These materials adapted by Amelia McNamara from the RStudio CC BY-SA materials Introduction to R (2014) and Master the Tidyverse (2017).

Introduction to R & RStudio: deck 05: Tidy data, tidy tools

Amelia McNamara
Visiting Assistant Professor of Statistical and Data Sciences
Smith College

January 2018

%>%

Steps

```
boys_2015 <- filter(babynames, year == 2015, sex == "M")
boys_2015 <- select(boys_2015, name, n)
boys_2015 <- arrange(boys_2015, desc(n))
boys_2015
```

1. Filter babynames to just boys born in 2015
2. Select the name and n columns from the result
3. Arrange those columns so that the most popular names appear near the top.

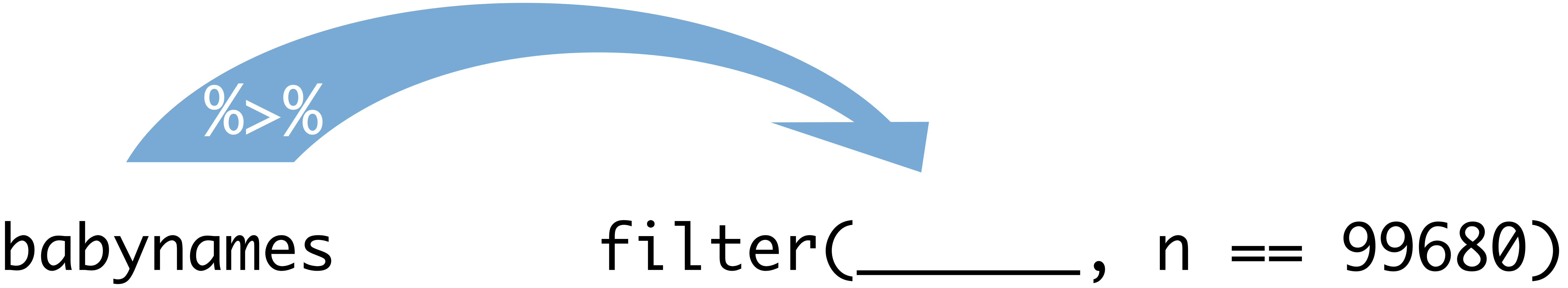
Steps

```
boys_2015 <- filter(babynames, year == 2015, sex == "M")
boys_2015 <- select(boys_2015, name, n)
boys_2015 <- arrange(boys_2015, desc(n))
boys_2015
```

Steps

```
arrange(select(filter(babynames, year == 2015,
  sex == "M"), name, n), desc(n))
```

The pipe operator %>%



babynames   filter(______, n == 99680)

Passes result on left into first argument of function on right. So, for example, these do the same thing. Try it.

```
filter(babynames, n == 99680)
babynames %>% filter(n == 99680)
```

Pipes

```
boys_2015 <- filter(babynames, year == 2015, sex == "M")
boys_2015 <- select(boys_2015, name, n)
boys_2015 <- arrange(boys_2015, desc(n))
boys_2015
```

```
babynames %>%
  filter(year == 2015, sex == "M") %>%
  select(name, n) %>%
  arrange(desc(n))
```

```
foo_foo <- little_bunny()
```

```
foo_foo %>%
  hop_through(forest) %>%
  scoop_up(field_mouse) %>%
  bop_on(head)
```

vs.

```
foo_foo2 <- hop_through(foo_foo, forest)
foo_foo3 <- scoop_up(foo_foo2, field_mouse)
bop_on(foo_foo3,  head)
```

Shortcut to type %>%

Cmd + Shift + M (Mac)

Ctrl + Shift + M (Windows)

Your Turn 6

Use **%>%** to write a sequence of functions that:

1. Filter babynames to just the girls that were born in 2015
2. Select the **name** and **n** columns
3. Arrange the results so that the most popular names are near the top.

05:00

```
babynames %>%
  filter(year == 2015, sex == "F") %>%
  select(name, n) %>%
  arrange(desc(n))
#         name     n
#  1      Emma 20355
#  2    Olivia 19553
#  3    Sophia 17327
#  4       Ava 16286
#  5  Isabella 15504
#  6       Mia 14820
#  7   Abigail 12311
#  8     Emily 11727
#  9 Charlotte 11332
# 10    Harper 10241
# ... with 18,983 more rows
```

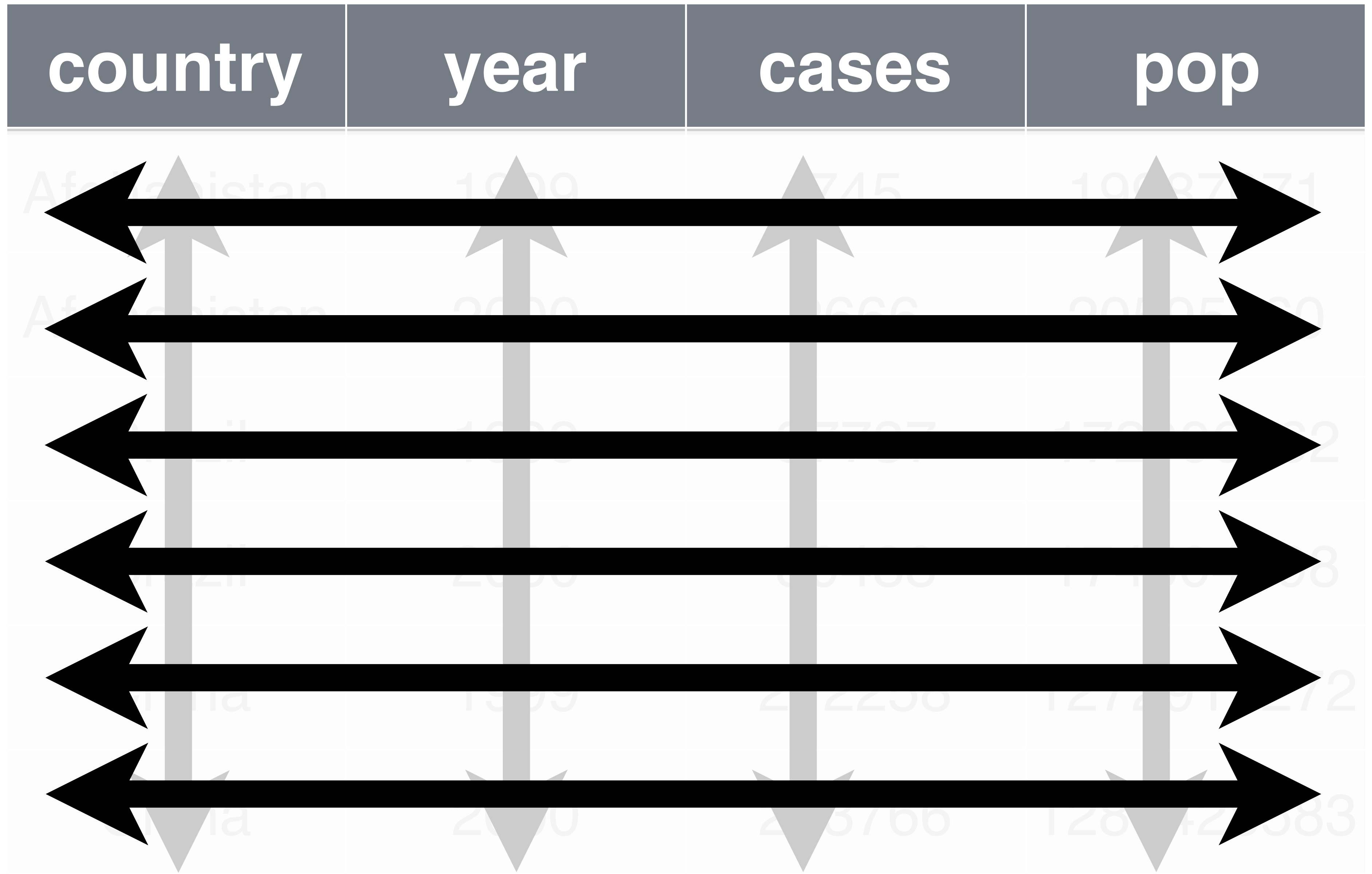
Tidy data

Tidy data

Tidy functions all expect and return the same data structure, known as **tidy data**:

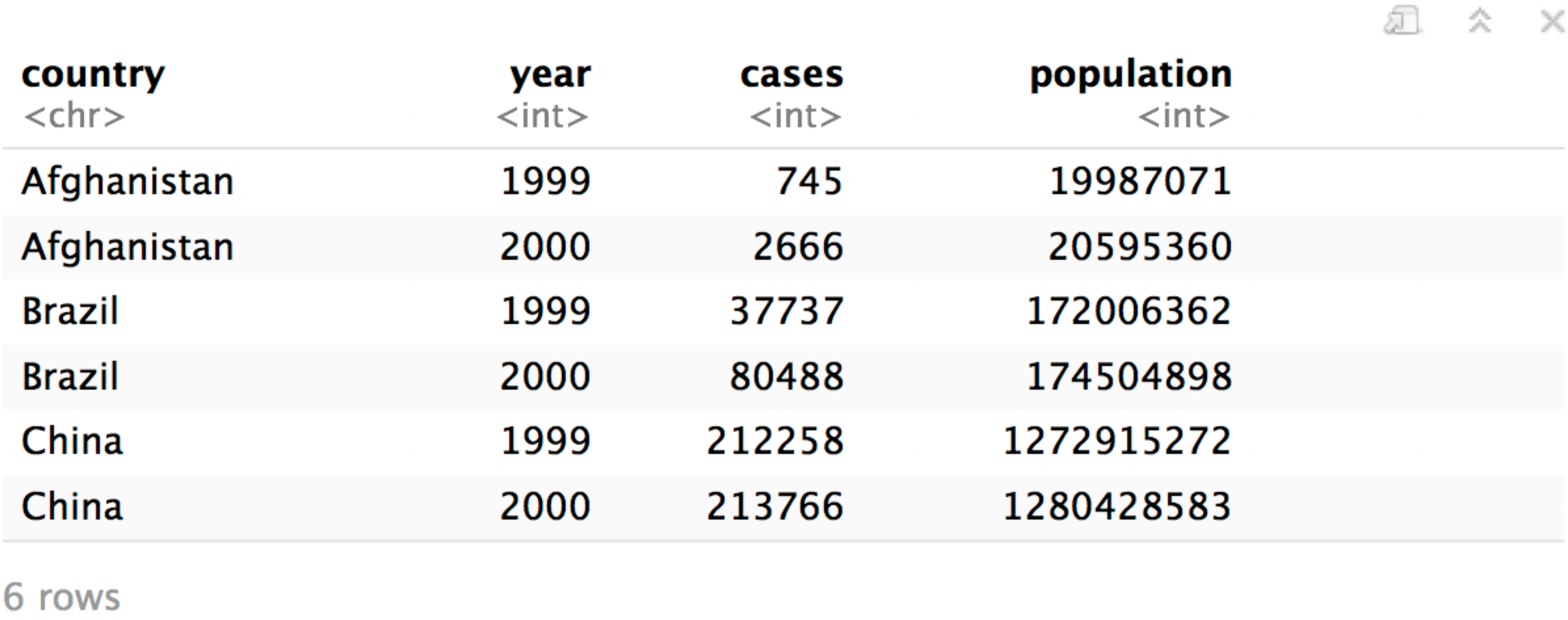
1. A **data frame** that contains
2. **variables** in the **columns** and
3. **cases** in the **rows**.

Tidy data



A data set is **tidy** iff:

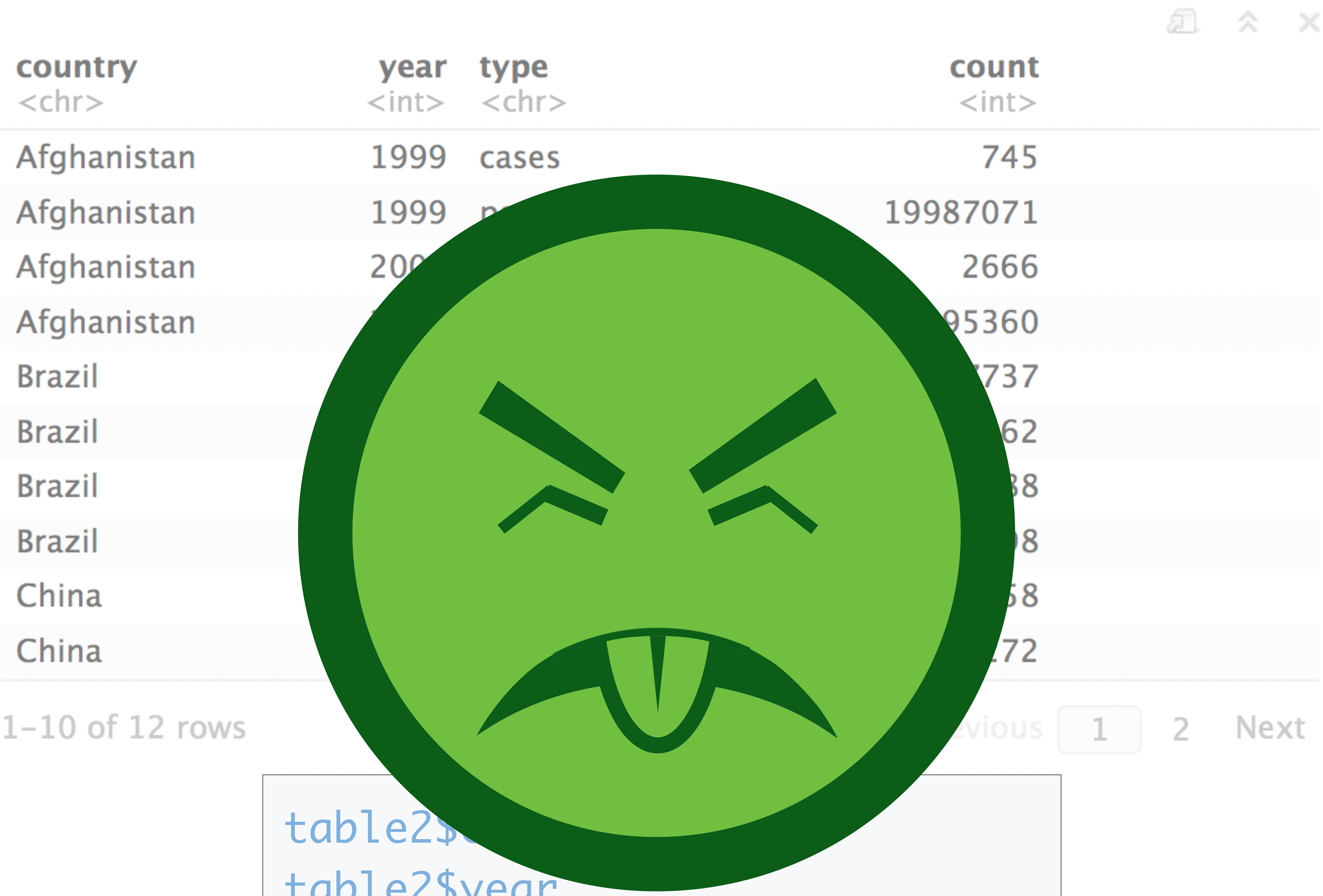
1. Each **variable** is in its own **column**
2. Each **case** is in its own **row**
3. Each **value** is in its own **cell**



country	year	cases	population
<chr>	<int>	<int>	<int>
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

6 rows

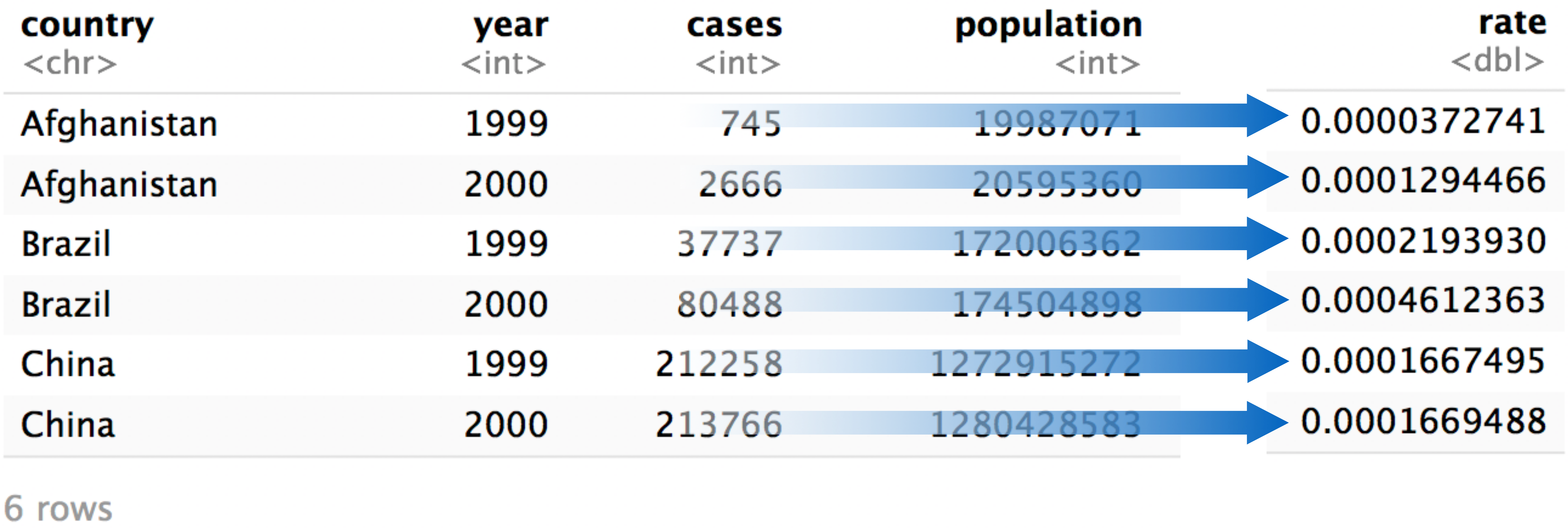
```
table1$country
table1$year
table1$cases
table1$population
```



country <chr>	year <int>	type <chr>	count <int>
Afghanistan	1999	cases	745
Afghanistan	1999	p	19987071
Afghanistan	200		2666
Afghanistan			95360
Brazil			737
Brazil			62
Brazil			88
Brazil			98
China			58
China			72

1–10 of 12 rows

```
table2$
table2$year
table2$count[c(1,3,5,7,9,11)]
table2$count[c(2,4,6,8,10,12)]
```



country <chr>	year <int>	cases <int>	population <int>	rate <dbl>
Afghanistan	1999	745	19987071	0.0000372741
Afghanistan	2000	2666	20595360	0.0001294466
Brazil	1999	37737	172006362	0.0002193930
Brazil	2000	80488	174504898	0.0004612363
China	1999	212258	1272915272	0.0001667495
China	2000	213766	1280428583	0.0001669488

6 rows

```
table1$cases / table1$population -> table1$rate
```

Tidy tools

Tidy tools

Functions are easiest to use when they are:

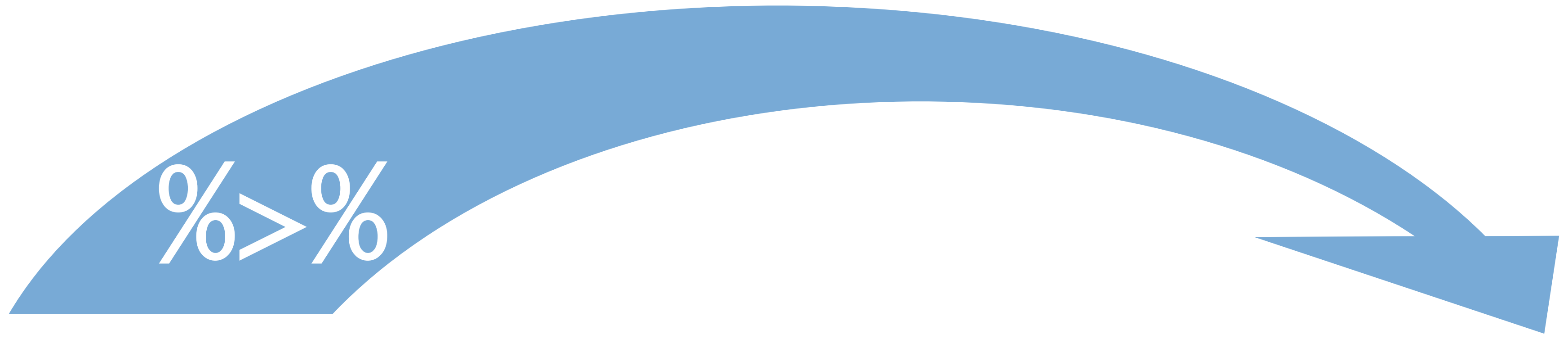
1. **Simple** - They do one thing, and they do it well
2. **Composable** - They can be combined with other functions for multi-step operations
3. **Smart** - They can use R objects as input.

Tidy functions do these things in a specific way.

1. **Simple** - They do one thing, and they do it well

filter() - extract **cases**
arrange() - reorder **cases**
group_by() - group **cases**
select() - extract **variables**
mutate() - create new **variables**
summarise() - summarise **variables** / create **cases**

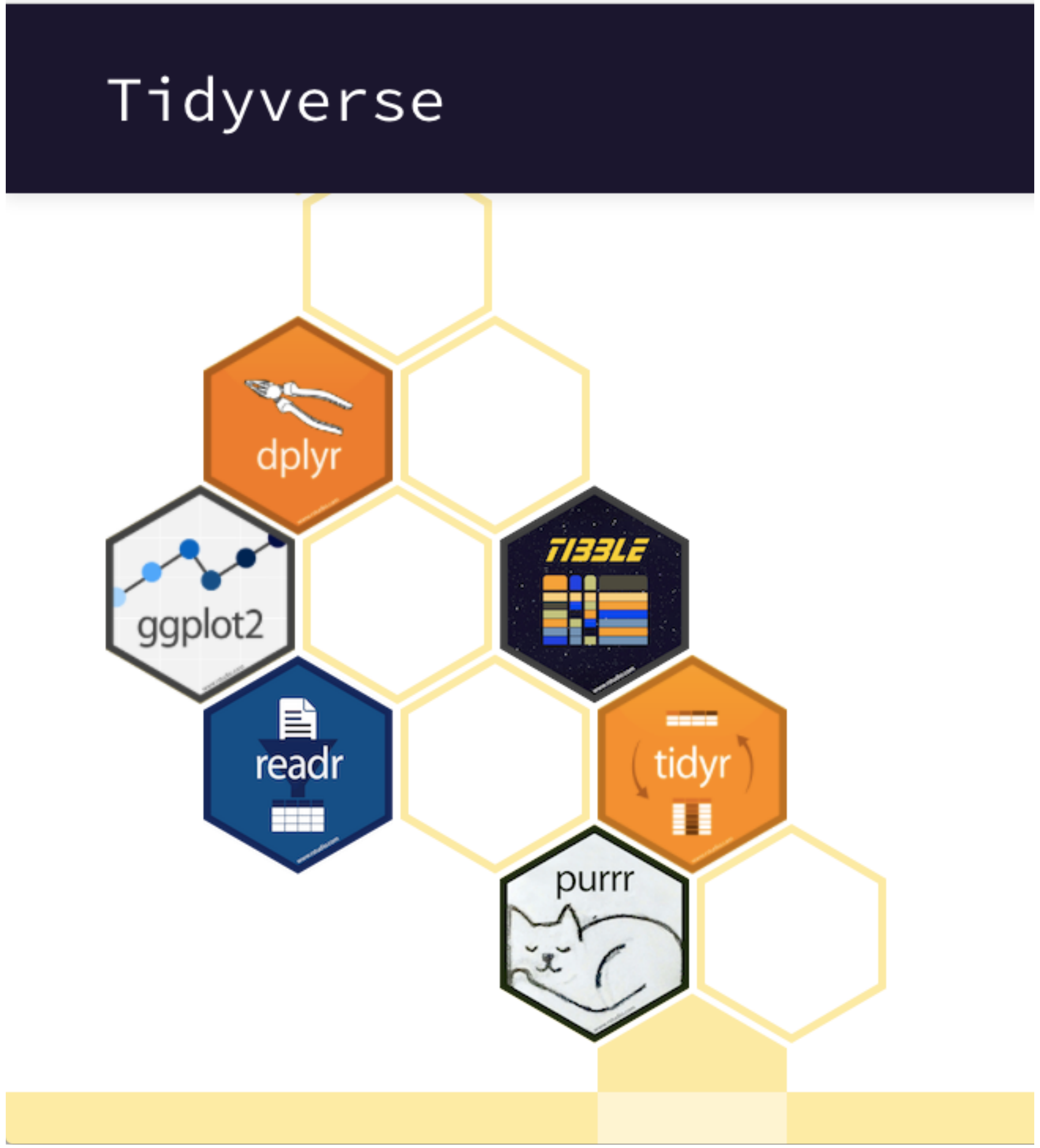
2. **Composable** - They can be combined with other functions for multi-step operations



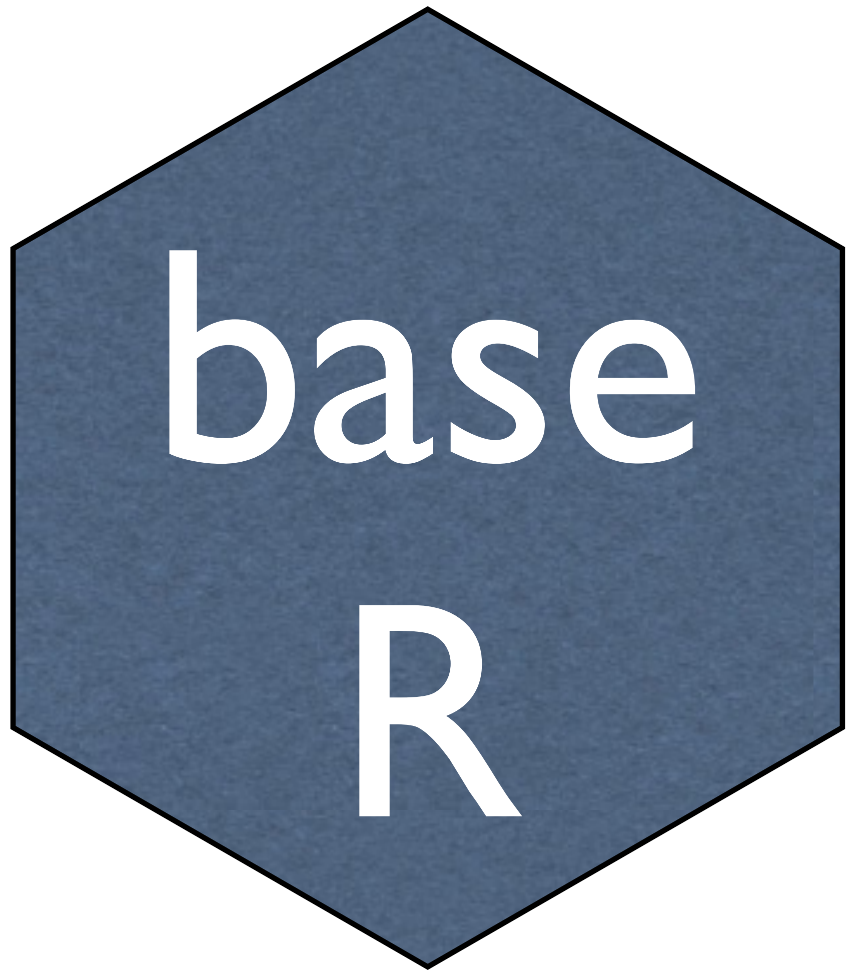
```
babynames      mutate(______, percent = prop * 100)
```

Each dplyr function takes a data frame as its first argument and returns a data frame. As a result, you can directly pipe the output of one function into the next.

It's not either/or



v.



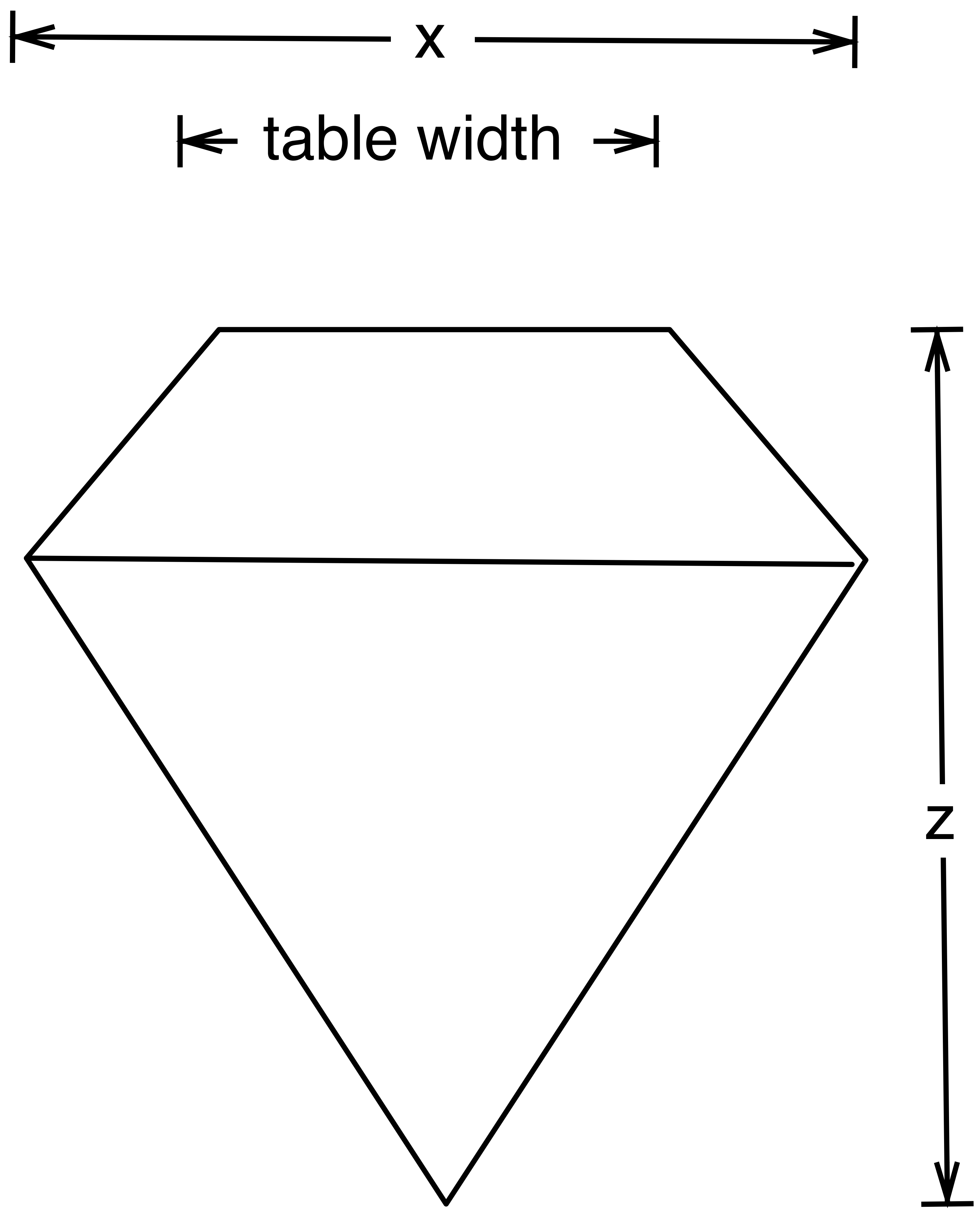
?

No! It's all R, it all works together. More like different philosophy

Diamonds

Diamonds data

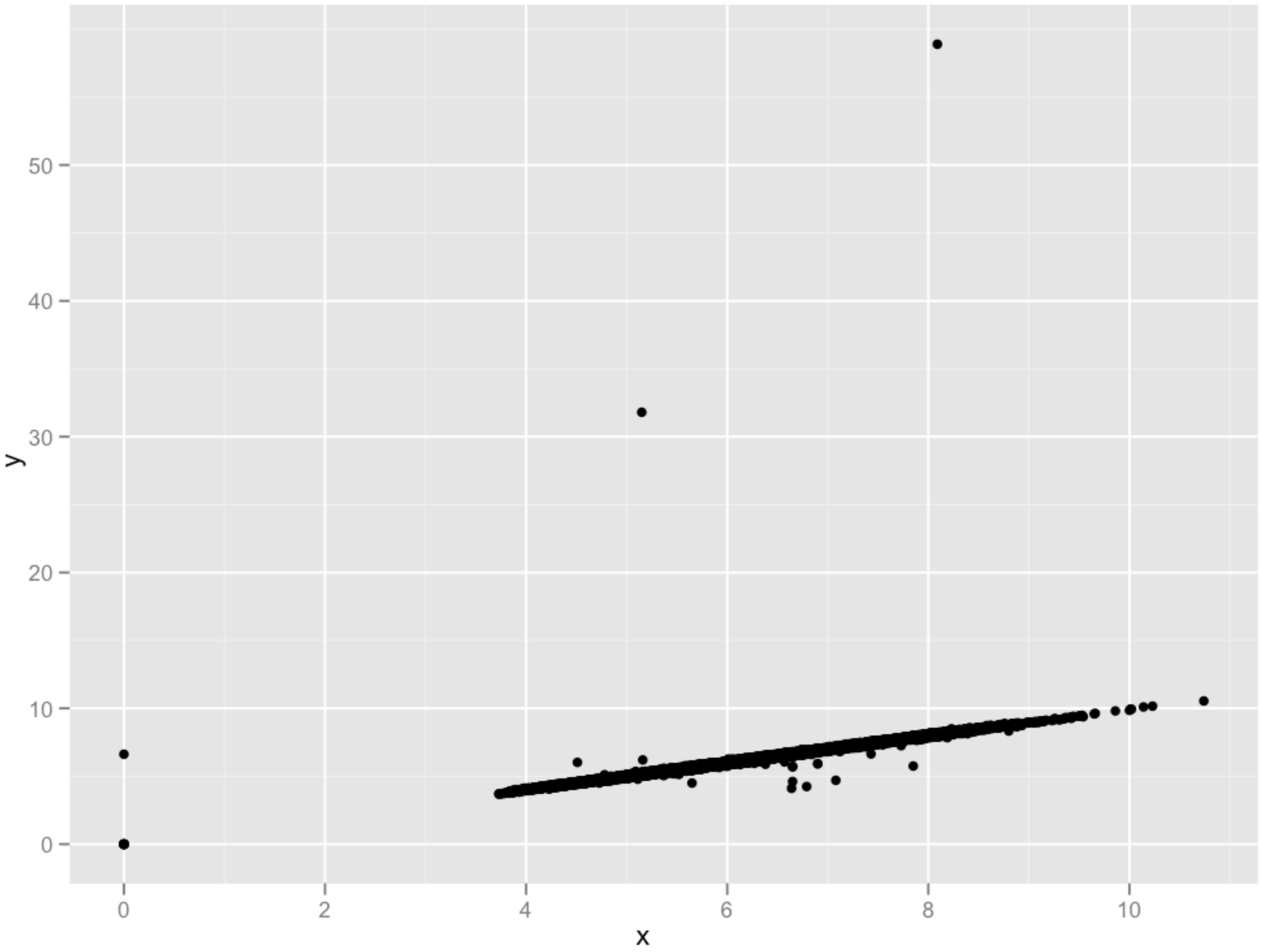
- ~**54,000** round diamonds from http://www.diamondse.info/
- comes in the ggplot2 package
- Carat, colour, clarity, cut
- Total depth, table, depth, width, height
- Price



depth = z / diameter

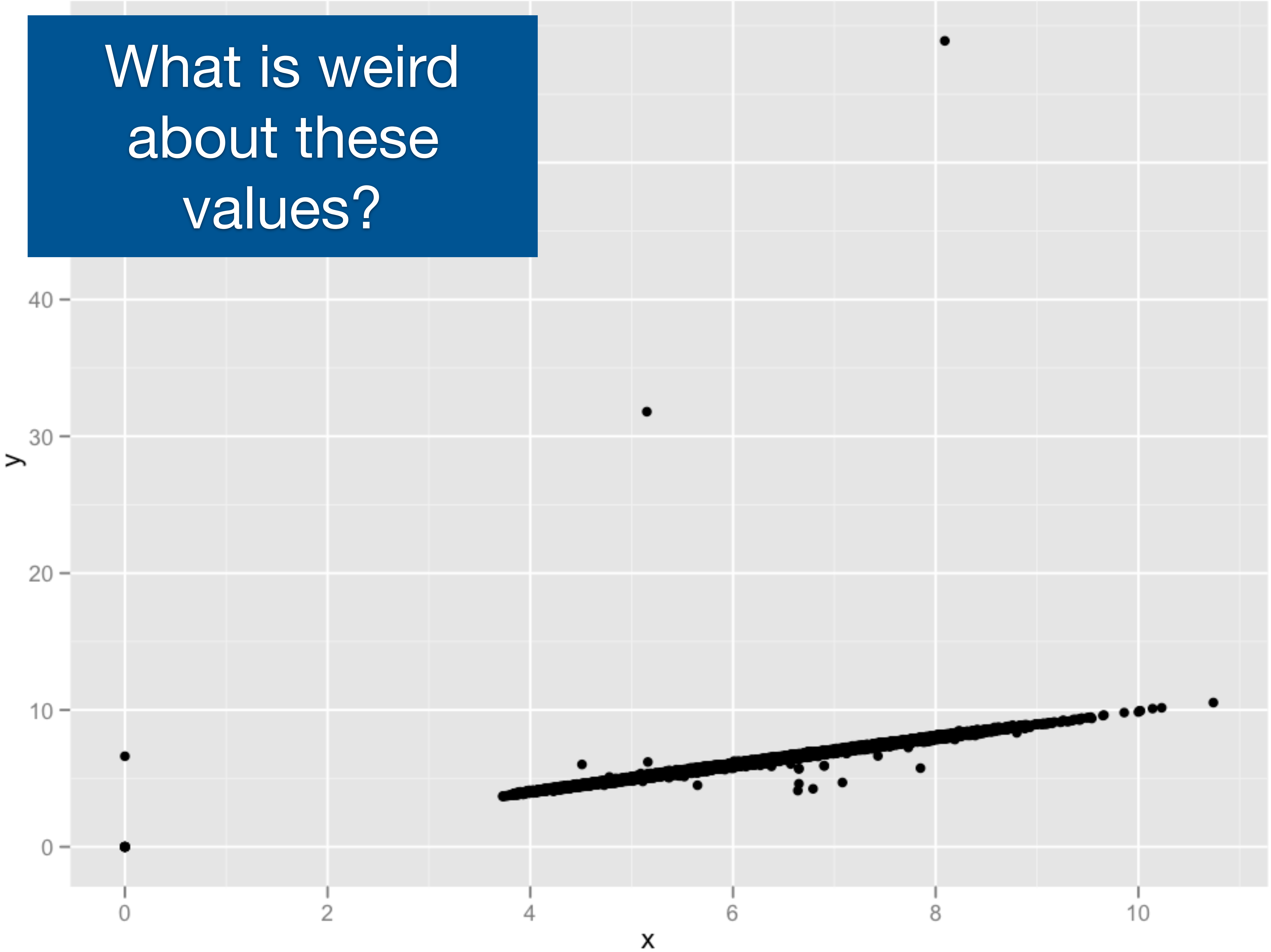
table = table width / x * 100

x, y, z in mm

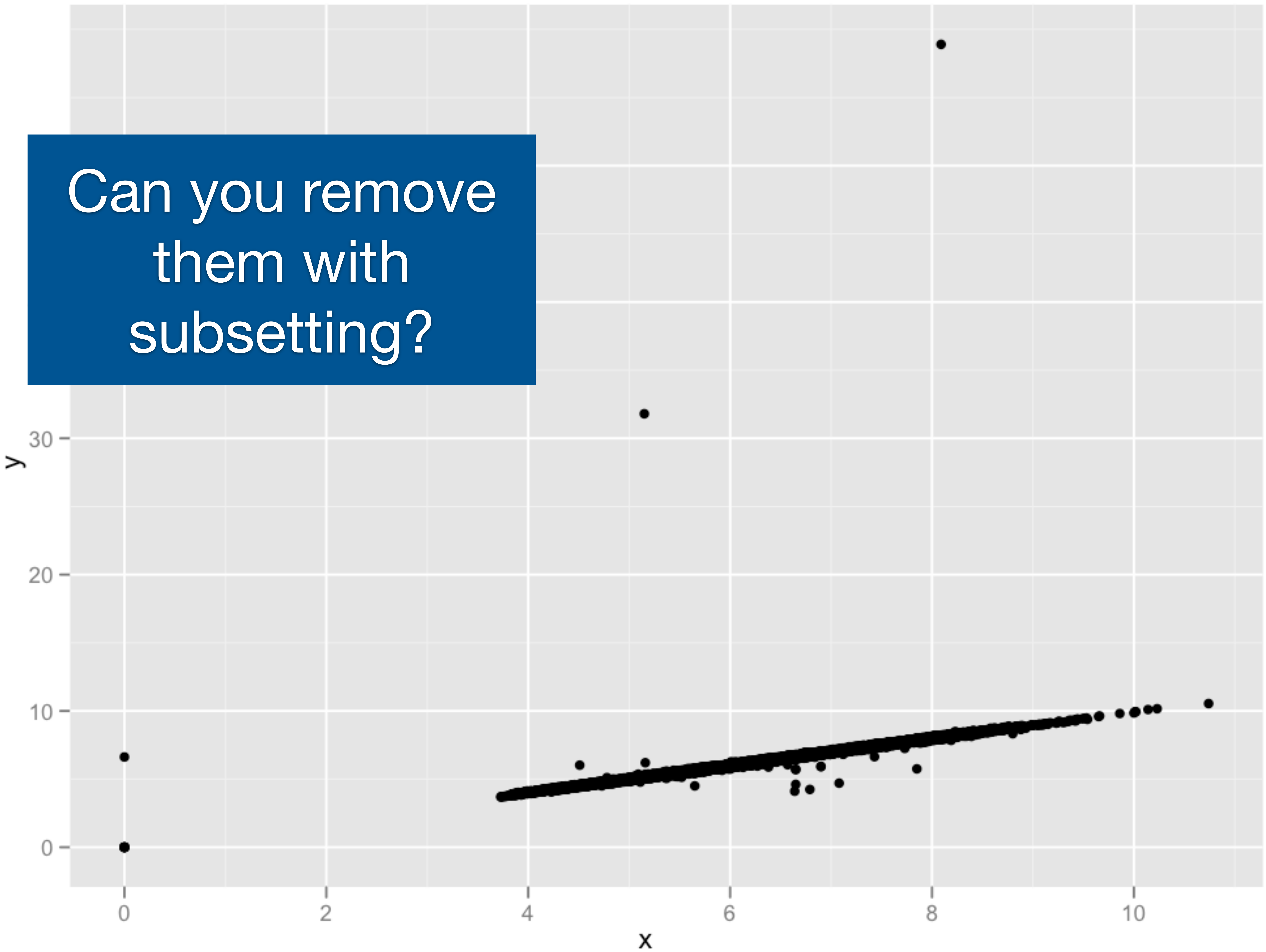


```
qplot(x, y, data = diamonds)
```

What is weird about these values?



```
qplot(x, y, data = diamonds)
```



Can you remove them with subsetting?

```
qplot(x, y, data = diamonds)
```