**Chapter 5**

These describe the **type of each variable**:

* int stands for integers.
* dbl stands for doubles, or real numbers.
* chr stands for character vectors, or strings.
* dttm stands for date-times (a date + a time).

There are three other common types of variables:

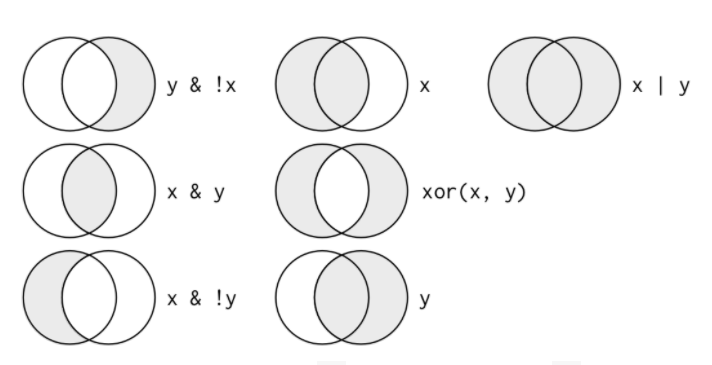
* lgl stands for logical, vectors that contain only TRUE or FALSE.
* fctr stands for factors, which R uses to represent categorical variables with fixed possible values.
* date stands for dates.

**Functions**

* Pick observations by their values (filter()).
* Reorder the rows (arrange()).
* Pick variables by their names (select()).
* Create new variables with functions of existing variables (mutate()).
* Collapse many values down to a single summary (summarise()).

These can all be used in conjunction with group\_by() which changes the scope of each function from operating on the entire dataset to operating on it group-by-group.

**Logical operators**



To filter flights that were either in November or December

**filter**(flights, month == 11 | month == 12)

**not** filter(flights, month == (11 | 12))

**OR**

nov\_dec <- **filter**(flights, month **%in%** **c**(11, 12))

**MORGANS LAW**

!(x & y) is the same as !x | !y, and !(x | y) is the same as !x & !y.

If you want to determine if a value is missing, use is.na():

**is.na**(x)

**FILTER**

filter() **only includes rows where the condition is TRUE;** it excludes both FALSE and NA values. If you want to preserve missing values, ask for them explicitly

**to include NAs**

**filter**(df, **is.na**(x) | x > 1)

**ARRANGE**

arrange() instead of selecting rows, it changes their order

Use desc() to re-order by a column in descending order

**arrange**(flights, **desc**(dep\_delay))

if NA’s first

**arrange**(flights, **desc**(**is.na**(dep\_time)), dep\_time)

The flights will first be sorted by desc(is.na(dep\_time))

**SELECT**

select() allows you to rapidly zoom in on a useful subset using operations based on the names of the variables

There are a number of **helper functions** you can use within select():

* starts\_with("abc"): matches names that begin with “abc”.
* ends\_with("xyz"): matches names that end with “xyz”.
* contains("ijk"): matches names that contain “ijk”.
* matches("(.)\\1"): selects variables that match a regular expression. This one matches any variables that contain repeated characters. You’ll learn more about regular expressions in [strings](https://r4ds.had.co.nz/strings.html#strings).
* num\_range("x", 1:3): matches x1, x2 and x3.

If using select to rename values; all the values that aren’t mentioned are dropped so it’s better to use rename(), which is a variant of select() that keeps all the variables that aren’t explicitly mentioned

**rename**(flights, tail\_num = tailnum)

Another option is to use select() in conjunction with the everything() helper. This is useful if you have a handful of variables you’d like to move to the start of the data frame.

**MUTATE**

add new columns that are functions of existing columns mutate()

mutate() always adds new columns at the end of your dataset so we’ll start by creating a **narrower dataset** so we can see the new variables

If you only want to keep the new variables, use transmute()

* Arithmetic operators: +, -, \*, /, ^. These are all vectorised This is most useful when one of the arguments is a single number: air\_time / 60, hours \* 60 + minute, etc.

Arithmetic operators are also useful in conjunction with the aggregate functions. For example, x / sum(x) calculates the proportion of a total, and y - mean(y) computes the difference from the mean.

* Modular arithmetic: %/% (integer division) and %% (remainder), where x == y \* (x %/% y) + (x %% y). Modular arithmetic is a handy tool because it allows you to break integers up into pieces. For example, in the flights dataset, you can compute hour and minute from dep\_time with:

**transmute**(flights,

dep\_time,

hour = dep\_time %/% 100,

minute = dep\_time %% 100

**Logs**: log(), log2(), log10()

**Offsets**: lead() and lag() allow you to refer to leading or lagging values

This allows you to compute running differences (e.g. x - lag(x)) or find when values change (x != lag(x))

**Cumulative and rolling aggregates**: R provides functions for running sums, products, mins and maxes: cumsum(), cumprod(), cummin(), cummax(); and dplyr provides cummean() for cumulative means.

**Logical comparisons**, <, <=, >, >=, !=, and ==, which you learned about earlier. If you’re doing a complex sequence of logical operations it’s often a good idea to store the interim values in new variables

**Ranking**: there are a number of ranking functions, but you should start with min\_rank()

use desc(x) to give the largest values the smallest ranks

**SUMMARISE**

The last key verb is summarise(). It collapses a data frame to a single row:

**summarise**(flights, delay = **mean**(dep\_delay, na.rm = TRUE))

summarise() is not terribly useful unless we pair it with group\_by().

**PIPE**

**Instead of**

by\_dest <- **group\_by**(flights, dest)

delay <- **summarise**(by\_dest,

count = **n**(),

dist = **mean**(distance, na.rm = TRUE),

delay = **mean**(arr\_delay, na.rm = TRUE)

)

delay <- **filter**(delay, count > 20, dest != "HNL")

**use**

delays <- flights %>%

**group\_by**(dest) %>%

**summarise**(

count = **n**(),

dist = **mean**(distance, na.rm = TRUE),

delay = **mean**(arr\_delay, na.rm = TRUE)

) %>%

**filter**(count > 20, dest != "HNL")

This focuses on the transformations, not what’s being transformed, which makes the code easier to read. You can read it as a series of imperative statements: group, then summarise, then filter. As suggested by this reading, a good way to pronounce %>% when reading code is **“then”.**

**MISSING VALUES**

 if there’s any missing value in the input, the output will be a missing value. Fortunately, all aggregation functions have an na.rm argument which removes the missing values prior to computation

**removing the cancelled flights**

not\_cancelled <- flights %>%

**filter**(!**is.na**(dep\_delay), !**is.na**(arr\_delay))

not\_cancelled %>%

**group\_by**(year, month, day) %>%

**summarise**(mean = **mean**(dep\_delay))

**COUNTS**

count (n()), or a count of non-missing values (sum(!is.na(x)))

**SUMMARY FUNCTIONS**

* **Measures of location**: we’ve used mean(x), but median(x)

not\_cancelled %>%

**group\_by**(year, month, day) %>%

**summarise**(

avg\_delay1 = **mean**(arr\_delay),

avg\_delay2 = **mean**(arr\_delay[arr\_delay > 0]) *# the average positive delay*

* **Measures of spread**: sd(x), IQR(x), mad(x). The root mean squared deviation, or standard deviation sd(x), is the standard measure of spread. The interquartile range IQR(x) and median absolute deviation mad(x) are robust equivalents that may be more useful if you have outliers.

not\_cancelled %>%

**group\_by**(dest) %>%

**summarise**(distance\_sd = **sd**(distance)) %>%

**arrange**(**desc**(distance\_sd))

* **Measures of rank**: min(x), quantile(x, 0.25), max(x). Quantiles are a generalisation of the median

not\_cancelled %>%

**group\_by**(year, month, day) %>%

**summarise**(

first = **min**(dep\_time),

last = **max**(dep\_time)

)

* **Measures of position**: first(x), nth(x, 2), last(x). These work similarly to x[1], x[2], and x[length(x)] but let you set a default value if that position does not exist (i.e. you’re trying to get the 3rd element from a group that only has two elements).
* **Counts**: You’ve seen n(), which takes no arguments, and returns the size of the current group. To count the number of non-missing values, use sum(!is.na(x)). To count the number of distinct (unique) values, use n\_distinct(x)

*# Which destinations have the most carriers?*

not\_cancelled %>%

**group\_by**(dest) %>%

**summarise**(carriers = **n\_distinct**(carrier)) %>%

**arrange**(**desc**(carriers))

* **Counts and proportions of logical values**: sum(x > 10), mean(y == 0). When used with numeric functions, TRUE is converted to 1 and FALSE to 0. This makes sum() and mean() very useful: sum(x) gives the number of TRUEs in x, and mean(x) gives the proportion

*# How many flights left before 5am? (these usually indicate delayed*

*# flights from the previous day)*

not\_cancelled %>%

**group\_by**(year, month, day) %>%

**summarise**(n\_early = **sum**(dep\_time < 500))

**GROUPING BY MULT VARIABLES**

When you group by multiple variables, each summary peels off one level of the grouping.

daily <- **group\_by**(flights, year, month, day)

**UNGROUP**

If you need to remove grouping, and return to operations on ungrouped data, use ungroup().

daily %>%

**ungroup**() %>% *# no longer grouped by date*

**summarise**(flights = **n**()) *# all flights*

**GROUPED MUTATES**

Grouping is most useful in conjunction with summarise(), but you can also do convenient operations with mutate() and filter()

The worst of the group

flights\_sml %>%

**group\_by**(year, month, day) %>%

**filter**(**rank**(**desc**(arr\_delay)) < 10)

**all groups bigger than**

popular\_dests <- flights %>%

**group\_by**(dest) %>%

**filter**(**n**() > 365)

popular\_dests

A grouped filter is a grouped mutate followed by an ungrouped filter