Section 1

Connecting to SAS

Recall history

SAS

From late 1960s, North Carolina State.

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

Commercialized, corporate ethos.

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

Long history: well-tested.

R

From 1993, New Zealand.

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

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

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

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

Big user community makes sure everything works.

Connecting to SAS

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

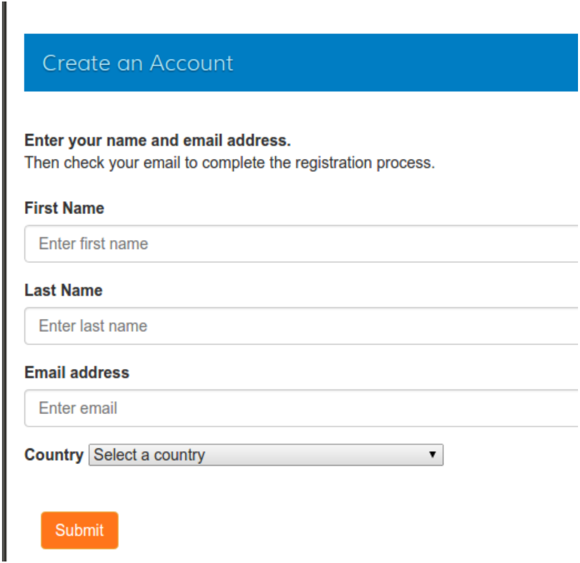
Getting registered for online version

- Go to `https://odamid.oda.sas.com`. Get to this:

- 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):



Go check your e-mail

and look for something like this:

Dear Megan Butler,

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

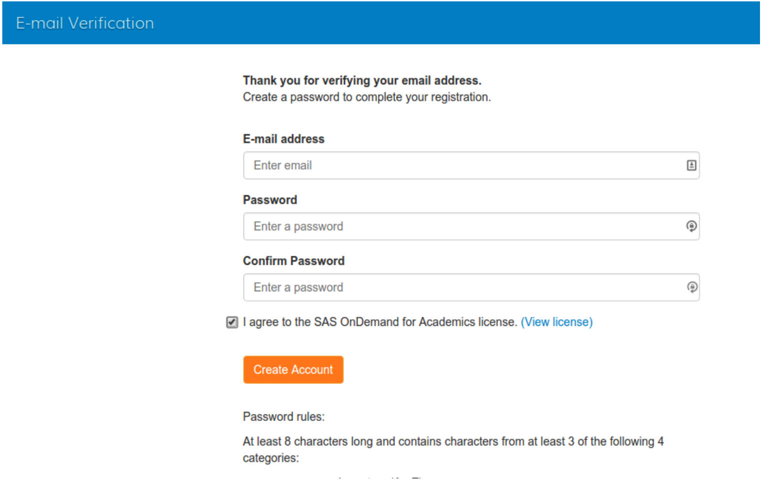
A SAS Profile was created just for you. **Click on the link below to activate your profile and complete the registration process:**

https://odamid.oda.sas.com/SASODARegistration/activate.html?token=8BC03221-340E-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.

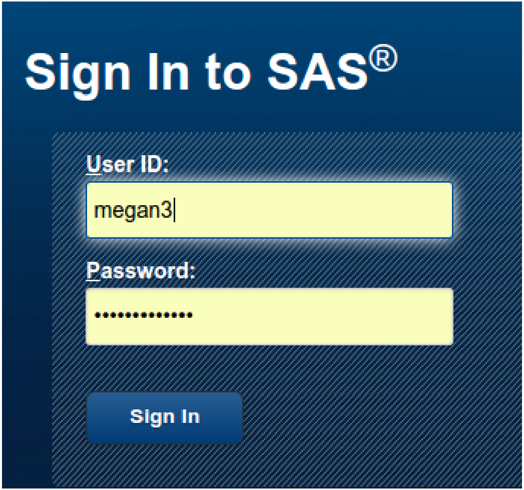
Choose a password



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

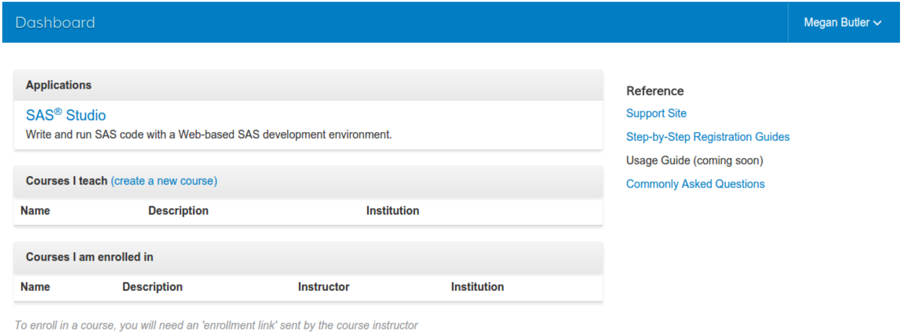
Log into SAS

Go back to the page you bookmarked earlier:



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

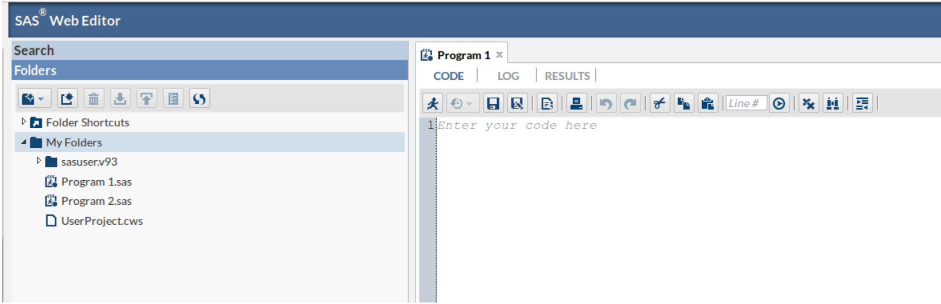
The dashboard



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

SAS, as you see it

Something like this:

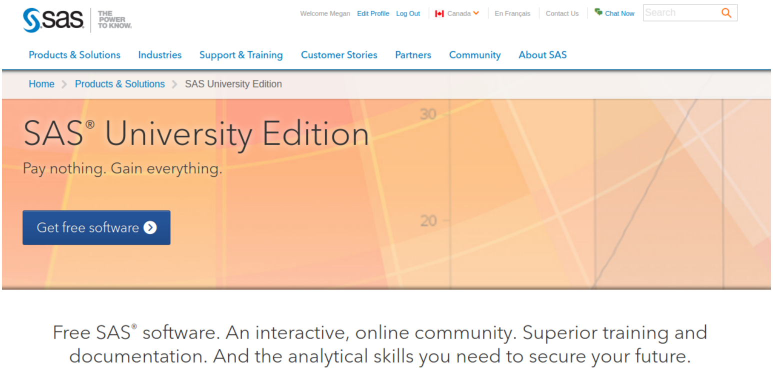


Installing SAS on your own machine

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

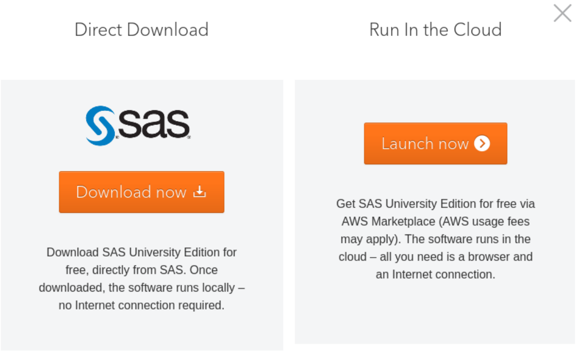
Getting SAS for your own machine

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



And then

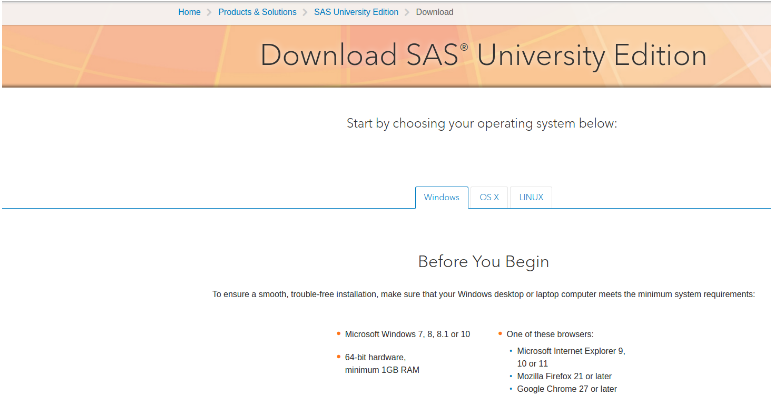
- Click Get Free Software. See this:



- Click Download Now on the *left*.

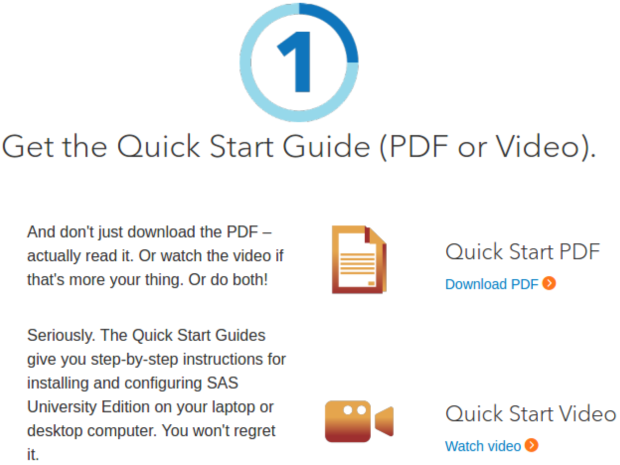
Select operating system

- by clicking appropriate tab, eg:



Starting setup

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



Download VirtualBox

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

2

Install Oracle VirtualBox virtualization software* on your machine.

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

Download VirtualBox for Windows

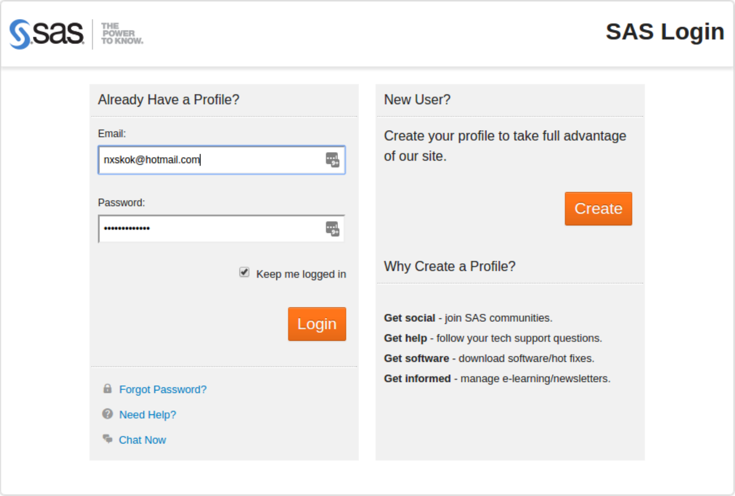
* Note: In addition to Oracle VirtualBox, SAS University Edition also works with VMware Workstation Player virtualization software. If you prefer to use VMware Workstation Player, you can download it here: VMware Workstation Player download page. Charges may apply.

Scroll down some more

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

Creating a "profile"

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



Finally, step 4

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

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

a. Install the latest release of Oracle VirtualBox using the link provided by your site administrator, or see https://www.virtualbox.org/wiki/Downloads.

b. See the SAS University Edition download page (at http://www.sas.com/en_us/software/university-edition/download.html) to get the SAS University Edition vApp. When downloading the SAS University Edition vApp, you might be prompted by your browser to save or run the file. Click **Save** to save this file in your `Downloads` directory.

Quick Start step 2

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

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

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

Step 3: setting up file access

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

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

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

Start SAS

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

Step 4: Start the SAS University Edition vApp.
In VirtualBox, select the SAS University Edition vApp, and then select **Machine > Start.** It might take a few minutes for the virtual machine to start.
Note: When the virtual machine is running, the screen with the SAS logo is replaced with a black console screen (called the Welcome window). You can minimize this window, but **do not close the Welcome window until you are ready to end your SAS session**.

Step 5: Open the SAS University Edition.
a. In a web browser on your local computer, enter `http://localhost:10080`.
b. From the SAS University Edition: Information Center, click **Start SAS Studio**.

SAS Studio online and on your machine

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

Accessing data files in SAS Studio

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

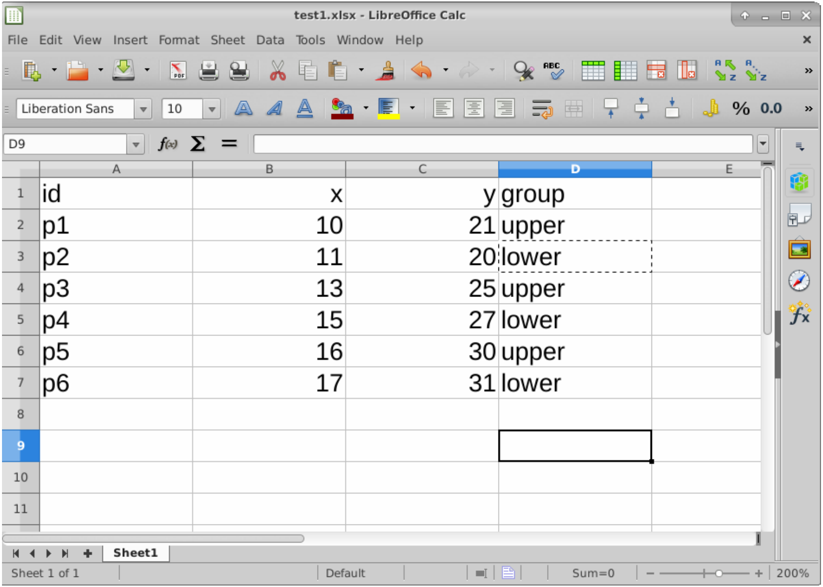
Section 2

Reading data from files

Introduction

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

A spreadsheet



Save as `.csv`

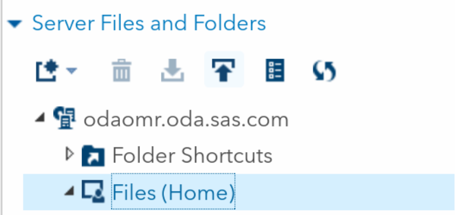
- `.csv` or "comma-separated values" is a way of turning spreadsheet values into plain text.
- but does *not* preserve formulas. (This is a reason for doing *all* your calculations in your statistical software, and *only* having data in your spreadsheet.)
- File, Save As Text CSV (or similar).

The .csv file

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

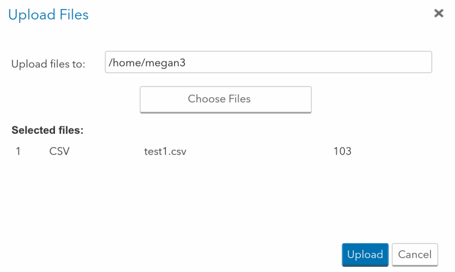
Reading files in SAS

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



...Continued

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



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

Reading in the data

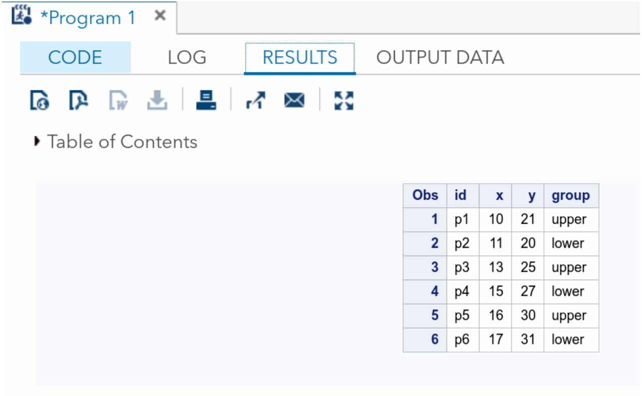
- In SAS Studio, click New (leftmost button under Server Files and Folders) and select New SAS Program.
- On the right, in the Code tab, type code like this, only instead of `ken` put *your* username:

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

proc print;
```

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

Running the code

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



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

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

That code

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

Alternatively

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

Summarizing a data set

- Replace the `proc print` with `proc means`:

  ```
  proc means;
  ```

  That gives the mean, SD, min, max and #observations for each variable (below). Like R `group_by`, `summarize`.

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

```
                          The MEANS Procedure

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

Reading from a URL

- A little extra setup:

```
filename myurl url
  "http://www.utsc.utoronto.ca/~butler/c32/global.csv";

proc import
  datafile=myurl
  dbms=csv
  out=global
  replace;
  getnames=yes;

proc print;
```

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

Did it work?

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

Space-delimited files

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

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

Reading in these data

- Change the `proc import`:

```
filename myurl url
  "http://www.utsc.utoronto.ca/~butler/c32/coffee.tx
proc import
  datafile=myurl
  dbms=dlm
  out=coffee
  replace;
  delimiter=' ';
  getnames=yes;
```

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

Did it work?

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

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

Obs	cup	tempdiff
1	SIGG	12
2	SIGG	16
3	SIGG	9
4	SIGG	23
5	SIGG	11
6	SIGG	20.5
7	SIGG	12.5
8	SIGG	20.5
9	SIGG	24.5
10	Starbucks	13
11	Starbucks	7
12	Starbucks	7
13	Starbucks	17.5
14	Starbucks	10
15	Starbucks	15.5

Reading soap data in SAS

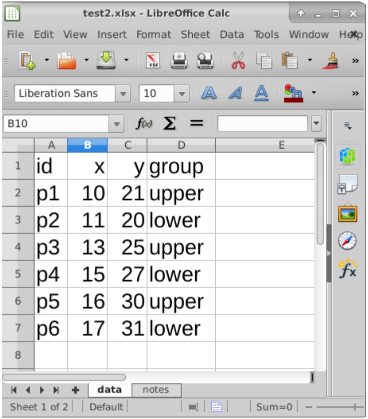
```
filename myurl
 url
 "http://www.utsc.utoronto.ca/~butler/c32/soap.txt";

proc import
  datafile=myurl
  dbms=dlm
  out=soap
  replace;
  delimiter=' ';
  getnames=yes;

proc print data=soap(obs=10);
```

Ten rows of the soap data

Obs	case	scrap	speed	line
1	1	218	100	a
2	2	248	125	a
3	3	360	220	a
4	4	351	205	a
5	5	470	300	a
6	6	394	255	a
7	7	332	225	a
8	8	321	175	a
9	9	410	270	a
10	10	260	170	a

Reading an Excel sheet directly

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



- It is now a workbook with a second sheet called "notes" (that we don't want).

Reading Excel spreadsheet into SAS

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

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

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

The spreadsheet as data set

- Did it work? Yes:

```
proc print;
```

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

Reading Excel files from the Web

- Recall that R's `read_excel` required us to download-save spreadsheet first then read it from local file.
- SAS has no such requirements here.
- Define a `filename myurl url` as before, and use it in the appropriate `proc import`.

Section 3

Graphs

Our data

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

Reading data into SAS

- Upload file to SAS Studio first.
- Or get from `http://www.utsc.utoronto.ca/~butler/c32/ais.txt` and use `filename myurl url` thing first.
- R equivalent: `read_tsv`.
- A bit trickier because we can't *type* tab: have to use special code `'09'x` (ASCII code 09 in hex):

```
filename myurl url
  "http://www.utsc.utoronto.ca/~butler/c32/ais.txt";
proc import
  datafile=myurl
  dbms=dlm
  out=sports
  replace;
  delimiter='09'x;
  getnames=yes;
```

Some of the data, tiny

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

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

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

Or, summarized

```
proc means;
```

The MEANS Procedure

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

Kinds of graph

Reminder: depends on number and type of variables you have:

Categ.	Quant.	Graph	R equiv.
1	0	bar chart	`geom_bar`
0	1	histogram	`geom_histogram`
2	0	grouped bar charts	`geom_bar`, `fill`
1	1	side-by-side boxplots	`geom_boxplot`
0	2	scatterplot	`geom_point`
2	1	grouped boxplots	`geom_boxplot`, `colour`
1	2	scatterplot with points identified by group (eg. by colour)	`geom_point`, `colour`

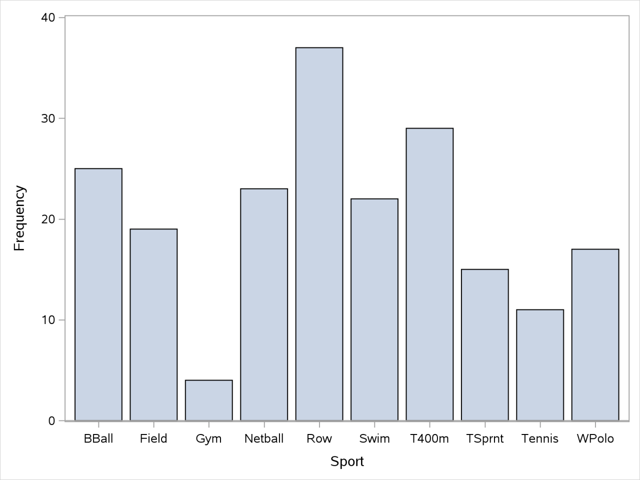
With more variables, *separate plots by groups*: `paneling` in SAS (`facetting` in R).

Workhorse graphing procedure

- SAS also has standard graphing procedure, that we use for all our SAS graphs.
- **proc sgplot**
- Use in different ways to get precise graph we want.
- Start with bar chart of the sports played by the athletes.

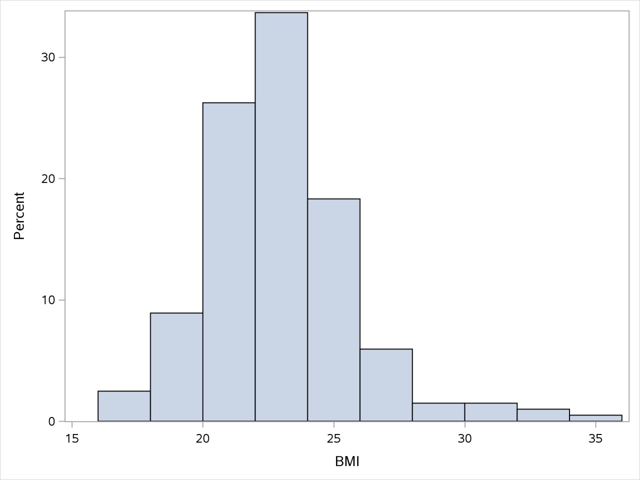
Bar chart in SAS

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



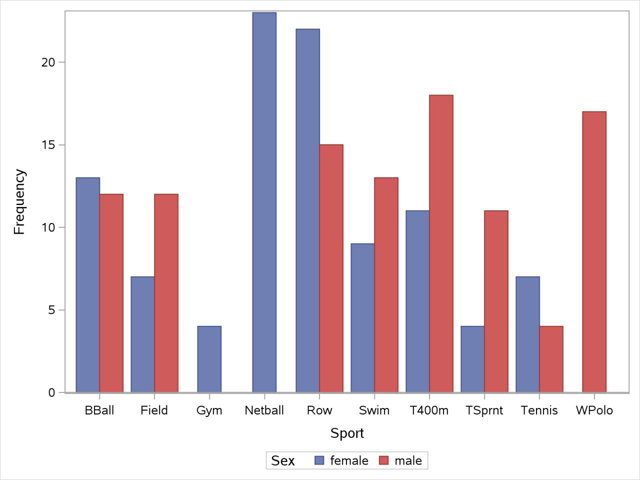
Histogram of body mass index, in SAS

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



Grouped bar plot in SAS

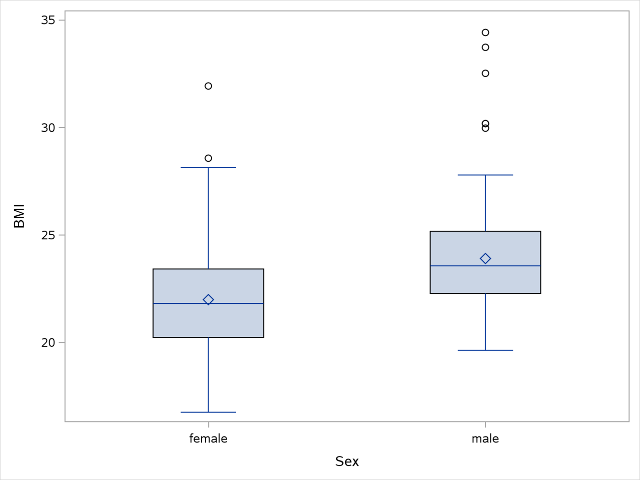
```
proc sgplot;
  vbar Sport / group=Sex groupdisplay=cluster;
```



BMI by gender

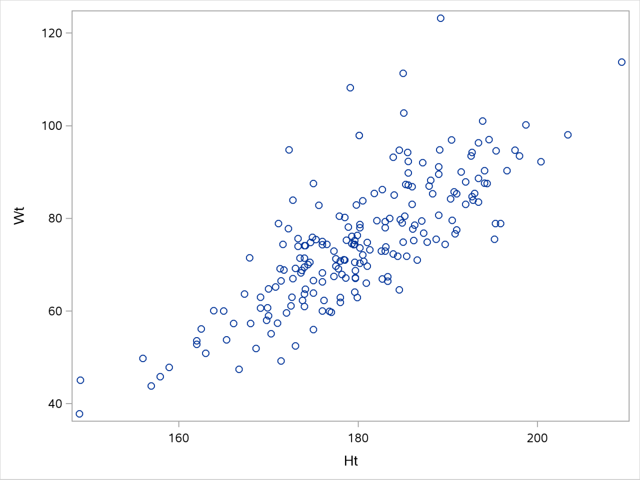
Side-by-side boxplots:

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



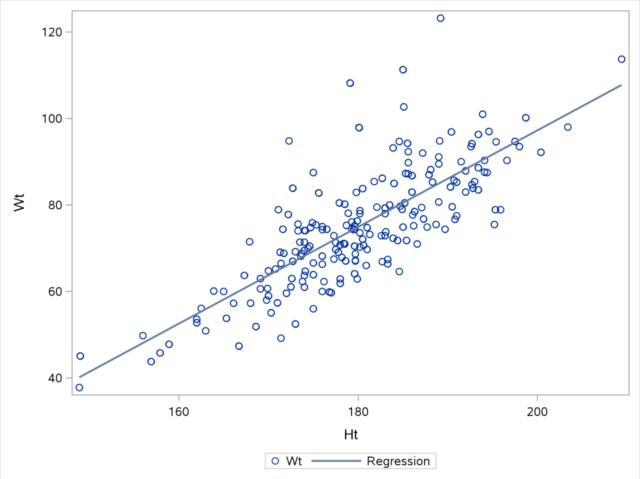
Height vs. weight

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



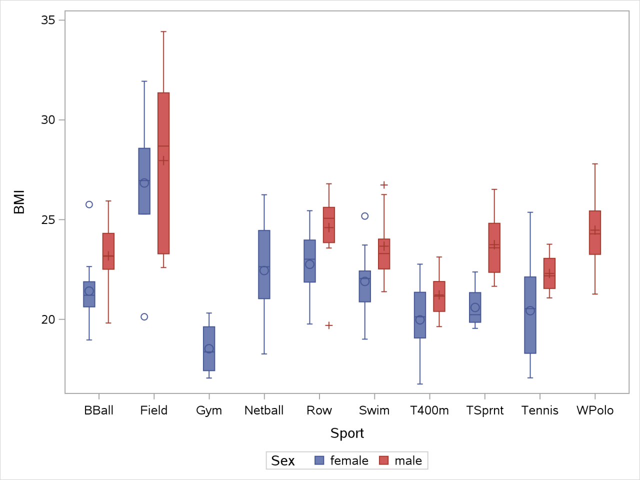
and again, with regression line

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



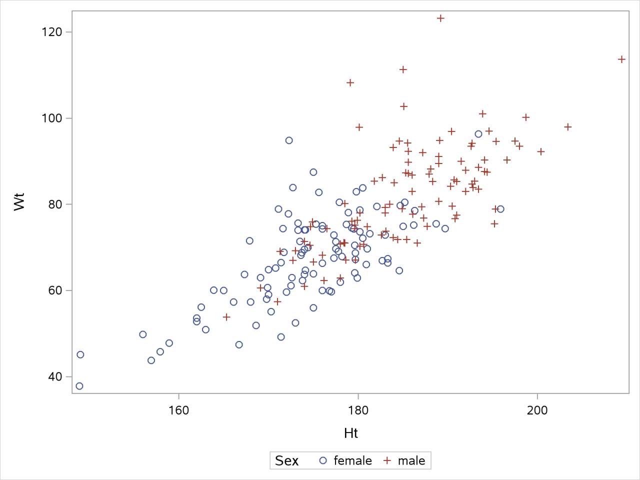
BMI by sport and gender

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



Scatterplot by gender

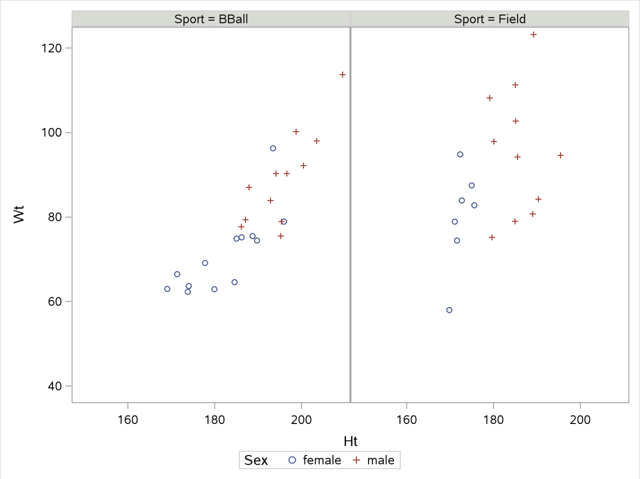
```
proc sgplot;
  scatter x=Ht y=Wt / group=Sex;
```



Height by weight for each sport

Separate plot for each sport, first two panels here:

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



Section 4

More detailed summaries of data

Summarizing data in SAS

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

Specifying summaries, variables and groups

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

Quartiles of athlete weight

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

```
                      The MEANS Procedure

                   Analysis Variable : Wt

            Lower           Upper        Quartile
         Quartile        Quartile           Range
     --------------------------------------------
       66.5000000      84.2000000      17.7000000
     --------------------------------------------
```

Mean and SD of height by gender

- Thus:

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

```
                    The MEANS Procedure

                 Analysis Variable : Ht

                 N
  Sex          Obs           Mean        Std Dev
  ------------------------------------------------
  female       100    174.5940000      8.2422026

  male         102    185.5058824      7.9034874
  ------------------------------------------------
```

How many athletes from each sport?

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

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

- Results over.

Results

```
                    The MEANS Procedure

                 Analysis Variable : BMI

                              N
                  Sport      Obs       N
                  ---------------------
                  BBall       25      25

                  Field       19      19

                  Gym          4       4

                  Netball     23      23

                  Row         37      37

                  Swim        22      22

                  T400m       29      29

                  TSprnt      15      15

                  Tennis      11      11

                  WPolo       17      17
```

A perhaps better way to count

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

The FREQ Procedure

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

SD of all the (numerical) columns

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

```
proc means stddev;
```

```
                 The MEANS Procedure

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

Section 5

Inference

Inference in SAS

Blue Jays data again:

```
filename myurl url
  "http://www.utsc.utoronto.ca/~butler/c32/jays15-home.c
proc import
  datafile=myurl
    dbms=csv
    out=jays
    replace;
  getnames=yes;
```

Checking what I read in

- Especially important in SAS:

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

```
Obs          row          game date                   box      team venue opp result

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

Obs         runs       Oppruns      innings wl       position gb   winner

  1            1             2            . 4-3             2 1    Odorizzi
  2            2             3            . 4-4             3 2    Geltz
  3           12             7            . 5-4             2 1    Buehrle
  4            2             4            . 5-5             4 1.5  Archer
  5            7             8            . 5-6             4 2.5  Martin
  6            6             5           10 6-6             3 1.5  Cecil

Obs loser      save                 game_time  Daynight    attendance  streak

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

Doing a t-test

of a difference from last year's attendance. Null mean is previous year's mean attendance. R: `t.test`:

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

N	Mean	Std Dev	Std Err	Minimum	Maximum
25	25070.2	11006.7	2201.3	14184.0	48414.0

Mean	95% CL Mean		Std Dev	95% CL Std Dev	
25070.2	20526.8	29613.5	11006.7	8594.3	15312.0

DF	t Value	Pr > |t|
24	-1.93	0.0650

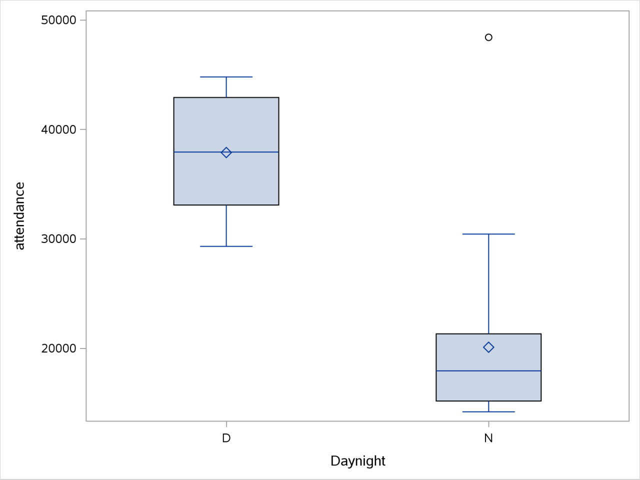
Same CI (20527 to 29614) as R, also same P-value 0.0650.

Day and night games

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

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

The boxplot



Comments

- Attendances on average *much* higher for day games than night ones. Why?
- We should be cautious about doing a t-test here. Why?
- What is that upper outlier in the night games?

Another example: learning to read

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

The data, in SAS

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

```
filename myurl url
  "http://www.utsc.utoronto.ca/~butler/c32/drp.txt";
proc import
  datafile=myurl
  dbms=dlm
  out=reading
  replace;
  delimiter=' ';
  getnames=yes;

  proc print;
```

The data, some, tiny

Obs	group	score
1	t	24
2	t	61
3	t	59
4	t	46
5	t	43
6	t	44
7	t	52
8	t	43
9	t	58
10	t	67
11	t	62
12	t	57
13	t	71
14	t	49
15	t	54
16	t	43
17	t	53
18	t	57
19	t	49
20	t	56
21	t	33
22	c	42
23	c	33
24	c	46
25	c	37
26	c	43
27	c	41
28	c	10
29	c	42
30	c	55
31	c	19
32	c	17
33	c	55

Analysis

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

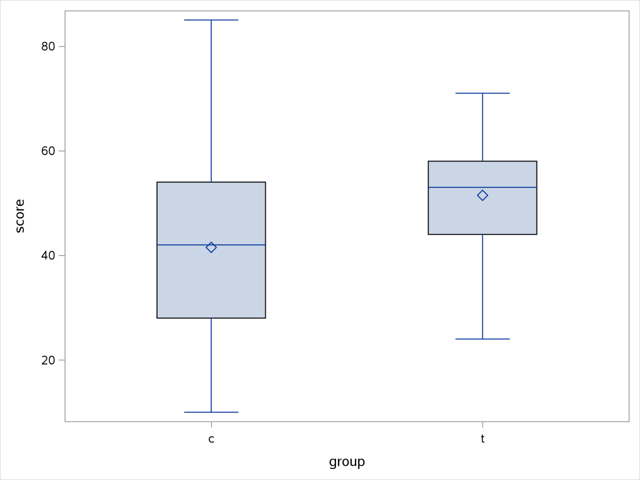
Comparing means

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

```
                              The MEANS Procedure

                           Analysis Variable : score

               N
group        Obs     N            Mean         Std Dev         Minimum         Maximum
-----------------------------------------------------------------------------
c             23    23      41.5217391      17.1487332      10.0000000      85.0000000

t             21    21      51.4761905      11.0073568      24.0000000      71.0000000
-----------------------------------------------------------------------------
```

Boxplots

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



Comments

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

The t-test

In R, was `t.test(score~group)`:

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

```
                    The TTEST Procedure

                     Variable:  score

Method           Variances        DF    t Value    Pr < t

Pooled           Equal            42      -2.27    0.0143
Satterthwaite    Unequal      37.855      -2.31    0.0132
```

plus a lot more output.

Comments and Conclusions

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

Errors in testing

Reminder of what can happen:

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

- Prob. of *not* making type II error called **power** $(= 1 - \beta)$. *High* power good.

Calculating power in SAS

- The magic `proc` is `proc power`.
- We did before: a one-sample t-test with $n = 15$, $H_0 : \mu = 10$ vs. $H_a : \mu \neq 10$, and a true $\mu = 8$:

```
proc power;
  onesamplemeans
  test=t
  nullmean=10
  mean=8
  stddev=4
  ntotal=15
  power=.;
```

- R equivalent was `power.t.test` (or simulation).

The results

```
The POWER Procedure
One-Sample t Test for Mean

Fixed Scenario Elements

Distribution                  Normal
Method                         Exact
Null Mean                         10
Mean                               8
Standard Deviation                 4
Total Sample Size                 15
Number of Sides                    2
Alpha                           0.05

Computed Power

Power

0.438
```

Calculating sample size

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

Using `proc power`

Explicitly leave `ntotal` missing, and supply value for `power`:

```
proc power;
  onesamplemeans
  test=t
  nullmean=10
  mean=8
  stddev=4
  ntotal=.
  power=0.80;
```

Results

```
The POWER Procedure
One-Sample t Test for Mean

Fixed Scenario Elements

Distribution                  Normal
Method                         Exact
Null Mean                         10
Mean                               8
Standard Deviation                 4
Nominal Power                    0.8
Number of Sides                    2
Alpha                           0.05

        Computed N Total

        Actual        N
         Power    Total

         0.808       34
```

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

Power curves

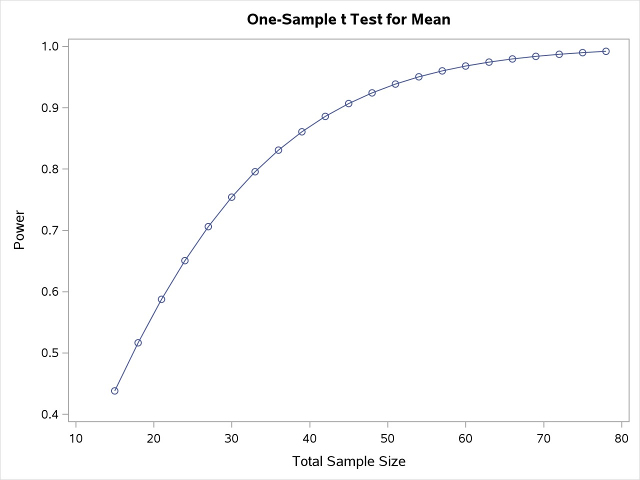
- Rather than calculating power for one sample size, or sample size for one power, might want a *picture* of relationship between sample size and power.
- Or, likewise, picture of relationship between true mean and power.
- Called **power curve**.
- SAS makes these automatically (have to learn how).

Power curves in SAS

Hint: when plotting power curves, supply values for everything except power. In `plot` line, specify what you want as x on the plot. (Power goes on y-axis.) You may have to experiment with limits of x-scale.

```
proc power plotonly;
  onesamplemeans
    test=t
    nullmean=10
    mean=8
    stddev=4
    ntotal=15
    power=.;
  plot x=n min=15 max=80;
```

The graph



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

Power curves for *mean*

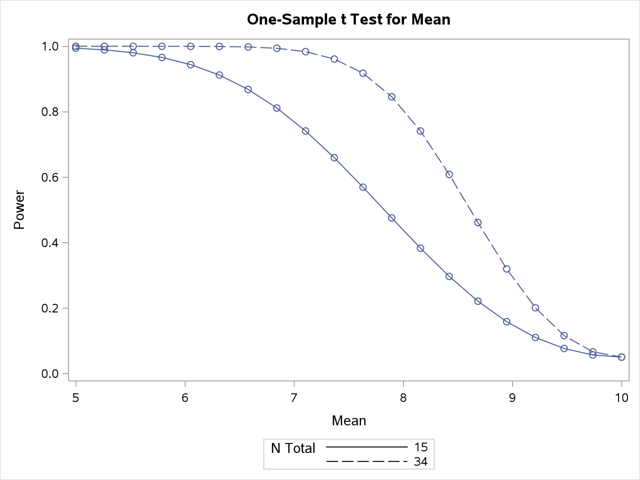
How wrong does the null hypothesis have to be, to have good chance of correctly rejecting the null?

Try for sample sizes $n = 15$ and $n = 34$. (As before, $\sigma = 4$.)

```
proc power plotonly;
  onesamplemeans
    test=t
    nullmean=10
    mean=8
    stddev=4
    ntotal=15 34
    power=.;
  plot x=effect min=5 max=10;
```

- I specify two different sample sizes as shown.
- This time I want "effect size" (mean) on x-axis.

The SAS power curves



Comments over.

Comments

- When true `mean=10`, H_0 actually *true*, and probability of rejecting it then is $\alpha = 0.05$.
- As the null gets more wrong, becomes easier to correctly reject it.
- No matter how wrong H_0 is, always have a greater chance of correctly rejecting it with larger sample size.
- Previously, true mean 8, producing power 0.42 and 0.80.
- With $n = 34$, a mean less than about 7 is almost certain to be correctly rejected. (With $n = 15$, the mean needs to be less than about 5.)

Calculating power for a two-sample t-test

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

Recall sample statistics:

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

```
                         The MEANS Procedure

                     Analysis Variable : score

         N
group  Obs    N          Mean       Std Dev       Minimum       Maximum
-----------------------------------------------------------------------
c       23   23    41.5217391    17.1487332    10.0000000    85.0000000

t       21   21    51.4761905    11.0073568    24.0000000    71.0000000
-----------------------------------------------------------------------
```

What to consider

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

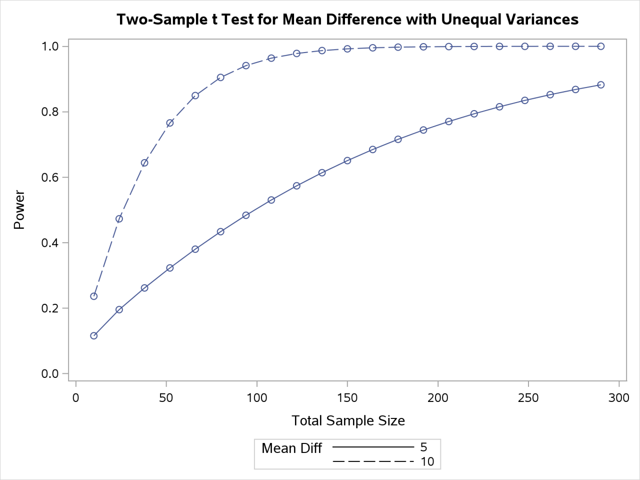
Code

```
proc power plotonly;
  twosamplemeans
    test=diff_satt
    sides=1
    meandiff=5 10
    stddev=15
    ntotal=44
    power=.;
  plot x=n min=10 max=300;
```

Code comments

- `twosamplemeans`
- `test=diff_satt` to specify Satterthwaite two-sample test
- `sides=1` (1-sided test; this is number 1)
- `meandiff` specifies true differences between means. Use two different values.
- Population SDs taken to be 15 for both groups.
- Leave power blank to plot power against something else.
- On `plot` specify what goes on x-axis and its limits.

The power curves



Comments

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

Or, more accurately. . .

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

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

Results

```
                    The POWER Procedure
   Two-Sample t Test for Mean Difference with Unequal Variances

                  Fixed Scenario Elements

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

                      Computed Power

                      Actual
                       Alpha    Power

                      0.0499    0.309
```

Power actually went *up* a tiny bit.

Unequal sample sizes

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

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

Results

Power for 22 in each group was 29%:

```
                The UNIVARIATE Procedure
                    Variable:  Time

               Tests for Location: Mu0=160

Test            -Statistic-    -----p Value------

Student's t     t  1.824364    Pr > |t|    0.0784
Sign            M         2    Pr >= |M|   0.5847
Signed Rank     S        50    Pr >= |S|   0.3118
                 Computed Power

                 Actual
                  Alpha    Power

                 0.0505    0.225
```

Unequal sample sizes bring power down to 22.5%.

Duality between test and CI

The data:

```
filename myurl url
  "http://www.utsc.utoronto.ca/~butler/c32/duality.txt";
proc import
  datafile=myurl
    dbms=dlm
    out=duality
    replace;
  delimiter=' ';
  getnames=yes;


proc print;
```

Output over.

The data, small

Obs	y	group
1	10	1
2	11	1
3	11	1
4	13	1
5	13	1
6	14	1
7	14	1
8	15	1
9	16	1
10	13	2
11	13	2
12	14	2
13	17	2
14	18	2
15	19	2

Test and CI at default $\alpha = 0.05$

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

```
                          The TTEST Procedure

                            Variable:  y

group         Method              Mean       95% CL Mean        Std Dev

1                              13.0000     11.4627  14.5373      2.0000
2                              15.6667     12.8769  18.4564      2.6583
Diff (1-2)    Pooled           -2.6667     -5.2580  -0.0754      2.2758
Diff (1-2)    Satterthwaite    -2.6667     -5.5626   0.2292

            group         Method          95% CL Std Dev

            1                             1.3509   3.8315
            2                             1.6593   6.5198
            Diff (1-2)    Pooled          1.6499   3.6665
            Diff (1-2)    Satterthwaite

      Method           Variances        DF     t Value     Pr > |t|

      Pooled           Equal            13       -2.22       0.0446
      Satterthwaite    Unequal      8.7104       -2.09       0.0668
```

95% CI $(-5.56, 0.23)$ contains null mean of 0, P-value greater than $\alpha = 0.05$ (do not reject 0).

90% CI

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

```
                      The TTEST Procedure

                         Variable:  y

group          Method              Mean       90% CL Mean         Std Dev

1                               13.0000     11.7603   14.2397      2.0000
2                               15.6667     13.4798   17.8535      2.6583
Diff (1-2)     Pooled           -2.6667     -4.7909   -0.5425      2.2758
Diff (1-2)     Satterthwaite    -2.6667     -5.0103   -0.3230

            group          Method              90% CL Std Dev

            1                                  1.4365    3.4220
            2                                  1.7865    5.5539
            Diff (1-2)     Pooled              1.7352    3.3806
            Diff (1-2)     Satterthwaite

      Method           Variances        DF     t Value     Pr > |t|

      Pooled           Equal            13       -2.22       0.0446
      Satterthwaite    Unequal      8.7104       -2.09       0.0668
```

90% CI $(-5.01, -0.32)$ *does not* contain zero, P-value less than $\alpha = 0.10$ (reject 0 at this α).

If you have a test but no CI

- you make a CI by including all the parameter values that would *not be rejected* by your test.
- Use:
  - $\alpha = 0.01$ for a 99% CI,
  - $\alpha = 0.05$ for a 95% CI,
  - $\alpha = 0.10$ for a 90% CI,

  and so on.

Testing for non-normal data

Same data as before:

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

Reading in data

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

Checking: all looks good

```
proc print;
```

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

t-test

$n = 30$ data values:

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

N	Mean	Std Dev	Std Err	Minimum	Maximum
30	201.2	123.8	22.6015	64.0000	527.0

Mean	95% CL Mean		Std Dev	95% CL Std Dev	
201.2	162.8	Infty	123.8	98.5899	166.4

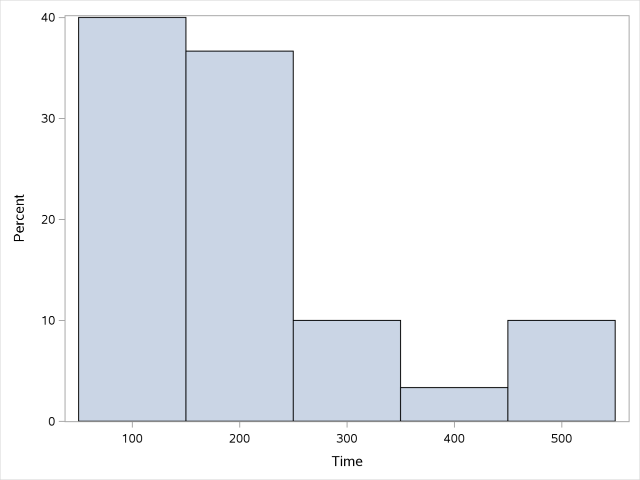
DF	t Value	Pr > t
29	1.82	0.0392

Comments

- All looks good, and we have shown that the mean time to complete this form is greater than 160 minutes (P-value 0.0392).
- **But**, the *t*-test assumes approximately normally-distributed data. We don't have that. Histogram:

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

The histogram



Times are *skewed to the right*.

The sign test

- To test whether the *median* is greater than 160?
- *Count* how many observations above and below 160.
- If too many above, reject null that median is 160, in favour of alternative, median greater.

Doing the sign test in SAS

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

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

```
                     The UNIVARIATE Procedure
                         Variable:  Time

                   Tests for Location: Mu0=160

          Test           -Statistic-    -----p Value------

          Student's t    t  1.824364    Pr > |t|    0.0784
          Sign           M         2    Pr >= |M|   0.5847
          Signed Rank    S        50    Pr >= |S|   0.3118
                           Computed Power

                           Actual
                            Alpha    Power

                           0.0505    0.225
```

Comments

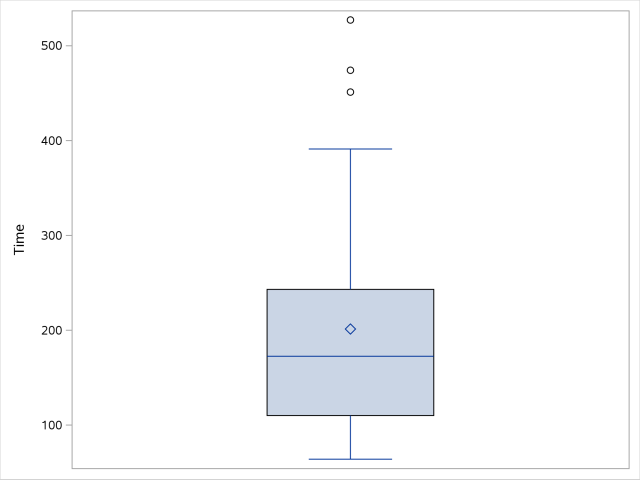
- P-values are (take half of the two-sided SAS ones):

Test	P-value
t	0.0392
Sign	0.2923

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

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

The boxplot



Concluding comments (about this)

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

Confidence interval for the median

- The sign test does not naturally come with a confidence interval for the median.
- So we use the "duality" between test and confidence interval to say: the (95%) confidence interval for the median contains exactly those values of the null median that would not be rejected by the *two-sided* sign test (at $\alpha = 0.05$).
- Uses `proc univariate` (don't have to calculate anything ourselves).

CI for median using `proc univariate`

This is attributed in the SAS documentation to Hahn and Meeker, but it's the same procedure as we used in R:

```
proc univariate cipctldf;
  var Time;
```

The output

The UNIVARIATE Procedure
Variable: Time

Quantiles (Definition 5)

Level	Quantile
100% Max	527.0
99%	527.0
95%	474.0
90%	421.0
75% Q3	243.0
50% Median	172.5
25% Q1	110.0
10%	77.5
5%	65.0
1%	64.0
0% Min	64.0

Quantiles (Definition 5)

Level	95% Confidence Limits Distribution Free		-------Order Statistics------- LCL Rank	UCL Rank	Coverage
100% Max					
99%	.	.	.	.	.
95%	391	527	27	30	72.46
90%	264	527	24	30	93.18
75% Q3	194	451	18	28	96.78
50% Median	119	215	10	21	95.72
25% Q1	71	150	3	13	96.78
10%	64	104	1	7	93.18
5%	64	84	1	4	72.46
1%	.	.	.	.	.
0% Min					

CI for median

- is 119 to 215.
- Same interval as `smmr` gave in R.
- There is no way that 160 would be rejected as the median.

Some different data, and a different test

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

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

Matched pairs

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

Reading data, in SAS

```
filename myurl url
  "http://www.utsc.utoronto.ca/~butler/c32/analgesic.txt
proc import
  datafile=myurl
    dbms=dlm
    out=pain
    replace;
  delimiter=' ';
  getnames=yes;
```

The data

```
proc print;
```

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

Matched pairs *t*-test

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

N	Mean	Std Dev	Std Err	Minimum	Maximum
12	-2.1333	3.4092	0.9841	-12.4000	0.6000

Mean	95% CL Mean		Std Dev	95% CL Std Dev	
-2.1333	-4.2994	0.0327	3.4092	2.4150	5.7884

DF	t Value	Pr > |t|
11	-2.17	0.0530

R equivalent: `t.test(...,paired=T)`

Comments

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

Alternative way to do matched pairs

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

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

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

The new data set pain2

```
proc print;
```

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

Now do t-test on differences

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

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

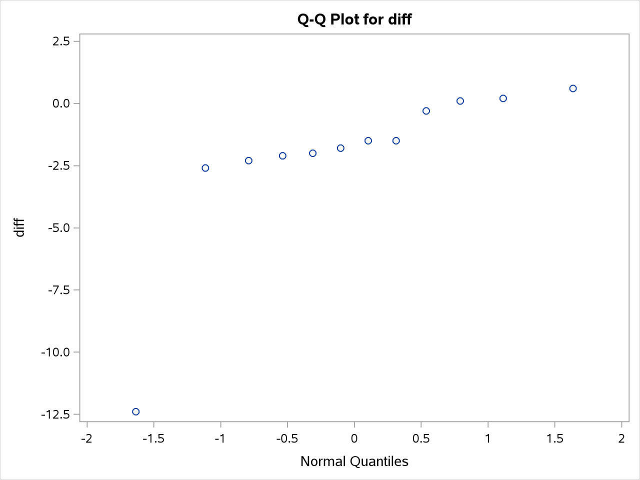
```
 N         Mean      Std Dev      Std Err      Minimum      Maximum

12      -2.1333       3.4092       0.9841     -12.4000       0.6000
    Mean        95% CL Mean         Std Dev       95% CL Std Dev

 -2.1333     -4.2994    0.0327       3.4092       2.4150    5.7884
                     DF     t Value     Pr > |t|

                     11       -2.17       0.0530
```

Assessing normality

- Matched pairs analyses assume (theoretically) that differences normally distributed.
- 1-sample and 2-sample t-tests assume (each) group normally distributed.
- Though we know that t-tests generally behave well even without normality.
- Assess normality with 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, as over.
- R equivalent: `stat_qq`.

Drawing it in SAS

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

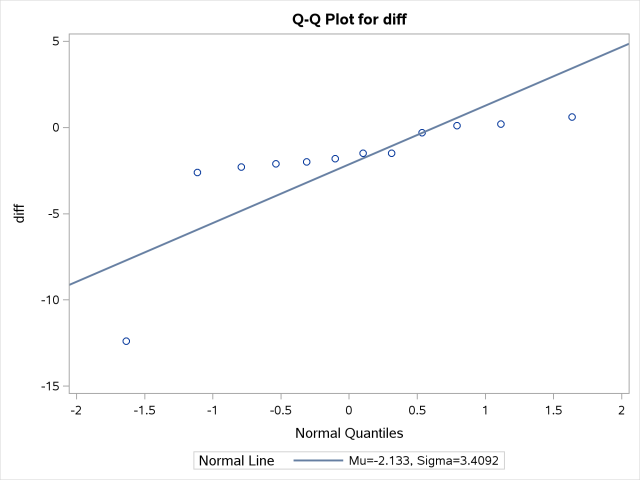


Getting a line on SAS normal quantile plot

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

Having SAS estimate them

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



Another way to estimate μ and σ

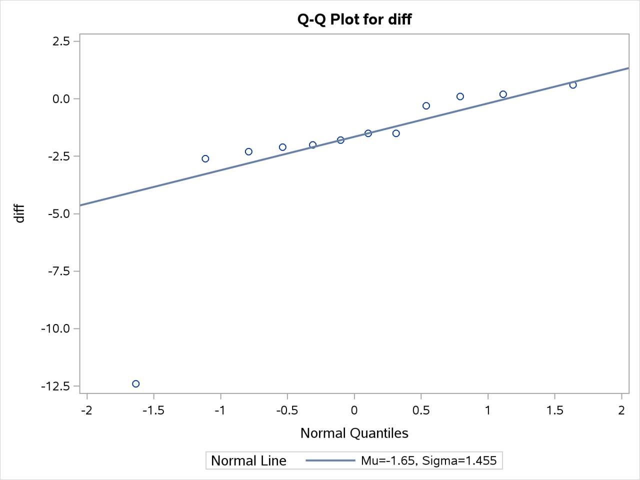
- Problem above is that SD was grossly inflated by outlier.
- On standard normal, quartiles about ± 0.675:

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

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

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

Using improved μ and σ

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



Matched-pairs sign test in SAS

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

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

```
                 The UNIVARIATE Procedure
                    Variable:  diff

               Tests for Location: Mu0=0

Test              -Statistic-    -----p Value------

Student's t       t  -2.16771    Pr > |t|     0.0530
Sign              M        -3    Pr >= |M|    0.1460
Signed Rank       S       -32    Pr >= |S|    0.0088
```

Results

- P-value for t-test 0.0530, for sign test 0.1460.
- Sign test says "no evidence of difference between drugs A and B", while t-test says marginal evidence of difference.

Mood's median test

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

Mood's median test for kids' reading data

```
filename myurl url
  "http://www.utsc.utoronto.ca/~butler/c32/drp.txt";
proc import
  datafile=myurl
  dbms=dlm
  out=reading
  replace;
  delimiter=' ';
  getnames=yes;

  proc print;
```

The data (tiny)

Obs	group	score
1	t	24
2	t	61
3	t	59
4	t	46
5	t	43
6	t	44
7	t	52
8	t	43
9	t	58
10	t	67
11	t	62
12	t	57
13	t	71
14	t	49
15	t	54
16	t	43
17	t	53
18	t	57
19	t	49
20	t	56
21	t	33
22	c	42
23	c	33
24	c	46
25	c	37
26	c	43
27	c	41
28	c	10
29	c	42
30	c	55
31	c	19
32	c	17
33	c	55

Doing Mood's median test

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

```
                        The NPAR1WAY Procedure

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

                       Sum of      Expected       Std Dev         Mean
group          N       Scores      Under H0      Under H0        Score

t             21         14.0         10.50      1.675750     0.666667
c             23          8.0         11.50      1.675750     0.347826

                  Average scores were used for ties.
```

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

Results

```
      Median Two-Sample Test

Statistic                14.0000
Z                         2.0886
One-Sided Pr >  Z         0.0184
Two-Sided Pr > |Z|        0.0367
      Median One-Way Analysis

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

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

Jumping rats

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

Analysis in SAS

Read in data and do ANOVA. R equivalent: aov.

```
filename myurl url
  "http://www.utsc.utoronto.ca/~butler/c32/jumping.txt";
proc import
  datafile=myurl
    dbms=dlm
    out=rats
    replace;
  delimiter=' ';
  getnames=yes;

proc anova;
  class group;
  model density=group;
```

Results (some)

The ANOVA Procedure

Dependent Variable: density

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

Source	DF	Anova SS	Mean Square	F Value	Pr > F
group	2	7433.866667	3716.933333	7.98	0.0019

Tukey in SAS

R equivalent: `TukeyHSD`.

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

Strategy: if you intend to do Tukey (if the ANOVA comes out significant), submit *all* these lines the first time. If the ANOVA F-test is not significant, *ignore* the Tukey.

Tukey output (some)

```
                    The ANOVA Procedure

     Tukey's Studentized Range (HSD) Test for density

Means with the same letter are not significantly different.


Tukey Grouping            Mean      N    group

             A         638.700     10    Highjum

             B         612.500     10    Lowjump
             B
             B         601.100     10    Control
```

Mood's median test

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

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

```
                      The NPAR1WAY Procedure

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

                          Sum of      Expected       Std Dev          Mean
group            N        Scores      Under H0      Under H0         Score

Control         10           1.0           5.0      1.313064          0.10
Lowjump         10           4.0           5.0      1.313064          0.40
Highjum         10          10.0           5.0      1.313064          1.00

                   Average scores were used for ties.
```

Rest of output

```
                Median One-Way Analysis

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

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

Welch ANOVA in SAS

R equivalent: `oneway.test`.

The instruction to do the Welch ANOVA goes on the `means` line, where the Tukey would go if we were doing that:

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

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

Results

```
                    The ANOVA Procedure

                 Welch's ANOVA for density

       Source            DF     F Value    Pr > F

       group         2.0000        8.82    0.0023
       Error        17.4054
  Level of            -----------density-----------
  group          N             Mean          Std Dev

  Control       10       601.100000       27.3636011
  Highjum       10       638.700000       16.5935061
  Lowjump       10       612.500000       19.3290225
```

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

Games-Howell

- The approved multiple-comparisons test for Welch's ANOVA is Games-Howell, which can be done this way:

```
proc mixed;
  class group;
  model density=group / ddfm=satterth;
  repeated / group=group;
  lsmeans group / adjust=tukey adjdfe=row;
```

- In `group=group`, first `group` is always `group`, second one is name of your categorical variable. (Here that *was* `group`.)
- There are (many other) details in the code, not explained here.

Results

The Mixed Procedure

Differences of Least Squares Means

Effect	group	_group	Estimate	Standard Error	DF	t Value	Pr > |t|
group	Control	Highjum	-37.6000	10.1198	14.8	-3.72	0.0021
group	Control	Lowjump	-11.4000	10.5942	16.2	-1.08	0.2977
group	Highjum	Lowjump	26.2000	8.0558	17.6	3.25	0.0045

Differences of Least Squares Means

Effect	group	_group	Adjustment	Adj P
group	Control	Highjum	Tukey-Kramer	0.0056
group	Control	Lowjump	Tukey-Kramer	0.5417
group	Highjum	Lowjump	Tukey-Kramer	0.0120

High jumping significantly different from others (again).

Section 6

Tidying and organizing data

Some tidying and organizing in SAS

- SAS is less flexible than R's `tidyverse` tools, but some of the previous can be done (with effort).
- Basic idea: create a new dataset using `data` and `set`, and then provide additional code to say what to do to the previous dataset.
- Read Australian athletes data again:

```
filename myurl url
  "http://www.utsc.utoronto.ca/~butler/c32/ais.txt";
proc import
  datafile=myurl
  dbms=dlm
  out=sports
  replace;
  delimiter='09'x;
  getnames=yes;
```

Check the data

```
proc print data=sports(obs=10);
```

Obs	Sex	Sport	RCC	WCC	Hc
1	female	Netball	4.56	13.3	42.2
2	female	Netball	4.15	6	38
3	female	Netball	4.16	7.6	37.5
4	female	Netball	4.32	6.4	37.7
5	female	Netball	4.06	5.8	38.7
6	female	Netball	4.12	6.1	36.6
7	female	Netball	4.17	5	37.4
8	female	Netball	3.8	6.6	36.5
9	female	Netball	3.96	5.5	36.3
10	female	Netball	4.44	9.7	41.4

Obs	Hg	Ferr	BMI	SSF
1	13.6	20	19.16	49
2	12.7	59	21.15	110.2
3	12.3	22	21.4	89
4	12.3	30	21.03	98.3
5	12.8	78	21.77	122.1
6	11.8	21	21.38	90.4
7	12.7	109	21.47	106.9
8	12.4	102	24.45	156.6
9	12.4	71	22.63	101.1
10	14.1	64	22.8	126.4

Obs	_Bfat	LBM	Ht	Wt
1	11.29	53.14	176.8	59.9
2	25.26	47.09	172.6	63
3	19.39	53.44	176	66.3
4	19.63	48.78	169.9	60.7
5	[illegible]	[illegible]	[illegible]	[illegible]

Choosing variables

keep to say which ones you want:

```
data sports2;
  set sports;
  keep Sport Sex Ht Wt;

proc print data=sports2(obs=8);
```

Obs	Sex	Sport	Ht	Wt
1	female	Netball	176.8	59.9
2	female	Netball	172.6	63
3	female	Netball	176	66.3
4	female	Netball	169.9	60.7
5	female	Netball	183	72.9
6	female	Netball	178.2	67.9
7	female	Netball	177.3	67.5
8	female	Netball	174.1	74.1

Un-choosing variables

drop to say which ones you don't want. Note the double-dash to denote "this through that":

```
data sports3;
  set sports;
  drop RCC--LBM;

proc print data=sports3(obs=8);
```

Obs	Sex	Sport	Ht	Wt
1	female	Netball	176.8	59.9
2	female	Netball	172.6	63
3	female	Netball	176	66.3
4	female	Netball	169.9	60.7
5	female	Netball	183	72.9
6	female	Netball	178.2	67.9
7	female	Netball	177.3	67.5
8	female	Netball	174.1	74.1

Comments

- Normally don't worry about explicitly dropping variables you don't need; you just ignore them in your analysis.
- `keep` and `drop` mostly for final "tidy" version of datasets that you create.
- Can also feed `proc print` the columns to display, with `var`.
- For example, might want to discard intermediate steps of a calculation.
- `keep` and `drop` equivalent to R `select`.

Calculating a new variable

Put the calculation in the `data` step, as we have seen before. R equivalent: `mutate`.

```
data sports4;
  set sports;
  Wt_lb=Wt*2.2;
  keep Sport Wt Wt_lb;


proc print data=sports4(obs=7);
```

```
Obs    Sport              Wt     Wt_lb

  1    Netball          59.9    131.78
  2    Netball            63    138.60
  3    Netball          66.3    145.86
  4    Netball          60.7    133.54
  5    Netball          72.9    160.38
  6    Netball          67.9    149.38
  7    Netball          67.5    148.50
```

Choosing rows by row number

SAS has a special variable `_N_` that holds the row number. R equivalent: `slice`.

```
data sports5;
  set sports;
  if _N_>=16 and _N_<=25;

proc print;
```

Obs	Sex	Sport	RCC	WCC	Hc
1	female	Netball	4.25	10.7	39.5
2	female	Netball	4.46	10.9	39.7
3	female	Netball	4.4	9.3	40.4
4	female	Netball	4.83	8.4	41.8
5	female	Netball	4.23	6.9	38.3
6	female	Netball	4.24	8.4	37.6
7	female	Netball	3.95	6.6	38.4
8	female	Netball	4.03	8.5	37.7
9	female	BBall	3.96	7.5	37.5
10	female	BBall	4.41	8.3	38.2

Obs	Hg	Ferr	BMI	SSF

Choosing each of a number of rows

```
data sports6;
  set sports;
  if _N_ in (10, 13, 17, 42);

proc print;
```

Obs	Sex	Sport	RCC	WCC	Hc
1	female	Netball	4.44	9.7	41.4
2	female	Netball	4.02	9.1	37.7
3	female	Netball	4.46	10.9	39.7
4	female	Row	4.37	8.1	41.8

Obs	Hg	Ferr	BMI	SSF
1	14.1	64	22.8	126.4
2	12.7	107	23.01	77
3	13.7	102	23.99	115.9
4	14.3	53	23.47	98

Obs	_Bfat	LBM	Ht	Wt

Choosing rows where a condition is true

`if` like R's `filter`, but note that SAS uses *one* equals sign in testing for equality:

```
data sports7;
  set sports;
  if Sport="Tennis";


proc print;
```

Obs	Sex	Sport	RCC	WCC	Hc
1	female	Tennis	4	4.2	36.6
2	female	Tennis	4.4	4	40.8
3	female	Tennis	4.38	7.9	39.8
4	female	Tennis	4.08	6.6	37.8
5	female	Tennis	4.98	6.4	44.8
6	female	Tennis	5.16	7.2	44.3
7	female	Tennis	4.66	6.4	40.9
8	male	Tennis	5.66	8.3	50.2
9	male	Tennis	5.03	6.4	42.7
10	male	Tennis	4.97	8.8	43
11	male	Tennis	5.38	6.3	46

Multiple conditions 1/2

Join them with actual words `and`, `or`:

```
data sports8;
  set sports;
  if Sport="Tennis" and RCC<5;

proc print;
  var Sex--RCC;
```

Obs	Sex	Sport	RCC
1	female	Tennis	4
2	female	Tennis	4.4
3	female	Tennis	4.38
4	female	Tennis	4.08
5	female	Tennis	4.98
6	female	Tennis	4.66
7	male	Tennis	4.97

Multiple conditions 2/2

```
data sports9;
  set sports;
  if Sport="Tennis" or RCC>5;


proc print;
  var Sex--RCC BMI;
```

Obs	Sex	Sport	RCC	BMI
1	female	Row	5.02	19.76
2	female	T400m	5.31	21.35
3	female	Field	5.33	25.27
4	female	TSprnt	5.16	20.3
5	female	Tennis	4	25.36
6	female	Tennis	4.4	22.12
7	female	Tennis	4.38	21.25
8	female	Tennis	4.08	20.53
9	female	Tennis	4.98	17.06
10	female	Tennis	5.16	18.29
11	female	Tennis	4.66	18.37
12	male	Swim	5.13	22.46
13	male	Swim	5.09	23.68
14	male	Swim	5.17	23.15
15	male	Swim	5.11	22.32
16	male	Swim	5.03	24.02
17	male	Swim	5.32	23.29
18	male	Swim	5.34	22.81
19	male	Swim	5.33	21.38
20	male	Row	5.04	25.84

Using data `where` a condition is true

- Rather than creating a new data set containing the values that satisfy a condition, we can tell SAS which data to use right in a `proc`.
- As near as SAS gets to R pipeline.
- Key idea: put `where` and a logical condition as the *first* line of the `proc`.
- For example, mean BMI of tennis players:

```
proc means;
  where sport="Tennis";
  var BMI;
```

Mean and SD of BMI for tennis players

The MEANS Procedure

Analysis Variable : BMI

N	Mean	Std Dev	Minimum	Maximum
11	21.1054545	2.4626789	17.0600000	25.3600000

Arranging values in order

This is `proc sort`, which produces an output data set that is the "most recent" one. R equiv: `arrange`.

```
proc sort data=sports;
  by RCC;

proc print;
  var Sex--RCC;
```

Obs	Sex	Sport	RCC
1	female	Netball	3.8
2	female	Netball	3.9
3	female	T400m	3.9
4	female	Row	3.91
5	female	Netball	3.95
6	female	Row	3.95
7	female	Netball	3.96
8	female	BBall	3.96
9	female	Tennis	4
10	female	Netball	4.02
11	female	Netball	4.03

Using a second variable as tiebreaker

```
proc sort data=sports;
  by RCC BMI;

proc print;
  var Sex--RCC BMI;
```

Obs	Sex	Sport	RCC	BMI
1	female	Netball	3.8	24.45
2	female	T400m	3.9	19.37
3	female	Netball	3.9	20.06
4	female	Row	3.91	22.27
5	female	Netball	3.95	19.87
6	female	Row	3.95	24.54
7	female	BBall	3.96	20.56
8	female	Netball	3.96	22.63
9	female	Tennis	4	25.36
10	female	Netball	4.02	23.01
11	female	Netball	4.03	23.35
12	female	Netball	4.06	21.77
13	female	Swim	4.07	20.42
14	female	Tennis	4.08	20.53

Descending order

```
proc sort data=sports;
  by descending BMI;

proc print;
  var Sex--RCC BMI;
```

Obs	Sex	Sport	RCC	BMI
1	male	Field	5.48	34.42
2	male	Field	4.96	33.73
3	male	Field	5.48	32.52
4	female	Field	4.75	31.93
5	male	Field	5.01	30.18
6	male	Field	5.01	30.18
7	male	Field	5.09	29.97
8	female	Field	4.58	28.57
9	female	Field	4.51	28.13
10	male	WPolo	5.34	27.79
11	male	WPolo	4.9	27.56
12	male	Field	5.11	27.39
13	female	Field	4.81	26.95
14	male	WPolo	5.08	26.86

Displaying the seven heaviest athletes

```
proc sort data=sports;
  by descending Wt;

data sports10;
  set sports;
  if _N_<=7;
  keep Sport Wt;

proc print;
```

Obs	Sport	Wt
1	Field	123.2
2	BBall	113.7
3	Field	111.3
4	Field	108.2
5	Field	102.7
6	WPolo	101
7	BBall	100.2

Tidying data

- Data rarely come to us as we want to use them.
- Before we can do analysis, typically have organizing to do.
- This is typical of ANOVA-type data, "wide format":
- 20 pigs are randomly allocated to one of four feeds. At the end of the study, the weight of each pig is recorded, and we want to know whether there are any differences in mean weights among the feeds.
- Problem: want the weights all in *one* column, with 2nd column labelling which feed each weight was from. Untidy!

Tidy and untidy data (Wickham)

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

Tidying data in SAS

- Hard. Illustrate the SAS version of `gather` on the pigs data, that we have to read in first.
- Each line of this dataset has to produce *four* lines of the long data set.

```
filename myurl url
  "http://www.utsc.utoronto.ca/~butler/c32/pigs1.txt
proc import
  datafile=myurl
  dbms=dlm out=pigs replace;
  delimiter=' ';
  getnames=yes;


proc print;
```

Obs	feed1	feed2	feed3	feed4
1	60.8	68.7	92.6	87.9
2	57	67.7	92.1	84.2
3	65	74	90.2	83.1
4	58.6	66.3	96.5	85.7

Making the long data set, the tedious way

```
data pigs2;
  set pigs;
  feed='feed1';
  weight=feed1;
  output;
  feed='feed2';
  weight=feed2;
  output;
  feed='feed3';
  weight=feed3;
  output;
  feed='feed4';
  weight=feed4;
  output;
  keep feed weight;
```

The long data set

```
proc print;
```

Obs	feed	weight
1	feed1	60.8
2	feed2	68.7
3	feed3	92.6
4	feed4	87.9
5	feed1	57.0
6	feed2	67.7
7	feed3	92.1
8	feed4	84.2
9	feed1	65.0
10	feed2	74.0
11	feed3	90.2
12	feed4	83.1
13	feed1	58.6
14	feed2	66.3
15	feed3	96.5
16	feed4	85.7
17	feed1	61.7
18	feed2	69.8
19	feed3	99.1
20	feed4	90.3

Using a SAS array to reduce repetition

```
data pigs3;
  set pigs;
  array feed_array [4] feed1-feed4;
  do i=1 to 4;
    weight=feed_array[i];
    feed=vname(feed_array[i]);
    output;
  end;
  keep pig feed weight;
```

- In SAS, an array is a mechanism for referring to a group of variables together, here the four feed variables. The *i*-th element of the array refers to the *i*-th feed variable.
- In the loop (indented), `weight` is set to the *value* of the appropriate one of the `feed` variables, while `feed` is set to the *name* of that `feed` variable. Compare the coding without the loop.

The long data set, again

```
proc print;
```

Obs	weight	feed
1	60.8	feed1
2	68.7	feed2
3	92.6	feed3
4	87.9	feed4
5	57.0	feed1
6	67.7	feed2
7	92.1	feed3
8	84.2	feed4
9	65.0	feed1
10	74.0	feed2
11	90.2	feed3
12	83.1	feed4
13	58.6	feed1
14	66.3	feed2
15	96.5	feed3
16	85.7	feed4
17	61.7	feed1
18	69.8	feed2
19	99.1	feed3
20	90.3	feed4

The ANOVA, again, with output part 1

```
proc anova;
  class feed;
  model weight=feed;
  means feed / tukey;
```

```
                          The ANOVA Procedure

                     Dependent Variable: weight

                                  Sum of
Source                 DF        Squares    Mean Square    F Value    Pr > F

Model                   3    3520.525500    1173.508500     119.14    <.0001

Error                  16     157.600000       9.850000

Corrected Total        19    3678.125500
Source                 DF        Anova SS    Mean Square    F Value    Pr > F

feed                    3    3520.525500    1173.508500     119.14    <.0001
```

The mean weights are not all the same for each feed.

Tukey output

```
Means with the same letter are not significantly different.
```

Tukey Grouping	Mean	N	feed
A	94.100	5	feed3
B	86.240	5	feed4
C	69.300	5	feed2
D	60.620	5	feed1

All of the feeds have significantly different mean weight, with feed 3 being the best and feed 1 the worst.

Section 7

Case study 2: Electricity, peak hour demand and total energy usage

Another regression example (SAS)

- Electric utility company wants to relate peak-hour demand (kW) to total energy usage (kWh) during a month.
- Important planning problem, because generation system must be large enough to meet maximum demand.
- Data from 53 residential customers from August.
- Read in data and draw scatterplot:

```
filename myurl url
  "http://www.utsc.utoronto.ca/~butler/c32/global.cs
proc import
    datafile=myurl
    dbms=dlm
    out=util
    replace;
    delimiter=' ';
    getnames=yes;
```

Check data

The first few rows, which look reasonable:

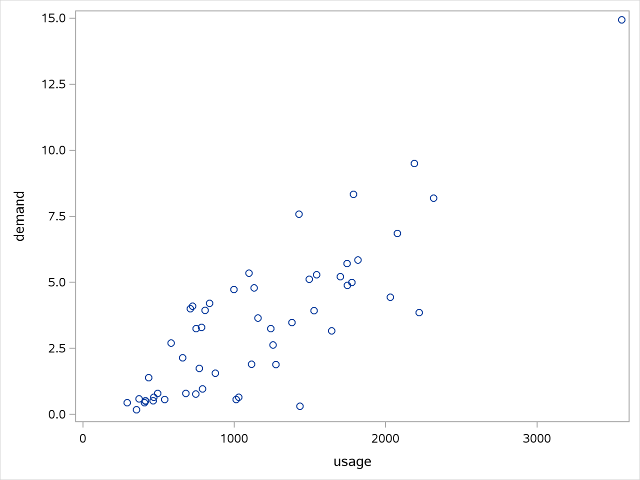
```
proc print data=util(obs=8);
```

Obs	usage	demand
1	679	0.79
2	292	0.44
3	1012	0.56
4	493	0.79
5	582	2.7
6	1156	3.64
7	997	4.73
8	2189	9.5

Make a scatterplot:

```
proc sgplot;
  scatter x=usage y=demand;
```

Scatterplot



Fitting a regression

- Concern: outlier top right (though appears to be legit values)
- Trend basically straight, and outlier appears to be on it.
- So try fitting regression:

```
proc reg;
  model demand=usage;
```

Regression output

The REG Procedure
Model: MODEL1
Dependent Variable: demand

Root MSE	1.57720	R-Square	0.7046
Dependent Mean	3.41321	Adj R-Sq	0.6988
Coeff Var	46.20882		

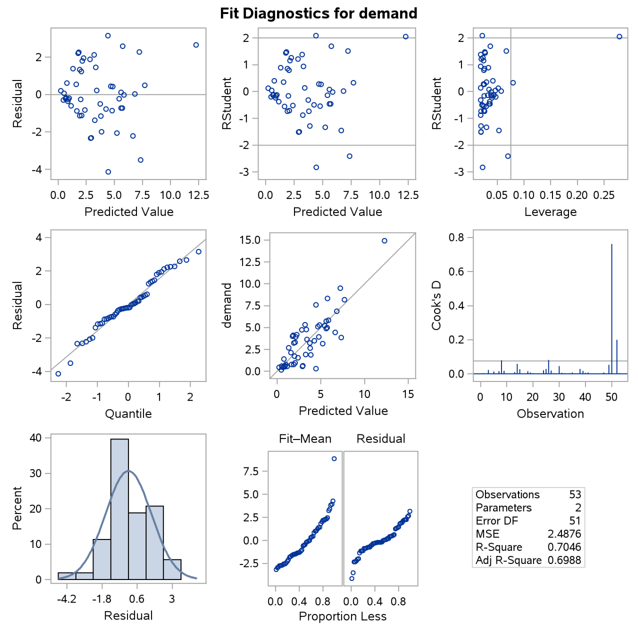
Parameter Estimates

Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	1	-0.83130	0.44161	-1.88	0.0655
usage	1	0.00368	0.00033390	11.03	<.0001

Comments

- R-squared 70%: not bad!
- Statistically significant slope: `demand` really does depend on `usage`.
- But should look at residuals.
- Output from regression also includes array of "diagnostic plots", over:

Regression diagnostics



Observations	53
Parameters	2
Error DF	51
MSE	2.4876
R-Square	0.7046
Adj R-Square	0.6988

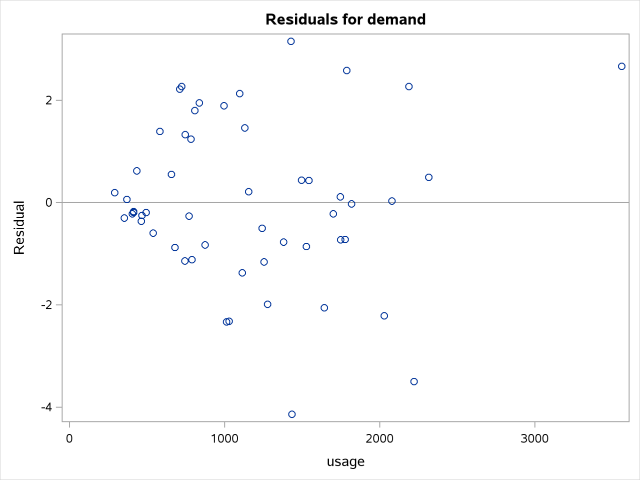
What the diagnostic plots show

Counting from top left, along the rows:

1. Regular residual plot, against fitted values
2. Standardized residuals against fitted values, with the advantage that the standardized residuals behave like z-scores
3. Standardized residuals against leverages (high leverage means unusual x)
4. normal quantile plot of residuals
5. response against predicted
6. Cook's distance (overall influence) against observation number
7. histogram (with normal curve) of residuals
8. I never use this one!
9. summary of regression

Over, residuals against x's (only one here, `usage`).

Residual plot



General comments on these plots

- I usually look at plots #1 and #4 of the diagnostic plots, and maybe the big plot of residuals against x.
- Plot of residuals against fitted values shows (if it has a pattern) any problems with the regression.
- Residuals should be approx. normal. Normal quantile plot shows if they are not.
- Plot of residuals against x's show any problems with that particular x (eg. nonlinearity). With only one x, same conclusion as residual plot.
- Look at leverages/Cook's distances to see if any unusually large ones.

Comments for these data

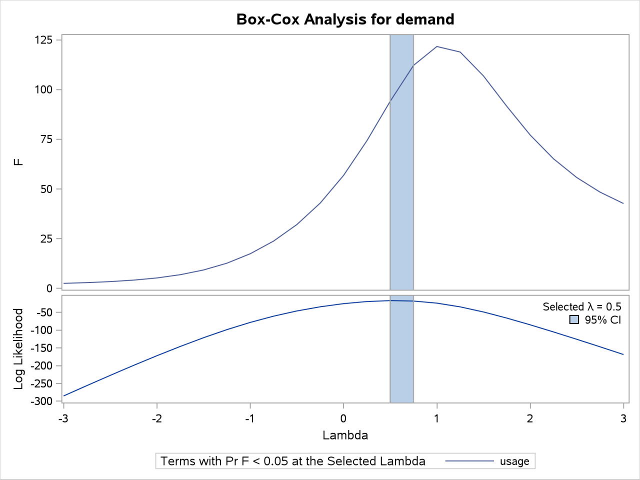
- No trend in residuals vs. fitted
- but: residuals for `demand` close to 0 are themselves close to zero
- and: residuals for larger `demand` tend to get farther from zero
- at least up to `demand` 5 or so.
- One of the assumptions hiding behind regression is that residuals should be of equal size, not "fanning out" as here.
- Remedy: transformation of response variable.
- Note: there is one point with large leverage, the observation with large usage and demand.

But what transformation?

- Best way: consult with person who brought you the data.
- Can't do that here!
- No idea what transformation would be good.
- Let data choose: "Box-Cox transformation".
- Scale is that of "ladder of powers": power transformation, but 0 is log.
- SAS: `proc transreg`:

```
proc transreg;
  model boxcox(demand)=identity(usage);
```

Output (graph)

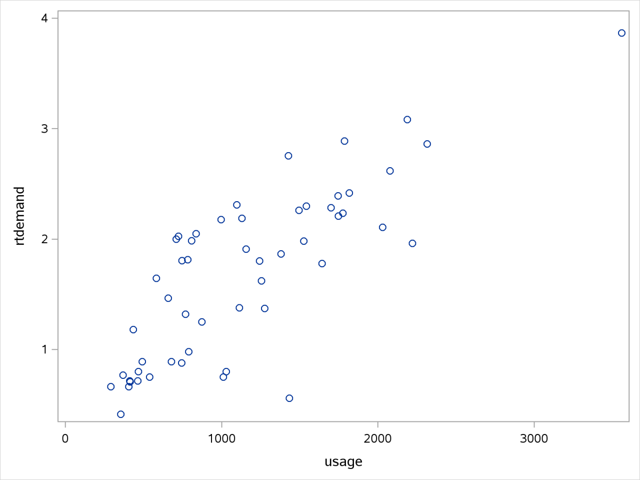


Comments

- SAS finds best transformation, here power 0.60 or so.
- Also gives you a CI for power, here 0.50 to 0.75.
- Ideal transformation should be defensible power, typically from set $\{-1, -0.5, 0, 0.5, 1, 2\}$. Here that would be power 0.5, which would be square root.
- Try that and see how it looks.
- Create another new data set by bringing in everything from old one and make a scatterplot:

```
data trans;
  set util;
  rtdemand=sqrt(demand);

proc sgplot;
  scatter x=usage y=rtdemand;
```

New scatterplot



Regression with new response variable

- Scatter plot still looks straight.
- Data set `trans` is most recently-created (default) one, so used in scatterplot above and `proc reg` below.

```
proc reg;
  model rtdemand=usage;
```

Output

The REG Procedure
Model: MODEL1
Dependent Variable: rtdemand

Root MSE	0.46404	R-Square	0.6485
Dependent Mean	1.68040	Adj R-Sq	0.6416
Coeff Var	27.61503		

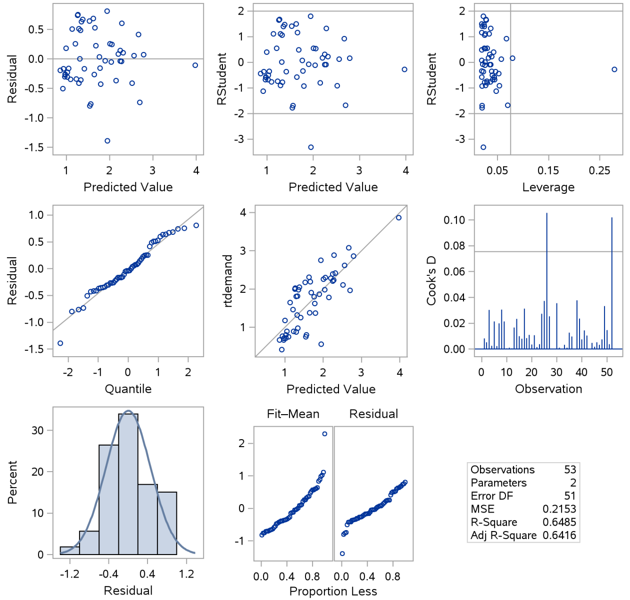
Parameter Estimates

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

Comments

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

Residual diagnostics for 2nd regression

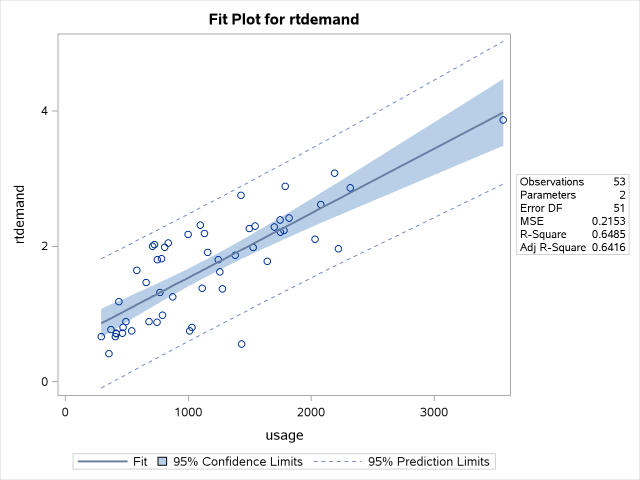


Observations	53
Parameters	2
Error DF	51
MSE	0.2153
R-Square	0.6485
Adj R-Square	0.6416

Comments

- Better. No trends, approx. constant variability.
- One mildly suspicious outlier at the bottom.
- Can trust this regression.
- Better a lower R-squared from a regression we can trust than a higher one from one we cannot.
- Look at scatterplot of `rtdemand` against `usage` with regression line on it (in graphics output from regression).

Scatterplot with fitted line



Predictions

- When we transformed the response variable, have to think carefully about predictions. Using `usage=1000`, and with R as calculator:

```
int=0.58223
slope=0.00095286
pred=int+slope*1000
pred
```

```
## [1] 1.53509
```

- It's a prediction, but of the response variable in regression, which was `rtdemand`, square root of demand.
- To predict actual demand, need to undo the transformation.
- Undoing square root is *squaring*:

```
pred^2
```

```
## [1] 2.356501
```

More predictions

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

```
usage=c(1000,2000,3000)
rt.demand=int+slope*usage
demand=rt.demand^2
demand

## [1]  2.356501  6.189895 11.839173
```

- Transformations are non-linear changes.
- Here, though the `usage` values equally spaced, predicted `demand` values are not.
- Larger gap between 2nd and 3rd than 1st and 2nd.

Section 8

Regression with categorical variables

The pig feed data

- Read in pig feed data (after tidying):

```
filename myurl url
  "http://www.utsc.utoronto.ca/~butler/c32/pigs.txt"
proc import
  datafile=myurl
  out=pigs
  dbms=dlm
  replace;
  getnames=yes;
  delimiter=' ';

proc print;
```

The data

Obs	pig	feed	weight
1	1	feed1	60.8
2	2	feed1	57
3	3	feed1	65
4	4	feed1	58.6
5	5	feed1	61.7
6	1	feed2	68.7
7	2	feed2	67.7
8	3	feed2	74
9	4	feed2	66.3
10	5	feed2	69.8
11	1	feed3	92.6
12	2	feed3	92.1
13	3	feed3	90.2
14	4	feed3	96.5
15	5	feed3	99.1
16	1	feed4	87.9
17	2	feed4	84.2
18	3	feed4	83.1
19	4	feed4	85.7
20	5	feed4	90.3

`proc glm`

- Regression with categorical variables goes with `proc glm`, not `proc reg`.
- Declare all the categorical variables with `class` before fitting model.

```
proc glm;
  class feed;
  model weight=feed / solution;
```

Output

The GLM Procedure

Dependent Variable: weight

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	3	3520.525500	1173.508500	119.14	<.0001
Error	16	157.600000	9.850000		
Corrected Total	19	3678.125500			

Source	DF	Type III SS	Mean Square	F Value	Pr > F
feed	3	3520.525500	1173.508500	119.14	<.0001

Parameter		Estimate		Standard Error	t Value	Pr > |t|
Intercept		86.24000000	B	1.40356688	61.44	<.0001
feed	feed1	-25.62000000	B	1.98494332	-12.91	<.0001
feed	feed2	-16.94000000	B	1.98494332	-8.53	<.0001
feed	feed3	7.86000000	B	1.98494332	3.96	0.0011
feed	feed4	0.00000000	B	.	.	.

Comments

- SAS gives the ANOVA-type output for `proc glm`.
- The F-statistic is the same as R's.
- *Last* feed `feed4` used as baseline, all else compared to that. Weight gain for feed 3 is highest, feed 1 is lowest.

The crickets

- On Assignment 8.5 (at current writing) we explored "crickets" data set.
- Male crickets rub their wings together to produce a chirping sound.
- Rate of chirping, called "pulse rate", depends on species and possibly on temperature.
- Sample of crickets of two species' pulse rates measured; temperature also recorded.
- Does pulse rate differ for species, especially when temperature accounted for?

The crickets, in SAS

- I saved the tidied data set from Assignment 8.5:

```
filename myurl url
  "http://www.utsc.utoronto.ca/~butler/c32/crickets2
proc import
  datafile=myurl
  out=crickets
  dbms=csv
  replace;
  getnames=yes;

proc print data=crickets(obs=20);
```

The data, some

Obs	species	temperature	pulse_rate
1	exclamationis	20.8	67.9
2	exclamationis	20.8	65.1
3	exclamationis	24	77.3
4	exclamationis	24	78.7
5	exclamationis	24	79.4
6	exclamationis	24	80.4
7	exclamationis	26.2	85.8
8	exclamationis	26.2	86.6
9	exclamationis	26.2	87.5
10	exclamationis	26.2	89.1
11	exclamationis	28.4	98.6
12	exclamationis	29	100.8
13	exclamationis	30.4	99.3
14	exclamationis	30.4	101.7
15	niveus	17.2	44.3
16	niveus	18.3	47.2
17	niveus	18.3	47.6
18	niveus	18.3	49.6
19	niveus	18.9	50.3
20	niveus	18.9	51.8

Predict pulse rate from other variables

- ... using `proc glm` since `species` is categorical:

```
proc glm;
  class species;
  model pulse_rate=temperature species / solution;
```

Output part 1

Something affects pulse rate:

The GLM Procedure

Dependent Variable: pulse_rate

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	8492.824970	4246.412485	1330.72	<.0001
Error	28	89.349869	3.191067		
Corrected Total	30	8582.174839			

Output part 2

For what, look at type III tests:

```
Source                      DF     Type III SS     Mean Square    F Value    Pr > F

temperature                  1     4376.082568     4376.082568    1371.35    <.0001
species                      1      598.003953      598.003953     187.40    <.0001
```

It's both temperature and species (removing either would be a mistake).

See over *how* temperature and species affect pulse rate.

Output part 3: parameter estimates

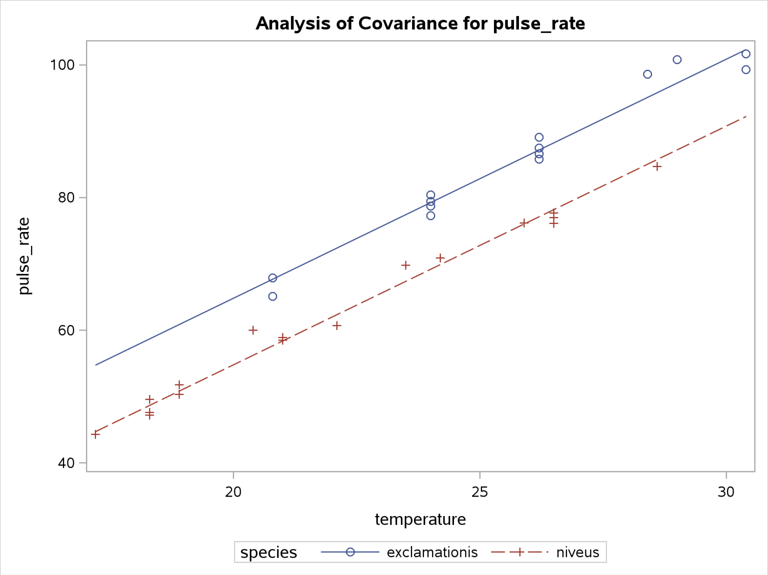
Parameter		Estimate		Standard Error	t Value	Pr > |t|
Intercept		-17.27619743	B	2.19552853	-7.87	<.0001
temperature		3.60275287		0.09728809	37.03	<.0001
species	exclamationis	10.06529123	B	0.73526224	13.69	<.0001
species	niveus	0.00000000	B	.	.	.

Slope for temperature is 3.6: increasing temperature by 1 degree increases pulse rate by 3.6.

Niveus now used as baseline; a cricket being *exclamationis* instead increases pulse rate by 10, for any fixed temperature.

...and a graph

This comes from `proc glm` output:



Conclusions from SAS

- Both temperature and species significantly affect pulse rate.
- As temperature goes up, pulse rate goes up (for both species).
- *Exclamationis* has a pulse rate about 10 higher than *niveus* for all temperatures.
- Data suggests that model fitted, with parallel straight lines for each species, fits well.

Section 9

Dates and times

Reading dates in SAS

- Consider this data file:

```
date,status,dunno
2011-08-03,hello,August 3 2011
2011-11-15,still here,November 15 2011
2012-02-01,goodbye,February 1 2012
```

- `proc import` will make guesses about what you have, *as long as it is consistently formatted*:

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

What that reads in

```
Obs          date     status             dunno

  1    2011-08-03     hello        03AUG11:00:00:00
  2    2011-11-15     still here   15NOV11:00:00:00
  3    2012-02-01     goodbye      01FEB12:00:00:00
```

- SAS made a guess at the dates with month names in them: it guessed they were "datetimes", which explains the mysterious midnight times.
- Not clear from looking at this whether the column `date` actually *is* dates, or just text. To check, look in Log tab for the word `format`. I got:

  ```
  format date yymmdd10. ;
  format status $10. ;
  format dunno datetime. ;
  ```

- This tells you how the values have been displayed: the `date` is indeed a date with year first, and `dunno` is indeed a "datetime".

Display formatted dates in SAS

- If you don't like how your dates are displayed, you can change it, eg.:

```
proc print;
  format date mmddyy8.;
```

```
Obs        date    status            dunno

  1    08/03/11    hello         03AUG11:00:00:00
  2    11/15/11    still here    15NOV11:00:00:00
  3    02/01/12    goodbye       01FEB12:00:00:00
```

- Even though dates were originally in ISO year-month-day format, they can be output in any format (eg. US format here).
- SAS can input/output dates in many formats; you just have to find name of one you need. See eg. `https://v8doc.sas.com/sashtml/lrcon/zenid-63.htm`.

Constructing dates from year, month and day

- You might have separate columns containing year, month, day.
- Strategy (both R and SAS): glue them together into something that can be recognized as date:

```
filename myurl url
  "http://www.utsc.utoronto.ca/~butler/c32/pieces.tx
proc import
  datafile=myurl
    dbms=dlm
    out=pieces
    replace;
  delimiter=' ';
  getnames=yes;

data makedates;
  set pieces;
  sasdate=mdy(month,day,year);
```

The resulting data set

```
proc print;
  format sasdate yymmdd10.;
```

Obs	year	month	day	sasdate
1	1970	1	1	1970-01-01
2	2007	9	4	2007-09-04
3	1940	4	15	1940-04-15

The `format` displays the dates in ISO format. If you omit it:

```
proc print;
```

Obs	year	month	day	sasdate
1	1970	1	1	3653
2	2007	9	4	17413
3	1940	4	15	-7200

you get *days since Jan 1, 1960*.

Month names

- If your data file contains month *names*, may need to organize as text that SAS can read as a date. Example, monthly sales of a product:

```
year,month,sales
2011,November,102
2011,December,131
2012,January,97
2012,February,108
2012,March,113
```

- Read data as is, see how it came out.

Reading in

- Try it:

```
filename myurl url
  "http://www.utsc.utoronto.ca/~butler/c32/monthly.c
proc import
  datafile=myurl
  out=sales1
  dbms=csv
  replace;
  getnames=yes;

proc print;
```

- Still have separate year and month, so need to combine ourselves:

```
Obs          year     month            sales

 1           2011     November          102
 2           2011     December          131
 3           2012     January            97
 4           2012     February          108
```

Making dates of these

- Two-step process:
  - construct a piece of text that looks like a date (`cat`)
  - turn that into a genuine date (`input`)
- All done in a `data` step (creating new variables)
- Have to invent day-of-month; here pretend 16th of month.

Making it work

- Something like this. Can use any format for output, but in `input` must use format respecting the text you made:

```
data sales2;
  set sales1;
  date_text=cat('16 ',month,' ',year);
  real_date=input(date_text,anydtdte20.);

proc print;
  var sales date_text real_date;
  format real_date yymmdd10.;
```

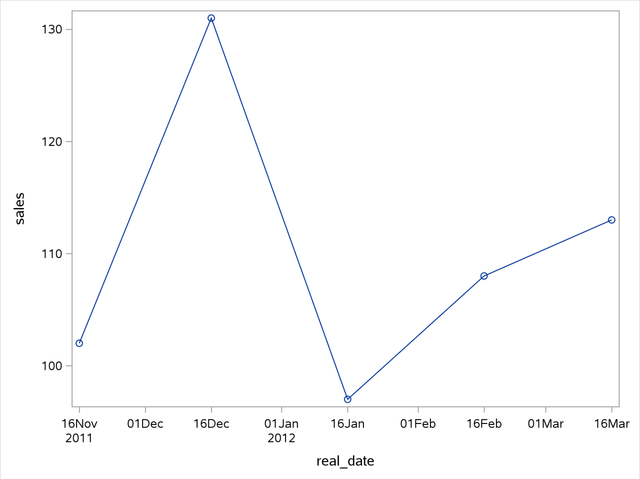
- Results:

```
Obs          sales      date_text             real_date

 1             102    16 November 2011      2011-11-16
 2             131    16 December 2011      2011-12-16
 3              97    16 January  2012      2012-01-16
 4             108    16 February 2012      2012-02-16
 5             113    16 March    2012      2012-03-16
```

Plotting sales against time

Now that we have real dates, this is easy. `series` joins points by lines:

```
proc sgplot;
  series x=real_date y=sales / markers;
  format real_date monyy7.;
```



Extracting things in SAS

Recall:

```
proc print data=dates;
```

Obs	date	status	dunno
1	2011-08-03	hello	03AUG11:00:00:00
2	2011-11-15	still here	15NOV11:00:00:00
3	2012-02-01	goodbye	01FEB12:00:00:00

Extract day, month, year thus:

```
data moredates;
  set dates;
  d=day(date);
  m=month(date);
  y=year(date);
```

The results

```
proc print;
```

Obs	date	status	dunno	d	m	y
1	2011-08-03	hello	03AUG11:00:00:00	3	8	2011
2	2011-11-15	still here	15NOV11:00:00:00	15	11	2011
3	2012-02-01	goodbye	01FEB12:00:00:00	1	2	2012

Dates and times in SAS

- If it looks like a date-and-time, SAS will read it as one, for example:
- Since date-times might have spaces, delimit by something other than space!

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

Resulting data set

```
proc print;
```

```
Obs    occasion          when

 1      first      01JAN70:07:50:01
 2      second     04SEP07:15:30:00
 3      third      15APR40:06:45:10
 4      fourth     10FEB16:12:26:40
```

Constructing date-times

- SAS has function `dhms` from which we can construct date-times from pieces such as these:
- which we read in the usual way:

```
filename myurl url
  "http://www.utsc.utoronto.ca/~butler/c32/manypiece
proc import
  datafile=myurl
    dbms=dlm
    out=many
    replace;
  delimiter=' ';
  getnames=yes;
```

Creating the date-times

Start from dataset read in from file and then create what you need, throwing away original variables (not needed any more):

```
data dtm;
  set many;
  thedate=mdy(month,day,year);
  sasdt=dhms(thedate,hour,minute,second);
  keep thedate hour minute second sasdt;
```

The result

```
proc print;
```

Obs	hour	minute	second	thedate	sasdt
1	10	0	0	1362	117712800
2	13	24	30	20956	1810646670

which doesn't account for the new variables being date or date/time, or better:

```
proc print;
format thedate yymmdd10. thetime time8.
  sasdt datetime.;
```

Obs	hour	minute	second	thedate	sasdt
1	10	0	0	1963-09-24	24SEP63:10:00:00
2	13	24	30	2017-05-17	17MAY17:13:24:30

Handling date-times

- In SAS, date-times are *seconds since midnight Jan 1, 1960*.
- In R, the zero date was Jan 1, *1970*.
- Thus, subtracting date-times gives a number of seconds, which we might then have to translate into something useful. Hospital data:

```
admit,discharge
1981-12-10 22:00:00,1982-01-03 14:00:00
2014-03-07 14:00:00,2014-03-08 09:30:00
2016-08-31 21:00:00,2016-09-02 17:00:00
```

- Read in like this:

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

Create the lengths of stay

- In a new dataset, calculate the lengths of stay, converting seconds to days:

```
data hospitalstay;
  set stays;
  stay=(discharge-admit)/60/60/24;


proc print;
```

- The stay should be displayed as a decimal number, so no special treatment required. Length of stay agrees with R:

Obs	admit	discharge	stay
1	10DEC81:22:00:00	03JAN82:14:00:00	23.6667
2	07MAR14:14:00:00	08MAR14:09:30:00	0.8125
3	31AUG16:21:00:00	02SEP16:17:00:00	1.8333

Section 10

Miscellaneous stuff in SAS

SAS: More than one observation per line of data file

- Suppose you have a data file like this:

  but the data are *all* values of one variable x (so there are 12 values altogether).

- How to get *one* column called x?
- Strategy: read values in the usual way, then process.
- Here there are no variable names, so:

```
filename myurl url
  "http://www.utsc.utoronto.ca/~butler/c32/many.txt";
proc import
  datafile=myurl
    dbms=dlm out=many replace;
  delimiter=' ';
  getnames=no;
```

- Note last line, not the usual.

So far

```
proc print;
```

```
        V             V             V             V             V             V
O       A             A             A             A             A             A
b       R             R             R             R             R             R
s       1             2             3             4             5             6

1            3             4             5             6             7             7
2            8             9             3             4             8             6
```

We have six variables with names like `VAR2`, each "variable" having two values (two lines of data file).

Solution for this

Solution very like the SAS version of gather, using an array:

```
data one;
  set many;
  array x_array VAR1-VAR6;
  do i=1 to 6;
    x=x_array[i];
    output;
  end;
  keep x;
```

Did it work?

```
proc print;
```

Obs	x
1	3
2	4
3	5
4	6
5	7
6	7
7	8
8	9
9	3
10	4
11	8
12	6

Same data file as values of `x` and `y`

- Recall:
- Suppose now a value of `x` and a value of `y`, then another `x` and another `y`, and so on, so 3 is `x`, 4 is `y`, 5 is `x`, 6 is `y` and so on.
- Read in as before using `proc import` to get data set with `VAR1` through `VAR6`, then loop from 1 to *3* (3 `x-y` pairs), pulling out the right things.

Making x and y

- This code, adapted from previous:

```
data two;
  set many;
  array xy_array VAR1-VAR6;
  do i=1 to 3;
    x=xy_array[2*i-1];
    y=xy_array[2*i];
    output;
  end;
  keep x y;
```

- Tricky part: when $i = 1$, want items 1 and 2 from the array; when $i = 2$, want items 3 and 4, etc.
- Twice the value of i will give the second value we want (the one for `y`), so one less than that will give the value we want for `x`.

Did it work?

We seem to have been successful. You can check that the right values got assigned to `x` and `y` in the right order.

```
proc print;
```

Obs	x	y
1	3	4
2	5	6
3	7	7
4	8	9
5	3	4
6	8	6

Permanent data sets

- Can we read in data set *once* and not every time?
- Yes, use *this mechanism* when creating, for example pigs data:

```
filename myurl url
  "http://www.utsc.utoronto.ca/~butler/c32/pigs1.txt
libname mydata V9 '/home/ken';
proc import
  datafile=myurl
    dbms=dlm
    out=mydata.pigs1
    replace;
  delimiter=' ';
  getnames=yes;
```

- First, define a `libname` that tells SAS which folder this dataset will go in.
- Then, on `out=`, use a two-part name: the `libname`, then dataset name.

Comments

- In folder defined by `libname`, will be a file called `pigs1.sas7bdat` (!) on SAS Studio. In my case, in my main SAS Studio folder.
- Can use subfolders, using / forward slash syntax, in `libname`.
- Whenever you need to use it, add `data='/home/username/pigs1'` to a `proc` line (replacing `username` with your username, and replacing `pigs1` with your data set name).
- Closing SAS breaks connection with temporary (ie. *non*-permanent) data sets. To get those back, need to run `proc import` lines again.

`proc means` without reading in data

- Imagine we closed down SAS Studio and opened it up again. Then:

```
proc means data='/home/ken/pigs1';
```

- with output

```
                          The MEANS Procedure

Variable    N          Mean       Std Dev       Minimum       Maximum
---------------------------------------------------------------------
feed1       5    60.6200000     3.0646370    57.0000000    65.0000000
feed2       5    69.3000000     2.9266021    66.3000000    74.0000000
feed3       5    94.1000000     3.6131704    90.2000000    99.1000000
feed4       5    86.2400000     2.8962044    83.1000000    90.3000000
---------------------------------------------------------------------
```

Saving permanent data sets another way

- Can also create a new data set, using `data` step, and make *that* permanent. For example, suppose we take data set `two` from before (containing variables `x` and `y`):

```
proc print data=two;
```

```
Obs    x    y

 1     3    4
 2     5    6
 3     7    7
 4     8    9
 5     3    4
 6     8    6
```

- Then add a variable `z` to it, saving in permanent data set `three`.

```
libname mydata V9 '/home/ken';
data mydata.three; /* permanent data set to save in */
  set two; /* this has variables x and y in it */
  z=x+y;
```

The new permanent data set

- Imagine I closed down SAS Studio and opened it up again:

```
proc print data='/home/ken/three';
```

Obs	x	y	z
1	3	4	7
2	5	6	11
3	7	7	14
4	8	9	17
5	3	4	7
6	8	6	14

Why permanent data sets?

- It is a lot of work (for us) to read in data sets from file every time. I can never remember the syntax for `proc import` (I usually copy an old one).
- It can take a lot of effort to get data in the right format for analysis. Rather than do that every time, we can save a permanent data set once the dataset is in the right shape.
- For big data, we don't want to repeat the effort of reading and processing more than once. (This can take a *long* time.) Better to create one permanent dataset and use it for each of our analyses.

How does SAS know which data set to use?

Two rules:

1. Any `proc` can have `data=` on it. Tells SAS to use that data set. Can be
   - unquoted dataset name (created by `proc import` or by processing a dataset read in that way)
   - quoted data set name (permanent one on disk created as above)
2. Without `data=`, *most recently created data set*. Typically data set created by `proc import` or `data` step. Also, data set created by `out=` counts.

Does permanent data set count as "most recently created"? No, or at least not always. If unsure, use `data=`.

Embellishments to plots

- Histogram with kernel density curve
- Smooth trend on scatterplot
- Plotting several series of data
- Labelling points on plots

Use Australian athletes data

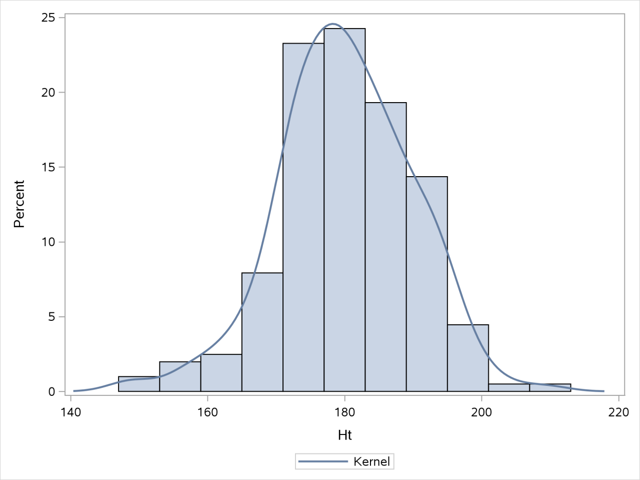
```
filename myurl url
  "http://www.utsc.utoronto.ca/~butler/c32/ais.txt";
proc import
  datafile=myurl
  dbms=dlm
  out=sports
  replace;
  delimiter='09'x;
  getnames=yes;
```

Kernel density curve on histogram

- A kernel density curve smooths out a histogram and gives sense of shape of distribution.
- `geom_density` in R on a `geom_histogram`.
- Athlete heights:

```
proc sgplot;
  histogram Ht;
  density Ht / type=kernel;
```

Histogram of heights with kernel density

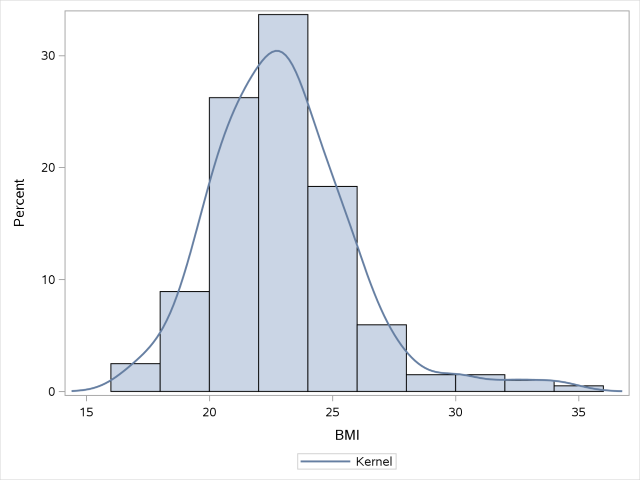


More or less symmetric.

Kernel density for BMI

```
proc sgplot;
  histogram BMI;
  density BMI / type=kernel;
```

Histogram with kernel density



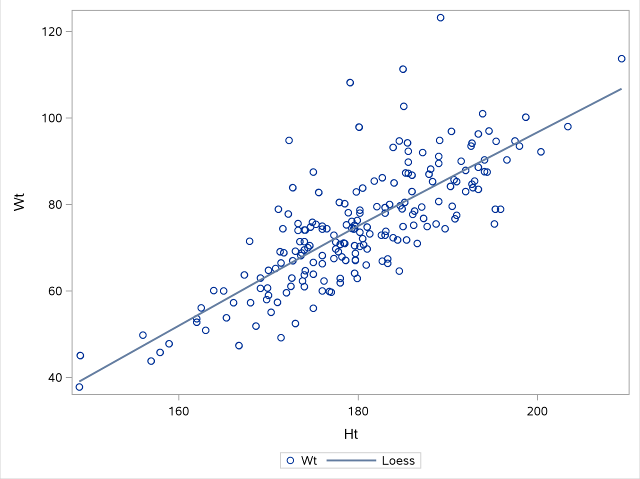
Rather more clearly skewed right.

Loess curve

- R equivalent: `geom_smooth` *without* `method="lm"`.
- Smooth curve through scatterplot called *Loess curve* in SAS: Code like this:

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

Loess curve on plot



Loess curve says this is as straight as you could wish for.

Loess curve for windmill data

- Read into SAS thus:

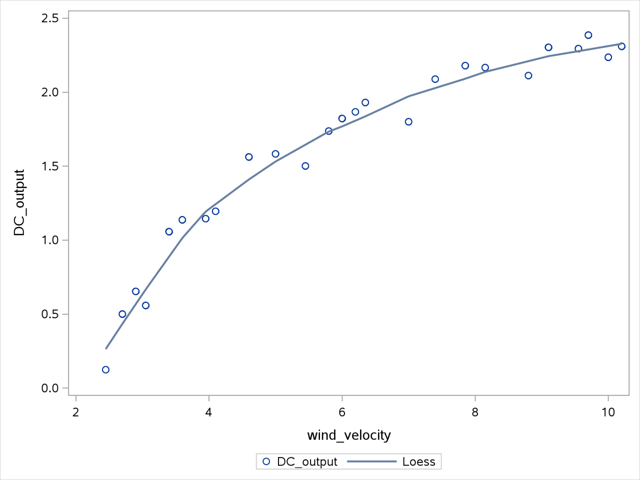
```
filename myurl url
  "http://www.utsc.utoronto.ca/~butler/c32/windmill.
proc import
  datafile=myurl
    dbms=csv
    out=windmill
    replace;
  getnames=yes;


proc means;
```

```
                               The MEANS Procedure

Variable            N            Mean         Std Dev         Minimum         Maximum
------------------------------------------------------------------------------------
wind_velocity      25       6.1320000       2.5294466       2.4500000      10.2000000
DC_output          25       1.6096000       0.6522777       0.1230000       2.3860000
------------------------------------------------------------------------------------
```

To make the scatterplot with loess curve

```
proc sgplot;
  scatter x=wind_velocity y=DC_output;
  loess x=wind_velocity y=DC_output;
```

The plot with curve



This time, relationship is definitely curved.

Multiple series on one plot: the oranges data

- Data file like this (circumferences of 5 trees each at 7 times):

```
age A B C D E
118 30 30 30 33 32
484 51 58 49 69 62
664 75 87 81 111 112
1004 108 115 125 156 167
1231 115 120 142 172 179
1372 139 142 174 203 209
1582 140 145 177 203 214
```

- Columns don't line up because the delimiter is "exactly one space", and some of the values are longer than others.
- In R, `gather` data to put x and y for plot in single columns. Here, use original columns.

Reading the data

```
filename myurl url
  "http://www.utsc.utoronto.ca/~butler/c32/oranges.txt"
proc import
  datafile=myurl
    dbms=dlm
    out=trees
    replace;
  delimiter=' ';
  getnames=yes;
```

Did it work?

```
proc print;
```

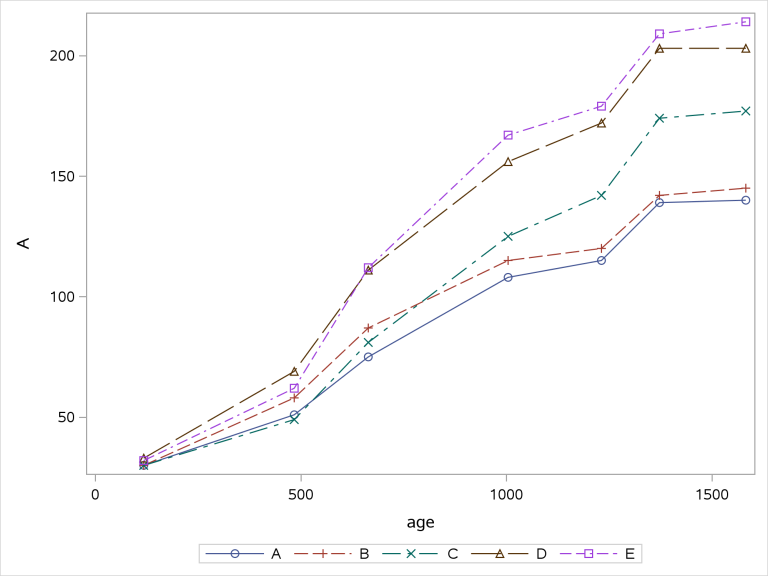
Obs	age	A	B	C	D	E
1	118	30	30	30	33	32
2	484	51	58	49	69	62
3	664	75	87	81	111	112
4	1004	108	115	125	156	167
5	1231	115	120	142	172	179
6	1372	139	142	174	203	209
7	1582	140	145	177	203	214

Multiple series

- Growth curve for *each* tree, joined by lines.
- `series` joins points by lines.
- `markers` displays actual data points too.
- Do each series one at a time.

```
proc sgplot;
  series x=age y=a / markers;
  series x=age y=b / markers;
  series x=age y=c / markers;
  series x=age y=d / markers;
  series x=age y=e / markers;
```

The growth curves



Labelling points on a plot

- Often, a data set comes with an identifier variable.
- We would like to label each point on a plot with its identifier, to see which individual is which.
- Commonly (but not only) done on scatterplot.

Example: the cars data

- 38 cars. For each:
  - Name of car (identifier)
  - Gas mileage (miles per US gallon)
  - Weight (US tons)
  - Number of cylinders in engine
  - Horsepower of engine
  - Country of origin

Reading in

`.csv` file, so:

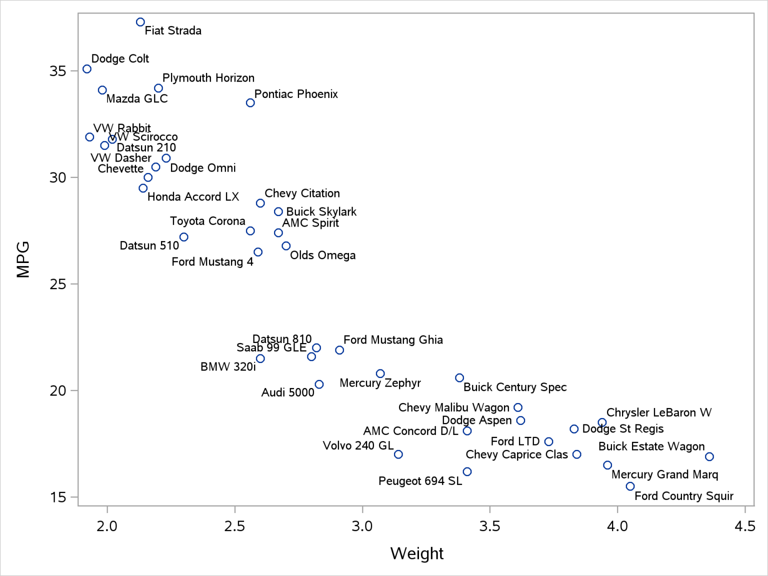
```
filename myurl url
  "http://www.utsc.utoronto.ca/~butler/c32/cars.csv";
proc import
  datafile=myurl
  dbms=csv
  out=cars
  replace;
  getnames=yes;
```

Adding labels to scatterplot

- Expect heavier car to have worse (lower) gas mileage, so make scatterplot of gas mileage (y) against weight (x).
- Want to see which car is which, so label points.
- R: `geom_text` (or `geom_text_repel`).
- The magic word is `datalabel`:

```
proc sgplot;
  scatter y=mpg x=weight / datalabel=car;
```

The plot


Fiat Strada
Dodge Colt
Plymouth Horizon
Mazda GLC
Pontiac Phoenix
VW Rabbit
VW Scirocco
Datsun 210
VW Dasher
Chevette
Dodge Omni
Honda Accord LX
Chevy Citation
Buick Skylark
Toyota Corona
AMC Spirit
Datsun 510
Olds Omega
Ford Mustang 4
Datsun 810
Saab 99 GLE
Ford Mustang Ghia
BMW 320i
Mercury Zephyr
Audi 5000
Buick Century Spec
Chevy Malibu Wagon
Chrysler LeBaron W
Dodge Aspen
Dodge St Regis
AMC Concord D/L
Ford LTD
Volvo 240 GL
Chevy Caprice Clas
Buick Estate Wagon
Mercury Grand Marq
Peugeot 694 SL
Ford Country Squir
35
30
25
20
15
MPG
2.0
2.5
3.0
3.5
4.0
4.5
Weight

Comments

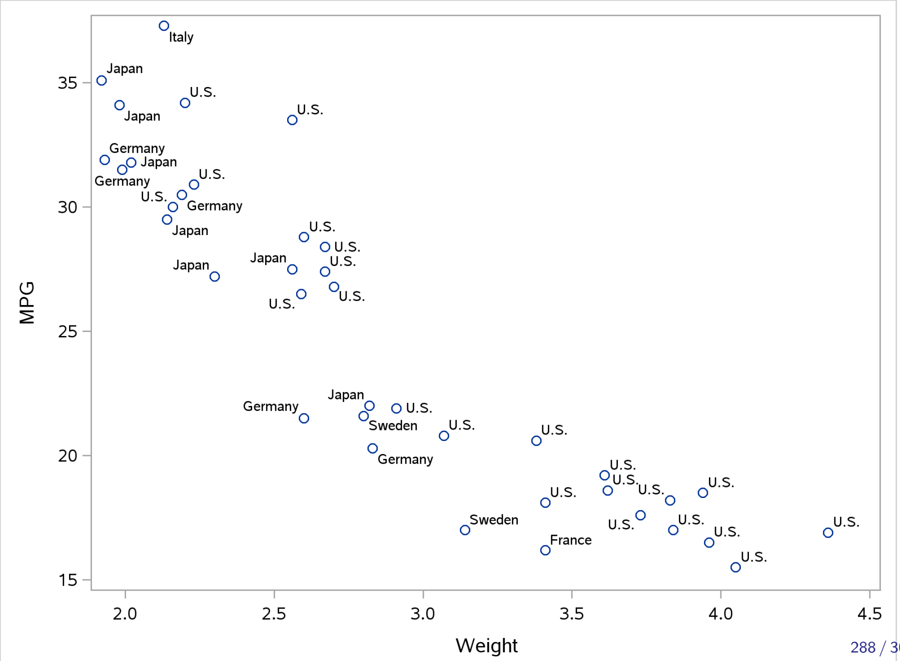
- Each car labelled with its name, either left, right, above or below, whichever makes it clearest. (Some intelligence applied to placement, like `geom_text_repel` in R.)
- Cars top left are "nimble": light in weight, good gas mileage.
- Cars bottom right are "boats": heavy, with terrible gas mileage.

Labelling by country

Same idea:

```
proc sgplot;
  scatter x=weight y=mpg / datalabel=country;
```

Labelled by country



Labelling only some of the observations

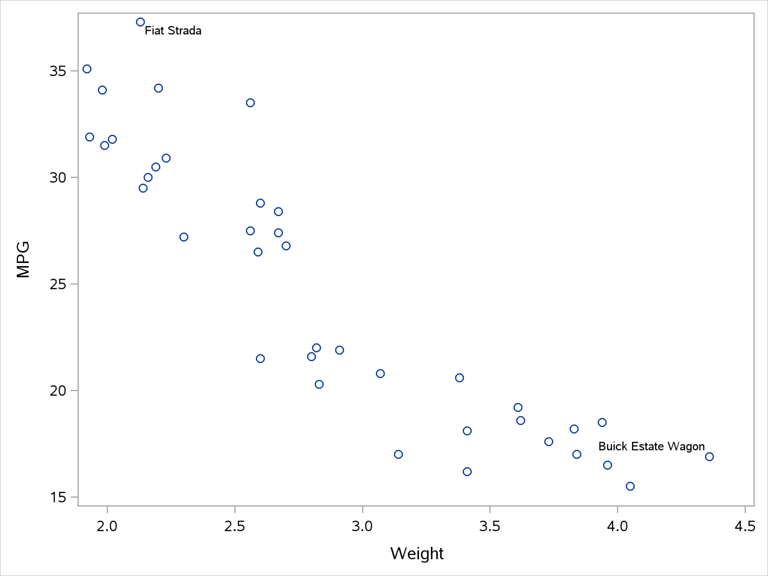
- Create a new data set with all the old variables plus a new one that contains the text to plot.
- For example, label most fuel-efficient car (#4) and heaviest car (#9).
- "Observation number" given by SAS special variable `_n_`, like `row_number` in R.
- Note the syntax: "if then do" followed by "end".

```
data cars2;
  set cars;
  if (_n_=4 or _n_=9) then do;
    newtext=car;
  end;
```

- For any cars not selected, `newtext` will be blank. Then, using the new data set that we just created:

```
proc sgplot;
  scatter x=weight y=mpg / datalabel=newtext;
```

The plot

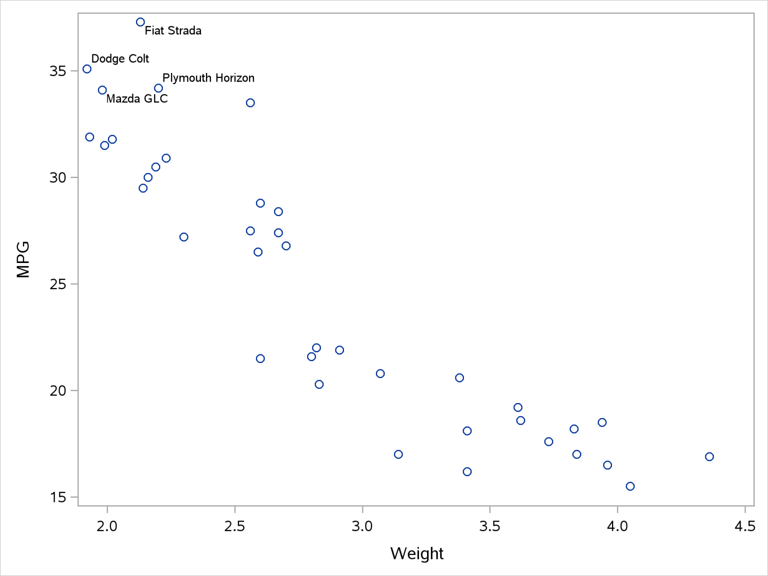


Fiat Strada
Buick Estate Wagon
35
30
25
20
15
MPG
2.0
2.5
3.0
3.5
4.0
4.5
Weight

Or label cars with `mpg` greater than 34

```
data cars3;
  set cars;
  if mpg>34 then do;
     newtext=car;
  end;


proc sgplot;
  scatter x=weight y=mpg / datalabel=newtext;
```

High-mpg cars



Section 11

Vector and matrix algebra

Vector and matrix algebra in SAS

- SAS has this through `proc iml`, which we have to learn about.

Vectors and scalars in `proc iml`

- Define vectors and scalars as below.
- To do a calculation, define the answer into a variable, and then `print` it. Note that 2 has gotten added to each element of `u`:

```
proc iml;
  k=2;
  u={2 3 6 5 7};
  ans=k+u;
  print ans;
```

```
                    ans

      4      5      8      7      9
```

Adding vectors

- Each run of `proc iml` is independent, so you have to redefine anything you want to use. This is vector addition, as before:

```
proc iml;
  u={2 3 6 5 7};
  v={1 8 3 4 2};
  ans=u+v;
  print ans;
```

```
                        ans

        3        11        9        9        9
```

Elementwise and scalar multiplication

- Elementwise vector multiplication does not work in `proc iml`.
- Scalar multiplication, though, exactly as you would expect:

```
proc iml;
  k=2;
  u={2 3 6 5 7};
  ans=k*u;
  print ans;
```

---

ans				
4	6	12	10	14

Matrices in `proc iml`

- Enter a matrix like a vector, but row by row, with a comma separating rows:

```
proc iml;
  A={1 3,2 4};
  B={5 6,7 8};
  print A;
  print B;
```

```
        A

1           3
2           4
        B

5           6
7           8
```

Adding and multiplying matrices

- These are genuine matrix addition and multiplication (no elementwise multiplication):

```
proc iml;
  A={1 3,2 4};
  B={5 6,7 8};
  ans1=A+B;
  print ans1;
  ans2=A*B;
  print ans2;
```

```
ans1

 6     9
 9    12
ans2

26    30
38    44
```

Reading a matrix in from a file 1/2

- If your matrix is in a file like this:
- read into data set as usual:

```
filename myurl url
  "http://www.utsc.utoronto.ca/~butler/c32/m.txt";
proc import
  datafile=myurl
    dbms=dlm
    out=mymatrix
    replace;
  delimiter=' ';
  getnames=no;
```

- Columns get names `VAR1`, `VAR2`, etc.

Reading a matrix in from a file 2/2

- and then use in `proc iml` thus:

```
proc iml;
  use mymatrix;
  read all var {VAR1 VAR2} into M;
  v={1,3};
  ans=M*v;
  print ans;
```

```
ans

   37
   29
   21
```

Solve a system of equations

- Suppose we wish to solve, for x and y:

$$\begin{aligned} x + 3y &= 1 \\ 2x + 4y &= 2 \end{aligned}$$

- Can be done with matrix algebra by defining

$$A = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}, w = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

- Then solve $Az = w$ as $z = A^{-1}w$.
- Thus, strategy is to find inverse *first*.

Code and result

```
proc iml;
  A={1 3,2 4};
  w={1,2};
  Ainv=inv(A);
  print Ainv;
  ans=Ainv*w;
  print ans;
```

```
     Ainv

    -2       1.5
     1      -0.5
     ans

          1
          0
```

Thus solution is $x = 1, y = 0$. These solve original equations.

Row and column vectors in `proc iml`

- Without commas gives a *row* vector in `proc iml`.
- *With* commas gives a column vector:

```
proc iml;
  r={1 2 3};
  c={4,5,6};
  print r;
  print c;
```

```
              r

     1        2        3
              c

              4
              5
              6
```

Inner product

- Make sure both vectors are *column* vectors, and then matrix-multiply the transpose of the first (a row vector) by the second:

```
proc iml;
  a={1,2,3};
  b={4,5,6};
  ans=t(a)*b;
  print ans;
```

```
                    ans

                        32
```