R for the Infrequent UseR

Shannon Dunnigan

GTMNERR

Thursday, September 23, 2021

Who am I?

Why use R?

- It’s FREE
- Integration
- Reproducibility
- Automation
- Community of support

Thomas Mock @thomas_mock · Mar 21

Finalizing the slides on my #rstats presentation "Finding the YOU in the R community"

So I have a questions for the R community (at least on Twitter):

❓ "Why be an involved useR?"

15 7 43

Kim Cressman
@swmpkim

Following

Replying to @thomas_mock

Because it's fun and you can learn SO MUCH, without having to seek it out, just by watching what other people do. I have a lot of "OMG, you can do THAT?" moments.

I also learn a lot from what other people *ask*.

And sometimes I can help someone else, which is always nice.

8:42 PM - 21 Mar 2019

3 Likes

3

From Dr. Thomas Mock's presentation, "A Gentle Introduction to Tidy Statistics in R"

Thomas Mock @thomas_mock · Mar 21

Finalizing the slides on my #rstats presentation "Finding the YOU in the R community"

So I have a questions for the R community (at least on Twitter):

❓ "Why be an involved useR?"

15 7 43

Luuuda
@ludmila_janda

Following

Replying to @thomas_mock

for all these feels:

🤯 omg you can do that in R now?!

✋ oh I think I can help with this issue

👉 oh man I've so been there

🤩 thank you so much for helping me with this thing i've been going crazy trying to figure out

📊 whoa awesome data viz

📚 ohhh sweet resources

6:37 PM - 21 Mar 2019

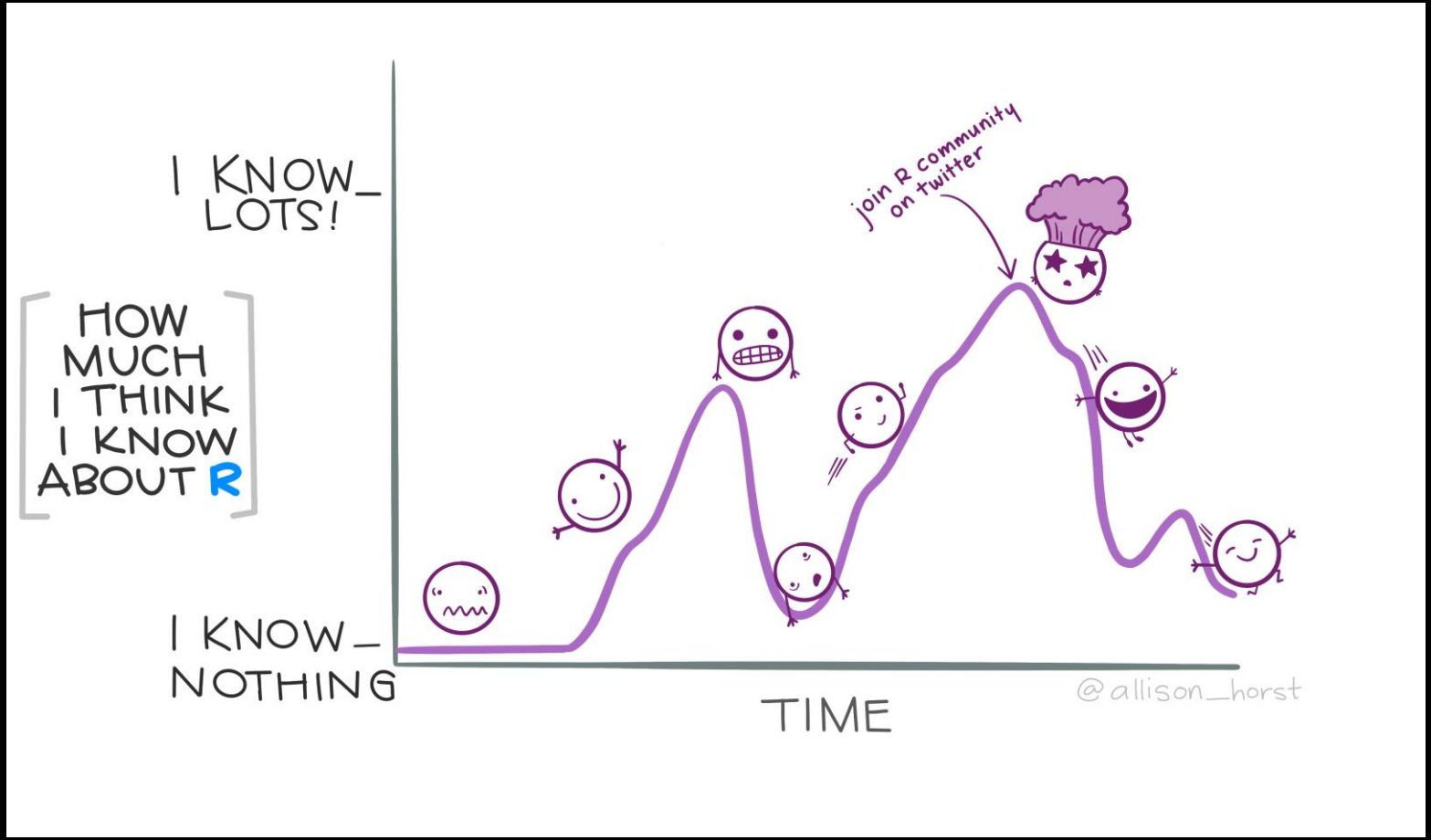
3 Retweets 22 Likes

1 3 22

Who are you?

"The Infrequent UseR"

R Knowledge Rollercoaster by Allison Horst



This Workshop

What we will cover

- Reintroduction
- Best Practice for Reproducible Workflows
- Making a reproducible report with R Markdown and RStudio
- How to get and ask for help

What we will *not* cover

- Data tidying and wrangling
- Statistical analyses
- Customization of R Markdown outputs

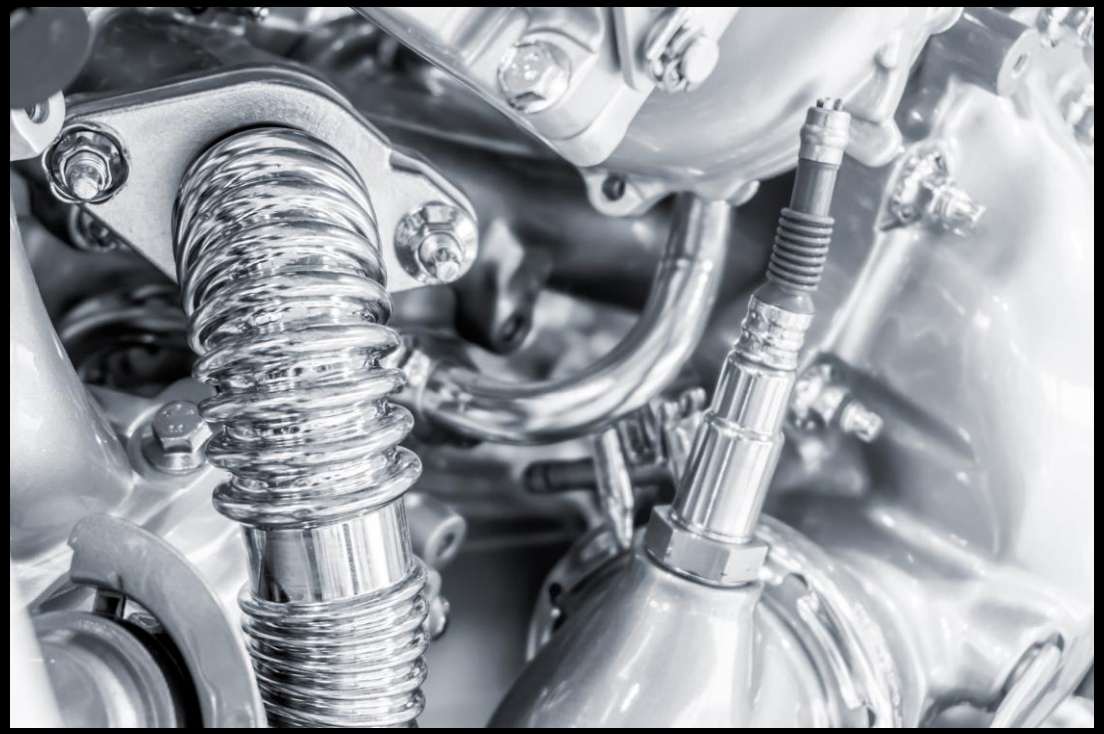
but stay tuned…

“Hello, R, it’s me. I know it’s been a while…”

A Reintroduction to R

R vs. RStudio

R



RStudio

R vs R Packages

R

R Packages

Before you begin…

We recommend versions R 4.0+ and RStudio 1.4+ for this workshop

- Check your versions of R and RStudio
- If you need to update:
  1. Manually install from CRAN (recommended if you do not care about old packages)
  2. Windows only: use `installr` package – run this through base R rather than RStudio
  3. MacOS only: use `updateR` - very similar to `installr` in Windows, but also requires the `devtools` package.

Packages

- Installing packages

```
# install the ggplot2 package
install.packages("ggplot2")

# install the ggplot2, here, readxl, and janitor packages
install.packages(c("ggplot2", "here", "readxl", "janitor"))

# install using a vector object
my_packages <- c("ggplot2", "here", "readxl", "janitor")
install.packages(my_packages)
```

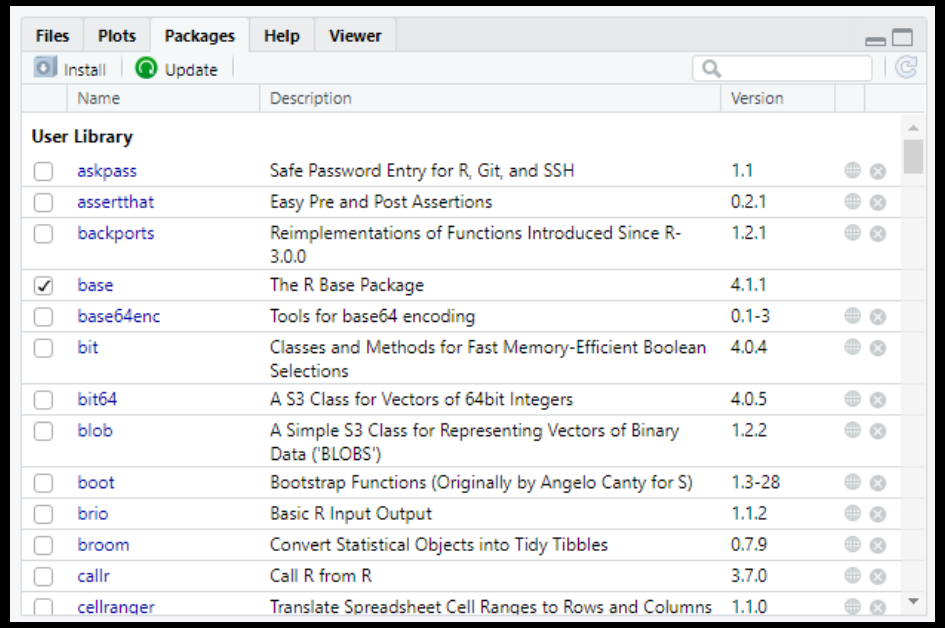
Or in RStudio, manage packages using Tools > Install Packages

Packages

- Installing packages
- How to check what packages are installed already

```
# check what packages are installed already
installed.packages()
```

  - In RStudio, use the “Packages” tab
    - User Library (top): additional packages
    - System Library (bottom): base packages

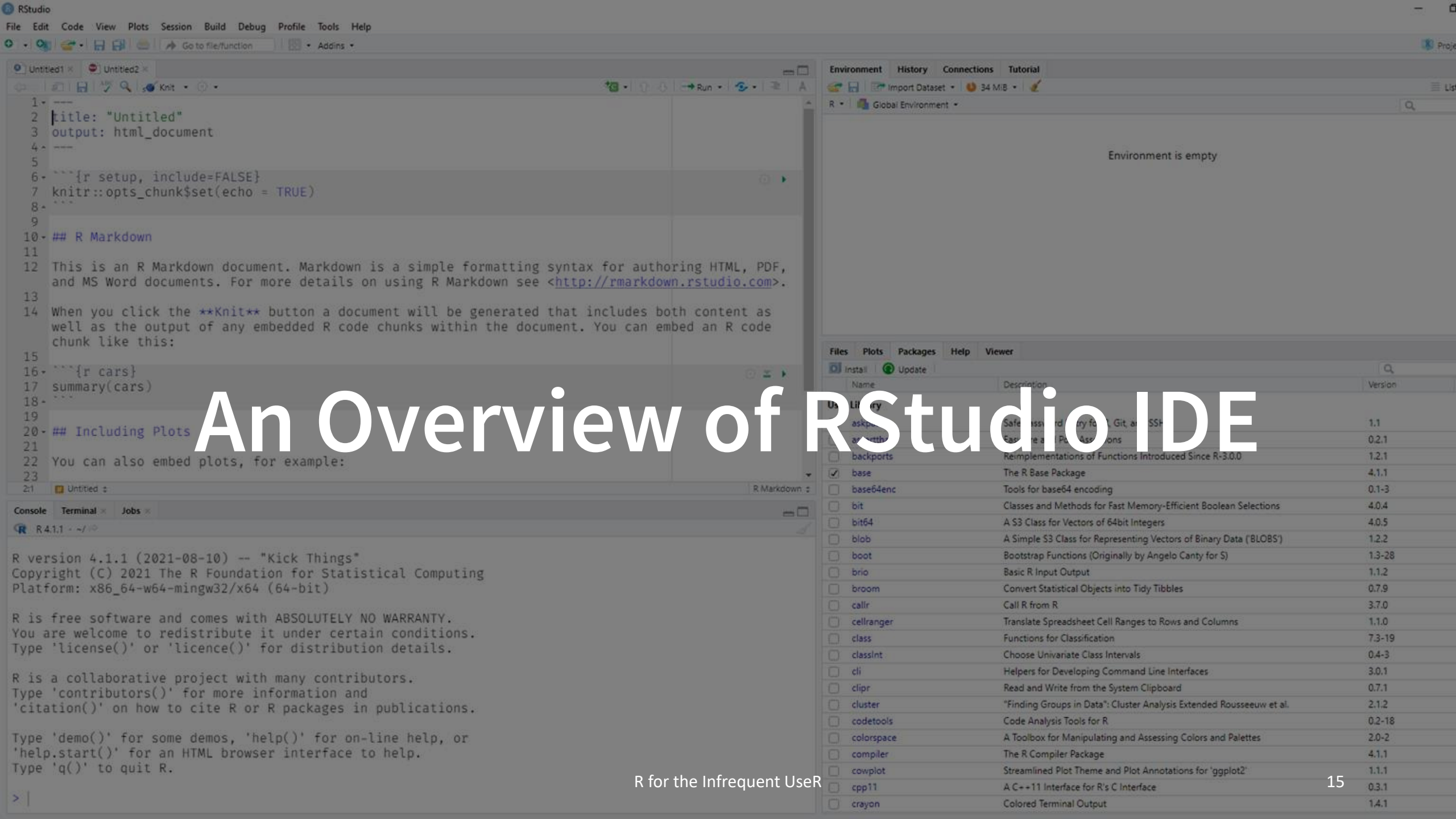


Packages

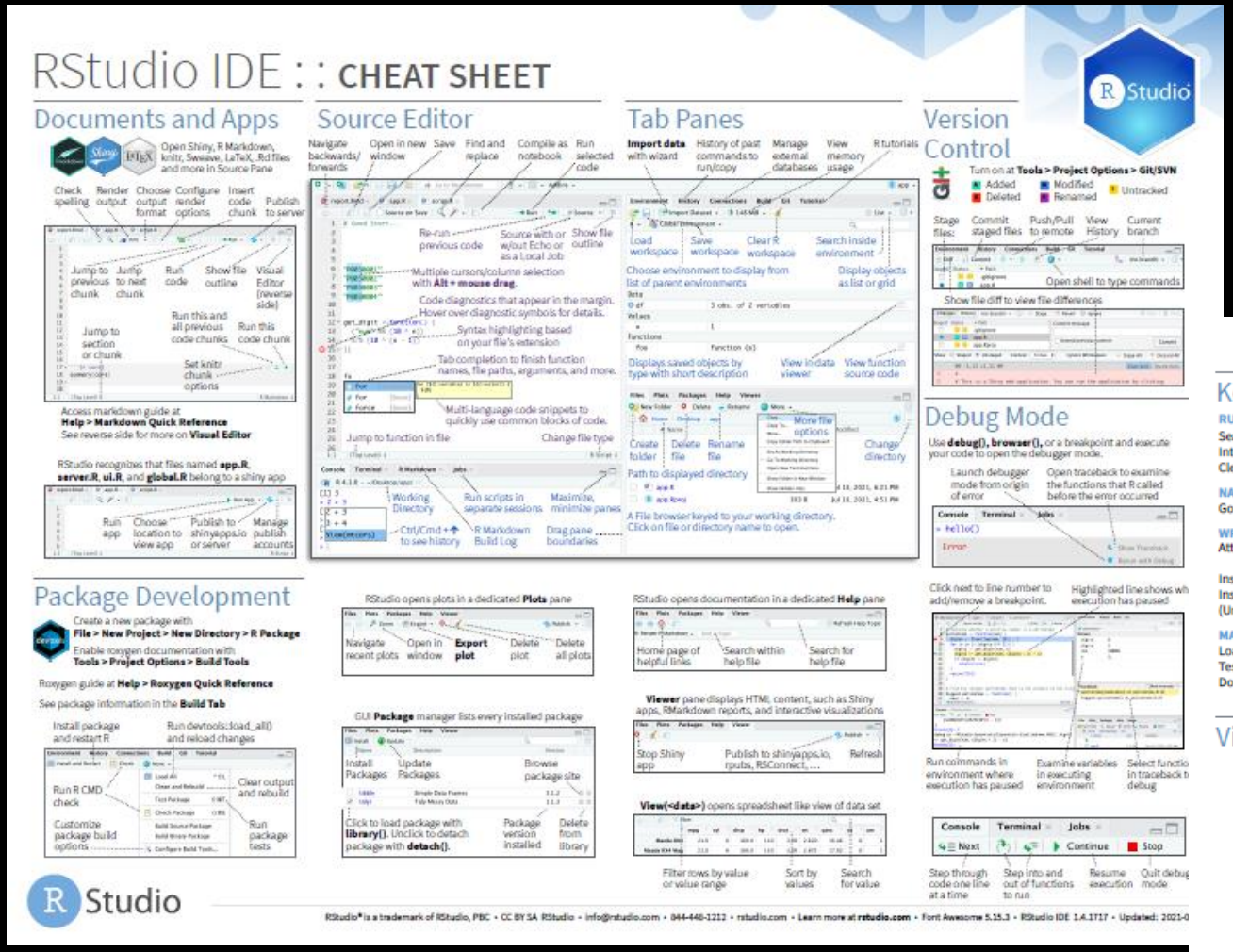
- Installing packages
- How to check what packages are installed already
- Updating packages

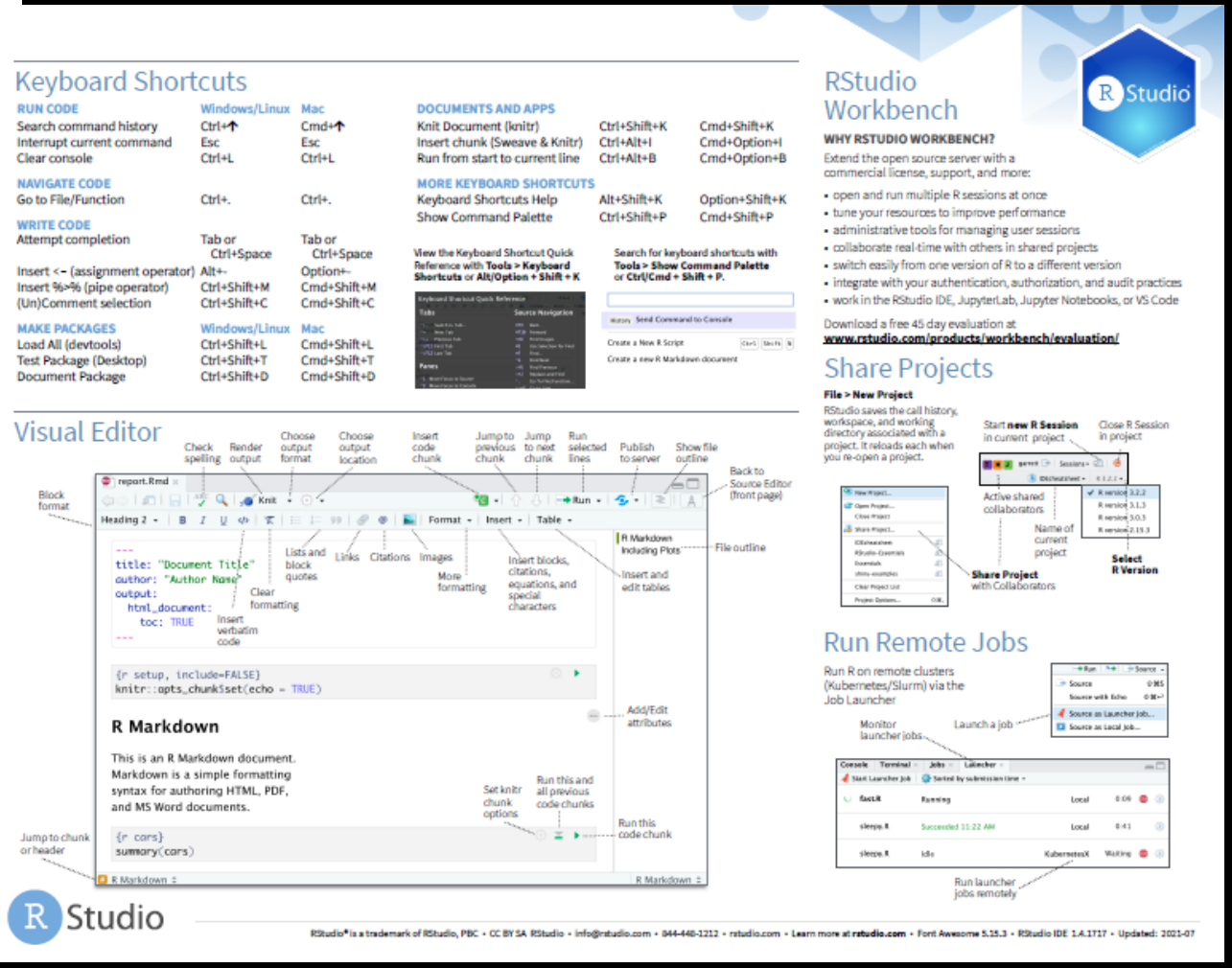
```
# list all packages where update is available
old.packages()

# update all available packages
update.packages()

# update all without prompts for permission/clarification
update.packages(ask = FALSE)
```

An Overview of RStudio IDE



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





A Reproducible Workflow

Adaptations from Jenny Bryan and Jim Hester's work
"*What They Forgot to Teach You About R*"

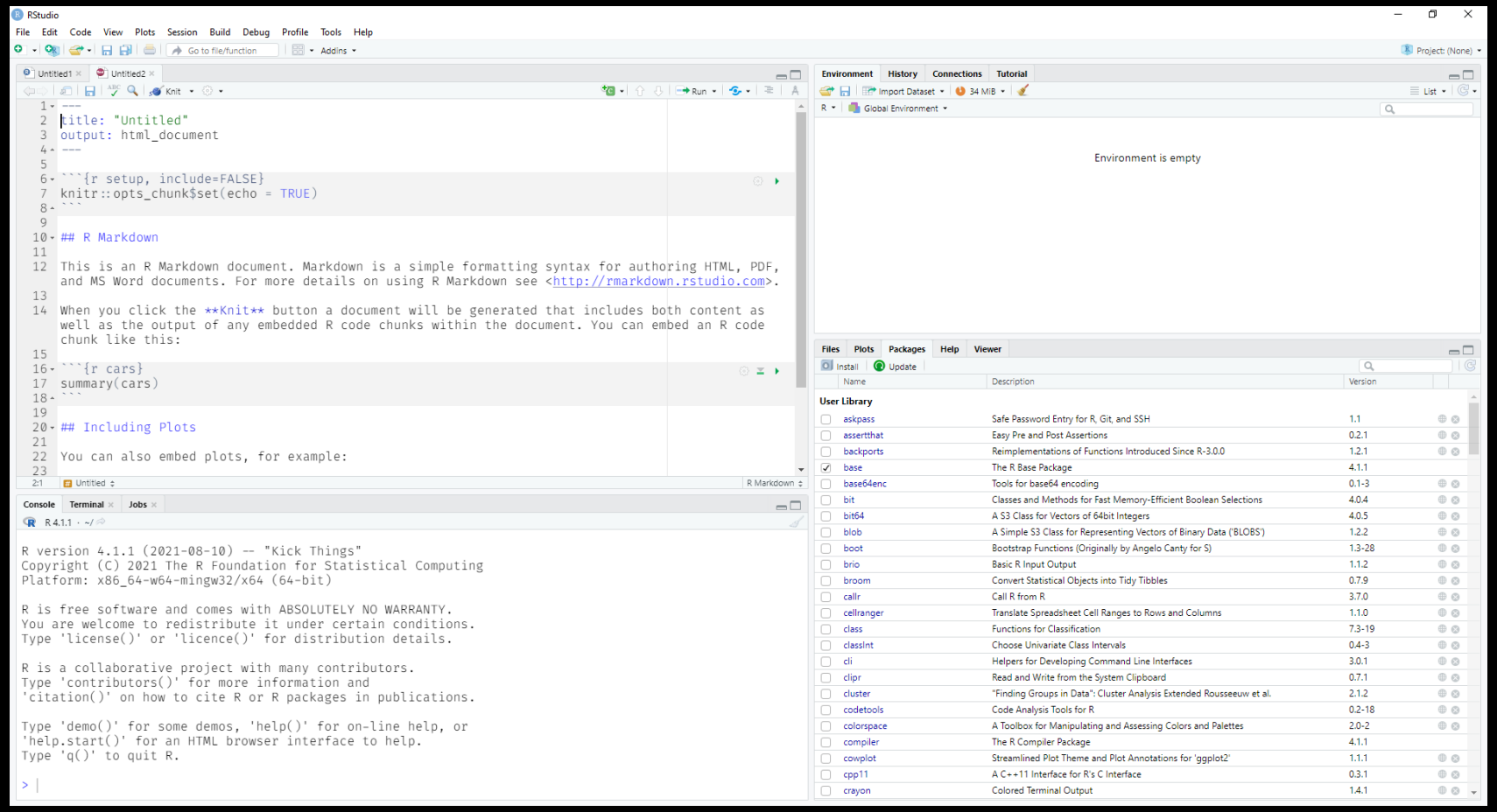
https://rstats.wtf/

Reproducibility: Use an IDE

Environment

Build and Git integration

Source Code



Files

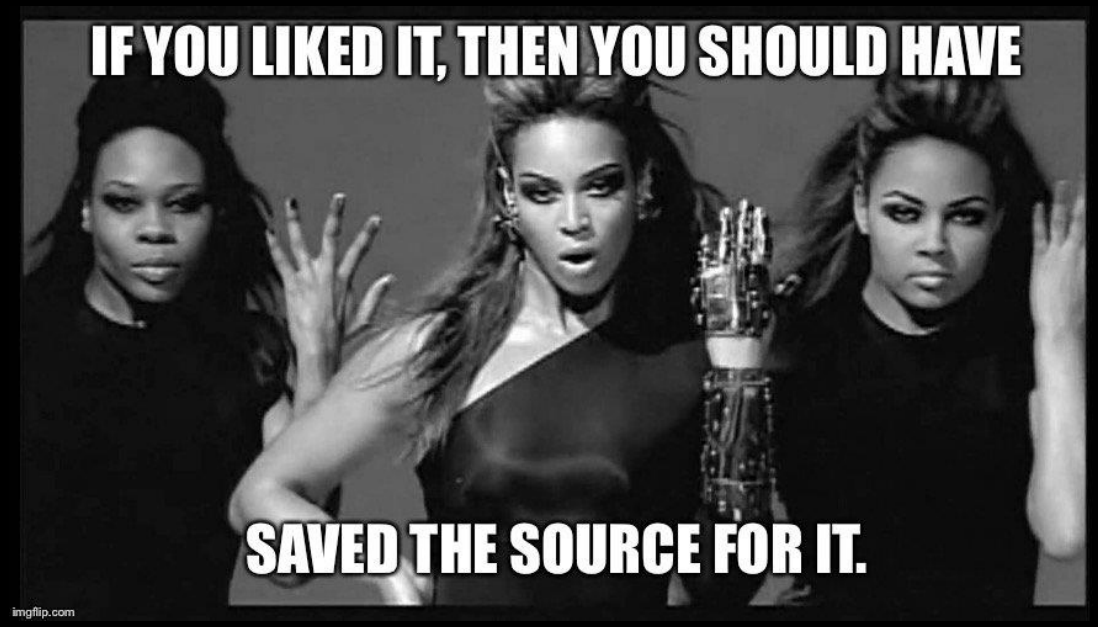
Plots

Viewer

Console

Reproducibility:
Save the Source, not the workspace

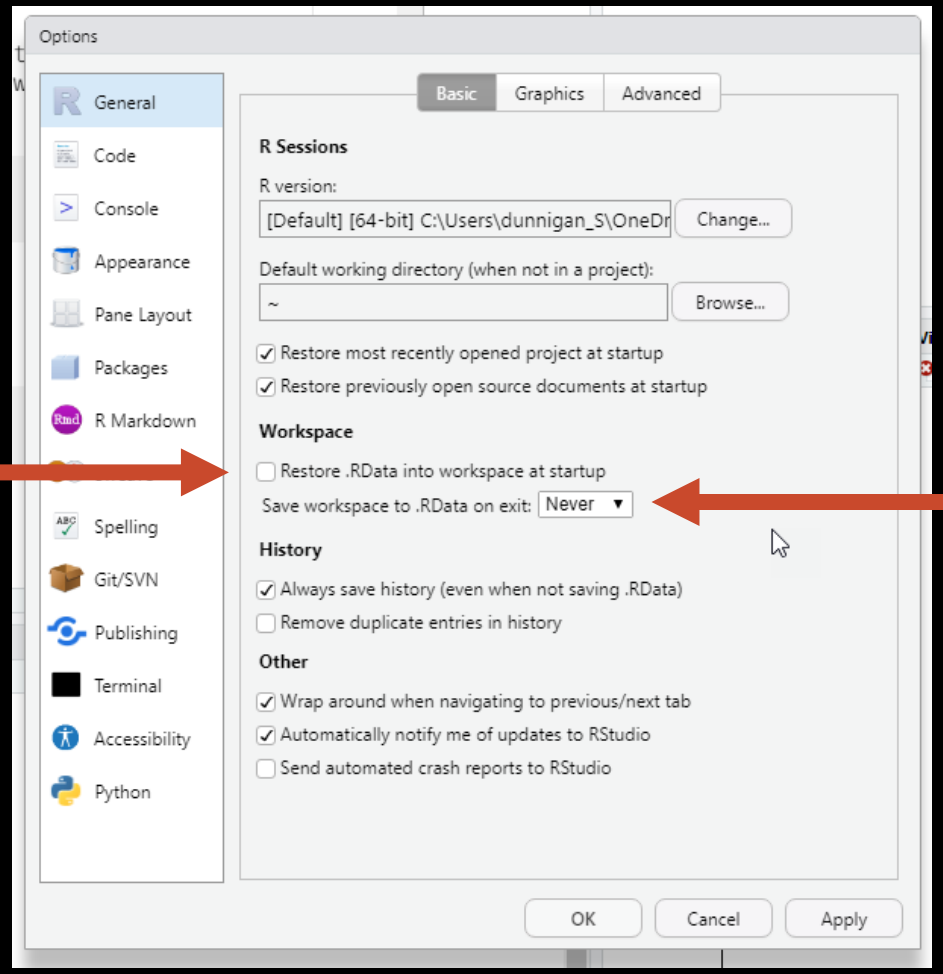
Hugh Jackman in “Swordfish” (2001) – great scene



From WTF(too good not to reshare)

Reproducibility:
Save the Source, not the workspace

- In RStudio go to Tools > Global Options
- Or `usethis::use_blank_slate()`



Reproducibility: Saving Objects

- Some analyses take a long time to execute
- Break analysis into natural phases
- Isolate computationally demanding steps: Script -> outputs
- Write objects to file

```
saveRDS(object, here("output", "my_object.rds"))
# load these objects in subsequent scripts
my_object <- readRDS(here("output", "my_object.rds"))

# Or use save() and load() with .RData
save (object, here("output", "my_object.RData"))
load(here("output", "my_object.RData"))
```

Reproducibility: Project-oriented Workflow

Name	Status	Date modified	Type	Size
.git		9/9/2021 11:00 AM	File folder	
.Rproj.user		9/9/2021 11:00 AM	File folder	
backgroundinfo		9/9/2021 11:00 AM	File folder	
data		9/9/2021 11:00 AM	File folder	
images		9/9/2021 11:00 AM	File folder	
output		9/9/2021 11:00 AM	File folder	
R		9/9/2021 11:00 AM	File folder	
reportdocs		9/9/2021 11:00 AM	File folder	
.gitignore		2/10/2021 12:03 PM	Text Document	1 KB
.Rhistory		7/8/2021 4:28 PM	RHISTORY File	18 KB
_config.yml		6/18/2019 8:06 AM	YML File	1 KB
guana		7/8/2021 3:55 PM	R Project	1 KB
my-styles		8/1/2019 11:59 AM	Microsoft Word D...	18 KB
README		2/10/2021 12:02 PM	Chrome HTML Do...	618 KB
README.md		2/17/2021 6:06 PM	MD File	1 KB

Reproducibility: Project-oriented Workflow

Name	Status	Date modified	Type	Size
.git		9/9/2021 11:00 AM	File folder	
.Rproj.user		9/9/2021 11:00 AM	File folder	
backgroundinfo		9/9/2021 11:00 AM	File folder	
data		9/9/2021 11:00 AM	File folder	
images		9/9/2021 11:00 AM	File folder	
output		9/9/2021 11:00 AM	File folder	
R		9/9/2021 11:00 AM	File folder	
reportdocs		9/9/2021 11:00 AM	File folder	
.gitignore		2/10/2021 12:03 PM	Text Document	1 KB
.Rhistory		7/8/2021 4:28 PM	RHISTORY File	18 KB
_config.yml		6/18/2019 8:06 AM	YML File	1 KB
guana		7/8/2021 3:55 PM	R Project	
my-styles		8/1/2019 11:59 AM	Microsoft Word D...	18 KB
README		2/10/2021 12:02 PM	Chrome HTML Do...	618 KB
README.md		2/17/2021 6:06 PM	MD File	1 KB

Reproducibility: Use Safe Paths

"If the first line of your #rstats script is `setwd("C:\Users\jenny\path\that\only\I\have")`, *I will come into your lab and SET YOUR COMPUTER ON FIRE."*

Rage tweets by @jennybc and @tpoi

Reproducibility: Project-oriented Workflow

Name	Status	Date modified	Type	Size
.git		9/9/2021 11:00 AM	File folder	
.Rproj.user		9/9/2021 11:00 AM	File folder	
backgroundinfo		9/9/2021 11:00 AM	File folder	
data		9/9/2021 11:00 AM	File folder	
images		9/9/2021 11:00 AM	File folder	
output		9/9/2021 11:00 AM	File folder	
R		9/9/2021 11:00 AM	File folder	
reportdocs		9/9/2021 11:00 AM	File folder	
.gitignore		2/10/2021 12:03 PM	Text Document	1 KB
.Rhistory		7/8/2021 4:28 PM	RHISTORY File	18 KB
_config.yml		6/18/2019 8:06 AM	YML File	1 KB
guana		7/8/2021 3:55 PM	R Project	
my-styles		8/1/2019 11:59 AM	Microsoft Word D...	18 KB
README		2/10/2021 12:02 PM	Chrome HTML Do...	618 KB
README.md		2/17/2021 6:06 PM	MD File	1 KB

Artwork by
Allison Horst
(https://github.com/allisonhorst)

So, what does that look like?

```
# load here package
library(here)
here() starts at C:/Users/Dunnigan_S/2021-infrequent-useR

# run here
here::here('data', 'fun.xlsx')
[1] "C:/Users/Dunnigan_S/2021-infrequent-useR/data/fun.xlsx"

# read in 'fun' data
dat <- readxl::read_xlsx(here::here('data', 'fun.xlsx'))
```

Reproducibility:
Use standardized naming conventions

- **TL;DR** - machine and human readable and plays well with default ordering (put something numeric first).

Really great presentation by Jenny Bryan about naming files that she gave at the Reproducible Science Workshop: https://speakerdeck.com/jennybc/how-to-name-files

"plays well with default ordering"

```
2013-06-26_BRAFWTNEGASSAY_Plasmid-Cellline-100-1MutantFraction_H01.csv
2013-06-26_BRAFWTNEGASSAY_Plasmid-Cellline-100-1MutantFraction_H02.csv
2013-06-26_BRAFWTNEGASSAY_Plasmid-Cellline-100-1MutantFraction_H03.csv
2013-06-26_BRAFWTNEGASSAY_Plasmid-Cellline-100-1MutantFraction_platefile.csv
2014-02-26_BRAFWTNEGASSAY_FFPEDNA-CRC-1-41_A01.csv
2014-02-26_BRAFWTNEGASSAY_FFPEDNA-CRC-1-41_A02.csv
2014-02-26_BRAFWTNEGASSAY_FFPEDNA-CRC-1-41_A03.csv
2014-02-26_BRAFWTNEGASSAY_FFPEDNA-CRC-1-41_A04.csv
```

```
01_marshal-data.r
02_pre-dea-filtering.r
03_dea-with-limma-voom.r
04_explore-dea-results.r
90_limma-model-term-name-fiasco.r
helper01_load-counts.r
helper02_load-exp-des.r
helper03_load-focus-statinf.r
helper04_extract-and-tidy.r
```

put something numeric first

From Jenny Bryan's "How to name things" presentation.

Use the ISO 8601 standard for dates (YYYY-MM-DD)

Deliberate use of "_" and "-" allows us to recover meta-data from the filenames.

From Jenny Bryan's "How to name things" presentation.

2013-06-26_BRAFWTNEGASSAY_Plasmid-Cellline-100-1MutantFraction_H01.csv
2013-06-26_BRAFWTNEGASSAY_Plasmid-Cellline-100-1MutantFraction_H02.csv
2013-06-26_BRAFWTNEGASSAY_Plasmid-Cellline-100-1MutantFraction_H03.csv
2013-06-26_BRAFWTNEGASSAY_Plasmid-Cellline-100-1MutantFraction_platefile.csv

```
> flist <- list.files(pattern = "Plasmid") %>% head

> stringr::str_split_fixed(flist, "[_\\.]", 5)
     [,1]         [,2]             [,3]                                  [,4]  [,5]
[1,] "2013-06-26" "BRAFWTNEGASSAY" "Plasmid-Cellline-100-1MutantFraction" "A01" "csv"
[2,] "2013-06-26" "BRAFWTNEGASSAY" "Plasmid-Cellline-100-1MutantFraction" "A02" "csv"
[3,] "2013-06-26" "BRAFWTNEGASSAY" "Plasmid-Cellline-100-1MutantFraction" "A03" "csv"
[4,] "2013-06-26" "BRAFWTNEGASSAY" "Plasmid-Cellline-100-1MutantFraction" "B01" "csv"
[5,] "2013-06-26" "BRAFWTNEGASSAY" "Plasmid-Cellline-100-1MutantFraction" "B02" "csv"
[6,] "2013-06-26" "BRAFWTNEGASSAY" "Plasmid-Cellline-100-1MutantFraction" "B03" "csv"
```

date assay sample set well

This happens to be R but also possible in the shell, Python, etc.

Examples

NO

fig 2.png

Jim’s master of all data.xlsx

Plankton 2021 raw*edited.csv

abstract.docx

YES

fig02_linegraph-chla-timeseries.png

jims-filename-is-better-master.xlsx

2021-plankton-edited.csv

2021-09-23_abstract-for-wksp.docx

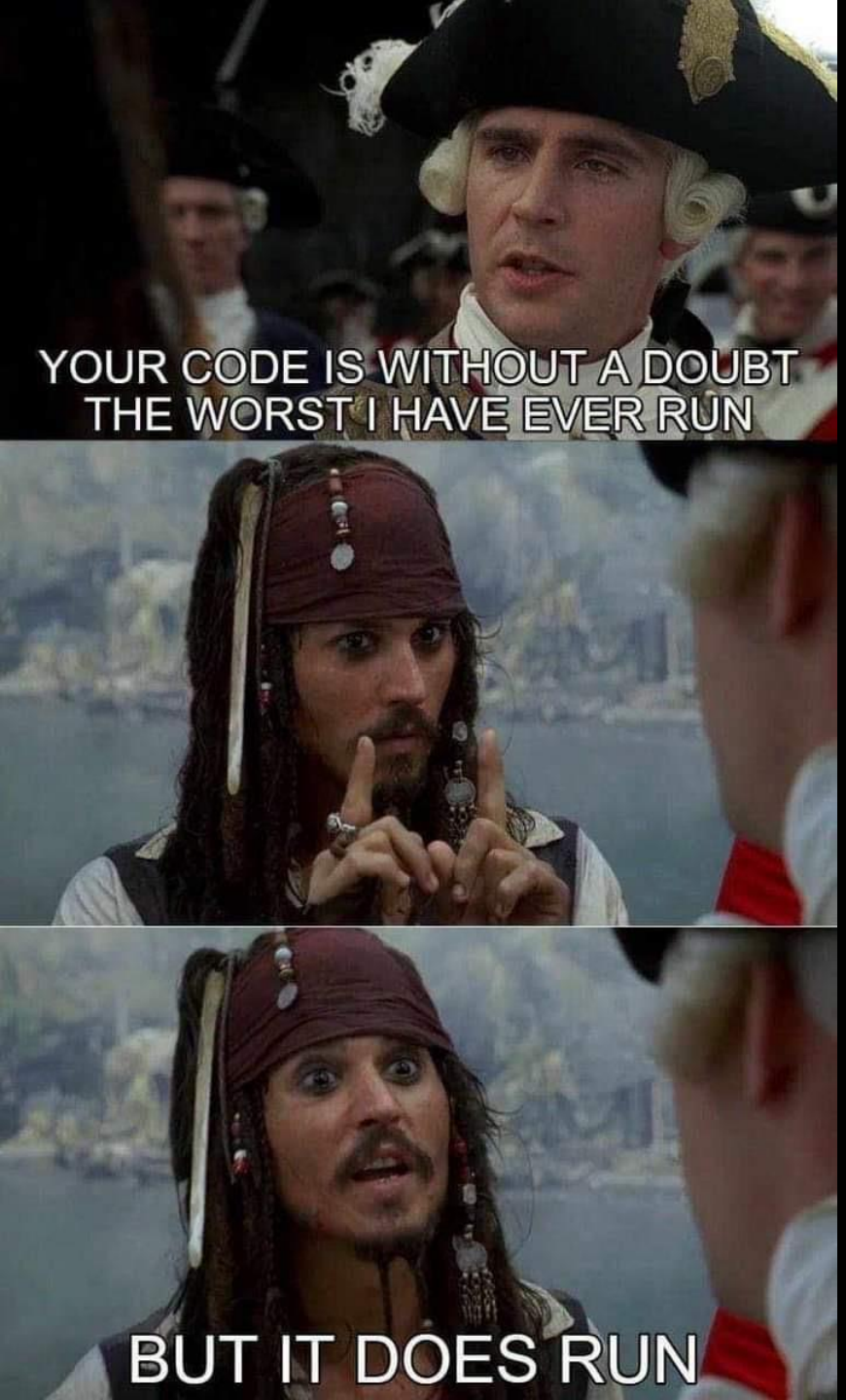
The `janitor` package

Artwork by Allison Horst (https://github.com/allisonhorst)

The `janitor` package

```
> head(env)
# A tibble: 6 x 16
  Site        StationCode Date                `Time (24 hr)` ActivityType ComponentShort ComponentLong
  <chr>       <chr>       <dttm>              <chr>          <chr>        <chr>          <chr>
1 PINE ISLAND gtmpinut1.1 2021-01-12 00:00:00 13:06          Field        Depth          Water depth
2 PINE ISLAND gtmpinut1.1 2021-01-12 00:00:00 13:06          Field        Depth_S        Sample depth
3 PINE ISLAND gtmpinut1.1 2021-01-12 00:00:00 13:06          Field        SECCHI         Secchi Disk
4 PINE ISLAND gtmpinut1.1 2021-01-12 00:00:00 13:06          Field        WIND_S         Wind Speed
```

```
> clean_names(env) %>% head()
# A tibble: 6 x 16
  site        station_code date                time_24_hr activity_type component_short component_long
  <chr>       <chr>        <dttm>              <chr>      <chr>         <chr>           <chr>
1 PINE ISLAND gtmpinut1.1  2021-01-12 00:00:00 13:06      Field         Depth           Water depth
2 PINE ISLAND gtmpinut1.1  2021-01-12 00:00:00 13:06      Field         Depth_S         Sample depth
3 PINE ISLAND gtmpinut1.1  2021-01-12 00:00:00 13:06      Field         SECCHI          Secchi Disk
4 PINE ISLAND gtmpinut1.1  2021-01-12 00:00:00 13:06      Field         WIND_S          Wind Speed
```



Reproducibility: Coding style

"Good coding style is like using correct punctuation. You can manage without it, but it sure makes things easier to read."

– Hadley Wickham, *Advanced R*

- Style Guides exist (like Hadley's or Google)
- You may write it, but others will read it.
- Be consistent.
- Check out the `formatR` package by Yihui Xie

Image is shared from an R users Facebook group, and I wish I could give the original creator credit.

Reproducibility: Coding Style

- Lowercase variable and object names with underscores (_) to separate words within a name.
- Spaces! Code is already difficult to read. giveyoureyesabreak
- Closing curly braces `}` go on their own line
- Use <- and not =, for assignment
- Pipes! `%>%` from the `magrittr` package

Additional Best Practice Tips

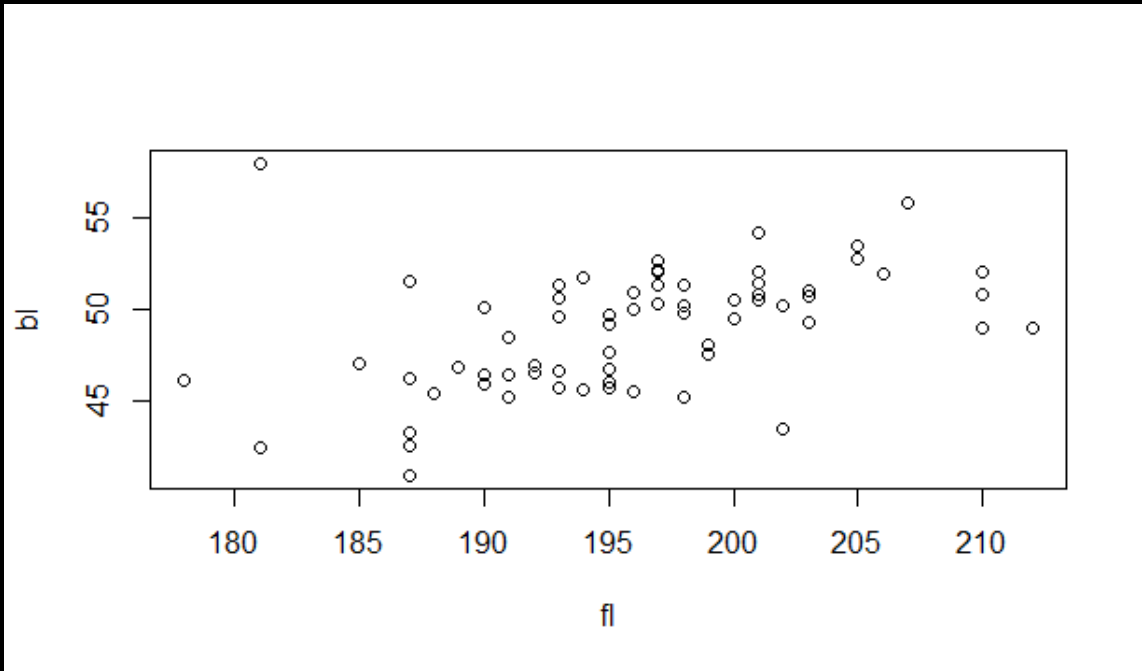
Tidy data, data frames, and moving away from the spreadsheet

Leave it in the data frame

Adapted from Jenny Bryan's RStudio webinar "*Thinking inside the box: you can do that inside a dataframe?!*"

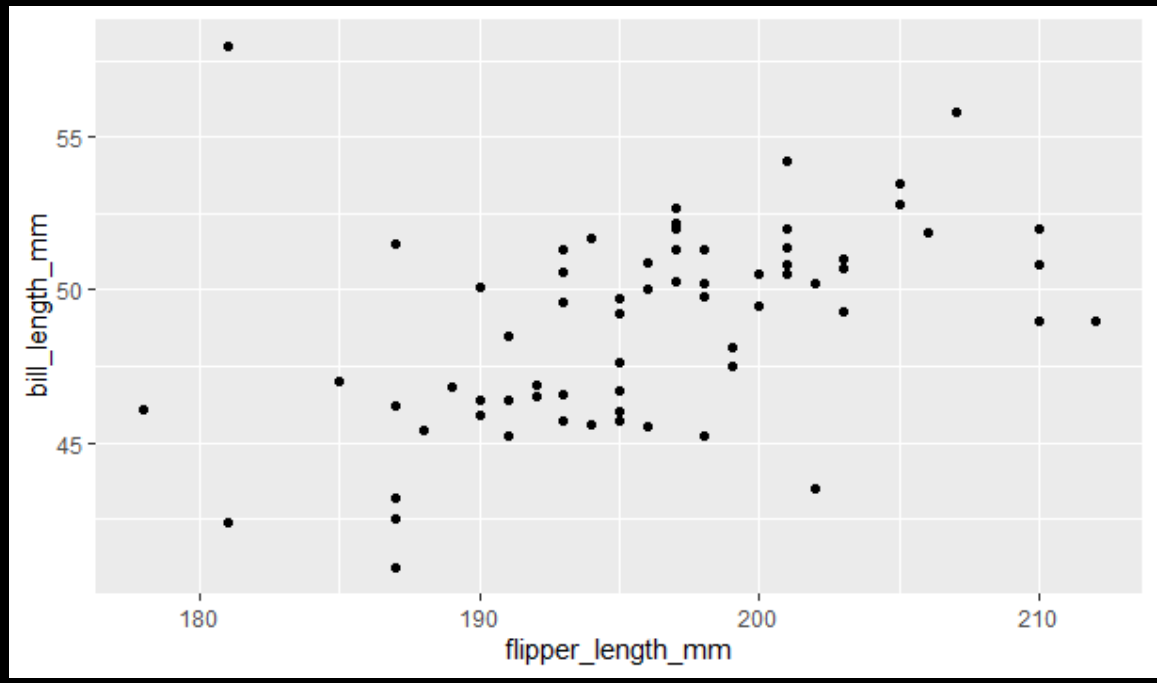
- Don't create little excerpts and copies of your data

```
library(palmerspenguins) # for the dataset
bl <- penguins[277:344, 3]
fl <- penguins[277:344, 5]
plot(bl ~ fl)
```



Bring intent into your code

```
library(tidyverse)
penguins %>%
  dplyr::filter(species == “Chinstrap”) %>%
  ggplot(aes(x = flipper_length_mm,
             y = bill_length_mm)) +
  geom_point()
```



Tidy data

A data set is **tidy** iff:

1. Each **variable** is in its own **column**
2. Each **observation** is in its own **row**
3. Each **value** is in its own **cell**

Highly recommend these papers:

Wickham, 2014: *Tidy Data*

Broman and Woo, 2017: *Data Organization in Spreadsheets*

Tidy data

A data set is **tidy** iff:

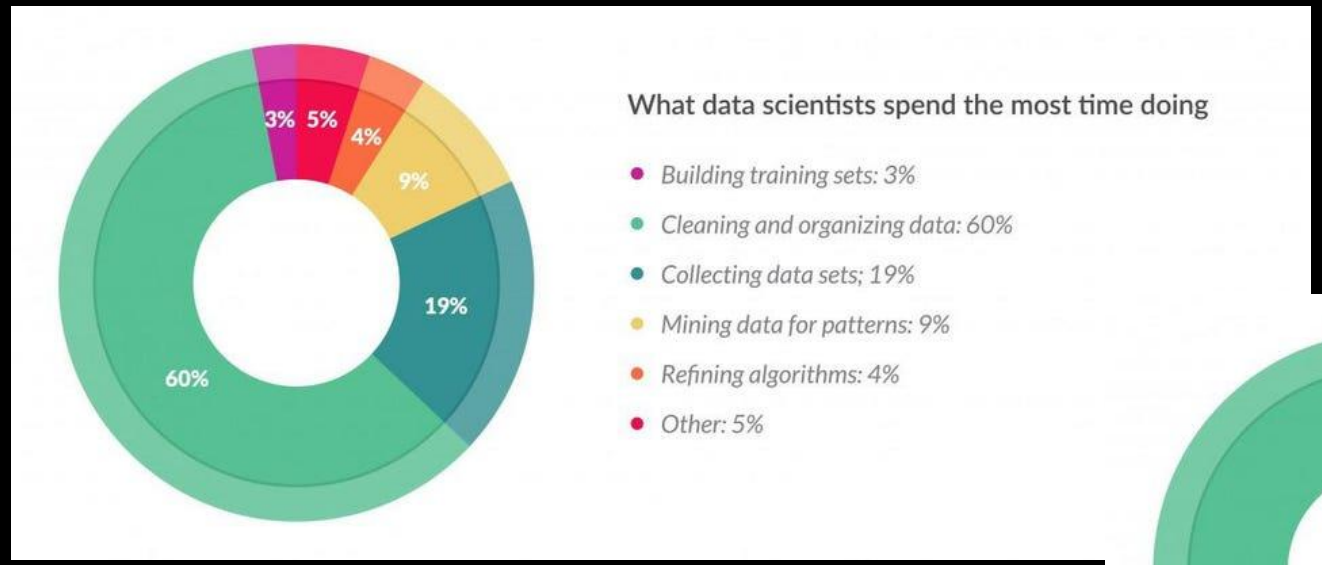
1. Each **variable** is in its own **column**
2. Each **observation** is in its own **row**
3. Each **value** is in its own **cell**

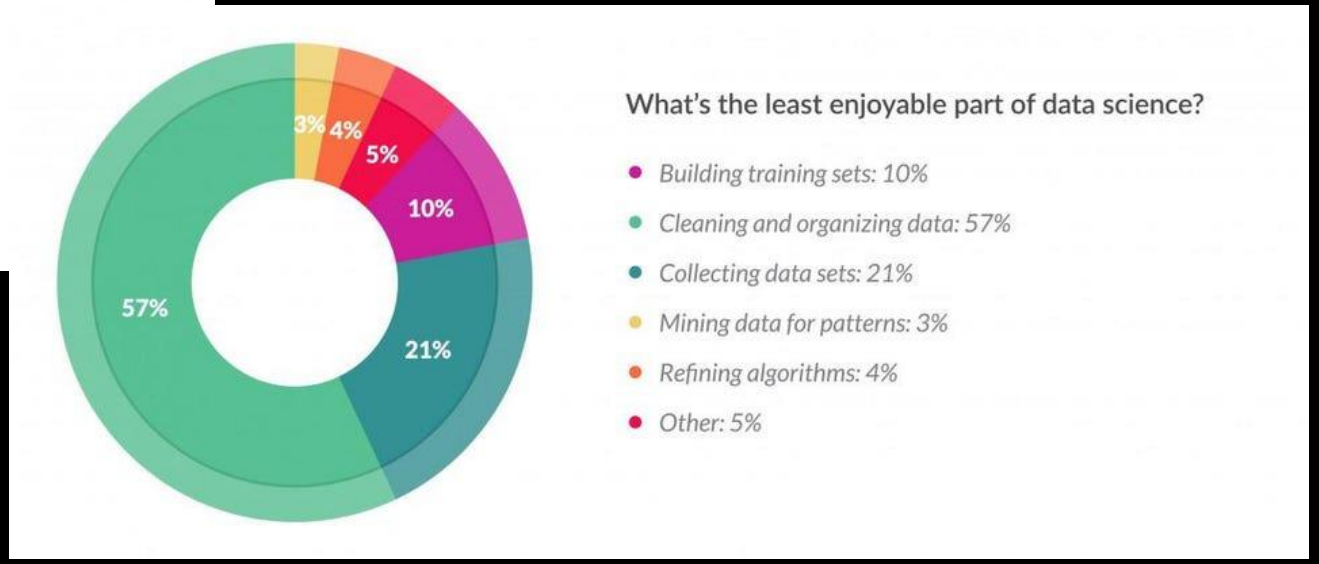
variable: all values that measure the same underlying *attribute*

observation: all values measured on the same *unit*

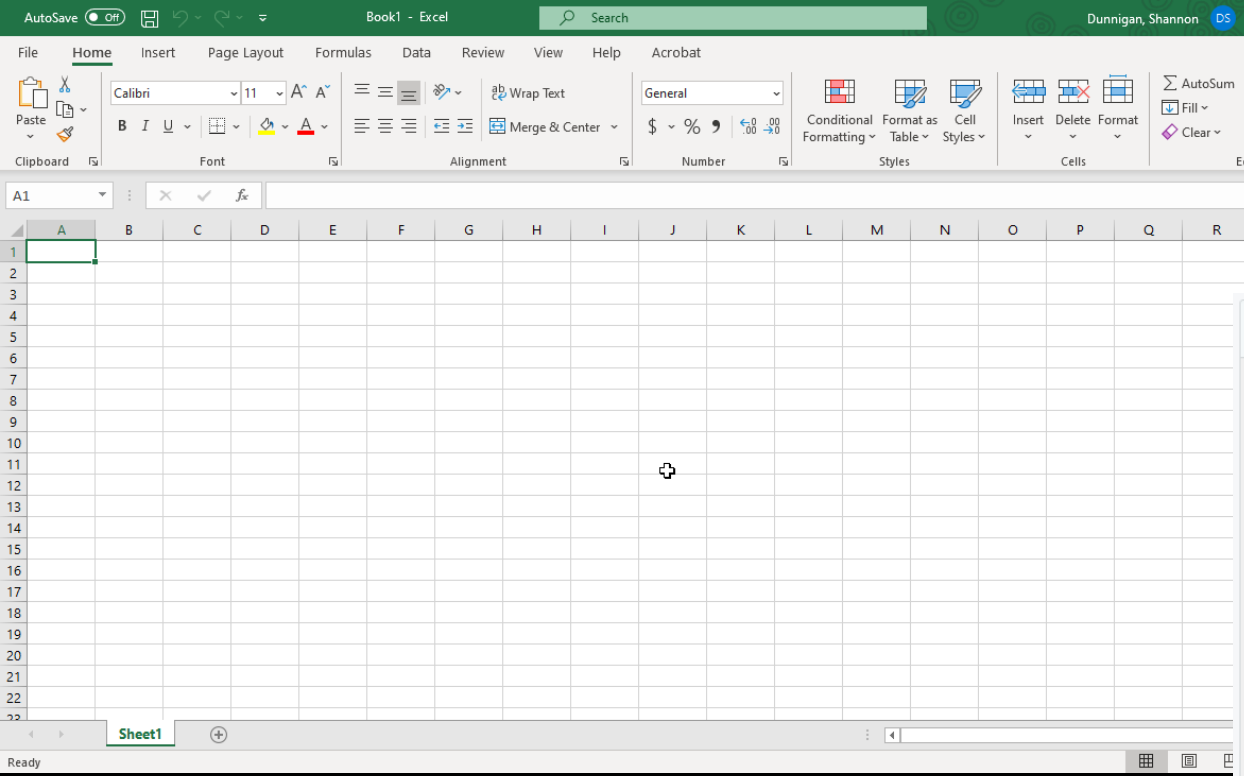
value: belongs to one **variable** and one **observation**

“…most time-consuming, least enjoyable data science task…”





Overcoming the spreadsheet hurdle



01.1_load_wrangle_LIMS.R

Source on Save | Run | Source

```r

# 02 wrangle-tidy data ------------------------------------------------------
# rename some columns in lims to what we use in SWMP
# convert datetimes into POSIXct format
# make all entries in station_code and component_long columns lowercase (easier coding)
# remove field blanks
lims2 <- lims %>%
         dplyr::rename(station_code = field_id,
                       component_long = component,
                       datetimestamp = date_sampled) %>%
         dplyr::mutate(datetimestamp = as.POSIXct(strptime(datetimestamp,
                                                           "%d-%b-%Y %H:%M", tz='EST')),
                       date_analyzed = as.POSIXct(strptime(date_analyzed,
                                                           "%d-%b-%Y %H:%M", tz='EST')),
                       station_code = tolower(station_code),
                       component_long = tolower(component_long)) %>%
         dplyr::filter(station_code != "field blank")

# correct so that TKN and TKN-F are different
# fixing the LIMS entry so that kjeldahl nitrogen, dissolved is different from kjeldahl nitrogen
tkn_f <- lims2 %>%
         dplyr::filter(analysis == "W-TKN-F") %>%
         dplyr::mutate(component_long = "kjeldahl nitrogen, dissolved")

# merge the renamed TKNF data with all the other data and then join with the `names` df to get t
lims3 <- lims2 %>%
         dplyr::filter(analysis != "W-TKN-F") %>%
         dplyr::bind_rows(tkn_f) %>%
         dplyr::left_join(names, by = "component_long") %>%
         dplyr::mutate(cdmo_name = forcats::as_factor(cdmo_name)) # I think conv. to factor hel

## clean up environment ---
rm(tkn_f, lims2, names)

# 03 make LIMS data wide ----------------------------------------------------
# to make the LIMS data in wide format for entry into in-house datafile

lims_wide_results <- lims3 %>%
                     dplyr::select(station_code, datetimestamp, cdmo_name, component_long, re
```

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)

Group Cases

Use **group_by**(.data, ..., .add = FALSE, .drop = TRUE) to create a "grouped" copy of a table grouped by columns in ... dplyr functions will manipulate each "group" separately and combine the results.

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

Use **rowwise**(.data, ...) to group data into individual rows. dplyr functions will compute results for each row. Also apply functions to list-columns. See tidyr cheat sheet for list-column workflow.

```
starwars %>%
  rowwise() %>%
  mutate(film_count = length(films))
```

ungroup(x, ...) Returns ungrouped copy of table.
ungroup(g_mtcars)

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 with the lowest and highest values.
slice_min(mtcars, mpg, prop = 0.25)

slice_head(.data, ..., n, prop) and **slice_tail()** Select the first or last rows.
slice_head(mtcars, n = 5)

Logical and boolean operators to use with filter()

==	<	<=	is.na()	%in%	|
!=	>	>=	!is.na()	!	&

xor()

See **?base::Logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, ..., .by_group = FALSE) Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES

add_row(.data, ..., .before = NULL, .after = NULL) Add one or more rows to a table.
add_row(cars, speed = 1, dist = 1)

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)

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

MANIPULATE MULTIPLE VARIABLES AT ONCE

across(.cols, .funs, ..., .names = NULL) Summarise or mutate multiple columns in the same way.
summarise(mtcars, across(everything(), mean))

c_across(.cols) Compute across columns in row-wise data.
transmute(rowwise(UKgas), total = sum(c_across(1:2)))

MAKE NEW VARIABLES

Apply **vectorized functions** to columns. Vectorized functions take vectors as input and return vectors of the same length as output (see back).

vectorized function

mutate(.data, ..., .keep = "all", .before = NULL, .after = NULL) Compute new column(s). Also **add_column()**, **add_count()**, and **add_tally()**.
mutate(mtcars, gpm = 1 / mpg)

transmute(.data, ...) Compute new column(s), drop others.
transmute(mtcars, gpm = 1 / mpg)

rename(.data, ...) Rename columns. Use **rename_with()** to rename with a function.
rename(cars, distance = dist)

R Studio

Vectorized Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

OFFSET

dplyr::**lag()** - offset elements by 1
dplyr::**lead()** - offset elements by -1

CUMULATIVE AGGREGATE

dplyr::**cumall()** - cumulative all()
dplyr::**cumany()** - cumulative any()
cummax() - cumulative max()
dplyr::**cummean()** - cumulative mean()
cummin() - cumulative min()
cumprod() - cumulative prod()
cumsum() - cumulative sum()

RANKING

dplyr::**cume_dist()** - proportion of all values <=
dplyr::**dense_rank()** - rank w ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

+, -, *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons
dplyr::**between()** - x >= left & x <= right
dplyr::**near()** - safe == for floating point numbers

MISCELLANEOUS

dplyr::**case_when()** - multi-case if_else()
starwars %>%
  mutate(type = case_when(
    height > 200 | mass > 200 ~ "large",
    species == "Droid" ~ "robot",
    TRUE ~ "other"
)
dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function

COUNT

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

POSITION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICAL

mean() - proportion of TRUE's
sum() - # of TRUE's

ORDER

dplyr::**first()** - first value
dplyr::**last()** - last value
dplyr::**nth()** - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

tibble::**rownames_to_column()**
Move row names into col.
a <- rownames_to_column(mtcars, var = "C")

tibble::**column_to_rownames()**
Move col into row names.
column_to_rownames(a, var = "C")

Also tibble::**has_rownames()** and tibble::**remove_rownames()**.

Combine Tables

COMBINE VARIABLES

bind_cols(..., .name_repair) Returns tables placed side by side as a single table. Column lengths must be equal. Columns will NOT be matched by id (to do that look at Relational Data below), so be sure to check that both tables are ordered the way you want before binding.

RELATIONAL DATA

Use a **"Mutating Join"** to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

left_join(x, y, by = NULL, copy = FALSE, suffix = c("_x", "_y"), ..., keep = FALSE, na_matched = "na") Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix = c("_x", "_y"), ..., keep = FALSE, na_matches = "na") Join matching values from x to y.

inner_join(x, y, by = NULL, copy = FALSE, suffix = c("_x", "_y"), ..., keep = FALSE, na_matches = "na") Join data. Retain only rows with matches.

full_join(x, y, by = NULL, copy = FALSE, suffix = c("_x", "_y"), ..., keep = FALSE, na_matches = "na") Join data. Retain all values, all rows.

COLUMN MATCHING FOR JOINS

Use **by = c("col1", "col2", ...)** to specify one or more common columns to match on.
left_join(x, y, by = "A")

Use a named vector, **by = c("col1" = "col2")**, to match on columns that have different names in each table.
left_join(x, y, by = c("C" = "D"))

Use **suffix** to specify the suffix to give to unmatched columns that have the same name in both tables.
left_join(x, y, by = c("C" = "D"), suffix = c("_1", "_2"))

COMBINE CASES

bind_rows(..., .id = NULL) Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured).

Use a **"Filtering Join"** to filter one table against the rows of another.

semi_join(x, y, by = NULL, copy = FALSE, ..., na_matches = "na") Return rows of x that have a match in y. Use to see what will be included in a join.

anti_join(x, y, by = NULL, copy = FALSE, ..., na_matches = "na") Return rows of x that do not have a match in y. Use to see what will not be included in a join.

Use a **"Nest Join"** to inner join one table to another into a nested data frame.

nest_join(x, y, by = NULL, copy = FALSE, keep = FALSE, name = NULL, ...) Join data, nesting matches from y in a single new data frame column.

SET OPERATIONS

intersect(x, y, ...)
Rows that appear in both x and y.

setdiff(x, y, ...)
Rows that appear in x but not y.

union(x, y, ...)
Rows that appear in x or y. (Duplicates removed). **union_all()** retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

R Studio

Your turn: What kinds of things have you found helpful?

A Reproducible Report

https://github.com/skdunnigan/2021-infrequent-useR

What is R Markdown?

- File format for making document using R
- Written in markdown (easy, plain text format)
- Chunks of embedded code
- Designed to be used with the `rmarkdown` package
- “render” combines knit and convert to produce the file