UNIVERSITY OF CAMBRIDGE
Bioinformatics Training

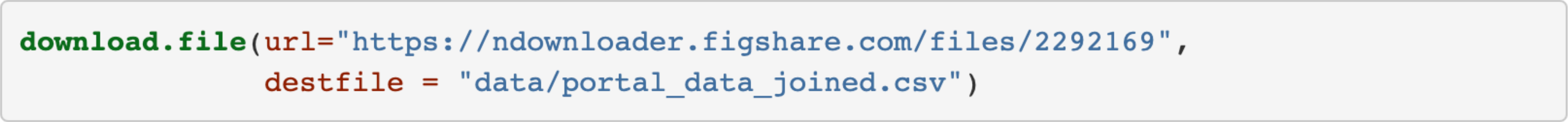
Introduction to R
for Biologists

STARTING WITH DATA

Author: Gita Yadav

Starting with Data

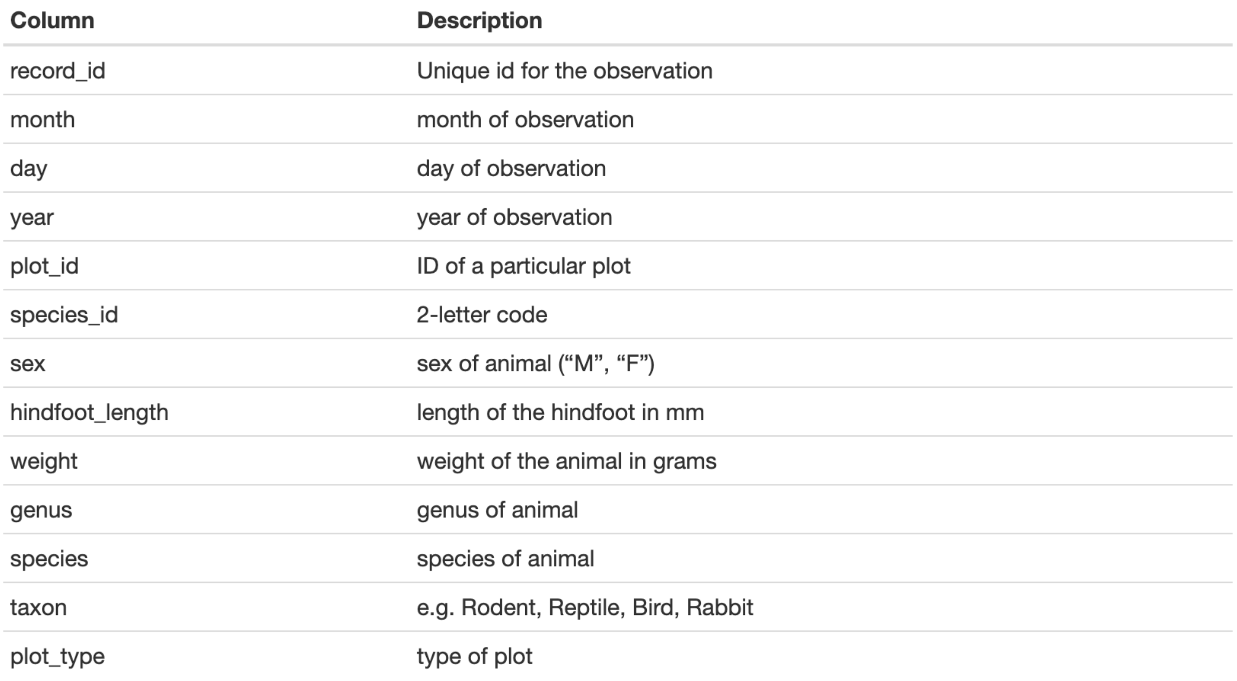
- Critical to Understand Your Data!
- Download the data and Look at it
- Use R function download.file() to download the CSV file that contains the data



```
download.file(url="https://ndownloader.figshare.com/files/2292169",
              destfile = "data/portal_data_joined.csv")
```

View Data

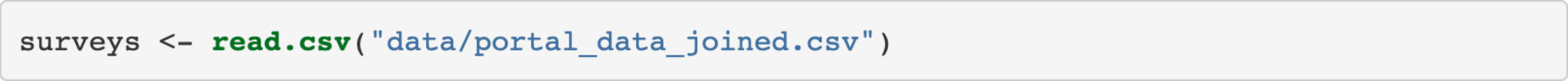
- Go to **Files** section in **RStudio**,
- Click on the filename in **data** folder
- Click **View** File



Column	Description
record_id	Unique id for the observation
month	month of observation
day	day of observation
year	year of observation
plot_id	ID of a particular plot
species_id	2-letter code
sex	sex of animal (“M”, “F”)
hindfoot_length	length of the hindfoot in mm
weight	weight of the animal in grams
genus	genus of animal
species	species of animal
taxon	e.g. Rodent, Reptile, Bird, Rabbit
plot_type	type of plot

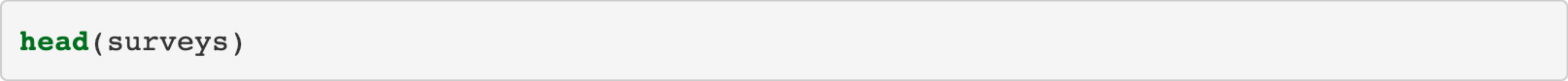
Reading in Data from a file

- We’ve **looked** at the **raw** format of the file (CSV format)
- Let us **Load** the data into R
- Use function read.csv() to load the data into an object of class data.frame

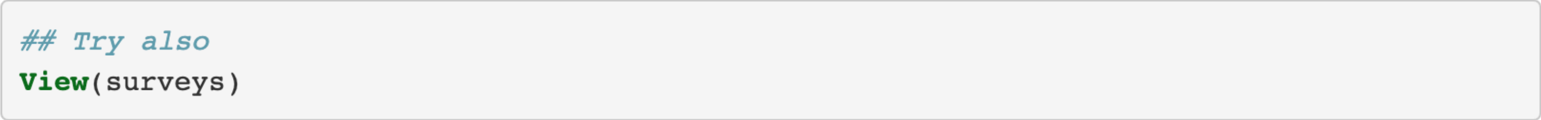


```
surveys <- read.csv("data/portal_data_joined.csv")
```

- Look at the data



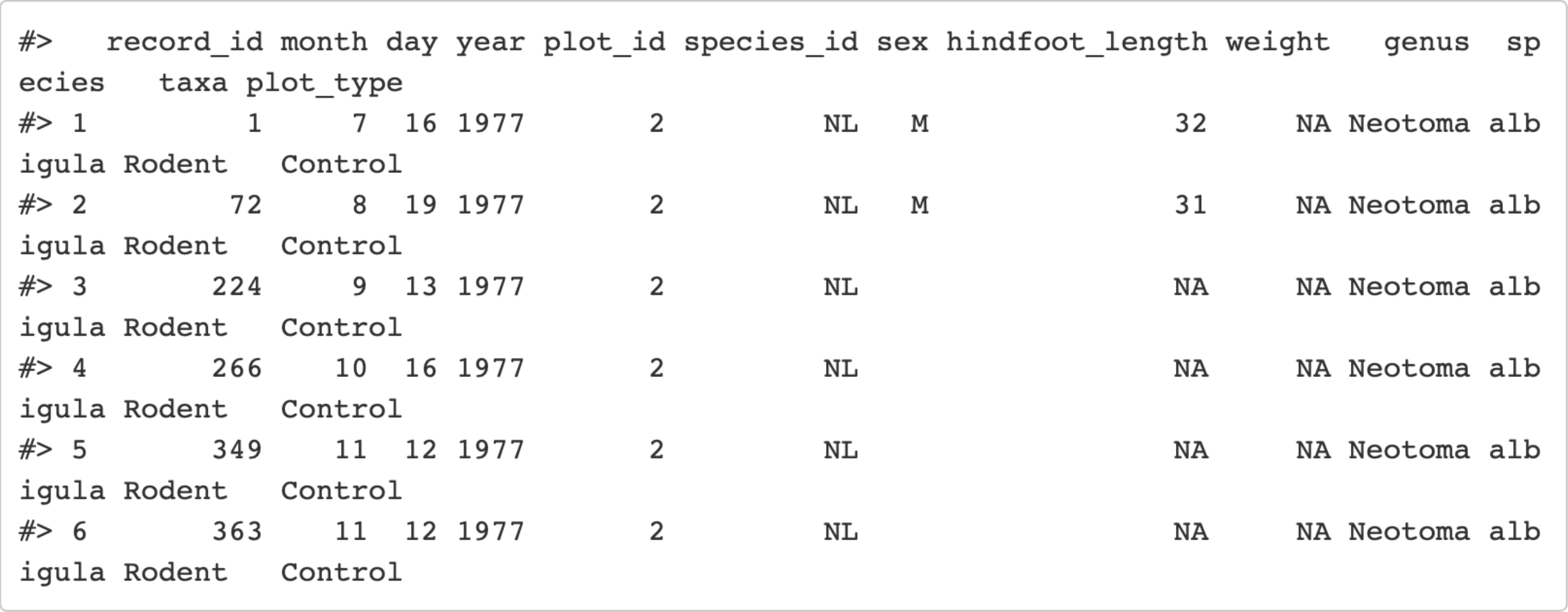
```
head(surveys)
```



```
## Try also
View(surveys)
```

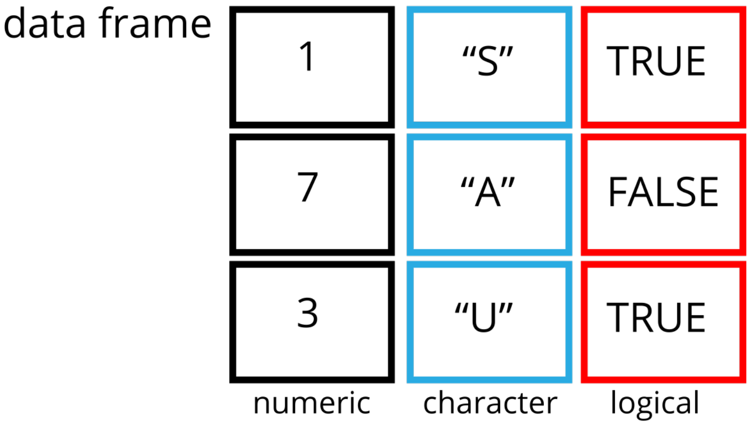
Reading in Data from a file

surveys



```
#>   record_id month day year plot_id species_id sex hindfoot_length weight   genus   sp
ecies    taxa plot_type
#> 1         1     7  16 1977       2         NL   M              32     NA Neotoma alb
igula Rodent   Control
#> 2        72     8  19 1977       2         NL   M              31     NA Neotoma alb
igula Rodent   Control
#> 3       224     9  13 1977       2         NL                  NA     NA Neotoma alb
igula Rodent   Control
#> 4       266    10  16 1977       2         NL                  NA     NA Neotoma alb
igula Rodent   Control
#> 5       349    11  12 1977       2         NL                  NA     NA Neotoma alb
igula Rodent   Control
#> 6       363    11  12 1977       2         NL                  NA     NA Neotoma alb
igula Rodent   Control
```

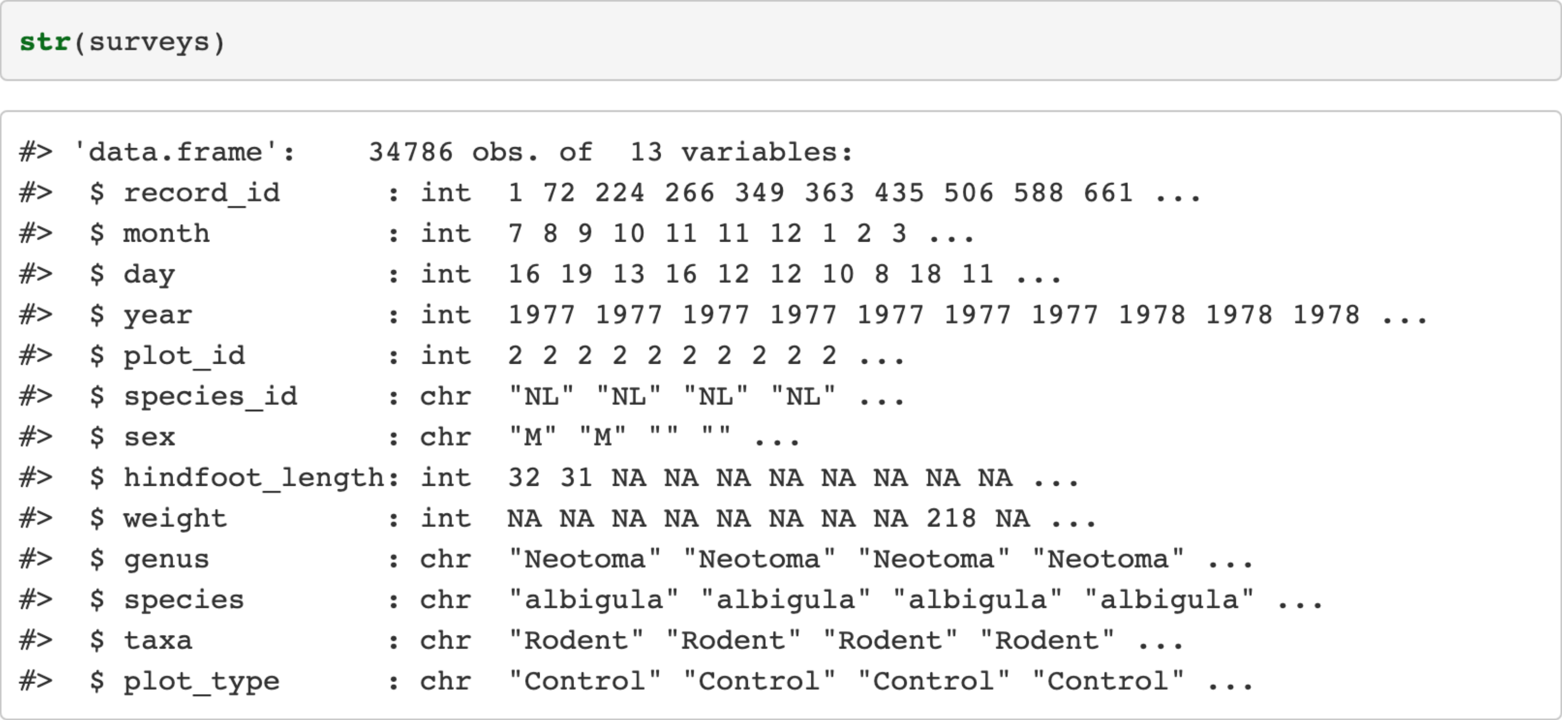
Data frame



data frame

1	“S”	TRUE
7	“A”	FALSE
3	“U”	TRUE
numeric	character	logical

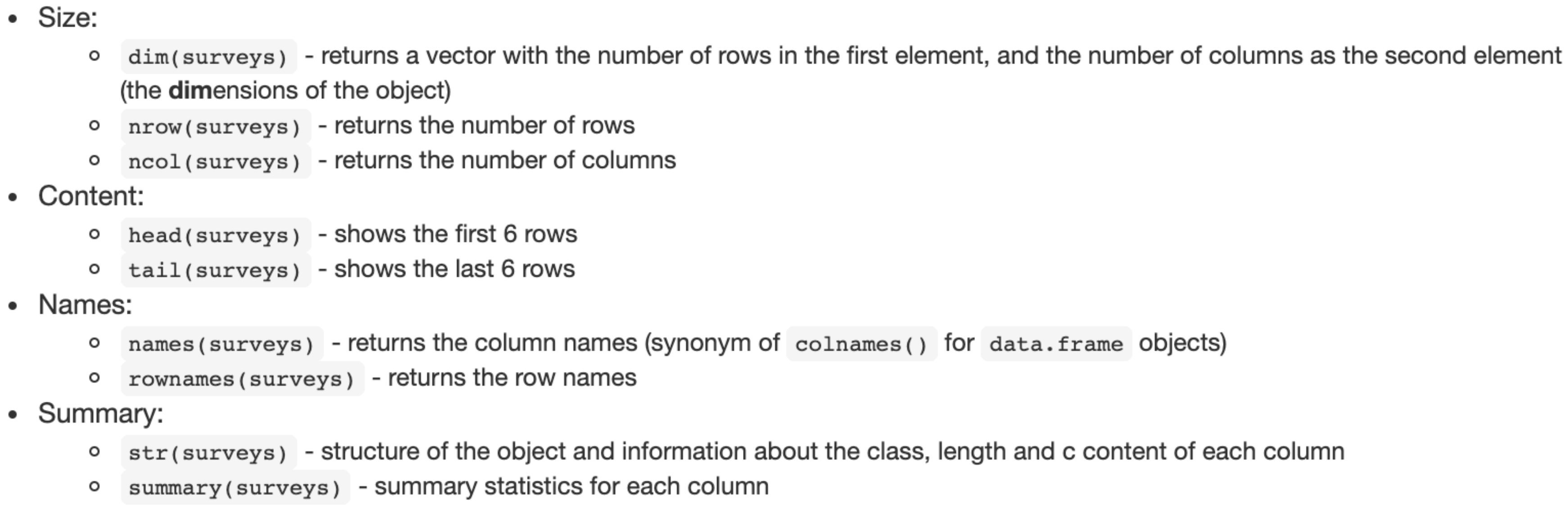
Data frame



```r
str(surveys)
```

```
#> 'data.frame':    34786 obs. of  13 variables:
#>  $ record_id      : int  1 72 224 266 349 363 435 506 588 661 ...
#>  $ month          : int  7 8 9 10 11 11 12 1 2 3 ...
#>  $ day            : int  16 19 13 16 12 12 10 8 18 11 ...
#>  $ year           : int  1977 1977 1977 1977 1977 1977 1977 1978 1978 1978 ...
#>  $ plot_id        : int  2 2 2 2 2 2 2 2 2 2 ...
#>  $ species_id     : chr  "NL" "NL" "NL" "NL" ...
#>  $ sex            : chr  "M" "M" "" "" ...
#>  $ hindfoot_length: int  32 31 NA NA NA NA NA NA NA NA ...
#>  $ weight         : int  NA NA NA NA NA NA NA NA 218 NA ...
#>  $ genus          : chr  "Neotoma" "Neotoma" "Neotoma" "Neotoma" ...
#>  $ species        : chr  "albigula" "albigula" "albigula" "albigula" ...
#>  $ taxa           : chr  "Rodent" "Rodent" "Rodent" "Rodent" ...
#>  $ plot_type      : chr  "Control" "Control" "Control" "Control" ...
```

Inspecting a data.frame object

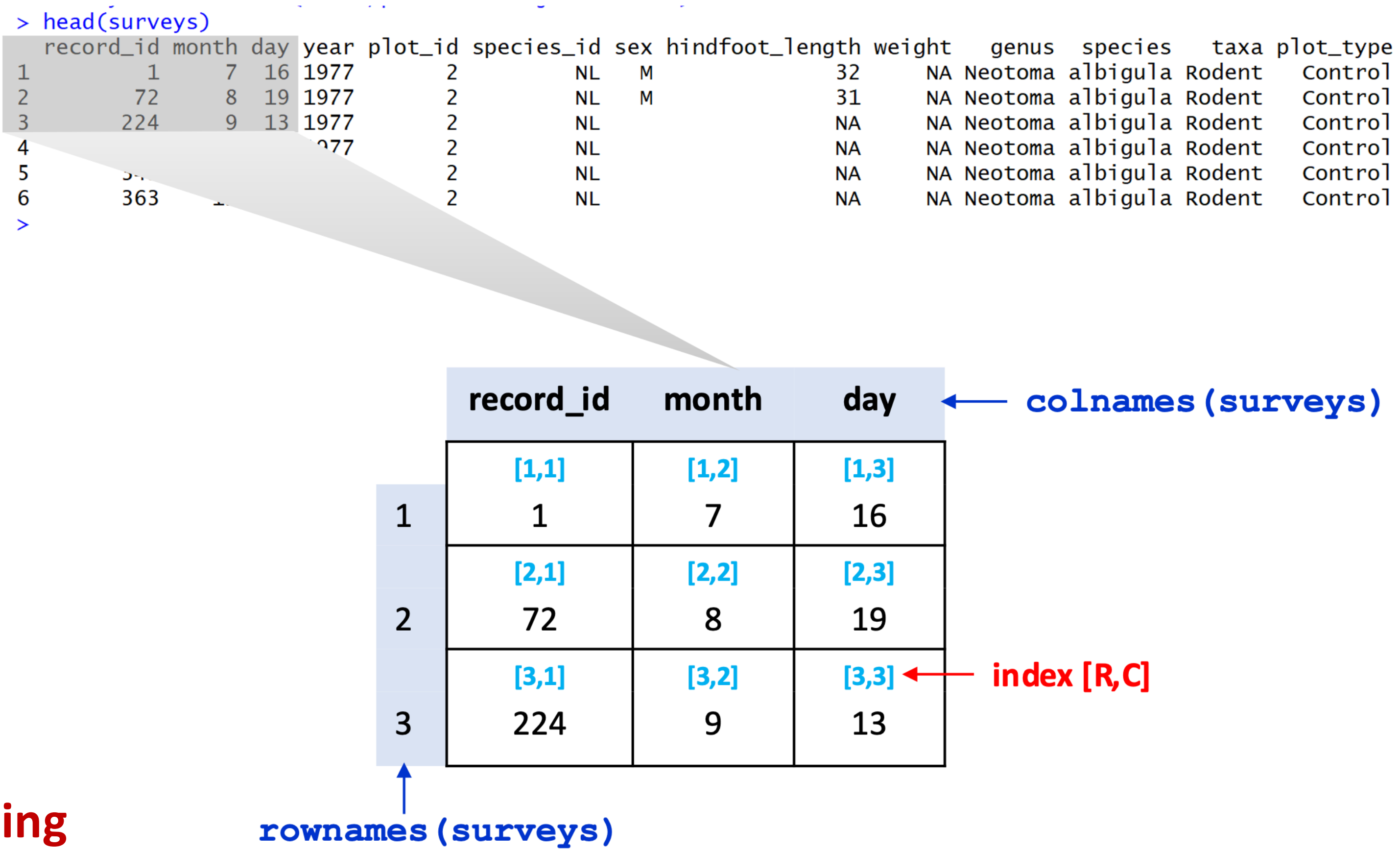


- Size:
  - `dim(surveys)` - returns a vector with the number of rows in the first element, and the number of columns as the second element (the **dim**ensions of the object)
  - `nrow(surveys)` - returns the number of rows
  - `ncol(surveys)` - returns the number of columns
- Content:
  - `head(surveys)` - shows the first 6 rows
  - `tail(surveys)` - shows the last 6 rows
- Names:
  - `names(surveys)` - returns the column names (synonym of `colnames()` for `data.frame` objects)
  - `rownames(surveys)` - returns the row names
- Summary:
  - `str(surveys)` - structure of the object and information about the class, length and c content of each column
  - `summary(surveys)` - summary statistics for each column

Challenge!

- **Based on the output of str(surveys), can you answer the following questions?**
- **What is the class of the object surveys?**
- **How many rows and how many columns are in this object?**
- **How many taxa have been recorded during these surveys?**

Indexing and sub-setting data frames

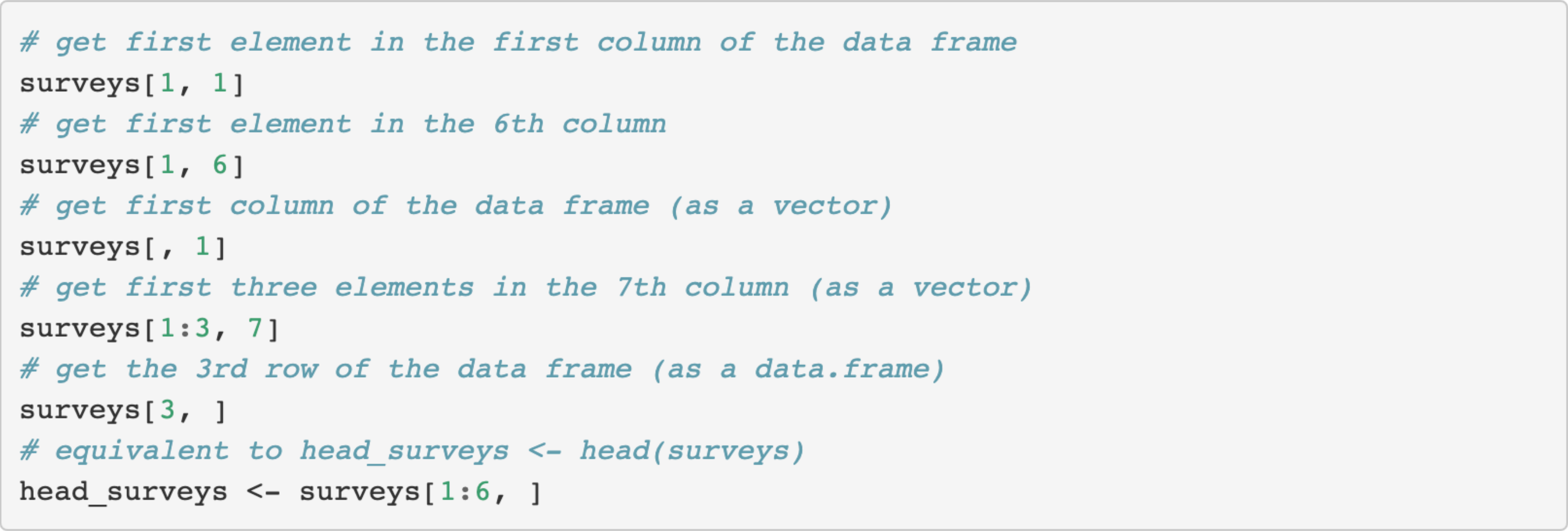


```
> head(surveys)
  record_id month day year plot_id species_id sex hindfoot_length weight   genus  species   taxa plot_type
1         1     7  16 1977       2         NL   M              32     NA Neotoma albigula Rodent   Control
2        72     8  19 1977       2         NL   M              31     NA Neotoma albigula Rodent   Control
3       224     9  13 1977       2         NL                  NA     NA Neotoma albigula Rodent   Control
4                    '977       2         NL                  NA     NA Neotoma albigula Rodent   Control
5       3.                      2         NL                  NA     NA Neotoma albigula Rodent   Control
6       363     1               2         NL                  NA     NA Neotoma albigula Rodent   Control
>
```

	record_id	month	day
1	[1,1] 1	[1,2] 7	[1,3] 16
2	[2,1] 72	[2,2] 8	[2,3] 19
3	[3,1] 224	[3,2] 9	[3,3] 13

Numeric Indexing

Numeric Indexing



```r
# get first element in the first column of the data frame
surveys[1, 1]
# get first element in the 6th column
surveys[1, 6]
# get first column of the data frame (as a vector)
surveys[, 1]
# get first three elements in the 7th column (as a vector)
surveys[1:3, 7]
# get the 3rd row of the data frame (as a data.frame)
surveys[3, ]
# equivalent to head_surveys <- head(surveys)
head_surveys <- surveys[1:6, ]
```

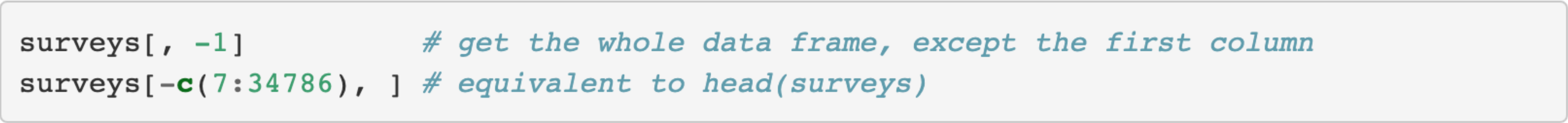
Numeric Indexing (cont.)

- `:` is an R operator to create a sequence of numeric vectors

(Integers in increasing or decreasing order)

- Try 1:10 and 10:1
- It is equivalent to the function seq(from, to)

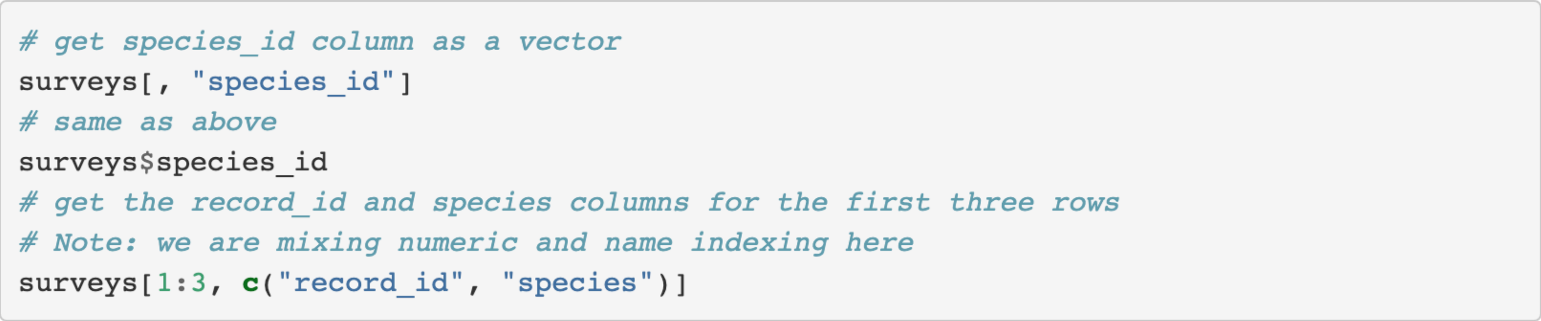
- You can also exclude certain indices of a data frame using the “-” sign:



```
surveys[, -1]             # get the whole data frame, except the first column
surveys[-c(7:34786), ] # equivalent to head(surveys)
```

Name Indexing

- Data frames can **also** be subset by calling row names and column names directly!
- This is known as **name indexing**
- Examples:

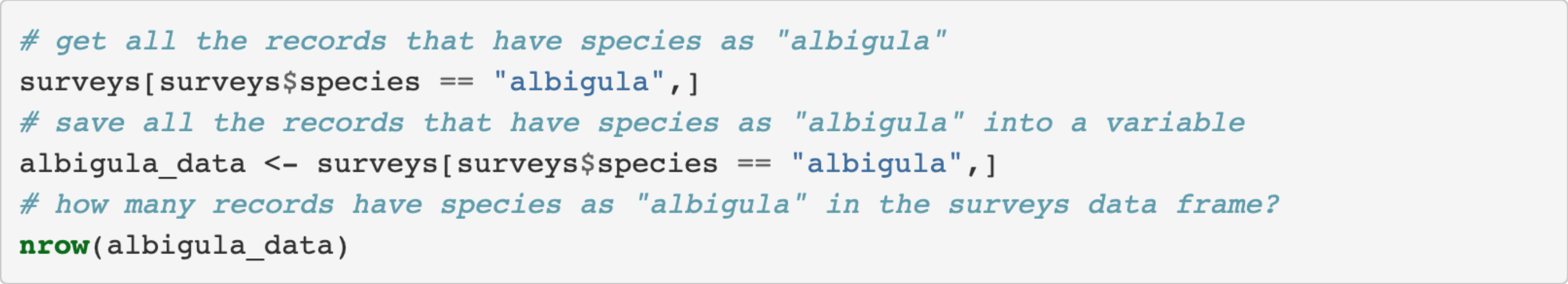


```
# get species_id column as a vector
surveys[, "species_id"]
# same as above
surveys$species_id
# get the record_id and species columns for the first three rows
# Note: we are mixing numeric and name indexing here
surveys[1:3, c("record_id", "species")]
```

- In RStudio, you can use the **autocompletion** feature to get the full/correct names!

Logical Indexing

- Another way to retrieve data from a data frame is by logical indexing
- Let’s perform a logical operation on **surveys**



```
# get all the records that have species as "albigula"
surveys[surveys$species == "albigula",]
# save all the records that have species as "albigula" into a variable
albigula_data <- surveys[surveys$species == "albigula",]
# how many records have species as "albigula" in the surveys data frame?
nrow(albigula_data)
```

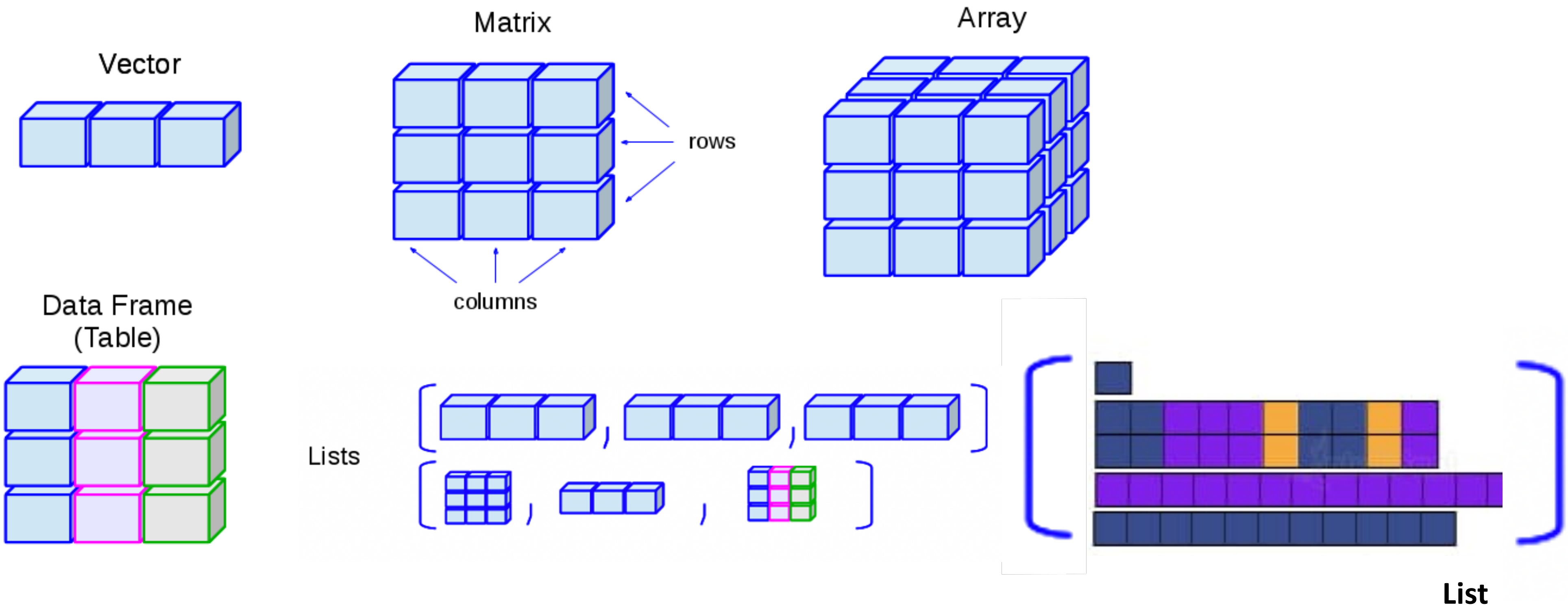
- Who’s this fellow By the way!

Neotoma albigula

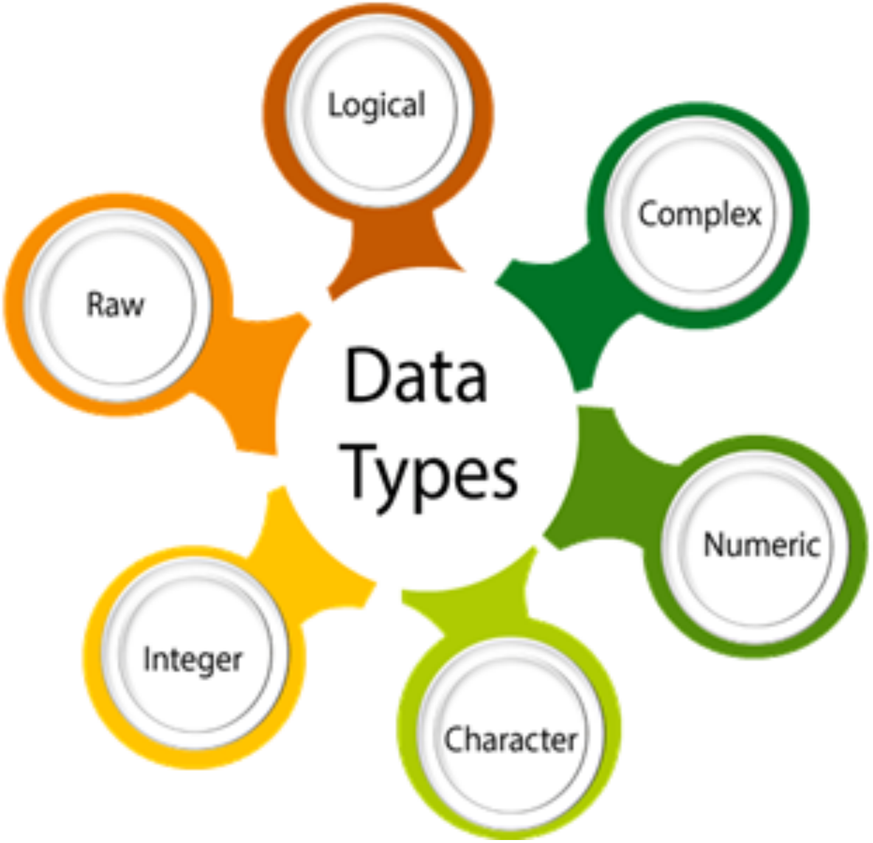
Challenge!

- Create a data.frame (surveys_200) containing only the data in row 200 of the surveys dataset.
- Notice how nrow() gave you the number of rows in a data.frame?
  - Use that number to pull out just that last row in the data frame.
  - Compare that with what you see as the last row using tail() to make sure it's meeting expectations.
  - Pull out that last row using nrow() instead of the row number.
  - Create a new data frame (surveys_last) from that last row.
- Use nrow() to extract the row that is in the middle of the data frame. Store the content of this row in an object named surveys_middle.
- Combine nrow() with the - notation above to reproduce the behavior of head(surveys), keeping just the first through 6th rows of the surveys dataset.

Data Structures (Recap)



Data Type (Recap)

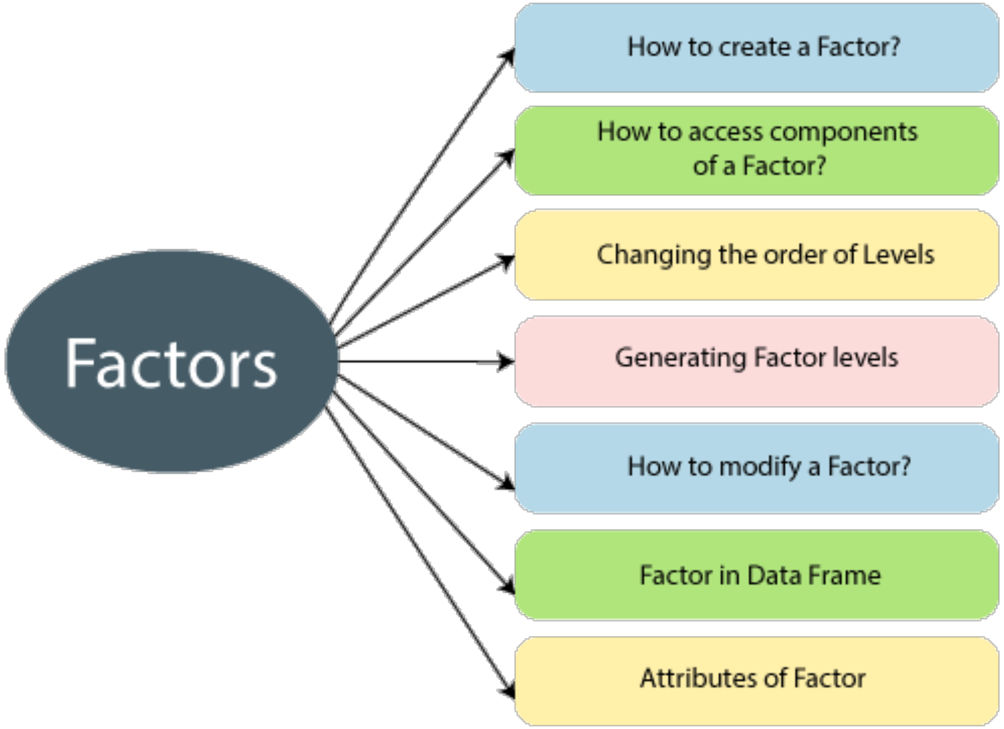


Factors

- **Categorical** Variables in Statistics
- Can take Limited set of Values (levels)
  - Example:

    "Gender" = {Male, Female}

    "Meal" = {Breakfast, Lunch, Dinner}
- No Intrinsic Ordering (alphabetical)
- Order can be changed
  - Why/When would you wish this!?



Factors

How to create a Factor?

How to access components of a Factor?

Changing the order of Levels

Generating Factor levels

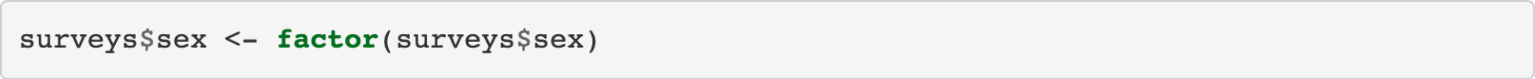
How to modify a Factor?

Factor in Data Frame

Attributes of Factor

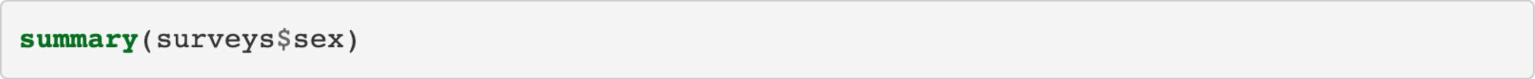
Factors

- Let’s convert survey columns that contain categorical data to type factor with the factor() function:



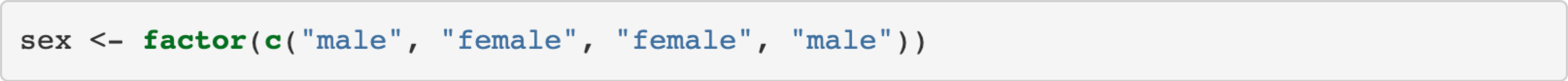
```
surveys$sex <- factor(surveys$sex)
```

- We can see that the conversion has worked by using the summary() function



```
summary(surveys$sex)
```

- By default, R always sorts levels in alphabetical order

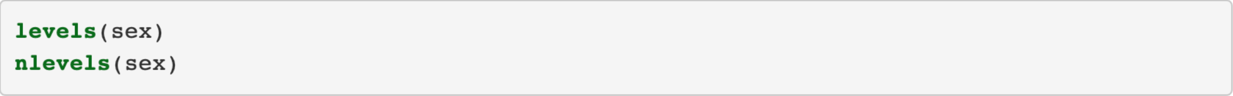


```
sex <- factor(c("male", "female", "female", "male"))
```

- R will assign 1 to the level "female" and 2 to the level "male" (because **f** comes before **m**, even though the first element in this vector is "male")

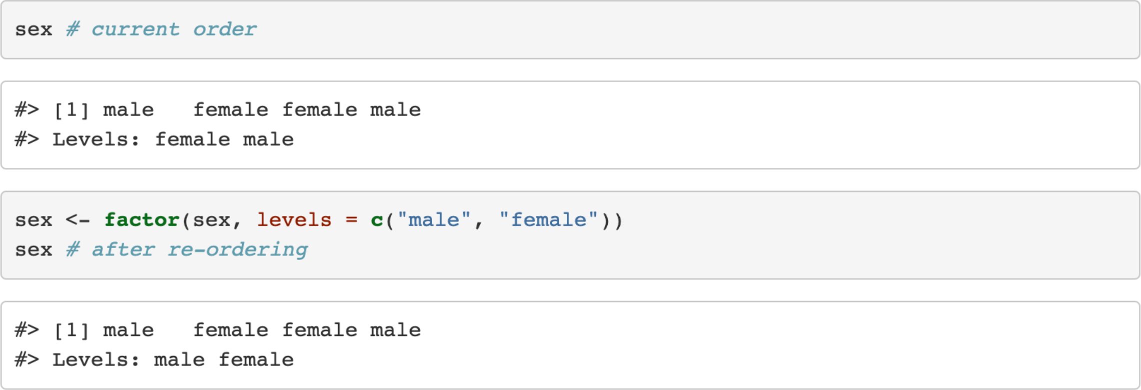
Factors

- Use function levels() to check this! [You can find the number of levels using nlevels()]



```
levels(sex)
nlevels(sex)
```

- **Order** of factors does not usually matter. But It might! Eg. It’s meaningful (e.g., “low”, “medium”, “high”), or improves your visualization, or it is required by the analysis
- Here, one way to reorder our levels in the sex vector would be:



```
sex # current order
```

```
#> [1] male   female female male
#> Levels: female male
```

```
sex <- factor(sex, levels = c("male", "female"))
sex # after re-ordering
```

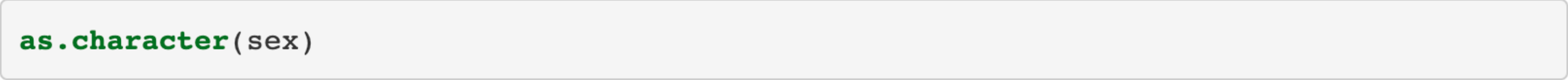
```
#> [1] male   female female male
#> Levels: male female
```

Challenge!

- **Change the columns taxa and genus in the surveys data frame into a factor.**
- **Using the functions you learned before, can you find out…**
  - **How many rabbits were observed?**
  - **How many different genera are in the genus column?**

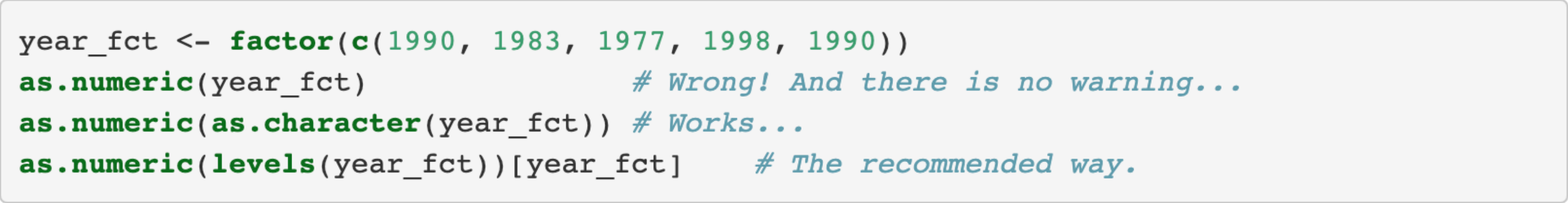
Converting Factors

- One way to convert a **factor** to a **character** vector, you use as.character(x)



```
as.character(sex)
```

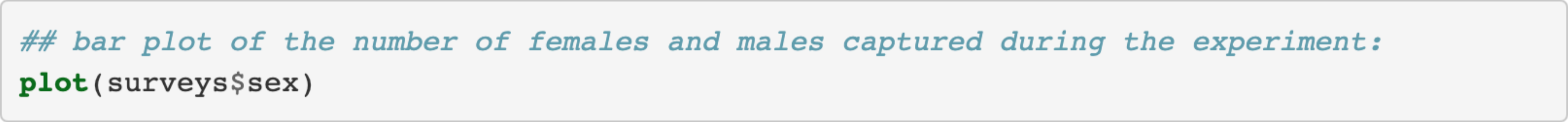
- The as.numeric() function returns the **index values** of the factor, not its levels, so it will result in an entirely new (and unwanted in this case) set of numbers.
- One method to avoid this is to convert factors to characters, and then to numbers.
- **Another method** is to use the levels() function



```
year_fct <- factor(c(1990, 1983, 1977, 1998, 1990))
as.numeric(year_fct)                 # Wrong! And there is no warning...
as.numeric(as.character(year_fct)) # Works...
as.numeric(levels(year_fct))[year_fct]    # The recommended way.
```

Renaming Factors

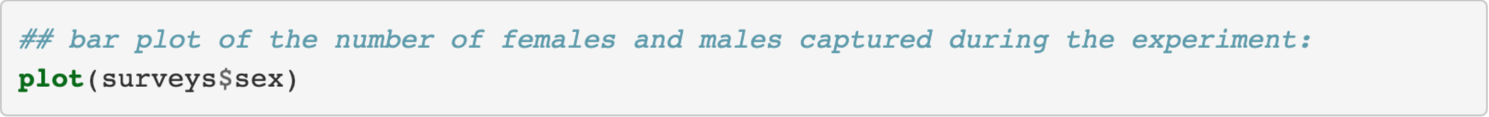
- Quick glance at the number of observations represented by each factor level. Use plot() function:



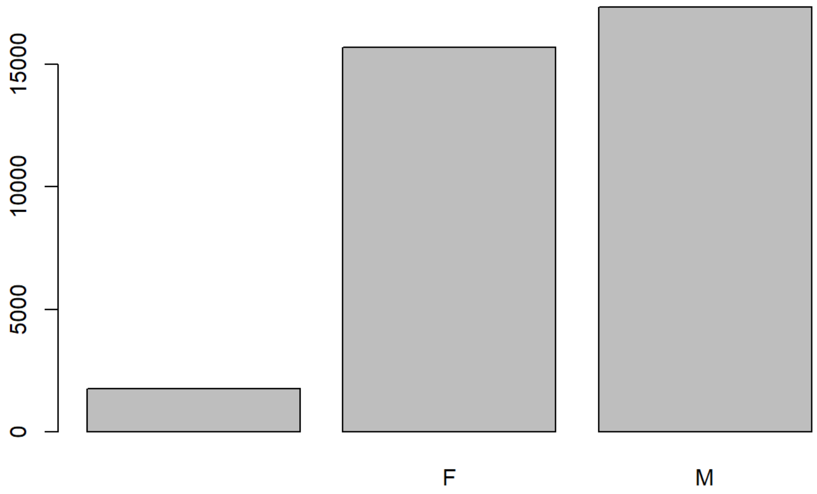
```
## bar plot of the number of females and males captured during the experiment:
plot(surveys$sex)
```

Renaming Factors

- Quick glance at the number of observations represented by each factor level. Use plot() function:



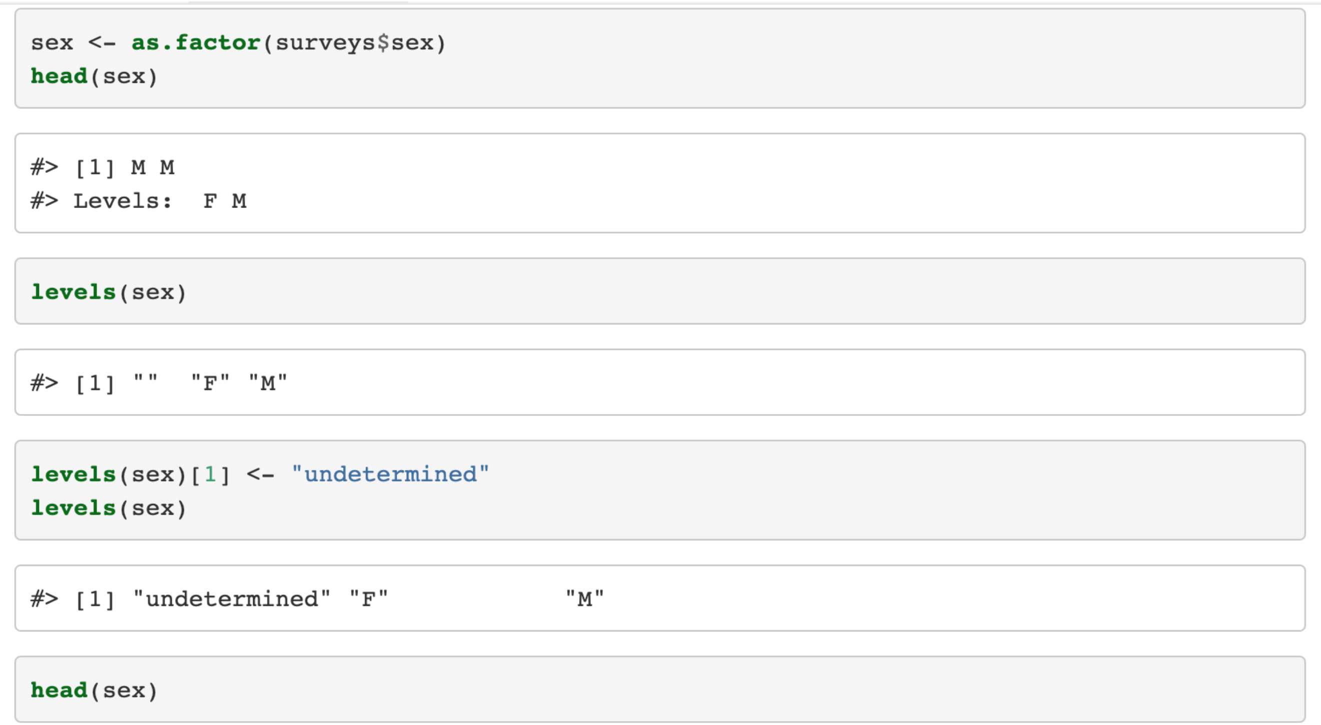
```
## bar plot of the number of females and males captured during the experiment:
plot(surveys$sex)
```



- **Note**: For 1700 individuals - sex information hasn’t been recorded. **How to show them in the plot**?!

Renaming Factors

- To show them in the plot, we can turn the **missing values** into a **factor** level.
- **New label** to the new factor level. [Copy **sex** column to avoid modifying the working copy of the **data frame!]**



```
sex <- as.factor(surveys$sex)
head(sex)
```

```
#> [1] M M
#> Levels:   F M
```

```
levels(sex)
```

```
#> [1] ""  "F" "M"
```

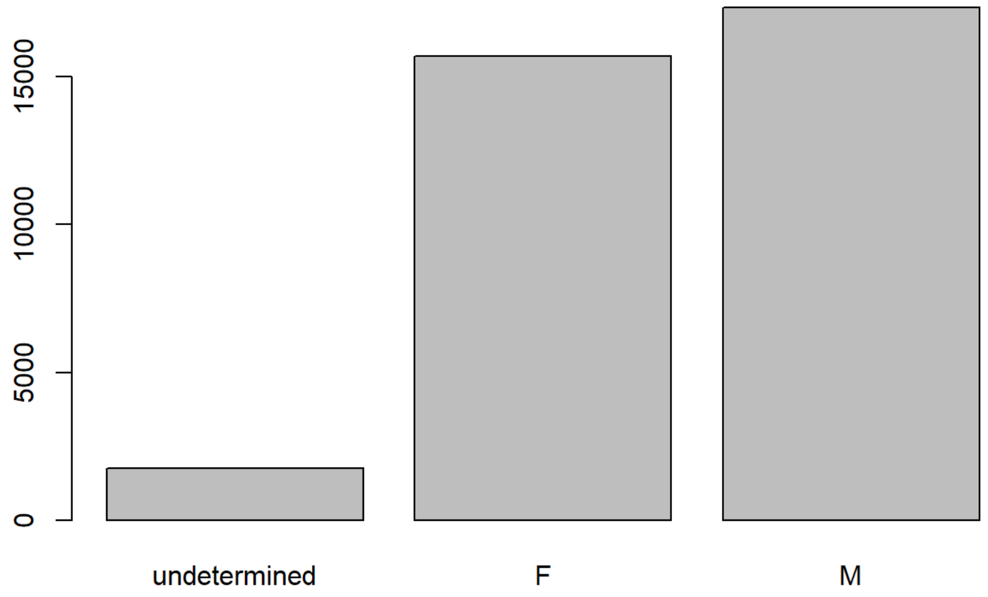
```
levels(sex)[1] <- "undetermined"
levels(sex)
```

```
#> [1] "undetermined" "F"            "M"
```

```
head(sex)
```

Renaming Factors

- Now let’s plot the data again!
- plot (sex)



Challenge!

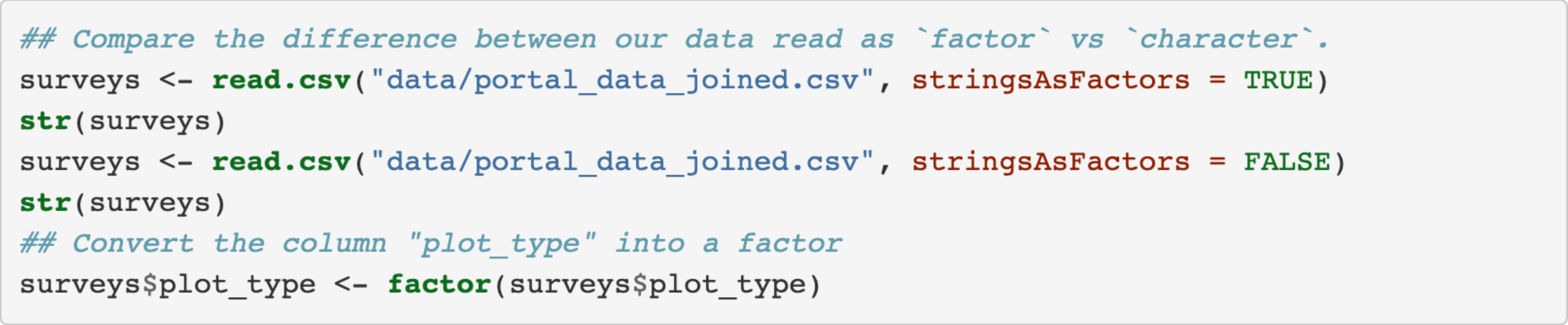
- **Rename “F” and “M” to “female” and “male” respectively.**

- **Now that we have renamed the factor level to “undetermined”, can you recreate the barplot such that “undetermined” is last (after “male”)?**

Using STRINGS as Factors

- Depending on what you want to do with the data, when you have a column with categorical data you may want to keep these columns as character or else you may want to change them to factor.

- To do so, read.csv() and read.table() have an argument called stringsAsFactors which can be set to TRUE.

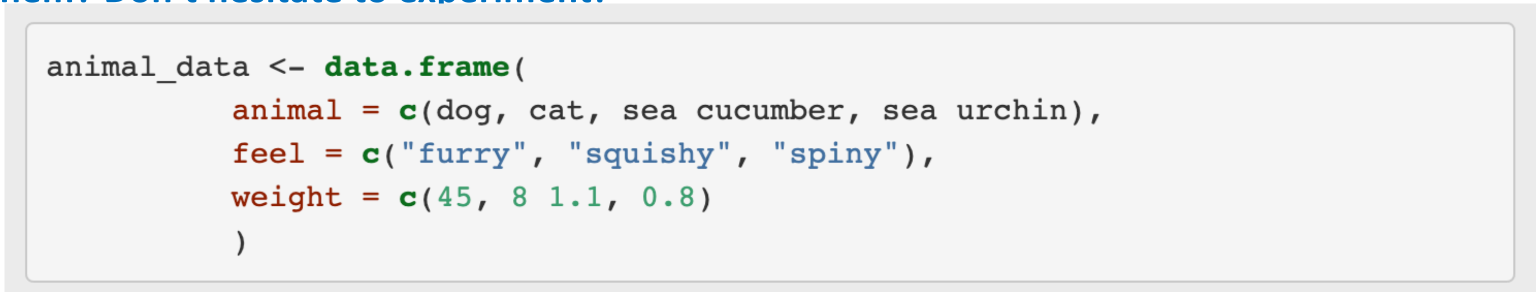


```
## Compare the difference between our data read as `factor` vs `character`.
surveys <- read.csv("data/portal_data_joined.csv", stringsAsFactors = TRUE)
str(surveys)
surveys <- read.csv("data/portal_data_joined.csv", stringsAsFactors = FALSE)
str(surveys)
## Convert the column "plot_type" into a factor
surveys$plot_type <- factor(surveys$plot_type)
```

- This is sometimes a blessing, sometimes an annoyance!
- **Be AWARE that it exists**! (Not in the latest version of R though!)

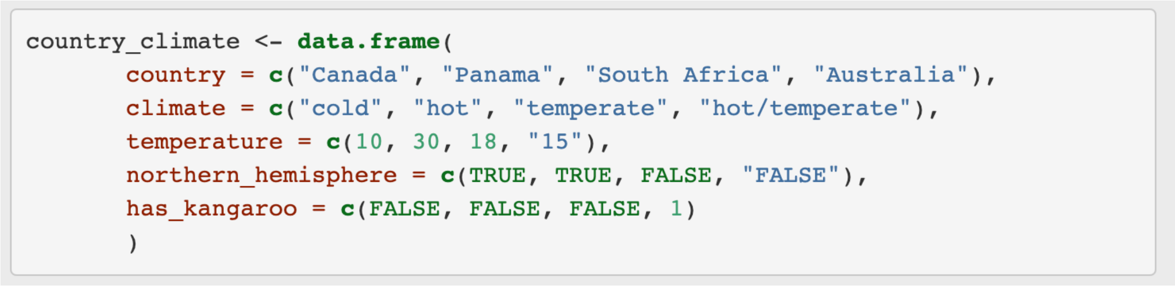
Challenge!

1. We have seen how data frames are created when using read.csv(), but they can also be created by hand with the data.frame() function. There are a few mistakes in this hand-crafted data.frame. Can you spot and fix them? Don't hesitate to experiment!



```
animal_data <- data.frame(
          animal = c(dog, cat, sea cucumber, sea urchin),
          feel = c("furry", "squishy", "spiny"),
          weight = c(45, 8 1.1, 0.8)
          )
```

2. Can you predict the class for each of the columns in the following example? Check your guesses using str(country_climate):
   - Are they what you expected? Why? Why not?
   - What would have been different if we had added stringsAsFactors = FALSE when creating the data frame?
   - What would you need to change to ensure that each column had the accurate data type?



```
country_climate <- data.frame(
       country = c("Canada", "Panama", "South Africa", "Australia"),
       climate = c("cold", "hot", "temperate", "hot/temperate"),
       temperature = c(10, 30, 18, "15"),
       northern_hemisphere = c(TRUE, TRUE, FALSE, "FALSE"),
       has_kangaroo = c(FALSE, FALSE, FALSE, 1)
       )
```

NEXT

DATA VISUALISATION