

# Welcome to STAC32!

## Applications of Statistical Methods

Instructor: Ken Butler

## In this course

- ▶ Data analysis: use mostly known (to you) statistical methods to gain insight about data.
- ▶ Use two software packages, SAS and R — I assume you know nothing about these.
- ▶ Look at lots of examples.
- ▶ “Get your hands dirty”.
- ▶ Writing reports.
- ▶ Other tasks in R and SAS.

# Part I

## Housekeeping

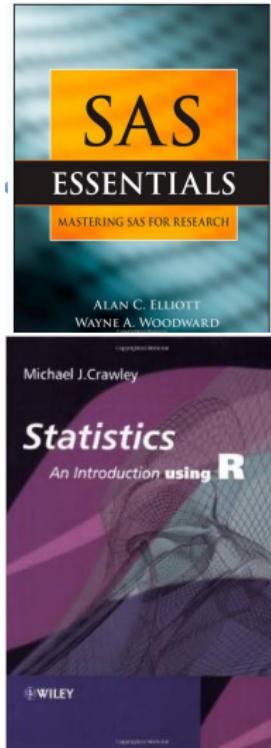
## The instructor

- ▶ Ken Butler, office IC 471, e-mail:  
<mailto:butler@utsc.utoronto.ca>
- ▶ Lectures Tuesday 13:00-14:00 in IC 220, Thursday 12:00-13:00 in IC 230, Thursday 13:00-14:00 in B469/498. Second hour of Thursday class in computer lab.
- ▶ Office hours: Tuesday 14:00-16:00, Thursday 14:00-15:00 or longer if needed. Or by appointment (e-mail me, address above, to set one up).
- ▶ Link to course website:  
<http://www.utsc.utoronto.ca/~butler/c32/>
- ▶ I use Blackboard only for grades.
- ▶ E-mail: Use only UTSC/UToronto e-mail address. I aim to respond within two working days. Non-simple questions: office hours.

## Text(s)

Recommended:

- ▶ “SAS Essentials: Mastering SAS for research” by Alan C. Elliott and Wayne A. Woodward, publ. Jossey-Bass. ISBN 978-0-470-46129-7.
- ▶ “Statistics: an Introduction using R” by Michael J. Crawley, publ. Wiley. ISBN 978-0-470-02298-6.



## Structure of course

- ▶ 3 hours/week of class time.
- ▶ Of these, 1 hour in computer lab:
  - ▶ practice what you learned from lecture
  - ▶ I give you lab exercises to work on (with small questions to hand in).
  - ▶ Get help if needed.
  - ▶ Small question(s) to hand in.

## Course material

- ▶ Demo of what R and SAS can do
- ▶ Installation and connection
- ▶ Using SAS
- ▶ Using R
- ▶ Basics of data analysis, with examples
- ▶ Statistical inference
- ▶ Writing reports and reproducible research
- ▶ More data analysis (including regression and multiple regression)
- ▶ A look at the intricacies of R
- ▶ A look at the intricacies of SAS

## Assessment

Item	Weight
Lab session hand-in (every week).	5
Assignments (every 2 weeks or so). Can study with classmates, but what you hand in must be <b>entirely your own work.</b>	25
Midterm exam, 2 hours, open book (see below)	20
Data analysis report. Data analysis on data set of your choosing, using R/SAS/both, report written as described in class	15
Final exam, 3 hours, open book (see below)	35

“Open book” means these are permissible:

- ▶ the SAS text
- ▶ the R text
- ▶ my lecture notes
- ▶ any other notes that you have made in this course.

## Missed work and documentation

- ▶ No make-up assignments or tests in this course.
- ▶ Work handed in late may or may not be accepted (instructor's discretion; instructor's decision is final).
- ▶ If you miss assessed work due to illness/injury, complete form at <http://www.illnessverification.utoronto.ca/> and submit to instructor *within 10 working days of due date*.
- ▶ Weight of missed work *with appropriate documentation* transferred to other assessments of same type.

## Academic integrity

- ▶ Read and understand  
<http://www.utoronto.ca/academicintegrity/>
- ▶ Academic dishonesty devalues *your* degree *and* those of *all* other students.
- ▶ Cheating and plagiarism taken very seriously.
- ▶ Examples of academic offences:
  - ▶ Using someone else's words/ideas without acknowledgement.
  - ▶ Obtaining or providing unauthorized assistance on an assignment.
  - ▶ On test/exam, looking at someone else's answers or *allowing someone else to look at yours*.
  - ▶ Pretending to be someone else.
  - ▶ Falsifying or altering required documentation (eg. doctor's notes).
- ▶ Penalties include (depending on severity) a mark of zero for the work concerned, a mark of zero for the entire course, or suspension/expulsion from the University.

## Accessibility statement

*Students with diverse learning styles and needs are welcome in this course. In particular, if you have a disability/health consideration that may require accommodations, please feel free to approach me and/or the AccessAbility Services Office as soon as possible. I will work with you and AccessAbility Services to ensure you can achieve your learning goals in this course. Enquiries are confidential. The UTSC AccessAbility Services staff (located in S302) are available by appointment to assess specific needs, provide referrals and arrange appropriate accommodations: (416) 287-7560 or by e-mail as below.*

[ability@utsc.utoronto.ca](mailto:ability@utsc.utoronto.ca)

## Part II

Hello world

## Demonstration

- ▶ Begin with brief demonstration of how our software works.
- ▶ Enter (a little) data.
- ▶ Calculate summary statistics.
- ▶ Draw histogram and boxplot.

# R

Enter data and find mean:

```
R> x=c(10,11,13,14,14,15,15,17,20,25,29)
```

```
R> mean(x)
```

```
[1] 16.63636
```

# R

Enter data and find mean:

```
R> x=c(10,11,13,14,14,15,15,17,20,25,29)
```

```
R> mean(x)
```

```
[1] 16.63636
```

Standard deviation:

```
R> sd(x)
```

```
[1] 5.852738
```

# R

Enter data and find mean:

```
R> x=c(10,11,13,14,14,15,15,17,20,25,29)
```

```
R> mean(x)
```

```
[1] 16.63636
```

Standard deviation:

```
R> sd(x)
```

```
[1] 5.852738
```

Median and quartiles (5-number summary):

```
R> quantile(x)
```

```
0% 25% 50% 75% 100%
```

```
10.0 13.5 15.0 18.5 29.0
```

# R

Enter data and find mean:

```
R> x=c(10,11,13,14,14,15,15,17,20,25,29)
```

```
R> mean(x)
```

```
[1] 16.63636
```

Standard deviation:

```
R> sd(x)
```

```
[1] 5.852738
```

Median and quartiles (5-number summary):

```
R> quantile(x)
```

0% 25% 50% 75% 100%

10.0 13.5 15.0 18.5 29.0

Inter-quartile range

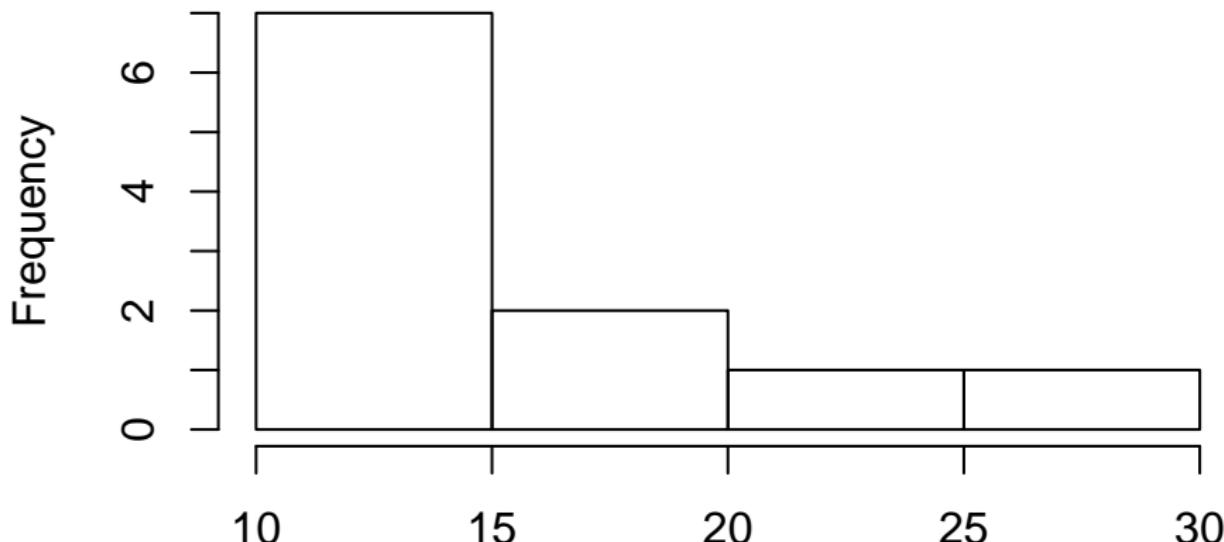
```
R> IQR(x)
```

```
[1] 5
```

## Histogram

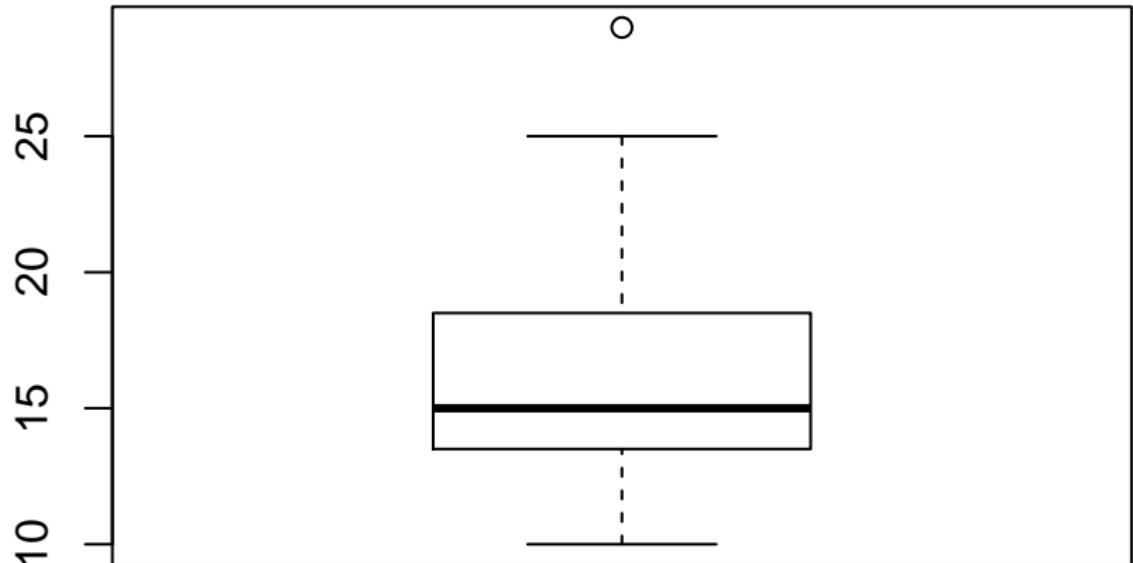
```
R> hist(x)
```

**Histogram of x**



## Boxplot

```
R> boxplot(x)
```



## Summary / R's way of working

- ▶ Mean 16.6, median 15.
- ▶ Quartiles 13.5, 18.5
- ▶ SD 5.85, IQR 5.
  
- ▶ Ask it to do one thing at a time
- ▶ Get one result at a time
- ▶ Decide what to do next

# SAS

Enter data first, then ask for summaries (output next):

```
SAS>   data xx;  
SAS>     input x @@;  
SAS>     group=1;  
SAS>     cards;  
SAS>       10 11 13 14 14 15 15 17 20 25 29  
SAS>     ;  
SAS>  
SAS>   proc univariate;  
SAS>     var x;
```

# Summary statistics output

The UNIVARIATE Procedure

Variable: x

## Moments

N	11	Sum Weights	11
Mean	16.6363636	Sum Observations	183
Std Deviation	5.85273829	Variance	34.2545455
Skewness	1.1877081	Kurtosis	0.77406932
Uncorrected SS	3387	Corrected SS	342.545455
Coeff Variation	35.1803941	Std Error Mean	1.76466699

## Basic Statistical Measures

Location	Variability		
Mean	16.63636	Std Deviation	5.85274
Median	15.00000	Variance	34.25455
Mode	14.00000	Range	19.00000
		Interquartile Range	7.00000

Note: The mode displayed is the smallest of 2 modes with a count of 2.

## Tests for Location: Mu0=0

Test	-Statistic-	-----	p Value-----
Student's t	t	9.42748	Pr >  t  <.0001
Sign	M	5.5	Pr >=  M  0.0010
Signed Rank	S	33	Pr >=  S  0.0010

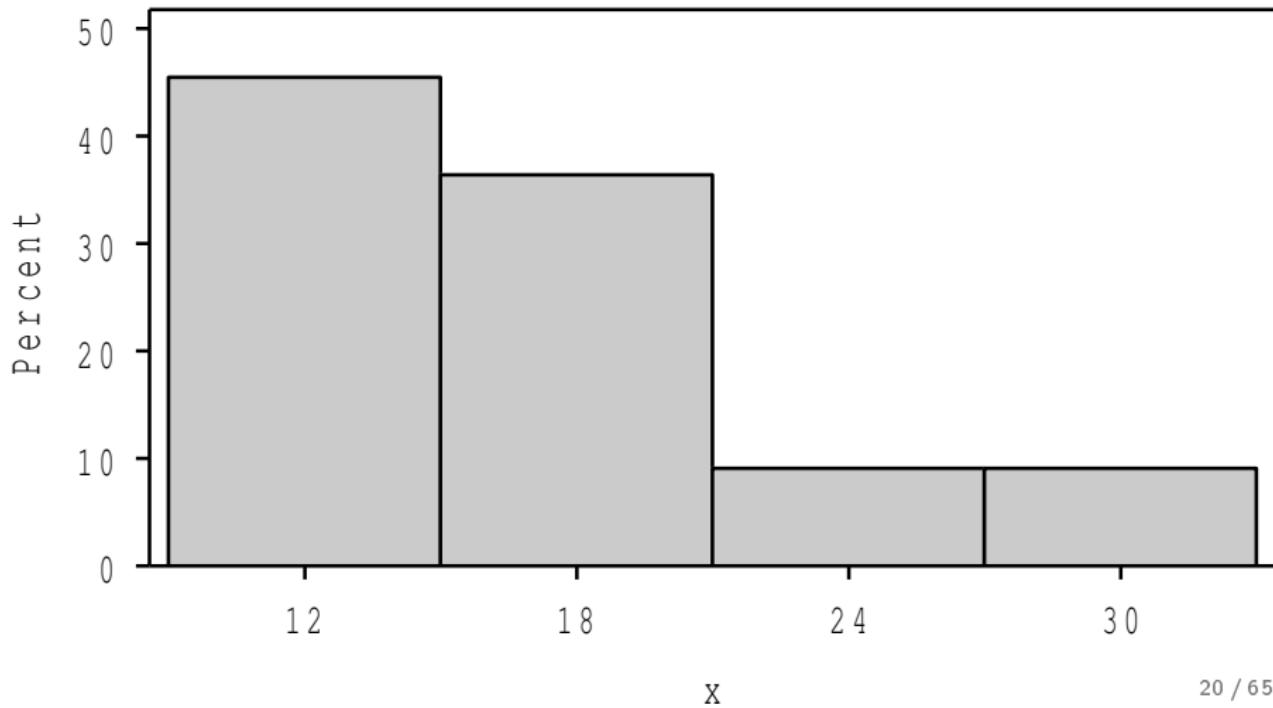
## Quantiles (Definition 5)

Quantile Estimate

100% Max	29
99%	29
95%	29
90%	25
75% Q3	20
50% Median	15
25% Q1	13
10%	11

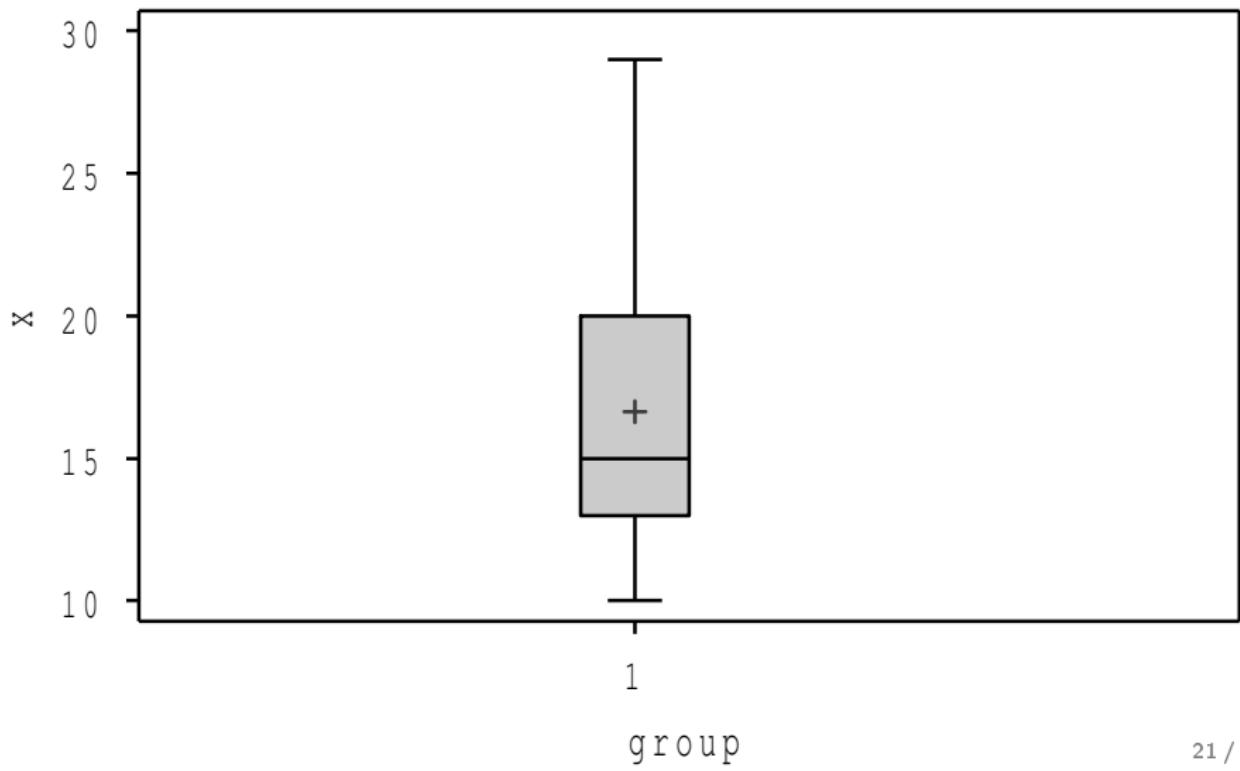
## Histogram

```
SAS> proc univariate noint;
SAS>   var x;
SAS>   histogram;
```



## Boxplot

```
SAS> proc boxplot;  
SAS> plot x*group / boxstyle=schematic;
```



## Summary / SAS's way of working

- ▶ Mean 16.6, median 15.
- ▶ Quartiles 13 and 20.
- ▶ SD 5.85, IQR 7.
- ▶ SAS boxplot does *not* have outlier on it.
  
- ▶ Assemble commands for SAS, “submit” all at once.
- ▶ Get *pile* of output, search for what you need.
- ▶ Often end up ignoring most of output!

## Comparison of results

Statistic	R	SAS	comment
Mean	16.6	16.6	check
Median	15	15	check
SD	5.85	5.85	check
Q1	13.5	13	??
Q3	18.5	20	??
IQR	5	7	??

There are many definitions of quartile: SAS has 5, R has 9 (!), and the defaults are *different*. By course (depending on where you are coming from):

Course	R	SAS
STAB22	Keep median	Throw median
STAB57	#4	#1

## Checking for outliers

Is the highest value 29 an outlier?

Go up  $1.5 \times IQR$  from Q3 (down same from Q1):

Using R's values:

```
R> 18.5+1.5*5
```

```
[1] 26
```

29 is higher than this, so is an outlier. But, using SAS's values:

```
R> 20+1.5*7
```

```
[1] 30.5
```

This time, 29 is *not* an outlier.

Confusing, eh?

# Part III

## 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.
  - ▶ In those days, 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.
  - ▶ Open-source, free. Core group, anyone can contribute.
  - ▶ Grew out of commercial software S, which appeared when graphics terminals new (emphasis on graphics).
  - ▶ R style: enter commands one at a time, see output/graphics right away.
  - ▶ 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).
- ▶ Details in Chapter 1 of  
<http://www.utsc.utoronto.ca/~butler/r/book.pdf>.
- ▶ Startup by starting R Studio.

## 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.
- ▶ The hard part is getting registered for it.

## Getting registered

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

The screenshot shows the SAS Sign In page. At the top, there is a blue header bar with the text "Sign In to SAS®" and the SAS logo "sas" next to it. Below the logo is the tagline "THE POWER TO KNOW.". On the right side of the header is a link labeled "About". The main area has a dark blue background with diagonal stripes. It contains two input fields: "User ID:" and "Password:", each with a red asterisk (\*) indicating a required field. To the right of each input field is a small box showing the character count: "9" for User ID and "0" for Password. Below the input fields is a blue "Sign In" button. At the bottom of the page, there is a white footer section containing links: "DON'T HAVE AN ACCOUNT? [REGISTER HERE](#)", "FORGOT YOUR USER ID? [CLICK HERE](#)", and "FORGOT YOUR PASSWORD? [CLICK HERE](#)". The footer also includes the copyright notice "Copyright © 2002 - 2013 by SAS Institute Inc., Cary, NC USA".

- ▶ 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**

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=BBC03221-340E-9BEF-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:



The image shows the SAS Sign In page. At the top right, there is a link to "About". Below the header, the SAS logo is displayed with the tagline "THE POWER TO KNOW." To the left of the logo, the text "Sign In to SAS®" is written. The main area contains two input fields: one for "User ID" and one for "Password", both marked with an asterisk (\*) indicating they are required fields. Below these fields is a blue "Sign In" button. At the bottom of the page, there are links for users who don't have an account ("REGISTER HERE") or forgot their user ID or password ("CLICK HERE"). Copyright information for SAS Institute Inc. is also present at the bottom.

About

## Sign In to SAS®

**sas** | THE POWER TO KNOW.

User ID: \*

Password: \*

**Sign In**

DON'T HAVE AN ACCOUNT? [REGISTER HERE](#)  
FORGOT YOUR USER ID? [CLICK HERE](#)  
FORGOT YOUR PASSWORD? [CLICK HERE](#)

Copyright © 2002 - 2013 by SAS Institute Inc., Cary, NC USA

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

# The dashboard

The screenshot shows the SAS Dashboard. At the top, there's a blue header bar with the word "Dashboard" on the left and a user profile "Megan Butler" on the right. Below the header, the main content area is divided into several sections:

- Applications**: A section containing a link to "SAS® Studio".

**SAS® Studio**  
Write and run SAS code with a Web-based SAS development environment.
- Courses I teach** (create a new course): A section showing a table with columns for Name, Description, and Institution. There are no visible rows of data.
- Courses I am enrolled in**: A section showing a table with columns for Name, Description, Instructor, and Institution. There are no visible rows of data.
- Reference**: A section with links to "Support Site", "Step-by-Step Registration Guides", "Usage Guide (coming soon)", and "Commonly Asked Questions".

At the bottom of the dashboard, there's a note: "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.)

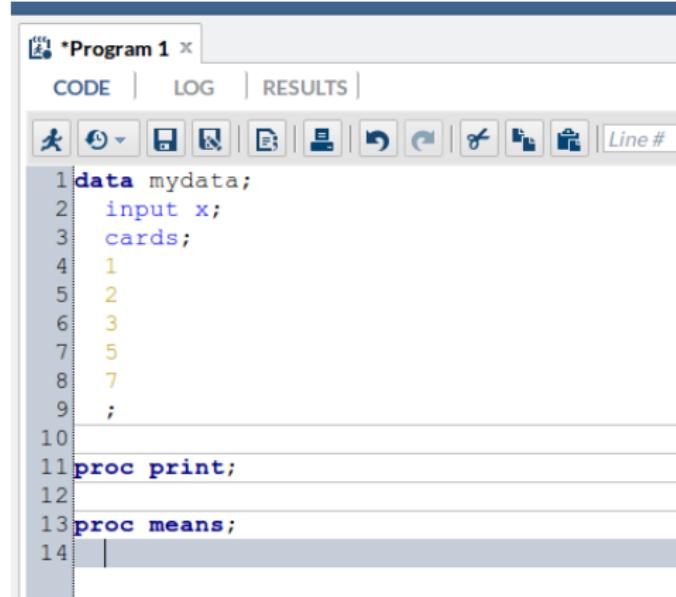
# SAS, as you see it

Something like this:

The screenshot shows the SAS Web Editor interface. On the left, there is a sidebar titled "SAS® Web Editor" with a "Folders" section. It contains a search bar, a toolbar with icons for creating new files, and a tree view of folders and files. The "My Folders" section is expanded, showing "sasuser.v93", "Program 1.sas", "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. The "CODE" tab is active, showing the placeholder text "Enter your code here".

## Trying out SAS

Go to the right side of the window, under Program 1, click on Code, and type the following into the window with the numbered lines:

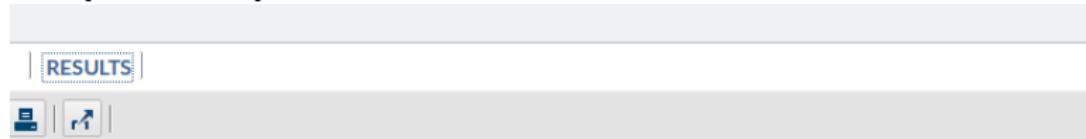


```
1 data mydata;
2   input x;
3   cards;
4   1
5   2
6   3
7   5
8   7
9   ;
10
11 proc print;
12
13 proc means;
14
```

When you have this to your satisfaction, click the “running humanoid” under Code. (This is called “submitting” in SAS jargon.)

## Output

If everything was correct, the Results tab under Program 1 will be selected, and you'll see your results, thus:



The screenshot shows a software window with a toolbar at the top. On the left of the toolbar is a vertical bar with a downward arrow icon. Next to it is a button with a blue border containing the word "RESULTS". To the right of that is another button with a blue border containing a double-headed arrow icon. The main area of the window is currently empty, indicating no results have been displayed yet.

Obs	x
1	1
2	2
3	3
4	5
5	7

The MEANS Procedure				
Analysis Variable : x				
N	Mean	Std Dev	Minimum	Maximum
5	3.6000000	2.4083189	1.0000000	7.0000000

These are: a listing of your data (from proc print) and a summary of the data, including mean and SD (from proc means).

## Errors

Let me make a deliberate mistake: I left off the semicolon on the end of `proc print`. When I submitted this, the Log tab popped up with a whole bunch of stuff including this:

```
48      ;
49
50      proc print
51
52      proc means;
      _____
22
202
ERROR 22-322: Syntax error, expecting one of the following: ;, BLANKLINE, DATA, DOUBLE, HEA
               SPLIT, STYLE, SUMLABEL, UNIFORM, WIDTH.
ERROR 202-322: The option or parameter is not recognized and will be ignored.
53
54      ;
```

When SAS hit the `proc means`, it was expecting a semicolon (to finish off the `proc print;`), but didn't see one. So the error was *just before* where the mark was.

The tactic is: fix the *first* error and submit again. (That first error might have caused a bunch of others.)

Sometimes the last thing won't run. If that happens to you, type a `run;` on a line by itself at the end of the code.

## More stuff

- ▶ To save your code, click on one of the symbols under Log. (Save, Save As). SAS will append a .sas to the name you supply. You'll see your code file appear on the left, under Folders.
- ▶ To open previously-saved code, double-click on the code file name under Folders. A new tab will appear with the code file.
- ▶ To create a new code file, click the second button under Folders (its tooltip says “New SAS program”).
- ▶ These files all live on the SAS server. To make a copy of a file on your own computer, click on the file in the Folders window, and click the down-arrow “download” button. You'll be prompted to open or save it.
- ▶ You can also upload files. Click the up-arrow “upload” button, and click Choose Files to select the file to upload. The folder with a name like /home/ken is your file storage on the SAS server.
- ▶ You can create subfolders. Click on the leftmost button under Folders, and give the new folder a name.

## Using Web Editor to save a data file

Create a new “SAS Program” (only it won’t be) and type/copy these, as shown, into the Code window:

```
a 20  
a 21  
a 16  
b 11  
b 14  
b 17  
b 15  
c 13  
c 9  
c 12  
c 13
```

- ▶ Save into file `threegroups.txt`. Select My Folders as folder to save in. SAS puts `.sas` on the file name. Find the file in the Folders window, right-click, select Rename, and make the filename what you really wanted.

## Analysis 1: reading in and verifying the data

- ▶ Now you can remove it from the right-hand side by clicking the X on its tab.
- ▶ Create a new SAS program, and enter its code thus. Don't type the SAS>.

```
SAS> data groups;  
SAS>   infile '/home/ken/threegroups.txt';  
SAS>   input group $ y;  
SAS>  
SAS> proc print;
```

- ▶ Replace ken with your Web Editor ID.
- ▶ Code much cleaner with data in separate file.

## Output 1

Obs	group	y
1	a	20
2	a	21
3	a	16
4	b	11
5	b	14
6	b	17
7	b	15
8	c	13
9	c	9
10	c	12
11	c	13

Go back to Code tab, enter code on next slide below what was there before, and submit whole thing again.

## Analysis/output 2: means by group

```
SAS> proc means;  
SAS>   class group;  
SAS>   var y;
```

The MEANS Procedure

Analysis Variable : y

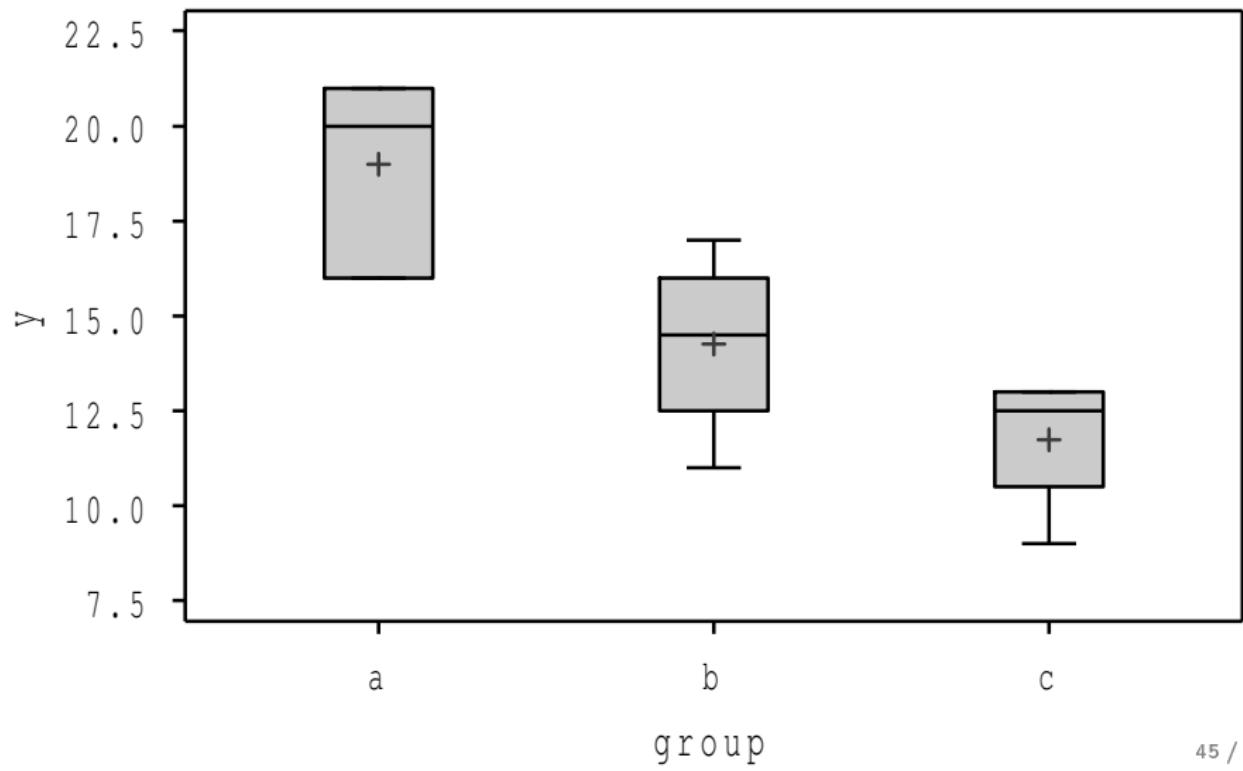
group	Obs	N	Mean	Std Dev	Minimum
a	3	3	19.0000000	2.6457513	16.0000000
b	4	4	14.2500000	2.5000000	11.0000000
c	4	4	11.7500000	1.8929694	9.0000000

Analysis Variable : y

group	Obs	Maximum
a	3	21.0000000
b	4	17.0000000
c	4	13.0000000

## Analysis 3: boxplots

```
SAS> proc boxplot;  
SAS> plot y*group / boxtyle=schematic;
```



## Conclusions

Both boxplots and proc means support idea that group A has largest values and group C has smallest.

## Copying into SAS

- ▶ Copying *into* SAS mostly easy: copy as normal, paste into Code tab.
- ▶ If copying from spreadsheet, like this,

	A	B	C	D
1	1	4	7	
2	2	5	8	
3	3	6	9	
4	10	11	12	
5				
6				
7				

values separated by *tabs*. Steps:

- ▶ Copy into SAS code tab as usual.
- ▶ Save into file, eg. x.dat.
- ▶ Read in as below (note expandtabs):

## Reading a spreadsheet

```
SAS> data x;  
SAS>   infile '/home/ken/x.dat' expandtabs;  
SAS>   input a b c;  
SAS>  
SAS> proc print;  
SAS>  
SAS> run;
```

Obs	a	b	c
1	1	4	7
2	2	5	8
3	3	6	9
4	10	11	12

Without expandtabs, get many incomprehensible error messages, or no data at all!

## Copying out of SAS

- ▶ Can copy text output into eg. Word doc. SAS tables get copied to Word tables, but *things can get misaligned*. Be prepared to do some re-formatting.
- ▶ For copying graphs, I need to:
  - ▶ right-click graph, Copy Image
  - ▶ Paste Special (paste as bitmap)Might be easier for you. Experiment.
- ▶ Copy and paste code from Code window. SAS code should be fixed-pitch font (eg. Courier) in your document.
- ▶ If all else fails, take screen shots (alt-PrintScreen), paste into Word doc as images.

## Using R

- ▶ Mimic above “analysis” using R.
- ▶ Run commands one at a time, see results right away.
- ▶ See *errors* right away too!
- ▶ Start up R Studio, go to Console window (bottom left). See > prompt: waiting for you. Try:

```
R> x=c(1,2,3,5,7)
```

- ▶ “Glue values together into list, and call it x”.
- ▶ No comment equals no error.

## Using R

- ▶ Mimic above “analysis” using R.
- ▶ Run commands one at a time, see results right away.
- ▶ See *errors* right away too!
- ▶ Start up R Studio, go to Console window (bottom left). See > prompt: waiting for you. Try:

```
R> x=c(1,2,3,5,7)
```

- ▶ “Glue values together into list, and call it x”.
- ▶ No comment equals no error.
- ▶ Display anything in R by entering its name:

```
R> x
```

```
[1] 1 2 3 5 7
```

- ▶ showing that x really does contain those values.

## Basic statistics

- ▶ Mean:

```
R> mean(x)
```

```
[1] 3.6
```

## Basic statistics

- ▶ Mean:

```
R> mean(x)
```

```
[1] 3.6
```

- ▶ SD:

```
R> sd(x)
```

```
[1] 2.408319
```

## Basic statistics

- ▶ Mean:

```
R> mean(x)
```

```
[1] 3.6
```

- ▶ SD:

```
R> sd(x)
```

```
[1] 2.408319
```

- ▶ Quartiles by function `quantile` (note spelling):

```
R> quantile(x)
```

0%	25%	50%	75%	100%
1	2	3	5	7

- ▶ Errors come out in red immediately. Output (results) in black.
- ▶ Command history: up and down arrows take you to all the commands you entered.

## Projects and R scripts

- ▶ Can use R from Console window, copy commands and output into Word.
- ▶ But better organization by using a Project and R script.
- ▶ **Project**: container for commands, data, stuff associated with one piece of work:
  - ▶ Project-Create Project.
  - ▶ Use current folder for project or create new one.
  - ▶ “Browse” to navigate to folder.
  - ▶ R Studio switches to new project.

## Projects and R scripts

- ▶ Can use R from Console window, copy commands and output into Word.
- ▶ But better organization by using a Project and R script.
- ▶ **Project:** container for commands, data, stuff associated with one piece of work:
  - ▶ Project-Create Project.
  - ▶ Use current folder for project or create new one.
  - ▶ “Browse” to navigate to folder.
  - ▶ R Studio switches to new project.
- ▶ **Script:** like string of commands fed into SAS, but more flexible.
  - ▶ File-New-R Script. Creates top left window for commands to use (and re-use).
  - ▶ File-Save as usual. No file extension needed (R Studio supplies one.)

## Projects and R scripts

- ▶ Can use R from Console window, copy commands and output into Word.
- ▶ But better organization by using a Project and R script.
- ▶ **Project:** container for commands, data, stuff associated with one piece of work:
  - ▶ Project-Create Project.
  - ▶ Use current folder for project or create new one.
  - ▶ “Browse” to navigate to folder.
  - ▶ R Studio switches to new project.
- ▶ **Script:** like string of commands fed into SAS, but more flexible.
  - ▶ File-New-R Script. Creates top left window for commands to use (and re-use).
  - ▶ File-Save as usual. No file extension needed (R Studio supplies one.)
  - ▶ To run:
    - ▶ “Source” runs everything.
    - ▶ “Run” (or Control-Enter) runs code on current line.
    - ▶ Select several lines: Run or Control-Enter runs selected lines.

## Projects and R scripts

- ▶ Can use R from Console window, copy commands and output into Word.
- ▶ But better organization by using a Project and R script.
- ▶ **Project:** container for commands, data, stuff associated with one piece of work:
  - ▶ Project-Create Project.
  - ▶ Use current folder for project or create new one.
  - ▶ “Browse” to navigate to folder.
  - ▶ R Studio switches to new project.
- ▶ **Script:** like string of commands fed into SAS, but more flexible.
  - ▶ File-New-R Script. Creates top left window for commands to use (and re-use).
  - ▶ File-Save as usual. No file extension needed (R Studio supplies one.)
  - ▶ To run:
    - ▶ “Source” runs everything.
    - ▶ “Run” (or Control-Enter) runs code on current line.
    - ▶ Select several lines: Run or Control-Enter runs selected lines.
  - ▶ Commands and output appear in Console window; copy-paste to a report.

## Reading data from a file

- ▶ “basic” way: one observation per line, values for all variables separated by whitespace. (Like SAS.)
- ▶ The top-left of R-studio also text editor.
- ▶ Create new data file with File, New, Text File
- ▶ Open existing data file, eg. `threegroups.txt` that we used with SAS.
- ▶ With R, put the variable names on the first line of the data file, like this. Saved as `threegroups.txt`:

```
group y
a 20
a 21
a 16
b 11
...
...
```

## Reading data in

- ▶ Tell R that first row is headers:

```
R> mydata=read.table("threegroups.txt",header=T)
```

```
R> mydata
```

	group	y
1	a	20
2	a	21
3	a	16
4	b	11
5	b	14
6	b	17
7	b	15
8	c	13
9	c	9
10	c	12
11	c	13

- ▶ If no headers, say header=F, R supplies column names.

## Data frames

- ▶ `mydata` holding data values called **data frame**: rectangular array with rows and columns, rows observations (individuals) and columns variables.
- ▶ Access variable like this:

```
R> mydata$y  
[1] 20 21 16 11 14 17 15 13 9 12 13
```

- ▶ Or like this:

```
R> attach(mydata)  
R> y  
[1] 20 21 16 11 14 17 15 13 9 12 13
```

- ▶ Logic: no variable `y`, but if attach a data frame, variables looked for there first. Here, `y` must be `mydata$y`.
- ▶ Problem with `attach`: many extra variables; “where did that `y` and `group` come from anyway?” (“polluting the name space”).
- ▶ When finished with attached variables: `detach(mydata)`, now no “extra” `y` or `group` any more.

## Means by group

- ▶ Not quite as easy as SAS:

```
R> aggregate(y~group,mydata,mean)
```

	group	y
1	a	19.00
2	b	14.25
3	c	11.75

- ▶ This works whether or not you attached the data frame.
- ▶ Three things:
  - ▶ “Model formula”: variable calculating for, squiggle, grouping variable.
  - ▶ Data frame containing those variables.
  - ▶ Thing to calculate.

## Means and other things by group

- ▶ IQR by group like this:

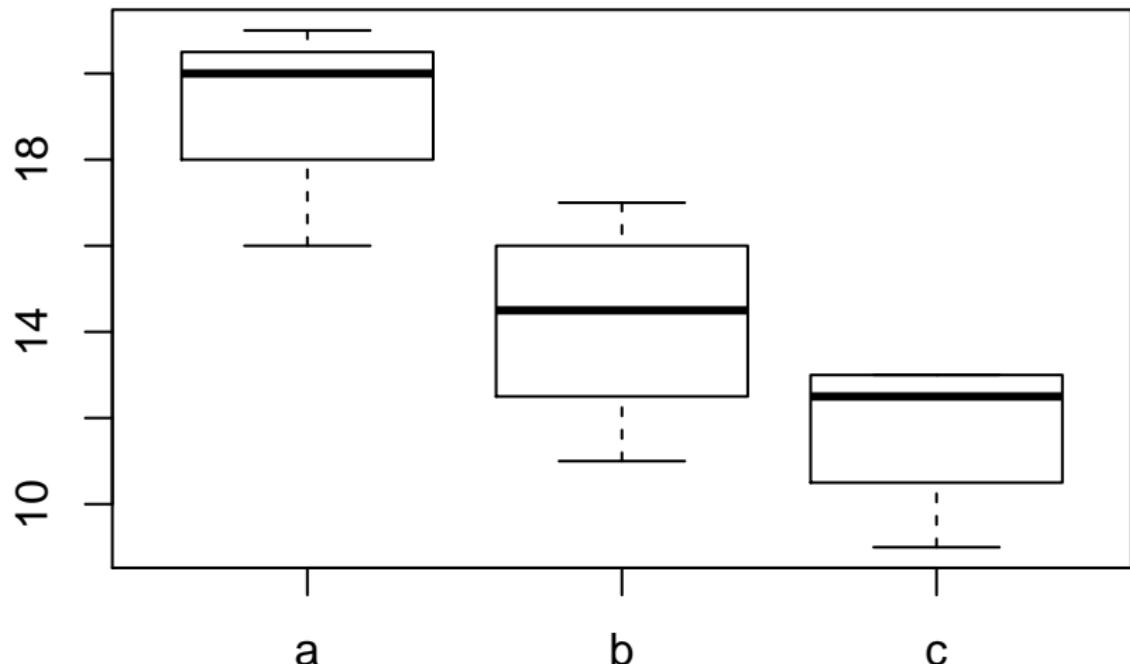
```
R> aggregate(y~group, data=mydata, IQR)
```

group	y
1	a 2.50
2	b 2.25
3	c 1.75

- ▶ Feed in calculation variable, grouping variable, data frame, thing to calculate.
- ▶ See model formula again in a few seconds when we draw a boxplot.

## Boxplot

```
R> boxplot(y~group,data=mydata)
```



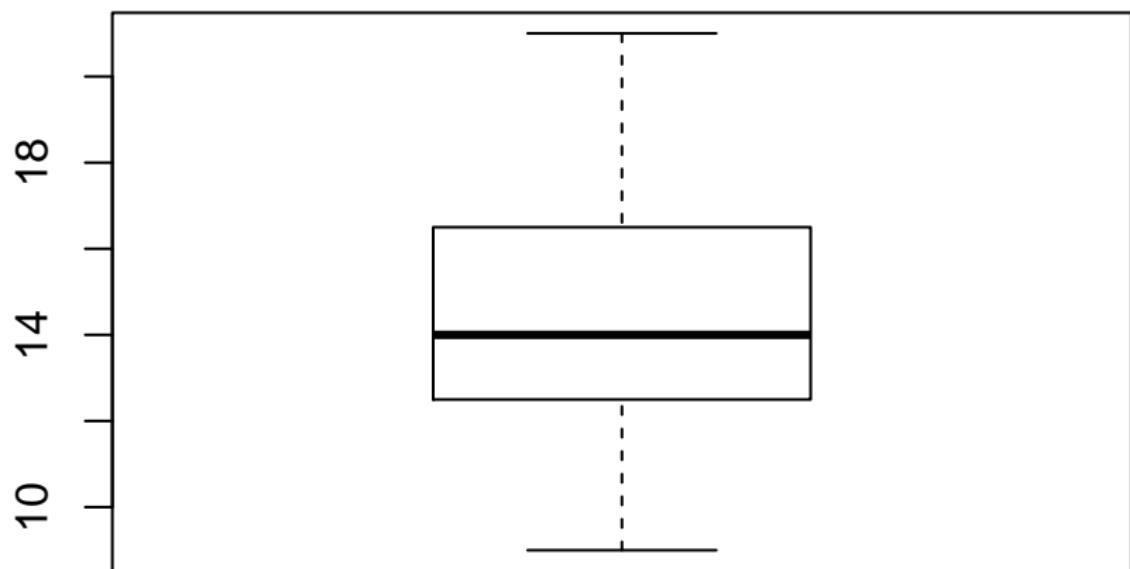
## Comments

- ▶ “Silly” boxplots with not much data.
- ▶ Different from SAS because different quartile definition.
- ▶ In R Studio, plot appears bottom right.
- ▶ Can omit grouping variable to get boxplot of all values.

## Another boxplot

To get boxplot of *all* values in *y*, not subdivided by group, do this:

```
R> boxplot(mydata$y)
```

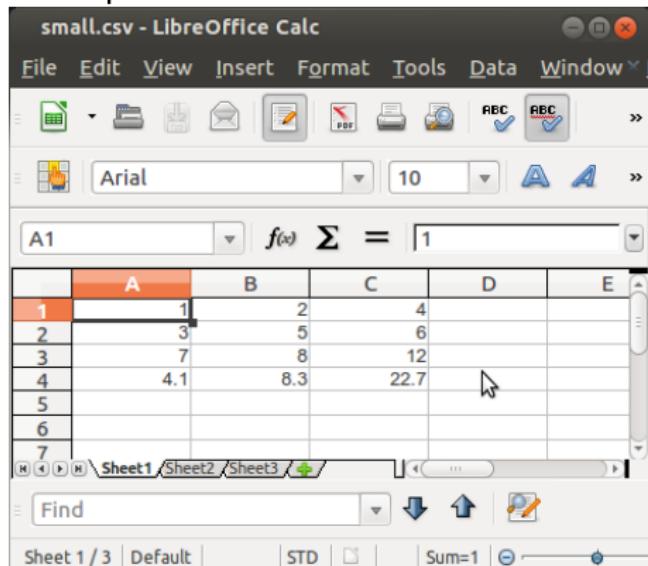


## Multiple graphs in R Studio

- ▶ If you made the last two boxplots in R Studio, second one came up on top of first one.
- ▶ Use arrows below Plots tab to cycle among your graphs.
- ▶ Limit of 30 graphs saved.

## Reading data from a spreadsheet

- ▶ Best way, for R: save data as .csv file (File, Save As)
- ▶ .csv saves values, not formulas.
- ▶ Example:



The screenshot shows a LibreOffice Calc spreadsheet window titled "small.csv - LibreOffice Calc". The menu bar includes File, Edit, View, Insert, Format, Tools, Data, and Window. The toolbar contains icons for new, open, save, print, and other functions. The formula bar shows "A1" selected, with "f(x)" and "Σ = 1" entered. The main grid has 6 rows and 5 columns. Row 1 contains column headers A through E. Rows 2 through 6 contain numerical data: Row 2 (B2, C2, D2) has values 3, 5, 6; Row 3 (B3, C3, D3) has values 7, 8, 12; Row 4 (B4, C4, D4) has values 4.1, 8.3, 22.7. The bottom status bar shows "Sheet 1 / 3 | Default | STD | Sum=1".

	A	B	C	D	E
1	1	2	4		
2	3	5	6		
3	7	8	12		
4	4.1	8.3	22.7		
5					
6					
7					

- ▶ Columns have no names.
- ▶ Save as small.csv in project folder.

## Reading into R

```
R> zz=read.csv("small.csv",header=F)
```

```
R> zz
```

	V1	V2	V3
1	1.0	2.0	4.0
2	3.0	5.0	6.0
3	7.0	8.0	12.0
4	4.1	8.3	22.7

- ▶ No column names; R supplied some. Can change.

```
R> mynames=c("Foo","Blah","Ding")
```

```
R> names(zz)=mynames
```

```
R> zz
```

	Foo	Blah	Ding
1	1.0	2.0	4.0
2	3.0	5.0	6.0
3	7.0	8.0	12.0
4	4.1	8.3	22.7

- ▶ Data frame now has supplied names.

## Reading .csv files into SAS

dlm short for “delimiter”:

```
SAS> data stuff;  
SAS>   infile '/home/ken/small.csv' dlm=', ';  
SAS>   input foo blah ding;  
SAS>  
SAS> proc print;
```

Obs	foo	blah	ding
1	1.0	2.0	4.0
2	3.0	5.0	6.0
3	7.0	8.0	12.0
4	4.1	8.3	22.7

## Alternatively in R

via copy and paste:

- ▶ Open new Text File in R Studio.
- ▶ Paste spreadsheet values into empty window.
- ▶ Save as eg. small.txt.
- ▶ Into R via `read.table`

```
R> fred=read.table("small.txt",header=F)
```

```
R> fred
```

	V1	V2	V3
1	1.0	2.0	4.0
2	3.0	5.0	6.0
3	7.0	8.0	12.0
4	4.1	8.3	22.7

Don't know whether pasting introduced tabs, but R handled it OK.

## read.table vs. read.csv

- ▶ `read.csv` simplified version of `read.table`, especially for reading `.csv` files. These are the same:

```
R> fred1=read.csv("small.txt",header=F)
```

and

```
R> fred2=read.table("small.txt",sep=",",header=F)
```

## Part IV

A data analysis example

## 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 ( $\text{mg}/\text{cm}^3$ ) measured at end.

## The data

- ▶ One response variable (bone density)
- ▶ One categorical explanatory variable (treatment group)
- ▶ Data file: name of group, bone density (separated by spaces).
- ▶ Analysis: exploratory (side by side boxplots, means by group)
- ▶ Inference: does bone density really depend on treatment (later).
- ▶ Two parallel analyses, start with SAS.

## Reading data

- ▶ Enter code below under Code tab in SAS Studio.
- ▶ First line of data file has variable names, so skip that (`firstobs=2`)
- ▶ `input` line has names of variables.
- ▶ `group` is text (not numbers) so tell SAS that with `$`.
- ▶ Grab the data file and copy it to a new data file. Rename it to `jumping.txt`.
- ▶ Use `/home/something/jumping.txt` on `infile` line.

```
SAS> data rats;  
SAS>   infile 'jumping.txt' firstobs=2;  
SAS>   input group $ density;
```

Submit these lines. No output, but check log window to make sure everything good.

## My log window

```
21  data rats;  
22  infile 'jumping.txt' firstobs=2;  
23  input group $ density;  
24  
25  run;
```

NOTE: The infile 'jumping.txt' is:

Filename=/home/ken/teaching/c32/notes/jumping.txt,  
Owner Name=ken, Group Name=ken,  
Access Permission=rw-rw-r--,  
Last Modified=Fri May 10 15:33:22 2013,  
File Size (bytes)=415

NOTE: 30 records were read from the infile 'jumping.txt'.

The minimum record length was 12.

The maximum record length was 13.

NOTE: The data set WORK.RATS has 30 observations and 2 variables.

NOTE: DATA statement used (Total process time):

real time 0.03 seconds

cpu time 0.00 seconds

## Checking the data

via proc print. Verify that data read in is as it should be.

SAS>	proc print;		15	Lowjump	599
Obs	group	density	16	Lowjump	632
1	Control	611	17	Lowjump	631
2	Control	621	18	Lowjump	588
3	Control	614	19	Lowjump	607
4	Control	593	20	Lowjump	596
5	Control	593	21	Highjump	650
6	Control	653	22	Highjump	622
7	Control	600	23	Highjump	626
8	Control	554	24	Highjump	626
9	Control	603	25	Highjump	631
10	Control	569	26	Highjump	622
11	Lowjump	635	27	Highjump	643
12	Lowjump	605	28	Highjump	674
13	Lowjump	638	29	Highjump	643
14	Lowjump	594	30	Highjump	650

## That data set

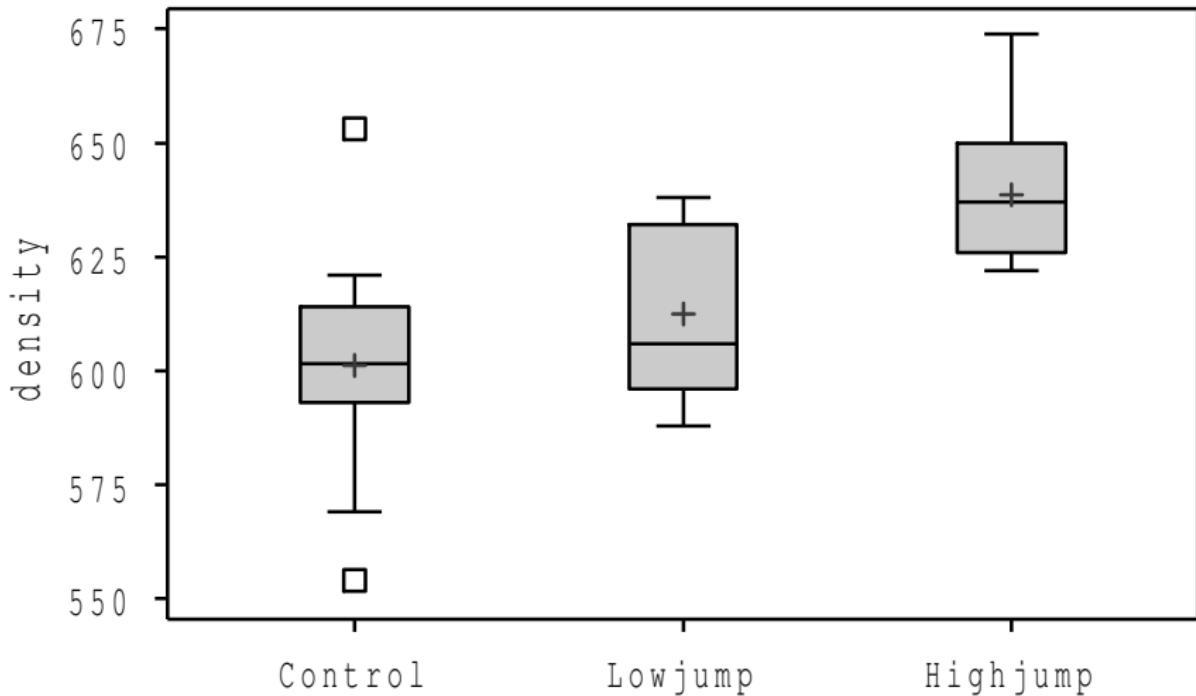
- ▶ Data set available until shut down SAS.
- ▶ Later see how to save as “permanent” data set, can use without needing data step.
- ▶ Check with original data file to make sure values correct.

## Make a graph

In this case, a side-by-side boxplot:

```
SAS> proc boxplot;
```

```
SAS> plot density*group / boxstyle=schematic;
```

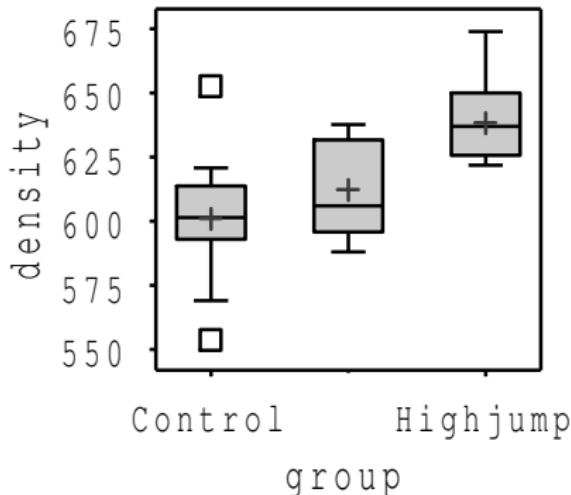


## What do we learn?

```
SAS> proc boxplot;
```

```
SAS> plot density*group /
```

```
SAS> boxstyle=schematic;
```



- ▶ Medians for control and lowjump groups similar.
- ▶ Median for highjump group higher.
- ▶ Similar for means (+ on SAS boxplots).
- ▶ lowjump has largest IQR
- ▶ lowjump has small IQR but big-time outliers
- ▶ so SD might tell different story than IQR
- ▶ highjump values skewed to right (longer upper whisker)
- ▶ control skewed, or not?

## Reading in the data with R

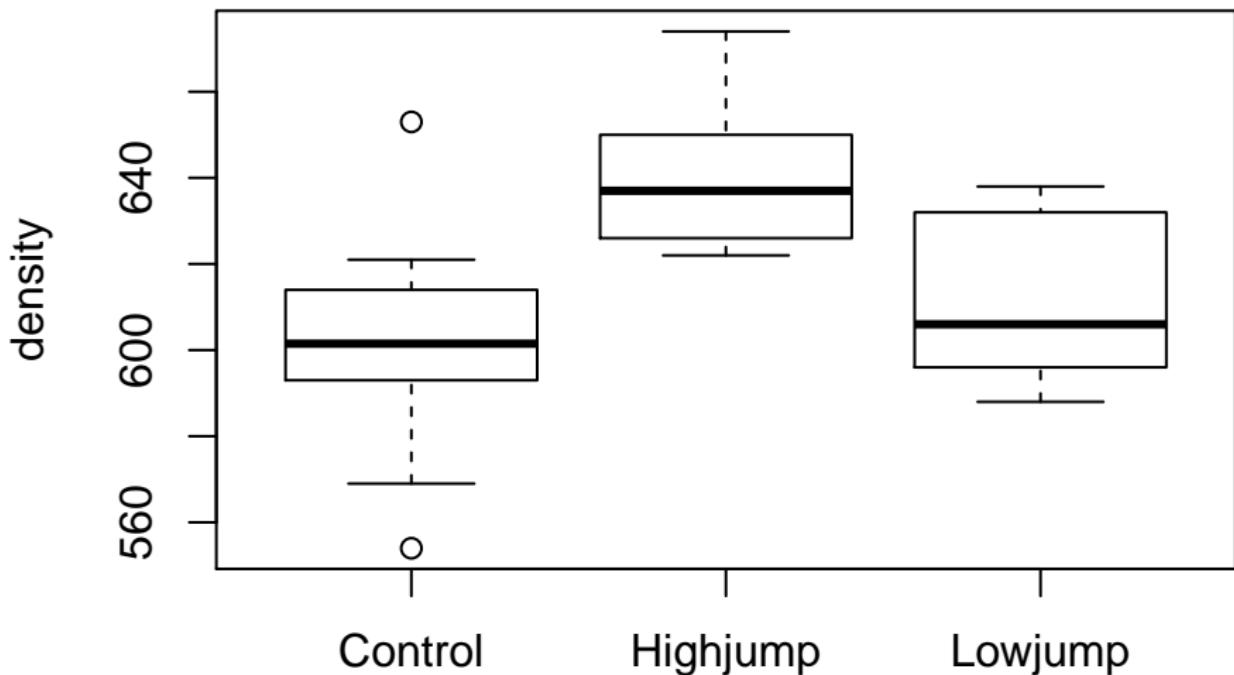
```
R> rats=read.table("jumping.txt",header=T)
```

```
R> rats
```

	group	density			
1	Control	611	15	Lowjump	599
2	Control	621	16	Lowjump	632
3	Control	614	17	Lowjump	631
4	Control	593	18	Lowjump	588
5	Control	593	19	Lowjump	607
6	Control	653	20	Lowjump	596
7	Control	600	21	Highjump	650
8	Control	554	22	Highjump	622
9	Control	603	23	Highjump	626
10	Control	569	24	Highjump	626
11	Lowjump	635	25	Highjump	631
12	Lowjump	605	26	Highjump	622
13	Lowjump	638	27	Highjump	643
14	Lowjump	594	28	Highjump	674
			29	Highjump	643
			30	Highjump	650

## Side-by-side boxplots with R

```
R> boxplot(density~group,data=rats,ylab="density")
```



## Comments

```
R> boxplot(density~group, data=rats, ylab="density")
```

- ▶ Use model formula.
- ▶ Specify data frame to get variables from
- ▶ Specify what to put on *y*-axis of boxplots.
- ▶ Groups came out *in alphabetical order*.
- ▶ Medians for control and lowjump similar, highjump higher.
- ▶ Variability picture confused: control IQR smallest, but outliers.
- ▶ highjump values skewed to right.

To make the groups come out in ascending order

```
R> attach(rats)
R> levels(group)
[1] "Control"  "Highjump" "Lowjump"
```

In alphabetical order. Order we want is

```
R> right.order=levels(group)[c(1,3,2)]
R> right.order
```

```
[1] "Control"  "Lowjump" "Highjump"
```

So we define *ordered factor* that respects the order given:

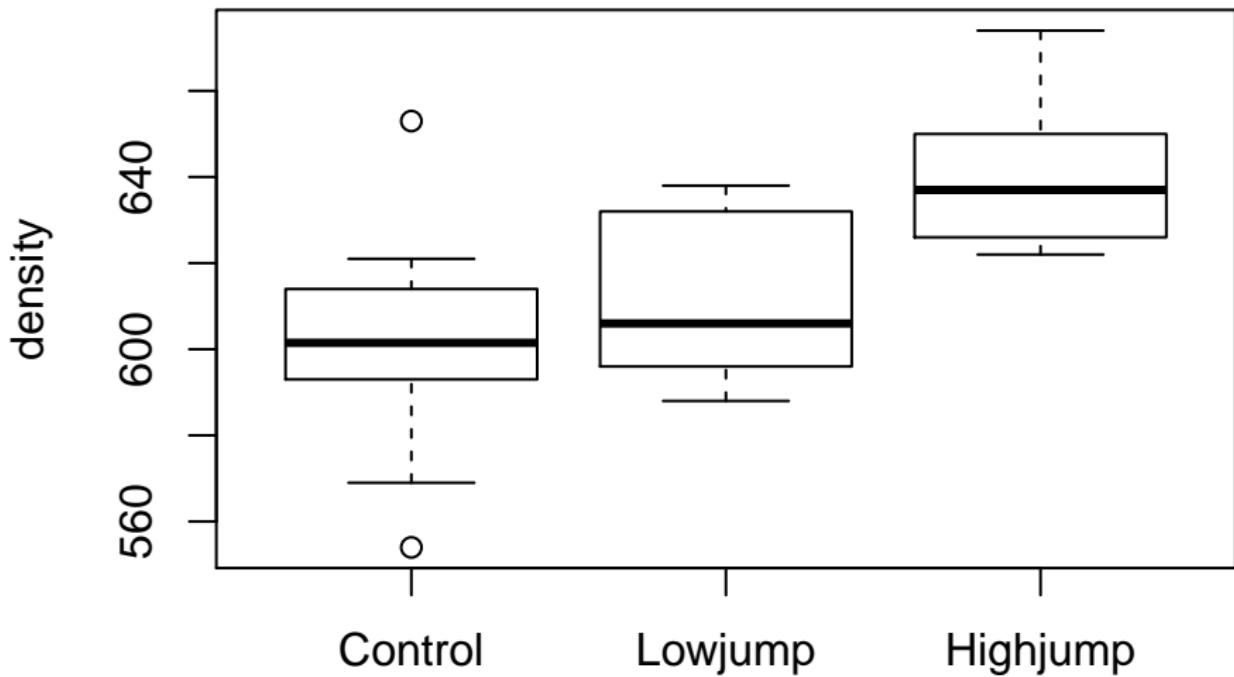
```
R> g2=ordered(group,levels=right.order)
R> levels(g2)
```

```
[1] "Control"  "Lowjump" "Highjump"
```

and try boxplot again:

```
R> boxplot(density~g2,data=rats,ylab="density")
R> detach(rats)
```

## Revised boxplot



## Means and SDs by group

SAS makes this easier:

```
SAS> proc means;  
SAS>   var density;  
SAS>   class group;
```

The MEANS Procedure

Analysis Variable : density					
	N	Obs	N	Mean	Std Dev
group					Minimum
Control	10	10	601.1000000	27.3636011	554.0000000
Highjump	10	10	638.7000000	16.5935061	622.0000000
Lowjump	10	10	612.5000000	19.3290225	588.0000000

Analysis Variable : density

	N	Obs	Maximum
group			
Control	10	653.0000000	
Highjump	10	674.0000000	
Lowjump	10	638.0000000	

## and in R

Way I like best:

```
R> aggregate(density~group,data=rats,mean)
```

	group	density
1	Control	601.1
2	Highjump	638.7
3	Lowjump	612.5

```
R> aggregate(density~group,data=rats,sd)
```

	group	density
1	Control	27.36360
2	Highjump	16.59351
3	Lowjump	19.32902

- ▶ highjump mean higher than for other groups.
- ▶ Bone density most variable for control — outliers.
- ▶ IQR was *smallest* for control.

## Means and selecting stuff

```
R> mean(rats$density)
```

```
[1] 617.4333
```

Mean bone density for *all* rats.

Find rats that are in control group:

```
R> attach(rats)
```

```
R> iscontrol=(group=="Control")
```

```
R> iscontrol
```

```
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE  
[13] FALSE  
[25] FALSE FALSE FALSE FALSE FALSE FALSE
```

`group=="Control"` is *logical* statement: for each rat, is true or false (get 30 answers).

## Bone densities for control group and their mean

```
R> density[iscontrol]
```

```
[1] 611 621 614 593 593 653 600 554 603 569
```

In English, “the bone densities of the rats for which group is control”. This is an ordinary vector, so find its mean in the usual way:

```
R> mean(density[iscontrol])
```

```
[1] 601.1
```

## Selecting rows/columns of a data frame

Select rows of the `rats` data frame for rats that were in the control group.  
Problem: data frames have rows *and* columns. Here's row 5 (rat 5) and column 2 (density):

```
R> rats[5, 2]
```

```
[1] 593
```

To select *all* of a row or column, leave index blank (but include comma):

```
R> rats[5, ]
```

	group	density
5	Control	593

## Selecting all the columns for rats in control group

and thus to select all the columns for those rats in the control group:

```
R> rats[iscontrol,]
```

	group	density
1	Control	611
2	Control	621
3	Control	614
4	Control	593
5	Control	593
6	Control	653
7	Control	600
8	Control	554
9	Control	603
10	Control	569

## Old-fashioned programmer's way of finding group means

- ▶ “What groups are there?”

```
R> levels(group)  
[1] "Control"  "Highjump" "Lowjump"
```

- ▶ Loop over them:

```
R> for (i in levels(group))  
R>   {  
R>     is.group=(group==i)  
R>     d=density[is.group]  
R>     print(c(i,mean(d)))  
R>   }
```

```
[1] "Control"  "601.1"  
[1] "Highjump" "638.7"  
[1] "Lowjump"  "612.5"
```

## = and ==

You might have been confused by = and == appearing on the same line of code. They look almost the same, but they have very different meanings:

- ▶ A double == means “take the things on each side, make it TRUE if they’re equal and FALSE if they’re not”.

```
R> 6-4==2
```

```
[1] TRUE
```

- ▶ A single = means “work out the thing on the right, and save it in the variable named on the left”.

```
R> c(1,2,3)==c(2,2,3)
```

```
[1] FALSE TRUE TRUE
```

- ▶ Both of these get used together sometimes. This is legit, though I’d rather write it as `x=(1:3==2)` for clarity.

```
R> 1:3==2
```

```
[1] FALSE TRUE FALSE
```

```
R> x=2+3 ; x
```

```
[1] 5
```

```
R> x=1:3==2 ; x
```

```
[1] FALSE TRUE FALSE 88 / 653
```

## Mean bone density of highjump group

Short and scary:

```
R> mean(density[group=="Highjump"])
[1] 638.7
```

Or, same thing, longer and less scary:

```
R> ishigh=(group=="Highjump")
R> highdens=density[ishigh]
R> mean(highdens)
[1] 638.7
```

I like to

- ▶ first get a logical vector that picks out whatever it is I want to select.
- ▶ pick out what I want to calculate on
- ▶ do the calculation.

Tidy up:

```
R> detach(rats)
```

## Next steps

- ▶ What is research question?
- ▶ “Is it *really* true that amount of jumping related to bone density”?
- ▶ or, “are the differences we observed just chance?”
- ▶ This requires **statistical inference**.

# Part V

## Statistical 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?

## 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*.

## Setting up the study

- ▶ Response variable: eg. score from standard reading test.
- ▶ Could: give all kids in our study our reading program, and see how well they did.
- ▶ But: kids in this study might get more attention than normal, encouraging them to do better.
- ▶ So should compare against “placebo”: usual reading program, but done under conditions of this study.
- ▶ Comparing like with like.

## Setting up the study

- ▶ Response variable: eg. score from standard reading test.
- ▶ Could: give all kids in our study our reading program, and see how well they did.
- ▶ But: kids in this study might get more attention than normal, encouraging them to do better.
- ▶ So should compare against “placebo”: usual reading program, but done under conditions of this study.
- ▶ Comparing like with like.
- ▶ Decide which children get new reading program: *randomize* .
  - ▶ If eg. give new program to kids needing it most, would make new program look worse than it really was: **bias**.

## Doing randomization

- ▶ Suppose we have 44 children available. Label them, eg. by numbering them 1 through 44.
- ▶ Randomly choose 22 to get new reading program:

```
R> s=sample(1:44, 22)
```

```
R> sort(s)
```

```
[1] 2 5 6 8 9 10 13 14 15 16 21 22 23 24 26 28 30 38 39 41 42 44
```

- ▶ Or, sample all 44, which shuffles them:

```
R> sample(1:44, 44)
```

```
[1] 6 28 38 25 32 40 9 30 19 7 39 14 15 26 36 29 2 24 23 27 17 34  
[26] 35 11 43 13 18 31 1 22 3 20 4 16 33 42 12 41 10 21 8
```

Take first 22 to get new reading program.

## Selecting more than 2 groups

- ▶ Use shuffle idea again, eg. jumping rats. Control group:

```
R> s=sample(1:30, 30)
```

```
R> s[1:10]
```

```
[1] 6 29 26 19 25 3 22 16 21 28
```

- ▶ Low jump group:

```
R> s[11:20]
```

```
[1] 30 20 12 23 7 17 4 11 1 9
```

- ▶ High jump group:

```
R> s[21:30]
```

```
[1] 2 8 27 10 18 14 15 13 5 24
```

## Why are the groups the same size?

- ▶ Want best chance of finding any differences among means that exist.  
*(Power of test.)*
- ▶ Population means must be as well estimated as possible.
- ▶ Ensure that no mean is too badly estimated.
- ▶ Make smallest group as big as possible.
- ▶ That is, make all groups as near as possible same size.

## Back to the kids

- ▶ No pairing, matching: try *two-sample t-test*.
- ▶ Data in file drp.txt with header line, group then reading test score.
- ▶ For test, need approx. normality, but don't need equal variability.
- ▶ Use SAS:

```
SAS> data kids;  
SAS>   infile 'drp.txt' firstobs=2;  
SAS>   input group $ score;
```

## Comparing means

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

The MEANS Procedure

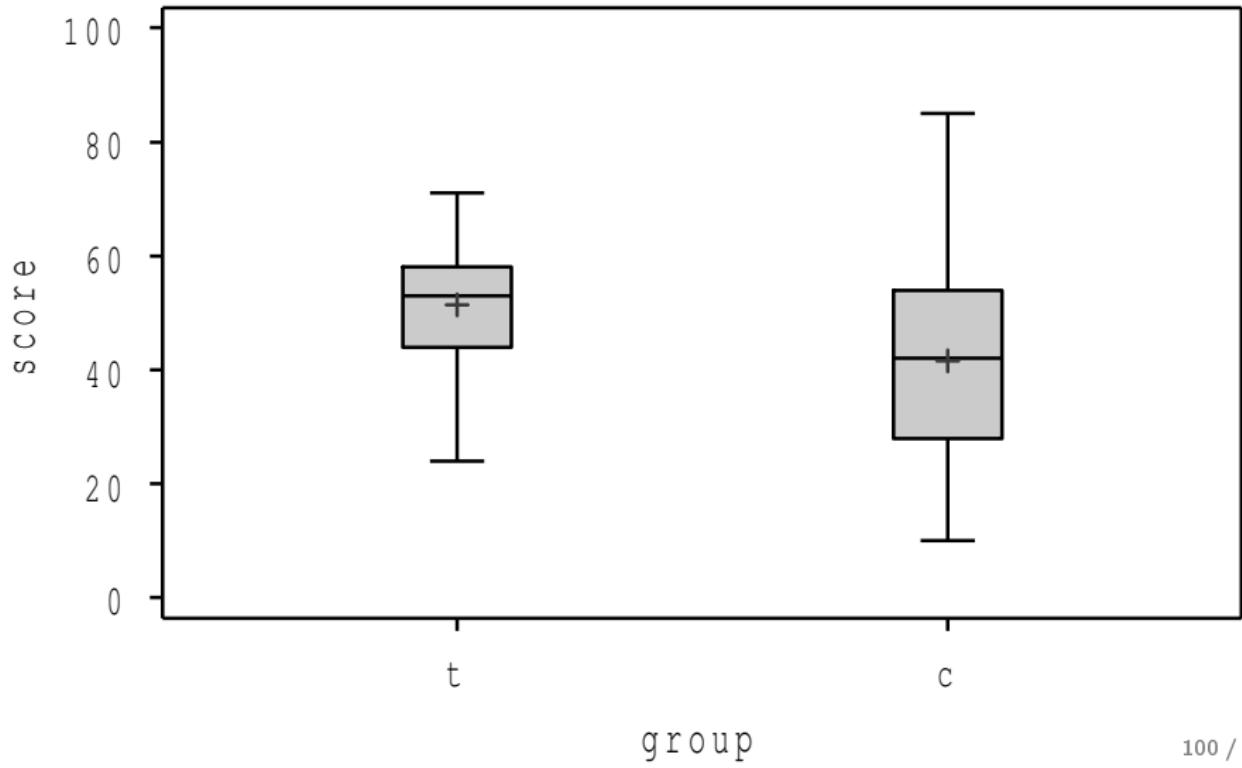
Analysis Variable : score						
	N					
group	Obs	N	Mean	Std Dev	Minimum	Maximum
<hr/>						
c	23	23	41.5217391	17.1487332	10.0000000	85.0000000
t	21	21	51.4761905	11.0073568	24.0000000	71.0000000
<hr/>						

Analysis Variable : score		
	N	
group	Obs	Maximum
<hr/>		
c	23	85.0000000
t	21	71.0000000
<hr/>		

## Boxplots

```
SAS> proc boxplot;
```

```
SAS> plot score*group / boxstyle=schematic;
```



## 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.

## The *t*-test

```
SAS> proc ttest side=l;  
SAS>   var score;  
SAS>   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 1 (lower), u (upper) or 2 (two-sided, can be omitted.) This is 1 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.
- ▶ Cannot 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=1`.

## Doing it with R

- ▶ I skip the boxplots (you do it).
- ▶ c (control) before t (treatment) alphabetically, so proper alternative is "less".
- ▶ R does Satterthwaite test by default:

```
R> kids=read.table("drp.txt",header=T)
R> 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
```

- ▶ As before: new reading program really helps.

## The pooled $t$ -test

- ▶ Pooled (equal variances) test this way:

```
R> t.test(score~group,data=kids,alternative="less",
R>       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:

```
R> 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 (more on that later).
- ▶ New reading program increases average scores by somewhere between about 1 and 19 points.

## Randomization test

- ▶  $t$ -test assumes normality (though often OK without).
- ▶ Different approach. Suppose reading method has no effect.
- ▶ Then each observation might have come from either treatment or control.
- ▶ We observed in data difference in means of 10.
- ▶ If we assign Treatment and Control to the observations at random, how likely would we be to see a difference in means of 10 or bigger?
- ▶ Randomization idea: if null hypothesis true, how could we randomly assign the observations we had to groups?

## Randomization test in R

- ▶ Since we are designing this ourselves, R is tool to use.
- ▶ Ingredients:
  - ▶ sample to shuffle treatment groups
  - ▶ aggregate to calculate means for shuffled groups
  - ▶ little bit of calculation to get difference in sample means.

```
R> attach(kids)
R> myshuffle=sample(group,length(group))
R> myshuffle
[1] t t c c c t t t t c c c c c c c t c t t t c t c c t
[39] c t t t t c
Levels: c t
```

- ▶ This makes shuffled groups.

## Randomization difference in means

- ▶ Now find group means:

```
R> themeans=aggregate(score~myshuffle,data=kids,mean)
R> themeans
```

myshuffle	score
1	c 47.13043
2	t 45.33333

- ▶ Difference in means is second thing in second column minus first thing in second column:

```
R> meandiff=themeans[2,2]-themeans[1,2]
R> meandiff
[1] -1.797101
```

- ▶ Simulated treatment < simulated control.

# Again!

```
R> myshuffle=sample(group,length(group))
R> themeans=aggregate(score~myshuffle,data=kids,mean)
R> meandiff=themeans[2,2]-themeans[1,2]
R> meandiff
[1] -1.797101
```

Exactly the same as before! Bizarre.  
(Correct, though: the two `myshuffles` are different.)

## One more time...

```
R> myshuffle=sample(group,length(group))
R> themeans=aggregate(score~myshuffle,data=kids,mean)
R> meandiff=themeans[2,2]-themeans[1,2]
R> meandiff
[1] -0.7950311
```

Difference in means still negative, but closer to zero.

Looks like a difference in means of 10 is improbably large, by chance.  
Do this, say, 1000 times?

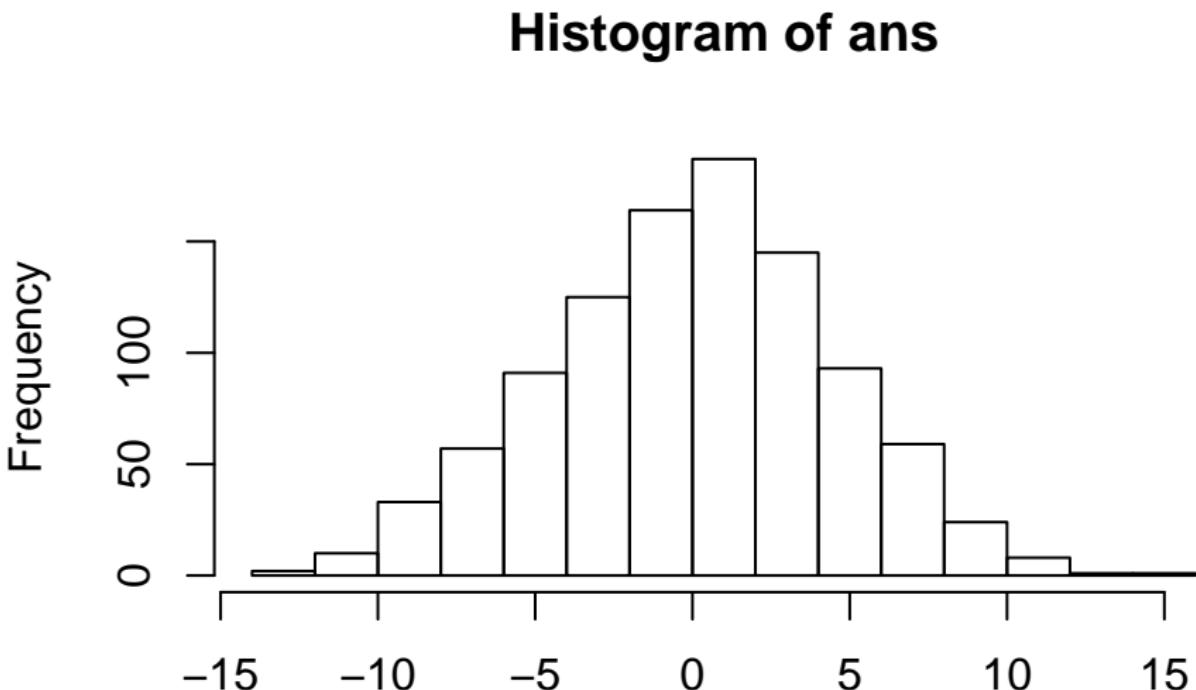
# 1000 times

```
R> nsim=1000
R> ans=numeric(nsim)
R> for (i in 1:nsim)
R> {
R>   myshuffle=sample(group,
R>                     length(group))
R>   themeans=aggregate(score~myshuffle,
R>                      data=kids,mean)
R>   ans[i]=themeans[2,2]-themeans[1,2]
R> }
```

- ▶ Do 1000 simulations.
- ▶ Initialize `ans` to contain 1000 “empty numbers”.
- ▶ Loop:  $i = 1, 2, \dots, 1000$  in turn.
- ▶ Inside loop: do randomization  $i$  as above, but store results in  $i$ -th element of `ans`.
- ▶ Investigate randomization by looking at `ans`.

## Histogram of randomized mean differences

R> hist(ans)



## How many randomizations gave mean difference > 10?

- ▶ By chance, mean difference of 10 or bigger appears unlikely.
- ▶ Use a logical vector to pick out the elements of `ans` that we want, then make a table of them:

```
R> isbig=(ans>=10);  
R> table(isbig)
```

isbig	
FALSE	TRUE
990	10

- ▶ 10 out of 1000.
- ▶ Our randomization test gives a P-value of  $10/1000 = 0.01$ , so would reject null hypothesis “new reading program has no effect” in favour of alternative “new reading program helps”.
- ▶ P-value similar to 0.013 from two-sample *t*.

## Why 1000 randomizations?

- ▶ No uniquely good answer.
- ▶ More randomizations is better.
- ▶ But didn't want to wait too long for it to finish!
- ▶ If you think 10,000 (or a million) better, replace `nsim` in my code by desired value and run again.

## Jargon for testing

**Alternative hypothesis:** what we are trying to prove (new reading program is effective).

**Null hypothesis:** “there is no difference” (new reading program no better than current program). *Must contain “equals”.*

**One-sided alternative:** trying to prove *better* (as with reading program).

**Two-sided alternative:** trying to prove *different*.

**Test statistic:** something expressing difference between data and null (eg. difference in sample means, *t* statistic).

**P-value:** probability of observing test statistic value *as extreme or more extreme, if null is true.*

## 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 by  $t$  or randomization distribution.
- ▶ Mean reading score was 10 higher for new program.
- ▶ Difference of 10 was unusually big (P-value small from  $t$ -test and randomization 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:

		Decision	
Truth	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  $\alpha$ . Usually fix  $\alpha$ , eg.  $\alpha = 0.05$ .
- ▶ Prob. of type II error denoted  $\beta$ . Determined by the planned experiment. Low  $\beta$  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  $\theta$ .
- ▶ Suppose  $H_0$  wrong. What does that say about  $\theta$ ?
- ▶ 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  $\theta$  is:
  - ▶ If  $\theta = 496$ , should be able to reject  $H_0 : \theta = 10$  even for small sample, so  $\beta$  should be small (power large).
  - ▶ If  $\theta = 11$ , might have hard time rejecting  $H_0$  even with large sample, so  $\beta$  would be larger (power smaller).
- ▶ *Power depends on true parameter value, and on sample size.*
- ▶ So we play “what if”: “if  $\theta$  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.

## Calculating power for a *t*-test

Think about reading programs again. Need to specify:

- ▶ What kind of *t*-test (here 2-sample, not 1-sample or paired)
- ▶ 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 (here 5 points).

Using  $n = 22$  kids in each group (has to be same), we get this :-(

```
R> power.t.test(n=22, delta=5, sd=15, type="two.sample",
R>           alternative="one.sided")
```

Two-sample t test power calculation

```
      n = 22
      delta = 5
      sd = 15
sig.level = 0.05
power = 0.2887273
alternative = one.sided
```

## Detecting a larger difference

Larger difference, say 10 points under same conditions, should be easier to detect. Change delta=5 to delta=10:

```
R> power.t.test(n=22,delta=10,sd=15,type="two.sample",
R> alternative="one.sided")
```

Two-sample t test power calculation

```
      n = 22
      delta = 10
      sd = 15
sig.level = 0.05
      power = 0.7020779
alternative = one.sided
```

NOTE: n is number in \*each\* group

Much better, power around 70%. Much likelier to detect this bigger difference.

## Increasing the sample size

Making  $n$  bigger should improve power. For detecting difference of 5, increase sample size from 22 in each group to 40:

```
R> power.t.test(n=40,delta=5,sd=15,type="two.sample",alterna
```

```
Two-sample t test power calculation
```

```
    n = 40
    delta = 5
    sd = 15
    sig.level = 0.05
    power = 0.4336523
    alternative = one.sided
```

NOTE: n is number in \*each\* group

Power went up from 29% to 43%. Often need *much* bigger sample size to make much difference.

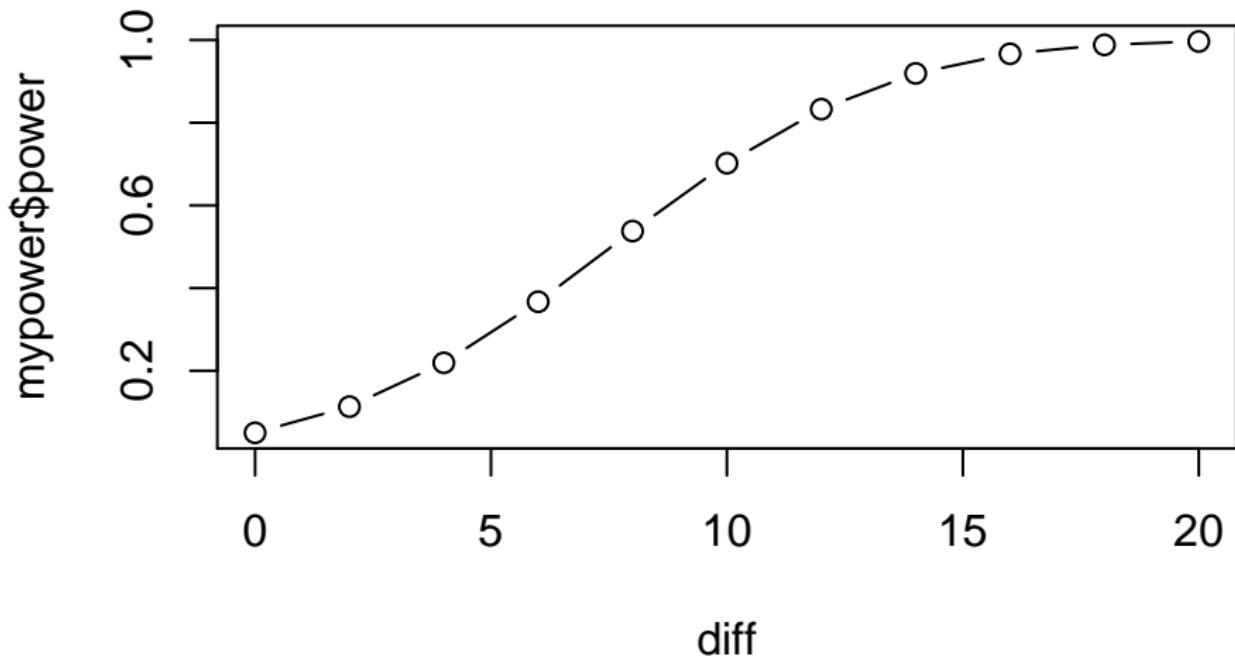
## Making a graph of power

- ▶ “Power curve”.
- ▶ Two ways:
  - ▶ Keep sample size fixed, change true difference between means
  - ▶ Keep difference fixed, change sample size.
- ▶ Feed `power.t.test` a vector of true differences or sample sizes. Get back vector of power values:

```
R> my.del=seq(0,20,2)
R> mypower=power.t.test(n=22,delta=my.del,sd=15,
R>   type="two.sample",alternative="one.sided")
R> mypower$power
[1] 0.0500000 0.1131924 0.2192775 0.3670836 0.5380092 0.7020777
[8] 0.9192728 0.9667502 0.9883915 0.9965808
```

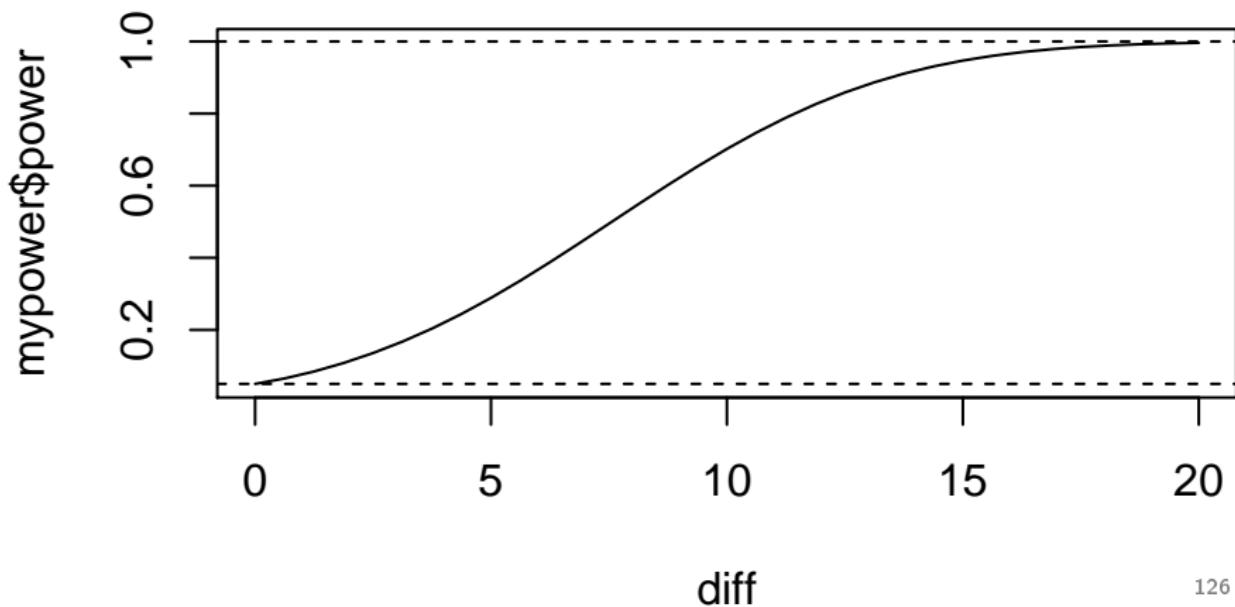
## The power curve

```
R> plot(my.del,mypower$power,type="b",xlab="diff")
```



## Another look at the power curve

```
R> plot(my.del,mypower$power,type="n",xlab="diff")
R> lines(spline(my.del,mypower$power))
R> abline(h=0.05,lty="dashed")
R> abline(h=1,lty="dashed")
```



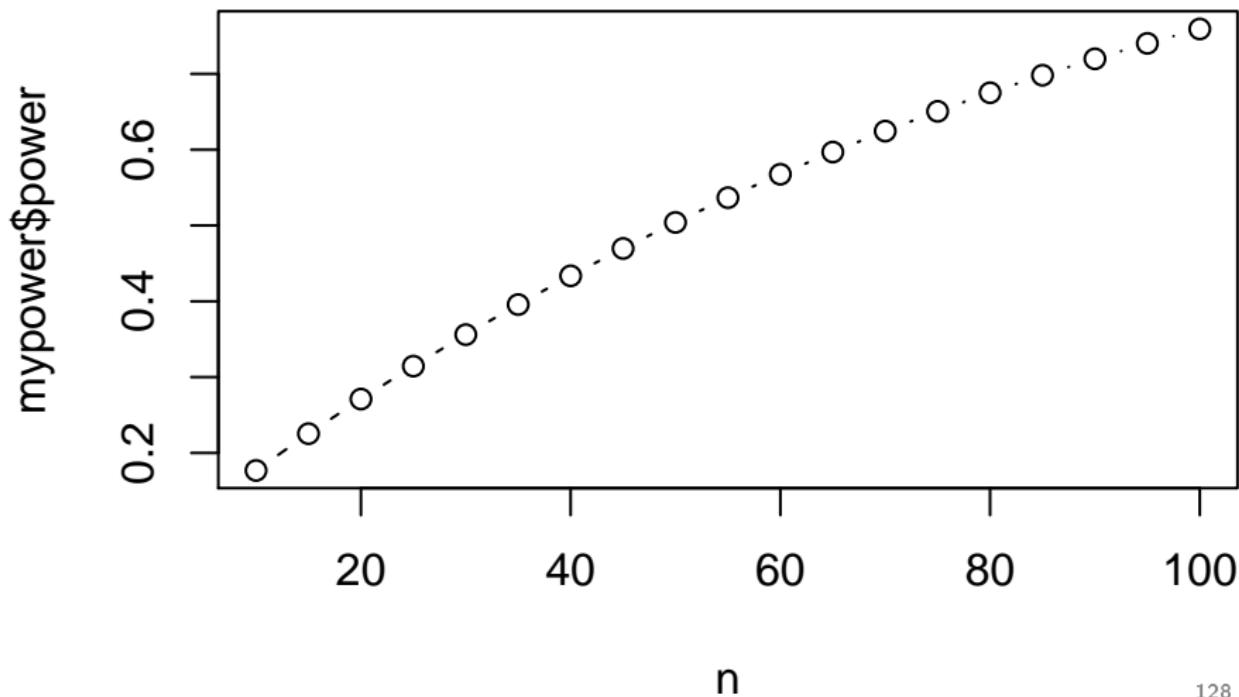
## Power curve for sample size, fixed difference of 5

Sample sizes, 10 to 100 in steps of 5:

```
R> my.n=seq(10,100,5)
R> mypower=power.t.test(n=my.n,delta=5,sd=15,
R>   type="two.sample",alternative="one.sided")
R> mypower$power
[1] 0.1768868 0.2254334 0.2710950 0.3145661 0.3560927 0.3957733 0.4344538
[8] 0.4697559 0.5041065 0.5367294 0.5676549 0.5969194 0.6245648 0.6524857
[15] 0.6751905 0.6982766 0.7199535 0.7402799 0.7593159
```

## The power curve

```
R> plot(my.n,mypower$power,type="b",xlab="n")
```



## Finding sample size

- ▶ Instead of being given a sample size and finding a power, we can be given a power and asked for the sample size needed to achieve it.
- ▶ Eg. to detect a difference of 5 between means, how big a sample do I need to get 80% power?
- ▶ Leave out n= and put in power=0.80 instead; solves for n:

```
R> power.t.test(power=0.80,delta=5,sd=15,type="two.sample",
R>   alternative="one.sided")
```

Two-sample t test power calculation

```
      n = 111.9686
      delta = 5
      sd = 15
      sig.level = 0.05
      power = 0.8
  alternative = one.sided
```

NOTE: n is number in \*each\* group

## Comments

- ▶ To have 80% chance of correctly detecting difference of 5 in means, need 112 children in each group.
- ▶ Sample size calculations apt to give depressing results!

# SAS

SAS has proc power which does all the same stuff (a little more flexibly):

- ▶ Two-sample  $t$  for means: twosamplemeans.
- ▶ Satterthwaite test: diff\_satt
- ▶ One-sided.
- ▶ Desired difference in means 5.
- ▶ SD in each group 15
- ▶ Total sample size 44 (22 in each group)
- ▶ Want to find power for this: power=..

## SAS code

```
SAS> proc power;  
SAS>      twosamplemeans  
SAS>      test=diff_satt  
SAS>      sides=1  
SAS>      meandiff=5  
SAS>      stddev=15  
SAS>      ntotal=44  
SAS>      power=.;  
SAS>  
SAS> run;
```

Note punctuation: twosamplemeans to power really all one line, so semicolon only after power=..

## SAS output

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

Standard Deviation                   15

Total Sample Size                   44

Null Difference                       0

Nominal Alpha                         0.05

Group 1 Weight                       1

Group 2 Weight                       1

Computed Power

Actual

Alpha      Power

0.0498     0.288

Power is 0.29, as with R. Note that we didn't need any data.

## Power for several alternatives and sample sizes

If you put several values on a line, SAS does all combinations of them, eg. mean differences 5 and 10, sample size 22 and 40 per group:

```
SAS> proc power;
SAS> two-samplemeans
SAS>   test=diff_satt
SAS>   sides=1
SAS>   meandiff=5 10
SAS>   stddev=15
SAS>   ntotal=44 80
SAS>   power=.;
```

# Output

The POWER Procedure

Two-Sample t Test for Mean Difference with Unequal Variances

Fixed Scenario Elements

Distribution                                   Normal

Method    Exact

Number of Sides                               1

Standard Deviation                           15

Null Difference                               0

Nominal Alpha                                0.05

Group 1 Weight                               1

Group 2 Weight                               1

Computed Power

	Mean	N	Actual	
Index	Diff	Total	Alpha	Power
1	5	44	0.0498	0.288
2	5	80	0.0499	0.433
3	10	44	0.0498	0.701
4	10	80	0.0499	0.905

## Comments

- ▶ Consistent with R results before.
- ▶ Increasing sample size increases power a bit.
- ▶ Increasing true mean difference increases power dramatically.

## Sample size calculation

To calculate sample size required, leave ntotal blank and fill in power.  
Here, mean difference 5, desired power 0.80:

```
SAS> proc power;  
SAS>   twosamplemeans  
SAS>     test=diff_satt  
SAS>     sides=1  
SAS>     meandiff=5  
SAS>     stddev=15  
SAS>     ntotal=.  
SAS>     power=0.80;
```

## 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

Standard Deviation                   15

Nominal Power                         0.8

Null Difference                       0

Nominal Alpha                         0.05

Group 1 Weight                       1

Group 2 Weight                       1

Computed N Total

Actual      Actual      N

Alpha      Power      Total

0.05      0.800      224

Need 224 children *total*, or 112 in each group. As R.

## 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.

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

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

## Results

Power for 22 in each group was 29%:

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
Standard Deviation	15
Group 1 Sample Size	10
Group 2 Sample Size	34
Null Difference	0
Nominal Alpha	0.05

Computed Power

Actual

Alpha	Power
0.0505	0.225

Unequal sample sizes bring power down to 22.5%.

## Confidence intervals

- ▶ Hypothesis test: “could my parameter be *this*”, where you give value for *this*.
  - ▶ In reading example, *this* was null-hypothesis mean difference of 0. From data, concluded that *this* not zero, so new reading program was effective.
- ▶ Confidence interval: “what could my parameter be?”.
  - ▶ “how much difference in mean score between reading programs?”
- ▶ As with testing, uncertainty involved.
- ▶ “I think parameter between *this* and *that*.”
- ▶ Always possible to be wrong. Make compromise, eg. 95% CI: in 95% of all possible samples, confidence interval statement will be true about parameter. (And only 5% chance of being wrong.)
- ▶ Higher confidence gives more chance of being right, but in exchange, gives longer, less informative interval.
- ▶ Confidence intervals are two-sided things, so correspond with *two-sided* test (defaults in R and SAS).

## CI in SAS

```
SAS> proc ttest;  
SAS>   var score;  
SAS>   class group;
```

The TTEST Procedure

Variable: score

group	Method	Mean	95% CL	Mean	Std Dev
c		41.5217	34.1061	48.9374	17.1487
t		51.4762	46.4657	56.4867	11.0074
Diff (1-2)	Pooled	-9.9545	-18.8176	-1.0913	14.5512
Diff (1-2)	Satterthwaite	-9.9545	-18.6759	-1.2330	
group	Method	95% CL	Std Dev		
c		13.2627	24.2715		
t		8.4213	15.8954		
Diff (1-2)	Pooled	11.9981	18.4947		
Diff (1-2)	Satterthwaite				

## Comments

- ▶ Same code as for test, look at different part of output.
- ▶ Look under 95% CL mean and Satterthwaite line, interval from  $-18.68$  to  $-1.23$ .
- ▶ Control minus treatment, so new reading method better on average by between 1 and 19 points.
- ▶ Would a 90% CI be longer or shorter than a 95% one? Code next page. Put appropriate alpha on proc ttest line.

## 90% CI

```
SAS> proc ttest alpha=0.10;  
SAS>   var score;  
SAS>   class group;
```

The TTEST Procedure

Variable: score

group	Method	Mean	90% CL	Mean	Std Dev
c		41.5217	35.3816	47.6618	17.1487
t		51.4762	47.3334	55.6190	11.0074
Diff (1-2)	Pooled	-9.9545	-17.3414	-2.5675	14.5512
Diff (1-2)	Satterthwaite	-9.9545	-17.2176	-2.6913	
group	Method	90% CL	Std Dev		
c		13.8098	22.8992		
t		8.7834	14.9440		
Diff (1-2)	Pooled	12.3693	17.7758		
Diff (1-2)	Satterthwaite				

## Comparison

- ▶ 95%:  $-18.68$  to  $-1.23$ .
- ▶ 90%:  $-17.22$  to  $-2.69$ .
- ▶ 90% CI shorter, but has more chance to be wrong.
- ▶ Both CIs wide: we don't know much about effect that new reading method has. (Solution: more data.)

## Same thing in R

Default is 95% CI, get via `t.test` as before:

```
R> 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
95 percent confidence interval:
-18.67588 -1.23302
sample estimates:
mean in group c mean in group t
41.52174          51.47619
```

Same as SAS.

## 90% CI in R

Add `conf.level`, which is actual confidence level, unlike SAS:

```
R> t.test(score~group,data=kids,conf.level=0.90)
```

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

90 percent confidence interval:

-17.217609 -2.691293

sample estimates:

mean in group c mean in group t

41.52174 51.47619

Again, same as SAS.

## 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 (R):

```
R> group1=c(10,11,11,13,13,14,14,15,16)
R> group2=c(13,13,14,17,18,19)
R> t.test(group1,group2)
```

Welch Two Sample t-test

```
data: group1 and group2
t = -2.0937, df = 8.71, p-value = 0.0668
alternative hypothesis: true difference in means is not equal to
95 percent confidence interval:
-5.5625675  0.2292342
sample estimates:
mean of x mean of y
13.00000 15.66667
```

## 90% CI

```
R> t.test(group1,group2,conf.level=0.90)
```

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

```
data: group1 and group2
```

```
t = -2.0937, df = 8.71, p-value = 0.0668
```

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

```
90 percent confidence interval:
```

```
-5.010308 -0.323025
```

```
sample estimates:
```

```
mean of x mean of y
```

```
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  $0.2$ , includes 0.
- ▶ 90% CI from  $-5.0$  to  $-0.3$ , does not include 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. Following *always* true.

- ▶ Reject  $H_0$  at level  $\alpha$  if and only if  $H_0$  value *outside* corresponding CI.
- ▶ Fail to reject  $H_0$  at level  $\alpha$  if and only if  $H_0$  value *inside* corresponding CI.
- ▶ Idea: “Plausible” parameter value inside CI, not rejected; “Implausible” parameter value outside CI, rejected.

See my lecture notes for how to do this in SAS. (Some trickery involved because of the way I laid out the data.)

## 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  $\alpha = 0.05$  for a 95% CI,  $\alpha = 0.10$  for a 90% CI, and so on.

## The sign test

- ▶ Test for population *median*.
- ▶ Data:  
 $R> x=c(3, 4, 5, 5, 6, 8, 11, 14, 19, 37)$
- ▶ Let  $M$  denote pop. median. Test  $H_0 : M = 12$  vs.  $H_a : M \neq 12$ .
- ▶ If  $H_0$  true, each data value either above or below 12, prob. 0.5 of each (ignore exactly 12's).
- ▶ Test statistic: # values above 12.
- ▶ If  $H_0$  true, test statistic has *binomial distribution*,  $n = 10, p = 0.5$ .
- ▶ Observed 3 values above 12 (and 7 below).
- ▶ P-value comes from prob. of 3 successes or less in that binomial distribution.

# Probability distribution calculator

- ▶ R knows about many distributions, eg. `norm`, `binom`.
- ▶ All have prefixes:
  - ▶ `d`: probability (or density) of exactly given value
  - ▶ `p`: probability of given value *or less* (CDF)
  - ▶ `q`: value that has that much probability less (inverse CDF).

<code>R&gt; pnorm(1.96)</code>	Standard normal
<code>[1] 0.9750021</code>	Prob of less than 1.96
<code>R&gt; qnorm(0.95)</code>	Value with prob 0.95 less than it
<code>[1] 1.644854</code>	
<code>R&gt; dbinom(1,4,0.3)</code>	Binomial, $n = 4, p = 0.3$
<code>[1] 0.4116</code>	Prob of exactly 1 success
<code>R&gt; pbinom(1,4,0.3)</code>	Prob of 1 success or less
<code>[1] 0.6517</code>	
<code>R&gt; qbinom(0.9,4,0.3)</code>	Value with prob (at least) 0.9 less than it
<code>[1] 2</code>	

## Sign test

- ▶ Data:

```
R> x
```

```
[1] 3 4 5 5 6 8 11 14 19 37
```

- ▶ Hypotheses about median  $M$ :  $H_0 : M = 12$ ,  $H_a : M \neq 12$ .
- ▶ Test statistic: number of values above 12.

```
R> table(x>12)
```

	FALSE	TRUE
--	-------	------

	7	3
--	---	---

3 values above.

- ▶ P-value: prob. of 3 successes or less in binomial  $n = 10$ ,  $p = 0.5$ , times 2 (two-sided test):

```
R> 2*pbinom(3,10,0.5)
```

```
[1] 0.34375
```

- ▶ No evidence that the median differs from 12.

## Sign test in SAS

Lives in proc univariate. Note how null median specified:

```
SAS> data x;  
SAS>   input x@@;  
SAS>   cards;  
SAS>     3 4 5 5 6 8 11 14 19 37  
SAS>   ;  
SAS>  
SAS> proc univariate location=12;
```

## SAS sign test results

The UNIVARIATE Procedure

Variable: x

Tests for Location: Mu0=12

Test	-Statistic-	-----p Value-----
Student's t	t -0.24399	Pr >  t  0.8127
Sign	M -2	Pr >=  M  0.3438
Signed Rank	S -9.5	Pr >=  S  0.3730

P-value 0.3438, same as R.

## Making R sign test into function

- ▶ Get CI for median as “what values of  $M$  would sign test not reject for?”
- ▶ Trying lots of values of  $M$ , do sign test for each.
- ▶ Efficiency: write R *function* to do sign test and return P-value:

```
R> sign.test=function(mydata,med)
R> {
R>   tbl=table(mydata<med)
R>   stat=min(tbl)
R>   n=length(mydata)
R>   pval=2*pbisnom(stat,n,0.5)
R>   return(pval)
R> }
```

- ▶ Function takes data and null median as input.
- ▶ Test statistic is #values greater or less than  $M$ , whichever is smaller.
- ▶ Returns two-sided P-value.

## Using function to get CI

Idea: call function repeatedly on same data but different null  $M$ . Values of  $M$  with P-value  $> 0.05$  inside 95% CI, others outside.

First test on actual problem:

```
R> sign.test(x, 12)
[1] 0.34375
```

All right. Try some values, making sure not to use any data values:

```
R> sign.test(x, 15)
[1] 0.109375
```

```
R> sign.test(x, 20)
[1] 0.02148437
```

15 inside, 20 outside.

## Completing the interval

Lower end:

```
R> sign.test(x, 4.5)
```

```
[1] 0.109375
```

```
R> sign.test(x, 3.5)
```

```
[1] 0.02148437
```

4.5 inside, 3.5 outside.

So 95% CI for pop. median goes from 4.5 to 15.

Not a very good CI, for two reasons:

- ▶ Sample size small.
- ▶ Sign test not very powerful.

But it *is* a confidence interval.

## Some data

Take a look at these data:

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

## Matched pairs

- ▶ Data are comparison of 2 drugs for effectiveness at reducing pain.
- ▶ 12 subjects 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.

## Analysis, in SAS

Uses a *t*-test. Note paired line.

```
SAS> data analgesic;  
SAS>     infile "analgesic.txt";  
SAS>     input subject druga drugb;  
SAS>  
SAS> proc ttest;  
SAS>     paired druga*drugb;
```

## Output of matched pairs *t*-test

The TTEST Procedure

Difference: 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	5.7884
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 differences in data step:

```
SAS> data analgesic;  
SAS>     infile "analgesic.txt";  
SAS>     input subject druga drugb;  
SAS>     diff=druga-drugb;  
SAS>  
SAS> proc ttest h0=0;  
SAS>     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.

## Results

The TTEST Procedure

Variable: diff

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	5.7884
DF	t Value	Pr >  t			
11	-2.17	0.0530			

## Paired test in R (1)

I forgot to give the variables names in the data file. Use what we have.

```
R> pain=read.table("analgesic.txt",header=F)
```

```
R> pain
```

	V1	V2	V3
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 in R (2)

```
R> attach(pain)
R> t.test(V2, V3, paired=T)
```

Paired t-test

```
data: V2 and V3
t = -2.1677, df = 11, p-value = 0.05299
alternative hypothesis: true difference in means is not equal to
95 percent confidence interval:
-4.29941513  0.03274847
sample estimates:
mean of the differences
-2.133333
```

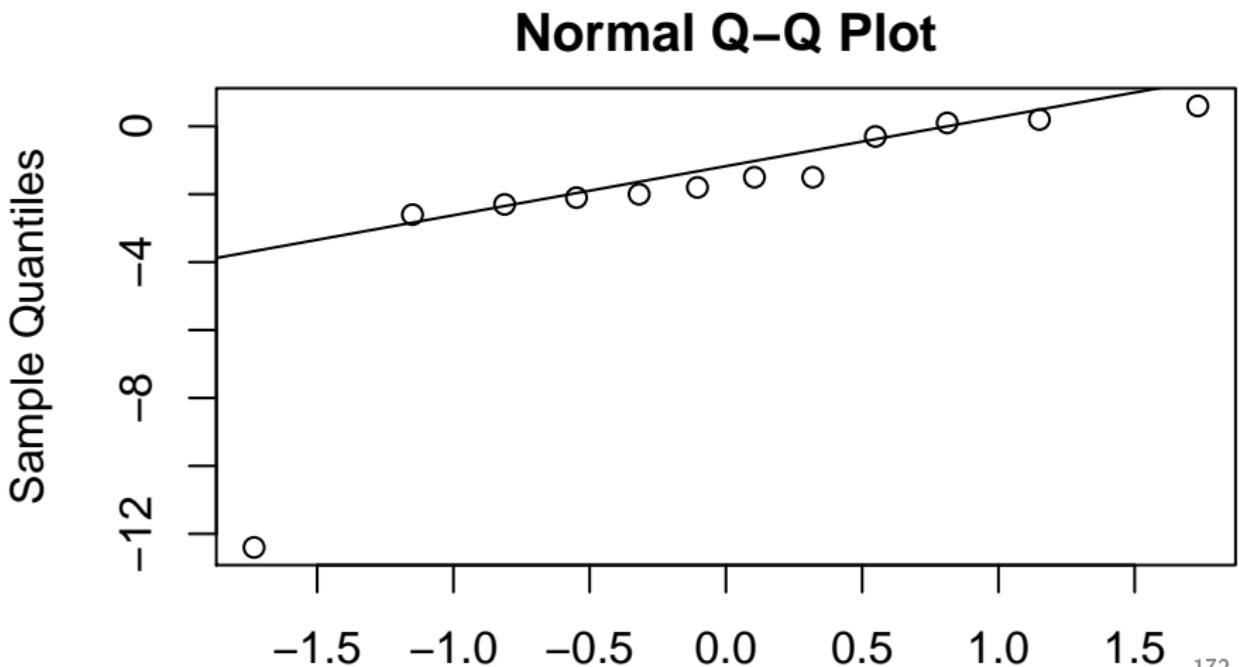
P-value as before. Likewise, you can calculate the differences yourself and do a 1-sample  $t$ -test on them.

## Assessing normality

- ▶ Analyses assume (theoretically) that differences 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.

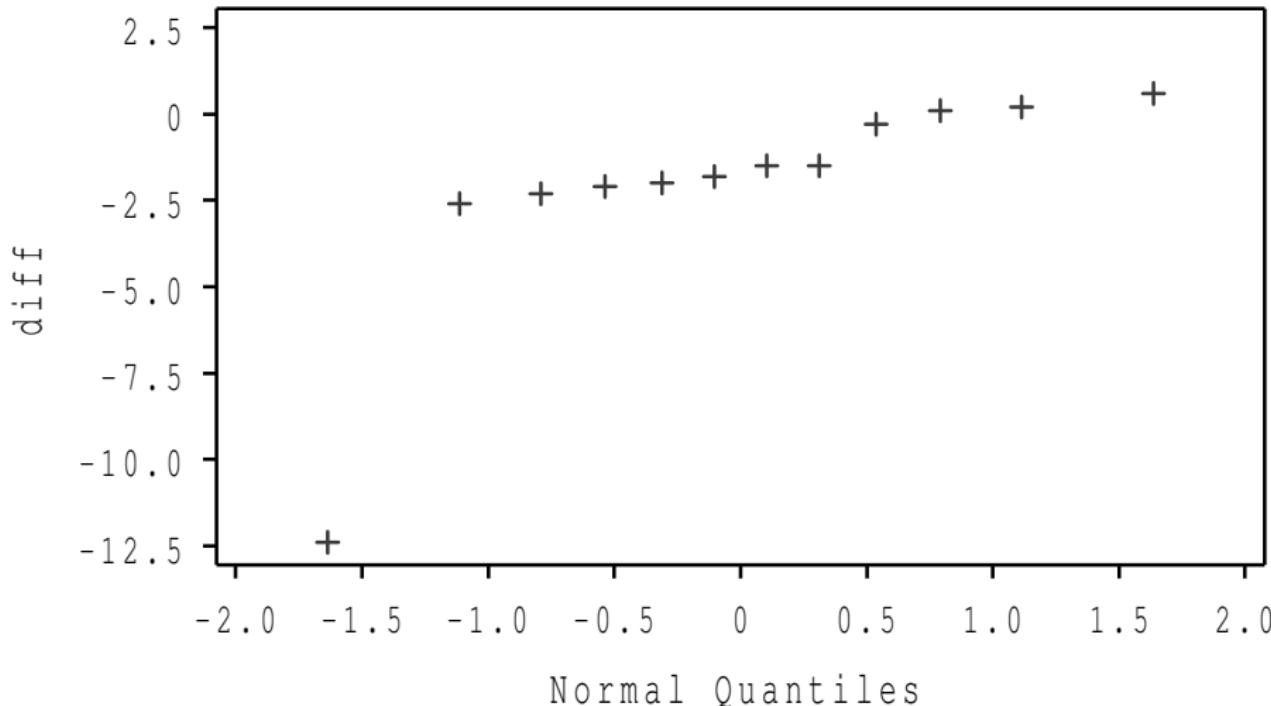
## The normal quantile plot

```
R> diff=V2-V3  
R> qqnorm(diff)  
R> qqline(diff)
```



And in SAS...

```
SAS> proc univariate noprint;  
SAS> qqplot diff;
```

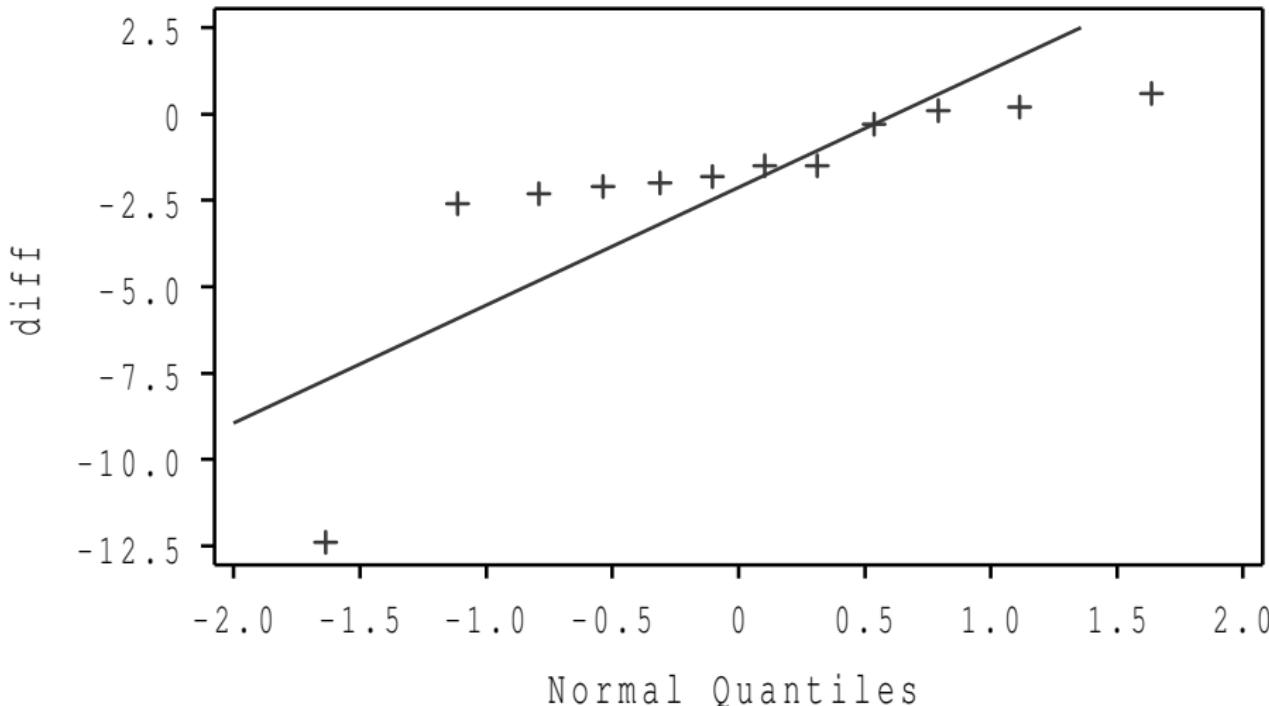


## Getting a line on SAS normal quantile plot

- ▶ 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

```
SAS> proc univariate noprint;  
SAS> qqplot diff / normal(mu=est sigma=est);
```



## Another way to estimate mean and SD

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

```
R> qnorm(0.25); qnorm(0.75)
```

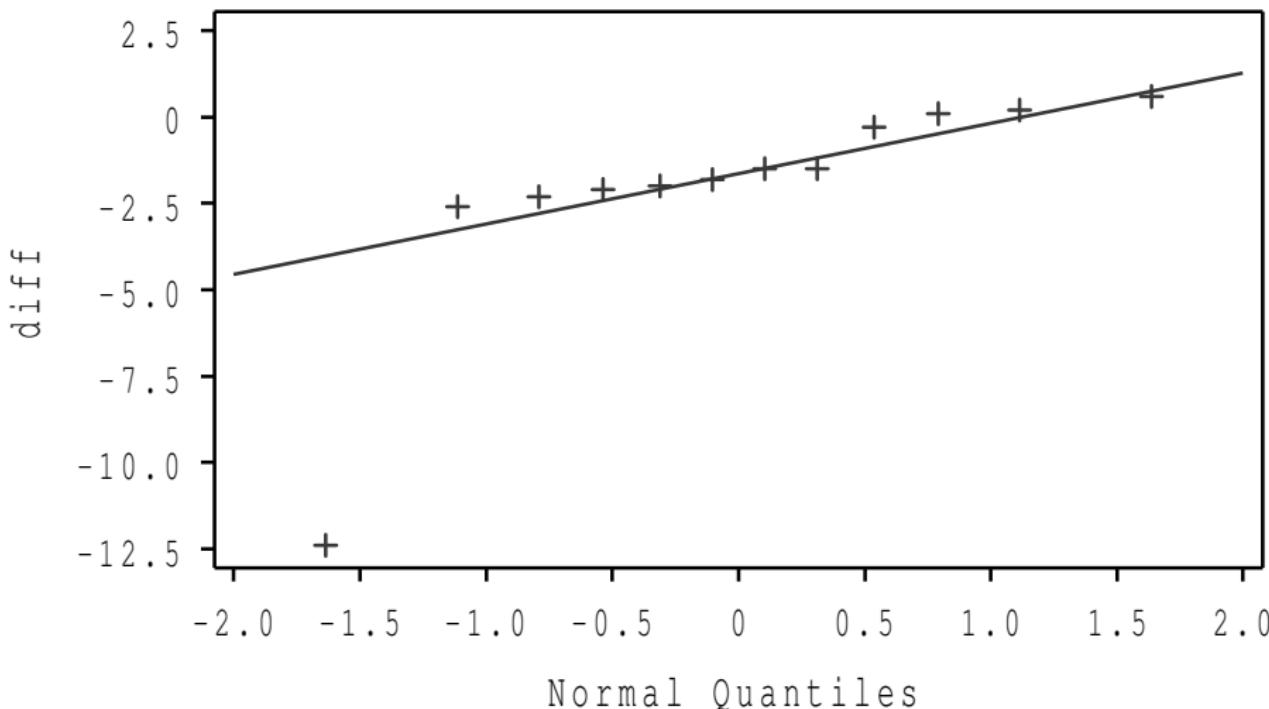
```
[1] -0.6744898
```

```
[1] 0.6744898
```

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

## Using improved $\mu$ and $\sigma$

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



## Dealing with non-normality

- ▶ One approach: do nothing, since the  $t$ -test takes care of non-normality.
- ▶ Another: make a sign test, and test whether *median* difference is zero.  
In SAS, just ask for it.
- ▶ Another: do a randomization test.

## Sign test in SAS

```
SAS> proc univariate;  
SAS>   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

P-value 0.1460.

## Sign test in R

- ▶ SAS's P-value 0.1460, not close to significant. Did the outlier exaggerate the significance in the  $t$ -test?
- ▶ In R, either use the `sign.test` function that we wrote before:

```
R> sign.test(diff, 0)
```

```
[1] 0.1459961
```

- ▶ Or, count the number of (say) positive differences (3 out of 12), and calculate the P-value as

```
R> pbinom(3, 12, 0.5)*2
```

```
[1] 0.1459961
```

- ▶ I got 95% sign-test CI for median difference as  $-2.2$  to  $0.1$ , same idea as before.

## Randomization test

- ▶ How to do randomization? What can be switched around if  $H_0$  of no difference is true?
- ▶ Randomly allocate results for each person to drug A or B.
- ▶ Observed sample mean difference:

```
R> mean(diff)
```

```
[1] -2.133333
```

- ▶ Switching drugs (from observed data) means switching *sign* of difference A minus B. So generate random sample of  $-1$  and  $1$  of size 12 (with replacement), like this:

```
R> pm=c(1, -1)
```

```
R> random.pm=sample(pm, 12, replace=T)
```

```
R> random.pm
```

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

## Generating a randomization sample and mean

- ▶ Then take observed differences and generate randomized ones by multiplying `diff` by these random signs:

```
R> random.diff=diff*random.pm
```

```
R> random.diff
```

```
[1] 1.5 -2.1 -0.3 -0.2 -2.6 -0.1 -1.8 0.6 1.5
```

```
R> mean(random.diff)
```

```
[1] -0.9666667
```

- ▶ This randomization gave a sample mean difference close to  $-1$ , while our observed value was more negative,  $-2.13$ .

## Doing this lots of times

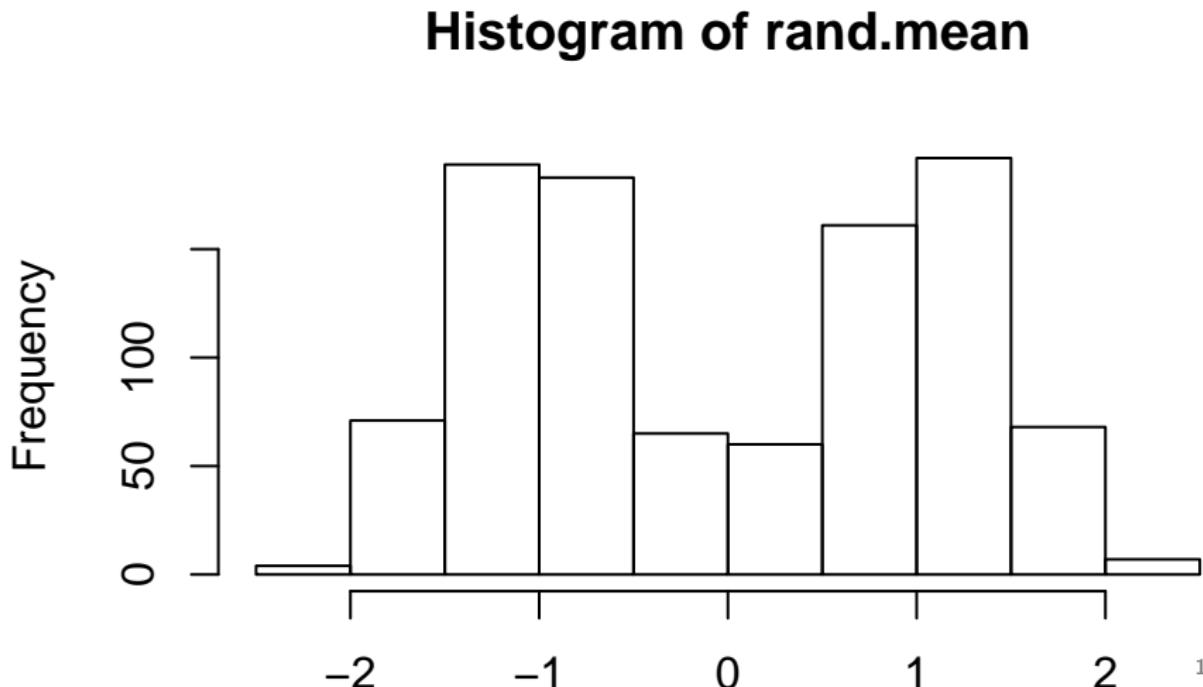
Make a loop, as before. Create empty vector to hold results, and fill it in as we go. Using pm defined above.

```
R> nsim=1000;  
R> rand.mean=numeric(nsim)  
R> for (i in 1:nsim)  
R> {  
R>   random.pm=sample(pm,12,replace=T)  
R>   random.diff=diff*random.pm  
R>   rand.mean[i]=mean(random.diff)  
R> }
```

## Histogram of randomization distribution

First thing is to have a look at a histogram:

```
R> hist(rand.mean)
```



## The randomization distribution

- If  $t$ -test were plausible, this would look normal. Does it?

## The randomization distribution

- ▶ If  $t$ -test were plausible, this would look normal. Does it?
- ▶ Why not? The outlier difference,  $-12.5$ . If counted as positive, mean probably positive; if counted as negative, mean probably negative.
- ▶ This true almost regardless of other values (whether plus or minus).
- ▶ Randomization test can be trusted, though.
- ▶ P-value: how many of randomized means came out less than observed  $-2.13$ , doubled:

```
R> tab=table(rand.mean<=mean(diff))
```

```
R> tab
```

FALSE	TRUE
-------	------

998	2
-----	---

```
R> 2*tab[2]/nsim
```

```
TRUE
```

```
0.004
```

- ▶ Randomization P-value tiny; no doubt now that mean pain relief for two drugs different.

## Randomization test for median difference

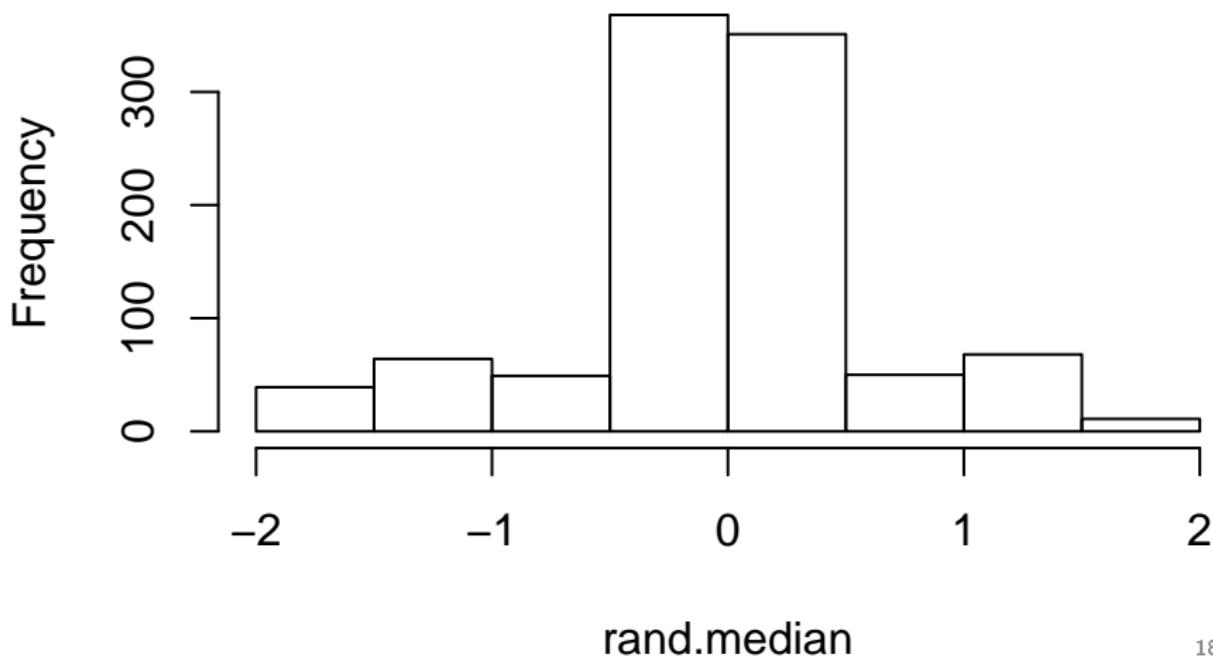
- ▶ With outlier, should we be using mean?
- ▶ Randomization test flexible; change mean to median:

```
R> nsim=1000;  
R> rand.median=numeric(nsim)  
R> for (i in 1:nsim)  
R> {  
R>   random.pm=sample(pm, 12, replace=T)  
R>   random.diff=diff*random.pm  
R>   rand.median[i]=median(random.diff)  
R> }  
R> hist(rand.median)
```

- ▶ Change mean to median everywhere.

## Histogram of randomization distribution for median

**Histogram of rand.median**



## P-value for randomization test for median

- ▶ Randomization distribution has only one peak now.
- ▶ Sample median difference was  $-1.65$  (`median(diff)`).
- ▶ P-value is proportion of these at least as extreme:

```
R> tab=table(rand.median<=median(diff))
```

```
R> tab
```

	FALSE	TRUE
--	-------	------

980	20
-----	----

```
R> 2*tab[2]/nsim
```

TRUE

0.04

- ▶ This time, *just* reject hypothesis that two drugs equally good.

## Comparison

Test results seem to be very different:

Test	Statistic	P-value
<i>t</i> -test	mean	0.0530
randomization	mean	0.0040
sign test	median	0.1460
randomization	median	0.0400

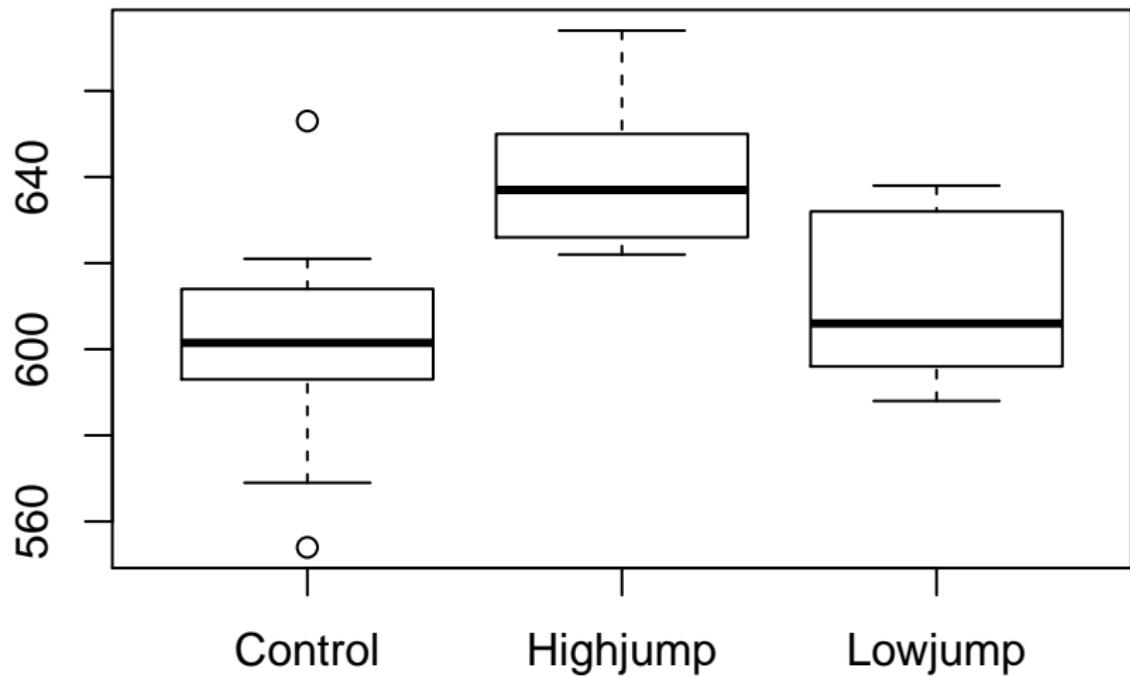
- ▶ Not that much agreement.
- ▶ Sign test P-value much larger than others (lack of power?)
- ▶ *t*-test probably not trustworthy (very non-normal dist. of differences).
- ▶ Using mean probably not wise (outlier difference)
- ▶ Randomization test for median probably best.

## Jumping rats

- ▶ Recall jumping rats.
- ▶ See whether larger amount of exercise (jumping) went with higher bone density.
- ▶ 30 rats randomly assigned to one of 3 treatment groups:
  - ▶ Control (no jumping)
  - ▶ Low jumping
  - ▶ High jumping
- ▶ Random assignment: rats in each group similar in all important ways.
- ▶ So entitled to draw conclusions about cause and effect.
- ▶ Recall boxplots:

```
R>     rats=read.table("jumping.txt",header=T)
R>     boxplot(density~group,data=rats)
```

## Boxplots



## Testing: ANOVA

```
R> rats.aov=aov(density~group,data=rats)
R> 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.
- ▶ Null hypothesis: all 3 groups have same mean.
- ▶ Alternative: “not all 3 means the same”, at least one is different from others.
- ▶ Reject null, but not very useful finding.

## Same thing in SAS

Read in data and do ANOVA:

```
SAS> data rats;  
SAS>   infile 'jumping.txt' firstobs=2;  
SAS>   input group $ density;  
SAS>  
SAS> proc anova;  
SAS>   class group;  
SAS>   model density=group;
```

## Results (some)

The ANOVA Procedure

Dependent Variable: density

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

Source	DF	Anova SS		F Value
		Anova SS	Mean Square	
group	2	7433.86667	3716.93333	7.98
Source	Pr > F			
group		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:

```
R> attach(rats)
R> controls=density[group=="Control"]
R> lows=density[group=="Lowjump"]
R> highs=density[group=="Highjump"]
```

## Control vs. low

```
R> t.test(controls, lows)
```

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

```
data: controls and lows
```

```
t = -1.0761, df = 16.191, p-value = 0.2977
```

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

```
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

```
R> t.test(controls,highs)
```

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

```
data: controls and highs
```

```
t = -3.7155, df = 14.831, p-value = 0.002109
```

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

```
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

```
R> t.test(lows,highs)
```

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

```
data: lows and highs
```

```
t = -3.2523, df = 17.597, p-value = 0.004525
```

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

```
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  $\alpha$  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

```
R> rats.aov=aov(density~group,data=rats)
R> 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	<i>t</i> -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

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

## Tukey output (some)

The ANOVA Procedure

Tukey's Studentized Range (HSD) Test for density

Means with the same letter are not significantly different.

T

u

k

e

y

G

r

o

u

p

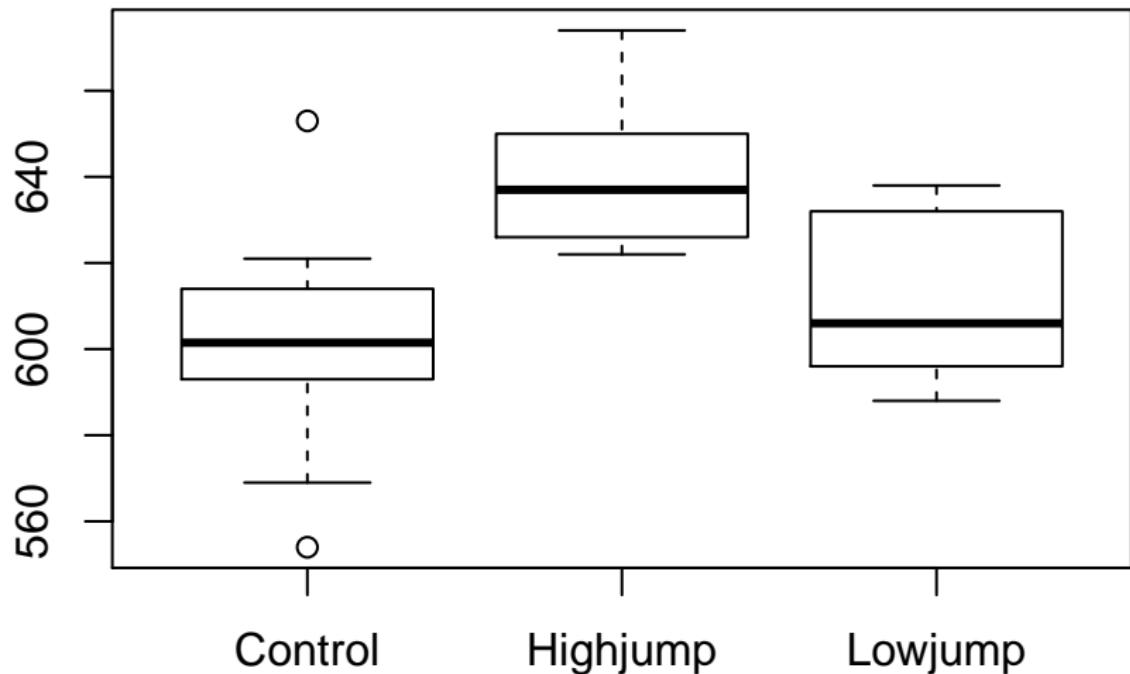
i

n

group	Mean	N
Highjump	638.700	10
Lowjump	612.500	10
Control	601.100	10

## Checking assumptions

```
R> boxplot(density~group,data=rats)
```



## 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:

- ▶ Transformation of response (usually works best when SD increases with mean)
- ▶ Different test, eg. randomization test.
- ▶ Can also try Mood's Median Test (if you know about chi-squared tests). Based on medians, like a generalization of sign test. Not very powerful.

## Randomization test

- ▶ Like two-sample randomization test: randomly shuffle group memberships.
- ▶ Choices about test statistic, eg.
  - ▶ usual ANOVA  $F$ -statistic
  - ▶ Tukey-like highest sample mean minus lowest.
- ▶ Go with Tukey-like idea.
- ▶ Calculate observed value of test statistic:

```
R> group.means=aggregate(density~group,data=rats,mean)  
R> group.means
```

	group	density
1	Control	601.1
2	Highjump	638.7
3	Lowjump	612.5

Actual means in second column of `group.means`:

```
R> z=group.means[,2]  
R> obs=max(z)-min(z)  
R> obs  
[1] 37.6
```

## Randomly shuffling groups

Actual groups are:

```
R> rats$group
```

```
[1] Control Control Control Control Control Control Control Control  
[9] Control Control Lowjump Lowjump Lowjump Lowjump Lowjump Lowjump  
[17] Lowjump Lowjump Lowjump Lowjump Highjump Highjump Highjump  
[25] Highjump Highjump Highjump Highjump Highjump Highjump Highjump  
Levels: Control Highjump Lowjump
```

Shuffle like this:

```
R> n.total=length(rats$group)
```

```
R> shuffled.groups=sample(rats$group,n.total)
```

```
R> shuffled.groups
```

```
[1] Highjump Control Control Lowjump Lowjump Lowjump Control  
[9] Lowjump Lowjump Highjump Control Control Highjump Highjump Highjump  
[17] Lowjump Highjump Lowjump Highjump Control Highjump Control  
[25] Highjump Highjump Lowjump Control Lowjump Control  
Levels: Control Highjump Lowjump
```

## Calculate means and highest minus lowest

Same idea as for actual data, but using `shuffled.groups` instead of `group`:

```
R> my.group.means=aggregate(density~shuffled.groups,data=rats,mean)
R> z=my.group.means[,2]
R> max(z)-min(z)

[1] 10.2
```

This time, largest minus smallest was 10.2.

Now do it a bunch of times.

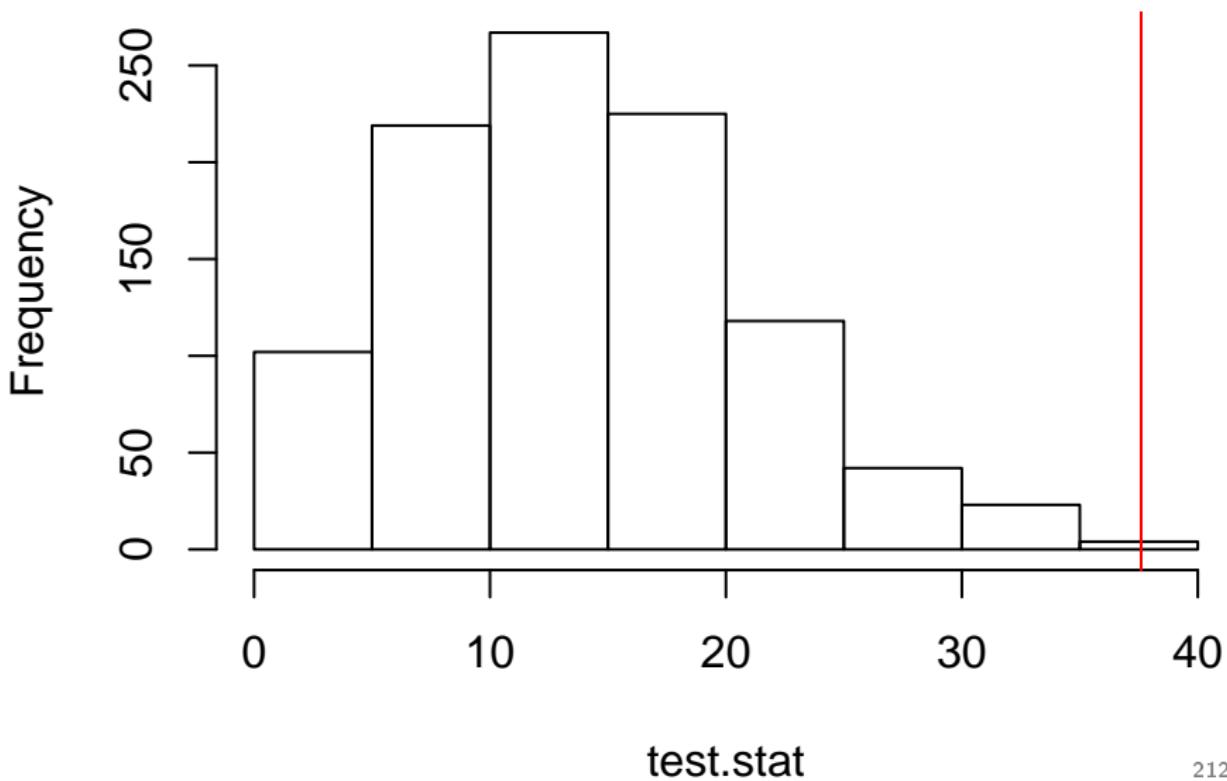
## Obtaining randomization distribution

I'm using n.total from before (it's 30, the total number of rats).

```
R> nsim=1000
R> test.stat=numeric(nsim)
R> for (i in 1:nsim)
R>   {
R>     shuffled.groups=sample(rats$group,n.total)
R>     my.group.means=aggregate(density~shuffled.groups,
R>       data=rats,mean)
R>     z=my.group.means[,2]
R>     test.stat[i]=max(z)-min(z)
R>   }
R> hist(test.stat)
R> abline(v=obs,col="red")
```

Randomization distribution of test.stat

## Histogram of test.stat



## Comments and P-value

- ▶ Distribution shape skewed to right (lower limit of 0).
- ▶ Red line marks observed highest minus lowest: seems unusually high.
- ▶ If group means really different (anyhow), highest minus lowest will be *large*, so one-tailed test.
- ▶ (Same logic as one-sidedness of  $F$ -test in ANOVA.)
- ▶ P-value the usual way:

```
R> table(test.stat>=obs)
```

FALSE	TRUE
998	2

- ▶ P-value  $2/1000 = 0.002$ .

## Using randomization test like Tukey

- ▶ Compare *any* pair of means with this randomization distribution. One above was highjump vs. control. How about highjump vs. lowjump? Calculate observed difference first:

```
R> hilo=group.means[2,2]-group.means[3,2]
```

```
R> hilo
```

```
[1] 26.2
```

```
R> table(test.stat>=hilo)
```

```
FALSE  TRUE
```

```
 942    58
```

P-value 0.058.

## lowjump vs. control

- ▶ And now lowjump vs. control:

```
R> loc=group.means[3,2]-group.means[1,2]
```

```
R> loc
```

```
[1] 11.4
```

```
R> table(test.stat>=loc)
```

	FALSE	TRUE
--	-------	------

392	608
-----	-----

giving P-value 0.608.

## Compare randomization P-values with Tukey ones

Groups	Tukey P-value	Randomization P-value
highjump-control	0.0016	0.0020
highjump-lowjump	0.0298	0.0580
lowjump-control	0.4744	0.6080

- ▶ The randomization P-values are larger.
- ▶ Indeed, the highjump-lowjump comparison is no longer significant at  $\alpha = 0.05$ .
- ▶ This might be an allowance for the groups having unequal spreads (Tukey assumes equal SDs).

## Part VI

Writing a report (and doing science)

## Writing a report

- ▶ The first part of doing research is doing research,
- ▶ The second part of doing research is communicating your findings to the world.
- ▶ That means writing a report or paper to describe what you did.
- ▶ Research should be **reproducible**: anyone else should be able to re-do what you did and verify that it is correct.
- ▶ So your report should contain all the relevant detail for someone to *replicate* (or not) your findings.
- ▶ Need to support your conclusions so that your readers will believe you.
- ▶ This is how science (or human knowledge generally) works.

## Falsifiability

- ▶ To say something meaningful about how the world works, you need a statement that could be proved wrong, like

*No alien spaceships have ever landed in Roswell, NM*

- ▶ How to prove this wrong? Just find an alien spaceship there.
- ▶ Some unfalsifiable statements:
  - ▶ An alien spaceship crashed in Roswell, NM.
  - ▶ A giant white gorilla lives in the Himalayas.
  - ▶ Loch Ness contains a giant reptile.
- ▶ If you disagree, you might be met with “the government hid the evidence”, “the snow covered the gorilla’s tracks”, “the monster is hiding in the mud at the bottom of Loch Ness”.
- ▶ All you ever find is an absence of evidence. And you can never disprove these. But they are not science.
- ▶ Science consists of a vast array of falsifiable statements that have never been falsified, despite everyone’s best efforts. This is what it means for something in science to be “true”.

## Falsifiability part 2

- ▶ “Life arose by evolution” also falsifiable, but never has been falsified.
- ▶ Evolution provides the best explanation we know of how life arose.
- ▶ All the many things in it that could be falsified have not been.

## Clinical trials

- ▶ How do we know that drugs work?
- ▶ Randomized controlled trials.
- ▶ Have to show that new drug works significantly better than current drug (or placebo).
- ▶ New drug *could* be worthless (“falsifiable”), but drug trials show it’s actually effective.
- ▶ Drug trials typically use *huge* numbers of patients, so drug has to work many times. It might not, but usually it does.
- ▶ New drug, or new piece of science, has to prove its worth before being accepted.

## A typical journal article

Watch the line wrap!

<http://0-jap.physiology.org.library.pcc.edu/content/100/3/839.full#abstract-1>

# Writing a report

- ▶ Tell the story of what you did.
- ▶ Typical structure:

Introduction: “setting the scene”

Literature review: “what others have done”

Methods: “what you did”

Results: “what happened”

Conclusions: “what the results mean”

Discussion: “what the conclusions mean to the world”.

- ▶ Details of structure vary by discipline/requirements.
- ▶ Journal articles often begin with Abstract (summary of what article is about).
- ▶ Writing for *your readers*: make it *as easy as possible* to read and understand.

# The Introduction

- ▶ What is your study all about?
- ▶ Why are you doing it?
- ▶ What are you expecting (hoping) to see?
- ▶ What research hypotheses do you have?
- ▶ “As concise as possible without detracting from clarity”.

## Literature Review

- ▶ What do other people say?
- ▶ Include *references*, eg. “Dribblington and Smith (1992) demonstrate that blowfish, in fact, suck.” (I made this up.)
- ▶ Summarize in a few words, as above, what each cited work says (in the context of what your report is about).
- ▶ Include each cited work in your list of References (typically at the end of the report).
- ▶ Sometimes combined with the Introduction.

# Methods

- ▶ What did you do?
- ▶ Sometimes has more descriptive title.
- ▶ What experiment did you conduct?
- ▶ How did you design it?
- ▶ What results did you collect?
- ▶ How did you collect them (eg. technology)?
- ▶ *Not* the place for justifying what you did (that belongs in Introduction).
- ▶ Matter-of-fact, *clear*.

# Results

- ▶ For a simple analysis: one block of results, report and move on.
- ▶ More complicated analysis: might need to split up into several sections, like this (only at greater length):
  - ▶ First I did *this* and got *these* results,
  - ▶ so I did *that* and got *those* results,
  - ▶ and that suggested *the other* which produced *some other* results.
- ▶ Take the time to tell your story clearly.
- ▶ You might need a mini-Methods section before each Results piece. If that makes sense, write it that way.

## Purely statistical analyses

- ▶ Such an analysis is mainly Results.
- ▶ “Methods” is just “where I got the data from”, which can be moved into Introduction (no Methods).
- ▶ Analysis can be long and multi-faceted, so can have separate sections for the various parts, eg.
  - ▶ Exploratory analysis (histograms, tables of means, scatterplots).
  - ▶ First-stage analysis (eg.  $t$ -tests, regression analysis)
  - ▶ Second-stage analysis (eg. variations on the above suggested by the results, eg. sign test, transformations in regression).
- ▶ The Conclusions (see below) is where you talk about *why* you did what you did. Results is for showing what analysis you did, and what numbers came out the other end.
- ▶ Use graphs to illustrate results where you can, rather than large tables of numbers.
- ▶ Point: display results so that Conclusions (coming up) clearest.

## “How many significant digits should I quote?”

*Only quote figures that you are pretty sure of — perhaps plus one more.*

- ▶ Know how accurate your data values are (generally, accuracy given in data set).
- ▶ Derived quantities, like mean, SD might deserve one extra decimal.
- ▶ Regression slopes trickier:
  - ▶ How many significant digits in  $y$ ?
  - ▶ How many significant digits in  $x$ ?
  - ▶ Take smaller of those. Perhaps plus one more.
- ▶ Just because SAS gives you six decimals doesn't mean *you* should give six decimals!

## Conclusions: “what do my results mean to me?”

- ▶ Get to sell your results and their consequences.
- ▶ Definitely *do* want to add your opinions here!
- ▶ Go back to your Introduction for guidance:
  - ▶ did you find out what you were hoping to?
  - ▶ how do your results lead to that conclusion?
  - ▶ if you didn't find out what you were hoping to, why not?

## Discussion: “what do my results mean to the world?”

- ▶ Place results in broader context.
- ▶ Why are these results important? What implication do they have?
- ▶ What are the limitations of my study that should make me hesitant about generalizing my results? (Eg. small sample sizes, something omitted in the study that could affect results.)
- ▶ How do your results stack up against the ones in the literature (compare the Literature Review)?
- ▶ Do your results suggest future lines of research? (They almost always do.)

## Final bits

- ▶ Acknowledge any people or organizations that helped you.
- ▶ List of references (of the works you cited in the Literature Review and elsewhere).
- ▶ Done!

Journal articles often have Abstract, short summary of what article is about.

- ▶ Sometimes Abstract has (headline) results.
- ▶ Doesn't usually have references.
- ▶ Adhere to norms of journal.

## The example report

- ▶ I know nothing about physiology!
- ▶ Abstract: really Introduction and Literature Review. Gives away some of the results. (*I think Abstract should stand on its own, telling you whether paper is worth reading.*)
- ▶ Key Words: some journals like this. Can search for Key Words that interest you.
- ▶ Here, citations are *numbered*. Online citations often link to References list.
- ▶ “What you did” here called Materials and Methods (common in sciences).
- ▶ Note technology detail.
- ▶ Results section has not just significant results, but *non-significant* ones as well.
- ▶ Just enough detail to allow for replication. (Clashes with limited space available in journals.)
- ▶ Conclusions section called Discussion.
- ▶ Plain-English last sentence.

# Reproducible Research and R

- ▶ Old way: submit your paper to journal, edit down, be prepared for requests from people who wanted to know more.
- ▶ New way: put paper online, including all code and data.
- ▶ Anyone can then reproduce exactly what you did, and discuss their findings with you.
- ▶ Problem: how do you know your analysis and conclusions in sync?
- ▶ Usual procedure: *copy* code and output into your report. But might copy old version. How do you know you got the most recent version?
- ▶ Solution (in R): construct document with specially formatted code. When you process the document, that code gets run (and results pasted in). So code you show *guaranteed* to produce results claimed.

## R Markdown

- ▶ Markup language (like HTML) containing instructions about how document should appear.
- ▶ In R Studio, File, New, R Markdown. See example document.
- ▶ Some preparatory work (see lecture notes) to be done the first time.
- ▶ Some R Markdown “markup” elements:
  - ▶ A row of ===== with blank line below makes title.
  - ▶ **\*\*text\*\*** produces **text**; **\*text\*** makes *text*.
  - ▶ (most important) The line ````{r}` starts an R **code chunk**. All the lines until closing ````` are interpreted as R code. R Studio colours code chunk grey so you can see it.
  - ▶ To include a plot, simply include the plot command(s) among your R code.
- ▶ To process an R Markdown document, save. Find “knit HTML”. Click it. After some processing, see output (HTML) in previewer window.

# Part of the output

RStudio: Preview HTML

Preview: ~/teaching/c32/notes/junk.html | Log | Save As | Publish | Find

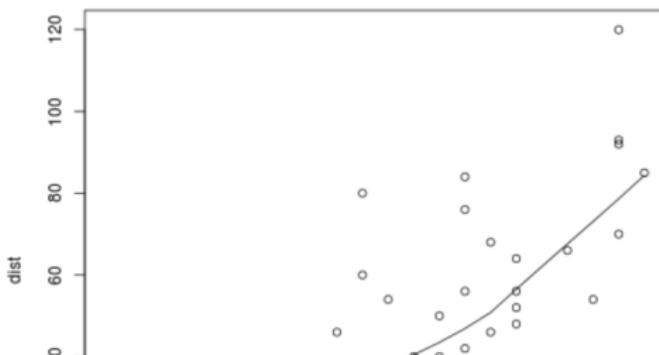
as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##          speed            dist
##  Min.   : 4.0   Min.   :  2
##  1st Qu.:12.0   1st Qu.: 26
##  Median :15.0   Median : 36
##  Mean   :15.4   Mean   : 43
##  3rd Qu.:19.0   3rd Qu.: 56
##  Max.   :25.0   Max.   :120
```

You can also embed plots, for example:

```
plot(cars)
lines(lowess(cars))
```



## Some tips

- ▶ To add a new code chunk, look for Chunks button and select Insert Chunk. Or, Control-Alt-I.
- ▶ Play with the template document! Make a histogram, do a regression, etc., etc.
- ▶ You can also run the code chunks in R Studio, to check that they are working properly. These are the other options in Chunks.
- ▶ If you see an error in the HTML, or something you want to change, **go back to the R Markdown, change it there, and then Knit HTML again**. The R Markdown document is your “base”; the HTML is produced *from* it.
  - ▶ The HTML document can always be produced again, so it doesn’t matter if you lose it. The R Markdown document, on the other hand, needs to be kept safe.
- ▶ You can save the HTML document and view it with a web browser.
- ▶ Click the MD button (left of Knit HTML) for more things you can do with R Markdown.

# Making a presentation using R Markdown: part 1

- ▶ Uses tool pandoc (download/install; free. Instructions in lecture notes).
- ▶ Use R as command line.

- ▶ Windows (DOS): `dir` lists all files in a folder:

```
R> mycommand="dir"
```

```
R> system(mycommand)
```

- ▶ Idea: construct command, then pass to `system`.

- ▶ Linux:

```
R> mycommand="ls test.*"
```

```
R> system(mycommand)
```

test.doc

test.log

test.lst

test.md

test.Rmd

test.sas

- ▶ See the R Markdown file `test.Rmd`, also a “Markdown” file `test.md` actually used for constructing other things. Contains your text plus *output* from R commands.

## Making a presentation using R Markdown: part 2

- ▶ We take the Markdown file `test.md` and make a presentation HTML file `test_pres.html` from it.
- ▶ I use a different name for presentation HTML file to avoid confusion with HTML file from Knit HTML.
- ▶ First construct a rather complicated command, and then pass it into `system`:

```
R> mycommand="pandoc -t slidify -s test.md -o test_pres.html"
R> system(mycommand)
```

- ▶ Now open `test_pres.html` in a web browser. It probably goes off the bottom of the page. But at least it looks like a presentation.

## Fixing problems

- ▶ In the likely event that you don't like what you see, go back to **R Markdown file** and change it. To start a new page in a presentation, put in a heading with =====. This makes the title of a new slide.
- ▶ Then you have to:
  - ▶ Knit HTML again
  - ▶ Run pandoc again
  - ▶ Reload the presentation in your web browser (control-R in Firefox).
- ▶ Full-screen mode (F11 in Firefox) previews actual presentation and shows you how much will fit on one screen. (This will vary according to resolution of device you give the actual presentation on.)

## More Markdown tips

- ▶ Bullet points like this:

- \* Here is the first point
  - \* and this is the second
  - \* Here, finally, is the third point.

Or start the lines with -. Use a blank line to end.

- ▶ Inserting images from current folder or Web:

- ![caption text](image.png)
  - ![text](http://example.com/logo.png)

Latter needs you to be online while running pandoc, though not (I think) while giving presentation.

- ▶ Can give graphs captions by changing code chunk line to this:

- ```{r, fig.cap="Scatterplot of x against y"}

- ▶ Using number from R output in text:

- We had `r length(x)` simulated values altogether.

- ▶ Get “typewriter font” like this:

- This presentation was made using `pandoc`.

## Even more Markdown tips

- ▶ Code to display but not run:

```
```
```

```
for (i in 1:10)
{
  print(i)
}
```

```
```
```

Insert code chunk and take out {r}.

- ▶ Handle  $\text{\LaTeX}$  formulas and equations by running pandoc this way:  
`pandoc --mathjax -t slidy -s test.md -o test_pres.html`
- ▶ A nicer way of interfacing R with  $\text{\LaTeX}$  is via Sweave. If you are running SAS on the machine where you're preparing the document, you can use Statweave instead to incorporate code and output from both.
- ▶ I encourage you to learn  $\text{\LaTeX}$ ! Much better than Word for mathematical, scientific (and for that matter academic) documents.

## Part VII

More examples

## 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.

## The data

Some of it:

```
R> windmill=read.csv("windmill.csv",header=T)
R> head(windmill,n=10)
```

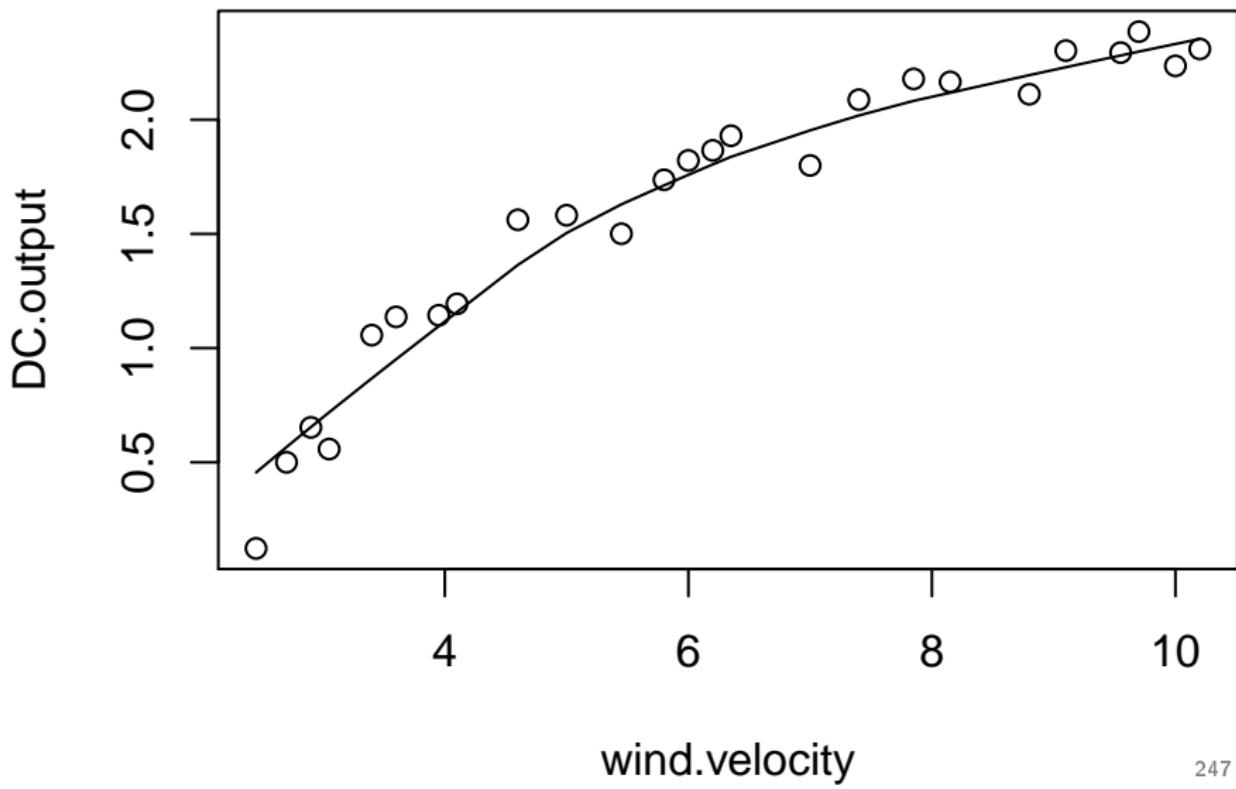
	wind.velocity	DC.output
1	5.00	1.582
2	6.00	1.822
3	3.40	1.057
4	2.70	0.500
5	10.00	2.236
6	9.70	2.386
7	9.55	2.294
8	3.05	0.558
9	8.15	2.166
10	6.20	1.866

# 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.
  - ▶ attach data frame for convenience.

```
R> attach(windmill)
R> plot(DC.output~wind.velocity)
R> lines(lowess(DC.output~wind.velocity))
```

## Scatterplot



## Comments

- ▶ Definitely a relationship: as wind velocity increases, so does DC output. (As you'd expect.)
- ▶ Is relationship linear? To help judge, *lowess curve* smooths scatterplot trend. ("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:

```
R> windmill.lm=lm(DC.output~wind.velocity)
```

```
R> summary(windmill.lm)
```

Call:

```
lm(formula = DC.output ~ wind.velocity)
```

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

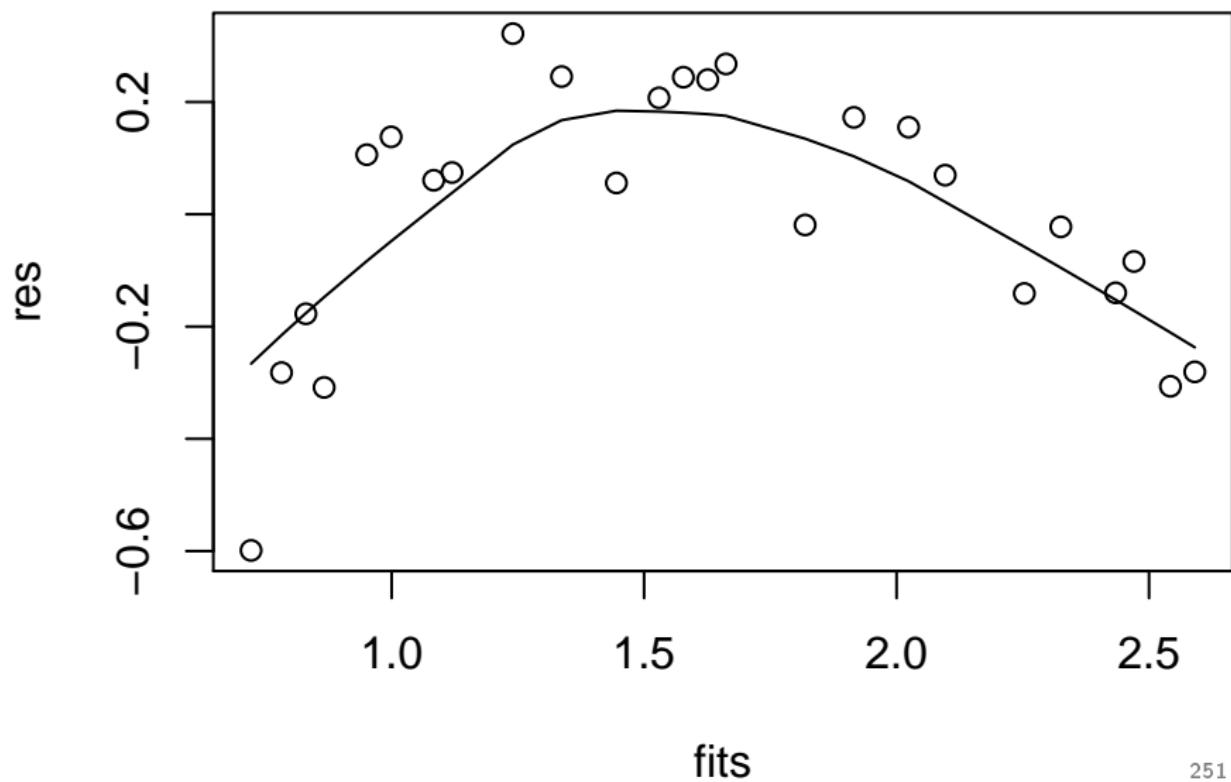
Multiple R-squared: 0.8745, Adjusted R-squared: 0.869 249 / 653

## Comments

- ▶ Strategy: `lm` actually fits the regression. Store results in a variable.  
*Then* look at the results, eg. via `summary`.
- ▶ 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.
  - ▶ Can plot the regression object; get 4 plots, at least 2 of which are mysterious.
  - ▶ Can obtain residuals and predicted values directly and plot them.
  - ▶ We go second way:

```
R> fits=fitted(windmill.lm)
R> res=resid(windmill.lm)
R> plot(fits,res)
R> lines(lowess(fits,res))
```

## Plot of residuals against fitted values



## Comments on residual plot

- ▶ Residual plot should be a random scatter of points.
- ▶ Should be no pattern “left over” after fitting the regression.
- ▶ lowess curve 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:

- ▶ defining a new variable to be  $x^2$
- ▶ adding it to right side of model formula with +:

```
R> vel2=wind.velocity^2  
R> windmill2.lm=lm(DC.output~wind.velocity+vel2)  
R> summary(windmill2.lm)
```

- ▶ This actually *multiple regression*.
- ▶ Call it *parabola model*.

## Parabola model output

Call:

```
lm(formula = DC.output ~ wind.velocity + vel2)
```

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 ***
vel2	-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

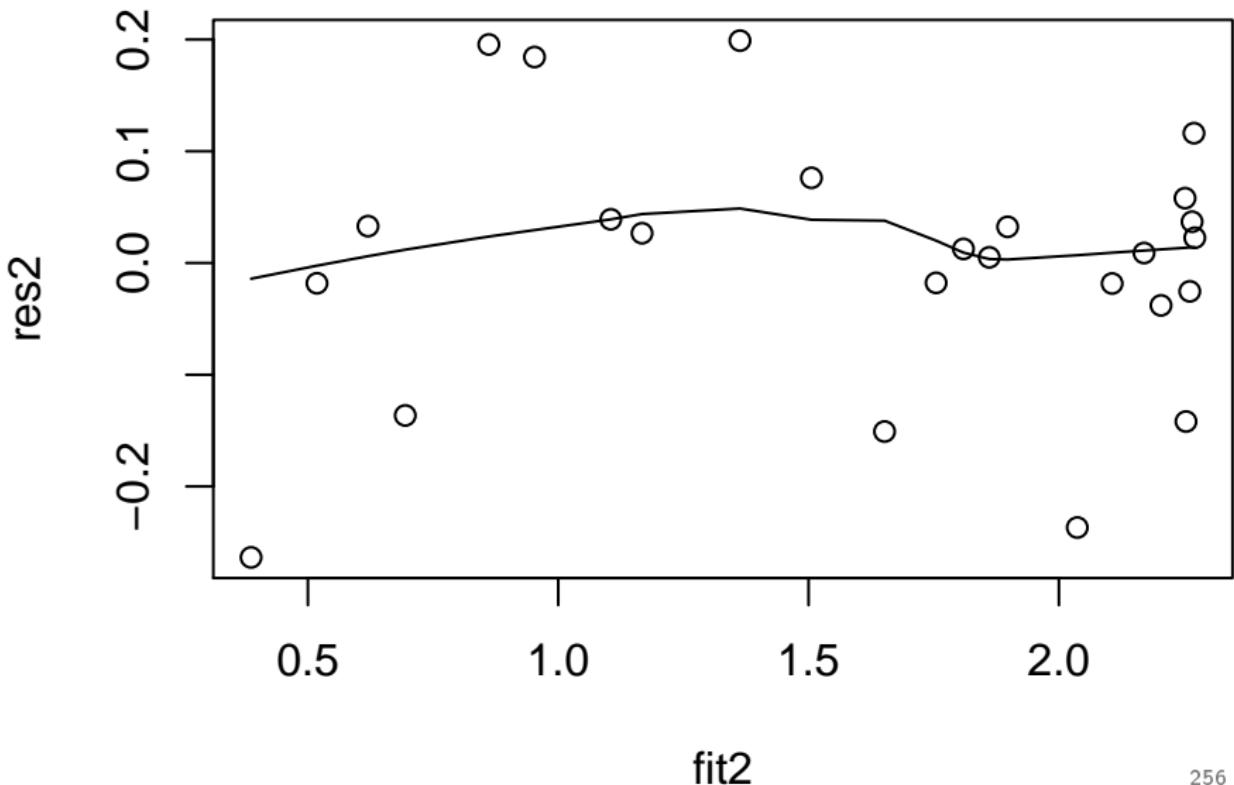
F-statistic: 328.3 on 2 and 22 DF, p-value: < 2.2e-16

## 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:

```
R> fit2=fitted(windmill2.lm)
R> res2=resid(windmill2.lm)
R> plot(fit2,res2)
R> lines(lowess(fit2,res2))
```

## Residual plot from parabola model



## Scatterplot with fitted line and curve

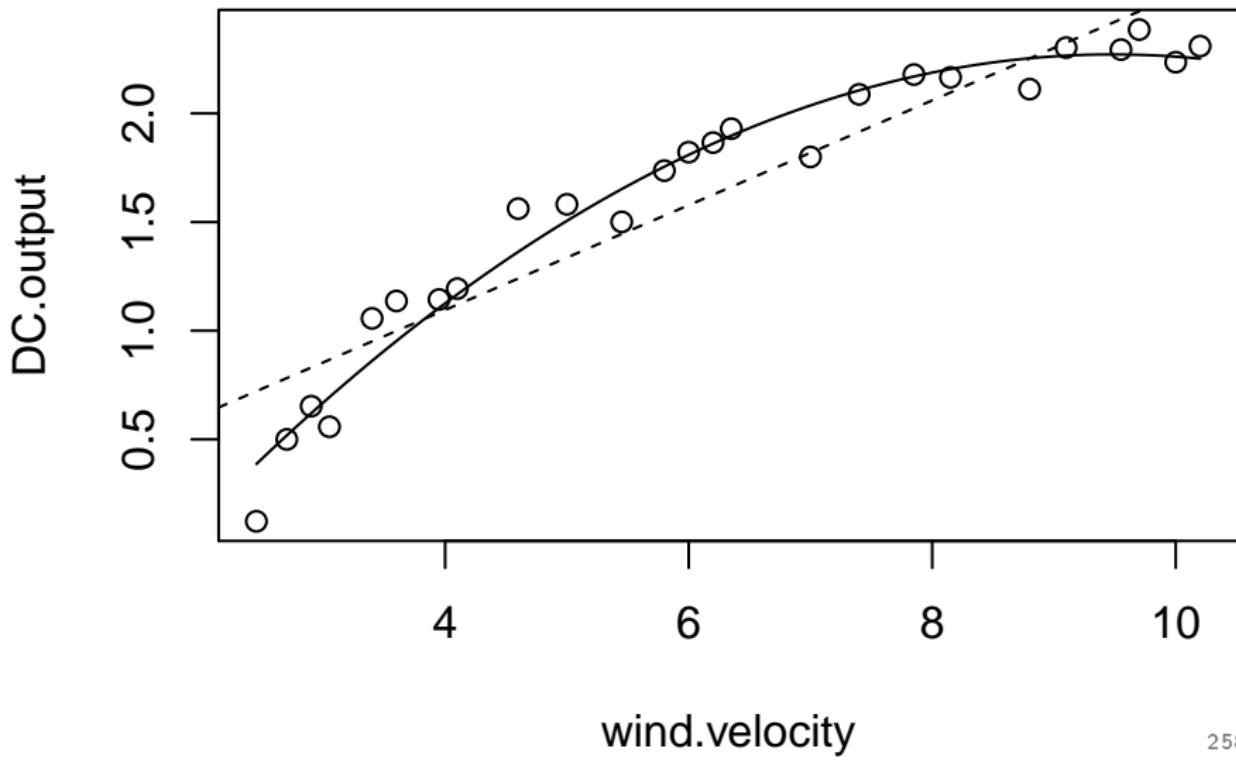
- ▶ Residual plot basically random; lowess curve basically flat. Good.
- ▶ Scatterplot with curves like this:

```
R> plot(DC.output~wind.velocity)
R> abline(windmill.lm, lty="dashed")
R> lines(spline(wind.velocity, fit2))
```

- ▶ This plots:
  - ▶ scatterplot
  - ▶ fitted line (`abline` plots a line by intercept and slope, which it can get from fitted model object `windmill.lm`). Line plotted dashed (to distinguish from curve).
  - ▶ `lines` plots curves (!), `spline` makes smooth curve out of the fitted values from parabola.
- ▶ Investigate why `lines(wind.velocity, fit2)` doesn't work.

## Scatterplot with fitted line and curve

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*.
- ▶ 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)

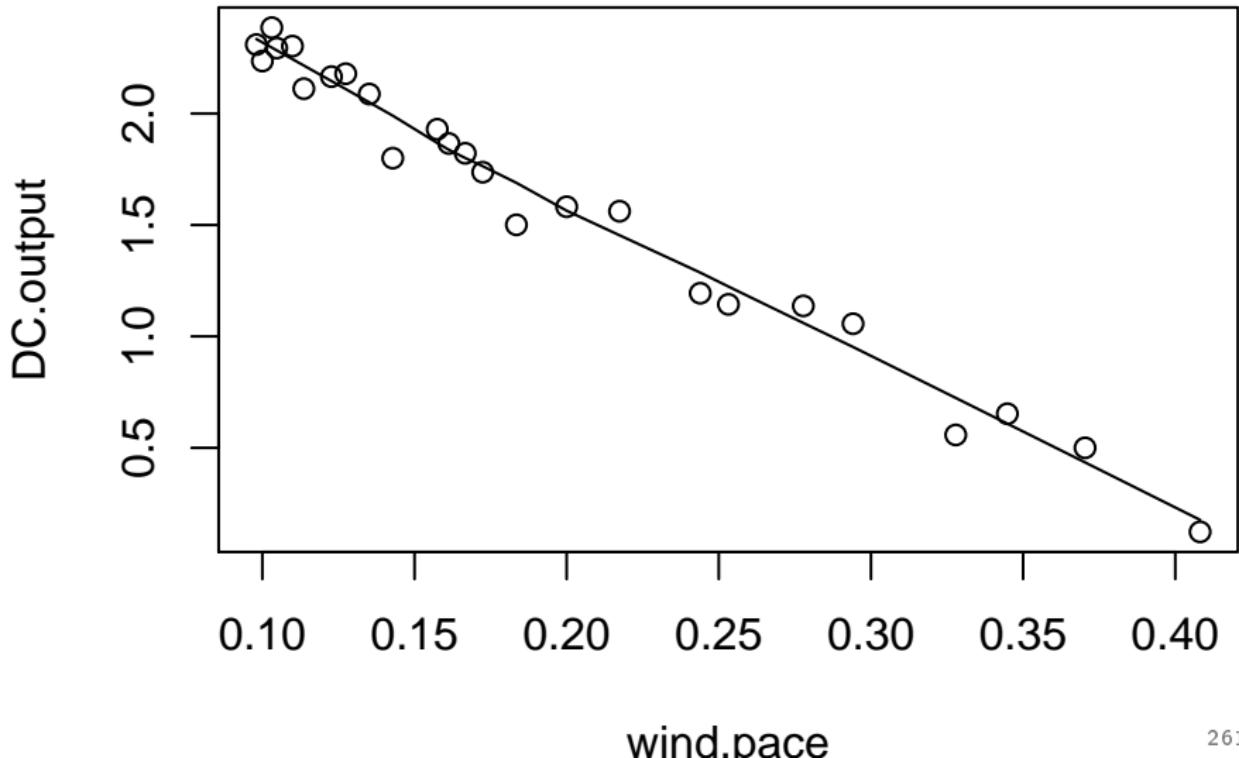
```
R> wind.pace=1/wind.velocity  
R> plot(DC.output~wind.pace)  
R> lines(lowess(DC.output~wind.pace))
```

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

```
R> windmill3.lm=lm(DC.output~wind.pace)  
R> summary(windmill3.lm)
```

## Scatterplot for wind.pace

That's pretty straight.



## Regression output

Call:

```
lm(formula = DC.output ~ wind.pace)
```

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

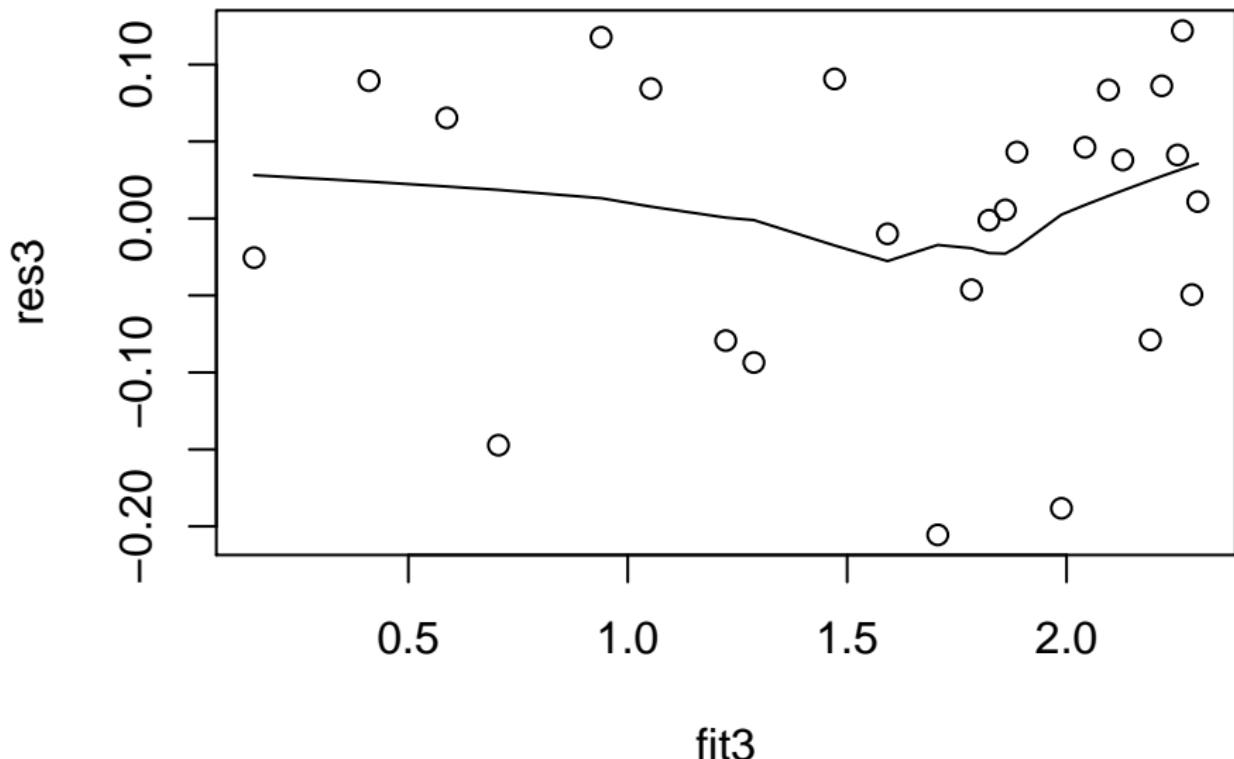
F-statistic: 1128 on 1 and 23 DF, p-value: < 2.2e-16

## 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:

```
R> fit3=fitted(windmill3.lm)
R> res3=resid(windmill3.lm)
R> plot(fit3,res3)
R> lines(lowess(fit3,res3))
```

## Residual plot for asymptote model



## Plotting trend on scatterplot

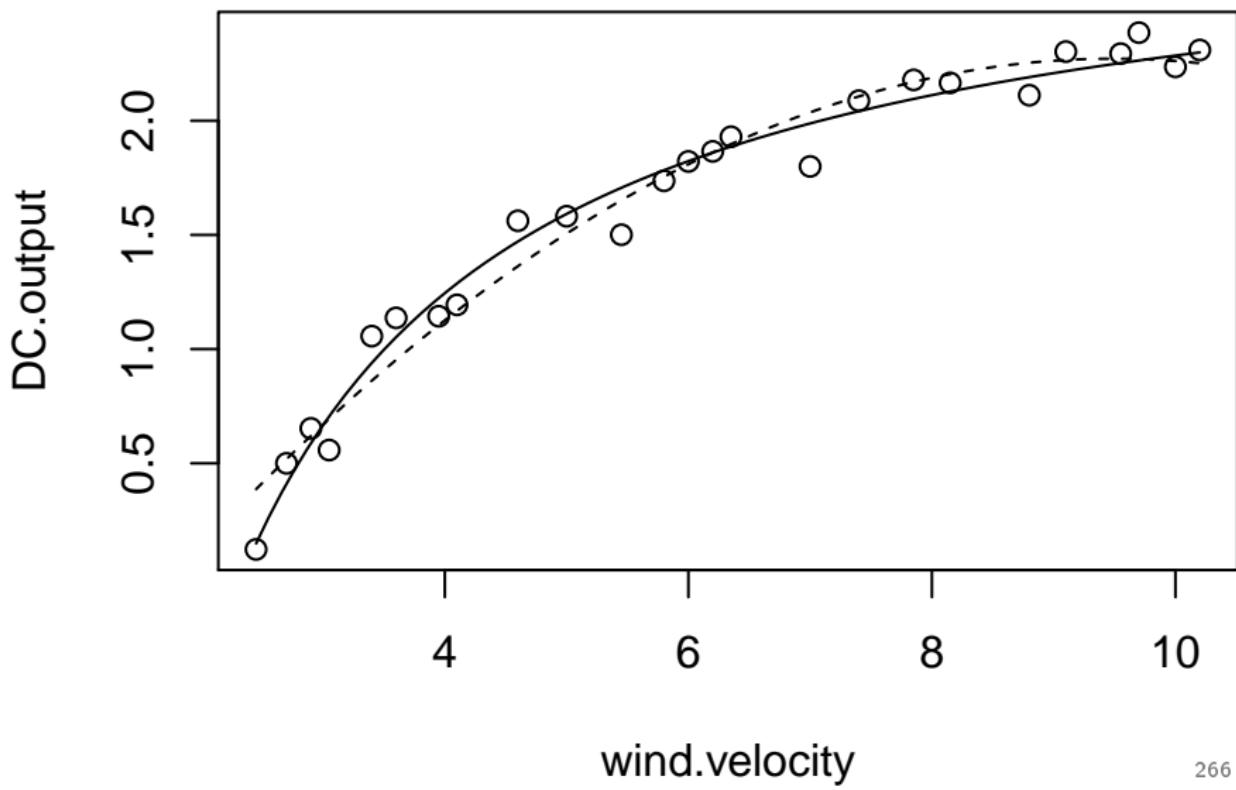
- ▶ Residual plot not bad. But residuals go up to 0.10 and down to -0.20, suggesting possible skewness (not normal). I think it's OK overall.
- ▶ Trend on scatterplot: something like this. Plot parabola model, dashed, for comparison.

```
R> plot(DC.output~wind.velocity)
R> lines(spline(wind.velocity,fit3))
R> lines(spline(wind.velocity,fit2),lty="dashed")
```

Strategy as before:

- ▶ make scatterplot
- ▶ plot the asymptote model predictions (model #3)
- ▶ plot the parabola model predictions using a dashed curve (model #2)

## Scatterplot with fitted curves



## Comments

- ▶ Predictions are very similar.
- ▶ Predictions from asymptote model as good, and from simpler model, so go with those.
- ▶ Go back to model summary:

## Asymptote model summary

```
R> summary(windmill3.lm)
```

Call:

```
lm(formula = DC.output ~ wind.pace)
```

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 268 / 653

## 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.
- ▶ Eyeballed 95% CI for asymptote:  $3 \pm 2(0.05) = (2.9, 3.1)$ .
- ▶ 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 `windmill3.lm` 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 0.5 to 16 (steps of 0.5), and predict `DC.output` according to the two models:

```
R> wv=seq(0.5,16,0.5)
```

```
R> wv
```

```
[1] 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5  
[16] 8.0 8.5 9.0 9.5 10.0 10.5 11.0 11.5 12.0 12.5 13.0  
[31] 15.5 16.0
```

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

## Setting up data frames (1)

- ▶ Parabola model had `wind.velocity` and `vel2`, wind velocity squared, so we need:

```
R> wind.vel.pred=data.frame(wind.velocity=wv,vel2=wv^2)
R> head(wind.vel.pred)
```

	wind.velocity	vel2
1	0.5	0.25
2	1.0	1.00
3	1.5	2.25
4	2.0	4.00
5	2.5	6.25
6	3.0	9.00

## Setting up data frames (2)

- ▶ Asymtote model had just `wind.pace`, which was one over wind velocity:

```
R> wind.pace.pred = data.frame(wind.pace=1/wv)  
R> head(wind.pace.pred)
```

```
wind.pace  
1 2.0000000  
2 1.0000000  
3 0.6666667  
4 0.5000000  
5 0.4000000  
6 0.3333333
```

## Doing predictions

- ▶ predict needs two things: a fitted model object like `windmill2.lm`, and data frame of values to predict from. Result is a vector of predictions as long as `wv` was:

```
R> pred.2=predict(windmill2.lm,wind.vel.pred)  
R> pred.3=predict(windmill3.lm,wind.pace.pred)
```

## Doing predictions

- ▶ predict needs two things: a fitted model object like `windmill2.lm`, and data frame of values to predict from. Result is a vector of predictions as long as `wv` was:

```
R> pred.2=predict(windmill2.lm,wind.vel.pred)
```

```
R> pred.3=predict(windmill3.lm,wind.pace.pred)
```

- ▶ Plot data, for comparison. Extend horizontal scale to include all values in `wv`, and make vertical scale go to 3 (asymptote):

```
R> plot(DC.output~wind.velocity,xlim=range(wv),ylim=c(0,3))
```

## Doing predictions

- ▶ predict needs two things: a fitted model object like `windmill2.lm`, and data frame of values to predict from. Result is a vector of predictions as long as `wv` was:

```
R> pred.2=predict(windmill2.lm,wind.vel.pred)  
R> pred.3=predict(windmill3.lm,wind.pace.pred)
```

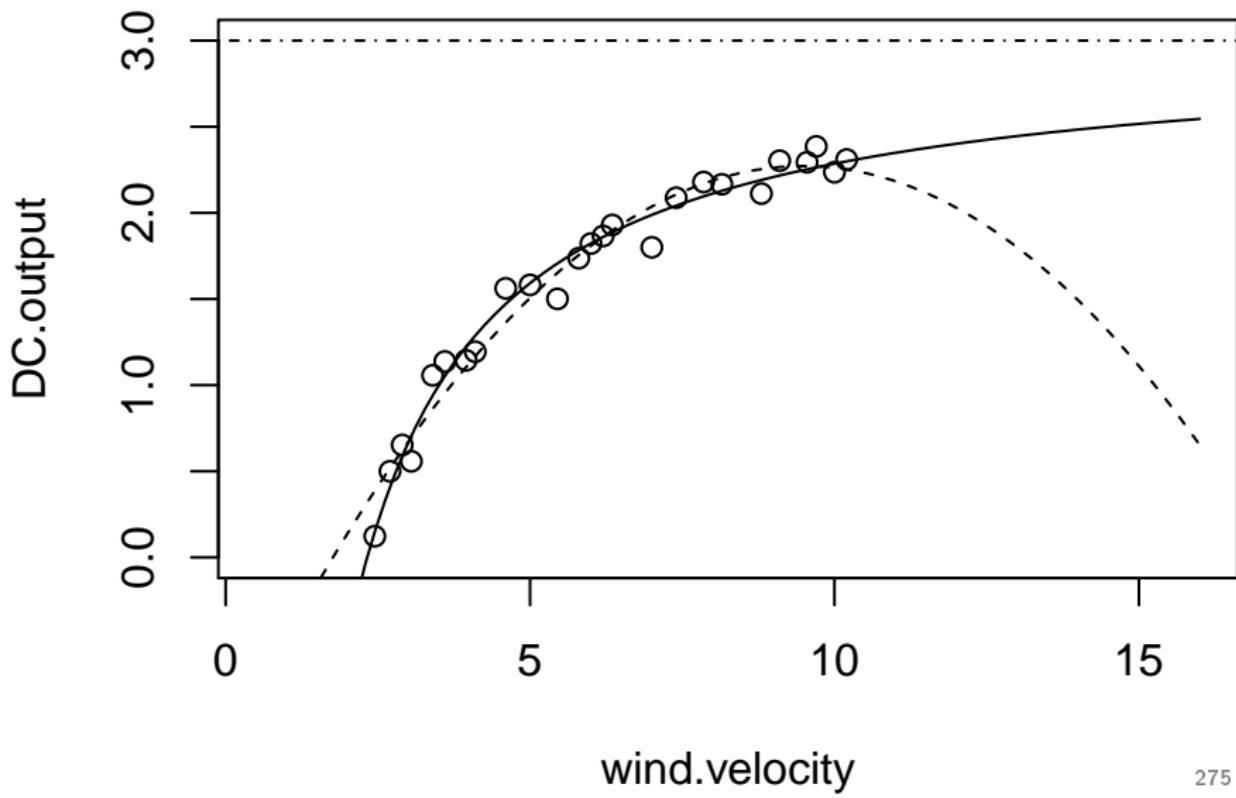
- ▶ Plot data, for comparison. Extend horizontal scale to include all values in `wv`, and make vertical scale go to 3 (asymptote):

```
R> plot(DC.output~wind.velocity,xlim=range(wv),ylim=c(0,3))
```

- ▶ For each model, make smooth curve of predictions, drawing parabola-model ones dashed. Also, add horizontal line at 3 (where we think asymptote is):

```
R> plot(DC.output~wind.velocity,xlim=range(wv),ylim=c(0,3))  
R> lines(spline(wv,pred.3))  
R> lines(spline(wv,pred.2),lty="dashed")  
R> abline(h=3,lty="dotdash")
```

## The plot



## Comments (1)

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

```
R> coef(windmill2.lm)
```

(Intercept)	wind.velocity	vel2
-1.15589824	0.72293590	-0.03812088

- ▶ Nope, goes to  $-1.15$  (intercept), actually significantly different from zero.

## Comments (2)

- ▶ What about asymptote model?

```
R> coef(windmill3.lm)
```

(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? Doesn't appear to be.)
- ▶ Best we can do now is to predict `DC.output` to be zero for small `wind.velocity`.
- ▶ This assumes a “threshold” wind velocity below which no electricity is generated at all.

## Summary

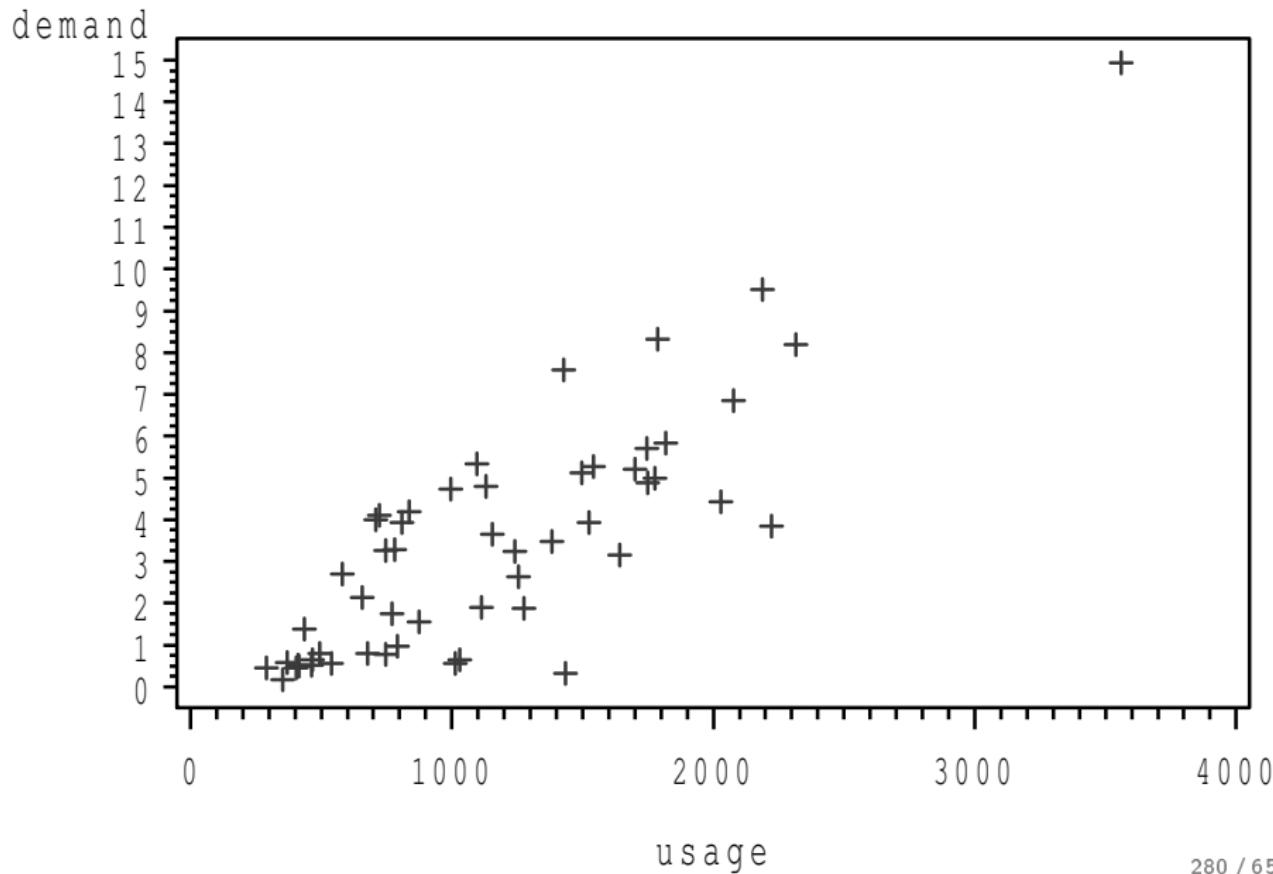
- ▶ Often, in data analysis, there is no completely satisfactory conclusion, as here.
- ▶ Have to settle for model that works OK, with restrictions.
- ▶ Always something else you can try.
- ▶ At some point you have to say “I stop.”

## 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:

```
SAS> data util;  
SAS>   infile 'utility.txt';  
SAS>   input usage demand;  
SAS>  
SAS> proc gplot;  
SAS>   plot demand*usage;
```

## 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:

```
SAS> proc reg;  
SAS>   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					
	Parameter	Standard			

Variable	DF	Estimate	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

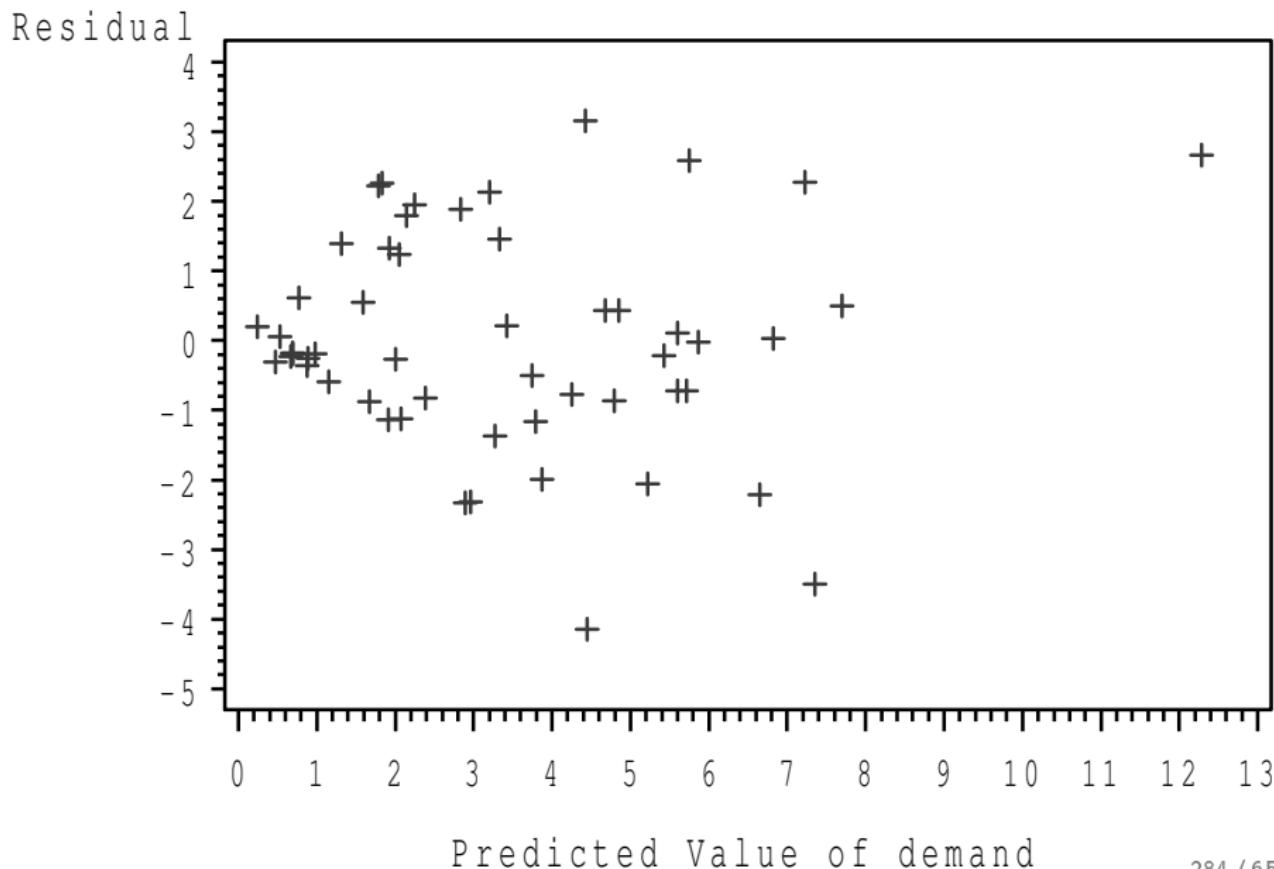
- ▶ R-squared 70%: not bad!
- ▶ Statistically significant slope: demand really does depend on usage.
- ▶ But should look at residuals.
- ▶ SAS way: obtain *output data set*, then look at that:

```
SAS> proc reg;  
SAS>   model demand=usage;  
SAS>   output out=util2 r=res p=fit;
```

- ▶ Output data set contains residuals, called `res`, fitted values, called `fit`. (Plus all same output as before.)
- ▶ Now make scatterplot of residuals against fitted values:

```
SAS> proc gplot;  
SAS>   plot res*fit;
```

## Residual plot



## Comments

- ▶ No trend
- ▶ 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

## 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:

```
SAS> proc transreg data=util;
```

```
SAS>   model boxcox(demand)=identity(usage);
```

# Output

The TRANSREG Procedure

Box-Cox Transformation Information for demand

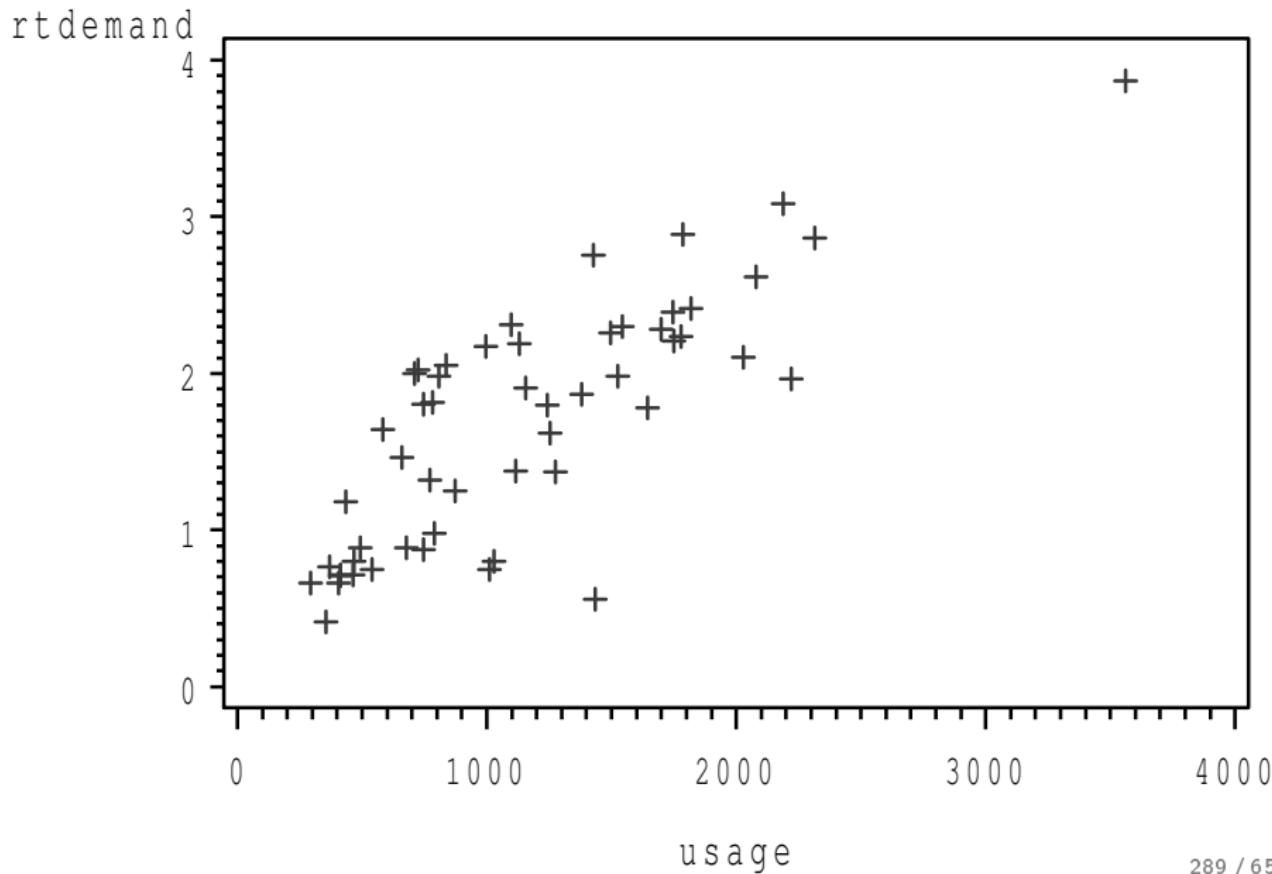
Lambda	R-Square	Log Like
-3.00	0.05	-285.806
-2.75	0.05	-256.529
-2.50	0.06	-227.764
-2.25	0.07	-199.652
-2.00	0.09	-172.389
-1.75	0.12	-146.248
-1.50	0.15	-121.590
-1.25	0.20	-98.852
-1.00	0.25	-78.489
-0.75	0.32	-60.876
-0.50	0.39	-46.203
-0.25	0.46	-34.473
0.00	0.53	-25.618
0.25	0.59	-19.687
0.50 +	0.65	-17.023 <
0.75	0.69	-18.287 *
1.00	0.70	-24.150
1.25	0.70	-34.686
1.50	0.68	-49.133
1.75	0.64	-66.342
2.00	0.60	-85.312
2.25	0.56	-105.376

## 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. 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:

```
SAS> data trans;  
SAS>   set util;  
SAS>   rtDemand=sqrt(demand);  
SAS>  
SAS> proc gplot;  
SAS>   plot rtDemand*usage;
```

## 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:

```
SAS> proc reg;  
SAS>   model rtDemand=usage;  
SAS>   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

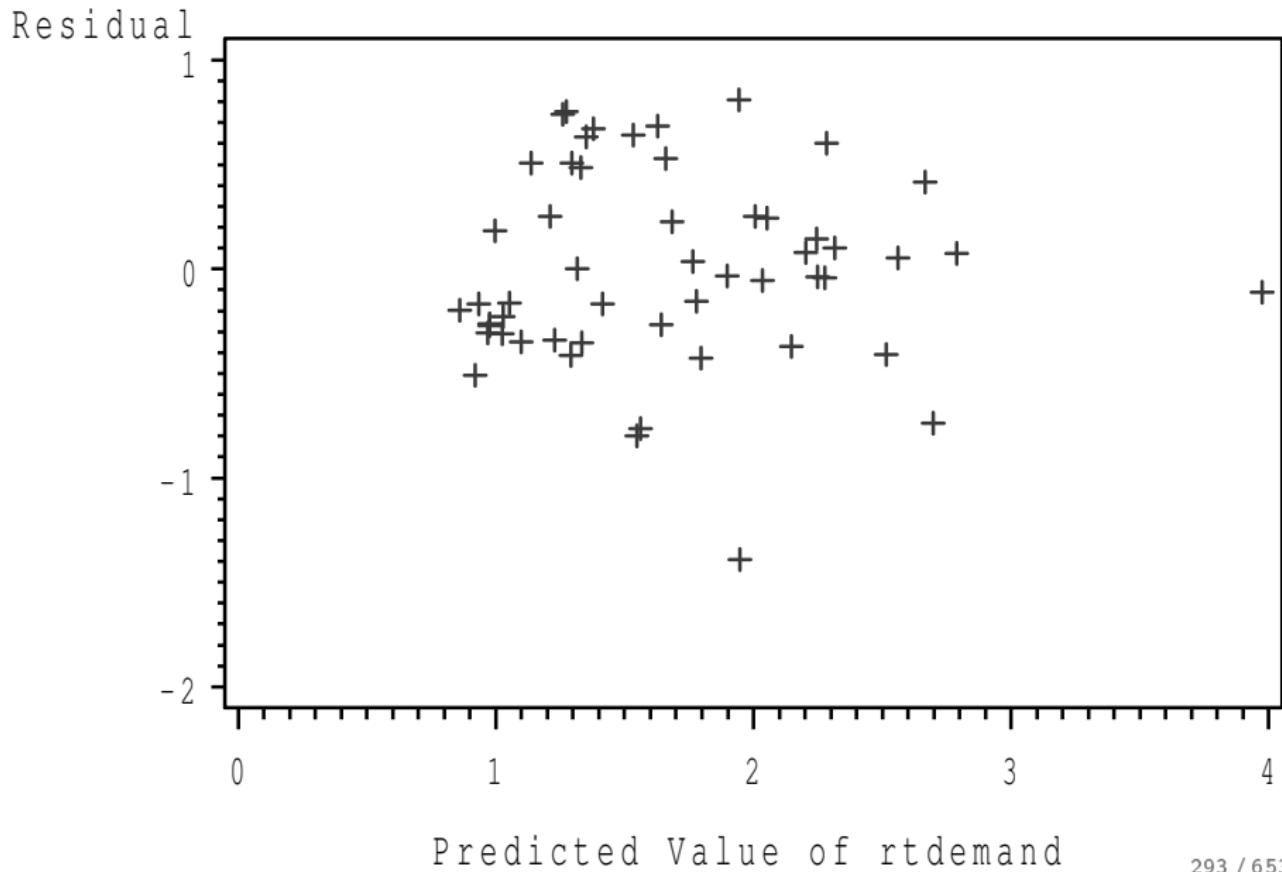
Variable	DF	Parameter Estimate	Standard Error	t Value	Pr >  t
Intercept	1	0.58223	0.12993	4.48	<.0001
usage	1	0.00095286	0.00009824	9.70	<.0001

## Comments

- ▶ R-squared actually decreased (from 70% to 65%).
- ▶ Slope still strongly significant.
- ▶ Should take a look at residuals now:

```
SAS> proc gplot;  
SAS> plot res*fit;
```

## Residual plot for 2nd regression



## Comments; scatter plot with line on it

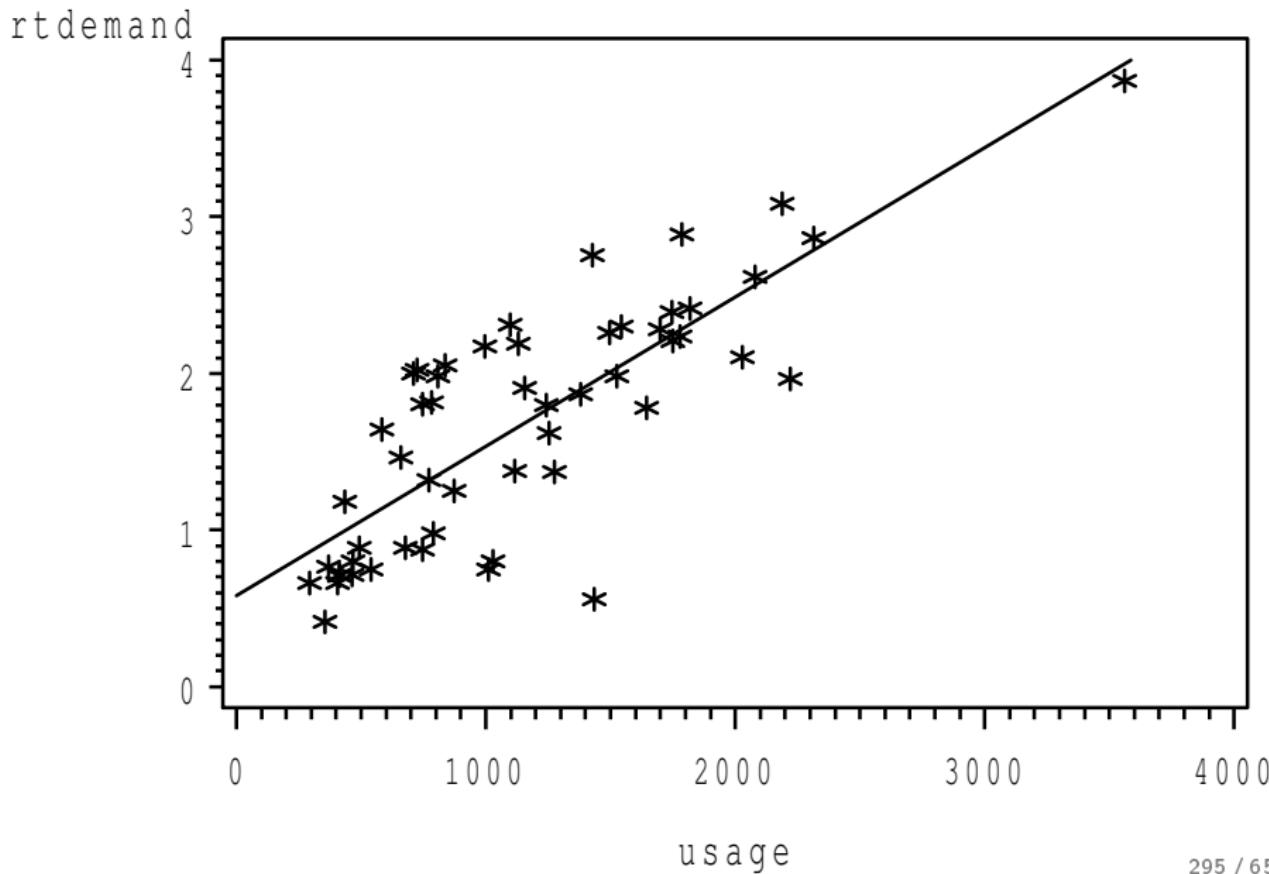
- ▶ Better. No trends, constant variability.
- ▶ One mildly suspicious outlier at the bottom.
- ▶ Can trust this regression.
- ▶ Better a lower R-squared from a regression we can trust than a higher one from one we cannot.
- ▶ Make scatterplot of rtDemand against usage with regression line on it. Uses ideas from SAS-stuff chapter of notes. The idea of the symbol1 is to plot the points in black stars (c=black, v=star) and then to draw a regression line (i=r1) using a solid line (l=1).

```
SAS> symbol1 c=black i=r1 l=1 v=star;
```

```
SAS> proc gplot;
```

```
SAS> plot rtDemand*usage;
```

## Scatterplot with fitted line



## Predictions

- ▶ When we transformed the response variable, have to think carefully about predictions. Using `usage=1000`:

```
R> int=0.58223
```

```
R> slope=0.00095286
```

```
R> pred=int+slope*1000
```

```
R> 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*:

```
R> pred^2
```

```
[1] 2.356501
```

## More predictions

- ▶ For usage 1000, 2000, 3000 all at once:

```
R> usage=c(1000,2000,3000)
```

```
R> rt.demand=int+slope*usage
```

```
R> demand=rt.demand^2
```

```
R> 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.

## 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 There were two data collection periods. run is 1 for run 1, 0 for run 2.
- ▶ rut.depth is response. Does it depend on other variables, and how?
- ▶ In R this time.

## Getting set up

- ▶ Read in data:

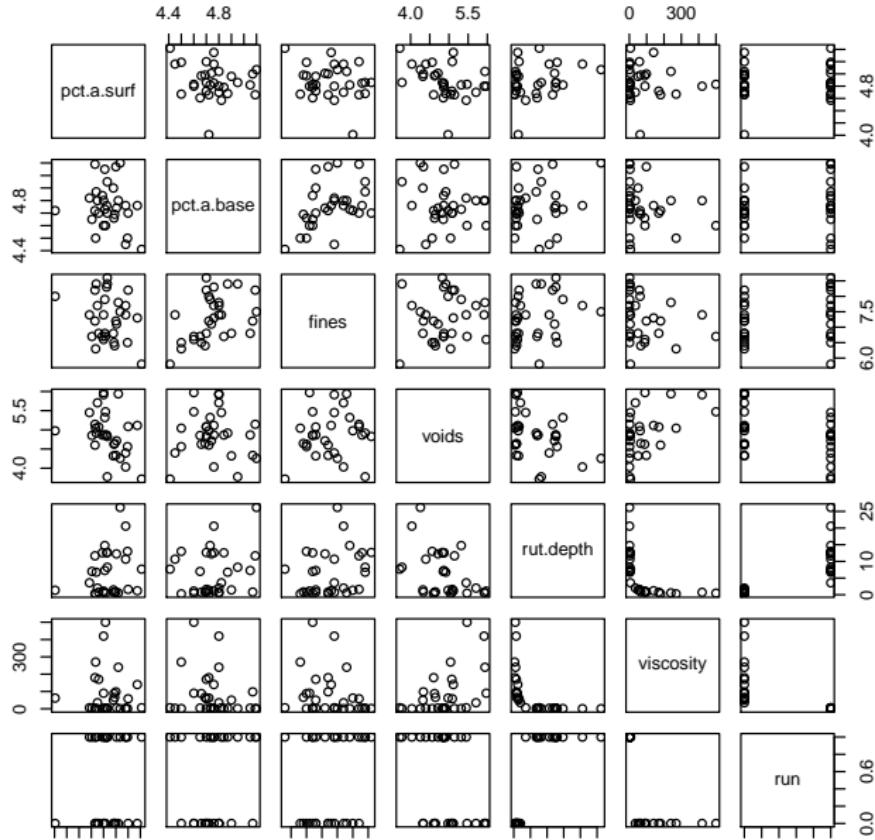
```
R> asphalt=read.table("asphalt.txt",header=T)
R> head(asphalt)
```

	pct.a.surf	pct.a.base	fines	voids	rut.depth	viscosity	run
1	4.68	4.87	8.4	4.916	6.75	2.8	1
2	5.19	4.50	6.5	4.563	13.00	1.4	1
3	4.82	4.73	7.9	5.321	14.75	1.4	1
4	4.85	4.76	8.3	4.865	12.60	3.3	1
5	4.86	4.95	8.4	3.776	8.25	1.7	1
6	5.16	4.45	7.4	4.397	10.67	2.9	1

- ▶ Quantitative variables with one response: multiple regression.
- ▶ Some issues here that don't come up in "simple" regression; handle as we go. (STAB27 ideas.)
- ▶ Too many variables for scatterplot, so do *lots*. What does plotting a data frame do?

```
R> plot(asphalt)
```

# Pairs plot



## Interpreting the pairs plot

- ▶ Array of scatterplots, one for each pair of variables.
- ▶ Get rough idea of what's going on.
- ▶ Look along 5th row to see relationships with `rut.depth`.
- ▶ Trends weak at best:
  - ▶ Downward (nonlinear) trend with viscosity
  - ▶ upward trend with run: most of high obs. in run 1.
  - ▶ Also: viscosity and run are related. Hard to know whether `rut.depth` really depends on viscosity or run or both. Confounding.
- ▶ Non-linearity of `rut.depth`-viscosity relationship should concern us.
- ▶ Take this back to asphalt engineer: suggests *log* of viscosity.
- ▶ Check that relationship of log-viscosity with `rut.depth` more nearly linear:

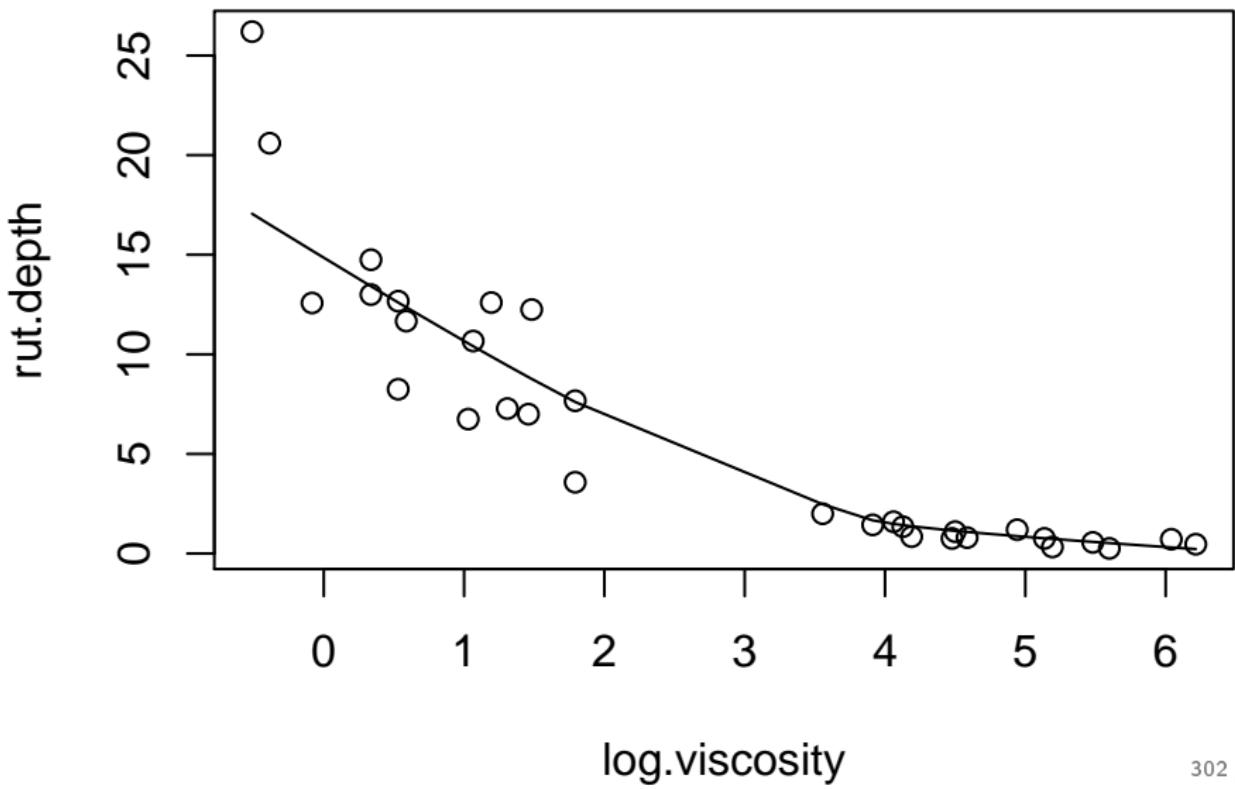
```
R> attach(asphalt)
```

```
R> log.viscosity=log(viscosity)
```

```
R> plot(rut.depth~log.viscosity)
```

```
R> lines(lowess(rut.depth~log.viscosity))
```

## Log-viscosity vs. rut depth



## 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, explanatories on right joined by plusses:

```
R> rut.lm=lm(rut.depth~pct.a.surf+pct.a.base+fines+
R>     voids+log.viscosity+run)
R> summary(rut.lm)
```

## Regression output

Call:

```
lm(formula = rut.depth ~ pct.a.surf + pct.a.base + fines + voids +  
log.viscosity + run)
```

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 look at residual plot, try taking out *everything* non-significant, and test whether that's justifiable.

```
R> rut2.lm=lm(rut.depth~log.viscosity)
```

```
R> summary(rut2.lm)
```

## Taking out everything but log.viscosity

Call:

```
lm(formula = rut.depth ~ log.viscosity)
```

Residuals:

Min	1Q	Median	3Q	Max
-5.487	-1.635	-0.538	1.624	10.816

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	13.9827	0.9315	15.01	3.29e-15 ***
log.viscosity	-2.7434	0.2688	-10.21	4.15e-11 ***
---				
Signif. codes:	0 ‘***’	0.001 ‘**’	0.01 ‘*’	0.05 ‘.’
	0.1 ‘ ’	1		

Residual standard error: 3.204 on 29 degrees of freedom

Multiple R-squared: 0.7822, Adjusted R-squared: 0.7747

F-statistic: 104.2 on 1 and 29 DF, p-value: 4.152e-11

## What next?

- ▶ R-squared hasn't dropped much (81%–78%)
- ▶ suggesting taking out all those other variables hasn't done much damage
- ▶ but we haven't *tested* that.
- ▶ Do this using `anova` on two models, smaller first:
  - ▶ `rut.lm` has everything in it
  - ▶ `rut2.lm` has just `log.viscosity`, smaller.
- ▶ Null: two models equally good. Alternative: bigger model better.
- ▶ This:  
`R> anova(rut2.lm,rut.lm)`

## Results of anova

### Analysis of Variance Table

Model 1: rut.depth ~ log.viscosity

Model 2: rut.depth ~ pct.a.surf + pct.a.base + fines + voids + run

	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
1	29	297.62				
2	24	265.10	5	32.525	0.5889	0.7084

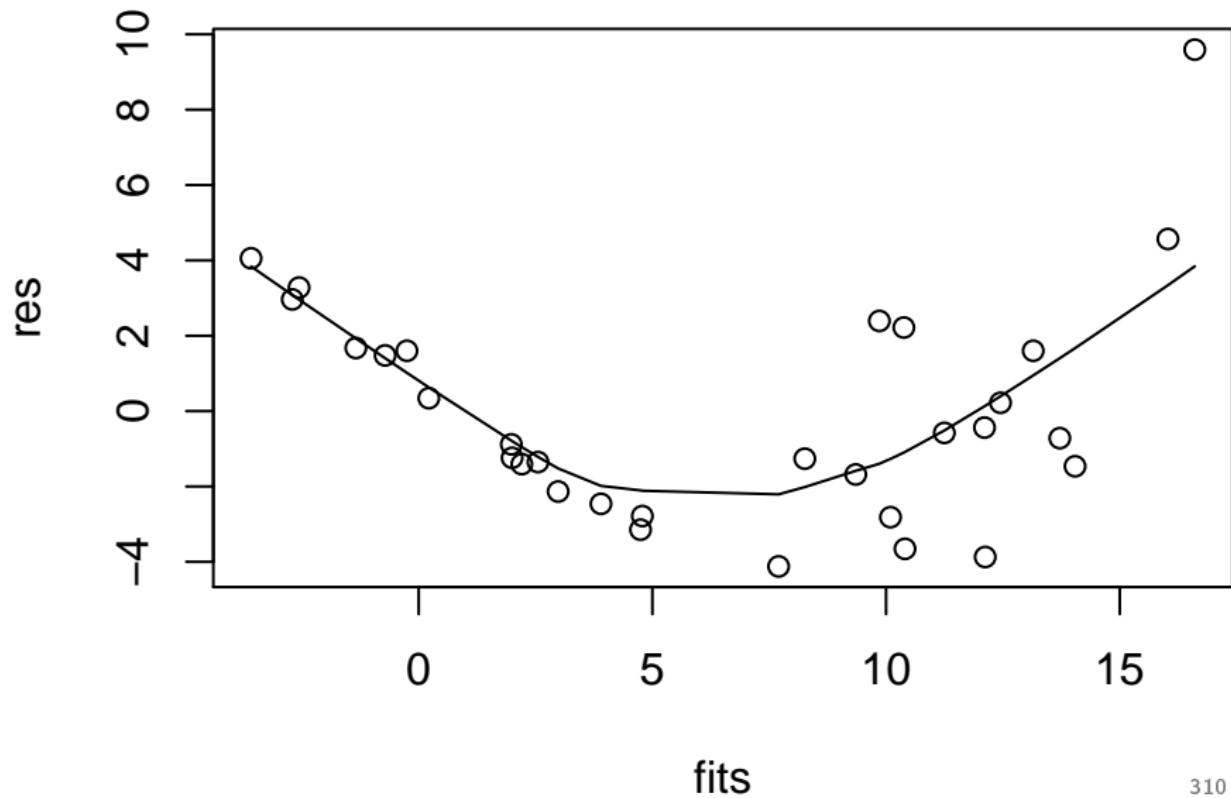
- ▶ P-value not at all small. Cannot reject null.
- ▶ Go with smaller model.
- ▶ Do not need extra complication of larger model.
- ▶ (Showing you this mechanism in case you need it.)

But... we didn't check residuals yet!

Calculate residuals, fitted values; plot them with lowess curve:

```
R> fits=fitted(rut.lm)
R> res=resid(rut.lm)
R> plot(fits,res)
R> lines(lowess(fits,res))
```

## The residual plot

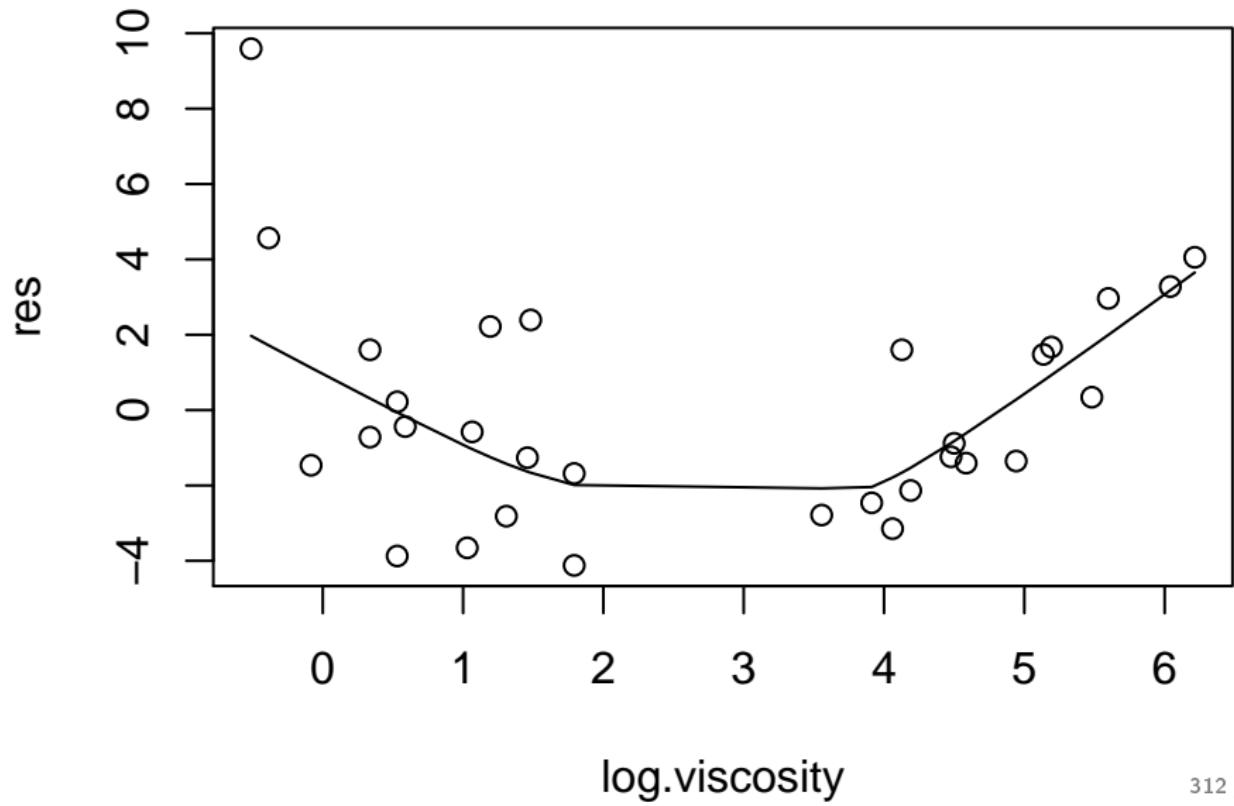


## Bendiness!

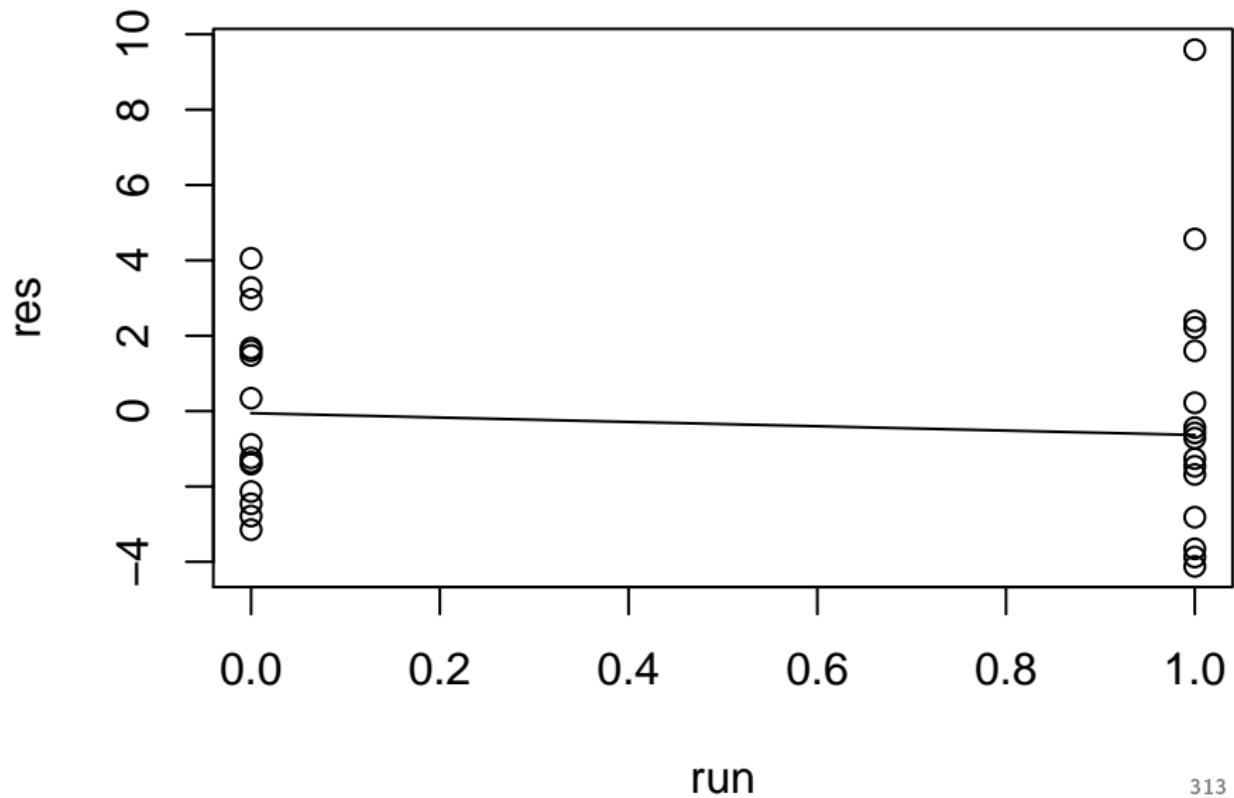
- ▶ Curved relationship: should we fix by transforming response?
- ▶ But, can also plot residuals against **explanatory variables** (or anything else).
- ▶ Maybe transforming one or more explanatory variables might be better.
- ▶ Art, not science; may be no one best way to go.
- ▶ Should look at residuals against all six. Just do some.
- ▶ Start with residuals against log.viscosity:

```
R> plot(log.viscosity,res)
R> lines(lowess(log.viscosity,res))
```

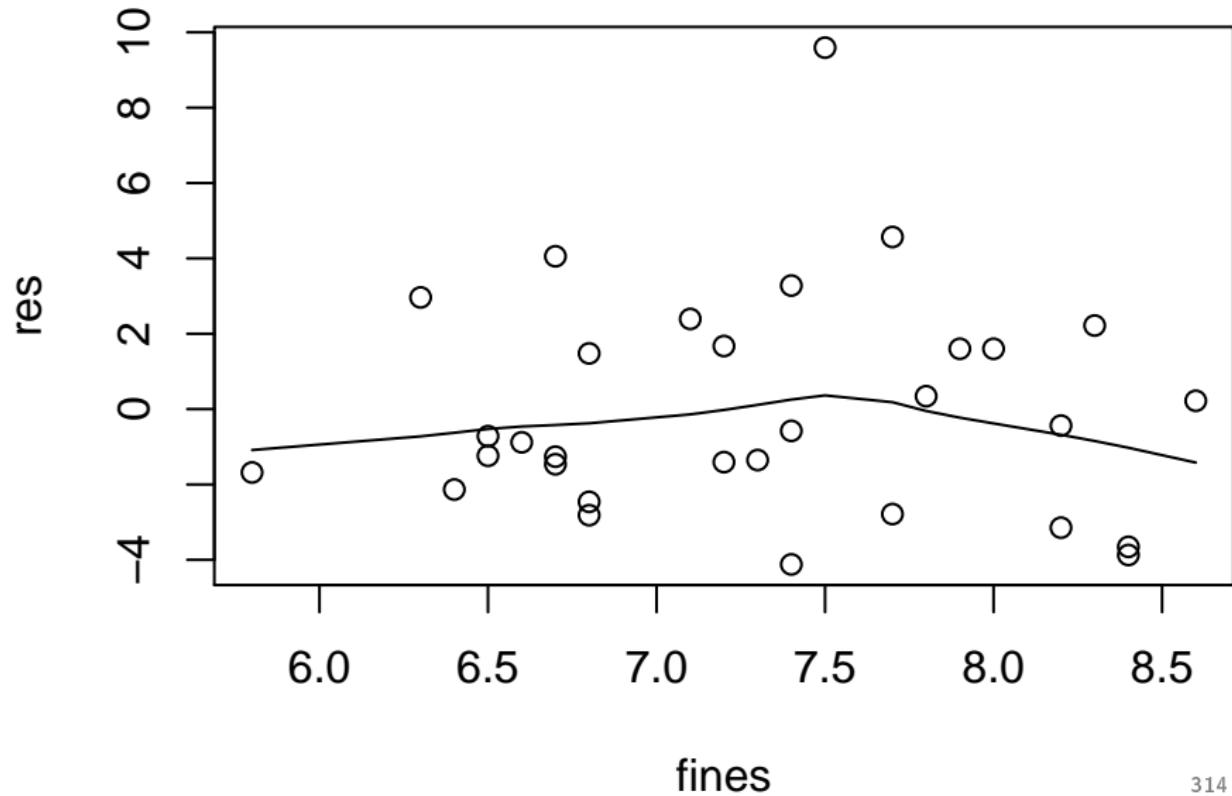
## Residuals against log.viscosity



## Residuals against run



## Residuals against fines



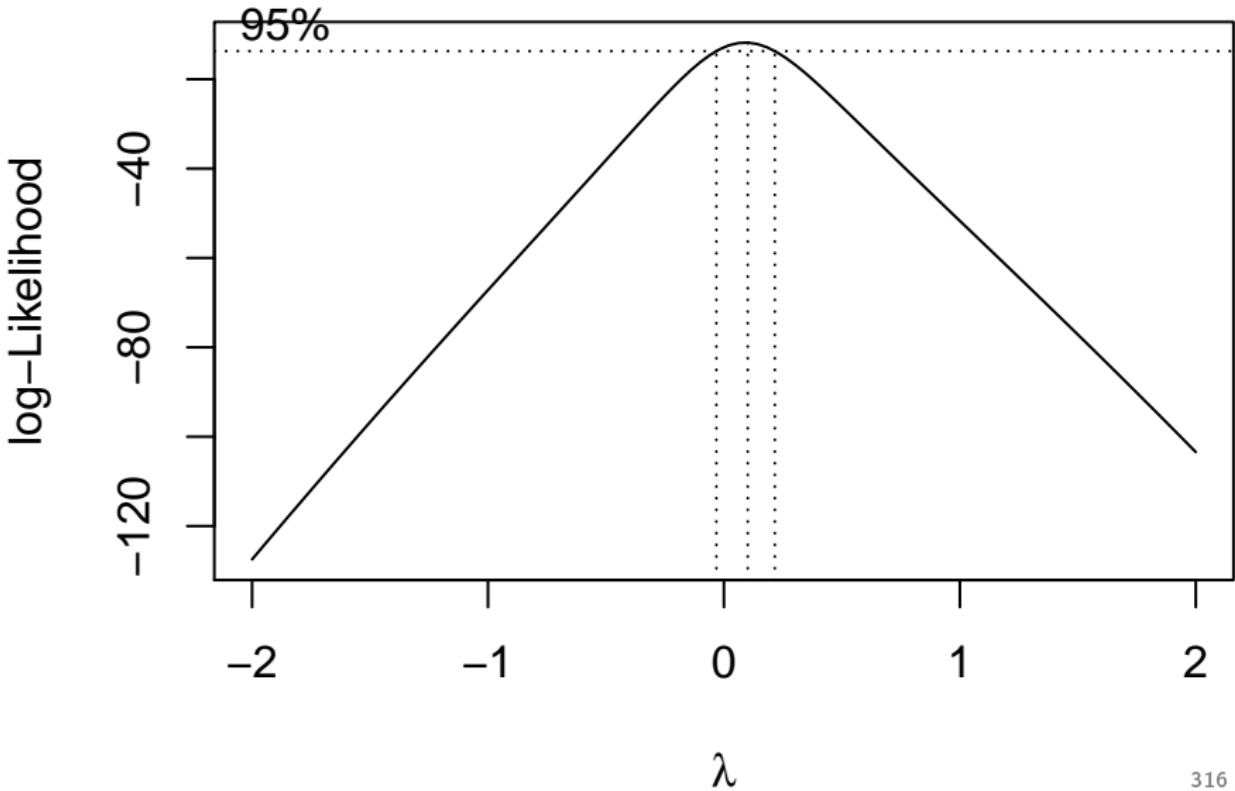
## Comments

- ▶ Residuals vs. log.viscosity: definite curve.
- ▶ Residuals vs. run: no curve (only 2 values of run), but greater variability when run=1.
- ▶ Residuals vs. fines: possible curve.
- ▶ My take: curves everywhere, so fix by transforming *response*.
- ▶ If only some plots showed curves, try transforming “offending” explanatories.
- ▶ What transformation of rut.depth? Could try log, or use Box-Cox again. R’s boxcox produces a plot (next page).

```
R> library(MASS)
```

```
R> boxcox(rut.depth~pct.a.surf+pct.a.base+fines+voids+
R>     log.viscosity+run)
```

## 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 new response variable is

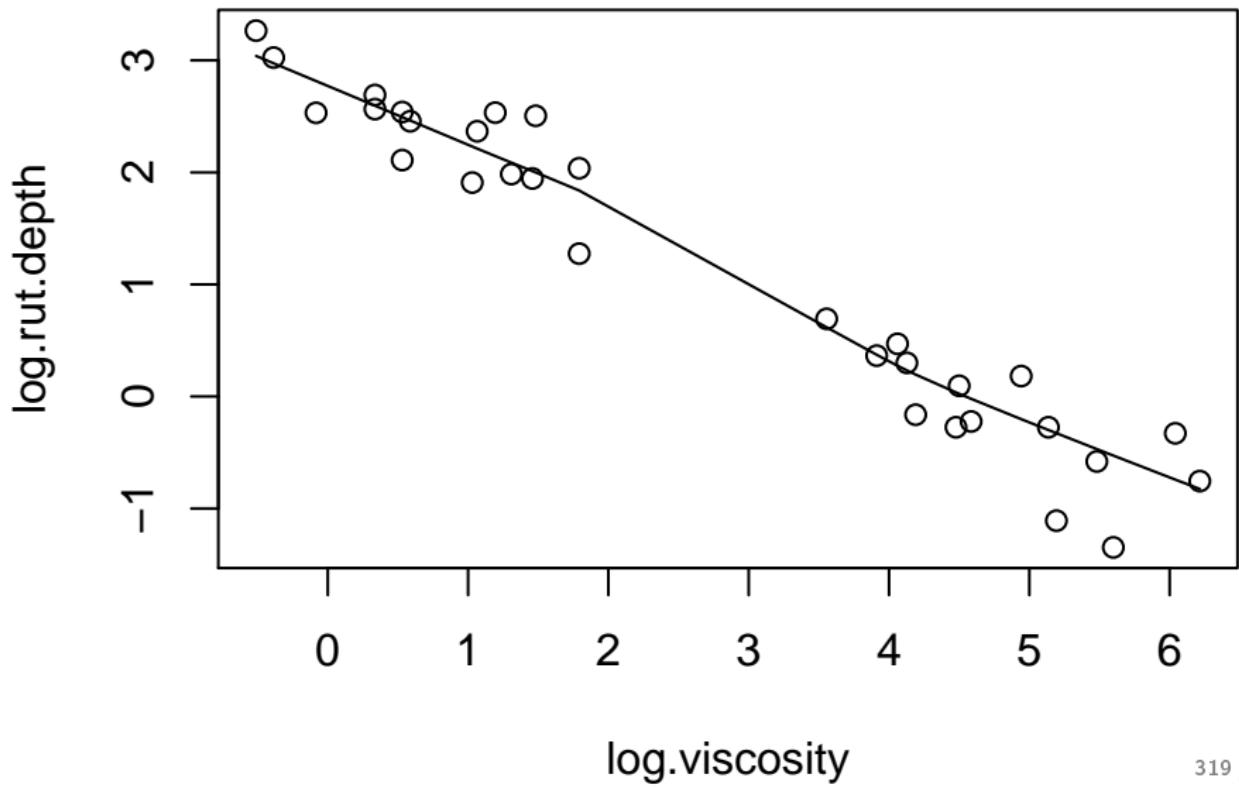
```
R> log.rut.depth=log(rut.depth)
```

## Modelling with transformed response

- ▶ Start from the beginning again.
- ▶ Model `log.rut.depth` in terms of everything else, see what can be removed.
- ▶ Check resulting residual plots.
- ▶ First, though: have we straightened out relationship with `log.viscosity`?
- ▶ 

```
R> plot(log.viscosity, log.rut.depth)
R> lines(lowess(log.viscosity, log.rut.depth))
```

## log.viscosity against log.rut.depth



## Comments

- ▶ Previous plot *much* straighter.
- ▶ Fit everything now:

```
R> log.rut.lm=lm(log.rut.depth~pct.a.surf+pct.a.base+
R>       fines+voids+log.viscosity+run)
R> summary(log.rut.lm)
```

# Output

Call:

```
lm(formula = log.rut.depth ~ pct.a.surf + pct.a.base + fines +  
    voids + log.viscosity + run)
```

Residuals:

Min	1Q	Median	3Q	Max
-0.53072	-0.18563	-0.00003	0.20017	0.55079

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	-1.57299	2.43617	-0.646	0.525
pct.a.surf	0.58358	0.23198	2.516	0.019 *
pct.a.base	-0.10337	0.36891	-0.280	0.782
fines	0.09775	0.09407	1.039	0.309
voids	0.19885	0.12306	1.616	0.119
log.viscosity	-0.55769	0.08543	-6.528	9.45e-07 ***
run	0.34005	0.33889	1.003	0.326
---				
Signif. codes:	0 ‘***’	0.001 ‘**’	0.01 ‘*’	0.05 ‘.’
	1			

Residual standard error: 0.3088 on 24 degrees of freedom

## Taking out everything non-significant

Two ways to go now. First: remove everything but pct.a.surf and log.viscosity:

```
R> log.rut2.lm=lm(log.rut.depth~pct.a.surf+log.viscosity)
```

and check that removing all those variables wasn't too much:

```
R> anova(log.rut2.lm,log.rut.lm)
```

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 + void  
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

- ▶ Safe this time, but might not have been.
- ▶ Taking out some x's might make others significant.

## Take out least significant in sequence

```
R> summary(log.rut.lm)
```

Call:

```
lm(formula = log.rut.depth ~ pct.a.surf + pct.a.base + fines +  
    voids + log.viscosity + run)
```

Residuals:

Min	1Q	Median	3Q	Max
-0.53072	-0.18563	-0.00003	0.20017	0.55079

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	-1.57299	2.43617	-0.646	0.525
pct.a.surf	0.58358	0.23198	2.516	0.019 *
pct.a.base	-0.10337	0.36891	-0.280	0.782
fines	0.09775	0.09407	1.039	0.309
voids	0.19885	0.12306	1.616	0.119
log.viscosity	-0.55769	0.08543	-6.528	9.45e-07 ***
run	0.34005	0.33889	1.003	0.326
---				

Take out pct.a.base:

```
R> log.rut3.lm=lm(log.rut.depth~pct.a.surf+fines+voids+log.viscosity+run)
R> summary(log.rut3.lm)
```

Call:

```
lm(formula = log.rut.depth ~ pct.a.surf + fines + voids + log.viscosity + run)
```

Residuals:

Min	1Q	Median	3Q	Max
-0.51610	-0.18785	-0.02248	0.18364	0.57160

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	-2.07850	1.60665	-1.294	0.2076
pct.a.surf	0.59299	0.22526	2.632	0.0143 *
fines	0.08895	0.08701	1.022	0.3165
voids	0.20047	0.12064	1.662	0.1091
log.viscosity	-0.55249	0.08184	-6.751	4.48e-07 ***
run	0.35977	0.32533	1.106	0.2793
---				

Take out fines:

```
R> log.rut4.lm=lm(log.rut.depth~pct.a.surf+voids+log.viscosity  
R> summary(log.rut4.lm)
```

Call:

```
lm(formula = log.rut.depth ~ pct.a.surf + voids + log.viscosity  
    run)
```

Residuals:

Min	1Q	Median	3Q	Max
-0.57275	-0.20080	0.01061	0.17711	0.59774

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	-1.25533	1.39147	-0.902	0.3753
pct.a.surf	0.54837	0.22118	2.479	0.0200 *
voids	0.23188	0.11676	1.986	0.0577 .
log.viscosity	-0.58039	0.07723	-7.516	5.59e-08 ***
run	0.29468	0.31931	0.923	0.3646

Take out run:

```
R> log.rut5.lm=lm(log.rut.depth~pct.a.surf+voids+log.viscosity)
R> summary(log.rut5.lm)
```

Call:

```
lm(formula = log.rut.depth ~ pct.a.surf + voids + log.viscosity)
```

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 ‘.’ 0.1 326 / 6531

## 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, than by taking out 4 at once:

```
R> coef(log.rut2.lm)
```

	(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:  
`R> step(log.rut.lm,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

```
R> library(leaps)
R> leaps=regsubsets(log.rut.depth~pct.a.surf+pct.a.base+fines+voids+log.viscosity+runoff, nbest=10)
R> s=summary(leaps)
R> cbind(s$rsq,s$which)
```

	(Intercept)	pct.a.surf	pct.a.base	fines	voids	log.viscosity	runoff
1	0.9452562	1	0	0	0	0	1 0
1	0.8624107	1	0	0	0	0	0 1
2	0.9508647	1	1	0	0	0	1 0
2	0.9479541	1	0	0	0	1	1 0
3	0.9578631	1	1	0	0	1	1 0
3	0.9534561	1	1	0	1	0	1 0
4	0.9591996	1	1	0	0	1	1 1
4	0.9589206	1	1	0	1	1	1 0
5	0.9608365	1	1	0	1	1	1 1
5	0.9593265	1	1	1	1	1	1 0
6	0.9609642	1	1	1	1	1	1 1

## 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).
- ▶ One solution (STAB27): *adjusted R-squared*, where adding worthless variable makes it go *down*.

## All possible regressions, adjusted R-squared

```
R> cbind(s$adjr2, s$which)
```

	(Intercept)	pct.a.surf	pct.a.base	fines	voids	log.viscosity	run
1	0.9433685	1	0	0	0	0	1 0
1	0.8576662	1	0	0	0	0	0 1
2	0.9473550	1	1	0	0	0	1 0
2	0.9442365	1	0	0	0	1	1 0
3	0.9531812	1	1	0	0	1	1 0
3	0.9482845	1	1	0	1	0	1 0
4	0.9529226	1	1	0	0	1	1 1
4	0.9526007	1	1	0	1	1	1 0
5	0.9530038	1	1	0	1	1	1 1
5	0.9511918	1	1	1	1	1	1 0
6	0.9512052	1	1	1	1	1	1 1

Backward-elimination model comes out best again.

## 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.

## Part VIII

### Data munging

## Getting the data into the right format

- ▶ Data rarely reach us as we wish to use them:
  - ▶ Errors.
  - ▶ Values in wrong format.
  - ▶ Rows that should be columns or vice versa.
- ▶ Computer science: “munging”, re-arranging data into form you want.
- ▶ Often have to do this before we begin analysis.
- ▶ R and SAS have suitable tools.

## Handling data

- ▶ **Do not edit the data you receive; always work with a copy.**
- ▶ Create new folder for each project.
- ▶ Within that, create folder data to hold original data, **read-only**.
- ▶ Also create folders R and/or SAS, as appropriate.
- ▶ Start with snippets of code in main folder. Once they work, create larger scripts and move to R/SAS folders.
- ▶ See <http://nicercode.github.io/blog/2013-04-05-projects/>.
- ▶ When done, create file analysis.R (analysis.sas) that runs whole analysis.
- ▶ Create folder doc for report. (Might use R Markdown for this.)

## Be a computer scientist

- ▶ Comment everything. (Can use files README in various places documenting where everything from.)
- ▶ Don't use new or old in filenames. If necessary, create subfolder with today's date, like 2013-07-30, and copy everything there before changing it.
- ▶ Aim: be able to reproduce everything you did, even if some of the output gets lost.

## The cars data

- ▶ Use for illustration later.
- ▶ Measurements on a number of cars:
  - ▶ Car name (text)
  - ▶ Gas mileage (miles per US gallon)
  - ▶ Weight (US tons)
  - ▶ Cylinders in engine
  - ▶ Horsepower of engine
  - ▶ Country where car made (text).
- ▶ Make SAS “permanent data set” out of these in a moment.

## Permanent data sets (from SAS chapter)

- ▶ Can we read in data set *once* and not every time?
- ▶ Yes, use *filename* (in single quotes) when creating:

```
SAS> options ls=70;  
SAS> data 'cars';  
SAS> infile 'cars.txt' firstobs=2;  
SAS> input car $25. mpg weight cylinders hp country $;
```

- ▶ Car names max of 25 chars long. Country names max of 8, so no special treatment needed.
- ▶ SAS stores file called *cars.sas7bdat* (!) on *cms-chorus*.
- ▶ Whenever you need it, add *data='cars'* to a *proc* line.
- ▶ Can use subfolders, using syntax for *machine that SAS running on*. (See notes for what to do when SAS running under Windows.)
- ▶ Closing SAS breaks connection with data sets read in with *data* step. To get those back, need to run *data* step lines again.

## Checking data

- ▶ Direct check that data values are correct.
- ▶ Looking for outliers (suspicious values).
- ▶ Examining missing values.

## Direct checking

- ▶ Ideal: physically check that every value in data frame / dataset agrees with original source.
- ▶ Works for small data sets.
- ▶ If too much data, take *random sample* of rows and check those.
- ▶ Balance: checking more rows improves chances of finding errors, but takes you more time.
- ▶ Both SAS and R can take random samples of data to check.
- ▶ Illustrate by sampling 5 rows from my cars data set.

## Random sampling in SAS

```
proc surveyselect:
```

```
SAS> proc surveyselect data='cars' sampsize=5 out=cars1;
```

```
SAS>   id _all_;
```

```
SAS>
```

```
SAS> proc print;
```

Obs	car	mpg	weight	cylinders	hp	country
1	VW Rabbit	31.9	1.925	4	71	Germany
2	Buick Century Special	20.6	3.380	6	105	U.S.
3	Volvo 240 GL	17.0	3.140	6	125	Sweden
4	Dodge St Regis	18.2	3.830	8	135	U.S.
5	Mercury Grand Marquis	16.5	3.955	8	138	U.S.

## Random sampling in R

Already saw `sample` for randomization:

```
R> cars=read.csv("cars.csv",header=TRUE)
```

```
R> myrows=sample(nrow(cars),5)
```

```
R> myrows
```

```
[1] 30 28 29 3 17
```

```
R> cars[myrows,]
```

	Car	MPG	Weight	Cylinders	Horsepower	Country
30	Chevette	30.0	2.16	4	68	U.S.
28	Dodge St Regis	18.2	3.83	8	135	U.S.
29	Toyota Corona	27.5	2.56	4	95	Japan
3	Mercury Zephyr	20.8	3.07	6	85	U.S.
17	VW Scirocco	31.5	1.99	4	71	Germany

## Comments

- ▶ In each case, check sampled rows against data source (car magazine).
- ▶ Data source may also contain summaries of data, like mean, SD. Check these against data: if they are wrong, at least one error.
- ▶ Having right mean and SD not *proof* that data values correct, but increases confidence.

## Checking sample statistics

Might know that MPG has mean 24.76, min 15.5 and max 37.3:

```
SAS> proc means data='cars';
SAS>   var mpg;
```

The MEANS Procedure

Analysis Variable : mpg				
N	Mean	Std Dev	Minimum	Maximum
38	24.7605263	6.5473138	15.5000000	37.3000000

```
R> mean(cars$MPG)
```

```
[1] 24.76053
```

```
R> quantile(cars$MPG)
```

0%	25%	50%	75%	100%
15.500	18.525	24.250	30.375	37.300

Check. (Max and min easy to find from original data.)

## Looking for outliers

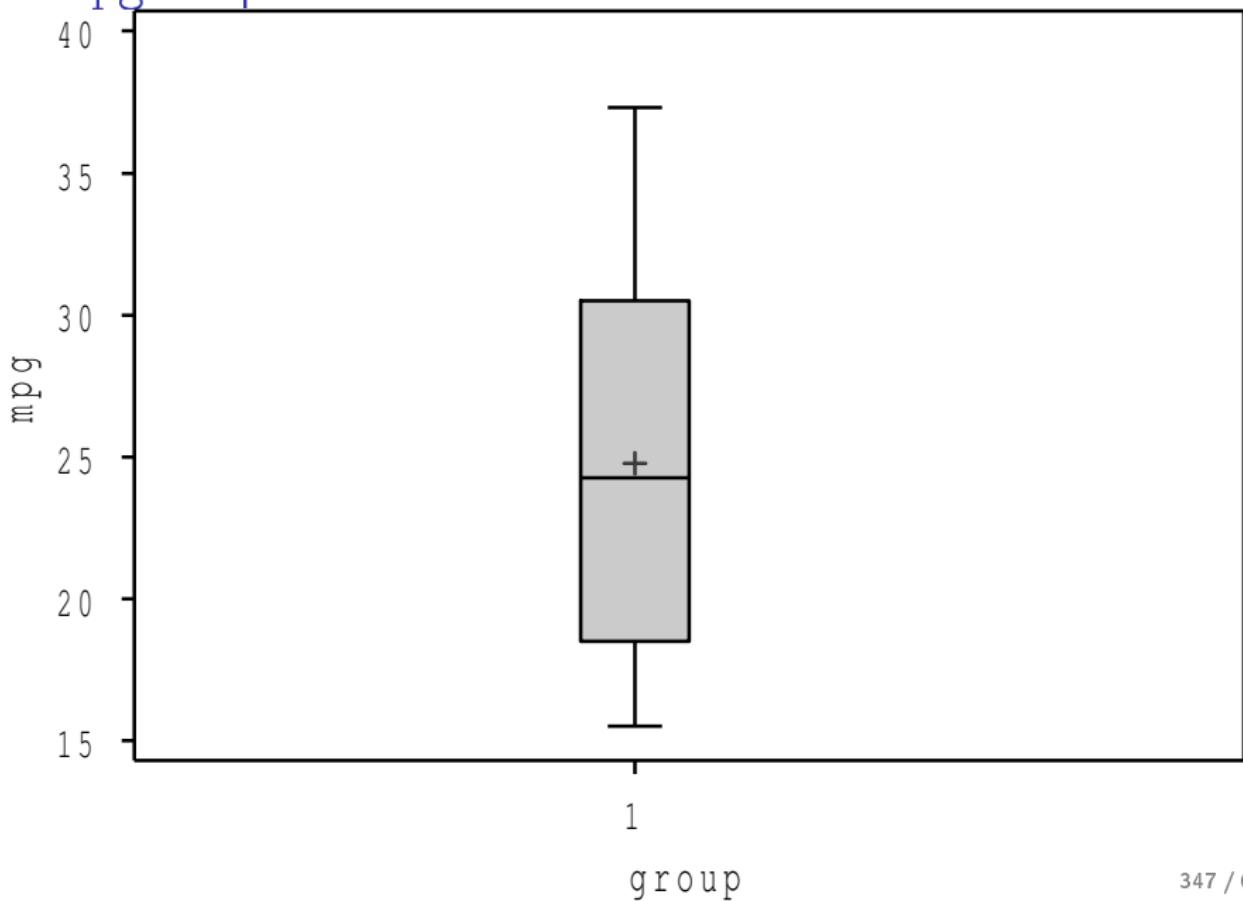
- ▶ Errors often show up as values suspiciously big or small: outliers.
- ▶ Errors that are outliers can distort means, SDs, correlations, regression slopes.
- ▶ But perfectly legitimate values might be outliers.
- ▶ Need to find and check outliers: might be legit, might be errors.

## Tool 1: the boxplot

- ▶ Check each variable in turn.
- ▶ In SAS, make a fake group variable if needed.
- ▶ Example: car mpg values:

```
SAS> data newcars;  
SAS>   set 'cars';  
SAS>   group=1;  
SAS>  
SAS> proc boxplot data=newcars;  
SAS>   plot mpg*group / boxtyle=schematic;
```

## Car mpg boxplot

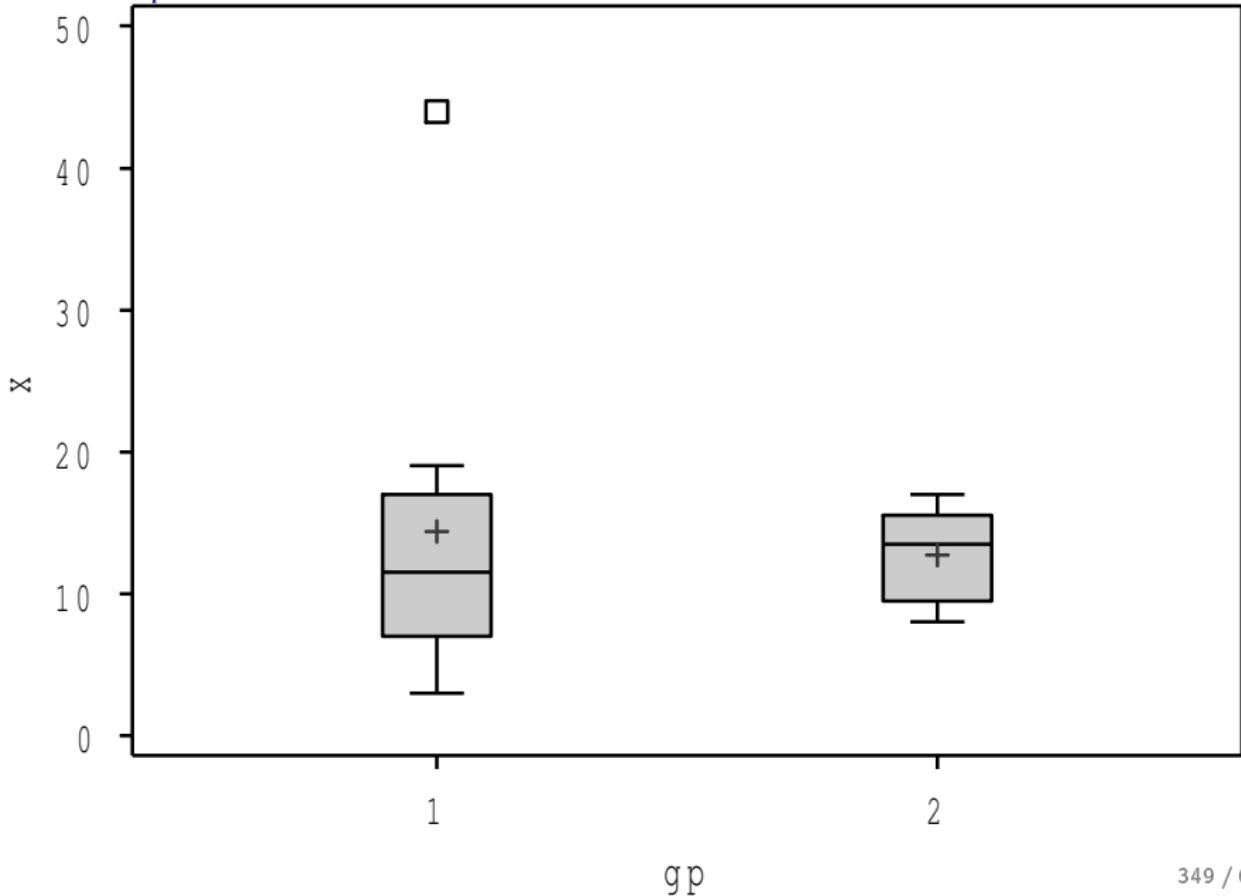


## Comments

- ▶ No outliers here.
- ▶ Data in groups: check each group separately.
- ▶ Another SAS example. Can you guess where the outlier is?

```
SAS> data mydata;  
SAS>   input gp x @@;  
SAS>   cards;  
SAS>   1 3 1 5 1 17 1 16 1 11 1 44 1 10 1 7 1 12 1 19  
SAS>   2 10 2 13 2 15 2 14 2 17 2 9 2 8 2 16  
SAS> ;  
SAS>  
SAS> proc boxplot;  
SAS>   plot x*gp / boxtype=schematic;
```

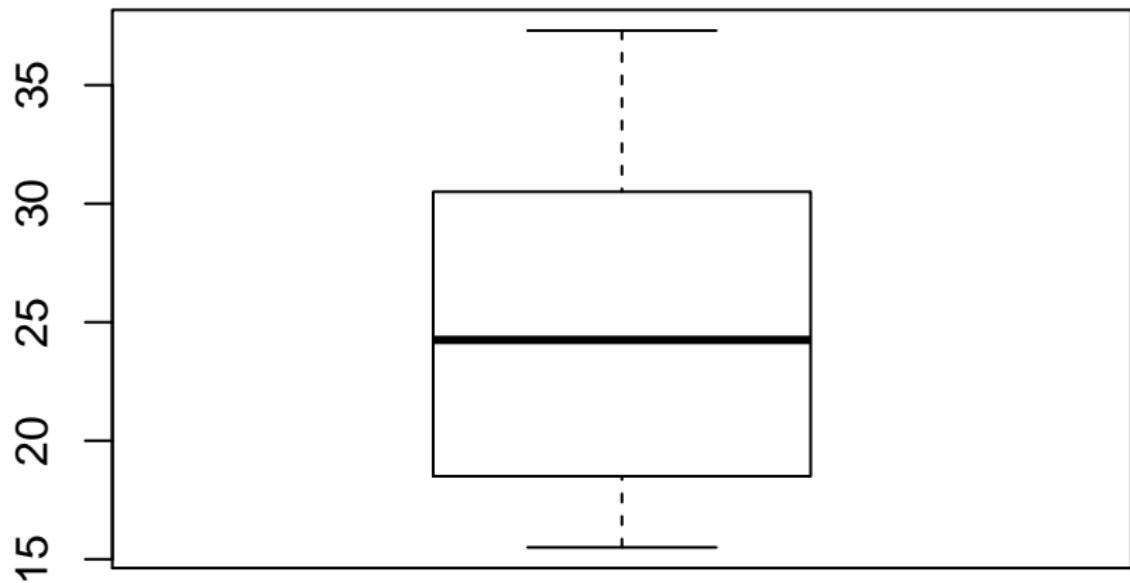
## The boxplot



## Comment and R with car MPG

- ▶ That value 44 should have been 14.
- ▶ R has no need for fake groups:  
*R> boxplot(cars\$MPG)*
- ▶ The MPG values have no outliers, as we see on next page.

## R boxplot



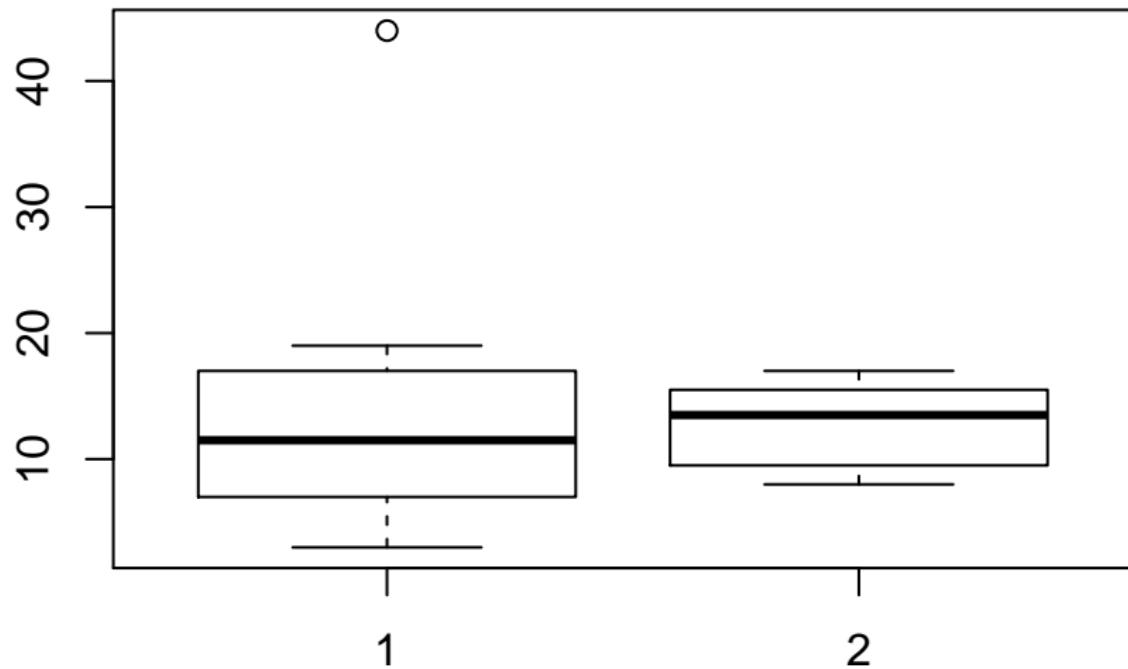
## R boxplot for outlier data

- ▶ Same data as for SAS just now.
- ▶ Code:

```
R> od=read.table("outlier.txt",header=T)
R> boxplot(x~g,data=od)
```

- ▶ Again, outlier shows up clearly:

## The boxplot



## What to do after finding an outlier

- ▶ Go back and check the original data source.
- ▶ If suspected outlier is wrong, correct it (if possible)
- ▶ If suspected outlier is correct, have to live with it.

## Outliers in multiple dimensions

- ▶ In regression, concept called **leverage**: measures whether an observation has unusual  $x$ 's *in combination*, even if none unusual by themselves.
- ▶ Example:

u	v
0	4
1	5
2	5
3	7
5	9
6	12
7	13
7	12
2	12

- ▶ Last observation: neither  $u = 2$  nor  $v = 12$  unusual by themselves.
- ▶ But when  $u = 2$ , expect  $v$  about 5; when  $v = 12$ , expect  $u$  about 7.
- ▶ *Combination of u and v unusual.*

## How to detect a combination-outlier?

- ▶ Do multiple regression with *all* variables as explanatory, and *anything* as response, eg. a constant:

```
R> uv=read.table("leverage.txt",header=T)
R> z=rep(1,9) # there are 9 observations
R> uv.lm=lm(z~u+v,data=uv)
```

- ▶ Then get the leverage values using `hatvalues` on the fitted model object:

```
R> h=hatvalues(uv.lm)
```

# The leverages

In the last column:

R> `cbind(uv, h)`

	u	v	h
1	0	4	0.3653835
2	1	5	0.2572723
3	2	5	0.2688846
4	3	7	0.1504384
5	5	9	0.1894560
6	6	12	0.2197643
7	7	13	0.3171921
8	7	12	0.3164954
9	2	12	0.9151135

- ▶ Leverage for last observation noticeably higher than others.
- ▶ How big is big? Guideline  
 $L = 2(k + 1)/n$ ,  $k$  number of variables,  
 $n$  number of observations.
- ▶ Here  $k = 2$ ,  $n = 9$ , so  
 $L = (2)(3)/9 = 0.67$ .
- ▶ Some people use 3 instead of 2 in definition of  $L$ . I want to find *possible* outliers, to track down possible errors.
- ▶ Leverage for last observation clearly that high, no others close.

## Same thing in SAS

- ▶ Define fake “response” variable in data step. This defines z to be 1 for every observation:

```
SAS> data uv;  
SAS>   infile 'leverage.txt' firstobs=2;  
SAS>   input u v;  
SAS>   z=1;
```

- ▶ Then obtain leverage values like this:

```
SAS> proc reg;  
SAS>   model z=u v / influence;
```

## Output (part)

The REG Procedure

Model: MODEL1

Dependent Variable: z

Obs	Residual	Output Statistics		
		RStudent	Hat Diag	Cov Ratio
1	0	.	0.3654	.
2	0	.	0.2573	.
3	0	.	0.2689	.
4	0	.	0.1504	.
5	0	.	0.1895	.
6	0	.	0.2198	.
7	0	.	0.3172	.
8	0	.	0.3165	.
9	0	.	0.9151	.

Output Statistics

-----DFBETAS-----

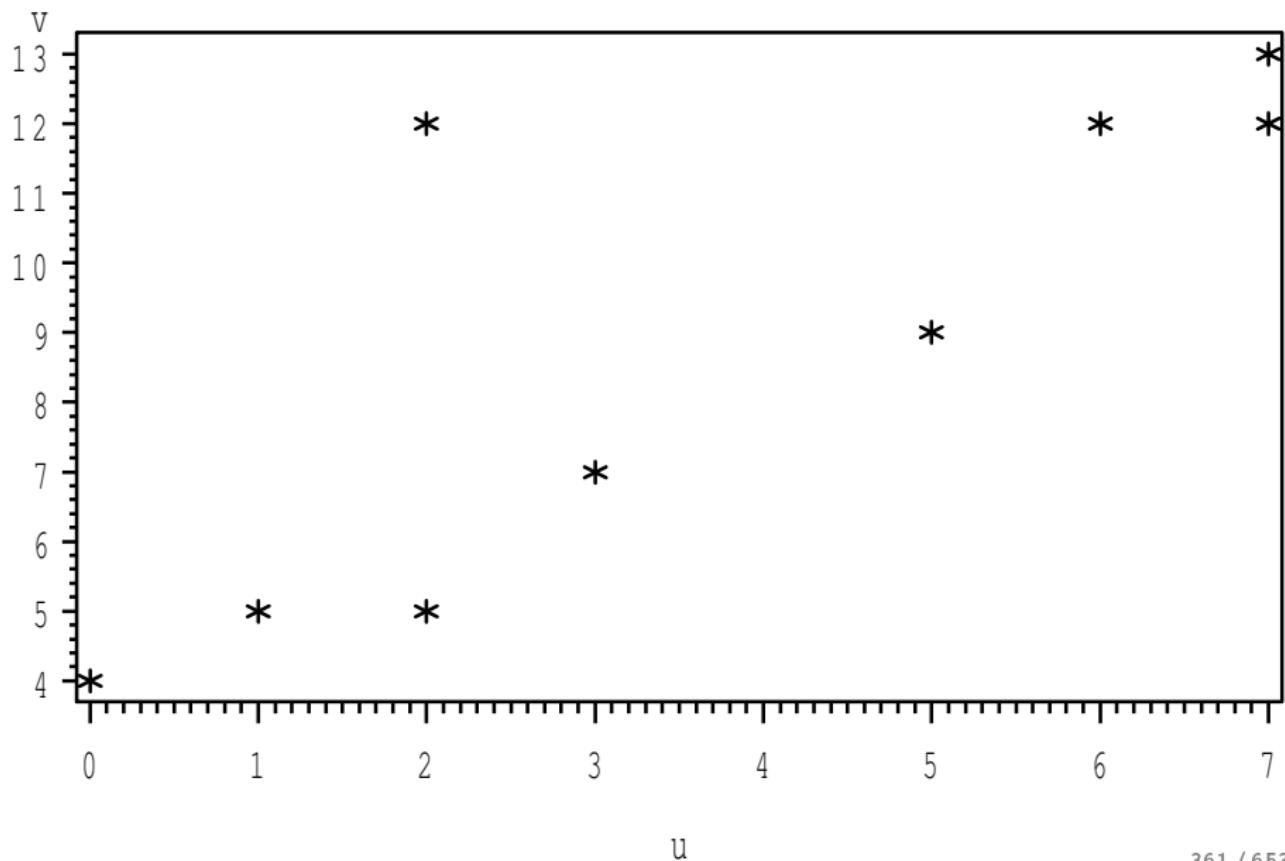
## Comments

- ▶ Leverages as before, with last one being 0.9151, larger than other (bigger than  $L = 0.67$ ).
- ▶ In this case, had 2 variables, so could also plot them (either way around):

```
SAS> proc gplot;  
SAS>   plot v*u;
```

- ▶ With more than 2 variables, this would not help.

# Plot of $u$ and $v$



## Comments

- ▶ Observation 9 rather stands out from the others by being off the pattern of the other points, top left.
- ▶ Can also apply leverage idea to *one* variable, and with groups.
- ▶ Recall data set mydata with 2 groups, outlier.
- ▶ Just pick out group 1 first by creating new data set:

```
SAS> data mydata2;  
SAS>   set mydata;  
SAS>   z=1;  
SAS>   if gp eq 1;  
SAS>  
SAS> proc reg;  
SAS>   model z=x / influence;
```

## Leverages

The REG Procedure

Model: MODEL1

Dependent Variable: z

Obs	Residual	Output Statistics		
		RStudent	Hat Diag	Cov Ratio
1	0	.	0.2068	.
2	0	.	0.1726	.
3	0	.	0.1056	.
4	0	.	0.1021	.
5	0	.	0.1095	.
6	0	.	0.8203	.
7	0	.	0.1159	.
8	0	.	0.1450	.
9	0	.	0.1047	.
10	0	.	0.1174	.

Output Statistics

## Comments

- ▶ The 6th observation has a much bigger leverage than everything else (0.8203).
- ▶ The cutoff is  $L = (2)(1 + 1)/10 = 0.4$ , so this value definitely an outlier.
- ▶ Go back to original data: the 6th value in group 1 is 44.
- ▶ Do the same kind of thing to verify that there are no outliers in group 2.

And in R:

- ▶ Pick out the values in group 1 first:

Warning message:

In rm(x) : object 'x' not found

R> attach(od)

R> is1=(g==1)

R> x1=x[is1]

R> x1

[1] 3 5 17 16 11 44 10 7 12 19

- ▶ Then make the fake response variable z, as long as x1 is, and run a regression predicting it from your explanatory variable x1:

R> z=rep(1,length(x1))

R> z.lm=lm(z~x1)

## The leverages

- ▶ And now have a look at the leverages, with data beside for comparison:

```
R> cbind(x1, hatvalues(z.lm))
```

x1

1	3	0.2068399
2	5	0.1726406
3	17	0.1055574
4	16	0.1021046
5	11	0.1095035
6	44	0.8202894
7	10	0.1159158
8	7	0.1450181
9	12	0.1047353
10	19	0.1173956

- ▶ Again, 6th observation (44) is outlier.
- ▶ Similarly, check group 2.

## Another way of handling groups

- ▶ Include the grouping variable as explanatory.
- ▶ Grouping variable should be an R factor (though with 2 groups, doesn't matter):

```
R> z=rep(1,length(x))
```

```
R> z.lm=lm(z~x+factor(g))
```

## The leverages this time

```
R> ans[ans$g==2, ]
```

	x	g	
11	10	2	0.1308357
12	13	2	0.1250482
13	15	2	0.1289066
14	14	2	0.1262057
15	17	2	0.1389382
16	9	2	0.1358515
17	8	2	0.1424107
18	16	2	0.1331507

```
R> ans=cbind(x,g,hatvalues(z.lm))
R> ans[ans$g==1, ]
```

	x	g	
1	3	1	0.2002855
2	5	1	0.1681843
3	17	1	0.1052165
4	16	1	0.1019755
5	11	1	0.1089204
6	44	1	0.7761016
7	10	1	0.1149394
8	7	1	0.1422563
9	12	1	0.1044448
10	19	1	0.1163284

- ▶ not quite so extreme now, but still bigger than cutoff  $2(2 + 1)/18 = 0.33$ .
- ▶ no outliers in group 2.

## Missing data

- ▶ Often a problem in data analysis!
- ▶ First question: “*why* are those values missing?”
- ▶ One (least bad) answer: missing **at random**.
  - ▶ whether or not value missing has nothing to do with anything else.
- ▶ With data missing at random:
  - ▶ can throw away observations with any missing values in them.  
(Wasteful?)
  - ▶ carry out analysis as if missing data never existed.
  - ▶ Software can often handle missing data automatically.
- ▶ Data missing *for a reason* is a big problem (like having non-random sample). For example, might be harder to observe value of  $y$  if  $x$  is large, then you miss important information about  $x$ - $y$  relationship.

## Missing values in SAS

- ▶ SAS uses . for “missing”. Change our outlier to missing:

```
SAS> data mydata;  
SAS>   input gp x @@;  
SAS>   cards;  
SAS>   1 3 1 5 1 17 1 16 1 11 1 . 1 10 1 7 1 12 1 19  
SAS>   2 10 2 13 2 15 2 14 2 17 2 9 2 8 2 16  
SAS> ;  
SAS>  
SAS> proc means;  
SAS>   var x;  
SAS>   class gp;
```

# Output

The MEANS Procedure

Analysis Variable : x

gp	Obs	N	Mean	Std Dev	Minimum
		N			
1	10	9	11.11111111	5.5100918	3.0000000
2	8	8	12.7500000	3.3700360	8.0000000

Analysis Variable : x

gp	Obs	N	Maximum
1	10		19.0000000
2	8		17.0000000

- ▶ There were 10 values in group 1, but SAS only used 9 of them to find the mean.
- ▶ Missing value was thrown away.

## Missing values in R

- ▶ In R, NA is code for missing:

```
R> od$x[6]=NA
```

```
R> aggregate(x~g, od, mean)
```

g	x
1	11.11111
2	12.75000

- ▶ Same results as SAS.
- ▶ So aggregate handles missing values with no extra effort.

## But sometimes you need to be more careful

- ▶ Pick out all the group 1 values including the missing one:

```
R> x1=od$x[is1]
```

```
R> x1
```

```
[1] 3 5 17 16 11 NA 10 7 12 19
```

- ▶ Find the mean:

```
R> mean(x1)
```

```
[1] NA
```

- ▶ That *includes* the missing value. To exclude it, do this:

```
R> mean(x1,na.rm=T)
```

```
[1] 11.11111
```

- ▶ aggregate removes missing values *first*, before calculating any means.
- ▶ R documentation tells you how missing values treated: read it!

## Informative missing values

- ▶ Take a look at this:

```
R> x=c(3,3,4,5,6,7,8)
R> y=c(10,10,11,12,NA,NA,12)
R> z=is.na(y)
R> aggregate(x~z,FUN=mean)
```

z	x	
1	FALSE	4.6
2	TRUE	6.5

1	FALSE	4.6
2	TRUE	6.5

- ▶ x *higher* on average when y missing than when observed.
- ▶ x and y are correlated:  

```
R> cor(x,y,use="complete.obs")
[1] 0.8439249
```
- ▶ So those missing values for y probably higher than the average y.
- ▶ The missing y values are *informative*.

# Imputation

- ▶ Sometimes missing values are informative (as above)
- ▶ or you have many variables, and throwing away a whole observation because one variable is missing is too wasteful.
- ▶ If you can guess value to replace NA with, can act as if you have complete data.
- ▶ Supplying values for missing data: **imputation**.

## Ways to do imputation

- ▶ Sometimes know from subject matter that missing values “at least *this*”, which gives you at least a crude value.
- ▶ Replace missing values by mean of their variable. Example above:

```
R> rbind(x,y)
```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]
x	3	3	4	5	6	7	8
y	10	10	11	12	NA	NA	12

```
R> mean(y,na.rm=T)
```

```
[1] 11
```

Replace missing values by 11, which seems too small, since their x's are higher than average.

- ▶ *Predict* the missing values by doing a regression. Use the complete data for estimation, then predict the missing values from the variables that are known. (SAS easier for this.)

## Regression imputation in SAS

- ▶ Enter data, then run regression obtaining output data set with predicted values in it, which we then look at:

```
SAS> data xy;  
SAS>   input x y @@;  
SAS>   cards;  
SAS>   3 10 3 10 4 11 5 12 6 . 7 . 8 12  
SAS> ;  
SAS>  
SAS> proc reg;  
SAS>   model y=x;  
SAS>   output out=xy2 p=pred;  
SAS>  
SAS> proc print;
```

## Results

Obs	x	y	pred
1	3	10	10.3488
2	3	10	10.3488
3	4	11	10.7558
4	5	12	11.1628
5	6	.	11.5698
6	7	.	11.9767
7	8	12	12.3837

- ▶ Imputed missing y values are 11.57 and 11.98.
- ▶ Make sense: higher than average y (11) since missing values went with higher than average x.

## Problems

- ▶ Problem: x-y relationship not exactly a straight line, so these values *too accurate!*
- ▶ Using these values will probably make later model fit better than it should, and make us over-optimistic about conclusions.
- ▶ Improved idea: look at typical size of regression residuals, here about 0.5. Add random quantity to each imputed value, mean 0, SD (here) 0.5. Random variability in imputed values about same as data.
- ▶ (Can do all this in R, a bit more awkward: see lecture notes.)

## Re-arranging data: keeping, dropping and creating variables

- Sometimes decide we want to make a bunch of new variables, or get rid of several, before we do the analysis. Can be a good idea to create new data set/frame with just those variables in it.
- Cars data: keep only Car, Weight, MPG, and Cylinders, get rid of others, create a new variable gpm that is reciprocal of MPG. In R:

```
R> cars=read.csv("cars.csv",header=T)
R> to.keep=c("Car","Weight","MPG","Cylinders")
R> my.cars=cars[,to.keep]
R> my.cars$gpm=1/cars$MPG
```

- Strategy:
  - make a list of variables to keep
  - select them out of the old data frame
  - create new variables with \$ to attach them to new data frame
- If you create variable just called gpm, won't be attached to new data frame.

## Keeping, dropping and creating variables in SAS

- ▶ Use a data step:

```
SAS> data mycars;  
SAS>   set 'cars';  
SAS>   gpm=1/mpg;  
SAS>   keep car weight mpg cylinders gpm;
```

- ▶ Need to explicitly keep new variable, but can either define first or keep first (doesn't matter).

- ▶ Or, drop unwanted variables:

```
SAS> data mycars;  
SAS>   set 'cars';  
SAS>   drop hp country;  
SAS>   gpm=1/mpg;
```

- ▶ Either way works. Use easier.

## Selecting variables in R

- ▶ Use subset.
- ▶ Feed in: data frame, select is list of variables to keep (no quotes needed):

```
R> mycars=subset(cars,select=c(Car,MPG,Weight,Cylinders))  
R> head(mycars)
```

	Car	MPG	Weight	Cylinders
1	Buick Skylark	28.4	2.67	4
2	Dodge Omni	30.9	2.23	4
3	Mercury Zephyr	20.8	3.07	6
4	Fiat Strada	37.3	2.13	4
5	Peugeot 694 SL	16.2	3.41	6
6	VW Rabbit	31.9	1.93	4

## Dropping variables in R

- ▶ To drop, use “negative” variable names:

```
R> mycars=subset(cars,select=-c(Horsepower,Country))  
R> head(mycars)
```

	Car	MPG	Weight	Cylinders
1	Buick Skylark	28.4	2.67	4
2	Dodge Omni	30.9	2.23	4
3	Mercury Zephyr	20.8	3.07	6
4	Fiat Strada	37.3	2.13	4
5	Peugeot 694 SL	16.2	3.41	6
6	VW Rabbit	31.9	1.93	4

- ▶ Everything *but* Horsepower and Country remains.

## Dropping observations

- ▶ Outliers should be checked, but if they are not mistakes they should *stay in the data set*.
- ▶ To remove observation containing errors, easiest: find what number row. Eg. suppose want to drop Fiat Strada (row 4), do analysis without.
- ▶ In R, make new data frame omitting it:

```
R> mycars=cars[-4,]
```

- ▶ In SAS, make new data set like this:

```
SAS> data mycars;  
SAS>   set 'cars';  
SAS>   if _n_=4 then delete;
```

## Deleting several observations

- ▶ Want to delete obs. 4 and 6 (Fiat Strada and VW Rabbit).
- ▶ R:

```
R> mycars=cars[-c(4,6),]
```

- ▶ SAS:

```
SAS> data mycars;  
SAS>   set 'cars';  
SAS>   if _n_=4 | _n_=6 then delete;
```

- ▶ In SAS, | means “or”; & means “and”.

## Dropping some variables and observations

- ▶ Just do both. SAS:

```
SAS> data mycars;  
SAS>   set 'cars';  
SAS>   keep car mpg weight;  
SAS>   if _n_=4 | _n_=6 then delete;  
SAS>  
SAS> proc print;
```

- ▶ R:

```
R> keep.vars=c("Car", "MPG", "Weight")  
R> drop.obs=c(4,6)  
R> mycars=cars[-drop.obs,keep.vars]  
R> mycars
```

## SAS output (top)

- Fiat Strada and VW Rabbit missing (check).

Obs	car	mpg	weight
1	Buick Skylark	28.4	2.670
2	Dodge Omni	30.9	2.230
3	Mercury Zephyr	20.8	3.070
4	Peugeot 694 SL	16.2	3.410
5	Plymouth Horizon	34.2	2.200
6	Mazda GLC	34.1	1.975
7	Buick Estate Wagon	16.9	4.360
8	Audi 5000	20.3	2.830
9	Chevy Malibu Wagon	19.2	3.605
10	Dodge Aspen	18.6	3.620
11	VW Dasher	30.5	2.190
12	Ford Mustang 4	26.5	2.585
13	Dodge Colt	35.1	1.915
14	Datsun 810	22.0	2.815
15	VW Scirocco	31.5	1.990

## R output (top)

- Rows numbered according to *original* data frame: 4 and 6 missing.

	Car	MPG	Weight
1	Buick Skylark	28.4	2.67
2	Dodge Omni	30.9	2.23
3	Mercury Zephyr	20.8	3.07
5	Peugeot 694 SL	16.2	3.41
7	Plymouth Horizon	34.2	2.20
8	Mazda GLC	34.1	1.98
9	Buick Estate Wagon	16.9	4.36
10	Audi 5000	20.3	2.83
11	Chevy Malibu Wagon	19.2	3.61
12	Dodge Aspen	18.6	3.62
13	VW Dasher	30.5	2.19
14	Ford Mustang 4	26.5	2.59
15	Dodge Colt	35.1	1.92
16	Datsun 810	22.0	2.82
17	VW Scirocco	31.5	1.99

## The oranges data

- ▶ Imagine you have 5 orange trees. Want to see how they grow, so measure circumference at various times. Data:

row	ages	A	B	C	D	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

## Create SAS permanent data set

```
SAS> data 'oranges';
SAS>   infile "oranges.txt" firstobs=2;
SAS>   input row age a b c d e;
SAS>
SAS> proc print;
```

Obs	row	age	a	b	c	d	e
1	1	118	30	30	30	33	32
2	2	484	51	58	49	69	62
3	3	664	75	87	81	111	112
4	4	1004	108	115	125	156	167
5	5	1231	115	120	142	172	179
6	6	1372	139	142	174	203	209
7	7	1582	140	145	177	203	214

- ▶ Trees (columns) at different #days (rows).

## Interchanging rows and columns

- ▶ Oranges data (SAS):

```
SAS> proc print data='oranges';
```

Obs	row	age	a	b	c	d	e
1	1	118	30	30	30	33	32
2	2	484	51	58	49	69	62
3	3	664	75	87	81	111	112
4	4	1004	108	115	125	156	167
5	5	1231	115	120	142	172	179
6	6	1372	139	142	174	203	209
7	7	1582	140	145	177	203	214

- ▶ Fine for eg. mean circumference for each tree, averaged over times.

```
SAS> proc means data='oranges';
```

```
SAS>   var a--e;
```

## Column means

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
-----					
a	7	94.0000000	42.9806158	30.0000000	140.0000000
b	7	99.5714286	43.2930215	30.0000000	145.0000000
c	7	111.1428571	58.8598011	30.0000000	177.0000000
d	7	135.2857143	66.3242396	33.0000000	203.0000000
e	7	139.2857143	71.8974137	32.0000000	214.0000000
-----					

- ▶ Note selection of several variables at once.
- ▶ More meaningful: average for each *time* over *trees*.
- ▶ Rows should be columns and columns rows — proc transpose first.

## Transposing a data set

```
SAS> proc transpose data='oranges' name=tree prefix=age;  
SAS>   var a--e;  
SAS>  
SAS> proc print;  
SAS>
```

Obs	tree	age1	age2	age3	age4	age5	age6	age7
1	a	30	51	75	108	115	139	140
2	b	30	58	87	115	120	142	145
3	c	30	49	81	125	142	174	177
4	d	33	69	111	156	172	203	203
5	e	32	62	112	167	179	209	214

## Finding means for each age

```
SAS> proc means;  
SAS>   var age1--age7;
```

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
-----					
age1	5	31.0000000	1.4142136	30.0000000	33.0000000
age2	5	57.8000000	8.1670068	49.0000000	69.0000000
age3	5	93.2000000	17.2394896	75.0000000	112.0000000
age4	5	134.2000000	25.9364608	108.0000000	167.0000000
age5	5	145.6000000	29.2284108	115.0000000	179.0000000
age6	5	173.4000000	32.8374786	139.0000000	209.0000000
age7	5	175.8000000	33.2821273	140.0000000	214.0000000
-----					

## The same stuff in R

- ▶ Original data:

```
R> oranges=read.table("oranges.txt",header=T)
R> oranges
```

	row	ages	A	B	C	D	E
1	1	118	30	30	30	33	32
2	2	484	51	58	49	69	62
3	3	664	75	87	81	111	112
4	4	1004	108	115	125	156	167
5	5	1231	115	120	142	172	179
6	6	1372	139	142	174	203	209
7	7	1582	140	145	177	203	214

## Column and row means

- ▶ Use `apply`.
- ▶ Second thing is 1 for rows, 2 for columns.
- ▶ Last thing is what to calculate.
- ▶ Want to skip first two columns of oranges.

```
R> apply(oranges[,3:7], 2, mean)
```

A	B	C	D	E
94.00000	99.57143	111.14286	135.28571	139.28571

- ▶ Row means:

```
R> apply(oranges[,3:7], 1, mean)
```

```
[1] 31.0 57.8 93.2 134.2 145.6 173.4 175.8
```

- ▶ Last one shows steady increase over time, until last one.

## Row or column SDs

- ▶ Same idea, but replace mean by sd, eg. row SDs:

```
R> apply(oranges[,3:7],1,sd)
```

```
[1] 1.414214 8.167007 17.239490 25.936461 29.228411 32.83
```

- ▶ Column SDs:

```
R> apply(oranges[,3:7],2,sd)
```

A	B	C	D	E
42.98062	43.29302	58.85980	66.32424	71.89741

- ▶ Row SDs (all same age) small compared to column SDs (all same tree, which grow).

## Transposing in R

- ▶ Can exchange rows and columns in R (eg. if you want to do something else with data apart from finding mean, SD):

```
R> oo=t(oranges[,3:7])
```

```
R> oo
```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]
A	30	51	75	108	115	139	140
B	30	58	87	115	120	142	145
C	30	49	81	125	142	174	177
D	33	69	111	156	172	203	203
E	32	62	112	167	179	209	214

- ▶ This is R matrix, not same as data frame (can't add column names and use column names as variables via attach).
- ▶ See how to fix that on next page.

## Adding names and making data frame

- ▶ Turn from R matrix into `data.frame`, add names to name columns (new variables):

```
R> colnames(oo)=oranges$ages
```

```
R> ood=data.frame(oo)
```

```
R> ood
```

	X118	X484	X664	X1004	X1231	X1372	X1582
A	30	51	75	108	115	139	140
B	30	58	87	115	120	142	145
C	30	49	81	125	142	174	177
D	33	69	111	156	172	203	203
E	32	62	112	167	179	209	214

- ▶ Column names have gained X's, so are legal variable names to attach.

## Wide format and long format

- ▶ This is **wide format**:

R> oranges

	row	ages	A	B	C	D	E
1	1	118	30	30	30	33	32
2	2	484	51	58	49	69	62
3	3	664	75	87	81	111	112
4	4	1004	108	115	125	156	167
5	5	1231	115	120	142	172	179
6	6	1372	139	142	174	203	209
7	7	1582	140	145	177	203	214

- ▶ Several observations per line.
- ▶ Think of circumference as response variable, which depends on time (trees as replicates).
- ▶ R's modelling functions want responses *all in one column* (with time and tree identified).
- ▶ Command called `reshape` that turns wide format into long format.

## Long format for oranges data

	row	ages	tree	circum	id		row	ages	tree	circum	id	
1.1	1	118	1	30	1		5.3	5	1231	3	142	5
2.1	2	484	1	51	2		6.3	6	1372	3	174	6
3.1	3	664	1	75	3		7.3	7	1582	3	177	7
4.1	4	1004	1	108	4		1.4	1	118	4	33	1
5.1	5	1231	1	115	5		2.4	2	484	4	69	2
6.1	6	1372	1	139	6		3.4	3	664	4	111	3
7.1	7	1582	1	140	7		4.4	4	1004	4	156	4
1.2	1	118	2	30	1		5.4	5	1231	4	172	5
2.2	2	484	2	58	2		6.4	6	1372	4	203	6
3.2	3	664	2	87	3		7.4	7	1582	4	203	7
4.2	4	1004	2	115	4		1.5	1	118	5	32	1
5.2	5	1231	2	120	5		2.5	2	484	5	62	2
6.2	6	1372	2	142	6		3.5	3	664	5	112	3
7.2	7	1582	2	145	7		4.5	4	1004	5	167	4
1.3	1	118	3	30	1		5.5	5	1231	5	179	5
2.3	2	484	3	49	2		6.5	6	1372	5	209	6
3.3	3	664	3	81	3		7.5	7	1582	5	214	7
4.3	4	1004	3	125	4							

## How to make long format: the reshape command

- ▶ I did:

```
R> reshape(oranges, varying=3:7, v.names="circum",  
R>       timevar="tree", direction="long")
```

- ▶ Specify:

- ▶ Data frame
- ▶ Which columns to combine together to make “long” response
- ▶ What to call that long response
- ▶ What those columns are repeated measurements over
- ▶ What format to make (can also make “wide” from “long”).

- ▶ You *will* get this wrong the first time!

## How to make it (slightly) easier

- ▶ Give the columns that are all responses names like this:

```
R> names(oranges)=c("row","ages","circum.A","circum.B",
R>   "circum.C","circum.D","circum.E")
R> oranges
```

	row	ages	circum.A	circum.B	circum.C	circum.D	circum.E
1	1	118	30	30	30	33	32
2	2	484	51	58	49	69	62
3	3	664	75	87	81	111	112
4	4	1004	108	115	125	156	167
5	5	1231	115	120	142	172	179
6	6	1372	139	142	174	203	209
7	7	1582	140	145	177	203	214

- ▶ Two-part names (separated by .): response name, tree name. Then, with no v.names:

```
R> oranges.long=reshape(oranges,varying=3:7,
R>   timevar="tree",direction="long")
```

## Matched pairs and reshape in reverse

- ▶ How well does particular brand of tire help drivers to stop on wet pavement vs. dry pavement?
- ▶ Data:

```
R> braking=read.table("braking.txt",header=T)
R> head(braking)
```

	car	cond	dist
1	1	dry	150
2	1	wet	201
3	2	dry	147
4	2	wet	220
5	3	dry	136
6	3	wet	192

- ▶ Long format.
- ▶ This is matched pairs (2 stopping distances per car).
- ▶ For matched pairs, R needs *wide* format.
- ▶ Want one line per car, with both wet and dry stopping distance on it.

## Making wide format

- ▶ Response variable dist (get two on each line)
- ▶ Different responses will be values of cond.
- ▶ What does each line represent? A car.
- ▶ Thus:

```
R> braking.wide=reshape(braking,v.names="dist",
R>   timevar="cond",idvar="car",direction="wide")
```

## The wide data frame

```
R> braking.wide
```

	car	dist.dry	dist.wet
1	1	150	201
3	2	147	220
5	3	136	192
7	4	134	146
9	5	130	182
11	6	134	173
13	7	134	202
15	8	128	180
17	9	136	192
19	10	158	206

- ▶ R even constructs suitable variable names for you.

## Matched pairs analysis

- ▶ Sample of “all possible cars”.
- ▶ Wet distances greater than dry distances, so test not of interest (will be significant).
- ▶ Confidence interval, wet minus dry, most interesting.
- ▶ Uses `t.test` with `paired` option.
- ▶ Calculate variable `diff` for later.

```
R> attach(braking.wide)
R> t.test(dist.wet,dist.dry,paired=T)
R> diff=dist.wet-dist.dry
R> detach(braking.wide)
```

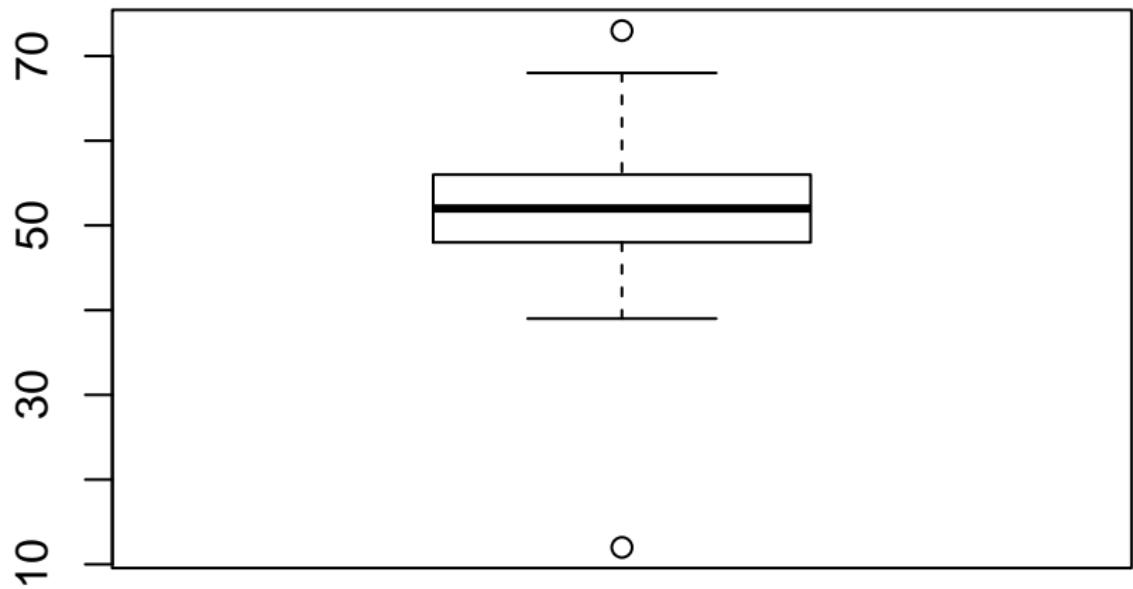
## t.test output

Paired t-test

```
data: dist.wet and dist.dry
t = 9.6233, df = 9, p-value = 4.921e-06
alternative hypothesis: true difference in means is not equal to
95 percent confidence interval:
 38.78192 62.61808
sample estimates:
mean of the differences
                  50.7
```

- ▶ P-value *is* very small.
- ▶ Stopping distances between about 40 and 60 (feet) longer in wet (95% confidence).
- ▶ Small sample. Should have checked differences for outliers, eg. by boxplot:  
*R> boxplot(diff)*

## The boxplot

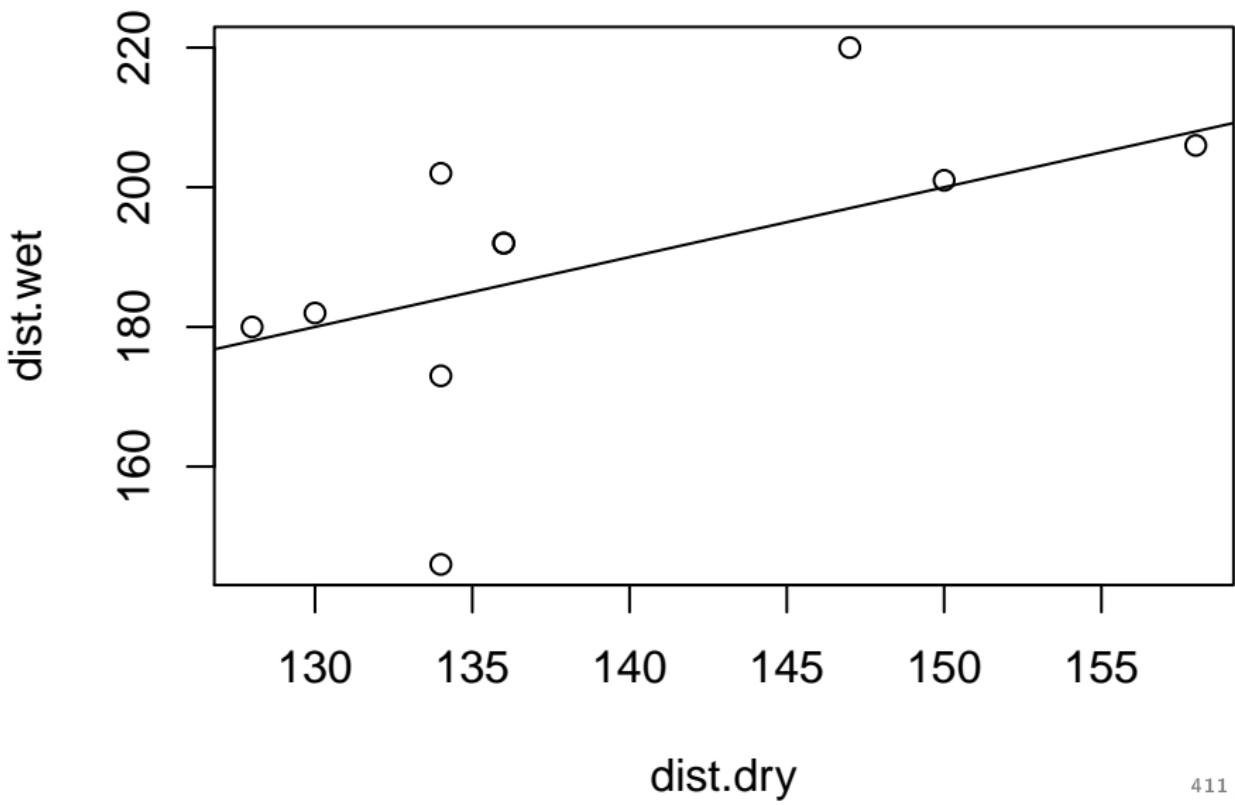


## Comments

- ▶ One of those differences, about 10, *a lot* smaller than others (car 4, whose wet stopping distance suspiciously short).
- ▶ Differences may not be sufficiently normal (with small sample).
- ▶ Should have doubts about paired *t*-test.
- ▶ Plot actual stopping distances (2 variables, scatterplot).
- ▶ Mean difference wet ( $y$ ) minus dry ( $x$ ) about 50, so  $y = x + 50$  approx. Plot that line too:

```
R> plot(braking.wide[, -1])  
R> abline(a=50, b=1)
```

## Scatterplot



## Comments

- ▶ Point bottom left of plot unusual.
- ▶ `dist.wet` about 150; expected about 180.
- ▶ Big outlier difference on boxplot.

## In SAS: reading data

- Top of data file like this:

```
car cond dist
1 dry 150
1 wet 201
2 dry 147
2 wet 220
3 dry 136
3 wet 192
```

- Each car's data on 2 lines of file, but want as 1 line of data set.
- Want: car number, word "dry", dry distance, newline, car number again, word "wet", wet distance.
- Slash in middle of input line means "go to next line of file":

```
SAS> data braking;
SAS>   infile "braking.txt" firstobs=2;
SAS>   input car cond $ distdry / car2 cond2 $ distwet;
SAS>   drop car car2 cond cond2;
SAS>
SAS> proc print;
```

## The data set as read in

Obs	distdry	distwet
1	150	201
2	147	220
3	136	192
4	134	146
5	130	182
6	134	173
7	134	202
8	128	180
9	136	192
10	158	206

## Paired *t*-test

- ▶ Note specification of which variables are paired:

```
SAS> proc ttest;
```

```
SAS>   paired distwet*distdry;
```

The TTEST Procedure

Difference: distwet - distdry

N	Mean	Std Dev	Std Err	Minimum	Maximum
10	50.7000	16.6603	5.2685	12.0000	73.0000
Mean	95% CL Mean		Std Dev	95% CL	Std Dev
50.7000	38.7819	62.6181	16.6603	11.4596	30.4153
DF	t Value	Pr >  t			
9	9.62	<.0001			

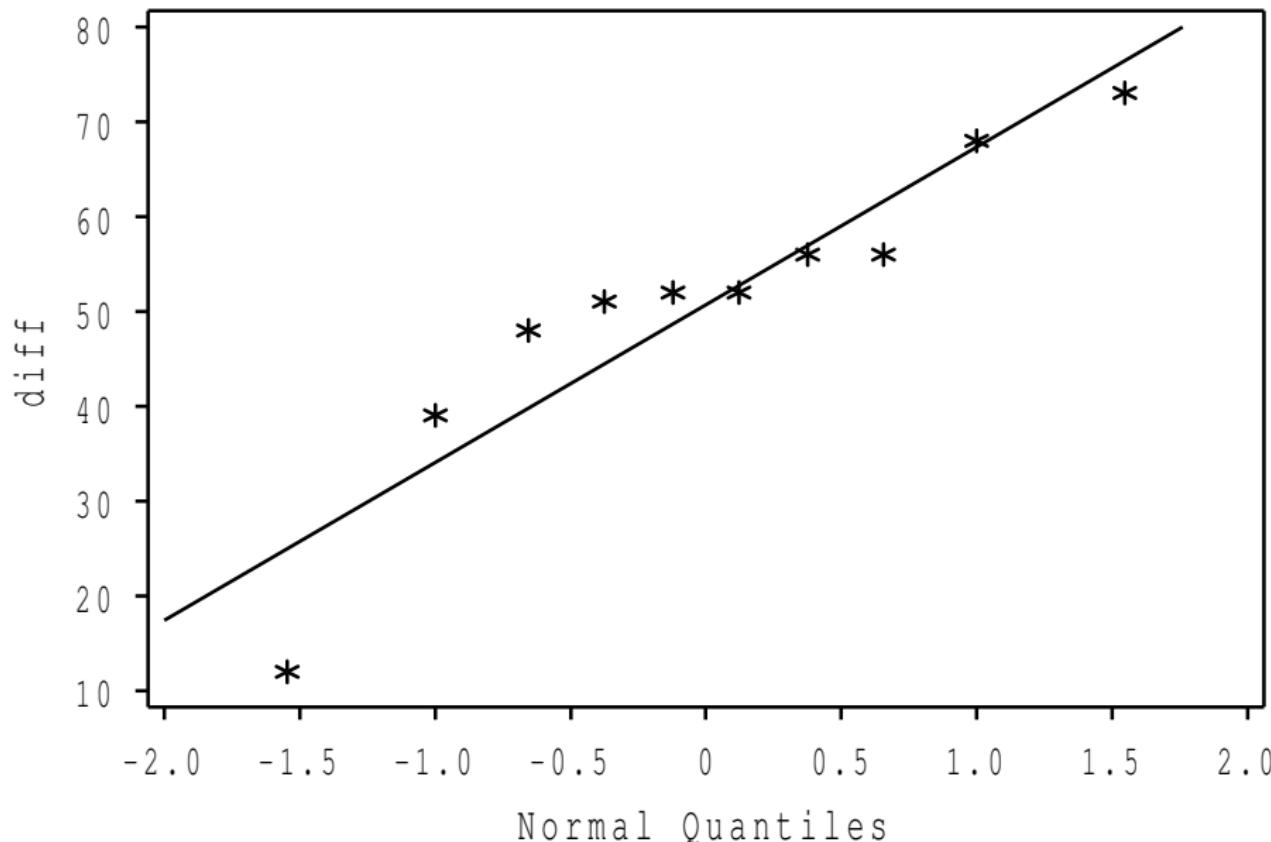
- ▶ As before, P-value very small, 95% CI from about 40 to 60.

## Getting normal quantile plot of differences

- ▶ First, find differences (by making a new data set)
- ▶ Run proc univariate, getting no printed output
- ▶ Ask for a normal quantile plot, with estimated mean and SD for the normal:

```
SAS> data fred;  
SAS>   set braking;  
SAS>   diff=distwet-distdry;  
SAS>  
SAS> proc univariate noprint;  
SAS>   qqplot diff /  
SAS>     normal(mu=est sigma=est color=black);
```

## The normal quantile plot



## Comments

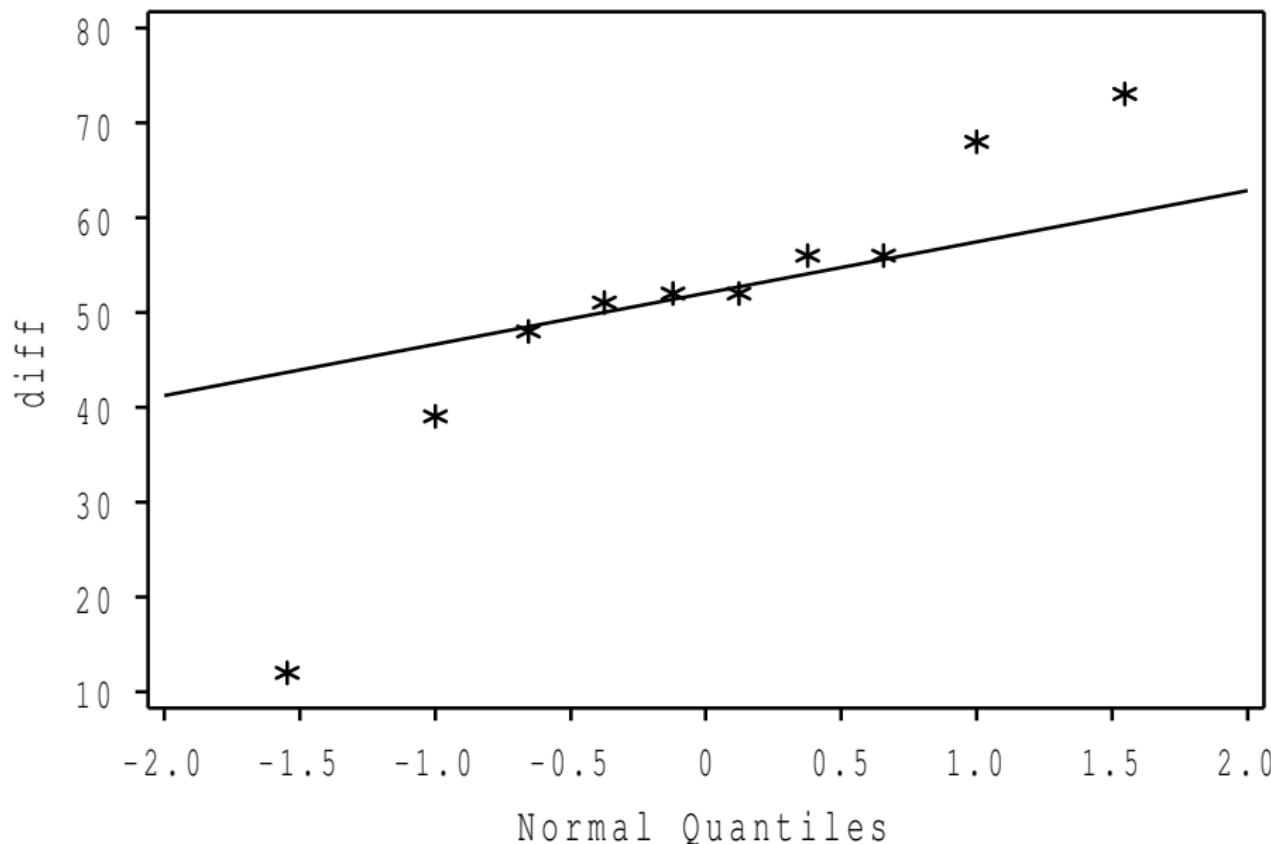
- ▶ Smallest difference quite a bit smaller than the rest, but not that far from the line.
- ▶ SAS uses sample mean and SD if you ask it to estimate.
- ▶ If there are outliers, SD inflated and mean pulled towards outlier.
- ▶ Bad! Would prefer to use median and IQR.

## Estimating $\mu$ and $\sigma$ using median and IQR

- ▶ Quartiles for standard normal are  $z = \pm 0.67$  (check table).
- ▶ IQR for any normal therefore  $2(0.67)\sigma = 1.34\sigma$ .
- ▶ Estimate  $\sigma$  by setting equal to IQR, here 7.25.
- ▶  $1.34\sigma = 7.25 \Rightarrow \sigma = 5.41$ .
- ▶ Estimate  $\mu$  by median, 52:

```
SAS> proc univariate noprint;  
SAS>   qqplot diff /  
SAS>     normal(mu=52 sigma=5.41 color=black);
```

## Revised normal quantile plot



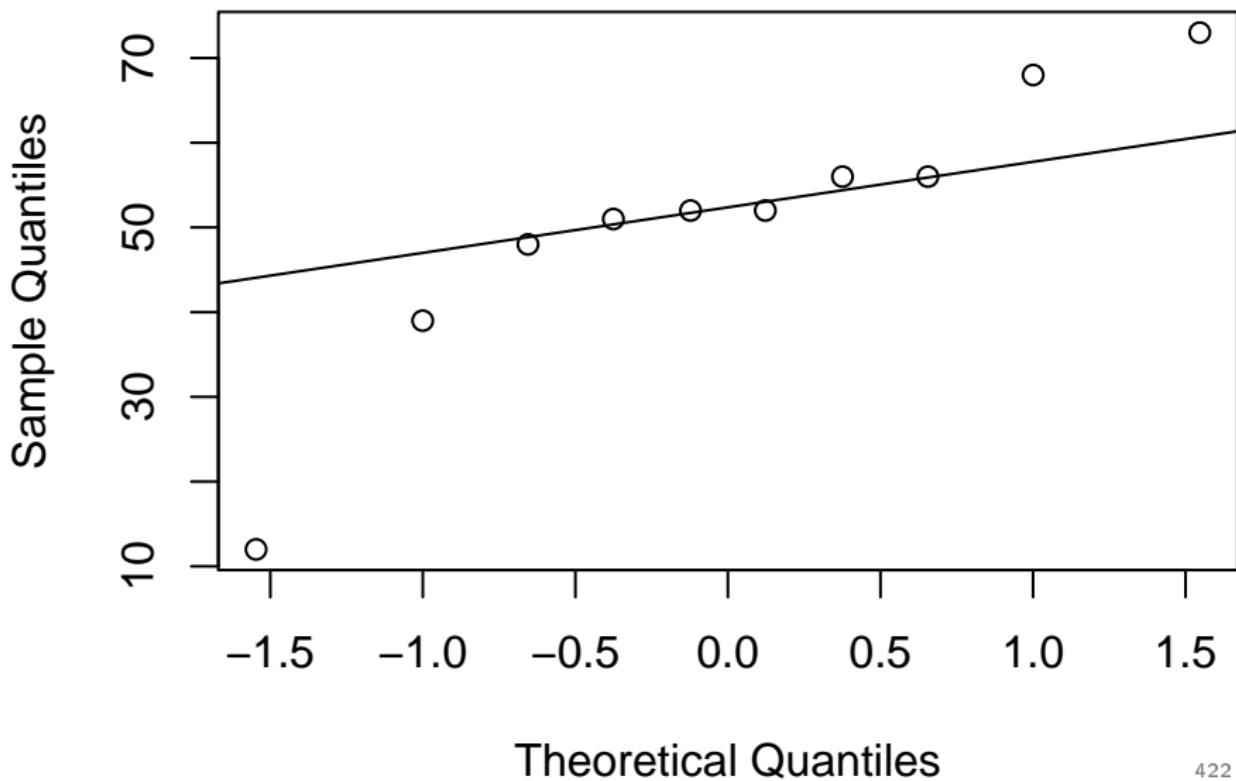
## Comments

- ▶ Now see that lowest value much too small for normal.
- ▶ Largest two values also a bit too big.
- ▶ R's normal quantile plot also uses quartiles to estimate where the line should go:

```
R> qqnorm(diff)  
R> qqline(diff)
```

R's normal quantile plot

## Normal Q-Q Plot



## Comments

- ▶ The *default* R normal quantile plot is the same as the SAS one based on the quartiles.
- ▶ Using quartiles to estimate  $\mu$  and  $\sigma$  will not be affected by outliers, so I think this is a better idea than using sample mean and SD.
- ▶ A distribution will typically fail to be normal *because* of outliers.
- ▶ Can test any (continuous) distribution using the same qqplot idea.

# Part IX

R stuff

## More R stuff

- ▶ R has a thousand tiny parts, all working together, but to use them, need to know their *names*.
- ▶ I discovered them by searching, like this:

*how to label vertical axis R*

to find out that I needed (here) `ylab`.

- ▶ Looked at help via `?ylab` to find out how it worked.
- ▶ Read in the `cars` data to use for examples:

```
R> cars=read.csv("cars.csv")  
R> attach(cars)
```

- ▶ Take a look at help for `median`, via `?median` (in Console). 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)`.

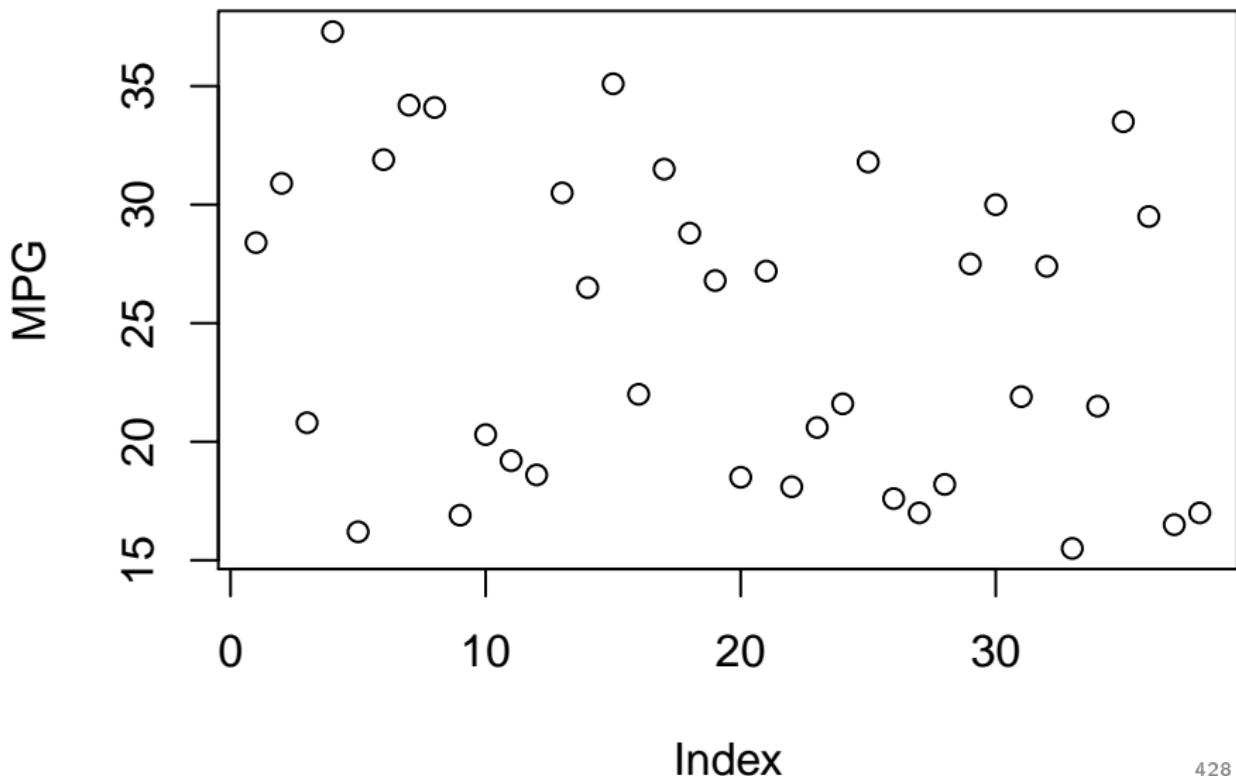
## Plotting stuff

- ▶ plot is “generic function”: what it does depends on what you feed it.
- ▶ We know that plotting two variables makes a scatterplot.
- ▶ Plotting only one plots against time (“index plot”). Car MPG values:  
*R> plot(MPG)*
- ▶ Plotting categorical variable vs. numeric makes side by side boxplots:  
*R> plot(Country, MPG)*
- ▶ Plotting two categorical variables produces a “spineplot”. Make Cylinders into a factor first.

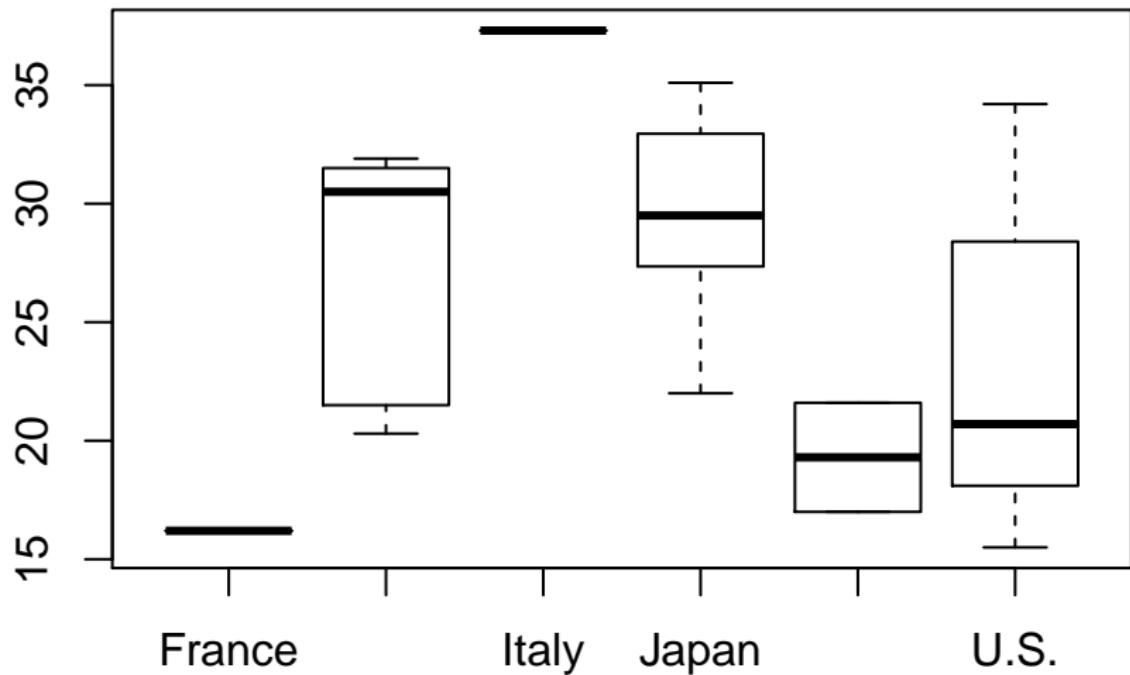
```
R> cyl.fac=factor(Cylinders)
R> plot(cyl.fac, Country)
R> table(Country, cyl.fac)
```

	cyl.fac	4	5	6	8
Country	France	0	0	1	0
	Germany	4	1	0	0
	Italy	1	0	0	0
	Japan	6	0	1	0
	Sweden	1	0	1	0
	U.S.	7	0	7	8

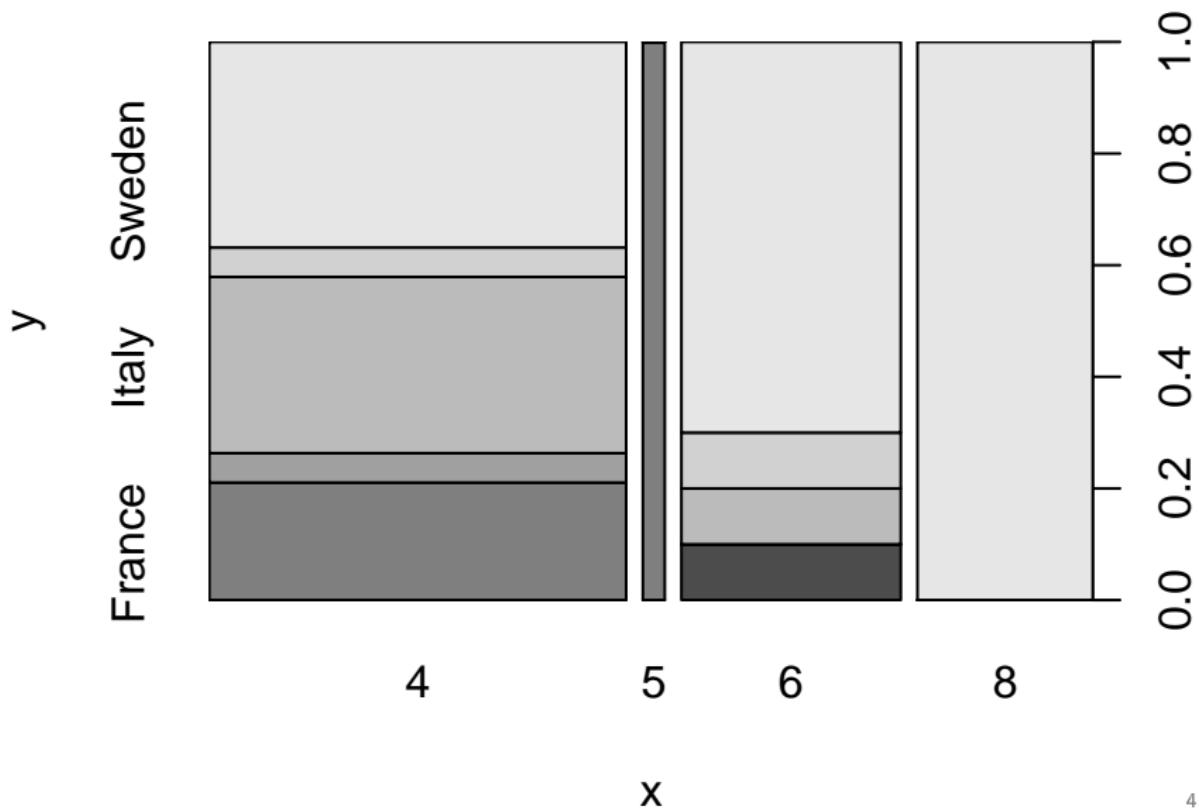
## Index plot



## Boxplots

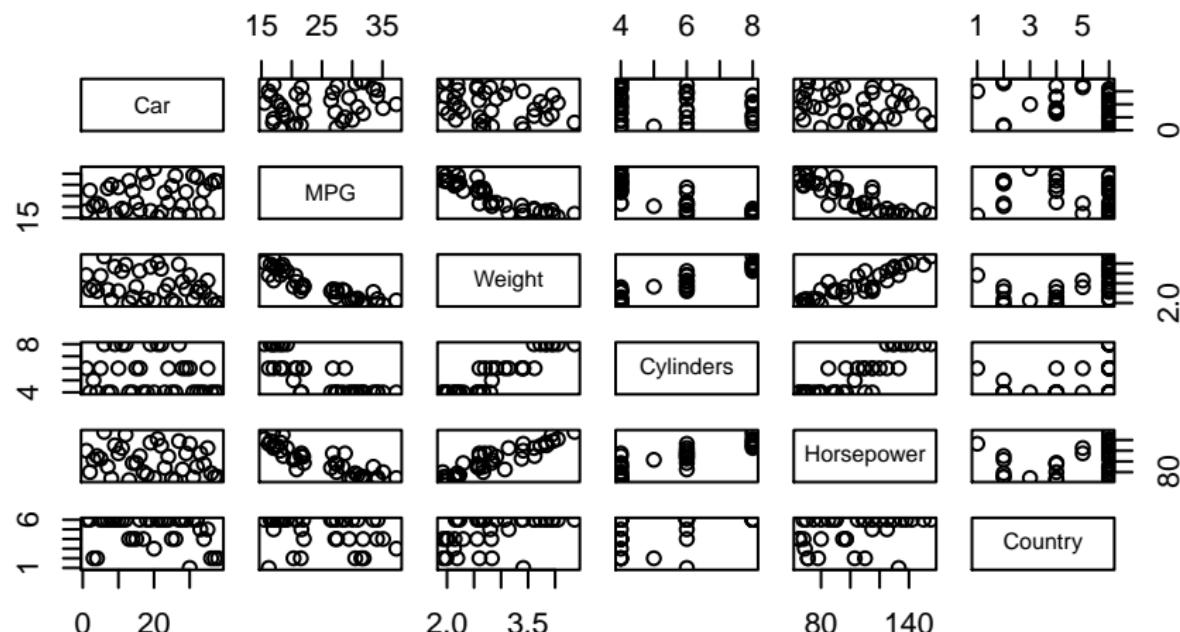


# Spineplot



# Plotting a data frame

```
R> plot(cars)
```

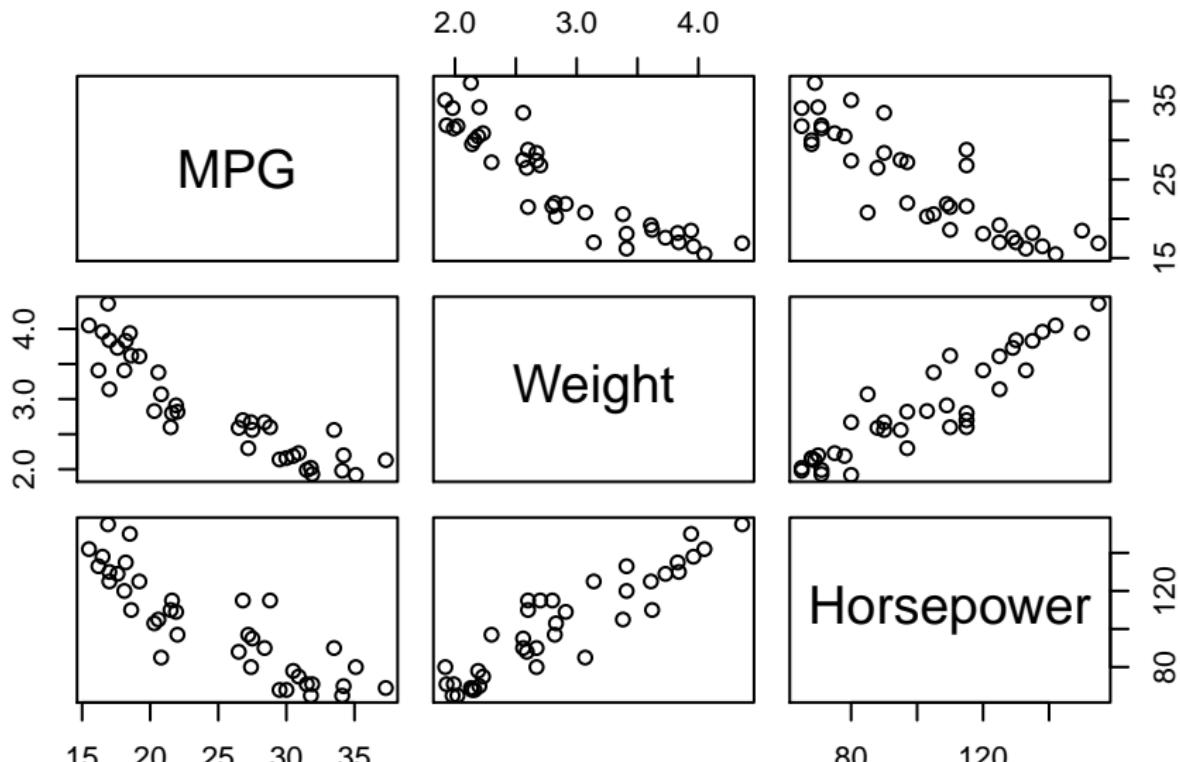


## Comments

- ▶ Plots mostly self-explanatory.
- ▶ Spineplot: area of region represents number of cars from country (France dark, US light) with each number of cylinders. All 8-cylinder cars from US.
- ▶ Plotting a data frame gives **pairs plot**: all possible scatterplots of pairs of variables. Includes categorical variables (coded to numbers).
- ▶ Can (just) see that weight goes up with horsepower and both go down with MPG.
- ▶ Below: more useful (and visible!) pairs plot. Columns 2, 3 and 5 have the actual quantitative variables.  

```
R> plot(cars[,c(2,3,5)])
```
- ▶ See also `methods(plot)` and `?plot`.

## More useful pairs plot



## Finding function names

- ▶ The function `apropos` helps you find out what a function is called when you have part of the name:

```
R> apropos("histogram")
character(0)
```

- ▶ Nothing. Try something shorter:

```
R> apropos("hist")
[1] "hist"          "hist.default" "history"       "loadhisto
```

- ▶ `hist` looks most plausible. Try `?hist` and see whether that's the one.  
(It is.)

## Annotating plots (help in `?par`)

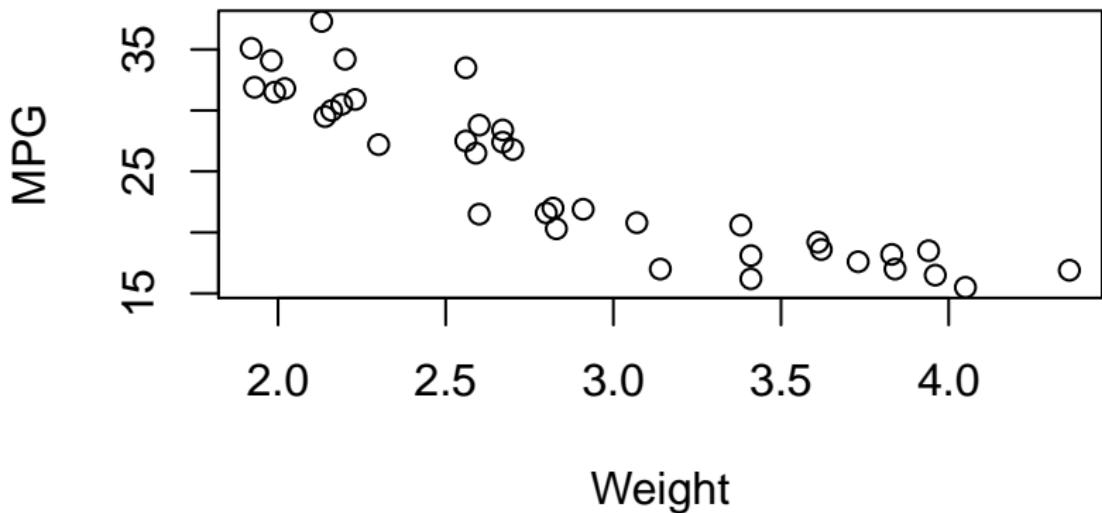
- ▶ adding a title to a plot (`main`)
- ▶ changing the axis labels (`xlab`, `ylab`)
- ▶ adding points to a plot (`points`)
- ▶ adding lines or curves to a plot (`lines`; `abline`)
- ▶ changing line types (`lty`)
- ▶ changing the size of text, numbers on axes (`cex.axis`, `cex.lab`)
- ▶ adding text to a plot, eg. to label points (`text`)
- ▶ colouring groups of points (`col`)
- ▶ using different plotting symbols (`pch`)
- ▶ changing the size of text (`cex`)
- ▶ types of plot
- ▶ plotting multiple sets of lines/points
- ▶ arrays of plots (`mfrow`, `mfcol`)

## Adding a title to a plot

- ▶ Illustrating with cars data:

```
R> plot(Weight,MPG,main="Plot of gas mileage against weight",  
R>       sub="A thoroughly unnecessary subtitle under the x axis")
```

**Plot of gas mileage against weight**

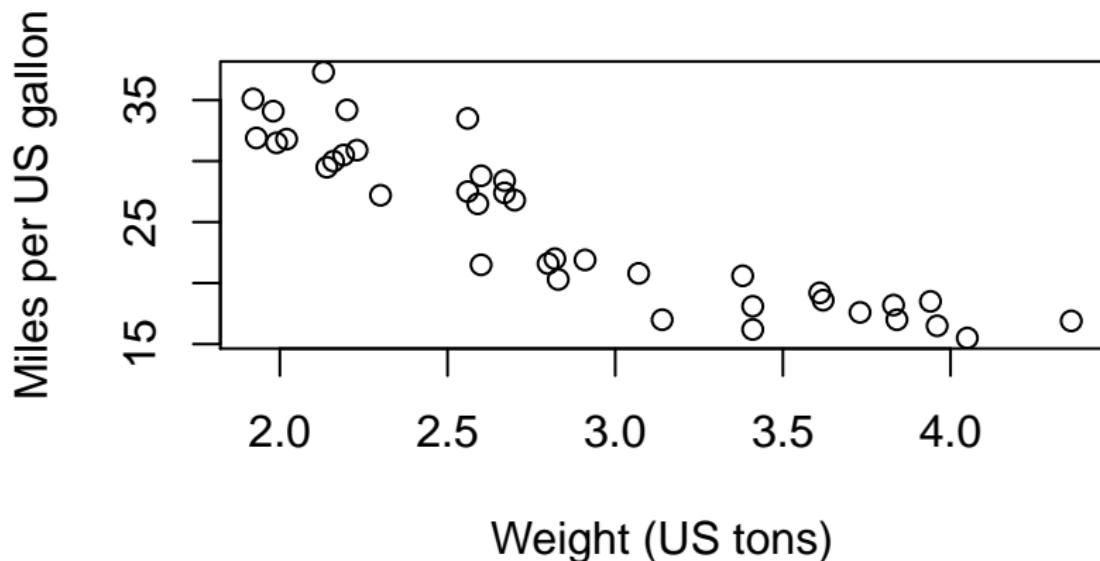


Weight

A thoroughly unnecessary subtitle under the x axis

## Axis labels

```
R> plot(Weight,MPG,xlab="Weight (US tons)",  
R>       ylab="Miles per US gallon")
```



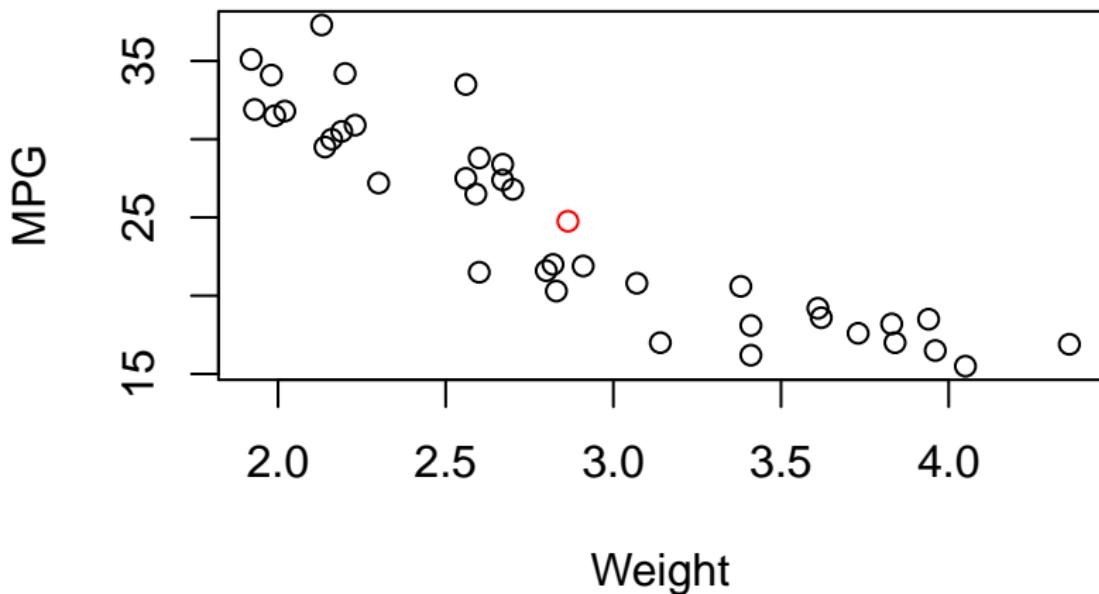
## Adding points to a plot

- Find mean Weight and MPG and add them to the plot in red:

```
R> mean.weight=mean(Weight) ; mean.MPG=mean(MPG)
```

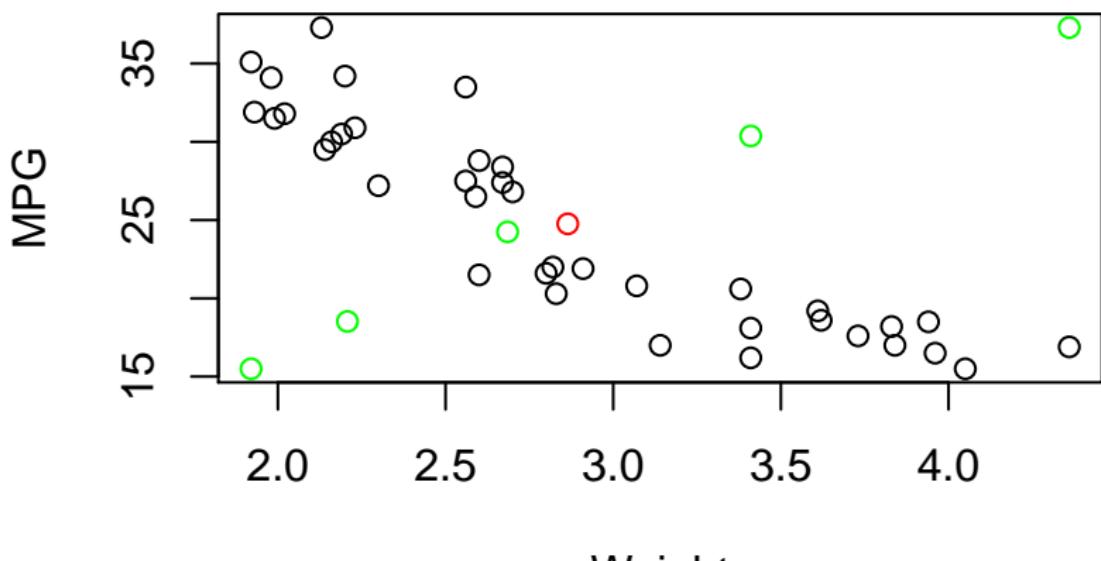
```
R> plot(Weight,MPG)
```

```
R> points(mean.weight,mean.MPG,col="red")
```



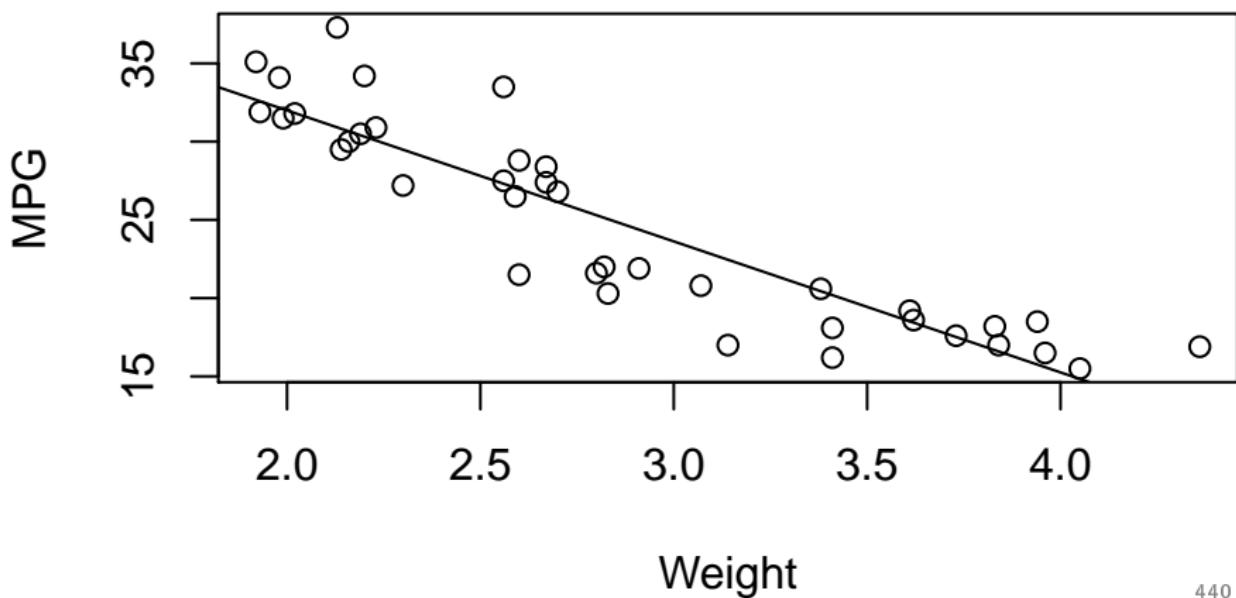
## Adding vectors of points to a plot

```
R> q.weight=quantile(Weight) ; q.MPG=quantile(MPG)
R> plot(Weight,MPG)
R> points(mean.weight,mean.MPG,col="red")
R> points(q.weight,q.MPG,col="green")
```



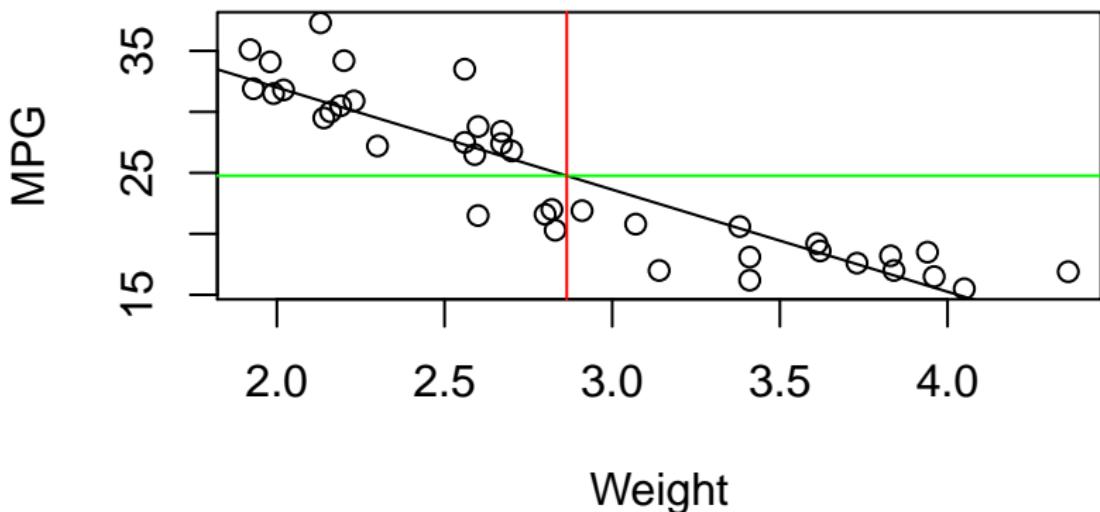
## Adding a regression line to a plot

```
R> plot(Weight,MPG)
R> MPG.lm=lm(MPG~Weight)
R> abline(MPG.lm)
```



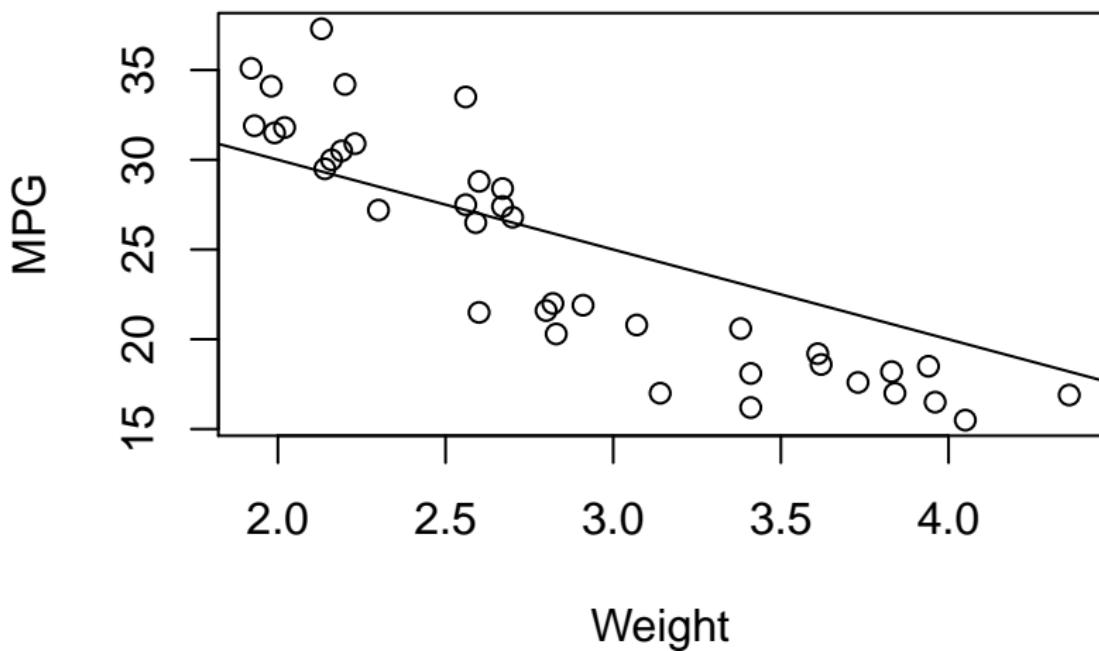
## Horizontal and vertical lines

```
R> plot(Weight,MPG)
R> MPG.lm=lm(MPG~Weight)
R> abline(MPG.lm)
R> abline(h=mean.MPG,col="green")
R> abline(v=mean.weight,col="red")
```



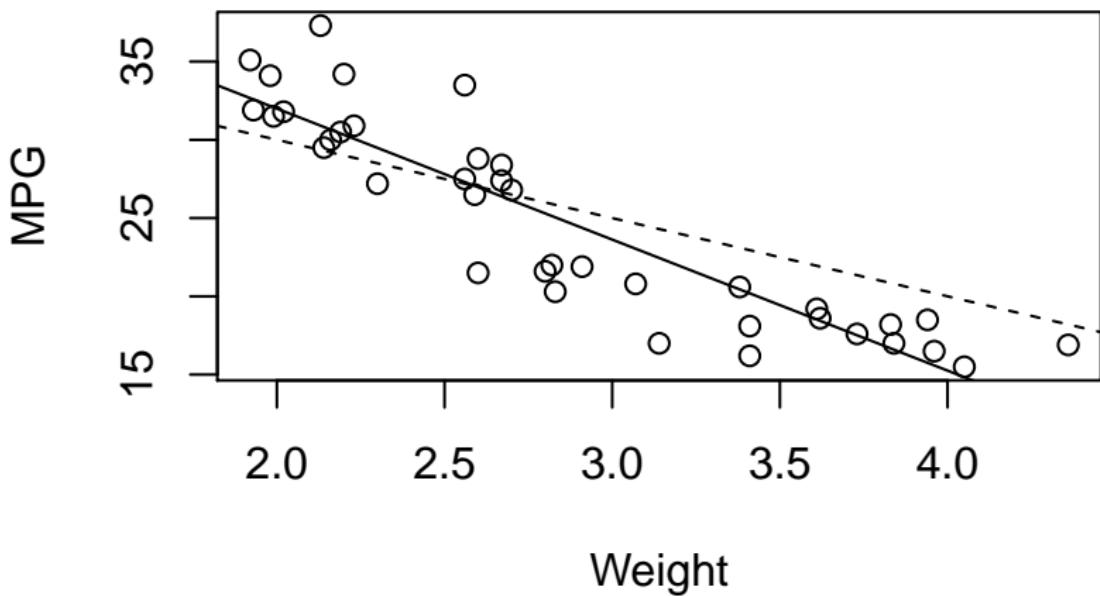
## Line with given intercept and slope

```
R> plot(Weight, MPG)  
R> abline(a=40, b=-5)
```



## Line types

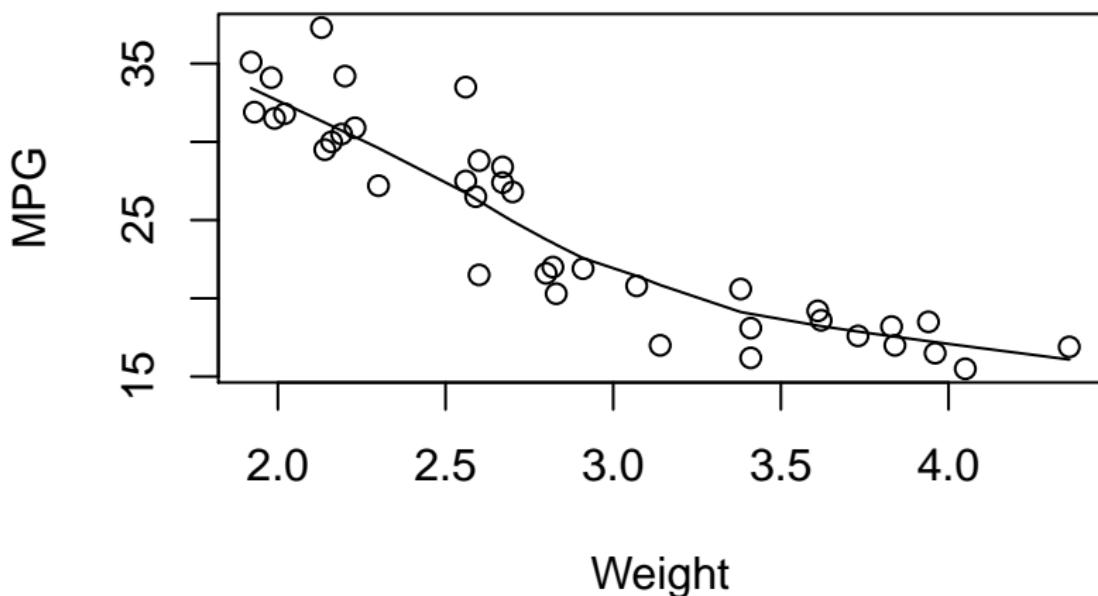
```
R> plot(Weight, MPG)
R> abline(MPG.lm)
R> abline(a=40, b=-5, lty="dashed")
```



## Lowess curve (scatterplot smoother)

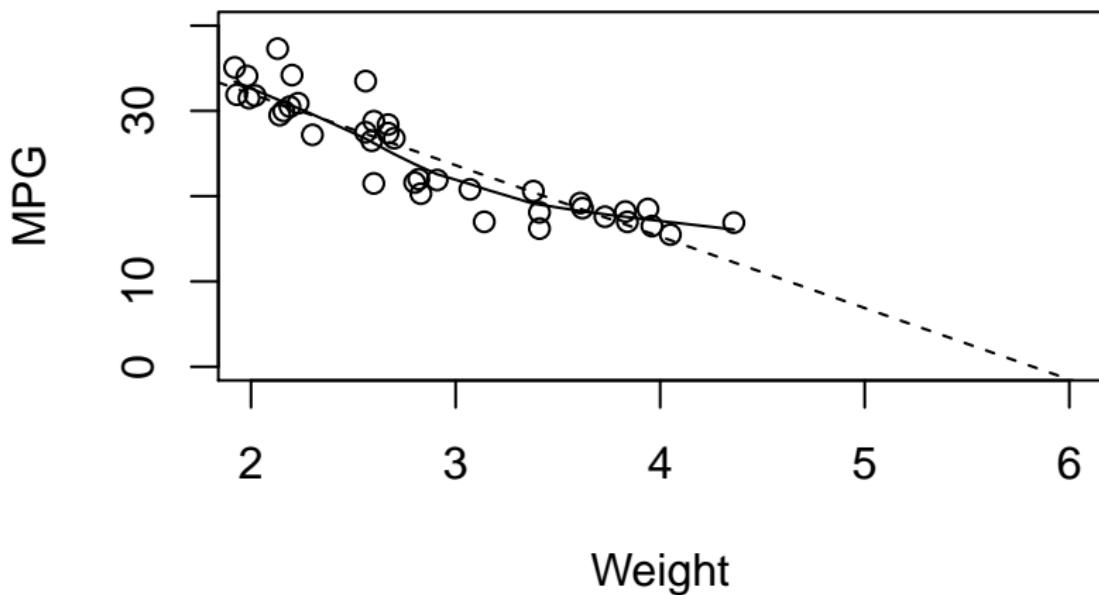
```
R> plot(Weight, MPG)
```

```
R> lines(lowess(Weight, MPG))
```



## Extending/shrinking axes; prediction for weight=6?

```
R> plot(Weight,MPG,xlim=c(2,6),ylim=c(0,40))  
R> abline(MPG.lm,lty="dashed")  
R> lines(lowess(Weight,MPG))
```



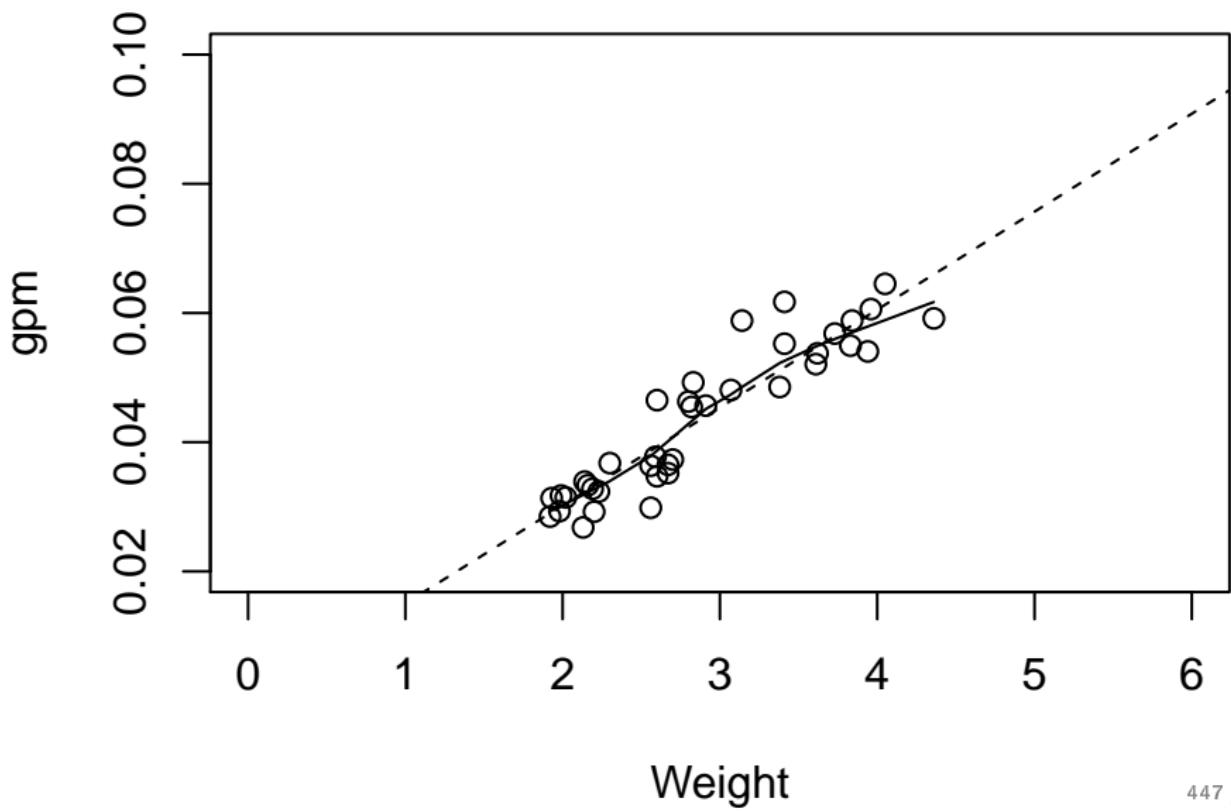
## Reciprocal transformation

- ▶ Does predicting  $1/\text{MPG}$  from Weight do any better?

```
R> gpm=1/MPG  
R> plot(Weight,gpm,xlim=c(0,6),ylim=c(0.02,0.1))  
R> gpm.lm=lm(gpm~Weight)  
R> abline(gpm.lm,lty="dashed")  
R> lines(lowess(Weight,gpm))
```

- ▶ Plot next page.
- ▶ Is trend/lowess straighter?

## The plot



## Comments

- ▶ Trend with gallons per mile is straighter.
- ▶ Prediction for Weight=6 is still extrapolation, but more confident in straight line now.
- ▶ For Weight=6, predicted gpm about 0.09, so predicted MPG about  $1/0.09 = 11$ .
- ▶ Not obviously meaningless.

## Labelling points

- ▶ Which car heaviest? Which has best gas mileage? Label on scatterplot.
- ▶ Find heaviest and most fuel-efficient cars (black magic):

```
R> cars[Weight==max(Weight),]
```

	Car	MPG	Weight	Cylinders	Horsepower	Country
9	Buick Estate Wagon	16.9	4.36		8	155

```
R> cars[MPG==max(MPG),]
```

	Car	MPG	Weight	Cylinders	Horsepower	Country
4	Fiat Strada	37.3	2.13		4	69

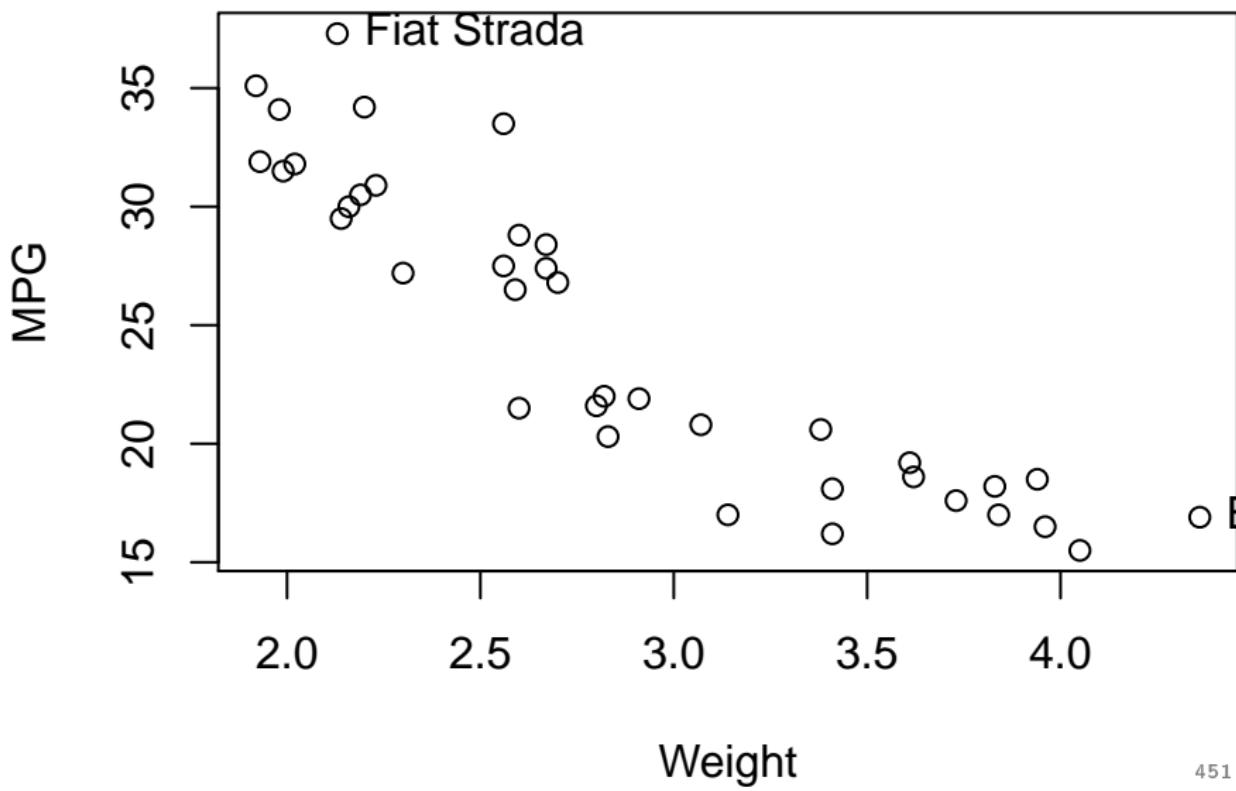
- ▶ Car #9 heaviest; car #4 best MPG.
- ▶ Label these cars on scatterplot with name of car (to right of point).

## The plot with these points labelled

- ▶ **text:**
  - ▶  $x$  and  $y$  coordinates of place to put text
  - ▶ Text to put there (Car is name of car)
  - ▶ (optional) pos=4 puts text to right of circle rather than on top of it (default):

```
R> plot(Weight,MPG)
R> text(Weight[9],MPG[9],Car[9],pos=4)
R> text(Weight[4],MPG[4],Car[4],pos=4)
```

## The resulting plot



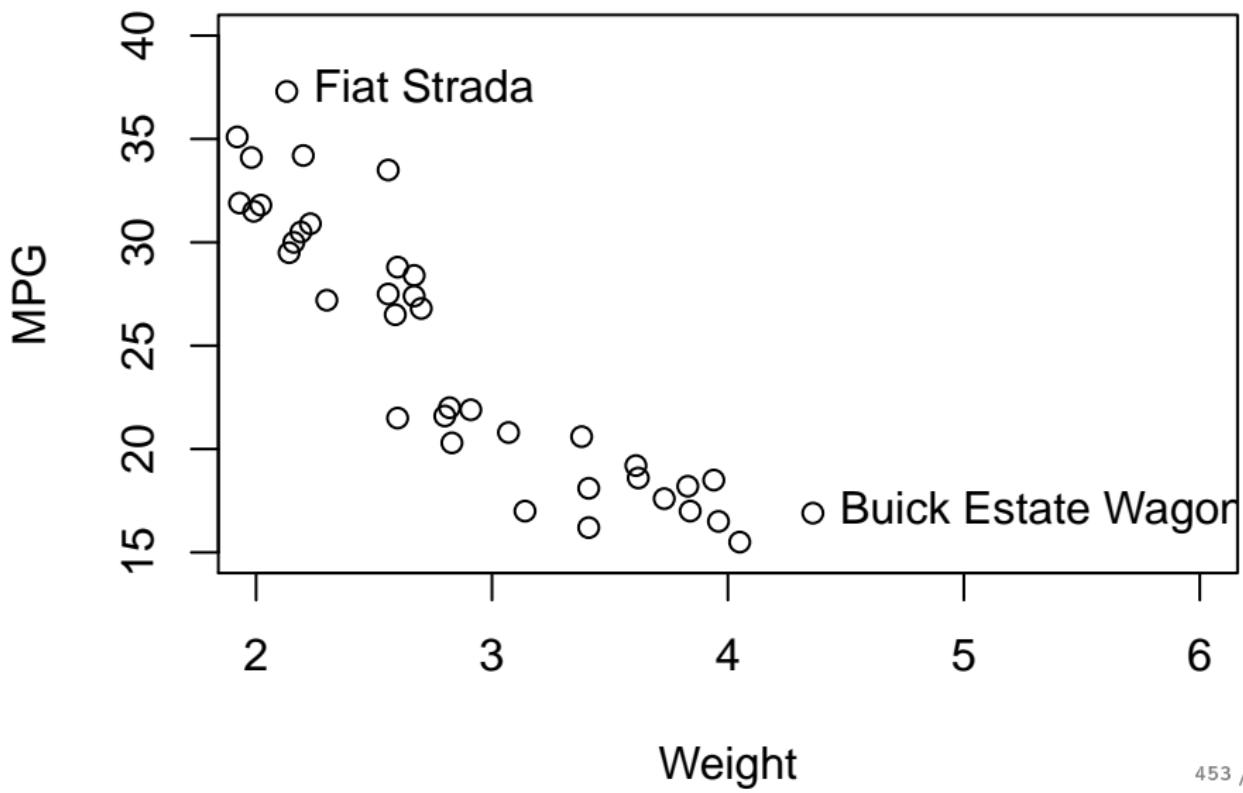
# Ooops

- ▶ Labels went off the graph.
- ▶ Fix using `xlim` and `ylim`:

```
R> plot(Weight,MPG,xlim=c(2,6),ylim=c(15,40))  
R> text(Weight[9],MPG[9],Car[9],pos=4)  
R> text(Weight[4],MPG[4],Car[4],pos=4)
```

- ▶ Get limits by experiment.

## Revised plot



## Colours

- ▶ col works to colour anything:

```
R> plot(Weight,MPG,col="red",xlim=c(2,6),ylim=c(15,40))
R> text(Weight[9],MPG[9],Car[9],pos=4,col="green")
R> text(Weight[4],MPG[4],Car[4],pos=4,col="blue")
R> abline(MPG.lm,col="brown")
R> lines(lowess(Weight,MPG),col="purple")
```

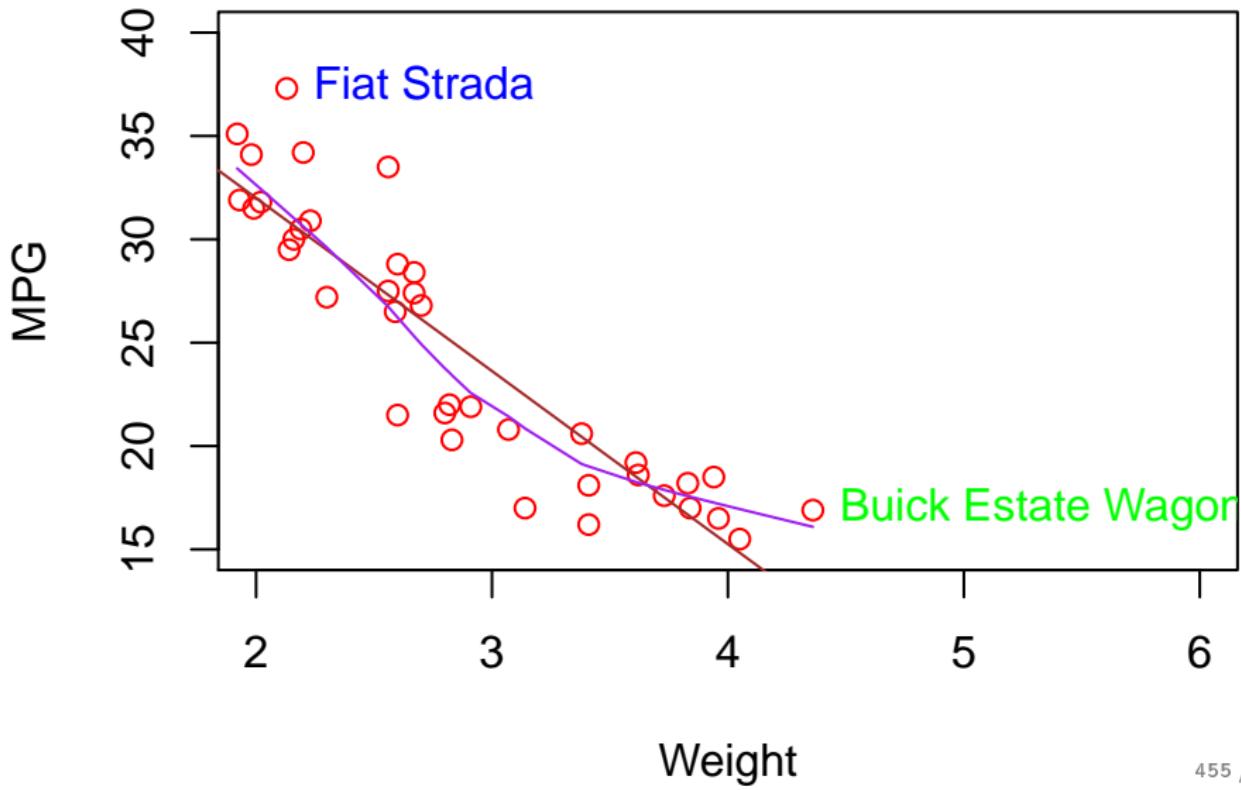
- ▶ Gives previous plot with all kinds of lines.
- ▶ colors() gives R's 657 colour names.
- ▶ Colours can also be referred to by number:

```
R> palette()
```

```
[1] "black"     "red"       "green3"    "blue"      "cyan"      "magenta"
[8] "gray"
```

- ▶ 1 is black, 2 is red and so on.
- ▶ Go around again starting with black.

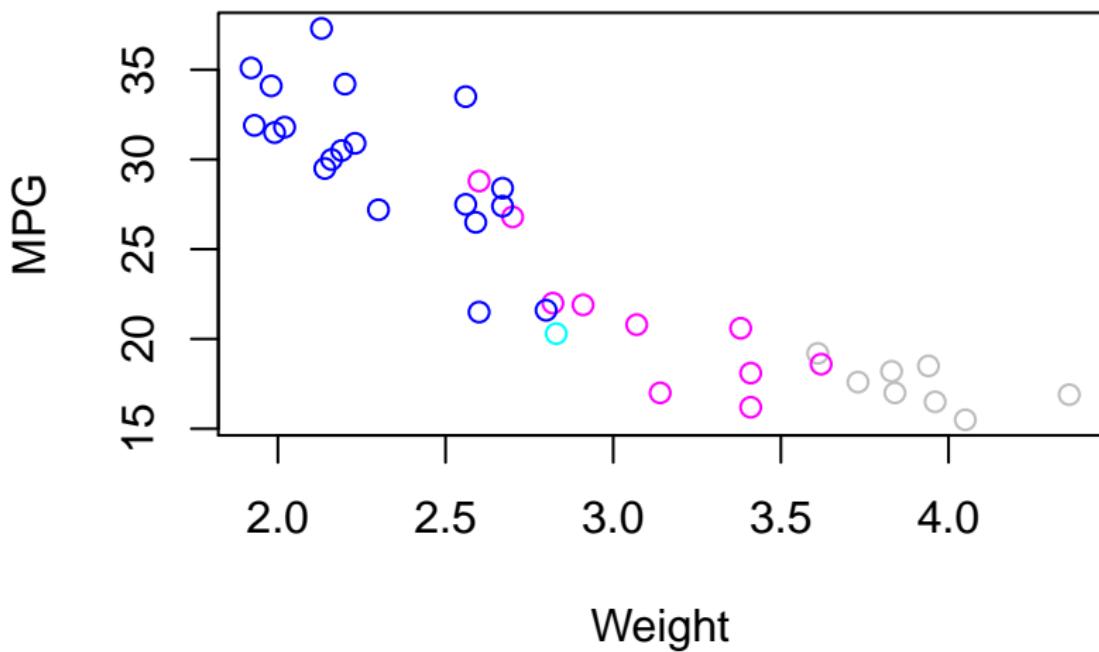
## Colourful plot



## Colouring by value

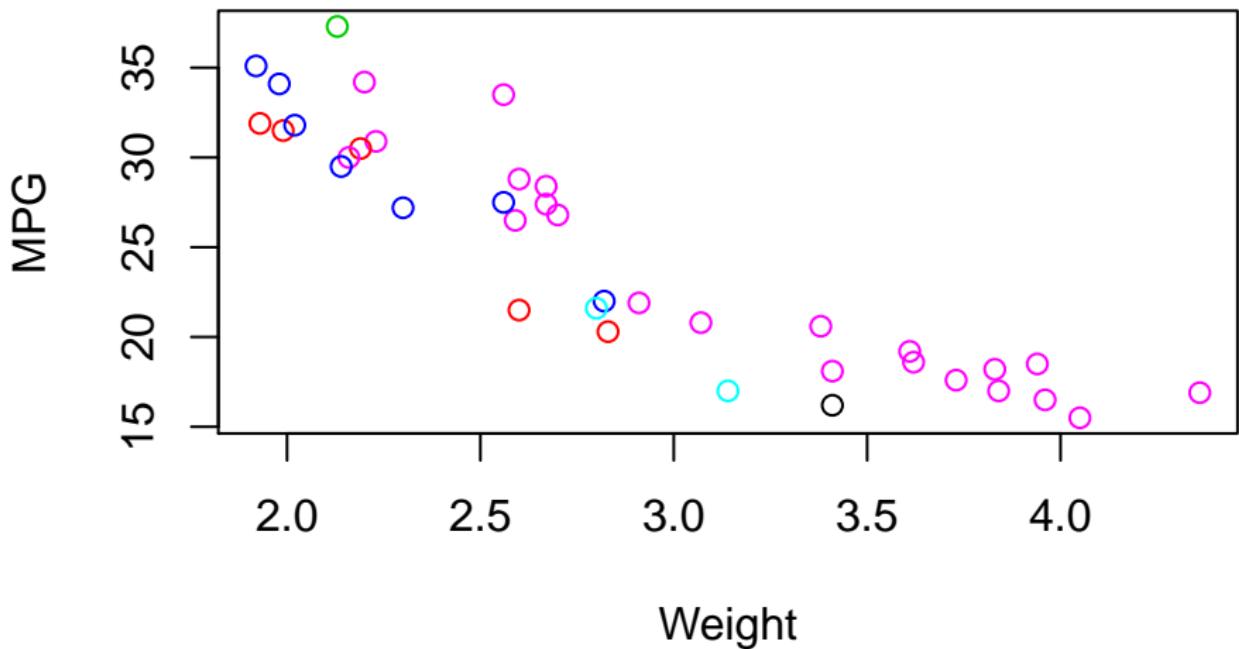
Can feed col a vector of numbers:

```
R> plot(Weight, MPG, col=Cylinders)
```



Factors get converted to numbers

R> `plot(Weight, MPG, col=Country)`

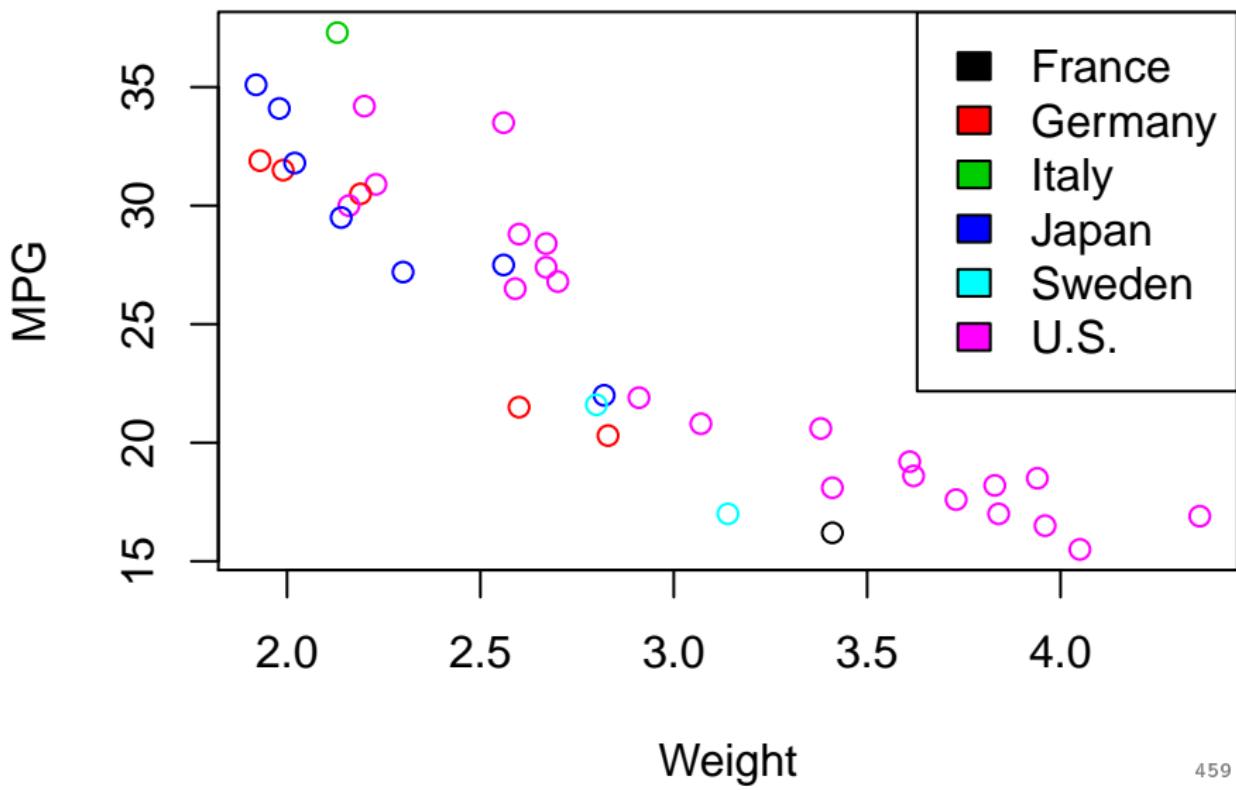


## Adding a legend

- ▶ But don't know which colour goes with which country: *legend*.
- ▶ Make a list of things that should be listed in legend (countries), one each. (Factor: `levels`.)
- ▶ Place to put legend: top right.
- ▶ What things identify countries: numbers 1–6.

```
R> plot(Weight, MPG, col=Country)
R> countries=levels(Country)
R> legend("topright", legend=countries, fill=1:6)
```

## Plot with legend

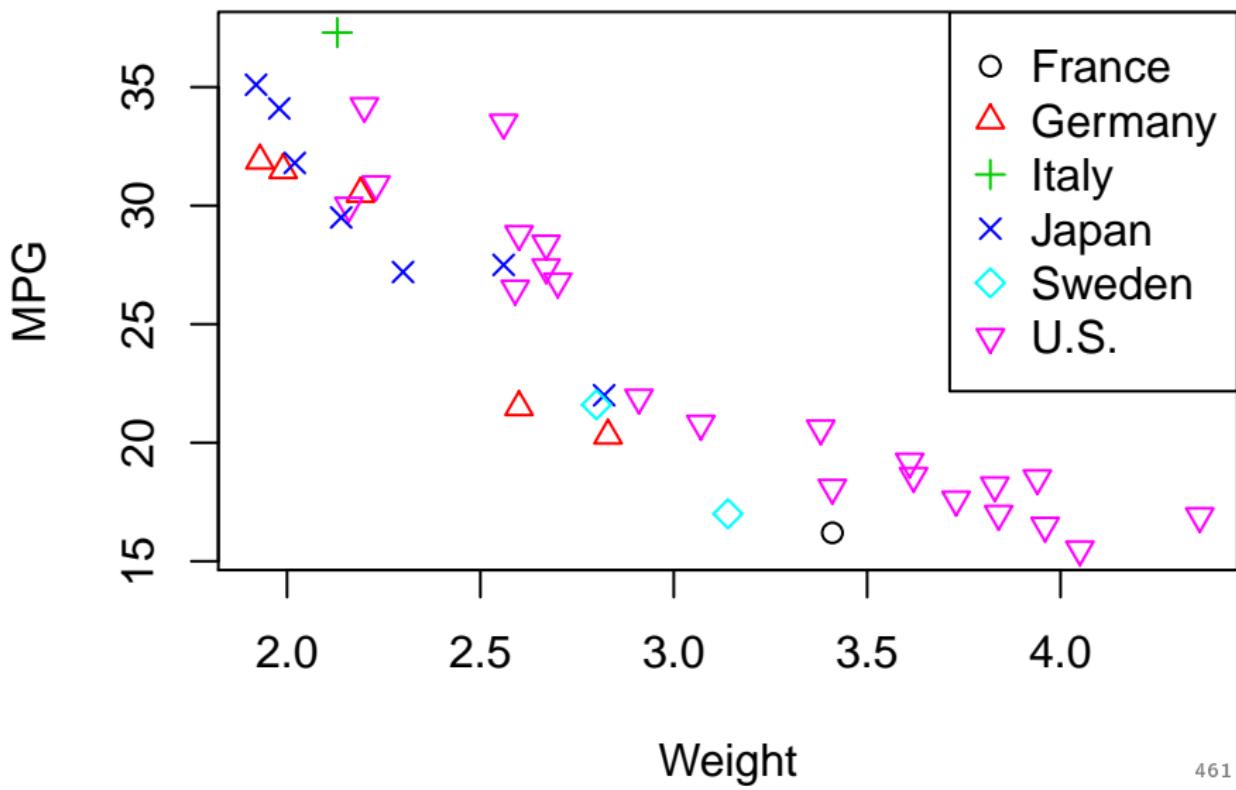


## Alternatively...

- ▶ Plotting character is pch (1, circle, is default).
- ▶ About 25 pchs, get them by number.
- ▶ pch doesn't work with factors; use as.numeric on factor.
- ▶ So distinguish countries by colours and plotting characters like this.  
Note: both col and pch in plot line, so also in legend:

```
R> plot(Weight, MPG, col=Country, pch=as.numeric(Country))  
R> countries=levels(Country)  
R> legend("topright", legend=countries, pch=1:6, col=1:6)
```

## The resulting plot

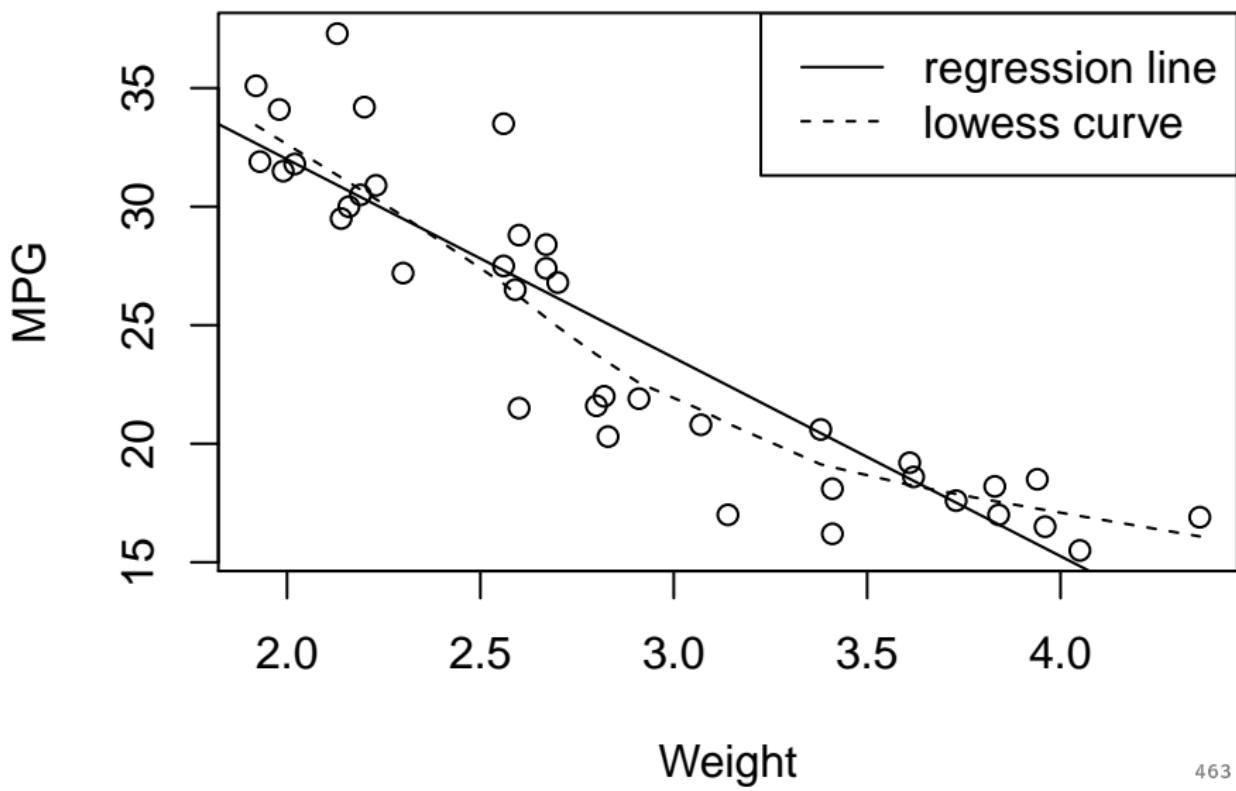


## Using legends to distinguish line types

- ▶ Scatter plot with regression line and lowess curve using different line types, shown on legend.
- ▶ Legend needs to contain a list of the line type names, plus the line types.
- ▶ Make vectors of these first, then pick out the appropriate ones when we draw the lines.

```
R> curve.types=c("regression line","lowess curve")
R> line.types=c("solid","dashed")
R> plot(Weight,MPG)
R> abline(MPG.lm,lty=line.types[1])
R> lines(lowess(Weight,MPG),lty=line.types[2])
R> legend("topright",legend=curve.types,lty=line.types)
```

# The plot

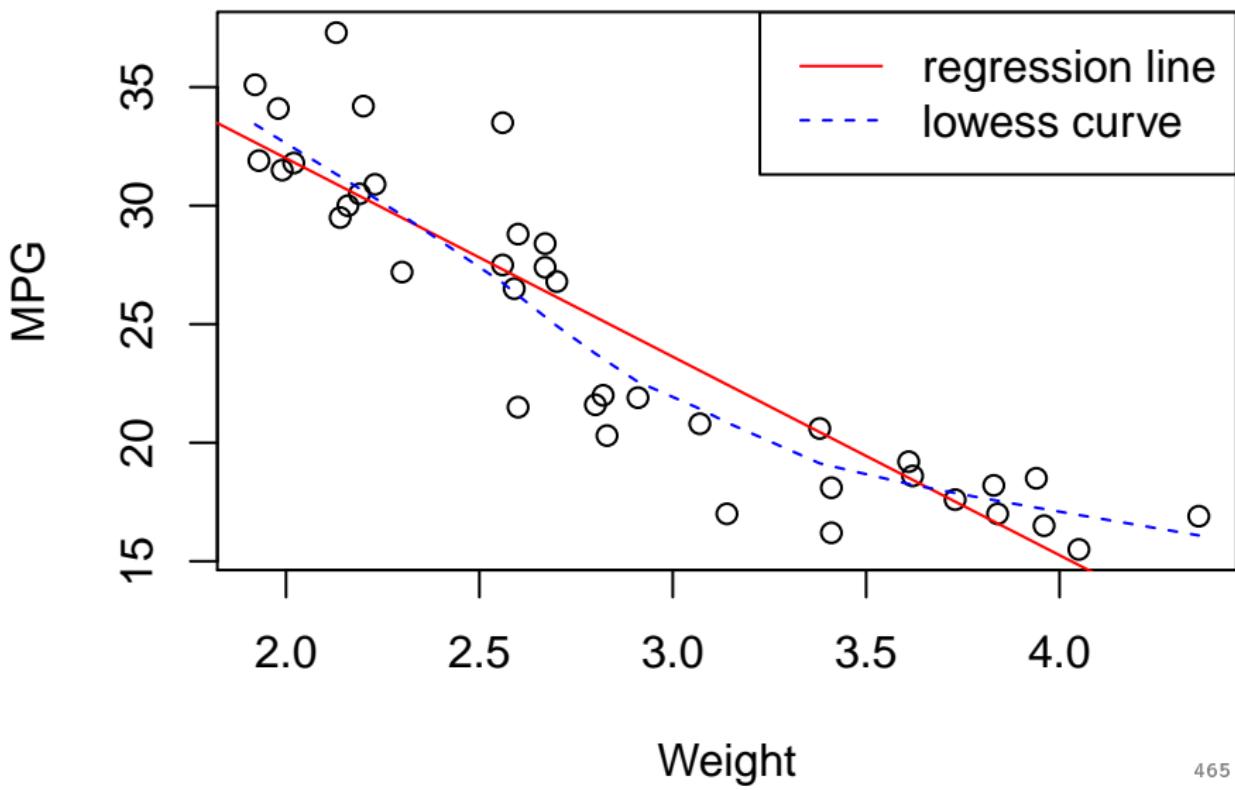


## Adding colours

- ▶ As before, but define a vector of colours.
- ▶ Pick out appropriate colour when drawing each line/curve
- ▶ Add colours to legend via col.

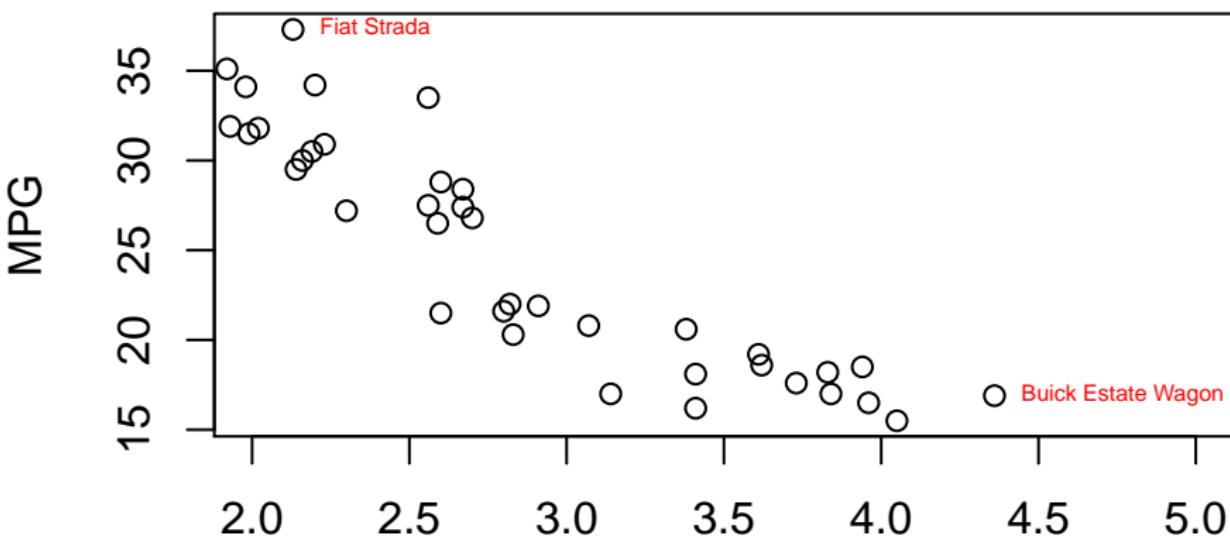
```
R> curve.types=c("regression line","lowess curve")
R> line.types=c("solid","dashed")
R> colours=c("red","blue")
R> plot(Weight,MPG)
R> abline(MPG.lm,lty=line.types[1],col=colours[1])
R> lines(lowess(Weight,MPG),lty=line.types[2],
R>       col=colours[2])
R> legend("topright",legend=curve.types,lty=line.types,
R>        col=colours)
```

## The plot, with colours



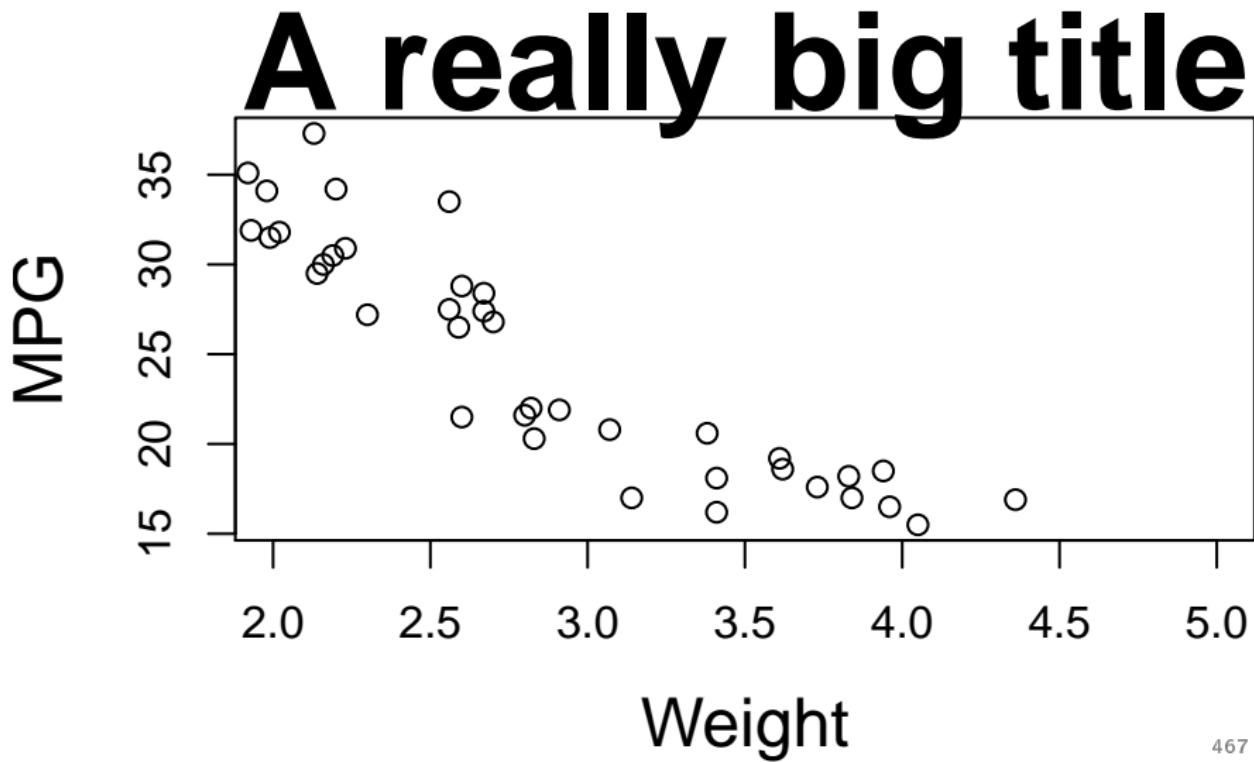
## Text size via cex

```
R> plot(Weight,MPG,xlim=c(2,5))  
R> text(Weight[9],MPG[9],Car[9],pos=4,col="red",cex=0.5)  
R> text(Weight[4],MPG[4],Car[4],pos=4,col="red",cex=0.5)
```



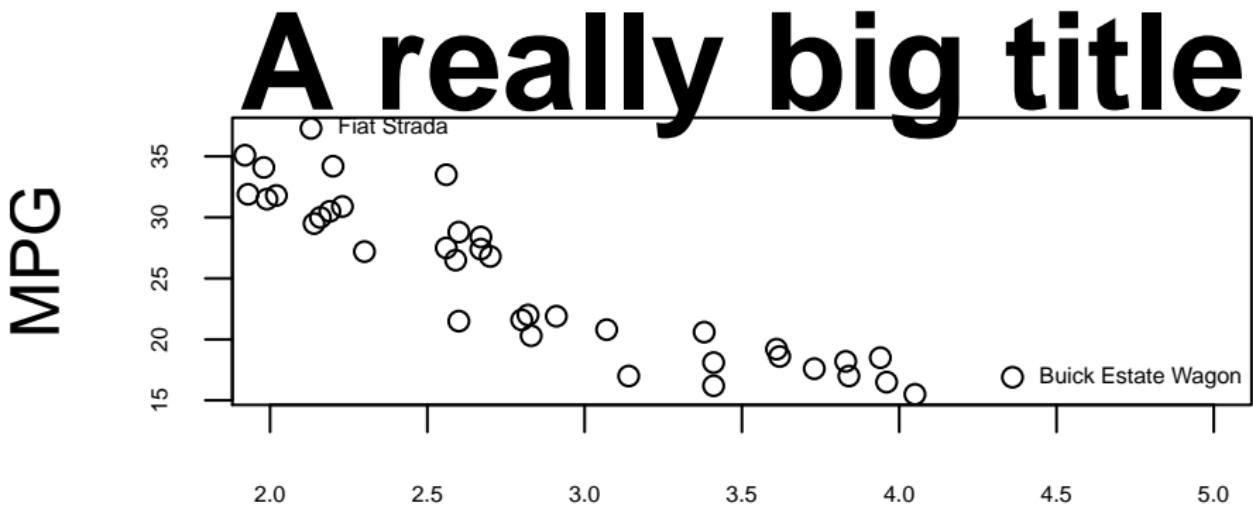
Title and axis text *really big*

```
R> plot(Weight,MPG,xlim=c(2,5),cex.lab=1.5,cex.main=3,  
R>     main="A really big title")
```



## Tiny

```
R> plot(Weight,MPG,xlim=c(2,5),cex.lab=1.5,cex.main=3,  
R>   main="A really big title",cex.axis=0.5,cex.sub=0.5,sub="A  
R> text(Weight[9],MPG[9],Car[9],pos=4,cex=0.5)  
R> text(Weight[4],MPG[4],Car[4],pos=4,cex=0.5)
```

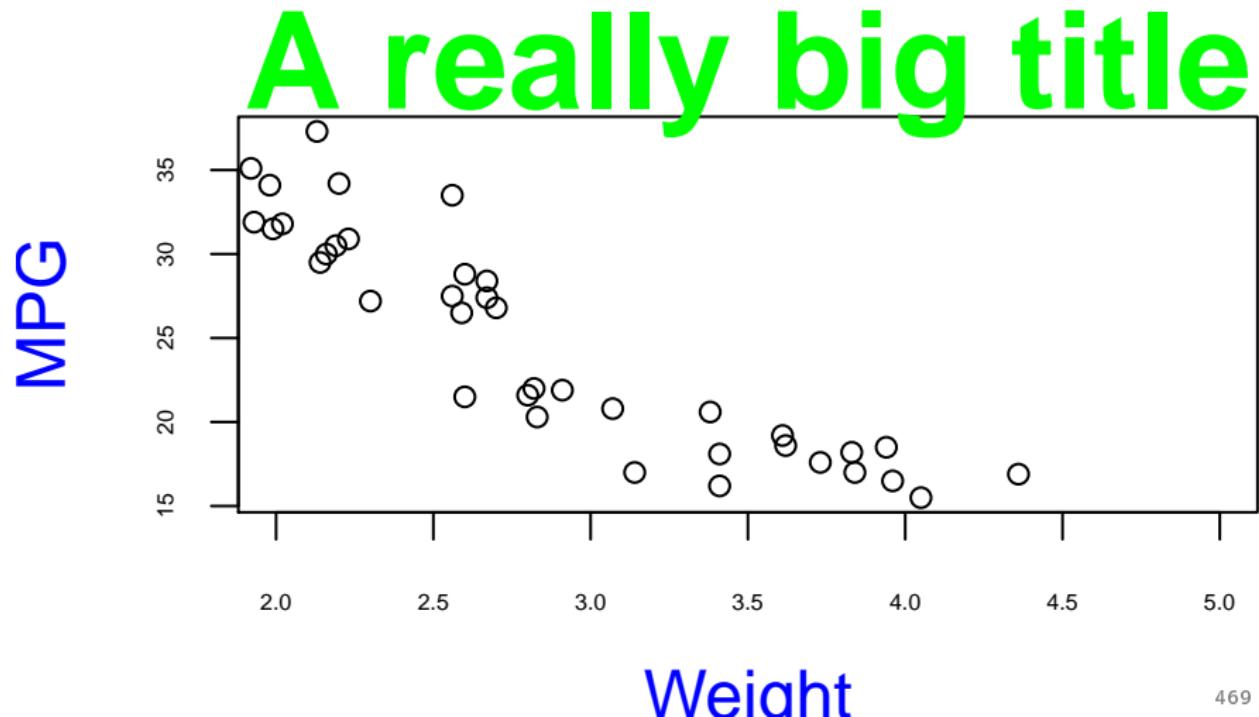


Weight

A tiny subtitle

## Colours via col.something

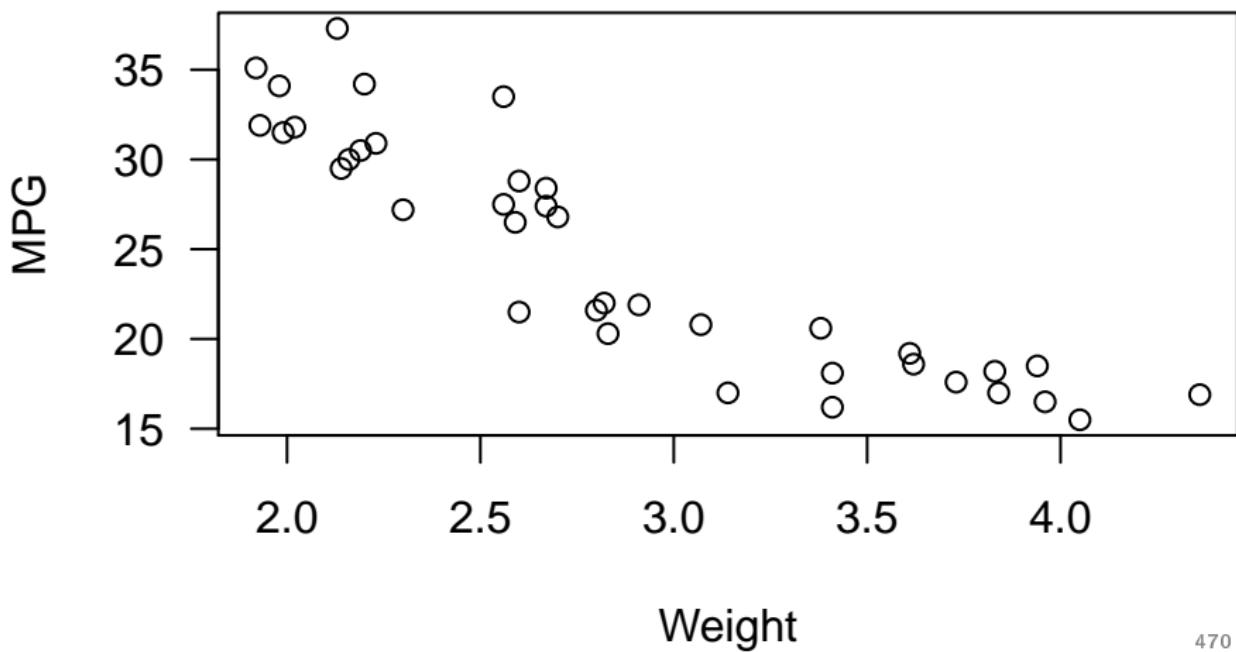
```
R> plot(Weight,MPG,xlim=c(2,5),cex.lab=1.5,cex.main=3,  
R>   main="A really big title",cex.axis=0.5,cex.sub=0.5,sub=  
R>   "A tiny subtitle",col.main="green",col.lab="blue")
```



## Numbers on axes

`las=1` makes all numbers horizontal:

```
R> plot(Weight, MPG, las=1)
```



## Multiple series

- ▶ On a spreadsheet, often plot multiple “series” on same graph.
- ▶ Oranges data:

```
R> oranges=read.table("oranges.txt",header=T)  
R> oranges
```

	row	ages	A	B	C	D	E
1	1	118	30	30	30	33	32
2	2	484	51	58	49	69	62
3	3	664	75	87	81	111	112
4	4	1004	108	115	125	156	167
5	5	1231	115	120	142	172	179
6	6	1372	139	142	174	203	209
7	7	1582	140	145	177	203	214

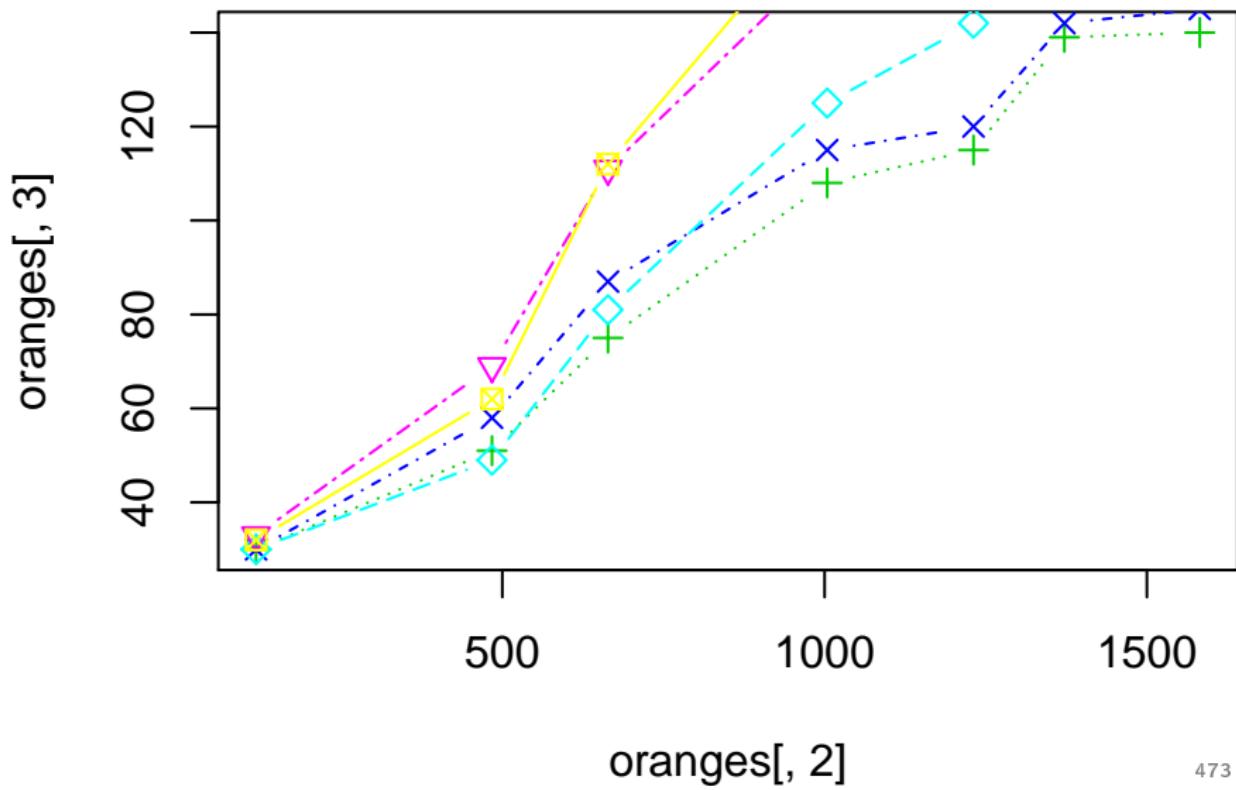
- ▶ Plot circumference against age for each tree A through E.
- ▶ Plot columns 3,4,5,6,7 against 2.
- ▶ Strategy: start with *empty* plot, add 5 “series” to plot one at a time.

## Code to make plot

- ▶ Start with `type="n"` to plot *nothing*, but set up axes etc.
- ▶ `type="b"` means plot both points and lines (join points by lines).
- ▶ Colour, line type and plotting character all accept numbers, so distinguish “growth curves” by all of them.

```
R> plot(oranges[,2],oranges[,3],type="n")
R> for (i in 3:7)
R>   {
R>     lines(oranges[,2],oranges[,i],type="b",col=i,lty=i,pch=i)
R>   }
```

# Plot, try #1



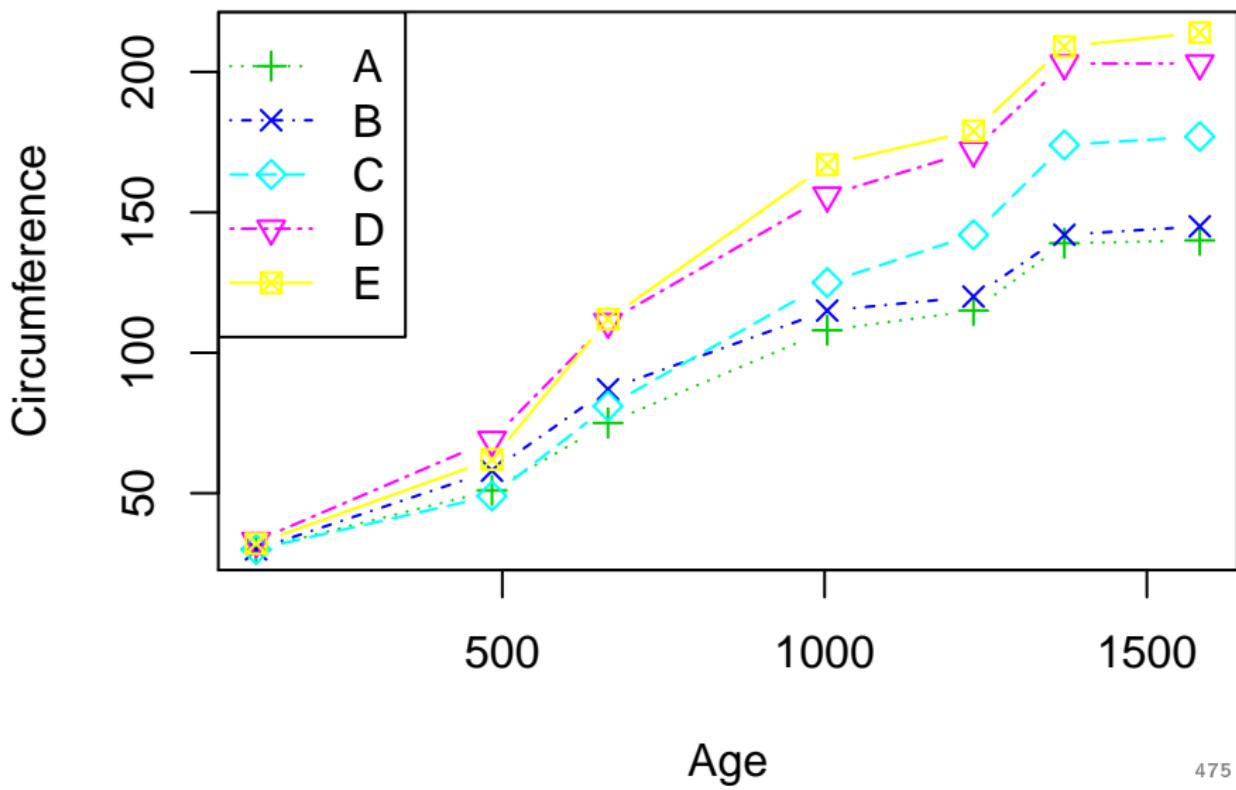
## Critique

Some problems with the plot:

- ▶ Growth curves go off top — need *y* axis to go up to the largest circumference of *any* of the trees.
- ▶ Axis labels are not meaningful.
- ▶ Need a legend to distinguish the trees.
- ▶ Put first two on plot command, legend after the loop:

```
R> circum.range=range(oranges[,3:7])
R> tree.names=c("A", "B", "C", "D", "E")
R> plot(oranges[,2],oranges[,3],type="n",ylim=circum.range,
R>       xlab="Age",ylab="Circumference")
R> for (i in 3:7)
R>   {
R>     lines(oranges[,2],oranges[,i],type="b",col=i,lty=i,
R>            pch=i)
R>   }
R> legend("topleft",legend=tree.names,col=3:7,lty=3:7,pch=3:7)
```

## Revised plot

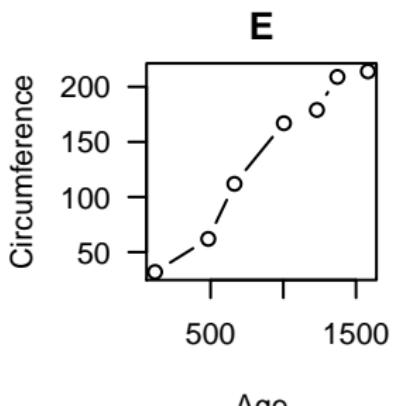
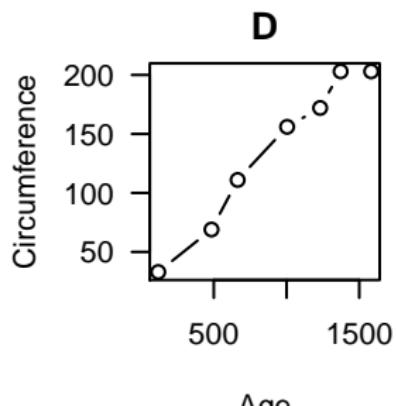
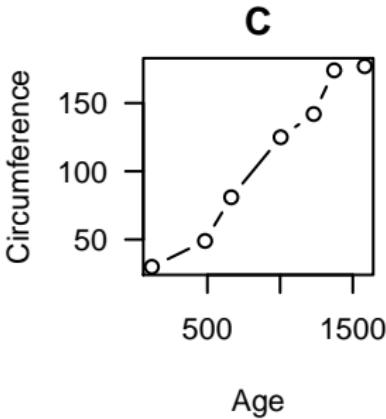
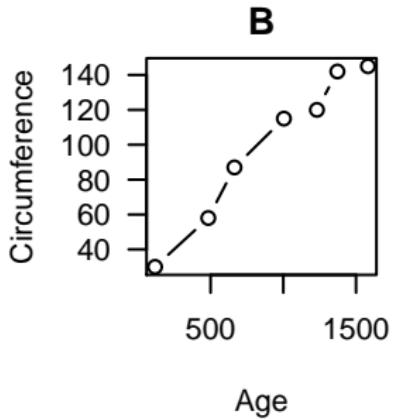
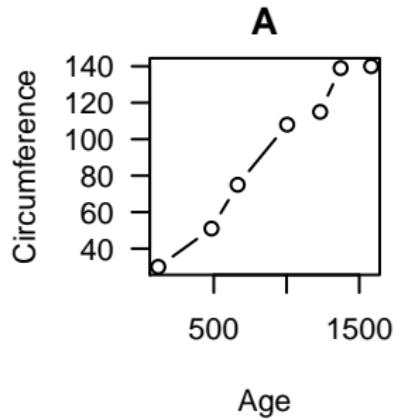


## Multiple plots on one page

- ▶ Sometimes want to plot several things small on one page.
- ▶ Use `mfrow` to set up array with given # of rows, columns. Call as shown.
- ▶ Demonstrate with oranges data in 2 rows and 3 columns (5 trees, so one plot blank).
- ▶ Some trickery to identify names of trees:

```
R> par(mfrow=c(2,3))
R> for (i in 3:7)
R> {
R>   plot(oranges[,2],oranges[,i],
R>         xlab="Age",ylab="Circumference",
R>         main=names(oranges)[i],type="b",las=1)
R> }
```

## The array of plots



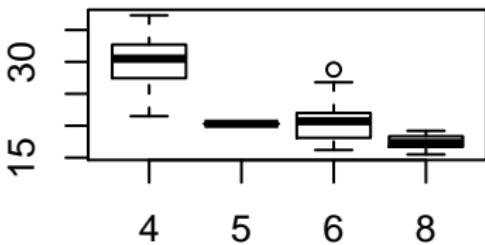
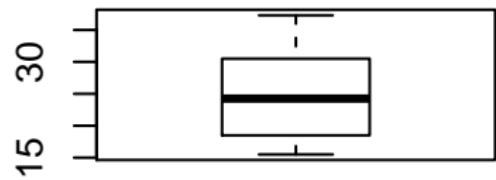
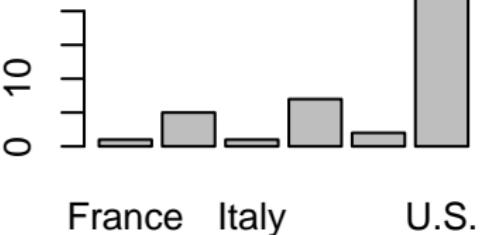
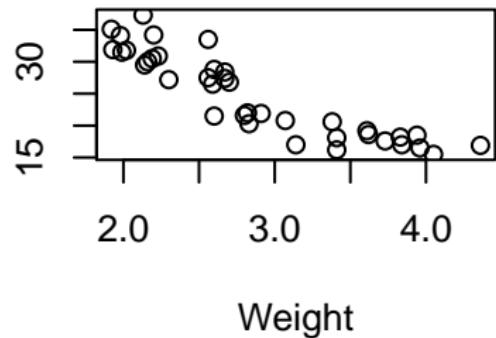
## Comments

- ▶ Shapes of growth patterns look very similar.
- ▶ I prefer the five curves on one plot.
- ▶ Plots don't have to be the same kind of thing. Any R graph will do.
- ▶ Pointless example using car data:

```
R> par(mfrow=c(2, 2))
R> plot(Weight, MPG)
R> plot(Country)
R> boxplot(MPG)
R> boxplot(MPG~Cylinders)
R> par(mfrow=c(1, 1))
```

- ▶ Last line resets plotting window so that next graph takes up whole region as normal.
- ▶ Otherwise next graph will be top left of a  $2 \times 2$  array, with other plots blank!

## The pointless plot(s)



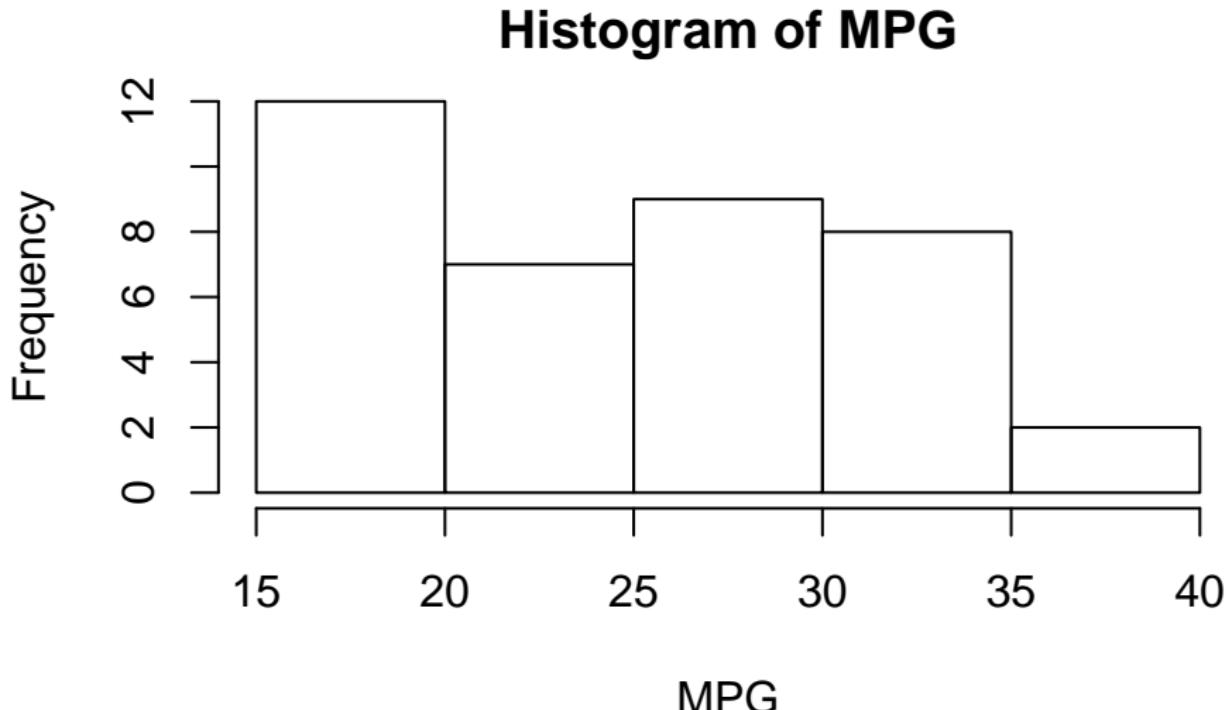
## Part X

Histograms and normal quantile plots

## Histograms: changing bar width

- ▶ Default histogram of car MPG:

*R> hist(MPG)*

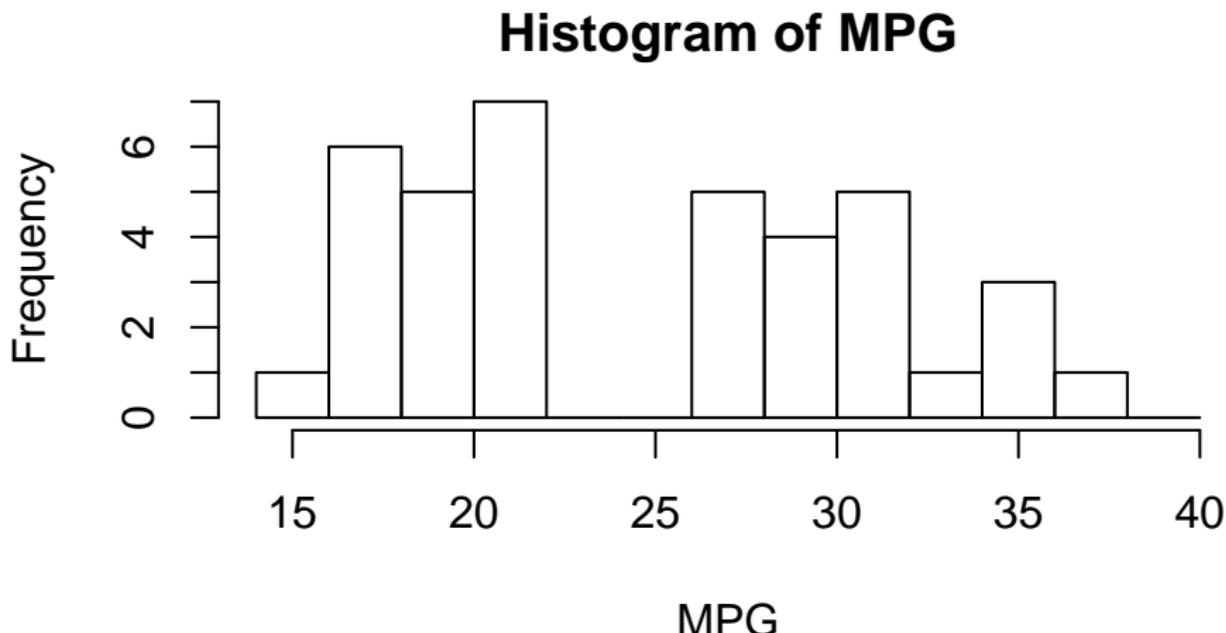


## Same histogram with more (too many?) bars

- ▶ via `breaks`. Set up “14 to 40 by 2’s” first:

```
R> mybreaks=seq(14,40,2)
```

```
R> hist(MPG,breaks=mybreaks)
```



## Kernel density curve

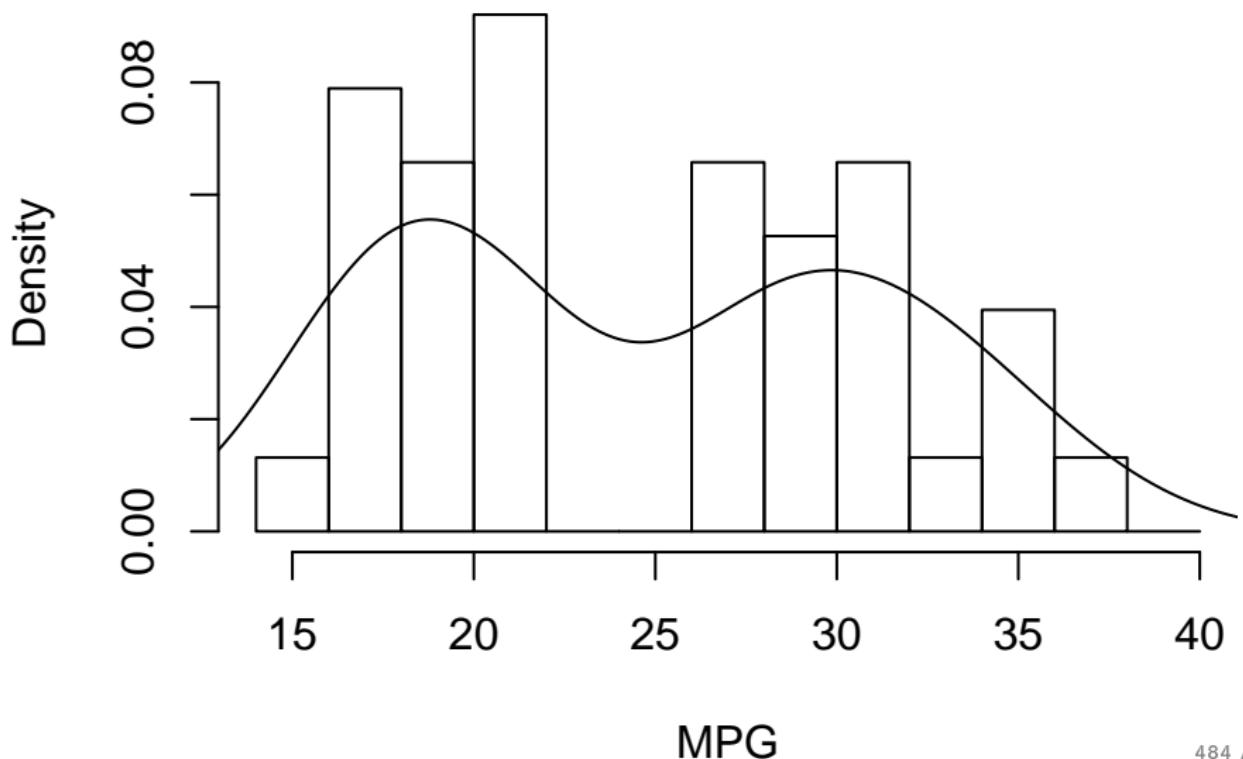
- ▶ Smoothed version of histogram.
- ▶ Shows us what underlying distribution might look like.
- ▶ Steps:
  - ▶ Make histogram show *relative frequencies* and use lots of bars
  - ▶ Add kernel density curve using lines

```
R> hist(MPG, breaks=mybreaks, probability=T)
```

```
R> lines(density(MPG))
```

## Histogram with kernel density curve

### Histogram of MPG



## Comments

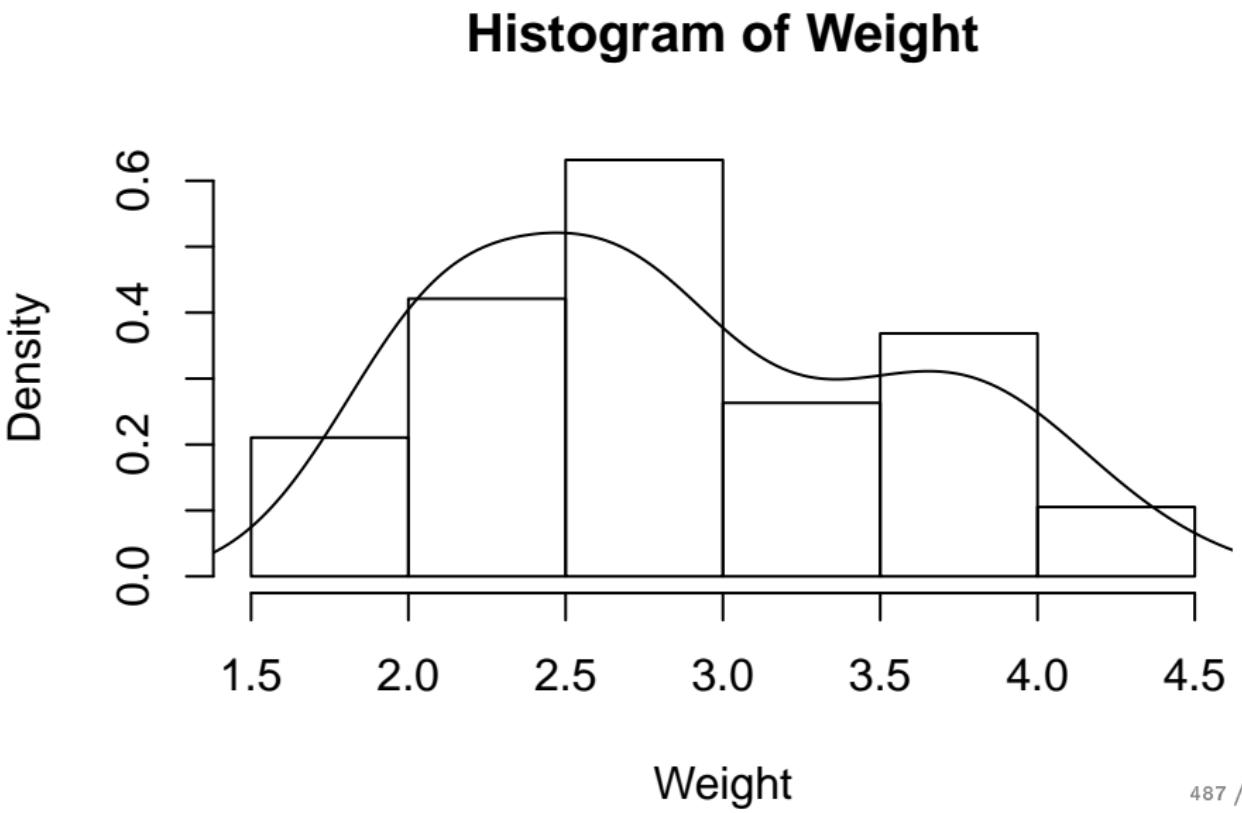
- ▶ Kernel density curve has two peaks, reflecting lack of cars with MPG around 25.
- ▶ Histogram with this many bars very “jumpy”; kernel density curve smooths pattern a lot.
- ▶ Density scale: 6 of the 38 values between 16 and 18; fraction of whole:  
 $R > 6/38$   
[1] 0.1578947
- ▶ but: interval is  $18 - 16 = 2$  wide, so density is  $6/38/2$  per unit MPG:  
 $R > (6/38)/2$   
[1] 0.07894737
- ▶ This last is height of 16 – 18 bar: just less than 0.08.

## The weights

- ▶ Are there two peaks in the car weights as well?

```
R> hist(Weight, probability=T)  
R> lines(density(Weight))
```

## Histogram with kernel density

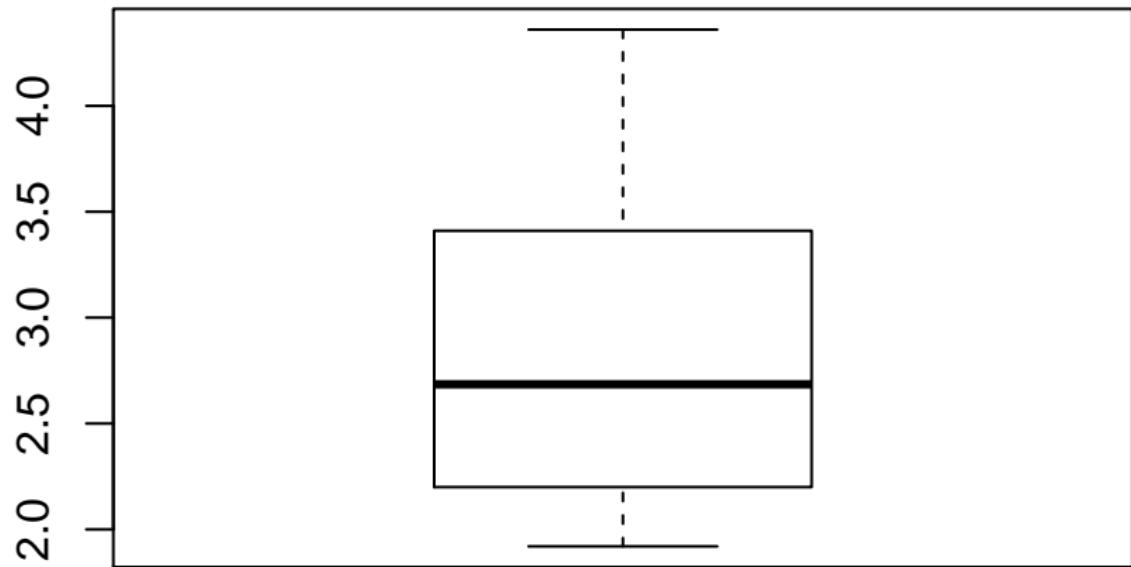


## Comments

- ▶ R less convinced of bimodality this time.
- ▶ Pattern more right-skewedness.
- ▶ What does a boxplot say?

```
R> boxplot(Weight)
```

## Boxplot of weight



## Part XI

### Assessing normality

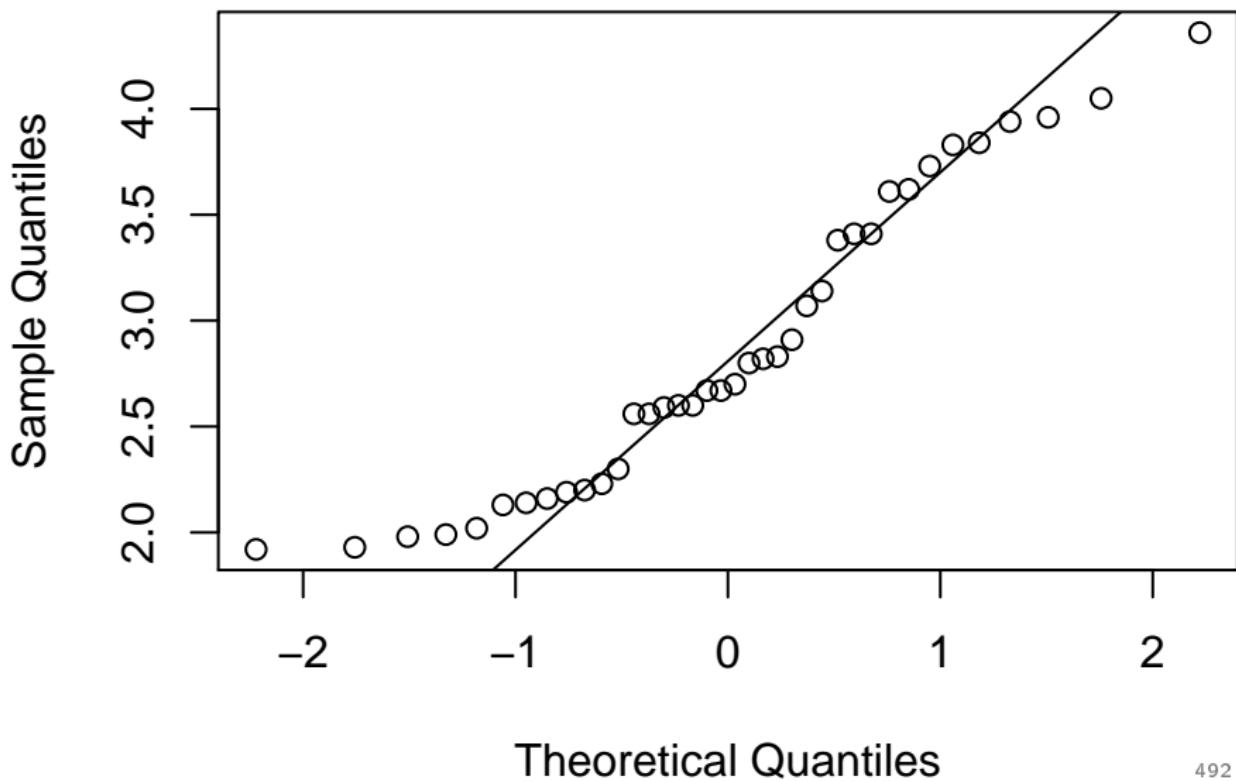
## Normal quantile plot

- ▶ Histogram, (especially) boxplot don't give focused assessment of whether a distribution is normal.
- ▶ Need *normal quantile plot*.
- ▶ Plot data values against *what you'd expect if normal distribution correct*.
- ▶ If normal *is* correct, get straight line. If not, get a curve.
- ▶ Car weights:

```
R> qqnorm(Weight)  
R> qqline(Weight)
```

Normal quantile plot for Weight

## Normal Q-Q Plot



## Comments

- ▶ If data were perfectly normal, values exactly on line.
- ▶ Data stray off the line a bit at the ends: low values especially are too big/bunched up for normal.
- ▶ Weights are not normal.
- ▶ How much deviation from the line might there be if data *really* normal?
- ▶ Generate some random normal data and find out:

```
R> z=rnorm(100)
```

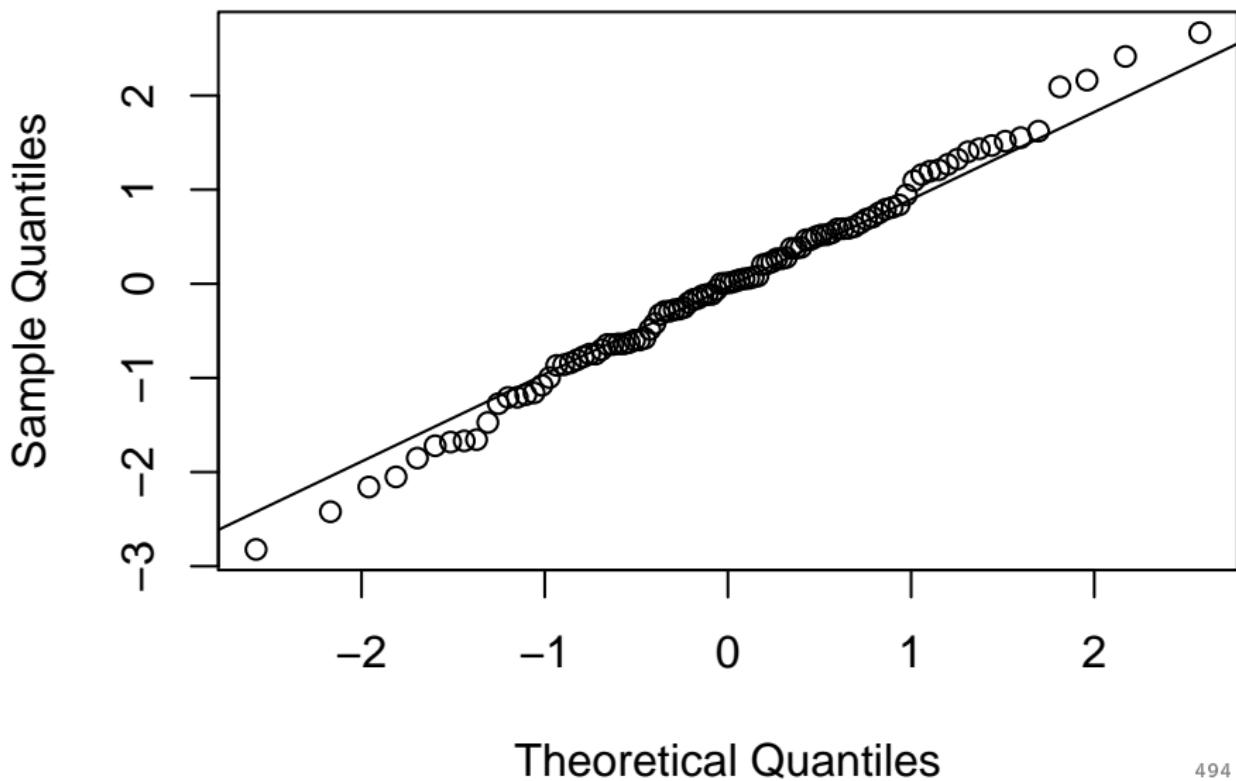
```
R> qqnorm(z)
```

```
R> qqline(z)
```

- ▶ See (over) that:
  - ▶ overall pattern of points is straight, not curved
  - ▶ points at extremes are not “drifting away” from line.

Normal quantile plot for genuinely normal data

## Normal Q-Q Plot



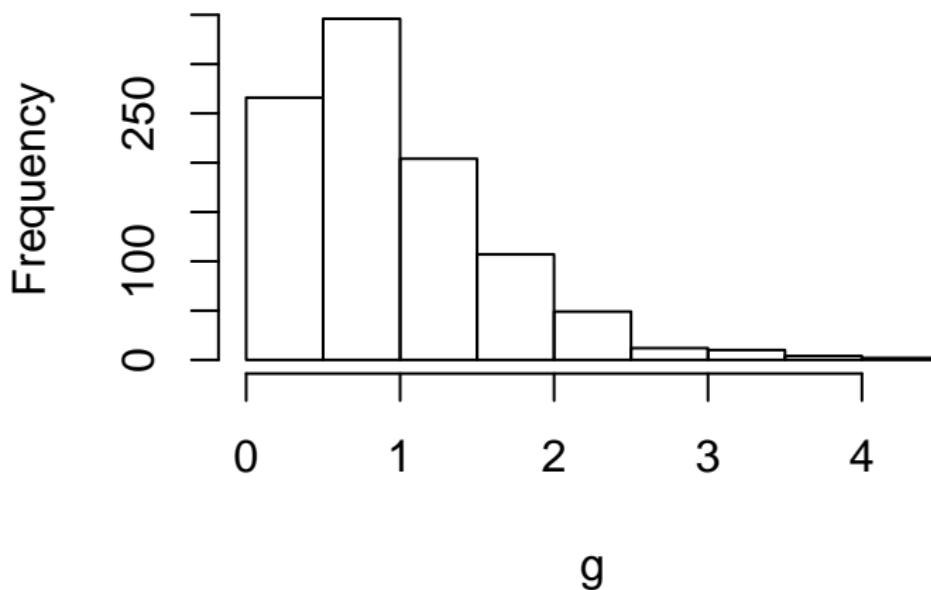
## Right-skewed data

- The *gamma distribution* is skewed to right:

```
R> g=rgamma(1000, 2, 2)
```

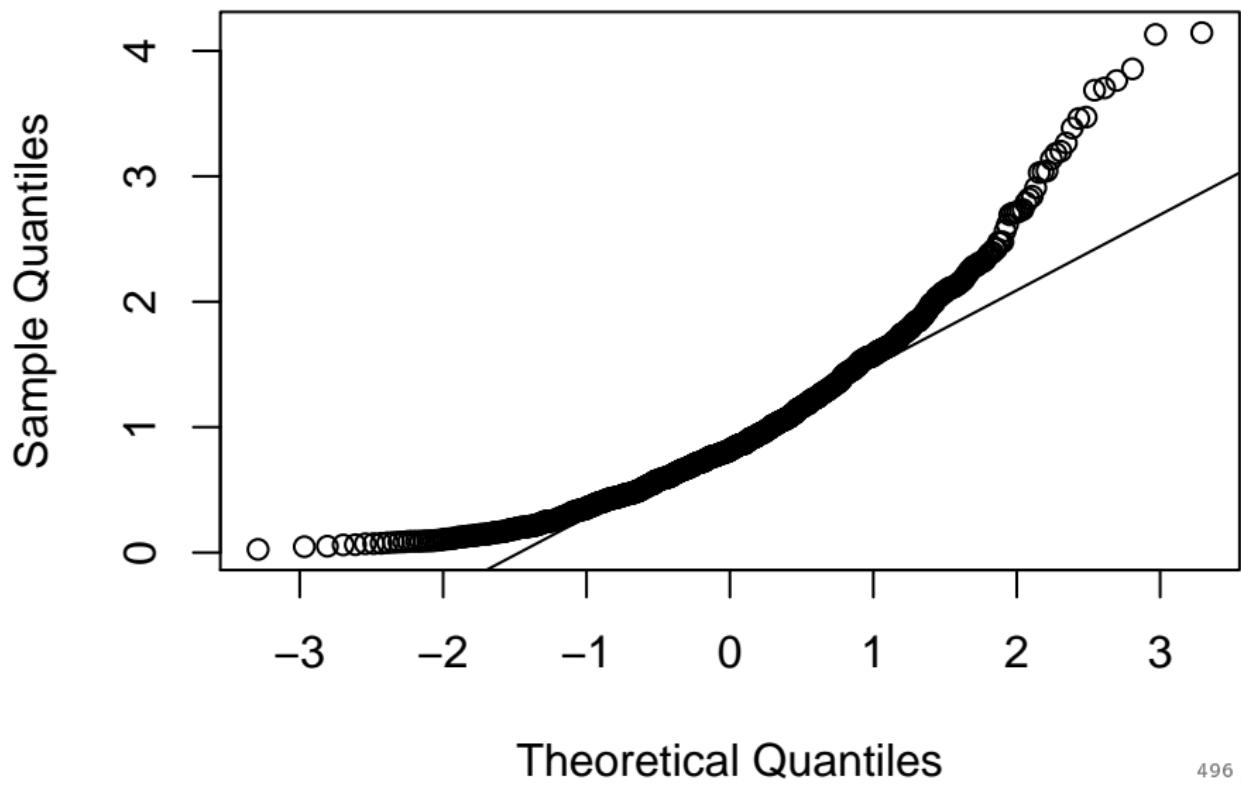
```
R> hist(g)
```

Histogram of g



Normal quantile plot for gamma data

## Normal Q-Q Plot



## Comments

- ▶ *Seriously* non-normal!
- ▶ Big-time curve on plot; points don't follow line at all.
- ▶ Observations at top end *too spread out* for normal.
- ▶ Observations at bottom end *bunched up* for normal.
- ▶ Skewness in direction of spread-out values: skewed *right*.

## Car MPGs

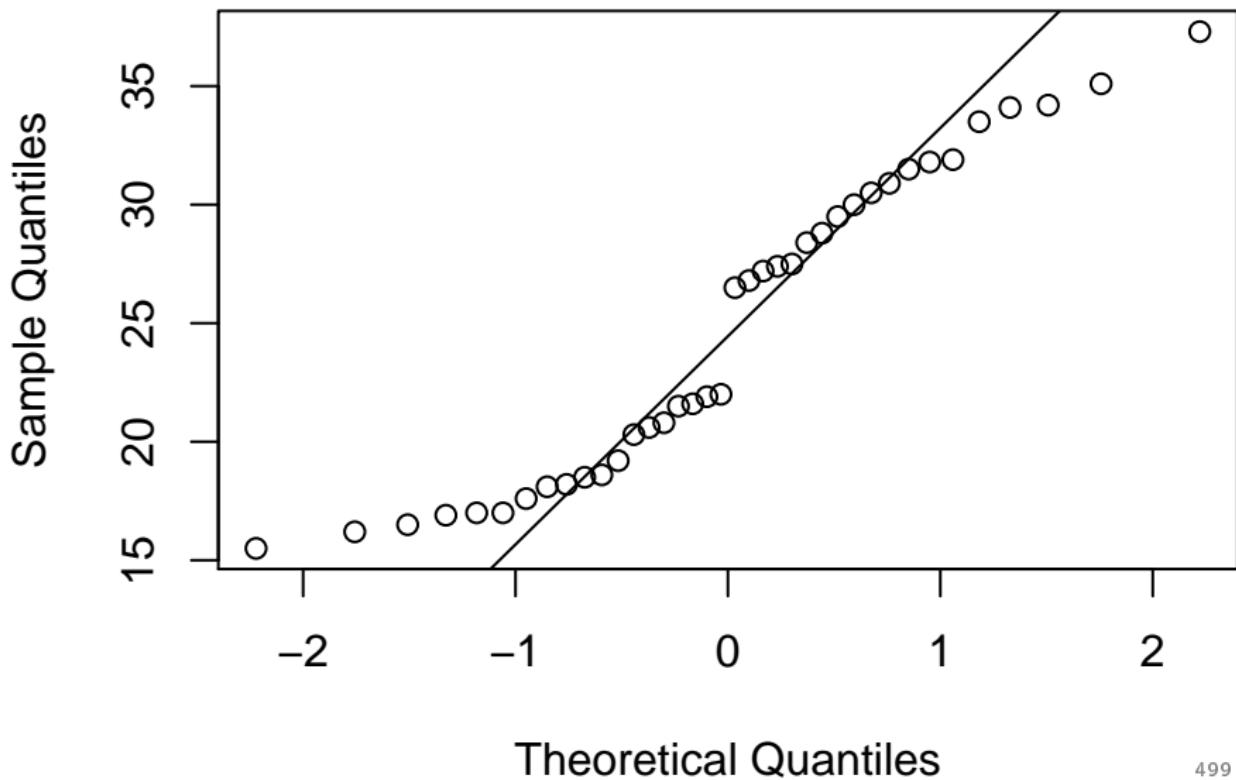
- ▶ Distribution had “hole” in middle — some low MPGs, and some high ones: not normal.
- ▶ How does this show up on normal quantile plot?

```
R> qqnorm(MPG)
```

```
R> qqline(MPG)
```

Normal quantile plot for car MPG

## Normal Q-Q Plot



## Comments

- ▶ “Hole” shows up as vertical gap.
- ▶ Almost S-bend in data values.
- ▶ High ones not high enough.
- ▶ Low ones not low enough.
- ▶ Data too *bunched up* to be normal (short tails).

## 68–95–99.7 and the MPG data

- ▶ MPG mean and SD:

```
R> xbar=mean(MPG) ; xbar ; s=sd(MPG) ; s  
[1] 24.76053  
[1] 6.547314
```

- ▶ If normal correct, 95% of values should be between

```
R> xbar-2*s ; xbar+2*s  
[1] 11.6659  
[1] 37.85515
```

- ▶ How many actually are? Look at sorted data:

```
R> sort(MPG)  
[1] 15.5 16.2 16.5 16.9 17.0 17.0 17.6 18.1 18.2 18.5 18.6  
[16] 21.5 21.6 21.9 22.0 26.5 26.8 27.2 27.4 27.5 28.4 28.8  
[31] 31.5 31.8 31.9 33.5 34.1 34.2 35.1 37.3
```

- ▶ All of them! So data *too bunched-up* to be normal, as normal quantile plot said.

## Part XII

Selecting and combining data

## Rows and columns

- ▶ Basic arrangement: **data frame**. Top few rows of cars:

```
R> head(cars)
```

	Car	MPG	Weight	Cylinders	Horsepower	Country
1	Buick Skylark	28.4	2.67	4	90	U.S.
2	Dodge Omni	30.9	2.23	4	75	U.S.
3	Mercury Zephyr	20.8	3.07	6	85	U.S.
4	Fiat Strada	37.3	2.13	4	69	Italy
5	Peugeot 694 SL	16.2	3.41	6	133	France
6	VW Rabbit	31.9	1.93	4	71	Germany

- ▶ Variable (column):

```
R> cars$Cylinders
```

```
[1] 4 4 6 4 6 4 4 4 8 5 8 6 4 4 4 6 4 6 6 8 4 6 6 4 4 8 6
```

- ▶ Row and/or column by *number* (Fiat Strada's MPG):

```
R> cars[4,2]
```

```
[1] 37.3
```

## Rows and columns (more)

- ▶ All of a row (eg. 4th):

```
R> cars[4,]
```

	Car	MPG	Weight	Cylinders	Horsepower	Country
4	Fiat Strada	37.3	2.13		4	69 Italy

- ▶ All of a column (eg. 2nd):

```
R> cars[, 2]
```

```
[1] 28.4 30.9 20.8 37.3 16.2 31.9 34.2 34.1 16.9 20.3 19.2  
[16] 22.0 31.5 28.8 26.8 18.5 27.2 18.1 20.6 21.6 31.8 17.6  
[31] 21.9 27.4 15.5 21.5 33.5 29.5 16.5 17.0
```

- ▶ Referring to variables by name, without \$:

```
R> attach(cars)
```

```
R> Cylinders
```

```
[1] 4 4 6 4 6 4 4 4 8 5 8 6 4 4 4 6 4 6 6 8 4 6 6 4 4 8 6
```

## Several rows/columns

- ▶ Several rows/columns. `6:9` means “6 through 9”, and `c(3,5)` means “3 and 5 only”:

```
R> cars[6:9,c(3,5)]
```

	Weight	Horsepower
6	1.93	71
7	2.20	70
8	1.98	65
9	4.36	155

- ▶ Or using names:

```
R> cars[6:9,c("Car","Cylinders")]
```

	Car	Cylinders
6	VW Rabbit	4
7	Plymouth Horizon	4
8	Mazda GLC	4
9	Buick Estate Wagon	8

- ▶ Likewise, leave row/column blank to select all.

## The logical vector

- ▶ How to select eg. “cars with 8 cylinders” or “cars whose gas mileage bigger than 30”?
- ▶ Idea:

```
R> Cylinders==8
```

```
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE  
[13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE  
[25] FALSE TRUE FALSE TRUE FALSE FALSE FALSE FALSE TRUE  
[37] TRUE TRUE
```

Go through cars: for each, TRUE if car has 8 cylinders, FALSE if not.

- ▶ == is “logical equals”: true if thing on left equals thing on right.  
Distinguish from =, saving right in variable on left.
- ▶ Can use both (I like to add brackets):  

```
R> has8=(Cylinders==8)
```

Work out logical vector as above, save in variable has8 to use later.
- ▶ Purpose: use logical vector to *select* things (that satisfy the logical condition that made the logical vector).

## About those cars that have 8 cylinders

- ▶ Which countries do the 8-cylinder cars come from?

*R> Country[has8]*

[1] U.S. U.S. U.S. U.S. U.S. U.S. U.S.

Levels: France Germany Italy Japan Sweden U.S.

All from the US.

- ▶ Gas mileages of 8-cylinder cars?

*R> MPG[has8]*

[1] 16.9 19.2 18.5 17.6 18.2 15.5 16.5 17.0

Bad.

## Greater than and less than

- ▶ Which cars have MPG of 30 or more? Directly:

```
R> cars[MPG>=30, ]
```

		Car	MPG	Weight	Cylinders	Horsepower	Country
2		Dodge Omni	30.9	2.23		4	75 U.S.
4		Fiat Strada	37.3	2.13		4	69 Italy
6		VW Rabbit	31.9	1.93		4	71 Germany
7		Plymouth Horizon	34.2	2.20		4	70 U.S.
8		Mazda GLC	34.1	1.98		4	65 Japan
13		VW Dasher	30.5	2.19		4	78 Germany
15		Dodge Colt	35.1	1.92		4	80 Japan
17		VW Scirocco	31.5	1.99		4	71 Germany
25		Datsun 210	31.8	2.02		4	65 Japan
30		Chevette	30.0	2.16		4	68 U.S.
35		Pontiac Phoenix	33.5	2.56		4	90 U.S.

- ▶ For “strictly greater than 30”, use `>30` instead of `>=30`.
- ▶ Do “less than” similarly.

## Greater and less, part 2

- Or get logical vector first, and use *that* for selection:

```
R> hi.mpg=(MPG>=30)
```

```
R> head(cars[hi.mpg,])
```

	Car	MPG	Weight	Cylinders	Horsepower	Country
2	Dodge Omni	30.9	2.23	4	75	U.S.
4	Fiat Strada	37.3	2.13	4	69	Italy
6	VW Rabbit	31.9	1.93	4	71	Germany
7	Plymouth Horizon	34.2	2.20	4	70	U.S.
8	Mazda GLC	34.1	1.98	4	65	Japan
13	VW Dasher	30.5	2.19	4	78	Germany

- How many cylinders do the high-MPG cars have?

```
R> Cylinders[hi.mpg]
```

```
[1] 4 4 4 4 4 4 4 4 4 4 4 4
```

All 4. Not a surprise. (Conditional distribution of number of cylinders given that MPG 30 or more.)

## And, or, not: “Not”

- ▶ How many cars *not* from the US? Set up logical vector first:

```
R> not.us=(Country!="U.S.")
```

```
R> not.us
```

```
[1] FALSE FALSE FALSE TRUE TRUE TRUE FALSE TRUE FALSE  
[13] TRUE FALSE TRUE TRUE TRUE FALSE FALSE FALSE TRUE  
[25] TRUE FALSE TRUE FALSE TRUE FALSE FALSE FALSE FALSE  
[37] FALSE FALSE
```

(note exact matching.)

- ▶ Use table to count:

```
R> table(not.us)
```

not.us

	FALSE	TRUE
22	16	

16 of 38 cars are not from US.

“And”

- ▶ Which cars from US *and* have 4-cylinder engines?

```
R> cars[Country=="U.S." & Cylinders==4,]
```

	Car	MPG	Weight	Cylinders	Horsepower	Country
1	Buick Skylark	28.4	2.67	4	90	U.S.
2	Dodge Omni	30.9	2.23	4	75	U.S.
7	Plymouth Horizon	34.2	2.20	4	70	U.S.
14	Ford Mustang	26.5	2.59	4	88	U.S.
30	Chevette	30.0	2.16	4	68	U.S.
32	AMC Spirit	27.4	2.67	4	80	U.S.
35	Pontiac Phoenix	33.5	2.56	4	90	U.S.

“Or”

- ▶ Or set up logical vector first:

```
R> i.want=(Country=="U.S." & Cylinders==4)
```

```
R> cars[i.want,]
```

	Car	MPG	Weight	Cylinders	Horsepower	Country
1	Buick Skylark	28.4	2.67	4	90	U.S.
2	Dodge Omni	30.9	2.23	4	75	U.S.
7	Plymouth Horizon	34.2	2.20	4	70	U.S.
14	Ford Mustang	26.5	2.59	4	88	U.S.
30	Chevette	30.0	2.16	4	68	U.S.
32	AMC Spirit	27.4	2.67	4	80	U.S.
35	Pontiac Phoenix	33.5	2.56	4	90	U.S.

## “Or”

- ▶ “Or” in R is |.
- ▶ So, “either from Japan or has 6 cylinders (or both)”:  
`R> cars[Country=="Japan" | Cylinders==6,]`

	Car	MPG	Weight	Cylinders	Horsepower	Country
3	Mercury Zephyr	20.8	3.07	6	85	U.S.
5	Peugeot 694 SL	16.2	3.41	6	133	France
8	Mazda GLC	34.1	1.98	4	65	Japan
12	Dodge Aspen	18.6	3.62	6	110	U.S.
15	Dodge Colt	35.1	1.92	4	80	Japan
16	Datsun 810	22.0	2.82	6	97	Japan
18	Chevy Citation	28.8	2.60	6	115	U.S.
19	Olds Omega	26.8	2.70	6	115	U.S.
21	Datsun 510	27.2	2.30	4	97	Japan
22	AMC Concord D/L	18.1	3.41	6	120	U.S.
23	Buick Century Special	20.6	3.38	6	105	U.S.
25	Datsun 210	31.8	2.02	4	65	Japan
27	Volvo 240 GL	17.0	3.14	6	125	Sweden
29	Toyota Corona	27.5	2.56	4	95	Japan
31	Ford Mustang Ghia	21.9	2.91	6	109	U.S.
36	Honda Accord LX	29.5	2.14	4	68	Japan

## Part XIII

### The apply family

## Doing things all at once

- ▶ R very good at “applying” things to entire data frames, vectors.
- ▶ For example, calculating means by subgroup.
- ▶ If you’re a programmer, might do these tasks using loops.
- ▶ But no need in R (and faster to use “apply” functions).

## The apply function

- ▶ Go back to orange tree circumferences:

```
R> circum=oranges[,3:7]
```

```
R> circum
```

	A	B	C	D	E
1	30	30	30	33	32
2	51	58	49	69	62
3	75	87	81	111	112
4	108	115	125	156	167
5	115	120	142	172	179
6	139	142	174	203	209
7	140	145	177	203	214

```
R> apply(circum,1,mean)
```

```
[1] 31.0 57.8 93.2 134.2 145.6 173.4 175.8
```

- ▶ These are row means: mean circumference at each time.
- ▶ apply needs 3 things: data frame (or matrix), dimension to work on (1 is rows, 2 columns), function to calculate for each row/column.

## A meaningless one

- ▶ Any function that produces a number can be used with `apply`, eg. `median`:

```
R> circum
```

	A	B	C	D	E
1	30	30	30	33	32
2	51	58	49	69	62
3	75	87	81	111	112
4	108	115	125	156	167
5	115	120	142	172	179
6	139	142	174	203	209
7	140	145	177	203	214

```
R> apply(circum, 2, median)
```

	A	B	C	D	E
	108	115	125	156	167

- ▶ Column medians (actually 4th number in each column, since values in order).

## A funky one

- ▶ The first quartile Q1 for each row:

```
R> apply(circum, 1, quantile, probs=0.25)  
[1] 30 51 81 115 120 142 145
```

- ▶ Problem: quantile needs extra stuff: *which* quantile?
  - ▶ quantile for just first row looks like this:
- ```
R> quantile(circum[1,], probs=0.25)
```
- ▶ Add extra, probs=0.25, onto the end of call to apply.

## Means etc. by groups

- ▶ Groups defined by values of categorical variable: tapply
- ▶ Back to cars: mean MPG (quantitative) for each Country (categorical).
- ▶ Use tapply. Feed in quantitative, categorical, function:

```
R> tapply(MPG, Country, mean)
```

|  | France   | Germany  | Italy    | Japan    | Sweden   | U.S.     |
|--|----------|----------|----------|----------|----------|----------|
|  | 16.20000 | 27.14000 | 37.30000 | 29.60000 | 19.30000 | 22.99545 |

- ▶ For combination of categorical variables, use list, eg. by Country and Cylinders:

```
R> tapply(MPG, list(Country, Cylinders), mean)
```

|         | 4        | 5    | 6        | 8      |
|---------|----------|------|----------|--------|
| France  | NA       | NA   | 16.20000 | NA     |
| Germany | 28.85000 | 20.3 | NA       | NA     |
| Italy   | 37.30000 | NA   | NA       | NA     |
| Japan   | 30.86667 | NA   | 22.00000 | NA     |
| Sweden  | 21.60000 | NA   | 17.00000 | NA     |
| U.S.    | 30.12857 | NA   | 22.22857 | 17.425 |

## Using tapply with function returning several values

- ▶ like eg. `quantile`, returns 5-number summary by default:

```
R> tapply(MPG, Country, quantile)
```

\$France

|     | 0%   | 25%  | 50%  | 75%  | 100% |
|-----|------|------|------|------|------|
| MPG | 16.2 | 16.2 | 16.2 | 16.2 | 16.2 |

\$Japan

|     | 0%    | 25%   | 50%   | 75%   | 100%  |
|-----|-------|-------|-------|-------|-------|
| MPG | 22.00 | 27.35 | 29.50 | 32.95 | 35.10 |

\$Germany

|     | 0%   | 25%  | 50%  | 75%  | 100% |
|-----|------|------|------|------|------|
| MPG | 20.3 | 21.5 | 30.5 | 31.5 | 31.9 |

\$Sweden

|     | 0%    | 25%   | 50%   | 75%   | 100%  |
|-----|-------|-------|-------|-------|-------|
| MPG | 17.00 | 18.15 | 19.30 | 20.45 | 21.60 |

\$Italy

|     | 0%   | 25%  | 50%  | 75%  | 100% |
|-----|------|------|------|------|------|
| MPG | 37.3 | 37.3 | 37.3 | 37.3 | 37.3 |

\$U.S.

|     | 0%     | 25%    | 50%    | 75%    | 100%   |
|-----|--------|--------|--------|--------|--------|
| MPG | 15.500 | 18.125 | 20.700 | 28.150 | 34.200 |

- ▶ Ugly. It's an R list.

## The aggregate function

Alternative to tapply is aggregate. This uses a model formula, and you have to specify the data frame from which stuff comes. Mean MPG by Country:

```
R> aggregate(MPG~Country, data=cars, mean)
```

|   | Country | MPG      |
|---|---------|----------|
| 1 | France  | 16.20000 |
| 2 | Germany | 27.14000 |
| 3 | Italy   | 37.30000 |
| 4 | Japan   | 29.60000 |
| 5 | Sweden  | 19.30000 |
| 6 | U.S.    | 22.99545 |

## By Country and Cylinders

```
R> aggregate(MPG~Country+Cylinders, data=cars, mean)
```

|    | Country | Cylinders | MPG      |
|----|---------|-----------|----------|
| 1  | Germany | 4         | 28.85000 |
| 2  | Italy   | 4         | 37.30000 |
| 3  | Japan   | 4         | 30.86667 |
| 4  | Sweden  | 4         | 21.60000 |
| 5  | U.S.    | 4         | 30.12857 |
| 6  | Germany | 5         | 20.30000 |
| 7  | France  | 6         | 16.20000 |
| 8  | Japan   | 6         | 22.00000 |
| 9  | Sweden  | 6         | 17.00000 |
| 10 | U.S.    | 6         | 22.22857 |
| 11 | U.S.    | 8         | 17.42500 |

## Five-number summary of MPG by Country

```
R> aggregate(MPG~Country,data=cars,quantile)
```

|   | Country | MPG.0% | MPG.25% | MPG.50% | MPG.75% | MPG.100% |
|---|---------|--------|---------|---------|---------|----------|
| 1 | France  | 16.200 | 16.200  | 16.200  | 16.200  | 16.200   |
| 2 | Germany | 20.300 | 21.500  | 30.500  | 31.500  | 31.900   |
| 3 | Italy   | 37.300 | 37.300  | 37.300  | 37.300  | 37.300   |
| 4 | Japan   | 22.000 | 27.350  | 29.500  | 32.950  | 35.100   |
| 5 | Sweden  | 17.000 | 18.150  | 19.300  | 20.450  | 21.600   |
| 6 | U.S.    | 15.500 | 18.125  | 20.700  | 28.150  | 34.200   |

And even...

Five-number summary of MPG by Country-Cylinders combo:

```
R> aggregate(MPG~Country+Cylinders,data=cars,quantile)
```

|    | Country | Cylinders | MPG.0% | MPG.25% | MPG.50% | MPG.75% | MPG.100% |
|----|---------|-----------|--------|---------|---------|---------|----------|
| 1  | Germany | 4         | 21.500 | 28.250  | 31.000  | 31.600  | 31.900   |
| 2  | Italy   | 4         | 37.300 | 37.300  | 37.300  | 37.300  | 37.300   |
| 3  | Japan   | 4         | 27.200 | 28.000  | 30.650  | 33.525  | 35.100   |
| 4  | Sweden  | 4         | 21.600 | 21.600  | 21.600  | 21.600  | 21.600   |
| 5  | U.S.    | 4         | 26.500 | 27.900  | 30.000  | 32.200  | 34.200   |
| 6  | Germany | 5         | 20.300 | 20.300  | 20.300  | 20.300  | 20.300   |
| 7  | France  | 6         | 16.200 | 16.200  | 16.200  | 16.200  | 16.200   |
| 8  | Japan   | 6         | 22.000 | 22.000  | 22.000  | 22.000  | 22.000   |
| 9  | Sweden  | 6         | 17.000 | 17.000  | 17.000  | 17.000  | 17.000   |
| 10 | U.S.    | 6         | 18.100 | 19.600  | 20.800  | 24.350  | 28.800   |
| 11 | U.S.    | 8         | 15.500 | 16.800  | 17.300  | 18.275  | 19.200   |

## The split function

- ▶ Another way to break things into groups (and then process separately) is via `split`:

```
R> split(MPG, Country)
```

```
$France
```

```
[1] 16.2
```

```
$Germany
```

```
[1] 31.9 20.3 30.5 31.5 21.5
```

```
$Italy
```

```
[1] 37.3
```

```
$Japan
```

```
[1] 34.1 35.1 22.0 27.2 31.8 27.5 2
```

```
$Sweden
```

```
[1] 21.6 17.0
```

```
$U.S.
```

```
[1] 28.4 30.9 20.8 34.2 16.9 19.2
```

```
[16] 30.0 21.9 27.4 15.5 33.5 16.5
```

- ▶ Creates a list.

## Applying a function to elements of a list

- ▶ Done via `sapply`:

```
R> my.list=split(MPG, Country)
R> sapply(my.list, quantile)
```

|      | France | Germany | Italy | Japan | Sweden | U.S.   |
|------|--------|---------|-------|-------|--------|--------|
| 0%   | 16.2   | 20.3    | 37.3  | 22.00 | 17.00  | 15.500 |
| 25%  | 16.2   | 21.5    | 37.3  | 27.35 | 18.15  | 18.125 |
| 50%  | 16.2   | 30.5    | 37.3  | 29.50 | 19.30  | 20.700 |
| 75%  | 16.2   | 31.5    | 37.3  | 32.95 | 20.45  | 28.150 |
| 100% | 16.2   | 31.9    | 37.3  | 35.10 | 21.60  | 34.200 |

- ▶ Get a nice table.
- ▶ Transposed from aggregate output.
- ▶ Also, this uses row/column *names* as opposed to embedding categories in output.

## Part XIV

### Vector and matrix algebra in R

## Vector addition

- ▶ Define a vector, then “add 2” to it:

```
R> u=c(2,3,6,5,7)
```

```
R> k=2
```

```
R> u+k
```

```
[1] 4 5 8 7 9
```

Adds 2 to each element.

- ▶ Adding vectors:

```
R> u
```

```
[1] 2 3 6 5 7
```

```
R> v=c(1,8,3,4,2)
```

```
R> u+v
```

```
[1] 3 11 9 9 9
```

Elementwise addition. (MAT A23: vector addition.)

## Scalar multiplication

As per A23:

```
R> k
```

```
[1] 2
```

```
R> u
```

```
[1] 2 3 6 5 7
```

```
R> k*u
```

```
[1] 4 6 12 10 14
```

Each element of vector multiplied by 2.

## “Vector multiplication”

What about this?

R> u

```
[1] 2 3 6 5 7
```

R> v

```
[1] 1 8 3 4 2
```

R> 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.)

## Combining different-length vectors

- ▶ No error here. What happens?

```
R> u
```

```
[1] 2 3 6 5 7
```

```
R> w=c(1,2)
```

```
R> u+w
```

```
[1] 3 5 7 7 8
```

Warning message:

In `u + w` : longer object length is not a multiple of shorter

- ▶ 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.

## Matrices

- ▶ Create matrix like this:

```
R> A=matrix(1:4,nrow=2,ncol=2)
```

```
R> 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:

```
R> B=matrix(5:8,nrow=2,ncol=2,byrow=T)
```

```
R> 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?

R> A

```
[,1] [,2]  
[1,] 1 3  
[2,] 2 4
```

R> B

```
[,1] [,2]  
[1,] 5 6  
[2,] 7 8
```

R> 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?

R> A

|      | [,1] | [,2] |
|------|------|------|
| [1,] | 1    | 3    |
| [2,] | 2    | 4    |

R> B

|      | [,1] | [,2] |
|------|------|------|
| [1,] | 5    | 6    |
| [2,] | 7    | 8    |

R> 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:

R> A

```
[,1] [,2]  
[1,] 1 3  
[2,] 2 4
```

R> B

```
[,1] [,2]  
[1,] 5 6  
[2,] 7 8
```

R> A %\*% B

```
[,1] [,2]  
[1,] 26 30  
[2,] 38 44
```

## Linear algebra stuff

- ▶ To solve system of equations

$Ax = w$  for  $x$ :

R> A

[,1] [,2]

[1,] 1 3  
[2,] 2 4

R> w

[1] 1 2

R> solve(A, w)

[1] 1 0

- ▶ To find the inverse of  $A$ :

R> A

[,1] [,2]

[1,] 1 3  
[2,] 2 4

R> solve(A)

[,1] [,2]

[1,] -2 1.5  
[2,] 1 -0.5

- ▶ You can check that these are correct.

## Functions in R

- ▶ Every time you use something in R that is a name with () brackets after, you are using a *function*.
- ▶ Can write your own functions.
- ▶ This is major way of extending what R can do.
- ▶ Function has two parts: what goes in, what comes out. Example:

```
R> c(1,2,5)
```

```
[1] 1 2 5
```

In: 3 numbers 1, 2 and 5. Out: vector containing those three numbers. Or:

```
R> u=c(1,2,5)
```

```
R> w=c(8,9)
```

```
R> c(u,w,6)
```

```
[1] 1 2 5 8 9 6
```

which glues all the numbers in `u` and `w` together and adds a 6 on the end.

## The mean function

- ▶ Calculates mean (duh!)
- ▶ Input: one vector of numbers
- ▶ Output: a number:

*R> mean(u)*

[1] 2.666667

## The lm function

- ▶ Does linear regression.
- ▶ Input: a model formula:

```
R> y=c(w,11)
```

```
R> lm(y~u)
```

Call:

```
lm(formula = y ~ u)
```

Coefficients:

| (Intercept) | u      |
|-------------|--------|
| 7.3846      | 0.7308 |

- ▶ Output: what you just saw?
- ▶ No: lm has *print method* print.lm, but actually a lot more inside an lm object — actually R list:

```
R> print.default(lm(y~u))
```

## What `lm` *really* returns (some)

```
$coefficients
(Intercept)           u
 7.3846154   0.7307692
$df.residual
[1] 1
$residuals
      1          2          3
-0.11538462  0.15384615 -0.03846154
$xlevels
named list()
$effects
(Intercept)           u
-16.1658075  2.1513264  0.1961161
$call
lm(formula = y ~ u)
$terms
y ~ u
attr(,"variables")
list(y, u)
attr(,"factors")
$rank
[1] 2
$fitted.values
      1          2          3
 8.115385  8.846154 11.038462
$assign
[1] 0 1
attr(,"term.labels")
[1] " "
```

## Writing a function

- ▶ Write function for *range*, highest minus lowest.
- ▶ Input: a vector of numbers. Output: single number.
- ▶ Start like this:

```
R> my.range=function(x)
```

- ▶ I called my function `my.range`.
- ▶ Whatever input called *outside* function, *inside* it's called `x`.
- ▶ Doing-stuff part enclosed in curly brackets:

```
R> my.range=function(x)
```

```
R> {  
R>   a=max(x)  
R>   b=min(x)  
R>   a-b  
R> }
```

- ▶ Calculated range (largest minus smallest).
- ▶ Last line just calculates without saving in variable. How R function sends value back to outside world.

## Making a function in R Studio

- ▶ Just type all the lines into a R Script window.
- ▶ When done, select all the lines of the function, and Run them (via Control-Enter or the Run button).
- ▶ No output means no error (“defined machine for calculating ranges”).  
If errors, R tells you, and you go back and fix function.

## One-liners and curly brackets

- ▶ Said that function “body” has to be inside curly brackets.
- ▶ Exception: if body of function has only one line, curly brackets *optional*.
- ▶ `my.range` as one-liner:

```
R> my.r=function(x) max(x)-min(x)
```

- ▶ But — I’ve written so many one-liners that turned into many-liners that I like to put curly brackets in.
- ▶ This?

```
R> f=function(x,y) mean(resid(lm(y~x)))
```

- ▶ Hint: start from *inside*.

## Using a function you wrote

- ▶ Same way as any other function in R. Examples:

```
R> v=c(4,5,2,7)
```

```
R> my.range(v)
```

```
[1] 5
```

or

```
R> my.range(c(10,13,14))
```

```
[1] 4
```

or even:

```
R> my.range(1)
```

```
[1] 0
```

- ▶ Does the last one make sense?

## Optional arguments

- ▶ Example: standardize a vector of values, but want option of passing in a mean and SD to standardize with:

```
R> stand=function(x,calc=T,mean=0,std=1)
R> {
R>   if (calc)
R>     {
R>       mu=mean(x)
R>       sigma=sd(x)
R>       z=(x-mu)/sigma
R>     }
R>   else
R>     {
R>       z=(x-mean)/std
R>     }
R>   z
R> }
```

- ▶ Values on function line with = in them *optional*: if you don't provide a value, the one next to the = used instead.

## What's in that function

- ▶ Top line:

```
R> stand=function(x,calc=T,mean=0,std=1)
```

- ▶ calc says “do I want to standardize using sample mean and SD” (T) or use values I feed in (F)? If I don’t say, use sample values.
- ▶ mean and std are mean and SD to use if calc=F (ignored otherwise)
- ▶ mean and std have default values as well, so that no error if calc=F and no values given for them.
- ▶ Want function to work two different ways, depending on calc.

## What's in that function (2)

- ▶ If calc true, calculate sample mean and SD, use those:

```
R>     if (calc)
R>     {
R>       mu=mean(x)
R>       sigma=sd(x)
R>       z=(x-mu)/sigma
R>     }
```

- ▶ Otherwise, use supplied values:

```
R>     else
R>     {
R>       z=(x-mean)/std
R>     }
```

- ▶ Whichever way it was calculated, send z back to outside world:

```
R>     z
R>   }
```

## stand in action

- ▶ Using defaults (use mean and SD of data):

```
R> v=c(2,3,6,8)
```

```
R> stand(v)
```

```
[1] -0.9986254 -0.6354889  0.4539206  1.1801937
```

- ▶ Explicitly telling the function to calc (same thing):

```
R> stand(v,calc=T)
```

```
[1] -0.9986254 -0.6354889  0.4539206  1.1801937
```

- ▶ Use entered mean and SD:

```
R> stand(v,mean=6,std=2,calc=F)
```

```
[1] -2.0 -1.5  0.0  1.0
```

- ▶ Without names (have to be in right order):

```
R> stand(v,F,6,2)
```

```
[1] -2.0 -1.5  0.0  1.0
```

## stand in action (2)

- ▶ Standardizing a single value (have to give mean and std):

```
R> stand(5,mean=6,std=2,calc=F)
```

```
[1] -0.5
```

- ▶ Does this make any sense?

```
R> v
```

```
[1] 2 3 6 8
```

```
R> stand(v,calc=F)
```

```
[1] 2 3 6 8
```

- ▶ Gives an answer (“don’t change v”) because of my defensive programming.

## 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. n.  
Output: probability of succeeding for 1st time after exactly x trials (number).
- ▶ One-liner:

```
R> geometric=function(x,p) p*(1-p)^(x-1)
```

- ▶ Or with curly brackets:

```
R> geometric=function(x,p)
R> {
R>   p*(1-p)^(x-1)
R> }
```

- ▶ Testing:

```
R> geometric(1,0.4)
[1] 0.4
```

Prob. of succeeding first time same as p: good.

## Errors

- ▶ Chance of first success on second trial? Fail, then succeed:

```
R> geometric(2, 0.4)
```

```
[1] 0.24
```

$$(0.6)(0.4) = 0.24.$$

- ▶ What if user gives p outside of [0, 1], or x less than 1?
- ▶ Function dies with error. Or gives nonsense answer.
- ▶ Catch that first. Now:

```
R> geometric(0, 0.5)
```

```
[1] 1
```

and

```
R> geometric(2, 1.1)
```

```
[1] -0.11
```

- ▶ Ugh!

## Catching errors

- ▶ `stopifnot`: feed it some logical conditions, stops operation of function if any condition false. (If all true, nothing happens).
- ▶ If any condition false, R tells you which one.
- ▶ 3 things to check:  $p \geq 0$  or bigger,  $p \leq 1$  or smaller,  $x \geq 1$  or bigger:

```
R> geometric=function(x,p)
R> {
R>   stopifnot(p>=0,p<=1,x>=1)
R>   p*(1-p)^(x-1)
R> }
```

- ▶ Test:

```
R> geometric(2,0.5)
```

```
[1] 0.25
```

```
R> geometric(0,0.5)
```

```
Error: x >= 1 is not TRUE
```

```
R> geometric(2,1.1)
```

```
Error: p < 1 is not TRUE
```

Both fail, and `stopifnot` tells you why.

## Calling `geometric` with vector `x`

- ▶ What happens?
- ▶ Try it and see.

```
R> 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:

```
R> sum(geometric(1:5, 0.5))
```

```
[1] 0.96875
```

- ▶ Might know this as “cumulative probability”.

## 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.  $p$ :

```
R> c.geometric=function(x,p)
R> {
R>   probs=geometric(1:x,p)
R>   sum(probs)
R> }
```

- ▶ Easy to write, using our geometric function and stuff in R.

## Testing `c.geometric`

- ▶ Try the one we just did:

```
R> c.geometric(5, 0.5)
```

```
[1] 0.96875
```

Answer we had before.

- ▶ How about this:

```
R> 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.

- ▶ For total number of trials, mean number until 1st success is  $1/p$ :

- ▶  $p = 0.5$ , mean #trials is  $1/0.5 = 2$ .

- ▶  $p = 0.1$ , mean #trials is  $1/0.1 = 10$ .

## Using R's geometric calculator

- ▶ Called `pgeom`:

```
R> c.geometric(5, 0.5)
```

```
[1] 0.96875
```

```
R> c.geometric(20, 0.1)
```

```
[1] 0.8784233
```

```
R> pgeom(5, 0.5)
```

```
[1] 0.984375
```

```
R> 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:

```
R> pgeom(4, 0.5)
```

```
[1] 0.96875
```

```
R> pgeom(19, 0.1)
```

```
[1] 0.8784233
```

and *that* works.

## 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**:

```
R> c2.geometric=function(x,p)
R> {
R>   total=0
R>   for (i in 1:x)
R>     {
R>       prob=geometric(i,p)
R>       total=total+prob
R>     }
R>   total
R> }
```

- ▶ Check:

```
R> c2.geometric(5,0.5)
[1] 0.96875
```

## Passing on arguments

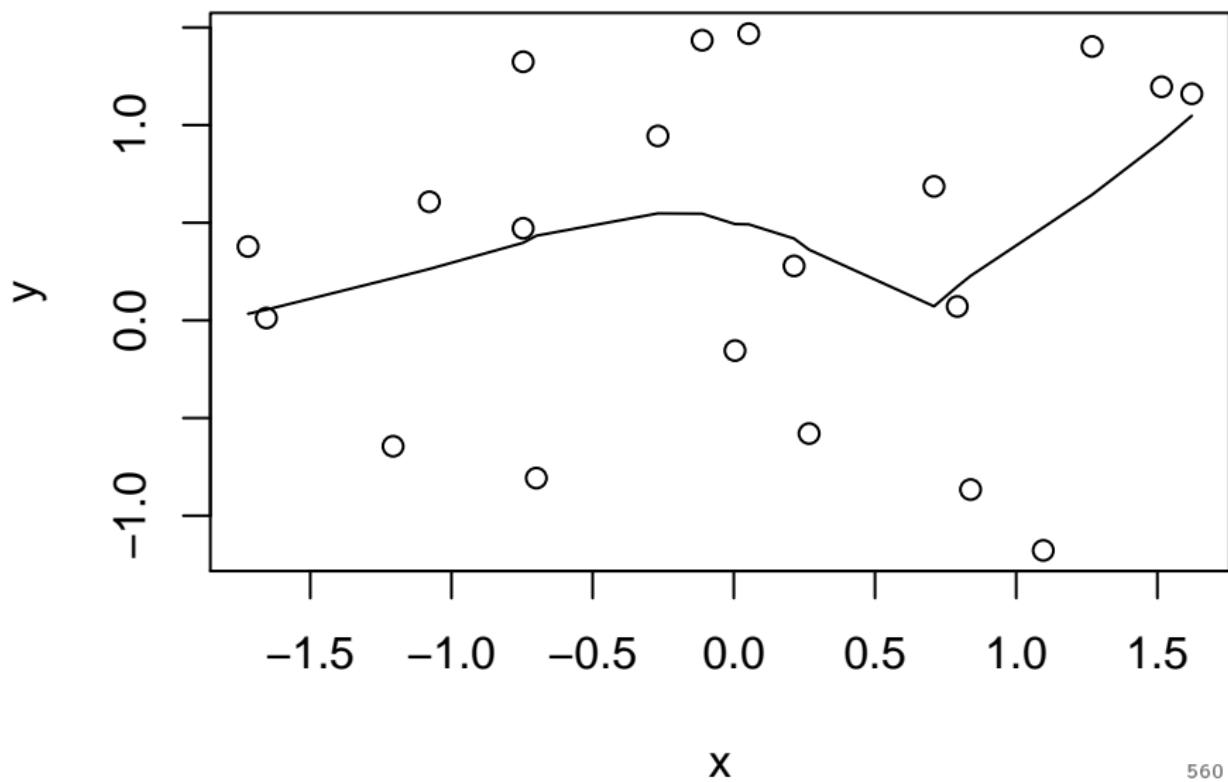
- ▶ Make a function that draws a scatterplot for two input variables  $x$  and  $y$ , and puts lowess curve on it.
- ▶ Not much to it:

```
R> scatter=function(x,y)
R> {
R>   plot(x,y)
R>   lines(lowess(x,y))
R> }
```

- ▶ Testing with random data:

```
R> set.seed(457299)
R> x=rnorm(20)
R> y=rnorm(20)
R> scatter(x,y)
```

# The plot



## Passing on arguments (2)

- ▶ What can we add to this? How about making colour, line type, etc., of lowess curve configurable?
- ▶ Pass col to scatter?

```
R> scatter(x,y,col="green")
```

```
Error in scatter(x, y, col = "green") :  
  unused argument (col = "green")
```

- ▶ scatter not expecting anything called col. The col has to go through scatter and into lowess.

## Passing on arguments (3)

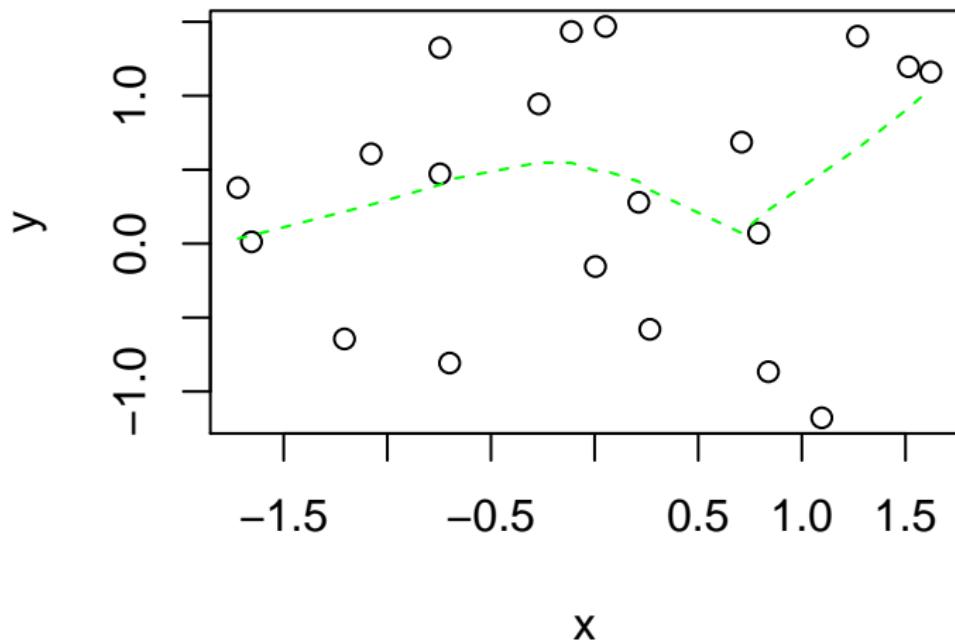
- ▶ One way: have arguments col, lty etc. on scatter, with defaults.  
But we might want to pass some other graphical parameters on to lines: painful.
- ▶ R has more flexible mechanism:

```
R> scatter2=function(x,y,...)
R> {
R>   plot(x,y)
R>   lines(lowess(x,y),...)
R> }
```

- ▶ ... means “other arguments”. Anything that scatter2 doesn't use will get passed on to lines. As long as lines understands it, all good.

## Trying it out

```
R> scatter2(x,y,col="green",lty="dashed")
```



## Passing something that lines doesn't know about

```
R> scatter2(x,y,col="red",hair="spiky")
```

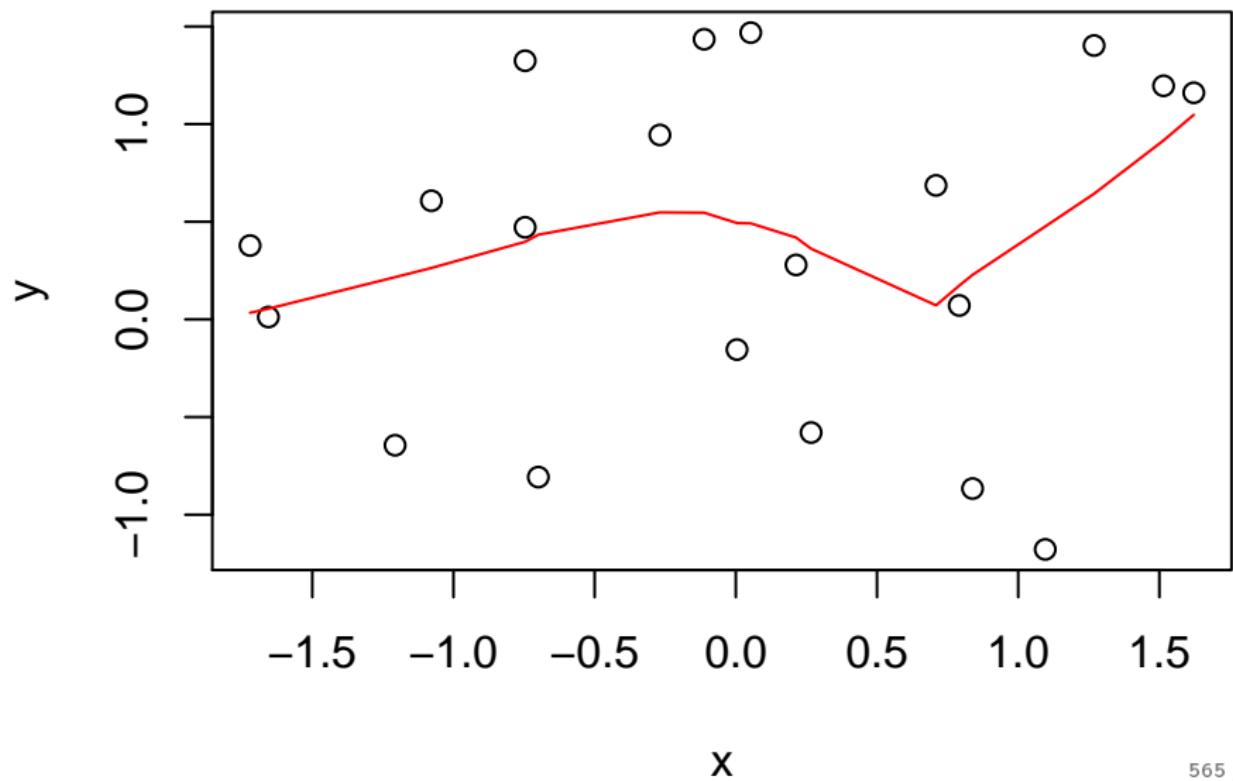
Warning message:

```
In plot.xy(xy.coords(x, y), type = type, ...):  
  "hair" is not a graphical parameter
```

Warning messages:

```
1: In plot.window(...) : "hair" is not a graphical parameter  
2: In plot.xy(xy, type, ...) : "hair" is not a graphical parameter  
3: In axis(side = side, at = at, labels = labels, ...):  
  "hair" is not a graphical parameter  
4: In axis(side = side, at = at, labels = labels, ...):  
  "hair" is not a graphical parameter  
5: In box(...) : "hair" is not a graphical parameter  
6: In title(...) : "hair" is not a graphical parameter  
7: In plot.xy(xy.coords(x, y), type = type, ...):  
  "hair" is not a graphical parameter
```

The plot still comes out



## Part XV

### Intricacies of SAS

## Reading data from a file

- ▶ This:

```
a 20  
a 21  
a 16  
b 11  
b 14  
b 17  
b 15  
c 13  
c 9  
c 12  
c 13
```

- ▶ got read in like this:

```
SAS> data groups;  
SAS>   infile 'threegroups.dat';  
SAS>   input group $ y;  
SAS>  
SAS> run;
```

## More than one observation per line

- ▶ Foregoing worked with:

- ▶ One obs. per line
  - ▶ Separated by whitespace.

- ▶ Suppose you have this:

3 4 5 6 7 7

8 9 3 4

- ▶ Eg. one variable x, then:

```
SAS> data xonly;  
SAS>   infile 'one.dat';  
SAS>   input x @@;  
SAS>  
SAS> proc print;
```

- ▶ @@ means “keep reading on same line until done”.

## The output

| Obs | x |
|-----|---|
| 1   | 3 |
| 2   | 4 |
| 3   | 5 |
| 4   | 6 |
| 5   | 7 |
| 6   | 7 |
| 7   | 8 |
| 8   | 9 |
| 9   | 3 |
| 10  | 4 |

If you leave off the @@?

- ▶ Data:

```
3 4 5 6 7 7  
8 9 3 4
```

- ▶ Code and output, without @@:

```
SAS> data xonly;  
SAS>   infile 'one.dat';  
SAS>   input x;  
SAS>  
SAS> proc print;
```

| Obs | x |
|-----|---|
| 1   | 3 |
| 2   | 8 |

- ▶ Only got the 1st value off each line.

## Two variables using @@

- ▶ Data:

```
3 4 5 6 7 7  
8 9 3 4
```

- ▶ Suppose values in data file are an x then a y, repeated:

```
SAS> data xonly;  
SAS>   infile 'one.dat';  
SAS>   input x y @@;  
SAS>  
SAS> proc print;
```

| Obs | x | y |
|-----|---|---|
| 1   | 3 | 4 |
| 2   | 5 | 6 |
| 3   | 7 | 7 |
| 4   | 8 | 9 |
| 5   | 3 | 4 |

## Skipping over header lines

- ▶ Data file like this:

```
x y  
3 4  
5 6  
7 7  
8 9  
3 4
```

- ▶ In SAS, supply variable names (on input line), so skip over header lines like this:

```
SAS> data xy;  
SAS>   infile 'two.dat' firstobs=2;  
SAS>   input xx yy;  
SAS>  
SAS> proc print;
```

- ▶ Can put any number on firstobs, depending on how many lines you want to skip.
- ▶ Variable names in data file (if any) *skipped*: SAS uses ones on input line.

## Data as read in

Note variable names:

| Obs | xx | yy |
|-----|----|----|
| 1   | 3  | 4  |
| 2   | 5  | 6  |
| 3   | 7  | 7  |
| 4   | 8  | 9  |
| 5   | 3  | 4  |

## Data separated by other things

- ▶ Separated by *commas*, so read in like this:

- ▶ Might have data like this:

3,4

5,6

7,7

8,9

3,4

- ▶ Eg. from spreadsheet saved as .csv.

```
SAS> data xy;
```

```
SAS>   infile 'three.dat' dlm=', ';
```

```
SAS>   input x y;
```

```
SAS>
```

```
SAS> proc print;
```

| Obs | x | y |
|-----|---|---|
| 1   | 3 | 4 |
| 2   | 5 | 6 |
| 3   | 7 | 7 |
| 4   | 8 | 9 |
| 5   | 3 | 4 |

## The singers

- ▶ Spreadsheet of female singer names, saved as .csv:

1,Bessie Smith  
2,Peggy Lee  
3,Aretha Franklin  
4,Diana Ross  
5,Dolly Parton  
6,Tina Turner  
7,Madonna  
8,Mary J Blige  
9,Salt n Pepa  
10,Aaliyah  
11,Beyonce

- ▶ Try reading in via SAS:

```
SAS> data singers;  
SAS> infile 'singers.csv' dlm=',';  
SAS> input number name $;  
SAS>  
SAS> proc print;
```

| Obs | number | name     |
|-----|--------|----------|
| 1   | 1      | Bessie S |
| 2   | 2      | Peggy Le |
| 3   | 3      | Aretha F |
| 4   | 4      | Diana Ro |
| 5   | 5      | Dolly Pa |
| 6   | 6      | Tina Tur |
| 7   | 7      | Madonna  |
| 8   | 8      | Mary J B |
| 9   | 9      | Salt n P |
| 10  | 10     | Aaliyah  |
| 11  | 11     | Beyonce  |

## Reading the whole names

- ▶ Only got 1st 8 characters of each singer's name (SAS default for text).
- ▶ Tell SAS that the names are 20 characters long:

```
SAS> data singers;
SAS>   infile 'singers.csv' dlm=',';
SAS>   input number name $20. ;
SAS>
SAS> proc print;
```

| Obs | number | name            |
|-----|--------|-----------------|
| 1   | 1      | Bessie Smith    |
| 2   | 2      | Peggy Lee       |
| 3   | 3      | Aretha Franklin |
| 4   | 4      | Diana Ross      |
| 5   | 5      | Dolly Parton    |
| 6   | 6      | Tina Turner     |
| 7   | 7      | Madonna         |
| 8   | 8      | Mary J Blige    |
| 9   | 9      | Salt n Pepa     |
| 10  | 10     | Aaliyah         |
| 11  | 11     | Beyonce         |

## Why this worked

- ▶ On input number name \$20.;, the 20. after dollar sign, specifying length of text, called informat.
- ▶ Singer's names have spaces, but this no problem, since delimiter is , .
- ▶ Possible trouble: commas *inside* the names, as in Robert Downey, Jr.
- ▶ Get around this by adding dsd to infile line.
- ▶ singer2.csv has Mr. Downey on the end:

```
SAS> data singers2;
SAS>     infile 'singers2.csv' dlm=',' dsd;
SAS>     input number name $20.;
SAS>
SAS> proc print;
```

## Data file and singers as read in

|                       | Obs | number | name               |
|-----------------------|-----|--------|--------------------|
| 1,Bessie Smith        | 1   | 1      | Bessie Smith       |
| 2,Peggy Lee           | 2   | 2      | Peggy Lee          |
| 3,Aretha Franklin     | 3   | 3      | Aretha Franklin    |
| 4,Diana Ross          | 4   | 4      | Diana Ross         |
| 5,Dolly Parton        | 5   | 5      | Dolly Parton       |
| 6,Tina Turner         | 6   | 6      | Tina Turner        |
| 7,Madonna             | 7   | 7      | Madonna            |
| 8,Mary J Blige        | 8   | 8      | Mary J Blige       |
| 9,Salt n Pepa         | 9   | 9      | Salt n Pepa        |
| 10,Aaliyah            | 10  | 10     | Aaliyah            |
| 11,Beyonce            | 11  | 11     | Beyonce            |
| 12,Robert Downey, Jr. | 12  | 12     | Robert Downey, Jr. |

## A gotcha

- ▶ If you tried this for yourself, this might not have worked.
- ▶ Issue: Singer names must be *at least* 20 characters long.
- ▶ If not, you have to add spaces to make them so.
- ▶ `singers3.csv` has additional spaces removed. With code:

```
SAS> data singers3;  
SAS>     infile 'singers3.csv' dlm=',' dsd;  
SAS>     input number name $20.;  
SAS>  
SAS> proc print;  
you get output shown on next page.
```

## Output from previous commands, data file on right

The \*\*\* marks actual end of lines in data file.

| Obs | number | name               |                       | *** |
|-----|--------|--------------------|-----------------------|-----|
| 1   | 1      | Bessie Smith       | 2,Peggy Lee           | *** |
| 2   | 2      | Peggy Lee          | 3,Aretha Franklin     | *** |
| 3   | 3      | Aretha Franklin    | 4,Diana Ross          | *** |
| 4   | 4      | Diana Ross         | 5,Dolly Parton        | *** |
| 5   | 5      | Dolly Parton       | 6,Tina Turner***      |     |
| 6   | 6      | 7,Madonna          | 7,Madonna             | *** |
| 7   | 8      | Mary J Blige       | 8,Mary J Blige        | *** |
| 8   | 9      | Salt n Pepa        | 9,Salt n Pepa         | *** |
| 9   | 10     | 11,Beyonce         | 10,Aaliyah***         |     |
| 10  | 12     | Robert Downey, Jr. | 11,Beyonce            | *** |
|     |        |                    | 12,Robert Downey, Jr. | *** |

- ▶ What??
- ▶ Names Tina Turner and Aaliyah not 20 chars long with spaces.
- ▶ SAS correctly reads number, but goes to next line to find 20-char name (including number on front of line).

## Windows, Mac and Linux/Unix end-of-line formats

- ▶ These operating systems have different ways of marking end of line.
- ▶ So if eg. you copy a Windows file to cms-chorus, SAS reads file, but gets most of values wrong.
- ▶ Quick fix: *add space to end of every line.*
- ▶ By time it reaches space, SAS has everything from line, so doesn't need to be concerned about end-of-line character.

## Reading spreadsheet data into SAS

- ▶ Two quick ways:
  - ▶ Save data to .csv, transfer to cms-chorus (but line-ending issue)
  - ▶ Copy and paste into Program Editor (quick and dirty). Save in `singsing.dat`, read in like this:

```
SAS> data sing;  
SAS>   infile "singsing.dat" expandtabs;  
SAS>   input singer $20. value;
```

- ▶ Read in actual spreadsheet using `proc import`:

```
SAS> proc import  
SAS>   out=singers  
SAS>   datafile='sing.xls'  
SAS>   dbms=xls  
SAS>   replace;  
SAS>   sheet="sheet1";  
SAS>   getnames=yes;  
SAS>  
SAS> proc print;
```

- ▶ `proc import` one long line, so ; only at end (for clarity)
- ▶ `out=`: name of SAS data set
- ▶ `datafile=`: Excel spreadsheet
- ▶ `sheet=`: which sheet in workbook
- ▶ `getnames=`: use column headers for variable names (`header=T` in R)

## Did it work?

| Obs | singer          | number |
|-----|-----------------|--------|
| 1   | Bessie Smith    | 1      |
| 2   | Peggy Lee       | 2      |
| 3   | Aretha Franklin | 3      |
| 4   | Diana Ross      | 4      |
| 5   | Dolly Parton    | 5      |
| 6   | Tina Turner     | 6      |
| 7   | Madonna         | 7      |
| 8   | Mary J Blige    | 8      |
| 9   | Salt n Pepa     | 9      |
| 10  | Aaliyah         | 10     |
| 11  | Beyonce         | 11     |

Yes!

## Dates

- ▶ Data file (date, another variable):

27may2013 857373

01apr2013 34757

03jan2013 38784

- ▶ Read in, using informat:

*SAS> data mydates;*

*SAS> infile 'dates.dat';*

*SAS> input thedate date9. x;*

*SAS>*

*SAS> proc print;*

*SAS>*

*SAS> proc print;*

*SAS> format thedate date9.;*

## Same data set, two different looks

| Obs | thedate | x      |
|-----|---------|--------|
| 1   | 19505   | 857373 |
| 2   | 19449   | 34757  |
| 3   | 19361   | 38784  |

| Obs | thedate   | x      |
|-----|-----------|--------|
| 1   | 27MAY2013 | 857373 |
| 2   | 01APR2013 | 34757  |
| 3   | 03JAN2013 | 38784  |

- ▶ First: SAS's internal representation of dates (days since Jan 1, 1960)
- ▶ Second: human-readable, using “outformat” (format).

## Other ways of inputting/outputting dates

```
SAS> proc print;  
SAS>   format thedate mmddyy10.;  
SAS>  
SAS> proc print;  
SAS>   format thedate ddmmyy10.;
```

| Obs | thedata    | x      |
|-----|------------|--------|
| 1   | 05/27/2013 | 857373 |
| 2   | 04/01/2013 | 34757  |
| 3   | 01/03/2013 | 38784  |

| Obs | thedata    | x      |
|-----|------------|--------|
| 1   | 27/05/2013 | 857373 |
| 2   | 01/04/2013 | 34757  |
| 3   | 03/01/2013 | 38784  |

These can be used on input line too, if input dates in that format.

## Permanent data sets

- ▶ Can we read in data set *once* and not every time?
- ▶ Yes, use *filename* (in single quotes) when creating:

```
SAS> options ls=70;
```

```
SAS> data 'cars';
```

```
SAS> infile 'cars.txt' firstobs=2;
```

```
SAS> input car $25. mpg weight cylinders hp country $;
```

- ▶ Car names max of 25 chars long. Country names max of 8, so no special treatment needed.
- ▶ SAS stores file called *cars.sas7bdat* (!) on *cms-chorus*.
- ▶ Whenever you need it, add *data='cars'* to a *proc* line.
- ▶ Can use subfolders, using syntax for *machine that SAS running on*. (See notes for what to do when SAS running under Windows.)
- ▶ Closing SAS breaks connection with data sets read in with *data* step. To get those back, need to run *data* step lines again.

# Means, without data step!

```
SAS> proc means data='cars';
SAS>   var mpg weight cylinders hp;
```

The MEANS Procedure

| Variable  | N  | Mean        | Std Dev    | Minimum    | Maximum     |
|-----------|----|-------------|------------|------------|-------------|
| -----     |    |             |            |            |             |
| mpg       | 38 | 24.7605263  | 6.5473138  | 15.5000000 | 37.3000000  |
| weight    | 38 | 2.8628947   | 0.7068704  | 1.9150000  | 4.3600000   |
| cylinders | 38 | 5.3947368   | 1.6030288  | 4.0000000  | 8.0000000   |
| hp        | 38 | 101.7368421 | 26.4449292 | 65.0000000 | 155.0000000 |
| -----     |    |             |            |            |             |

## Mean MPG by country

```
SAS> proc means;
```

```
SAS>   var mpg;
```

```
SAS>   class country;
```

The MEANS Procedure

Analysis Variable : mpg

| country | Obs | N  | Mean       | Std Dev   | Minimum   |
|---------|-----|----|------------|-----------|-----------|
|         |     |    |            |           | -----     |
| France  | 1   | 1  | 16.200000  | .         | 16.200000 |
| Germany | 5   | 5  | 27.140000  | 5.7348060 | 20.300000 |
| Italy   | 1   | 1  | 37.300000  | .         | 37.300000 |
| Japan   | 7   | 7  | 29.600000  | 4.5328431 | 22.000000 |
| Sweden  | 2   | 2  | 19.300000  | 3.2526912 | 17.000000 |
| U.S.    | 22  | 22 | 22.9954545 | 6.0542372 | 15.500000 |

Analysis Variable : mpg

| country | Obs | Maximum   |
|---------|-----|-----------|
|         |     | -----     |
| France  | 1   | 16.200000 |
| Germany | 5   | 31.900000 |

## 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 data set name (created by data step)
  - ▶ quoted data set name (permanent one on disk created as above)
2. Without data=, *most recently created data set*. Typically data set created by data step, though could also be spreadsheet via proc import. 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=.

## Part XVI

Creating new data sets from old ones

## Selecting individuals/observations

- ▶ Singers original data step:

```
SAS> data singers;  
SAS>     infile 'singers.csv' dlm=',';  
SAS>     input number name $20.;
```

- ▶ To select singers only 1 through 6:

```
SAS> data singers;  
SAS>     infile 'singers.csv' dlm=',';  
SAS>     input number name $20.;  
SAS>     if number<=6;
```

- ▶ Select individuals with if: “choose only these”.

## Did it work?

```
SAS> proc print;
```

| Obs | number | name            |
|-----|--------|-----------------|
| 1   | 1      | Bessie Smith    |
| 2   | 2      | Peggy Lee       |
| 3   | 3      | Aretha Franklin |
| 4   | 4      | Diana Ross      |
| 5   | 5      | Dolly Parton    |
| 6   | 6      | Tina Turner     |

Most recently created data set has only singers with numbers 6 or less.

## Omitting individuals

- Sometimes easier to focus on obs to leave out:

```
SAS> data singers;
```

```
SAS>     infile 'singers.csv' dlm=', ';
```

```
SAS>     input number name $20.;
```

```
SAS>     if number>8 then delete;
```

```
SAS>
```

```
SAS> proc print;
```

| Obs | number | name            |
|-----|--------|-----------------|
| 1   | 1      | Bessie Smith    |
| 2   | 2      | Peggy Lee       |
| 3   | 3      | Aretha Franklin |
| 4   | 4      | Diana Ross      |
| 5   | 5      | Dolly Parton    |
| 6   | 6      | Tina Turner     |
| 7   | 7      | Madonna         |
| 8   | 8      | Mary J Blige    |

## Selecting on text variable

- ▶ “Less than” means “earlier alphabetically”.
- ▶ Singers before M:

```
SAS> data singers;  
SAS>     infile 'singers.csv' dlm=', ';  
SAS>     input number name $20.;  
SAS>     if name<'M';  
SAS>  
SAS> proc print;
```

| Obs | number | name            |
|-----|--------|-----------------|
| 1   | 1      | Bessie Smith    |
| 2   | 3      | Aretha Franklin |
| 3   | 4      | Diana Ross      |
| 4   | 5      | Dolly Parton    |
| 5   | 10     | Aaliyah         |
| 6   | 11     | Beyonce         |

## Equality

- ▶ Selecting singer #7
- ▶ ie. singer whose number is equal to 7:

```
SAS> data singers;
SAS>     infile 'singers.csv' dlm=',';
SAS>     input number name $20.;
SAS>     if number=7;
SAS>
SAS> proc print;
Obs      number      name
1          7        Madonna
```

- ▶ Note that SAS uses = while R uses == for logical equals.

And, or

Note to me: write this one.

## Selecting variables

- ▶ if, delete selects/omits *individuals/observations.*
- ▶ To select variables, use keep or drop:

```
SAS> data singers;
```

```
SAS>     infile 'singers.csv' dlm=',';
```

```
SAS>     input number name $20.;
```

```
SAS>     keep name;
```

```
SAS>
```

```
SAS> proc print;
```

| Obs | name |
|-----|------|
|-----|------|

|   |              |
|---|--------------|
| 1 | Bessie Smith |
|---|--------------|

|   |           |
|---|-----------|
| 2 | Peggy Lee |
|---|-----------|

|   |                 |
|---|-----------------|
| 3 | Aretha Franklin |
|---|-----------------|

|   |            |
|---|------------|
| 4 | Diana Ross |
|---|------------|

|   |              |
|---|--------------|
| 5 | Dolly Parton |
|---|--------------|

|   |             |
|---|-------------|
| 6 | Tina Turner |
|---|-------------|

|   |         |
|---|---------|
| 7 | Madonna |
|---|---------|

|   |              |
|---|--------------|
| 8 | Mary J Blige |
|---|--------------|

|    |         |
|----|---------|
| 10 | Aaliyah |
|----|---------|

|    |         |
|----|---------|
| 11 | Beyonce |
|----|---------|

```
SAS> data singers;
```

```
SAS>     infile 'singers.csv' dlm=',';
```

```
SAS>     input number name $20.;
```

```
SAS>
```

```
SAS> proc print;
```

| Obs | name |
|-----|------|
|-----|------|

|   |              |
|---|--------------|
| 1 | Bessie Smith |
|---|--------------|

|   |           |
|---|-----------|
| 2 | Peggy Lee |
|---|-----------|

|   |                 |
|---|-----------------|
| 3 | Aretha Franklin |
|---|-----------------|

|   |            |
|---|------------|
| 4 | Diana Ross |
|---|------------|

|   |              |
|---|--------------|
| 5 | Dolly Parton |
|---|--------------|

|   |             |
|---|-------------|
| 6 | Tina Turner |
|---|-------------|

|   |         |
|---|---------|
| 7 | Madonna |
|---|---------|

|   |              |
|---|--------------|
| 8 | Mary J Blige |
|---|--------------|

|   |             |
|---|-------------|
| 9 | Salt n Pepa |
|---|-------------|

|    |         |
|----|---------|
| 10 | Aaliyah |
|----|---------|

|    |         |
|----|---------|
| 11 | Beyonce |
|----|---------|

## Cloning a data set (pointless!)

- ▶ Use `set` to bring in all the variables and individuals from another data set:

```
SAS> data singers;
SAS>   infile 'singers.csv' dlm=',';
SAS>   input number name $20.;
SAS>
SAS> data singers2;
SAS>   set singers;
```

- ▶ `singers2` exactly same as `singers`.
- ▶ `set` usually first step to doing something else with data.

## A less pointless cloning

- ▶ There is point in combining set with keep or drop or if to copy only individuals/variables you want.
- ▶ Example: cars data, keep only those cars with mpg bigger than 30:

```
SAS> data mycars;  
SAS>   set 'cars';  
SAS>   if mpg>30;  
SAS>  
SAS> proc print;
```

## High-gas-mileage cars

| Obs | car              | mpg  | weight | cylinders | hp | country |
|-----|------------------|------|--------|-----------|----|---------|
| 1   | Dodge Omni       | 30.9 | 2.230  | 4         | 75 | U.S.    |
| 2   | Fiat Strada      | 37.3 | 2.130  | 4         | 69 | Italy   |
| 3   | VW Rabbit        | 31.9 | 1.925  | 4         | 71 | Germany |
| 4   | Plymouth Horizon | 34.2 | 2.200  | 4         | 70 | U.S.    |
| 5   | Mazda GLC        | 34.1 | 1.975  | 4         | 65 | Japan   |
| 6   | VW Dasher        | 30.5 | 2.190  | 4         | 78 | Germany |
| 7   | Dodge Colt       | 35.1 | 1.915  | 4         | 80 | Japan   |
| 8   | VW Scirocco      | 31.5 | 1.990  | 4         | 71 | Germany |
| 9   | Datsun 210       | 31.8 | 2.020  | 4         | 65 | Japan   |
| 10  | Pontiac Phoenix  | 33.5 | 2.556  | 4         | 90 | U.S.    |

Keep only car name and gas mileage

```
SAS> data mycars;  
SAS>   set 'cars';  
SAS>   keep car mpg;  
SAS>  
SAS> proc print;
```

## Just two variables

| Obs | car                | mpg  | 20 | Chrysler LeBaron Wagon    | 18.5 |
|-----|--------------------|------|----|---------------------------|------|
| 1   | Buick Skylark      | 28.4 | 21 | Datsun 510                | 27.2 |
| 2   | Dodge Omni         | 30.9 | 22 | AMC Concord D/L           | 18.1 |
| 3   | Mercury Zephyr     | 20.8 | 23 | Buick Century Special     | 20.6 |
| 4   | Fiat Strada        | 37.3 | 24 | Saab 99 GLE               | 21.6 |
| 5   | Peugeot 694 SL     | 16.2 | 25 | Datsun 210                | 31.8 |
| 6   | VW Rabbit          | 31.9 | 26 | Ford LTD                  | 17.6 |
| 7   | Plymouth Horizon   | 34.2 | 27 | Volvo 240 GL              | 17.0 |
| 8   | Mazda GLC          | 34.1 | 28 | Dodge St Regis            | 18.2 |
| 9   | Buick Estate Wagon | 16.9 | 29 | Toyota Corona             | 27.5 |
| 10  | Audi 5000          | 20.3 | 30 | Chevette                  | 30.0 |
| 11  | Chevy Malibu Wagon | 19.2 | 31 | Ford Mustang Ghia         | 21.9 |
| 12  | Dodge Aspen        | 18.6 | 32 | AMC Spirit                | 27.4 |
| 13  | VW Dasher          | 30.5 | 33 | Ford Country Squire Wagon | 15.5 |
| 14  | Ford Mustang 4     | 26.5 | 34 | BMW 320i                  | 21.5 |
| 15  | Dodge Colt         | 35.1 | 35 | Pontiac Phoenix           | 33.5 |
| 16  | Datsun 810         | 22.0 | 36 | Honda Accord LX           | 29.5 |
| 17  | VW Scirocco        | 31.5 | 37 | Mercury Grand Marquis     | 16.5 |
| 18  | Chevy Citation     | 28.8 | 38 | Chevy Caprice Classic     | 17.0 |
| 19  | Olds Omega         | 26.8 |    |                           |      |

## Get rid of cylinders and hp

```
SAS> data mycars;  
SAS>   set 'cars';  
SAS>   drop cylinders hp;  
SAS>  
SAS> proc print;
```

# Those two variables gone

| Obs | car                       | mpg  | weight | country |
|-----|---------------------------|------|--------|---------|
| 1   | Buick Skylark             | 28.4 | 2.670  | U.S.    |
| 2   | Dodge Omni                | 30.9 | 2.230  | U.S.    |
| 3   | Mercury Zephyr            | 20.8 | 3.070  | U.S.    |
| 4   | Fiat Strada               | 37.3 | 2.130  | Italy   |
| 5   | Peugeot 694 SL            | 16.2 | 3.410  | France  |
| 6   | VW Rabbit                 | 31.9 | 1.925  | Germany |
| 7   | Plymouth Horizon          | 34.2 | 2.200  | U.S.    |
| 8   | Mazda GLC                 | 34.1 | 1.975  | Japan   |
| 9   | Buick Estate Wagon        | 16.9 | 4.360  | U.S.    |
| 10  | Audi 5000                 | 20.3 | 2.830  | Germany |
| 11  | Chevy Malibu Wagon        | 19.2 | 3.605  | U.S.    |
| 12  | Dodge Aspen               | 18.6 | 3.620  | U.S.    |
| 13  | VW Dasher                 | 30.5 | 2.190  | Germany |
| 14  | Ford Mustang 4            | 26.5 | 2.585  | U.S.    |
| 15  | Dodge Colt                | 35.1 | 1.915  | Japan   |
| 16  | Datsun 810                | 22.0 | 2.815  | Japan   |
| 17  | VW Scirocco               | 31.5 | 1.990  | Germany |
| 18  | Chevy Citation            | 28.8 | 2.595  | U.S.    |
| 19  | Olds Omega                | 26.8 | 2.700  | U.S.    |
| 20  | Chrysler LeBaron Wagon    | 18.5 | 3.940  | U.S.    |
| 21  | Datsun 510                | 27.2 | 2.300  | Japan   |
| 22  | AMC Concord D/L           | 18.1 | 3.410  | U.S.    |
| 23  | Buick Century Special     | 20.6 | 3.380  | U.S.    |
| 24  | Saab 99 GLE               | 21.6 | 2.795  | Sweden  |
| 25  | Datsun 210                | 31.8 | 2.020  | Japan   |
| 26  | Ford LTD                  | 17.6 | 3.725  | U.S.    |
| 27  | Volvo 240 GL              | 17.0 | 3.140  | Sweden  |
| 28  | Dodge St Regis            | 18.2 | 3.830  | U.S.    |
| 29  | Toyota Corona             | 27.5 | 2.560  | Japan   |
| 30  | Chevette                  | 30.0 | 2.155  | U.S.    |
| 31  | Ford Mustang Ghia         | 21.9 | 2.910  | U.S.    |
| 32  | AMC Spirit                | 27.4 | 2.670  | U.S.    |
| 33  | Ford Country Squire Wagon | 15.5 | 4.054  | U.S.    |

## Keeping only some individuals and variables

- ▶ Any variables not keep-ed are dropped.
- ▶ Any variables not drop-ed are kept.
- ▶ So only need one of keep and drop.
- ▶ But can combine with if:

```
SAS> data mycars;  
SAS>   set 'cars';  
SAS>   keep car mpg;  
SAS>   if weight>4;  
SAS>  
SAS> proc print;
```

| Obs | car                       | mpg  |
|-----|---------------------------|------|
| 1   | Buick Estate Wagon        | 16.9 |
| 2   | Ford Country Squire Wagon | 15.5 |

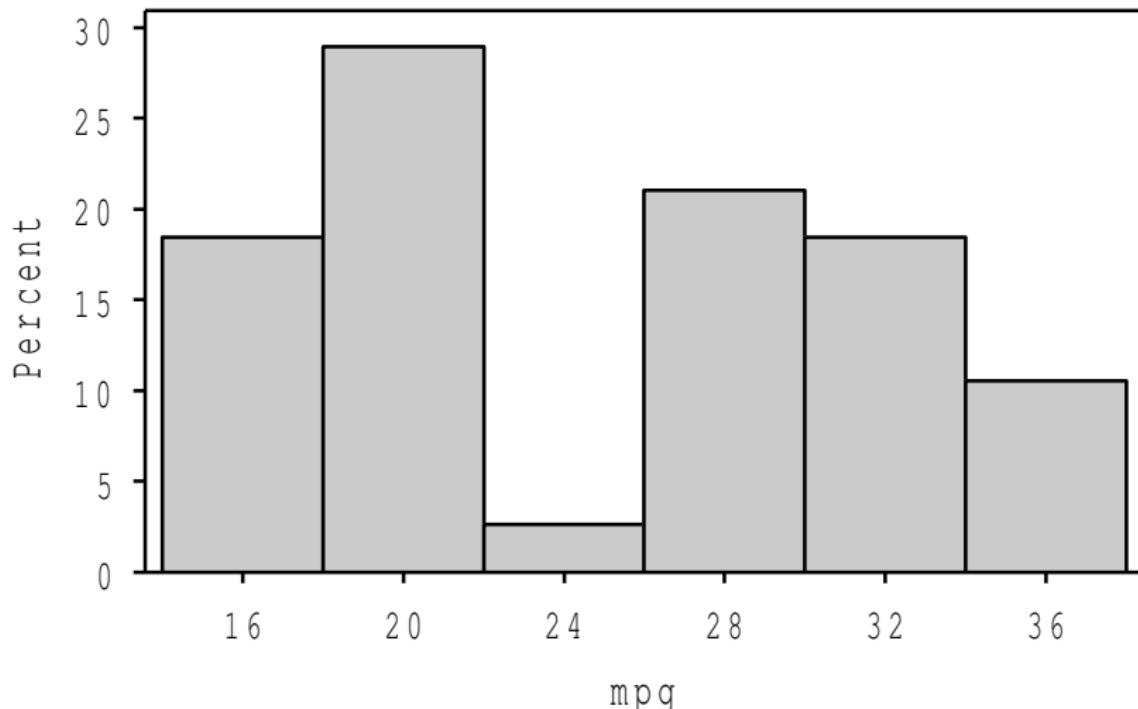
- ▶ Keeps only car name and gas mileage for those cars weighing over 4 tons.

## Part XVII

### Plots in SAS

## Histogram (reminder)

```
SAS> proc univariate data='cars' nopolish;  
SAS>   var mpg;  
SAS>   histogram;
```



## Changing bar endpoints

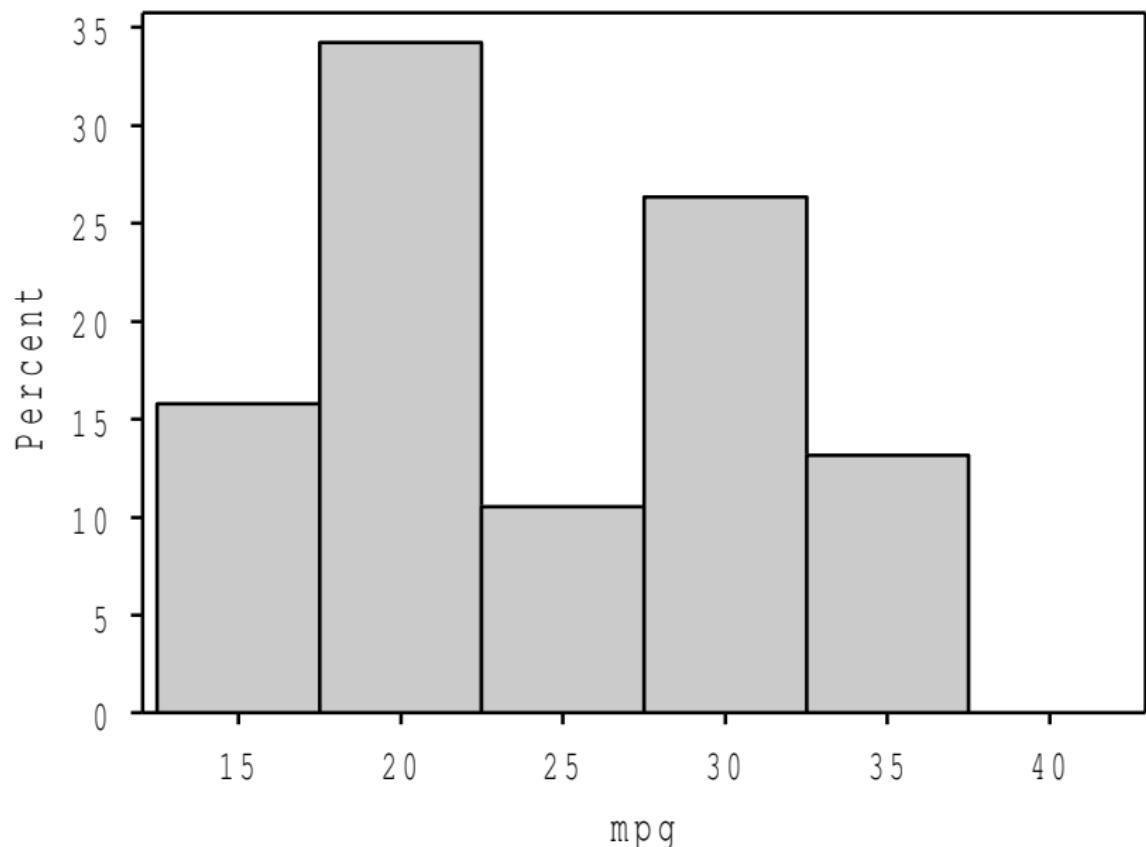
- ▶ Note “gap” in middle (bimodal distribution).
- ▶ Bar midpoints at 16, 20, 24, . . . , so bar *endpoints* at 14, 18, 22, 26,  
. . .
- ▶ Change midpoints like this:

```
SAS> proc univariate data='cars' nointeractive;  
SAS>   var mpg;  
SAS>   histogram / midpoints=15 20 25 30 35 40;
```

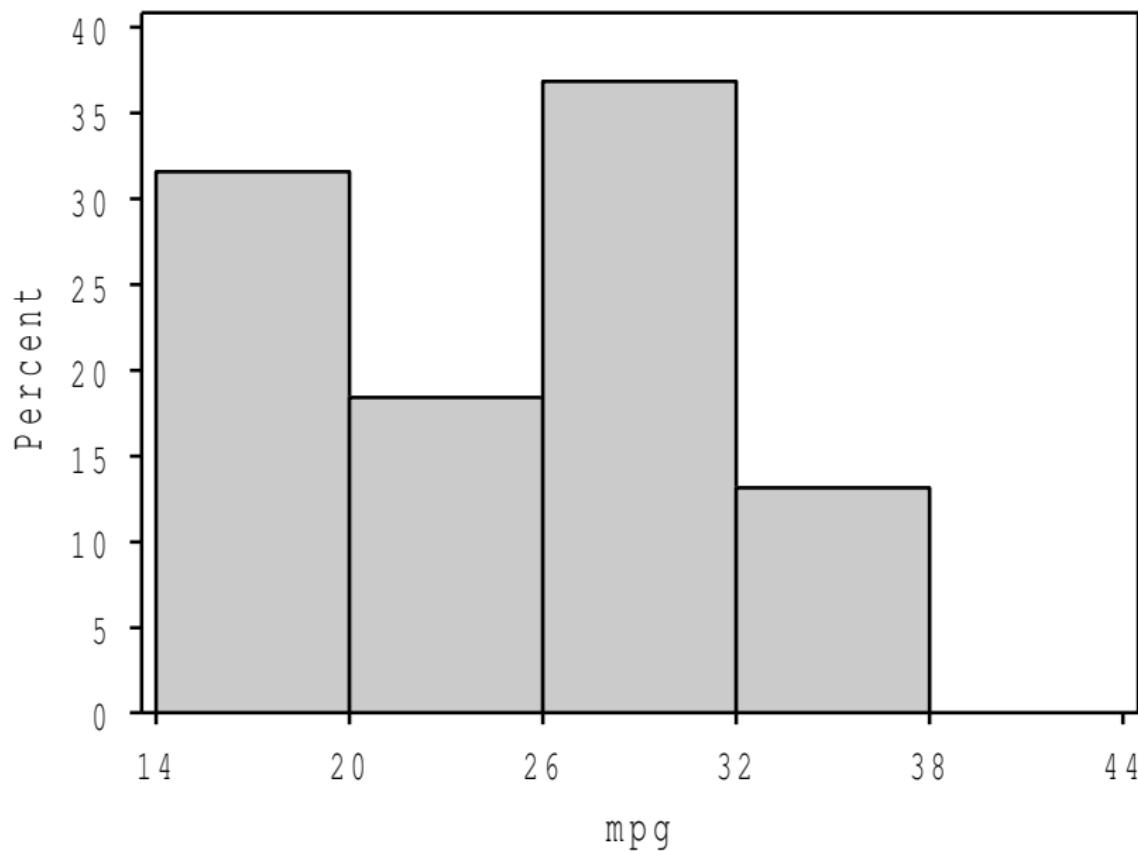
- ▶ Change endpoints like this:

```
SAS> proc univariate data='cars' nointeractive;  
SAS>   var mpg;  
SAS>   histogram / endpoints=14 20 26 32 38 44;
```

Midpoints 15, 20, 25, ...



Endpoints 14, 20, 26,...



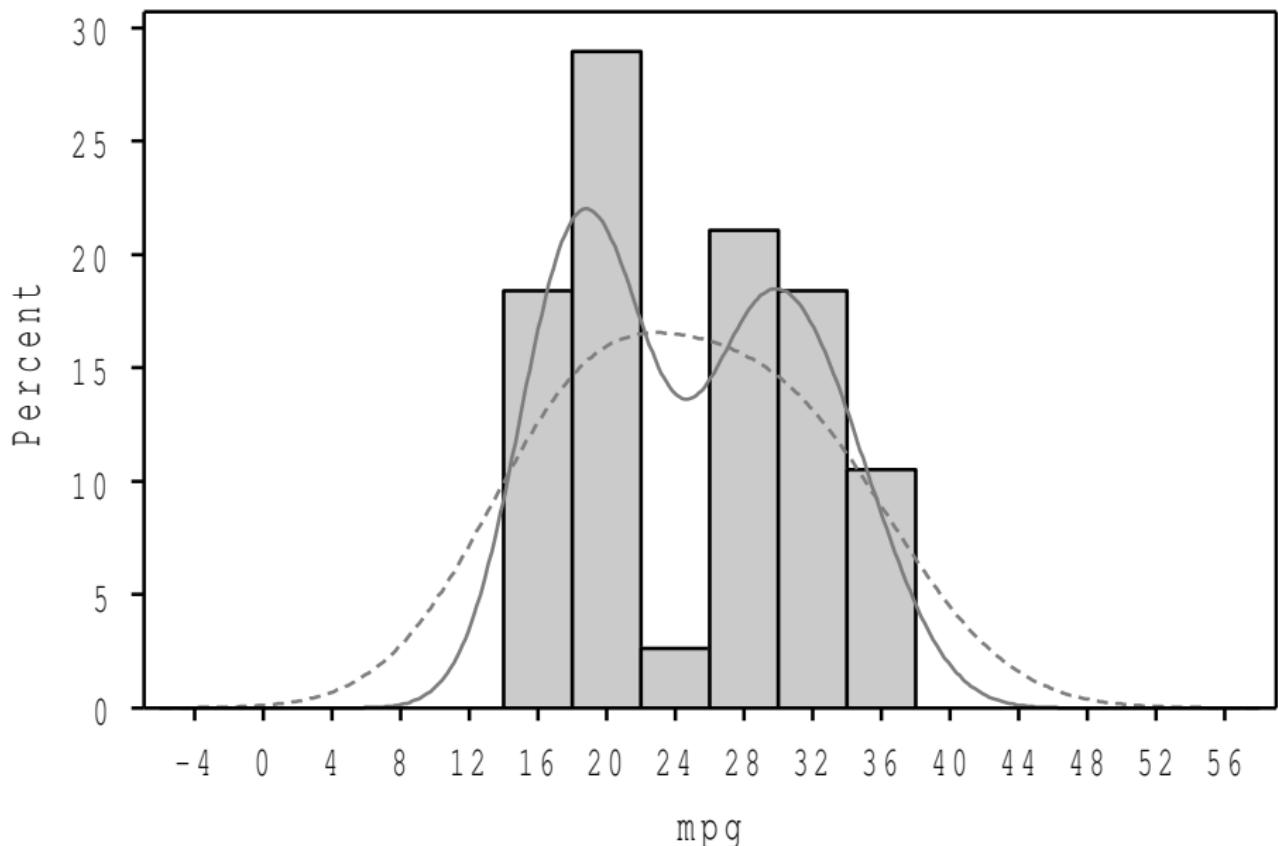
## Kernel density curve

- ▶ A kernel density curve smooths out a histogram and gives sense of shape of distribution.
- ▶ SAS's default is  $c=1$ . Add  $c=0.5$  for comparison.
- ▶ Car mpgs:

```
SAS> proc univariate data='cars' nointer;  
SAS>   var mpg;  
SAS>   histogram / kernel(c=0.5 1 l=1 2);
```

- ▶  $l=1\ 2$  does two line types: solid and dashed.

## Histogram of MPGs with kernel density

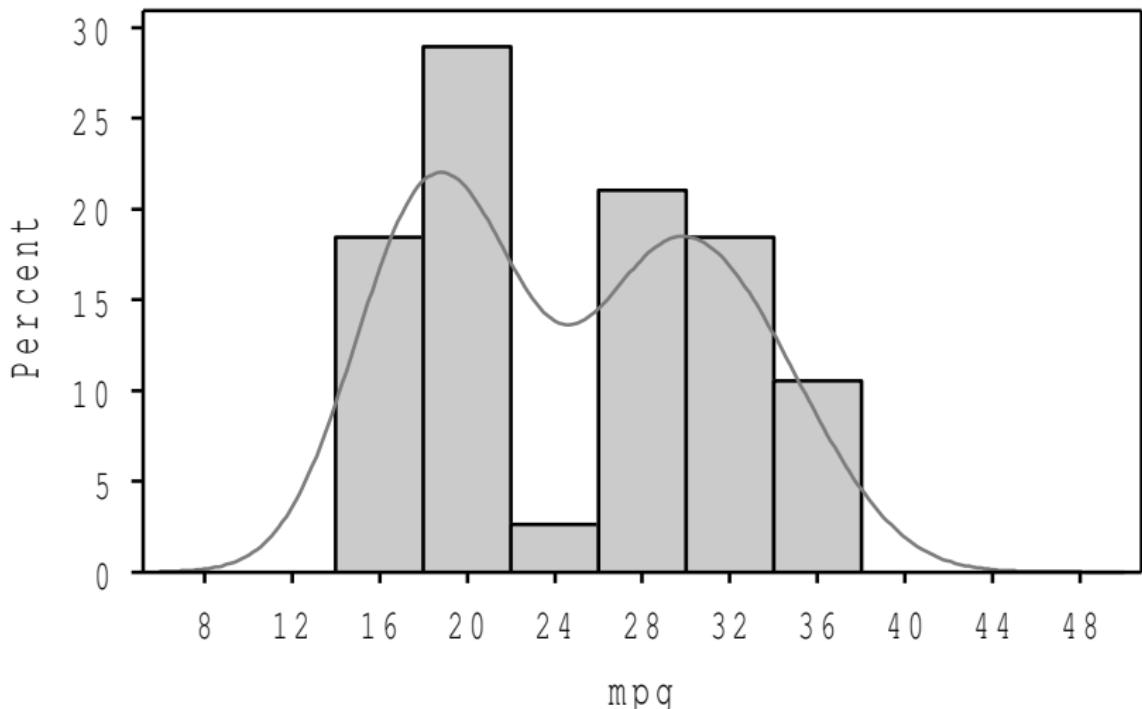


## Comments

- ▶  $c=1$  (dashed) seems *too* smooth, and misses “hole” in middle.
- ▶  $c=0.5$  still “smooth” (not jumpy), but conforms to data better.
- ▶ Recommend  $c=0.5$ ; gives results like R’s default.
- ▶  $c=0.5$  kernel density shows bimodalness of data.

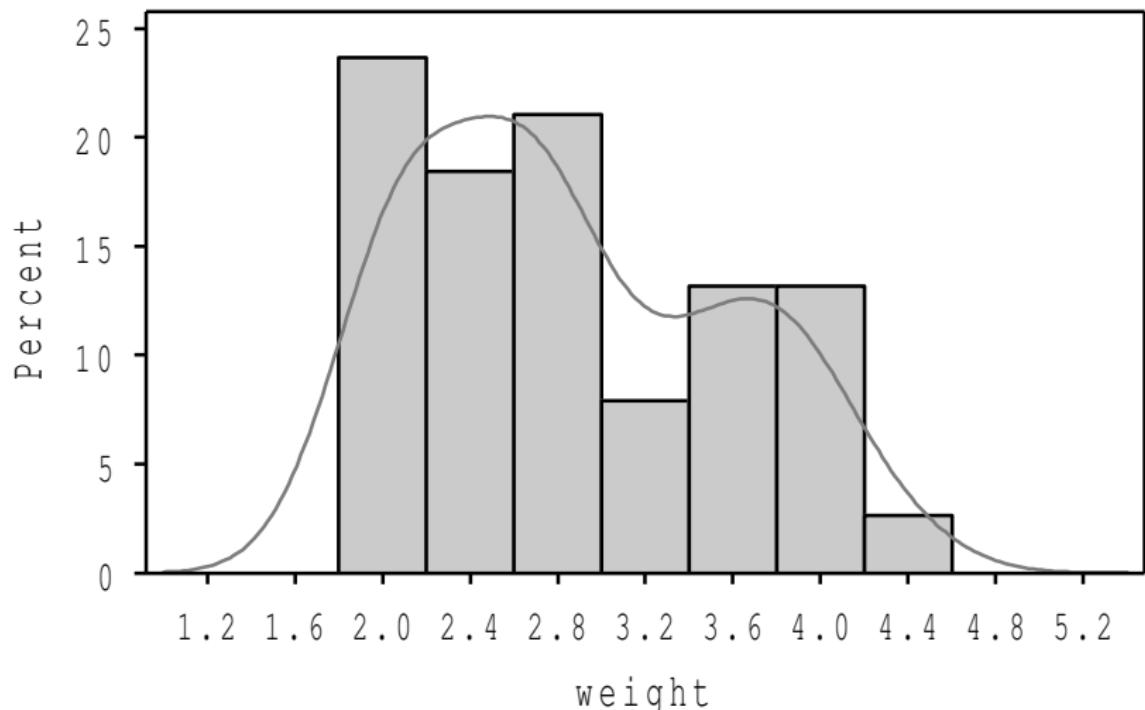
## Histogram of MPG with kernel density

```
SAS> proc univariate data='cars' noprint;  
SAS>   var mpg;  
SAS>   histogram / kernel(c=0.5);
```



## Kernel density for car weights

```
SAS> proc univariate data='cars' nopolish;  
SAS>   var weight;  
SAS>   histogram / kernel(c=0.5);
```



## Comments

- ▶ For MPG<sub>s</sub>, clear evidence of bimodal shape. Cars seem to divide into low-MPG and high-MPG groups.
- ▶ For weights, not so much evidence of bimodality. Looks more right-skewed.

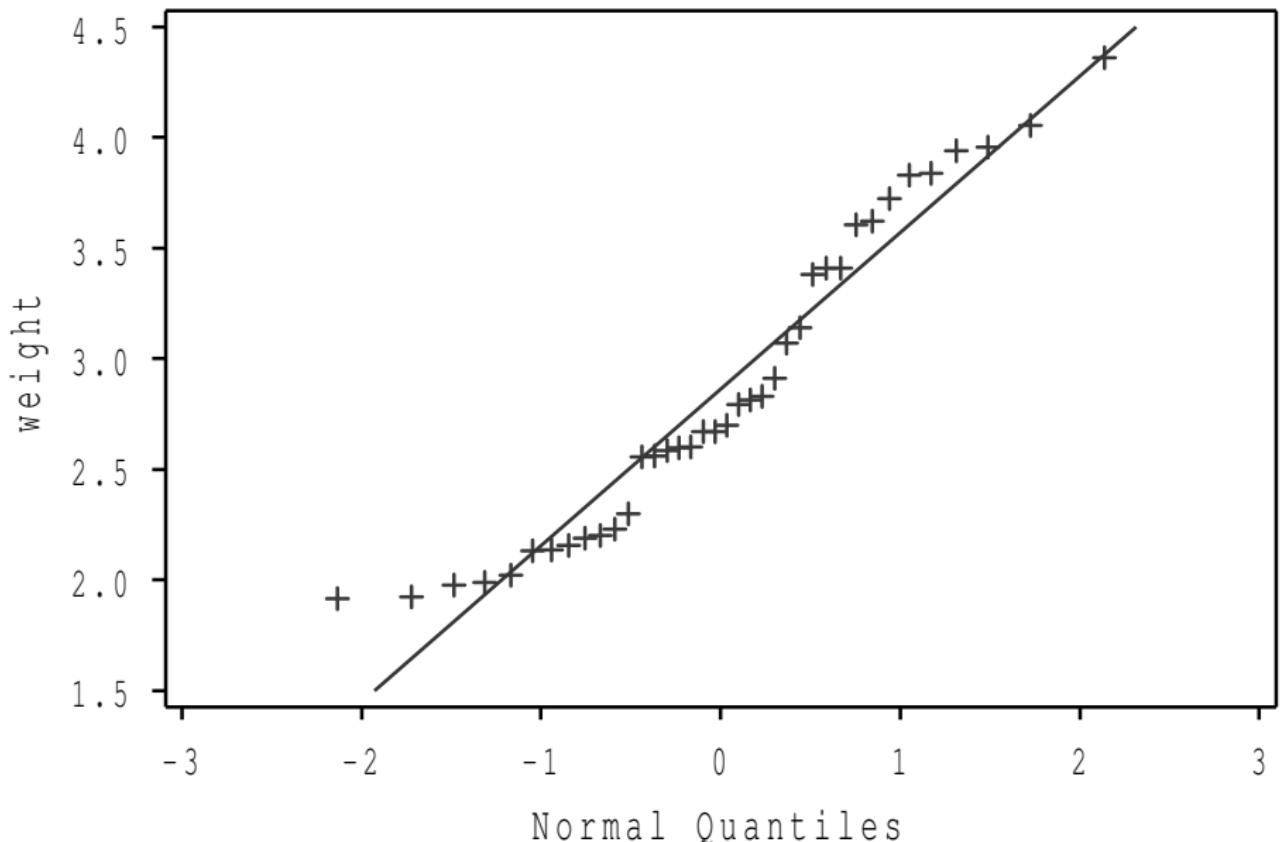
## Assessing normality

- ▶ Saw before that *normal quantile plot* was how to do this. For car weights:

```
SAS> proc univariate data='cars' nopolish;  
SAS>   var weight;  
SAS>   qqplot / normal(mu=est sigma=est);
```

- ▶ Bit after / says to plot the line that would be correct for normal distribution (like R's `qqline`) with mean and SD estimated from data.
- ▶ If normal dist no good, using sample mean and (especially) SD might be misleading.
- ▶ Major failure of normality is that values at *lower* end too bunched up: weights should go down below 2.0 tons.
- ▶ Also suggestion of S-bend shape, not straight line.

## The normal quantile plot

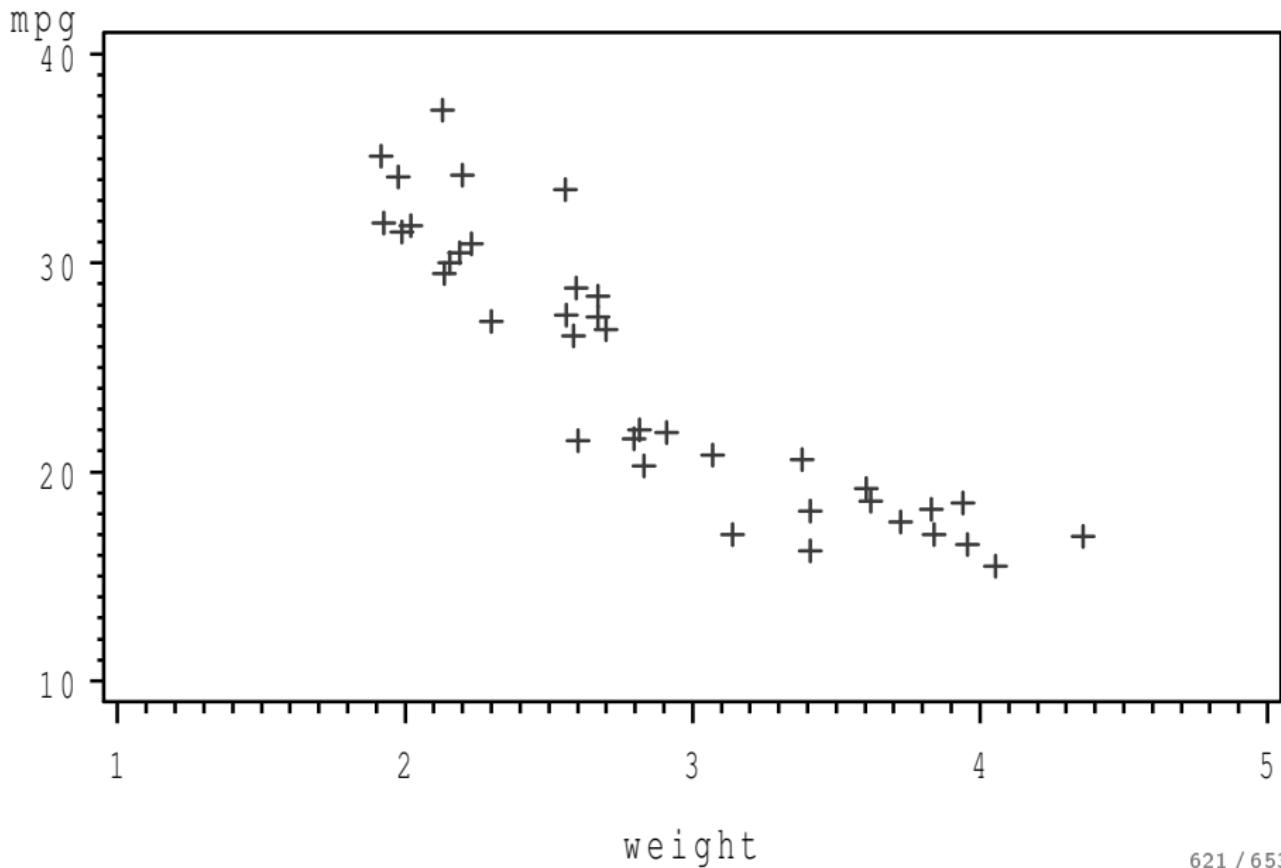


## Scatterplots

- ▶ How do two *quantitative* variables relate to each other?
- ▶ Plot their values against each other: *scatterplot*.
- ▶ proc gplot in SAS. Basic scatterplot, with *y*-variable *first*:

```
SAS> proc gplot data='cars';  
SAS>   plot mpg*weight;
```

## The scatterplot

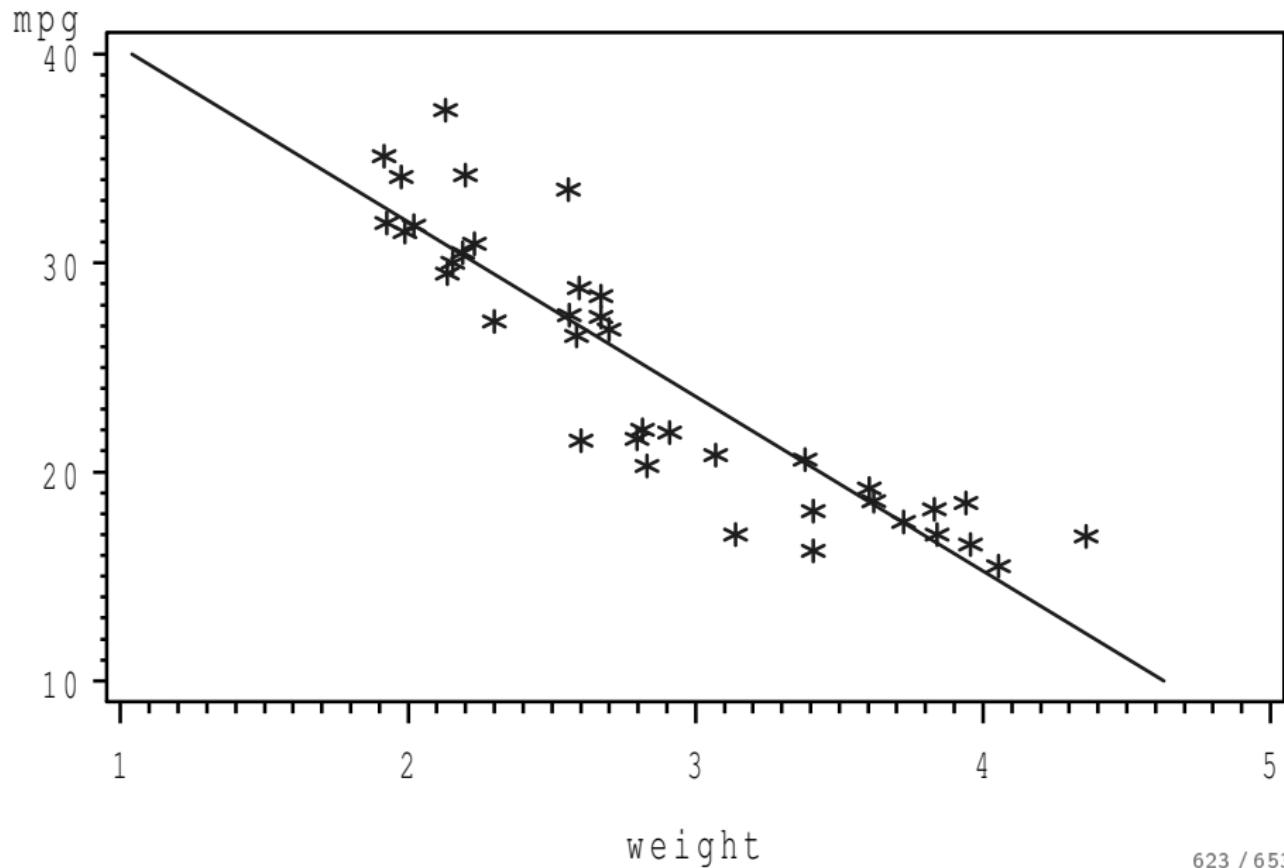


## Comments

- ▶ Clear downward trend: heavier car has worse MPG on average.
- ▶ Physics: a heavier car takes more fuel to start moving and keep moving (friction).
- ▶ Is it linear? Need to assess curvature.
- ▶ Start by adding regression line to plot. Key is `symbol`. Awkward, but follow steps (later). This plots everything blue (but these notes greyscale), with a regression line (`i=r1`) in solid (`l=1`), with the data points as stars:

```
SAS> symbol1 c=blue i=r1 l=1 v=star;  
SAS> proc gplot data='cars';  
SAS>   plot mpg*weight;
```

## Scatterplot with regression line



## Comments

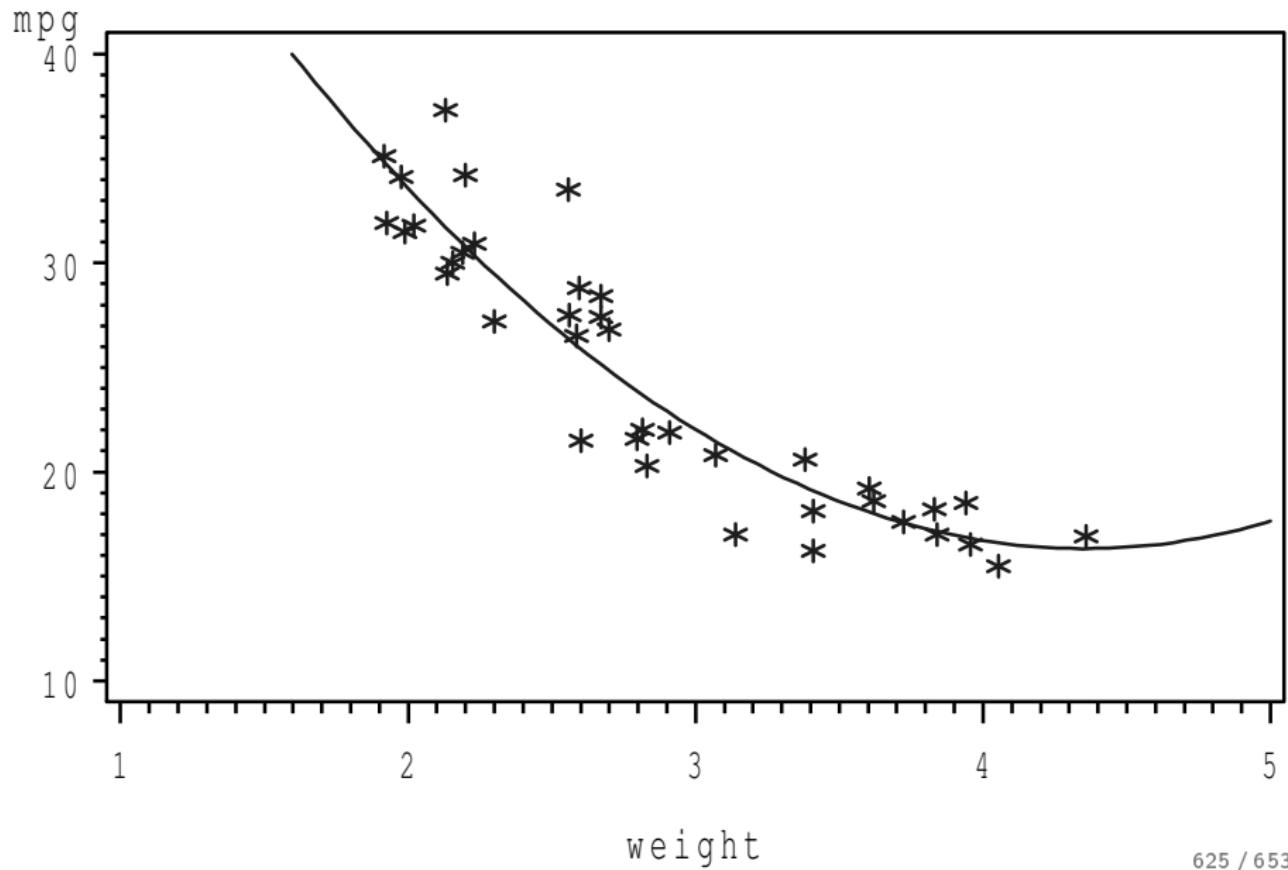
- ▶ A lot of points above the line at the left, below line in middle, above line at end.
- ▶ Suggests a curve may fit better than straight line.
- ▶ Here are things on symbol:
  - ▶ c= Colour. Use a colour name like red or green or black after the equals. No quotes needed.
  - ▶ i= Draw lines. Commonest one is r1 for regression line. There is also rq for a quadratic (parabola) regression.
  - ▶ l= Line type. 1 is solid, 2 is dots, and so on.
  - ▶ v= Plotting symbol. By name, for example square, star, circle, plus, dot.
- ▶ Plot a quadratic (parabola) on scatter plot like this:

```
SAS> symbol1 c=blue i=rq l=1 v=star;
```

```
SAS> proc gplot data='cars';
```

```
SAS>   plot mpg*weight;
```

## Scatterplot with parabola



## Comments

- ▶ I think a parabola fits better.
- ▶ Can put line and curve both on plot: see full notes.
- ▶ Worrying: at end, parabola starts going *up*, despite physics.
- ▶ Better: something like lowess curve (as R).

## Loess curve

- ▶ Loess curve (note spelling) in SAS: Code like this:

```
SAS> proc loess data='cars';  
SAS>   model mpg=weight;
```

- ▶ I can't get it to work here (!), but works just fine from Program Editor,
- ▶ Gives (lots of) output (some numerical, but lots of graphs).
- ▶ One called *fit plot* is the scatterplot with smooth trend.
- ▶ In this case, trend decreases, but at a decreasing rate as weight increases.
- ▶ You can get only the fit plot with

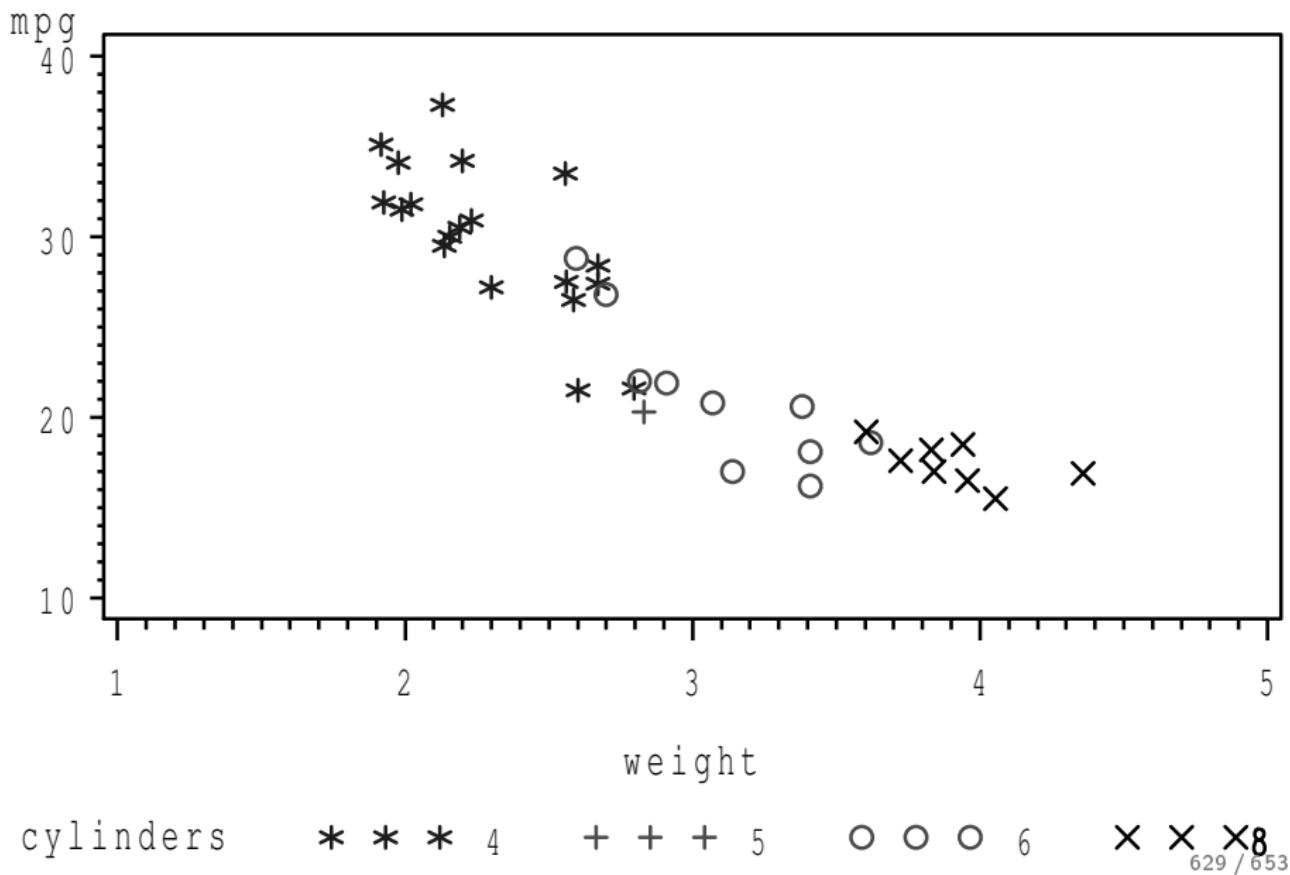
```
SAS> proc loess data='cars' plots(only)=fit;  
SAS>   model mpg=weight;
```

## Distinguishing points by colours or symbols

- ▶ This has to do with symbol again.
- ▶ Say we want to plot mpg by weight, with the points different colours and symbols according to what number of cylinders they are.
- ▶ 4 different numbers of cylinders, so define symbol1 through symbol4.
- ▶ Don't want i= (lines/curves) or l= (line types), so leave blank:

```
SAS> symbol1 c=blue i= v=star l=;
SAS> symbol2 c=red i= v=plus l=;
SAS> symbol3 c=green i= v=circle l=;
SAS> symbol4 c=black i= v=x l=;
SAS>
SAS> proc gplot data='cars';
SAS>   plot mpg*weight=cylinders;
```

## The plot



## Comments

- ▶ Stars denote 4-cylinder cars . . . crosses 8-cylinder. (Normally different colours, but this in greyscale).
- ▶ Legend at the bottom, so you can see which colour/symbol is which.
- ▶ Cars with more cylinders are heavier and have worse gas mileage.
- ▶ Symbol definition “sticky”: continues until you reset it with  
*SAS> options reset=all;*

or until you define new symbols.

## Multiple series on one plot: the oranges data

- ▶ Data file like this (circumferences of 5 trees each at 7 times):

| row | ages | A   | B   | C   | D   | 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 |

- ▶ Skip over first line of file; create permanent data set:

```
SAS> data 'oranges';
SAS>   infile "oranges.txt" firstobs=2;
SAS>   input row age a b c d e;
```

## Multiple series

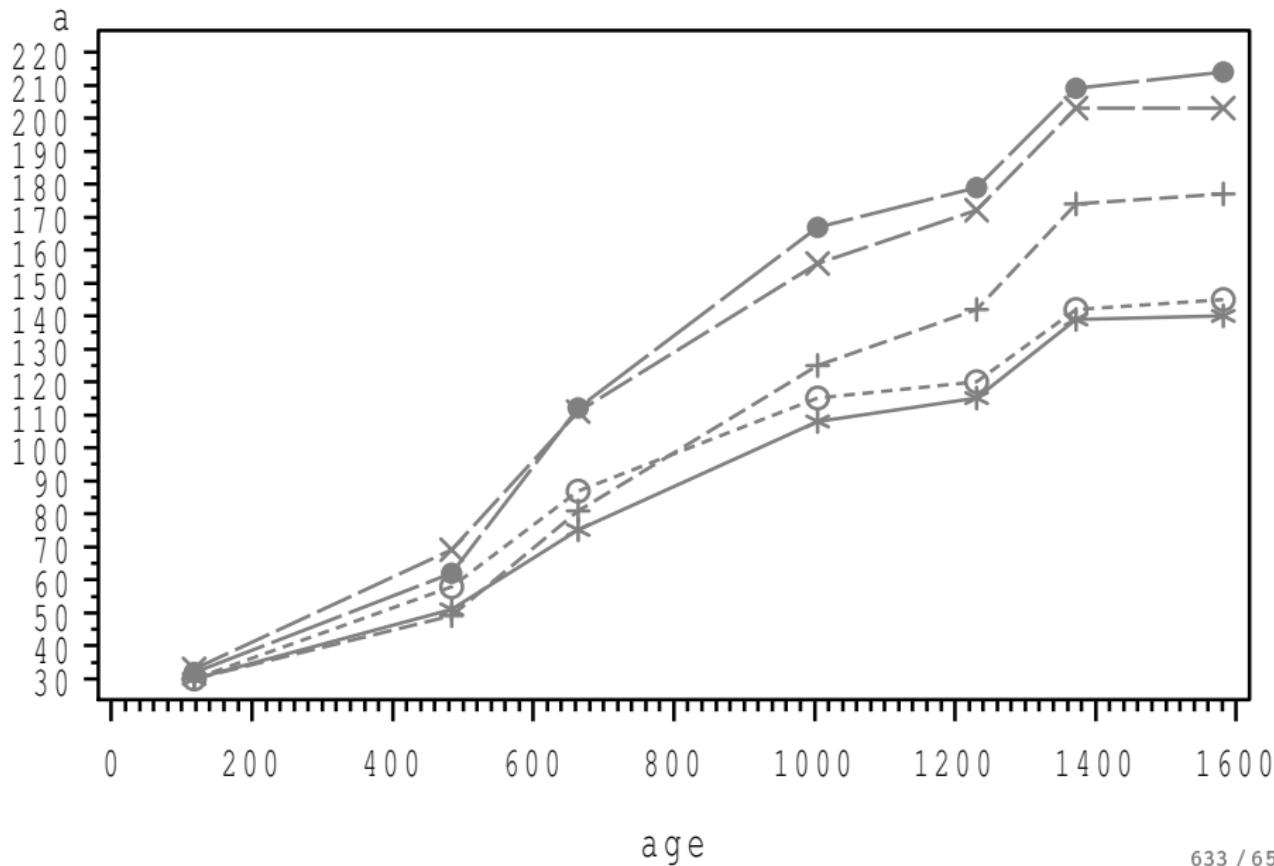
- ▶ Growth curve for each tree, joined by lines.
- ▶ Put one after another on plot line.
- ▶ Want 5 series to be different, so define 5 symbols first. Not using colours here (you can), just plot symbols.

```
SAS> symbol1 c= v=star i=join l=1;  
SAS> symbol2 c= v=circle i=join l=2;  
SAS> symbol3 c= v=plus i=join l=3;  
SAS> symbol4 c= v=x i=join l=4;  
SAS> symbol5 c= v=dot i=join l=5;  
SAS>
```

```
SAS> proc gplot;  
SAS>   plot a*age b*age c*age d*age e*age / overlay;
```

- ▶ `i=join` joins each observation to the next one (as opposed to `i=r1` which puts regression line through all at once).

## The growth curves



## Comments

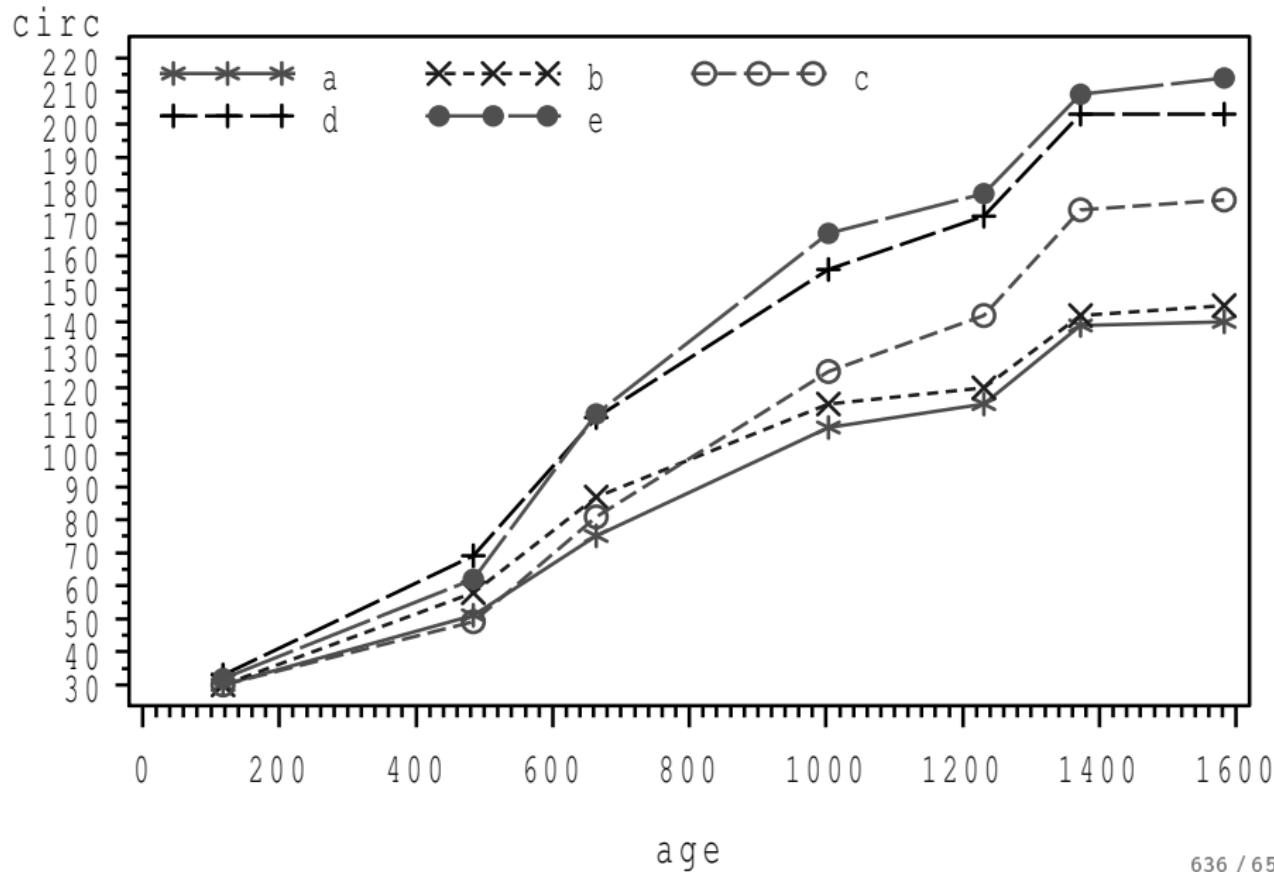
- ▶ overlay puts all plots on one set of axes.
- ▶ Plotting symbols (`v=`) and line types (`l=`) distinguish the trees. You can use colour as well (`c=`).
- ▶ 500 days is pretty good indicator of how big trees will get, and 700 days better. (Not much “crossing over” of growth curves.)
- ▶ Plot needs:
  - ▶ proper label for `y` axis
  - ▶ legend to identify trees
- ▶ Axis label: `axis2`
- ▶ Legend via `legend1`. Three things: any special text to label legend (“none” best), where to put it, plus `line mode=share` (don’t ask me).
- ▶ Code on next page.

## All the code

symbol stuff not necessary, because symbols are sticky. But here's the whole thing:

```
SAS> symbol1 c=red v=star i=j l=1;
SAS> symbol2 c=blue v=x i=j l=2;
SAS> symbol3 c=green v=circle i=j l=3;
SAS> symbol4 c=black v=plus i=j l=4;
SAS> symbol5 c=brown v=dot i=j l=5;
SAS>
SAS> axis2 label=('circ');
SAS> legend1 label=none
SAS>   position=(top left inside)
SAS>   mode=share;
SAS>
SAS> proc gplot;
SAS>   plot a*age b*age c*age d*age e*age /
SAS>     overlay legend=legend1 vaxis=axis2;
```

## Revised growth curves



## Comments

- ▶ Looks good.
- ▶ Howto for this stuff:  
<http://www2.sas.com/proceedings/sugi31/239-31.pdf>
- ▶ See also first part of Chapter 14 of SAS book.
- ▶ Undo symbol definitions by  
*SAS> goptions reset=all;*  
or by re-defining symbol1 etc., eg.:  
*SAS> symbol1 c=black v=plus i= l=;*

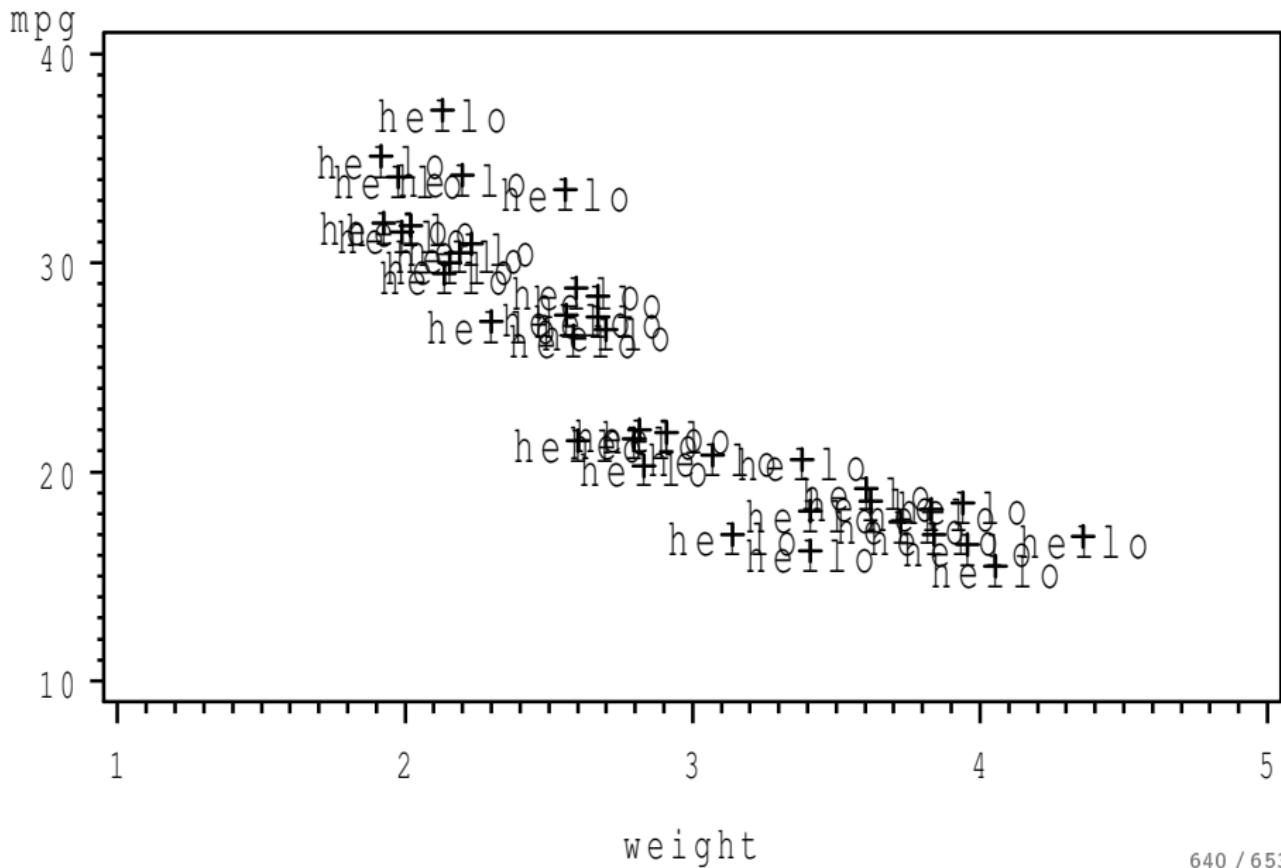
## Labelling points on a plot

- ▶ Section 16 of above how-to shows example of labelling points on a plot.
- ▶ Process: define new data set containing text to be plotted and locations for plotting (like `text` in R). Plus some other stuff. Which appears to be necessary.
- ▶ Cars: plot `mpg` against `weight`, label each point with word “Hello” (yes, silly).
- ▶ Define new data set containing original one plus some new stuff. Then plot, using `annotate(over)`.

## The code

```
SAS> data mytext1;  
SAS>   retain xsy ysys '2';  
SAS>   set 'cars';  
SAS>   x=weight;  
SAS>   y=mpg;  
SAS>   text='hello';  
SAS>  
SAS> proc gplot data='cars';  
SAS>   plot mpg*weight / annotate=mytext1;
```

## Labelled points



# Confusing?

Most certainly!

How-to ([sas.com](http://sas.com)) says:

*If you are really new to SAS/GRAFPH (or SAS in general), the following may seem a little obtuse. Do not worry, it's not just you, ANNOTATE is obtuse.*

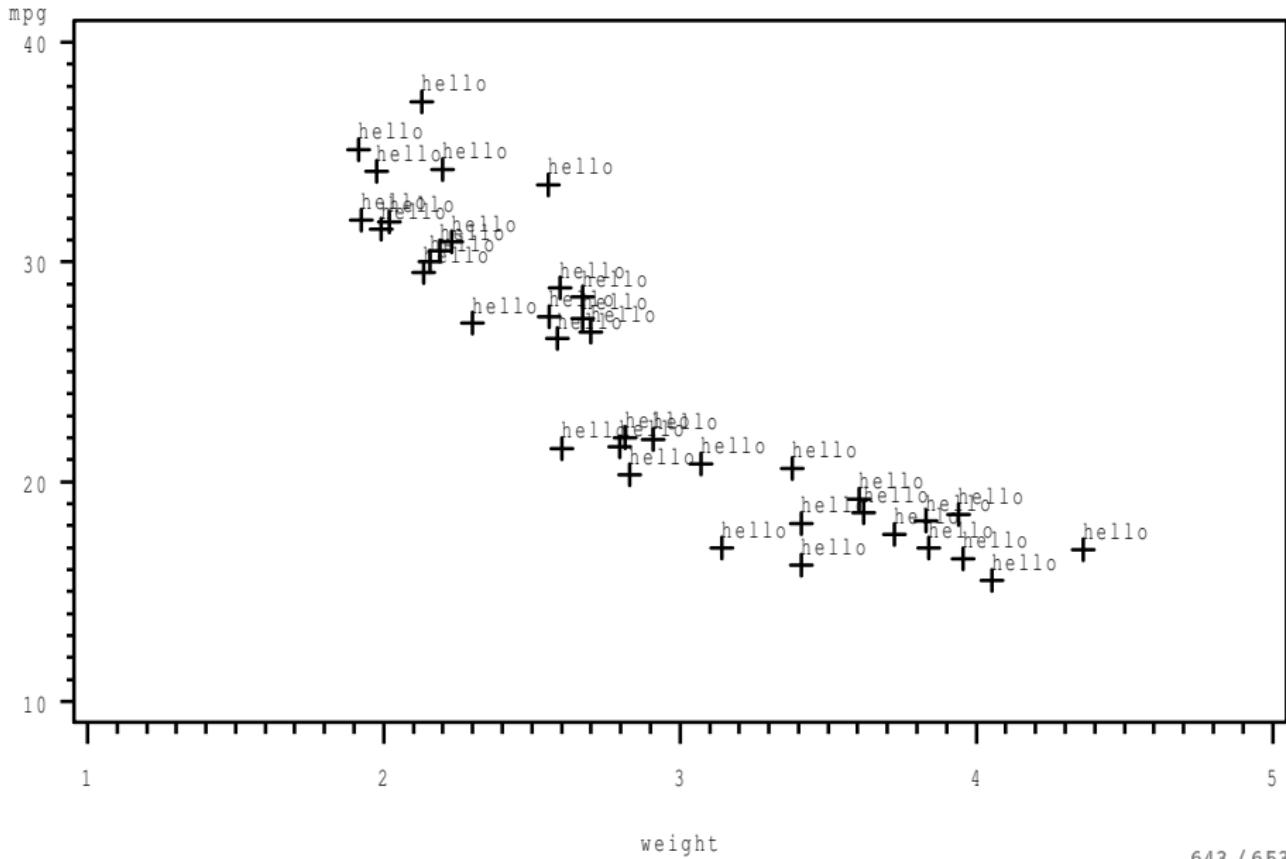
This is at least honest!

## Putting text next to the points

- ▶ Text labels *overwrite* the +'s. Also, too big.
- ▶ Would like something like R's pos to offset them (eg. to right).
- ▶ In SAS, this is position on retain line. (Why? Who knows.)  
position '3' puts text above and to right:
- ▶ htext resizes text (including axis labels). Like cex.

```
SAS> goptions htext=0.5;
SAS> data mytext2;
SAS>   retain xsyss ysys '2' position '3';
SAS>   set 'cars';
SAS>   x=weight;
SAS>   y=mpg;
SAS>   text='hello';
SAS>
SAS> proc gplot data='cars';
SAS>   plot mpg*weight / annotate=mytext2;
```

## With text top right

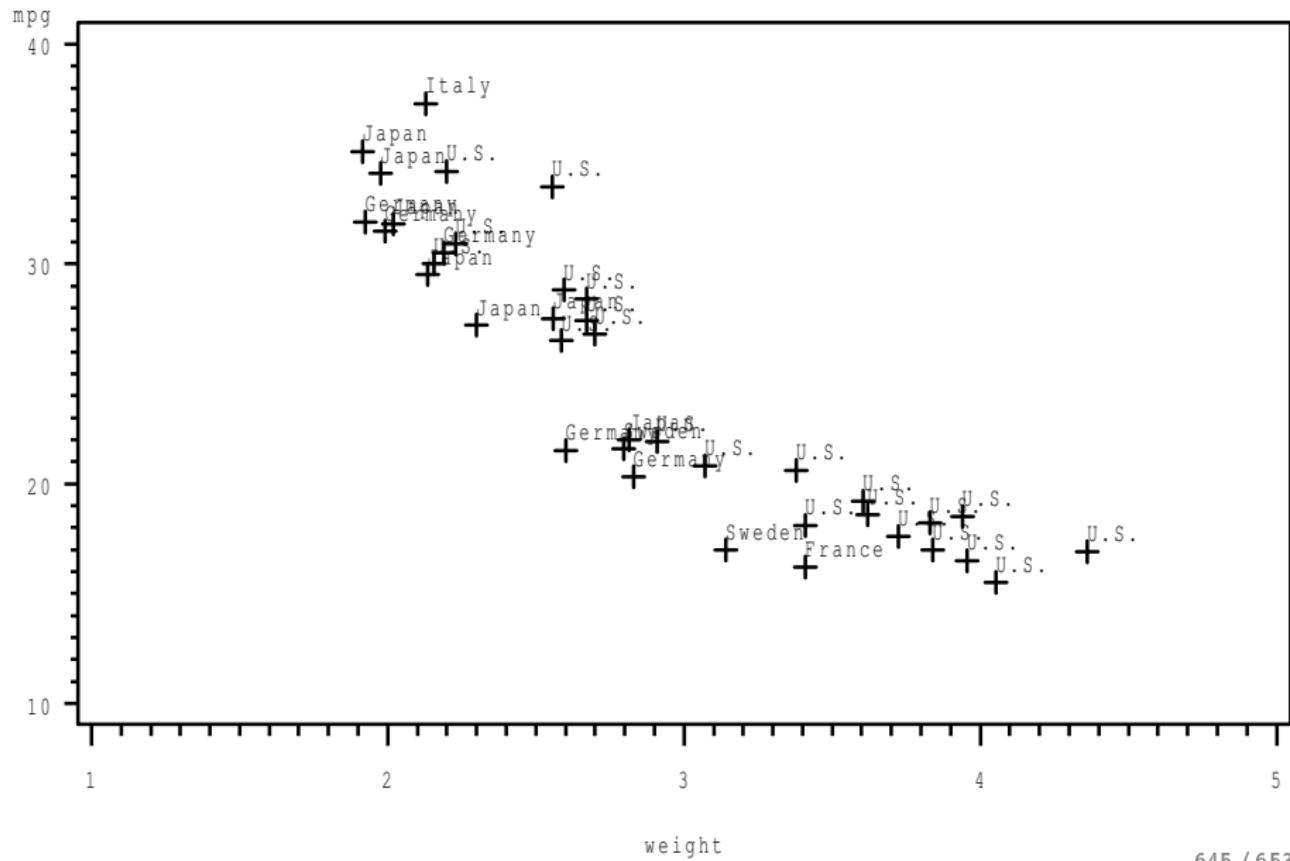


## Next steps

- ▶ Rather silly to label everything with “hello”.
- ▶ Usually want to name with value of variable, like country.
- ▶ Change text line in creation of new data set to name of (text) variable like country.
- ▶ htext curiously *not* sticky, so have to give it again.

```
SAS> goptions htext=0.5;
SAS> data mytext3;
SAS>   retain xsy s ysys '2' position '3';
SAS>   set 'cars';
SAS>   x=weight;
SAS>   y=mpg;
SAS>   text=country;
SAS>
SAS> proc gplot data='cars';
SAS>   plot mpg*weight / annotate=mytext3;
```

## Labelled by country



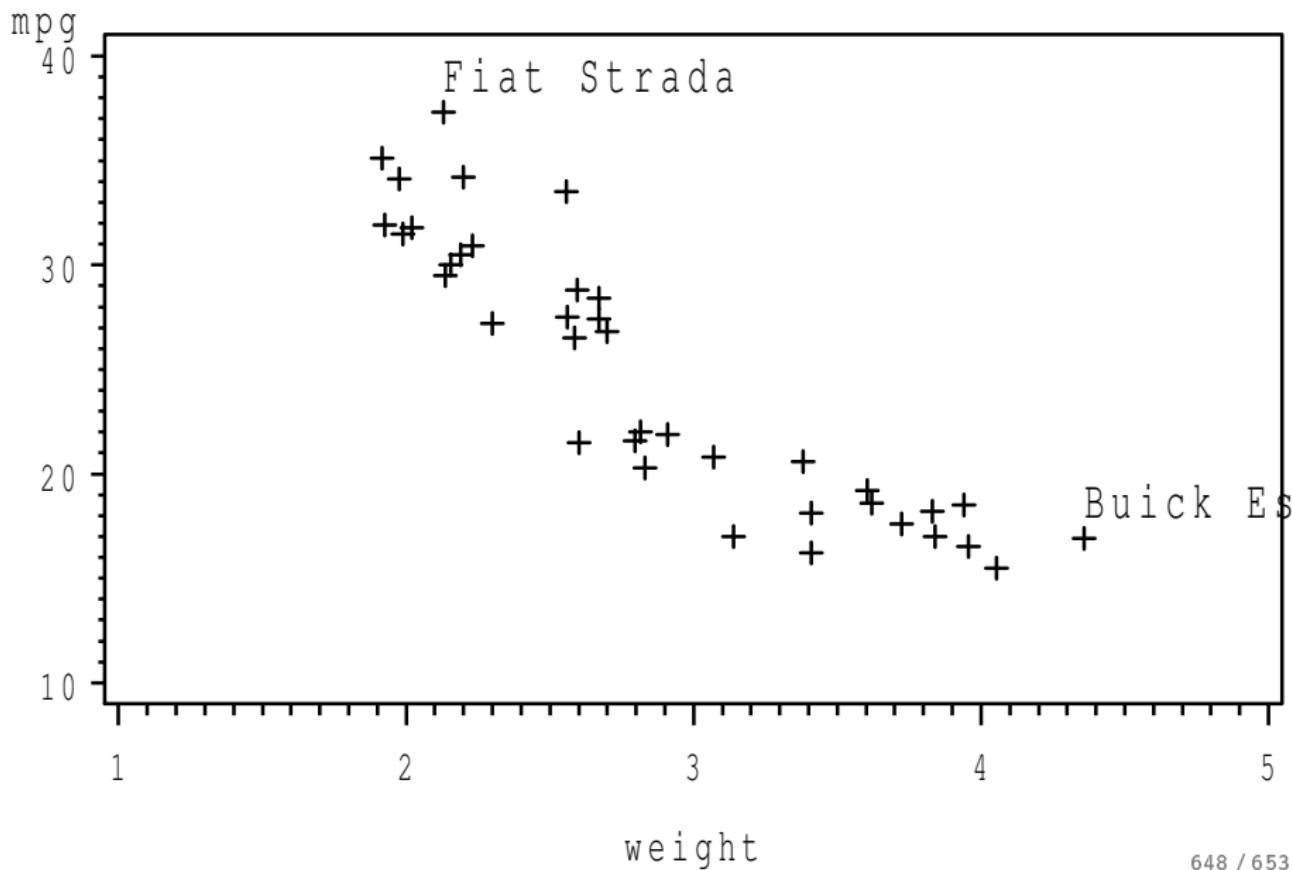
## Labelling only some of the observations

- ▶ When we define our new `mytext` data set, pick out just those cars we want.
- ▶ For example, label most fuel-efficient car (#4) and heaviest car (#9).
- ▶ “Observation number” given by SAS special variable `_n_`.
- ▶ Use `|` for “or”.
- ▶ Use `if`. Now have to do several things when `if` is true:
  - ▶ `if (condition) then do;` means “do all of these things until I tell you to stop”.
  - ▶ At end, put `end;;`.
  - ▶ Can optionally have an `else;;`, some statements, and another `end;;`.
- ▶ This time text is name of car (full size).

## The code

```
SAS> data mytext4;
SAS>   retain xsy ysys '2' position '3';
SAS>   set 'cars';
SAS>   if (_n_ eq 4 | _n_ eq 9) then do;
SAS>     x=weight;
SAS>     y=mpg;
SAS>     text=car;
SAS>   end;
SAS>
SAS> proc gplot data='cars';
SAS>   plot mpg*weight / annotate=mytext4;
```

## Heaviest and most fuel-efficient car, by name



## Extending the axis

- ▶ Buick Estate Wagon doesn't fit on the plot.
- ▶ Extend x-axis to fit it.
- ▶ Make axis go from 1 to 7 in steps of 0.5.
- ▶ Use mytext4 that we defined before for labelling.

```
SAS> axis1 order=(1 to 7 by 0.5);
```

```
SAS>
```

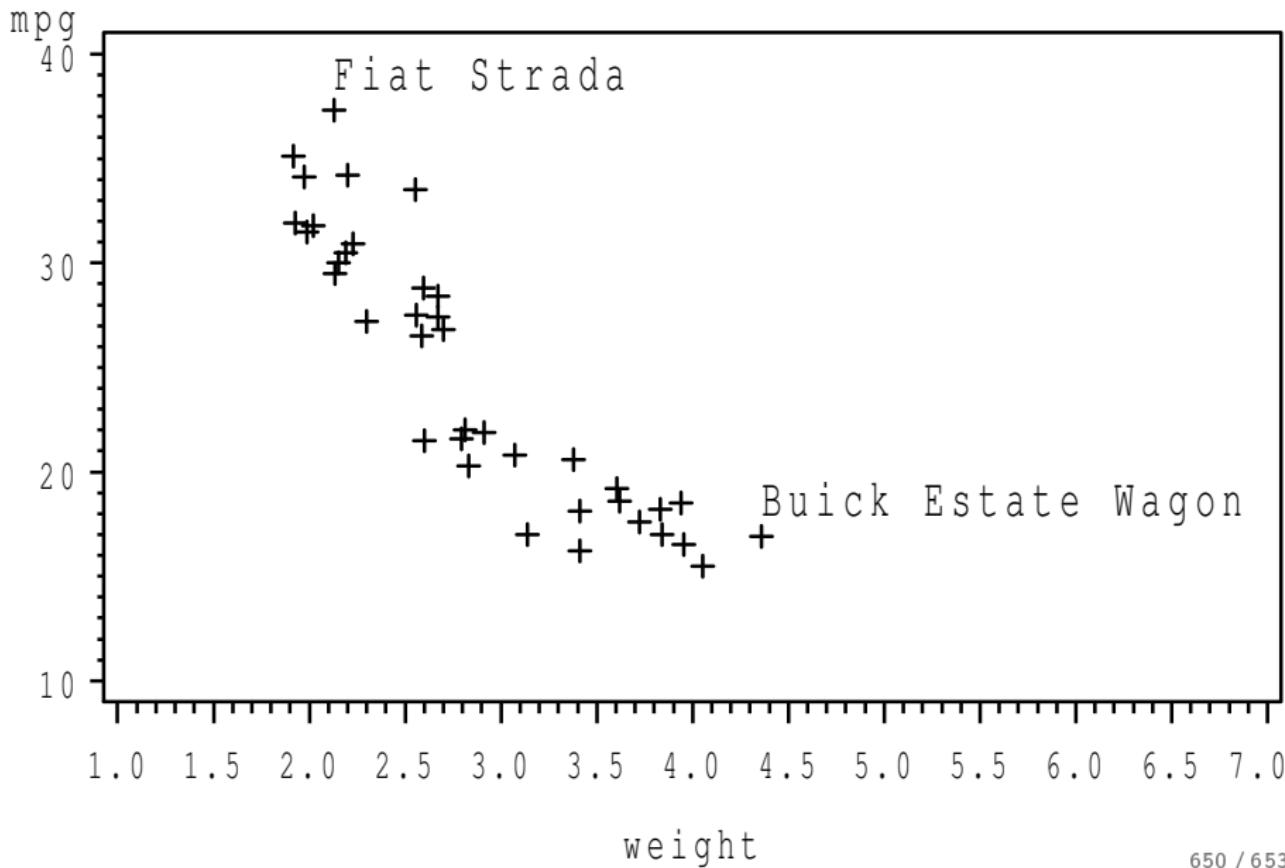
```
SAS> proc gplot data='cars';
```

```
SAS>   plot mpg*weight /
```

```
SAS>     annotate=mytext4
```

```
SAS>     haxis=axis1;
```

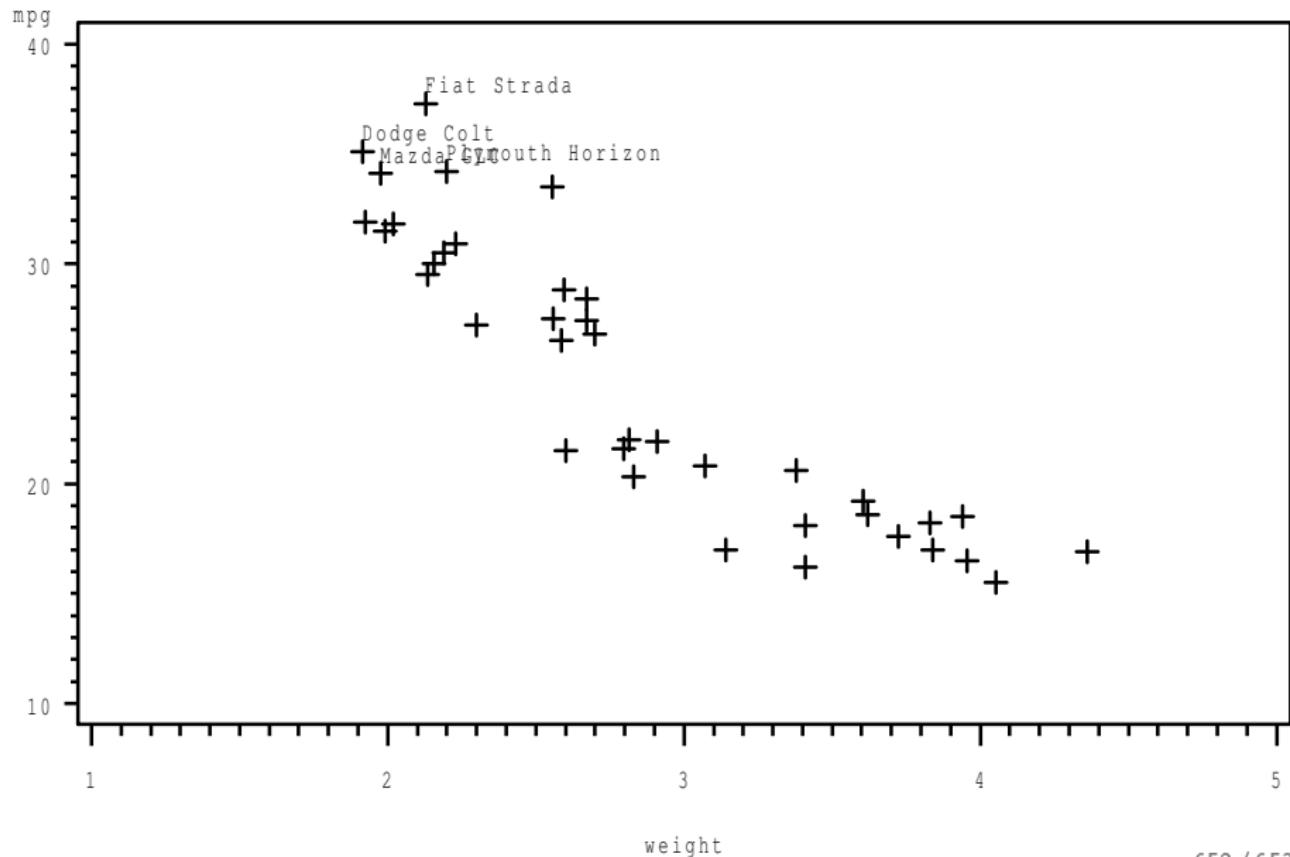
## Extended x-axis



Or label cars with mpg greater than 34

```
SAS> goptions htext=0.5;
SAS>
SAS> data mytext5;
SAS>   retain xsy ysys '2' position '3';
SAS>   set 'cars';
SAS>   if mpg>34 then do;
SAS>     x=weight;
SAS>     y=mpg;
SAS>     text=car;
SAS>   end;
SAS>
SAS> proc gplot data='cars';
SAS>   plot mpg*weight / annotate=mytext5;
```

## High-mpg cars



## Part XVIII

The End!