Tidyverse

Cheat Sheet

yDiv crash course

September 2022

Emilio Berti

emilio.berti@idiv.de

tibble

tibble(x, y)
Create a new tibble with columns x and y.

as_tibble(x)
Cast object x into a tibble.

Use as_tibble(x) if x is a matrix, a list, or a data.frame.

readr

Reading files

read_csv(file)
Read a comma-separated file.

read_delim(file, delim = ...)
Read a ...-separated file.

Use the option **show_col_types = FALSE** to silence printing of column types.

Writing files

write_csv(x, file)
Write table x as a comma-separated file.

write_delim(x, file , delim = ...)
Write table x as a ...-separated file.

tidyr

pivot_wider(df, **names_from** = x, **values_from** = y)
Make a long table a wide one by adding a new column for each value in x with entries the values of y.

pivot_longer(df, **cols** = ..., **names_to** = x, **values_to** = y)
Make a wide table a long one by creating two new column x, with levels the column names selected in *cols*, and y, with their values.

cols = ... takes *tidyselect* arguments.

tidyselect

everything()
Select all columns.

contains("...")
Select all columns matching pattern "...".

starts_with(" ... ")
Select all columns starting with pattern "...".

ends_with("...")
Select all columns ending with pattern "...".

tidyselect examples

select(df, **starts_with("Average"))**
Select all columns that start with the string "Average".

pivot_wider(df, **cols = contains("Site"))**
Select all columns that contain the string "Site".

where

cols = where(...)
Select all columns where statement ... is TRUE.

select(df, **where(is.numeric))**
Select all numeric columns.

mutate(df, **across(where(is.numeric), scale))**
Scale all numeric columns.

purrr

One input

map(x, **function)**
Apply a function to each element of x and return a list.

map_dbl(x, **function)**
Apply a function to each element of x and return a numeric vector.

map_chr(x, **function)**
Apply a function to each element of x and return a character vector.

These two syntaxes are equivalent:
map(z, **round)**
map(z, ∼**round**(.x))

The function can be passed as a function or as a formula, in which case .x refers to the first input and .y to the second.

Two inputs

map2(x, y, **function)**
Apply a function to each pair of elements x and y and return a list.

map2(x, y, ∼**round**(.x, .y))
Round x by y decimal digits.

The names .x and .y are conventions independent of the name of the inputs. E.g. **map2**(z, w, ∼**round**(.x, .y)) is correct.

Many inputs

pmap(list (...) , function)
Apply a function to each group on inputs ... and return a list.

pmap(list(x, y, z, w) ∼**..1 ^ ..2 + ..3 ^ ..4)**
Equivalent to $x^y + z^w$.

When there are more than two inputs (always passed as a list), then the convetion is to use ..1 for the first input, ..2 for the second, etc.

Both map2 and pmap can return numeric vectors instead of lists (**map2_dbl()** and **pmap_dbl()**) or character vectors (**map2_chr()** and **pmap_chr()**).

Other return types for purrr map families are logical vectors (**map_lgl()**), integer vectors (**map_int()**), and dataframes (**map_df()**)

dplyr

Modify columns

mutate(df, x **= ...)**
Create new column x.

transmute(df, x **= ..., y)**
Create new column x and retain only columns x and y.

Use **transmute()** when you want to create new columns and retain only some columns. This is equivalent to a **mutate()** followed by a **select()**.

Select columns

select(df, x, y**)**
Select columns x and y.

select(df, −x**)**
Select all columns except x.

Filter rows

filter (df, **condition)**
Retain only rows based on condition.

filter (df, x **> 5)**.
Select rows where x > 5.

filter (df, x **> 5,** y **< 3)**
Select rows where x > 5 AND y < 3.

filter (df, x **> 5 |** y **< 3)**
Select rows where x > 5 OR y < 3.

Grouping

group_by(df, x**)**
Group observation by the grouping variable x.

Usually, the grouping variable is categorical, e.g. a string or a factor.

Arrange rows

arrange(df, x**)**
Arrange rows with increasing values of x.

Use the option **decreasing = TRUE** to sort with decreasing values of x.

Extract one column

pull(df, x**)**
Extract column x and return it as a vector.

Summarize observations

summarize(df, **...)**
Create a new table containing the summary statistic ...

summarize(df, **Average = mean(**x**))**.
Summarize all observation to give the average value of x.

If the table is grouped, a row is returned for each group:
df **%>% group_by(**x**)%>% summarize(Avg = mean(**y**))**.
Return average value of y for each level of x.

Sample from tables

slice_head(df, n **= ...)**
Retain only the first ... rows.

slice_tail (df, n **= ...)**
Retain only the last ... rows.

slice_sample(df, n **= ...)**
Retain only ... random rows.
Use the option prop **= ...** , instead of n = ..., to retain a proportion of the rows.

Join two tables

left_join (x, y, by **=** z**)**
Join table x with table y according to a grouping variable z and retain only z values that occurr in x.

inner_join(x, y, by **=** z**)**
Join table x with table y according to a grouping variable z and retain only z values that occurr in both x and y.

full_join (x, y, by **=** z**)**
Join table x with table y according to a grouping variable z and retain all z values.

If the grouping variable has different name in the two tables (e.g. z and w), you must specify the comparison.
left_join (x, y, by **= c('z' = 'w'))**
Join x and y where z = w.