Data Transformation with dplyr Cheat Sheet

**Extract Cases**
Row functions return a subset of rows as a new table. Use a variant that ends in `_` for non-standard evaluation friendly code.

- `filter(data, …)` Extract rows that meet logical criteria. Also `filter_(data, …)`
- `distinct(data, …)` Remove rows with duplicate values. Also `distinct_(data, …)`
- `sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame())` Randomly select fraction of rows.
- `sample_n(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame())` Randomly select size rows.
- `slice(data, …)` Select rows by position. Also `slice_(data, …)`
- `top_n(x, n)` Select and order top n entries (by group if grouped data).
- `select(.data, …)` Extract columns by name. Also `select_if(.data, …)`
- `mutate(.data, …)` Add new column(s). Also `mutate_if(.data, …)`, `mutate_at(.data, …)`, `mutate_all(.data, …)`
- `arrange(data, …)` Order rows by values of a column (low to high). Use with `desc()` to order from high to low.

**Manipulate Cases**
- `mutate(.data, …)` Add new column(s). Also `add_row(.data, …)`
- `add_column(.data, …)` Add new column(s). Also `add_row(.data, …)`
- `rename(.data, …)` Rename columns. Also `rename(.data, …, new = 1:32)`

**Extract Variables**
Column functions return a set of columns as a new table. Use a variant that ends in `_` for non-standard evaluation friendly code.

- `select(data, …)` Extract columns by name. Also `select_if(data, …)`
- `rename(data, …)` Rename columns. Also `rename(.data, …, new = 1:32)`

**Group Cases**
Use `group_by()` to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

- `group_by(.data, …, add = FALSE)` Returns copy of table grouped by …
- `ungroup(x, …)` Returns ungrouped copy of table.

**Summarise Cases**
These apply summary functions to columns to create a new table. Summary functions take vectors as input and return one value (see back).

- `summarise(.data, …)` Compute table of summaries. Also `summarise_all(.data, …)`
- `count(x, …)` Count number of rows in each group defined by the variables in … Also `tally()`

**Logical and boolean operators to use with filter()**

- `<` is.na()
- `<=` is.na()
- `>` is.na()!
- `>=` is.na()!
- `&` xor()

**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).

- `mutate(.data, …)` Compute new column(s). Also `mutate_if(.data, …)`, `mutate_at(.data, …)`, `mutate_all(.data, …)`
- `transmute(.data, …)` Compute new column(s), drop others. Also `transmute_if(.data, …)`, `transmute_at(.data, …)`, `transmute_all(.data, …)`
- `top_n(x, n)` Select and order top n entries (by group if grouped data).
- `top_n(.data, …)`

**Arrange Cases**
Order rows by values of a column (low to high). Use with `desc()` to order from high to low.

- `arrange(.data, …)`
- `arrange(.data, …, .after = NULL)` Add new column(s). Also `add_column(.data, …, .after = NULL)`
- `rename(.data, …, .before = NULL)` Rename columns. Also `rename(.data, …)`

**Manipulate Variables**
- `mutate_at(.data, .cols, .funs, …)` Apply funs to specific columns. Use with `funs()`, `mutate_all(faithful, funs(log(.), log2(.)))`
- `remove(.data, …)` Remove variables.
- `rename(.data, …)` Rename columns. Also `rename(.data, …, new = 1:32)`
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.

**Offsets**

- `dplyr::lag()` - Offset elements by 1
- `dplyr::lead()` - Offset elements by -1

**Cumulative Aggregates**

- `dplyr::cumall()` - Cumulative all()
- `dplyr::cumany()` - Cumulative any()
- `dplyr::cummax()` - Cumulative max()
- `dplyr::cummean()` - Cumulative mean() (Note: This function is not documented in the provided text)
- `dplyr::cummin()` - Cumulative min()
- `dplyr::cumprod()` - Cumulative prod()
- `dplyr::cumsum()` - Cumulative sum()

**Rankings**

- `dplyr::cume_dist()` - Proportion of all values <=
- `dplyr::dense_rank()` - rank with 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

**Misc**

- `dplyr::between()` - x := left & x <= right
- `dplyr::case_when()` - multi-case if else()
- `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
- `dplyr::pmax()` - element-wise max()
- `dplyr::pmin()` - element-wise min()
- `dplyr::recode()` - Vectorized switch() to factors
- `dplyr::recode_factor()` - Vectorized switch() for factors

**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.

**Counts**

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

**Location**

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

**Logicals**

- `mean()` - Proportion of TRUE’s
- `sum()` - # of TRUE’s

**Position/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()` - mean 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. Then use `rownames_to_column()` to move row names into col, `column_to_rownames()` to move col in row names.

- `rownames_to_column()` - Move row names into col.
- `column_to_rownames()` - Move col in row names.

Also has `rownames()` and `remove_rownames()`

**Combine Variables**

- `bind_cols()` to paste tables beside each other as they are.
- `bind_rows()` to paste tables below each other as they are.

**Combine Tables**

- `left_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.
- `right_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.
- `inner_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.
- `full_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.

**Combine Cases**

- `intersect()` to filter one table against the other
- `setdiff()` to test whether two data sets contain the exact same rows (in any order)

**Extract Rows**

- `select()`
- `filter()`
- `arrange()`
- `slice()`
- `head()`
- `tail()`
- `slice_max()`
- `slice_min()`
- `slice_sample()`

- `x` + `y` = `z`

**RStudio** is a trademark of RStudio, Inc. • [DC Be](https://rstudio.com) • info@rstudio.com • 844-448-1212 • rstudio.com

Learn more with browseVignettes(package = c("dplyr", "tibble"))