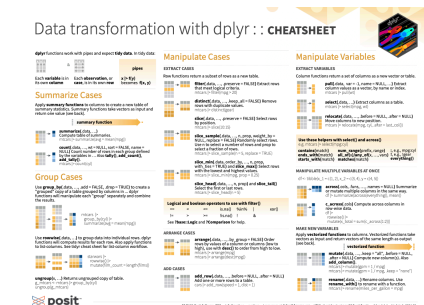**posit**

Data transformation with dplyr :: Cheatsheet

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**
- **pipes** `x |> f(y)` becomes `f(x,y)`

```
library(dplyr)
```

Summarize Cases

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

- `summarize(.data, ...)`: Compute table of summaries.

  ```
  mtcars |> summarize(avg = mean(mpg))
  ```

- `count(.data, ..., wt = NULL, sort = FLASE, name = NULL)`: Count number of rows in each group defined by the variables in `...`. Also `tally()`, `add_count()`, and `add_tally()`.

  ```
  mtcars |> count(cyl)
  ```

Download PDF

Data transformation with dplyr :: CHEATSHEET

Translations (PDF)

- Chinese
- German
- Portuguese
- Russian
- Spanish
- Turkish
- Ukrainian
- Uzbek

Group Cases

- Use `group_by(.data, ..., .add = FALSE, .drop = TRUE)` to created a "grouped" copy of a table grouped by columns in `...`. dplyr functions will manipulate each "group" separately and combine the results.

```
mtcars |>
  group_by(cyl) |>
  summarize(avg = mean(mpg))
```

- 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 cheatsheet for list-column workflow.

```
starwars |>
  rowwise() |>
  mutate(film_count = length(films))
```

- `ungroup(x, ...)`: Returns ungrouped copy of table.

Manipulate Cases

Extract Cases

Row functions return a subset of rows as a new table.

- `filter(.data, ..., .preserve = FALSE)`: Extract rows that meet logical criteria.

```
mtcars |> filter(mpg > 20)
```

- `distinct(.data, ..., .keep_all = FALSE)`: Remove rows with duplicate values.

```
mtcars |> distinct(gear)
```

- `slice(.data, ...,, .preserve = FALSE)`: Select rows by position.

```
mtcars |> slice(10:15)
```

- `slice_sample(.data, ..., n, prop, weight_by = NULL, replace = FALSE)`: Randomly select rows.
  Use `n` to select a number of rows and `prop` to select a fraction of rows.

```
mtcars |> slice_sample(n = 5, replace = TRUE)
```

- `slice_min(.data, order_by, ..., n, prop, with_ties = TRUE)` and `slice_max()`: Select rows with
  the lowest and highest values.

```
mtcars |> slice_min(mpg, prop = 0.25)
```

- `slice_head(.data, ..., n, prop)` and `slice_tail()`: Select the first or last rows.

```
mtcars |> slice_head(n = 5)
```

Logical and boolean operations to use with `filter()`

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

- `&`
- See `?base::Logic` and `?Comparison` for help.

Arrange cases

- `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.

```
mtcars |> arrange(mpg)
mtcars |> arrange(desc(mpg))
```

Add Cases

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

```
cars |> add_row(speed = 1, dist = 1)
```

Manipulate Variables

Extract Variables

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

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

```
mtcars |> pull(wt)
```

- `select(.data, ...)`: Extract columns as a table.

```
mtcars |> select(mpg, wt)
```

- `relocate(.data, ..., .before = NULL, .after = NULL)`: Move columns to new position.

```
mtcars |> relocate(mpg, cyl, after = last_col())
```

Use these helpers with `select()` and `across()`

```
mtcars |> select(mpg:cyl)
```

- `contains(match)`
- `num_range(prefix, range)`
- `:`, e.g., `mpg:cyl`
- `ends_with(match)`
- `all_of(x)` or `any_of(x, ..., vars)`
- `!`, e.g., `!gear`
- `starts_with(match)`
- `matches(match)`
- `everything()`

Manipulate Multiple Variables at Once

```
df <- tibble(x_1 = c(1, 2), x_2 = c(3, 4), y = c(4, 5))
```

- `across(.cols, .fun, ..., .name = NULL)`: summarize or mutate multiple columns in the same way.

  ```
  df |> summarize(across(everything(), mean))
  ```

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

  ```
  df |>
    rowwise() |>
    mutate(x_total = sum(c_across(1:2)))
  ```

Make New Variables

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

- `mutate(.data, ..., .keep = "all", .before = NULL, .after = NULL)`: Compute new column(s). Also `add_column()`.

```
mtcars |> mutate(gpm = 1 / mpg)
mtcars |> mutate(mtcars, gpm = 1 / mpg, .keep = "none")
```

- `rename(.data, ...)`: Rename columns. Use `rename_with()` to rename with a function.

```
mtcars |> rename(miles_per_gallon = mpg)
```

Vectorized Functions

To Use with `mutate()`

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

Offset

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

Cumulative Aggregate

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

Ranking

- `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
- `dplyr::between()`: x >= left & x <= right
- `dplyr::near()`: safe `==` for floating point numbers

Miscellaneous

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

```
starwars |>
  mutate(type = case_when(
    height > 200 | mass > 200 ~ "large",
    species == "Droid" ~ "robot",
    TRUE ~ "other"
  ))
```

- `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

- `pmax()`: element-wise max()

- `pmin()`: element-wise min()

Summary Functions

To Use with `summarize()`

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

Count

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

Position

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

Logical

- `mean()`: proportion of TRUEs
- `sum()`: # of TRUEs

Order

- `dplyr::first()`: first value
- `dplyr::last()`: last value
- `dplyr::nth()`: value in the 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.

- `tibble::rownames_to_column()`: Move row names into col.

```
a <- rownames_to_column(mtcars, var = "C")
```

- `tibble::columns_to_rownames()`: Move col into row names.

```
column_to_rownames(a, var = "C")
```

- Also `tibble::has_rownames()` and `tibble::remove_rownames()`.

Combine Tables

```
x <- tribble(
   ~A,   ~B, ~C,
  "a", "t",  1,
  "b", "u",  2,
  "c", "v",  3
)
```

```
y <- tribble(
  ~A,   ~B, ~D,
  "a", "t",   3,
  "b", "u",   2,
  "d", "w",   1
)
```

Combine Variables

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

Combine Cases

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

Relational Data

Use a **"Mutating Join"** to join one table to columns from another, matching values with the rows that the 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"), ..., keep = FALSE, na_matches = "na")`: Join matching values from `y` to `x`.
- `right_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ..., keep = FALSE, na_matches = "na")`: Join matching 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.

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

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

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

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

Column Matching for Joins

- Use `by = join_by(col1, col2, …)` to specify one or more common columns to match on.

```
left_join(x, y, by = join_by(A))
left_join(x, y, by = join_by(A, B))
```

- Use a logical statement, `by = join_by(col1 == col2)`, to match on columns that have different names in each table.

```
left_join(x, y, by = join_by(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 = join_by(C == D), suffix = c("1", "2"))
```

Set Operations

- `intersect(x, y, ...)`: Rows that appear in both `x` and `y`.
- `setdiff(x, y, ...)`: Rows that appear in `x` but not `y`.
- `union(x, y, ...)`: Rows that appear in x or y, duplicates removed. `union_all()` retains duplicates.
- Use `setequal()` to test whether two data sets contain the exact same rows (in any order).

Learn more at dplyr.tidyverse.org.

Updated: 2024-05.

```r
packageVersion("dplyr")
```

```
[1] '1.1.4'
```