Datatransformation med dplyr : : SNYDEARK

dplyr funktioner arbejder med *pipes* og forventer tidy data. I tidy data er:

Hver variabel i sin egen kolonne   &   Hver observation i sin egen række

pipes

x %>% f(y) bliver til f(x, y)

Opsummer rækker

Anvend opsummeringsfunktioner på kolonner for at lave en ny tabel med opsummeringer. Opsummeringsfunktioner tager vektorer som input og returnerer en værdi (se bagsiden).

opsummeringsfunktion

summarise(.data, …)
Lav tabel med opsummeringer.
summarise(mtcars, avg = mean(mpg))

count(.data, …, wt = NULL, sort = FALSE, name = NULL) Tæl antallet af rækker i hver gruppe defineret af variablene i … Også tally().
count(mtcars, cyl)

Gruppér rækker

Brug group_by(.data, …, .add = FALSE, .drop = TRUE) til at lave en "grupperet" kopi af en tabel grupperet af en kolonne i ... dplyr funktioner vil manipulere hver "gruppe" separat og kombinere resultaterne.

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

Brug rowwise(.data, …) til at gruppere data i individuelle rækker. dplyr funktioner vil lave resultater for hver række. Kan også anvende funktioner på liste-kolonner. Se tidyr snydearket for liste-kolonne workflow.

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

ungroup(x, …) returnerer en ikke-grupperet kopi af tabel.
ungroup(g_mtcars)

Manipuler rækker

UDTRÆK OBSERVATIONER

Rækkefunktioner returnerer specifikke rækker som en ny tabel.

filter(.data, …, .preserve = FALSE) Udtræk rækker der opfylder logiske kriterier.
filter(mtcars, mpg > 20)

distinct(.data, …, .keep_all = FALSE) Fjern rækker med duplikerede værdier.
distinct(mtcars, gear)

slice(.data, …, .preserve = FALSE) Vælg rækker efter position.
slice(mtcars, 10:15)

slice_sample(.data, …, n, prop, weight_by = NULL, replace = FALSE) Vælg rækker tilfældigt. Brug n til at vælge et antal rækker og prop til at vælge en andel af rækker.
slice_sample(mtcars, n = 5, replace = TRUE)

slice_min(.data, order_by, …, n, prop, with_ties = TRUE) and slice_max() Vælg rækker med de laveste og højeste værdier.
slice_min(mtcars, mpg, prop = 0.25)

slice_head(.data, …, n, prop) and slice_tail() Vælg de første eller sidste rækker.
slice_head(mtcars, n = 5)

Logiske og booleske operatorer der kan bruges med filter()

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

Se ?base::Logic og ?Comparison for hjælp.

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.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES

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

Manipuler variable

UDTRÆK VARIABLE

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.
pull(mtcars, wt)

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

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

Use these helpers with select() and across()
e.g. select(mtcars, 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

across(.cols, .funs, …, .names = NULL) Summarise or mutate multiple columns in the same way.
summarise(mtcars, across(everything(), mean))

c_across(.cols) Compute across columns in row-wise data.
transmute(rowwise(UKgas), 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).

vectorized function

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

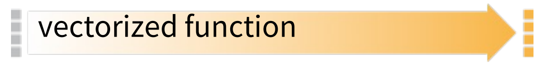
transmute(.data, …) Compute new column(s), drop others.
transmute(mtcars, gpm = 1 / mpg)

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

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.



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

CUMULATIVE AGGREGATE
dplyr::cumall() - cumulative all()
dplyr::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 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
pmax() - element-wise max()
pmin() - element-wise min()

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

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

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

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

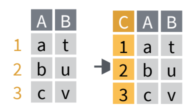
ORDER
dplyr::first() - first value
dplyr::last() - last value
dplyr::nth() - value in nth location of vector

RANK
quantile() - nth quantile
min() - minimum value
max() - maximum value

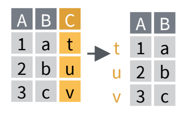
SPREAD
IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

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

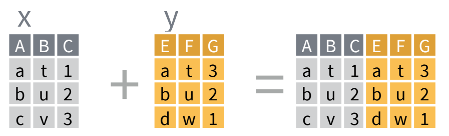


tibble::column_to_rownames()
Move col into row names.
column_to_rownames(a, var = "C")

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

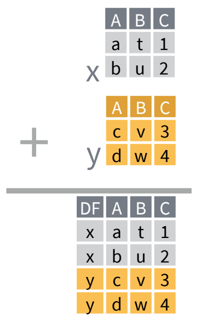
Combine Tables

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.

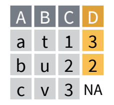
COMBINE CASES



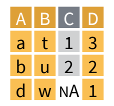
bind_rows(…, .id = NULL) Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured).

RELATIONAL DATA

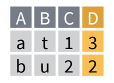
Use a "Mutating Join" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.



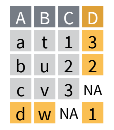
left_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), …, keep = FALSE, na_matched = "na") Join matching values from y to x.



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

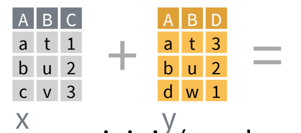


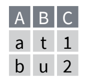
inner_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), …, keep = FALSE, na_matches = "na") Join data. Retain only rows with matches.



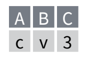
full_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), …, keep = FALSE, na_matches = "na") Join data. Retain all values, all rows.

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



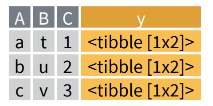


semi_join(x, y, by = NULL, copy = FALSE, …, na_matches = "na") Return rows of x that have a match in y. Use to see what will be included in a join.



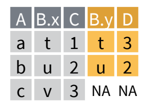
anti_join(x, y, by = NULL, copy = FALSE, …, na_matches = "na") Return rows of x that do not have a match in y. Use to see what will not be included in a join.

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

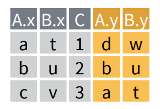


nest_join(x, y, by = NULL, copy = FALSE, keep = FALSE, name = NULL, …) Join data, nesting matches from y in a single new data frame column.

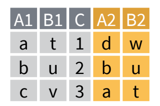
COLUMN MATCHING FOR JOINS



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

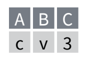


Use a named vector, by = c("col1" = "col2"), to match on columns that have different names in each table.
left_join(x, y, by = c("C" = "D"))

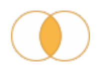


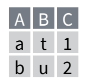
Use suffix to specify the suffix to give to unmatched columns that have the same name in both tables.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

SET OPERATIONS

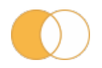


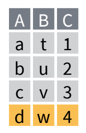
intersect(x, y, …)
Rows that appear in both x and y.



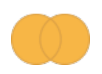


setdiff(x, y, …)
Rows that appear in x but not y.





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).