) ata 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

becomes f(x, y)

x %>% f(y) pipes

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

Summarise Cases

summary function



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

₩

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.



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.



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

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

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table



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



distinct(mtcars, gear, rows with duplicate values distinct(.data,...,.keep_all = FALSE) Remove



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



slice_sample(.data, ..., n, prop, weight_by =
NULL, replace = FALSE) Randomly select rows. slice_sample(mtcars, n = 5, replace = TRUE) select a fraction of rows. Use n to select a number of rows and prop to



with the lowest and highest values slice_min(mtcars, mpg, prop = 0.25, slice_min(.data, order_by, ..., n, prop _ties = TRUE) and **slice_max()** Select rows



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



xor()

See ?base::Logic and ?Comparison for help

·II

ARRANGE CASES



arrange(mtcars, desc(mpg)) 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

ADD CASES



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

Manipulate Variables

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



¥

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



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

+



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

Use these helpers with select() and across()

ends_with(match) contains(match)

starts_with(match)

num_range(prefix, range) :, e.g. mpg:cy
all_of(x)/any_of(x, ..., vars) -, e.g. -gear
matches(match) everything()

MANIPULATE MULTIPLE VARIABLES AT ONCE



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

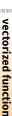
¥

row-wise data. c_across(.cols) Compute across columns in

transmute(rowwise(UKgas), total = sum(c_across(1:2))

MAKE NEW VARIABLES

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

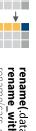




mutate(mtcars, gpm = 1 / mpg) after = NULL) Compute new column(s). Also
add_column(), add_count(), and add_tally(). mutate(.data, ..., .keep = "all", .before = NULL,



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



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



Vectorized Functions

TO USE WITH MUTATE ()

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

vectorized function

OFFSET

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

CUMULATIVE AGGREGATE

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

RANKING

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

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

MISCELLANEOUS

dplyr::case_when() - multi-case if_else()

mutate(type = case_when utate(type - cosc_...height > 200 | mass > 200 ~ "large", height - 200 | mass > 200 ~ "robot" ~ "robot" other"

dplyr::**na_if()** - replace specific values with NA element across a set of vectors dplyr:if_else() - element-wise if() + else() dplyr::coalesce() - first non-NA values by **pmax()** - element-wise max(

Summary Functions

TO USE WITH SUMMARISE (

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

summary function

COUNT

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

POSITION

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

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

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

mad() - median absolute deviation
sd() - standard deviation IQR() - Inter-Quartile Range **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.



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



pmin() - element-wise min()

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

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

Combine Tables

COMBINE VARIABLES



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

COMBINE CASES



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

RELATIONAL DATA

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



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



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



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



left_join(x, y, by = c("C" = suffix = c("1", "2")) have the same name in both tables. give to unmatched columns that Use **suffix** to specify the suffix to have different names in each table. left_join(x, y, by = c("C" = "D"))

A1 B1 C A2 a t 1 d b u 2 b c v 3 a

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





A B C a t 1 b u 2 **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.

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

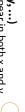
another into a nested data frame. Use a "Nest Join" to inner join one table to



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

SET OPERATIONS

C V B C Rows that appear in both x and y. intersect(x, y, ...



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



a t 1 C V 3 union(x, y, ...)
Rows that appear in x or y. (Duplicates removed). union_all() retains duplicates.

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

