Data Wrangling

Charles Nicholson, Ph.D

Data wrangl

process of c
structuring a
raw data int
format to m
appropriate
for a variety
downstream
such as anal

wrangle verb

wran·gle | \ ˈraŋ-gəl \

wrangled; wrangling \ ˈraŋ-g(ə-)liŋ \

Definition of *wrangle* (Entry 1 of 2)

intransitive verb

1 : to dispute angrily or peevishly : BICKER

2 : to engage in argument or controversy

transitive verb

1 : to obtain by persistent arguing or maneuvering : WANGLE

2 [back-formation from *wrangler*] : to herd and care for (livestock and especially horses) on the range

wrangle noun

Definition of *wrangle* (Entry 2 of 2)

1 : an angry, noisy, or prolonged dispute or quarrel

2 : the action or process of wrangling

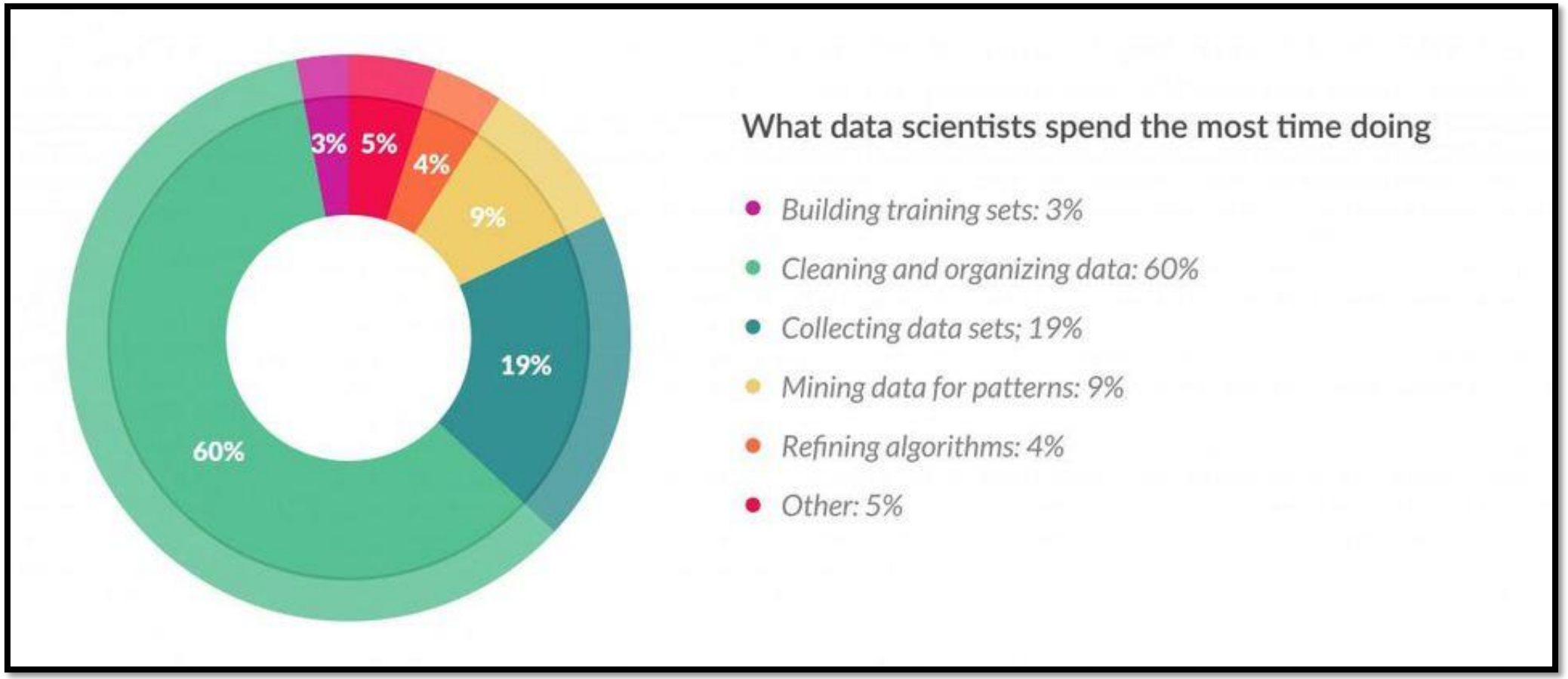
Data wrangling

- Includes all of the steps we discuss in Data Preparation
  - Improving data quality
  - Dealing with outliers
  - Missing value imputation
  - Feature engineering
- And more...

More…

- read in data from various formats and sources
- string manipulation; date manipulation
- filter rows, sub select columns
- sorting
- summarize by groups:
  - mean, standard deviation, counts, range, etc.
  - first or last observation in an ordered series
  - top n observations
- de-duplicate records
- sample observations
- join data sets:
  - left join, right join, inner join, full join
- category levels
  - binning numerical data into ordinal categories
  - re-ordering existing levels; collapsing categories
- transpose data
  - wide-form to long-form
  - long-form to wide-form

Data wrangling is a big part of the analytics process…



Forbes, 2016, “Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says”

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**

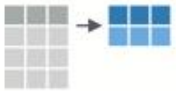
pipes

x %>% f(y) becomes **f(x, y)**

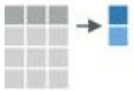
Summarise Cases

These 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(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

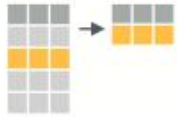
Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and

Manipulate Cases

EXTRACT CASES

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



filter(.data, …) Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*

distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values.
distinct(iris, Species)

sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select fraction of rows.
sample_frac(iris, 0.5, replace = TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

slice(.data, …) Select rows by position.
slice(iris, 10:15)

top_n(x, n, wt) Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

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

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

Manipulate Variables

EXTRACT VARIABLES

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

pull(.data, var = -1) Extract column values as a vector. Choose by name or index.
pull(iris, Sepal.Length)

select(.data, …)
Extract columns as a table. Also **select_if()**.
select(iris, Sepal.Length, Species)

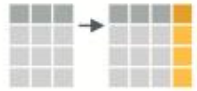
Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match)	**num_range**(prefix, range)	**:**, e.g. mpg:cyl
ends_with(match)	**one_of**(…)	**-**, e.g, -Species
matches(match)	**starts_with**(match)	

MAKE NEW VARIABLES

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

vectorized function



mutate(.data, …)
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)

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

mutate_all(tbl, funs **)** Apply funs to every

Data Wrangling
with dplyr and tidyr
Cheat Sheet

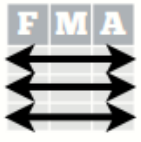
RStudio

Tidy Data - A foundation for wrangling in R

In a tidy data set:

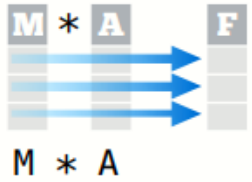
Each **variable** is saved in its own **column**

&



Each **observation** is saved in its own **row**

Tidy data complements R's **vectorized operations**. R will automatically preserve observations as you manipulate variables. No other format works as intuitively with R.



Syntax - Helpful conventions for wrangling

dplyr::**tbl_df(iris)**

Converts data to tbl class. tbl's are easier to examine than data frames. R displays only the data that fits onscreen:

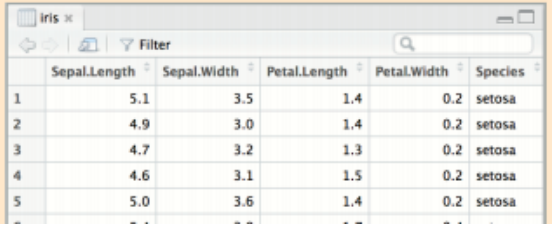
```
Source: local data frame [150 x 5]

   Sepal.Length Sepal.Width Petal.Length
1           5.1         3.5          1.4
2           4.9         3.0          1.4
3           4.7         3.2          1.3
4           4.6         3.1          1.5
5           5.0         3.6          1.4
..          ...         ...          ...
Variables not shown: Petal.Width (dbl),
  Species (fctr)
```

dplyr::**glimpse(iris)**

Information dense summary of tbl data.

utils::**View(iris)**

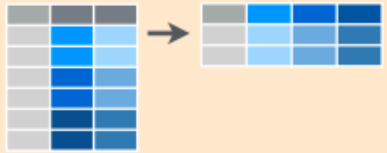
View data set in spreadsheet-like display (note capital V).



Reshaping Data - Change the layout of a data set

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

Gather columns into rows.



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

Spread rows into columns.

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

Separate one column into several.

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

Unite several columns into one.

dplyr::**data_frame(a = 1:3, b = 4:6)**

Combine vectors into data frame (optimized).

dplyr::**arrange(mtcars, mpg)**

Order rows by values of a column (low to high).

dplyr::**arrange(mtcars, desc(mpg))**

Order rows by values of a column (high to low).

dplyr::**rename(tb, y = year)**

Rename the columns of a data frame.

Subset Observations (Rows)

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)**

Subset Variables (Columns)

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

Select columns by name or helper function.

Helper functions for select - ?select

select(iris, **contains(".")**)

Select columns whose name contains a character string.

Data Import : : CHEAT SHEET

R's **tidyverse** is built around **tidy data** stored in **tibbles**, which are enhanced data frames.

The front side of this sheet shows how to read text files into R with **readr**.

The reverse side shows how to create tibbles with **tibble** and layout tidy data with **tidyr**.

Read Tabular Data - These functions share the common arguments:

Data types

readr functions guess

OTHER TYPES OF DATA

Try one of the following packages to import other types of files

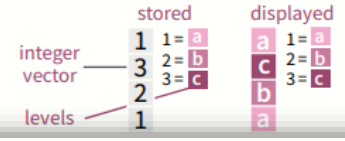
- **haven** - SPSS, Stata, and SAS files
- **readxl** - excel files (.xls and .xlsx)

Factors with forcats : : CHEAT SHEET

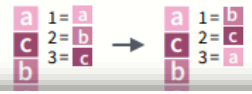
The **forcats** package provides tools for working with factors, which are R's data structure for categorical data.

Factors

R represents categorical data with factors. A **factor** is an integer vector with a **levels** attribute that stores a set of mappings between

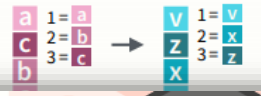


Change the order of levels



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

Change the value of levels



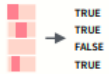
fct_recode(.f, ...) Manually change levels. Also **fct_relabel** which obeys purrr::map syntax to apply a function or expression to each level.
code(f, v = "a", x = "b", z = "c")
abel(f, ~ paste0("x", .x))

on(f, prefix = ""))
mize levels with random
rs. *fct_anon(f)*

String manipulation with stringr : : CHEAT SHEET

The **stringr** package provides a set of internally consistent tools for working with character strings, i.e. sequences of characters surrounded by quotation marks.

Detect Matches



str_detect(string, pattern) Detect the

Subset Strings

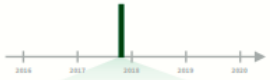
str_sub(string, start = 1L, end = -1L) Extract

Manage Lengths

str_length(string) The width of strings (i.e.

Dates and times with lubridate : : CHEAT SHEET

Date-times



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 wide variety of input formats.

GET AND SET COMPONENTS

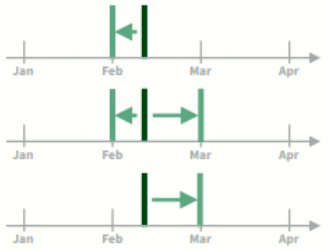
Use an accessor function to get a component.

d ## "2017-11-28"
day(d) ## 28

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

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

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")

rollback(dates, roll_to_first = FALSE, preserve_hms = TRUE)
Roll back to last day of previous

Some examples in R with dplyr and tidyr (and magrittr)