Introduction to the Tidyverse

Summer Institute in Data Science

Rolando J. Acosta

HARVARD
SCHOOL OF PUBLIC HEALTH

June 22, 2020

@RJANunez

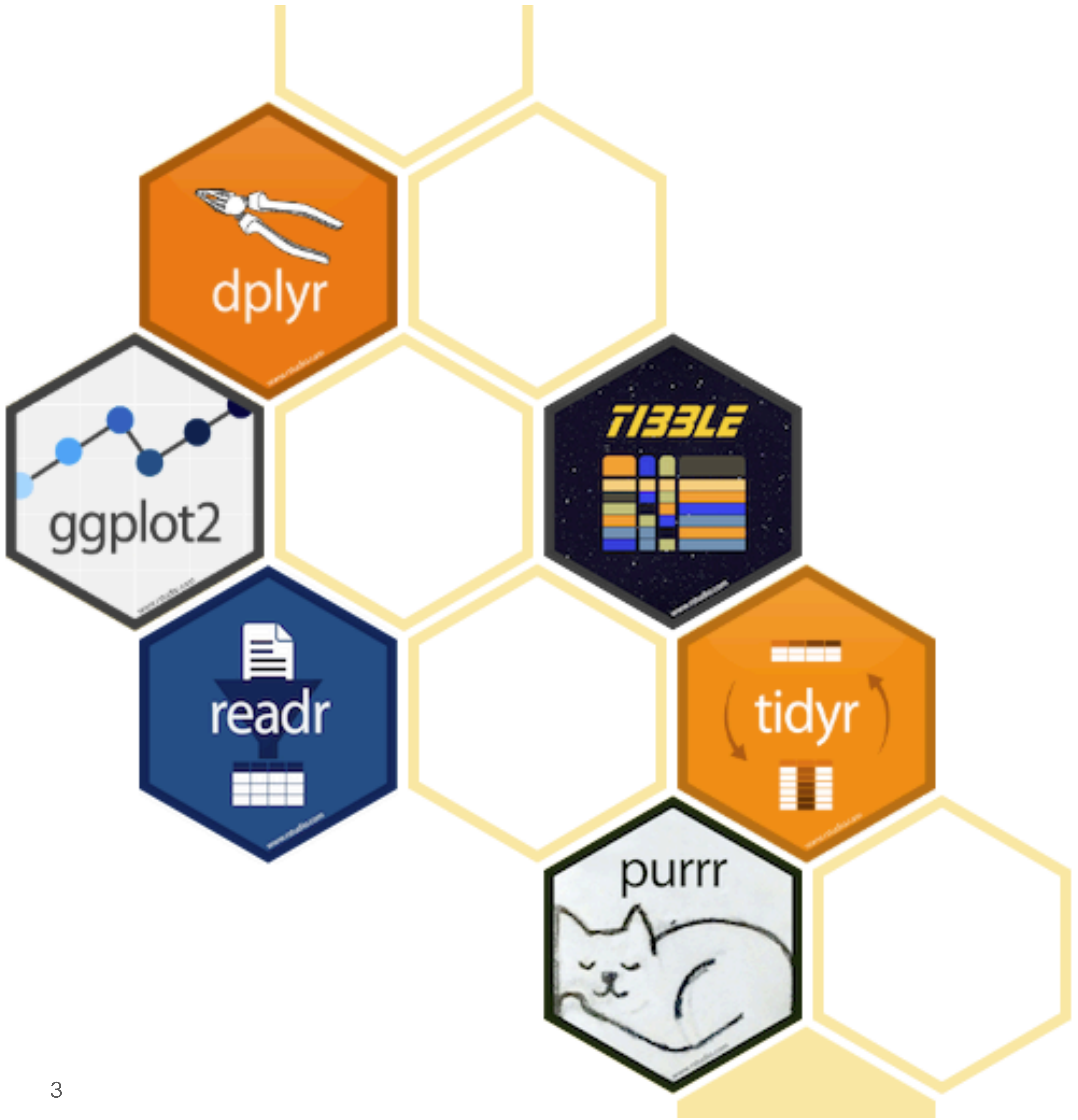
Outline

- What is tidyverse?
- Philosophy and design
- Tidy data
- The `dplyr` package

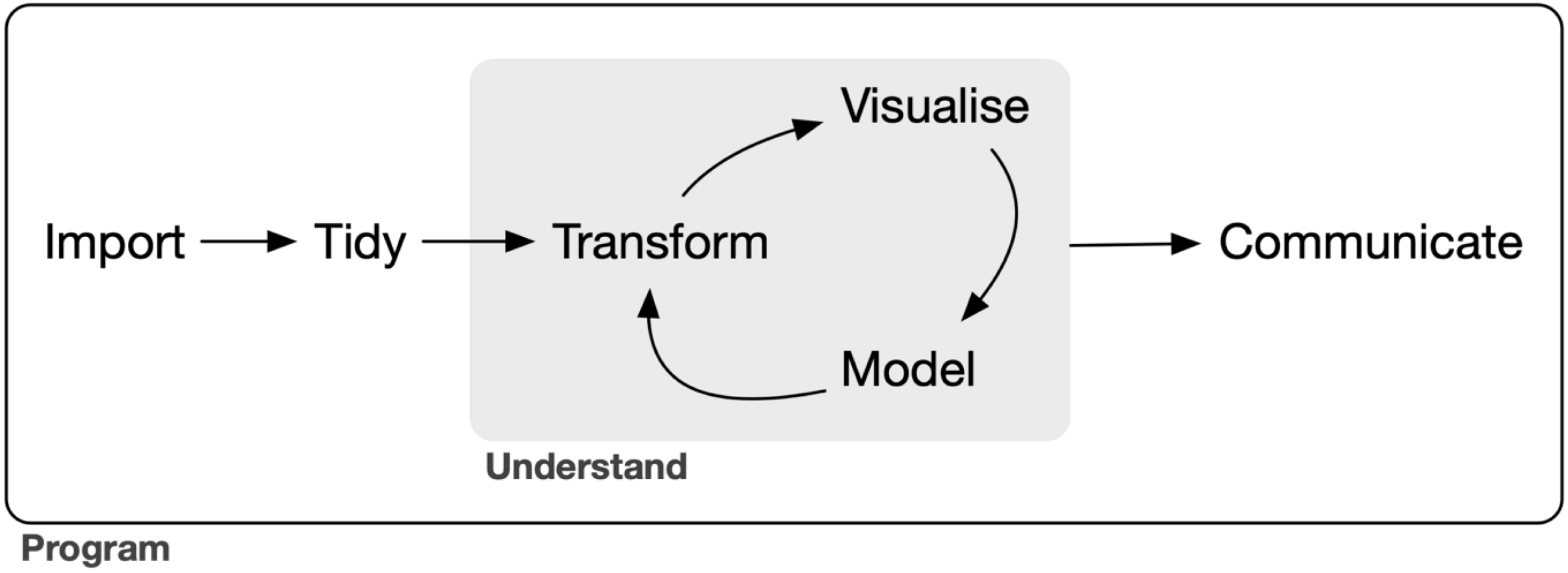
What is tidyverse?

"At a high level, the tidyverse is a language for solving data science challenges with R code. Its primary goal is to facilitate a conversation between a human and a computer about data. Less abstractly, the tidyverse is a collection of R packages that share a high-level design philosophy and low-level grammar and data structures, so that learning one package makes it easier to learn the next."

https://tidyverse.tidyverse.org/articles/paper.html



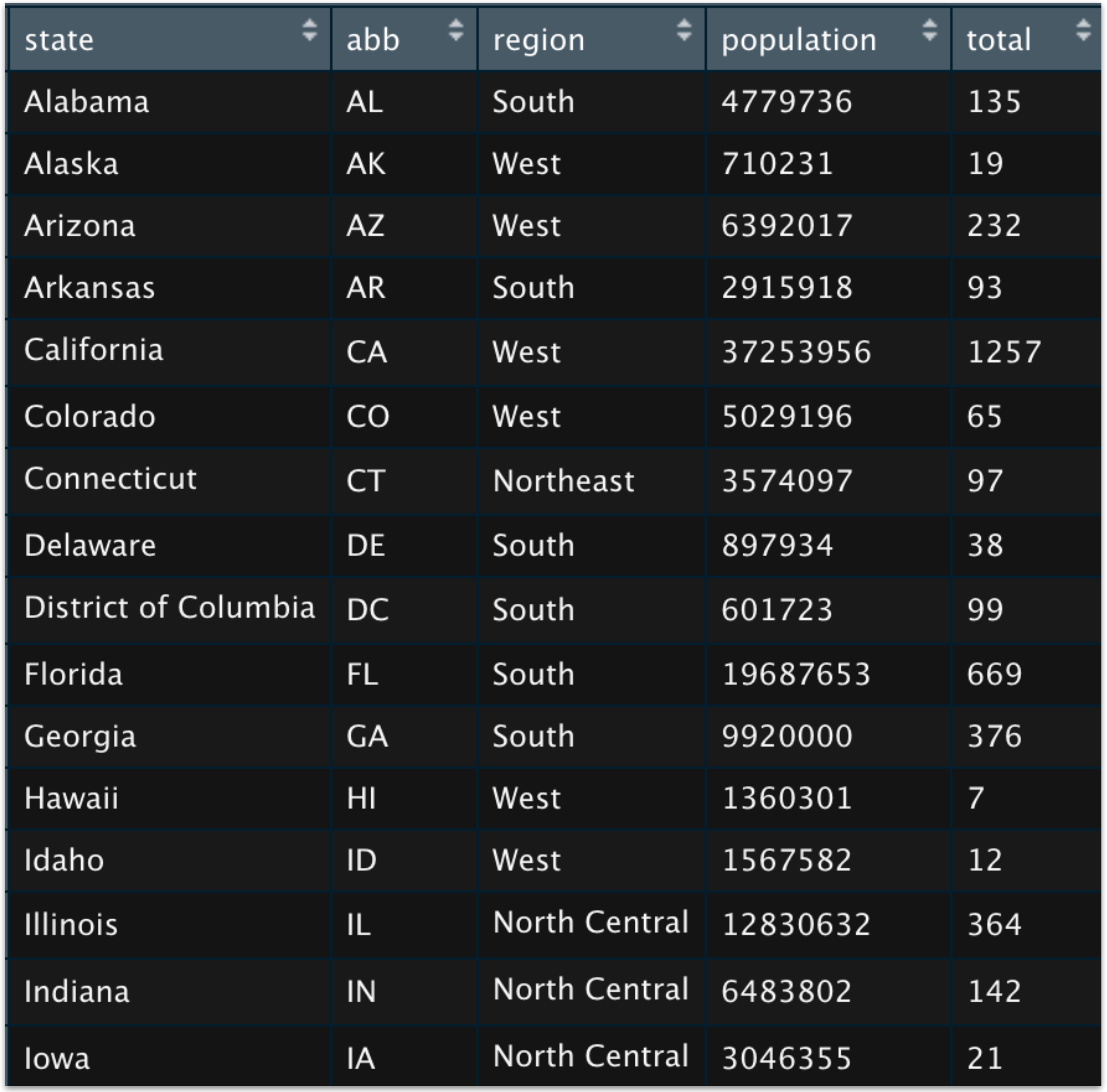
Philosophy and design



Tidy data: Ex 1

- A data frame is in ***tidy*** format if each row represents one observation and each column represents a different variable

```r
library(dslabs)
data(murders)
View(murders)
```



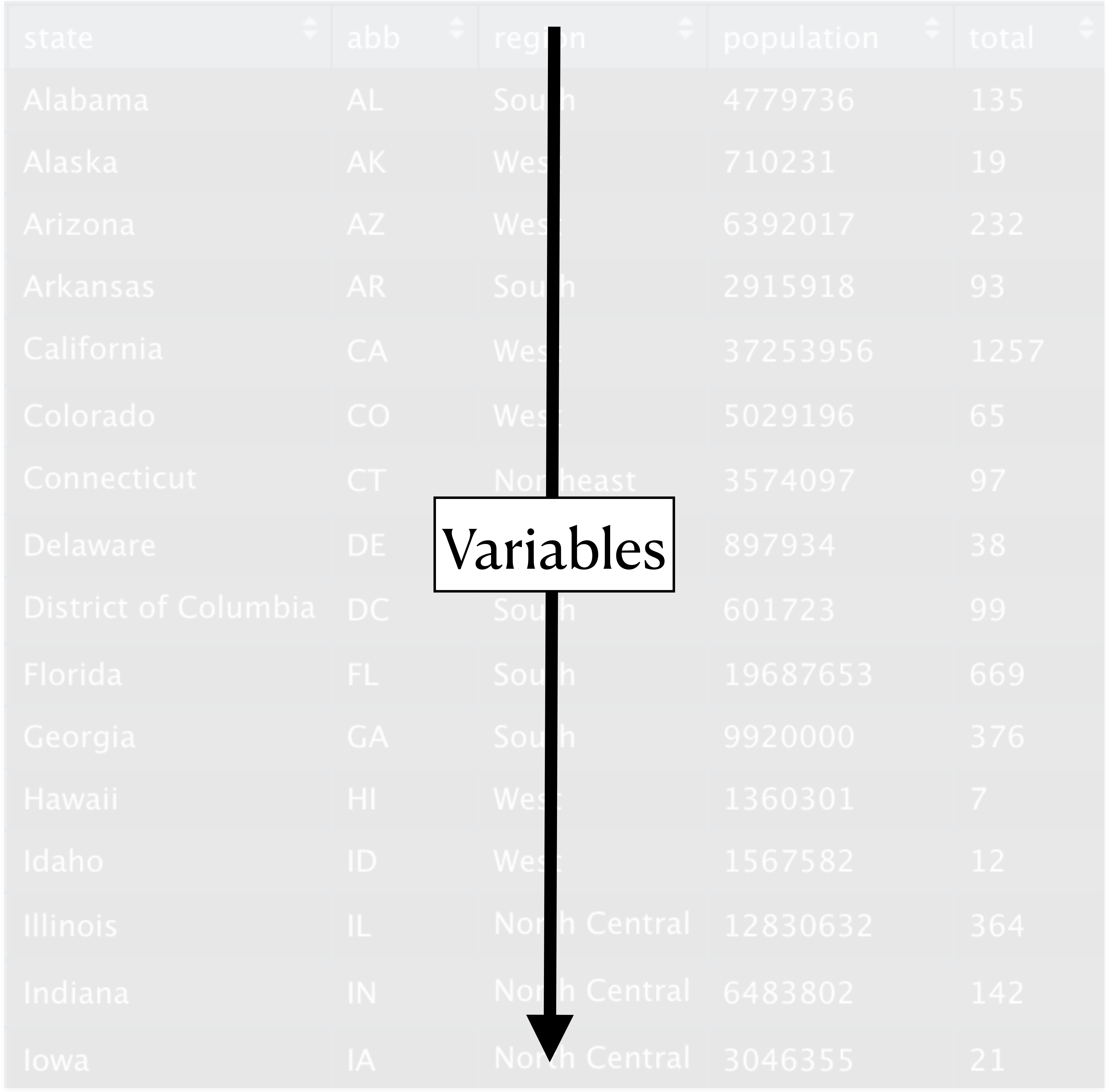
state	abb	region	population	total
Alabama	AL	South	4779736	135
Alaska	AK	West	710231	19
Arizona	AZ	West	6392017	232
Arkansas	AR	South	2915918	93
California	CA	West	37253956	1257
Colorado	CO	West	5029196	65
Connecticut	CT	Northeast	3574097	97
Delaware	DE	South	897934	38
District of Columbia	DC	South	601723	99
Florida	FL	South	19687653	669
Georgia	GA	South	9920000	376
Hawaii	HI	West	1360301	7
Idaho	ID	West	1567582	12
Illinois	IL	North Central	12830632	364
Indiana	IN	North Central	6483802	142
Iowa	IA	North Central	3046355	21

Example of tidy format

Tidy data: Ex 1

- A data frame is in ***tidy*** format if each row represents one observation and each column represents a different variable

```r
library(dslabs)
data(murders)
View(murders)
```



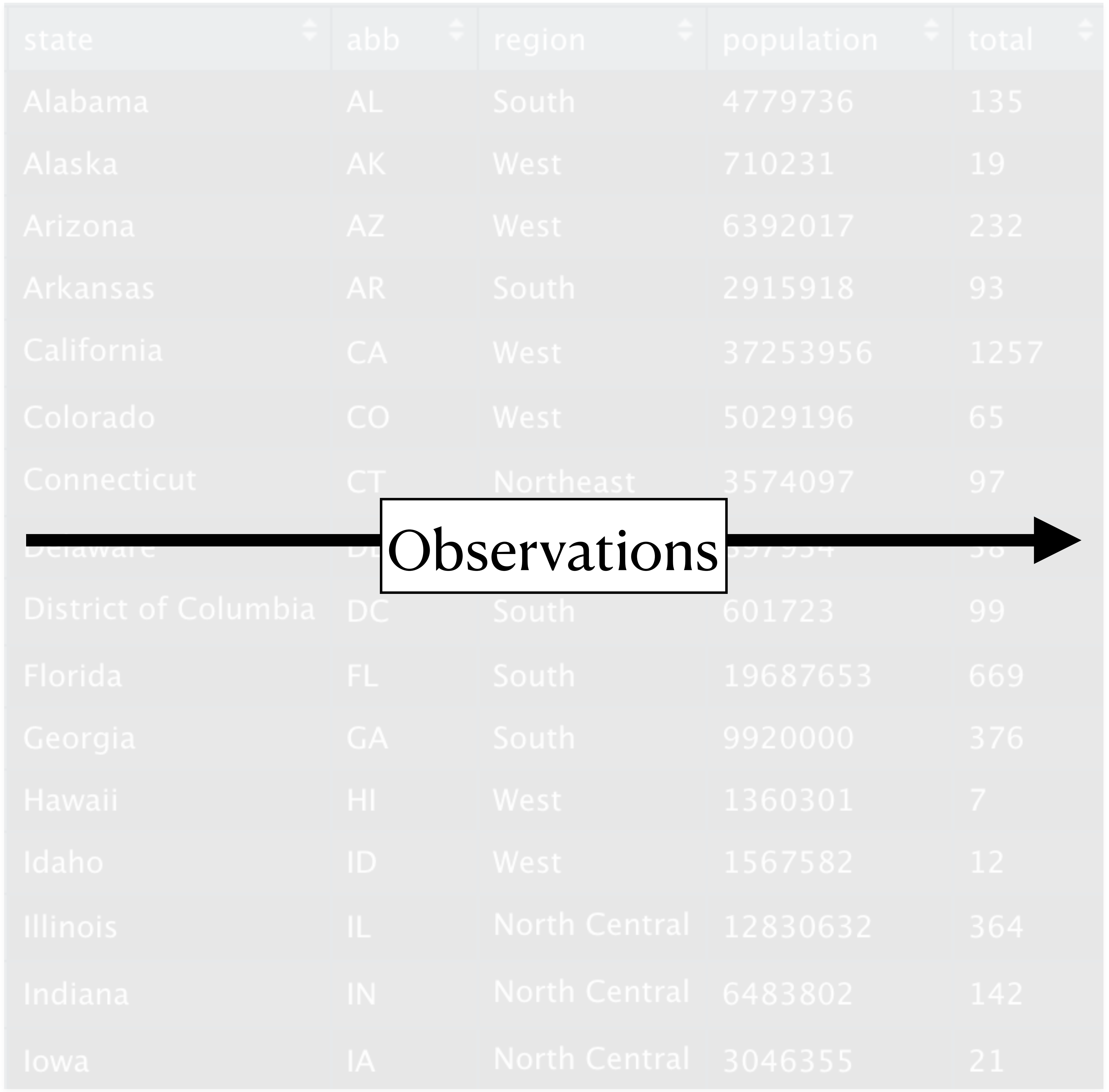
state	abb	region	population	total
Alabama	AL	South	4779736	135
Alaska	AK	West	710231	19
Arizona	AZ	West	6392017	232
Arkansas	AR	South	2915918	93
California	CA	West	37253956	1257
Colorado	CO	West	5029196	65
Connecticut	CT	Northeast	3574097	97
Delaware	DE	[illegible]	897934	38
District of Columbia	DC	South	601723	99
Florida	FL	South	19687653	669
Georgia	GA	South	9920000	376
Hawaii	HI	West	1360301	7
Idaho	ID	West	1567582	12
Illinois	IL	North Central	12830632	364
Indiana	IN	North Central	6483802	142
Iowa	IA	North Central	3046355	21

Example of tidy format

Tidy data: Ex 1

- A data frame is in ***tidy*** format if each row represents one observation and each column represents a different variable

```r
library(dslabs)
data(murders)
View(murders)
```



state	abb	region	population	total
Alabama	AL	South	4779736	135
Alaska	AK	West	710231	19
Arizona	AZ	West	6392017	232
Arkansas	AR	South	2915918	93
California	CA	West	37253956	1257
Colorado	CO	West	5029196	65
Connecticut	CT	Northeast	3574097	97
Delaware	DE	[illegible]	897934	38
District of Columbia	DC	South	601723	99
Florida	FL	South	19687653	669
Georgia	GA	South	9920000	376
Hawaii	HI	West	1360301	7
Idaho	ID	West	1567582	12
Illinois	IL	North Central	12830632	364
Indiana	IN	North Central	6483802	142
Iowa	IA	North Central	3046355	21

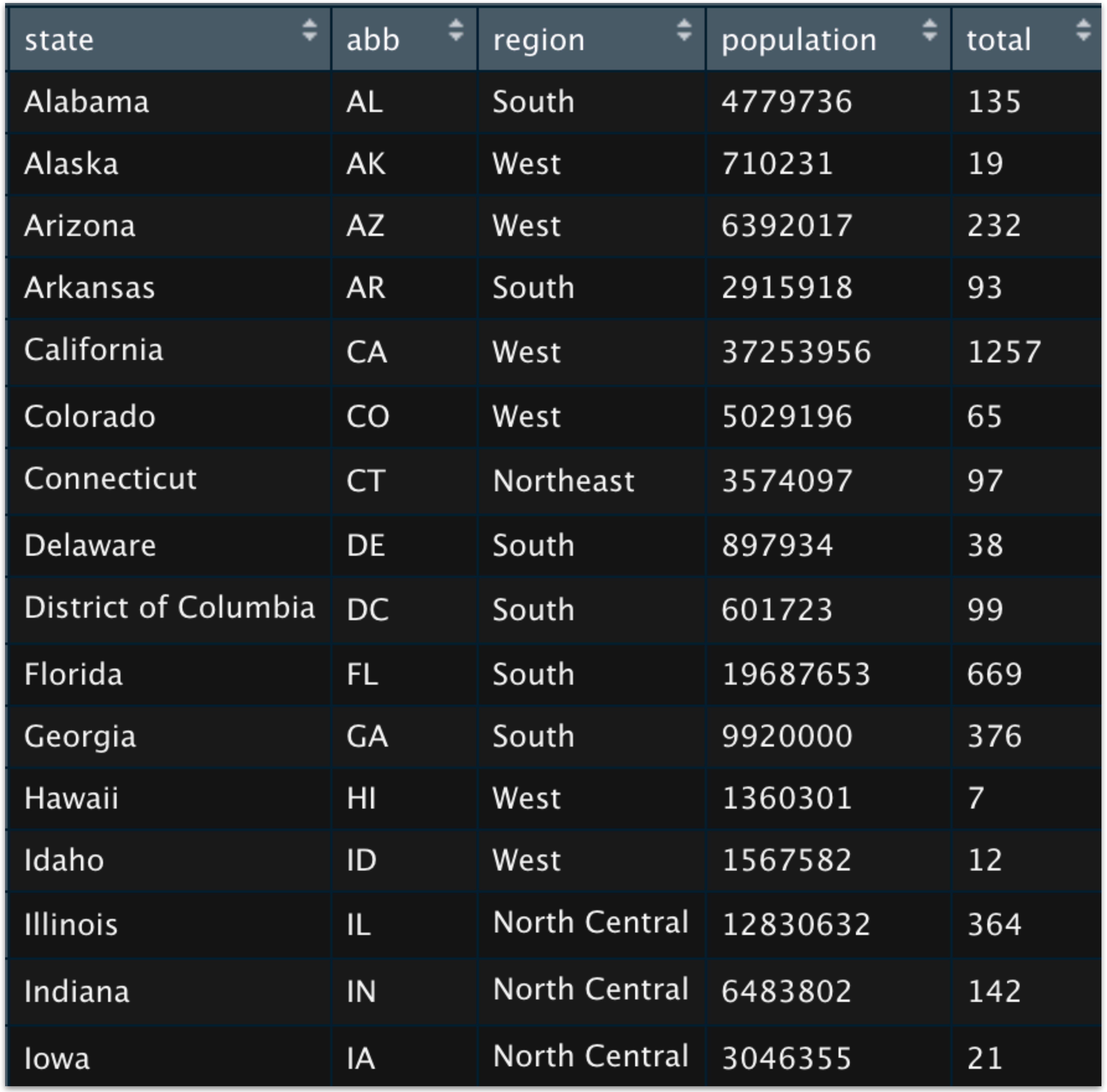
Example of tidy format

Tidy data: Ex 1

- A data frame is in ***tidy*** format if each row represents one observation and each column represents a different variable

```
library(dslabs)
data(murders)
View(murders)
```

→ loading *dslabs* package



state	abb	region	population	total
Alabama	AL	South	4779736	135
Alaska	AK	West	710231	19
Arizona	AZ	West	6392017	232
Arkansas	AR	South	2915918	93
California	CA	West	37253956	1257
Colorado	CO	West	5029196	65
Connecticut	CT	Northeast	3574097	97
Delaware	DE	South	897934	38
District of Columbia	DC	South	601723	99
Florida	FL	South	19687653	669
Georgia	GA	South	9920000	376
Hawaii	HI	West	1360301	7
Idaho	ID	West	1567582	12
Illinois	IL	North Central	12830632	364
Indiana	IN	North Central	6483802	142
Iowa	IA	North Central	3046355	21

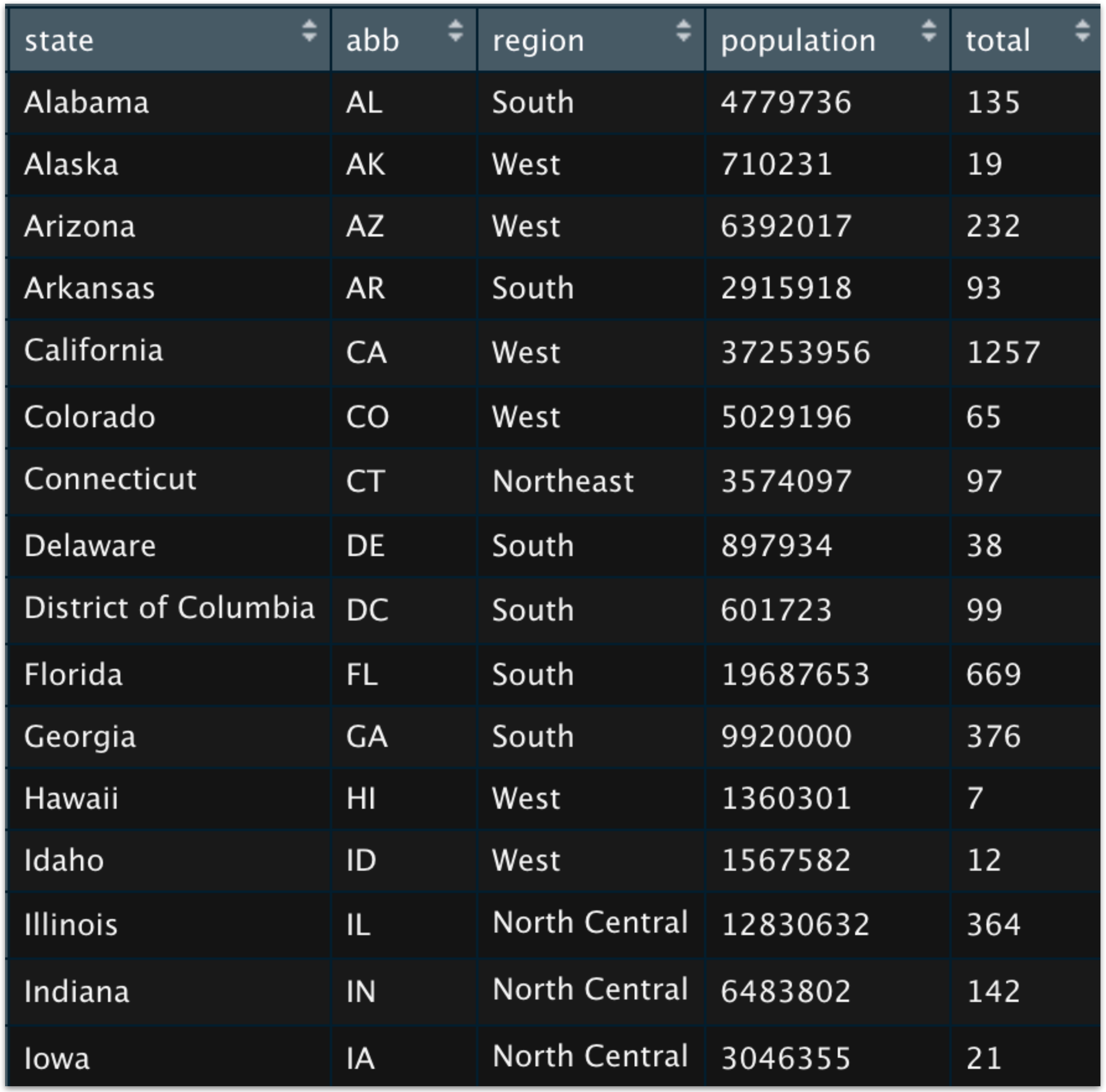
Example of tidy format

Tidy data: Ex 1

- A data frame is in ***tidy*** format if each row represents one observation and each column represents a different variable

```
library(dslabs)
data(murders)
View(murders)
```

$\longrightarrow$ loading *murders* data



state	abb	region	population	total
Alabama	AL	South	4779736	135
Alaska	AK	West	710231	19
Arizona	AZ	West	6392017	232
Arkansas	AR	South	2915918	93
California	CA	West	37253956	1257
Colorado	CO	West	5029196	65
Connecticut	CT	Northeast	3574097	97
Delaware	DE	South	897934	38
District of Columbia	DC	South	601723	99
Florida	FL	South	19687653	669
Georgia	GA	South	9920000	376
Hawaii	HI	West	1360301	7
Idaho	ID	West	1567582	12
Illinois	IL	North Central	12830632	364
Indiana	IN	North Central	6483802	142
Iowa	IA	North Central	3046355	21

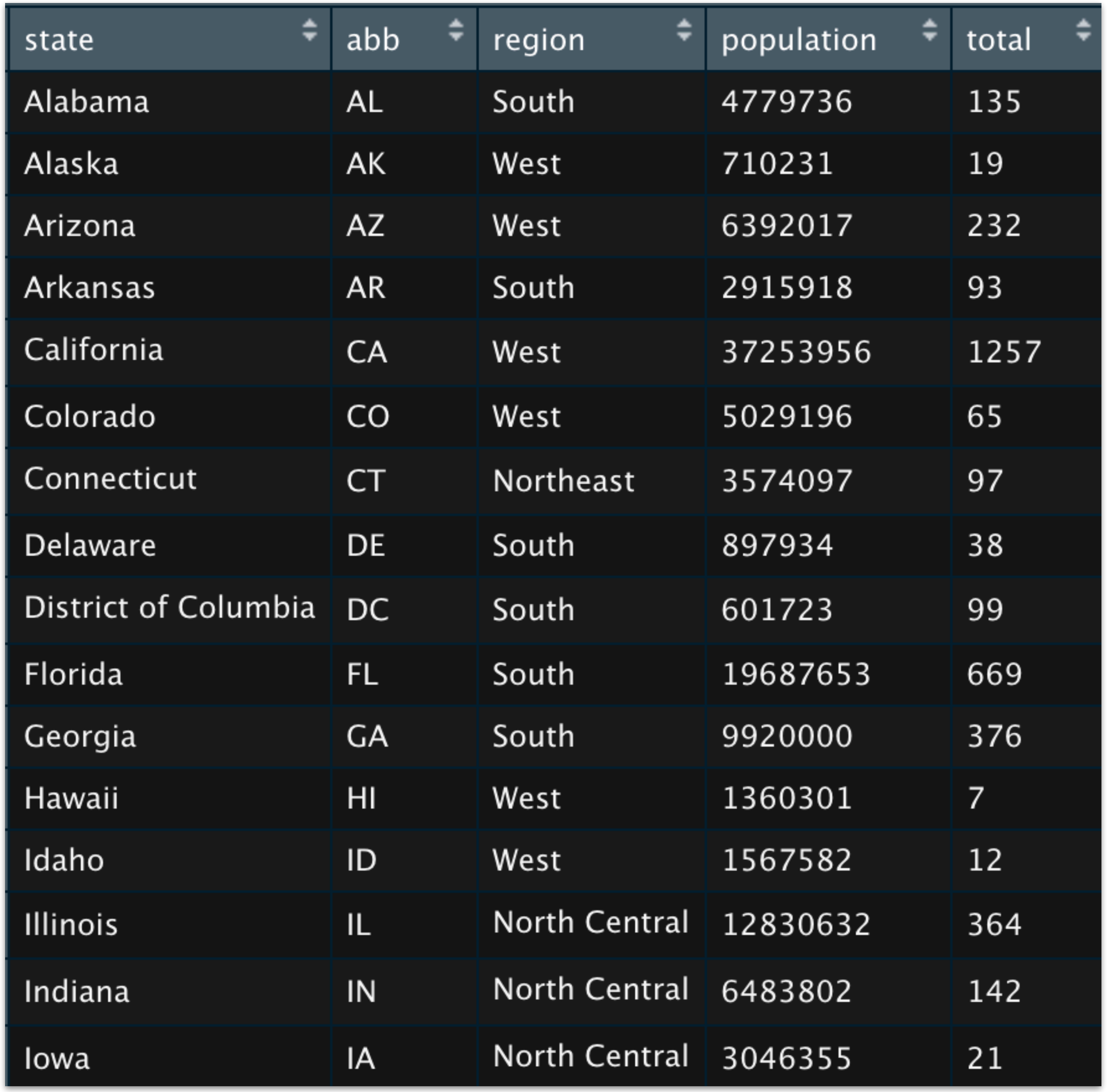
Example of tidy format

Tidy data: Ex 1

- A data frame is in ***tidy*** format if each row represents one observation and each column represents a different variable

```r
library(dslabs)
data(murders)
View(murders)
```

→ view *murders* data



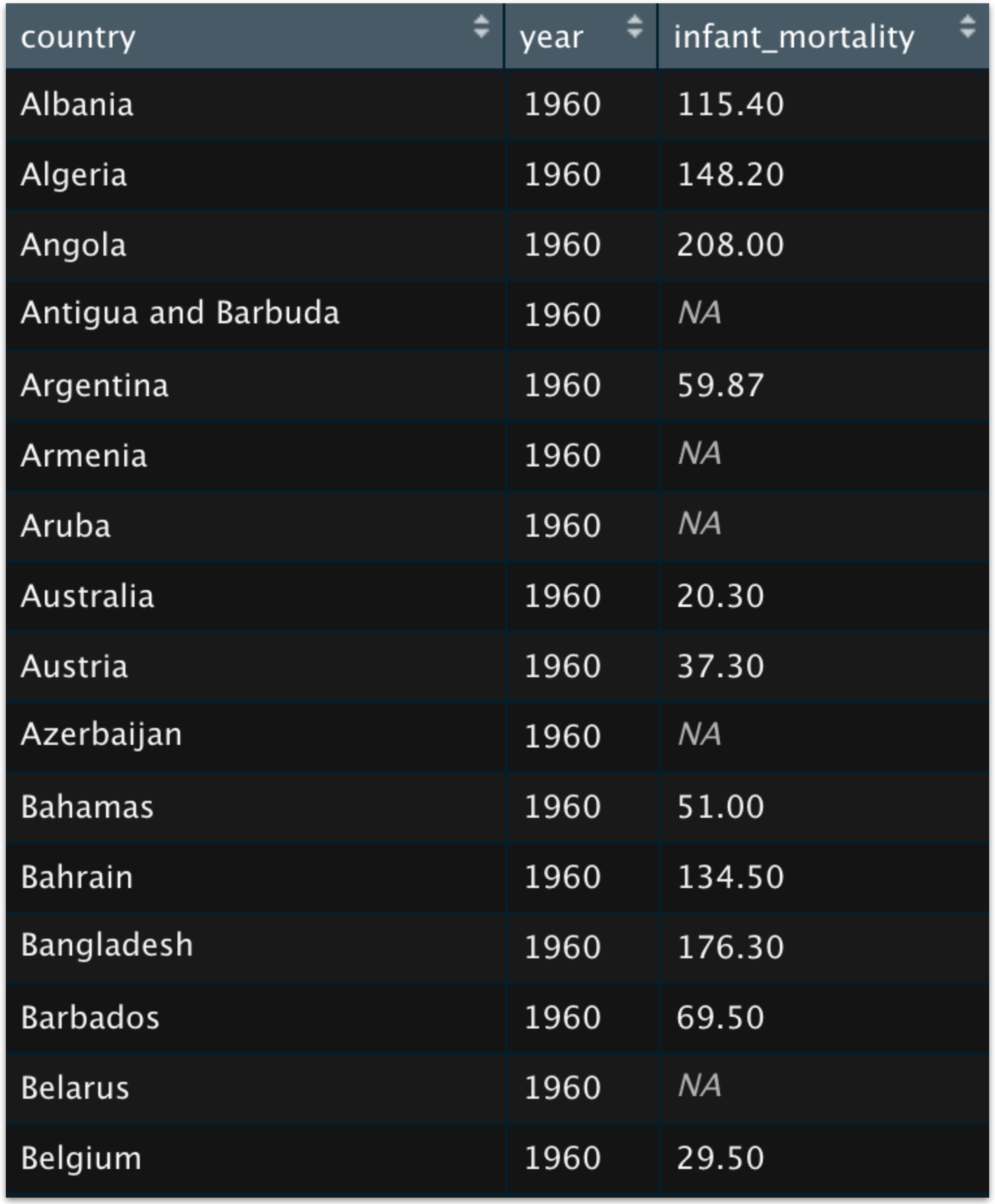
state	abb	region	population	total
Alabama	AL	South	4779736	135
Alaska	AK	West	710231	19
Arizona	AZ	West	6392017	232
Arkansas	AR	South	2915918	93
California	CA	West	37253956	1257
Colorado	CO	West	5029196	65
Connecticut	CT	Northeast	3574097	97
Delaware	DE	South	897934	38
District of Columbia	DC	South	601723	99
Florida	FL	South	19687653	669
Georgia	GA	South	9920000	376
Hawaii	HI	West	1360301	7
Idaho	ID	West	1567582	12
Illinois	IL	North Central	12830632	364
Indiana	IN	North Central	6483802	142
Iowa	IA	North Central	3046355	21

Example of tidy format

Tidy data: Ex 2

- A data frame is in ***tidy*** format if each row represents one observation and each column represents a different variable

```
library(dslabs)
data(gapminder)
View(gapminder)
```



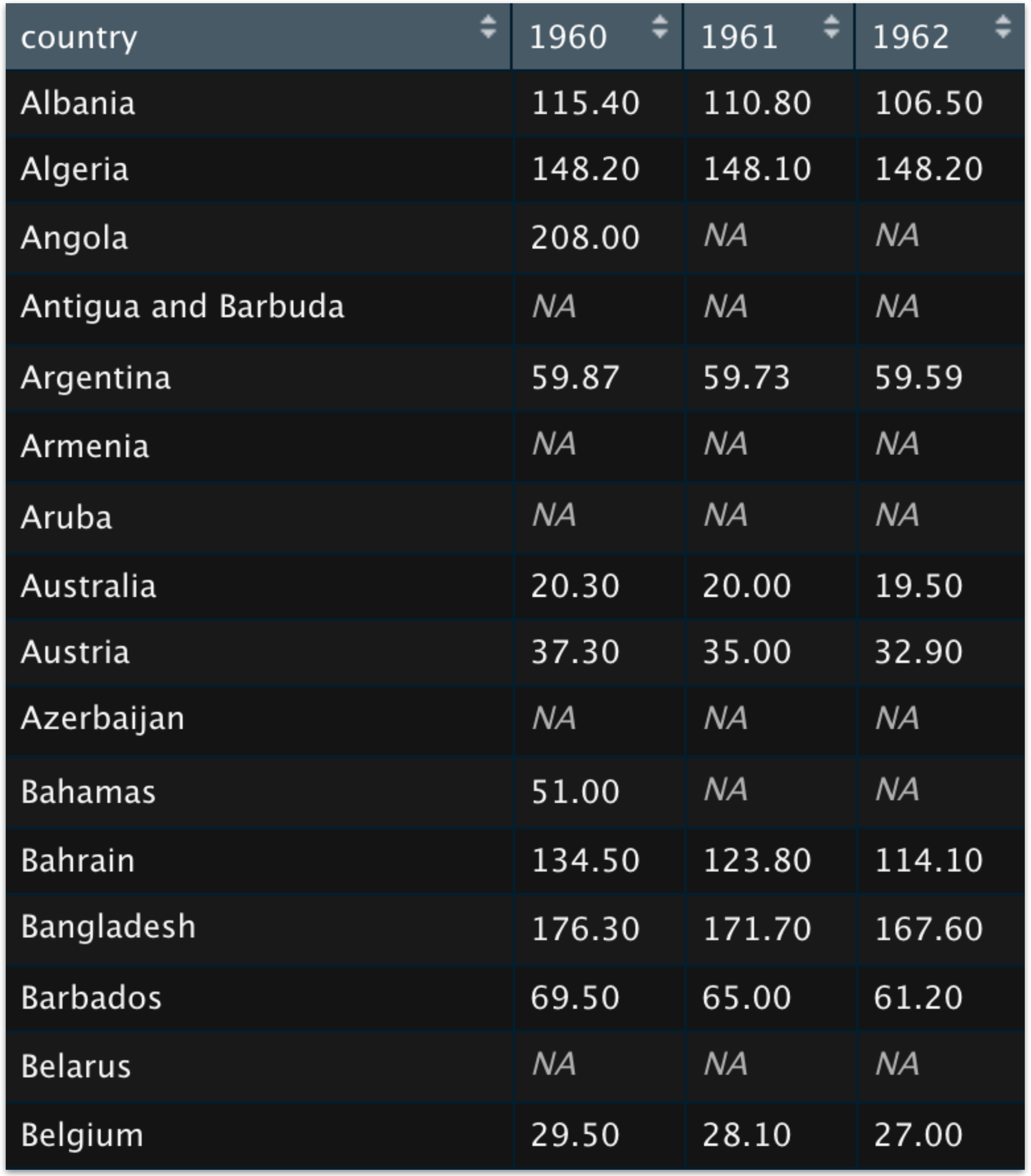
country	year	infant_mortality
Albania	1960	115.40
Algeria	1960	148.20
Angola	1960	208.00
Antigua and Barbuda	1960	*NA*
Argentina	1960	59.87
Armenia	1960	*NA*
Aruba	1960	*NA*
Australia	1960	20.30
Austria	1960	37.30
Azerbaijan	1960	*NA*
Bahamas	1960	51.00
Bahrain	1960	134.50
Bangladesh	1960	176.30
Barbados	1960	69.50
Belarus	1960	*NA*
Belgium	1960	29.50

Example of tidy format

Tidy data: Ex 2

- A data frame is in ***tidy*** format if each row represents one observation and each column represents a different variable

```
library(dslabs)
data(gapminder)
View(gapminder)
```



country	1960	1961	1962
Albania	115.40	110.80	106.50
Algeria	148.20	148.10	148.20
Angola	208.00	*NA*	*NA*
Antigua and Barbuda	*NA*	*NA*	*NA*
Argentina	59.87	59.73	59.59
Armenia	*NA*	*NA*	*NA*
Aruba	*NA*	*NA*	*NA*
Australia	20.30	20.00	19.50
Austria	37.30	35.00	32.90
Azerbaijan	*NA*	*NA*	*NA*
Bahamas	51.00	*NA*	*NA*
Bahrain	134.50	123.80	114.10
Bangladesh	176.30	171.70	167.60
Barbados	69.50	65.00	61.20
Belarus	*NA*	*NA*	*NA*
Belgium	29.50	28.10	27.00

Not an example of tidy format

Transforming data frames

- We can use functions from the package ***dplyr*** to transform data frames:
  - `mutate`
  - `filter`
  - `select`
  - The pipe operator (%>%)
  - `summarize`
  - `group_by`
  - `do`

Adding a column with `mutate()`

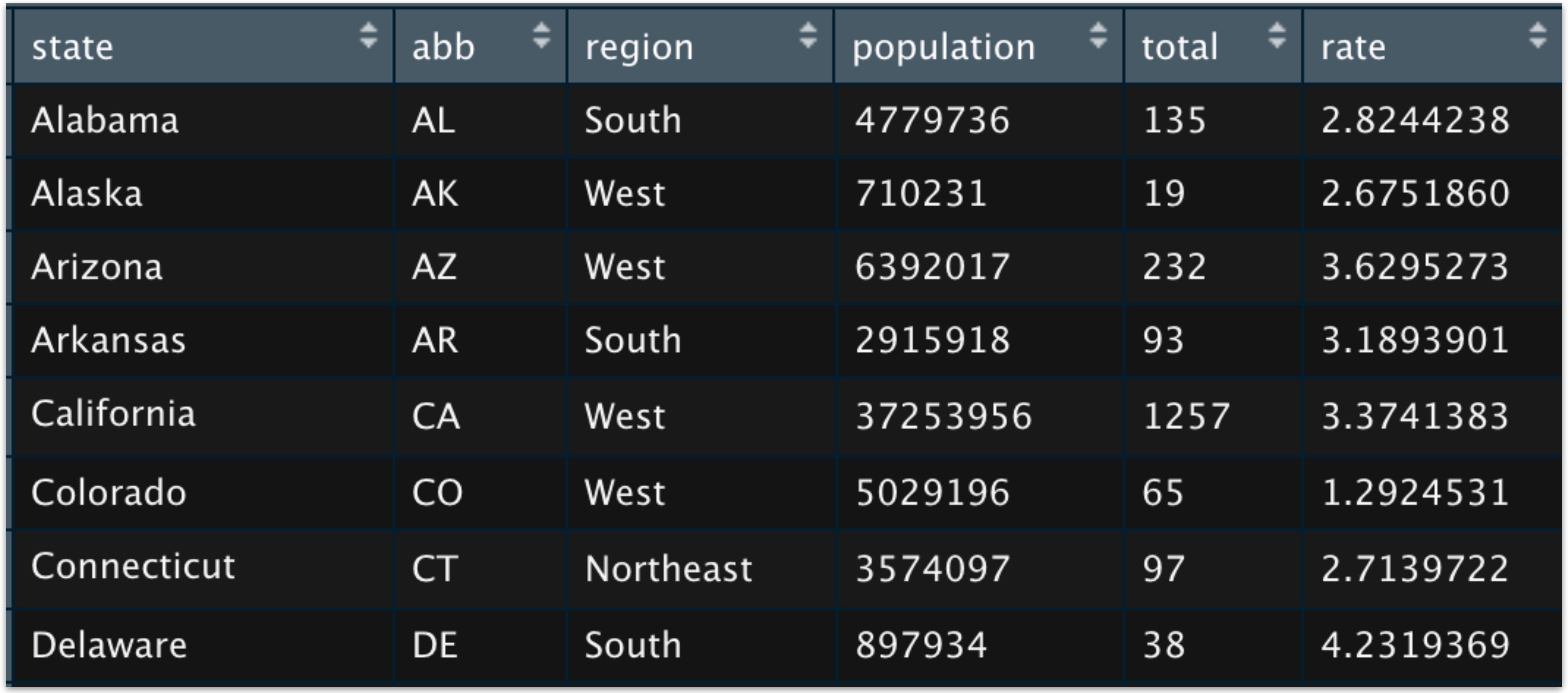
- Let's add a column with murder rates to the ***murders*** dataset.
- Syntax for the function `mutate`:

```
mutate(data frame, name = value)
```

- `data frame`: Name of data frame of interest
- `name`: Name of the new column
- `value`: Values that the variable should take

Adding a column with `mutate()`

```r
library(dslabs)
library(dplyr)
data("murders")
murders <- mutate(murders, rate = total / population * 100000)
```



state	abb	region	population	total	rate
Alabama	AL	South	4779736	135	2.8244238
Alaska	AK	West	710231	19	2.6751860
Arizona	AZ	West	6392017	232	3.6295273
Arkansas	AR	South	2915918	93	3.1893901
California	CA	West	37253956	1257	3.3741383
Colorado	CO	West	5029196	65	1.2924531
Connecticut	CT	Northeast	3574097	97	2.7139722
Delaware	DE	South	897934	38	4.2319369

Subsetting with `filter()`

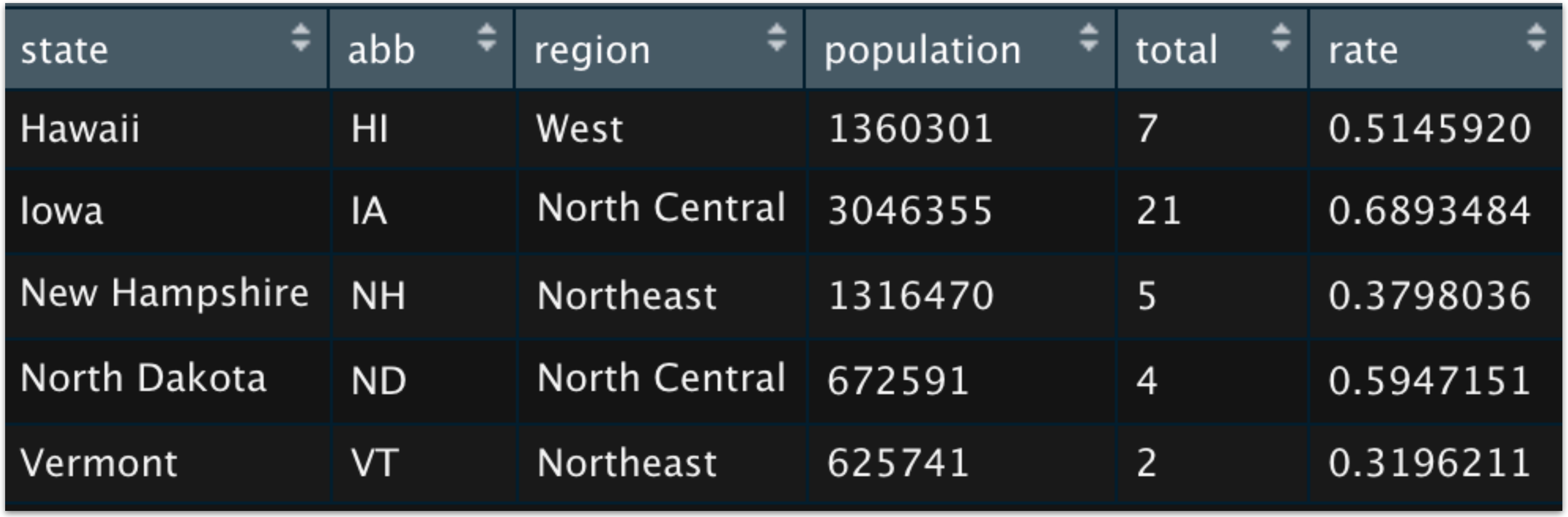
- Say that we want to only show entries with a murder rate lower than or equal to 0.71
- Syntax for the function `filter`:

```
filter(data frame, condition)
```

- `data frame`: Name of data frame of interest
- `condition`: A rule use to subset data

Subsetting with `filter()`

```r
filter(murders, rate <= 0.71)
```



state	abb	region	population	total	rate
Hawaii	HI	West	1360301	7	0.5145920
Iowa	IA	North Central	3046355	21	0.6893484
New Hampshire	NH	Northeast	1316470	5	0.3798036
North Dakota	ND	North Central	672591	4	0.5947151
Vermont	VT	Northeast	625741	2	0.3196211

Selecting columns with `select()`

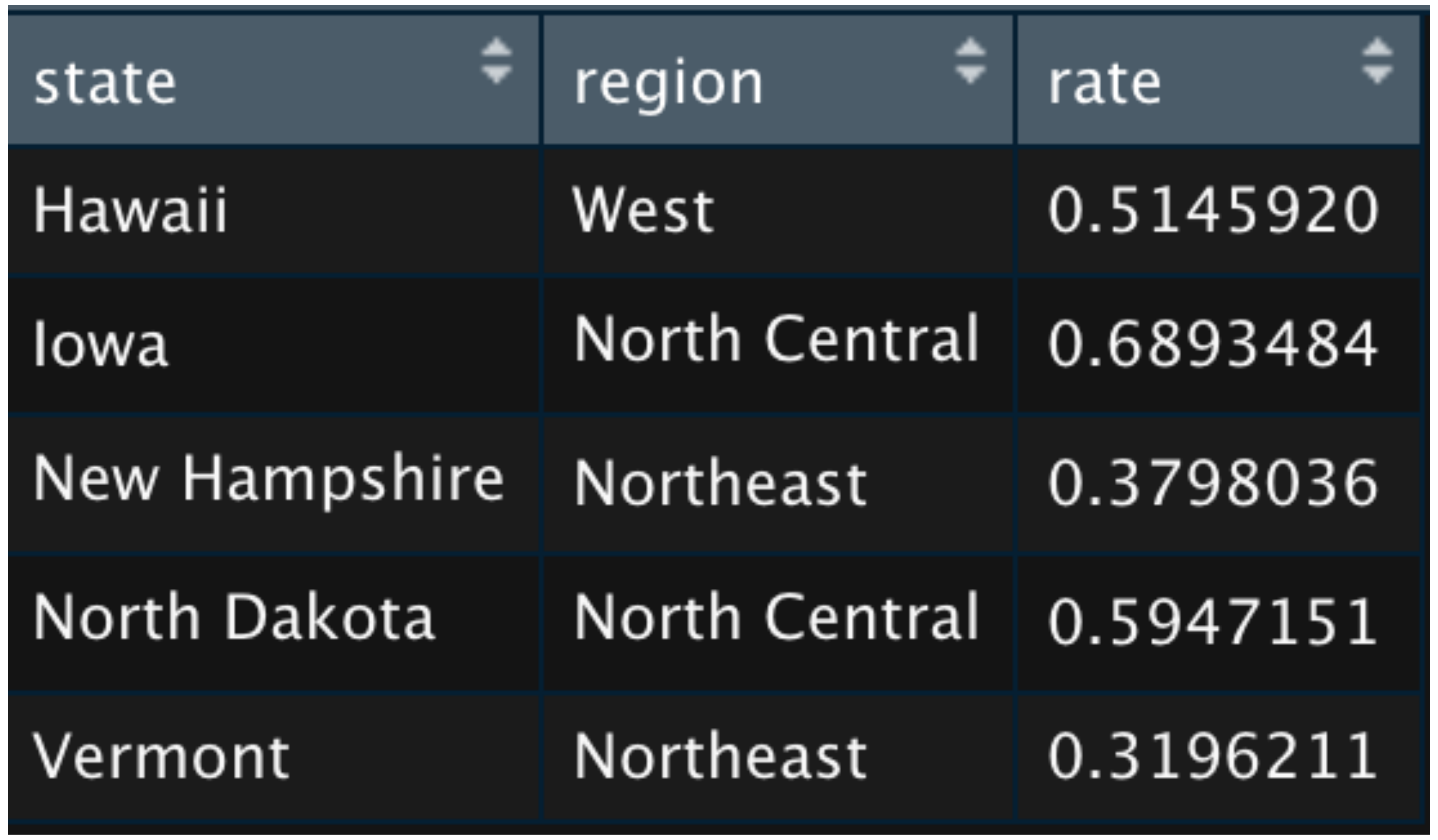
- In this example let's select a few columns from the original dataset and then filter as we did before
- Syntax for the function `select`:

```
select(data frame, columns)
```

- `data frame`: Name of data frame of interest
- `columns`: Name of the columns of interest

Selecting columns with `select()`

```r
new_table <- select(murders, state, region, rate)
filter(new_table, rate <= 0.71)
```



state	region	rate
Hawaii	West	0.5145920
Iowa	North Central	0.6893484
New Hampshire	Northeast	0.3798036
North Dakota	North Central	0.5947151
Vermont	Northeast	0.3196211

The pipe operator: %>%

- We used the following code in the previous slide:

```r
new_table <- select(murders, state, region, rate)
filter(new_table, rate <= 0.71)
```

- However, we can perform a series of operations by sending the results of one function to another with the pipe operator (%>%)

$$\begin{tikzcd} \text{original data} \arrow[r] & \texttt{select} \arrow[r] & \texttt{filter} \end{tikzcd}$$

The pipe operator: %>%

- Let's look at few examples:

```r
16 %>% sqrt()
```

```r
16 %>% sqrt() %>% log2()
```

- The first one yields 4 and the second 2
- Note that the pipe sends values to the first argument, so we can define other arguments as if the first one is defined

```r
16 %>% sqrt() %>% log(base = 2)
```

The pipe operator: %>%

- Original code

```
new_table <- select(murders, state, region, rate)
filter(new_table, rate <= 0.71)
```

- New code

```
murders %>%
   select(state, region, rate) %>%
   filter(rate <= 0.71)
```

- murders is the first argument to `select` and the result from `select` is the first argument to `filter`

Summarizing data

- An important step in any analysis is summarizing data:
  - mean
  - standard deviation
- Sometimes we can get more informative summaries by first splitting the data by groups and then summarizing
- Let's us introduce to functions to do this:
  - `summarize`
  - `group_by`

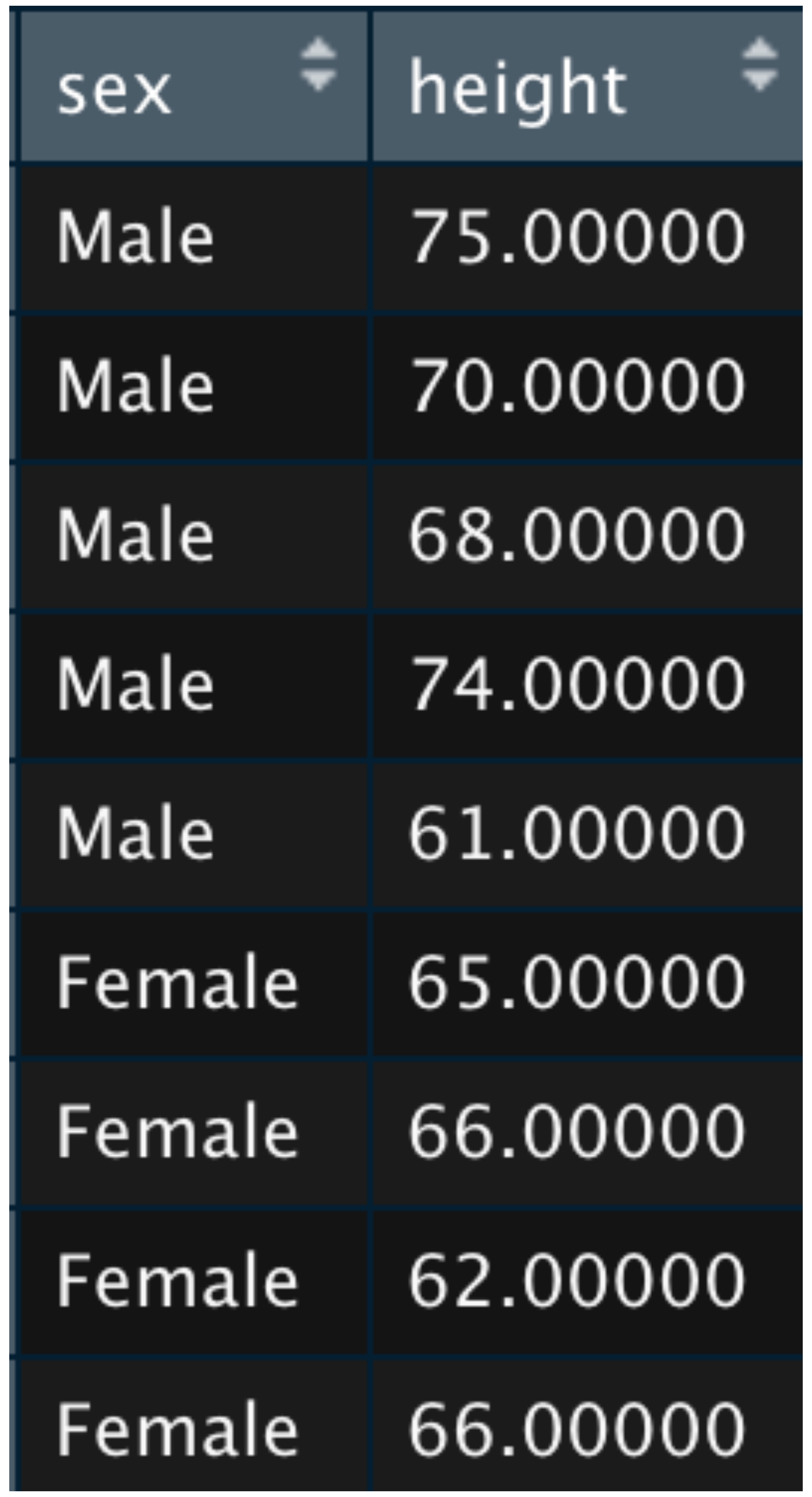
Summarizing data

- New dataset: The ***heights*** dataset includes height and sex reported by students in an in-class survey

```
library(dslabs)
library(dplyr)
data(“heights")
```

- The following code computes the mean and standard deviation for females

```
heights %>%
   filter(sex == “Female”) %>%
   summarize(average = mean(height),
             sta_dev = sd(height))
```



sex	height
Male	75.00000
Male	70.00000
Male	68.00000
Male	74.00000
Male	61.00000
Female	65.00000
Female	66.00000
Female	62.00000
Female	66.00000

Sample of ***heights*** dataset

Summarizing data

```
heights %>%
   filter(sex == “Female”) %>%
   summarize(average = mean(height), sta_dev = sd(height))
```

- This yields `average = 64.94` and `sta_dev = 3.76`

- We can compute any number of summary statistics:

```
heights %>%
   filter(sex == “Female”) %>%
   summarize(median = median(height), minimum = min(height),
             maximum = max(height))
```

Summarizing data

- Recall that we can get the minimum, median, and maximum statistics by looking at the `0%`, `50%`, and `100%` quantiles:

```
heights %>%
   filter(sex == “Female”) %>%
   summarize(range = quantile(height, c(0, 0.5, 1)))
```

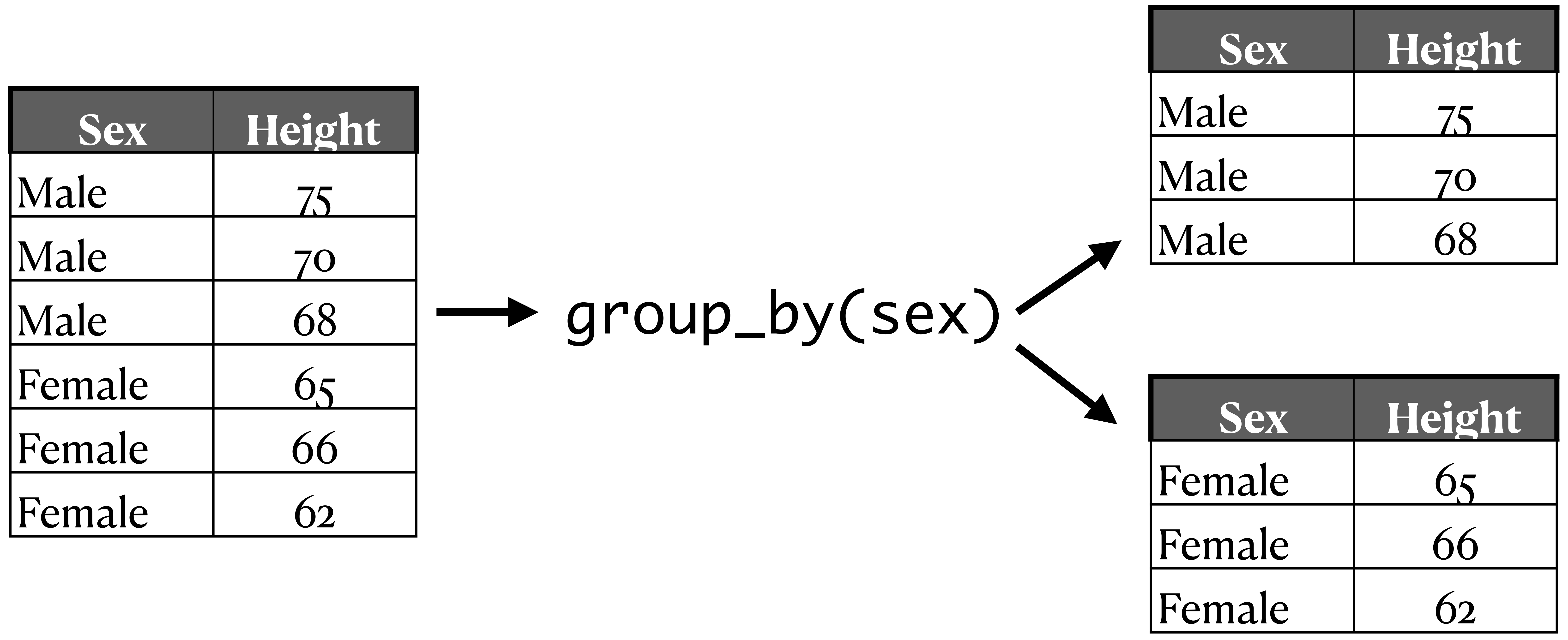
- but this return an error!
- This is because we can only call functions that return a single value within `summarize`
- One last example. Let’s compute the murder rate in the US

```
murders %>%
   summarize(rate = sum(total) / sum(population) * 100000)
```

Group and then summarize

- As stated before, is common to first split the data by groups and then provide summaries for each group. Let's compute the mean and standard deviation for males and females separately:

```
heights %>% group_by(sex)
```



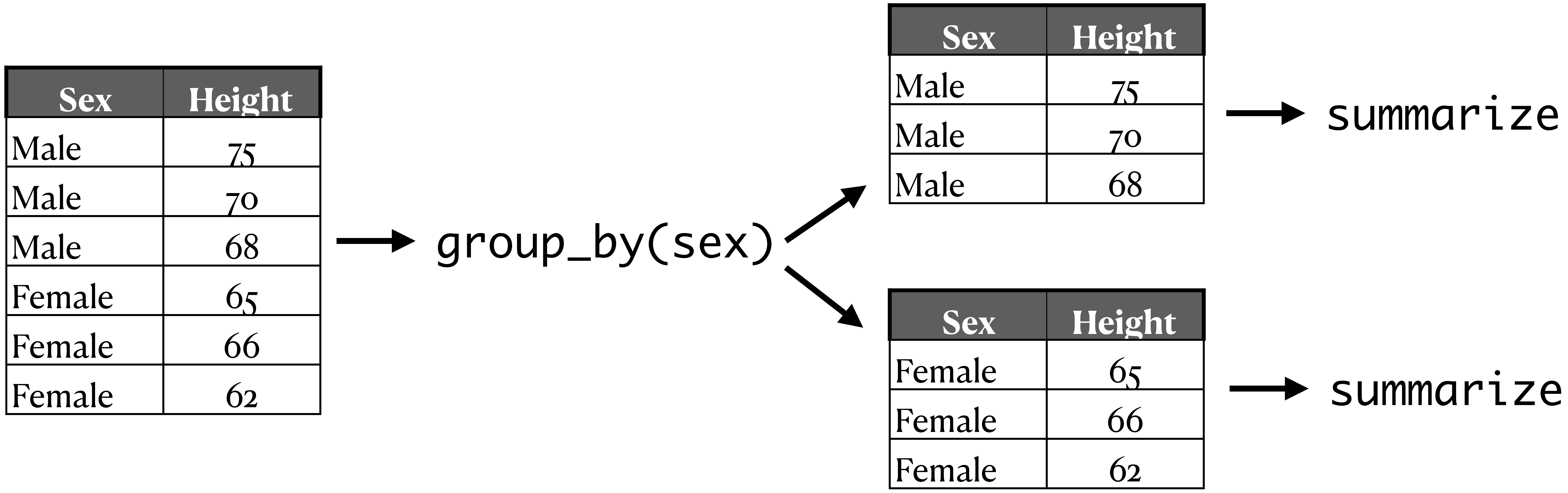
Sex	Height
Male	75
Male	70
Male	68
Female	65
Female	66
Female	62

Sex	Height
Male	75
Male	70
Male	68

Sex	Height
Female	65
Female	66
Female	62

Group and then summarize

```
heights %>%
   group_by(sex) %>%
   summarize(average = mean(height), sta_dev = sd(height))
```



Sex	Height
Male	75
Male	70
Male	68
Female	65
Female	66
Female	62

Sex	Height
Male	75
Male	70
Male	68

Sex	Height
Female	65
Female	66
Female	62

Group and then summarize

- Now suppose that we want to compute the median murder rate in the four US regions using the ***murders*** dataset. How can we do this?

Group and then summarize

- Now suppose that we want to compute the median murder rate in the four US regions using the ***murders*** dataset. How can we do this?

```
murders
```

- Start with the dataset that we want to use

Group and then summarize

- Now suppose that we want to compute the median murder rate in the four US regions using the ***murders*** dataset. How can we do this?

```
murders %>%
   group_by(region)
```

- Use the pipe operator to "send" the data to the `group_by` function
- Recall that the pipe makes ***murders*** the first argument in `group_by`
- Therefore, the only thing left is to specify which variable to group by

Group and then summarize

- Now suppose that we want to compute the median murder rate in the four US regions using the ***murders*** dataset. How can we do this?

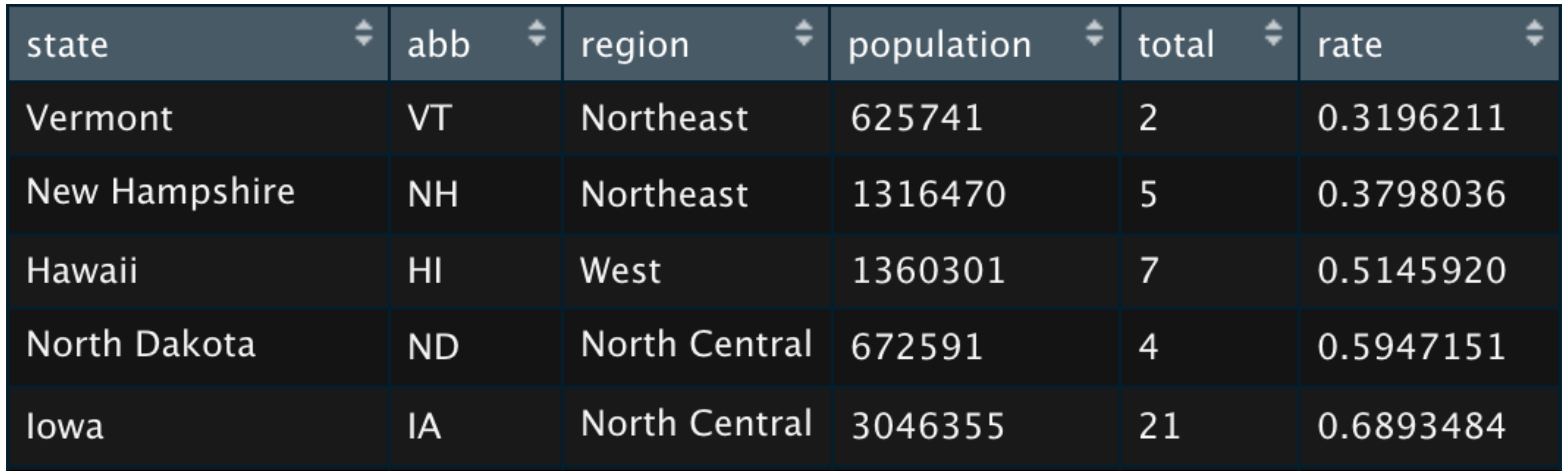
```
murders %>%
   group_by(region) %>%
   summarize(median_rate = median(rate))
```

- Finally, use `summarize` to get the median murder rates per region

Sorting data frames

- We can use `arrange` to sort dataframes

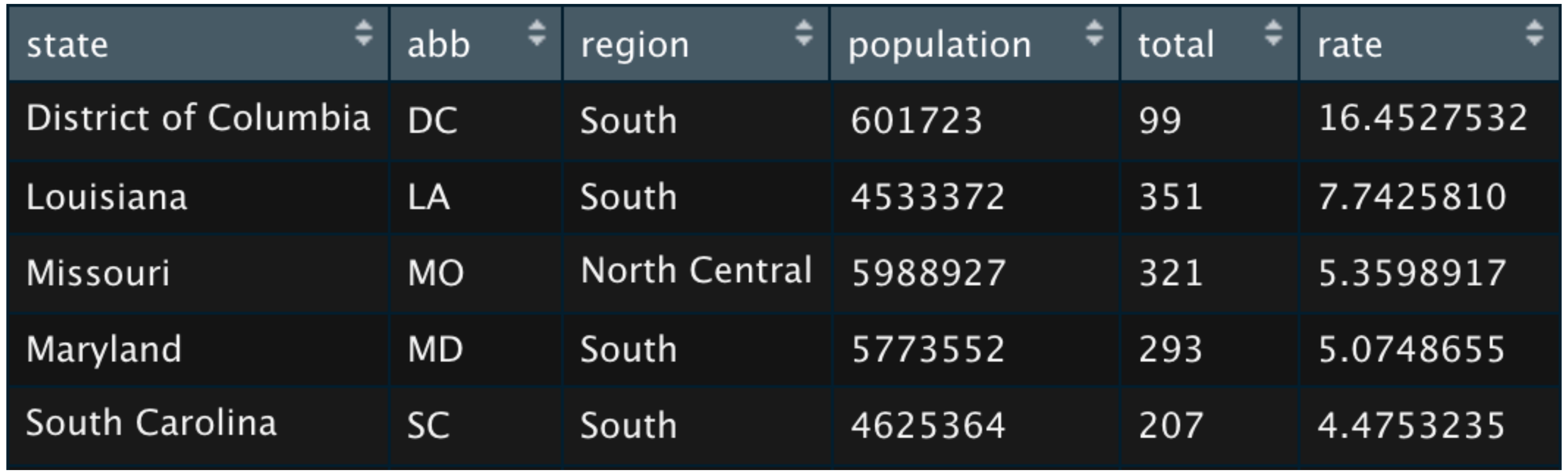
```
murders %>%
  arrange(rate)
```



state	abb	region	population	total	rate
Vermont	VT	Northeast	625741	2	0.3196211
New Hampshire	NH	Northeast	1316470	5	0.3798036
Hawaii	HI	West	1360301	7	0.5145920
North Dakota	ND	North Central	672591	4	0.5947151
Iowa	IA	North Central	3046355	21	0.6893484

- To sort in descending order we can use `desc`

```
murders %>%
  arrange(desc(rate))
```

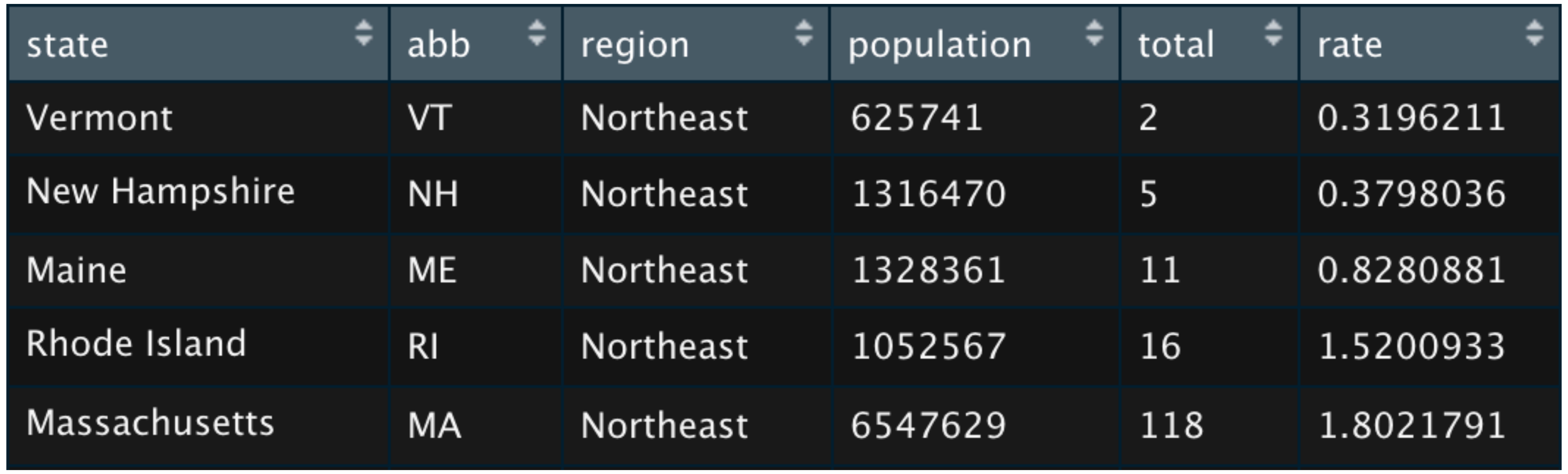


state	abb	region	population	total	rate
District of Columbia	DC	South	601723	99	16.4527532
Louisiana	LA	South	4533372	351	7.7425810
Missouri	MO	North Central	5988927	321	5.3598917
Maryland	MD	South	5773552	293	5.0748655
South Carolina	SC	South	4625364	207	4.4753235

Sorting data frames

- We can also do nested sorting

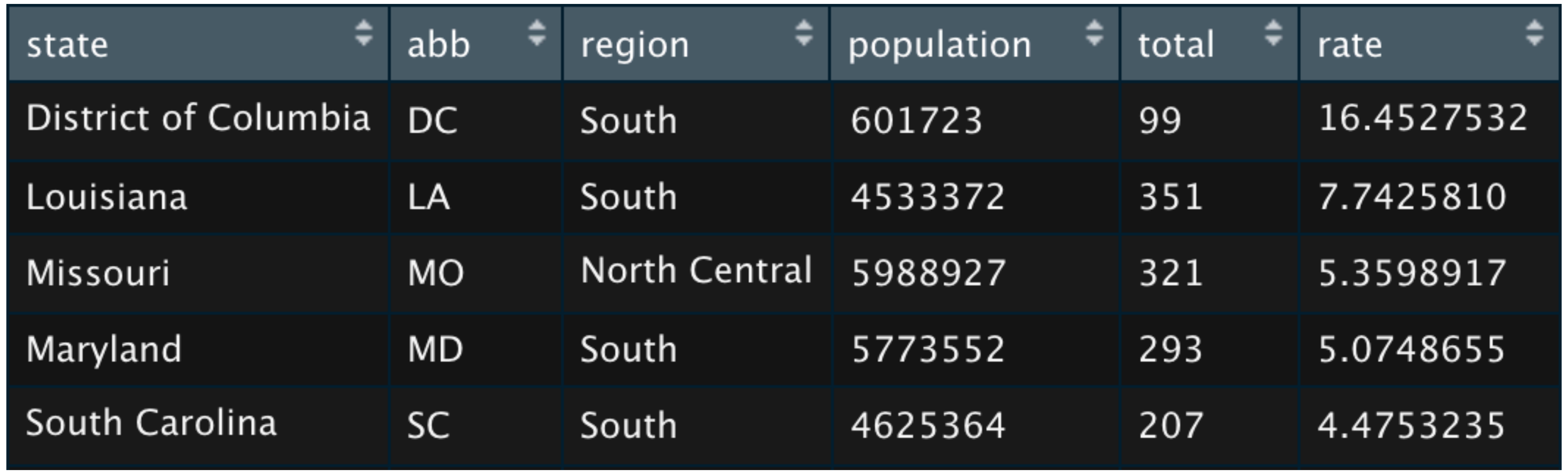
```r
murders %>%
  arrange(region, rate)
```



state	abb	region	population	total	rate
Vermont	VT	Northeast	625741	2	0.3196211
New Hampshire	NH	Northeast	1316470	5	0.3798036
Maine	ME	Northeast	1328361	11	0.8280881
Rhode Island	RI	Northeast	1052567	16	1.5200933
Massachusetts	MA	Northeast	6547629	118	1.8021791

- Finally, if we want to get top *n* observations we can use `top_n`

```r
murders %>%
  top_n(5, rate)
```



state	abb	region	population	total	rate
District of Columbia	DC	South	601723	99	16.4527532
Louisiana	LA	South	4533372	351	7.7425810
Missouri	MO	North Central	5988927	321	5.3598917
Maryland	MD	South	5773552	293	5.0748655
South Carolina	SC	South	4625364	207	4.4753235

Quick detour: Tibbles

- Tidy data must be stored in data frames
- A ***tibble*** is a special kind of data frame that have many appealing qualities
- All the functions we have seen so far return a ***tibble***
- More on this during the hands-on section

The do function

- Most R functions do not accept ***tibbles*** nor do they return data frames
- Recall the `quantile` example from before:

```
heights %>%
   filter(sex == “Female”) %>%
   summarize(range = quantile(height, c(0, 0.5, 1)))
```

- which yields the following error:

```
Error: expecting result of length one, got : 2
```

- The do function serves as a bridge between R

The do function

- Let's use the do function to get around this
- First, we have to write a function that takes a data frame as an argument and returns a data frame

```
my_summary <- function(dat){
  x <- quantile(dat$height, c(0, 0.5, 1))
  tibble(min = x[1], median = x[2], max = x[3])
}
```

- Now we can use the following code:

```
heights %>%
  group_by(sex) %>%
  do(my_summary(.))
```

The do function

- Let's use the do function to get around this
- First, we have to write a function that takes a data frame as an argument and returns a data frame

```r
my_summary <- function(dat){
  x <- quantile(dat$height, c(0, 0.5, 1))
  tibble(min = x[1], median = x[2], max = x[3])
}
```

- Now we can use the following code:

```r
heights %>%
  group_by(sex) %>%
  do(my_summary(.))
```

- Why the dot?

The `do` function

- The ***tibble*** created by `group_by` is piped to `do`
- Within the call to `do`, the name of this tibble is `.` and we want to send it to `my_summary`

References

1. Introduction to Data Science: Data analysis and prediction algorithms with R by Rafael A. Irizarry, Chapter 4. https://rafalab.github.io/dsbook/
2. R for Data Science by Grolemund & Wickham, Chapter 5. https://r4ds.had.co.nz/index.html

Referencias en español:

1. Introducción a la Ciencia de Datos: Análisis de datos y algoritmos de predicción con R por Rafael A. Irizarry, Capítulo 4. https://rafalab.github.io/dslibro/
2. R para Ciencia de Datos por Grolemund & Wickham, Capítulo 5. https://es.r4ds.hadley.nz

Your turn!