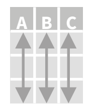
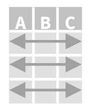
Data Transformation with dplyr : : CHEAT SHEET

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**

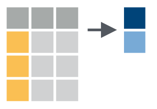
pipes

x %>% f(y) becomes **f(x, y)**

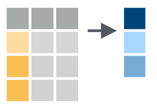
Summarise Cases

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



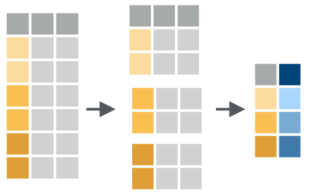
summarise(.data, …**)**
Compute table of summaries.
summarise(mtcars, avg = mean(mpg))



count(x, ..., wt = NULL, sort = FALSE**)**
Count number of rows in each group defined by the variables in … Also **tally()**.
count(mtcars, cyl)

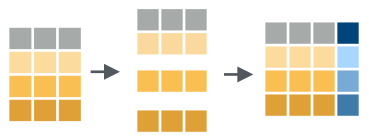
Group Cases

Use **group_by(**.data, ..., .add = FALSE**)** 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) %>%
summarise(avg = mean(mpg))

Use **rowwise(**.data, ...**)** to group data into individual rows. dplyr functions will compute results for each row. Also used to apply functions to list-columns without purrr functions.



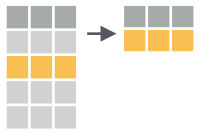
starwars %>%
rowwise() %>%
mutate(film_count = length(films))

ungroup(x, …**)** Returns ungrouped copy of table.
ungroup(g_mtcars)

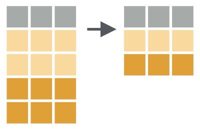
Manipulate Cases

EXTRACT CASES

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

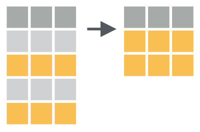


filter(.data, …**)** Extract rows that meet logical criteria.
filter(mtcars, mpg > 20)

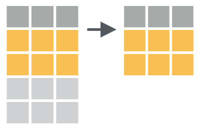


distinct(.data, ..., .keep_all = FALSE**)** Remove rows with duplicate values.
distinct(mtcars, gear)

slice(.data, …**)** Select rows by position.
slice(mtcars, 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.
slice_sample(mtcars, 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.
slice_min(mtcars, mpg, prop = 0.25)

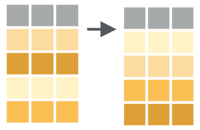
slice_head(.data, ..., n, prop**)** and **slice_tail()** Select the first or last rows.
slice_head(mtcars, n = 5)

Logical and boolean operators to use with filter()

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

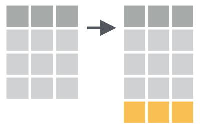
See **?base::Logic** and **?Comparison** for help.

ARRANGE CASES



arrange(.data, …**)** 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

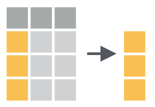


add_row(.data, ..., .before = NULL, .after = NULL**)** Add one or more rows to a table.
add_row(cars, speed = 1, dist = 1)

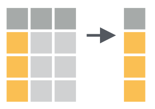
Manipulate Variables

EXTRACT VARIABLES

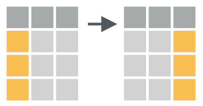
Column functions return a set of columns as a new vector or table.



pull(.data, var = -1**)** Extract column values as a vector. Choose by name or index.
pull(mtcars, wt)



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

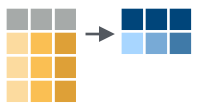


relocate(.data, ..., .before = NULL, .after = NULL**)** Move columns to new position.
relocate(mtcars, mpg, cyl, .after = last_col())

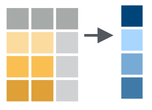
Use these helpers with select() and across()
e.g. select(mtcars, mpg:cyl)

contains(match**)**	**num_range(**prefix, range**)**	**:**, e.g. mpg:cyl
ends_with(match**)**	**one_of(**…**)**	**-**, e.g, -gear
matches(match**)**	**starts_with(**match**)**	**everything()**

MANIPULATE MULTIPLE VARIABLES AT ONCE



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

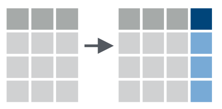


c_across(.cols**)** Compute across columns in row-wise data.
transmute(rowwise(UKgas), n = sum(c_across(1:2)))

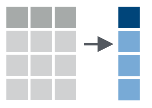
MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

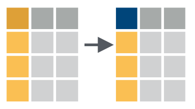
vectorized function



mutate(.data, …, .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.
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.

OFFSETS

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

CUMULATIVE AGGREGATES

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

RANKINGS

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

MISC

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()
dplyr::**recode()** - Vectorized switch()
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
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

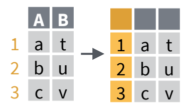
quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

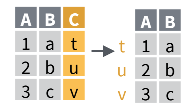
IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

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

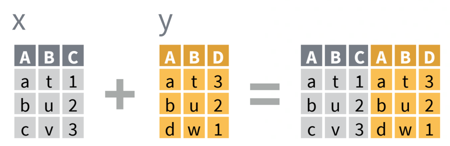


column_to_rownames()
Move col into row names.
column_to_rownames(a, var = "C")

Also **has_rownames()**, **remove_rownames()**

Combine Tables

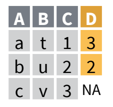
COMBINE VARIABLES



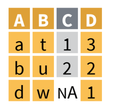
Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(…) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

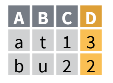
Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.



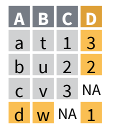
left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…**)**
Join matching values from y to x.



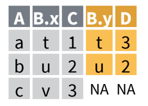
right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…**)**
Join matching values from x to y.



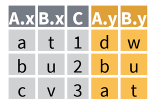
inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…**)**
Join data. Retain only rows with matches.



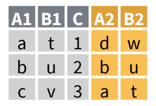
full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…**)**
Join data. Retain all values, all rows.



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

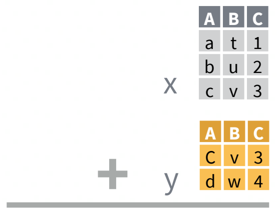


Use a named vector, **by = c("col1" = "col2")**, to match on columns that have different names in each table.
left_join(x, y, by = c("C" = "D"))

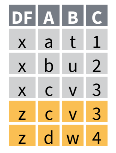


Use **suffix** to specify the suffix to give to unmatched columns that have the same name in both tables.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

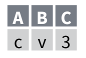
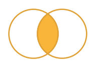
COMBINE CASES



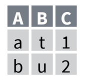
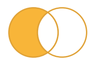
Use **bind_rows()** to paste tables below each other as they are.



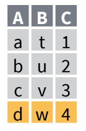
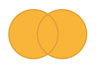
bind_rows(…, .id = NULL**)**
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

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

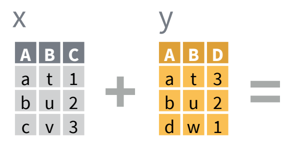
setdiff(x, y, …)
Rows that appear in x but not y.

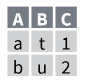
union(x, y, …)
Rows that appear in x or y. (Duplicates removed). union_all() retains duplicates.

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

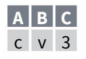
EXTRACT ROWS



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



semi_join(x, y, by = NULL, …**)**
Return rows of x that have a match in y.
USEFUL TO SEE WHAT WILL BE JOINED.



anti_join(x, y, by = NULL, …**)**
Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.