

# Data Transformation with dplyr : : CHEAT SHEET



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



## 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).



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

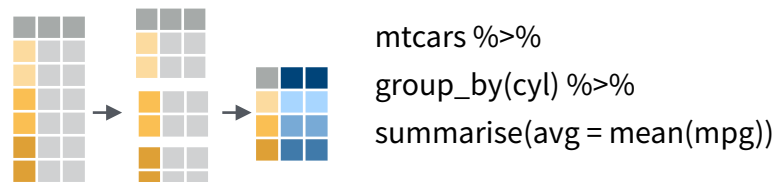
**count(x, ..., wt = NULL, sort = FALSE)**  
Count number of rows in each group defined by the variables in ... Also **tally()**.  
*count(iris, Species)*

### VARIATIONS

- summarise\_all()** - Apply funs to every column.
- summarise\_at()** - Apply funs to specific columns.
- summarise\_if()** - Apply funs to all cols of one type.

## Group Cases

Use **group\_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.



**group\_by(.data, ..., add = FALSE)**  
Returns copy of table grouped by ...  
*g\_iris <- group\_by(iris, Species)*

**ungroup(x, ...)**  
Returns ungrouped copy of table.  
*ungroup(g\_iris)*

## Manipulate Cases

### EXTRACT CASES

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

**filter(.data, ...)** Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*

**distinct(.data, ..., .keep\_all = FALSE)** Remove rows with duplicate values.  
*distinct(iris, Species)*

**sample\_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame())** Randomly select fraction of rows.  
*sample\_frac(iris, 0.5, replace = TRUE)*

**sample\_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame())** Randomly select size rows.  
*sample\_n(iris, 10, replace = TRUE)*

**slice(.data, ...)** Select rows by position.  
*slice(iris, 10:15)*

**top\_n(x, n, wt)** Select and order top n entries (by group if grouped data). *top\_n(iris, 5, Sepal.Width)*

**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(faithful, eruptions = 1, waiting = 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(iris, Sepal.Length)*

**select(.data, ...)**  
Extract columns as a table. Also **select\_if()**.  
*select(iris, Sepal.Length, Species)*

**Use these helpers with select (),**  
*e.g. select(iris, starts\_with("Sepal"))*

**contains(match)**     **num\_range(prefix, range)**     **:**, e.g. mpg:cyl  
**ends\_with(match)**     **one\_of(...)**     **-**, e.g. -Species  
**matches(match)**     **starts\_with(match)**

### 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).



**mutate(.data, ...)**  
Compute new column(s).  
*mutate(mtcars, gpm = 1/mpg)*

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

**mutate\_all(.tbl, .funs, ...)** Apply funs to every column. Use with **funs()**. Also **mutate\_if()**.  
*mutate\_all(faithful, funs(log(.), log2(.)))*  
*mutate\_if(iris, is.numeric, funs(log(.)))*

**mutate\_at(.tbl, .cols, .funs, ...)** Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for select().  
*mutate\_at(iris, vars(-Species), funs(log(.)))*

**add\_column(.data, ..., .before = NULL, .after = NULL)** Add new column(s). Also **add\_count()**, **add\_tally()**. *add\_column(mtcars, new = 1:32)*

**rename(.data, ...)** Rename columns.  
*rename(iris, Length = Sepal.Length)*





# Vector 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 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()` -  $x \geq \text{left} \ \& \ x \leq \text{right}$   
`dplyr::near()` - safe == for floating point numbers

## MISC

`dplyr::case_when()` - multi-case if\_else()  
`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

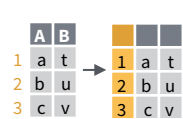
**quantile()** - nth quantile  
**min()** - minimum value  
**max()** - maximum value

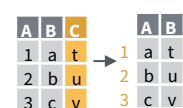
## SPREAD

**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.

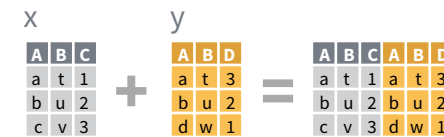
 **rownames\_to\_column()**  
Move row names into col.  
`a <- rownames_to_column(iris, var = "C")`

 **column\_to\_rownames()**  
Move col in row names.  
`column_to_rownames(a, var = "C")`

Also **has\_rownames()**, **remove\_rownames()**

# Combine Tables


## COMBINE VARIABLES

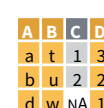


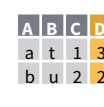
Use **bind\_cols()** to paste tables beside each other as they are.


**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.

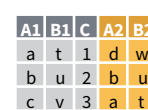
 **right\_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...)**  
Join matching values from x to y.

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

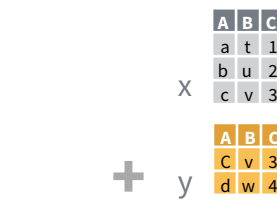
 **full\_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...)**  
Join data. Retain all values, all rows.

 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)

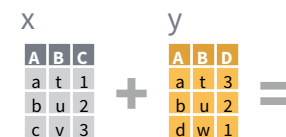
 **intersect(x, y, ...)**  
Rows that appear in both x and y.

 **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.