CPE 213 Data Models

(a.k.a. Data Modeling and Visualization)

Lecture 2: R Tutorial

Asst. Prof. Dr. Santitham Prom-on

Department of Computer Engineering, Faculty of Engineering
King Mongkut’s University of Technology Thonburi

Lecture 2 Overview

- R basic
- Data manipulation with tidyr and dplyr

http://fastdata.in.th/data-model-2021/

R Basics

Section 1

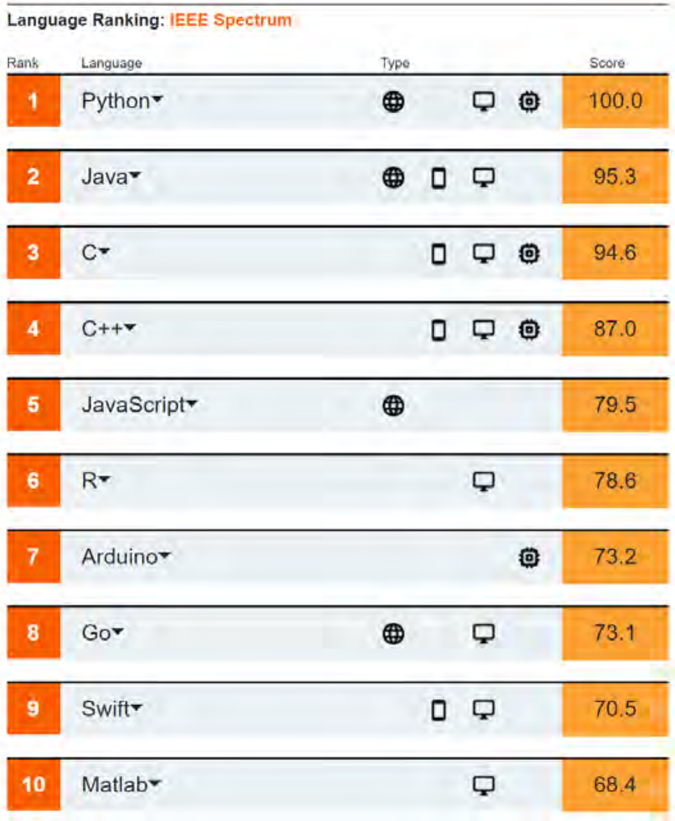
What is R?

- R is a scripting language for data manipulation and analysis

- R is originated from S language at Bell Laboratories (AT&T)
    - R and S were created with purposes of providing an interactive environment without requiring programming skills

- Over the years, through a lot of academic and commercial contributions, numerous packages were developed with various purposes and better qualities.

IEEE Ranking of Popular Programming Language 2016



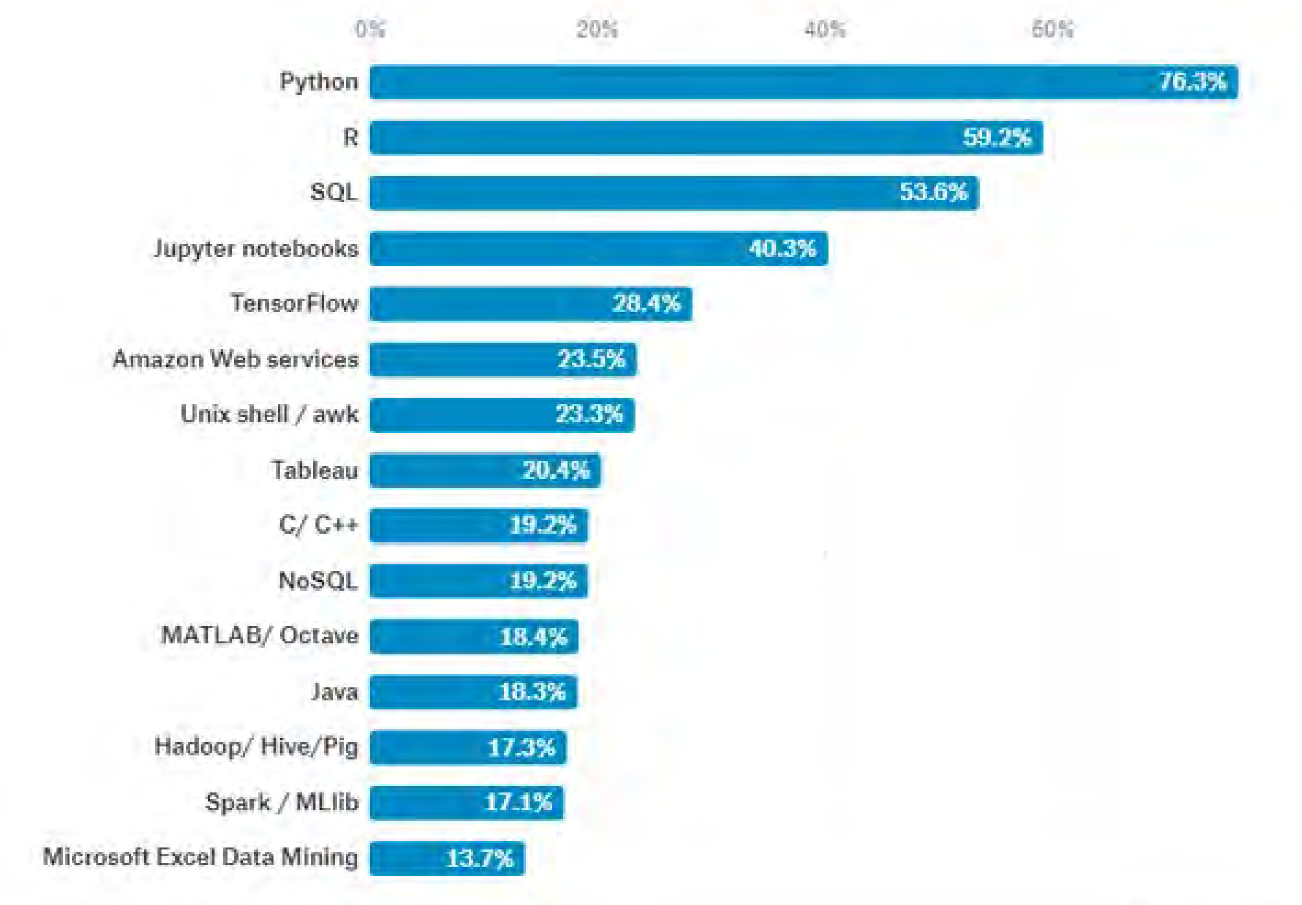
Language Ranking: IEEE Spectrum

Rank	Language	Type	Score
1	Python▾	🌐 🖥 ⚙	100.0
2	Java▾	🌐 📱 🖥	95.3
3	C▾	📱 🖥 ⚙	94.6
4	C++▾	📱 🖥 ⚙	87.0
5	JavaScript▾	🌐	79.5
6	R▾	🖥	78.6
7	Arduino▾	⚙	73.2
8	Go▾	🌐 🖥	73.1
9	Swift▾	📱 🖥	70.5
10	Matlab▾	🖥	68.4

https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2020

KDnuggets

Most Requested programming languages for Data Science in 2019



7,955 responses

Only displaying the top 15 answers. There are 38 answers not shown.

Why use R?

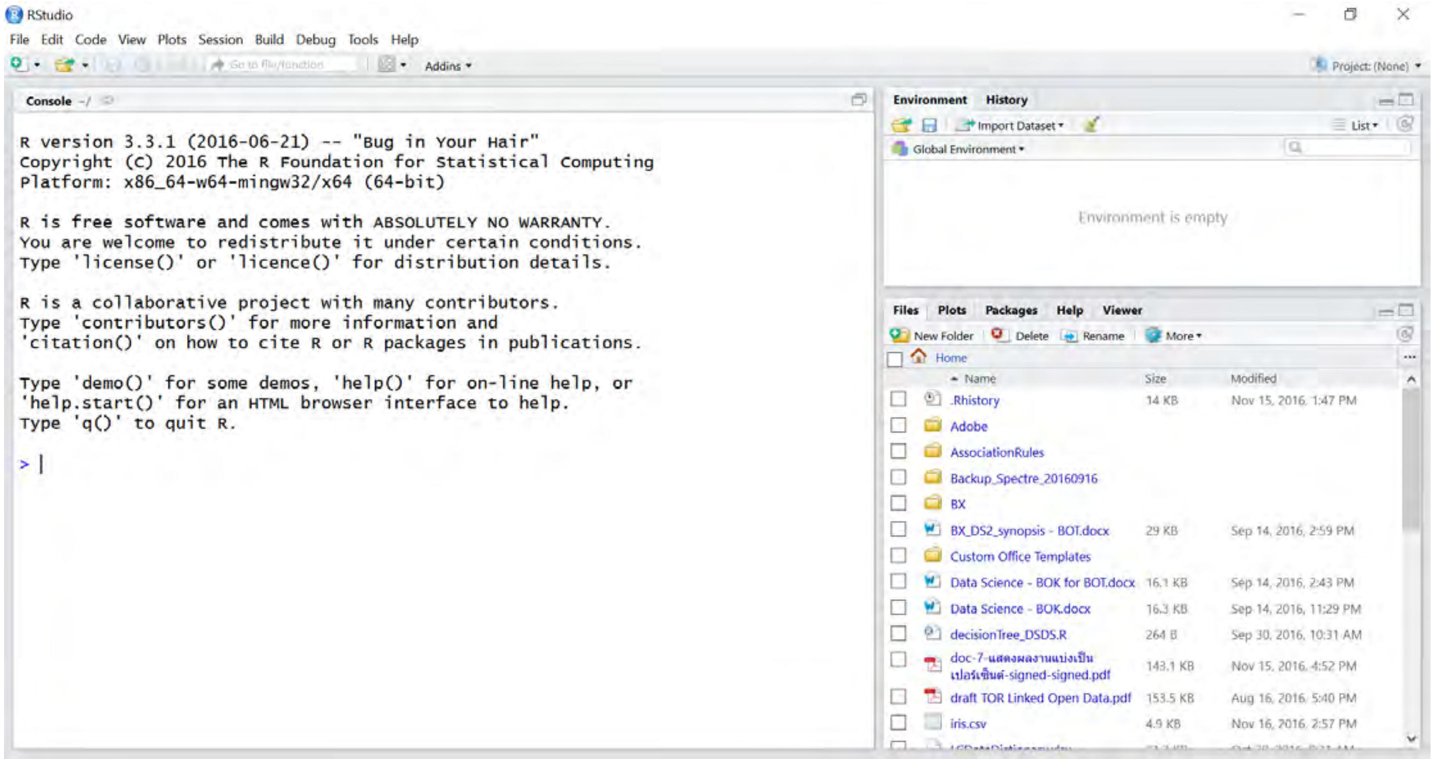
- Free
- Easy to get the job done
- Clear, more compact code
- Easier transition to distributed computing
- Great supporting community
- Object-oriented programming
- Functional programming

Learning R

- Learning R is not like learning C or Java languages
- R is a dynamic typing (no declaration required)
- R is functional (use function to run)
- R is data oriented (utilize data structure to do the work)
- R is learned through an interactive exploration of how to interact with the environment and data

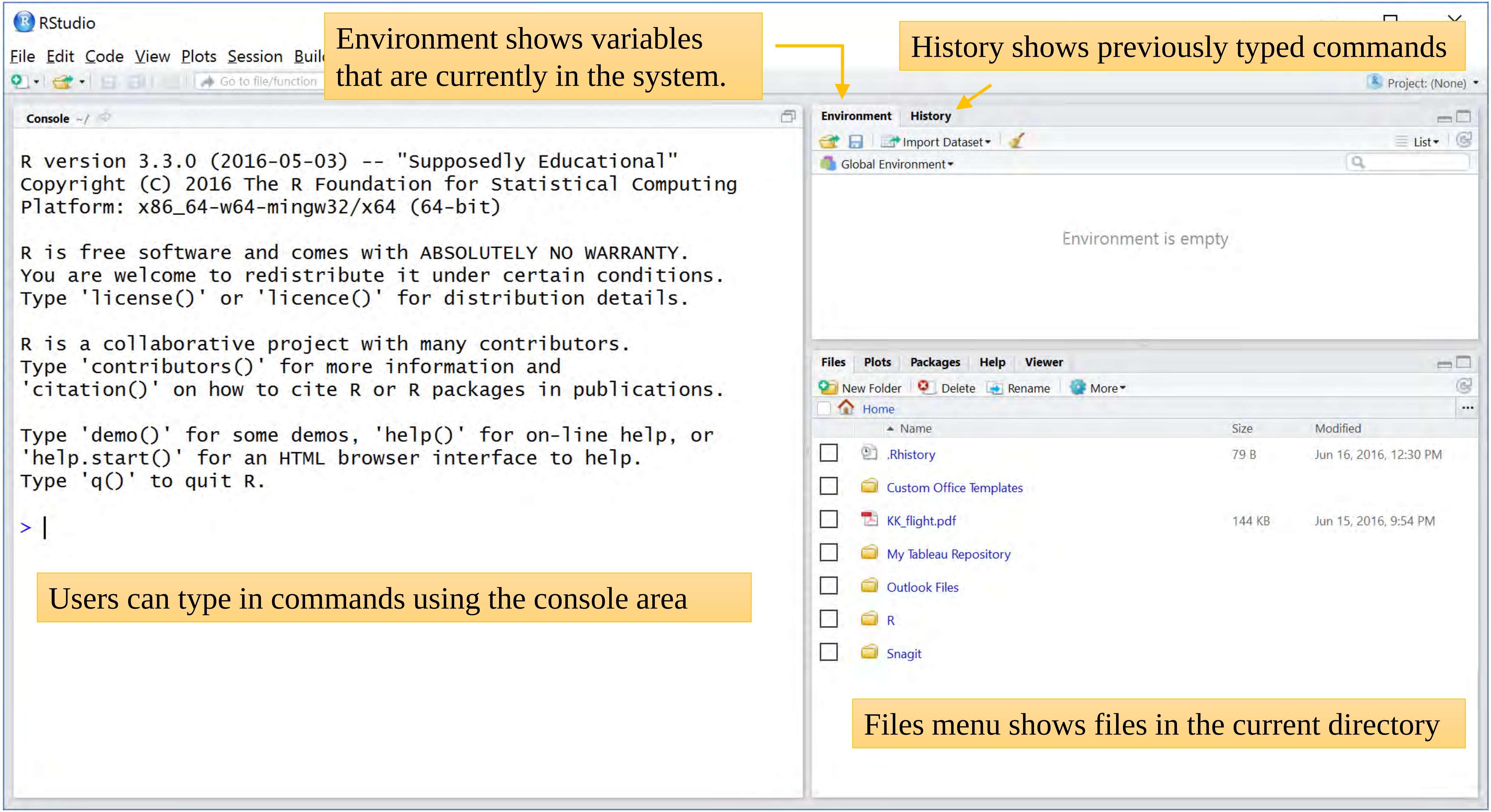
Rstudio (via www.rstudio.com)

- RStudio is an IDE for R
- RStudio makes R easier to use
  - Console
  - Syntax highlight editor
  - Plotting
  - History
  - Environment
  - Help
  - Debugging



```
R version 3.3.1 (2016-06-21) -- "Bug in Your Hair"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

Name	Size	Modified
.Rhistory	14 KB	Nov 15, 2016, 1:47 PM
Adobe		
AssociationRules		
Backup_Spectre_20160916		
BX		
BX_DS2_synopsis - BOT.docx	29 KB	Sep 14, 2016, 2:59 PM
Custom Office Templates		
Data Science - BOK for BOT.docx	16.1 KB	Sep 14, 2016, 2:43 PM
Data Science - BOK.docx	16.3 KB	Sep 14, 2016, 11:29 PM
decisionTree_DSDS.R	264 B	Sep 30, 2016, 10:31 AM
doc-7-แสดงผลงานแบ่งเป็นเปอร์เซ็นต์-signed-signed.pdf	143.1 KB	Nov 15, 2016, 4:52 PM
draft TOR Linked Open Data.pdf	153.5 KB	Aug 16, 2016, 5:40 PM
iris.csv	4.9 KB	Nov 16, 2016, 2:57 PM



Environment shows variables that are currently in the system.

History shows previously typed commands

```
R version 3.3.0 (2016-05-03) -- "Supposedly Educational"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

Environment is empty

Name	Size	Modified
.Rhistory	79 B	Jun 16, 2016, 12:30 PM
Custom Office Templates		
KK_flight.pdf	144 KB	Jun 15, 2016, 9:54 PM
My Tableau Repository		
Outlook Files		
R		
Snagit		

Users can type in commands using the console area

Files menu shows files in the current directory

Try this…

Type these commands and guess their functions

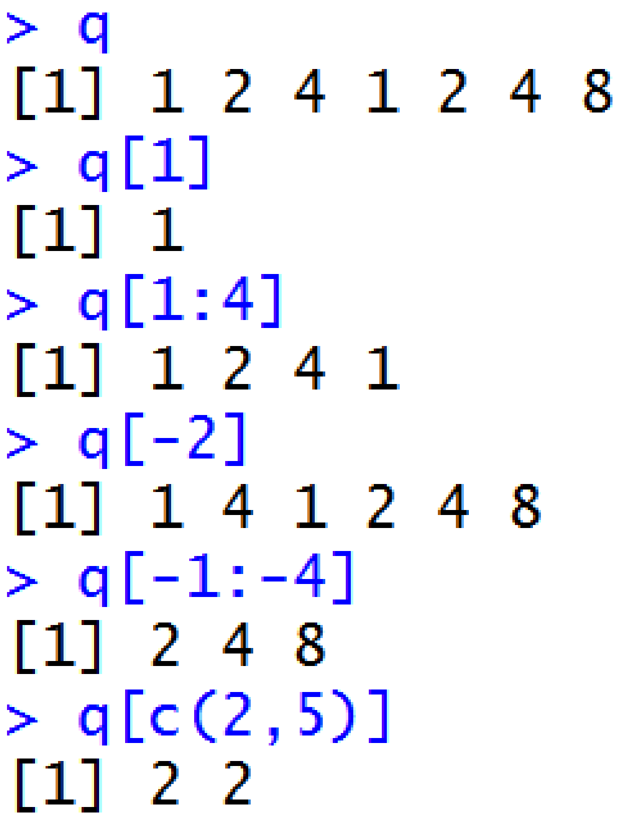
```
1+1

mean(runif(100))

1:10

2**10

abs(rnorm(10))
```

Variable assignment

```
x <- c(1,2,4)
```

- This function creates a vector of 1, 2 and 4
- There are no fixed types associated with variables
- `c()` is a function that concatenates three one-element vector
- The assignment operator in R are `<-` and =
- `<-` is a preferred assignment operator while = is used for parameter assignment
- Try this `q <- c(x,x,8)`

Variable printing and subsetting



```
> q
[1] 1 2 4 1 2 4 8
> q[1]
[1] 1
> q[1:4]
[1] 1 2 4 1
> q[-2]
[1] 1 4 1 2 4 8
> q[-1:-4]
[1] 2 4 8
> q[c(2,5)]
[1] 2 2
```

Print a variable q

Select an individual element of a vector, index 1

Subsetting range

Deselect

Deselect range

Selective subsetting

Calling functions

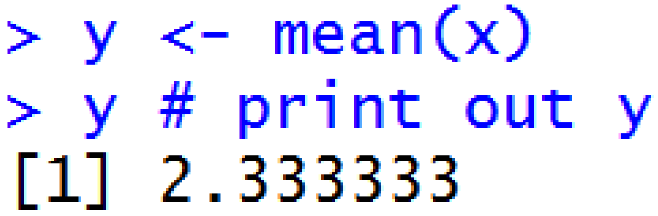
```
y <- mean(x)
y1 <- sd(x)
```

- The above code calculate mean and standard deviation of `x` and store them to `y` and `y1` respectively

Comment

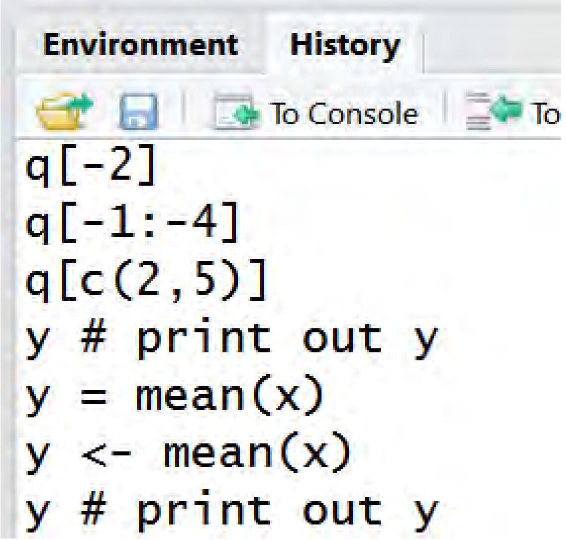
Try this

```
y # print out y
```



```
> y <- mean(x)
> y # print out y
[1] 2.333333
```

- Text after # is treated as comment
- They are not executed, but recorded in history

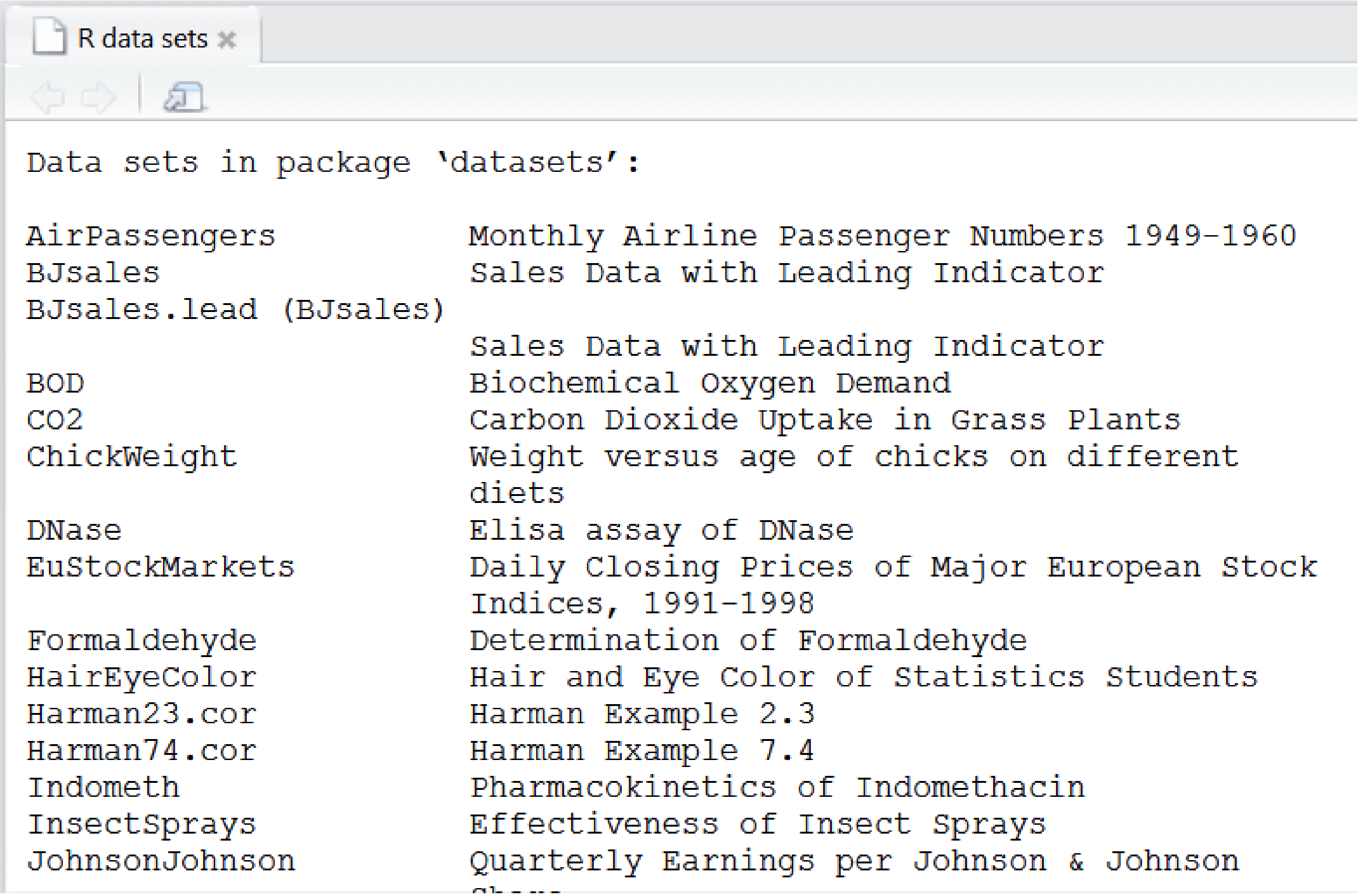


```
q[-2]
q[-1:-4]
q[c(2,5)]
y # print out y
y = mean(x)
y <- mean(x)
y # print out y
```

Internal Dataset

- R is preloaded with the internal datasets

- To view the whole list type `data()`



```
Data sets in package 'datasets':

AirPassengers            Monthly Airline Passenger Numbers 1949-1960
BJsales                  Sales Data with Leading Indicator
BJsales.lead (BJsales)
                         Sales Data with Leading Indicator
BOD                      Biochemical Oxygen Demand
CO2                      Carbon Dioxide Uptake in Grass Plants
ChickWeight              Weight versus age of chicks on different
                         diets
DNase                    Elisa assay of DNase
EuStockMarkets           Daily Closing Prices of Major European Stock
                         Indices, 1991-1998
Formaldehyde             Determination of Formaldehyde
HairEyeColor             Hair and Eye Color of Statistics Students
Harman23.cor             Harman Example 2.3
Harman74.cor             Harman Example 7.4
Indometh                 Pharmacokinetics of Indomethacin
InsectSprays             Effectiveness of Insect Sprays
JohnsonJohnson           Quarterly Earnings per Johnson & Johnson
                         Share
```

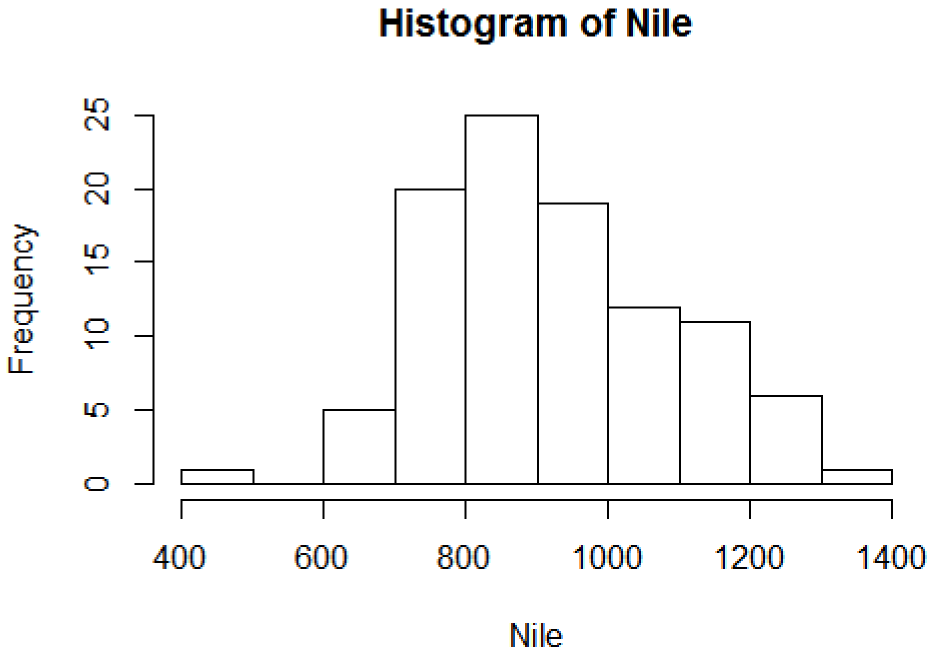
Internal Dataset

- You can load the internal dataset by typing the name
- Try the following

```
Nile
iris
mtcars
```

- To plot a histogram of Nile d

```
hist(Nile)
```



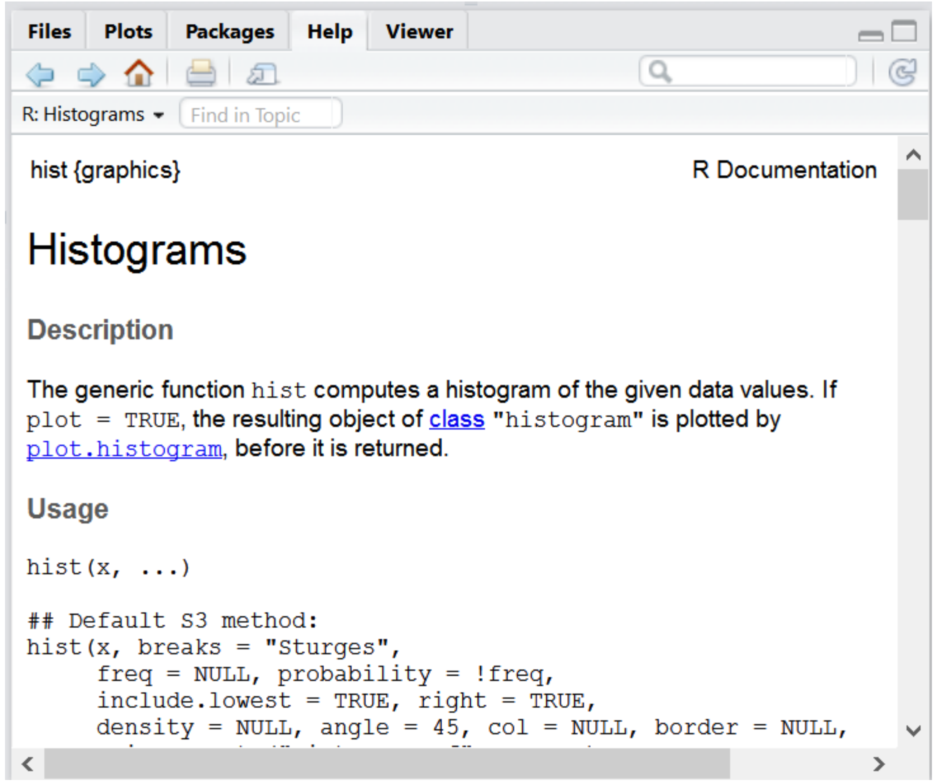
Help

- A skilled programmer always look for documentation
- To access the function, data or package documentations, try

```
help("hist")
```

or

```
?hist
?Nile
```



hist {graphics} R Documentation

Histograms

Description

The generic function `hist` computes a histogram of the given data values. If `plot = TRUE`, the resulting object of class `"histogram"` is plotted by `plot.histogram`, before it is returned.

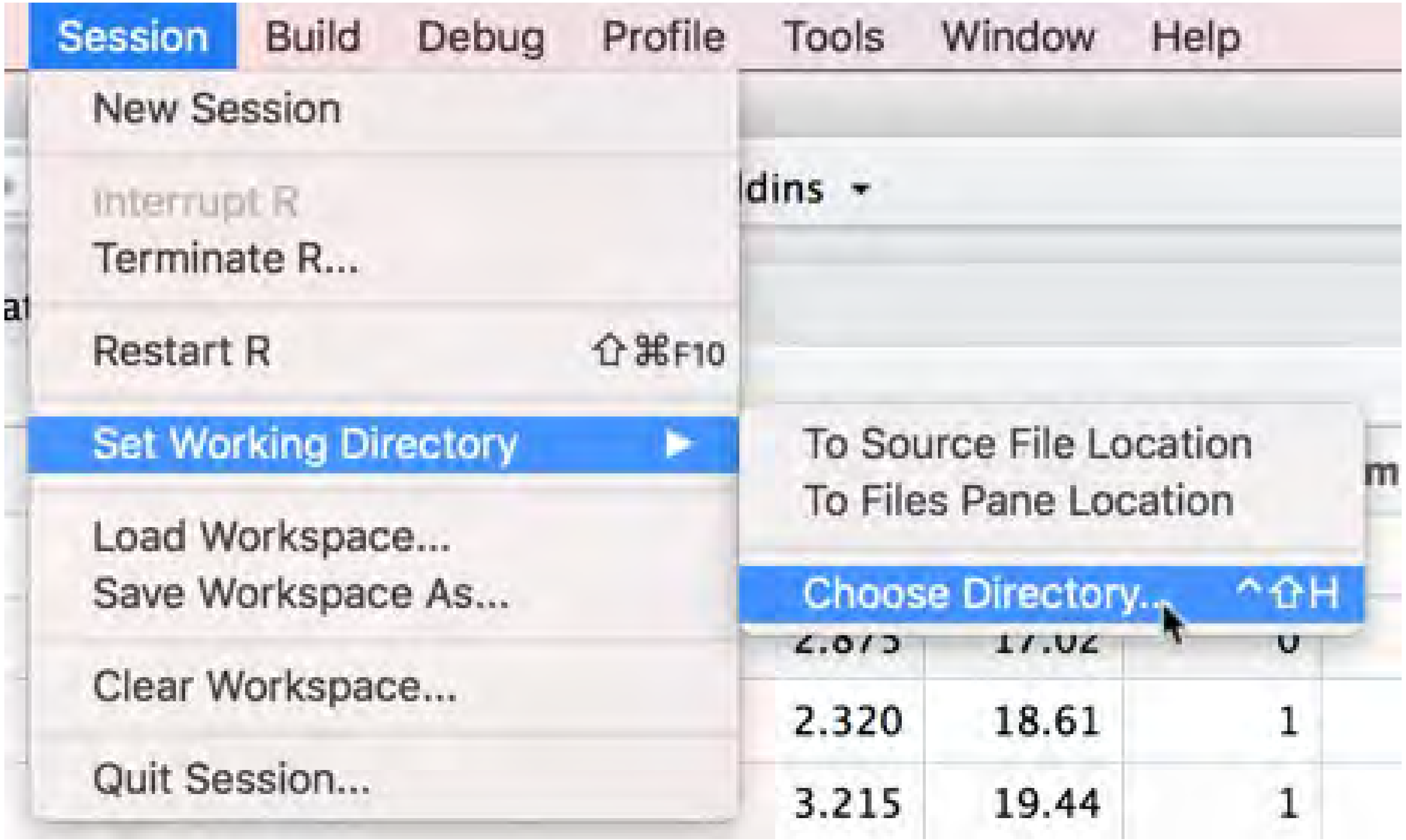
Usage

```
hist(x, ...)

## Default S3 method:
hist(x, breaks = "Sturges",
     freq = NULL, probability = !freq,
     include.lowest = TRUE, right = TRUE,
     density = NULL, angle = 45, col = NULL, border = NULL,
```

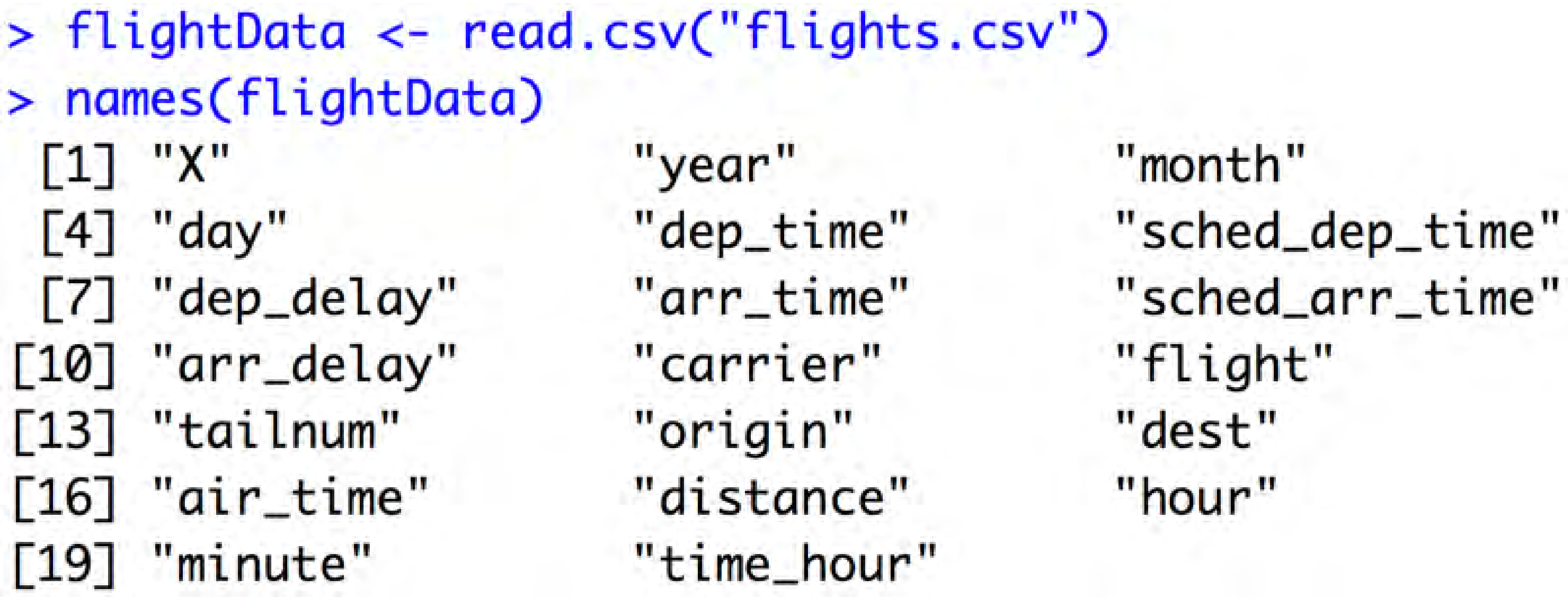
Working directory

- Working directory is the current directory that the console prompt is in.
- Working directory is like Linux or Windows command line current directory.
- R users need to set working directory in order to use the relative path or using the file directory without path.

Change working directory



Read Data

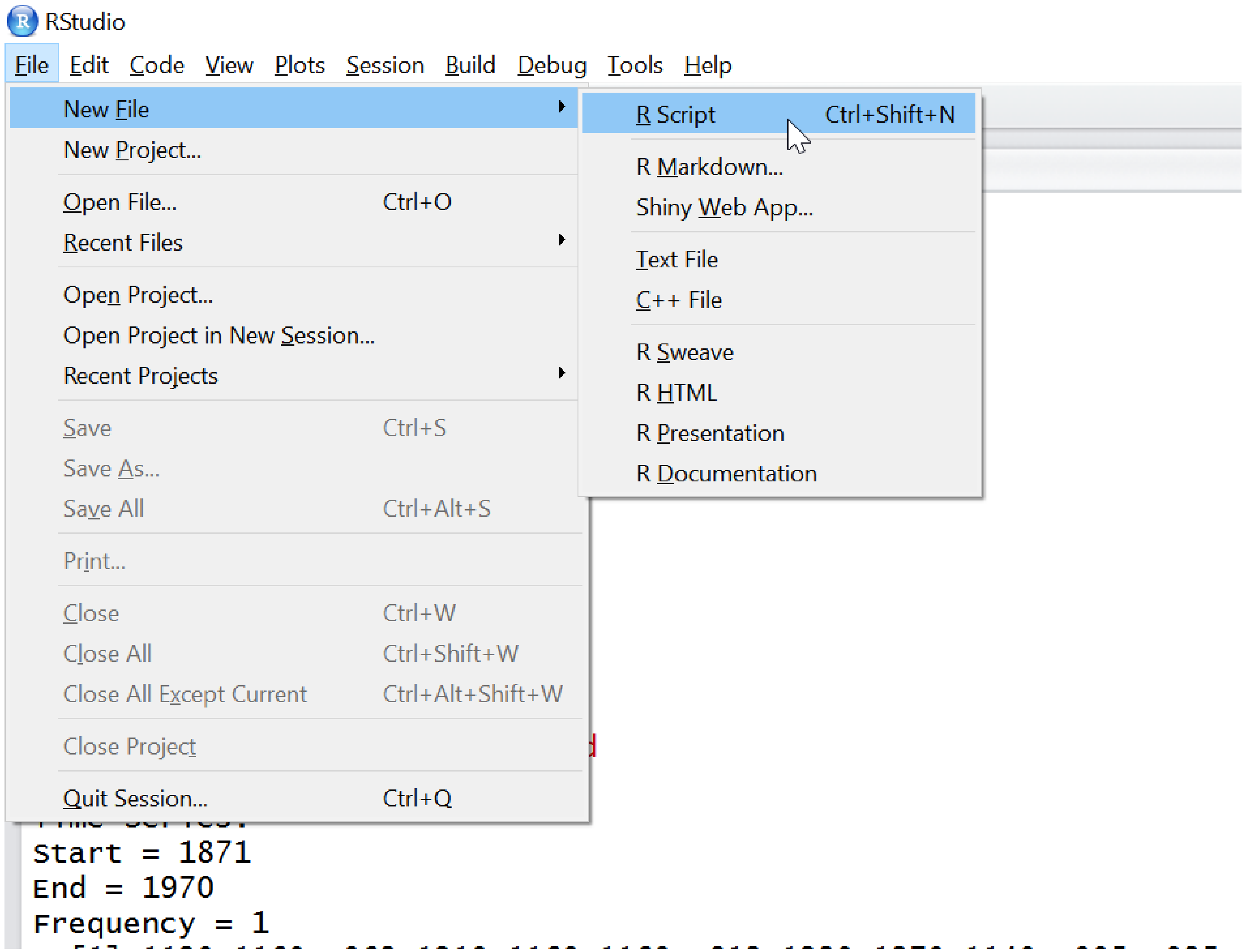


```
> flightData <- read.csv("flights.csv")
> names(flightData)
 [1] "X"              "year"           "month"
 [4] "day"            "dep_time"       "sched_dep_time"
 [7] "dep_delay"      "arr_time"       "sched_arr_time"
[10] "arr_delay"      "carrier"        "flight"
[13] "tailnum"        "origin"         "dest"
[16] "air_time"       "distance"       "hour"
[19] "minute"         "time_hour"
```

Introduction to function

- As in most programming languages, the heart of R programming consists of writing *functions*.
- A function is a group of instructions that takes inputs, uses them to compute other values, and returns a result.
- As a simple introduction, let’s define a function named oddcount(), whose purpose is to count the odd numbers in a vector of integers.
- We will write a function in RStudio editor, save it to file and load it to the workspace.

1. Create a new R script file

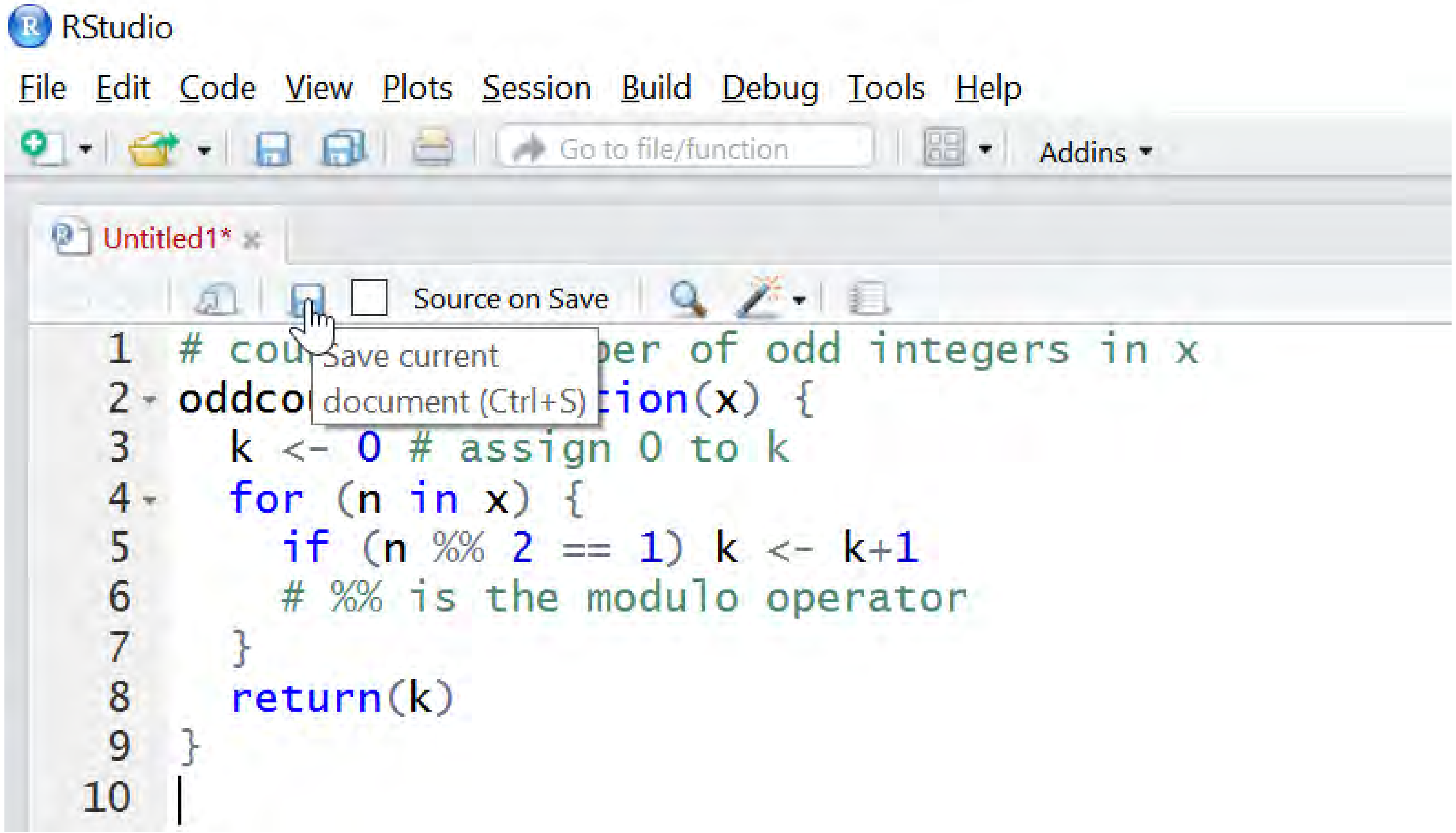


```
Time Series:
Start = 1871
End = 1970
Frequency = 1
```

2. Write the oddcount function

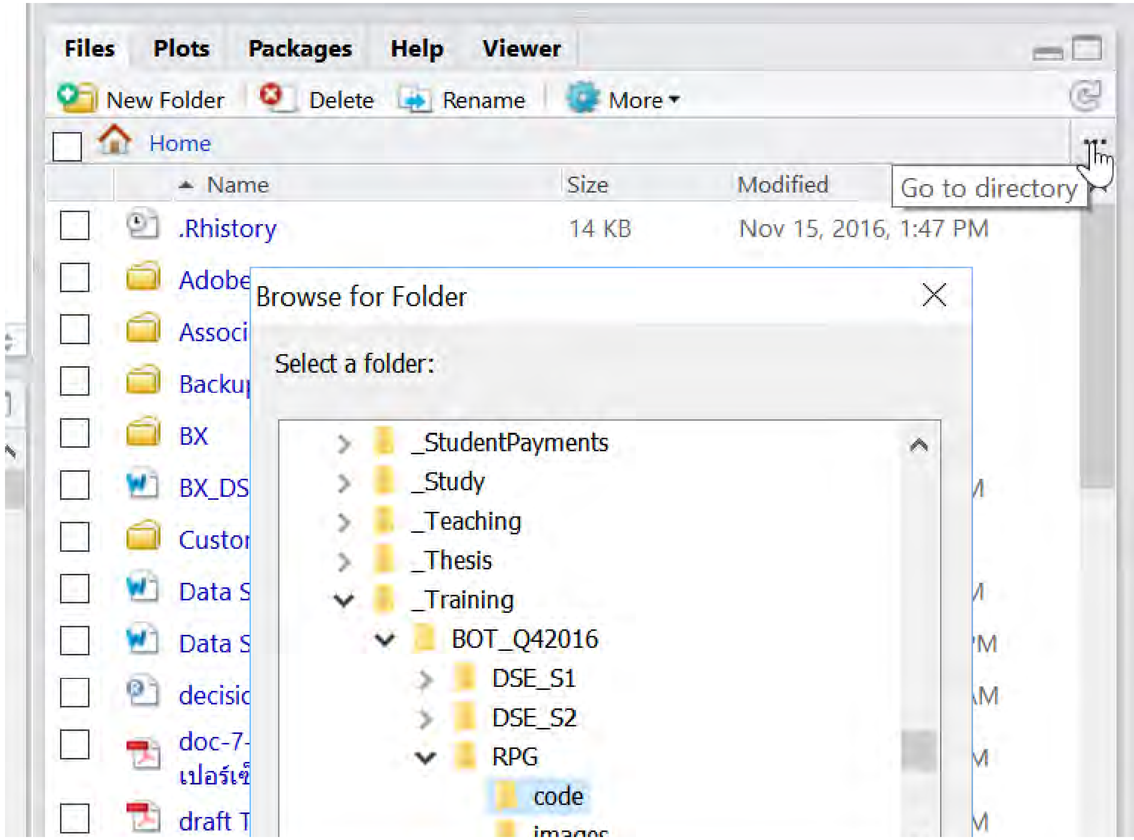
```
# counts the number of odd integers in x
oddcount <- function(x) {
  k <- 0 # assign 0 to k
  for (n in x) {
    if (n %% 2 == 1) k <- k+1
    # %% is the modulo operator
  }
  return(k)
}
```

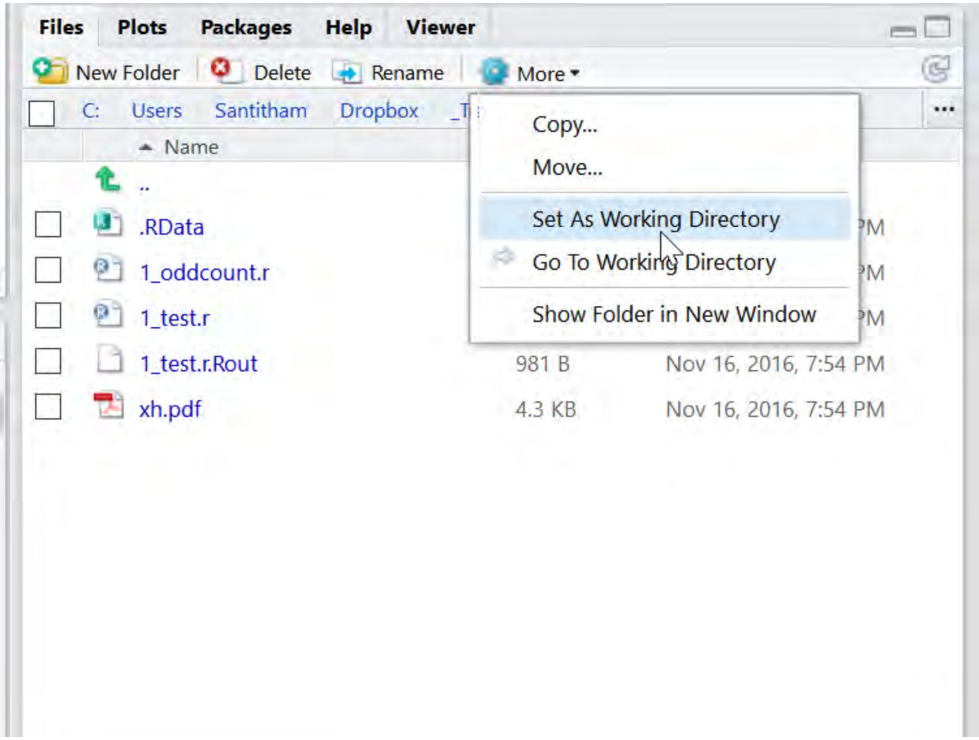
3. Save file with name “1_oddcount.r”


RStudio
File Edit Code View Plots Session Build Debug Tools Help
Go to file/function
Addins
Untitled1*
Source on Save
Save current document (Ctrl+S)

```
# cou                  ber of odd integers in x
oddco                  tion(x) {
  k <- 0 # assign 0 to k
  for (n in x) {
    if (n %% 2 == 1) k <- k+1
    # %% is the modulo operator
  }
  return(k)
}
```

4. Change working directory to where you save file





```
> setwd("C:/Users/Santitham/Dropbox/_Training/BOT_Q42016/RPG/code")
> |
```

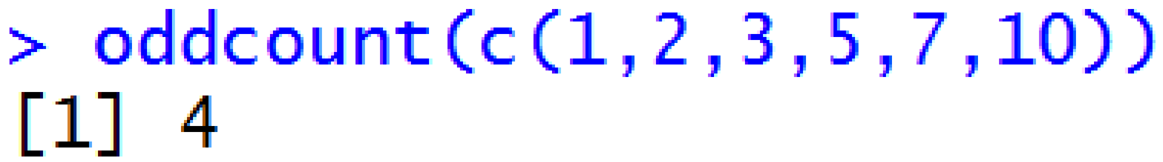
5. Load the function to use

To load the function

```
source("1_oddcount.r")
```

To use the function

```
oddcount(c(1,2,3,5,7,10))
```



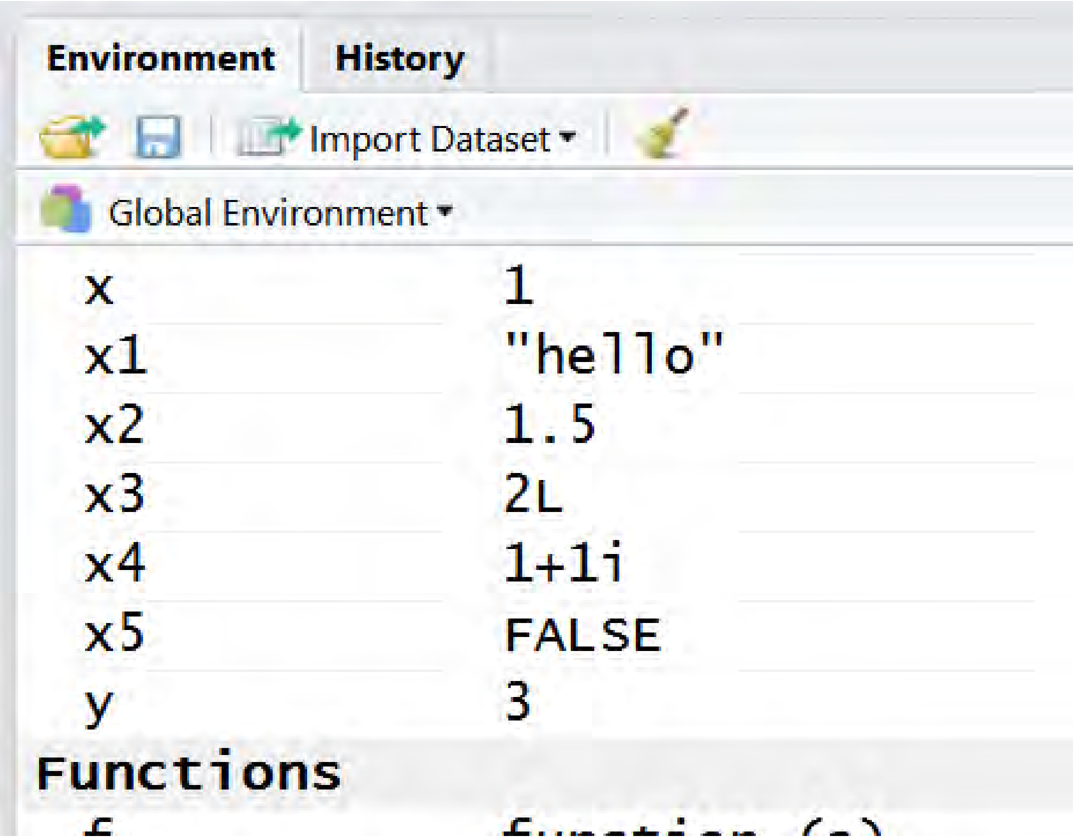
```
> oddcount(c(1,2,3,5,7,10))
[1] 4
```

R objects – basic classes

- R has five basic classes of objects
  - character
  - numeric (real numbers)
  - integer
  - complex
  - logical (True/False)
- Special values
  - Inf. Infinity occurs when the number is divided by zero
  - NaN. Not-a-number indicates undefined value, e.g. 0/0. It can also be thought as a missing value.

Try this

```
x1 <- 'hello'
x2 <- 1.5
x3 <- as.integer(2)
x4 <- 1+1i
x5 <- FALSE
```



```
x        1
x1       "hello"
x2       1.5
x3       2L
x4       1+1i
x5       FALSE
y        3
Functions
```

Compound data structure

- Vector
- Matrices and Arrays
- Lists
- Data Frames
- Factors and Tables

Vectors

- Vector is a basic foundation of other data structure

```
x <- 1
```

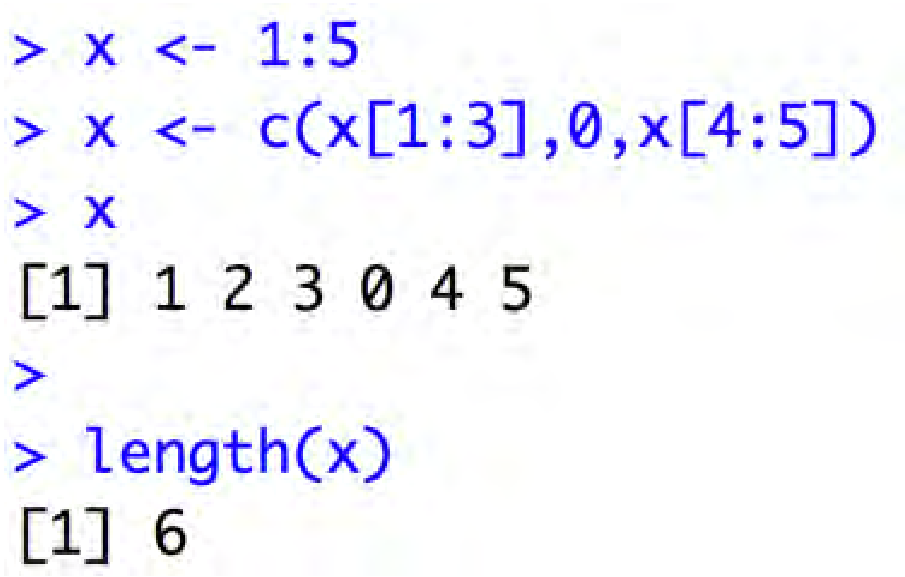
- This generates one-element vector

```
x <- c(2,4,6)
```

- This generates three-elements vector

Adding or deleting vector elements

- Vectors are stored contiguously, thus you cannot insert or delete elements
- The size of a vector is determined at its creation, so if you wish to add or delete elements, you’ll need to reassign the vector.



```
> x <- 1:5
> x <- c(x[1:3],0,x[4:5])
> x
[1] 1 2 3 0 4 5
>
> length(x)
[1] 6
```

Common vector operation
Arithmetic operator

- Adding/subtracting

```
x <- c(1,2,4)
x + c(5,0,1)
x - 1
x + c(2,1)
```

- (Element-wise) multiplication

```
x * c(5, 0, 4)
```

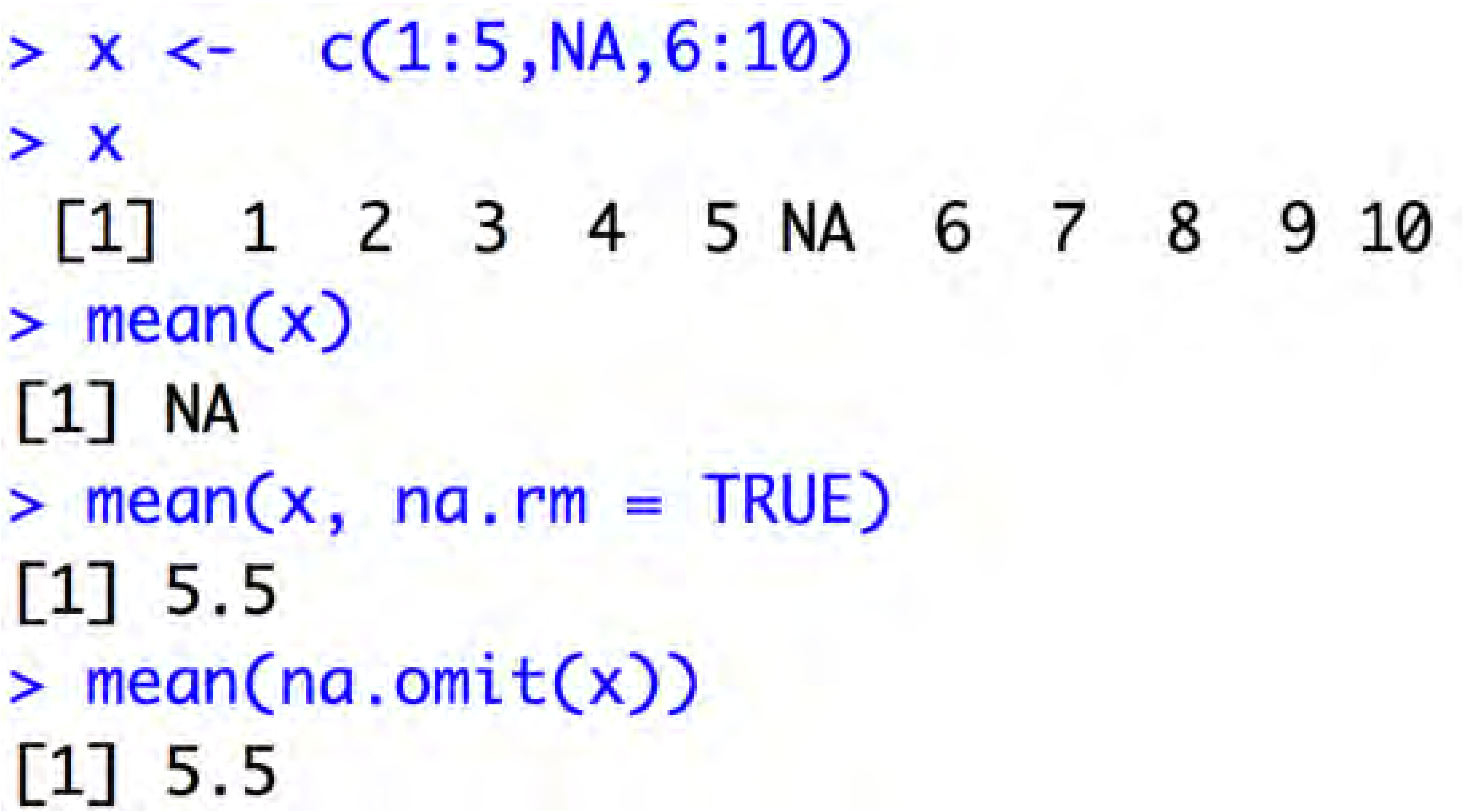
- (Element-wise) division

```
x / c(5, 4, -1)
```

- (Element-wise) modulus

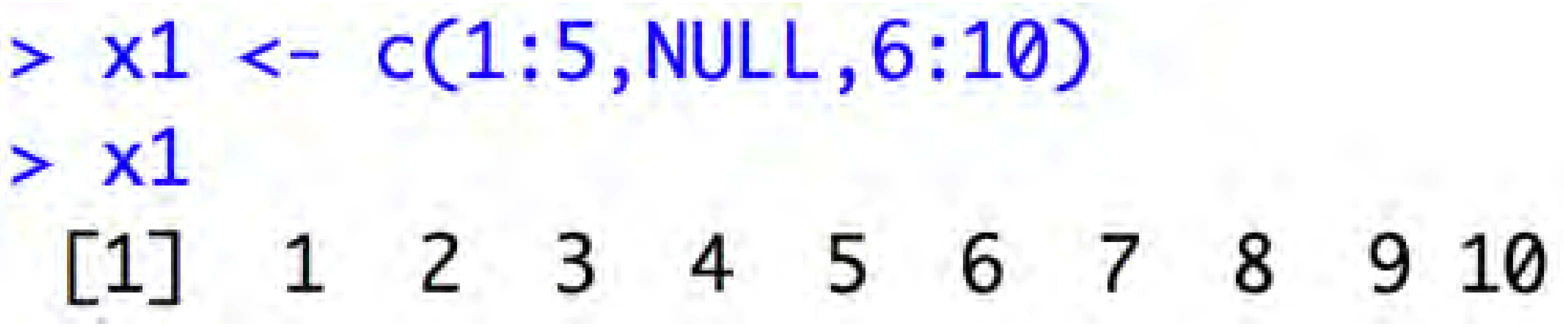
```
x %% c(5,4,-1)
```

NA: missing data



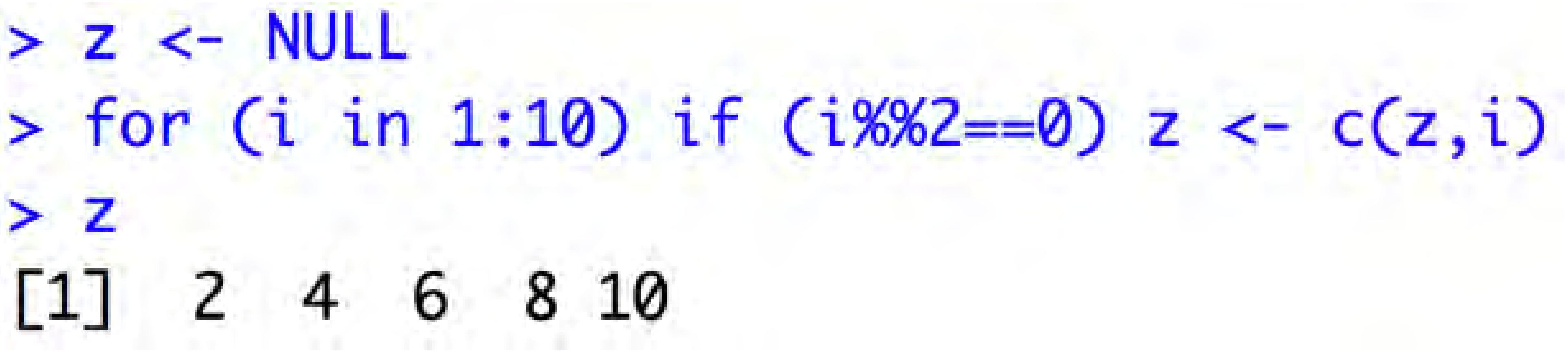
```
> x <-  c(1:5,NA,6:10)
> x
 [1]  1  2  3  4  5 NA  6  7  8  9 10
> mean(x)
[1] NA
> mean(x, na.rm = TRUE)
[1] 5.5
> mean(na.omit(x))
[1] 5.5
```

NULL: nothing, empty set



```
> x1 <- c(1:5,NULL,6:10)
> x1
 [1]  1  2  3  4  5  6  7  8  9 10
```

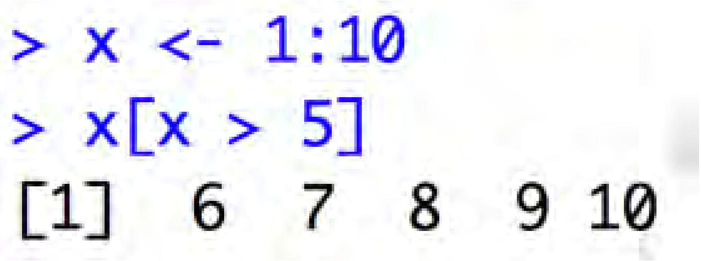
- NULL can be used as an initial empty variable



```
> z <- NULL
> for (i in 1:10) if (i%%2==0) z <- c(z,i)
> z
[1]  2  4  6  8 10
```

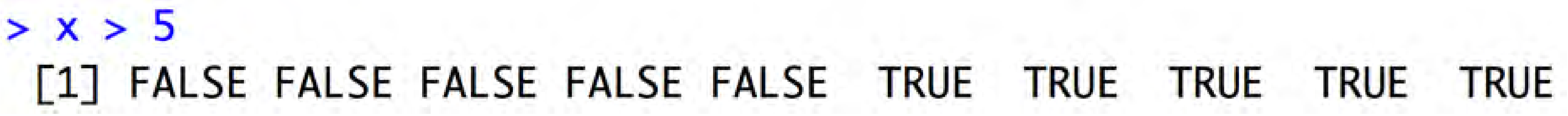
Filtering

- Filtering allows us to extract a vector’s elements that satisfy certain conditions.



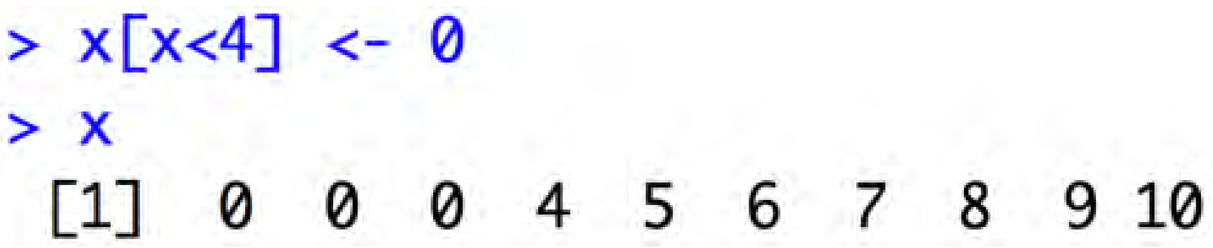
```
> x <- 1:10
> x[x > 5]
[1]  6  7  8  9 10
```

- This is because the index vector become a logical vector



```
> x > 5
 [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

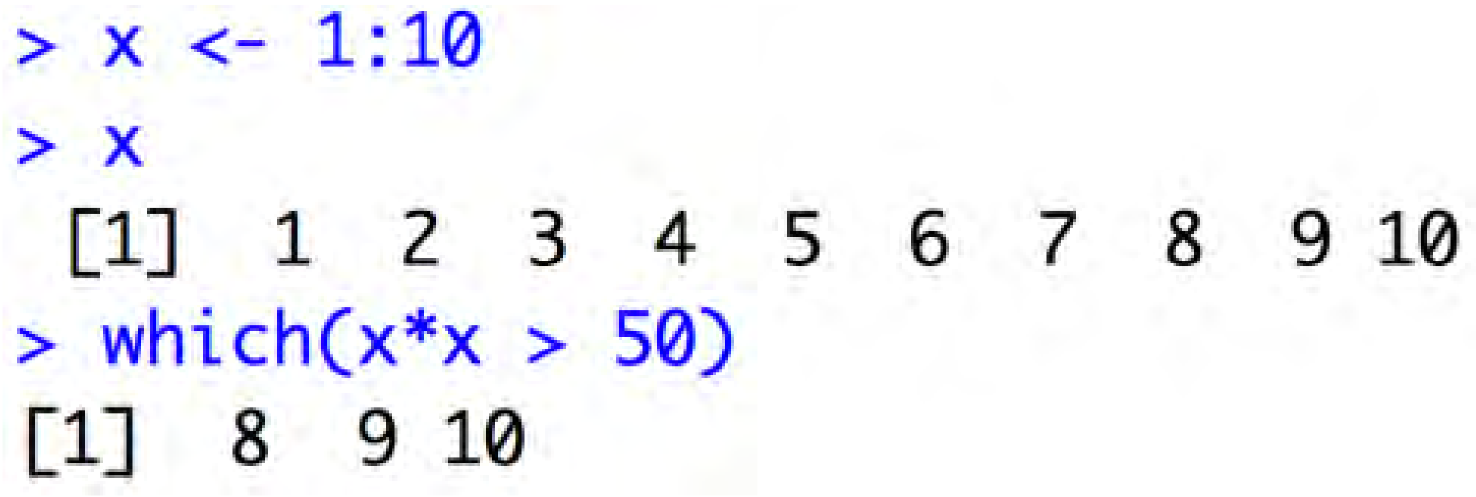
- We can assign value to the filtered position



```
> x[x<4] <- 0
> x
 [1]  0  0  0  4  5  6  7  8  9 10
```

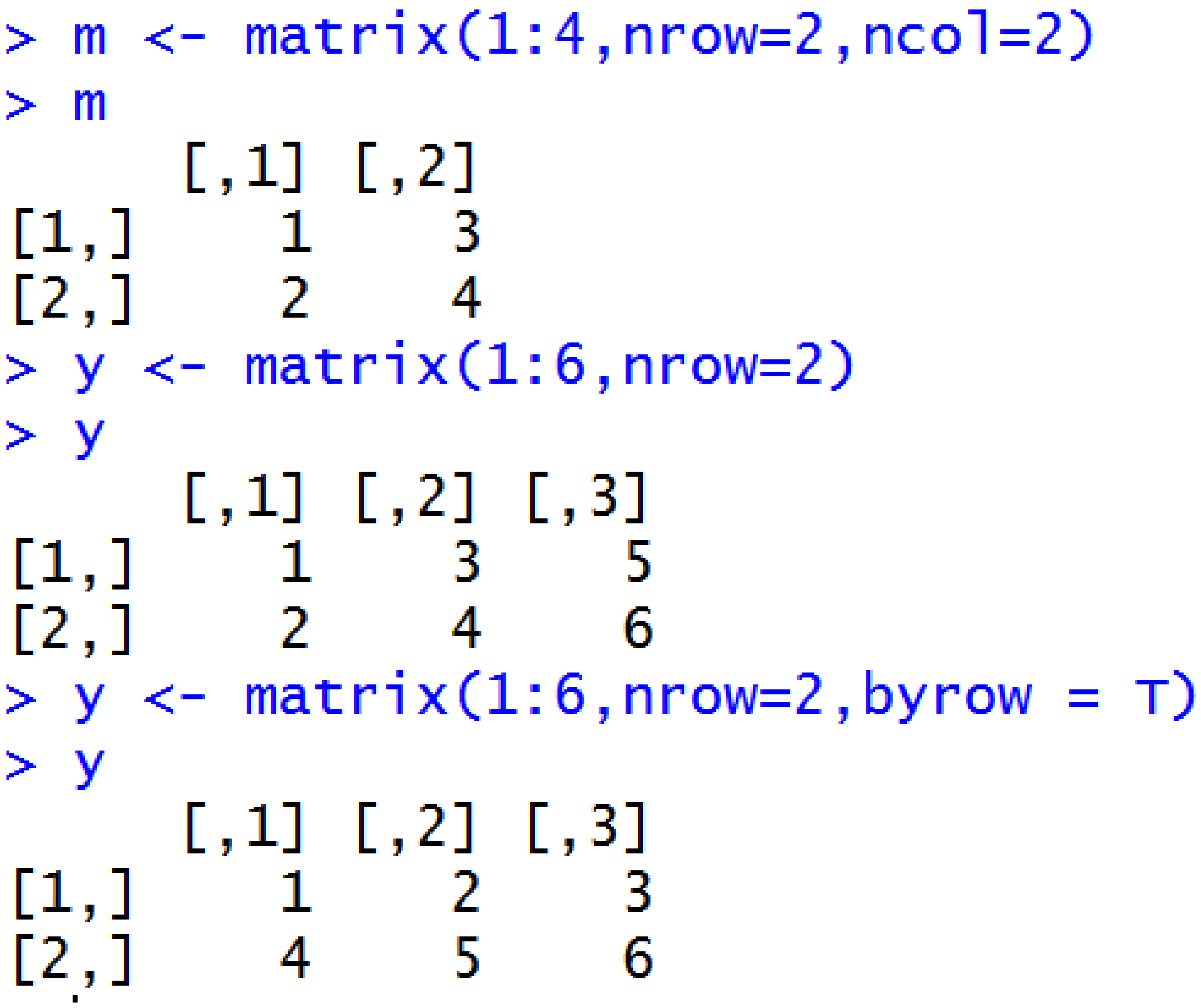
Selection function which()

- In some cases, though, we may just want to find the positions within a vector at which the condition occurs.
- We can do this using `which()`, as follows:



```
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> which(x*x > 50)
[1]  8  9 10
```

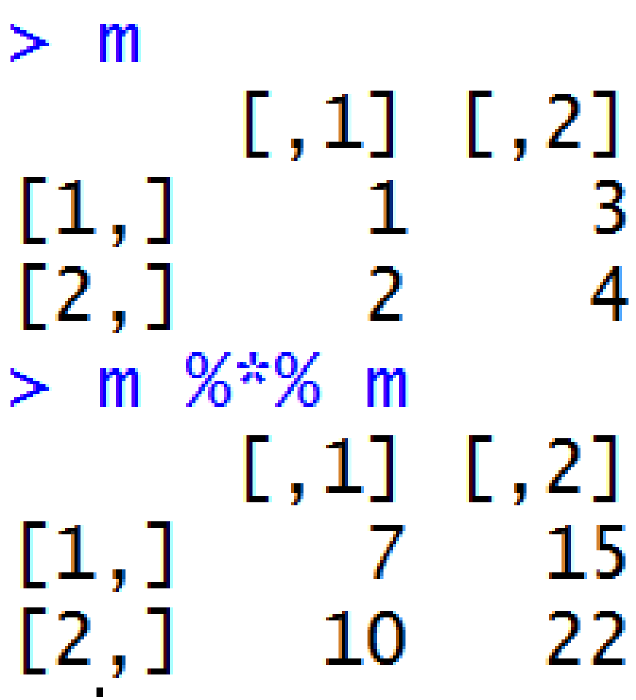
Matrix Creating matrix



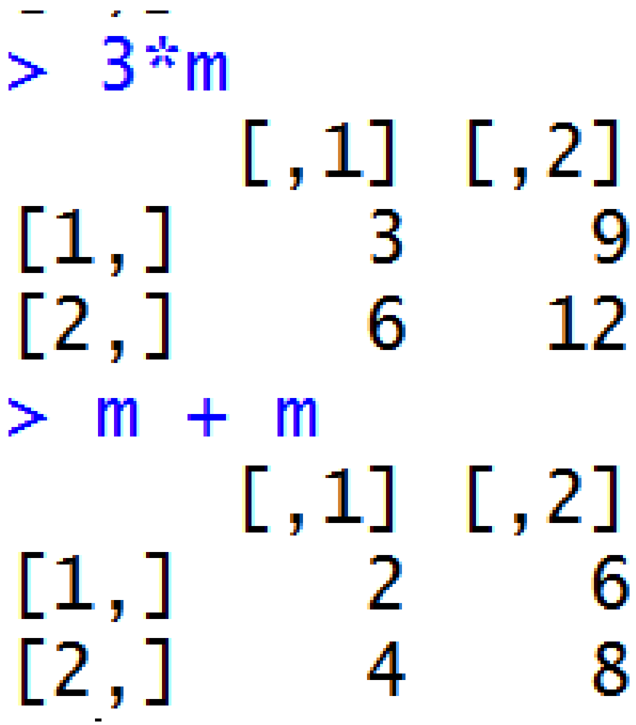
```
> m <- matrix(1:4,nrow=2,ncol=2)
> m
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> y <- matrix(1:6,nrow=2)
> y
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> y <- matrix(1:6,nrow=2,byrow = T)
> y
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Matrix operation

Matrix Multiplication by Scalar



```
> m
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> m %*% m
     [,1] [,2]
[1,]    7   15
[2,]   10   22
```

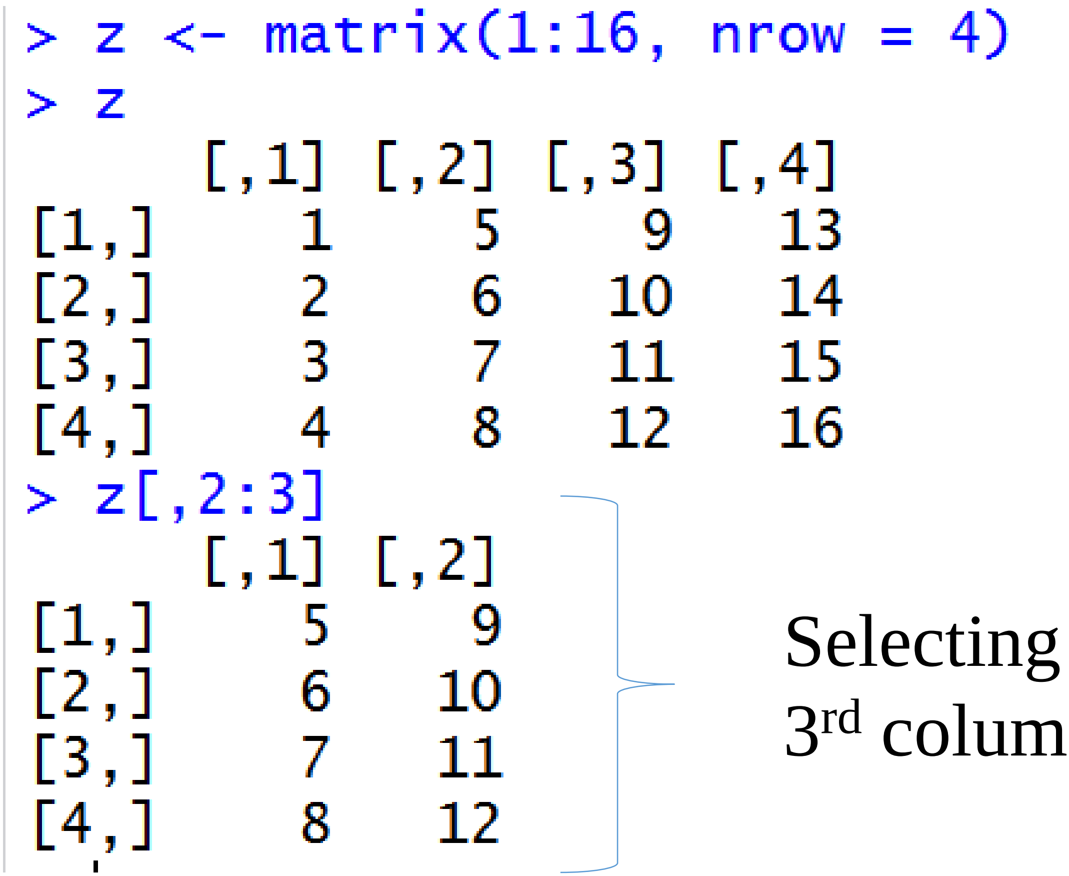


```
> 3*m
     [,1] [,2]
[1,]    3    9
[2,]    6   12
> m + m
     [,1] [,2]
[1,]    2    6
[2,]    4    8
```

Matrix Multiplication

Matrix Addition

Matrix indexing

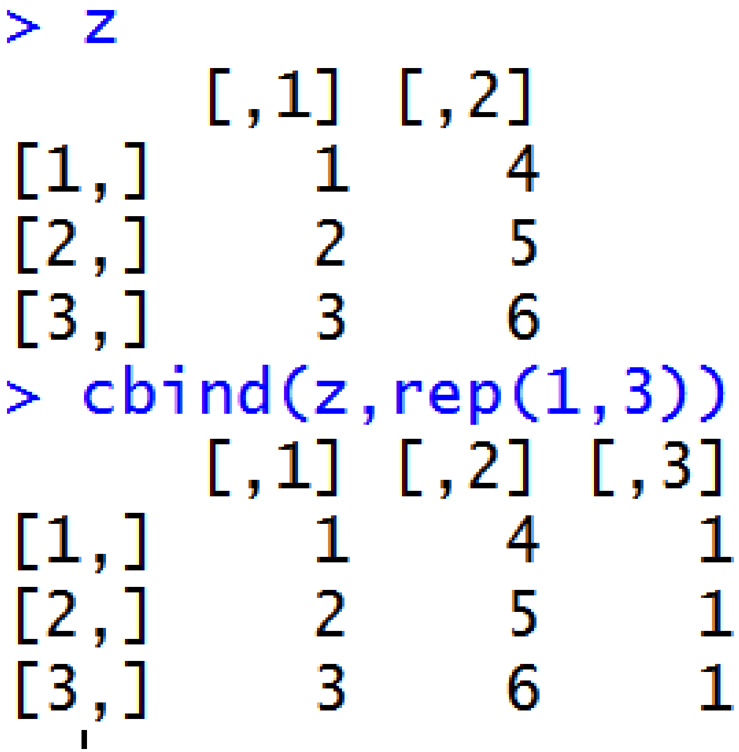


```
> z <- matrix(1:16, nrow = 4)
> z
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
> z[,2:3]
     [,1] [,2]
[1,]    5    9
[2,]    6   10
[3,]    7   11
[4,]    8   12
```

Selecting only 2nd and 3rd columns

Adding or deleting matrix rows and columns

- Just like vector, matrices are fixed length and dimension
- However, they can be reassigned
- cbind() bind the column together



```
> z
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> cbind(z,rep(1,3))
     [,1] [,2] [,3]
[1,]    1    4    1
[2,]    2    5    1
[3,]    3    6    1
```

Object coercion

If more than one class are introduced, the vector will be coerced.

Implicit

```
> y <- c(1.7, "a")
## character
> y <- c(TRUE, 2)
## numeric
> y <- c("a", TRUE)
## character
```

Explicit

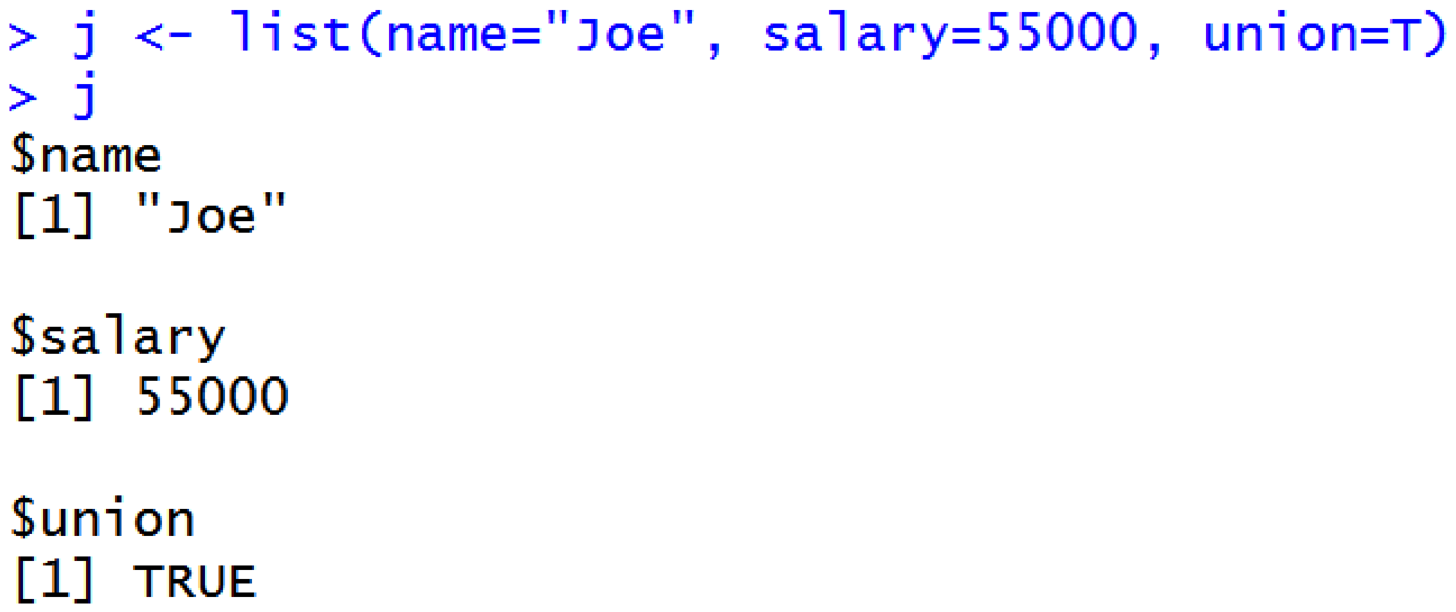
```
> x <- 0:6
> class(x)
 [1] "integer"
> as.numeric(x)
 [1] 0 1 2 3 4 5 6
> as.logical(x)
 [1] FALSE TRUE TRUE TRUE TRUE TRUE TRUE
> as.character(x)
 [1] "0" "1" "2" "3" "4" "5" "6"
```

List

- In contrast to a vector, in which all elements must be of the same mode, R’s list structure can combine objects of different types.
- Ordinary vectors are termed *atomic* vectors, since their components cannot be broken down into smaller components.
- In contrast, lists are referred to as *recursive* vectors.

Creating a list

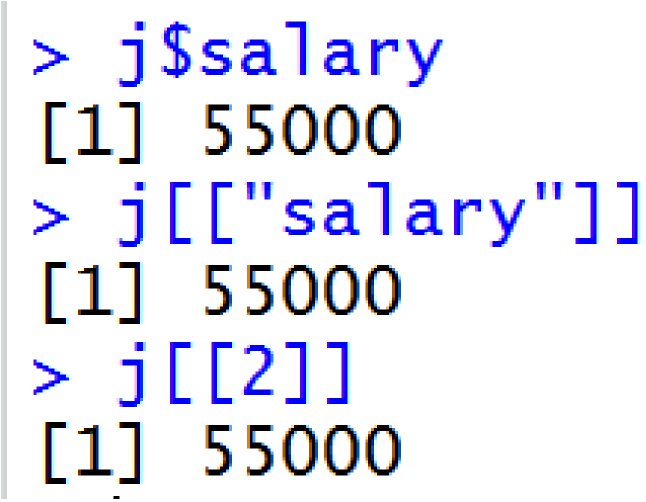
- Let us consider an employee database.
- For each employee, we wish to store the name, salary, and a Boolean indicating union membership.



```
> j <- list(name="Joe", salary=55000, union=T)
> j
$name
[1] "Joe"

$salary
[1] 55000

$union
[1] TRUE
```

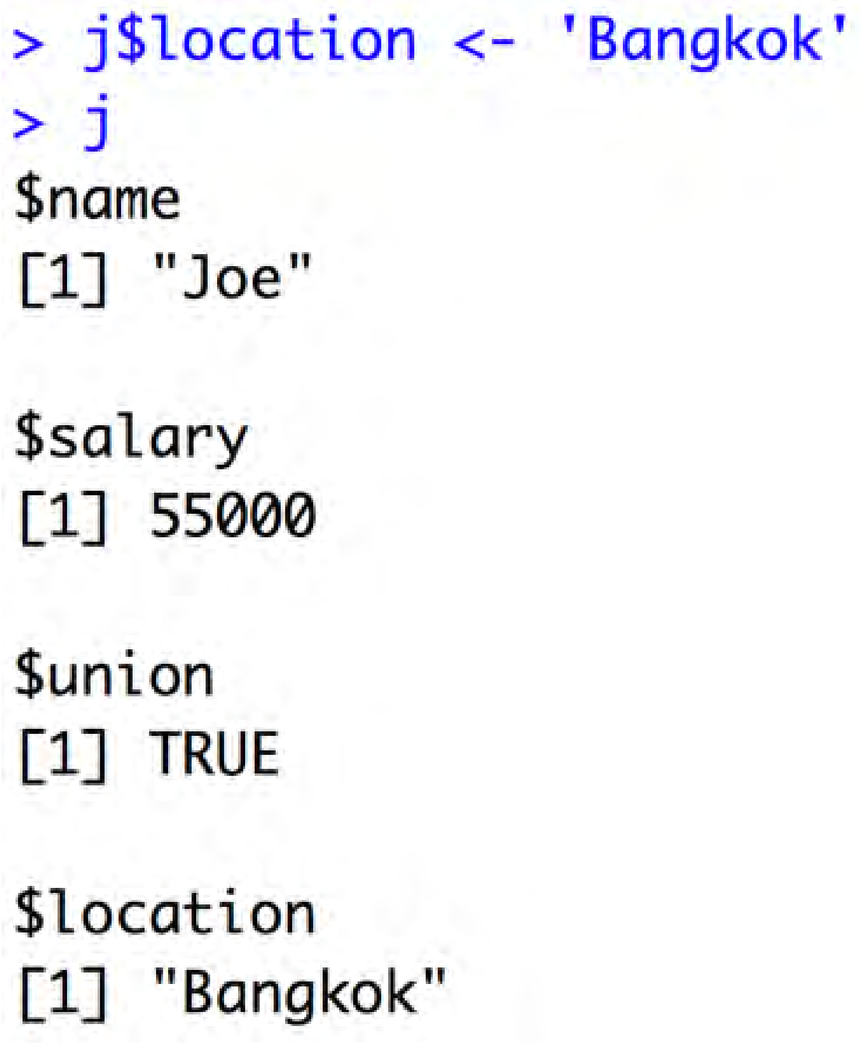
List indexing

- You can access a list component in several different ways:



```
> j$salary
[1] 55000
> j[["salary"]]
[1] 55000
> j[[2]]
[1] 55000
```

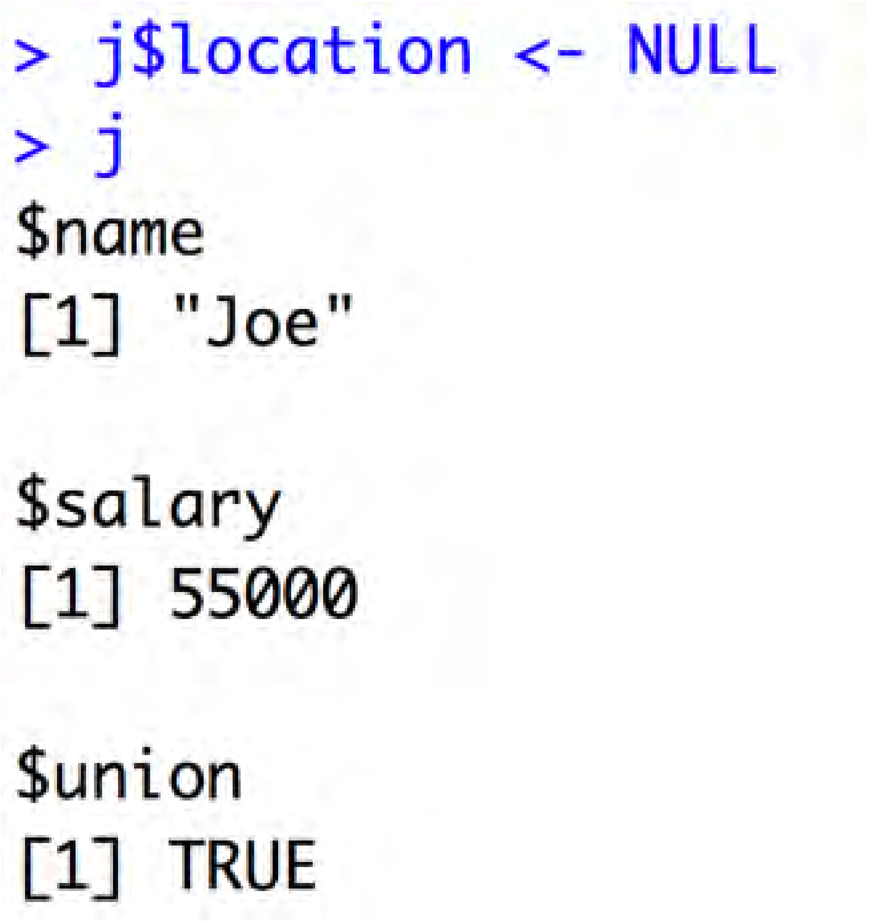
Adding list elements



```
> j$location <- 'Bangkok'
> j
$name
[1] "Joe"

$salary
[1] 55000

$union
[1] TRUE

$location
[1] "Bangkok"
```

Element can be added directly

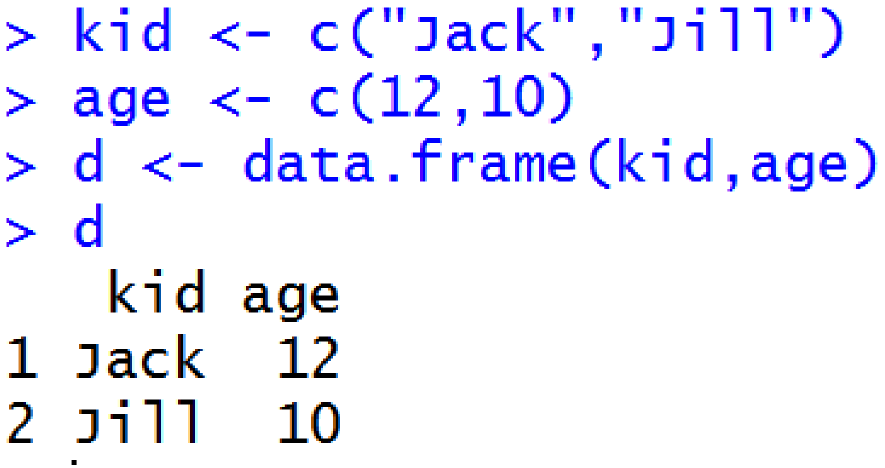
Deleting list elements



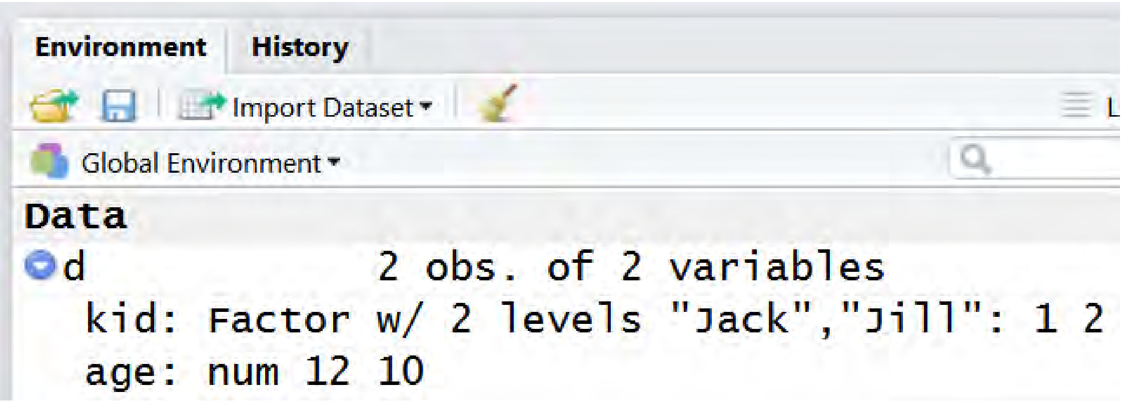
```
> j$location <- NULL
> j
$name
[1] "Joe"

$salary
[1] 55000

$union
[1] TRUE
```

Data Frames

- On an intuitive level, a *data frame* is like a matrix, with a two-dimensional rows and columns structure.
- On a technical level, a data frame is a list, with the components of that list being equal-length vectors.

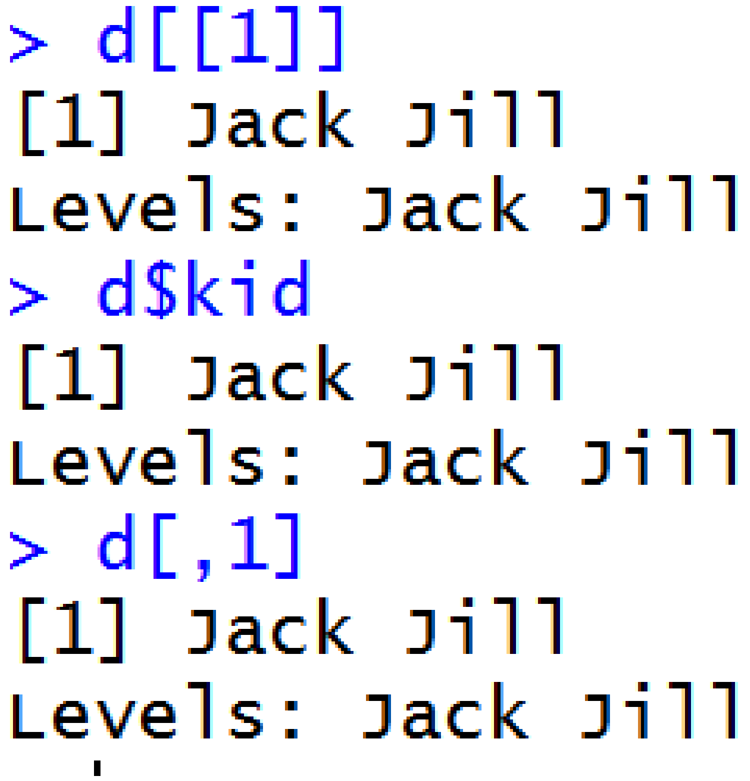


```
> kid <- c("Jack","Jill")
> age <- c(12,10)
> d <- data.frame(kid,age)
> d
   kid age
1 Jack  12
2 Jill  10
```



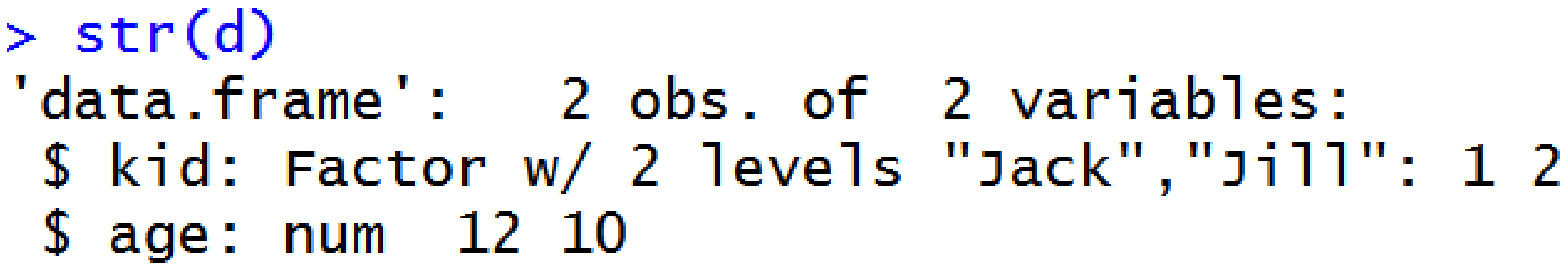
```
Data
d             2 obs. of 2 variables
  kid: Factor w/ 2 levels "Jack","Jill": 1 2
  age: num 12 10
```

Accessing data frame



```
> d[[1]]
[1] Jack Jill
Levels: Jack Jill
> d$kid
[1] Jack Jill
Levels: Jack Jill
> d[,1]
[1] Jack Jill
Levels: Jack Jill
```

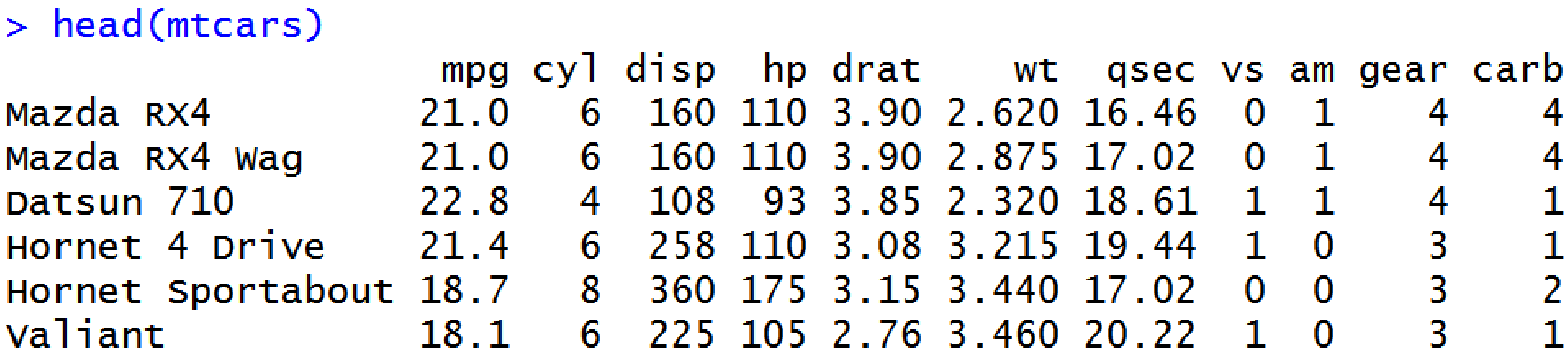
Object internal structure



```
> str(d)
'data.frame':   2 obs. of  2 variables:
 $ kid: Factor w/ 2 levels "Jack","Jill": 1 2
 $ age: num  12 10
```

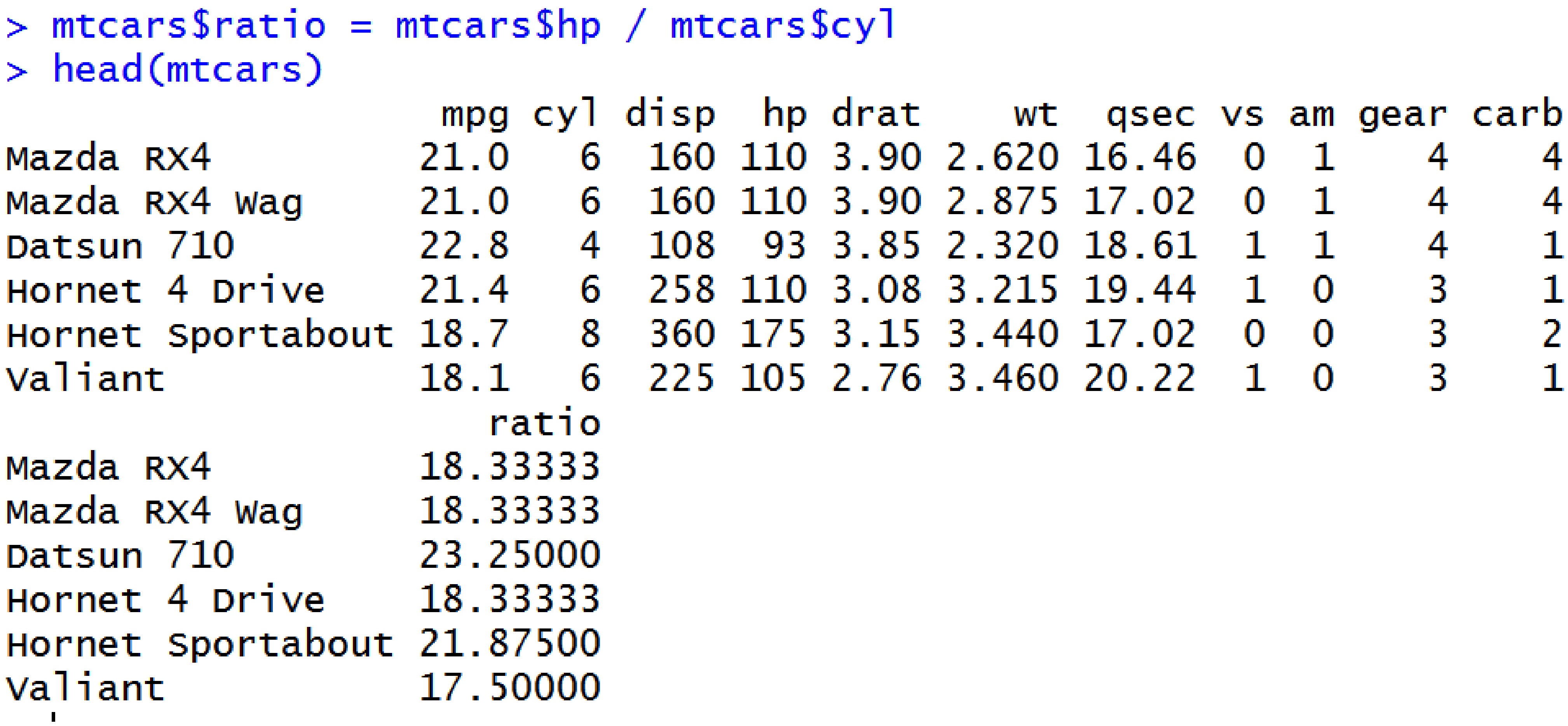
Example: mtcars

- You can view the top rows of the data frame by using head()



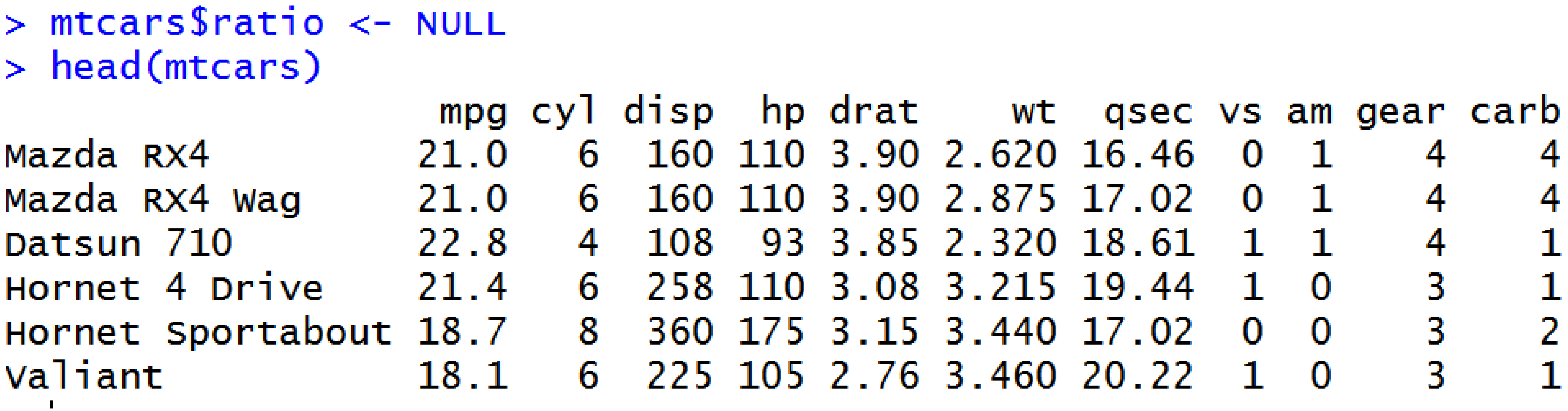
```
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Adding column to data frame



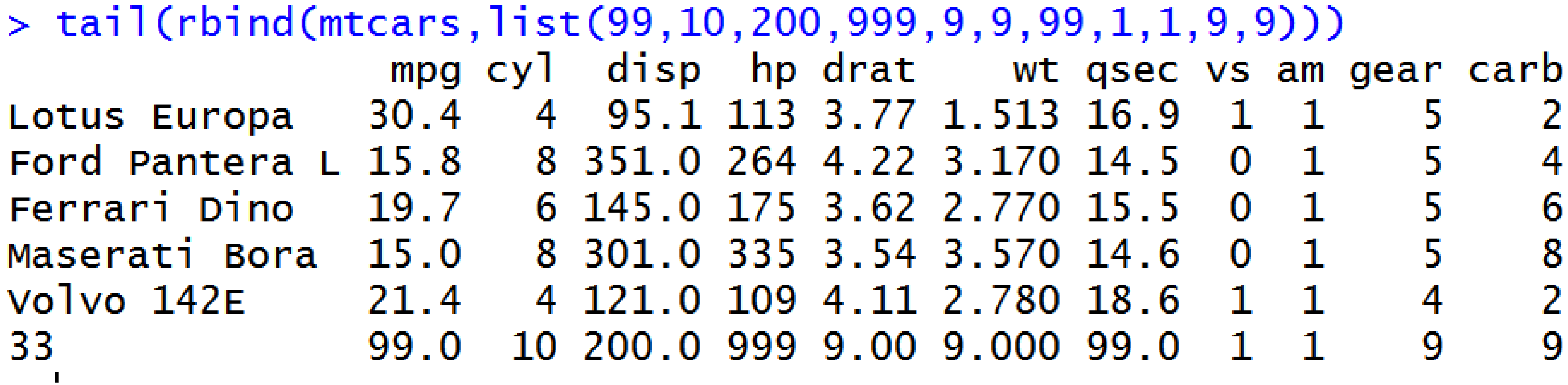
```
> mtcars$ratio = mtcars$hp / mtcars$cyl
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
                     ratio
Mazda RX4         18.33333
Mazda RX4 Wag     18.33333
Datsun 710        23.25000
Hornet 4 Drive    18.33333
Hornet Sportabout 21.87500
Valiant           17.50000
```

Delete column from data frame



```
> mtcars$ratio <- NULL
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Adding row to data frame



```
> tail(rbind(mtcars,list(99,10,200,999,9,9,99,1,1,9,9)))
                mpg cyl  disp  hp drat    wt qsec vs am gear carb
Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
33             99.0  10 200.0 999 9.00 9.000 99.0  1  1    9    9
```

rbind() produces a new data frame with additional row

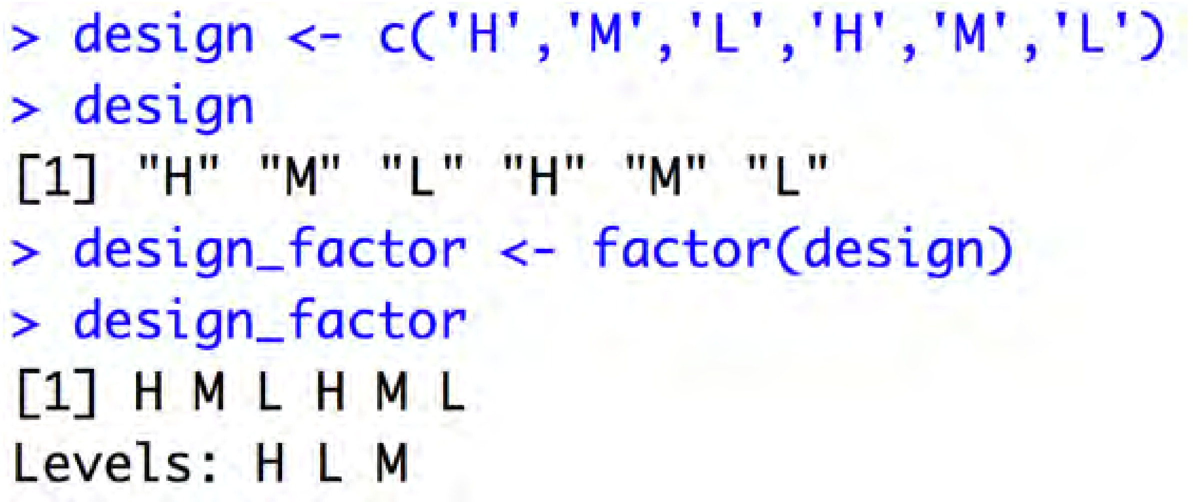
tail() function can be used to view the end of data frame

Factors

- Factors form the basis for many of R’s powerful operations, including many of those performed on tabular data.
- The motivation for factors comes from the notioncof *nominal*, or *categorical*, variables in statistics.
- These values are nonnumerical in nature, corresponding to categories such as RENT, MORTGAGE, OWN, although they may be coded using numbers.
- Statistical model relies on factors (and levels) to indicate group of data to be analyzed

Factors and levels

- An R *factor* might be viewed simply as a vector with a bit more information added (though, as seen below, it’s different from this internally).
- That extra information consists of a record of the distinct values in that vector, called *levels*. Here’s an example:



```
> design <- c('H','M','L','H','M','L')
> design
[1] "H" "M" "L" "H" "M" "L"
> design_factor <- factor(design)
> design_factor
[1] H M L H M L
Levels: H L M
```

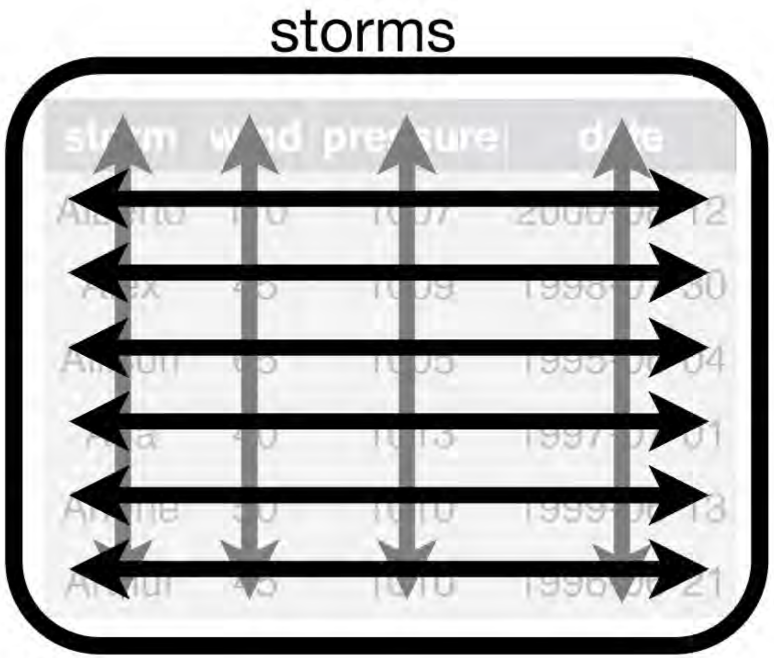
Data Preparation

Section 2

Two packages to help you work with the structure of data.

- tidyr
- dplyr

Tidy data

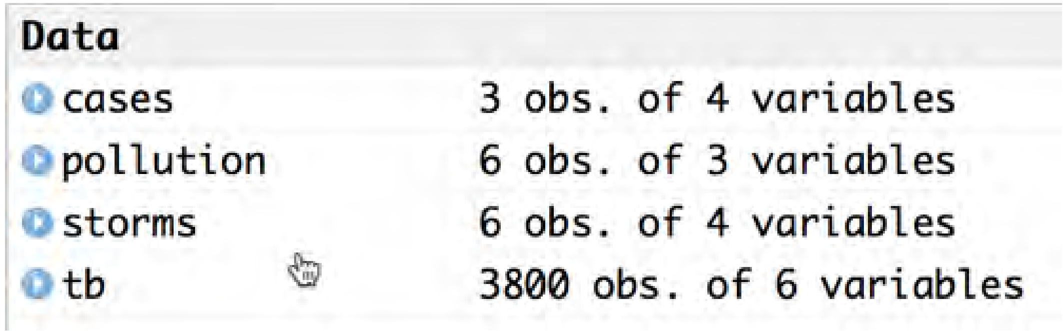


1 Each **variable** is saved in its own **column**.

2 Each **observation** is saved in its own **row**.

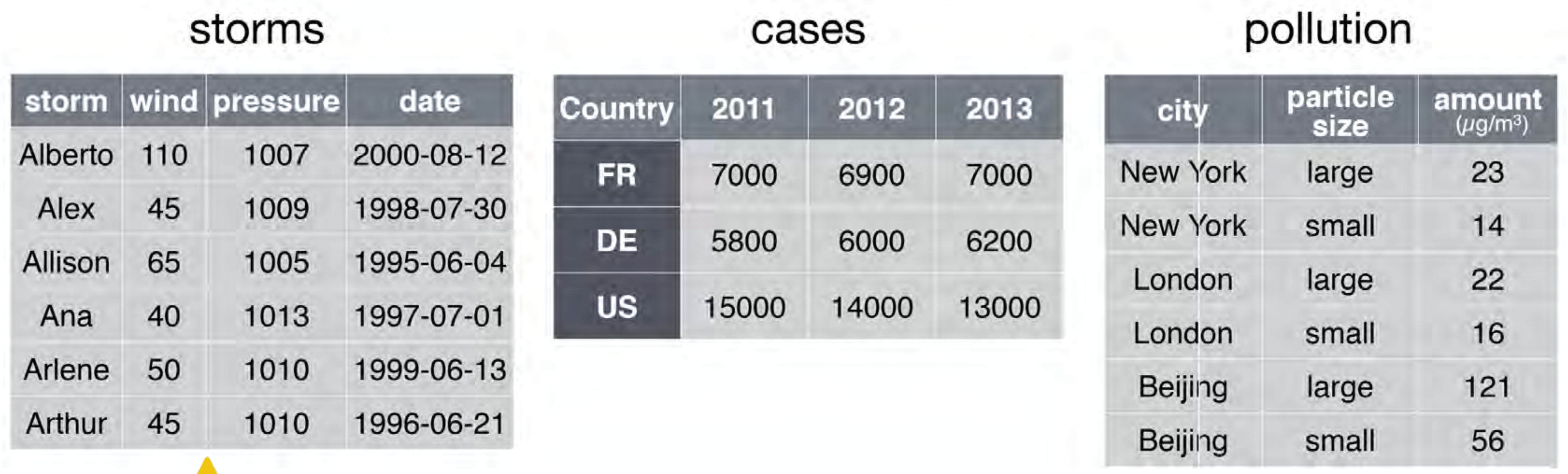
3 Each "type" of observation stored in a **single table** (here, storms).

Load data



```
Data
cases        3 obs. of 4 variables
pollution    6 obs. of 3 variables
storms       6 obs. of 4 variables
tb           3800 obs. of 6 variables
```

load("2-DataPrep.RData")



storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

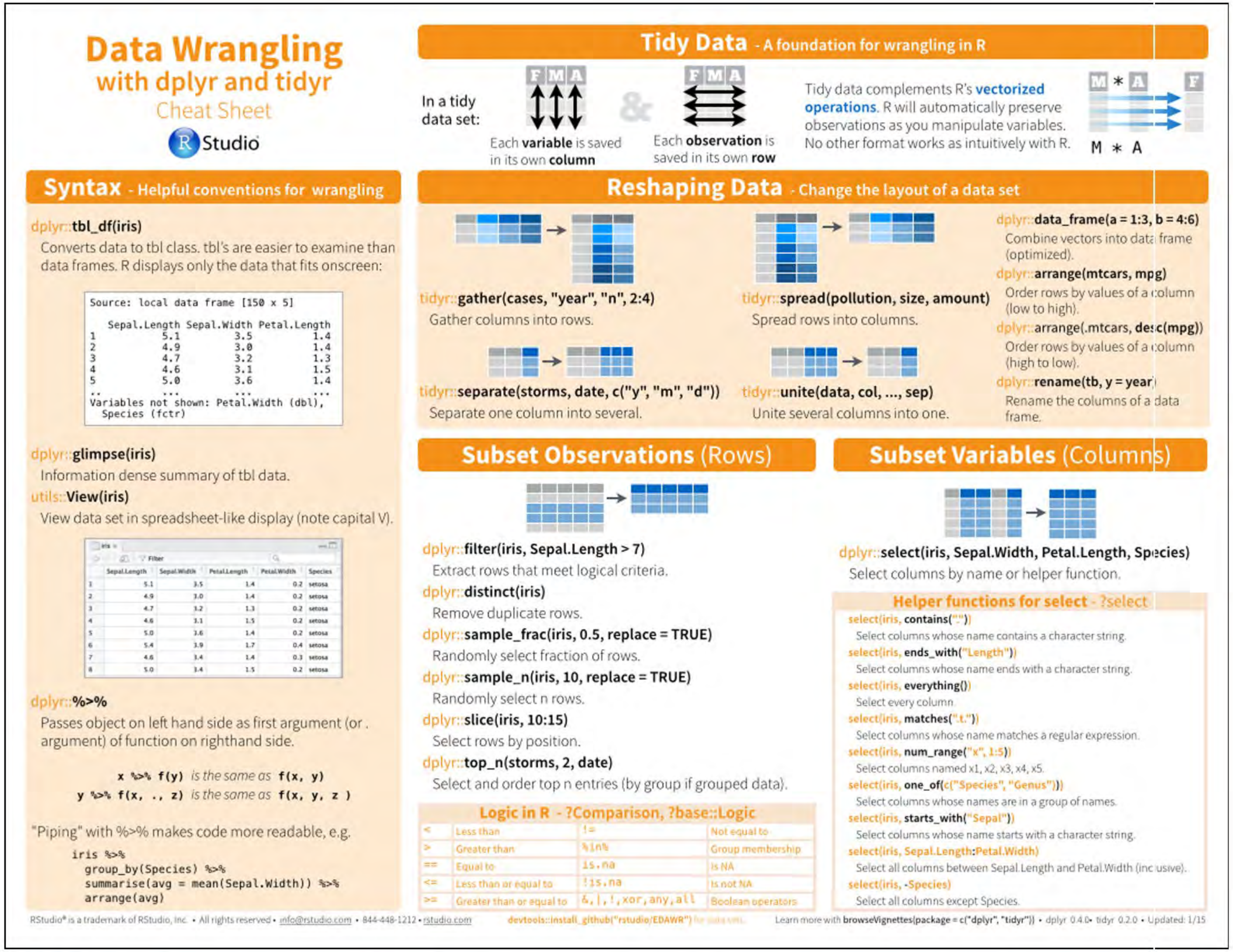
cases

Country	2011	2012	2013
FR	7000	6900	7000
DE	5800	6000	6200
US	15000	14000	13000

pollution

city	particle size	amount (μg/m^3)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

★ Tidy data



Data Wrangling with dplyr and tidyr Cheat Sheet

RStudio

Syntax - Helpful conventions for wrangling

dplyr::**tbl_df(iris)**

Converts data to tbl class. tbl's are easier to examine than data frames. R displays only the data that fits onscreen:

```
Source: local data frame [150 x 5]

  Sepal.Length Sepal.Width Petal.Length
1          5.1         3.5          1.4
2          4.9         3.0          1.4
3          4.7         3.2          1.3
4          4.6         3.1          1.5
5          5.0         3.6          1.4
..         ...         ...          ...
Variables not shown: Petal.Width (dbl),
  Species (fctr)
```

dplyr::**glimpse(iris)**

Information dense summary of tbl data.

utils::**View(iris)**

View data set in spreadsheet-like display (note capital V).

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa

dplyr::**%>%**

Passes object on left hand side as first argument (or . argument) of function on righthand side.

x %>% f(y) *is the same as* **f(x, y)**
y %>% f(x, ., z) *is the same as* **f(x, y, z)**

"Piping" with %>% makes code more readable, e.g.

```
iris %>%
  group_by(Species) %>%
  summarise(avg = mean(Sepal.Width)) %>%
  arrange(avg)
```

Tidy Data - A foundation for wrangling in R

In a tidy data set:

Each **variable** is saved in its own **column**

&

Each **observation** is saved in its own **row**

Tidy data complements R's **vectorized operations**. R will automatically preserve observations as you manipulate variables. No other format works as intuitively with R.

Reshaping Data - Change the layout of a data set

tidyr::**gather(cases, "year", "n", 2:4)**

Gather columns into rows.

tidyr::**spread(pollution, size, amount)**

Spread rows into columns.

tidyr::**separate(storms, date, c("y", "m", "d"))**

Separate one column into several.

tidyr::**unite(data, col, ..., sep)**

Unite several columns into one.

dplyr::**data_frame(a = 1:3, b = 4:6)**

Combine vectors into data frame (optimized).

dplyr::**arrange(mtcars, mpg)**

Order rows by values of a column (low to high).

dplyr::**arrange(.mtcars, desc(mpg))**

Order rows by values of a column (high to low).

dplyr::**rename(tb, y = year)**

Rename the columns of a data frame.

Subset Observations (Rows)

dplyr::**filter(iris, Sepal.Length > 7)**

Extract rows that meet logical criteria.

dplyr::**distinct(iris)**

Remove duplicate rows.

dplyr::**sample_frac(iris, 0.5, replace = TRUE)**

Randomly select fraction of rows.

dplyr::**sample_n(iris, 10, replace = TRUE)**

Randomly select n rows.

dplyr::**slice(iris, 10:15)**

Select rows by position.

dplyr::**top_n(storms, 2, date)**

Select and order top n entries (by group if grouped data).

Logic in R - ?Comparison, ?base::Logic

<	Less than	!=	Not equal to
>	Greater than	%in%	Group membership
==	Equal to	is.na	Is NA
<=	Less than or equal to	!is.na	Is not NA
>=	Greater than or equal to	&,|,!,xor,any,all	Boolean operators

Subset Variables (Columns)

dplyr::**select(iris, Sepal.Width, Petal.Length, Species)**

Select columns by name or helper function.

Helper functions for select - ?select

select(iris, **contains(".")**)

Select columns whose name contains a character string.

select(iris, **ends_with("Length")**)

Select columns whose name ends with a character string.

select(iris, **everything()**)

Select every column.

select(iris, **matches(".t.")**)

Select columns whose name matches a regular expression.

select(iris, **num_range("x", 1:5)**)

Select columns named x1, x2, x3, x4, x5.

select(iris, **one_of(c("Species", "Genus"))**)

Select columns whose names are in a group of names.

select(iris, **starts_with("Sepal")**)

Select columns whose name starts with a character string.

select(iris, Sepal.Length:Petal.Width)

Select all columns between Sepal.Length and Petal.Width (inclusive).

select(iris, -Species)

Select all columns except Species.

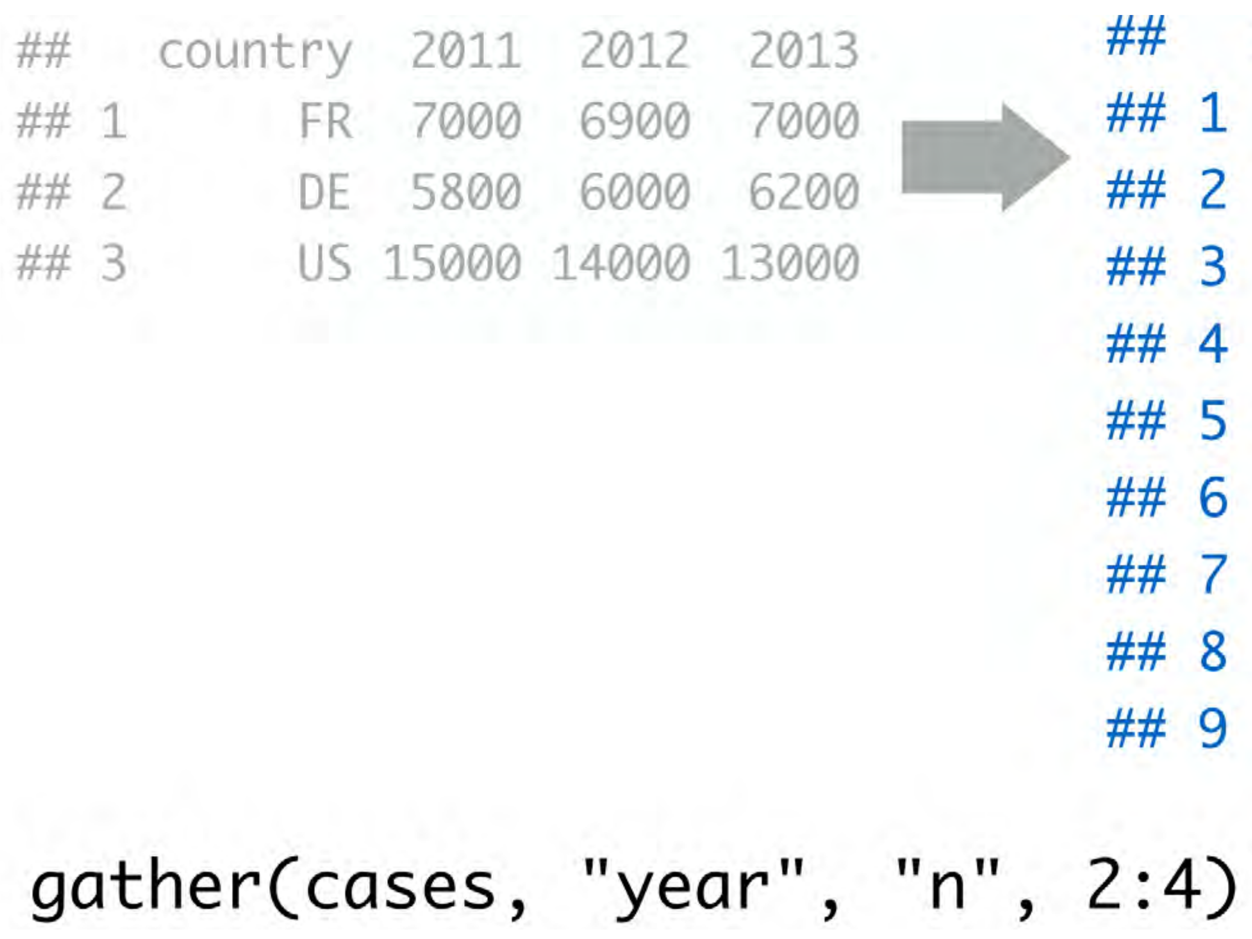
devtools::install_github("rstudio/EDAWR") for data sets. Learn more with **browseVignettes(package = c("dplyr", "tidyr"))** • dplyr 0.4.0• tidyr 0.2.0 • Updated: 1/15

http://www.rstudio.com/resources/cheatsheets/

Data Preparation Tasks

- Reshaping table: gather, spread
- Reshaping column: unite, separate
- Subsetting rows: filter, distinct, sample_n, top_n
- Subsetting columns: select
- Summarise data: summarise
- Group data: group_by
- Make new variables: mutate
- Integration: left_join, union, bind_cols

tidyr
Gather columns

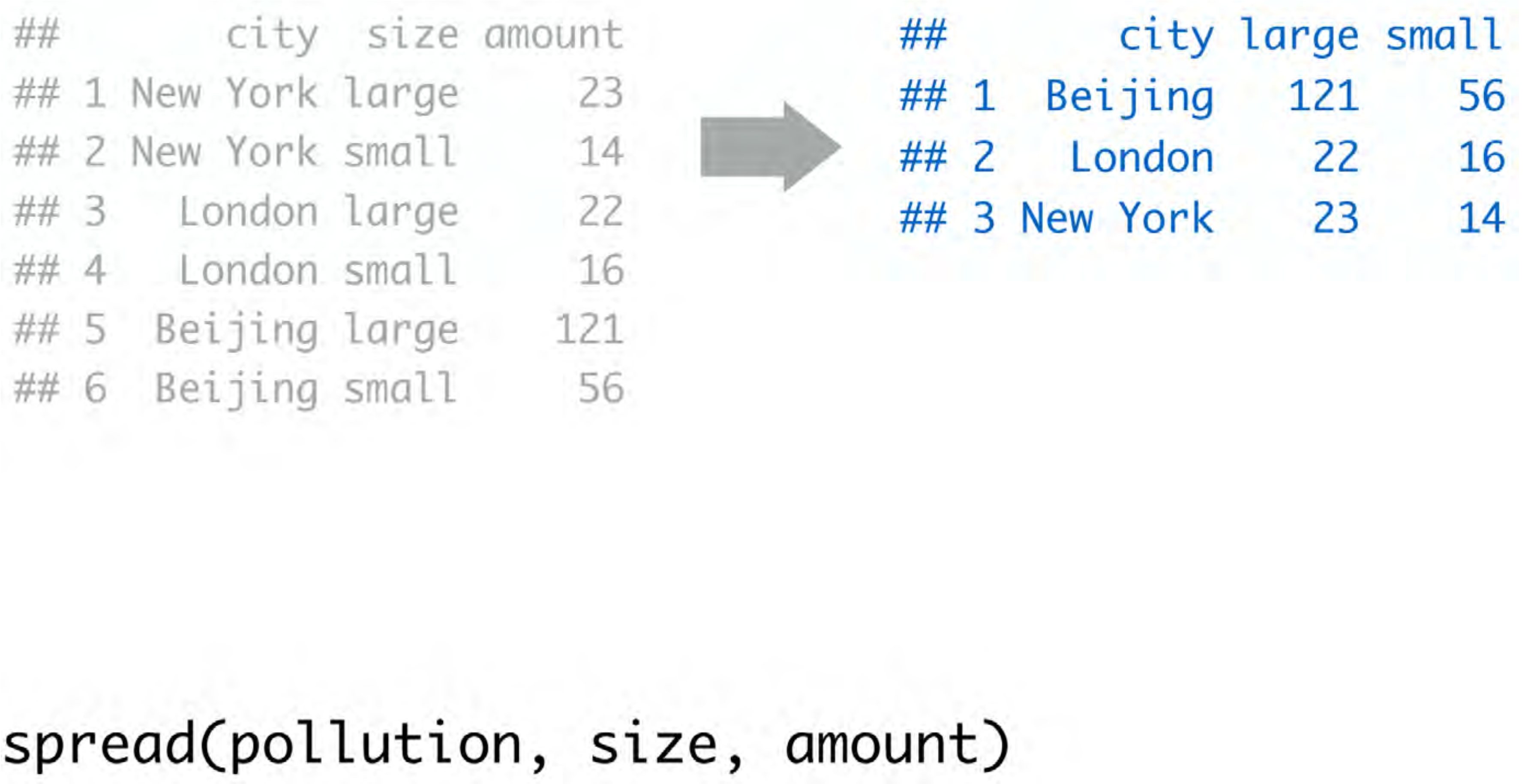


```
##    country  2011  2012  2013
## 1       FR  7000  6900  7000
## 2       DE  5800  6000  6200
## 3       US 15000 14000 13000
```

```
##    country year     n
## 1       FR 2011  7000
## 2       DE 2011  5800
## 3       US 2011 15000
## 4       FR 2012  6900
## 5       DE 2012  6000
## 6       US 2012 14000
## 7       FR 2013  7000
## 8       DE 2013  6200
## 9       US 2013 13000
```

```
gather(cases, "year", "n", 2:4)
```

tidyr
Spread columns



```
##       city  size amount
## 1 New York large     23
## 2 New York small     14
## 3   London large     22
## 4   London small     16
## 5  Beijing large    121
## 6  Beijing small     56
```

```
##       city large small
## 1  Beijing   121    56
## 2   London    22    16
## 3 New York    23    14
```

```
spread(pollution, size, amount)
```

separate()

Separate splits a column by a character string separator.

```
separate(storms, date, c("year", "month", "day"), sep = "-")
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

→

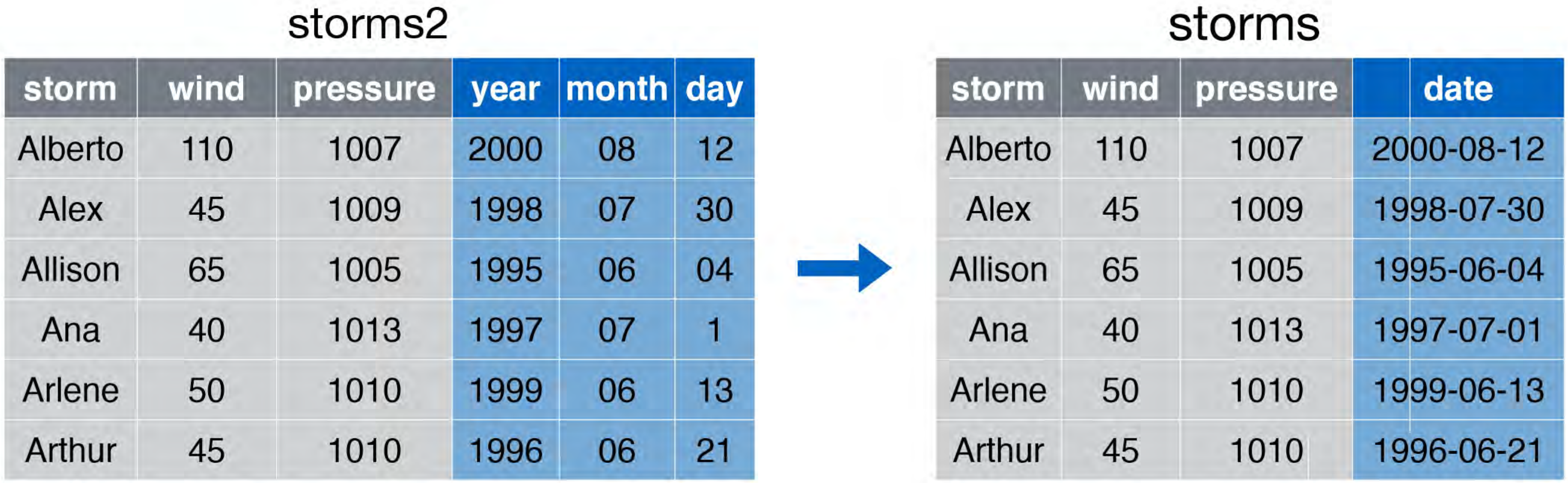
storms2

storm	wind	pressure	year	month	day
Alberto	110	1007	2000	08	12
Alex	45	1009	1998	07	30
Allison	65	1005	1995	06	04
Ana	40	1013	1997	07	1
Arlene	50	1010	1999	06	13
Arthur	45	1010	1996	06	21

unite()

Unite unites columns into a single column.

```
unite(storms2, "date", year, month, day, sep = "-")
```

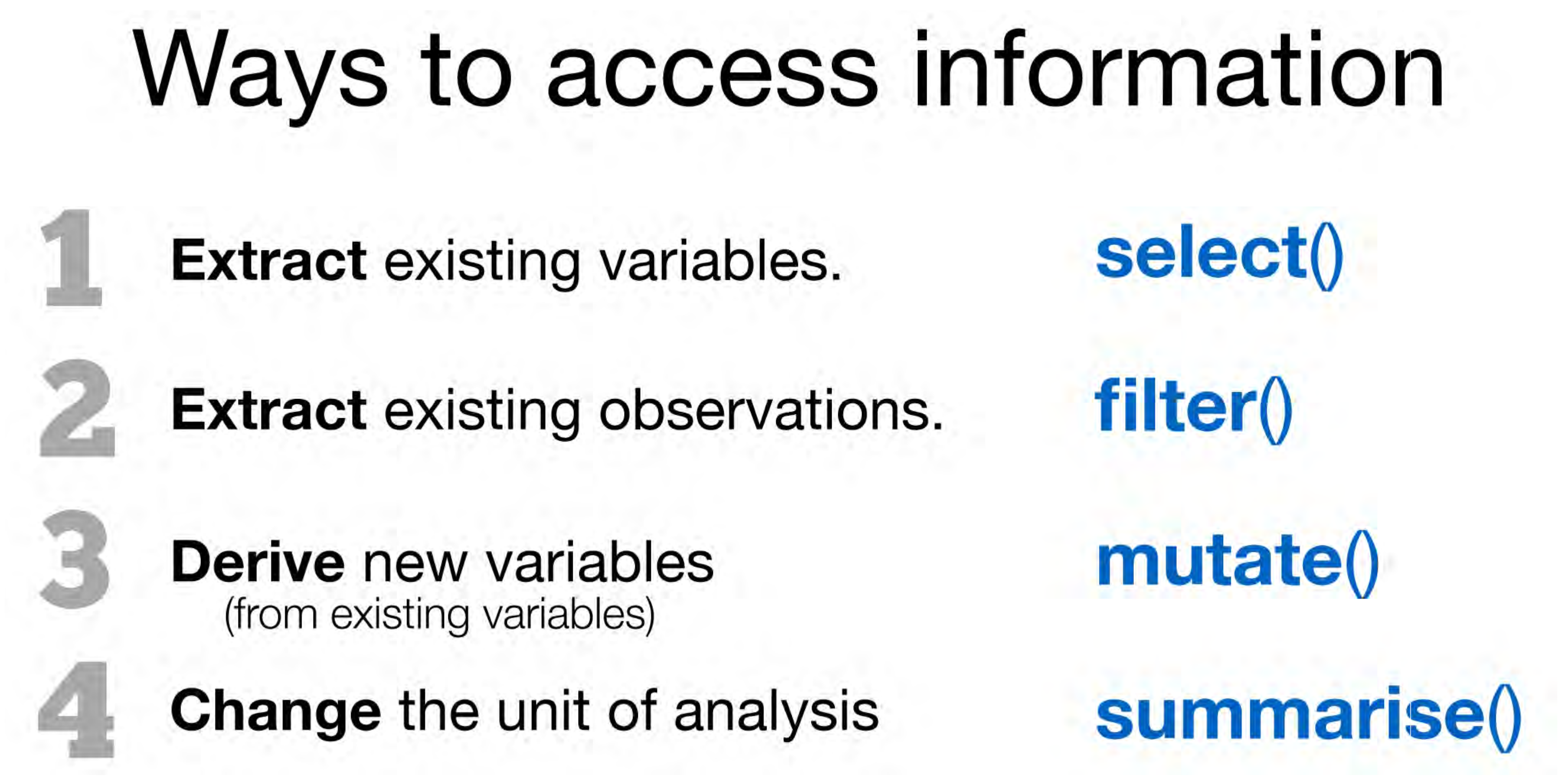


storms2

storm	wind	pressure	year	month	day
Alberto	110	1007	2000	08	12
Alex	45	1009	1998	07	30
Allison	65	1005	1995	06	04
Ana	40	1013	1997	07	1
Arlene	50	1010	1999	06	13
Arthur	45	1010	1996	06	21

storms

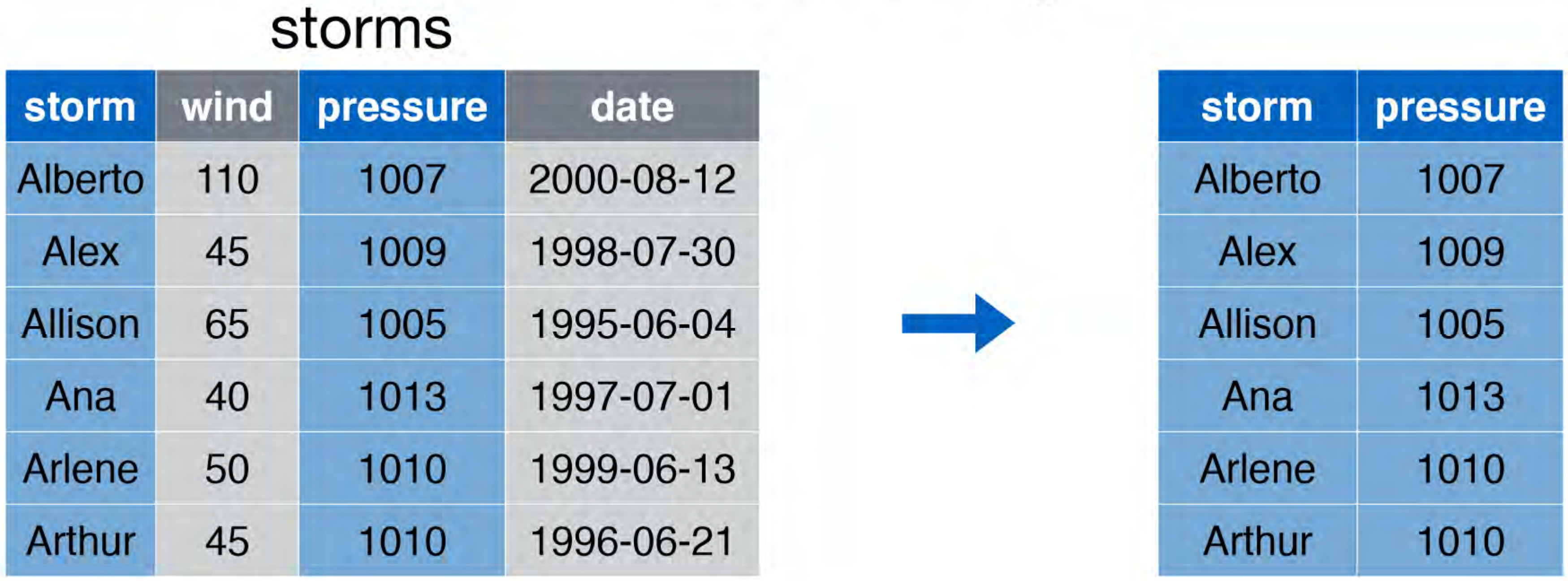
storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



Ways to access information

1	**Extract** existing variables.	select()
2	**Extract** existing observations.	filter()
3	**Derive** new variables (from existing variables)	mutate()
4	**Change** the unit of analysis	summarise()

select()



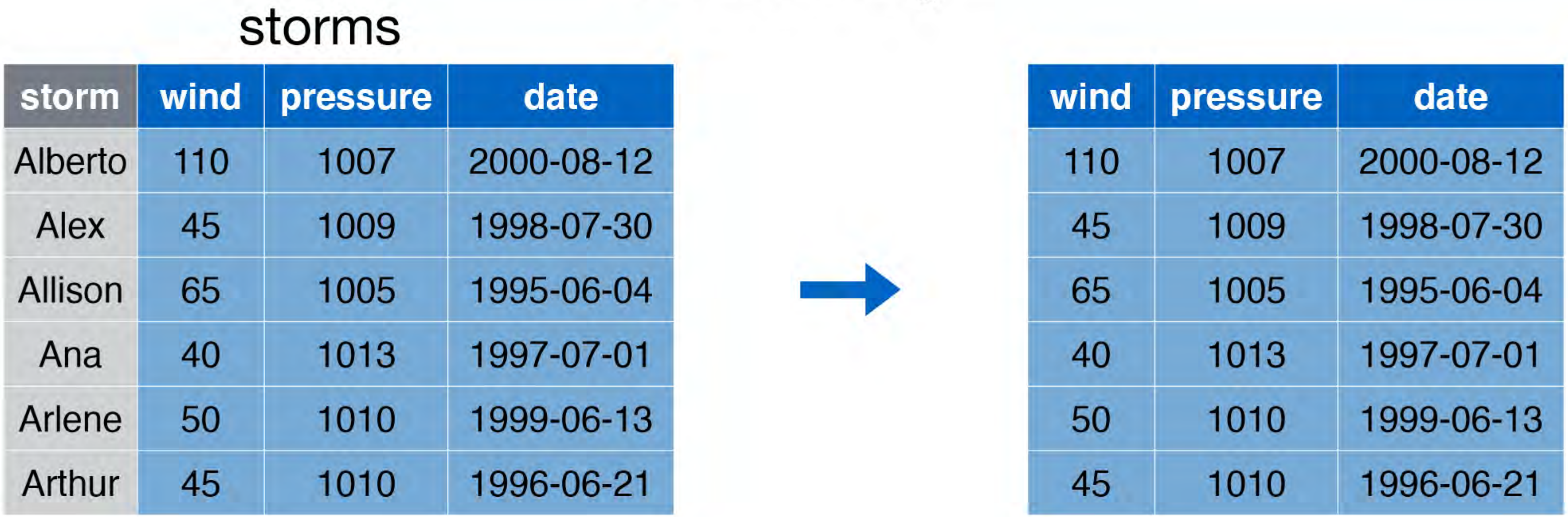
storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	pressure
Alberto	1007
Alex	1009
Allison	1005
Ana	1013
Arlene	1010
Arthur	1010

```
select(storms, storm, pressure)
```

select()



storms

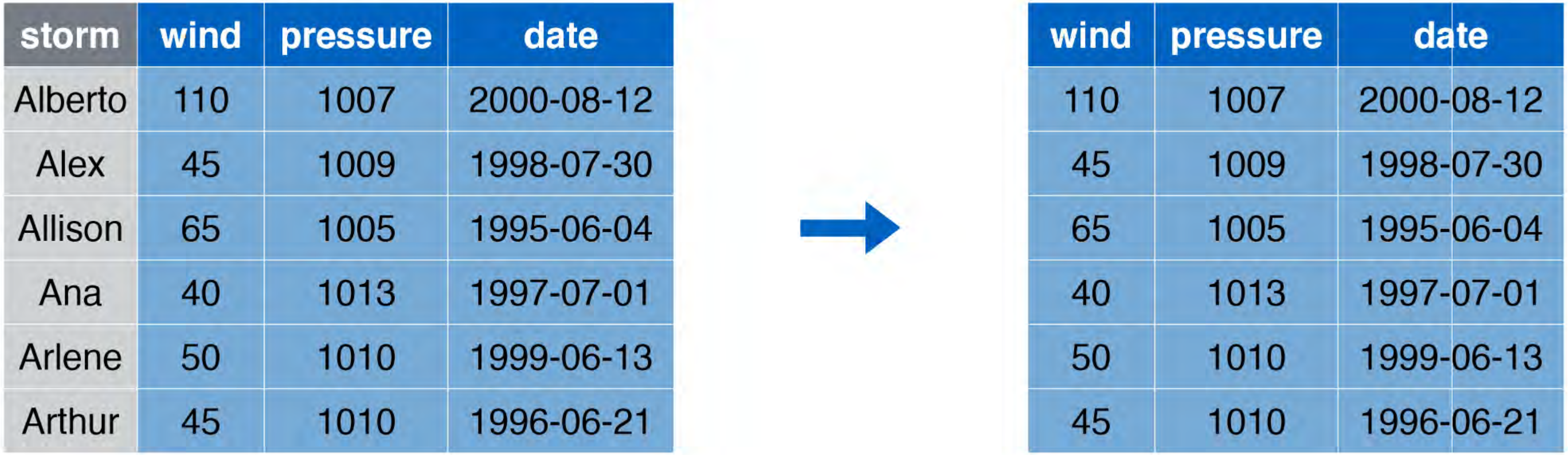
storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

→

wind	pressure	date
110	1007	2000-08-12
45	1009	1998-07-30
65	1005	1995-06-04
40	1013	1997-07-01
50	1010	1999-06-13
45	1010	1996-06-21

```
select(storms, -storm)
# see ?select for more
```

select()



storms

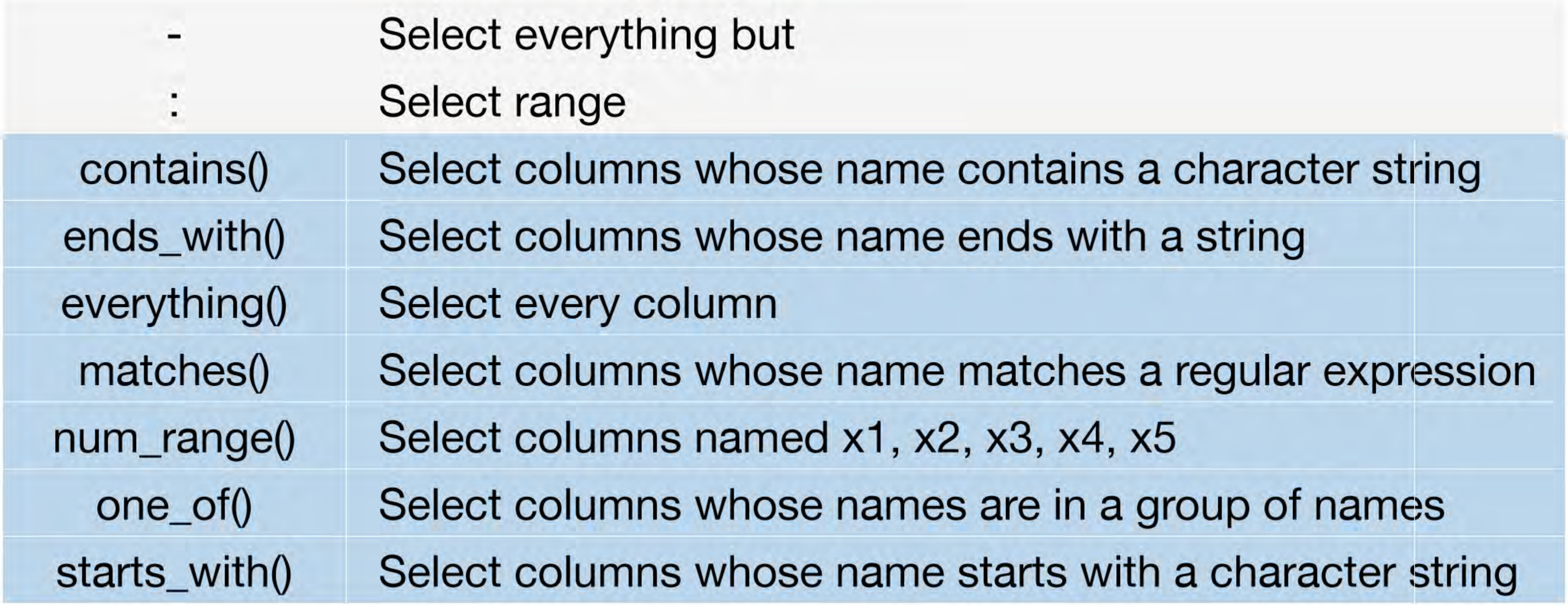
storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

wind	pressure	date
110	1007	2000-08-12
45	1009	1998-07-30
65	1005	1995-06-04
40	1013	1997-07-01
50	1010	1999-06-13
45	1010	1996-06-21

```
select(storms, wind:date)
# see ?select for more
```

Useful select functions

* Blue functions come in dplyr



-	Select everything but
:	Select range
contains()	Select columns whose name contains a character string
ends_with()	Select columns whose name ends with a string
everything()	Select every column
matches()	Select columns whose name matches a regular expression
num_range()	Select columns named x1, x2, x3, x4, x5
one_of()	Select columns whose names are in a group of names
starts_with()	Select columns whose name starts with a character string

filter()

storms

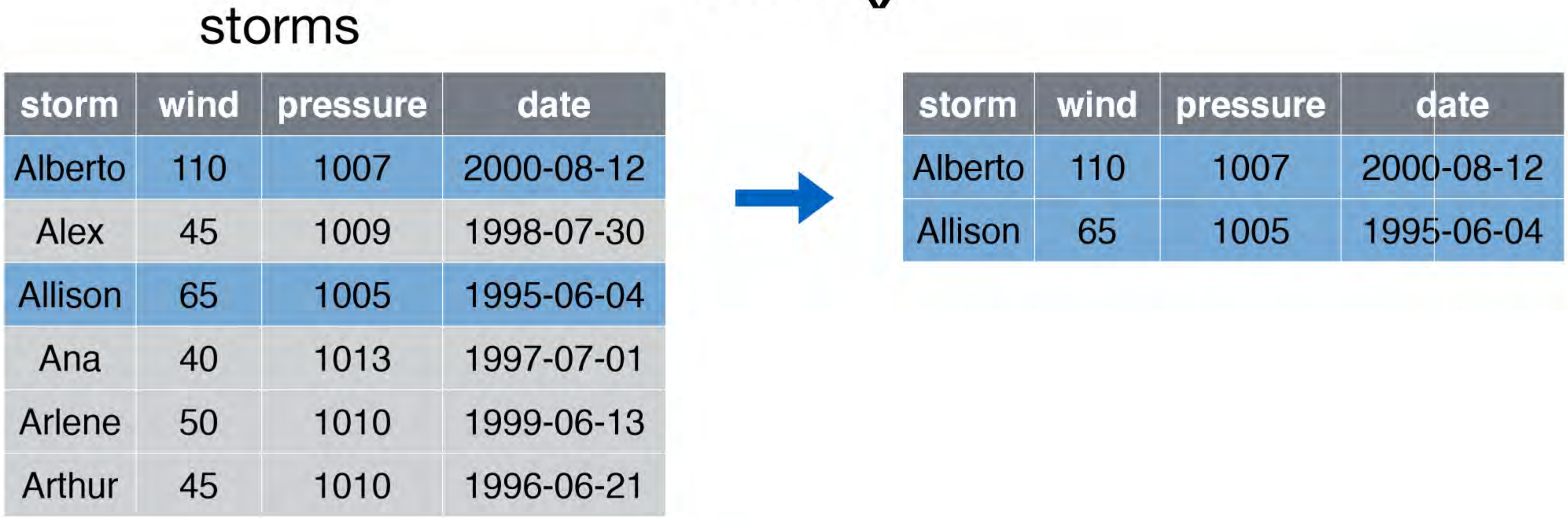
storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

→

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Allison	65	1005	1995-06-04
Arlene	50	1010	1999-06-13

```
filter(storms, wind >= 50)
```

filter()



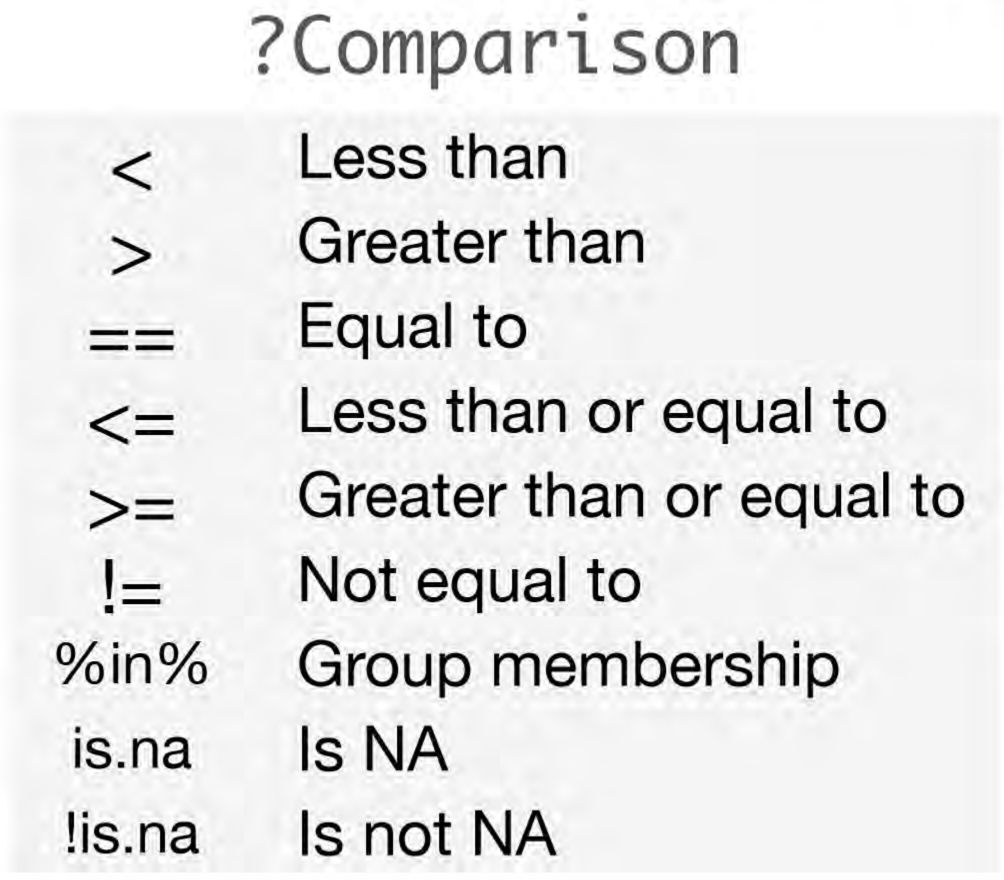
storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Allison	65	1005	1995-06-04

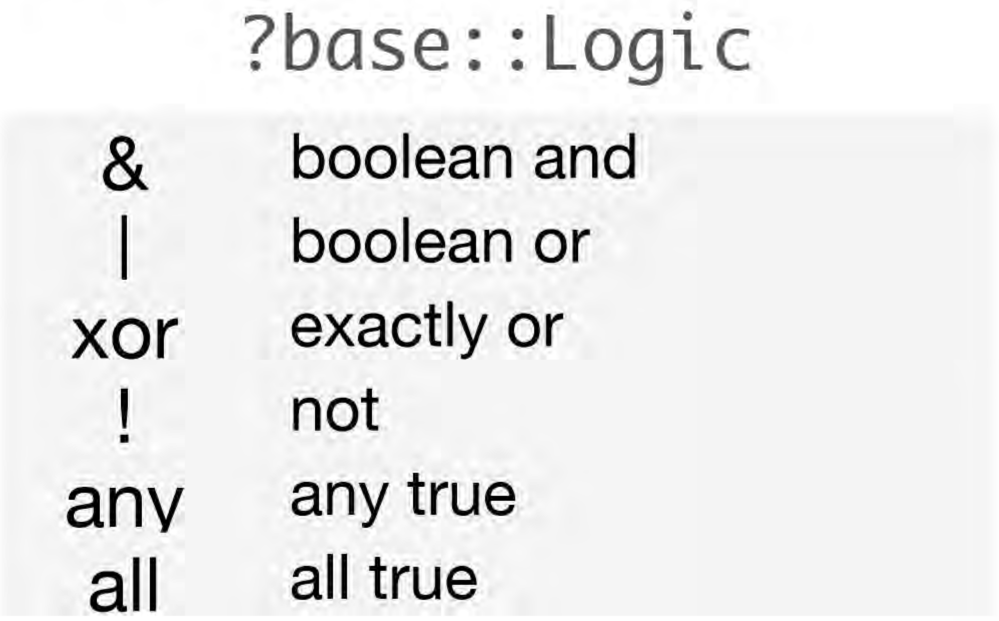
```
filter(storms, wind >= 50,
  storm %in% c("Alberto", "Alex", "Allison"))
```

logical tests in R



?Comparison

<	Less than
>	Greater than
==	Equal to
<=	Less than or equal to
>=	Greater than or equal to
!=	Not equal to
%in%	Group membership
is.na	Is NA
!is.na	Is not NA



?base::Logic

&	boolean and
|	boolean or
xor	exactly or
!	not
any	any true
all	all true

mutate()

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

→

storm	wind	pressure	date	ratio	inverse
Alberto	110	1007	2000-08-12	9.15	0.11
Alex	45	1009	1998-07-30	22.42	0.04
Allison	65	1005	1995-06-04	15.46	0.06
Ana	40	1013	1997-07-01	25.32	0.04
Arlene	50	1010	1999-06-13	20.20	0.05
Arthur	45	1010	1996-06-21	22.44	0.04

```
mutate(storms, ratio = pressure / wind, inverse = ratio^-1)
```

Useful mutate functions

* All take a vector of values and return a vector of values

** Blue functions come in dplyr

pmin(), pmax()	Element-wise min and max
cummin(), cummax()	Cumulative min and max
cumsum(), cumprod()	Cumulative sum and product
between()	Are values between a and b?
cume_dist()	Cumulative distribution of values
cumall(), cumany()	Cumulative all and any
cummean()	Cumulative mean
lead(), lag()	Copy with values one position
ntile()	Bin vector into n buckets
dense_rank(), min_rank(), percent_rank(), row_number()	Various ranking methods

"Window" functions

* All take a vector of values and return a vector of values

pmin(), pmax()

cummin(), cummax()

cumsum(), cumprod()

between()

cume_dist()

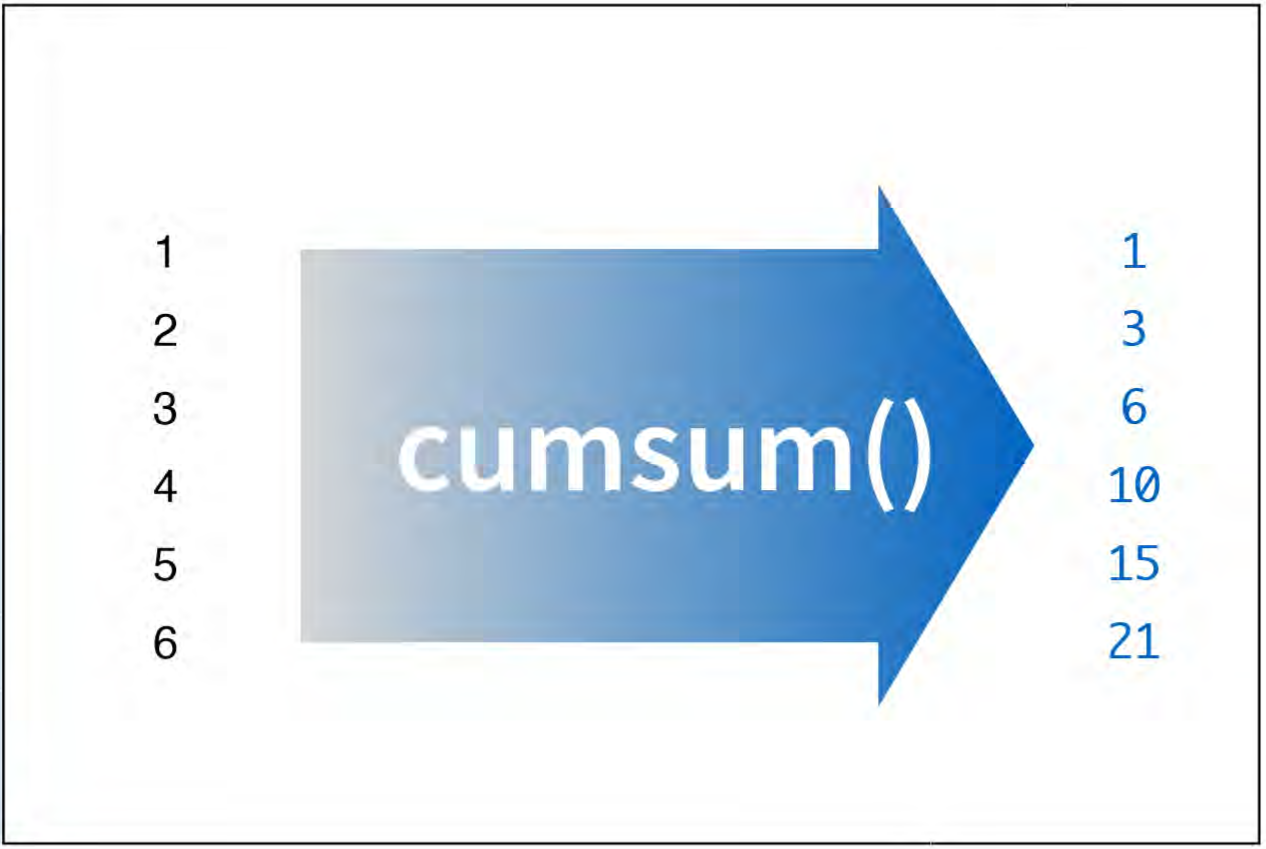
cumall(), cumany()

cummean()

lead(), lag()

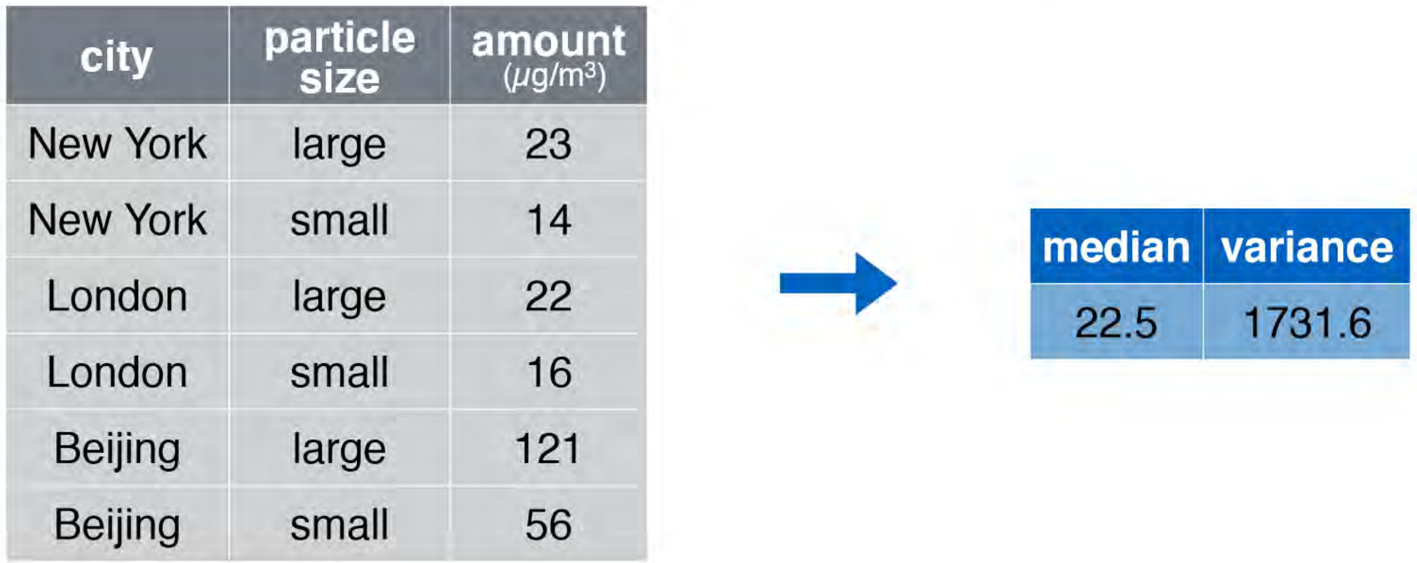
ntile()

dense_rank(), min_rank(),
percent_rank(), row_number()



1	cumsum()	1
2		3
3		6
4		10
5		15
6		21

summarise()

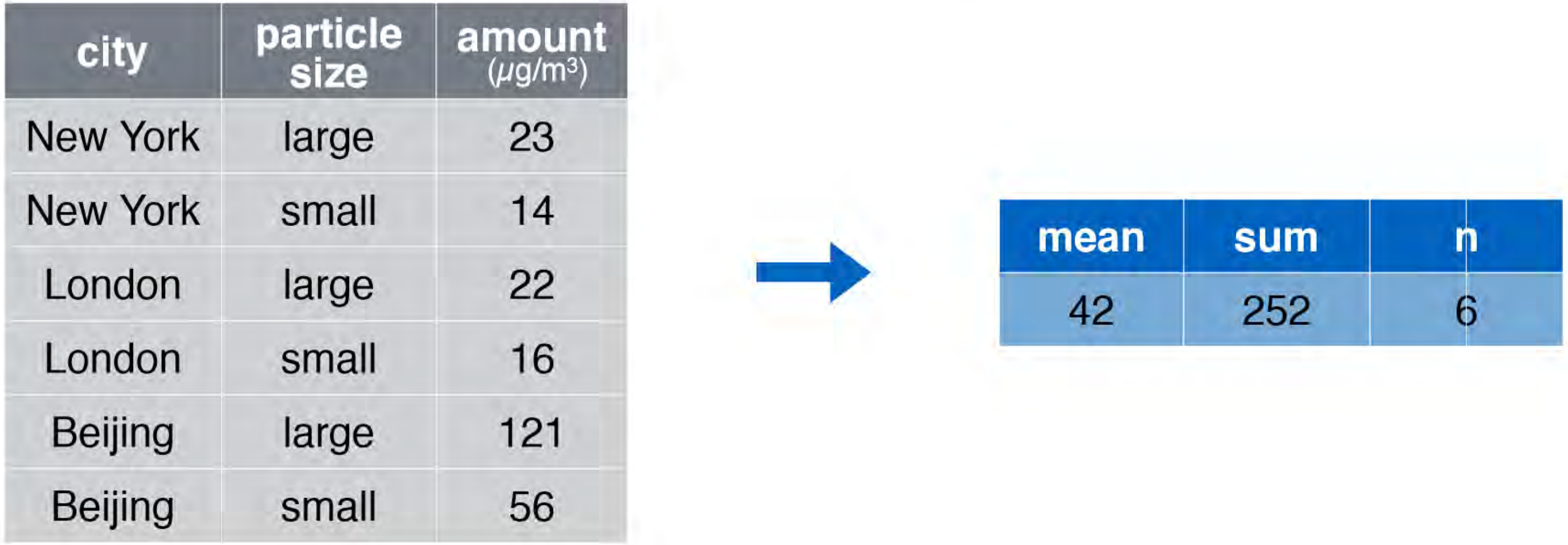


city	particle size	amount (μg/m^3)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

median	variance
22.5	1731.6

```
pollution %>% summarise(median = median(amount), variance = var(amount))
```

summarise()



city	particle size	amount ($\mu g/m^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

mean	sum	n
42	252	6

```
pollution %>% summarise(mean = mean(amount), sum = sum(amount), n = n())
```

Useful summary functions

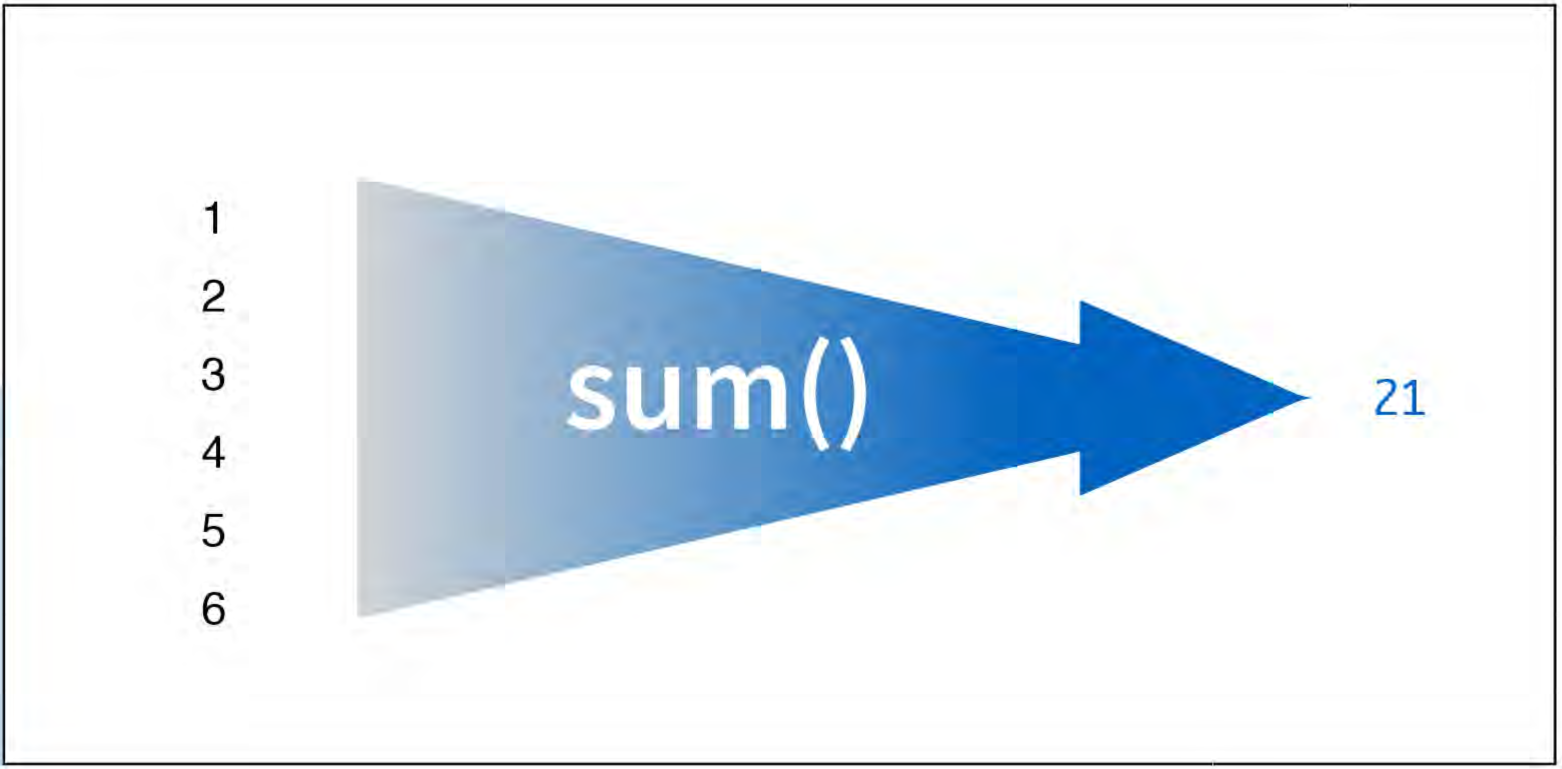
* All take a vector of values and return a single value
** Blue functions come in dplyr

min(), max()	Minimum and maximum values
mean()	Mean value
median()	Median value
sum()	Sum of values
var, sd()	Variance and standard deviation of a vector
first()	First value in a vector
last()	Last value in a vector
nth()	Nth value in a vector
n()	The number of values in a vector
n_distinct()	The number of distinct values in a vector

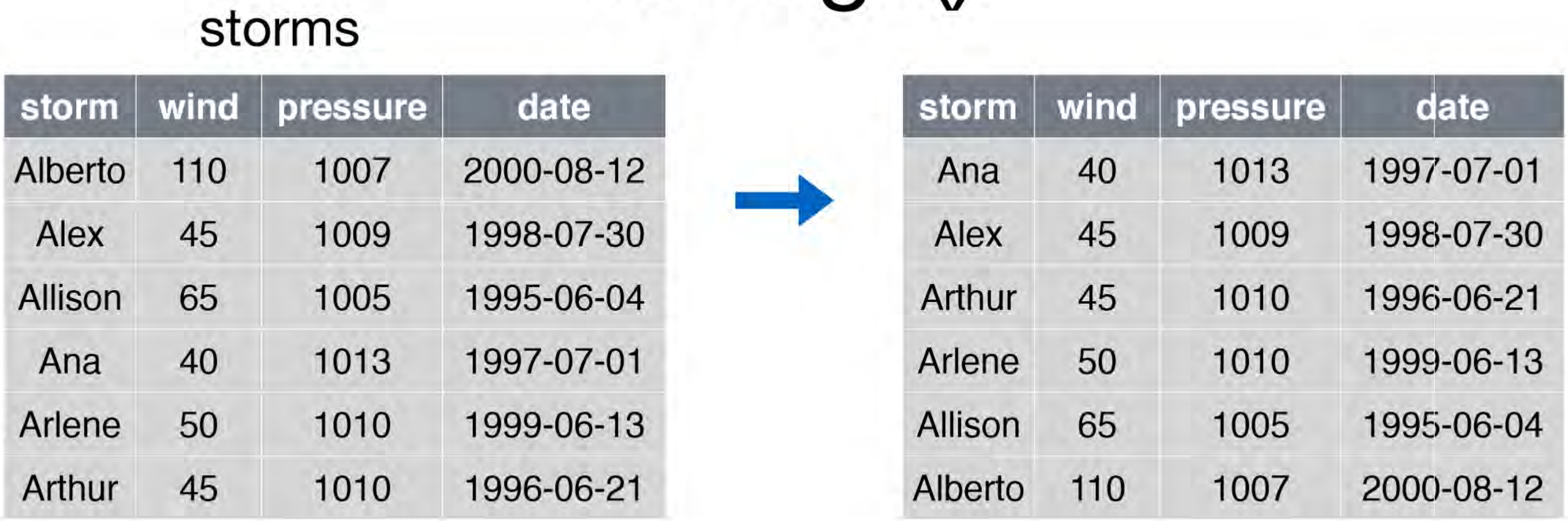
"Summary" functions

* All take a vector of values and return a single value

min(), max()
mean()
median()
sum()
var, sd()
first()
last()
nth()
n()
n_distinct()



arrange()



storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

→

storm	wind	pressure	date
Ana	40	1013	1997-07-01
Alex	45	1009	1998-07-30
Arthur	45	1010	1996-06-21
Arlene	50	1010	1999-06-13
Allison	65	1005	1995-06-04
Alberto	110	1007	2000-08-12

```
arrange(storms, wind)
```

arrange()

storms

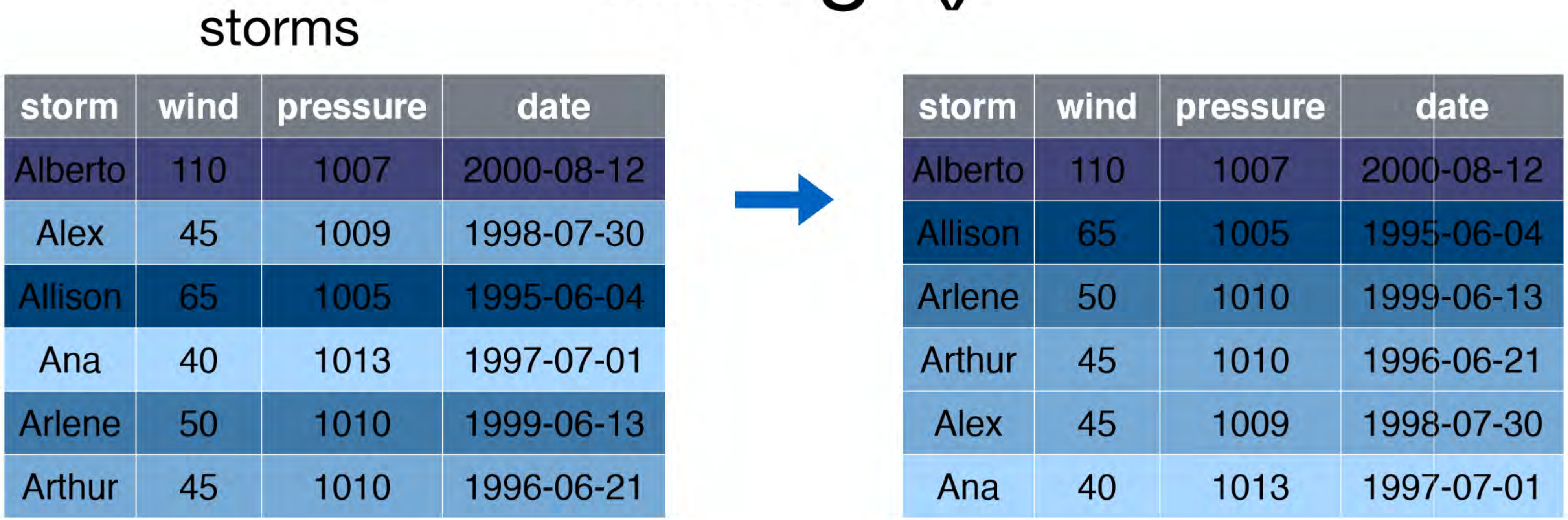
storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

→

storm	wind	pressure	date
Ana	40	1013	1997-07-01
Alex	45	1009	1998-07-30
Arthur	45	1010	1996-06-21
Arlene	50	1010	1999-06-13
Allison	65	1005	1995-06-04
Alberto	110	1007	2000-08-12

```
arrange(storms, wind)
```

arrange()



storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

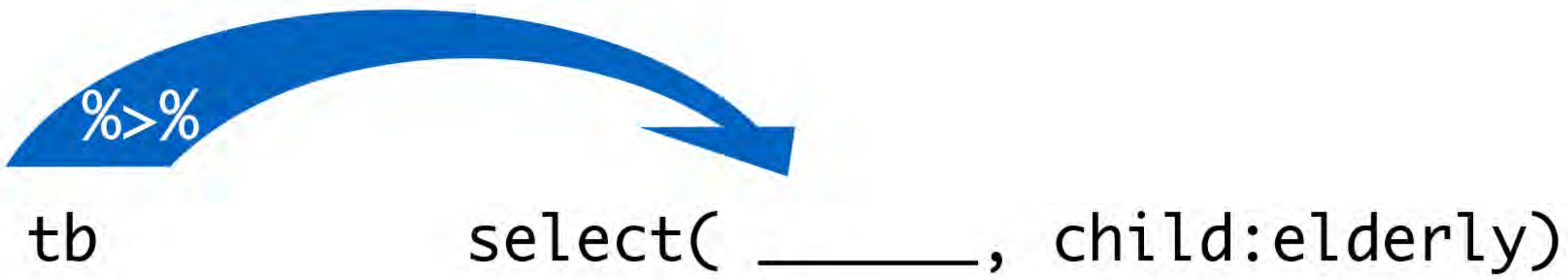
storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Allison	65	1005	1995-06-04
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21
Alex	45	1009	1998-07-30
Ana	40	1013	1997-07-01

```
arrange(storms, desc(wind))
```

The pipe operator %>%

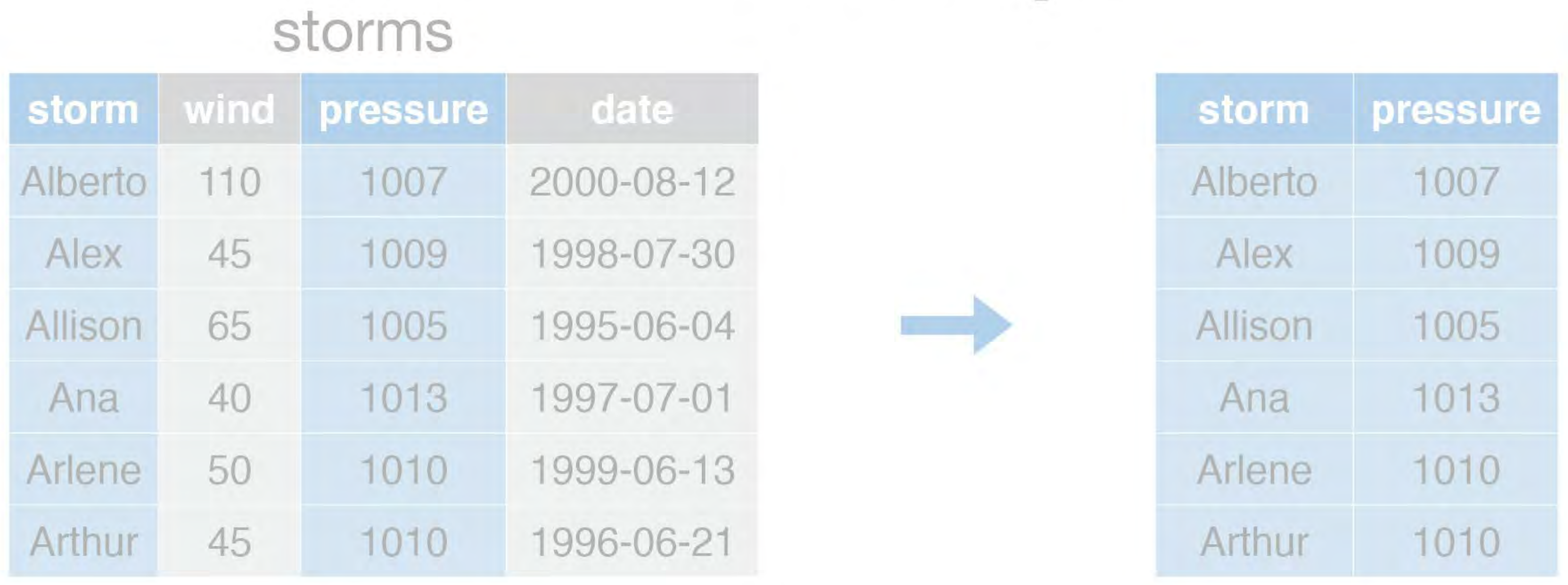
```
library(dplyr)
select(tb, child:elderly)
tb %>% select(child:elderly)
```

These do the same thing
Try it!



```
tb             select( _____, child:elderly)
```

select()



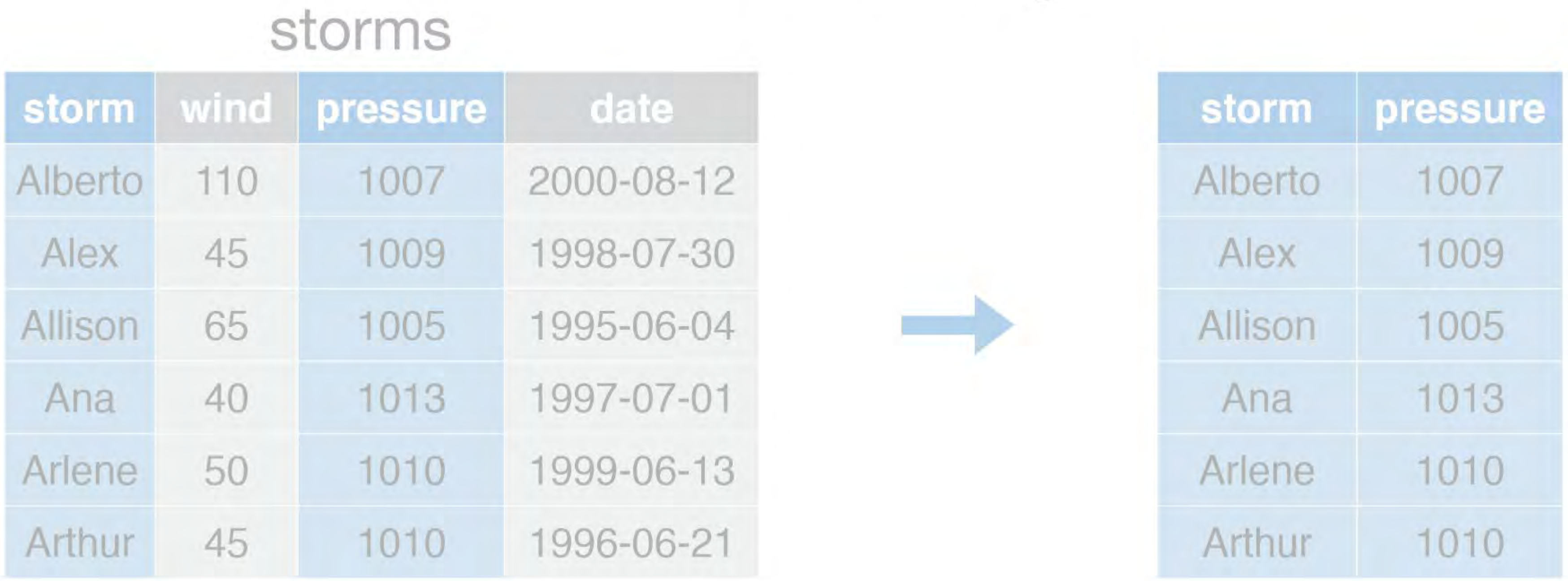
storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	pressure
Alberto	1007
Alex	1009
Allison	1005
Ana	1013
Arlene	1010
Arthur	1010

```
select(storms, storm, pressure)
```

select()



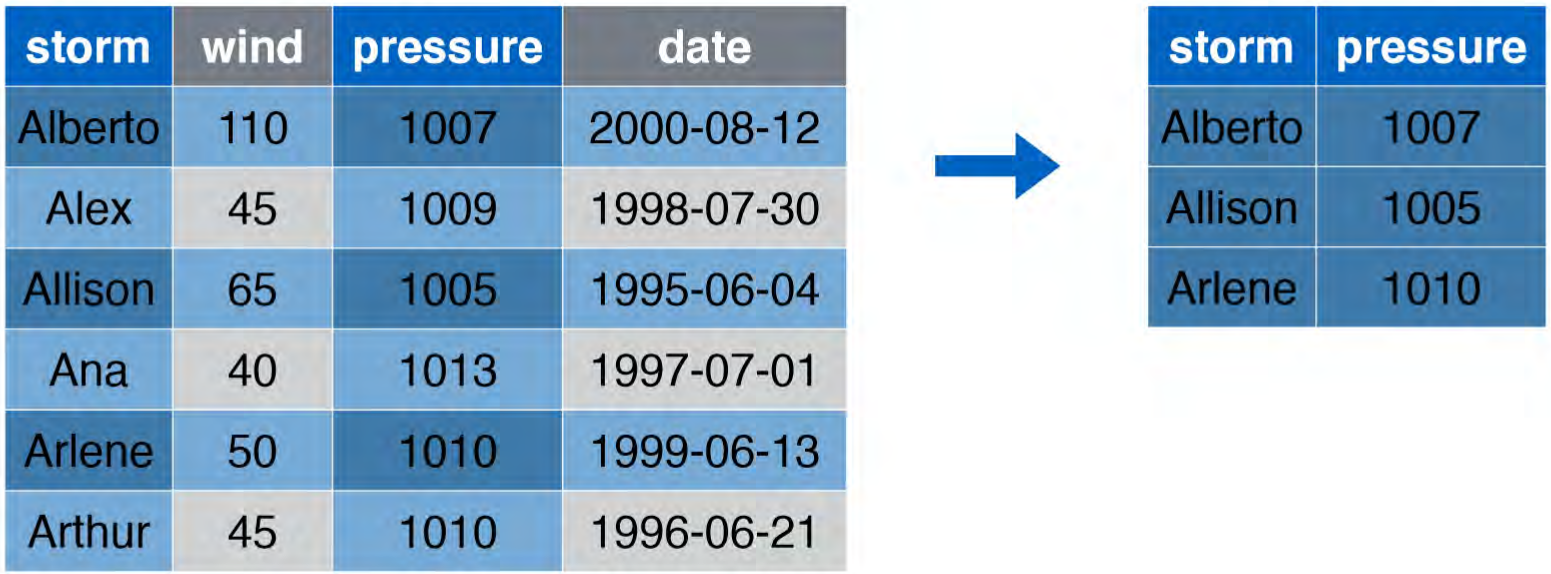
storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	pressure
Alberto	1007
Alex	1009
Allison	1005
Ana	1013
Arlene	1010
Arthur	1010

```
storms %>% select(storm, pressure)
```

storms

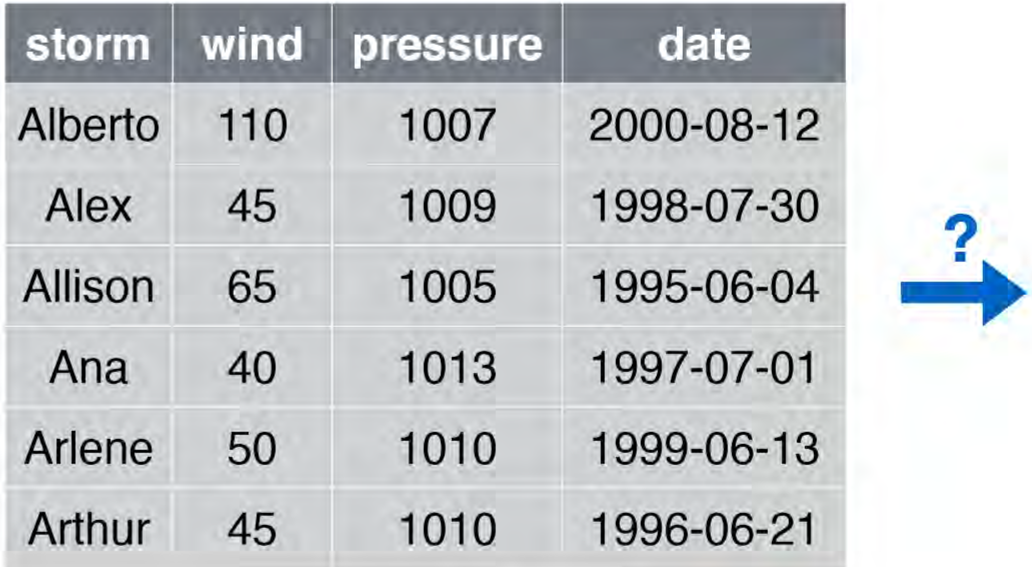


storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	pressure
Alberto	1007
Allison	1005
Arlene	1010

```
storms %>%
  filter(wind >= 50) %>%
  select(storm, pressure)
```

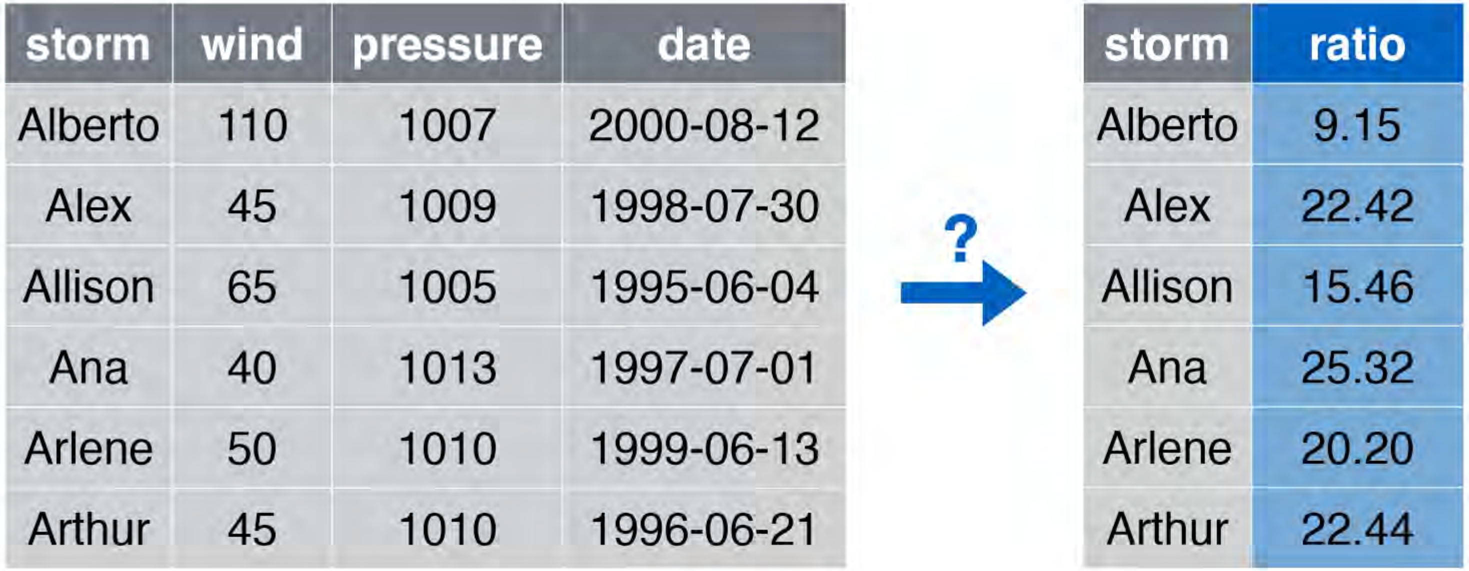
mutate()



storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

```
storms %>%
  mutate(ratio = pressure / wind) %>%
  select(storm, ratio)
```

mutate()

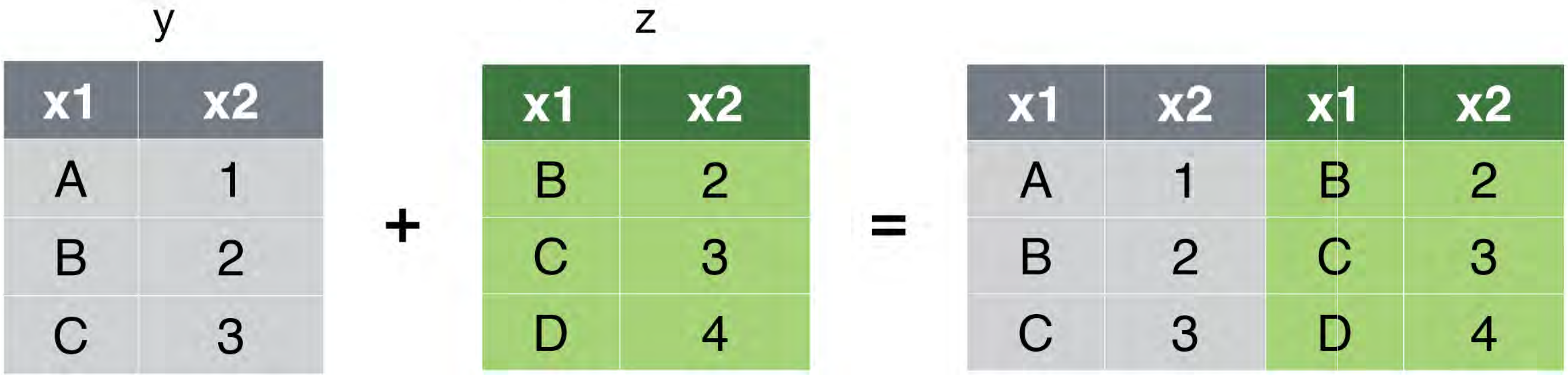


storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

storm	ratio
Alberto	9.15
Alex	22.42
Allison	15.46
Ana	25.32
Arlene	20.20
Arthur	22.44

```
storms %>%
  mutate(ratio = pressure / wind) %>%
  select(storm, ratio)
```

dplyr::**bind_cols()**



y

x1	x2
A	1
B	2
C	3

+

z

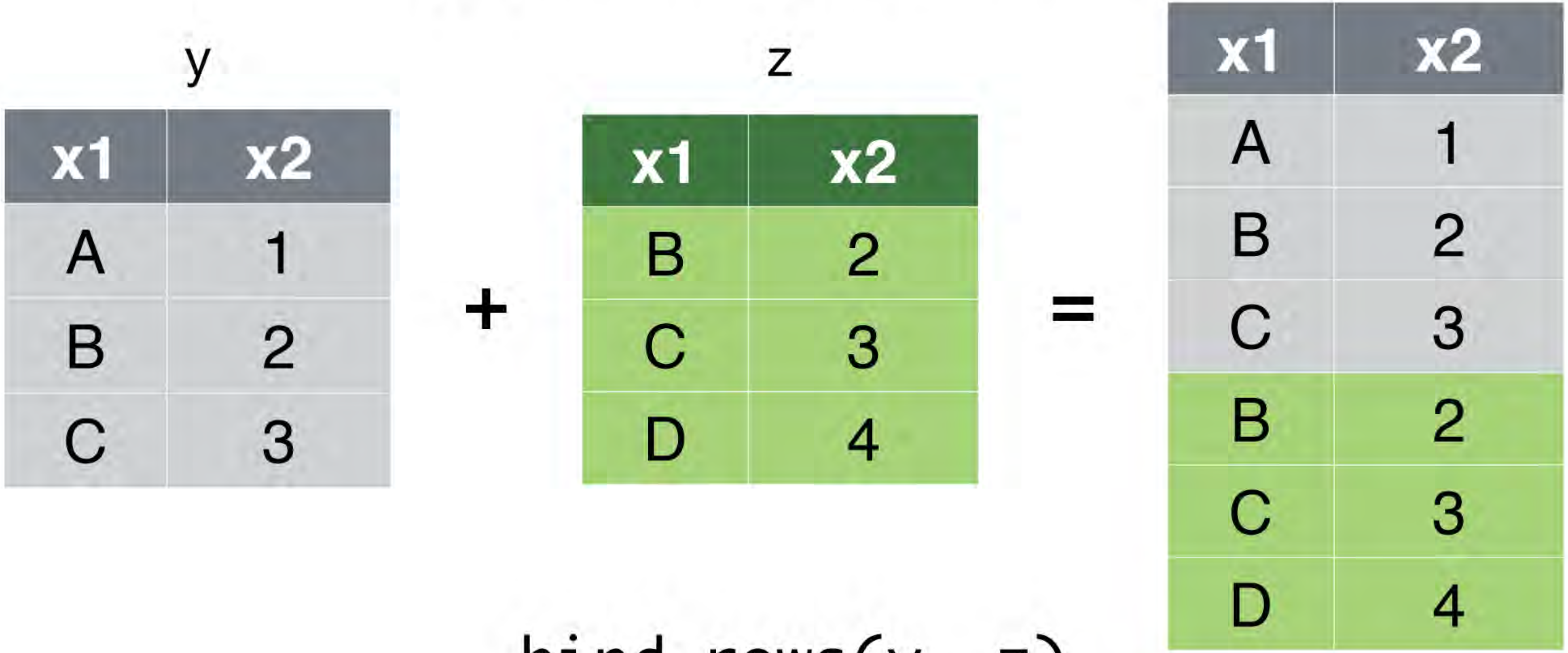
x1	x2
B	2
C	3
D	4

=

x1	x2	x1	x2
A	1	B	2
B	2	C	3
C	3	D	4

```
bind_cols(y, z)
```

dplyr::**bind_rows()**



y

x1	x2
A	1
B	2
C	3

+

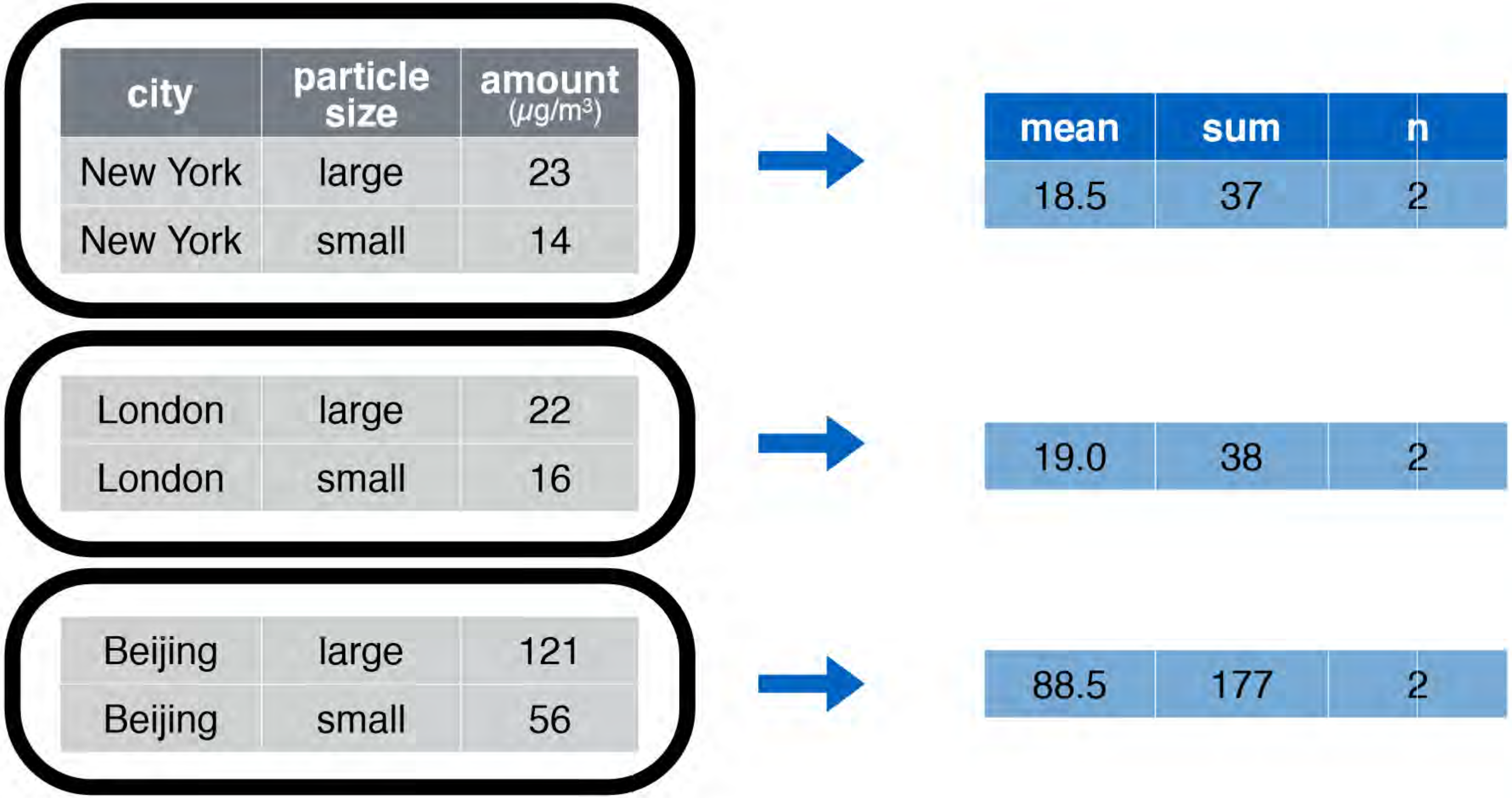
z

x1	x2
B	2
C	3
D	4

=

x1	x2
A	1
B	2
C	3
B	2
C	3
D	4

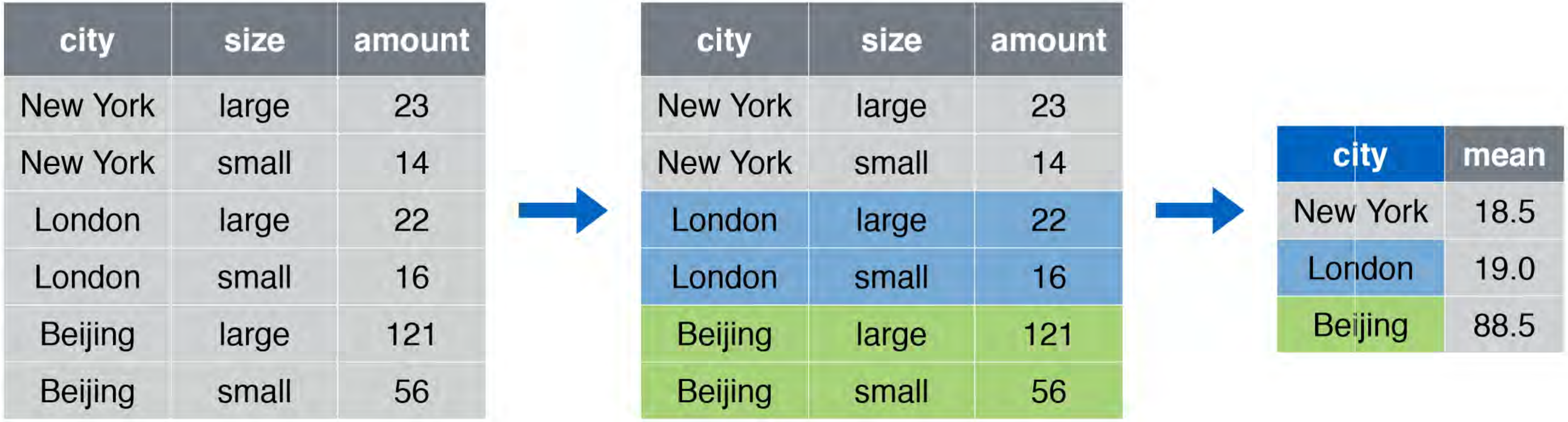
```
bind_rows(y, z)
```



city	particle size	amount ($\mu g/m^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

mean	sum	n
18.5	37	2
19.0	38	2
88.5	177	2

group_by() + summarise()



city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	mean
New York	18.5
London	19.0
Beijing	88.5

```
pollution %>% group_by(city) %>%  summarise(mean = mean(amount))
```

dplyr::**left_join()**

songs

song	name
Across the Universe	John
Come Together	John
Hello, Goodbye	Paul
Peggy Sue	Buddy

+

artists

name	plays
George	sitar
John	guitar
Paul	bass
Ringo	drums

=

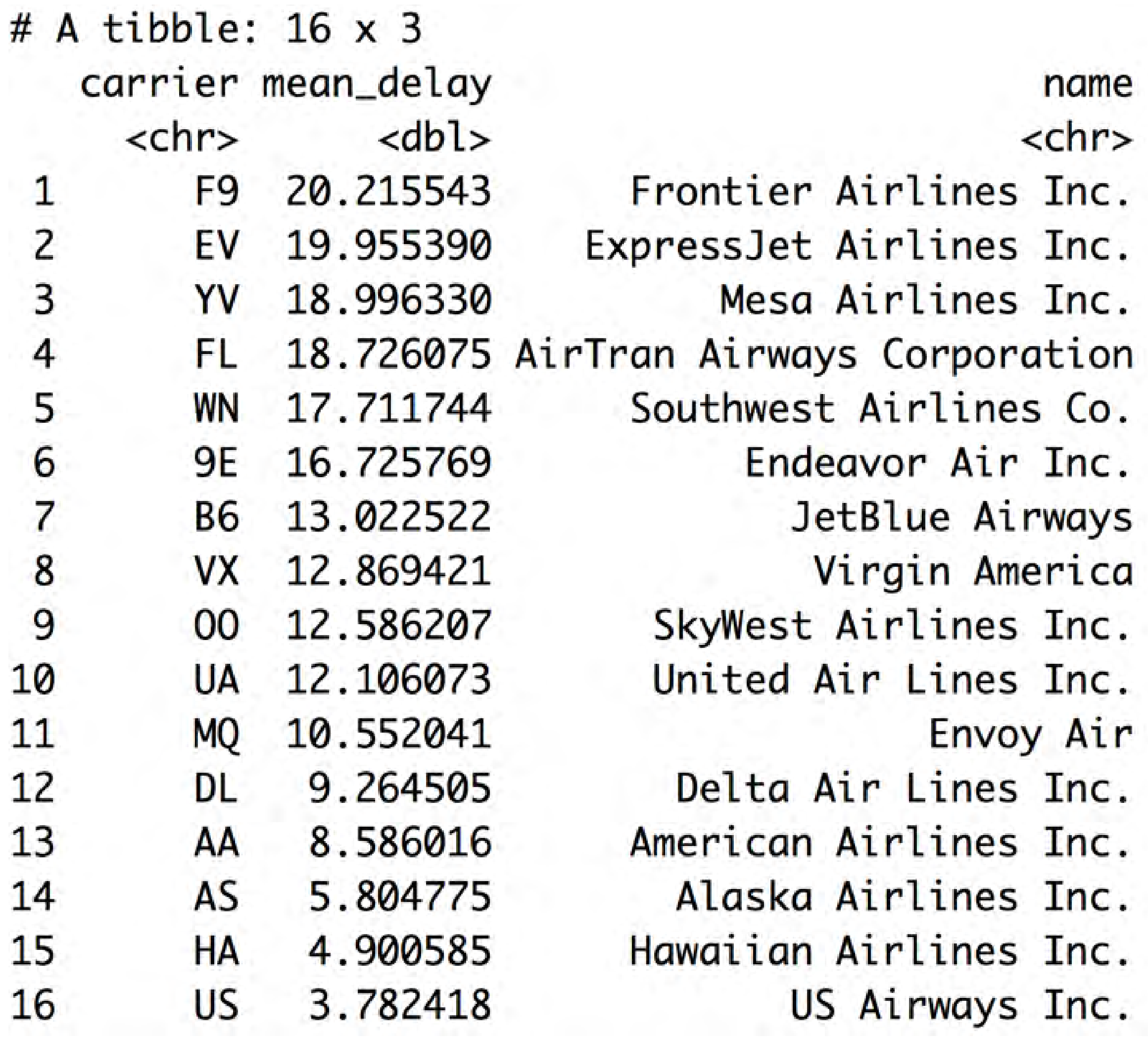
song	name	plays
Across the Universe	John	guitar
Come Together	John	guitar
Hello, Goodbye	Paul	bass
Peggy Sue	Buddy	**<NA>**

```
left_join(songs, artists, by = "name")
```

Lab

- Load two data frames “flights.csv” and “airlines.csv”
  - Hint: `read.csv`
- Select two columns (carrier, dep_delay)
- Filter NA out of dep_delay
- Use carrier as a group and calculate mean of dep_delay
- Sort the worst carrier by departure delay (dep_delay)
- Join airlines data to flight data using ‘carrier’ as a key
- Show the result

Expected Results



```
# A tibble: 16 x 3
   carrier mean_delay                        name
     <chr>      <dbl>                       <chr>
1       F9  20.215543      Frontier Airlines Inc.
2       EV  19.955390     ExpressJet Airlines Inc.
3       YV  18.996330          Mesa Airlines Inc.
4       FL  18.726075 AirTran Airways Corporation
5       WN  17.711744      Southwest Airlines Co.
6       9E  16.725769           Endeavor Air Inc.
7       B6  13.022522             JetBlue Airways
8       VX  12.869421              Virgin America
9       OO  12.586207       SkyWest Airlines Inc.
10      UA  12.106073       United Air Lines Inc.
11      MQ  10.552041                   Envoy Air
12      DL   9.264505        Delta Air Lines Inc.
13      AA   8.586016      American Airlines Inc.
14      AS   5.804775        Alaska Airlines Inc.
15      HA   4.900585      Hawaiian Airlines Inc.
16      US   3.782418             US Airways Inc.
```

Thank you

Question?