

ggplot2: Going further in the tidyverse



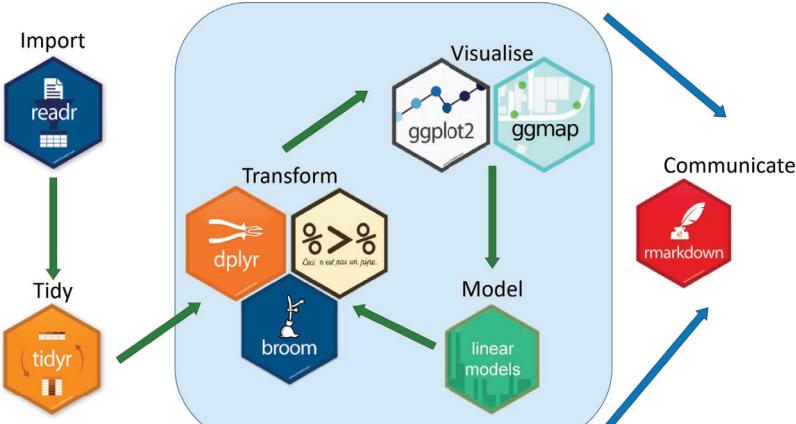
Michael Friendly

Psych 6135

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



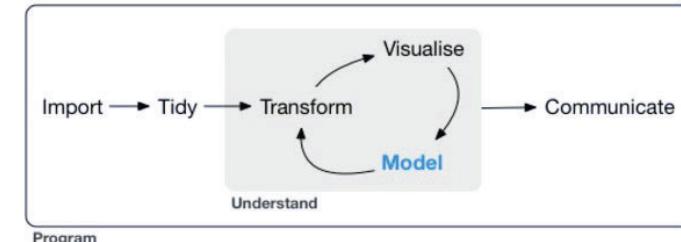
The tidyverse of R packages



3

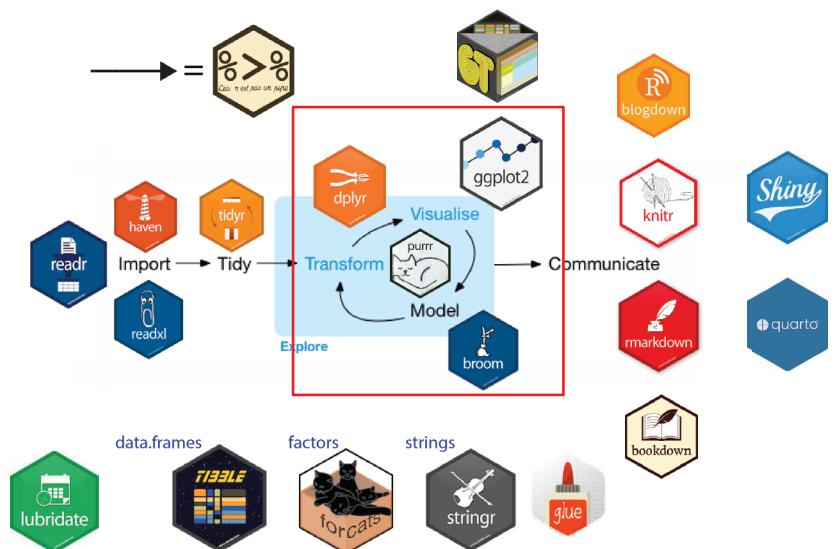
A larger view: Data science

- 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



2

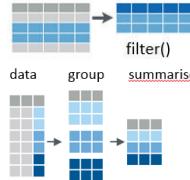
The tidyverse expands



4

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 
- Nice tables in R  
- Writing & publishing



5

Ready for some heavy lifting?



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

6

Data Import / Export

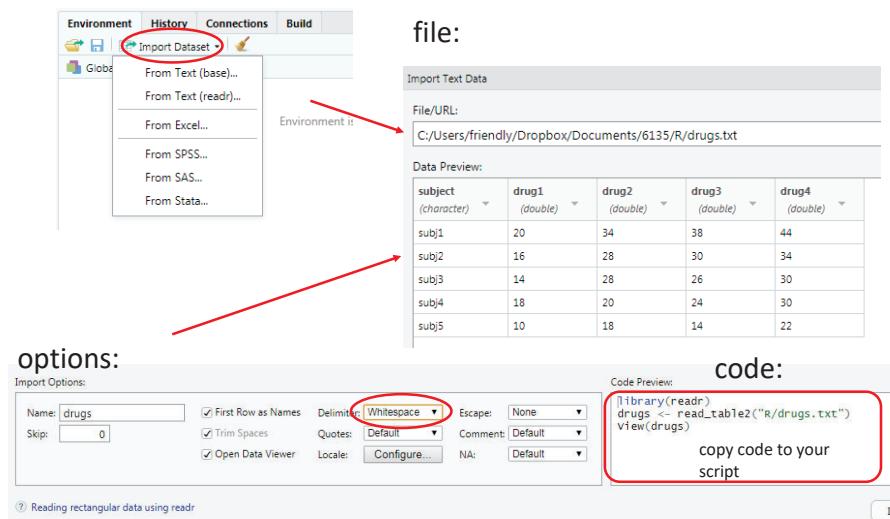
- The **readr** package is the modern, tidy way to import and export data
 - Tabular data:
 - comma delimited (read.csv)
 - any other delimiters (";" = read.csv2; <tab> = read_tsv)
 - Data types:
 - specify column types or let functions guess
- Other data formats



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

7

Data Import: RStudio



The screenshot shows the RStudio interface with the 'Import Dataset' menu open under the Environment tab. The 'file:' section shows a preview of a CSV file with columns subject, drug1, drug2, drug3, and drug4. The 'options:' section shows import settings like 'Name: drugs', 'Delimiter: Whitespace', and 'Escape: None'. The 'code:' section displays the R code: `library(readr)`, `drugs <- read_table2("R/drugs.txt")`, and `vview(drugs)`. A red box highlights the code area with the text 'copy code to your script'.

8

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 | detect matches, subset, replace |  |
| | glue | interpolate values into strings |  |
| Clean-up | janitor | Clean up variable names, find duplicate records |  |

9



Janitor

Unwanted header
Possibly duplicated rows
Illegal variable names
Rows / cols with no data

| A | B | C | D | E | F | G | H | I | J | K | L |
|---|-----------|-----------------|------------|-----------|------------------------|----------------|----------------|---------------|---------|---|---|
| 1 Data most recently refreshed on: Dec-27 2020 | | | | | | | | | | | |
| 2 First Name | Last Name | Employee Status | Subject | Hire Date | % allocated Full time? | do not edit! → | Certification | Certification | Active? | | |
| 3 Jason | Bourne | Teacher | PE | 3/6/2019 | 75% Yes | | Physical ed | Theater | YES | | |
| 4 Jason | Bourne | Teacher | Drafting | 1/14/2019 | 25% Yes | | Physical ed | Theater | YES | | |
| 5 Alicia | Keys | Teacher | Musical | 8/15/2001 | 100% Yes | | Instr. music | Vocal music | YES | | |
| 6 Ada | Lovelace | Teacher | Math | 3/8/2019 | 100% Yes | | PENDING | Computers | YES | | |
| 7 Desus | Nic | 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 | Football | 34700 | #N/A No | | Political sci. | YES | | | |
| 15 Michael | Larsen | Teacher | English | 40071 | 80% No | | Vocal music | English | YES | | |
| 16 | | | | | | | | | | | |

library(janitor)
Data <- read_excel(dirty.xlsx) |> clean_names() |> remove_empty() |> ...

Documentation: <https://sfirke.github.io/janitor/>

10



lubridate: Dates & times

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

- Identify the order of the year (**y**), month (**m**), day (**d**), hour (**h**), minute (**m**) and second (**s**) elements in your data.
- 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 `ymd_hms(), ydm_hm(), ydm_h()`
`ymd_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()`
`mdy("July 4th, 2000")`

4th of July '99 `dmy()`
`dmy("4th of July '99")`

2001: Q3 `q()`
`q("2001: Q3")`

2:01 `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
```

Learn more at: <http://lubridate.tidyverse.org>

11



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)(|I|j|)")`

Mutate Strings

`str_sub(..., sep = "")` Replace substrings by identifying the substrings with `str_sub()` and assigning into the results.
`str_sub(fruit, 1, 3) <- "`

`str_replace(string, pattern, replacement)` Replace the first matching pattern in each string.
`str_replace(fruit, "a", "e")`

`str_replace_all(string, pattern, replacement)` Replace all matched patterns in each string.
`str_replace_all(fruit, "a", "e")`

`str_to_lower(string, locale = "en")` Convert strings to lower case.
`str_to_lower(sentences)`

`str_to_upper(string, locale = "en")` Convert strings to upper case.
`str_to_upper(sentences)`

Join and Split

`str_c(..., sep = "")` 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.
`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}")`

Learn more at: <http://stringr.tidyverse.org>

12



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

13



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 level associated with each integer. It stores integers and categorical values. When you view a factor, R displays not the integers, but the values associated with them.

`a` → `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", "a"))`
`levels(f), levels(f) <- c("x", "y", "z")`

`b` → `b` 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` → `list` fct_count(f, sort = FALSE) Count the number of values with each level. `fct_count(f)`

Change the order of levels

`a` → `a` fct_relevel(f, ..., after = 0) Manually reorder factor levels.
`fct_relevel(f, c("b", "c", "a"))`

`c` → `c` fct_infreq(f, ordered = NA) Reorder levels by the frequency in which they appear in the data (highest frequency first).
`f3 <- factor(c("c", "c", "a", "a"))`
`fct_infreq(f3)`

`b` → `b` fct_inorder(f, ordered = NA) Reorder levels by order in which they appear in the data. `fct_inorder(f2)`

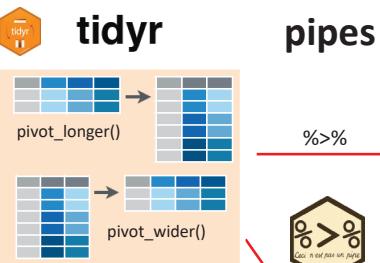
`a` → `a` fct_rev(f) Reverse level order.
`f4 <- factor(c("a", "b", "c"))`
`fct_rev(f4)`

`a` → `a` fct_shift(f) Shift levels to left or right, wrapping around end. `fct_shift(f4)`

Learn more at: <http://forcats.tidyverse.org>

14

Tidy tools: overview



Reshape data to be tidy

Manipulate & summarize tidy data

Visualize me!

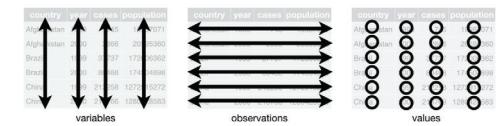
15

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 |

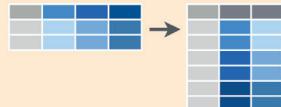
This organization is easy in Excel

But, this makes data analysis and graphing hard

16

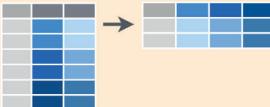
Tidy operations

gather(): Reshape wide to long
synonym: `tidy::pivot_longer()`



`tidy::gather(cases, "year", "n", 2:4)`
Gather columns into rows.

spread(): Reshape long to wide
synonym: `tidy::pivot_wider()`



`tidy::spread(pollution, size, amount)`
Spread rows into columns.



`tidy::separate(storms, date, c("y", "m", "d"))`
Separate one column into several.



`tidy::unite(data, col, ..., sep)`
Unite several columns into one.

Separate parts of a value into several variables

Join related variables into one

17

Tidying: reshaping wide to long

We can tidy the data by reshaping from wide to long format using `tidy::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 | 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 |

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

| key | value | columns |
|----------|----------|-----------|
| religion | income | frequency |
| Agnostic | <\$10k | 27 |
| Atheist | <\$10k | 12 |
| Buddhist | <\$10k | 27 |
| Catholic | <\$10k | 418 |
| Agnostic | \$10-20k | 34 |
| Atheist | \$10-20k | 27 |
| Buddhist | \$10-20k | 21 |
| Catholic | \$10-20k | 617 |
| Agnostic | \$20-30k | 60 |
| Atheist | \$20-30k | 37 |
| Buddhist | \$20-30k | 30 |
| Catholic | \$20-30k | 732 |
| Agnostic | \$30-40k | 81 |
| Atheist | \$30-40k | 52 |
| Buddhist | \$30-40k | 34 |
| 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

19



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
```

20



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 `%>%`

21



Using pipes: |> ggplot()



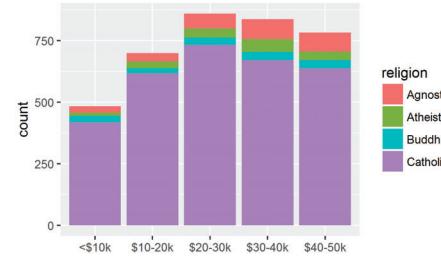
For the Pew data, mutate income → ordered factor and make a ggplot

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

reshape
make ordered
plot
as freq bars

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

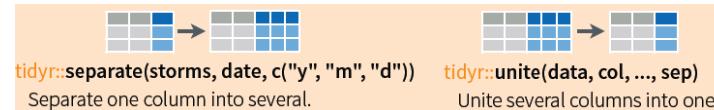
The result is piped to ggplot()



22

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



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()
```

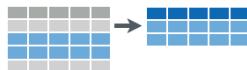
| | religion | income | frequency | low | high |
|---|----------|----------|-----------|-----|------|
| 1 | Agnostic | <\$10k | 27 | 0 | 10 |
| 2 | Atheist | <\$10k | 12 | 0 | 10 |
| 3 | Buddhist | <\$10k | 27 | 0 | 10 |
| 4 | Catholic | <\$10k | 418 | 0 | 10 |
| 5 | Agnostic | \$10-20k | 34 | 10 | 20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

NB: **tidy::separate()** now superceded by **separate_wider_{delim, position}()**

23

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

24

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("x")) Select columns whose name contains a character string. | select(iris, one_of(c("Species", "Genus"))) Select columns whose names are in a group of names. |
| select(iris, ends_with("Length")) Select columns whose name ends with a character string. | select(iris, starts_with("Sepal")) Select columns whose name starts with a character string. |
| select(iris, everything()) Select every column. | select(iris, Sepal.Length:Petal.Width) Select all columns between Sepal.Length and Petal.Width (inclusive). |
| select(iris, matches("x:t")) Select columns whose name matches a regular expression. | select(iris, -Species) Select all columns except Species. |
| select(iris, num_range("x", 1:5)) Select columns named x1, x2, x3, x4, x5. | |

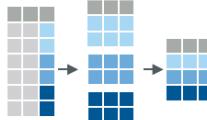
25

dplyr: group_by() and summarise()

- Fundamental operations in data munging are:

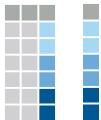
- grouping a dataset by one or more variables
- calculating one or more **summary** measures
- ungrouping**: expand to an ungrouped copy, if needed

data group summarise



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

data avg



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

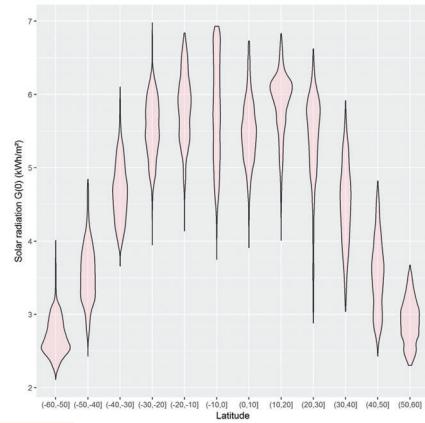
26

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 ...
 $ 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 ...
 $ Ann: num 3.19 3.19 3.19 3.19 3.19 3.19 3.19 3.19 3.19 ...
```

```
nasa %>
  filter(abs(Lat) < 60) %>%
  mutate(Lat = 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

28

Example: NASA data on solar radiation



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



- over 200 satellite-derived meteorology and solar energy parameters
- monthly averaged from 22 years of data
- data tables for a particular location
- GIS Web Mapping Application & Services



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

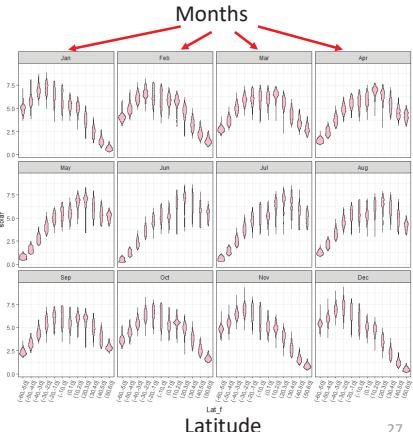
How to make this plot?

Q:
what are the basic plot elements?

A:

- x = Latitude, y= solar
- geom_violin()
- panels: month

Solar radiation

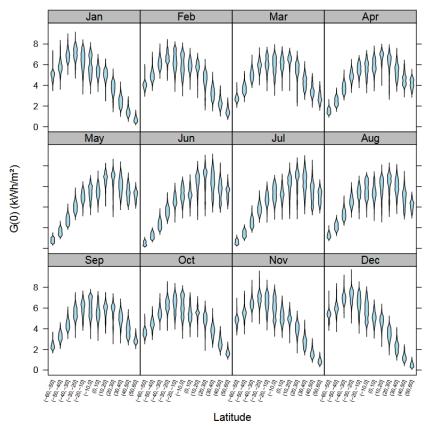


27

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 ...
 $ Feb: num 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 ...
 $ 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 ...
 $ Ann: num 3.19 3.19 3.19 3.19 3.19 3.19 3.19 3.19 3.19 ...
```



29

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(tidyverse)
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

30

tidying the data

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

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

```
> 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]
```

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

31

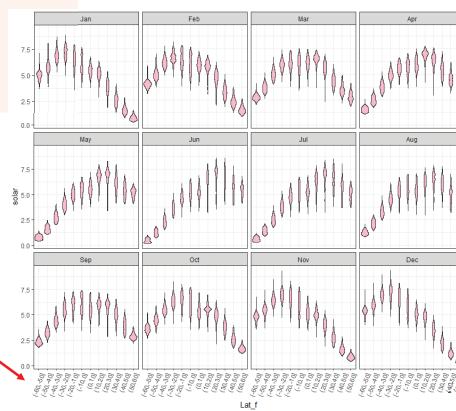
plotting the tidy data

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))
```

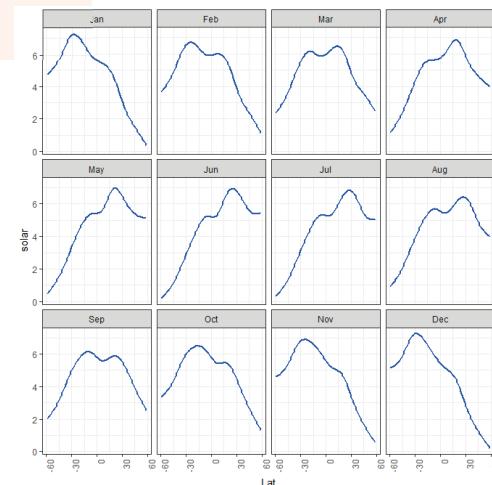
facet_wrap(~month) does the right thing: 12 months → 3×4 grid

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:
solar ~ Lon + month + 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

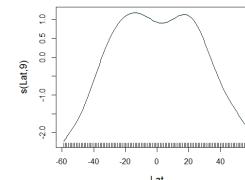
34

Visualize the model

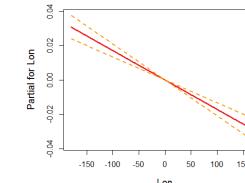
Effect plots show the fitted relationship between the response and model terms, averaged over other predictors.

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

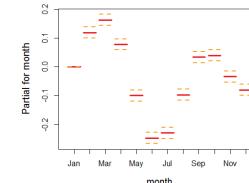
```
plot(nasa.gam, cex.lab=1.25)
termplot(nasa.gam, terms="month", se=TRUE, lwd.term=3, lwd.se=2, cex.lab=1.25)
termplot(nasa.gam, terms="Lon", se=TRUE, lwd.term=3, lwd.se=2, cex.lab=1.25)
```



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

35

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?

36



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, ...)

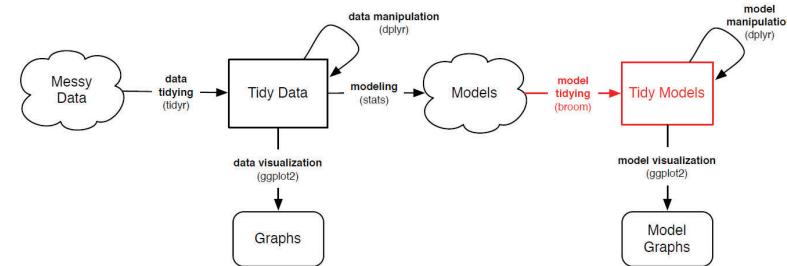


Image from: https://opr.princeton.edu/workshops/Downloads/2016Jan_BroomRobinson.pdf

37

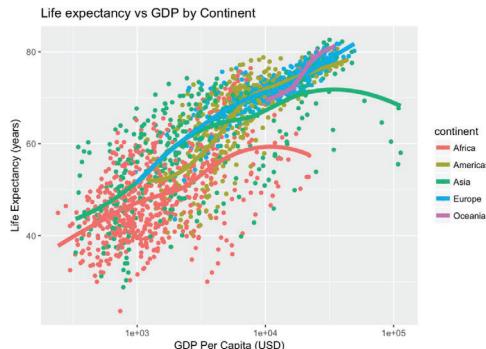
Example: gapminder data

```
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 

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 ***

---
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:  0.7999 
F-statistic: 969.7 on 7 and 1696 DF,  p-value: <2e-16

```

observation level

component level
(coefficients)

model level

102

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 169
```

`tidy()` gives the **model component** (term) statistics

| | 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-08 | 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-05 |
| 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 |

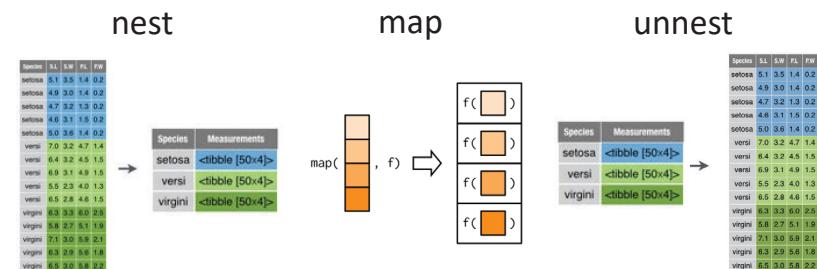
augment() gives the **observation level** statistics

```
> augment(gapmod) %>% slice(1:5)
# A tibble: 5 x 12
  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: .cooked <dbl>, .std.resid <dbl>
```

4

tidyverse:: “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
 - `tidyverse::nest()`, `purrr::map()`, `tidyverse::unnest()` is more general



See: https://cran.r-project.org/web/packages/broom/vignettes/broom_and_dplyr.html

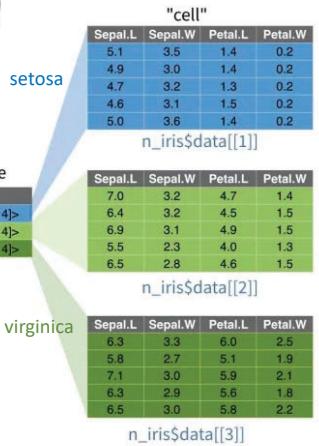
4

```
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 |



CC by RStudio

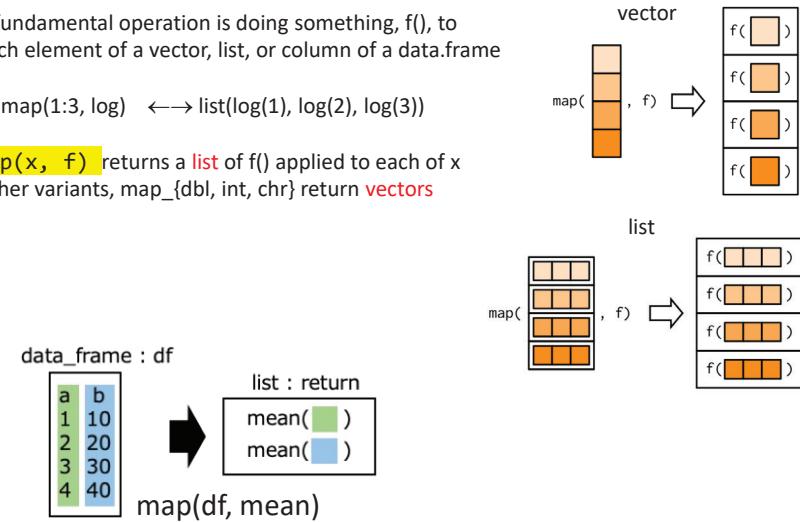
42

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



43

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?
- \rightarrow 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"
```

44

```
# 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>
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

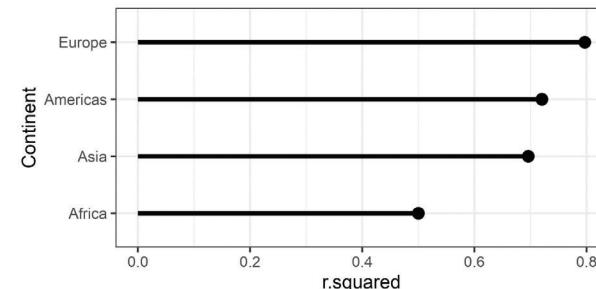
```
# A tibble: 16 x 6
  continent term            estimate std.error statistic p.value
  <fct>   <chr>           <dbl>    <dbl>    <dbl>    <dbl>
1 Asia     (Intercept) -6.20e+2   4.00e+1   -15.5   1.34e-42
2 Asia     year         3.23e-1   2.06e-2    15.7   2.41e-43
3 Asia     pop          5.13e-9   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
# ...
```

45

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")
```



47

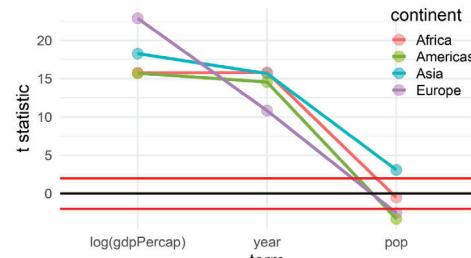
Visualizing coefficients

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

```
models %>% select(continent, tidied) %>% unnest(tidied) # get model stats
filter(term != "(Intercept)") %>% # ignore the intercept
mutate(term=factor(term, levels=c("log(gdpPercap)", "year", "pop"))) %>% # reorder terms sensibly
ggplot(aes(x=term, y=tstatistic, 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")) + # hlines for non-significance
  ylab("t statistic") +
  theme_minimal() + theme(legend.position=c(0.9, 0.8))
```

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, $p < 0.5!$

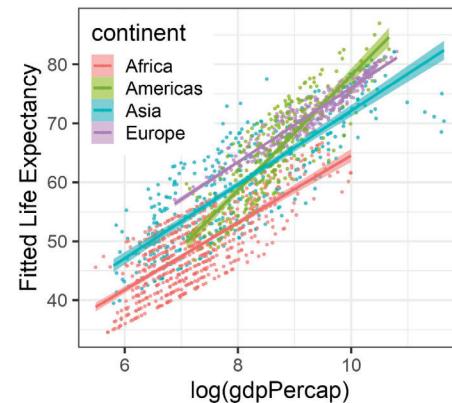


48

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



49

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

50

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 |

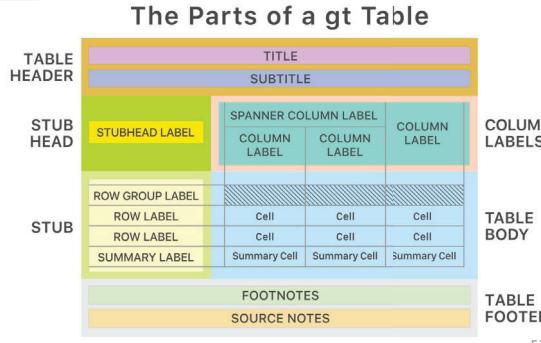
Too many decimals are used here, but you can control all that

51

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()
```



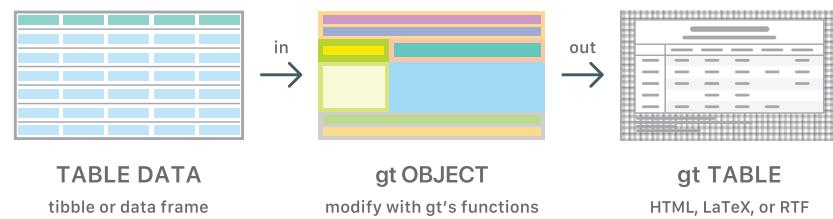
52

{gt} workflow

- Data table -> gt_tbl object

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

A Typical gt Workflow



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

53

```
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 Gaspé 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 |

54

```
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)) %>%
```

Add table column spanning headers

| Anderson's Iris Data (Collected in the Gaspé 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 |

55

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

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

| Anderson's Iris Data (Collected in the Gaspé 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 |

56

```
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 |

See: <https://vincentarelbundock.github.io/tinytable/>



tinytable

tinytable is a lightweight but full-featured table generator

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

57

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
```

58

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, ...

59

R Studio projects

R Studio projects are a handy way to organize your work

The .Rproj item opens the project in R Studio

60

R Studio projects

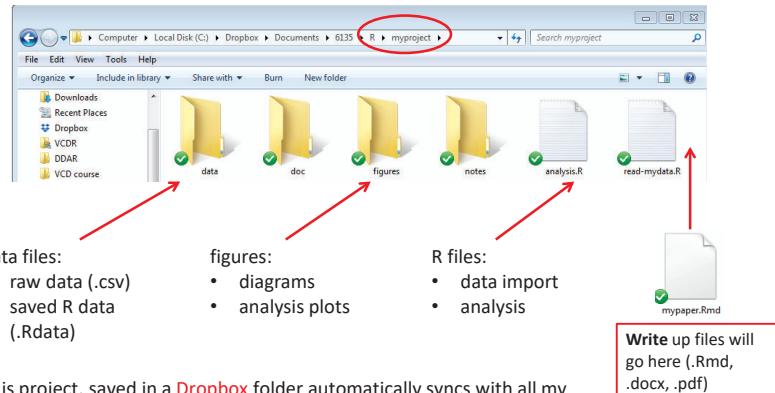
An R Studio project for a **research paper**: R files (scripts), Rmd files (text, R “chunks”)

The .Rproj item opens the project in R Studio

61

Organizing an R project

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



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

62

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")
```

63

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)

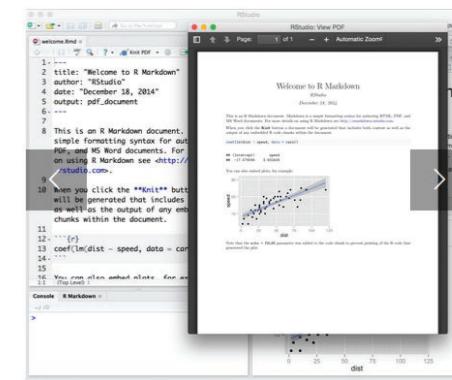
#' ## fit models
mymod.1 <- lm(y ~ X1 + X2 + X3, data=mydata)

#' ## 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

64

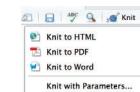
Reproducible analysis & reporting



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

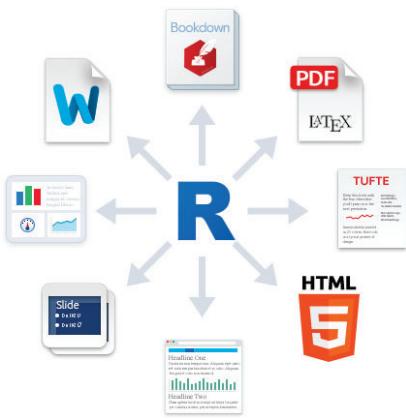
.Rmd files are just text files, using rmarkdown markup and knitr to run R on "code chunks"

A given document can be rendered in different output formats:



65

Output formats and templates



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

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

The ggplot2 book was written using .Rmd format.

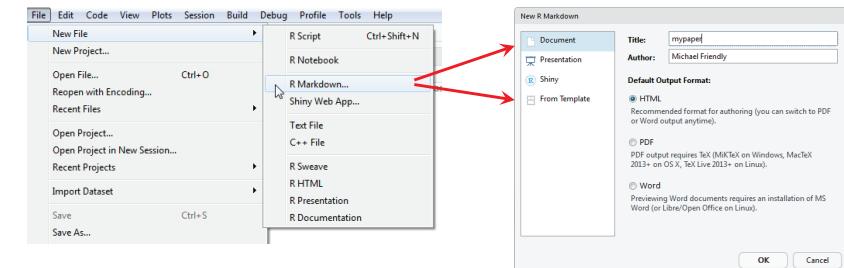
The [bookdown](#) package makes it easier to manage a book-length project – TOC, fig/table #s, cross-references, etc.
Also: [blogdown](#), [posterdown](#), ...

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

66

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



67

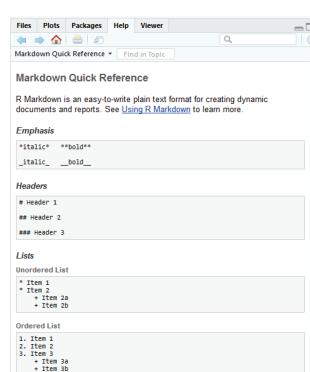
Writing it up

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

`mypaper.Rmd`, created from a template

```
analysis.R  read-mydata.R  mypaperRmd.R
1  title: "mypaper"
2  author: "Michael Friendly"
3  date: "January 29, 2018"
4  output: html_document
5
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
15 details on using R Markdown see <http://rmarkdown.rstudio.com>.
16 When you click the **Knit** button a document will be generated
17 R code chunks within the document. You can embed an R code chunk
18
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```
yaml header
Header 2
output code chunk
plot code chunk
```

Help -> Markdown quick reference



68

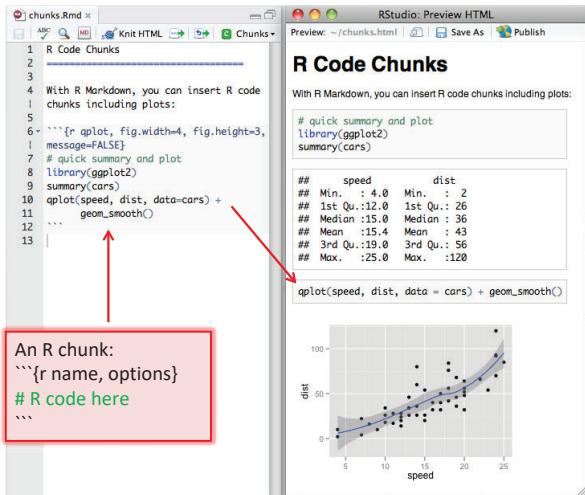
Rmarkdown basics

Rmarkdown uses simple formatting for all standard document elements

69

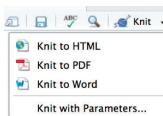
R code chunks

R code chunks are run by `knitr`, and the results are inserted in the output document



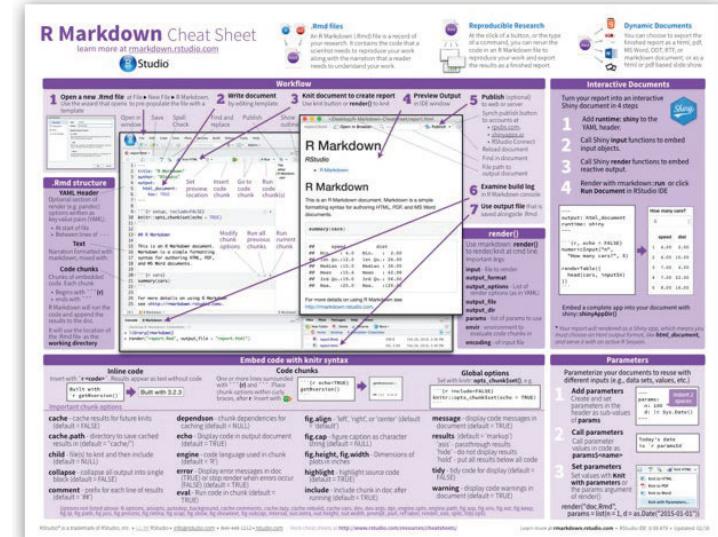
There are many options for controlling the details of chunk output – numbers, tables, graphs

Choose the output format:



70

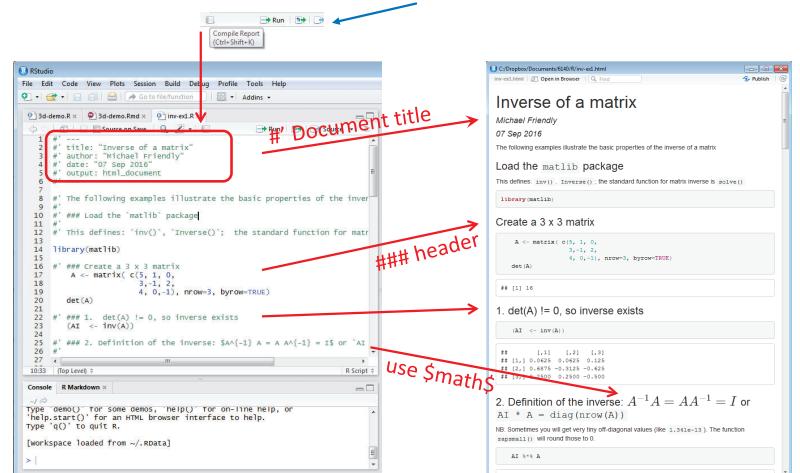
The R Markdown Cheat Sheet provides most of the details
<https://www.rstudio.com/wp-content/uploads/2016/03/rmarkdown-cheatsheet-2.0.pdf>



71

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



72



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)

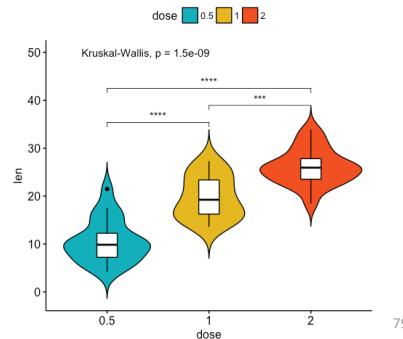
74

ggpubr

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

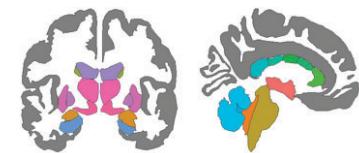
```
ggviolin(df, x = "dose", y = "len", fill = "dose",
  palette = c("#00AFBB", "#E7B800", "#FC4E07"),
  add = "boxplot", add.params = list(fill = "white")) +
  stat_compare_means(comparisons = my_comparisons, label = "p.signif") +
  stat_compare_means(label.y = 50)
```

see the examples at
<http://www.sthda.com/english/rpkgs/ggpubr/>



79

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))
```

80