials.
- ▶ Works because of how R handles vector arithmetic.
- ▶ R freebie: often get vector output from vector input with no extra coding.
- ▶ Above gives ingredients for “first success in 5 trials or less”: calculate prob of 1 to 5, then add up:

```
sum(geometric(1:5,0.5))
```

```
[1] 0.96875
```

- ▶ Might know this as “cumulative probability”.

## Function input

- If we use function as above, have to get inputs in *right order*:

```
geometric(2,0.8)
[1] 0.16

geometric(0.8,2)
Error: p <= 1 is not TRUE
```

- Second one fails because it thinks 2 is success probability.
- But if we *use the names*, can do any order:

```
geometric(x=2,p=0.8)
[1] 0.16

geometric(p=0.8,x=2)
[1] 0.16
```

## Cumulative probabilities as function

- ▶ Might be useful to have function for cumulative probabilities.
- ▶ Strategy: get individual probs as far as you wish to go, then add up.  
Eg. probability of 4 or less: need 1 through 4. In general,  $x$  or less with success prob.  $p$ :

```
c.geometric=function(x,p)
{
 probs=geometric(1:x,p)
 sum(probs)
}
```

- ▶ Easy to write, using our `geometric` function and stuff in R.

## Testing `c.geometric`

- ▶ Try the one we just did:

```
c.geometric(5,0.5)
[1] 0.96875
```

Answer we had before.

- ▶ How about this:

```
c.geometric(20,0.1)
[1] 0.8784233
```

If success probability only 0.1, might even take longer than 20 trials to get first success. So this is reasonable.

## Digression: mean trials until 1st success

- ▶ Mean number of trials until 1st success is  $1/p$ :
  - ▶  $p = 0.5$ , mean #trials is  $1/0.5 = 2$ .
  - ▶  $p = 0.1$ , mean #trials is  $1/0.1 = 10$ .
- ▶ Can we calculate this? Get all the probabilities, multiply each probability by its number of trials, add up.
- ▶ Might need to wait a large number of trials, so go up a long way:

```
sum(1:100*geometric(1:100, 0.5))
[1] 2

sum(1:100*geometric(1:100, 0.1))
[1] 9.997078
```

- ▶ Second one not quite 10, because there is a tiny probability of taking more than 100 trials.

## Using R's geometric calculator

- ▶ Called `pgeom`:

```
c.geometric(5,0.5)
```

```
[1] 0.96875
```

```
c.geometric(20,0.1)
```

```
[1] 0.8784233
```

```
pgeom(5,0.5)
```

```
[1] 0.984375
```

```
pgeom(20,0.1)
```

```
[1] 0.890581
```

- ▶ Oh. Not the same.
- ▶ Look in help for `pgeom`: this is *other* version of geometric, where you count *how many failures* happened before 1st success (#trials minus 1). So we need (compare `c.geometric` on *left* above):

```
pgeom(4,0.5)
```

```
[1] 0.96875
```

```
pgeom(19,0.1)
```

```
[1] 0.8784233
```

## Another way of writing cumulative geometric

- ▶ Suppose we hadn't thought to try a vector for  $x$ . What then?
- ▶ Calculate each probability in turn, add on to a running total, return total at end.
- ▶ Uses a **loop**:

```
c2.geometric=function(x,p)
{
 total=0
 for (i in 1:x)
 {
 prob=geometric(i,p)
 total=total+prob
 }
 total
}
```

## Checking

```
c2.geometric(5,0.5)
[1] 0.96875

c.geometric(5,0.5)
[1] 0.96875

c2.geometric(20,0.1)
[1] 0.8784233

c.geometric(20,0.1)
[1] 0.8784233
```

Same as before.

## Yet another way

- ▶ “for each number of trials from 1 to x, compute the probability of taking exactly that many trials, then add”:

```
probs=map_dbl(1:5,geometric,0.5)
sum(probs)

[1] 0.96875
```

- ▶ thus:

```
c3.geometric=function(x,p) {
 probs=map_dbl(1:x,geometric,p)
 sum(probs)
}
```

- ▶ testing:

```
c3.geometric(5,0.5)
[1] 0.96875
```

```
c3.geometric(20,0.1)
[1] 0.8784233
```

## Section 13

### Dates and times

## Packages for this section

```
library(lubridate)

Loading required package: methods

Attaching package: 'lubridate'

The following object is masked from 'package:base':

date
```

## Dates

- Dates are represented on computers as “days since an origin”, typically Jan 1, 1970, with a negative date being before the origin:

```
dates=c("1970-01-01", "2007-09-04", "1940-04-15")
d=as.Date(dates) ; d
[1] "1970-01-01" "2007-09-04" "1940-04-15"
as.numeric(d)
[1] 0 13760 -10853
```

- This means that we can do arithmetic with dates, eg.

```
d[2]+30
[1] "2007-10-04"
d[2]-d[3]
Time difference of 24613 days
```

## Reading in dates from a file

- ▶ `read_csv` and the others can guess that you have dates, if you format them as year-month-day, like column 1 of this `.csv`:

```
date,status,dunno
2011-08-03,hello,August 3 2011
2011-11-15,still here,November 15 2011
2012-02-01,goodbye,February 1 2012
```

- ▶ Then read them in:

```
ddd=read_csv("mydates.csv")

Parsed with column specification:
cols(
date = col_date(format = ""),
status = col_character(),
dunno = col_character()
)
```

- ▶ `read_csv` guessed that the 1st column is dates, but not 3rd.

## The data as read in

```
ddd

A tibble: 3 x 3
date status dunno
<date> <chr> <chr>
1 2011-08-03 hello August 3 2011
2 2011-11-15 still here November 15 2011
3 2012-02-01 goodbye February 1 2012
```

## Dates in R

- ▶ Preceding shows that dates should be stored as text in format yyyy-mm-dd (ISO standard).
- ▶ To deal with dates in other formats, use package lubridate and convert. For example, dates in US format with month first:

```
usdates=c("05/27/2012", "01/03/2016", "12/31/2015")
mdy(usdates)

[1] "2012-05-27" "2016-01-03" "2015-12-31"
```

- ▶ For UK-format dates with month *second*, one of these dates is legit, but the other two make no sense:

```
dmy(usdates)

Warning: 2 failed to parse.

[1] NA "2016-03-01" NA
```

## That last column in our data frame

- ▶ That is month, day, year, so:

```
d4=ddd %>% mutate(date2=mdy(dunno)) ; d4

A tibble: 3 x 4
date status dunno date2
<date> <chr> <chr> <date>
1 2011-08-03 hello August 3 2011 2011-08-03
2 2011-11-15 still here November 15 2011 2011-11-15
3 2012-02-01 goodbye February 1 2012 2012-02-01
```

- ▶ Column date2 was correctly converted from column dunno.

```
with(d4, all.equal(date, date2))
[1] TRUE
```

- ▶ The two columns of dates are all the same.

## Reading dates in SAS

- ▶ proc import will likewise make guesses about what you have:

```
proc import
 datafile='/home/ken/mydates.csv'
 dbms=csv
 out=dates
 replace;
getnames=yes;

proc print;
```

## What that reads in

| Obs | date       | status     | dunno            |
|-----|------------|------------|------------------|
| 1   | 2011-08-03 | hello      | 03AUG11:00:00:00 |
| 2   | 2011-11-15 | still here | 15NOV11:00:00:00 |
| 3   | 2012-02-01 | goodbye    | 01FEB12:00:00:00 |

- ▶ SAS made a guess at the dates with month names in them: it guessed they were “datetimes”, which explains the mysterious midnight times.
- ▶ Not clear from looking at this whether the column date actually *is* dates, or just text. To check, look in Log tab for the word format. I got:

```
format date yymmdd10. ;
format status $10. ;
format dunno datetime. ;
```

- ▶ This tells you how the values have been displayed: the date is indeed a date with year first, and dunno is indeed a “datetime”.

## Display formatted dates in SAS

- ▶ If you don't like how your dates are displayed, you can change it, eg.:

```
proc print;
 format date mmddyy8.;
```

| Obs | date     | status     | dunno            |
|-----|----------|------------|------------------|
| 1   | 08/03/11 | hello      | 03AUG11:00:00:00 |
| 2   | 11/15/11 | still here | 15NOV11:00:00:00 |
| 3   | 02/01/12 | goodbye    | 01FEB12:00:00:00 |

- ▶ Even though dates were originally in ISO year-month-day format, they can be output in any format (eg. US format here).
- ▶ SAS can input/output dates in many formats; you just have to find name of one you need. See eg.  
<https://v8doc.sas.com/sashelp/lrcon/zenid-63.htm>.

## Constructing dates from year, month and day

- ▶ You might have separate columns containing year, month, day.
- ▶ Strategy (both R and SAS): glue them together into something that can be recognized as date:

```
proc import
 datafile='/home/ken/pieces.txt'
 dbms=dlm
 out=pieces
 replace;
 delimiter=' '|;
 getnames=yes;

data makedates;
 set pieces;
 sasdate=mdy(month,day,year);
```

## The resulting data set

```
proc print;
 format sasdate yymmmdd10.;
```

| Obs | year | month | day | sasdate    |
|-----|------|-------|-----|------------|
| 1   | 1970 | 1     | 1   | 1970-01-01 |
| 2   | 2007 | 9     | 4   | 2007-09-04 |
| 3   | 1940 | 4     | 15  | 1940-04-15 |

The format displays the dates in ISO format. If you omit it:

```
proc print;
```

| Obs | year | month | day | sasdate |
|-----|------|-------|-----|---------|
| 1   | 1970 | 1     | 1   | 3653    |
| 2   | 2007 | 9     | 4   | 17413   |
| 3   | 1940 | 4     | 15  | -7200   |

you get *days since Jan 1, 1960*.

## In R

Starting from this file:

```
year month day
1970 1 1
2007 9 4
1940 4 15
```

```
dates0=read_delim("pieces.txt", " ")
Parsed with column specification:
cols(
year = col_integer(),
month = col_integer(),
day = col_integer()
)
```

```
newdates=dates0 %>%
 unite(dates,day,month,year) %>%
 mutate(d=dmy(dates))
```

## The results

```
newdates
```

```
A tibble: 3 x 2
dates d
<chr> <date>
1 1_1_1970 1970-01-01
2 4_9_2007 2007-09-04
3 15_4_1940 1940-04-15
```

- ▶ `unite` glues things together with an underscore between them (if you don't specify anything else). Syntax: first thing is new column to be created, other columns are what to make it out of.
- ▶ `unite` makes the original variable columns year, month, day disappear.
- ▶ The column `dates` is *text*, while `d` is a real date.

## Extracting information from dates

- ▶ Have seen how to construct dates from ingredients.
- ▶ If we have a date, how to extract those ingredients?
- ▶ lubridate has tools for that, eg. using the dates we just made:

```
thedates=newdates$d
month(thedates)

[1] 1 9 4

day(thedates)

[1] 1 4 15

wday(thedates,label=T)

[1] Thurs Tues Mon
Levels: Sun < Mon < Tues < Wed < Thurs < Fri < Sat
```

# Extracting things in SAS

Recall:

```
proc print data=dates;
```

| Obs | date       | status     | dunno            |
|-----|------------|------------|------------------|
| 1   | 2011-08-03 | hello      | 03AUG11:00:00:00 |
| 2   | 2011-11-15 | still here | 15NOV11:00:00:00 |
| 3   | 2012-02-01 | goodbye    | 01FEB12:00:00:00 |

Extract day, month, year thus:

```
data moredates;
 set dates;
 d=day(date);
 m=month(date);
 y=year(date);
```

## The results

```
proc print;
```

| Obs | date       | status     | dunno            | d  | m  | y    |
|-----|------------|------------|------------------|----|----|------|
| 1   | 2011-08-03 | hello      | 03AUG11:00:00:00 | 3  | 8  | 2011 |
| 2   | 2011-11-15 | still here | 15NOV11:00:00:00 | 15 | 11 | 2011 |
| 3   | 2012-02-01 | goodbye    | 01FEB12:00:00:00 | 1  | 2  | 2012 |

## Dates and times in SAS

- ▶ If it looks like a date-and-time, SAS will read it as one, for example:

```
occasion,when
first,1970-01-01 07:50:01
second,2007-09-04 15:30:00
third,1940-04-15 06:45:10
fourth,2016-02-10 12:26:40
```

- ▶ Since date-times might have spaces, delimit by something other than space!

```
proc import
 datafile='/home/ken/dt.csv'
 dbms=csv
 out=dt
 replace;
 getnames=yes;
```

## Resulting data set

```
proc print;
```

| Obs | occasion | _when            |
|-----|----------|------------------|
| 1   | first    | 01JAN70:07:50:01 |
| 2   | second   | 04SEP07:15:30:00 |
| 3   | third    | 15APR40:06:45:10 |
| 4   | fourth   | 10FEB16:12:26:40 |

## Constructing date-times

- ▶ SAS has function dhms from which we can construct date-times from pieces such as these:

```
year month day hour minute second
1963 9 24 10 0 0
2017 5 17 13 24 30
```

- ▶ which we read in the usual way:

```
proc import
 datafile='/home/ken/manypieces.txt'
 dbms=dlm
 out=many
 replace;
 delimiter=' ';
 getnames=yes;
```

## Creating the date-times

Start from dataset read in from file and then create what you need, throwing away original variables (not needed any more):

```
data dtm;
 set many;
 thedate=mdy(month,day,year);
 thetime=hms(hour,minute,second);
 sasdt=dhms(thedate,0,0,thetime);
 keep thedate thetime sasdt;
```

## The result

```
proc print;
```

| Obs | thedate | thetime | sasdt      |
|-----|---------|---------|------------|
| 1   | 1362    | 36000   | 117712800  |
| 2   | 20956   | 48270   | 1810646670 |

which doesn't account for the new variables being dates/times, or better:

```
proc print;
format thedate yymmd10. thetime time8.
 sasdt datetime.;
```

| Obs | thedate    | thetime  | sasdt            |
|-----|------------|----------|------------------|
| 1   | 1963-09-24 | 10:00:00 | 24SEP63:10:00:00 |
| 2   | 2017-05-17 | 13:24:30 | 17MAY17:13:24:30 |

## Dates and times in R

- ▶ Standard format for times is to put the time after the date, hours, minutes, seconds:

```
dd=c("1970-01-01 07:50:01", "2007-09-04 15:30:00",
"1940-04-15 06:45:10", "2016-02-10 12:26:40") ; dd
[1] "1970-01-01 07:50:01" "2007-09-04 15:30:00" "1940-04-15 06:45:10"
[4] "2016-02-10 12:26:40"
```

- ▶ Then read in using `ymd_hms`:

```
dt=ymd_hms(dd) ; dt
[1] "1970-01-01 07:50:01 UTC" "2007-09-04 15:30:00 UTC"
[3] "1940-04-15 06:45:10 UTC" "2016-02-10 12:26:40 UTC"
```

- ▶ Default timezone is “Universal Coordinated Time”. Change it via `tz=` and the name of a timezone:

```
dt=ymd_hms(dd, tz="America/Toronto") ; dt
[1] "1970-01-01 07:50:01 EST" "2007-09-04 15:30:00 EDT"
[3] "1940-04-15 06:45:10 EST" "2016-02-10 12:26:40 EST"
```

## Extracting time parts

- As you would expect:

```
dt
```

```
[1] "1970-01-01 07:50:01 EST" "2007-09-04 15:30:00 EDT"
[3] "1940-04-15 06:45:10 EST" "2016-02-10 12:26:40 EST"
```

```
hour(dt)
```

```
[1] 7 15 6 12
```

```
minute(dt)
```

```
[1] 50 30 45 26
```

```
second(dt)
```

```
[1] 1 0 10 40
```

```
tz(dt)
```

```
[1] "America/Toronto"
```

- Same times, but different time zone:

```
with_tz(dt, "Australia/Sydney")
```

```
[1] "1970-01-01 22:50:01 AEST" "2007-09-05 05:30:00 AEST"
[3] "1940-04-15 21:45:10 AEST" "2016-02-11 04:26:40 AEDT"
```

## Subtracting date-times

- ▶ We may need to calculate the time *between* two events. For example, these are the dates and times that some patients were admitted to and discharged from a hospital:

admit,discharge

1981-12-10 22:00:00,1982-01-03 14:00:00

2014-03-07 14:00:00,2014-03-08 09:30:00

2016-08-31 21:00:00,2016-09-02 17:00:00

- ▶ These ought to get read in and converted to date-times:

```
stays=read_csv("hospital.csv")

Parsed with column specification:
cols(
admit = col_datetime(format = ""),
discharge = col_datetime(format = "")
)
```

- ▶ and so it proves.

## Subtracting the date-times

- In the obvious way, this gets us an answer:

```
stays %>% mutate(stay=discharge-admit)

A tibble: 3 x 3
admit discharge stay
<dttm> <dttm> <time>
1 1981-12-10 22:00:00 1982-01-03 14:00:00 568.0 hours
2 2014-03-07 14:00:00 2014-03-08 09:30:00 19.5 hours
3 2016-08-31 21:00:00 2016-09-02 17:00:00 44.0 hours
```

- The number of hours is hard to interpret. The fractional number of days would be better:

```
stays %>% mutate(stay=(discharge-admit)/ddays(1))

A tibble: 3 x 3
admit discharge stay
<dttm> <dttm> <dbl>
1 1981-12-10 22:00:00 1982-01-03 14:00:00 23.666667
2 2014-03-07 14:00:00 2014-03-08 09:30:00 0.812500
3 2016-08-31 21:00:00 2016-09-02 17:00:00 1.833333
```

## Comments

- ▶ Date-times are stored internally as seconds-since-something, so that subtracting two of them will give, internally, a number of seconds.
- ▶ Just subtracting the date-times is displayed as a time (in units that R chooses for us).
- ▶ Functions `ddays(1)`, `dminutes(1)` etc. will give number of seconds in a day or a minute, thus dividing by them will give (fractional) days, minutes etc.
- ▶ This idea useful for calculating time from a start point until an event happens (in this case, a patient being discharged from hospital).

## In SAS

- ▶ In SAS, date-times are *seconds since midnight Jan 1, 1960*.
- ▶ Thus, subtracting date-times gives a number of seconds, which we might then have to translate into something useful.

```
proc import
 datafile='/home/ken/hospital.csv'
 dbms=csv
 out=stays
 replace;
 getnames=yes;
```

- ▶ In a new dataset, calculate the lengths of stay, converting seconds to days:

```
data hospitalstay;
 set stays;
 stay=(discharge-admit)/60/60/24;
```

```
proc print;
```

## The results

- ▶ Output below. The stay should be displayed as a decimal number, so no special treatment required. Length of stay agrees with R:

| Obs | admit            | discharge        | stay    |
|-----|------------------|------------------|---------|
| 1   | 10DEC81:22:00:00 | 03JAN82:14:00:00 | 23.6667 |
| 2   | 07MAR14:14:00:00 | 08MAR14:09:30:00 | 0.8125  |
| 3   | 31AUG16:21:00:00 | 02SEP16:17:00:00 | 1.8333  |

## Section 14

Miscellaneous stuff in R and SAS

## SAS: More than one observation per line of data file

- ▶ Suppose you have a data file like this:

```
3 4 5 6 7 7
8 9 3 4 8 6
```

but the data are *all* values of one variable x (so there are 12 values altogether).

- ▶ How to get *one* column called x?
- ▶ Strategy: read values in the usual way, then process.
- ▶ Here there are no variable names, so:

```
proc import
 datafile='/home/ken/many.txt'
 dbms=dlm out=many replace;
 delimiter=' '|;
 getnames=no;
```

- ▶ Note last line, not the usual.

## So far

```
proc print;
```

|   | V | V | V | V | V | V | V |
|---|---|---|---|---|---|---|---|
| 0 | A | A | A | A | A | R | A |
| b | R | R | R | R | R | R | R |
| s | 1 | 2 | 3 | 4 | 5 | 6 | 6 |
| 1 |   | 3 | 4 | 5 | 6 | 7 | 7 |
| 2 |   | 8 | 9 | 3 | 4 | 8 | 6 |

We have six variables with names like VAR2, each “variable” having two values (two lines of data file).

## Solution for this

Solution very like the SAS version of gather, using an array:

```
data one;
 set many;
 array x_array VAR1-VAR6;
 do i=1 to 6;
 x=x_array[i];
 output;
 end;
 keep x;
```

# Did it work?

```
proc print;
```

| Obs | x |
|-----|---|
| 1   | 3 |
| 2   | 4 |
| 3   | 5 |
| 4   | 6 |
| 5   | 7 |
| 6   | 7 |
| 7   | 8 |
| 8   | 9 |
| 9   | 3 |
| 10  | 4 |
| 11  | 8 |
| 12  | 6 |

## Same data file as values of x and y

- ▶ Recall:

```
3 4 5 6 7 7
8 9 3 4 8 6
```

- ▶ Suppose now a value of x and a value of y, then another x and another y, and so on, so 3 is x, 4 is y, 5 is x, 6 is y and so on.
- ▶ Read in as before using proc import to get data set with VAR1 through VAR6, then loop from 1 to 3 (3 x-y pairs), pulling out the right things.

## Making x and y

- ▶ This code, adapted from previous:

```
data two;
 set many;
 array xy_array VAR1-VAR6;
 do i=1 to 3;
 x=xy_array[2*i-1];
 y=xy_array[2*i];
 output;
 end;
 keep x y;
```

- ▶ Tricky part: when  $i = 1$ , want items 1 and 2 from the array; when  $i = 2$ , want items 3 and 4, etc.
- ▶ Twice the value of  $i$  will give the second value we want (the one for y), so one less than that will give the value we want for x.

## Did it work?

We seem to have been successful. You can check that the right values got assigned to x and y in the right order.

```
proc print;
```

| Obs | x | y |
|-----|---|---|
| 1   | 3 | 4 |
| 2   | 5 | 6 |
| 3   | 7 | 7 |
| 4   | 8 | 9 |
| 5   | 3 | 4 |
| 6   | 8 | 6 |

## Permanent data sets

- ▶ Can we read in data set *once* and not every time?
- ▶ Yes, use *this mechanism* when creating, for example pigs data:

```
libname mydata V9 '/home/ken';
proc import
 datafile='/home/ken/pigs1.txt'
 dbms=dlm
 out=mydata.pigs1
 replace;
 delimiter=' '|;
 getnames=yes;
```

- ▶ First, define a `libname` that tells SAS which folder this dataset will go in.
- ▶ Then, on `out=`, use a two-part name: the `libname`, then dataset name.

## Comments

- ▶ In folder defined by libname, will be a file called `pigs1.sas7bdat` (!) on SAS Studio. In my case, in my main SAS Studio folder.
- ▶ Can use subfolders, using / forward slash syntax, in libname.
- ▶ Whenever you need to use it, add `data='home/username/pigs1'` to a proc line (replacing `username` with your username, and replacing `pigs1` with your data set name).
- ▶ Closing SAS breaks connection with temporary (ie. *non-permanent*) data sets. To get those back, need to run `proc import` lines again.

## proc means without reading in data

- ▶ Imagine we closed down SAS Studio and opened it up again. Then:

```
proc means data='/home/ken/pigs1';
```

- ▶ with output

The MEANS Procedure

| Variable | N | Mean       | Std Dev   | Minimum    | Maximum    |
|----------|---|------------|-----------|------------|------------|
| pig      | 5 | 3.0000000  | 1.5811388 | 1.0000000  | 5.0000000  |
| feed1    | 5 | 60.6200000 | 3.0646370 | 57.0000000 | 65.0000000 |
| feed2    | 5 | 69.3000000 | 2.9266021 | 66.3000000 | 74.0000000 |
| feed3    | 5 | 94.1000000 | 3.6131704 | 90.2000000 | 99.1000000 |
| feed4    | 5 | 86.2400000 | 2.8962044 | 83.1000000 | 90.3000000 |

## Saving permanent data sets other ways

- ▶ If you run a proc that has an `out=` option, create permanent data set same way as from `proc import`: define a `libname` and put a two-part name on `out=` with the `libname` first.
- ▶ Can also create a new data set, using `data step`, and make *that* permanent. For example, suppose we take data set `two` from before (containing variables `x` and `y`) and add a variable `z` to it, saving in permanent data set `three`.
- ▶ Same idea: define `libname`, and use two-part data set name:

```
libname mydata V9 '/home/ken';
data mydata.three; /* permanent data set to save in */
 set two; /* this has variables x and y in it */
 z=x+y;
```

## The new permanent data set

- ▶ Imagine I closed down SAS Studio and opened it up again:

```
proc print data='/home/ken/three';
```

| Obs | x | y | z  |
|-----|---|---|----|
| 1   | 3 | 4 | 7  |
| 2   | 5 | 6 | 11 |
| 3   | 7 | 7 | 14 |
| 4   | 8 | 9 | 17 |
| 5   | 3 | 4 | 7  |
| 6   | 8 | 6 | 14 |

## Why permanent data sets?

- ▶ It is a lot of work (for us) to read in data sets from file every time. I can never remember the syntax for `proc import` (I usually copy an old one).
- ▶ It can take a lot of effort to get data in the right format for analysis. Rather than do that every time, we can save a permanent data set once the dataset is in the right shape.
- ▶ For big data, we don't want to repeat the effort of reading and processing more than once. (This can take a *long* time.) Better to create one permanent dataset and use it for each of our analyses.

## How does SAS know which data set to use?

Two rules:

1. Any proc can have data= on it. Tells SAS to use that data set. Can be
  - ▶ unquoted dataset name (created by proc import or by processing a dataset read in that way)
  - ▶ quoted data set name (permanent one on disk created as above)
2. Without data=, *most recently created data set*. Typically data set created by proc import or data step. Also, data set created by out= counts.

Does permanent data set count as “most recently created”? No, or at least not always. If unsure, use data=.

## Embellishments to plots

- ▶ Histogram with kernel density curve
- ▶ Smooth trend on scatterplot
- ▶ Plotting several series of data
- ▶ Labelling points on plots
- ▶ both SAS and R (eventually).

## Use Australian athletes data

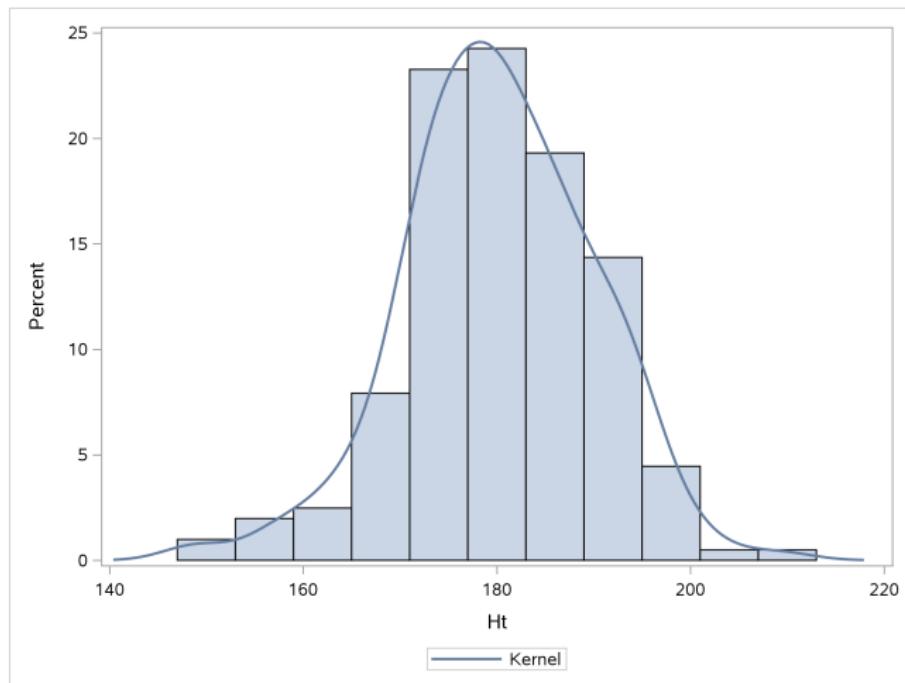
```
proc import
 datafile='/home/ken/ais.txt'
 dbms=dlm
 out=sports
 replace;
 delimiter='09'x;
 getnames=yes;
```

## Kernel density curve on histogram

- ▶ A kernel density curve smooths out a histogram and gives sense of shape of distribution.
- ▶ Athlete heights:

```
proc sgplot;
 histogram Ht;
 density Ht / type=kernel;
```

# Histogram of heights with kernel density

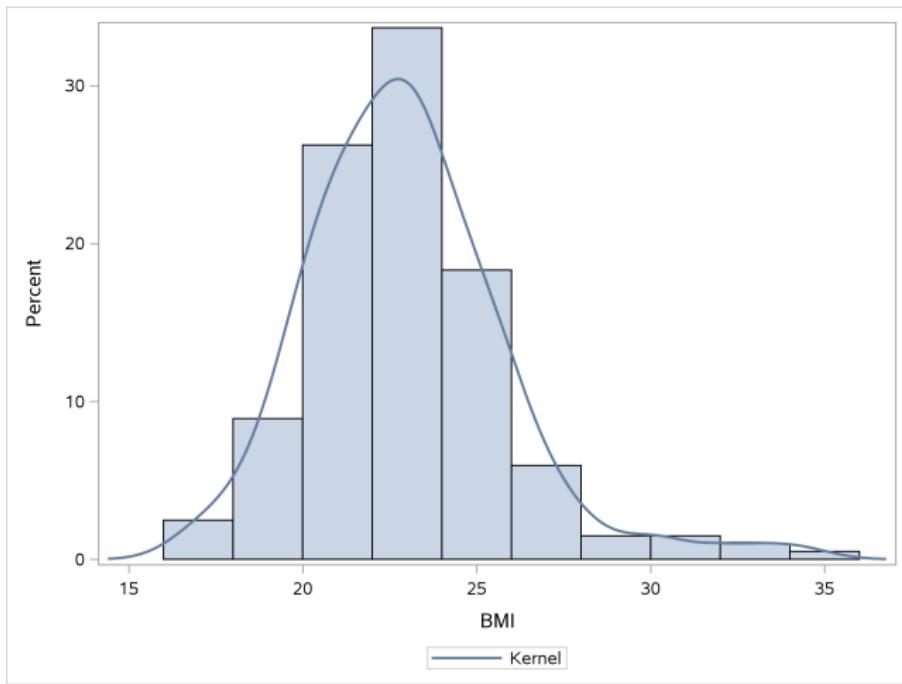


More or less symmetric.

## Kernel density for BMI

```
proc sgplot;
 histogram BMI;
 density BMI / type=kernel;
```

# Histogram with kernel density



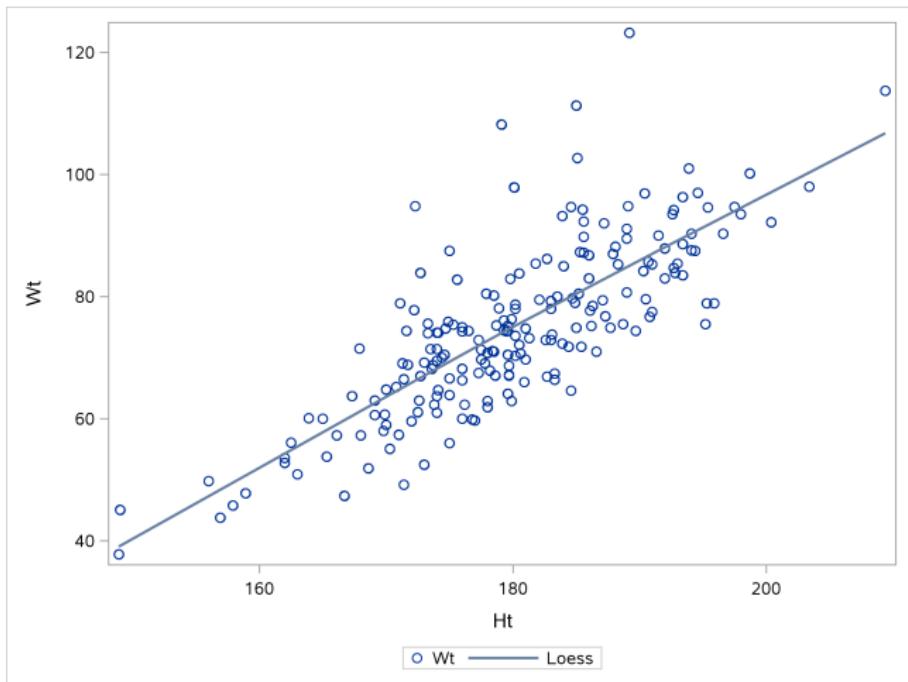
Rather more clearly skewed right.

## Loess curve

- ▶ Smooth curve through scatterplot called *Loess curve* in SAS: Code like this:

```
proc sgplot;
 scatter x=Ht y=Wt;
 loess x=Ht y=Wt;
```

## Loess curve on plot



Loess curve says this is as straight as you could wish for.

## Loess curve for windmill data

- ▶ Read into SAS thus:

```
proc import
 datafile='/home/ken/windmill.csv'
 dbms=csv
 out=windmill
 replace;
getnames=yes;
```

```
proc means;
```

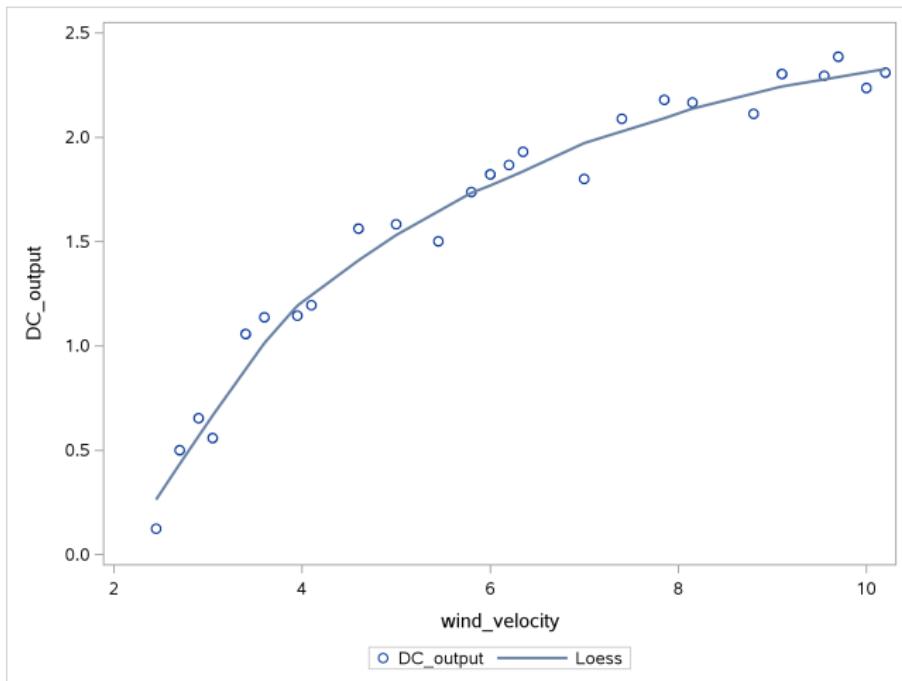
The MEANS Procedure

| Variable      | N  | Mean      | Std Dev   | Minimum   | Maximum    |
|---------------|----|-----------|-----------|-----------|------------|
| <hr/>         |    |           |           |           |            |
| wind_velocity | 25 | 6.1320000 | 2.5294466 | 2.4500000 | 10.2000000 |
| DC_output     | 25 | 1.6096000 | 0.6522777 | 0.1230000 | 2.3860000  |

## To make the scatterplot with loess curve

```
proc sgplot;
 scatter x=wind_velocity y=DC_output;
 loess x=wind_velocity y=DC_output;
```

## The plot with curve



This time, relationship is definitely curved.

## Multiple series on one plot: the oranges data

- ▶ Data file like this (circumferences of 5 trees each at 7 times):

| age  | A   | B   | C   | D   | E   |
|------|-----|-----|-----|-----|-----|
| 118  | 30  | 30  | 30  | 33  | 32  |
| 484  | 51  | 58  | 49  | 69  | 62  |
| 664  | 75  | 87  | 81  | 111 | 112 |
| 1004 | 108 | 115 | 125 | 156 | 167 |
| 1231 | 115 | 120 | 142 | 172 | 179 |
| 1372 | 139 | 142 | 174 | 203 | 209 |
| 1582 | 140 | 145 | 177 | 203 | 214 |

- ▶ Columns don't line up because the delimiter is "exactly one space", and some of the values are longer than others.

## Reading the data

```
proc import
 datafile='/home/ken/oranges.txt'
 dbms=dlm
 out=trees
 replace;
 delimiter=' ' ;
 getnames=yes;
```

# Did it work?

```
proc print;
```

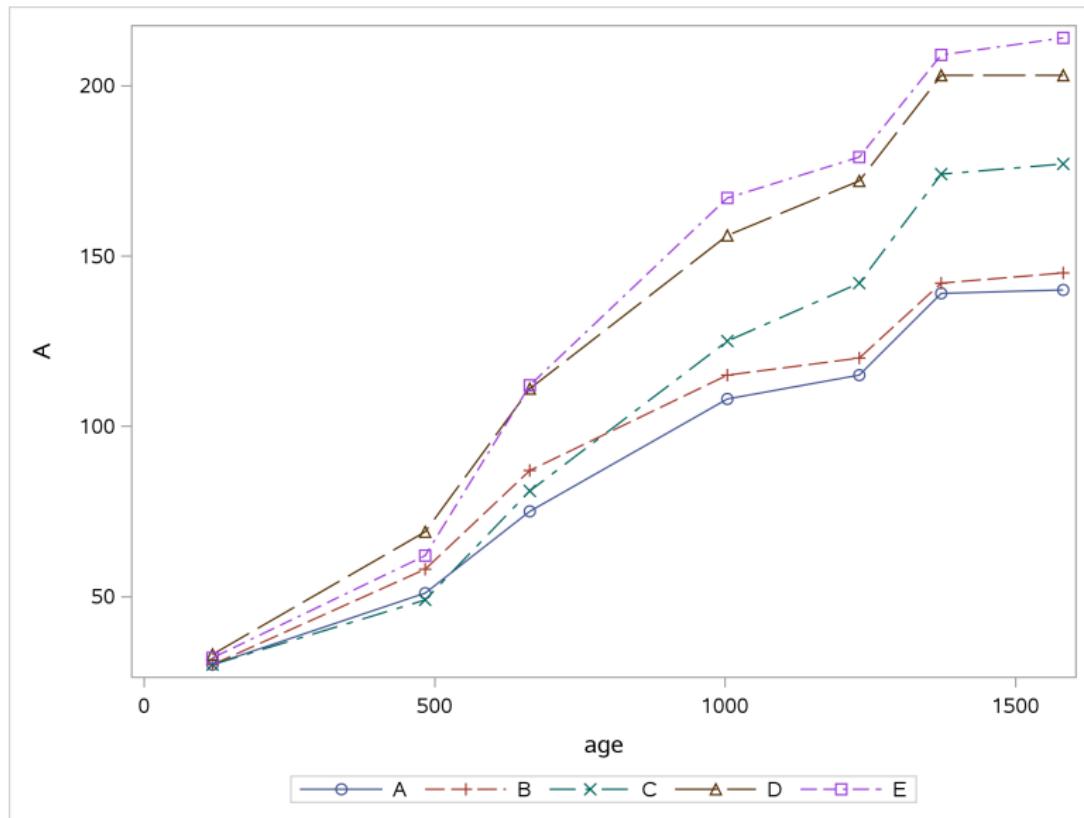
|   | a    | A   | B   | C   | D   | E   |
|---|------|-----|-----|-----|-----|-----|
| 0 | a    |     |     |     |     |     |
| b | g    |     |     |     |     |     |
| s | e    |     |     |     |     |     |
| 1 | 118  | 30  | 30  | 30  | 33  | 32  |
| 2 | 484  | 51  | 58  | 49  | 69  | 62  |
| 3 | 664  | 75  | 87  | 81  | 111 | 112 |
| 4 | 1004 | 108 | 115 | 125 | 156 | 167 |
| 5 | 1231 | 115 | 120 | 142 | 172 | 179 |
| 6 | 1372 | 139 | 142 | 174 | 203 | 209 |
| 7 | 1582 | 140 | 145 | 177 | 203 | 214 |

## Multiple series

- ▶ Growth curve for *each* tree, joined by lines.
- ▶ `series` joins points by lines.
- ▶ `markers` displays actual data points too.
- ▶ Do each series one at a time.

```
proc sgplot;
 series x=age y=a / markers;
 series x=age y=b / markers;
 series x=age y=c / markers;
 series x=age y=d / markers;
 series x=age y=e / markers;
```

# The growth curves



## Labelling points on a plot

- ▶ Often, a data set comes with an identifier variable.
- ▶ We would like to label each point on a plot with its identifier, to see which individual is which.
- ▶ Commonly (but not only) done on scatterplot.

## Example: the cars data

- ▶ 38 cars. For each:
  - ▶ Name of car (identifier)
  - ▶ Gas mileage (miles per US gallon)
  - ▶ Weight (US tons)
  - ▶ Number of cylinders in engine
  - ▶ Horsepower of engine
  - ▶ Country of origin

## Reading in

.csv file, so:

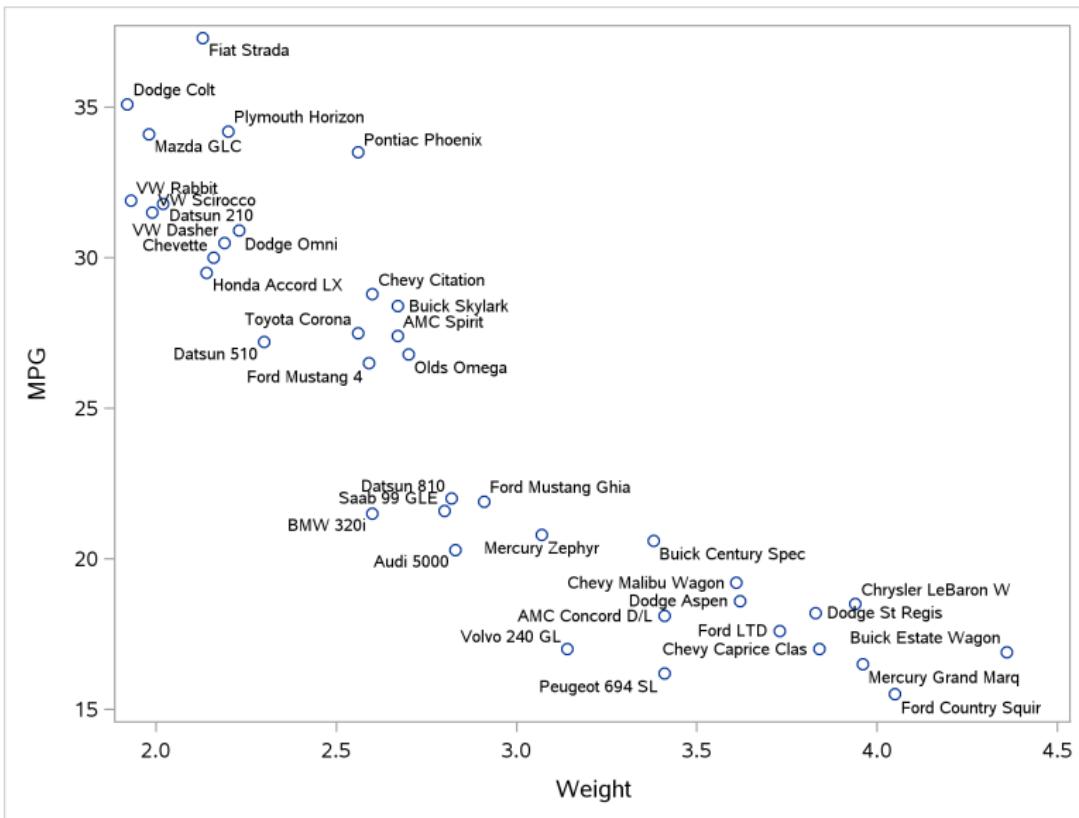
```
proc import
 datafile='/home/ken/cars.csv'
 dbms=csv
 out=cars
 replace;
 getnames=yes;
```

## Adding labels to scatterplot

- ▶ Expect heavier car to have worse (lower) gas mileage, so make scatterplot of gas mileage ( $y$ ) against weight ( $x$ ).
- ▶ Want to see which car is which, so label points.
- ▶ The magic word is `datalabel`:

```
proc sgplot;
 scatter y=mpg x=weight / datalabel=car;
```

# The plot



## Comments

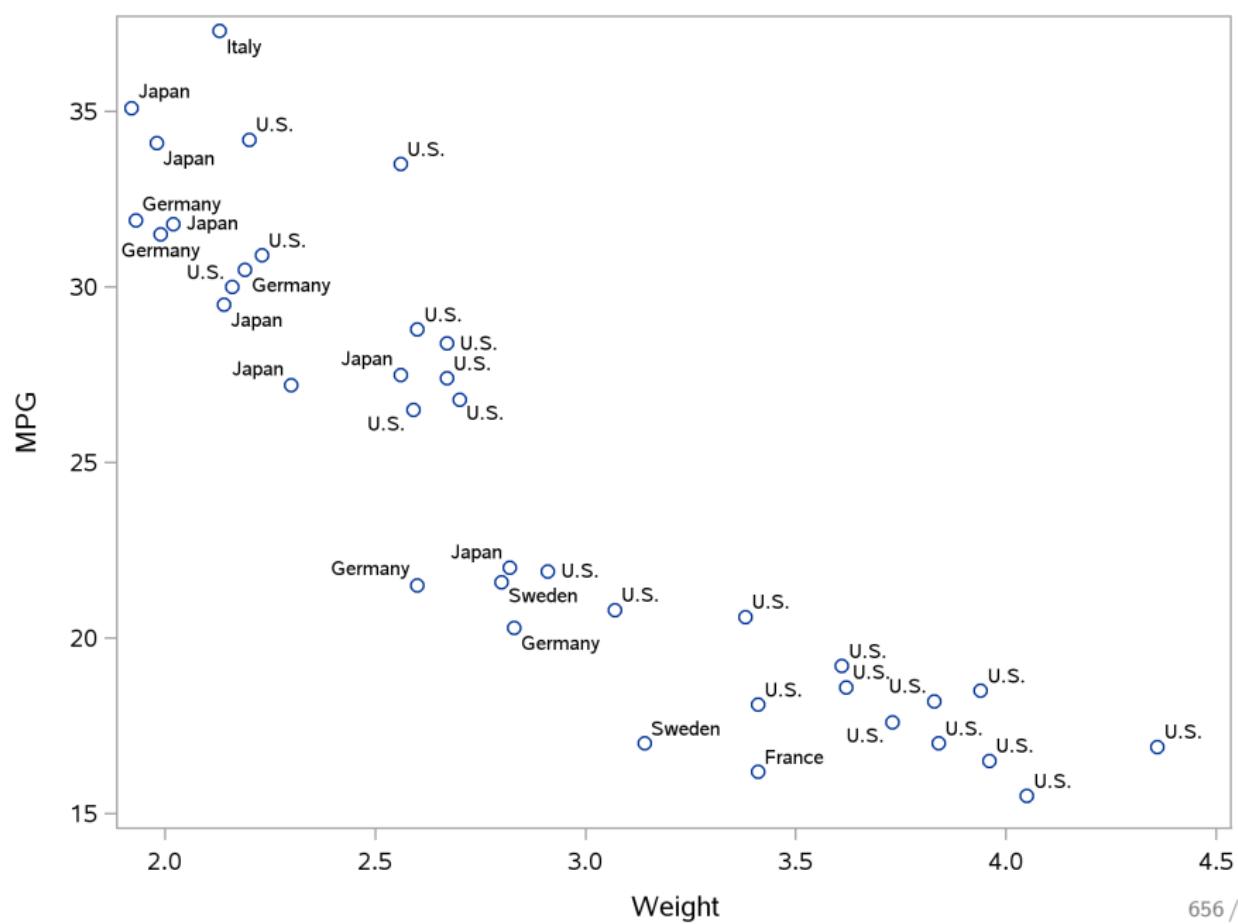
- ▶ Each car labelled with its name, either left, right, above or below, whichever makes it clearest. (Some intelligence applied to placement.)
- ▶ Cars top left are “nimble”: light in weight, good gas mileage.
- ▶ Cars bottom right are “boats”: heavy, with terrible gas mileage.

## Labelling by country

Same idea:

```
proc sgplot;
 scatter x=weight y=mpg / datalabel=country;
```

# Labelled by country



## Labelling only some of the observations

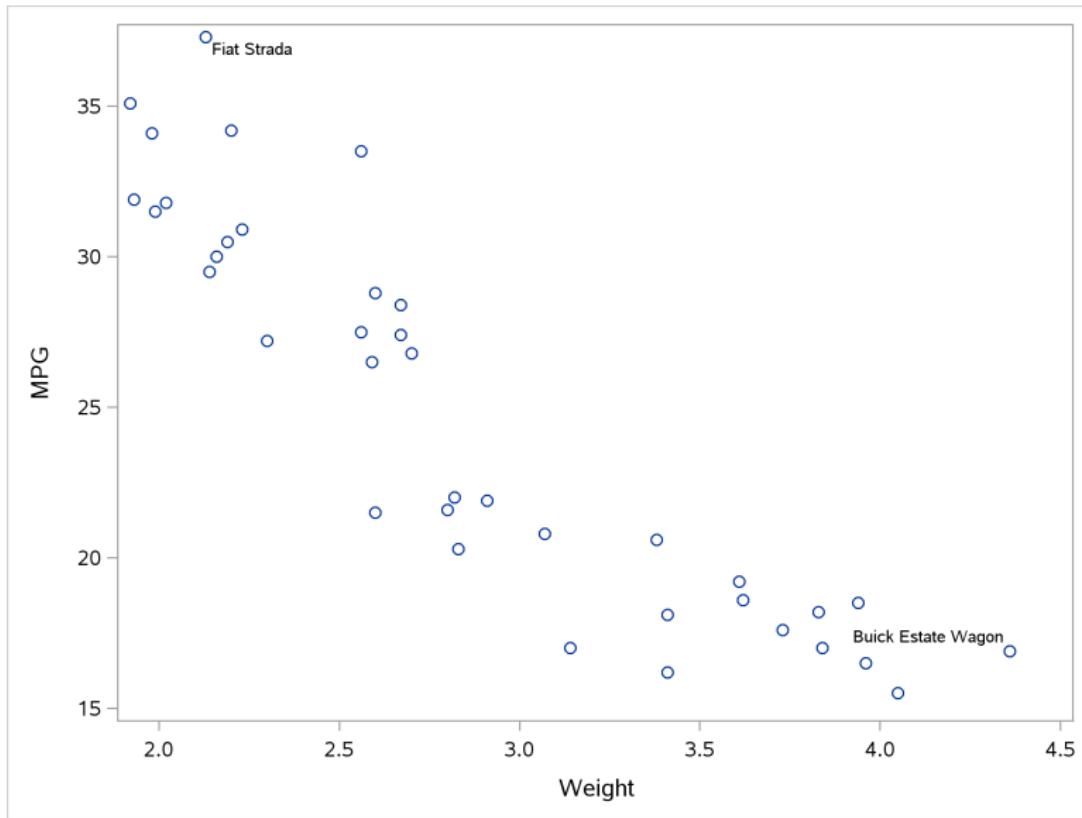
- ▶ Create a new data set with all the old variables plus a new one that contains the text to plot.
- ▶ For example, label most fuel-efficient car (#4) and heaviest car (#9).
- ▶ “Observation number” given by SAS special variable `_n_`.
- ▶ Note the syntax: “if then do” followed by “end”.

```
data cars2;
 set cars;
 if (_n_=4 or _n_=9) then do;
 newtext=car;
 end;
```

- ▶ For any cars not selected, `newtext` will be blank. Then, using the new data set that we just created:

```
proc sgplot;
 scatter x=weight y=mpg / datalabel=newtext;
```

# The plot

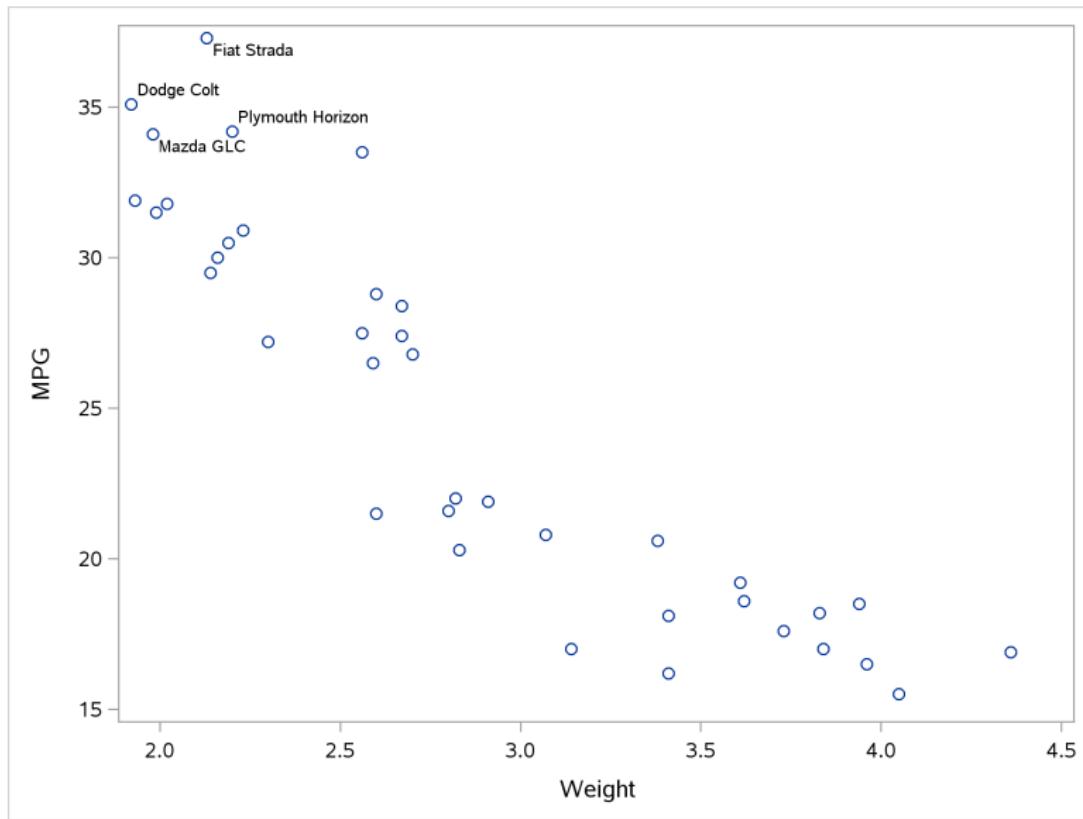


Or label cars with mpg greater than 34

```
data cars3;
 set cars;
 if mpg>34 then do;
 newtext=car;
 end;

proc sgplot;
 scatter x=weight y=mpg / datalabel=newtext;
```

# High-mpg cars



## More R stuff

- ▶ R has a thousand tiny parts, all working together, but to use them, need to know their *names*.
- ▶ Sometimes you *do* know the name, but you forget how it works. Then (at Console) type eg. `?median` or `help(median)`. Help appears in R Studio bottom right.

## Structure of help file

All R's help files laid out the same way:

- ▶ **Purpose:** what the function does
- ▶ **Usage:** how you make it go
- ▶ **Arguments:** what you need to feed in. Arguments with a `=` have *default* values. If the default is OK (it often is), you don't need to specify it.
- ▶ **Details:** more information about how the function works.
- ▶ **Value:** what comes back from the function.
- ▶ **References** to the literature, so that you can find out exactly how everything was calculated.
- ▶ **Examples.** Run these using eg. `example(median)`.

## If you don't know the name

- ▶ Then you have to find it out!
- ▶ If you know what it might be, `apropos(name)`:

```
apropos("read_")
```

```
[1] "read_csv" "read_csv2" "read_csv2_ch"
[4] "read_csv_chunked" "read_delim" "read_delim_ci"
[7] "read_excel" "read_file" "read_file_ra"
[10] "read_fwf" "read_lines" "read_lines_ci"
[13] "read_lines_raw" "read_log" "read_rds"
[16] "read_table" "read_table2" "read_tsv"
[19] "read_tsv_chunked" "read_xls" "read_xlsx"
[22] "spread_"
```

and then you investigate more via `help()`.

- ▶ Google-searching, eg: r ggplot add horizontal line. Often turns up questions on [stackexchange.com](https://stackexchange.com), which might be adapted to your needs.

# That Google search

About 14,900 results (0.59 seconds)

## geom\_hline, ggplot2 0.9.3.1

[docs.ggplot2.org/0.9.3.1/geom\\_hline.html](http://docs.ggplot2.org/0.9.3.1/geom_hline.html) ▾

Horizontal line. ... p <- ggplot(mtcars, aes(x = wt, y=mpg)) + geom\_point() p + ... To display different lines in different facets, you need to # create a data frame. p ...

## geom\_vline, ggplot2 0.9.3.1

[docs.ggplot2.org/0.9.3.1/geom\\_vline.html](http://docs.ggplot2.org/0.9.3.1/geom_vline.html) ▾

This geom allows you to annotate the plot with vertical lines (see geom\_hline and ... To display different lines in different facets, you need to # create a data frame. ... geom\_hline for horizontal lines, geom\_abline for lines defined by a slope and ...

## Lines (ggplot2) - Cookbook for R

[cookbook-r.com/Graphs/Lines\\_\(ggplot2\).html](http://cookbook-r.com/Graphs/Lines_(ggplot2).html) ▾

Basic lines; Automatically drawing lines for the mean; Using lines with facets ... bp # Add a horizontal line bp + geom\_hline(aes(yintercept=12)) # Make the line ...

## r - Add a horizontal line to plot and legend in ggplot2 - Stack Overflow

[stackoverflow.com/questions/.../add-a-horizontal-line-to-plot-and-legend-in-ggplot2](http://stackoverflow.com/questions/.../add-a-horizontal-line-to-plot-and-legend-in-ggplot2) ▾

Nov 6, 2012 - (1) Try this: cutoff <- data.frame( x = c(-Inf, Inf), y = 50, cutoff = factor(50) ) ggplot(the.data, aes( year, value )) + geom\_point(aes( colour = source )) + ...

## r - Add horizontal line to ggplot - Stack Overflow

[stackoverflow.com/questions/10748180/add-horizontal-line-to-ggplot](http://stackoverflow.com/questions/10748180/add-horizontal-line-to-ggplot) ▾

May 25, 2012 - A concrete solution to your question above could be to simply move your last line of code, p + geom\_hline(yintercept=400) , up to be part of the ...

- ▶ Looks like `geom_hline()`.
- ▶ Look up in help as `?ggplot2::geom_hline`.

# I never heard of `read_fwf`!

- ▶ Often `apropos` turns up things you never heard of.
- ▶ But now you have the name, you can look up the help:

`read_fwf {readr}`

R Documentation

Read a fixed width file into a tibble

## Description

A fixed width file can be a very compact representation of numeric data. It's also very fast to parse, because every field is in the same place in every line. Unfortunately, it's painful to parse because you need to describe the length of every field. `Readr` aims to make it as easy as possible by providing a number of different ways to describe the field structure.

## Usage

```
read_fwf(file, col_positions, col_types = NULL, locale = default_locale(),
 na = c("", "NA"), comment = "", skip = 0, n_max = Inf,
 guess_max = min(n_max, 1000), progress = show_progress())

fwf_empty(file, skip = 0, col_names = NULL, comment = "", n = 100L)

fwf_widths(widths, col_names = NULL)

fwf_positions(start, end = NULL, col_names = NULL)

fwf_cols(...)
```

---

## Arguments

- ▶ What does a fixed-width file look like?

## The original oranges data

|      |     |     |     |     |     |
|------|-----|-----|-----|-----|-----|
| 118  | 30  | 30  | 30  | 33  | 32  |
| 484  | 51  | 58  | 49  | 69  | 62  |
| 664  | 75  | 87  | 81  | 111 | 112 |
| 1004 | 108 | 115 | 125 | 156 | 167 |
| 1231 | 115 | 120 | 142 | 172 | 179 |
| 1372 | 139 | 142 | 174 | 203 | 209 |
| 1582 | 140 | 145 | 177 | 203 | 214 |

- ▶ The columns *line up*, so they are easy to read.
- ▶ That means that sometimes you have more than one space between them, and `read_delim` won't work.
- ▶ But each column is a *fixed number of characters* wide.
- ▶ No variable names, so have to supply them also when reading in.

## Guessing columns based on spaces

```
fname = "oranges-orig.txt"
oranges2 = read_fwf(fname,
 fwf_empty(fname, col_names = c("age", "A", "B", "C", "D", "E")))

Parsed with column specification:
cols(
age = col_integer(),
A = col_integer(),
B = col_integer(),
C = col_integer(),
D = col_integer(),
E = col_integer()
)
```

Note that we have to supply file name *twice*, so define it into variable to save typing.

## The data

```
oranges2
```

```
A tibble: 7 x 6
age A B C D E
<int> <int> <int> <int> <int> <int>
1 118 30 30 30 33 32
2 484 51 58 49 69 62
3 664 75 87 81 111 112
4 1004 108 115 125 156 167
5 1231 115 120 142 172 179
6 1372 139 142 174 203 209
7 1582 140 145 177 203 214
```

That worked.

## Reading columns based on widths

Here, there are 6 columns each 4 characters wide (including preceding space(s)), so:

```
oranges3=read_fwf(fname,fwf_widths(c(4,4,4,4,4,4),
 c("age","A","B","C","D","E")))

Parsed with column specification:
cols(
age = col_integer(),
A = col_integer(),
B = col_integer(),
C = col_integer(),
D = col_integer(),
E = col_integer()
)
```

## The data

```
oranges3
```

```
A tibble: 7 x 6
age A B C D E
<int> <int> <int> <int> <int> <int>
1 118 30 30 30 33 32
2 484 51 58 49 69 62
3 664 75 87 81 111 112
4 1004 108 115 125 156 167
5 1231 115 120 142 172 179
6 1372 139 142 174 203 209
7 1582 140 145 177 203 214
```

That worked also.

## No delimiters

The advantage to reading by width is that *you don't need any delimiters.*

For example, this file `rats7.txt`:

```
rat1T 8
rat2T11
rat3C 7
rat4C 4
rat5T12
```

has a rat identifier in the first 4 columns, a group T or C in the next 1 column, and then a response variable y in the next 2.

Delimiterless files used to be very common, because they take up very little disk space:

```
ls -l rats7.txt
```

```
-rw-rw-r-- 1 ken ken 40 May 23 11:59 rats7.txt
```

40 bytes:  $7 + 1 = 8$  for each line, times 5 lines.

## Reading in delimiterless data

- ▶ Read in with `read_fwf` and vector of widths:

```
rat7=read_fwf("rats7.txt",fwf_widths(c(4,1,2),
 c("id","group","y")))

Parsed with column specification:
cols(
id = col_character(),
group = col_character(),
y = col_integer()
)
```

## The “rat7” data

```
rat7

A tibble: 5 x 3
id group y
<chr> <chr> <int>
1 rat1 T 8
2 rat2 T 11
3 rat3 C 7
4 rat4 C 4
5 rat5 T 12
```

- ▶ That worked too. Note that `read_fwf` determined that `y` was a number and the other things were text.
- ▶ You need to have a separate document telling you how many characters each column is.

## Plotting series with R

- ▶ The oranges data:

```
oranges2
```

```
A tibble: 7 x 6
age A B C D E
<int> <int> <int> <int> <int> <int>
1 118 30 30 30 33 32
2 484 51 58 49 69 62
3 664 75 87 81 111 112
4 1004 108 115 125 156 167
5 1231 115 120 142 172 179
6 1372 139 142 174 203 209
7 1582 140 145 177 203 214
```

- ▶ Want to plot orange circumferences against age for each orange tree.
- ▶ Recall ggplot wants one column of x values and one column of y values, which we do not have.

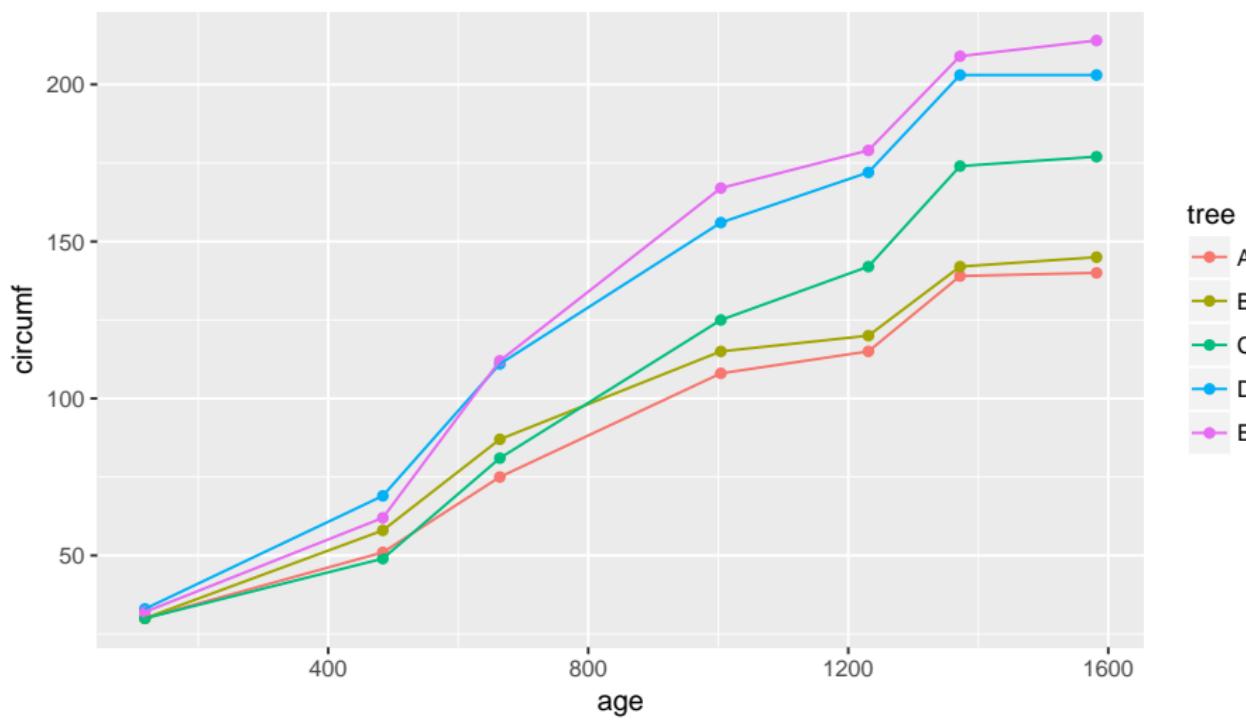
## Making right format and plot

Use `gather` to create columns we need, and then plot:

```
g=oranges2 %>% gather(tree,circumf,A:E) %>%
 ggplot(aes(x=age,y=circumf,colour=tree))+
 geom_point() + geom_line()
```

# The plot

g



## Data for other plots

- ▶ Re-use data on Australian athletes and cars to get corresponding plots to SAS's.

```
cars=read_csv("cars.csv")

Parsed with column specification:
cols(
Car = col_character(),
MPG = col_double(),
Weight = col_double(),
Cylinders = col_integer(),
Horsepower = col_integer(),
Country = col_character()
)
```

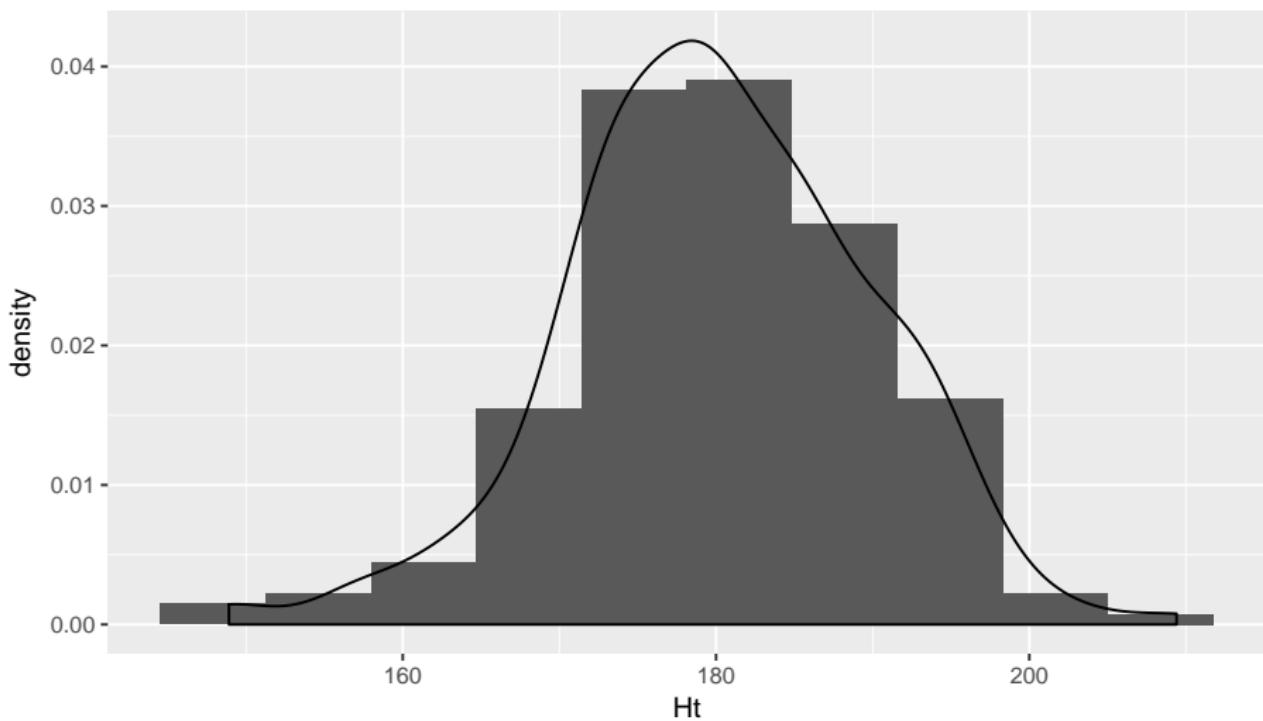
## Histogram with kernel density

- ▶ Athletes height and BMI, height first.
- ▶ Two things:
  - ▶ use density scale on histogram (0–1 or fraction of whole, rather than count)
  - ▶ add kernel density.

```
g=ggplot(athletes,aes(x=Ht))+
 geom_histogram(aes(y=..density..),bins=10)+
 geom_density()
```

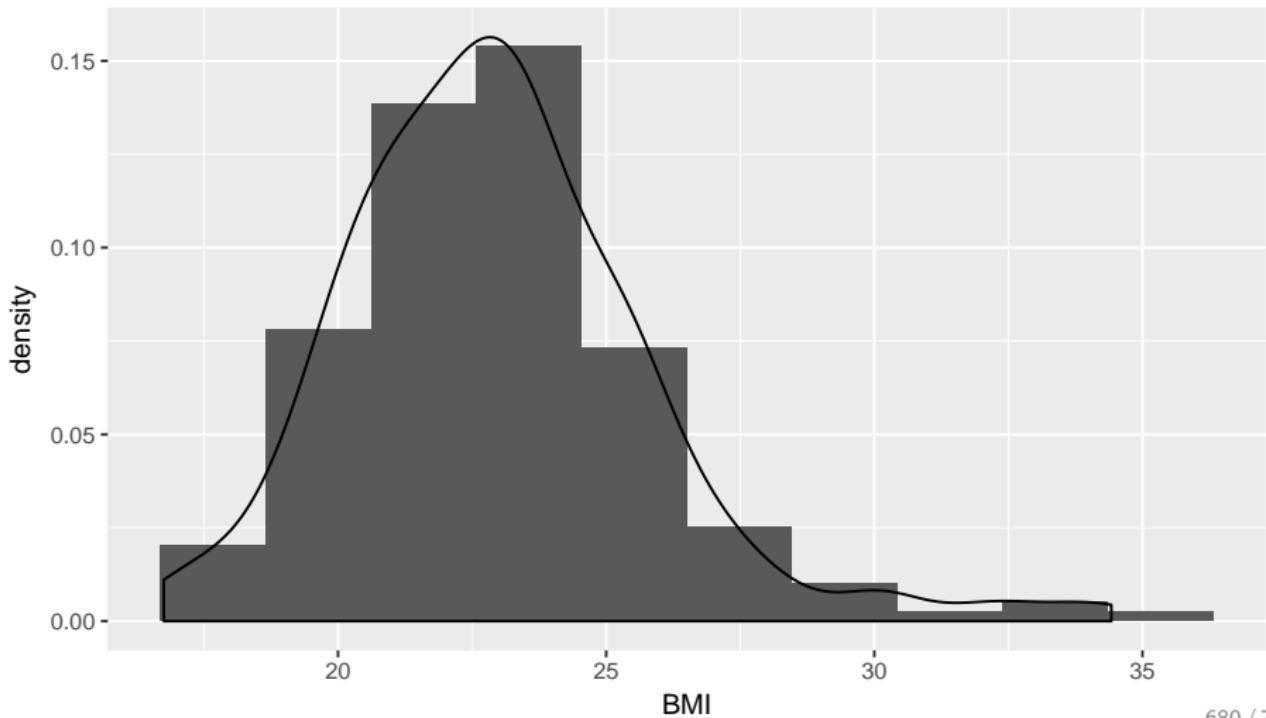
# The histogram

g



## Same idea for BMI

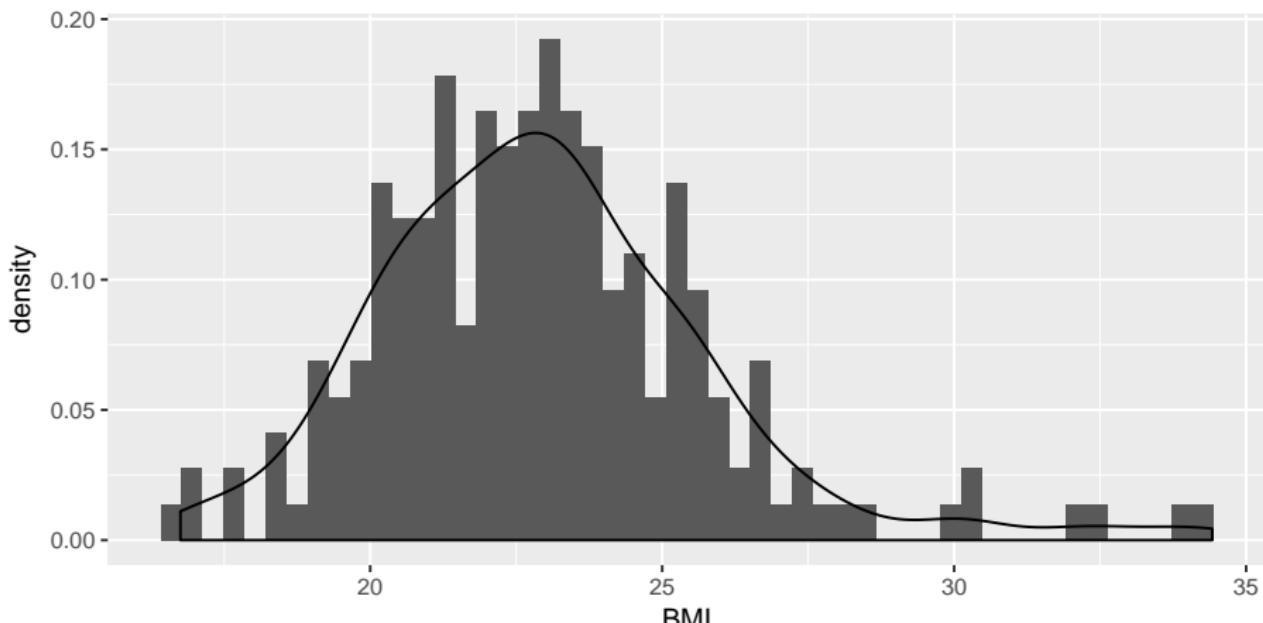
```
ggplot(athletes,aes(x=BMI))+
 geom_histogram(aes(y=..density..),bins=10)+
 geom_density()
```



## Too many bins

Kernel density still works for inappropriate number of bins:

```
ggplot(athletes,aes(x=BMI))+
 geom_histogram(aes(y=..density..),bins=50)+
 geom_density()
```

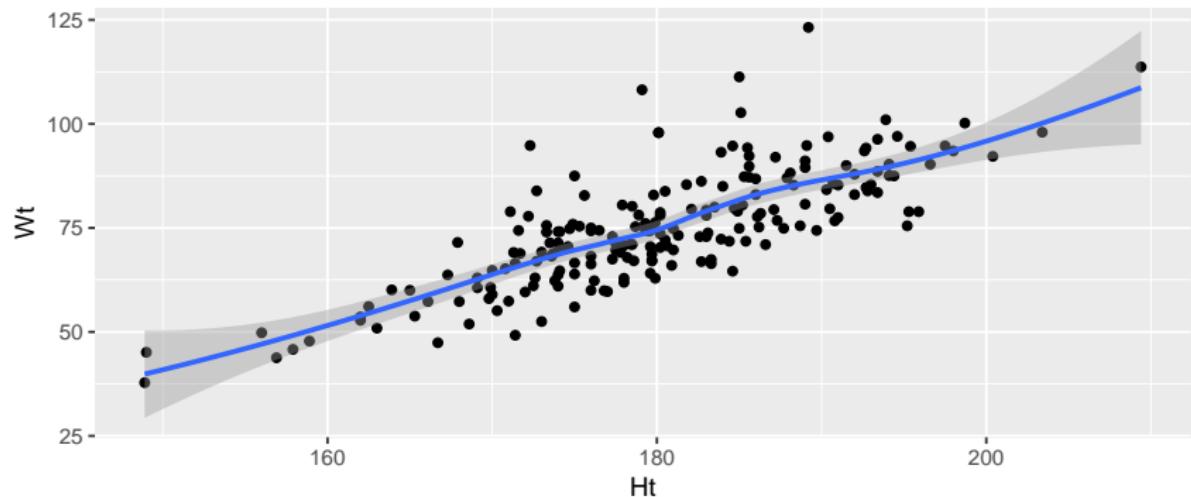


## Smooth trends

- ▶ Done by `geom_smooth` without method.
- ▶ Athletes height vs. weight:

```
ggplot(athletes, aes(x=Ht, y=Wt)) +
 geom_point() + geom_smooth()

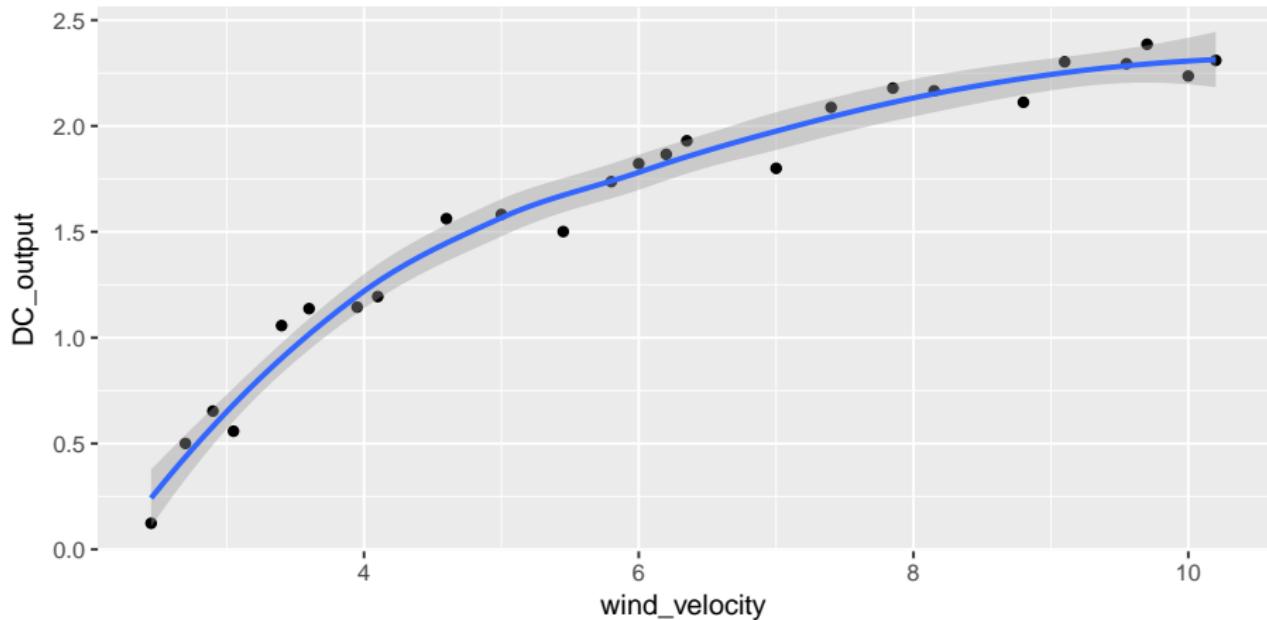
'geom_smooth()' using method = 'loess'
```



## And with windmill data

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

'geom_smooth()' using method = 'loess'
```

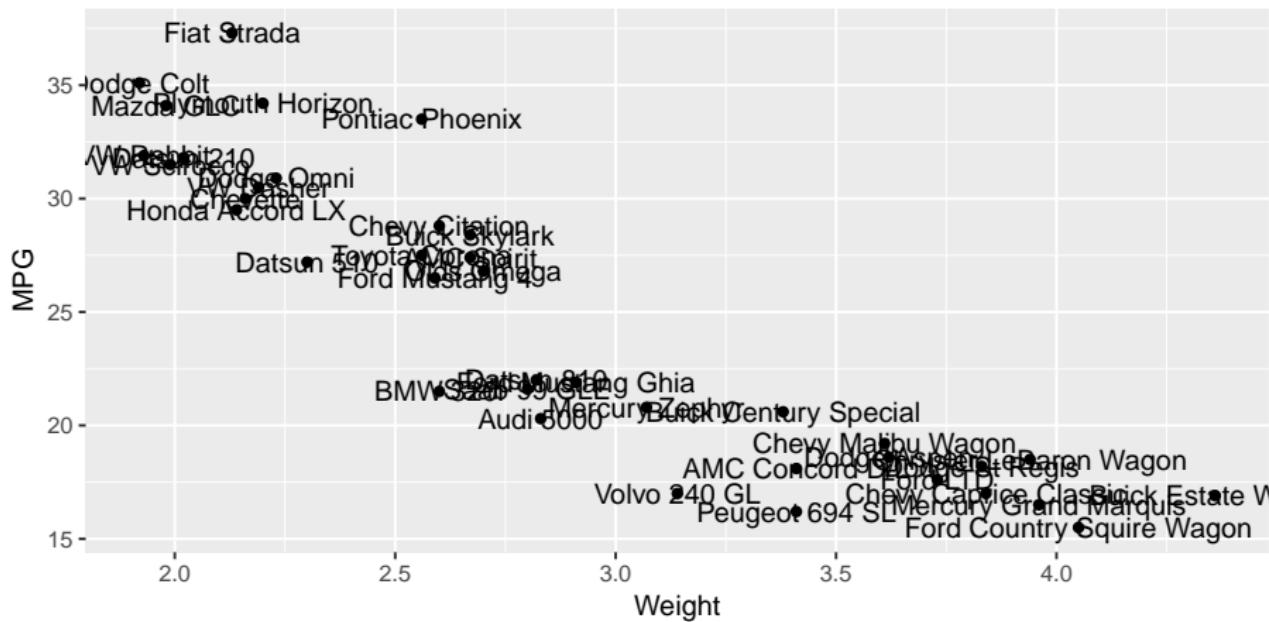


## Labelling observations

- ▶ There is a bad way and a good way.
- ▶ Bad way: `geom_text`
- ▶ Good way: `geom_text_repel` from package `ggrepel`.
- ▶ Illustrate on cars data with MPG vs. weight.
- ▶ Worst way first:

```
g=ggplot(cars,aes(x=Weight,y=MPG,label=Car))+
 geom_point()+geom_text()
```

## The worst plot



## Problems

- ▶ Labels are centred at points instead of off to side
- ▶ Text too big
- ▶ Labels overlap so you can't read them.
- ▶ If you put the labels next to the points, they might go off the plot.

## Adding labels to plot

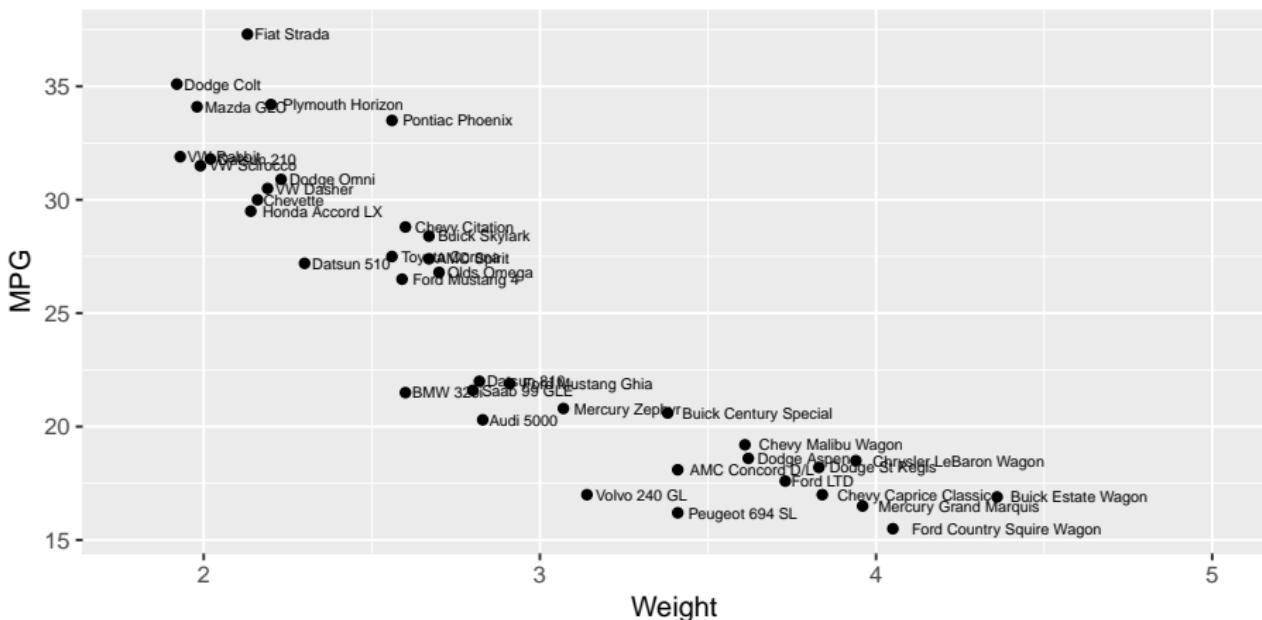
```
g=ggplot(cars,aes(x=Weight,y=MPG,label=Car))+
 geom_point() +
 geom_text(hjust=-0.1,size=2) +
 xlim(1.8,5.0)
```

## Comments

- ▶ `hjust` says where to put the labels relative to the points: 0.5 is centred over them, negative is on the right, greater than 1 is on the left.
- ▶ `vjust` similar to move labels up and down (less than 0, greater than 1 for above or below points).
- ▶ `size` controls size of text: 5 is default (so this is smaller).
- ▶ Not an obvious way to stop labels overlapping! But there is a solution, which we will see.
- ▶ `xlim` changes limits of `x-axis` (to stop labels going off side). Likewise `ylim`.

## Next attempt

g



Better, but labels still overlap. Need another solution.

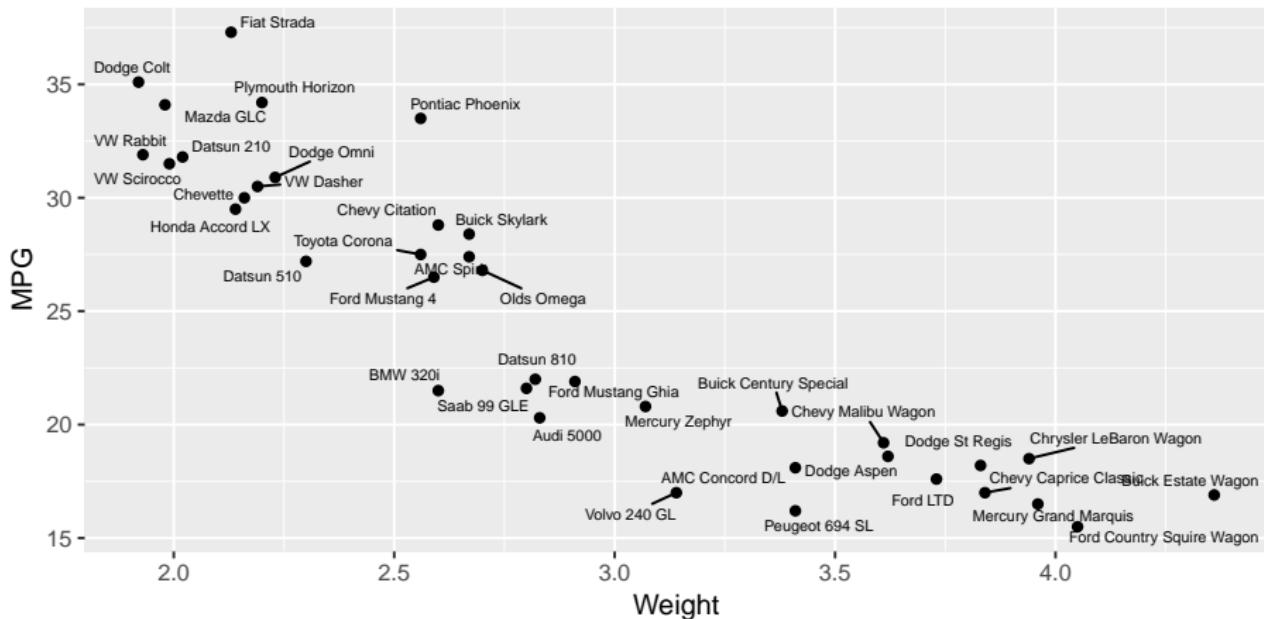
## Non-overlapping labels

- ▶ Key is to use package `ggrepel` and `geom_text_repel` from that package instead of `geom_text`:

```
library(ggrepel)
g=ggplot(cars, aes(x=Weight, y=MPG, label=Car))+
 geom_point()+
 geom_text_repel(size=2)
```

# Is that better?

g



Much better. I recommend this approach to labelling points on a graph.

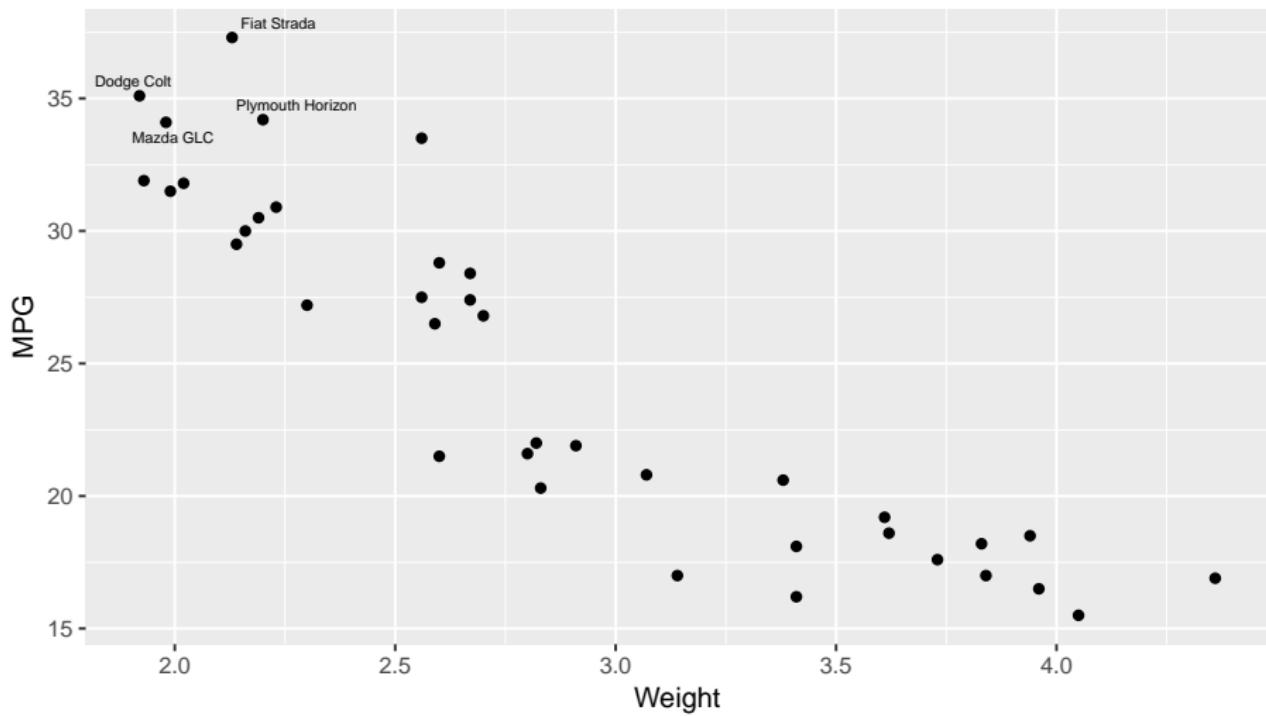
## Labelling some of the cars

- ▶ Maybe you want to draw attention only to *some* of the individuals
- ▶ for example labelling only certain cars or ones that satisfy a condition
- ▶ Mechanism: define a new label variable that contains:
  - ▶ the label, for the individual you want to label
  - ▶ blank text for those you don't (same idea as SAS)
- ▶ Handy function `ifelse`, like Excel =IF.
- ▶ Label cars with MPG over 34:

```
g=cars %>% mutate(newlabel=ifelse(MPG>34,Car,"")) %>%
 ggplot(aes(x=Weight,y=MPG,label=newlabel))+
 geom_point()+
 geom_text_repel(size=2)
```

# High gas-mileage cars

gg



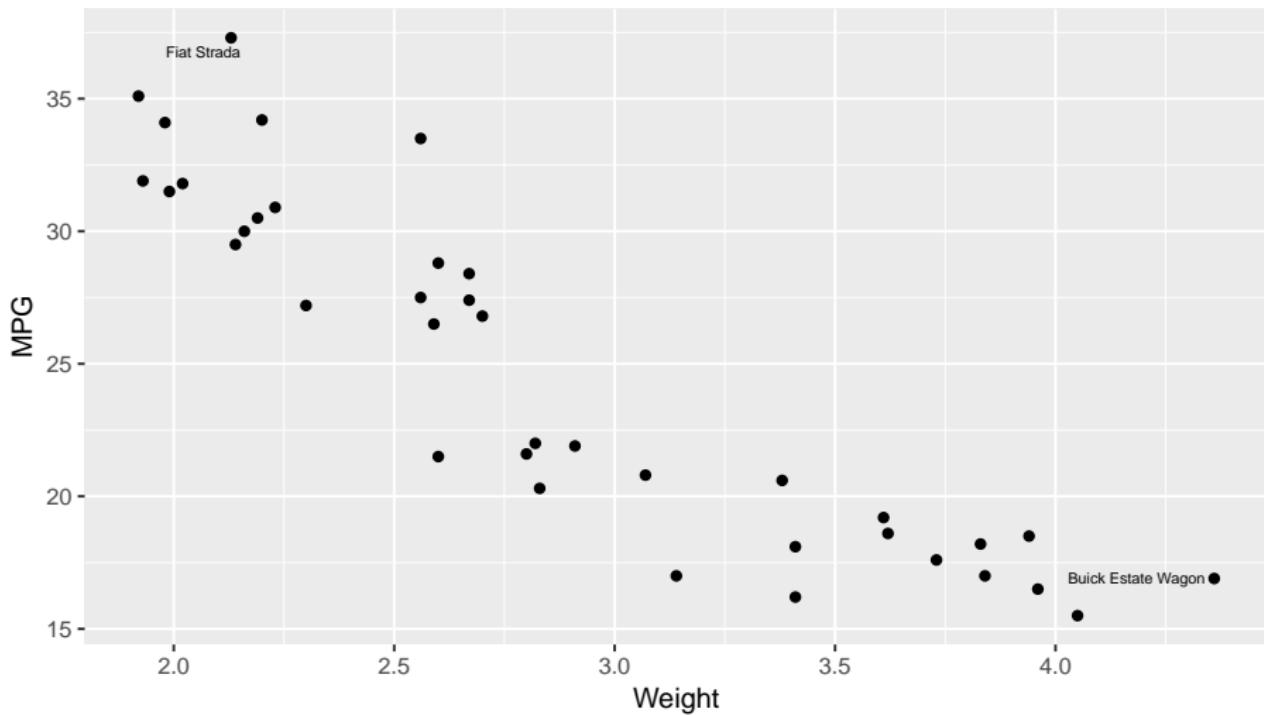
## Highest gas-mileage and highest-weight cars

- ▶ Found before that these were in rows 4 and 9 of data frame.
- ▶ How to use `ifelse` with row numbers? Define new column of row numbers, and then use it in `ifelse`, thus:

```
g=cars %>%
 mutate(row=row_number()) %>%
 mutate(newlabel=ifelse(row==4 | row==9, "Car", ""))
 ggplot(aes(x=Weight, y=MPG, label=newlabel)) +
 geom_point() +
 geom_text_repel(size=2)
```

# Heaviest and best-gas-mileage cars

g



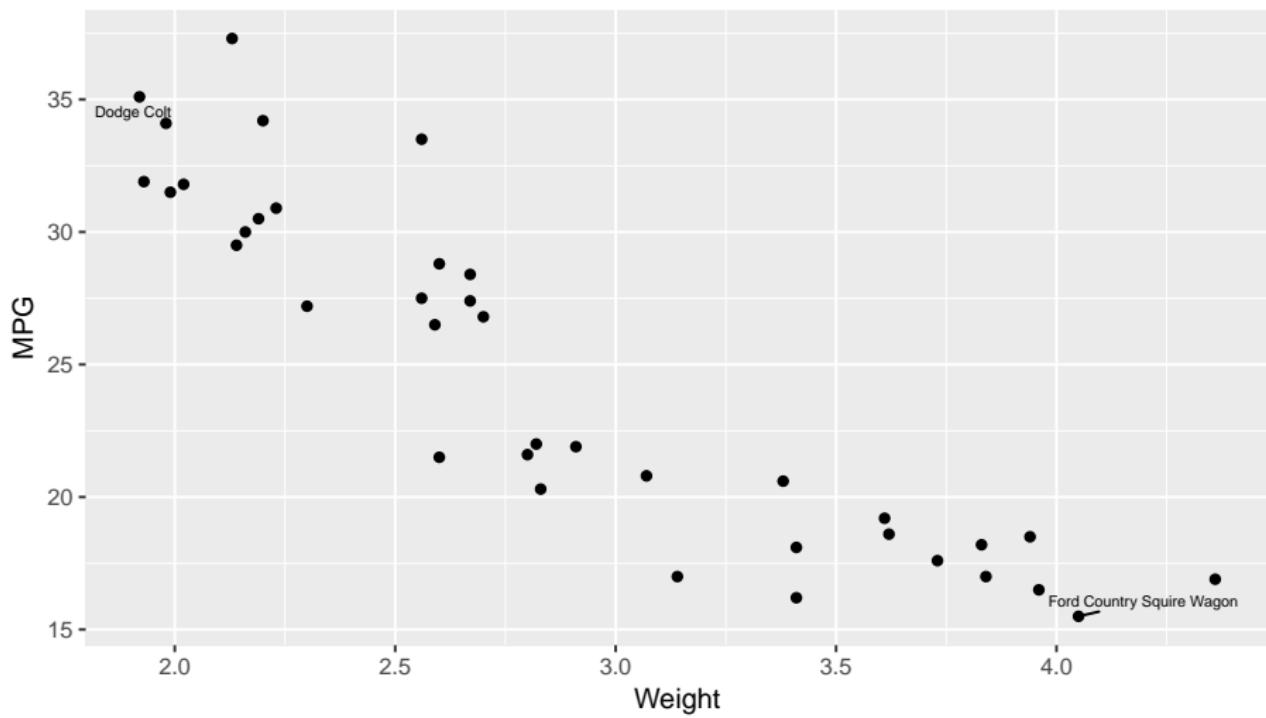
## Lightest weight and worst gas-mileage cars

- ▶ Suppose you didn't know which cars were the ones you wanted. Then you have to find them first.
- ▶ Now try for lightest weight and worst gas-mileage cars:

```
g=cars %>% mutate(tolabel= (Weight==min(Weight) |
 MPG==min(MPG))) %>%
 mutate(newlabel=ifelse(tolabel,Car,"")) %>%
 ggplot(aes(x=Weight,y=MPG,label=newlabel))+
 geom_point() +
 geom_text_repel(size=2)
```

# The plot

gg



## Miscellaneous graph things

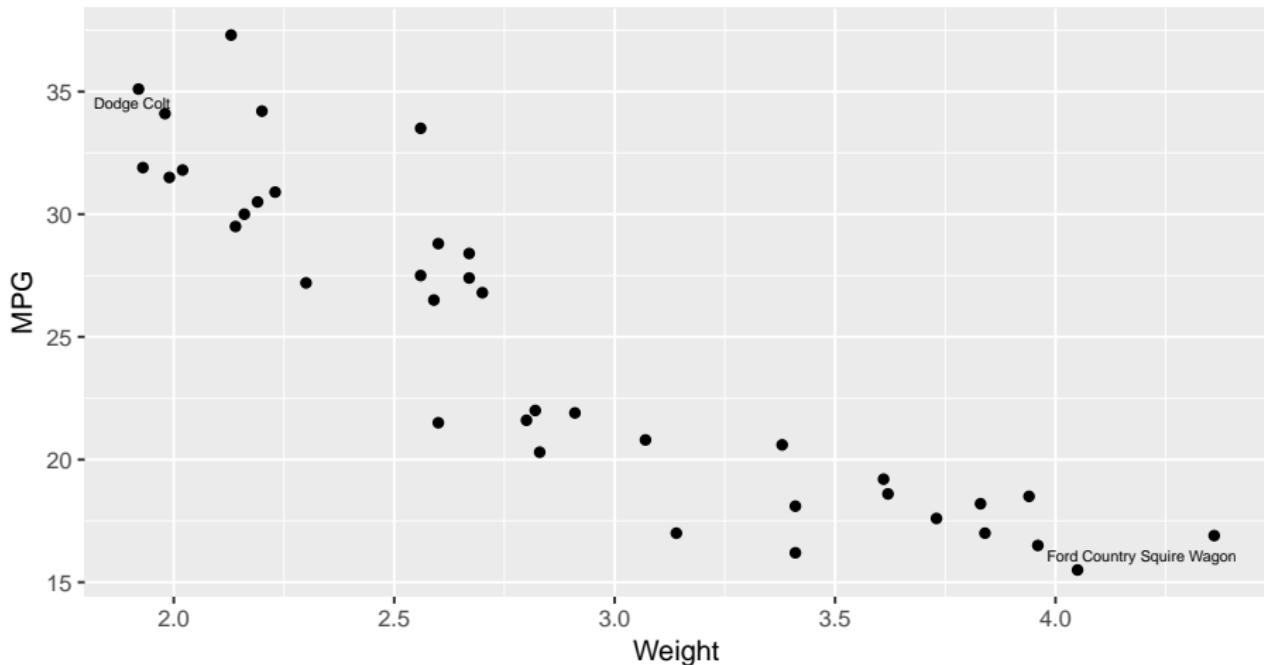
- ▶ Title for graph
- ▶ Axis labels
- ▶ Adding extra data to plot

We use previous graph as base (to save drawing again).

## With title

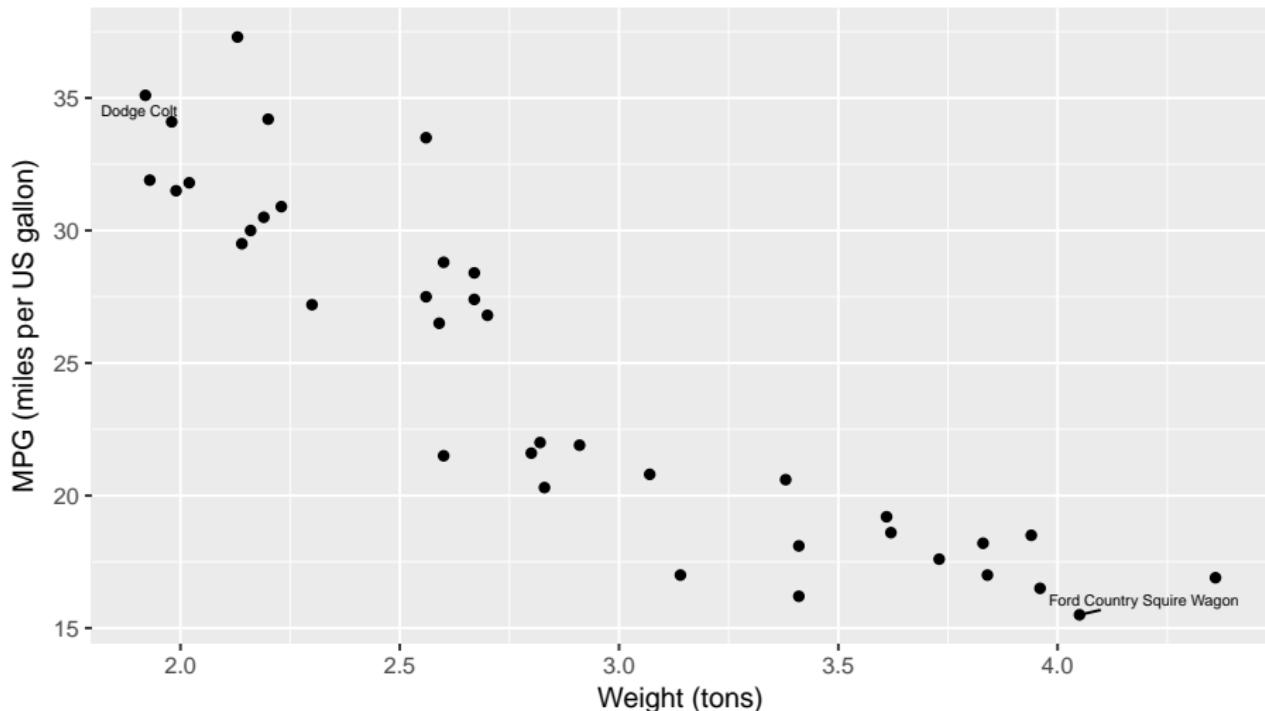
```
g+ggtitle("Gas mileage against weight")
```

Gas mileage against weight



## Axis labels

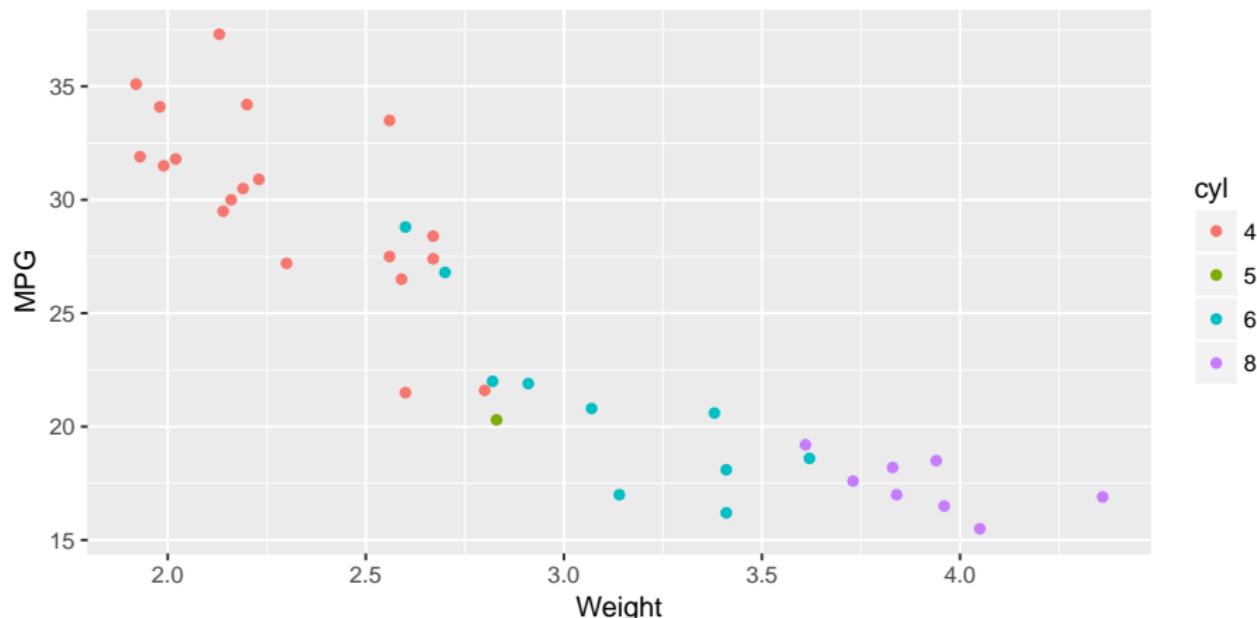
```
g+xlab("Weight (tons)")+ylab("MPG (miles per US gallon)")
```



## Labelling points by group

Cylinders, though a number, should be treated as categorical:

```
g3=cars %>% mutate(cyl=factor(Cylinders)) %>%
 ggplot(aes(x=Weight,y=MPG,colour=cyl))+
 geom_point() ; g3
```



## Adding new data: averages by cylinders

- ▶ First make data frame of new data to add:

```
summ=cars %>% group_by(Cylinders) %>%
 summarize(mw=mean(Weight),mm=mean(MPG))
summ

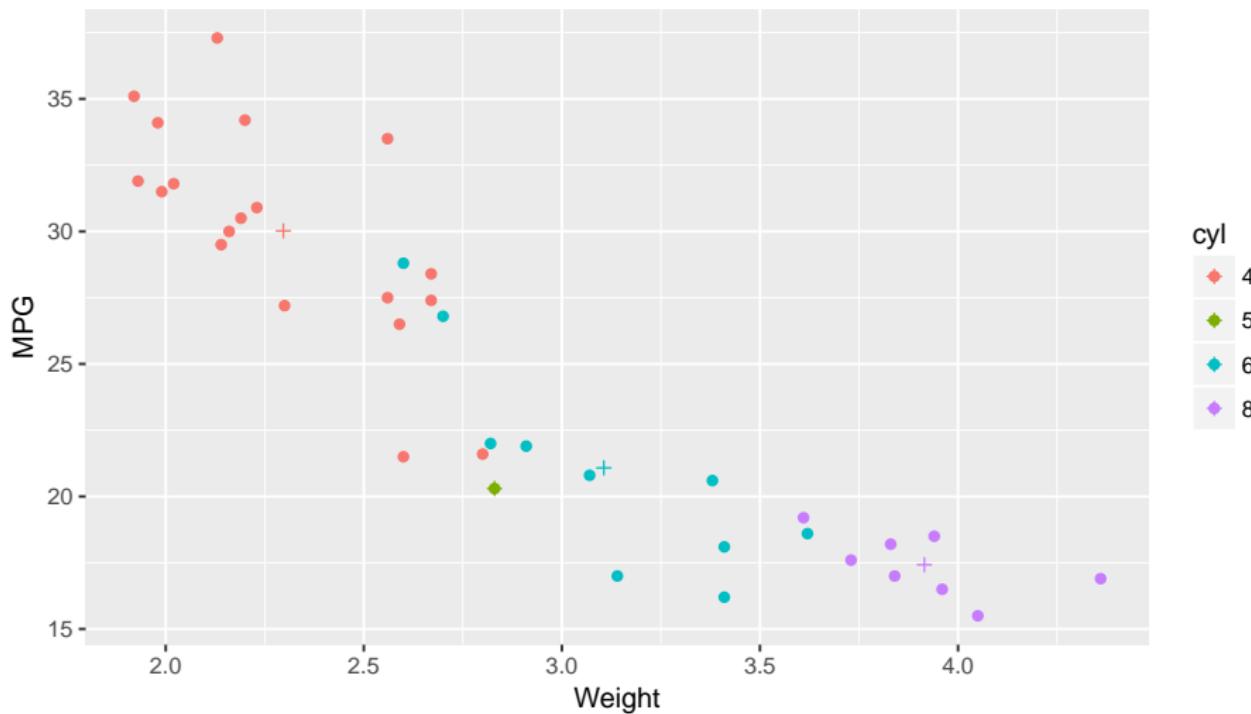
A tibble: 4 x 3
Cylinders mw mm
<int> <dbl> <dbl>
1 4 2.296842 30.02105
2 5 2.830000 20.30000
3 6 3.106000 21.08000
4 8 3.915000 17.42500
```

- ▶ then to plot averages on graph, add a new `geom_point` with a new data frame:

```
g4=g3+geom_point(data=summ,aes(x=mw,y=mm,
 colour=as.factor(Cylinders)),shape=3)
```

# The plot, group mean marked by +

g4



## Thinking back to SAS

We thought about a couple of issues in SAS that we should also address in R:

- ▶ Permanence of data in R
- ▶ Reading data with multiple obs per line, one variable
- ▶ Reading data with multiple obs per line, two variables

We talk briefly about those below.

## Permanence

- ▶ When you close R Studio, you are offered the option to “save your workspace”. If you choose “yes”, all of the data frames and other things you have created are saved, so that when you open R Studio in the same project later, you will be able to access all of these things. (“Everything is permanent” in that sense.)
- ▶ If you choose not to save your workspace, you will have to recreate all your objects next time (eg. re-read data from files). But you have a script to do that, don’t you?
- ▶ There is a school of thought that says you should *not* save your workspace, but keep scripts to re-create everything.
  - ▶ Pro: keeps your workspace “clean” of old objects that you created but don’t need any more, and you know exactly why everything is there.
  - ▶ Con: some objects take time and effort to re-create, and you won’t want to do that every time.
  - ▶ It is possible (beyond our scope) to save and re-load large/complicated objects so that they don’t have to be re-created.

## Multiple observations on one line

- ▶ Recall:

```
3 4 5 6 7 7
8 9 3 4 8 6
```

- ▶ These are all values of a variable `x`, six values on each line (thus 12 values in total).
- ▶ Read the data as R wants to. There are no column names, which we have to say explicitly (column names created)

```
many=read_delim("many.txt", " ", col_names=F)
```

```
Parsed with column specification:
```

```
cols(
X1 = col_integer(),
X2 = col_integer(),
X3 = col_integer(),
X4 = col_integer(),
X5 = col_integer(),
X6 = col_integer()
)
```

## Gather

- ▶ Idea: use `gather` to turn all six columns into one.
- ▶ This works, but we have to create a “dummy” column (named `col` here), that we ignore, to fill the “different” slot:

```
many %>% gather(col,x,X1:X6)
```

```
A tibble: 12 x 2
col x
<chr> <int>
1 X1 3
2 X1 8
3 X2 4
4 X2 9
5 X3 5
6 X3 3
7 X4 6
8 X4 4
9 X5 7
10 X5 8
11 X6 7
12 X6 6
```

## x and y, alternately

- ▶ Now suppose the values in the file are an x and a y, then another x and another y, and so on.
- ▶ `gather` works best on one “unit”, here an x-y pair, so strategy: create x-y pairs, gather, then separate out pairs.
- ▶ Start from here:

many

```
A tibble: 2 x 6
X1 X2 X3 X4 X5 X6
<int> <int> <int> <int> <int> <int>
1 3 4 5 6 7 7
2 8 9 3 4 8 6
```

- ▶ Tool to create pairs is `unite`, which we have to do three times (3 pairs).

## Creating the pairs

```
many %>%
 unite(xy1,X1,X2) %>%
 unite(xy2,X3,X4) %>%
 unite(xy3,X5,X6)

A tibble: 2 x 3
xy1 xy2 xy3
* <chr> <chr> <chr>
1 3_4 5_6 7_7
2 8_9 3_4 8_6
```

Next, we gather up those columns.

## Gathering up the pairs

```
many %>%
 unite(xy1,X1,X2) %>%
 unite(xy2,X3,X4) %>%
 unite(xy3,X5,X6) %>%
 gather(col,xy,xy1:xy3)

A tibble: 6 x 2
col xy
<chr> <chr>
1 xy1 3_4
2 xy1 8_9
3 xy2 5_6
4 xy2 3_4
5 xy3 7_7
6 xy3 8_6
```

Next, separate out those columns.

## Separating out

many %>%

```
unite(xy1,X1,X2) %>%
 unite(xy2,X3,X4) %>%
 unite(xy3,X5,X6) %>%
 gather(col,xy,xy1:xy3) %>%
 separate(xy,c("x","y"))
```

```
A tibble: 6 x 3
col x y
* <chr> <chr> <chr>
1 xy1 3 4
2 xy1 8 9
3 xy2 5 6
4 xy2 3 4
5 xy3 7 7
6 xy3 8 6
```

Same data as SAS (though not the same order of rows).

## Section 15

Vector and matrix algebra

## Vector and matrix algebra in R and SAS

- ▶ R has this built in, so we only have to see how it works.
- ▶ SAS has this through proc iml, which we have to learn about.
- ▶ Start with R, and flip back and forth.

## Vector addition

- ▶ Define a vector, then “add 2” to it:

```
u=c(2,3,6,5,7)
```

```
k=2
```

```
u+k
```

```
[1] 4 5 8 7 9
```

Adds 2 to each element.

- ▶ Adding vectors:

```
u
```

```
[1] 2 3 6 5 7
```

```
v=c(1,8,3,4,2)
```

```
u+v
```

```
[1] 3 11 9 9 9
```

Elementwise addition. (MAT A23: vector addition.)

## Scalar multiplication

As per A23:

```
k
[1] 2

u
[1] 2 3 6 5 7

k*u
[1] 4 6 12 10 14
```

Each element of vector multiplied by 2.

## “Vector multiplication”

What about this?

```
u
[1] 2 3 6 5 7

v
[1] 1 8 3 4 2

u*v
[1] 2 24 18 20 14
```

Each element of `u` multiplied by *corresponding* element of `v`. Could be called *elementwise multiplication*. (Not to be confused with “outer” or “vector” product from A23, or indeed “inner” or “scalar” multiplication, for which the answer is a number.)

## Combining different-length vectors

- ▶ No error here (you get a warning). What happens?

```
u
[1] 2 3 6 5 7
w=c(1,2)
u+w
[1] 3 5 7 7 8
```

- ▶ Add 1 to first element of u, add 2 to second.
- ▶ Go back to beginning of w to find something to add: add 1 to 3rd element of u, 2 to 4th element.
- ▶ Keep re-using shorter vector until reach length of longer one.
- ▶ R: “recycling”.
- ▶ Same idea is used when multiply a vector by a number: the number keeps getting recycled.

## Vectors and scalars in proc iml

- ▶ Define vectors and scalars as below.
- ▶ To do a calculation, define the answer into a variable, and then print it. Note that 2 has gotten added to each element of u:

```
proc iml;
k=2;
u={2 3 6 5 7};
ans=k+u;
print ans;
```

---

ans

4

5

8

7

9

## Adding vectors

- ▶ Each run of proc iml is independent, so you have to redefine anything you want to use. This is vector addition, as before:

```
proc iml;
 u={2 3 6 5 7};
 v={1 8 3 4 2};
 ans=u+v;
 print ans;
```

---

ans

3

11

9

9

9

## Elementwise and scalar multiplication

- ▶ Elementwise vector multiplication does not work in proc iml.
- ▶ Scalar multiplication, though, exactly as you would expect:

```
proc iml;
 k=2;
 u={2 3 6 5 7};
 ans=k*u;
 print ans;
```

---

ans

4

6

12

10

14

# Matrices (in R)

- ▶ Create matrix like this:

```
A=matrix(1:4,nrow=2,ncol=2)
A
[,1] [,2]
[1,] 1 3
[2,] 2 4
```

- ▶ First: stuff to make matrix from, then how many rows and columns.
- ▶ R goes *down columns* by default. To go along rows instead:

```
B=matrix(5:8,nrow=2,ncol=2,byrow=T)
B
[,1] [,2]
[1,] 5 6
[2,] 7 8
```

- ▶ One of `nrow` and `ncol` enough, since R knows how many things in the matrix.

## Adding matrices

What happens if you add two matrices?

A

```
[,1] [,2]
[1,] 1 3
[2,] 2 4
```

B

```
[,1] [,2]
[1,] 5 6
[2,] 7 8
```

A+B

```
[,1] [,2]
[1,] 6 9
[2,] 9 12
```

Nothing surprising here. This is matrix addition as we (and A23) know it.

# Multiplying matrices

- ▶ Now, what has happened here?

```
A
```

```
[,1] [,2]
[1,] 1 3
[2,] 2 4
```

```
B
```

```
[,1] [,2]
[1,] 5 6
[2,] 7 8
```

```
A*B
```

```
[,1] [,2]
[1,] 5 18
[2,] 14 32
```

- ▶ Not matrix multiplication (as per A23).
- ▶ Elementwise multiplication. Also called *Hadamard product* of A and B.

# Legit matrix multiplication

Like this:

A

```
[,1] [,2]
[1,] 1 3
[2,] 2 4
```

B

```
[,1] [,2]
[1,] 5 6
[2,] 7 8
```

A %\*% B

```
[,1] [,2]
[1,] 26 30
[2,] 38 44
```

## Matrices in proc iml

- ▶ Enter a matrix like a vector, but row by row, with a comma separating rows:

```
proc iml;
 A={1 3,2 4};
 B={5 6,7 8};
 print A;
 print B;
```

---

A

|   |   |
|---|---|
| 1 | 3 |
| 2 | 4 |

B

|   |   |
|---|---|
| 5 | 6 |
| 7 | 8 |

## Adding and multiplying matrices

- ▶ These are genuine matrix addition and multiplication (no elementwise multiplication):

```
proc iml;
 A={1 3,2 4};
 B={5 6,7 8};
 ans1=A+B;
 print ans1;
 ans2=A*B;
 print ans2;
```

---

ans1

|   |    |
|---|----|
| 6 | 9  |
| 9 | 12 |

ans2

|    |    |
|----|----|
| 26 | 30 |
| 38 | 44 |

## Reading a matrix in from a file 1/2

- ▶ If your matrix is in a file like this:

```
10 9
8 7
6 5
```

- ▶ read into data set as usual:

```
proc import
 datafile='/home/ken/m.txt'
 dbms=dlm
 out=mymatrix
 replace;
 delimiter=' ' ;
 getnames=no;
```

- ▶ Columns get names VAR1, VAR2, etc.

## Reading a matrix in from a file 2/2

- ▶ and then use in proc iml thus:

```
proc iml;
 use mymatrix;
 read all var {VAR1 VAR2} into M;
 v={1,3};
 ans=M*v;
 print ans;
```

---

ans

37

29

21

## Reading matrix from file in R

- ▶ The usual:

```
M=read_delim("m.txt", " ", col_names=F)

Parsed with column specification:
cols(
X1 = col_integer(),
X2 = col_integer()
)

class(M)

[1] "tbl_df" "tbl" "data.frame"
```

- ▶ except that M is not an R matrix, and thus this doesn't work:

```
v=c(1,3)
M %*% v

Error in M %*% v: requires numeric/complex
matrix/vector arguments
```

# Making a real matrix

Do this first:

```
M=as.matrix(M)
```

and then all is good:

```
M %*% v
[,1]
[1,] 37
[2,] 29
[3,] 21
```

## Linear algebra stuff

- ▶ To solve system of equations

$Ax = w$  for  $x$ :

```
A
[,1] [,2]
[1,] 1 3
[2,] 2 4

w
[1] 1 2

solve(A,w)
[1] 1 0
```

- ▶ To find the inverse of  $A$ :

```
A
[,1] [,2]
[1,] 1 3
[2,] 2 4

solve(A)
[,1] [,2]
[1,] -2 1.5
[2,] 1 -0.5
```

- ▶ You can check that these are correct.

## And in proc iml

- ▶ Solve  $Ax = w$  as  $x = A^{-1}w$ .
- ▶ Thus, strategy is to find inverse *first*:

```
proc iml;
A={1 3,2 4};
w={1,2};
Ainv=inv(A);
print Ainv;
ans=Ainv*w;
print ans;
```

---

Ainv

|     |      |
|-----|------|
| -2  | 1.5  |
| 1   | -0.5 |
| ans |      |

1  
0

## Row and column vectors in proc iml

- ▶ Without commas gives a *row* vector in proc iml.
- ▶ *With* commas gives a column vector:

```
proc iml;
 r={1 2 3};
 c={4,5,6};
 print r;
 print c;
```

r

1                  2                  3

c

4  
5  
6

## Inner product

- ▶ Make sure both vectors are *column* vectors, and then matrix-multiply the transpose of the first (a row vector) by the second:

```
proc iml;
 a={1,2,3};
 b={4,5,6};
 ans=t(a)*b;
 print ans;
```

---

ans

32

## Inner product in R

- ▶ Vectors in R are column vectors, so just do the matrix multiplication:

```
a=c(1,2,3)
b=c(4,5,6)
t(a) %*% b

[,1]
[1,] 32
```

Note that the answer is actually a  $1 \times 1$  matrix.

- ▶ Or as the sum of the elementwise multiplication:

```
sum(a*b)

[1] 32
```