**dpolyr** 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**.

### 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(.)` 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 columns that meet logical criteria.

### 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.

- `mtcars %>% group_by(cyl) %>% summarise(avg = mean(mpg))`

- `group_by(.data, ..., add = FALSE)` Returns copy of table grouped by ...
  - `g_iris <- group_by(iris, Species)`

### 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.

- `sample_frac(tbl, size = 1, replace = FALSE)` Randomly select fraction of rows.

- `sample_n(tbl, size, replace = FALSE)` Randomly select size rows.

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

See `?base::logic` and `?Comparison` for help.

### 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_if(iris, Sepal.Length > 5)`

### 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, mpg1 = mpg)`

- `mutate_all(.tbl, .funs, ...) Apply funs to every column. Use with `funs()`.

  - `mutate_all(iris, funs( log1(), log2()))`

### ADD CASES

- `add_row(.data, ..., before = NULL, .after = NULL)` Add one or more rows to a table.

  - `add_row(iris, eruptions = 1, sitting = 1)`

- `rename(.data, ...) Rename columns.

  - `rename(iris, Length = Sepal.Length)`
Vector 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.

**vectorized function**

**offsets**

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

cumulative aggregates

dplyr::cumsum() - cumulative sum(dplyr::cummax() - cumulative max(dplyr::cummin() - cumulative min(dplyr::cumprod() - cumulative product(dplyr::cumsum() - cumulative sum

**rankings**

dplyr::cumcount() - 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

dplyr::between() - x <= x < y
dplyr::near() - safe == for floating point numbers

**misc**

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
max() - element-wise max()
dplyr::min() - element-wise min()
dplyr::recod() - vectorized switch()
dplyr::recod_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.

**summary function**

**counts**

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

**location**

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

**logics**

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() - median 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.

rownames_to_column() - move row names into column

Also has rownames(), remove_rownames()

Combine Tables

**combine variables**

Use bind_cols() to paste tables beside each other as they are.

bind_cols(...)

**combine cases**

Use bind_rows() to paste tables below each other as they are.

bind_rows(..., id = NULL)

intersect(x, y, ...)

setdiff(x, y, ...)

union(x, y, ...)

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

**extract rows**

Use a "filtering join" to filter one table against the rows of another.

**semi-join**

semi_join(x, y, by = NULL, ...)

**anti-join**

anti_join(x, y, by = NULL, ...)

Use a "mutating 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.

left_join(x, y, by = NULL, copy = FALSE, suffix = c("_x", "_y"), ...)

right_join(x, y, by = NULL, copy = FALSE, suffix = c("_x", "_y"), ...)

inner_join(x, y, by = NULL, copy = FALSE, suffix = c("_x", "_y"), ...)

full_join(x, y, by = NULL, copy = FALSE, suffix = c("_x", "_y"), ...)

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

left_join(x, y, by = "A")

Use a named vector, by = c("col1" = "col2"), to match on columns that have different names in each table.

left_join(x, y, by = c("C" = "D"))

Use suffix to specify the suffix to give to unmatched columns that have the same name in both tables.

left_join(x, y, by = c("C" = "D"), suffix = c("_1", "_2"))