**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
- x \geq y \implies f(y) becomes f(x, y)

### Summarise Cases

These apply **summary functions** to columns to create a new table. Summary functions take vectors as input and return one value (see back).

- **summarise(.)** Compute table of summaries. Also `summarise()`. `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)`
- **slice_(.)** Select rows by position. Also `slice(_)`. `slice(iris, 10:15)`
- **top_n(, n, wt)** Select and order top n entries (by group if grouped data). `top_n(iris, 5, Sepal.Width)`
- **distinct(, keep_all = FALSE)** Remove rows with duplicate values. Also `distinct(_)`. `distinct(iris, Species)`
- **sample_frac(.tbl, size = 1, replace = FALSE)** 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)`

### 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.

- **mtcars %>% group_by(cyl) %>%** `summarise(avg = mean(mpg))`
- **group_by(.data, ..., add = FALSE)** Returns copy of table grouped by ... `g_iris <- group_by(iris, Species)`
- **ungroup(.)** Returns ungrouped copy of table. `ungroup(g_iris)`

### Manipulate Cases

**EXTRACT CASES**

Row functions return a subset of rows as a new table. Use a variant that ends in _ for non-standard evaluation friendly code.

- **filter(.data, ...)** Extract rows that meet logical criteria. Also `filter(.)`. `filter(iris, Sepal.Length > 7)`
- **distinct(.data, ..., keep_all = FALSE)** Remove rows with duplicate values. Also `distinct(_)`. `distinct(iris, Species)`
- **sample_frac(.tbl, size = 1, replace = FALSE)** 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_(.)** Select rows by position. Also `slice(_)`. `slice(iris, 10:15)`
- **top_n(.data, n, wt)** Select and order top n entries (by group if grouped data). `top_n(iris, 5, Sepal.Width)`

### 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()`. `mutate_all(.iris, funs(log(.), log2(.)))`
- **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(.)))`
- **mutate_if(.tbl, predicate, .funs, ...)** Apply funs to all columns of one type. Use with `funs()`. `mutate_if(.iris, is.numeric, funs(log(.)))`

### ARRANGE CASES

- **arrange(.data, ...)** Order rows by values of a column (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)`
- **add_column(.data, ..., before = NULL, after = NULL)** Add new column(s). `add_column(mtcars, new = 1:32)`
- **rename(.data, ...)** Rename columns. `rename(iris, Length = Sepal.Length)`
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()
cummean() - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsd() - Cumulative sd()
cumsum() - Cumulative sum()

RANKINGS

dplyr::n_distinct() - # of uniques
sum(is.na()) - # of non-NA's

MATH

+ - / ** ^ %/% %% - arithmetic ops
log1(), log2(), log10() - logs
<= < >= - logical comparisons

MISC

dplyr::between() - x <= left & x <= right
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

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
n_distinct() - number of values/rows
sum(is.na()) - # of non-NA's

LOCATIONS

mean() - mean, also mean(t.is.na())
median() - median

LOGICALS

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

RANKS

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

SPREAD

IQR() - Inter-Quartile Range
mad() - mean 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()