Data Transformation with siuba . **CHEAT SHEET**

siuba 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)**

Summarize 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

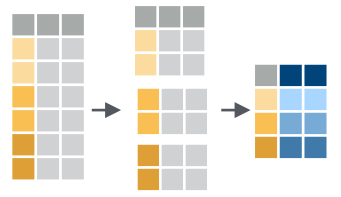
summarize(__data, …)
Compute table of summaries.
summarize(mtcars, avg = _.mean.mpg())

count(x, ..., wt = None, 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.
siuba functions will manipulate each "group" separately and then combine the results.



```
(mtcars >>
   group_by(_.cyl) >>
   summarize(avg = _.mpg.mean())
)
```

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()

<	<=	_.isna()	_.isin()	|	^
>	>=	~_.isna()	~	&	

See **help('operator')** and python's operator docs.

ARRANGE CASES

arrange(__data, …**)** Order rows by values of a column or columns (low to high), use with a minus sign (-) to order from high to low.
arrange(mtcars, _.mpg)
arrange(mtcars, -_.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.
select(iris, Sepal.Length, Species)

Use these helpers with select (),
e.g. select(iris, _.startswith("Sepal"))

_.mpg	_.**contains**(match)	:, e.g. _["mpg":"hp"]
_["mpg"]	_.**endswith**(match)	_[5:10]
	_.**detect**(match)	-, e.g. -_.mpg

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, …**)**
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 Series as input and return Series of the same length as output.

vectorized function

OFFSETS

vector.**lag()** - Offset elements by 1
vector.**lead()** - Offset elements by -1

CUMULATIVE AGGREGATES

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

RANKINGS

vector.**cume_dist()** - Proportion of all values <=
vector.**dense_rank()** -rank w ties = min, no gaps
vector.**min_rank()** - rank with ties = min
vector.**ntile()** - bins into n bins
vector.**percent_rank()** -min_rank scaled to [0,1]
vector.**row_number()** - rank with ties = "first"

MATH

+, - , **, /, //, % - arithmetic ops
numpy.**log()** - logs
<, <=, >, >=, !=, == - logical comparisons
vector.**between()** - (x >= left) & (x <= right)
vector::**near()** - safe == for floating numbers

MISC

siuba.**case_when()** - multi-case if_else()

*iris >> mutate(Species = **case_when({***
_.Species == "versicolor": "versi",
_.Species == "virginica": "virgi",
True: _.Species}))

vector.**coalesce()** - first non-NA values by element across a set of vectors
suiba.**if_else()** - element-wise if() + else()
vector.**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
_.**replace({'a': 'b'})** - Replace 'a' values to 'b'

Summary Functions

TO USE WITH SUMMARIZE ()

summarize() applies summary functions to columns to create a new table. Summary functions take Series as input and return single values as output.

summary function

COUNTS

vector.**n()** - number of values/rows
_.**nunique()** - # of uniques
_.**notna().sum()** - # of non-NA's

LOCATION

_.**mean()** - mean, also _.**notna().mean()**
_.**median()** - median

LOGICALS

_.**mean()** - Proportion of True's
_.**sum()** - # of True's

POSITION/ORDER

vector.**first()** - first value
vector.**last()** - last value
vector.**nth()** - value in nth location of vector

RANK

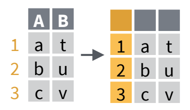
_.**quantile()** - nth quantile
_.**min()** - minimum value
_.**max()** - maximum value

SPREAD

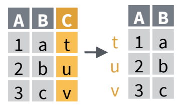
_.**mad()** - median absolute deviation
_.**std()** - standard deviation
_.**var()** - variance
(no method for inter-quartile range)

Pandas Indexes

Tidy data does not use indexes, which store a variable outside of the columns. To work with pandas indexes, first move them into a column.



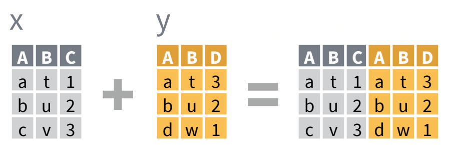
df.reset_index()
Move row names into col.
df.reset_index().rename(…)



df.set_index()
Move col in row names.
df.set_index("C")

Combine Tables

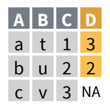
COMBINE VARIABLES



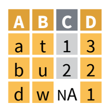
Use **bind_cols()** to paste tables beside each other as they are.

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

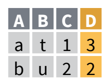
Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.



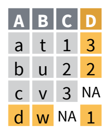
left_join(x, y, on = None)
Join matching values from y to x.



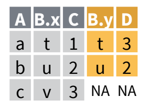
right_join(x, y, on = None)
Join matching values from x to y.



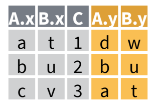
inner_join(x, y, on = None)
Join data. Retain only rows with matches.



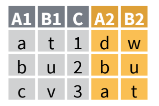
full_join(x, y, on = None)
Join data. Retain all values, all rows.



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

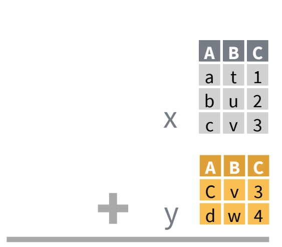


Use a named vector, **on = {"col1": "col2"}**, to match on columns that have different names in each table.
left_join(x, y, on = {"C": "D"})

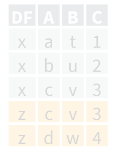


Use **suffix** to specify the suffix to give to unmatched columns that have the same name in both tables.
left_join(x, y, on = {"C": "D"}, suffix = ["1", "2"])

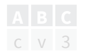
COMBINE CASES



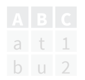
Use **bind_rows()** to paste tables below each other as they are.



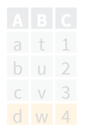
bind_rows(…, .id = NULL)
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)



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



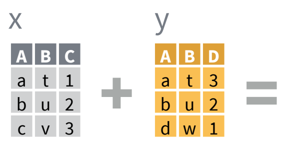
setdiff(x, y, …)
Rows that appear in x but not y.



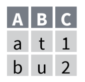
union(x, y, …)
Rows that appear in x or y.
(Duplicates removed). union_all() retains duplicates.

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

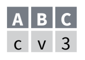
EXTRACT ROWS



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



semi_join(x, y, on = None, …)
Return rows of x that have a match in y.
USEFUL TO SEE WHAT WILL BE JOINED.



anti_join(x, y, on = None, …)
Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.

Adapted from Data Transformation with dplyr (CC BY SA RStudio) • This work CC BY SA Michael Chow • Minimal changes have been made to allow comparison with dplyr • Updated: 2020-05-27