

Introduction to dplyr

Vincent Guillemot



Cheatsheet

data-transformation.pdf

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

$x \mid> f(y)$ becomes $f(x, y)$

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 (see back).

summary function



summarize(.data, ...)
Compute table of summaries.
mtcars > summarize(avg = mean(mpg))



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

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.



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 tidy cheat sheet for list-column workflow.



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

ungroup(x, ...) Returns ungrouped copy of table.
g_mtcars <- mtcars > group_by(cyl)
ungroup(g_mtcars)

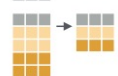
Manipulate Cases

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 operators to use with filter()

=	<	<=	is.na()	%in%		xor()
!=	>	>=	!is.na()	!	&	

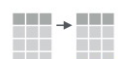
See **?base::Logic** and **?Comparison** for help.

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)

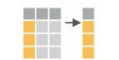
Manipulate Variables

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()

e.g. mtcars > select(mpg:cyl)

contains(match)	num_range(prefix, range)	! e.g., mpg:cyl
ends_with(match)	all_of(x)/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, .funs, ..., .names = 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 back).



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

==== **vectorized function** =====>

OFFSET

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

CUMULATIVE AGGREGATE

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

RANKING

dplyr::cume_dist() - proportion of all values <=
dplyr::dense_rank() - rank w 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
dplyr::pmax() - element-wise max()
dplyr::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.

==== **summary function** =====>

COUNT

dplyr::n() - number of values/rows
dplyr::n_distinct() - # of uniques
summarize(n = n()) - # of non-NA's

POSITION

mean() - mean, also mean(na.rm=TRUE)
median() - median

LOGICAL

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

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.

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

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

Combine Tables

COMBINE VARIABLES

x + y =
a t 1 3
b u 2 2
c v 3
a t 1 3
a t 1 3
d w 1
a t 1 3
b u 2 2
c v 3
d w 1

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.

RELATIONAL DATA

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"), ..., 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.

COLUMN MATCHING FOR JOINS

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"))

COMBINE CASES

x + y =
a t 1
b u 2
a t 1
b u 2
c v 3
d w 4
a t 1
b u 2
c v 3
d w 4
a t 1
b u 2
c v 3
d w 4

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 (as pictured).

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

x + y =
a t 1
b u 2
a t 1
b u 2
c v 3
d w 1

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.

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).



Exercise: read the covid data

- Create an R script named “data-manipulation.R” in your project
- Load the following libraries:
 - dplyr (manipulate data)
 - readr (import data)
- Read the data with function read_csv, assign the result to an object called “covid”
- What is the class of this “covid” object?

What is a tibble?

“stricter checking and better formatting than the traditional data frame.”

In practice: no very very big difference with our usual dataframes...

BUT... no row names!

Going back and forth:

- `base::as.data.frame`
- `tibble::as_tibble`

```
> covid
# A tibble: 32 × 582
   id      time status ABCB1  ABL1  ADA  AHR AICDA
  <chr> <dbl> <chr>  <dbl> <dbl> <dbl> <dbl> <dbl>
1 HC59      0 HC      7.43  6.33  6.87  9.31  3.97
2 HC60      0 HC      7.71  6.28  7.34  8.78  4.6
3 HC68      0 HC      7.01  6.02  7.09  8.69  3.95
4 HC98      0 HC      7.44  6.47  7.31  8.74  3.73
5 HC74      0 HC      7.48  6.53  7.14  9.6   3.89
6 HC88      0 HC      7.25  6.58  7.13 10.1   3.93
7 HC91      0 HC      7.52  6.21  7.06  8.84  4.17
8 HC94      0 HC      6.84  6.51  7.59  9.41  3.77
9 HC75      0 HC      7     6.22  7.07  9.34  3.82
10 HC76     0 HC      7.27  6.41  6.41  9.4   4.88
# i 22 more rows
# i 574 more variables: AIRE <dbl>, APP <dbl>,
# ARG1 <dbl>, ARG2 <dbl>, ARHGDIB <dbl>,
# ATG10 <dbl>, ATG12 <dbl>, ATG16L1 <dbl>,
# ATG5 <dbl>, ATG7 <dbl>, ATM <dbl>, B2M <dbl>,
# B3GAT1 <dbl>, BATF <dbl>, BATF3 <dbl>, BAX <dbl>,
# BCAP31 <dbl>, BCL10 <dbl>, BCL2 <dbl>, ...
# i Use `print(n = ...)` to see more rows, and `colnames`
# `()` to see all variable names
```

These are pipes!

```
print(covid, n = 2)
```

|> : available in base R

```
covid |> print(n = 2)
```

%>% : available through a package (magrittr)

```
covid %>% print(n = 2)
```

ERROR

Error in covid %>% print() : could not find function "%>% »

SOLUTION

```
library(dplyr)
```

Two very useful dplyr verbs

- Select columns with **select()**

```
covid |> select(IL1A)
```

- Filter rows with **filter()**

```
covid |> filter(status == "HC")
```

- **Both:**

```
covid |>
```

```
  filter(status == "HC") |>
```

```
  select(IL1A)
```

How to use a verb?

```
covid |>
```

```
  verb1(arg = val) |>
```

```
  verb2(other_arg = other_val)
```


Grouping and summarizing

Grouping with `group_by`

```
covid |>
  group_by(id) |>
  select(
    id, time,
    status, IFNG)
```

```
# A tibble: 32 × 4
# Groups:   id [13]
   id      time status  IFNG
  <chr> <dbl> <chr> <dbl>
1 HC59      0 HC      4.31
2 HC60      0 HC      5.35
```

Summarizing with `summarize`

```
covid |>
  group_by(status) |>
  summarize(
    mean_IFNG = mean(IFNG))
```

```
# A tibble: 2 × 2
  status mean_IFNG
  <chr>      <dbl>
1 HC      4.99
2 nCOV    4.12
```

Exercise

Compute the mean expression of IL1R1 in healthy controls using pipes and dplyr verbs (or base R if you prefer).

NB: There are several possible solutions. Try to find at least two.