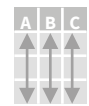


# Data Transformation with dplyr : : CHEAT SHEET



**dplyr** functions work with pipes and expect **tidy data**. In tidy data:



&



**pipes**

Each **variable** is in its own **column**

Each **observation**, or **case**, is in its own **row**

**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. See tidyverse cheatsheet for list-column workflow.



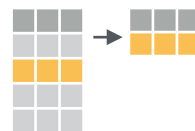
`starwars %>%  
rowwise() %>%  
mutate(film_count = length(films))`

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

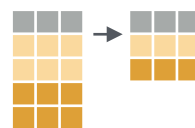
## Manipulate Cases

### EXTRACT CASES

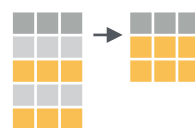
Row functions return a subset of rows as a new table.



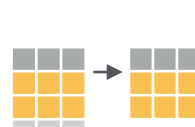
**filter(.data, ...)** Extract rows that meet logical criteria.  
`filter(mtcars, mpg > 20)`



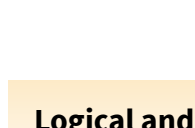
**distinct(.data, ..., .keep\_all = FALSE)** Remove rows with duplicate values.  
`distinct(mtcars, gear)`



**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()	!	&	

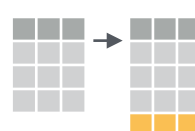
See **?base::Logic** and **?Comparison** for help.

### 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)**  
**ends\_with(match)**  
**matches(match)**

**num\_range(prefix, range)**  
**one\_of(...)**  
**starts\_with(match)**

**: , e.g. mpg:cyl**  
**-, e.g. -gear**  
**everything()**

### MANIPULATE MULTIPLE VARIABLES AT ONCE



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

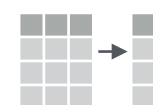


**c\_across(.cols)** Compute across columns in row-wise data.  
`transmute(rowwise(UKgas), n = sum(c_across(1:2)))`

### 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.

**vectorized function**

### 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.

**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

### 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

**x** + **y** = 

A	B	C	A	B	D
a	t	1	a	t	3
b	u	2	b	u	2
c	v	3	c	v	3

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

**x** + **y** = 

A	B	C
a	t	1
b	u	2
c	v	3

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

**x** + **y** = 

A	B	C
a	t	1
b	u	2
c	v	3

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.