Week 6: Data Wrangling in R

POP77001 Computer Programming for Social Scientists

Tom Paskhalis

17 October 2022

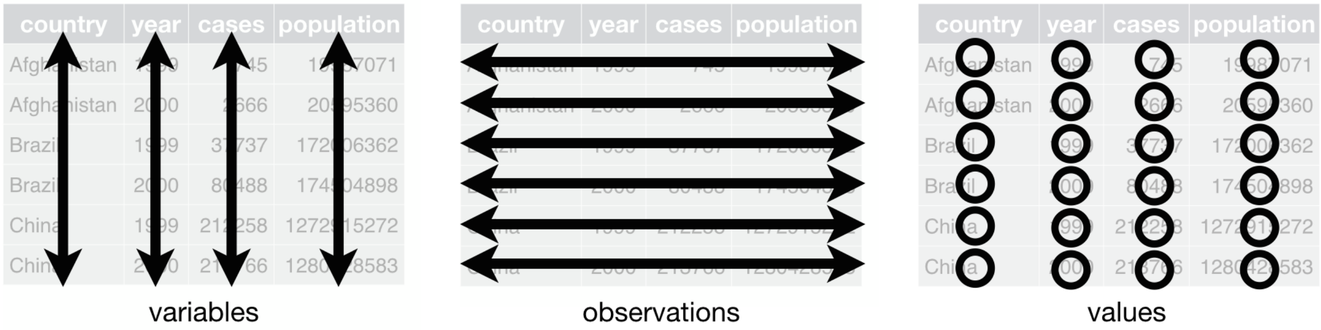
Module website: tinyurl.com/POP77001

Overview

- Data frames in base R
- Alternatives to data frames
- `tidyverse` packages
- Working with tabular data
- Data input and output
- Summary statistics

Tidy data

- Tidy data is a specific subset of rectangular data, where:
  - Each variable is in a column
  - Each observation is in a row
  - Each value is in a cell



Source: [R for Data Science]

Data frames

- Data frame is one of the object types available in base R.
- Despite their matrix-like appearance, data frames are lists of equal-sized vectors.
- Data frames can be created with `data.frame()` function with named vectors as input.

Data frames

- Data frame is one of the object types available in base R.
- Despite their matrix-like appearance, data frames are lists of equal-sized vectors.
- Data frames can be created with `data.frame()` function with named vectors as input.

In [2]:
```
df <- data.frame(
    x = 1:4,
    y = c("a", "b", "c", "d"),
    z = c(TRUE, FALSE, FALSE, TRUE)
)
df
```

```
  x y z
1 1 a  TRUE
2 2 b FALSE
3 3 c FALSE
4 4 d  TRUE
```

Data frame example

Data frame example

In [3]:

```
# str() function applied to data frame is useful in determining variabl
str(df)
```

```
'data.frame':	4 obs. of  3 variables:
 $ x: int  1 2 3 4
 $ y: chr  "a" "b" "c" "d"
 $ z: logi  TRUE FALSE FALSE TRUE
```

Data frame example

In [3]:

```
# str() function applied to data frame is useful in determining variabl
str(df)
```

```
'data.frame':	4 obs. of  3 variables:
 $ x: int  1 2 3 4
 $ y: chr  "a" "b" "c" "d"
 $ z: logi  TRUE FALSE FALSE TRUE
```

In [4]:

```
# dim() function behaves similar to matrix, showing N rows and N column
dim(df)
```

```
[1] 4 3
```

Data frame example

In [3]:
```
# str() function applied to data frame is useful in determining variabl
str(df)
```

```
'data.frame':	4 obs. of  3 variables:
 $ x: int  1 2 3 4
 $ y: chr  "a" "b" "c" "d"
 $ z: logi  TRUE FALSE FALSE TRUE
```

In [4]:
```
# dim() function behaves similar to matrix, showing N rows and N colum
dim(df)
```

```
[1] 4 3
```

In [5]:
```
# In contrast to matrix length() of data frame displays the length of u
length(df)
```

```
[1] 3
```

Creating data frame example

Creating data frame example

In [6]:

```
l <- list(x = 1:5, y = letters[1:5], z = rep(c(TRUE, FALSE), length.out
l
```

```
$x
[1] 1 2 3 4 5

$y
[1] "a" "b" "c" "d" "e"

$z
[1]  TRUE FALSE  TRUE FALSE  TRUE
```

Creating data frame example

In [6]:

```
l <- list(x = 1:5, y = letters[1:5], z = rep(c(TRUE, FALSE), length.out
l
```

```
$x
[1] 1 2 3 4 5

$y
[1] "a" "b" "c" "d" "e"

$z
[1]  TRUE FALSE  TRUE FALSE  TRUE
```

In [7]:

```
df <- data.frame(l)
df
```

```
  x y z
1 1 a  TRUE
2 2 b FALSE
3 3 c  TRUE
4 4 d FALSE
5 5 e  TRUE
```

Creating data frame example

In [6]:

```r
l <- list(x = 1:5, y = letters[1:5], z = rep(c(TRUE, FALSE), length.out
l
```

```
$x
[1] 1 2 3 4 5

$y
[1] "a" "b" "c" "d" "e"

$z
[1]  TRUE FALSE  TRUE FALSE  TRUE
```

In [7]:

```r
df <- data.frame(l)
df
```

```
  x y z
1 1 a  TRUE
2 2 b FALSE
3 3 c  TRUE
4 4 d FALSE
5 5 e  TRUE
```

In [8]:

```r
str(df)
```

```
'data.frame':	5 obs. of  3 variables:
```

```
 $ x: int  1 2 3 4 5
 $ y: chr  "a" "b" "c" "d" ...
 $ z: logi  TRUE FALSE TRUE FALSE TRUE
```

Subsetting data frame

- In subsetting data frames the techniques of subsetting matrices and lists are combined
- If you subset with a single vector, it behaves as a list
- If you subset with two vectors, it behaves as a matrix

Subsetting data frame example

Subsetting data frame example

In [9]:

```
# Like a list
df[c("x", "z")]
```

```
  x     z
1 1  TRUE
2 2 FALSE
3 3  TRUE
4 4 FALSE
5 5  TRUE
```

Subsetting data frame example

In [9]:
```
# Like a list
df[c("x", "z")]
```

```
  x     z
1 1  TRUE
2 2 FALSE
3 3  TRUE
4 4 FALSE
5 5  TRUE
```

In [10]:
```
# Like a matrix
df[,c("x", "z")]
```

```
  x     z
1 1  TRUE
2 2 FALSE
3 3  TRUE
4 4 FALSE
5 5  TRUE
```

Subsetting data frame example

In [9]:
```
# Like a list
df[c("x", "z")]
```

```
  x     z
1 1  TRUE
2 2 FALSE
3 3  TRUE
4 4 FALSE
5 5  TRUE
```

In [10]:
```
# Like a matrix
df[,c("x", "z")]
```

```
  x     z
1 1  TRUE
2 2 FALSE
3 3  TRUE
4 4 FALSE
5 5  TRUE
```

In [11]:
```
df[df$y == "b",]
```

```
  x y     z
2 2 b FALSE
```

Building data frame

- `rbind()` (row bind) - appends a row to data frame
- `cbind()` (column bind) - appends a column to data frame
- Both require compatible sizes (number of rows/columns)

Building data frame examples

- Adding columns

Building data frame examples

- Adding columns

In [12]:
```
rand <- rnorm(5)
rand
```

```
[1] -1.6395385 -0.6401171  1.4880066 -0.4978420 -1.3442429
```

Building data frame examples

- Adding columns

In [12]:

```
rand <- rnorm(5)
rand
```

```
[1] -1.6395385 -0.6401171  1.4880066 -0.4978420 -1.3442429
```

In [13]:

```
df <- cbind(df, rand)
df
```

```
  x y z       rand
1 1 a  TRUE -1.6395385
2 2 b FALSE -0.6401171
3 3 c  TRUE  1.4880066
4 4 d FALSE -0.4978420
5 5 e  TRUE -1.3442429
```

Building data frame examples continued

- Adding rows

Building data frame examples continued

- Adding rows

In [14]:

```
# Note that a row has to be a list as it contains different data types
r <- list(6, letters[6], FALSE, rnorm(1))
r
```

```
[[1]]
[1] 6

[[2]]
[1] "f"

[[3]]
[1] FALSE

[[4]]
[1] -0.2291225
```

Building data frame examples continued

- Adding rows

In [14]:
```
# Note that a row has to be a list as it contains different data types
r <- list(6, letters[6], FALSE, rnorm(1))
r
```

```
[[1]]
[1] 6

[[2]]
[1] "f"

[[3]]
[1] FALSE

[[4]]
[1] -0.2291225
```

In [15]:
```
df <- rbind(df, r)
df
```

```
  x y z      rand
1 1 a  TRUE -1.6395385
2 2 b FALSE -0.6401171
3 3 c  TRUE  1.4880066
```

```
4 4 d FALSE -0.4978420
5 5 e  TRUE -1.3442429
6 6 f FALSE -0.2291225
```

Issues with data frame

- While very versatile (and available out-of-the-box) data frames have their drawbacks:
    - Individual cells (observations) cannot themselves be lists;
    - Somewhat limited (and inconsistent) data manipulation functions;
    - Memory inefficient (**copy-on-modify** semantics);
    - No parallelisation.

Alternatives to data frame

- Two major alternatives to/enhanced versions of data frames are:
    - `tibble` from `tibble` package (part of `tidyverse` package ecosystem)
    - `data.table` from `data.table`
- `tibble` provides features enhancing user experience (readability, ease of manipulation)
- `data.table` provides speed

Data table - fast data frame

- As opposed to data frames, data tables are updated by reference.
- This frees up a lot of RAM for big data!
- It provides low-level parallelism.
- SQL-like operations for data manipulation.
- Has no external dependencies (other than base R itself)

Data table - fast data frame

- As opposed to data frames, data tables are updated by reference.
- This frees up a lot of RAM for big data!
- It provides low-level parallelism.
- SQL-like operations for data manipulation.
- Has no external dependencies (other than base R itself)

In [16]:

```
dt <- data.table::data.table(
    x = 1:4,
    y = c("a", "b", "c", "d"),
    z = c(TRUE, FALSE, FALSE, TRUE)
)
dt
```

```
  x y     z
1 1 a  TRUE
2 2 b FALSE
3 3 c FALSE
4 4 d  TRUE
```

`tidyverse` packages

- `tidyverse` package ecosystem - rich collection of data science packages
- Designed with consistent interfaces and generally higher usability than base R function
- Notable packages:
    - `readr` - data input/output (also `readxl` for spreadsheets, `haven` for SPSS/Stata)
    - `dplyr` - data manipulation (also `tidyr` for pivoting)
    - `ggplot2` - data visualisation
    - `lubridate` - working with dates and time
    - `tibble` - enhanced data frame

```
install.packages("tidyverse")
```

Tibble - user-friendly data frame

- Tibbles are designed to be backward compatible with base R data frames
- Console printing of tibbles is cleaner (prettified, only first 10 rows by default)
- Tibbles can have columns that themselves contain lists as elements
- Tibbles can be created with `tibble::tibble()` function
- Or objects can be coerced into a tibble using `tibble::as_tibble()` function

Tibble - user-friendly data frame

- Tibbles are designed to be backward compatible with base R data frames
- Console printing of tibbles is cleaner (prettified, only first 10 rows by default)
- Tibbles can have columns that themselves contain lists as elements
- Tibbles can be created with `tibble::tibble()` function
- Or objects can be coerced into a tibble using `tibble::as_tibble()` function

In [17]:

```r
tb <- tibble::tibble(
    x = 1:4,
    y = c("a", "b", "c", "d"),
    z = c(TRUE, FALSE, FALSE, TRUE)
)
tb
```

```
  x y z
1 1 a  TRUE
2 2 b FALSE
3 3 c FALSE
4 4 d  TRUE
```

Tibbles work (mostly) like data frames

Tibbles work (mostly) like data frames

In [18]:
```
str(tb)
```

```
tibble [4 × 3] (S3: tbl_df/tbl/data.frame)
 $ x: int [1:4] 1 2 3 4
 $ y: chr [1:4] "a" "b" "c" "d"
 $ z: logi [1:4] TRUE FALSE FALSE TRUE
```

Tibbles work (mostly) like data frames

In [18]:
```
str(tb)
```

```
tibble [4 × 3] (S3: tbl_df/tbl/data.frame)
 $ x: int [1:4] 1 2 3 4
 $ y: chr [1:4] "a" "b" "c" "d"
 $ z: logi [1:4] TRUE FALSE FALSE TRUE
```

In [19]:
```
dim(tb)
```

```
[1] 4 3
```

Tibbles work (mostly) like data frames

In [18]:
```
str(tb)
```

```
tibble [4 × 3] (S3: tbl_df/tbl/data.frame)
 $ x: int [1:4] 1 2 3 4
 $ y: chr [1:4] "a" "b" "c" "d"
 $ z: logi [1:4] TRUE FALSE FALSE TRUE
```

In [19]:
```
dim(tb)
```

```
[1] 4 3
```

In [20]:
```
tb[c("x", "z")]
```

```
  x z
1 1  TRUE
2 2 FALSE
3 3 FALSE
4 4  TRUE
```

Tibbles work (mostly) like data frames

In [18]:
```
str(tb)
```

```
tibble [4 × 3] (S3: tbl_df/tbl/data.frame)
 $ x: int [1:4] 1 2 3 4
 $ y: chr [1:4] "a" "b" "c" "d"
 $ z: logi [1:4] TRUE FALSE FALSE TRUE
```

In [19]:
```
dim(tb)
```

```
[1] 4 3
```

In [20]:
```
tb[c("x", "z")]
```

```
  x z
1 1  TRUE
2 2 FALSE
3 3 FALSE
4 4  TRUE
```

In [21]:
```
tb[tb$y == "b",]
```

```
  x y z
1 2 b FALSE
```

Manipulating columns in base R

- Adding/modifying columns

Manipulating columns in base R

- Adding/modifying columns

In [22]:

```
# New columns can also be created/modified by assignment (if the RHS ot
tb["r"] <- rnorm(4)
tb
```

```
  x y z     r
1 1 a  TRUE -0.63905096
2 2 b FALSE -0.40466580
3 3 c FALSE  0.49230918
4 4 d  TRUE  0.09646717
```

Manipulating columns in base R

- Adding/modifying columns

In [22]:

```
# New columns can also be created/modified by assignment (if the RHS ob
tb["r"] <- rnorm(4)
tb
```

```
  x y z     r
1 1 a  TRUE -0.63905096
2 2 b FALSE -0.40466580
3 3 c FALSE  0.49230918
4 4 d  TRUE  0.09646717
```

In [23]:

```
# Individual columns can also be selected with $ operator
tb$r <- tb$r + 5
tb
```

```
  x y z     r
1 1 a  TRUE 4.360949
2 2 b FALSE 4.595334
3 3 c FALSE 5.492309
4 4 d  TRUE 5.096467
```

Manipulating columns in base R continued

- Renaming columns

Manipulating columns in base R continued

- Renaming columns

In [24]:

```
# names() attribute for data frames/tibbles contains column names
names(tb)
```

```
[1] "x" "y" "z" "r"
```

Manipulating columns in base R continued

- Renaming columns

In [24]:

```
# names() attribute for data frames/tibbles contains column names
names(tb)
```

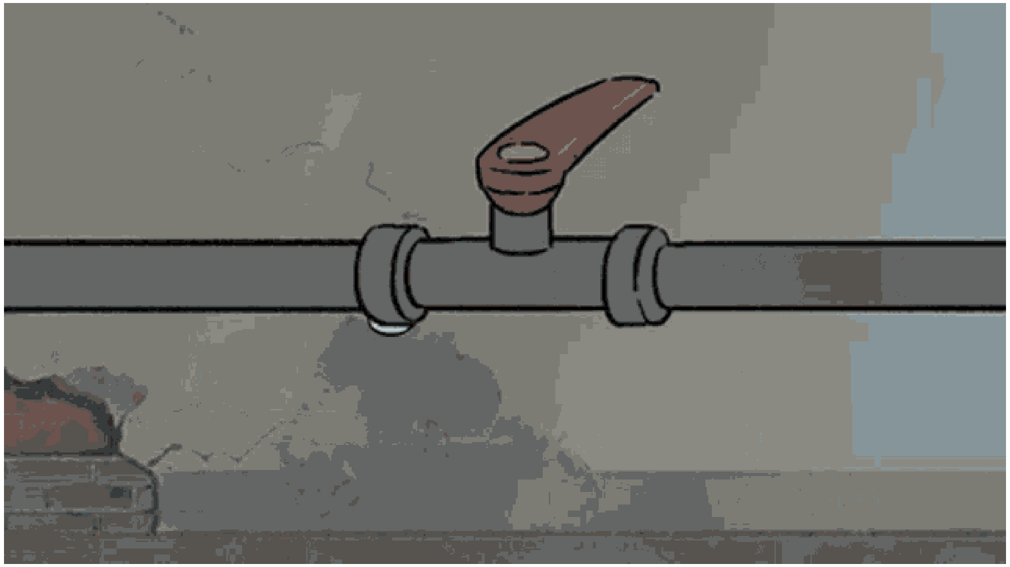
```
[1] "x" "y" "z" "r"
```

In [25]:

```
names(tb)[4] <- "rand"
tb
```

	x	y	z	rand
1	1	a	TRUE	4.360949
2	2	b	FALSE	4.595334
3	3	c	FALSE	5.492309
4	4	d	TRUE	5.096467

Data preparation



Source: Tenor

Data manipulation with `dplyr`

- `dplyr` - is one of the core packages for data manipulation in `tidyverse`
- Its principal functions are:
  - `filter()` - subset rows from data
  - `mutate()` - add new/modify existing variables
  - `rename()` - rename existing variable
  - `select()` - subset columns from data
  - `arrange()` - order data by some variable
- For data summary:
  - `group_by()` - aggregate data by some variable
  - `summarise()` - create a summary of aggregated variables

Data manipulation with `dplyr`

- `dplyr` - is one of the core packages for data manipulation in `tidyverse`

- Its principal functions are:

    - `filter()` - subset rows from data
    - `mutate()` - add new/modify existing variables
    - `rename()` - rename existing variable
    - `select()` - subset columns from data
    - `arrange()` - order data by some variable

- For data summary:

    - `group_by()` - aggregate data by some variable
    - `summarise()` - create a summary of aggregated variables

In [26]:

```
library("dplyr")
```

```
Attaching package: ‘dplyr’


The following objects are masked from ‘package:stats’:
```

```
    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

Data manipulation with `dplyr` examples

- Subsetting

Data manipulation with `dplyr` examples

- Subsetting

In [27]:
```
dplyr::filter(tb, y == 'b', z == FALSE)
```

```
  x y z     rand
1 2 b FALSE 4.595334
```

Data manipulation with `dplyr` examples

- Subsetting

In [27]:
```
dplyr::filter(tb, y == 'b', z == FALSE)
```

```
  x y z      rand
1 2 b FALSE 4.595334
```

In [28]:
```
# Note that dplyr functions do not require enquoted variable names
dplyr::select(tb, x, z)
```

```
  x z
1 1  TRUE
2 2 FALSE
3 3 FALSE
4 4  TRUE
```

Data manipulation with `dplyr` examples

- Subsetting

In [27]:
```
dplyr::filter(tb, y == 'b', z == FALSE)
```

```
  x y z     rand
1 2 b FALSE 4.595334
```

In [28]:
```
# Note that dplyr functions do not require enquoted variable names
dplyr::select(tb, x, z)
```

```
  x z
1 1  TRUE
2 2 FALSE
3 3 FALSE
4 4  TRUE
```

In [29]:
```
# We can also use helpful tidyselect functions for more complex rules
dplyr::select(tb, tidyselect::starts_with('r'))
```

```
  rand
1 4.360949
2 4.595334
3 5.492309
4 5.096467
```

Data manipulation with `dplyr` examples continued

- Renaming/modifying columns

Data manipulation with `dplyr` examples continued

- Renaming/modifying columns

In [30]:

```
# Data is not modified in-place, you need to re-assign the results
tb <- dplyr::rename(tb, random = rand)
```

Data manipulation with `dplyr` examples continued

- Renaming/modifying columns

In [30]:
```r
# Data is not modified in-place, you need to re-assign the results
tb <- dplyr::rename(tb, random = rand)
```

In [31]:
```r
dplyr::mutate(tb, random_8plus = ifelse(random >= 8, TRUE, FALSE))
```

```
  x y z     random   random_8plus
1 1 a  TRUE 4.360949 FALSE
2 2 b FALSE 4.595334 FALSE
3 3 c FALSE 5.492309 FALSE
4 4 d  TRUE 5.096467 FALSE
```

`%>%` operator

- Users of `tidyverse` packages are encouraged to use pipe operator `%>%`
- It allows to chain data transformations without creating intermediate variables
- It passes the result of the previous operation as a first first argument to the next
- Base R now also includes its own pipe operator `|>` but it is still relatively uncommon

```
<result> <- <input> %>%
  <function_name>(., arg_1, arg_2, ..., arg_n)

<result> <- <input> %>%
  <function_name>(arg_1, arg_2, ..., arg_n)
```

%>% operator examples

`%>%` operator examples

In [32]:
```
tb
```

```
  x y z     random
1 1 a  TRUE 4.360949
2 2 b FALSE 4.595334
3 3 c FALSE 5.492309
4 4 d  TRUE 5.096467
```

`%>%` operator examples

In [32]:

```
tb
```

```
  x y z     random
1 1 a  TRUE 4.360949
2 2 b FALSE 4.595334
3 3 c FALSE 5.492309
4 4 d  TRUE 5.096467
```

In [33]:

```
tb <- tb %>%
  dplyr::mutate(random_2 = rnorm(4)) %>%
  dplyr::filter(z == FALSE)
```

`%>%` operator examples

In [32]:
```
tb
```

```
  x y z     random
1 1 a  TRUE 4.360949
2 2 b FALSE 4.595334
3 3 c FALSE 5.492309
4 4 d  TRUE 5.096467
```

In [33]:
```
tb <- tb %>%
  dplyr::mutate(random_2 = rnorm(4)) %>%
  dplyr::filter(z == FALSE)
```

In [34]:
```
tb
```

```
  x y z     random   random_2
1 2 b FALSE 4.595334 -0.9099916
2 3 c FALSE 5.492309 -0.1632015
```

`%>%` operator vs built-in `|>` operator

- Since R version 4.1.0 (mid-2021), there is a built-in `|>` pipe operator

`%>%` operator vs built-in `|>` operator

- Since R version 4.1.0 (mid-2021), there is a built-in `|>` pipe operator

In [35]:

```
# Pipe %>% can also be used with non-dplyr functions
tb$x %>% .[2]
```

```
[1] 3
```

`%>%` operator vs built-in `|>` operator

- Since R version 4.1.0 (mid-2021), there is a built-in `|>` pipe operator

In [35]:
```
# Pipe %>% can also be used with non-dplyr functions
tb$x %>% .[2]
```

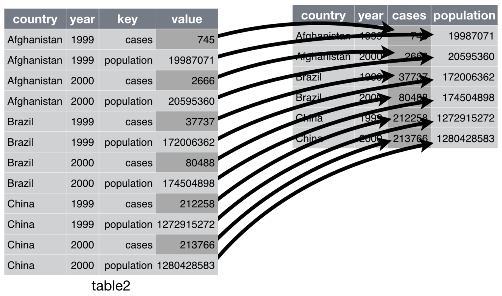
```
[1] 3
```

In [36]:
```
# Base R pipe operator |> is more restrictive (e.g. tb$x |> `[`(2) does
tb |> nrow()
```

```
[1] 2
```

Pivoting data

- Sometimes you want to pivot you data by:
  - Spreading some variable across columns (`tidyr::pivot_wider()`)
  - Gathering some columns in a variable pair (`tidyr::pivot_longer()`)



pivot_wider()	pivot_longer()

Source: R for Data Science

Pivoting data example

Pivoting data example

In [37]:

```
tb2 <- tibble::tibble(
  country = c("Afghanistan", "Brazil"),
  `1999` = c(745, 2666),
  `2000` = c(37737, 80488)
)
tb2
```

```
  country     1999 2000
1 Afghanistan  745 37737
2 Brazil      2666 80488
```

Pivoting data example

In [37]:

```
tb2 <- tibble::tibble(
  country = c("Afghanistan", "Brazil"),
  `1999` = c(745, 2666),
  `2000` = c(37737, 80488)
)
tb2
```

```
  country     1999 2000
1 Afghanistan  745 37737
2 Brazil      2666 80488
```

In [38]:

```
tb2 <- tb2 %>%
  # Note that pivoting functions come 'tidyr' package
  tidyr::pivot_longer(cols = c("1999", "2000"), names_to = "year", valu
tb2
```

```
  country     year cases
1 Afghanistan 1999   745
2 Afghanistan 2000 37737
3 Brazil      1999  2666
4 Brazil      2000 80488
```

Pivoting data example

In [37]:
```
tb2 <- tibble::tibble(
  country = c("Afghanistan", "Brazil"),
  `1999` = c(745, 2666),
  `2000` = c(37737, 80488)
)
tb2
```

```
  country     1999 2000
1 Afghanistan  745 37737
2 Brazil      2666 80488
```

In [38]:
```
tb2 <- tb2 %>%
  # Note that pivoting functions come 'tidyr' package
  tidyr::pivot_longer(cols = c("1999", "2000"), names_to = "year", valu
tb2
```

```
  country     year cases
1 Afghanistan 1999   745
2 Afghanistan 2000 37737
3 Brazil      1999  2666
4 Brazil      2000 80488
```

In [39]:
```
tb2 <- tb2 %>%
  tidyr::pivot_wider(names_from = "year", values_from = "cases")
tb2
```

```
  country     1999 2000
1 Afghanistan  745 37737
2 Brazil      2666 80488
```

Data formats in R

- `.csv` (Comma-separated value) files for storing tabular data
  - Standard file format for storing data that is highly interoperable across systems
  - Human-readable and can be opened in a simple text processor
- `.rds` (R data serialization) files allow to store single R object
  - Can store arbitrary R objects (e.g. fitted model), similar to Python `pickle`
  - Offers data compression
  - Only works within R
- `.rda` (R data) files for saving and loading multiple R objects
  - Offers data compression
  - Compares unfavourably to rds and, generally, should be avoided
- `.feather` / `.parquet` - big data formats associated with Apache Hadoop ecosystem
  - Cutting-edge performance (compression and read/write access)
  - Not human-readable
  - Relatively new, could be an overkill for some tasks

Functions for data I/O

- `.csv` (Comma-separated value)
    - `read.csv()` / `write.csv()` - base R functions
    - `readr::read_csv()` / `readr::write_csv()` - functions from `readr` package in `tidyverse`
- `.rds` (R data serialization)
    - `readRDS()` / `writeRDS()` - base R functions
    - `readr::read_rds()` / `readr::write_rds()` - functions from `readr` (no default compression)
- `.rda` (R data)
    - `save()` / `load()` - base R functions
- `.feather` / `.parquet`
    - `arrow::read_feather()` / `arrow::write_feather()` - functions from
    - `arrow::read_parquet()` / `arrow::write_parquet()` - `arrow` package in Apache Arrow

Reading data in R example

Reading data in R example

In [41]:

```
# We are skipping the first row as this dataset has a composite header
kaggle2021 <- readr::read_csv('../data/kaggle_survey_2021_responses.csv
```

```
Rows: 25973 Columns: 369

── Column specification ────────────────────────────────────────
────────────────────────────────────────────────────────────────
────────────────────────────────────────────────────────────────
─────────────────────

Delimiter: ","
chr (360): What is your age (# years)?, What is your gender? -
 Selected Choi...
dbl   (1): Duration (in seconds)
lgl   (8): In the next 2 years, do you hope to become more fami
liar with any...


i Use `spec()` to retrieve the full column specification for thi
s data.
i Specify the column types or set `show_col_types = FALSE` to qu
iet this message.
```

Reading data in R example

In [41]:
```
# We are skipping the first row as this dataset has a composite header
kaggle2021 <- readr::read_csv('../data/kaggle_survey_2021_responses.csv
```

```
Rows: 25973 Columns: 369

── Column specification ──────────────────────────────────────────
──────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────
──────────────────────

Delimiter: ","
chr (360): What is your age (# years)?, What is your gender? -
 Selected Choi...
dbl   (1): Duration (in seconds)
lgl   (8): In the next 2 years, do you hope to become more fami
liar with any...


i Use `spec()` to retrieve the full column specification for thi
s data.
i Specify the column types or set `show_col_types = FALSE` to qu
iet this message.
```

In [42]:
```
head(kaggle2021[,1:10])
```

A tibble: 6 × 10

Duration (in seconds)	What is your age (# years)?	What is your gender? - Selected Choice	In which country do you currently reside?	What is the highest level of formal education that you have attained or plan to attain within the next 2 years?	Select the title most similar to your current role (or most recent title if retired): - Selected Choice	For how ma years ha you be writing co and programmin
<dbl>	<chr>	<chr>	<chr>	<chr>	<chr>	<ch
910	50-54	Man	India	Bachelor’s degree	Other	5-10 ye
784	50-54	Man	Indonesia	Master’s degree	Program/Project Manager	20+ ye
924	22-24	Man	Pakistan	Master’s degree	Software Engineer	1-3 ye
575	45-49	Man	Mexico	Doctoral degree	Research Scientist	20+ ye
781	45-49	Man	India	Doctoral degree	Other	< 1 ye

Summarizing numeric variables

Summarizing numeric variables

In [43]:
```
# Note that summary() as opposed to pandas' describe() gives summary fo
summary(kaggle2021[,1:10])
```

```
 Duration (in seconds) What is your age (# years)?
 Min.   :    120       Length:25973
 1st Qu.:    443       Class :character
 Median :    656       Mode  :character
 Mean   :  11055
 3rd Qu.:   1038
 Max.   :2488653
 What is your gender? - Selected Choice
 Length:25973
 Class :character
 Mode  :character




 In which country do you currently reside?
 Length:25973
 Class :character
 Mode  :character




 What is the highest level of formal education that you have at
tained or plan to attain within the next 2 years?
```

```
 Length:25973
 Class :character
 Mode  :character




 Select the title most similar to your current role (or most re
cent title if retired): - Selected Choice
 Length:25973
 Class :character
 Mode  :character




 For how many years have you been writing code and/or programmi
ng?
 Length:25973
 Class :character
 Mode  :character




 What programming languages do you use on a regular basis? (Sel
ect all that apply) - Selected Choice - Python
 Length:25973
 Class :character
 Mode  :character




 What programming languages do you use on a regular basis? (Sel
```

```
ect all that apply) - Selected Choice - R
 Length:25973
 Class :character
 Mode  :character



 What programming languages do you use on a regular basis? (Sel
ect all that apply) - Selected Choice - SQL
 Length:25973
 Class :character
 Mode  :character
```

Summarizing categorical variables

Summarizing categorical variables

In [44]:

```
# table() function is rather flexible in allowing to tabulate a single
table(kaggle2021[3])
```

```
                        Man                    Nonbinary          Prefer no
t to say
                      20598                           88
355
Prefer to self-describe                            Woman
                         42                         4890
```

Summarizing categorical variables

In [44]:
```
# table() function is rather flexible in allowing to tabulate a single
table(kaggle2021[3])
```

```
                     Man                  Nonbinary            Prefer no
t to say
                   20598                         88
355
Prefer to self-describe                       Woman
                      42                       4890
```

In [45]:
```
# Wrapping it inside prop.table() gives proportions of each category
prop.table(table(kaggle2021[3]))
```

```
                     Man                  Nonbinary            Prefer no
t to say
             0.793054326                0.003388134                  0.0
13668040
Prefer to self-describe                       Woman
             0.001617064                0.188272437
```

Summarizing categorical variables

In [44]:

```
# table() function is rather flexible in allowing to tabulate a single
table(kaggle2021[3])
```

```
                    Man               Nonbinary          Prefer no
t to say
                  20598                      88
355
Prefer to self-describe                   Woman
                     42                    4890
```

In [45]:

```
# Wrapping it inside prop.table() gives proportions of each category
prop.table(table(kaggle2021[3]))
```

```
                    Man               Nonbinary          Prefer no
t to say
            0.793054326             0.003388134                0.0
13668040
Prefer to self-describe                   Woman
            0.001617064             0.188272437
```

In [46]:

```
# Wrapping it inside sort() gives value sorting, as opposed to alphabet
sort(table(kaggle2021[3]), decreasing = TRUE)[1]
```

Man: 20598

Next

- Tutorial: Working with data in R
- Assignment 2: Due at 12:00 on Monday, 24th October (submission on Blackboard)
- Next week: Reading week
- After reading week: Python 🐍