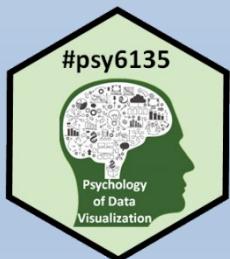
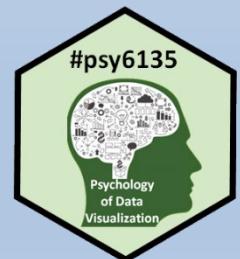


# ggplot2: Going further in the tidyverse



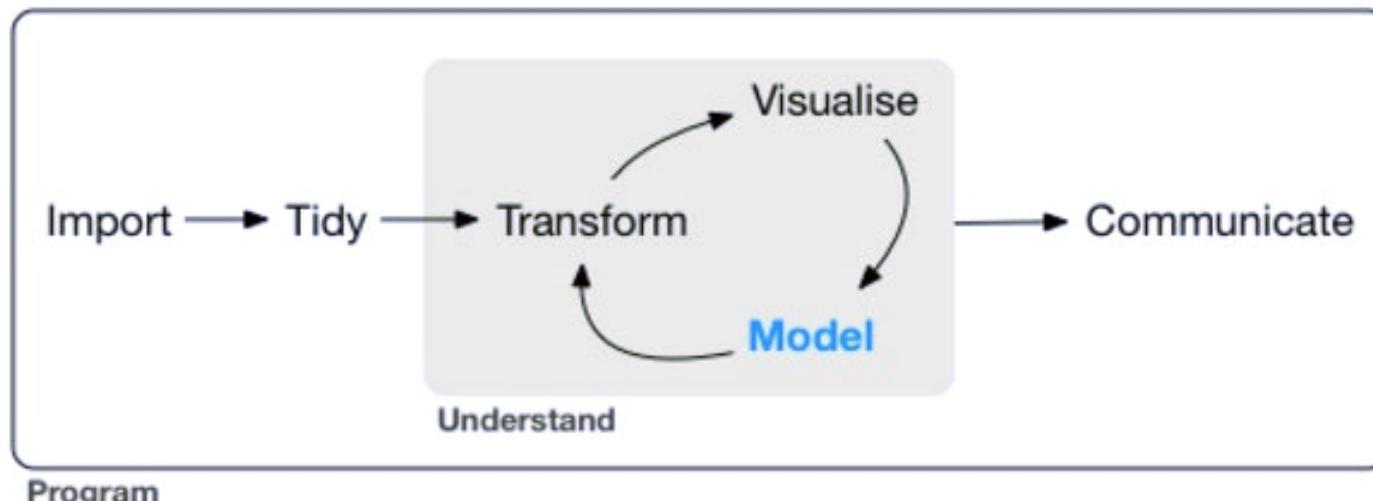
Michael Friendly  
Psych 6135

<https://friendly.github.io/6135/>



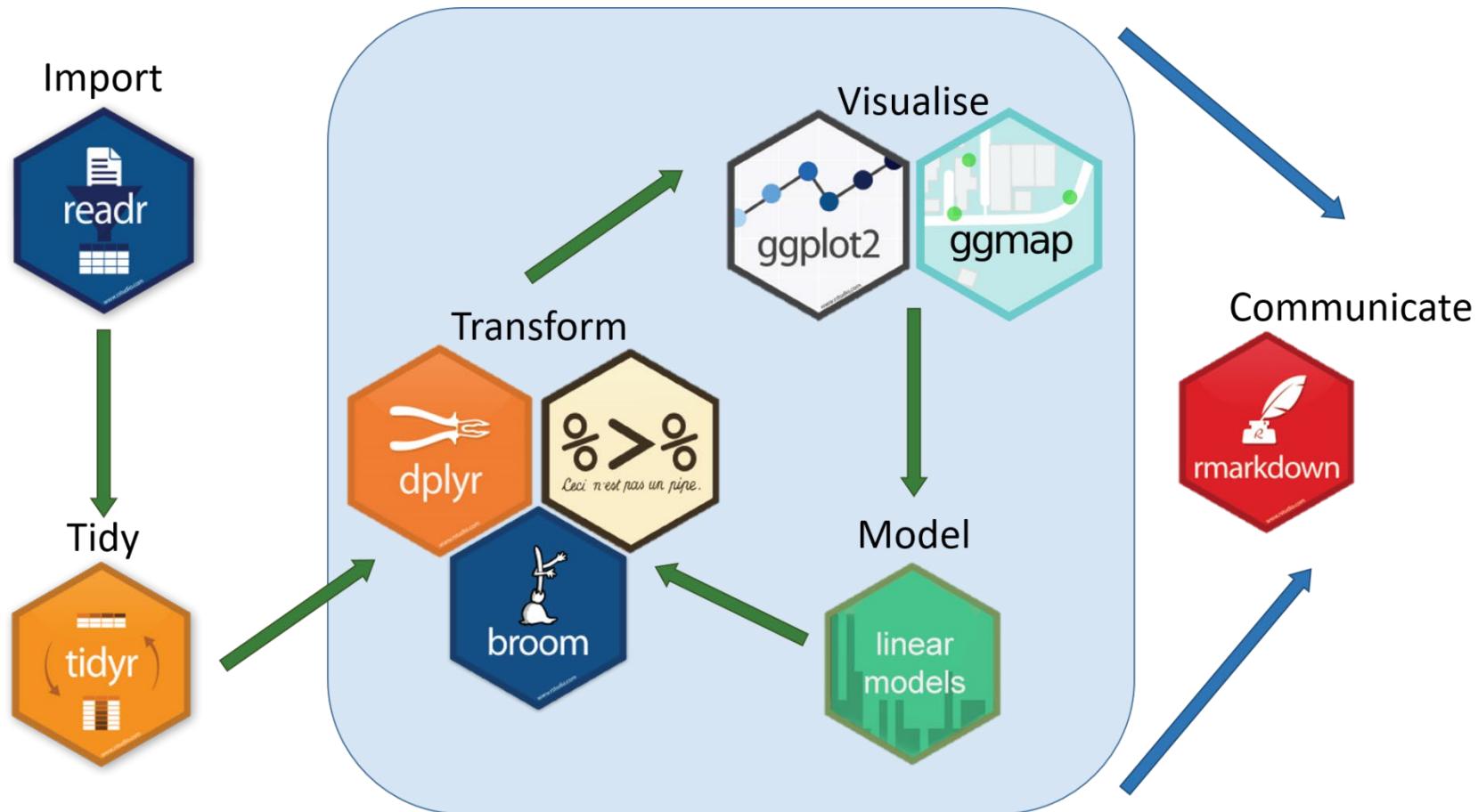
# A larger view: Data science

- Data science treats statistics & data visualization as parts of a larger process
  - Data import: text files, data bases, web scraping, ...
  - Data cleaning → “tidy data”
  - Model building & visualization
  - Reproducible report writing

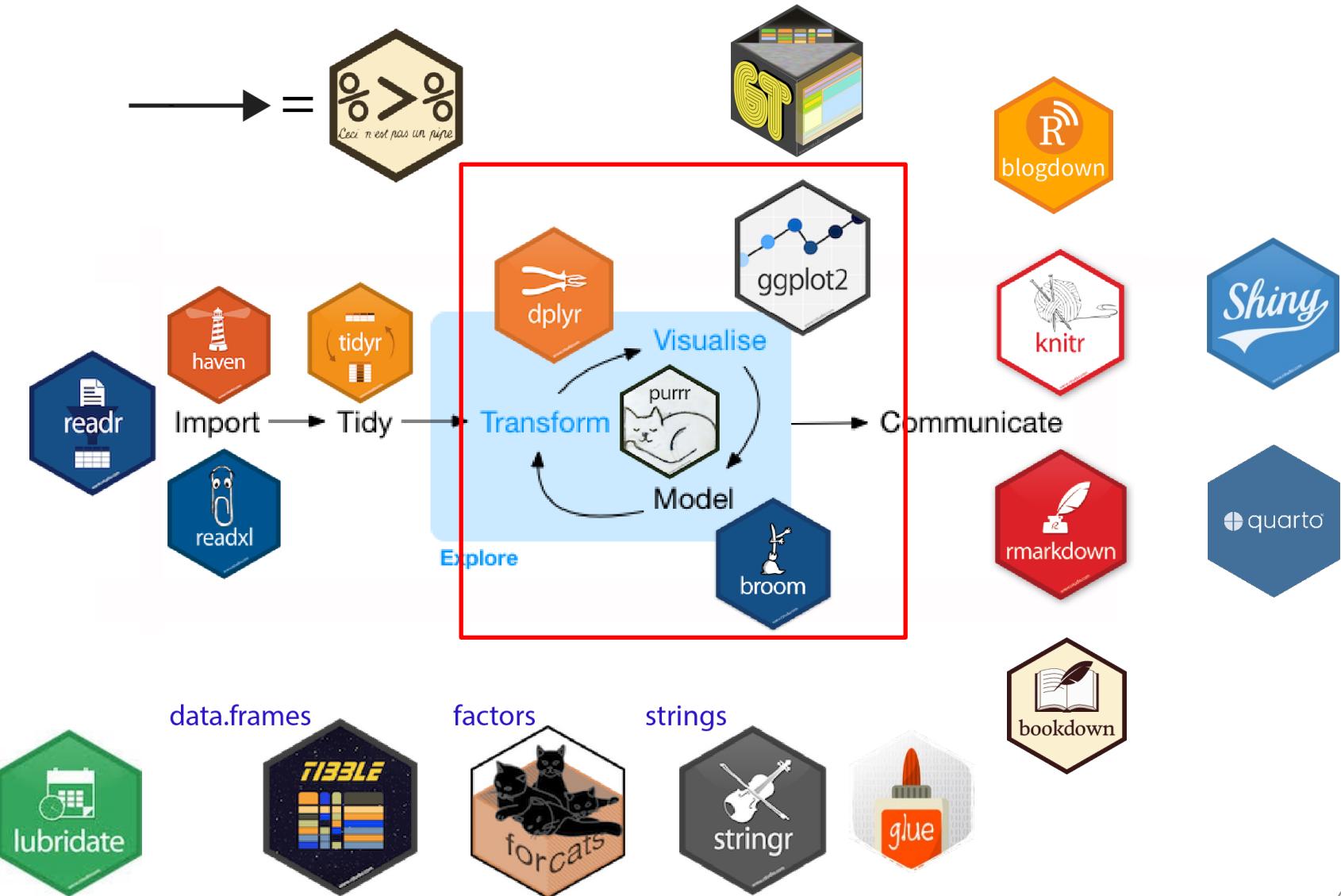




# The tidyverse of R packages



# The tidyverse expands

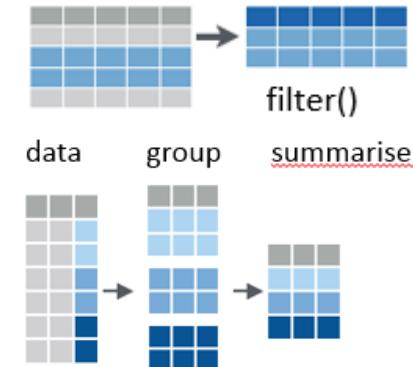


# Topics

- Data import / export
- Data wrangling: getting your data into shape



- dplyr & tidyr
- pipes: %>%, now: |>
- grouping & summarizing
- Example: NASA data on solar radiation

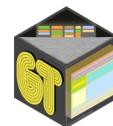


- Working with models: broom

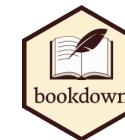


- Example: gapminder data

- Nice tables in R



- Writing & publishing



# Ready for some heavy lifting?



New ideas, ways of thinking & working are on the table!

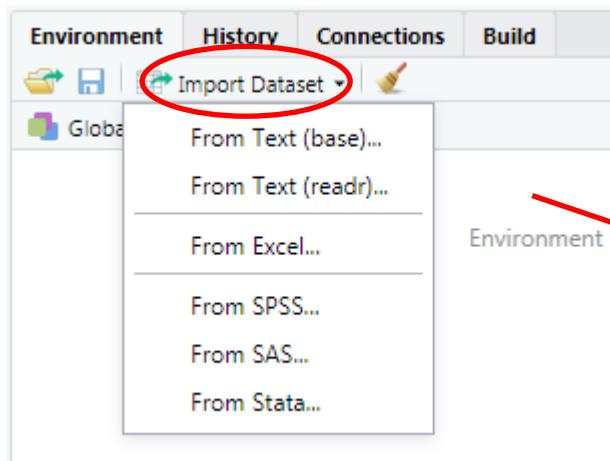
# Data Import / Export

- The `readr` package is the modern, tidy way to import and export data
  - Tabular data:
    - comma delimited (`read.csv`)
    - any other delimiters (“;” = `read.csv2`; <tab> = `read_tsv`)
  - Data types:
    - specify column types or let functions guess
- Other data formats



package	Data types
haven	SAS, SPSS, Stata
readxl	Excel files (.xls and .xlsx)
DBI	Databases (SQL, ...)
rvest	HTML (web scraping)

# Data Import: RStudio



file:

The 'Import Text Data' dialog box is open. The 'File/URL:' field contains the path 'C:/Users/friendly/Dropbox/Documents/6135/R/drugs.txt'. Below it is a 'Data Preview' table with the following data:

subject (character)	drug1 (double)	drug2 (double)	drug3 (double)	drug4 (double)
subj1	20	34	38	44
subj2	16	28	30	34
subj3	14	28	26	30
subj4	18	20	24	30
subj5	10	18	14	22

options:

The 'Import Options' dialog box is open. It includes fields for 'Name:' (set to 'drugs'), 'First Row as Names' (checked), 'Delimiter:' (set to 'Whitespace'), 'Escape:' (set to 'None'), 'Quotes:' (set to 'Default'), 'Comment:' (set to 'Default'), 'Locale:' (button), 'NA:' (set to 'Default'), and checkboxes for 'Trim Spaces' and 'Open Data Viewer'. A red circle highlights the 'Delimiter:' dropdown.

code:

The 'Code Preview' area displays the R code used to import the data:

```
library(readr)
drugs <- read_table2("R/drugs.txt")
View(drugs)
```

copy code to your  
script

# Data transformation tools

Some common data types can be messy when imported. Tidy tools are there to help

dates/times	lubridate	read dates/times in various formats; extract components	
factors	forcats	Change order of levels, drop levels, combine levels	
strings	stringr glue	detect matches, subset, replace interpolate values into strings	 
Clean-up	janitor	Clean up variable names, find duplicate records	



# Janitor

Unwanted header

Possibly duplicated rows

Illegal variable names

Rows / cols with no data

1	A	B	C	D	E	F	G	H	I	J	K	L
2	First Name	Last Name	Employee Status	Subject	Hire Date	% Allocated	Full time?	do not edit! --->	Certification	Certification	Active?	
3	Jason	Bourne	Teacher	PE	39690	75%	Yes		Physical ed	Theater	YES	
4	Jason	Bourne	Teacher	Drafting	1/14/2019	25%	Yes		Physical ed	Theater	YES	
5	Alicia	Keys	Teacher	Music	8/15/2001	100%	Yes		Instr. music	Vocal music	YES	
6	Ada	Lovelace	Teacher	#REF!	38572	100%	Yes		PENDING	Computers	YES	
7	Desus	Nice	Administration	Dean	2/25/2017	100%	Yes		PENDING		YES	
8	Chien-Shiung	Wu	Teacher	Physics	11037	50%	Yes		Science 6-12	Physics	YES	
9	Chien-Shiung	Wu	Teacher	Chemistry	11037	50%	Yes		Science 6-12	Physics	YES	
10												
11	James	Joyce	Teacher	English	9/20/1999	50%	No		English 6-12	YES		
12	Hedy	Lamarr	Teacher	Science	27919	50%	No		PENDING		YES	
13	Carlos	Boozer	Coach	Basketball	42221	#N/A	No		Physical ed		YES	
14	Young	Boozer	Coach		34700	#N/A	No			Political sci.	YES	
15	Micheal	Larsen	Teacher	English	40071	80%	No		Vocal music	English	YES	
16												

library(janitor)

Data <- read\_excel(dirty.xlsx) |> clean\_names() |> remove\_empty() |> ...



# lubridate: Dates & times

## PARSE DATE-TIMES (Convert strings or numbers to date-times)

1. Identify the order of the year (**y**), month (**m**), day (**d**), hour (**h**), minute (**m**) and second (**s**) elements in your data.
2. Use the function below whose name replicates the order. Each accepts a wide variety of input formats.

2017-11-28T14:02:00

**ymd\_hms()**, **ymd\_hm()**, **ymd\_h()**.  
`ymd_hms("2017-11-28T14:02:00")`

2017-22-12 10:00:00

**ydm\_hms()**, **ydm\_hm()**, **ydm\_h()**.  
`ydm_hms("2017-22-12 10:00:00")`

11/28/2017 1:02:03

**mdy\_hms()**, **mdy\_hm()**, **mdy\_h()**.  
`mdy_hms("11/28/2017 1:02:03")`

1 Jan 2017 23:59:59

**dmy\_hms()**, **dmy\_hm()**, **dmy\_h()**.  
`dmy_hms("1 Jan 2017 23:59:59")`

20170131

**ymd()**, **ydm()**.  
`ymd(20170131)`

July 4th, 2000

**mdy()**, **myd()**.  
`mdy("July 4th, 2000")`

4th of July '99

**dmy()**, **dym()**.  
`dmy("4th of July '99")`

2001: Q3

**yq()** Q for quarter.  
`yq("2001: Q3")`

2:01

**hms::hms()** Also `lubridate::hms()`, **hm()** and **ms()**, which return periods.\* `hms::hms(sec = 0, min = 1, hours = 2)`

## # parse dates in various formats

```
ymd("20210604")
#> [1] "2021-06-04"
mdy("06-04-2021")
#> [1] "2021-06-04"
dmy("04/06/2021")
#> [1] "2021-06-04"
```

## # extract date components

```
minard_bday <- ymd("1781-03-27")
year(minard_bday)
#> [1] 1781
month.name[month(minard_bday)]
#> [1] "March"
```

## # date arithmetic: how old is Minard?

```
year(today()) - year(minard_bday)
#> [1] 243
```



# stringr: Manipulating strings

## Detect Matches



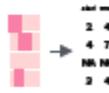
**str\_detect(string, pattern)** Detect the presence of a pattern match in a string.  
`str_detect(fruit, "a")`



**str\_which(string, pattern)** Find the indexes of strings that contain a pattern match.  
`str_which(fruit, "a")`



**str\_count(string, pattern)** Count the number of matches in a string.  
`str_count(fruit, "a")`

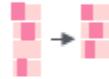


**str\_locate(string, pattern)** Locate the positions of pattern matches in a string. Also `str_locate_all`.  
`str_locate(fruit, "a")`

## Subset Strings



**str\_sub(string, start = 1L, end = -1L)** Extract substrings from a character vector.  
`str_sub(fruit, 1, 3); str_sub(fruit, -2)`



**str\_subset(string, pattern)** Return only the strings that contain a pattern match.  
`str_subset(fruit, "b")`

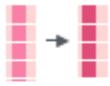


**str\_extract(string, pattern)** Return the first pattern match found in each string, as a vector. Also `str_extract_all` to return every pattern match.  
`str_extract(fruit, "[aeiou]")`

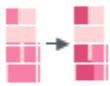


**str\_match(string, pattern)** Return the first pattern match found in each string, as a matrix with a column for each () group in pattern. Also `str_match_all`.  
`str_match(sentences, "(a|the) ([^ ]+)")`

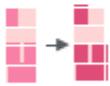
## Mutate Strings



**str\_sub()** <- value. Replace substrings by identifying the substrings with `str_sub()` and assigning into the results.  
`str_sub(fruit, 1, 3) <- "str"`



**str\_replace(string, pattern, replacement)** Replace the first matched pattern in each string.  
`str_replace(fruit, "a", "-")`



**str\_replace\_all(string, pattern, replacement)** Replace all matched patterns in each string.  
`str_replace_all(fruit, "a", "-")`

A STRING  
↓  
a string

**str\_to\_lower(string, locale = "en")** Convert strings to lower case.  
`str_to_lower(sentences)`

a string  
↓  
A STRING

**str\_to\_upper(string, locale = "en")** Convert strings to upper case.  
`str_to_upper(sentences)`

## Join and Split



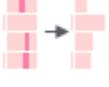
**str\_c(..., sep = "", collapse = NULL)** Join multiple strings into a single string.  
`str_c(letters, LETTERS)`



**str\_c(..., sep = "", collapse = "")** Collapse a vector of strings into a single string.  
`str_c(letters, collapse = "")`



**str\_dup(string, times)** Repeat strings times times.  
`str_dup(fruit, times = 2)`



**str\_split\_fixed(string, pattern, n)** Split a vector of strings into a matrix of substrings (splitting at occurrences of a pattern match). Also `str_split` to return a list of substrings.  
`str_split_fixed(fruit, " ", n=2)`



**str\_glue(..., .sep = "", .envir = parent.frame())** Create a string from strings and {expressions} to evaluate.  
`str_glue("Pi is {pi}")`



# glue: Interpolating strings

A common problem: Interpolating the [values](#) of variables into a text [string](#).  
Glue does this by embedding R expressions in {curly braces} which are evaluated and inserted into the argument string.

```
library(glue)
name <- "Michael"; food <- "pizza"
glue('My name is {name}. I like {food}.')
#> My name is Michael I like pizza.
```

`glue_data()` is useful with pipes and data frames

```
head(mtcars,4) %>%
  glue_data("{rownames(.)} has {hp} hp & {cyl} cylinders")
#> Mazda RX4 has 110 hp & 6 cylinders
#> Mazda RX4 Wag has 110 hp & 6 cylinders
#> Datsun 710 has 93 hp & 4 cylinders
#> Hornet 4 Drive has 110 hp & 6 cylinders
```

NB: `paste()` is the base-R version. `glue()` is much more powerful



# forcats: Working with factors

R represents categorical variables as **factors**, useful for analysis (e.g., ANOVA)  
In graphics, we often want to **recode** levels or **reorder** them

## Factors

R represents categorical data with factors. A **factor** is an integer vector with a **levels** attribute that stores a set of mappings between integers and categorical values. When you view a factor, R displays not the integers, but the values associated with them.

a	1=a
c	2=b
b	3=c
a	1=a

*Create a factor with factor()*  
`factor(x = character(), levels, labels = levels, exclude = NA, ordered = is.ordered(x), nmax = NA)` Convert a vector to a factor. Also **as\_factor**.  
`f <- factor(c("a", "c", "b", "a"), levels = c("a", "b", "c"))`

a	1=a
c	2=b
b	3=c
a	1=a

*Return its levels with levels()*  
`levels(x)` Return/set the levels of a factor. `levels(f); levels(f) <- c("x","y","z")`

*Use unclass() to see its structure*

## Inspect Factors

a	1=a
c	2=b
b	3=c
a	1=a

**fct\_count(f, sort = FALSE)**  
Count the number of values with each level. `fct_count(f)`

## Change the order of levels

a	1=a
c	2=b
b	3=c
a	1=a

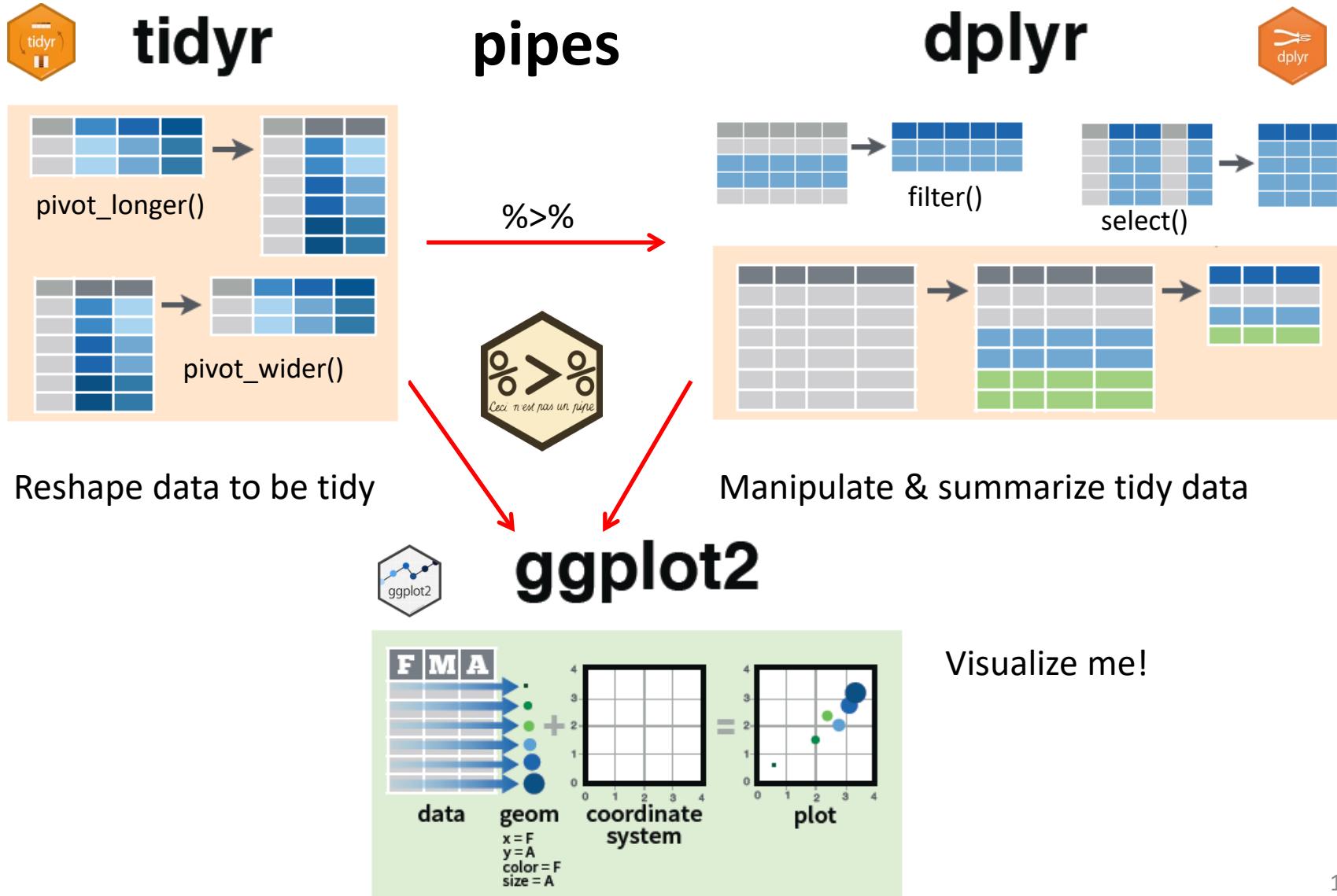
c	1=a
c	2=c
a	3=b

b	1=a
a	2=b

a	1=a
b	2=b
c	3=c

a	1=c
b	2=b
c	3=a

# Tidy tools: overview

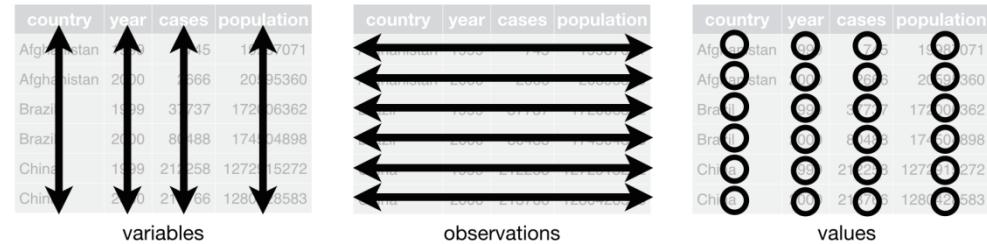


# Data wrangling with dplyr & tidyr

## What is Tidy Data?

A dataset is said to be tidy if:

- observations are in **rows**
- variables are in **columns**
- each value is in its own **cell**.



A “messy” dataset: Survey of income by religion from Pew Research

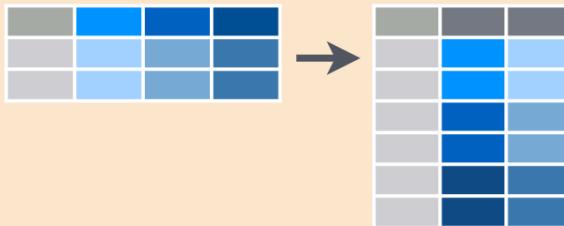
- Values of **income** are in separate columns, not one variable
- Column headers are **values**, not variable names
- Cell values are frequencies--- **implicit**, not explicit

religion	<\$10k	\$10-20k	\$20-30k	\$30-40k	\$40-50k	\$50-75k
Agnostic	27	34	60	81	76	137
Atheist	12	27	37	52	35	70
Buddhist	27	21	30	34	33	58
Catholic	418	617	732	670	638	1116

This organization is easy in Excel  
But, this makes data analysis and graphing hard

# Tidy operations

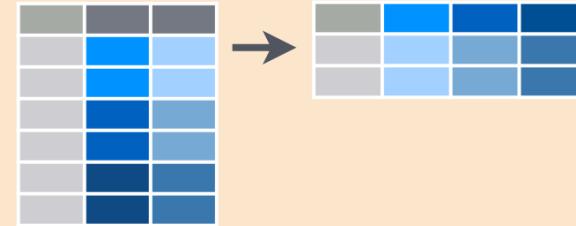
gather(): Reshape wide to long  
synonym: **tidyr::pivot\_longer()**



**tidyr::gather(cases, "year", "n", 2:4)**

Gather columns into rows.

spread(): Reshape long to wide  
synonym: **tidyr::pivot\_wider()**



**tidyr::spread(pollution, size, amount)**

Spread rows into columns.



**tidyr::separate(storms, date, c("y", "m", "d"))**

Separate one column into several.



**tidyr::unite(data, col, ..., sep)**

Unite several columns into one.

Separate parts of a value into several variables

Join related variables into one

# Tidying: reshaping wide to long

We can tidy the data by reshaping from wide to long format using `tidyverse::gather()` or `pivot_longer()`

```
> pew <- read.delim(  
  file = "http://stat405.had.co.nz/data/pew.txt",  
  header = TRUE,  
  stringsAsFactors = FALSE, check.names = FALSE)  
  
> (pew1 <- pew[1:4, 1:6]) # small subset
```

	religion	\$<10k	\$10-20k	\$20-30k	\$30-40k	\$40-50k
1	Agnostic	27	34	60	81	76
2	Atheist	12	27	37	52	35
3	Buddhist	27	21	30	34	33
4	Catholic	418	617	732	670	638

key      value      columns

```
>library(tidyverse)  
> gather(pew1, "income", "frequency", 2:6)
```

	religion	income	frequency
1	Agnostic	<\$10k	27
2	Atheist	<\$10k	12
3	Buddhist	<\$10k	27
4	Catholic	<\$10k	418
5	Agnostic	\$10-20k	34
6	Atheist	\$10-20k	27
7	Buddhist	\$10-20k	21
8	Catholic	\$10-20k	617
9	Agnostic	\$20-30k	60
10	Atheist	\$20-30k	37
11	Buddhist	\$20-30k	30
12	Catholic	\$20-30k	732
13	Agnostic	\$30-40k	81
14	Atheist	\$30-40k	52
15	Buddhist	\$30-40k	34
16	Catholic	\$30-40k	670
...	...	...	...

Another solution, using `reshape2::melt()`

```
> library(reshape2)  
> pew_tidy <- melt(  
  data = pew1,  
  id = "religion",  
  variable.name = "income",  
  value.name = "frequency"  
)
```

NB: income is a **character** variable; we might want to create an **ordered factor** or **numeric** version



# Using pipes: %>%, |>

- R is a **functional language**

- This means that  $f(x)$  returns a value, as in  $y <- f(x)$
- → That value can be passed to another function:  $g(f(x))$
- → And so on:  $h(g(f(x)))$

```
> x <- c(0.109, 0.359, 0.63, 0.996, 0.515, 0.142)
> exp(diff(log(x)))
[1] 3.29 1.75 1.58 0.52 0.28
```

- This gets messy and hard to read, unless you break it down step by step

```
> # Compute the logarithm of `x`, calculate lagged differences,
> # return the exponential function of the result
> log(x)
[1] -2.216 -1.024 -0.462 -0.004 -0.664 -1.952
> diff(log(x))          #calculate lagged diffs
[1]  1.19  0.56  0.46 -0.66 -1.29
> exp(diff(log(x)))    # convert back to original scale
[1] 3.29 1.75 1.58 0.52 0.28
```



# Using pipes: %>%, |>

- Pipes (|>) change the syntax to make this easier

```
> # use pipes  
> x |> log() |> diff() |> exp()  
[1] 3.29 1.75 1.58 0.52 0.28
```

- Basic rules

- x |> f() passes object on left hand side as first argument (or . argument) of function on right hand side
  - x |> f() is the same as f(x)
  - x |> f(y) is the same as f(x, y)
  - y %>% f(x, ., z) is the same as f(x, y, z)
- x %<>% f() does the same, but assigns the result to x
  - Shortcut for x <- x %>% f()

NB: pipes (%>%) originally in magrittr pkg; native pipe (|>) introduced in R 4.1  
Some things, like the “.” work only with %>%



# Using pipes: |> ggplot()

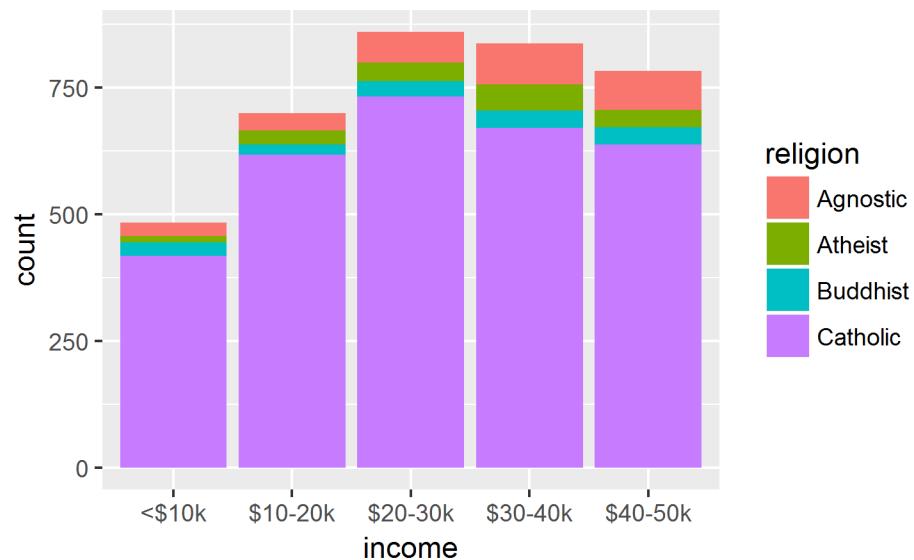


For the Pew data, mutate income → ordered factor and make a ggplot

```
pew1 |>  
gather("income", "frequency", 2:6) |>  
mutate(income = ordered(income, levels=unique(income))) |> # reshape  
ggplot(aes(x=income, fill=religion)) + # make ordered  
geom_bar(aes(weight=frequency)) # plot  
# as freq bars
```

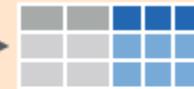
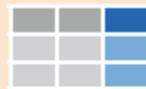
**mutate()** calculates or transforms column variables  
**ordered(income)** levels are now ordered appropriately.

The result is piped to ggplot()



# Tidying: separate() and unite()

It sometimes happens that several variables are crammed into one column, or parts of one variable are split across multiple columns



`tidyverse::separate(storms, date, c("y", "m", "d"))`

Separate one column into several.



`tidyverse::unite(data, col, ..., sep)`

Unite several columns into one.

For example, for the pew data, we might want separate income into low & high

```
pew_long %>%  
  mutate(inc = gsub("\\\\$k", "", income)) %>%  
  mutate(inc = gsub("<", "0-", inc)) %>%  
  separate(inc, c("low", "high"), "-") %>%  
  head()
```

	religion	income	frequency	low	high
1	Agnostic	<\$10k	27	0	10
2	Atheist	<\$10k	12	0	10
3	Buddhist	<\$10k	27	0	10
4	Catholic	<\$10k	418	0	10
5	Agnostic	\$10-20k	34	10	20
6	Atheist	\$10-20k	27	10	20

NB: `tidyverse::separate()` now superceded by `separate_wider_{delim, position}()`

# dplyr: Subset observations (rows)

dplyr implements a variety of verbs to select a subset of observations from a dataset



In a pipe expression, omit the dataset name

**dplyr::filter(iris, Sepal.Length > 7)**

Extract rows that meet logical criteria.

**dplyr::distinct(iris)**

Remove duplicate rows.

**dplyr::sample\_frac(iris, 0.5, replace = TRUE)**

Randomly select fraction of rows.

**dplyr::sample\_n(iris, 10, replace = TRUE)**

Randomly select n rows.

**dplyr::slice(iris, 10:15)**

Select rows by position.

**dplyr::top\_n(storms, 2, date)**

Select and order top n entries (by group if grouped data).

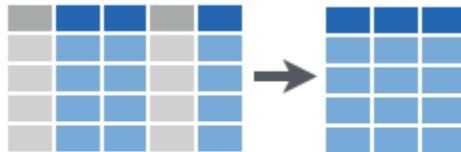
```
iris |> filter(Sepal.Length >7)  
iris |> filter(Species=="setosa")
```

```
iris |> sample_n(10)  
iris |> slice(1:50) # setosa
```

```
iris |> group_by(Species) |>  
      slice_max(Sepal.Width, n=2)
```

After group\_by() does  
operations w/in each group

# dplyr: Subset variables (columns)



`dplyr::select(iris, Sepal.Width, Petal.Length, Species)`

Select columns by name or helper function.

Many helper functions in dplyr allow selection by a **function** of variable names:

`select(iris, contains("!"))`

Select columns whose name contains a character string.

`select(iris, ends_with("Length"))`

Select columns whose name ends with a character string.

`select(iris, everything())`

Select every column.

`select(iris, matches(".t.))`

Select columns whose name matches a regular expression.

`select(iris, num_range("x", 1:5))`

Select columns named x1, x2, x3, x4, x5.

`select(iris, one_of(c("Species", "Genus")))`

Select columns whose names are in a group of names.

`select(iris, starts_with("Sepal"))`

Select columns whose name starts with a character string.

`select(iris, Sepal.Length:Petal.Width)`

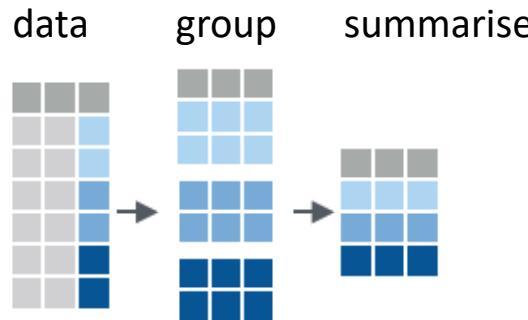
Select all columns between Sepal.Length and Petal.Width (inclusive).

`select(iris, -Species)`

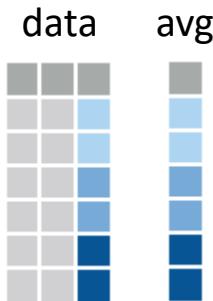
Select all columns except Species.

# dplyr: group\_by() and summarise()

- Fundamental operations in data munging are:
  - grouping a dataset by one or more variables
  - calculating one or more summary measures
  - ungrouping: expand to an ungrouped copy, if needed



```
mtcars |>  
group_by(cyl) |>  
summarise(avg=mean(mpg))
```



```
mtcars |>  
group_by(cyl) |>  
summarise(avg=mean(mpg)) |>  
ungroup()
```

# Example: NASA data on solar radiation



ATMOSPHERIC  
SCIENCE  
DATA CENTER



How does solar radiation vary with latitude, over months of the year?

How to make this plot?

Q:  
what are the basic plot elements?

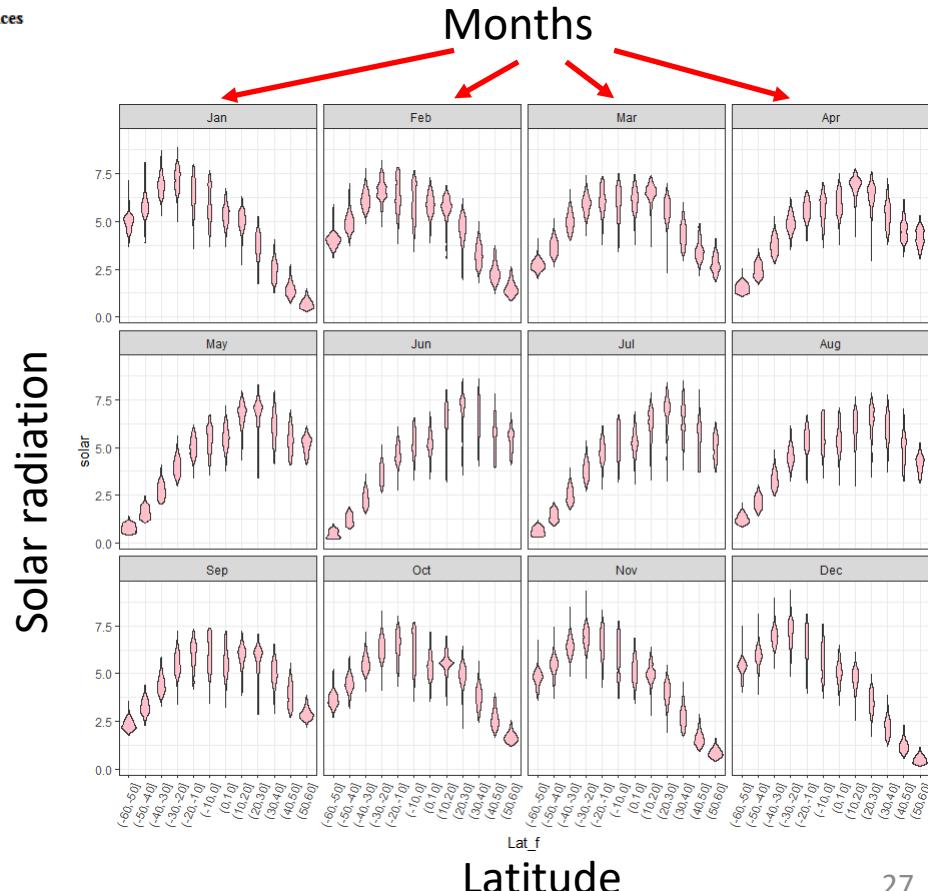
A:

- x = Latitude, y= solar
- geom\_violin()
- panels: month

## Surface meteorology and Solar Energy

*A renewable energy resource web site (release 6.0)*

sponsored by [NASA's Applied Science Program](#) in the Science Mission Directorate  
developed by [POWER: Prediction of Worldwide Energy Resource Project](#)

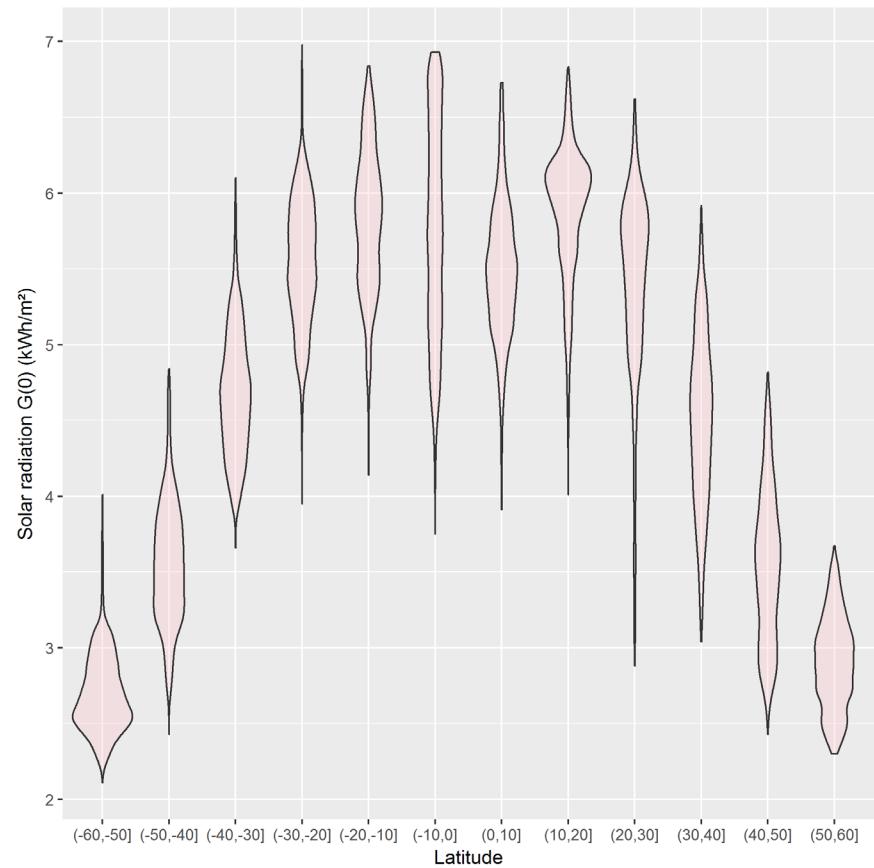


# NASA data: solar radiation

This is easy to do for the total **Annual** solar radiation, a column in the data

```
> str(nasa)
'data.frame': 64800 obs. of 15 variables:
 $ Lat: int -90 -90 -90 -90 -90 -90 -90 -90 -90 ...
 $ Lon: int -180 -179 -178 -177 -176 -175 -174 -173 -172 -171 ...
 $ Jan: num 9.63 9.63 9.63 9.63 9.63 9.63 9.63 9.63 9.63 9.63 ...
 $ Feb: num 5.28 5.28 5.28 5.28 5.28 5.28 5.28 5.28 5.28 5.28 ...
 $ Mar: num 0.75 0.75 0.75 0.75 0.75 0.75 0.75 0.75 0.75 0.75 ...
 $ Apr: num 0 0 0 0 0 0 0 0 0 0 ...
 $ May: num 0 0 0 0 0 0 0 0 0 0 ...
 $ Jun: num 0 0 0 0 0 0 0 0 0 0 ...
 $ Jul: num 0 0 0 0 0 0 0 0 0 0 ...
 $ Aug: num 0 0 0 0 0 0 0 0 0 0 ...
 $ Sep: num 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 ...
 $ Oct: num 3.24 3.24 3.24 3.24 3.24 3.24 3.24 3.24 3.24 3.24 ...
 $ Nov: num 8.28 8.28 8.28 8.28 8.28 8.28 8.28 8.28 8.28 8.28 ...
 $ Dec: num 11 11 11 11 11 11 ...
 $ Ann: num 3.19 3.19 3.19 3.19 3.19 3.19 3.19 3.19 3.19 3.19 ...
```

```
nasa %>%
  filter(abs(Lat) < 60) %>%
  mutate(Latf = cut(Lat, pretty(Lat, n=12))) %>%
  ggplot(aes(x=Latf, y=Ann)) +
  geom_violin(fill="pink", alpha=0.3) +
  labs(x="Latitude", y="Solar radiation G(0) (kWh/m2)")
```



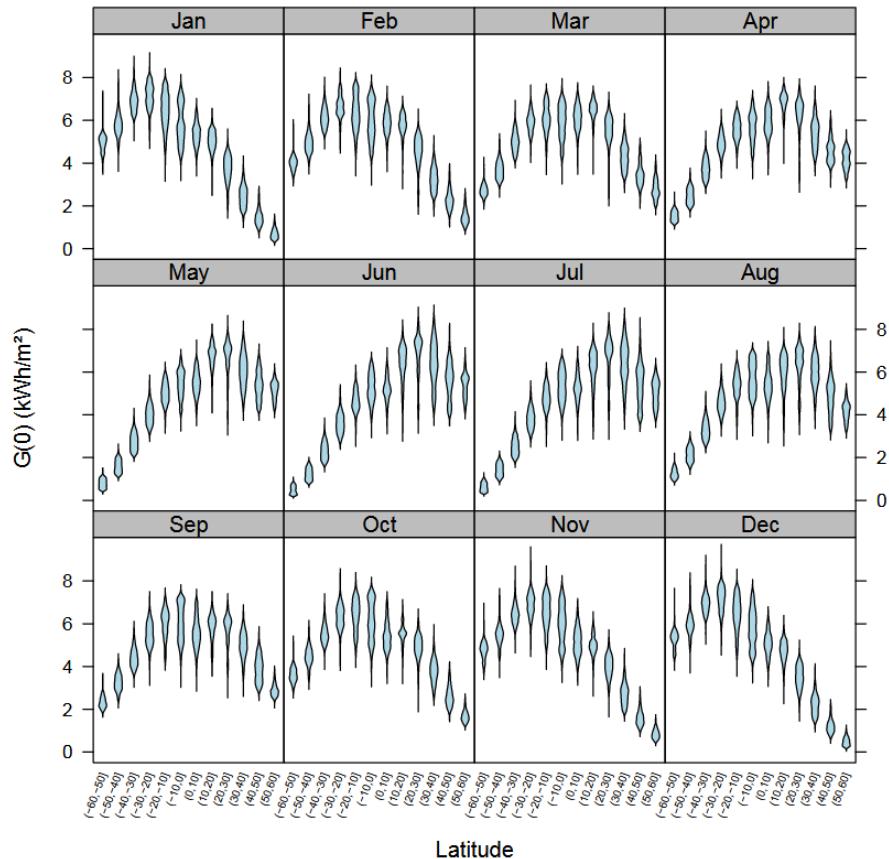
restrict latitude to [-60, 60]  
bin Lat into ordered levels, using cut()

violin plot for each

# Faceting & tidy data

This is complicated to do for the separate months, because the data structure is **untidy**--- months were in separate variables (wide format)

```
> str(nasa)
'data.frame': 64800 obs. of 15 variables:
 $ Lat: int -90 -90 -90 -90 -90 -90 -90 -90 -90 ...
 $ Lon: int -180 -179 -178 -177 -176 -175 -174 -173 -172 -171 ...
 $ Jan: num 9.63 9.63 9.63 9.63 9.63 9.63 9.63 9.63 9.63 ...
 $ Feb: num 5.28 5.28 5.28 5.28 5.28 5.28 5.28 5.28 5.28 ...
 $ Mar: num 0.75 0.75 0.75 0.75 0.75 0.75 0.75 0.75 0.75 ...
 $ Apr: num 0 0 0 0 0 0 0 0 0 ...
 $ May: num 0 0 0 0 0 0 0 0 0 ...
 $ Jun: num 0 0 0 0 0 0 0 0 0 ...
 $ Jul: num 0 0 0 0 0 0 0 0 0 ...
 $ Aug: num 0 0 0 0 0 0 0 0 0 ...
 $ Sep: num 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 ...
 $ Oct: num 3.24 3.24 3.24 3.24 3.24 3.24 3.24 3.24 3.24 ...
 $ Nov: num 8.28 8.28 8.28 8.28 8.28 8.28 8.28 8.28 8.28 ...
 $ Dec: num 11 11 11 11 11 11 ...
 $ Ann: num 3.19 3.19 3.19 3.19 3.19 3.19 3.19 3.19 3.19 ...
```



# tidying the data

To plot solar radiation against latitude by month (separate panels), we need to:

- remove the Ann column
- reshape the data for Jan:Dec to long format, so solar is all in one column
- restrict latitude to [-60, 60]
- bin Lat into ordered levels, using cut()

```
library(tidyr)
library(dplyr)
library(ggplot2)

nasa_long <- nasa |>
  select(-Ann) |>
  gather(month, solar, Jan:Dec, factor_key=TRUE) |>
  filter( abs(Lat) < 60 ) |>
  mutate( Lat_f = cut(Lat, pretty(Lat, 12)))
```

|> “pipes” data to the next stage

**select()** extracts or drops columns

**gather()** collapses columns into key-value pairs

**filter()** subsets observations

**mutate()** creates new variables

# tidying the data

```
> str(nasa_long)
'data.frame': 514080 obs. of 5 variables:
 $ Lat : int -59 -59 -59 -59 -59 -59 -59 -59 -59 -59 ...
 $ Lon : int -180 -179 -178 -177 -176 -175 -174 -173 -172 -171 ...
 $ month: Factor w/ 12 levels "Jan","Feb","Mar",...: 1 1 1 1 1 1 1 1 1 ...
 $ solar: num 5.19 5.19 5.25 5.25 5.17 5.17 5.15 5.15 5.15 5.15 ...
 $ Lat_f: Factor w/ 12 levels "(-60,-50]", "(-50,-40]", ...: 1 1 1 1 1 1 1 1 1 1 ...
> head(nasa_long)
  Lat Lon month solar  Lat_f
1 -59 -180 Jan  5.19 (-60,-50]
2 -59 -179 Jan  5.19 (-60,-50]
3 -59 -178 Jan  5.25 (-60,-50]
4 -59 -177 Jan  5.25 (-60,-50]
5 -59 -176 Jan  5.17 (-60,-50]
6 -59 -175 Jan  5.17 (-60,-50]
```

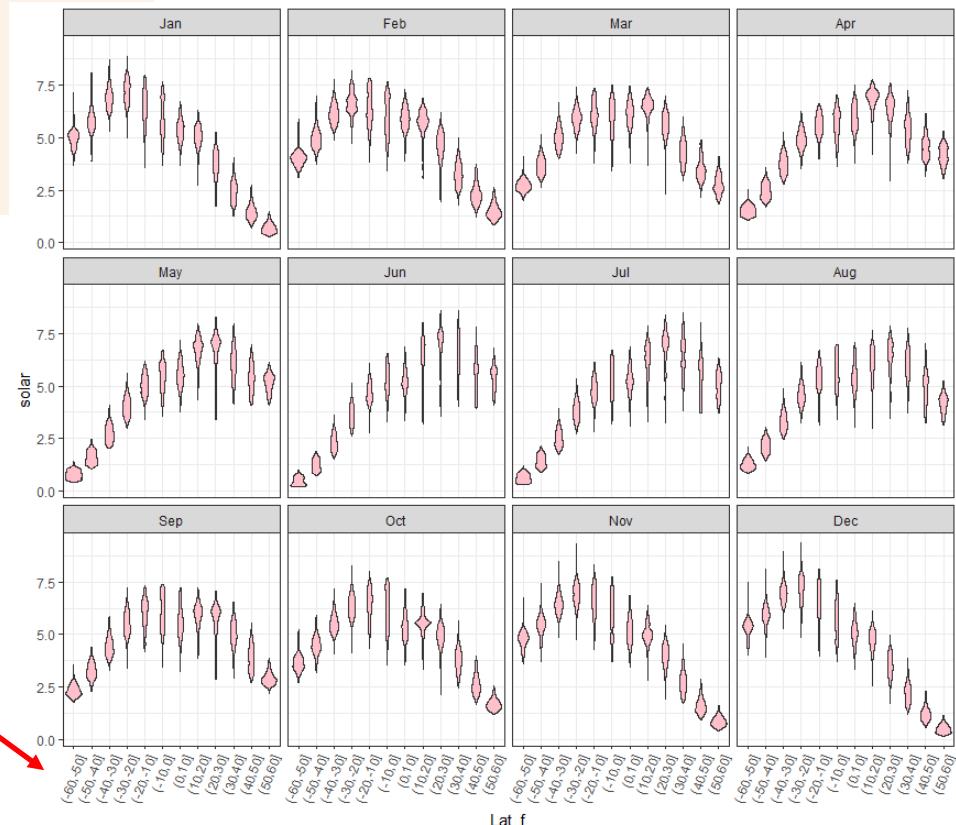
For ease of plotting, I created a factor version of Lat with 12 levels

The data are now in a form where I can plot solar against Lat or Lat\_f and facet by month

# plotting the tidy data

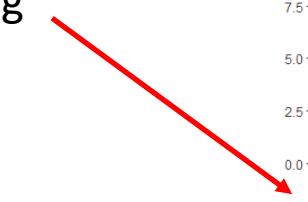
Using `geom_violin()` shows the shapes of the distributions for levels of `Lat_f`

```
ggplot(nasa_long, aes(x=Lat_f, y=solar)) +  
  geom_violin(fill="pink") +  
  facet_wrap(~ month) +  
  theme_bw() +  
  theme(axis.text.x =  
        element_text(angle = 70,  
                     hjust = 1))
```



`facet_wrap(~month)` does the right thing: 12 months → 3×4 grid

I had to rotate the x-axis labels for `Lat_f` to avoid overplotting



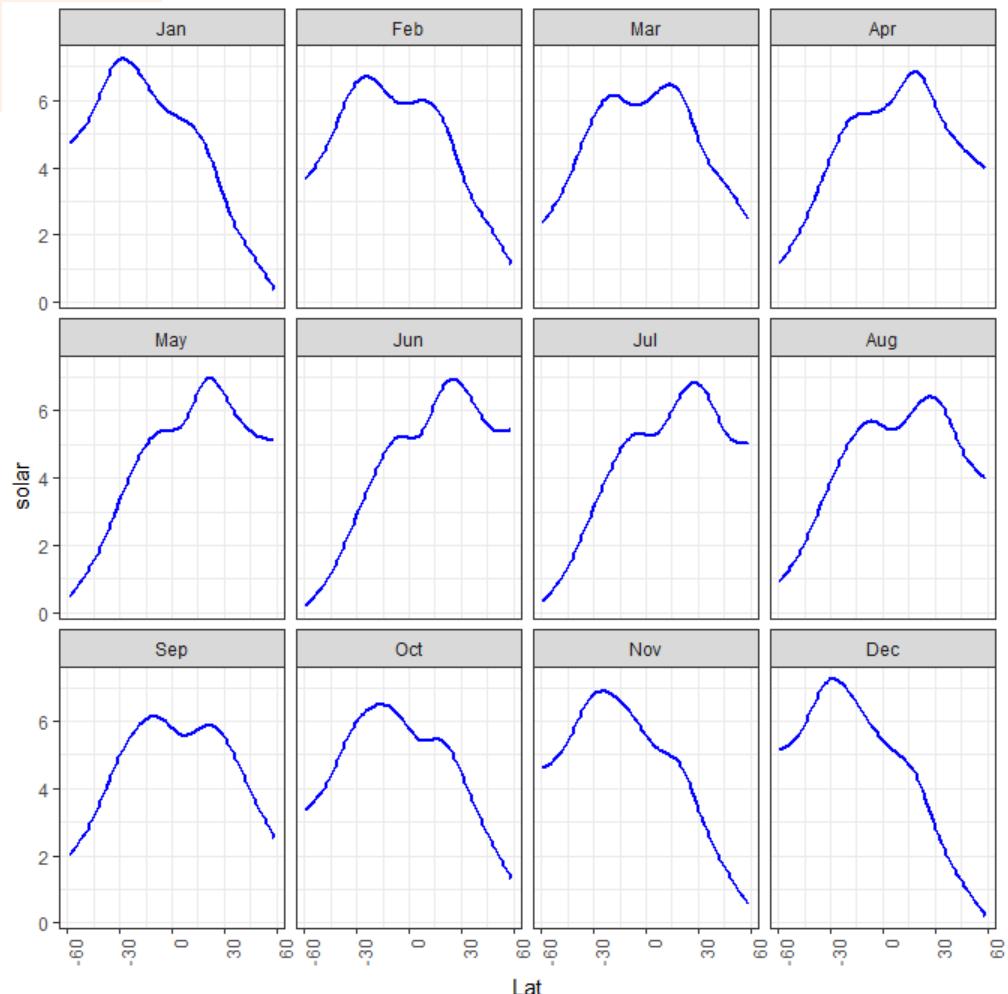
# plotting the tidy data: smoothing

```
ggplot(nasa_long, aes(x=Lat, y=solar)) +  
  geom_smooth(color="blue") +  
  facet_wrap(~ month) +  
  theme_bw()
```

Here we treat Lat as quantitative.  
`geom_smooth()` uses method = "gam" here because of large  $n$

The variation in the smoothed trends over the year suggest quite lawful behavior, shifting over the year

Can we express this as a statistical model ?



# Build a model

What we saw in the plot suggests a **generalized additive model**, with a smooth, **s(Lat)**. Similar to loess in spirit, but this is a statistical model that can be tested.

```
library(mgcv)
nasa.gam <- gam(solar ~ Lon + month + s(Lat), data=nasa_long)
summary(nasa.gam)
```

Family: gaussian  
Link function: identity

Formula:  
 $\text{solar} \sim \text{Lon} + \text{month} + \text{s(Lat)}$

Parametric coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	4.691e+00	6.833e-03	686.409	< 2e-16 ***
Lon	-1.713e-04	1.898e-05	-9.022	< 2e-16 ***
monthFeb	1.195e-01	9.664e-03	12.364	< 2e-16 ***
...	...			
monthDec	-8.046e-02	9.664e-03	-8.326	< 2e-16 ***

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' '

The violin plots suggest that variance is not constant. I'm ignoring this here by using the default gaussian model.

Model terms:

- Lon wasn't included before
- month is a factor, for the plots
- s(Lat) fits a **smoothed term** in latitude, averaged over other factors

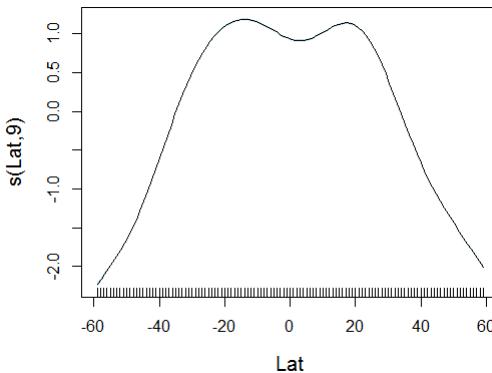
There are other model choices, but it is useful to visualize what we have done so far

# Visualize the model

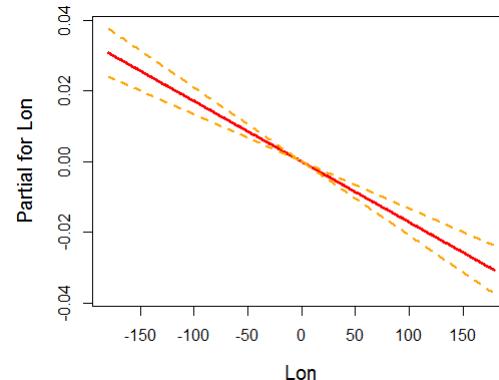
**Effect plots** show the fitted relationship between the response and model terms, averaged over other predictors.

The `{mgcv}` package has its own versions of these: `termplot()`

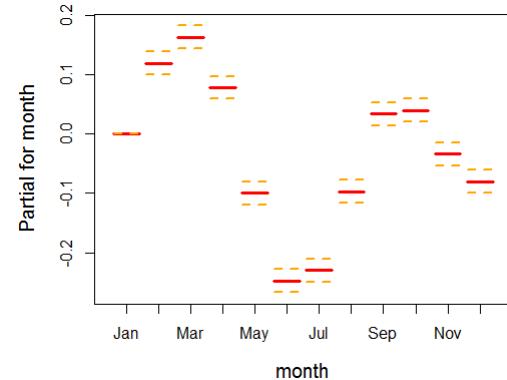
```
plot(nasa.gam, cex.lab=1.25)
termplot(nasa.gam, terms="month", se=TRUE, lwd.term=3, lwd.se=2, cex.lab=1.25)
termplot(nasa.gam, terms="Lon", se=TRUE, lwd.term=3, lwd.se=2, cex.lab=1.25)
```



why the dip at the equator?  
`s(Lat, 9)`: gam used 9 df



effect of longitude is very small, but signif. & maybe interpretable



month should be modeled as a cyclic time variable

# Visualizing models

- R modeling functions [`lm()`, `glm()`, ...] return model objects, but these are “messy”
  - extracting coefficients takes several steps: `data.frame(coef(mymod))`
  - some info ( $R^2$ ,  $F$ ,  $p.value$ ) is computed in `print()` method, not stored
  - can’t easily combine models
- Some have associated plotting functions
  - `plot(model)`: diagnostic plots
  - `car` package: many model plot methods
  - `effects` package: plot effects for model terms
- But what if you want to:
  - make a table of model summary statistics
  - fit a **collection** of models, compare, summarize or visualize them?



# broom: manipulating models

- The **broom** package turns model objects into tidy data frames
  - **glance**(models) extracts model-level summary statistics ( $R^2$ , df, AIC, BIC)
  - **tidy**(models) extracts coefficients, SE, p-values
  - **augment**(models) extracts observation-level info (residuals, ...)

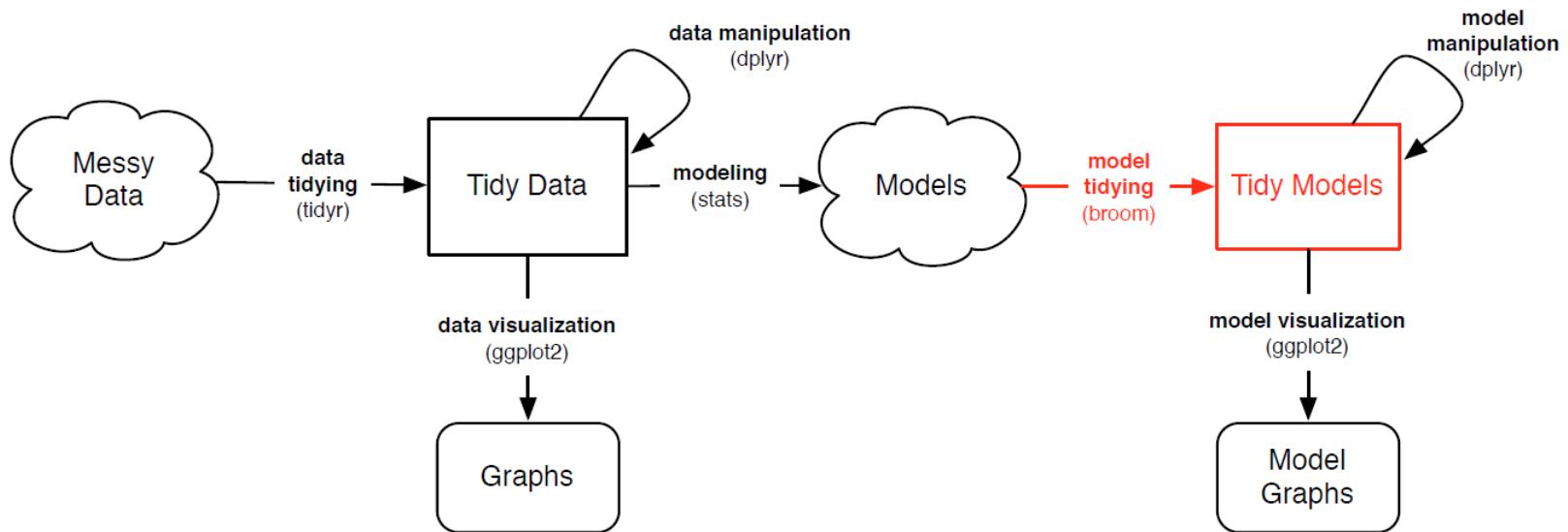


Image from: [https://opr.princeton.edu/workshops/Downloads/2016Jan\\_BroomRobinson.pdf](https://opr.princeton.edu/workshops/Downloads/2016Jan_BroomRobinson.pdf)

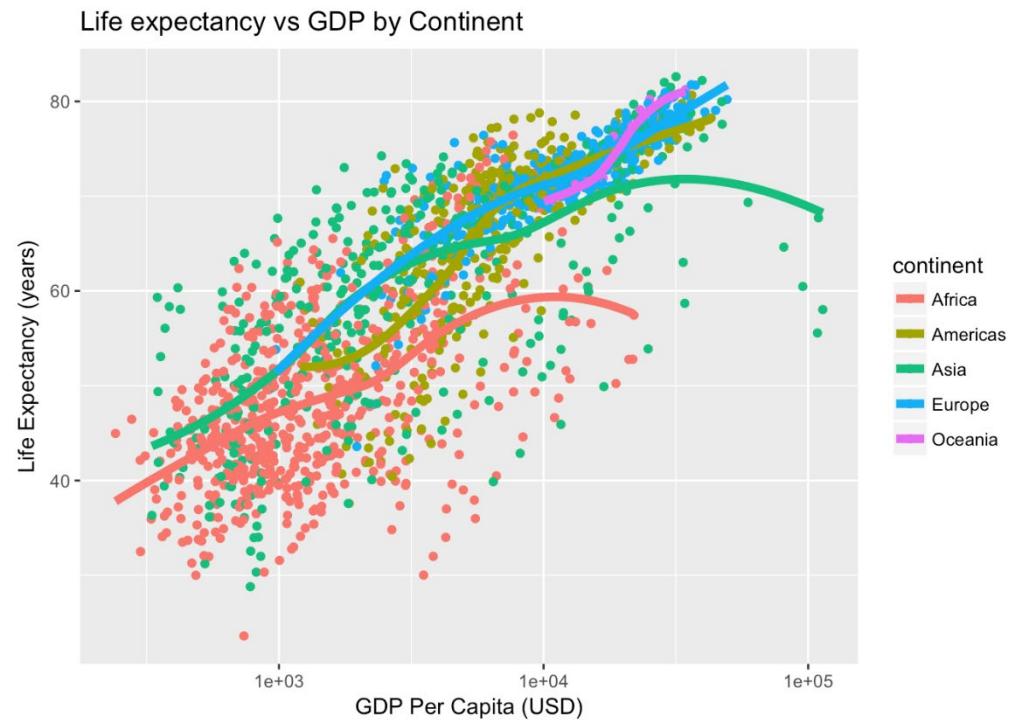
# Example: gapminder data

```
ggplot(aes(x = log(gdpPercap), y=lifeExp, color=continent), data=gapminder) +  
  geom_point() +  
  geom_smooth(method = "loess")
```

How to model this?

How to extract & plot model statistics?

How to fit & display multiple models for subsets?



# Example: gapminder data

Predict life expectancy from year, population, GDP and continent:

```
gapmod <- lm(lifeExp ~ year + pop + log(gdpPercap) + continent, data=gapminder)
summary(gapmod)
```

Call:

```
lm(formula = lifeExp ~ year + pop + log(gdpPercap) + continent, data = gapminder)
```

Residuals:

Min	1Q	Median	3Q	Max
-24.928	-3.285	0.314	3.699	15.221



observation level

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	-4.58e+02	1.67e+01	-27.43	< 2e-16 ***
year	2.38e-01	8.61e-03	27.58	< 2e-16 ***
pop	5.40e-09	1.38e-09	3.91	9.5e-05 ***
log (gdpPercap)	5.10e+00	1.60e-01	31.88	< 2e-16 ***
continentAmericas	8.74e+00	4.63e-01	18.86	< 2e-16 ***
continentAsia	6.64e+00	4.09e-01	16.22	< 2e-16 ***
continentEurope	1.23e+01	5.10e-01	24.11	< 2e-16 ***
continentOceania	1.26e+01	1.27e+00	9.88	< 2e-16 ***

component level  
(coefficients)



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

Residual standard error: 5.79 on 1696 degrees of freedom

Multiple R-squared: 0.8, Adjusted R-squared: 0.799

F-statistic: 969 on 7 and 1696 DF, p-value: <2e-16

model level



## glance() gives the model level summary statistics

```
> glance(gapmod)
   r.squared adj.r.squared sigma statistic p.value df logLik    AIC    BIC deviance df.residual
1       0.8          0.7992 5.789        969      0  8 -5406 10830 10879     56835        1696
```

## tidy() gives the model component (term) statistics

```
> tidy(gapmod)
# # # # #
```

	term	estimate	std.error	statistic	p.value
1	(Intercept)	-4.585e+02	1.671e+01	-27.433	1.982e-137
2	year	2.376e-01	8.613e-03	27.584	1.122e-138
3	pop	5.403e-09	1.381e-09	3.912	9.496e-05
4	log(gdpPercap)	5.103e+00	1.601e-01	31.876	4.096e-175
5	continentAmericas	8.739e+00	4.635e-01	18.856	3.758e-72
6	continentAsia	6.635e+00	4.091e-01	16.219	4.167e-55
7	continentEurope	1.230e+01	5.102e-01	24.113	1.044e-110
8	continentOceania	1.256e+01	1.270e+00	9.884	1.943e-22

## augment() gives the observation level statistics

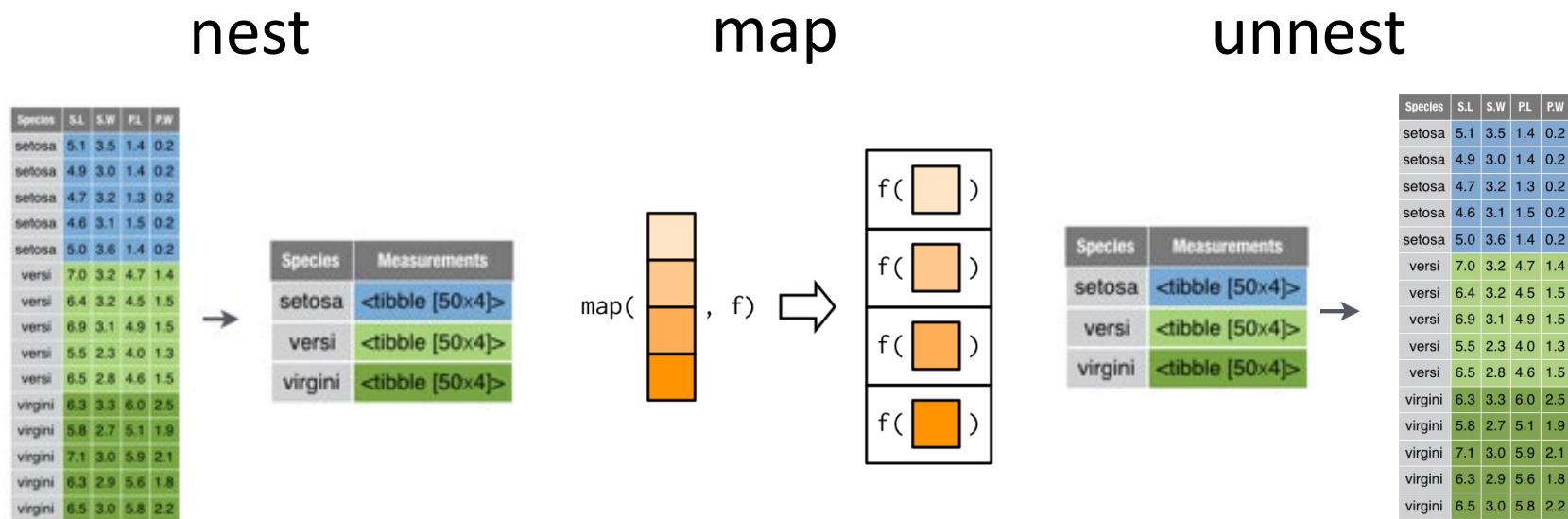
```
> augment(gapmod) %>% slice(1:5)
# # # # #
```

	lifeExp	year	pop	log.gdpPercap.	continent	.fitted	.se.fit	.resid	.hat	.sigma
	<dbl>	<int>	<int>	<dbl>	<fct>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	28.8	1952	8425333	6.66	Asia	46.0	0.408	-17.1	0.00496	5.78
2	30.3	1957	9240934	6.71	Asia	47.4	0.390	-17.1	0.00454	5.78
3	32.0	1962	10267083	6.75	Asia	48.8	0.376	-16.8	0.00423	5.78
4	34.0	1967	11537966	6.73	Asia	49.9	0.372	-15.9	0.00413	5.78
5	36.1	1972	13079460	6.61	Asia	50.5	0.382	-14.4	0.00435	5.78

# ... with 2 more variables: .cooks <dbl>, .std.resid <dbl>

# tidyr:: “nest – map – unnest” trick

- In many cases, we want to perform analysis for each subset of a dataset defined by one or more variables
  - dplyr::group\_by(), summarise(), ungroup() is one way
- tidyr::nest(), purrr::map(), tidyr::unnest() is more general



See: [https://cran.r-project.org/web/packages/broom/vignettes/broom\\_and\\_dplyr.html](https://cran.r-project.org/web/packages/broom/vignettes/broom_and_dplyr.html)

```
n_iris <- iris |> group_by(Species) |> nest()      # group by Species, then nest
n_iris <- iris |> nest(-Species)                      # nest all other cols
```

# tidyr nest()

Species	S.L	S.W	P.L	P.W
setosa	5.1	3.5	1.4	0.2
setosa	4.9	3.0	1.4	0.2
setosa	4.7	3.2	1.3	0.2
setosa	4.6	3.1	1.5	0.2
setosa	5.0	3.6	1.4	0.2
versi	7.0	3.2	4.7	1.4
versi	6.4	3.2	4.5	1.5
versi	6.9	3.1	4.9	1.5
versi	5.5	2.3	4.0	1.3
versi	6.5	2.8	4.6	1.5
virginica	6.3	3.3	6.0	2.5
virginica	5.8	2.7	5.1	1.9
virginica	7.1	3.0	5.9	2.1
virginica	6.3	2.9	5.6	1.8
virginica	6.5	3.0	5.8	2.2

Species	S.L	S.W	P.L	P.W
setosa	5.1	3.5	1.4	0.2
setosa	4.9	3.0	1.4	0.2
setosa	4.7	3.2	1.3	0.2
setosa	4.6	3.1	1.5	0.2
setosa	5.0	3.6	1.4	0.2
versi	7.0	3.2	4.7	1.4
versi	6.4	3.2	4.5	1.5
versi	6.9	3.1	4.9	1.5
versi	5.5	2.3	4.0	1.3
versi	6.5	2.8	4.6	1.5
virginica	6.3	3.3	6.0	2.5
virginica	5.8	2.7	5.1	1.9
virginica	7.1	3.0	5.9	2.1
virginica	6.3	2.9	5.6	1.8
virginica	6.5	3.0	5.8	2.2

nested data frame

Species	data
setosa	<tibble [50 x 4]>
versicolor	<tibble [50 x 4]>
virginica	<tibble [50 x 4]>

n\_iris

setosa

"cell"

Sepal.L	Sepal.W	Petal.L	Petal.W
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2

n\_iris\$data[[1]]

Sepal.L	Sepal.W	Petal.L	Petal.W
7.0	3.2	4.7	1.4
6.4	3.2	4.5	1.5
6.9	3.1	4.9	1.5
5.5	2.3	4.0	1.3
6.5	2.8	4.6	1.5

n\_iris\$data[[2]]

Sepal.L	Sepal.W	Petal.L	Petal.W
6.3	3.3	6.0	2.5
5.8	2.7	5.1	1.9
7.1	3.0	5.9	2.1
6.3	2.9	5.6	1.8
6.5	3.0	5.8	2.2

n\_iris\$data[[3]]

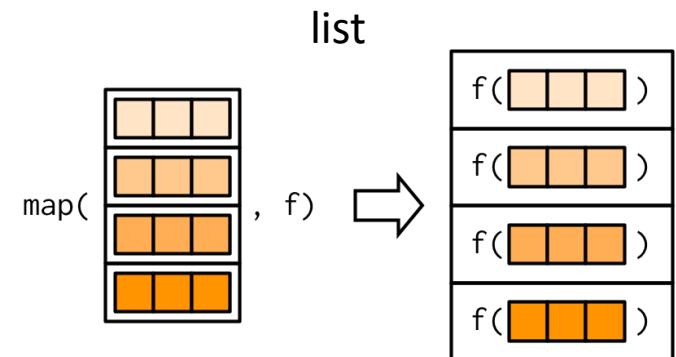
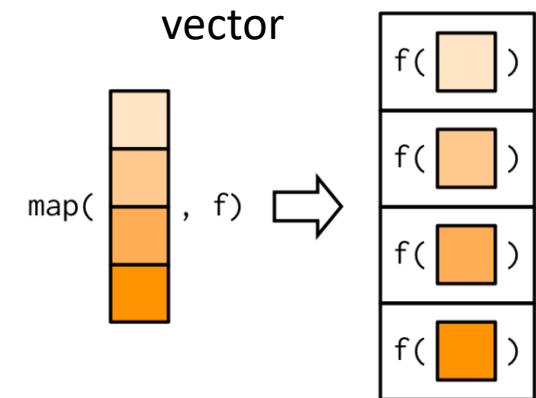
# purrr::map() & friends



A fundamental operation is doing something, `f()`, to each element of a vector, list, or column of a data.frame

`map(1:3, log)`  $\longleftrightarrow$  `list(log(1), log(2), log(3))`

`map(x, f)` returns a **list** of `f()` applied to each of `x`  
Other variants, `map_{dbl, int, chr}` return **vectors**



`data_frame : df`

a	b
1	10
2	20
3	30
4	40

list : return

`mean(` [green square] `)`  
`mean(` [blue square] `)`

`map(df, mean)`

# Fitting multiple models

There may be different effects by continent ( GDP x continent interaction)

- What if want to fit (and visualize) a **separate** model for each continent?
- → **nest by continent**, then {fit, tidy, glance, augment}

```
models <- gapminder %>%
  filter(continent != "Oceania") %>%          # only two countries
  nest(data = -continent) %>%
  mutate(
    fit = map(data, ~ lm(lifeExp ~ year + pop + log(gdpPercap), data = .x)),
    tidied = map(fit, tidy),
    glanced = map(fit, glance),
    augmented = map(fit, augment)
  )
```

What's in this object?

```
names(models)

[1] "continent" "data"       "fit"        "tidied"     "glanced"    "augmented"
```

```
# view model summaries  
models %>%  
  select(continent, glanced) %>%  
  unnest(glanced)
```

Model summary statistics

```
# A tibble: 4 x 13  
continent r.squared adj.r.squared sigma statistic p.value df logLik AIC  
<fct>     <dbl>        <dbl>    <dbl>      <dbl>    <dbl> <dbl> <dbl> <dbl>  
1 Asia       0.696        0.694    6.56      299.  5.27e-101    3 -1305. 2620.  
2 Europe     0.797        0.795    2.46      466.  7.42e-123    3 -833. 1675.  
3 Africa     0.500        0.498    6.48      207.  5.90e- 93    3 -2050. 4110.  
4 Americas   0.720        0.718    4.97      254.  1.39e- 81    3 -904. 1819.  
# ... with 4 more variables: BIC <dbl>, deviance <dbl>, df.residual <int>,
```

```
# model coefficients & tests  
models %>%  
  select(continent, tidied) %>%  
  unnest(tidied)
```

Coefficients

```
# A tibble: 16 x 6  
continent term          estimate std.error statistic p.value  
<fct>    <chr>        <dbl>    <dbl>      <dbl>    <dbl>  
1 Asia     (Intercept) -6.20e+2  4.00e+1   -15.5  1.34e-42  
2 Asia     year         3.23e-1  2.06e-2    15.7  2.41e-43  
3 Asia     pop          5.13e-9  1.66e-9    3.09  2.15e- 3  
4 Asia     log(gdpPerCap) 5.04e+0  2.76e-1    18.3  2.25e-54  
5 Europe   (Intercept) -1.72e+2  1.72e+1   -10.0  4.51e-21  
# ...
```

## Observations

```
# observation-level statistics
models %>%
  select(continent, augmented) %>%
  unnest(augmented)
```

```
# A tibble: 1,680 x 10
  continent lifeExp year      pop `log(gdpPercap)` .fitted     .hat .sigma .cooksdi
  <fct>     <dbl> <int>    <int>            <dbl>     <dbl> <dbl> <dbl> <dbl>
1 Asia       28.8  1952  8425333      6.66    43.7 0.0101  6.53 0.0133
2 Asia       30.3  1957  9240934      6.71    45.6 0.00822  6.53 0.0113
3 Asia       32.0  1962  10267083     6.75    47.4 0.00685  6.53 0.00957
4 Asia       34.0  1967  11537966     6.73    48.9 0.00616  6.53 0.00805
5 Asia       36.1  1972  13079460     6.61    49.9 0.00645  6.54 0.00727
6 Asia       38.4  1977  14880372     6.67    51.9 0.00640  6.54 0.00678
7 Asia       39.9  1982  12881816     6.89    54.6 0.00607  6.53 0.00771
8 Asia       40.8  1987  13867957     6.75    55.5 0.00795  6.53 0.0101
9 Asia       41.7  1992  16317921     6.48    55.8 0.0114   6.53 0.0134
10 Asia      41.8  1997  22227415     6.45    57.3 0.0138   6.53 0.0198
# ... with 1,670 more rows, and 1 more variable: .std.resid <dbl>
```

y

predictors

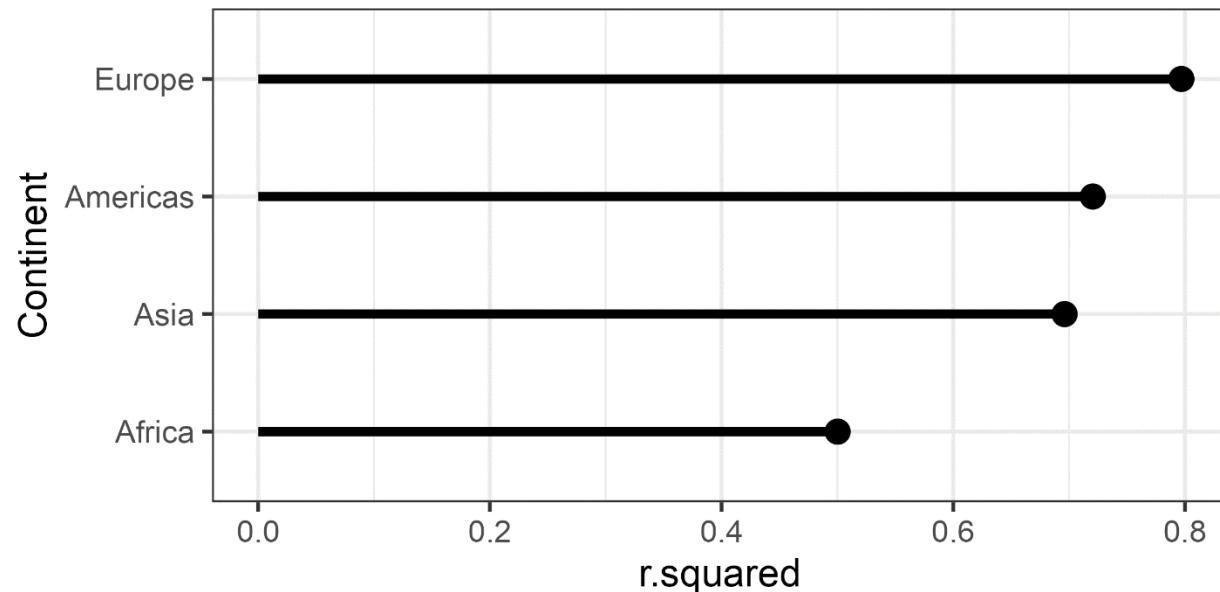
y

diagnostics

# Visualizing multiple models

One visual summary might be a plot of  $R^2$  values, ordered by continent

```
models %>%
  select(continent, glanced) %>% unnest(glanced) %>%
  ggplot(aes(r.squared, reorder(continent, r.squared))) +
  geom_point(size=4) +
  geom_segment(aes(xend = 0, yend = ..y...)) +
  ylab("Continent")
```



# Visualizing coefficients

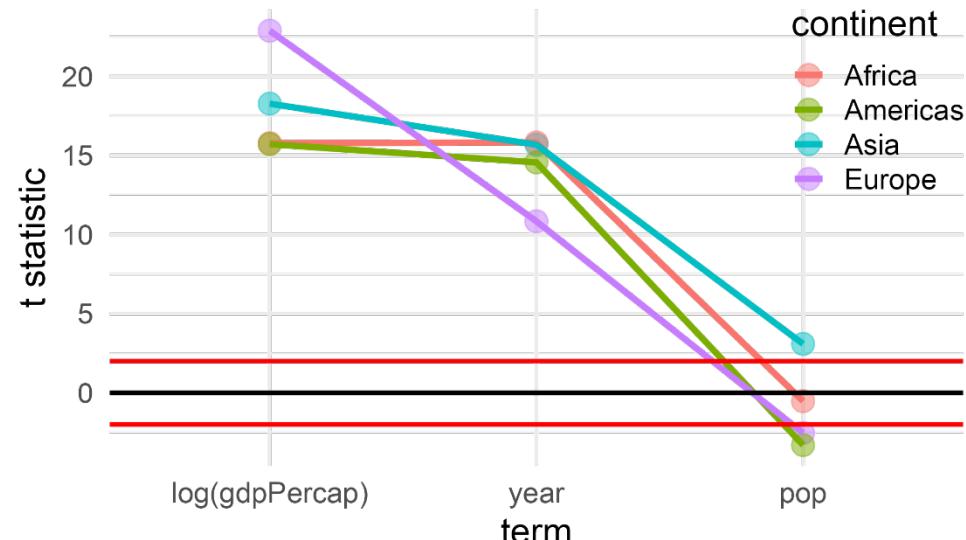
Coefficient plots are often useful, but these are on different scales.

```
models %>% select(continent, tidied) %>% unnest(tidied)
  filter(term != "(Intercept)") %>%
  mutate(term=factor(term, levels=c("log(gdpPercap)", "year", "pop"))) %>%
  ggplot(aes(x=term, y=statistic, color=continent, group=continent)) +
  geom_point(size=5, alpha=0.5) +
  geom_line(size=1.5) +
  geom_hline(yintercept=c(-2, 0, 2), color = c("red", "black", "red")) +
  ylab("t statistic") +
  theme_minimal() + theme(legend.position=c(0.9, 0.8))
```

# get model stats  
# ignore the intercept  
# reorder terms sensibly  
# hlines for non-significance

Here, I plot the  $t$ -statistics,  
 $t=b_{ij}/se(b_{ij})$  for all terms in  
all models.

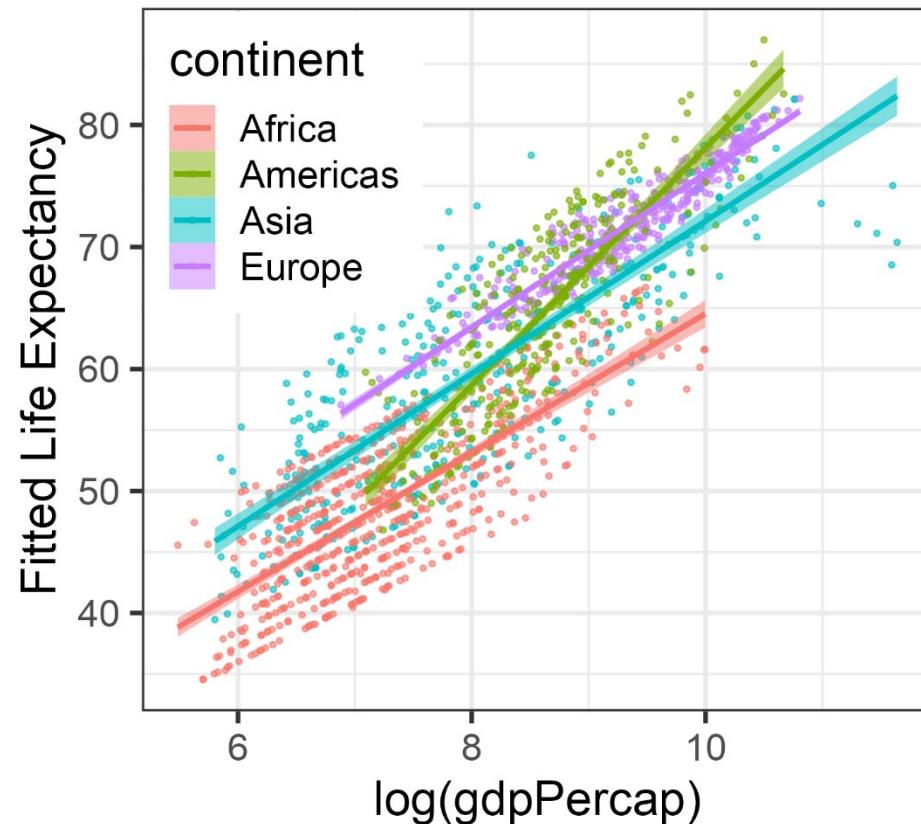
Any values outside  $\sim \pm 2$  are  
significant,  $p < 0.5!$



# Visualizing model fits

```
models %>% select(continent, augmented) %>% unnest(augmented) %>%
  ggplot(aes(x=`log(gdpPerCap)`, y=.fitted, color=continent, fill=continent)) +
  geom_point(size = 0.8, alpha=0.5) +
  geom_smooth(method = "lm", alpha=0.5) +
  ylab("Fitted Life Expectancy")
```

The slope for the Americas  
is noticeably larger than for  
other continents



# Nice tables in R

- Not a ggplot topic, but it is useful to know how you can produce beautiful tables in R
- There are many packages for this: See the [CRAN Task View on Reproducible Research](#)
  - `xtable`: Exports tables to LaTeX or HTML, with lots of control
  - `gt`: the ggplot of tables!
  - `flextable`: similar to gt, but with themes
  - `stargazer`: Well-formatted model summary tables, side-by-side
  - `apaStyle`: Generate APA Tables for MS Word
  - `tinytable`: Simple yet powerful tables for HTML, LaTeX, Word, Markdown

# Tables in R: xtable

Just a few examples, stolen from xtable: [vignette\("xtableGallery.pdf"\)](#)

```
fm1 <- aov(tlimth ~ sex + ethnicty + grade + disadvg, data = tli)
xtable(fm1)
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
sex	1	75.37	75.37	0.38	0.5417
ethnicty	3	2572.15	857.38	4.27	0.0072
grade	1	36.31	36.31	0.18	0.6717
disadvg	1	59.30	59.30	0.30	0.5882
Residuals	93	18682.87	200.89		

```
fm3 <- glm(disadvg ~ ethnicty*grade, data = tli, family = binomial)
xtable(fm3)
```

	Estimate	Std. Error	z value	Pr(> z )
(Intercept)	3.1888	1.5966	2.00	0.0458
ethnictyHISPANIC	-0.2848	2.4808	-0.11	0.9086
ethnictyOTHER	212.1701	22122.7093	0.01	0.9923
ethnictyWHITE	-8.8150	3.3355	-2.64	0.0082
grade	-0.5308	0.2892	-1.84	0.0665
ethnictyHISPANIC:grade	0.2448	0.4357	0.56	0.5742
ethnictyOTHER:grade	-32.6014	3393.4687	-0.01	0.9923
ethnictyWHITE:grade	1.0171	0.5185	1.96	0.0498

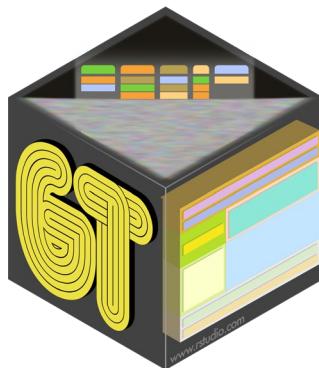
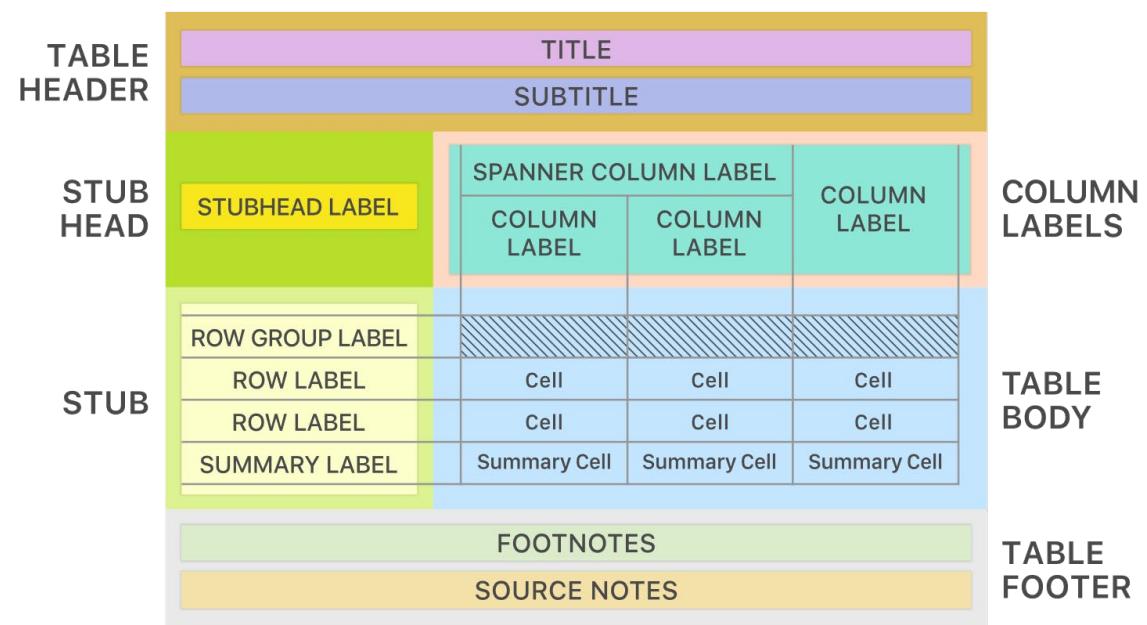
Too many decimals are used here, but you can control all that

# Tables with {gt}

- The {gt} package aims to provide a grammar of tables just as ggplot2 does for graphs
  - Designed to be simple to use, yet powerful

```
iris |> gt()
```

The Parts of a gt Table

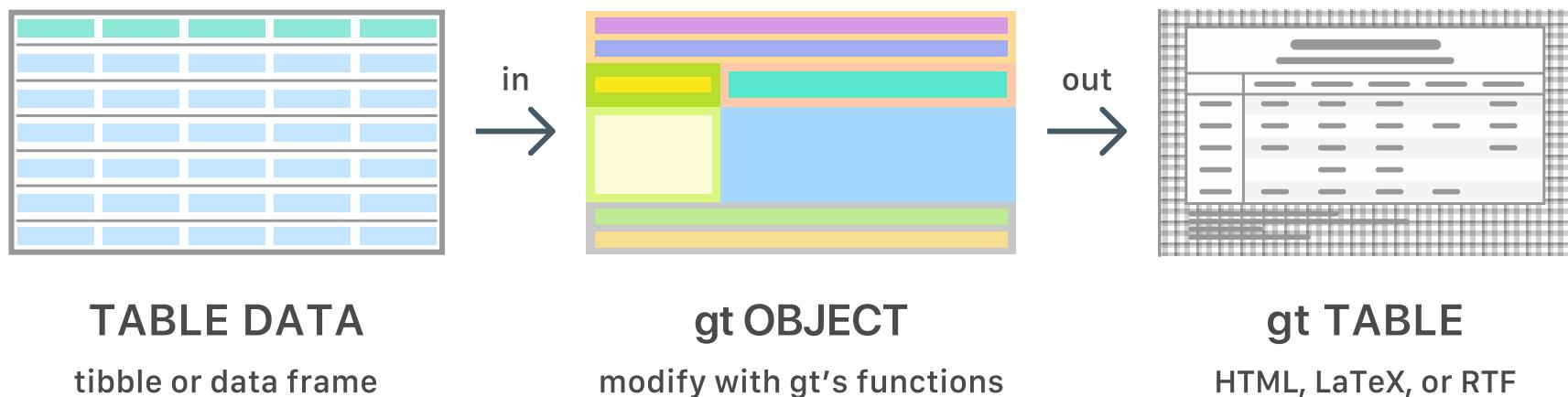


# {gt} workflow

- Data table -> gt\_tbl object

```
iris %>% gt() %>%  
  tab_header(...) %>%  
  tab_options(...) %>% gtsave()
```

## A Typical gt Workflow



See: <https://gt.rstudio.com/>

```
iris_tab <-
iris %>%
  slice(1:5) %>%
  gt() %>%
  tab_header(
    title = "Anderson's Iris Data",
    subtitle = "(Collected in ...)")
```

Sample 5 rows  
pipe to gt()

add header

---

Anderson's Iris Data  
(Collected in the Gaspe Peninsula)

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
6.4	3.1	5.5	1.8	virginica
6.9	3.1	5.4	2.1	virginica
4.4	3.2	1.3	0.2	setosa
7.7	2.8	6.7	2.0	virginica
5.0	2.3	3.3	1.0	versicolor

```

iris_tab <-
iris %>%
  slice(1:5) %>%
  gt() %>%
  tab_header(
    title = "Anderson's Iris Data",
    subtitle = "(Collected in ...)")

```

```

iris_tab <-
iris_tab %>%
  tab_spinner(
    label = "Sepal",
    columns = c(Sepal.Length,
                Sepal.Width)) %>%
  tab_spinner(
    label = "Petal",
    columns = c(Petal.Length,
                Petal.Width)) %>%

```

Add table column spanning headers

---

Anderson's Iris Data				
(Collected in the Gaspe Peninsula)				
Sepal		Petal		
Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
6.4	3.1	5.5	1.8	virginica
6.9	3.1	5.4	2.1	virginica
4.4	3.2	1.3	0.2	setosa
7.7	2.8	6.7	2.0	virginica
5.0	2.3	3.3	1.0	versicolor

---

```

iris_tab <-
iris %>%
  slice(1:5) %>%
  gt() %>%
  tab_header(
    title = "Anderson's Iris Data",
    subtitle = "(Collected in ...)")

```

```

iris_tab <-
iris_tab %>%
  tab_spacer()
  label = "Sepal",
  columns = c(Sepal.Length,
             Sepal.Width)) %>%
  tab_spacer()
  label = "Petal",
  columns = c(Petal.Length,
             Petal.Width)) %>%

```

```

iris_tab <- iris_tab %>%
  cols_label(                                Re-label columns
  Sepal.Length = "Length",
  Sepal.Width = "Width",
  Petal.Length = "Length",
  Petal.Width = "Width") %>%
  tab_options(                                Colorize headings
  heading.background.color = "#c6dbef",
  column_labels.background.color = "#edf8fb")

```

Anderson's Iris Data				
(Collected in the Gaspe Peninsula)				
Sepal		Petal		
Length	Width	Length	Width	Species
6.4	3.1	5.5	1.8	virginica
6.9	3.1	5.4	2.1	virginica
4.4	3.2	1.3	0.2	setosa
7.7	2.8	6.7	2.0	virginica
5.0	2.3	3.3	1.0	versicolor

# tinytable

tinytable is a lightweight but full-featured table generator

- simple interface, easy to learn
- flexible output for all formats

tinytable



```
library(tinytable)  
x <- mtcars[1:5, 1:5]  
tt(x)
```

mpg	cyl	disp	hp	drat
21.0	6	160	110	3.90
21.0	6	160	110	3.90
22.8	4	108	93	3.85
21.4	6	258	110	3.08
18.7	8	360	175	3.15

# tinytable

A few simple functions for constructing nice tables:

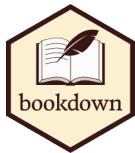
```
data |> tt() |>      # basic table  
  format_tt() |>      # format columns: significant digits, decimal points, ...  
  style_tt() |>      # align, colors, ...  
  theme_tt() |>      # themes for LaTeX, HTML, ...  
  group_tt()          # spanning labels for groups of rows/ columns  
  
save_tt(x, filename) # save to .docx, .html, .png, .pdf, .tex, or render in Rmarkdown
```

# Writing & publishing

A variety of R packages make it easy to write and publish research reports, slide presentations in various formats (HTML, Word, LaTeX, ...), all within R Studio



Write R scripts or documents comprising reproducible code and text



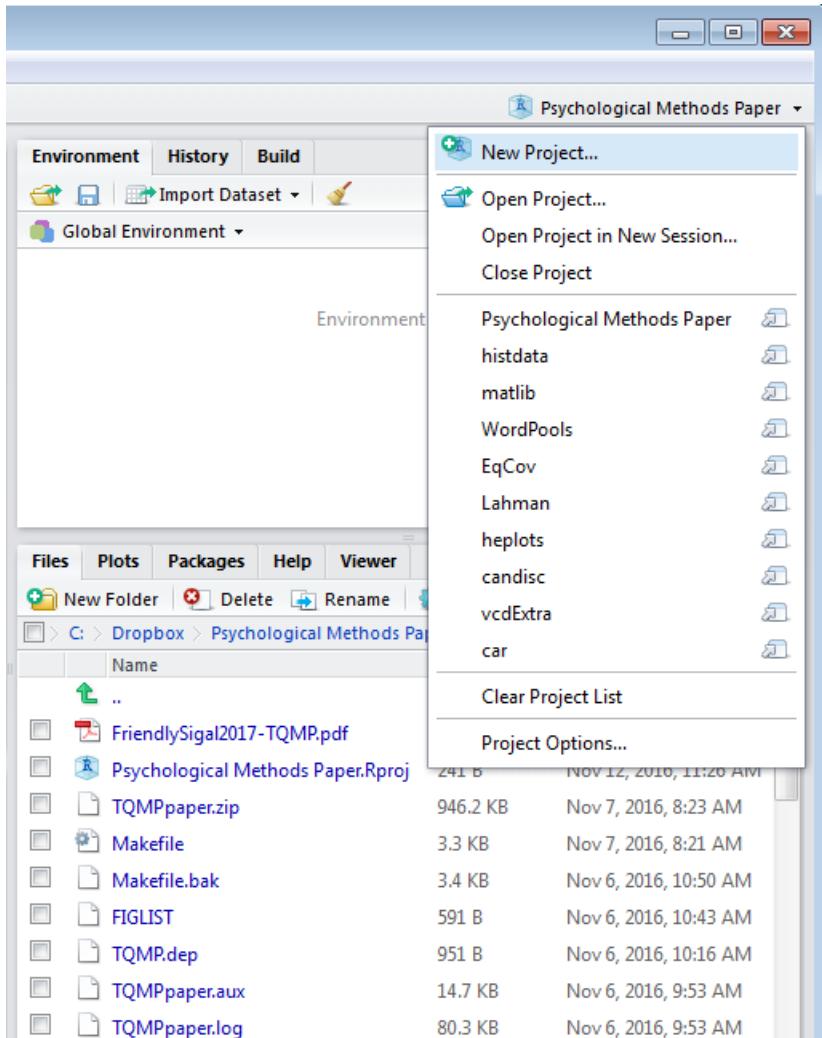
Special format for books, blogs, websites, ...

Organize your work with:

- R Studio projects
- Github for version control & collaboration

Publish online with github-pages, Netlify, ...

# R Studio projects



R Studio projects are a handy way to organize your work

The image shows two dialog boxes. The top one, "Create Project", has three options: "New Directory" (Start a project in a brand new working directory), "Existing Directory" (Associate a project with an existing working directory), and "Version Control" (Checkout a project from a version control repository). The bottom one, "Create New Project", has fields for "Directory name" (set to "C:/R/projects"), "Create project as subdirectory of" (also set to "C:/R/projects"), and checkboxes for "Create a git repository" and "Open in new session". A red arrow points from the ".Rproj" file in the R Studio interface to the "Create Project" dialog.

Lahman.Rproj

The .Rproj item opens the project in R Studio

# R Studio projects

An R Studio project for a **research paper**: R files (scripts), Rmd files (text, R “chunks”)

The screenshot shows the R Studio interface with a project titled "Psychological Methods Paper".

**File Explorer:** Shows the project structure with files like TQMPpaper.Rmd, parenting.Rmd, Rohwer-MMRA-ex.R, SocialCog.Rmd, parenting-ex.R, and a Global Environment.

**Code Editor:** Displays the content of TQMPpaper.Rmd, which includes:

```
1 title: "Graphical Methods for Multivariate Linear Models in Psychological Research: An R Tutorial"
2 shorttitle: "Graphical Methods for MLMs"
3 author:
4   - name: Michael Friendly
5     affiliation: 1
6   corresponding: yes # Define only one corresponding author
7   address: Psychology Department, York University, Toronto, Ontario, Canada, M3J1P3
8   email: friendly@yorku.ca
9   - name: Matthew Sigal
10    affiliation: 1
11 affiliation:
12   - id: 1
13   institution: York University
14 abstract: |
15   This paper is designed as a tutorial to highlight some recent developments for visualizing the relationships among response and predictor variables in multivariate linear
16
17
18
19
```

**Console:** Displays the R startup message and basic information about the R environment.

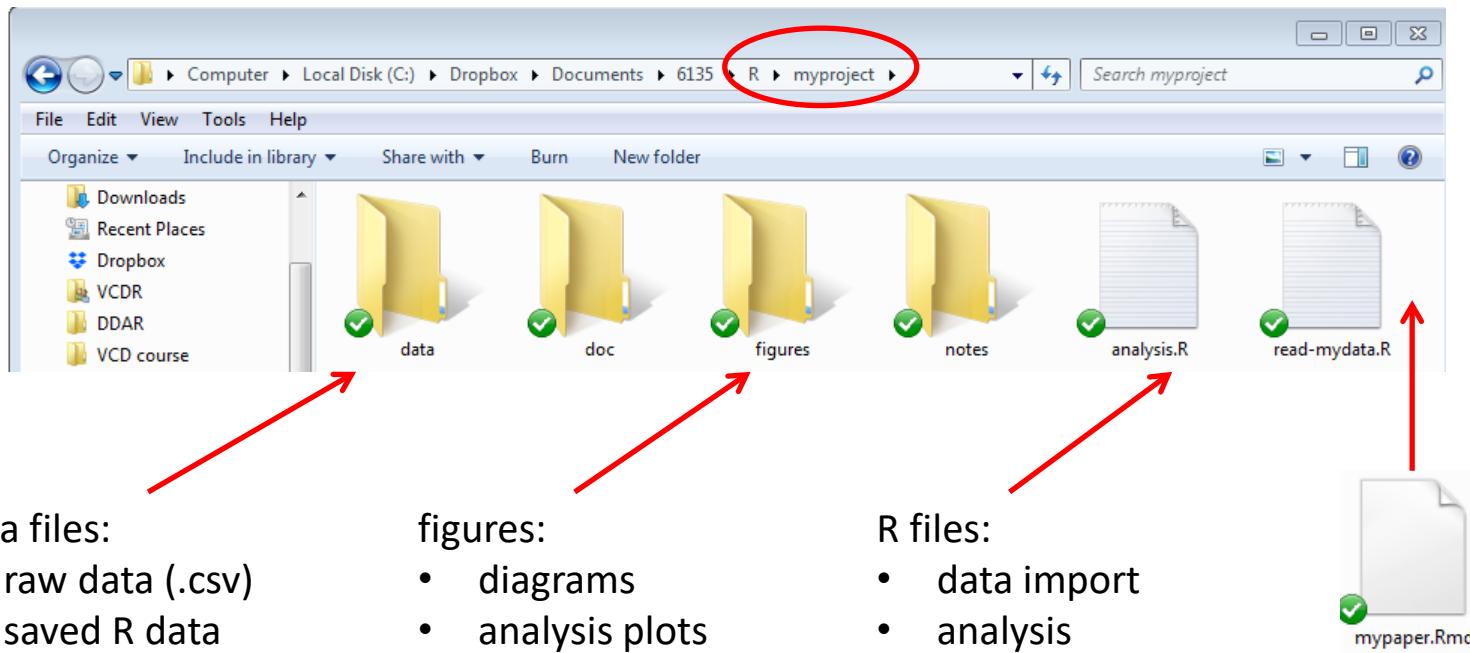
**Environment:** Shows a sidebar with various graphical methods and statistical concepts.

**Files:** Shows the contents of the project directory, including PDFs, Rproj files, and log files.

**Project Options:** A sidebar menu with options like New Project..., Open Project..., and Close Project.

# Organizing an R project

- Use a separate folder for each project
- Use sub-folders for various parts



Write up files will go here (.Rmd, .docx, .pdf)

This project, saved in a **Dropbox** folder automatically syncs with all my computers & collaborators. I use Git & **GitHub** for more serious work.

# Organizing an R project

- Use separate R files for different steps:
  - Data import, data cleaning, ... → save as an RData file
  - Analysis: load RData, ...

read-mydata.R

```
# read the data; better yet: use RStudio File -> Import Dataset ...
mydata <- read.csv("data/mydata.csv")

# data cleaning:
#     filter missing, make factors, transform variables, ....

# save the current state
save("data/mydata.RData")
```

# Organizing an R project

- Use separate R files for different steps:
  - Data import, data cleaning, ... → save as an RData file
  - Analysis: load RData, ...

analyse.R

```
#' ## load the data
load("data/mydata.RData")

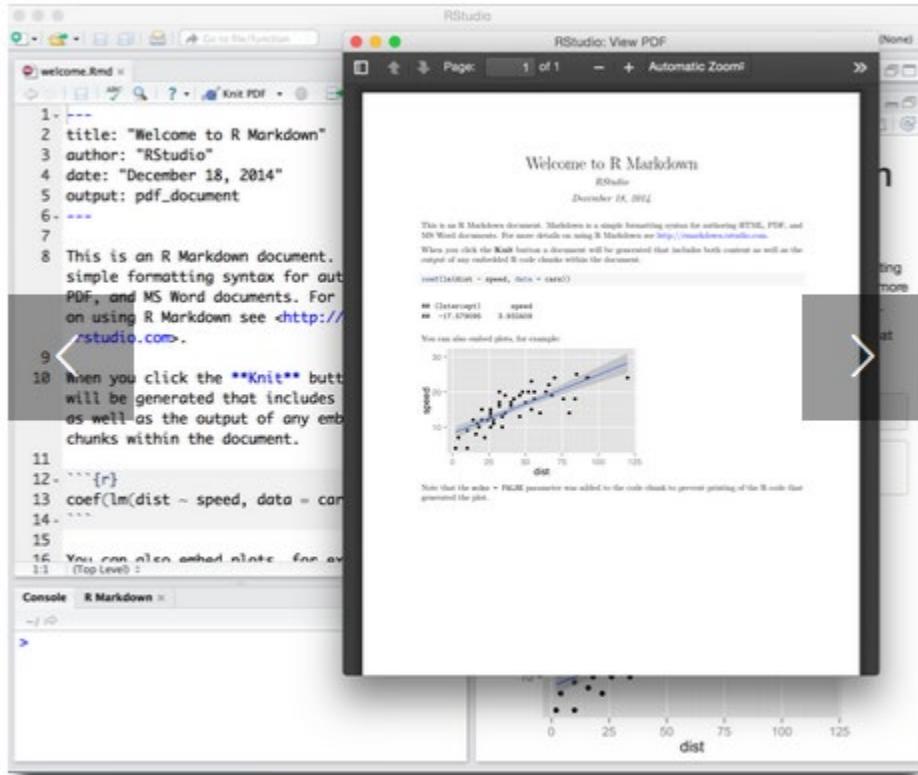
#' ## do the analysis – exploratory plots
plot(mydata)

#' ## fit models
mymod.1 <- lm(y ~ X1 + X2 + X3, data=mydata)

#' ## plot models, extract model summaries
plot(mymod.1)
summary(mymod.1)
```

NB: `#' ##` is a special R comment for a H2 heading in an R “notebook” script

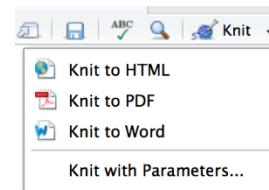
# Reproducible analysis & reporting



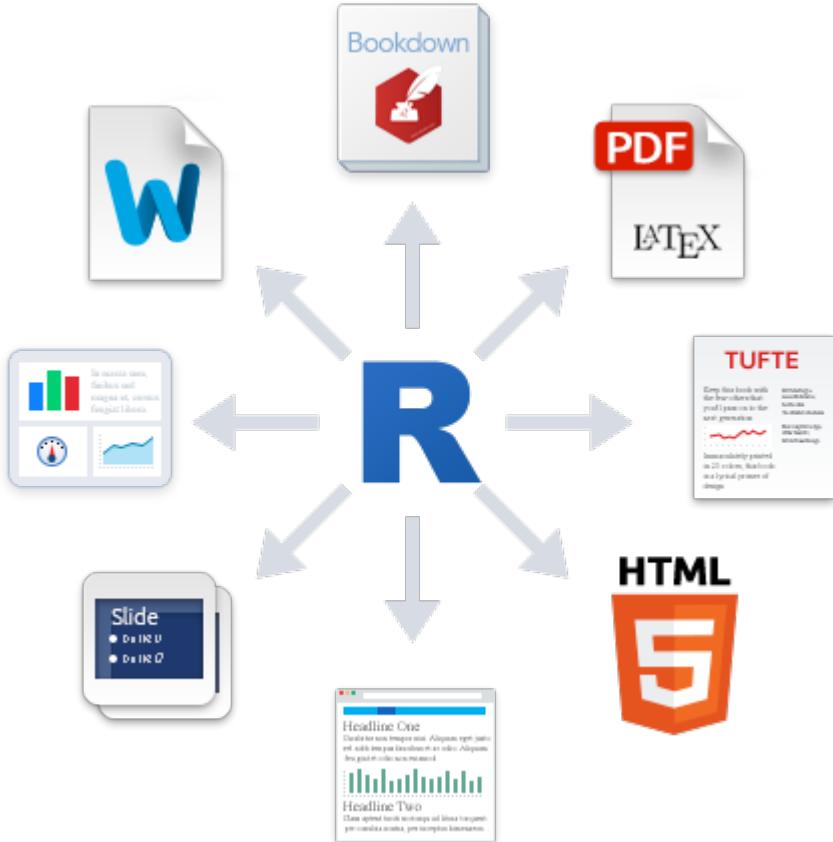
R Studio, together with the knitr and rmarkdown packages provide an easy way to combine writing, analysis, and R output into complete documents

.Rmd files are just text files, using rmarkdown markup and knitr to run R on “code chunks”

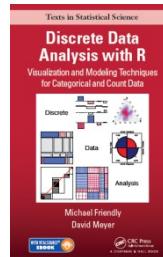
A given document can be rendered in different output formats:



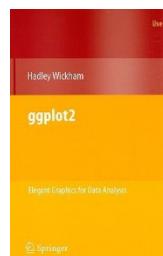
# Output formats and templates



The integration of R, R Studio, knitr, rmarkdown and other tools is now highly advanced.



My 2<sup>nd</sup> last book was written entirely in R Studio, using .Rnw syntax → LaTeX → PDF → camera ready copy



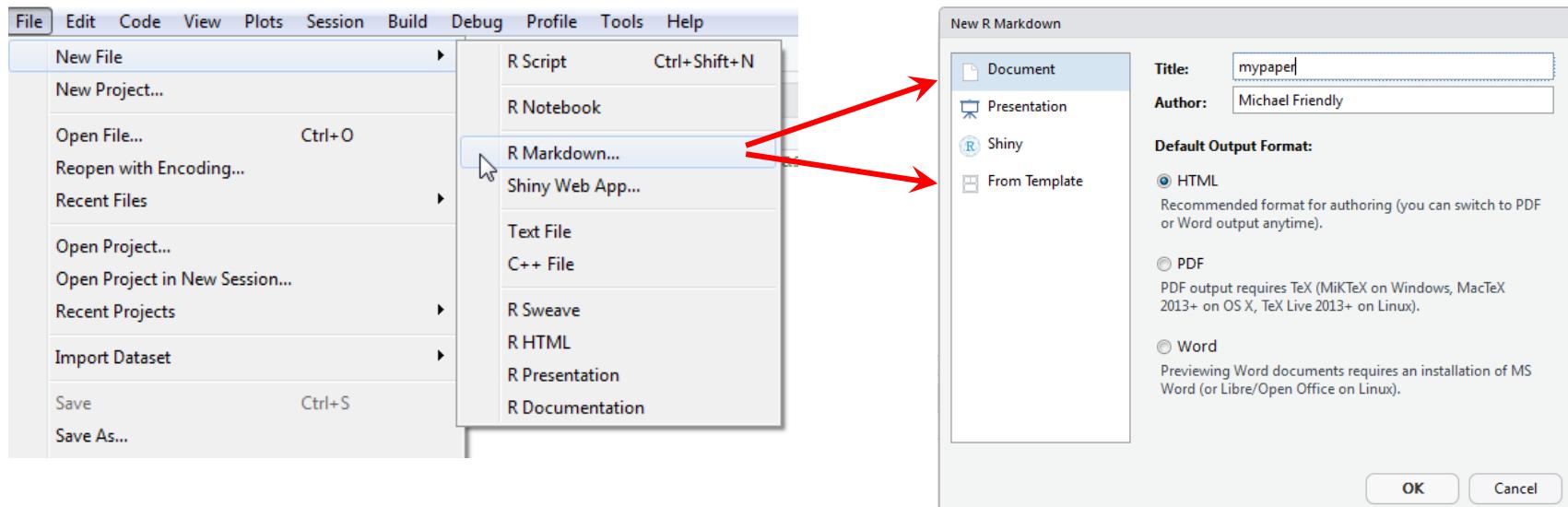
The ggplot2 book was written using .Rmd format.

The [bookdown](#) package makes it easier to manage a book-length project – TOC, fig/table #s, cross-references, etc.  
Also: [blogdown](#), [posterdown](#), ...

Templates are available for APA papers, slides, handouts, entire web sites, etc.

# Writing it up

- In R Studio, create a .Rmd file to use R Markdown for your write-up
  - lots of options: HTML, Word, PDF (needs LaTeX)
  - templates for various pub types



# Writing it up

- Use simple Markdown to write text
- Include code chunks for analysis & graphs

mypaper.Rmd, created from a template

```
1 ---  
2 title: "mypaper"  
3 author: "Michael Friendly"  
4 date: "January 29, 2018"  
5 output: html_document  
6 ---  
7  
8 ```{r setup, include=FALSE}  
9 knitr::opts_chunk$set(echo = TRUE)  
10  
11  
12 ## R Markdown  
13  
14 This is an R Markdown document. Markdown is a simple formatting  
details on using R Markdown see <http://rmarkdown.rstudio.com>.  
15  
16 when you click the **Knit** button a document will be generated  
R code chunks within the document. You can embed an R code chunk  
17  
18 ```{r cars}  
19 summary(cars)  
20  
21  
22 ## Including Plots  
23  
24 You can also embed plots, for example:  
25  
26 ```{r pressure, echo=FALSE}  
27 plot(pressure)  
28
```

Help -> Markdown quick reference

Files Plots Packages Help Viewer

Markdown Quick Reference Find in Topic

### Markdown Quick Reference

R Markdown is an easy-to-write plain text format for creating dynamic documents and reports. See [Using R Markdown](#) to learn more.

#### Emphasis

\*italic\* \*\*bold\*\*  
italic bold

#### Headers

# Header 1  
## Header 2  
### Header 3

#### Lists

##### Unordered List

\* Item 1  
\* Item 2  
+ Item 2a  
+ Item 2b

##### Ordered List

1. Item 1  
2. Item 2  
3. Item 3  
+ Item 3a  
+ Item 3b

# Rmarkdown basics

Rmarkdown uses simple formatting for all standard document elements

The image shows a comparison between an R Markdown file and its generated HTML output. On the left, the R Markdown file 'example.Rmd' is open in RStudio. It contains the following code:

```
1 # Header 1
2
3 This is an R Markdown document. Markdown is a
4 simple formatting syntax for authoring webpages.
5
6 Use an asterisk mark to provide emphasis, such
7 as *italics* or **bold**.
8
9 Create lists with a dash:
10 - Item 1
11 - Item 2
12 - Item 3
13
14 Use back ticks to
15 create a block of code
16
17
18 Embed LaTex or MathML equations,
19 $\frac{1}{n} \sum_{i=1}^n x_i$
20
21 Or even footnotes, citations, and a
22 bibliography. [^1]
23
24 [^1]: Markdown is great.
```

Red arrows point from specific sections of the R Markdown code to their corresponding rendered elements in the browser window on the right. The browser window displays the generated 'example.html' page with the following content:

# Header 1

This is an R Markdown document. Markdown is a simple formatting syntax for authoring web pages.

Use an asterisk mark to provide emphasis, such as *italics* or **bold**.

Create lists with a dash:

- Item 1
- Item 2
- Item 3

Use back ticks to  
create a block of code

Embed LaTex or MathML equations,  $\frac{1}{n} \sum_{i=1}^n x_i$

Or even footnotes, citations, and a bibliography.<sup>1</sup>

---

1. Markdown is great. ↪

# R code chunks

R code chunks are run by [knitr](#), and the results are inserted in the output document

The screenshot shows the RStudio interface. On the left, the R Markdown file 'chunks.Rmd' is open, displaying R code chunks. A red box highlights the first chunk, with a red arrow pointing from it to a callout box containing the definition of an R chunk. On the right, the preview window titled 'RStudio: Preview HTML' shows the rendered output. The output includes the rendered text of the first chunk, the R code for the second chunk, the resulting summary statistics table, and the resulting plot.

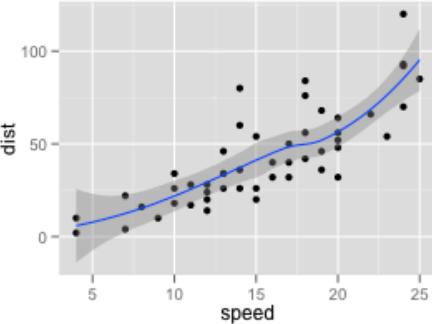
An R chunk:  
```{r name, options}  
# R code here  
```

With R Markdown, you can insert R code chunks including plots:

```
## quick summary and plot
library(ggplot2)
summary(cars)
```

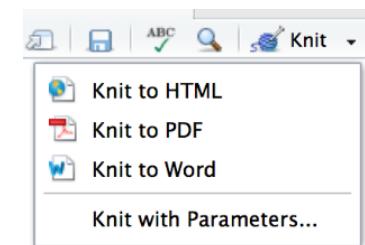
	speed	dist
## Min.	4.0	Min. : 2
## 1st Qu.	12.0	1st Qu.: 26
## Median	15.0	Median : 36
## Mean	15.4	Mean : 43
## 3rd Qu.	19.0	3rd Qu.: 56
## Max.	25.0	Max. :120

```
qplot(speed, dist, data = cars) + geom_smooth()
```



There are many options for controlling the details of chunk output – numbers, tables, graphs

Choose the output format:



# The R Markdown Cheat Sheet provides most of the details

<https://www.rstudio.com/wp-content/uploads/2016/03/rmarkdown-cheatsheet-2.0.pdf>

**R Markdown Cheat Sheet**  
learn more at [rmarkdown.rstudio.com](http://rmarkdown.rstudio.com)

**Workflow**

- 1 Open a new **.Rmd** file at File > New File > R Markdown. Use the wizard that opens to pre-populate the file with a template.
- 2 Write document by editing template.
- 3 Knit document to create report. Use knit button or `render()` to knit.
- 4 Preview Output in IDE window.
- 5 Publish (optional) to web or server. Sync publish button to accounts at [rstudio.com](#), [shinyapps.io](#), or RStudio Connect. Reload document. Find in document. File path to output document.
- 6 Examine build log in R Markdown console.
- 7 Use output file that is saved alongside .Rmd!

**.Rmd files**  
An R Markdown (.Rmd) file is a record of your research. It contains the code that a scientist needs to reproduce your work along with the narration that a reader needs to understand your work.

**Reproducible Research**  
At the click of a button, or the type of a command, you can run the code in an R Markdown file to reproduce your work and export the results as a finished report.

**Dynamic Documents**  
You can choose to export the finished report as a html, pdf, MS Word, ODT, RTF, or markdown document, or as a html or pdf based slide show.

**Interactive Documents**  
Turn your report into an interactive Shiny document in 4 steps:

- 1 Add runtime: shiny to the YAML header.
- 2 Call Shiny input functions to embed input objects.
- 3 Call Shiny render functions to embed reactive output.
- 4 Render with `markdown::run()` or click Run Document in RStudio IDE.

**Embed code with knitr syntax**

**Inline code**  
Insert with `r<code>`. Results appear as text without code.  
Built with `r getVersion()` → Built with 3.2.3

**Code chunks**  
One or more lines surrounded with `{}` and `---`. Place chunk-options within curly braces, after `r`. Insert with `knitr::getVersion()` → `knitr::getVersion()`

**Global options**  
Set with `knitr::opts_chunk$set()`, e.g.  
`knitr::opts_chunk$set(cache = TRUE)`

**Parameters**  
Parameterize your documents to reuse with different inputs (e.g., data sets, values, etc.)

- Add parameters**  
Create and set parameters in the header as sub-values of `params`.  
`param1: m1: 100  
param2: s1: Sys.Date()`
- Call parameters**  
Call parameter values in code as `params$<name>`.  
`today's date is: "param1"`
- Set parameters**  
Set values with `Knit with parameters` or the `params` argument of `render()`.  
`render("doc.Rmd",  
 params = list(n = 1, d = as.Date("2015-03-01")))`

RStudio® is a trademark of RStudio, Inc. • [rstudio.com](#) • 844-449-1212 • [rstudio.com](#)  
More cheat sheets at <http://www.rstudio.com/resources/cheat-sheets/>  
Learn more at [rmarkdown.rstudio.com](https://www.rstudio.com/rmarkdown/) • RStudio IDE 0.99.879 • Updated 02/16

# R notebooks

Often, you just want to “compile” an R script, and get the output embedded in the result, in HTML, Word, or PDF. Just type **Ctrl-Shift-K** or tap the **Compile Report** button

The screenshot illustrates the process of creating an R notebook. On the left, the RStudio interface shows an R Markdown file named 'inv-ex1.Rmd'. A red box highlights the first few lines of the code, which define the document's title, author, date, and output format:

```
# ---  
# title: "Inverse of a matrix"  
# author: "Michael Friendly"  
# date: "07 sep 2016"  
# output: html_document
```

A blue arrow points from the 'Compile Report (Ctrl+Shift+K)' button in the top toolbar to the same button in the R Markdown editor bar. Red arrows point from the annotated code to specific sections in the resulting HTML output on the right. The HTML page has a title 'Inverse of a matrix' and a header 'Michael Friendly 07 Sep 2016'. It contains R code for creating a matrix and calculating its inverse, along with the resulting matrix values.

Annotations in red text on the left side of the RStudio interface include:

- # Document title
- ### header
- use \$math\$

Annotations in red text on the right side of the HTML output include:

- 1.  $\det(A) \neq 0$ , so inverse exists
- 2. Definition of the inverse:  $A^{-1}A = AA^{-1} = I$  or  $AI * A = \text{diag}(nrow(A))$

NB: Sometimes you will get very tiny off-diagonal values (like  $1.341e-13$ ). The function `zapsmall()` will round those to 0.



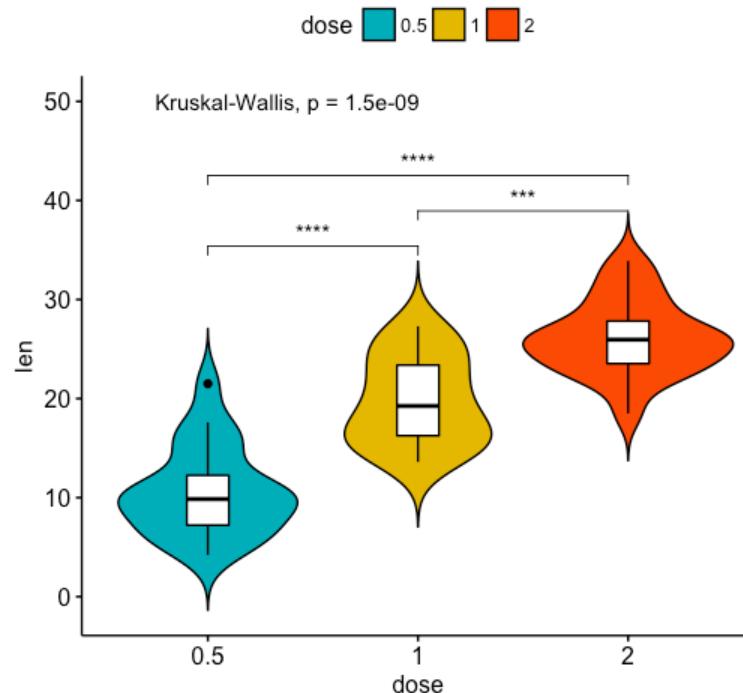
# Summary

- tidyverse thinking
  - data import → clean → make tidy
  - often need to reshape, wide ↔ long
  - tools for dates, factors, strings
- Models:
  - broom to extract tidy results
  - nest – map trick to fit/plot multiple models
- Writing it up
  - Learn to use table tools
  - Use Rstudio projects
  - Learn to use reproducible reporting (.Rmd)

# ggpubr

The `ggpubr` package provides some easy-to-use functions for creating and customizing publication ready plots.

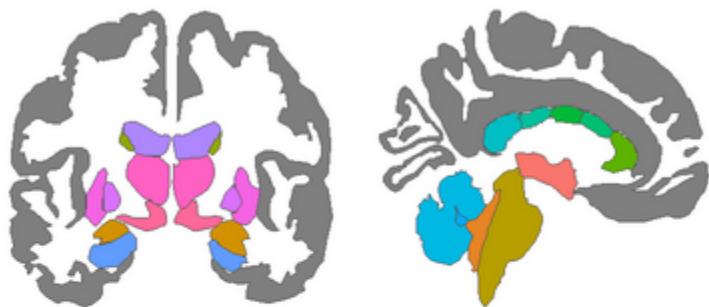
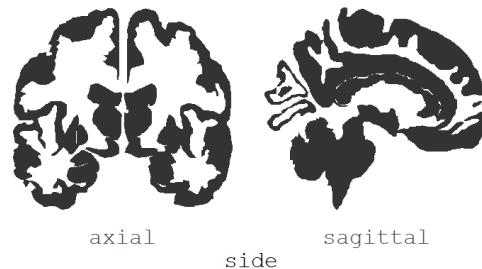
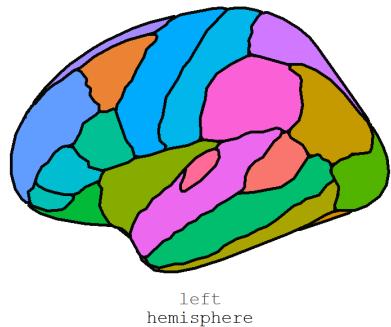
```
ggviolin(df, x = "dose", y = "len", fill = "dose",
          palette = c("#00AFBB", "#E7B800", "#FC4E07"),
          add = "boxplot", add.params = list(fill = "white")) +
  stat_compare_means(comparisons = my_comparisons, label = "p.signif") +
  stat_compare_means(label.y = 50)
```



see the examples at

<http://www.sthda.com/english/rpkgs/ggpubr/>

# ggseg: plotting brain atlases



```
# install.packages("remotes")
# remotes::install_github("LCBC-UiO/ggseg")
library(ggseg)
# install.packages("ggplot2")
library(ggplot2)

ggplot() +
  geom_brain(atlas = aseg) +
  theme_void() +
  theme(legend.position = "bottom",
        legend.text = element_text(size = 8)) +
  guides(fill = guide_legend(ncol = 4))
```