

Data frames and tibbles: tables in R

Introduction to R - Day 1

Instructor: **Selina Baldauf**

Freie Universität Berlin - Theoretical Ecology

2021-06-15 (updated: 2021-08-01)

Data frames

The built-in data structure for tables in R is a `data frame`.

- vectors in R can't represent a table with data that is connected via rows

Data frames are one of the **biggest and most important ideas** in R, and one of the things that make R different from other programming languages.

(Wickham, [Advanced R](#))

cities	population	area_km2
Istanbul	15100000	2576
Moscow	12500000	2561
London	9000000	1572
Saint Petersburg	5400000	1439
Berlin	3800000	891
Madrid	3200000	604
Kyiv	3000000	839
Rome	2800000	1285
Bucharest	2200000	228
Paris	2100000	105

Data frames

A data frame is a **named list of vectors** of the same length.

Basic properties of a data frame

- every **column is a vector**
- columns have a **header**
 - this is the **name** of the vector in the list
- within one column, all values are of the **same data type**
- every column has the same length

The diagram shows a data frame table with three columns: 'cities', 'population', and 'area_km2'. The 'cities' column is labeled 'character' with a downward arrow. The 'population' and 'area_km2' columns are labeled 'numeric' with a downward arrow. A red box highlights the header row. A blue arrow points downwards along the left side of the table, indicating the length of the vectors.

cities	population	area_km2
Istanbul	15100000	2576
Moscow	12500000	2561
London	9000000	1572
Saint Petersburg	5400000	1439
Berlin	3800000	891
Madrid	3200000	604
Kyiv	3000000	839
Rome	2800000	1285
Bucharest	2200000	228
Paris	2100000	105

Data frames

Data frames are created with the function `data.frame()`:

```
cities <- c(
  "Istanbul", "Moscow", "London",
  "Saint Petersburg", "Berlin", "Madrid",
  "Kyiv", "Rome", "Bucharest", "Paris")

population <- c(
  15.1e6, 12.5e6, 9e6, 5.4e6, 3.8e6,
  3.2e6, 3e6, 2.8e6, 2.2e6, 2.1e6)

area_km2 <- c(2576, 2561, 1572, 1439,
  891, 604, 839, 1285, 228, 105)
```

```
data.frame(
  cities = cities,
  population = population,
  area_km2 = area_km2
)
```

	cities	population	area_km2
## 1	Istanbul	15100000	2576
## 2	Moscow	12500000	2561
## 3	London	9000000	1572
## 4	Saint Petersburg	5400000	1439
## 5	Berlin	3800000	891
## 6	Madrid	3200000	604
## 7	Kyiv	3000000	839
## 8	Rome	2800000	1285
## 9	Bucharest	2200000	228
## 10	Paris	2100000	105

Tibbles

Tibbles are

a **modern reimaging of the data frame**. Tibbles are designed to be (as much as possible) **drop-in replacements** for data frames.

(Wickham, [Advanced R](#))

Have a look at [this book chapter](#) for a full list of the differences between data frames and tibbles and the advantages of using tibbles.

- tibbles have the same basic properties as data frames (named list of vectors)
- everything that you can do with data frames, you can do with tibbles



Tibbles

Tibbles are available from the `tibble` package.

Before we use tibbles, we need to install the package once using the function `install.packages`:

```
# This has to be done once  
install.packages("tibble")
```

Then, we need to load and attach the package to our current R session using `library`:

```
# This has to be done every time R restarts  
# Put it at the beginning of a script  
library(tibble)
```



Tibbles

Create a tibble using the `tibble()` function:

```
library(tibble)
```

```
tibble(  
  cities = cities,  
  population = population,  
  area_km2 = area_km2  
)
```

```
## # A tibble: 10 x 3  
##   cities      population area_km2  
##   <chr>          <dbl>    <dbl>  
## 1 Istanbul    15100000    2576  
## 2 Moscow      12500000    2561  
## 3 London       9000000    1572  
## 4 Saint Petersburg 5400000    1439  
## 5 Berlin       3800000     891  
## 6 Madrid       3200000     604  
## 7 Kyiv         3000000     839  
## 8 Rome         2800000    1285  
## 9 Bucharest    2200000     228  
## 10 Paris       2100000     105
```

Note: If you want to use a function from a package you can attach it using `library(package)` or you can use `package::function` to tell R where a function is from (e.g. `tibble::tibble()`). I will sometimes do this to clearly distinguish between base R and package functions.

Exploring tibbles

Look at the structure of an object using `str()`:

```
str(cities_tbl)
```

```
## tibble [10 x 3] (S3: tbl_df/tbl/data.frame)
##  $ cities      : chr [1:10] "Istanbul" "Moscow" "London" "Saint Petersburg" ...
##  $ population: num [1:10] 15100000 12500000 9000000 5400000 3800000 3200000 3000000 2800000
2200000 2100000
##  $ area_km2    : num [1:10] 2576 2561 1572 1439 891 ...
```

- This function shows you:
 - data type of object (`tbl_df/tbl/data.frame`)
 - extent of the data (10 rows times 3 columns)
 - column names and data types
- This function works for every R object and is very useful if code doesn't work and you don't know why

Exploring tibbles

How many rows?

```
nrow(cities_tbl)
```

```
## [1] 10
```

How many columns?

```
ncol(cities_tbl)
```

```
## [1] 3
```

What are the column headers?

```
names(cities_tbl)
```

```
## [1] "cities"      "population" "area_km2"
```

Exploring tibbles

Look at the entire table in a separate window with `view()`:

```
tibble::view(cities_tbl)
```

Get a quick summary of all columns:

```
summary(cities_tbl)
```

##	cities	population	area_km2
##	Length:10	Min. : 2100000	Min. : 105.0
##	Class :character	1st Qu.: 2850000	1st Qu.: 662.8
##	Mode :character	Median : 3500000	Median :1088.0
##		Mean : 5910000	Mean :1210.0
##		3rd Qu.: 8100000	3rd Qu.:1538.8
##		Max. :15100000	Max. :2576.0

Indexing tibbles

Indexing tibbles works similar to indexing vectors but with two dimensions instead of 1:

```
tibble [ row_index, col_index or col_name ]
```

- Missing row_index or col_index means *all rows* or *all columns* respectively.
- Indexing a tibble using `[]` always returns another tibble.

Indexing tibbles

```
# First row and first column  
cities_tbl[1, 1]
```

```
## # A tibble: 1 x 1  
##   cities  
##   <chr>  
## 1 Istanbul
```

This is the same as

```
cities_tbl[1, "cities"]
```

Indexing tibbles: rows

```
# rows 1 & 5, all columns:  
cities_tbl[c(1, 5), ]
```

Indexing tibbles: columns

```
# All rows, first 2 columns
cities_tbl[,1:2]
# same as
cities_tbl[,c("cities", "population")]
```

```
## # A tibble: 10 x 2
##   cities      population
##   <chr>         <dbl>
## 1 Istanbul    15100000
## 2 Moscow      12500000
## 3 London       9000000
## # ... with 7 more rows
```

Indexing tibbles: columns

Indexing columns by name is usually preferred to indexing by position

```
cities_tbl[ ,1:2] # okay  
cities_tbl[ ,c("cities", "population")] # better
```

Why?

- code is much easier to read
- code is more robust against
 - changes in column order
 - mistakes in the code (e.g. typos)

```
cities_tbl[ ,c(1,3)] # 3 instead of 2 -> wrong but no error  
cities_tbl[ ,c("cities", "popluation")] # typo -> wrong and error
```

💡 General rule: Good code produces errors when something unintended or wrong happens

Tibbles: Select columns with \$

Select an entire column from a tibble using `$` (this returns a vector instead of a tibble):

```
cities_tbl$cities
```

```
##      [1] "Istanbul"      "Moscow"         "London"         "Saint Petersburg" "Berlin"
##      [6] "Madrid"        "Kyiv"           "Rome"           "Bucharest"       "Paris"
```


Adding new columns

New columns can be added as vectors using the `$` operator. The vectors need to have the same length as the tibble has rows.

```
# add a country column
cities_tbl$country <- c(
  "Turkey", "Russia", "UK", "Russia", "Germany", "Spain",
  "Ukraine", "Italy", "Romania", "France"
)
```

```
## # A tibble: 10 x 4
##   cities      population area_km2 country
##   <chr>      <dbl>    <dbl> <chr>
## 1 Istanbul    15100000    2576 Turkey
## 2 Moscow      12500000    2561 Russia
## 3 London       9000000    1572 UK
## 4 Saint Petersburg 5400000    1439 Russia
## 5 Berlin       3800000     891 Germany
## 6 Madrid       3200000     604 Spain
## 7 Kyiv         3000000     839 Ukraine
## 8 Rome         2800000    1285 Italy
## 9 Bucharest    2200000     228 Romania
## 10 Paris       2100000     105 France
```

Adding new columns

Adding a new column **based on other columns**:

```
cities_tbl$density <- cities_tbl$population / cities_tbl$area_km2
```

```
## # A tibble: 10 x 5
##   cities      population area_km2 country density
##   <chr>          <dbl>    <dbl> <chr>    <dbl>
## 1 Istanbul    15100000    2576 Turkey    5862.
## 2 Moscow      12500000    2561 Russia    4881.
## 3 London       9000000    1572 UK        5725.
## 4 Saint Petersburg 5400000    1439 Russia    3753.
## 5 Berlin       3800000     891 Germany   4265.
## 6 Madrid       3200000     604 Spain     5298.
## 7 Kyiv         3000000     839 Ukraine   3576.
## 8 Rome         2800000    1285 Italy     2179.
## 9 Bucharest    2200000     228 Romania   9649.
## 10 Paris       2100000     105 France   20000
```

Adding new columns

Adding new columns **based on a condition**:

```
cities_tbl$category <- ifelse(cities_tbl$population > 5e6, # test  
                              "very large", # yes  
                              "large") # no
```

```
## # A tibble: 10 x 6  
##   cities      population area_km2 country density category  
##   <chr>          <dbl>    <dbl> <chr>    <dbl> <chr>  
## 1 Istanbul      15100000      2576 Turkey    5862. very large  
## 2 Moscow         12500000      2561 Russia    4881. very large  
## 3 London          9000000      1572 UK        5725. very large  
## 4 Saint Petersburg 5400000      1439 Russia    3753. very large  
## 5 Berlin          3800000       891 Germany   4265. large  
## 6 Madrid          3200000       604 Spain     5298. large  
## 7 Kyiv            3000000       839 Ukraine   3576. large  
## 8 Rome            2800000      1285 Italy     2179. large  
## 9 Bucharest       2200000       228 Romania   9649. large  
## 10 Paris          2100000       105 France    20000 large
```

Logical indexing

As with vectors, we can use us logical tests to **select rows** from a tibble. The basic structure is:

`tibble [logical indexing vector of length nrow(tibble), cols to select]`

Only rows that match `TRUE` in the indexing vector get selected.

```
# select all rows for which the population exceeds 15 M
cities_tbl[ cities_tbl$population > 15e6, ]
```

```
## # A tibble: 1 x 6
##   cities      population area_km2 country density category
##   <chr>         <dbl>    <dbl> <chr>    <dbl> <chr>
## 1 Istanbul    15100000      2576 Turkey    5862. very large
```

Logical indexing

What is happening in detail?

```
cities_tbl[cities_tbl$population > 15e6, ]
```

```
cities_tbl$population # vector with population
```

```
## [1] 15100000 12500000 9000000 5400000 3800000 3200000 3000000 2800000 2200000 2100000
```

```
cities_tbl$population > 15e6 # logical vector after relational test
```

```
## [1] TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

So we actually subset the tibble like this:

```
cities_tbl[c(TRUE, FALSE, FALSE, FALSE, FALSE,  
             FALSE, FALSE, FALSE, FALSE, FALSE), ]
```

Logical indexing

Some more examples:

```
cities_tbl[cities_tbl$cities != "Istanbul" & cities_tbl$area_km2 > 1000, ]
```

```
## # A tibble: 4 x 6
##   cities      population area_km2 country density category
##   <chr>          <dbl>    <dbl> <chr>    <dbl> <chr>
## 1 Moscow      12500000      2561 Russia   4881. very large
## 2 London       9000000      1572 UK       5725. very large
## 3 Saint Petersburg 5400000      1439 Russia   3753. very large
## 4 Rome        2800000      1285 Italy    2179. large
```

Logical indexing

Some more examples:

```
cities_tbl[cities_tbl$cities != "Istanbul" & cities_tbl$area_km2 > 1000, ]  
cities_tbl[cities_tbl$cities %in% c("Istanbul", "Madrid", "Paris"), ]
```

```
## # A tibble: 3 x 6  
##   cities      population area_km2 country density category  
##   <chr>          <dbl>    <dbl> <chr>    <dbl> <chr>  
## 1 Istanbul    15100000    2576 Turkey    5862. very large  
## 2 Madrid      3200000     604 Spain     5298. large  
## 3 Paris       2100000     105 France    20000 large
```

Changing values in tibbles

As with vectors, we can use indexing to change specific values in the tibble.

Idea:

1. Index row and column or the values you want to change
2. Overwrite them using the assignment operator `<-`

For example, the population of Madrid changed but we don't know the new population.

We can replace the population value from Madrid with NA:

```
cities_tbl[cities_tbl$cities == "Madrid", "population"] <- NA
```

```
cities_tbl[6, ] # row 6 is Madrid
```

```
## # A tibble: 1 x 6
##   cities population area_km2 country density category
##   <chr>         <dbl>    <dbl> <chr>    <dbl> <chr>
## 1 Madrid           NA        604 Spain    5298. large
```


Now you

Task 3: Tibbles

Find the task description [here](#)

Summary I

data frames and tibbles

- can be used to represent tables in R
- are pretty similar, however tibbles are slightly convenient and modern
- are **named lists of vectors of the same length**
 - every column is a vector
 - columns have a header which is the name of the vector in the list
 - within one column, values are of same data type
 - every column has the same length

tibbles

- to use tibbles, install the package once with `install.packages("tibble")`
- put `library(tibble)` at the beginning of your script to load package

Summary II

Creating tibbles and data frames

```
# data frame
data.frame(
  a = 1:3,
  b = c("a", "b", "c"),
  c = c(TRUE, FALSE, FALSE)
)
# tibble
tibble(
  a = 1:3,
  b = c("a", "b", "c"),
  c = c(TRUE, FALSE, FALSE)
)
# convert data frame to tibble
as_tibble(df)
```

Summary III

Looking at tibble structure

```
# structure of tibble and data types of columns  
str(tbl)  
# number of rows  
nrow(tbl)  
# number of columns  
ncol(tbl)  
# column headers  
names(tbl)  
# look at the data in a new window  
tibble::view(tbl)  
# summary of values from each column  
summary(tbl)
```

Summary IV

Indexing tibbles and selecting columns

Return result as tibble:

```
# rows and columns by position
tbl[1:3, c(1, 3)]
tbl[1:3, ] # all columns
tbl[, 3] # column 3, all rows
tbl[3] # same as above

# columns by name
tbl[, c("colA", "colB")]
tbl[c("colA", "colB")]
```

Return result as vector:

```
tbl$colA # select colA
```

Summary V

Logical indexing to select rows

- Index tibbles with a vector of the same length
- Use
 - logical and relational operators
 - `%in%`

```
tbl[tbl$colA == 5, ] # only rows where colA is 5 (all columns)
tbl[tbl$colA >= 10, ]
tbl[tbl$colB %in% c("hello", "cat", "apple"), ] # only rows where colB is "hello", "cat" or "apple"
tbl[tbl$colB == "hello" | tbl$colB == "cat" | tbl$colB == "apple", ] # same as above
```

Logical indexing to select columns

```
select_cols <- c("colA", "colB", "colC")
tbl[names(tbl) %in% select_cols]
tbl[c("colA", "colB", "colC")] # same as above
```

Summary VI

Add and remove columns

```
tbl$new_col <- c(1, 2, 3)
tbl$new_col <- tbl$colA / tbl$colB # new column based on other columns
tbl$new_col <- NULL # remove new_col
```