Data Cleaning & Exploration

RLadies

London, Ontario

Jaky Kueper

May 30, 2019

Outline

1. Overview of data cleaning and exploration.
2. Helpful packages and functions for 1.
3. Practice and skill-share.

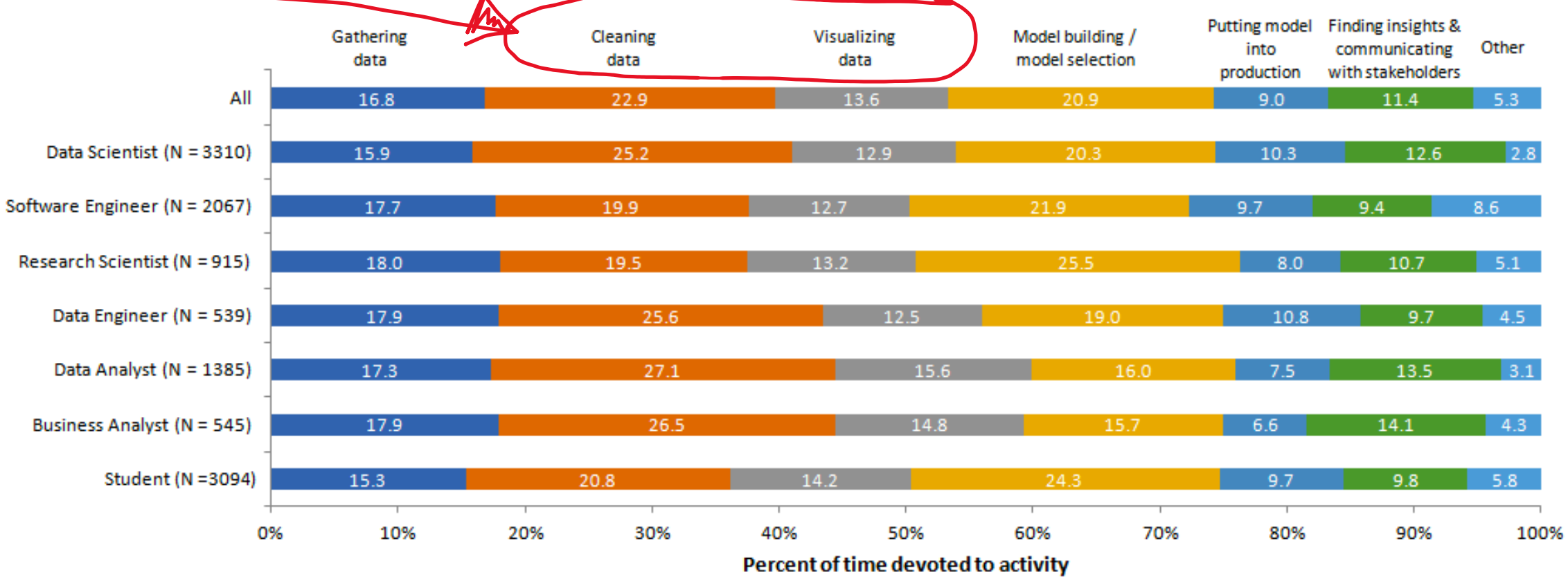
- Assumption: your data are square/rectangular and not super large, are not complete garbage (principles of data quality), and you have some sense of what's there.
- Disclaimer: Jaky studies epidemiology and computer science; she is a self-taught R-enthusiast.

Introduction to Data Cleaning

Data Cleaning and Exploration

- **Data Cleaning** is ”the process of amending or removing data in a database that is incorrect, incomplete, improperly formatted, or duplicated.” (Margaret Rouse)
  - AKA ‘tidying’ or ‘wrangling’ = ‘tidy’ + ‘transformation’ (Wickham & Grolemund)
  - AKA data cleansing
- **Data Exploration** is “the first step in data analysis and typically involves summarizing the main characteristics of a data set, including its size, accuracy, initial patterns in the data and other attributes.” (Margaret Rouse)
  - Note: we are not getting into things like exploratory risk factor analysis or descriptive epidemiology.
- What these entail depends on the data and what you want to do.
- How well you prepare data impacts quality of results!

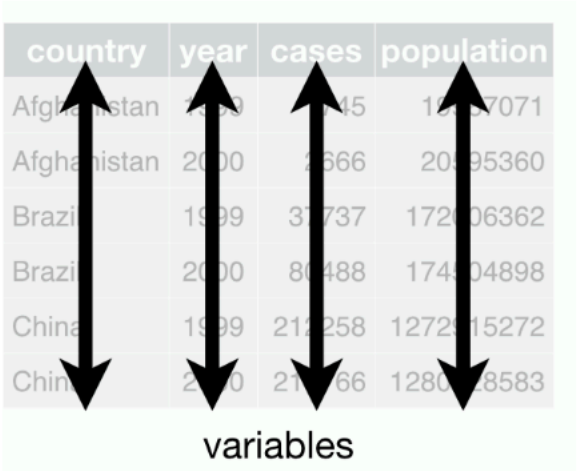
During a typical data science project at work or school, approximately what proportion of your time is devoted to the following?

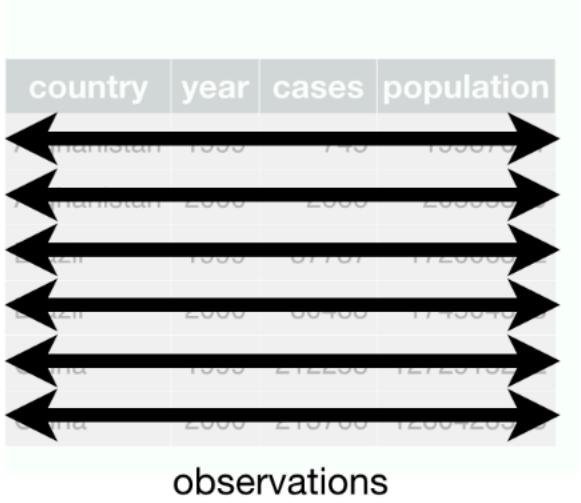


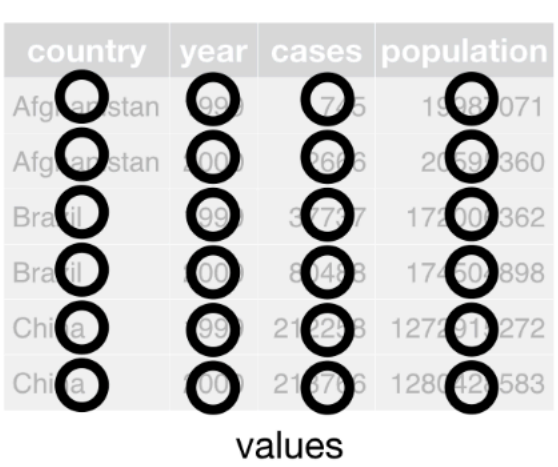
Note: Data are from the 2018 Kaggle ML and Data Science Survey. You can learn more about the study here: http://www.kaggle.com/kaggle/kaggle-survey-2018.
A total of 23859 respondents completed the survey; the percentages in the graph are based on a total of 15937 respondents who provided an answer to this question. Only selected job titles are presented.

BUSINESS BROADWAY
DATA SCIENCE | CUSTOMER ANALYTICS | MACHINE LEARNING

Tidy Data



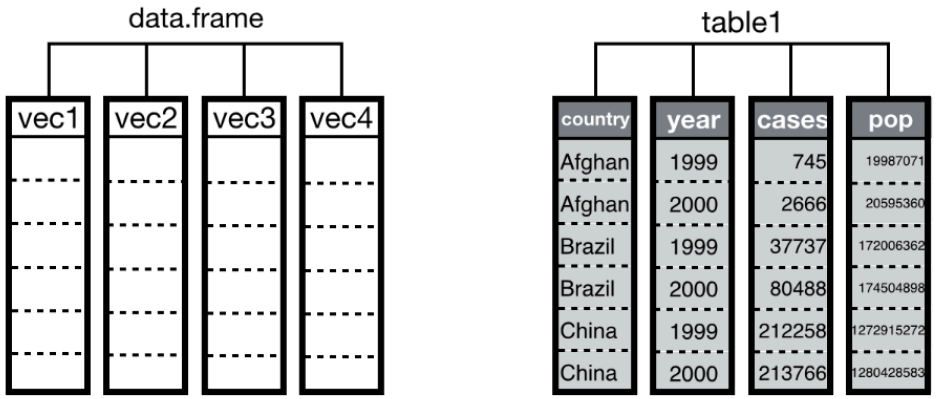




- Consistently display values AND relationships.

https://garrettgman.github.io/tidying/

Tidy Data

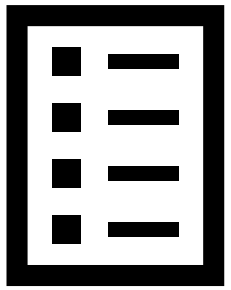


A data frame is a list of vectors that R displays as a table. When your data is tidy, the values of each variable fall in their own column vector.

- Best format for R's functions – saves time and energy later!

https://garrettgman.github.io/tidying/

Sample Workflow



1. Make a plan

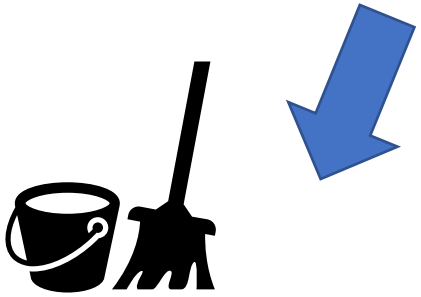
- Outline what you want to do
- Don’t avoid pen & paper.

2. Explore the data

- Get to know the dataset(s).
- Note changes to be made & questions.

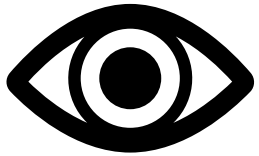
3. Gather Info to resolve uncertainties

- Ask questions!
- Research.
- Further explore the data.



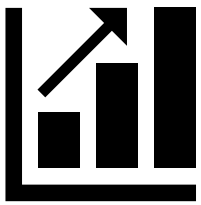
4. Get the data into a usable format

- Includes modifying or creating new columns.
- Create a reusable script.



5. Verify

- Re-explore to check distributions, format, etc.



6. Analyze!

- Data are ready to use.

A gift to your future self

- Create a script (or defined part of a script) that gets your data into ‘working order’.
  - Hit run and go.
  - Comment to remember what you did (or did not do) and why!
  - Promotes reproducibility and collaboration.
- Recycle functions or general processes for future projects.
- Make a copy of your data frame or set it up to easily be reloaded.
  - Ask me about hours spend re-pulling data in from an SQL database….

1. Make a Plan

Before diving into the data…

- Outline what you want to do
  - Goals and sub-goals
- Sample questions
  - Variables needed
  - Observations needed
  - The form of data (wide versus long)

2. Explore the Data

Our goal here is to get a sense of the data, not to make beautiful plots.

Initial Exploration

- Explore dataset characteristics
  - Size
  - Shape
  - Contents
  - Data types
  - Keys for relational databases
  - Sample observations

For a review of datatypes:

https://swcarpentry.github.io/r-novice-inflammation/13-supp-data-structures/

df; View(df); names(df); glimpse(df); str(df)
dim(df); nrow(df); ncol(df); length(df$col)
Wide: columns usually represent groups.
Summary(df); summary(df$colName)
Typeof(df$column)
df %>% count(primaryKey) %>% filter(n > 1)
Head(df); tail(df); df[c(3:5),]; df[“rowName”,]

Initial Exploration

- Explore dataset contents
  - Summary statistics / data profiling
- Questions to ask yourself
  - Anything unexpected?
  - Are any known values correct?
    - *Mean, median, range, mode, counts, etc.*

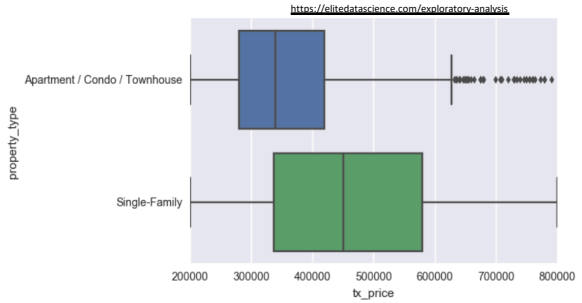
- Base functions to get a quick overview
  - *Summary(df)*
  - *Summary(df$numericColumn)*
  - *Table(df$characterColumn)*
- You may prefer to do column renaming or other tidying before this step.

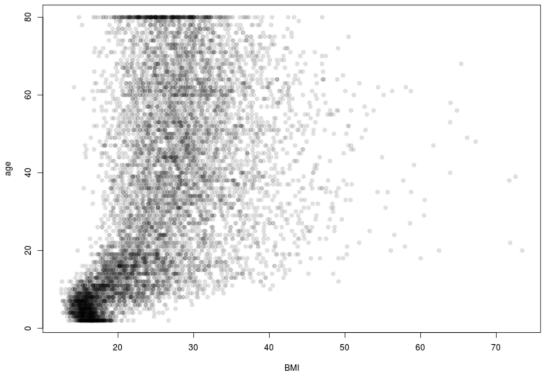
Initial Exploration: Within Column Plots

- Numerical distributions
  - Histograms
    - *Hist(df$numericColumn)*
  - Watch out for: Unexpected trends, ranges, outliers, scale/sub-ideal data types, etc.
- Categorical distributions
  - Bar plots
    - *barplot(vectororMatrixWithNumericCounts); barplot(table(df$characterColumn))*
  - Watch out for: small categories, categories to collapse, ordering, trends, etc.

Initial Exploration: Between Column Plots

- Categorical by numerical distributions
  - Box plots
    - *Boxplot(formula, df)*
- Numerical by numerical
  - Correlations
    - *Cor()*
  - Scatterplot
    - *Plot(df$var1, df$var2)*
- Categorical by categorical
  - Mosaic plot
    - *Mosaicplot(table(df$var1, df$var2))*





Plot tutorial: https://datawookie.netlify.com/blog/2013/05/plotting-categorical-variables/

3. Gather Information

Ask/Inquire

- Primary vs. secondary data
- Questions may be about anything you require to better understand and use the data!
  - Where the data come from.
  - Data definitions/dictionary.
  - Standard practices, e.g. units.

- Goal: to feel confident in decisions about how to manipulate the data going forward.
  - May require many consultations and looking for data in different places.

Example: Electronic Health record ‘level of education’ field

Field value	Actual Meaning
“ ”	
Do not know	
Prefer not to answer	
Null	
Unknown	
Undefined	
Other	
Defined value	

Example: Electronic Health record ‘level of education’ field

The value of consultation with people who generate or maintain the data!

Field value	Actual Meaning
“ ”	Not asked
Do not know	Client was asked or given the opportunity to provide info
Prefer not to answer	Client was asked or given the opportunity to provide info
Null	Not asked
Unknown	Not asked
Undefined	Client provided information that did not fit a predefined category
Other	Client provided information that did not fit a predefined category
Defined value	As stated (4 levels) 'No formal education’ to ‘Post secondary or equivalent’

Mix of important and non-important differences here.

Your goal will determine how you ‘clean’ these.

Example: Electronic Health record ‘level of education’ field

It is important to consult with people who generate or maintain the data

Field value	Actual Meaning
“ ”	Not asked
Do not know	Client was asked or given the opportunity to provide info
Prefer not to answer	Client was asked or given the opportunity to provide info
Null	Not asked
Unknown	Not asked
Undefined	Client provided information that did not fit a predefined category
Other	Client provided information that did not fit a predefined category
Defined value	As stated (4 levels) 'No formal education’ to ‘Post secondary or equivalent’

Treating all these the same may introduce bias.

(Not wanting to tell a care provider about your education is different than them never asking you about it.)

Example: Electronic Health record ‘level of education’ field

How these values are treated could change results.

Field value	Actual Meaning
“ ”	Not asked
Do not know	Client was asked or given the opportunity to provide info
Prefer not to answer	Client was asked or given the opportunity to provide info
Null	Not asked
Unknown	Not asked
Undefined	Client provided information that did not fit a predefined category
Other	Client provided information that did not fit a predefined category
Defined value	As stated (4 levels) 'No formal education’ to ‘Post secondary or equivalent’

Treating these differently may introduce bias

(Values are based on care provider behaviour and location / the type of EMR being used).

4. Cleaning

Remove Irrelevant and Unwanted Data

- Remove columns – complete variable removal
  - Declutter (e.g. unneeded, duplicate, highly correlated (unless using for e.g. missing data).
  - **Dplyr::select()** can specify what to keep or remove.

- Remove rows
  - Duplicate observations, e.g. if joined datasets
  - Unwanted observations, e.g. to refine target population
  - Outliers should only be removed if you have a good reason!
  - **Dplyr::filter()** keeps a subset of rows in a column.

Remodel

- Combine or recode analogous categorical variables
  - E.g. syntactic fixes (capitalization, spelling, synonyms), domain-knowledge decisions (similar categories), small categories (Keep? Remove? Combine?)
  - E.g. **dplyr::arrange()** to alter ordering
- Work with strings (bunch of characters)
  - E.g. search and replace.
- Missing data strategy
  - Remember missingness can be informative.
- Standardize
  - Rename variables
    - camelCaseIsMyFavourite
    - Avoid_spaces_and_most_nonalphanum_characters
  - Consistent units for numerical variables
- Convert between datatypes
  - *is.type()* gives True/False
  - *as.type()* converts it

Create (usually new columns)

- Calculate new variables
  - E.g. percentOfPopulation = cases / population * 100
- Dummy variables
  - Each level of a categorical variable is becomes a new column with values 0 or 1.
- Aggregate values
  - E.g. Diagnostic codes to identify everyone with a particular health condition.

- Note you may want to remove columns that become irrelevant.

5. Verify

Data Exploration

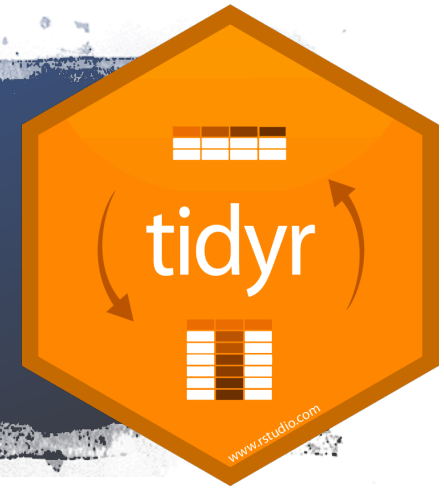
- Is everything ready to go?
- Can use similar tools as initial data exploration.
  - May want prettier presentation…Check out previous meetup materials!

6. Analyze

For another day!

Packages & Functions

Tidyr: reshape data



Long

country	year	key	value
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

table2

Wide

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

spread(table2, key, value)
spread(df, “keyColumnName”, “ValueColumnName”)

Distribute key:value columns into data cells.

New key

country	year	cases
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

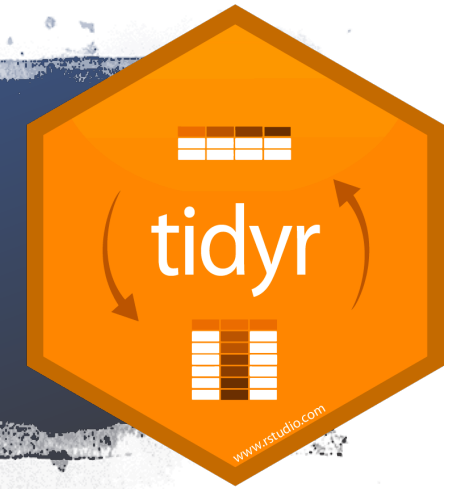
country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

table4

gather(table4, "year", "cases", 2:3)
gather(df, “keyColumnName”, “valueColumnName”, columnsToUse)

Collect column names and put them into a single ‘key’ column.

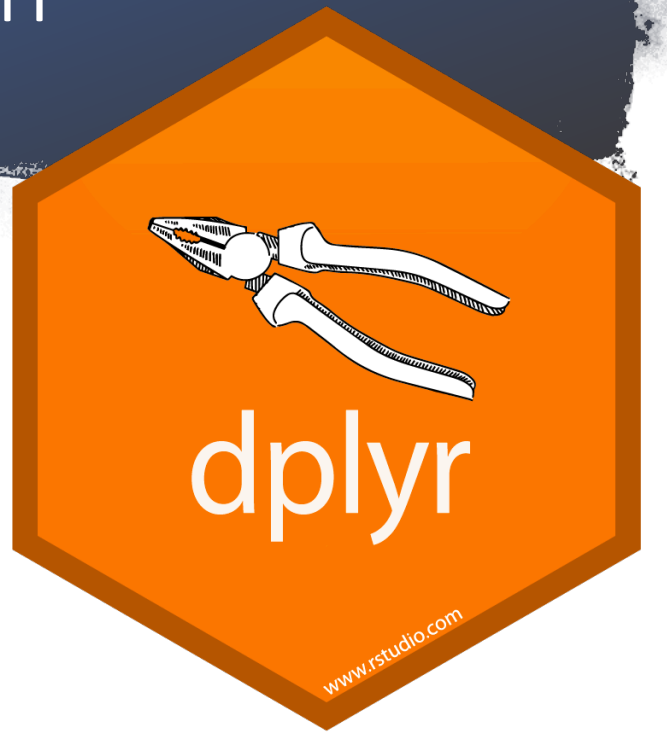
Tidyr: reshape data



- Other useful functions: separate() and unite()

- Overview & cheatsheet: https://tidyr.tidyverse.org/
- Check out: https://garrettgman.github.io/tidying/ or http://www.milanor.net/blog/reshape-data-r-tidyr-vs-reshape2/ for comparison with reshape2

Dplyr: data manipulation



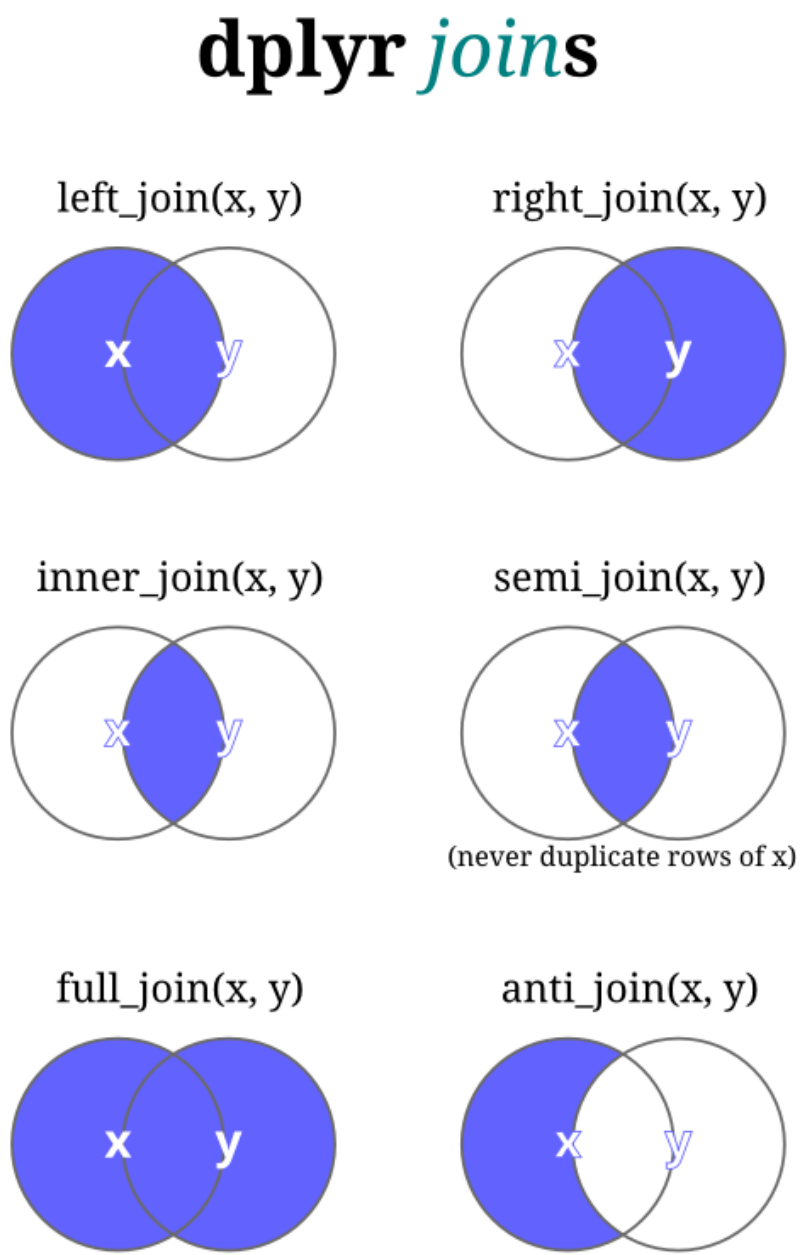
- Verbs (we saw some earlier) :
  - filter
  - select
  - mutate
  - arrange
  - summarize
  - group_by
  - Recode / rename

- Overwrite or save to a new dataframe.
- Overview & cheat sheet: https://dplyr.tidyverse.org/
- Tutorial with exercises: https://datacarpentry.org/2015-07-22-JamesMadison/etc/r-datacarpentry.html

Joins

- With dplyr: http://lindsaydbrin.github.io/CREATE_R_Workshop/Lesson_-_dplyr_join.html

- Relational databases: https://r4ds.had.co.nz/relational-data.html



https://rpubs.com/jcross/joins

Purrr: work with functions & vectors

Apply Functions

Map functions apply a function iteratively to each element of a list or vector.

map(.x, .f, ...) Apply a function to each element of a list or vector. *map(x, is.logical)*

map2(.x, .y, .f, ...) Apply a function to pairs of elements from two lists, vectors. *map2(x, y, sum)*

pmap(.l, .f, ...) Apply a function to groups of elements from list of lists, vectors. *pmap(list(x, y, z), sum, na.rm = TRUE)*

invoke_map(.f, .x = list(NULL), ..., .env=NULL) Run each function in a list. Also **invoke**. *l <- list(var, sd); invoke_map(l, x = 1:9)*

lmap(.x, .f, ...) Apply function to each list-element of a list or vector.
imap(.x, .f, ...) Apply .f to each element of a list or vector and its index.

OUTPUT

map(), map2(), pmap(), imap and **invoke_map** each return a list. Use a suffixed version to return the results as a specific type of flat vector, e.g. **map2_chr**, **pmap_lgl**, etc.

Use **walk**, **walk2**, and **pwalk** to trigger side effects. Each return its input invisibly.

function	returns
map	list
map_chr	character vector
map_dbl	double (numeric) vector
map_dfc	data frame (column bind)
map_dfr	data frame (row bind)
map_int	integer vector
map_lgl	logical vector
walk	triggers side effects, returns the input invisibly

SHORTCUTS - within a purrr function:

"name" becomes **function(x) x[["name"]]**, e.g. *map(l, "a")* extracts *a* from each element of *l*

~ .x .y becomes **function(.x, .y) .x .y**, e.g. *map2(l, p, ~ .x + .y)* becomes *map2(l, p, function(l, p) l + p)*

~ .x becomes **function(x) x**, e.g. *map(l, ~ 2 + .x)* becomes *map(l, function(x) 2 + x)*

~ ..1 ..2 etc becomes **function(..1, ..2,** etc**) ..1 ..2** etc, e.g. *pmap(list(a, b, c), ~ ..3 + ..1 - ..2)* becomes *pmap(list(a, b, c), function(a, b, c) c + a - b)*

RStudio

Work with Lists

FILTER LISTS

pluck(.x, ..., .default=NULL) Select an element by name or index, *pluck(x,"b")* ,or its attribute with **attr_getter**. *pluck(x,"b",attr_getter("n"))*

keep(.x, .p, ...) Select elements that pass a logical test. *keep(x, is.na)*

discard(.x, .p, ...) Select elements that do not pass a logical test. *discard(x, is.na)*

compact(.x, .p = identity) Drop empty elements. *compact(x)*

head_while(.x, .p, ...) Return head elements until one does not pass. Also **tail_while**. *head_while(x, is.character)*

RESHAPE LISTS

flatten(.x) Remove a level of indexes from a list. Also **flatten_chr**, **flatten_dbl**, **flatten_dfc**, **flatten_dfr**, **flatten_int**, **flatten_lgl**. *flatten(x)*

transpose(.l, .names = NULL) Transposes the index order in a multi-level list. *transpose(x)*

SUMMARISE LISTS

every(.x, .p, ...) Do all elements pass a test? *every(x, is.character)*

some(.x, .p, ...) Do some elements pass a test? *some(x, is.character)*

has_element(.x, .y) Does a list contain an element? *has_element(x, "foo")*

detect(.x, .f, ..., .right=FALSE, .p) Find first element to pass. *detect(x, is.character)*

detect_index(.x, .f, ..., .right = FALSE, .p) Find index of first element to pass. *detect_index(x, is.character)*

vec_depth(x) Return depth (number of levels of indexes). *vec_depth(x)*

JOIN (TO) LISTS

append(x, values, after = length(x)) Add to end of list. *append(x, list(d = 1))*

prepend(x, values, before = 1) Add to start of list. *prepend(x, list(d = 1))*

splice(...) Combine objects into a list, storing S3 objects as sub-lists. *splice(x, y, "foo")*

TRANSFORM LISTS

modify(.x, .f, ...) Apply function to each element. Also **map**, **map_chr**, **map_dbl**, **map_dfc**, **map_dfr**, **map_int**, **map_lgl**. *modify(x, ~.+ 2)*

modify_at(.x, .at, .f, ...) Apply function to elements by name or index. Also **map_at**. *modify_at(x, "b", ~.+ 2)*

modify_if(.x, .p, .f, ...) Apply function to elements that pass a test. Also **map_if**. *modify_if(x, is.numeric,~.+2)*

modify_depth(.x,.depth,.f,...) Apply function to each element at a given level of a list. *modify_depth(x, 1, ~.+ 2)*

WORK WITH LISTS

array_tree(array, margin = NULL) Turn array into list. Also **array_branch**. *array_tree(x, margin = 3)*

cross2(.x, .y, .filter = NULL) All combinations of .x and .y. Also **cross**, **cross3**, **cross_df**. *cross2(1:3, 4:6)*

set_names(x, nm = x) Set the names of a vector/list directly or with a function. *set_names(x, c("p", "q", "r"))* *set_names(x, tolower)*

Reduce Lists

reduce(.x, .f, ..., .init) Apply function recursively to each element of a list or vector. Also **reduce_right**, **reduce2**, **reduce2_right**. *reduce(x, sum)*

accumulate(.x, .f, ..., .init) Reduce, but also return intermediate results. Also **accumulate_right**. *accumulate(x, sum)*

Modify function behavior

compose() Compose multiple functions.

lift() Change the type of input a function takes. Also **lift_dl**, **lift_dv**, **lift_ld**, **lift_lv**, **lift_vd**, **lift_vl**.

rerun() Rerun expression n times.

negate() Negate a predicate function (a pipe friendly !)

partial() Create a version of a function that has some args preset to values.

safely() Modify func to return list of results and errors.

quietly() Modify function to return list of results, output, messages, warnings.

possibly() Modify function to return default value whenever an error occurs (instead of error).

Nested Data

A **nested data frame** stores individual tables within the cells of a larger, organizing table.

"cell" contents

nested data frame

n_iris

n_iris$data[[1]]

n_iris$data[[2]]

n_iris$data[[3]]

Use a nested data frame to:

- preserve relationships between observations and subsets of data
- manipulate many sub-tables at once with the **purrr** functions **map()**, **map2()**, or **pmap()**.

Use a two step process to create a nested data frame:
1. Group the data frame into groups with **dplyr::group_by()**
2. Use **nest()** to create a nested data frame with one row per group

n_iris <- iris %>% **group_by**(Species) %>% **nest**()

tidyr::**nest**(data, ..., .key = data)
For grouped data, moves groups into cells as data frames.

Unnest a nested data frame with **unnest()**:

n_iris %>% **unnest**()

tidyr::**unnest**(data, ..., .drop = NA, .id=NULL, .sep=NULL)
Unnests a nested data frame.

RStudio

List Column Workflow

Nested data frames use a **list column**, a list that is stored as a column vector of a data frame. A typical **workflow** for list columns:

1 **Make** a list column

2 **Work with** list columns

3 **Simplify** the list column

n_iris <- iris %>% **group_by**(Species) %>% **nest**()

mod_fun <- function(df) lm(Sepal.Length ~ ., data = df)

m_iris <- n_iris %>% **mutate**(model = **map**(data, mod_fun))

b_fun <- function(mod) coefficients(mod)[[1]]

m_iris %>% **transmute**(Species, beta = **map_dbl**(model, b_fun))

1. MAKE A LIST COLUMN - You can create list columns with functions in the **tibble** and **dplyr** packages, as well as **tidyr**'s nest()

tibble::**tribble**(...)
Makes list column when needed

```
tribble(~max, ~seq,
  3,  1:3,
  4,  1:4,
  5,  1:5)
```

tibble::**tibble**(...)
Saves list input as list columns
tibble(max = c(3, 4, 5), seq = list(1:3, 1:4, 1:5))

tibble::**enframe**(x, name="name", value="value")
Converts multi-level list to tibble with list cols
enframe(list('3'=1:3, '4'=1:4, '5'=1:5), 'max', 'seq')

dplyr::**mutate**(.data, ...) Also **transmute**()
Returns list col when result returns list.
mtcars %>% **mutate**(seq = **map**(cyl, seq))

dplyr::**summarise**(.data, ...)
Returns list col when result is wrapped with **list()**
mtcars %>% group_by(cyl) %>%
summarise(q = **list**(quantile(mpg)))

2. WORK WITH LIST COLUMNS - Use the purrr functions **map()**, **map2()**, and **pmap()** to apply a function that returns a result element-wise to the cells of a list column. **walk()**, **walk2()**, and **pwalk()** work the same way, but return a side effect.

purrr::**map**(.x, .f, ...)
Apply .f element-wise to .x as .f(.x)
n_iris %>% **mutate**(n = **map**(data, dim))

purrr::**map2**(.x, .y, .f, ...)
Apply .f element-wise to .x and .y as .f(.x, .y)
m_iris %>% **mutate**(n = **map2**(data, model, list))

purrr::**pmap**(.l, .f, ...)
Apply .f element-wise to vectors saved in .l
m_iris %>%
mutate(n = **pmap**(list(data, model, data), list))

3. SIMPLIFY THE LIST COLUMN (into a regular column)

Use the purrr functions **map_lgl()**, **map_int()**, **map_dbl()**, **map_chr()**, as well as tidyr's **unnest()** to reduce a list column into a regular column.

purrr::**map_lgl**(.x, .f, ...)
Apply .f element-wise to .x, return a logical vector
n_iris %>% **transmute**(n = **map_lgl**(data, is.matrix))

purrr::**map_int**(.x, .f, ...)
Apply .f element-wise to .x, return an integer vector
n_iris %>% **transmute**(n = **map_int**(data, nrow))

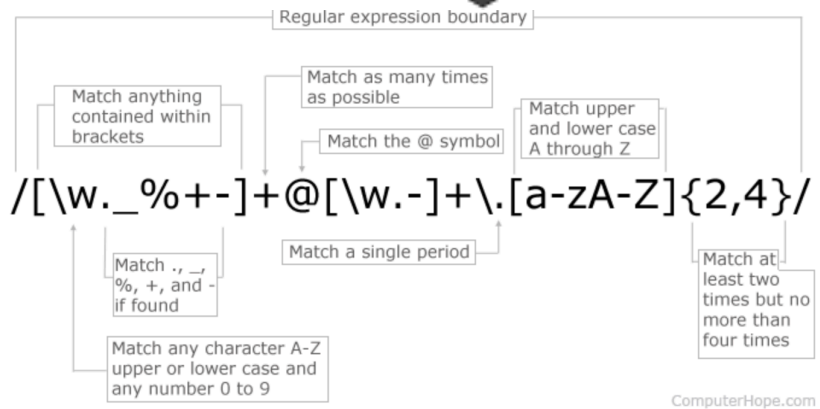
purrr::**map_dbl**(.x, .f, ...)
Apply .f element-wise to .x, return a double vector
n_iris %>% **transmute**(n = **map_dbl**(data, nrow))

purrr::**map_chr**(.x, .f, ...)
Apply .f element-wise to .x, return a character vector
n_iris %>% **transmute**(n = **map_chr**(data, nrow))

- Overview & cheatsheet: https://purrr.tidyverse.org/

Stringr: working with strings

- Strings are series of characters.
  - Create using ‘ ‘ or “ “
- Stringr helps you match, split, and manipulate character data.
  - Relies on regular expressions
- Can use to ‘tidy’ character variable types.
  - E.g. apply a stringr function to each row in a character variable column.
  - For loops or functions can be helpful.
- Overview & cheatsheet: https://www.rdocumentation.org/packages/stringr/versions/1.4.0



magrittR: code readability

- Primary operator is the pipe: %>%
  - 'evaluate the left hand side, and feed the result as input to the function on the right hand side'
- Main advantage: code readability
  - Use with things like subsetting, arranging, and mutating data.
- Overview: https://magrittr.tidyverse.org/
- Tutorial with dplyr: https://seananderson.ca/2014/09/13/dplyr-intro/

Lubridate: dates and time

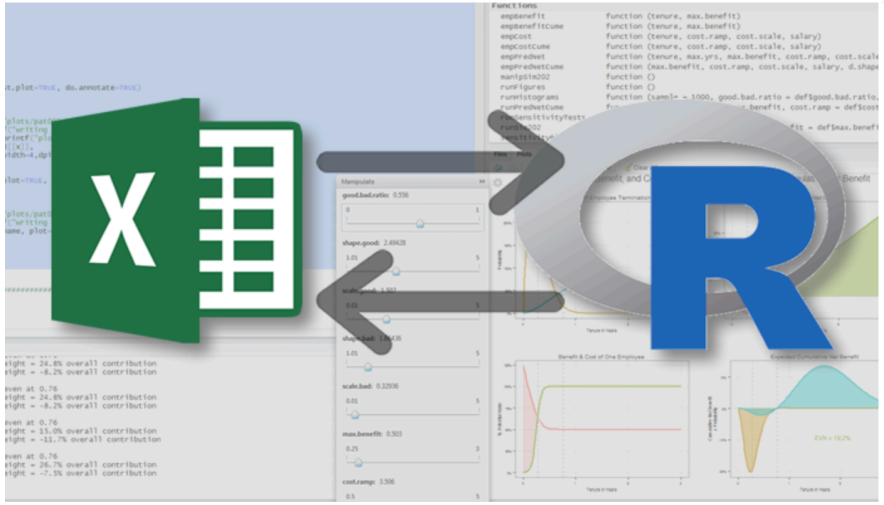
- Overview & cheat sheet: https://lubridate.tidyverse.org/

- How dates work in excel: https://www.excelcampus.com/functions/how-dates-work-in-excel/
  - Days since origin (the beginning of time: January 1, 1990 is common)
  - Reformatted to e.g. 05/30/2019

Tidyxl: messy excel data

- Human readable → machine readable

- Resources:
  - https://cran.r-project.org/web/packages/tidyxl/vignettes/tidyxl.html
  - https://github.com/nacnudus/tidyxl



Possibly helpful dictionary for those transitioning from Excel to R: https://paulvanderlaken.com/2018/07/31/transitioning-from-excel-to-r-dictionary-of-common-functions/

Practice

Data cleaning or anything from a previous meetup! Use each other for trouble shooting!

Practice time!

- Option 1: BYOD
  - Plan, try, chat, help, learn.
- Option 2:
  - Dataset: Starbucks
  - Play around with data manipulation.
  - Or make a goal, e.g.:
    - Does sugar, calorie, and fat content predict caffeine content of coffee beverages?
    - What type of beverage prep is the healthiest?
- Option 3: https://makingnoiseandhearingthings.com/2018/04/19/datasets-for-data-cleaning-practice/
- Buddy up if you like!
- Project on the screen if you like!

Resources

- "Great R packages for data import, wrangling and visualization"
  - https://www.computerworld.com/article/2921176/great-r-packages-for-data-import-wrangling-visualization.html
- https://garrettgman.github.io/tidying/
- Cheatsheet:
  - https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf
- Helpful blog from Thea's meetup:
  - https://whattheyforgot.org/save-source.html
- Table of useful R commands: https://www.calvin.edu/~scofield/courses/m143/materials/RcmdsFromClass.pdf