```
library(dplyr)

rladies_global %>%
  filter(city == 'Boston')
```

dplyr 0.7.0 - tidyeval / programming with dplyr

https://rladies.github.io/Boston

The tidyverse is a collection of R packages that share common philosophies and are designed to work together.

(and it includes dplyr)

There are three interrelated rules which make a dataset tidy:

1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.

Figure 12.1 shows the rules visually.

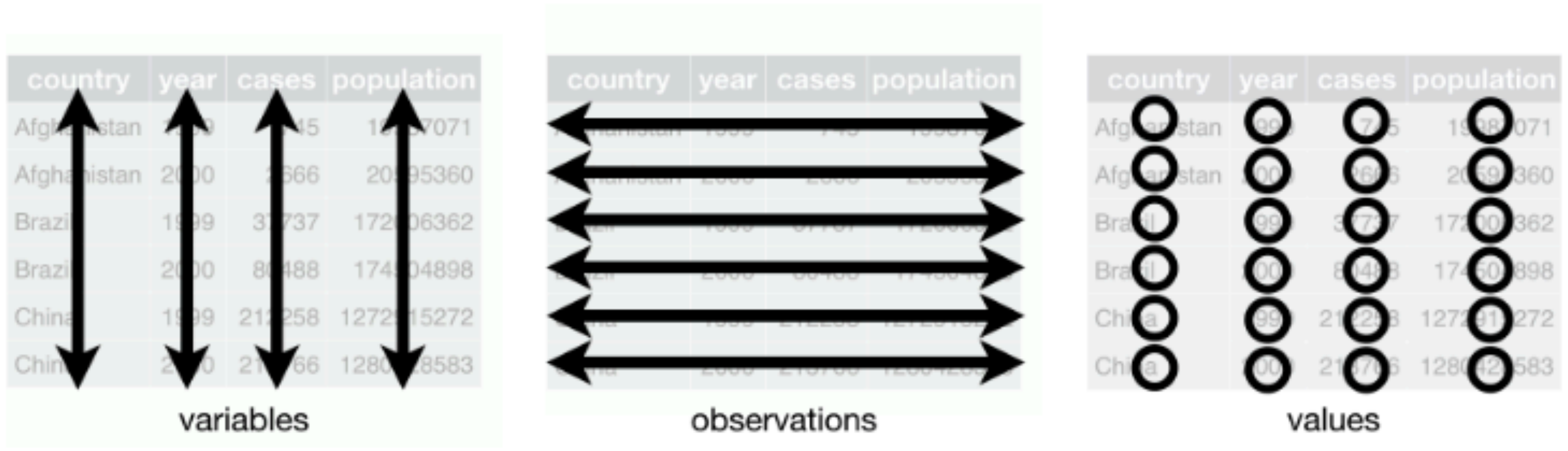

Figure 12.1: Following three rules makes a dataset tidy: variables are in columns, observations are in rows, and values are in cells.

These three rules are interrelated because it's impossible to only satisfy two of the three.

Lifted wholesale from http://r4ds.had.co.nz/tidy-data.html

dplyr verbs

- Pick observations by their values
  - `filter(cats, color == “multi”)`
- Reorder the rows
  - `arrange(cats, desc(weight))`
- Pick variables by their names
  - `select(cats, color, weight, name)`
- Create new variables with functions of existing variables
  - `mutate(cats, weight_kg = weight / 2.5)`
- Collapse many values down to a single summary
  - Use with group_by()
  - `summarise(cats, mean_weight = mean(weight))`

dplyr verbs

Pick observations by their values

```
filter(cats, color == “multi”)
```

```
cats[cats$color == “multi”,]
```

pipes

String together multiple “verbs” with pipes

- A pipe looks like this: %>%
- Verbs have implied first argument

```
filter(cats, color == “multi”)
```

equivalent to

```
cats %>% filter(., color == “multi”)
```

equivalent to

```
cats %>% filter(color == “multi”)
```

pipes

```
cats %>%
filter(color == “multi”) %>%
select(color, weight, sex) %>%
mutate(weight_kg = weight / 2.5) %>%
group_by(sex) %>%
summarize(mean_wt_kg = mean(weight_kg)) %>%
arrange(mean_wt_kg)
```

independent study

There is a LOT more to tidyverse and to dplyr

- http://www.datacarpentry.org/R-ecology-lesson/03-dplyr.html
- http://r4ds.had.co.nz/
- http://tidyverse.org/
- https://www.rstudio.com/resources/cheatsheets/

dplyr 0.7.0 - tidyeval / programming with dplyr

setup

- Install dplyr or tidyverse

```
    install.packages(“dplyr”)
```

- Install gapminder data package

```
    install.packages(“gapminder”)
```

- Load dplyr & gapminder

```
    library(dplyr)
    library(gapminder)
    data(gapminder)
```

- Check dplyr install

```
    packageVersion(“dplyr”)
```

tidyeval

“The biggest change is a new system for programming with dplyr, called **tidy evaluation**, or tidy eval for short. Tidy eval is a system for capturing expressions and later evaluating them in the correct context.”

EOHW

tidyeval

Core concept: “quoting”

1) Prepare the input
2) Tell dplyr you’ve prepared the input

part 1: quo(), !! and UQ()

```
my_var <- quo(continent)
# not my_var <- “continent”

gapminder %>%
     group_by(!!my_var) %>%
     summarise_at(vars(gdpPercap), mean)

gapminder %>% filter(UQ(my_var) == "Asia”)
# not filter(!!my_var == "Asia”)
```

part 1’: quo(), !!! and UQS()

```
library(stringr)
start <- quo(1)
end <- quo(4)
args <- list(start = start, end = end)

# str_sub subsets a string by given indices
gapminder %>%
mutate(ShortName = str_sub(country, !!!args))
```

part 2: setting variable names

```
# Use strings or quo() derivatives
str_name <- “neato_gdp”
name <- quo_name(quo(gdp))
custom <- paste0(“neato_”, name)


gapminder %>%
     mutate(!!custom := pop * gdpPercap)

gapminder %>%
     mutate(!!str_name := pop * gdpPercap)
```

part 3: enquo()

```
# write functions that take barewords
# as parameters

make_real_log <- function(df, incol){
     incol <- enquo(incol)
     name <- paste0(quo_name(incol),
                "_log")
     df %>%
          mutate(!!name := log2(1 + !!
incol))
}

# prevents log2(0) becoming -Inf
gapminder %>% make_real_log(pop)
```

part 4: the .data pronoun

```
# use strings instead of quo()
my_var <- “continent”

gapminder %>%
     group_by(.data[[my_var]]) %>%
     summarise_at(vars(gdpPercap), mean)

# prevent silent failure / R CMD check error
mutate_y <- function(df) {
  mutate(df, y = .data$a + .data$x)
}
```

part 5: tidbits

```
# pull() returns a vector
gapminder %>%
     pull(country) %>%
     unique()

# “_if” verb suffix
gapminder %>% summarise_if(is.numeric, mean)
gapminder %>% mutate_if(is.numeric, mean)

# and tidyeval is coming to ggplot2 & tidyr!
```

the problem (*my* problem) – user inputs

```
user_input = “country”

gapminder %>% group_by(.data[[user_input]])
%>% summarise(new = mean(lifeExp))

gapminder %>% group_by(!!
user_input := .data[[user_input]]) %>%
summarise(new = mean(lifeExp)) #2991

library(rlang)
gapminder %>% group_by(!!sym(user_input)) %>
% summarise(new = mean(lifeExp))
```

Discussion