

# Coding Resources (general)

keyword searches (\*stack overflow\*)

slack channel / colleagues

platform websites, ebooks

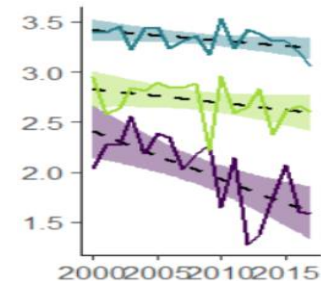
in-lab code reviews

# Recent web queries:

stack overflow:

- [How to use superscript with ggplot2](#)
- [Extracting the last n characters from a string in R](#)
- [Manually colouring plots with `scale\\_fill\\_manual` in ggplot2](#)
- [Construct a manual legend for a complicated plot](#)
- [Adding images below x-axis labels in ggplot2](#)

sthda.com



# Recent slack notes:

## 1. Function for “not in”:

because sometimes I forget it: ``%notin%` = function(x,y) !(x %in% y)`

## 1. Dealing with decimal places

WTF ggplot why would I want contour labels with 12 digits?

soln: `geom_text_contour(aes(label=scaleFUN(..level..)))`

also: `label_number(accuracy = 0.01)`

`label = label_number(suffix = "\u00b0C")` for °C

## 1. Dealing with dates: applying regular argument params to function

`as.mdy<-function(x) {as.Date(x,format="%m/%d/%Y") }`

`mutate_at(vars(end_with("Date")), as.mdy))`

# Resources from Rstudio.com

<https://www.rstudio.com/resources/cheatsheets/>

<https://rmarkdown.rstudio.com/lesson-1.html>

<https://www.rstudio.com/resources/webinars/>

<https://www.rstudio.com/resources/rstudioglobal-2021/>

Webinars & Videos

Data Science Hangout

Cheatsheets

Books

Education

Certified Partners

In-Person Workshops

RStudio Documentation

RStudio Blog

R Views Blog

AI Blog

Tidyverse Blog

# cheatsheets for R

## Data transformation with dplyr :: CHEAT SHEET



dplyr functions work with pipes and expect tidy data. In tidy data:



Each variable is in its own column

&



Each observation, or case, is in its own row



pipes

$x \%>\% f(y)$   
becomes  $f(x, y)$

### Summarise Cases

Apply summary functions to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function



**summarise(.data, ...)**  
Compute table of summaries.  
`summarise(mtcars, avg = mean(mpg))`

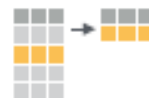


**count(.data, ..., wt = NULL, sort = FALSE, name = NULL)** Count number of rows in each group defined by the variables in ... Also `tally()`.  
`count(mtcars, cyl)`

### Manipulate Cases

#### EXTRACT CASES

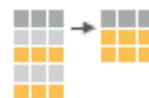
Row functions return a subset of rows as a new table.



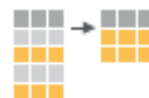
**filter(.data, ..., .preserve = FALSE)** Extract rows that meet logical criteria.  
`filter(mtcars, mpg > 20)`



**distinct(.data, ..., .keep\_all = FALSE)** Remove rows with duplicate values.  
`distinct(mtcars, gear)`



**slice(.data, ..., .preserve = FALSE)** Select rows by position.  
`slice(mtcars, 10:15)`



**slice\_sample(.data, ..., n, prop, weight\_by = NULL, replace = FALSE)** Randomly select rows. Use `n` to select a number of rows and `prop` to select a fraction of rows.  
`slice_sample(mtcars, n = 5, replace = TRUE)`

**slice\_min(.data, order\_by, ..., n, prop, with\_ties = TRUE)** and **slice\_max()** Select rows

### Manipulate Variables

#### EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.



**pull(.data, var = -1, name = NULL, ...)** Extract column values as a vector, by name or index.  
`pull(mtcars, wt)`



**select(.data, ...)** Extract columns as a table.  
`select(mtcars, mpg, wt)`



**relocate(.data, ..., .before = NULL, .after = NULL)** Move columns to new position.  
`relocate(mtcars, mpg, cyl, .after = last_col())`

Use these helpers with `select()` and `across()`  
e.g. `select(mtcars, mpg:cyl)`

<code>contains(match)</code>	<code>num_range(prefix, range)</code>	<code>!</code> , e.g. <code>mpg:cyl</code>
<code>ends_with(match)</code>	<code>all_of(x)/any_of(x, ..., vars)</code>	<code>~</code> , e.g. <code>-gear</code>
<code>starts_with(match)</code>	<code>matches(match)</code>	<code>everything()</code>

# cheatsheets for R

<https://www.rstudio.com/resources/cheatsheets/>

## Apply functions with purrr : : CHEAT SHEET

### Map Functions

#### ONE LIST

**map(.x, .f, ...)** Apply a function to each element of a list or vector, return a list.  
`x <- list(1:10, 11:20, 21:30)`  
`l1 <- list(x = c("a", "b"), y = c("c", "d"))`  
`map(l1, sort, decreasing = TRUE)`



**map\_dbl(.x, .f, ...)**  
Return a double vector.  
`map_dbl(x, mean)`



**map\_int(.x, .f, ...)**  
Return an integer vector.  
`map_int(x, length)`



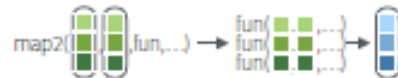
**map\_chr(.x, .f, ...)**  
Return a character vector.  
`map_chr(l1, paste, collapse = "")`



**map\_lgl(.x, .f, ...)**  
Return a logical vector.  
`map_lgl(x, is.integer)`

#### TWO LISTS

**map2(.x, .y, .f, ...)** Apply a function to pairs of elements from two lists or vectors, return a list.  
`y <- list(1, 2, 3); z <- list(4, 5, 6); l2 <- list(x = "a", y = "z")`  
`map2(x, y, ~ x * y)`



**map2\_dbl(.x, .y, .f, ...)**  
Return a double vector.  
`map2_dbl(y, z, ~ x / y)`



**map2\_int(.x, .y, .f, ...)**  
Return an integer vector.  
`map2_int(y, z, ~ '+' )`



**map2\_chr(.x, .y, .f, ...)**  
Return a character vector.  
`map2_chr(l1, l2, paste, collapse = "", sep = ":")`



**map2\_lgl(.x, .y, .f, ...)**  
Return a logical vector.  
`map2_lgl(l2, l1, ~ %in% )`

#### MANY LISTS

**pmap(.l, .f, ...)** Apply a function to groups of elements from a list of lists or vectors, return a list.  
`pmap(list(x, y, z), ~ ..1 * (..2 + ..3))`



**pmap\_dbl(.l, .f, ...)**  
Return a double vector.  
`pmap_dbl(list(y, z), ~ x / y)`



**pmap\_int(.l, .f, ...)**  
Return an integer vector.  
`pmap_int(list(y, z), ~ '+' )`



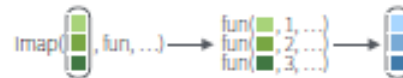
**pmap\_chr(.l, .f, ...)**  
Return a character vector.  
`pmap_chr(list(l1, l2), paste, collapse = "", sep = ":")`



**pmap\_lgl(.l, .f, ...)**  
Return a logical vector.  
`pmap_lgl(list(l2, l1), ~ %in% )`

#### LISTS AND INDEXES

**imap(.x, .f, ...)** Apply .f to each element and its index, return a list.  
`imap(y, ~ paste0(y, ":", ..x))`



**imap\_dbl(.x, .f, ...)**  
Return a double vector.  
`imap_dbl(y, ~ y)`



**imap\_int(.x, .f, ...)**  
Return an integer vector.  
`imap_int(y, ~ y)`



**imap\_chr(.x, .f, ...)**  
Return a character vector.  
`imap_chr(y, ~ paste0(y, ":", ..x))`



**imap\_lgl(.x, .f, ...)**  
Return a logical vector.  
`imap_lgl(l1, ~ is.character(y))`



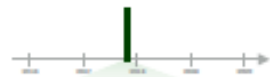
# cheatsheets for R

<https://www.rstudio.com/resources/cheatsheets/>

## Dates and times with lubridate :: CHEAT SHEET



### Date-times



2017-11-28 12:00:00

2017-11-28 12:00:00

A **date-time** is a point on the timeline, stored as the number of seconds since 1970-01-01 00:00:00 UTC

```
dt <- as_datetime(1511870400)
## "2017-11-28 12:00:00 UTC"
```

2017-11-28

A **date** is a day stored as the number of days since 1970-01-01

```
d <- as_date(17498)
## "2017-11-28"
```

12:00:00

An **hms** is a **time** stored as the number of seconds since 00:00:00

```
t <- hms::as_hms(85)
## 00:01:25
```

### 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 **tz** argument to set the time zone, e.g. `ymd(x, tz = "UTC")`.

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")`

### GET AND SET COMPONENTS

Use an accessor function to get a component.  
Assign into an accessor function to change a component in place.

```
d ## "2017-11-28"
day(d) ## 28
day(d) <- 1
d ## "2017-11-01"
```

2018-01-31 11:59:59

`date(x)` Date component. `date(dt)`

2018-01-31 11:59:59

`year(x)` Year. `year(dt)`  
`isoyear(x)` The ISO 8601 year.  
`epiyear(x)` Epidemiological year.

2018-01-31 11:59:59

`month(x, label, abbr)` Month.  
`month(dt)`

2018-01-31 11:59:59

`day(x)` Day of month. `day(dt)`  
`wday(x, label, abbr)` Day of week.  
`qday(x)` Day of quarter.

2018-01-31 11:59:59

`hour(x)` Hour. `hour(dt)`

2018-01-31 11:59:59

`minute(x)` Minutes. `minute(dt)`

2018-01-31 11:59:59

`second(x)` Seconds. `second(dt)`

### Round Date-times



`floor_date(x, unit = "second")`  
Round down to nearest unit.  
`floor_date(dt, unit = "month")`

`round_date(x, unit = "second")`  
Round to nearest unit.  
`round_date(dt, unit = "month")`

`ceiling_date(x, unit = "second", change_on_boundary = NULL)`  
Round up to nearest unit.  
`ceiling_date(dt, unit = "month")`

Valid units are second, minute, hour, day, week, month, bmonth, quarter, season, halfyear and year.

`rollback(dates, roll_to_first = FALSE, preserve_hms = TRUE)` Roll back to last day of previous month. Also `rollforward()`, `rollback(dt)`

### Stamp Date-times

`stamp()` Derive a template from an example string and return a new function that will apply the template to date-times. Also `stamp_date()` and `stamp_time()`.

1. Derive a template, create a function  
`sf <- stamp("Created Sunday, Jan 17, 1999 3:34")`
2. Apply the template to dates  
`sf(ymd("2010-04-05"))`  
`## [1] "Created Monday, Apr 05, 2010 00:00"`

Tip: use a date with day > 12

### Time Zones



# cheatsheets for R

<https://www.rstudio.com/resources/cheatsheets/>

## Use Python with R with reticulate :: CHEAT SHEET

The reticulate package lets you use Python and R together seamlessly in R code, in R Markdown documents, and in the RStudio IDE.

### Python in R Markdown

(Optional) Build Python env to use.

Add `knitr::knit_engines$set(python = reticulate::eng_python)` to the setup chunk to set up the reticulate Python engine (not required for knitr >= 1.18).

Suggest the Python environment to use, in your setup chunk.

Begin Python chunks with ````{python}`. Chunk options like `echo`, `include`, etc. all work as expected.

Use the `py` object to access objects created in Python chunks from R chunks.

Python chunks all execute within a single Python session so you have access to all objects created in previous chunks.

Use the `r` object to access objects created in R chunks from Python chunks.

Output displays below chunk, including matplotlib plots.

```
1- ```{r setup, include = FALSE}
2- library(reticulate)
3- virtualenv_create("fMRI-proj")
4- py_install("seaborn", envname = "fMRI-proj")
5- use_virtualenv("fMRI-proj")
6- ...
7-
8- ```{python, echo = FALSE}
9- import seaborn as sns
10- fmri = sns.load_dataset("fmri")
11- ...
12-
13- ```{r}
14- f1 <- subset(py$fmri, region == "parietal")
15- ...
16-
17- ```{python}
18- import matplotlib as mpl
19- sns.lmplot("timepoint", "signal", data=r.f1)
20- mpl.pyplot.show()
21- ...
```

```
1 library(reticulate)
2 py_install("seaborn")
3 use_virtualenv("r-reticulate")
4
5 sns <- import("seaborn")
6
7 fmri <- sns.load_dataset("fmri")
8 dim(fmri)
9
10 # creates tips
11 source_python("python.py")
12 dim(tips)
13
14 # creates tips in main
15 py_run_file("python.py")
16 dim(py$tips)
17
18 py_run_string("print(tips.shape)")
19
```

### Python in R

Call Python from R code in three ways:

#### IMPORT PYTHON MODULES

Use `import()` to import any Python module. Access the attributes of a module with `$`.

- `import(module, as = NULL, convert = TRUE, delay_load = FALSE)` Import a Python module. If `convert = TRUE`, Python objects are converted to their equivalent R types. Also `import_from_path()`. `import("pandas")`
- `import_main(convert = TRUE)` Import the main module, where Python executes code by default. `import_main()`
- `import_builtins(convert = TRUE)` Import Python's built-in functions. `import_builtins()`

#### SOURCE PYTHON FILES

Use `source_python()` to source a Python script and make the Python functions and objects it creates available in the calling R environment.





# cheatsheets for R

<https://www.rstudio.com/resources/cheatsheets/>

## Data visualization with ggplot2 :: CHEAT SHEET

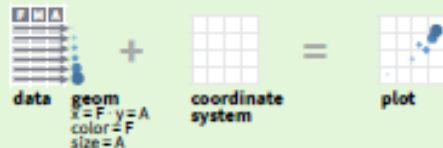


### Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and **geoms**—visual marks that represent data points.



To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.



Complete the template below to build a graph.

`ggplot(data = <DATA>) +`  
`<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),`  
`stat = <STAT>, position = <POSITION>) +`

required  
Not required

### Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables.  
Each function returns a layer.

#### GRAPHICAL PRIMITIVES

`a <- ggplot(economics, aes(date, unemploy))`  
`b <- ggplot(seals, aes(x = long, y = lat))`

- `a + geom_blank()` and `a + expand_limits()`  
Ensure limits include values across all plots.
- `b + geom_curve(aes(yend = lat + 1, xend = long + 1, curvature = 1))` - x, xend, y, yend, alpha, angle, color, curvature, linetype, size
- `a + geom_path(lineend = "butt", linejoin = "round", linemitre = 1)` - x, y, alpha, color, group, linetype, size
- `a + geom_polygon(aes(alpha = 50))` - x, y, alpha, color, fill, group, subgroup, linetype, size
- `b + geom_rect(aes(xmin = long, ymin = lat, xmax = long + 1, ymax = lat + 1))` - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size
- `a + geom_ribbon(aes(ymin = unemploy - 900, ymax = unemploy + 900))` - x, ymax, ymin, alpha, color, fill, group, linetype, size

#### LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

- `b + geom_abline(aes(intercept = 0, slope = 1))`
- `b + geom_hline(aes(yintercept = lat))`
- `b + geom_vline(aes(xintercept = long))`

#### TWO VARIABLES

both continuous

`e <- ggplot(mpg, aes(cty, hwy))`

- `e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1)` - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust
- `e + geom_point()` - x, y, alpha, color, fill, shape, size, stroke
- `e + geom_quantile()` - x, y, alpha, color, group, linetype, size, weight
- `e + geom_rug(sides = "bl")` - x, y, alpha, color, linetype, size
- `e + geom_smooth(method = lm)` - x, y, alpha, color, fill, group, linetype, size, weight
- `e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1)` - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

one discrete, one continuous

`f <- ggplot(mpg, aes(class, hwy))`

continuous bivariate distribution

`h <- ggplot(diamonds, aes(carat, price))`

- `h + geom_bin2d(binwidth = c(0.25, 500))` - x, y, alpha, color, fill, linetype, size, weight
- `h + geom_density_2d()` - x, y, alpha, color, group, linetype, size
- `h + geom_hex()` - x, y, alpha, color, fill, size

continuous function

`i <- ggplot(economics, aes(date, unemploy))`

- `i + geom_area()` - x, y, alpha, color, fill, linetype, size
- `i + geom_line()` - x, y, alpha, color, group, linetype, size
- `i + geom_step(direction = "hv")` - x, y, alpha, color, group, linetype, size

visualizing error

`df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)`  
`j <- ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se))`

# cheatsheets for R

<https://www.rstudio.com/resources/cheatsheets/>

## Data tidying with tidyr :: CHEAT SHEET

**Tidy data** is a way to organize tabular data in a consistent data structure across packages.

A table is tidy if:



Each **variable** is in its own **column**

&



Each **observation**, or **case**, is in its own **row**



Access **variables** as **vectors**



Preserve **cases** in vectorized operations

### Tibbles

AN ENHANCED DATA FRAME

Tibbles are a table format provided by the **tibble** package. They inherit the data frame class, but have improved behaviors:

- **Subset** a new tibble with `[]`, a vector with `[[` and `$`.
- **No partial matching** when subsetting columns.
- **Display** concise views of the data on one screen.



### Reshape Data

- Pivot data to reorganize values into a new layout.

table4a

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

→

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

**pivot\_longer**(data, cols, names\_to = "name", values\_to = "value", values\_drop\_na = FALSE)

"Lengthen" data by collapsing several columns into two. Column names move to a new names\_to column and values to a new values\_to column.

`pivot_longer(table4a, cols = 2:3, names_to = "year", values_to = "cases")`

table2

country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
A	2000	cases	2K
A	2000	pop	20M
B	1999	cases	37K
B	1999	pop	172M
B	2000	cases	80K
B	2000	pop	174M
C	1999	cases	212K
C	1999	pop	1T
C	2000	cases	213K
C	2000	pop	1T

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172M
B	2000	80K	174M
C	1999	212K	1T
C	2000	213K	1T

**pivot\_wider**(data, names\_from = "name", values\_from = "value")

The inverse of `pivot_longer()`. "Widen" data by expanding two columns into several. One column provides the new column names, the other the values.

`pivot_wider(table2, names_from = type, values_from = count)`

### Split Cells

- Use these functions to split or combine cells into individual, isolated values.

### Expand Tables

Create new combinations of variables or identify implicit missing values (combinations of variables not present in the data).

x

x1	x2	x3
A	1	3
B	1	4
B	2	3

→

x1	x2
A	1
A	2
B	1
B	2

**expand**(data, ...) Create a new tibble with all possible combinations of the values of the variables listed in ... Drop other variables.  
`expand(mtcars, cyl, gear, carb)`

x

x1	x2	x3
A	1	3
B	1	4
B	2	3

→

x1	x2	x3
A	1	3
A	2	NA
B	1	4
B	2	3

**complete**(data, ..., fill = list()) Add missing possible combinations of values of variables listed in ... Fill remaining variables with NA.  
`complete(mtcars, cyl, gear, carb)`

### Handle Missing Values

Drop or replace explicit missing values (NA).



# Contributed Cheatsheets

These cheatsheets have been generously contributed by R users.

## Advanced R



Environments, data structures, functions, subsetting and more, by Arianne Colton and Sean Chen. Updated February 2016.

DOWNLOAD

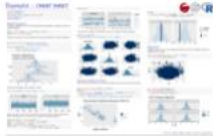
## Base R



Vectors, matrices, lists, data frames, functions and more in base R, by Mhairi McNeill. Updated March 2015.

DOWNLOAD

## bayesplot



Plotting for Bayesian Models in R, by Edward A. Roualdes. Updated May 2020.

DOWNLOAD

## BCEA



Bayesian cost effective analysis in R, by Gianluca Baio. Updated February 2021.

DOWNLOAD

## caret



Modeling and machine learning in R with the caret package, by Max Kuhn. Updated September 2017.

DOWNLOAD

## cartography



Thematic maps with spatial objects, by Timothée Giraud. Updated July 2018.

DOWNLOAD

## collapse



## data.table



## DeclareDesign



## jfa



Work with bayesian and classical statistical audit samples, by Koen Derks. Updated September 2021.

DOWNLOAD

## labelled



Manipulate labelled data, by Joseph Larmarange. Updated June 2020.

DOWNLOAD

## LaTeX



A reference to the LaTeX typesetting language, useful in combination with knitr and R Markdown, by Winston Chang. Updated January 2018.

DOWNLOAD

## leaflet



Interactive maps in R with leaflet, by Kejia Shi. Updated May 2017.

DOWNLOAD

## Machine Learning Modelling



A tabular guide to machine learning algorithms in R, by Arnaud Amsellem. Updated March 2018.

DOWNLOAD

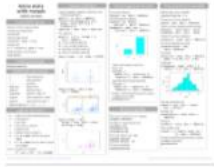
## mlr



The mlr package offers a unified interface to R's machine learning capabilities, by Aaron Cooley. Updated February 2018.

DOWNLOAD

## mosaic



The mosaic package is for teaching mathematics, statistics, computation and

## nardl



The nardl package estimates the nonlinear cointegrating autoregressive distributed

## nimble



Hierarchical statistical models that extend BUGS and JAGS, by the Nimble

# Visualizations with ggplot2 (K. Woo, accidental aRt)

- Data first grammar of graphics
- Common problems: mapping mishaps, scale snafus, theme threats
  - theme customization always defaults to most specific input
- [ggplot2-book.org](https://ggplot2-book.org)

<https://www.rstudio.com/resources/rstudioglobal-2021/always-look-on-the-bright-side-of-plots/>



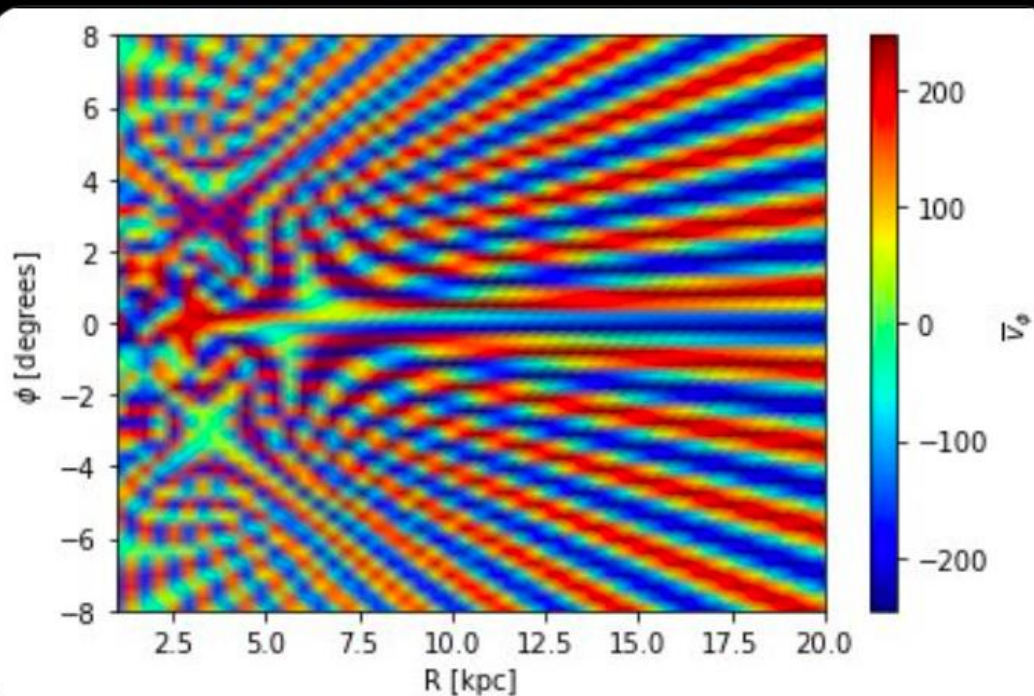
# @accidental\_aRt

↻ accidental aRt Retweeted



**Pau Ramos** @PBR117 · Jun 15, 2021

Everything in this plot is wrong (literally everything) and yet...There is some beauty in it! I'd say it qualifies as @accidental\_aRt

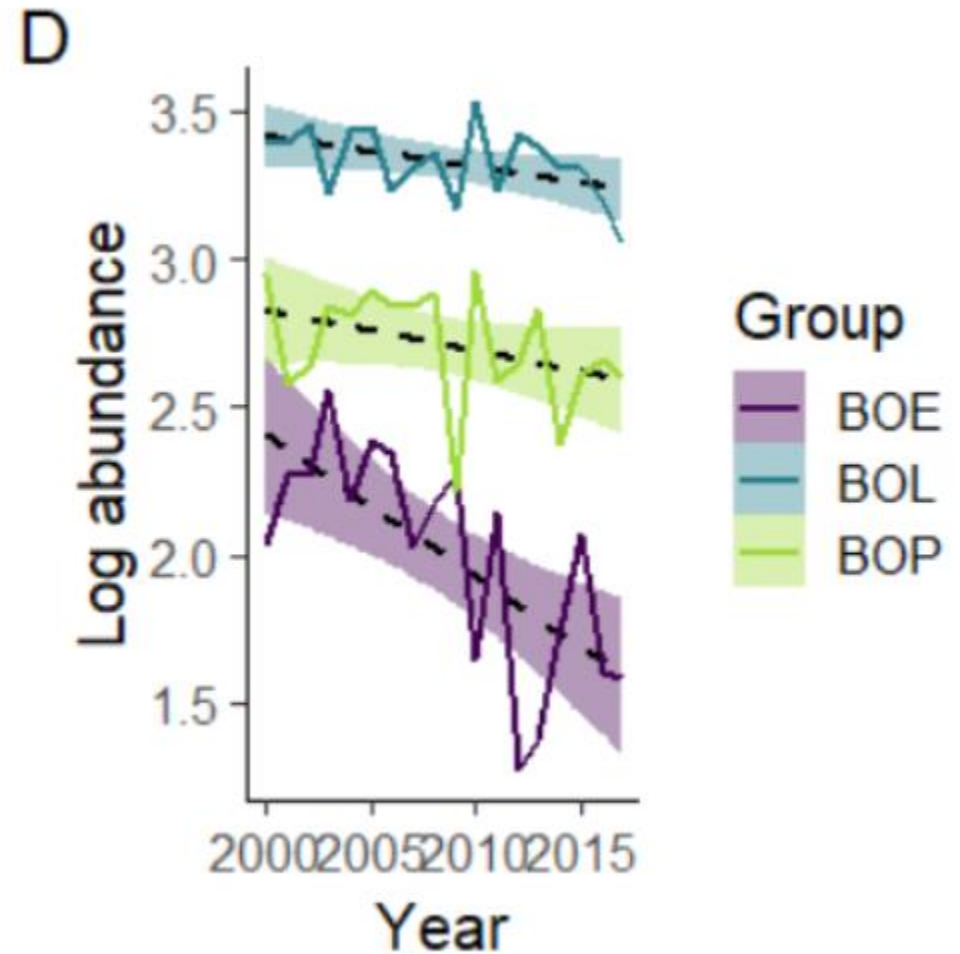
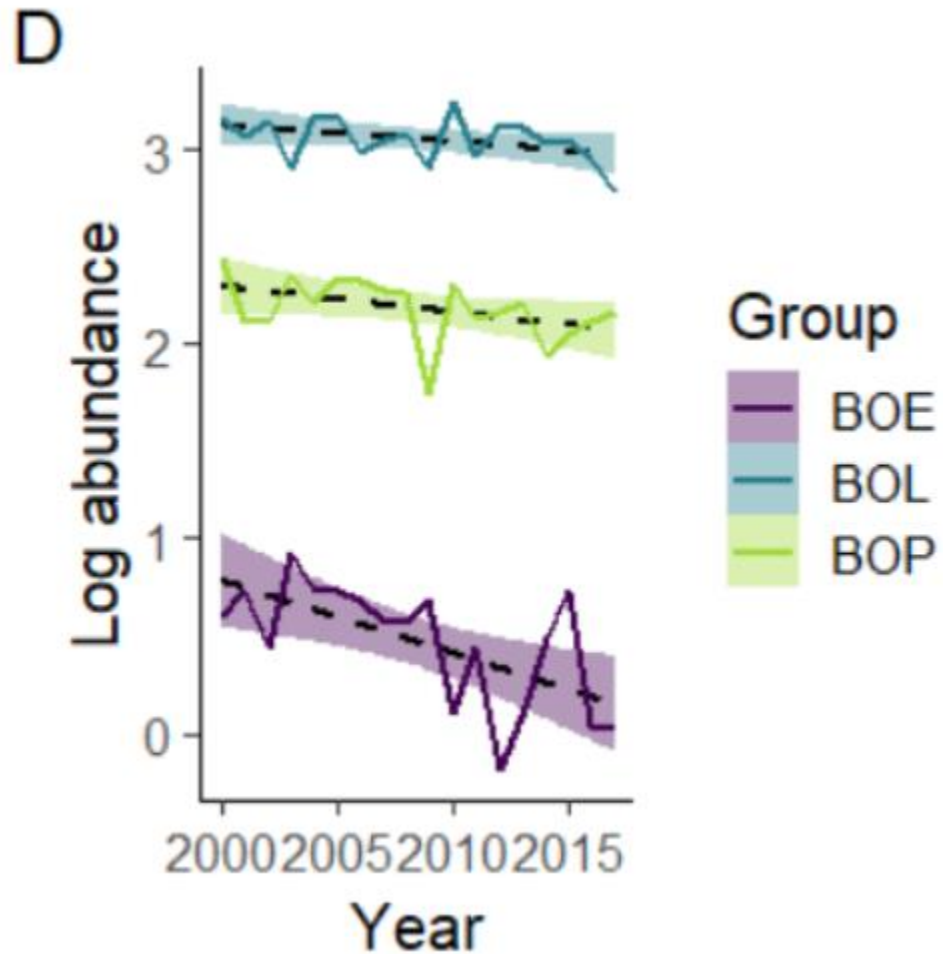


**Kanishka Misra** @kanishkamisra · Jan 25, 2021

"Definitely Accidental" art @accidental\_aRt



Same underlying data ... different order of operations





# Dealing with code maintenance in the light of version changes

(H. Wickham)

- Reframing frustrations with broken code & documentation
- Using best applicable functions (avoiding off-label uses)
- Tools to isolate environment to specific package versions

<https://www.rstudio.com/resources/rstudioglobal-2021/maintaining-the-house-the-tidyverse-built/>

# Tools to isolate environment to specific package versions

One option is to use renv

```
# Isolate this project  
renv::init()
```

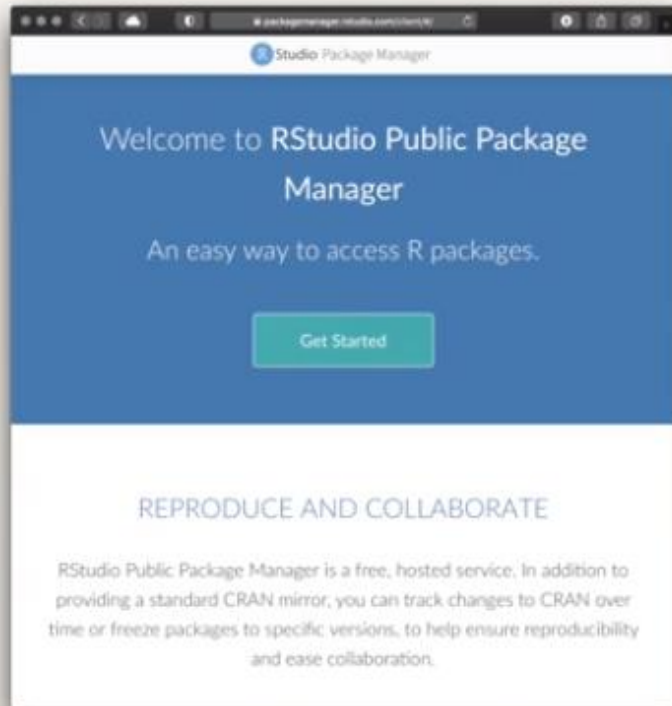
```
# Capture any recent package changes  
renv::snapshot()
```

```
# Elsewhere, restore exact environment  
renv::restore()
```

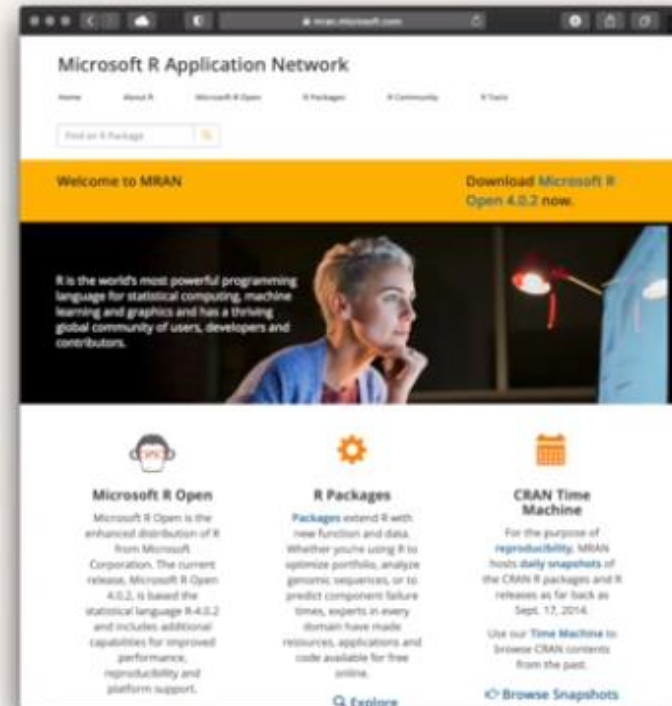


# Tools to isolate environment to specific package versions

Another option is to use a CRAN time capsule



<https://packagemanager.rstudio.com>



<https://mran.microsoft.com>

# Possible “homework” ideas for next discussion:

Watch 1 video from Rstudio or other instructional site

Report back on main points / key tips

20 minute R plot

- Pick a dataset using `data()`
- Define a question / hypothesis
- Consider filters, aggregations, transformations
- Create a relevant data visualization

Share our code to explore different approaches