Introduction to R & RStudio

Shelly Cooper

February 24, 2018

Why R?

FREE!

Open source

Learning R will help you learn other programming languages

- Example: MATLAB, Python, web programming

Flexible

- Can do anything you want it to do

(Did we mention it's free?)

What is R?

R is a programming language

- Statistics & graphing

RStudio is an environment that makes it easy to actually use R

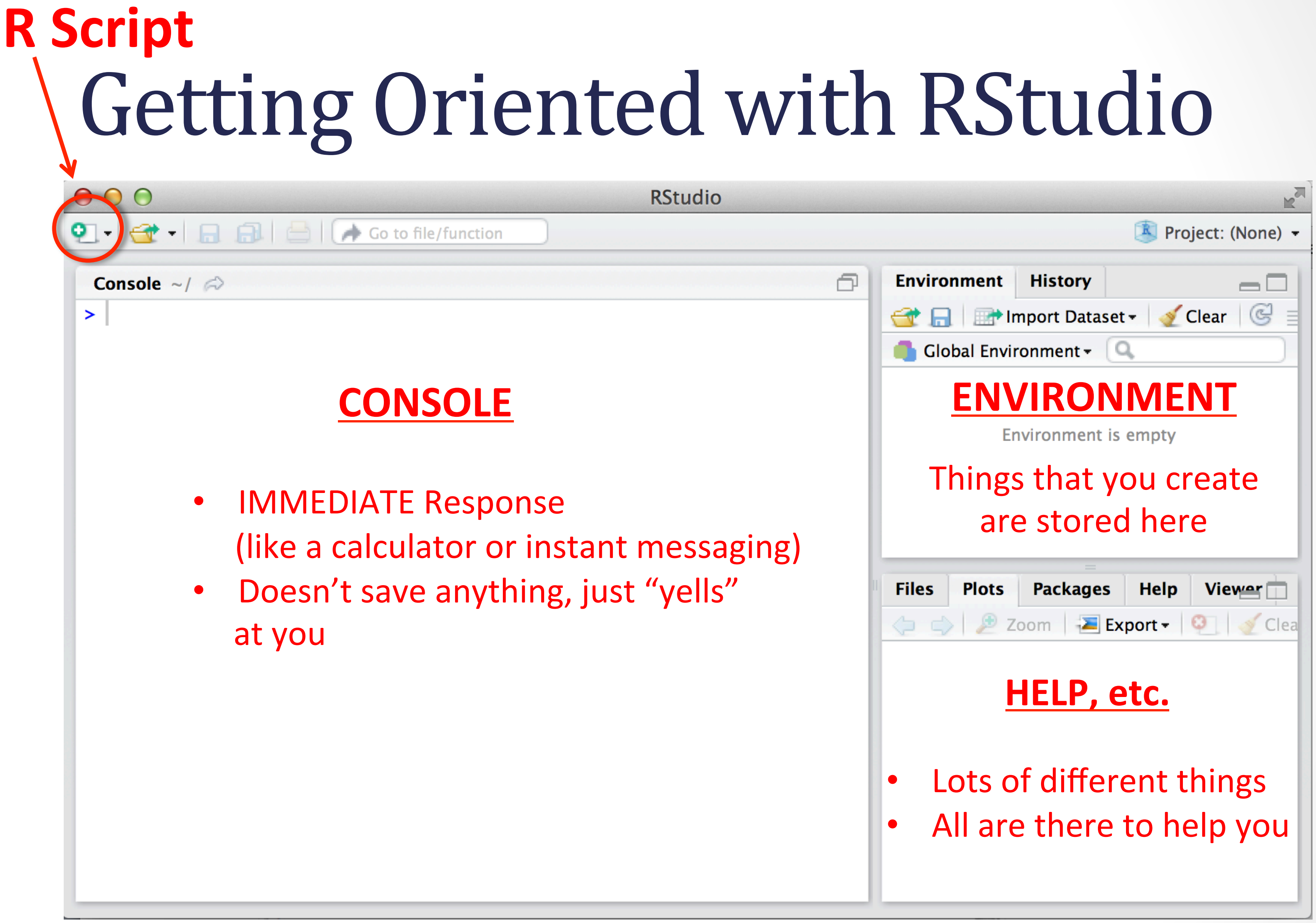
Ask questions

Make mistakes!

When in doubt, Google

SETTING UP R & RSTUDIO

Getting Oriented with RStudio



CONSOLE

- IMMEDIATE Response (like a calculator or instant messaging)
- Doesn't save anything, just "yells" at you

ENVIRONMENT

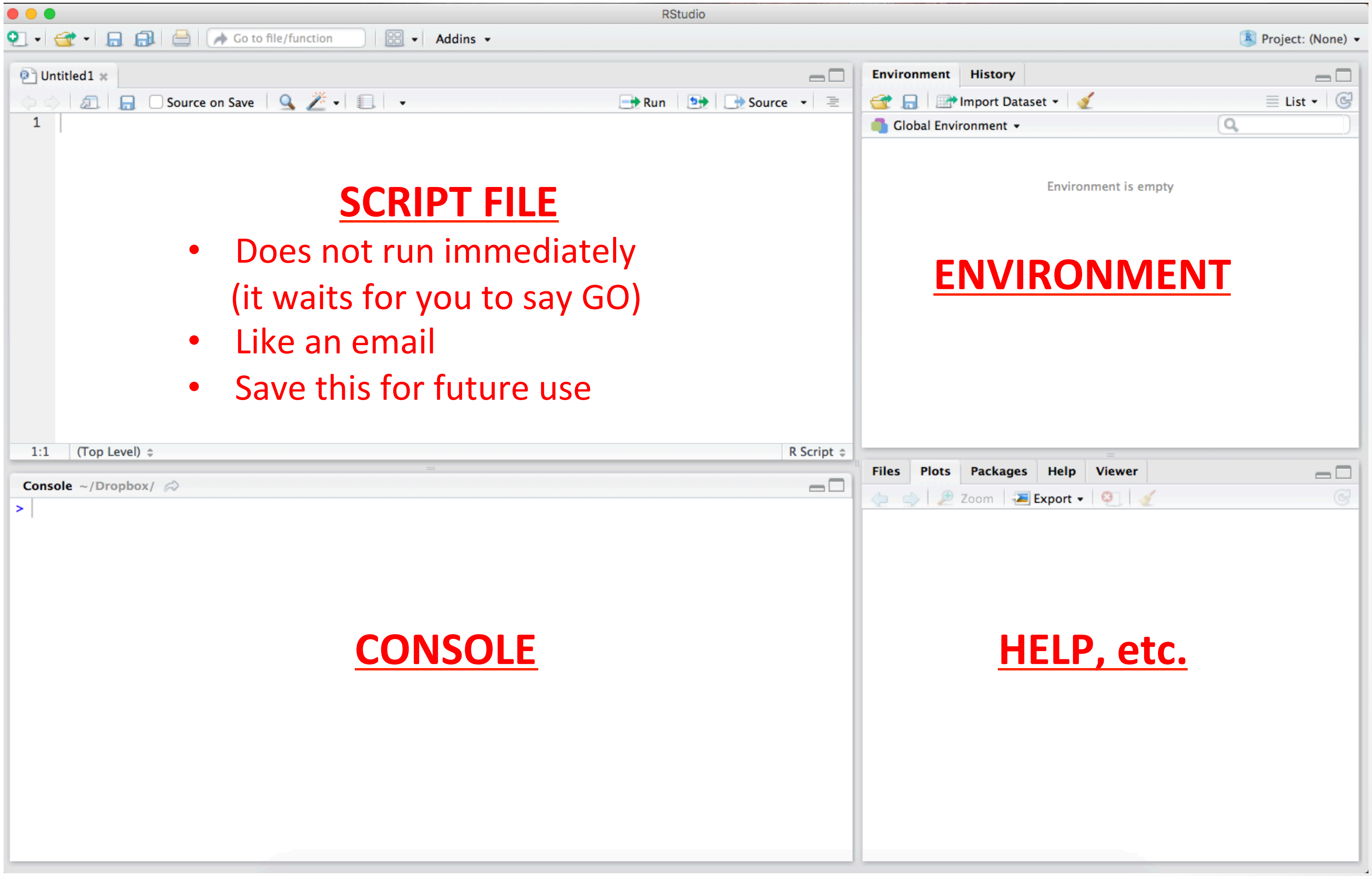
Environment is empty

Things that you create are stored here

HELP, etc.

- Lots of different things
- All are there to help you

Getting Oriented with RStudio



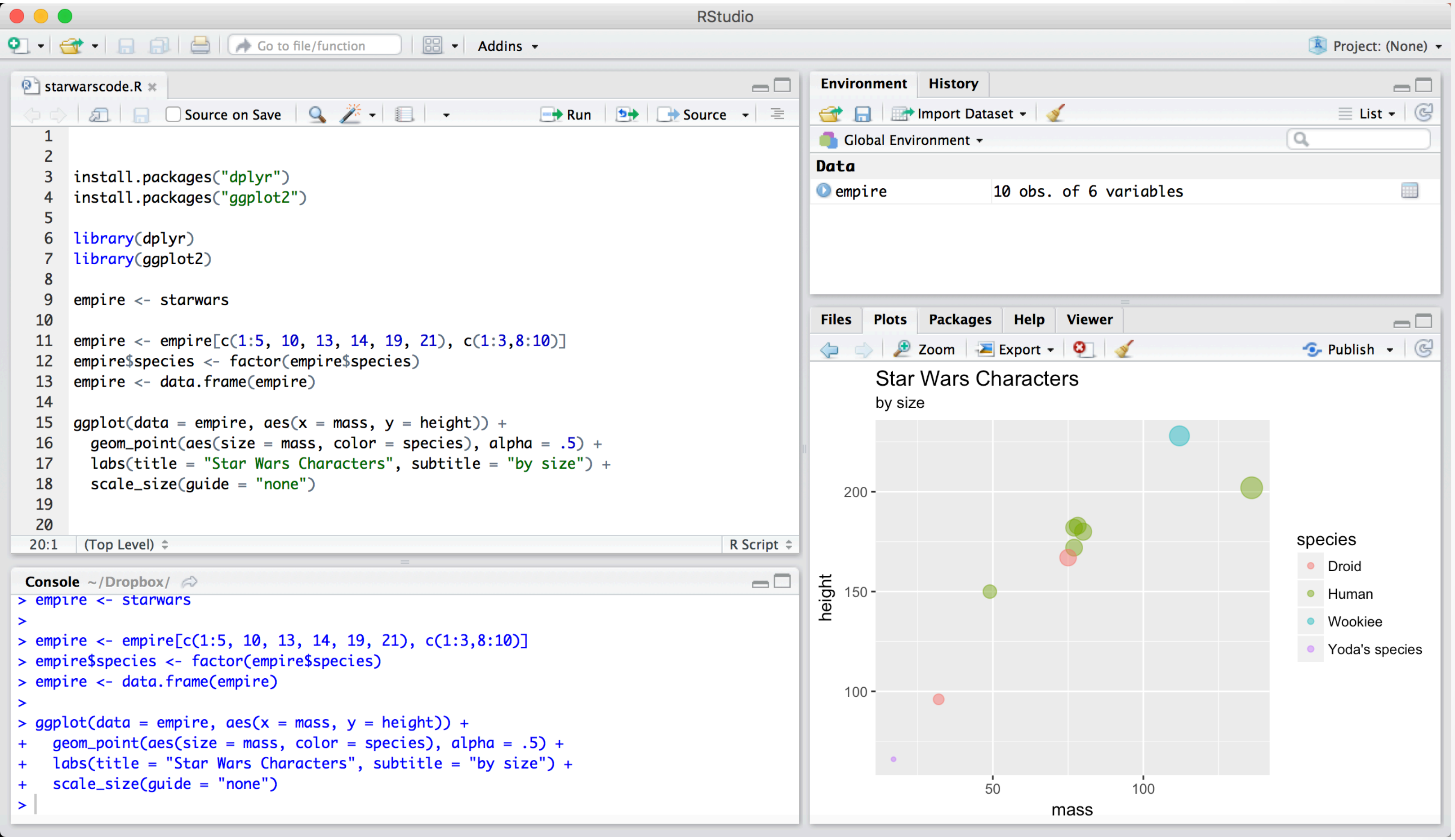
SCRIPT FILE

- Does not run immediately (it waits for you to say GO)
- Like an email
- Save this for future use

ENVIRONMENT

CONSOLE

HELP, etc.



```r


install.packages("dplyr")
install.packages("ggplot2")

library(dplyr)
library(ggplot2)

empire <- starwars

empire <- empire[c(1:5, 10, 13, 14, 19, 21), c(1:3,8:10)]
empire$species <- factor(empire$species)
empire <- data.frame(empire)

ggplot(data = empire, aes(x = mass, y = height)) +
  geom_point(aes(size = mass, color = species), alpha = .5) +
  labs(title = "Star Wars Characters", subtitle = "by size") +
  scale_size(guide = "none")


```

Data	
empire	10 obs. of 6 variables

```
> empire <- starwars
>
> empire <- empire[c(1:5, 10, 13, 14, 19, 21), c(1:3,8:10)]
> empire$species <- factor(empire$species)
> empire <- data.frame(empire)
>
> ggplot(data = empire, aes(x = mass, y = height)) +
+   geom_point(aes(size = mass, color = species), alpha = .5) +
+   labs(title = "Star Wars Characters", subtitle = "by size") +
+   scale_size(guide = "none")
>
```

EXAMPLE 1 IN R

How does R make sense of data?

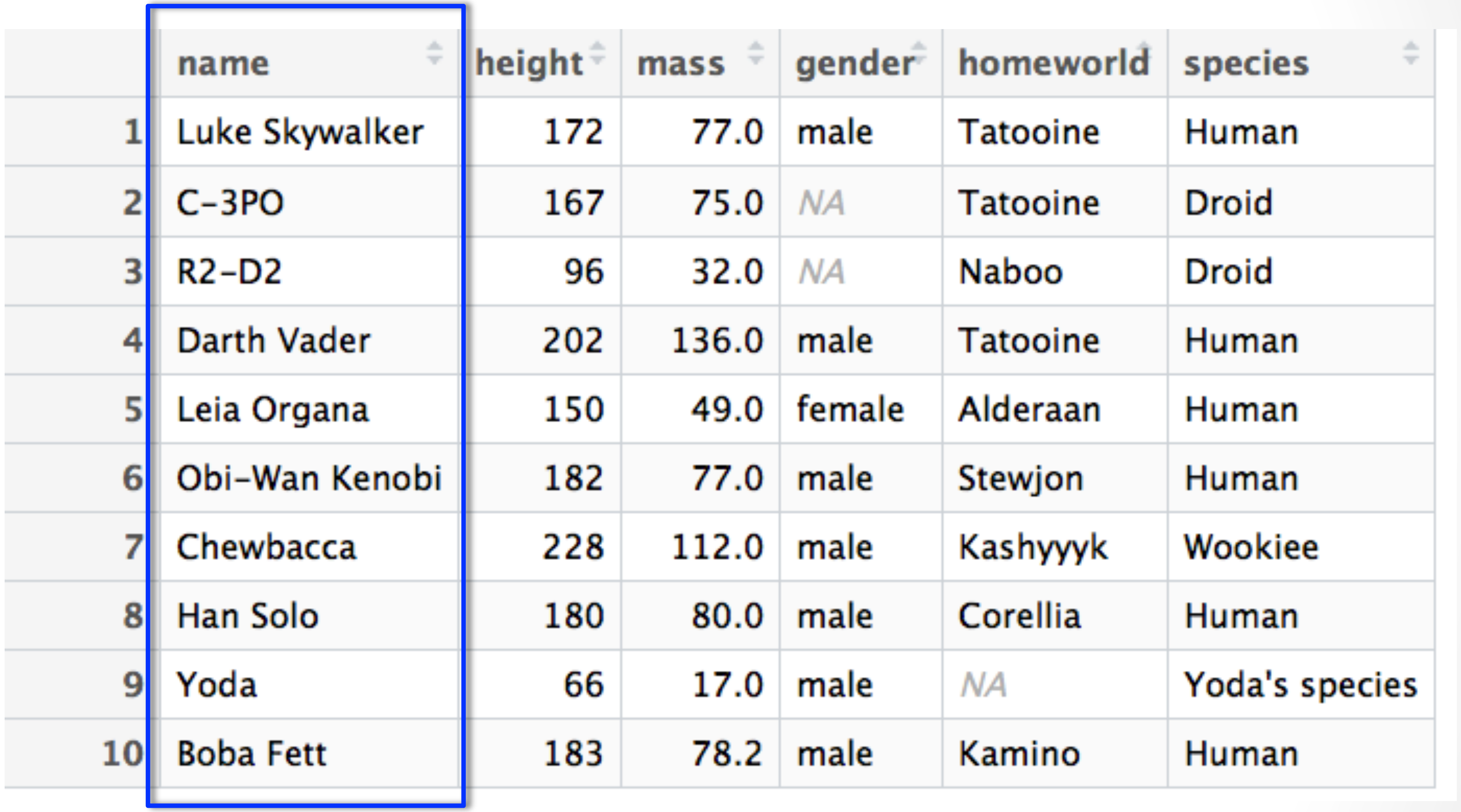
- What "words" does R know and use?
- Need a vocabulary so that you can talk to R (and R can understand you)

DATA FRAMES

Data Frames

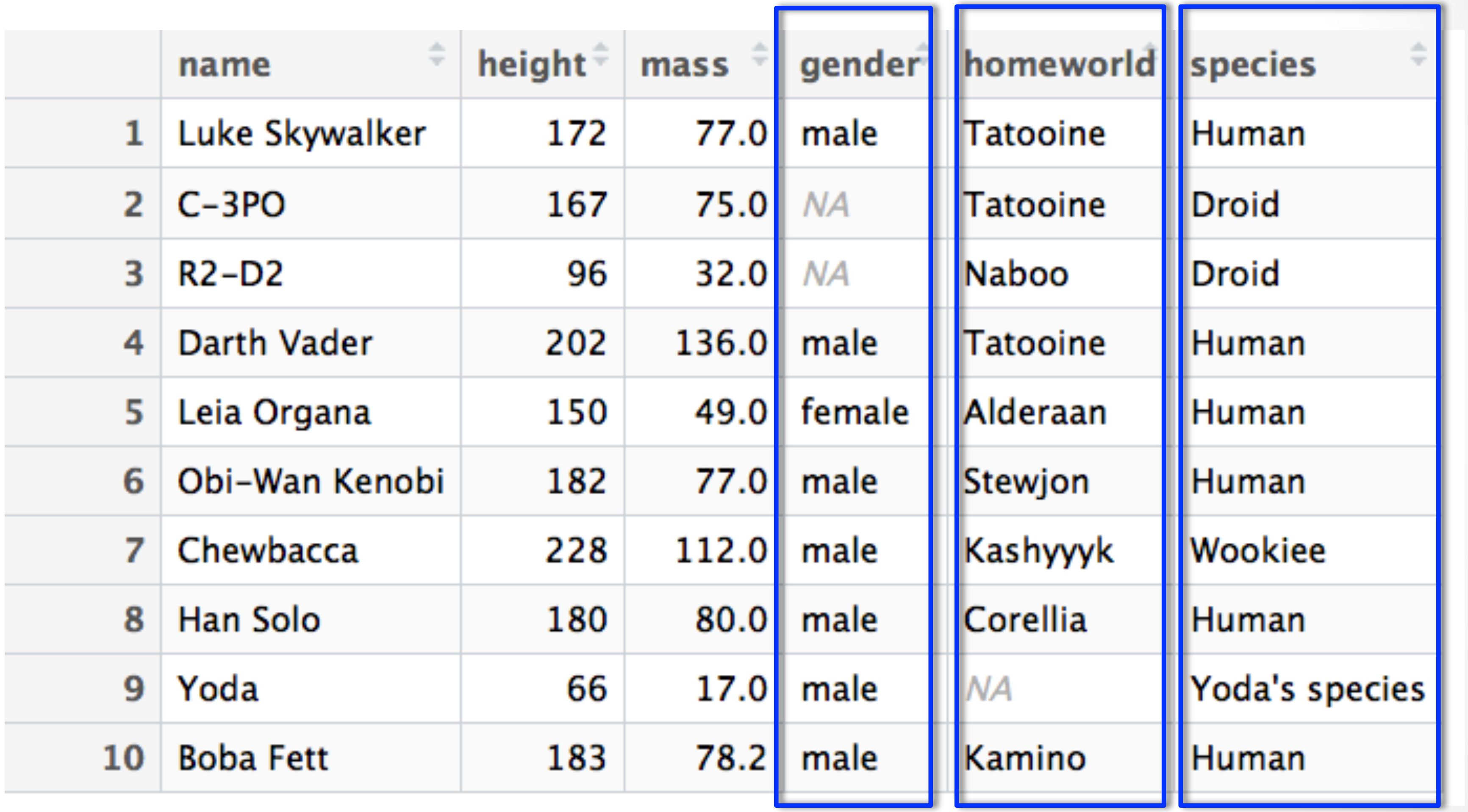
- Used for storing variables of equal length.
- In Excel, you would call this a “spreadsheet”.
- Many researchers might simply call this “data”.

Data Frames



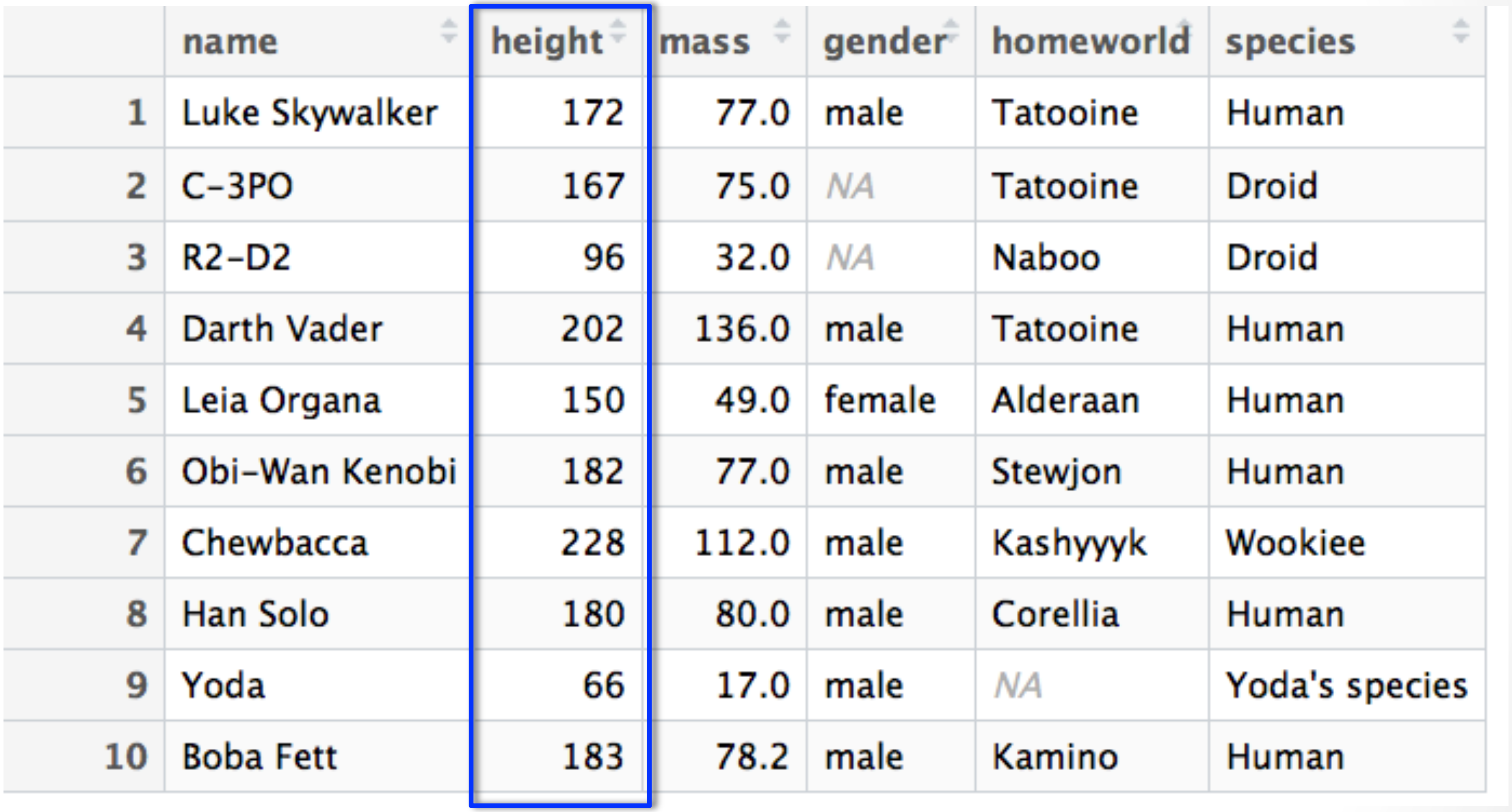
	name	height	mass	gender	homeworld	species
1	Luke Skywalker	172	77.0	male	Tatooine	Human
2	C-3PO	167	75.0	NA	Tatooine	Droid
3	R2-D2	96	32.0	NA	Naboo	Droid
4	Darth Vader	202	136.0	male	Tatooine	Human
5	Leia Organa	150	49.0	female	Alderaan	Human
6	Obi-Wan Kenobi	182	77.0	male	Stewjon	Human
7	Chewbacca	228	112.0	male	Kashyyyk	Wookiee
8	Han Solo	180	80.0	male	Corellia	Human
9	Yoda	66	17.0	male	NA	Yoda's species
10	Boba Fett	183	78.2	male	Kamino	Human

Data Frames



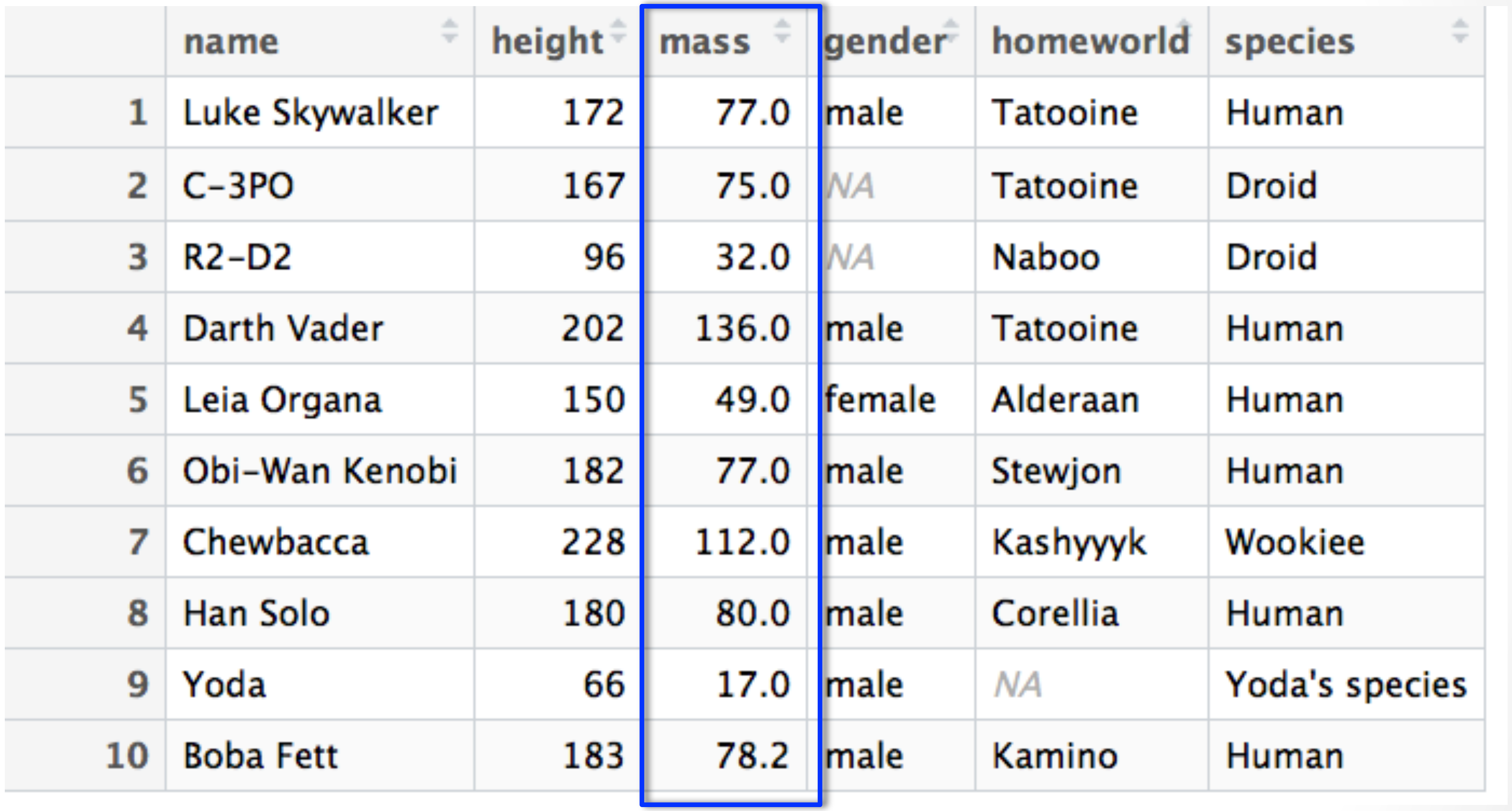
	name	height	mass	gender	homeworld	species
1	Luke Skywalker	172	77.0	male	Tatooine	Human
2	C-3PO	167	75.0	NA	Tatooine	Droid
3	R2-D2	96	32.0	NA	Naboo	Droid
4	Darth Vader	202	136.0	male	Tatooine	Human
5	Leia Organa	150	49.0	female	Alderaan	Human
6	Obi-Wan Kenobi	182	77.0	male	Stewjon	Human
7	Chewbacca	228	112.0	male	Kashyyyk	Wookiee
8	Han Solo	180	80.0	male	Corellia	Human
9	Yoda	66	17.0	male	NA	Yoda's species
10	Boba Fett	183	78.2	male	Kamino	Human

Data Frames



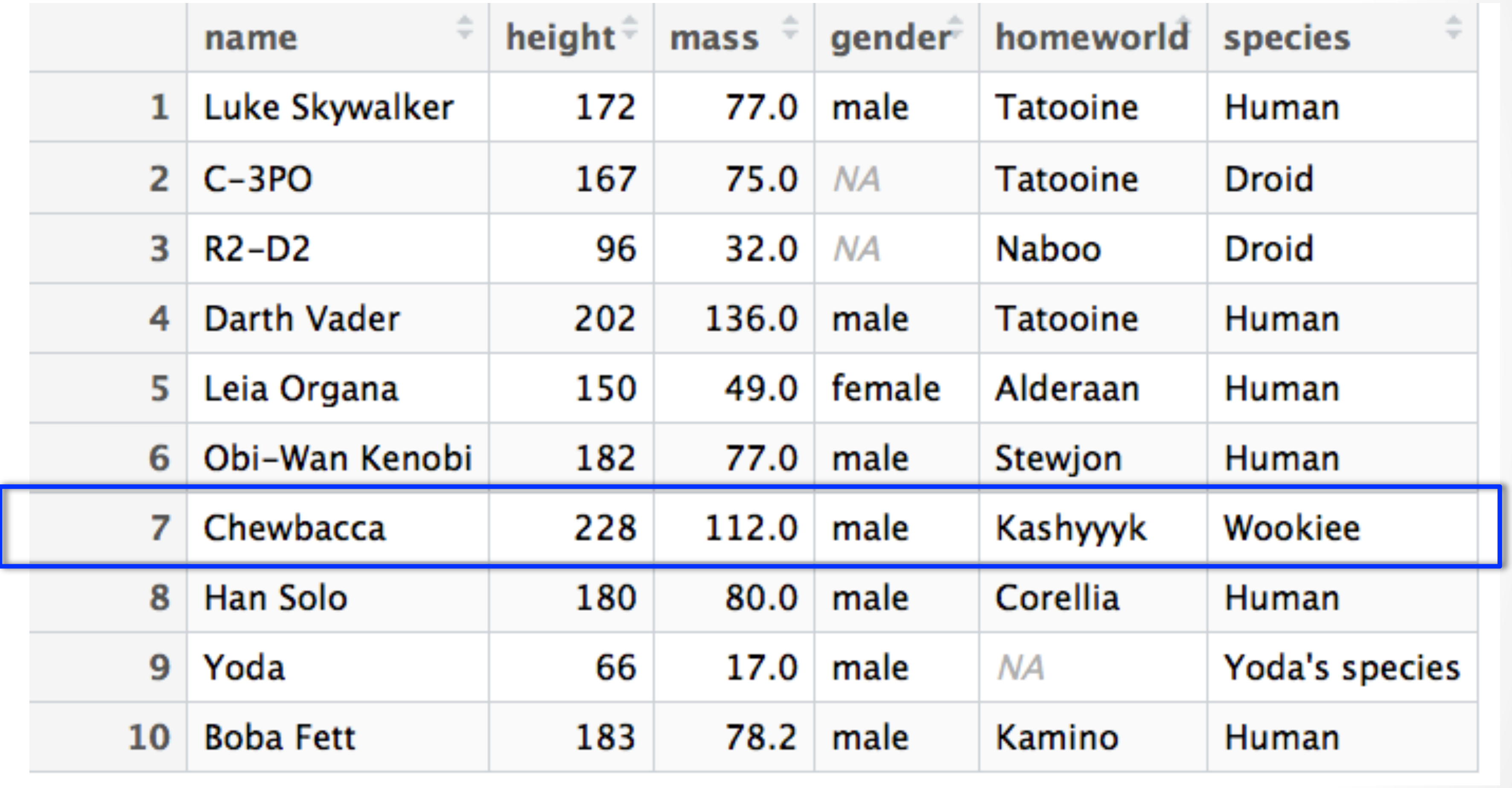
	name	height	mass	gender	homeworld	species
1	Luke Skywalker	172	77.0	male	Tatooine	Human
2	C-3PO	167	75.0	*NA*	Tatooine	Droid
3	R2-D2	96	32.0	*NA*	Naboo	Droid
4	Darth Vader	202	136.0	male	Tatooine	Human
5	Leia Organa	150	49.0	female	Alderaan	Human
6	Obi-Wan Kenobi	182	77.0	male	Stewjon	Human
7	Chewbacca	228	112.0	male	Kashyyyk	Wookiee
8	Han Solo	180	80.0	male	Corellia	Human
9	Yoda	66	17.0	male	*NA*	Yoda's species
10	Boba Fett	183	78.2	male	Kamino	Human

Data Frames



	name	height	mass	gender	homeworld	species
1	Luke Skywalker	172	77.0	male	Tatooine	Human
2	C-3PO	167	75.0	NA	Tatooine	Droid
3	R2-D2	96	32.0	NA	Naboo	Droid
4	Darth Vader	202	136.0	male	Tatooine	Human
5	Leia Organa	150	49.0	female	Alderaan	Human
6	Obi-Wan Kenobi	182	77.0	male	Stewjon	Human
7	Chewbacca	228	112.0	male	Kashyyyk	Wookiee
8	Han Solo	180	80.0	male	Corellia	Human
9	Yoda	66	17.0	male	NA	Yoda's species
10	Boba Fett	183	78.2	male	Kamino	Human

Data Frames



	name	height	mass	gender	homeworld	species
1	Luke Skywalker	172	77.0	male	Tatooine	Human
2	C-3PO	167	75.0	NA	Tatooine	Droid
3	R2-D2	96	32.0	NA	Naboo	Droid
4	Darth Vader	202	136.0	male	Tatooine	Human
5	Leia Organa	150	49.0	female	Alderaan	Human
6	Obi-Wan Kenobi	182	77.0	male	Stewjon	Human
7	Chewbacca	228	112.0	male	Kashyyyk	Wookiee
8	Han Solo	180	80.0	male	Corellia	Human
9	Yoda	66	17.0	male	NA	Yoda's species
10	Boba Fett	183	78.2	male	Kamino	Human

Data Frames

The way to create a data.frame:

- `data.frame()`

Exercise

Create a data.frame named `workshop`.

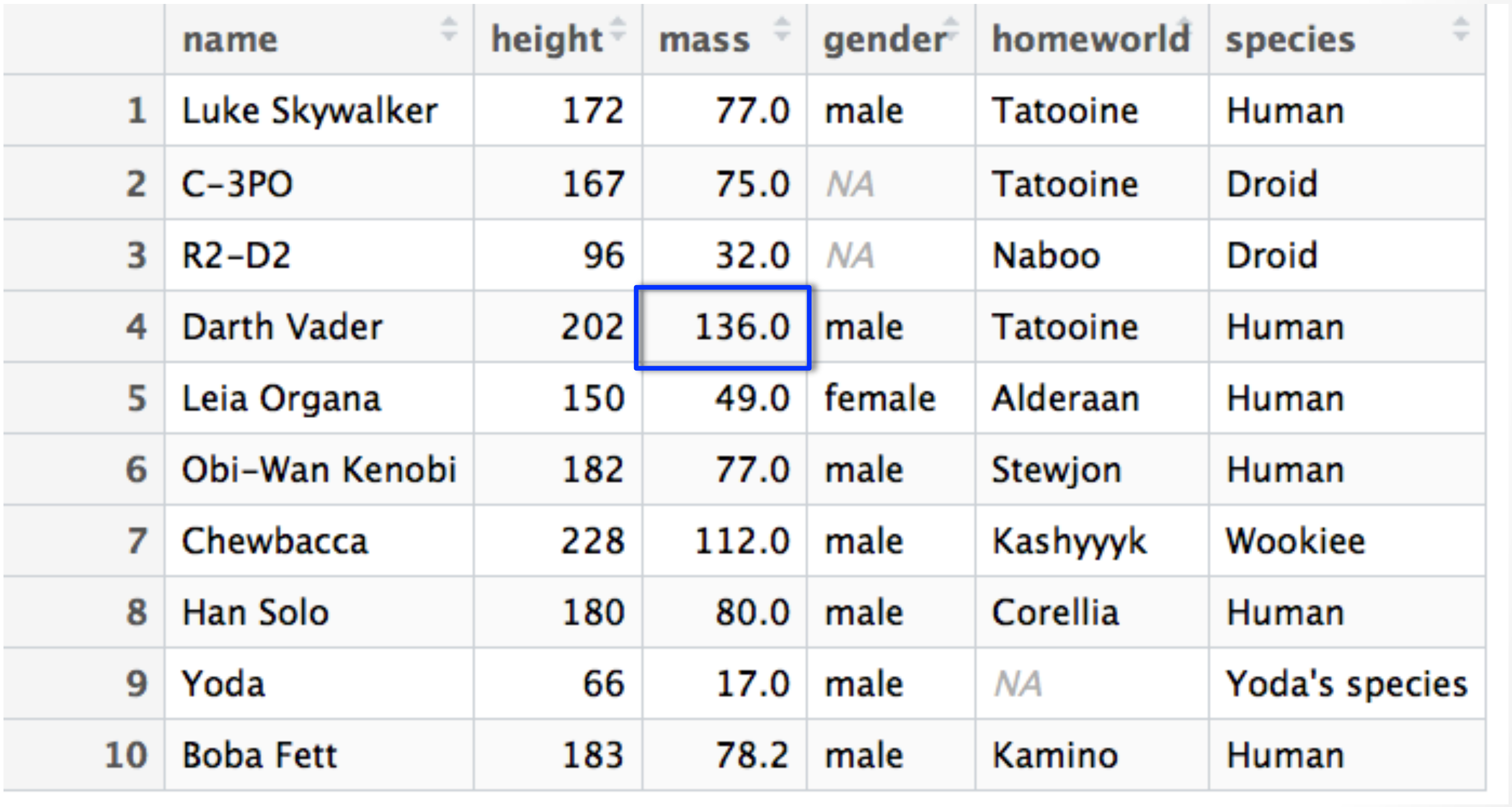
- Variables:
  - `subjectID: “Subject##”`
  - `bobsBurgers: T/F`
  - `countries: ##`
  - `personType: “dog” vs. “ant”`

OBJECTS

Object

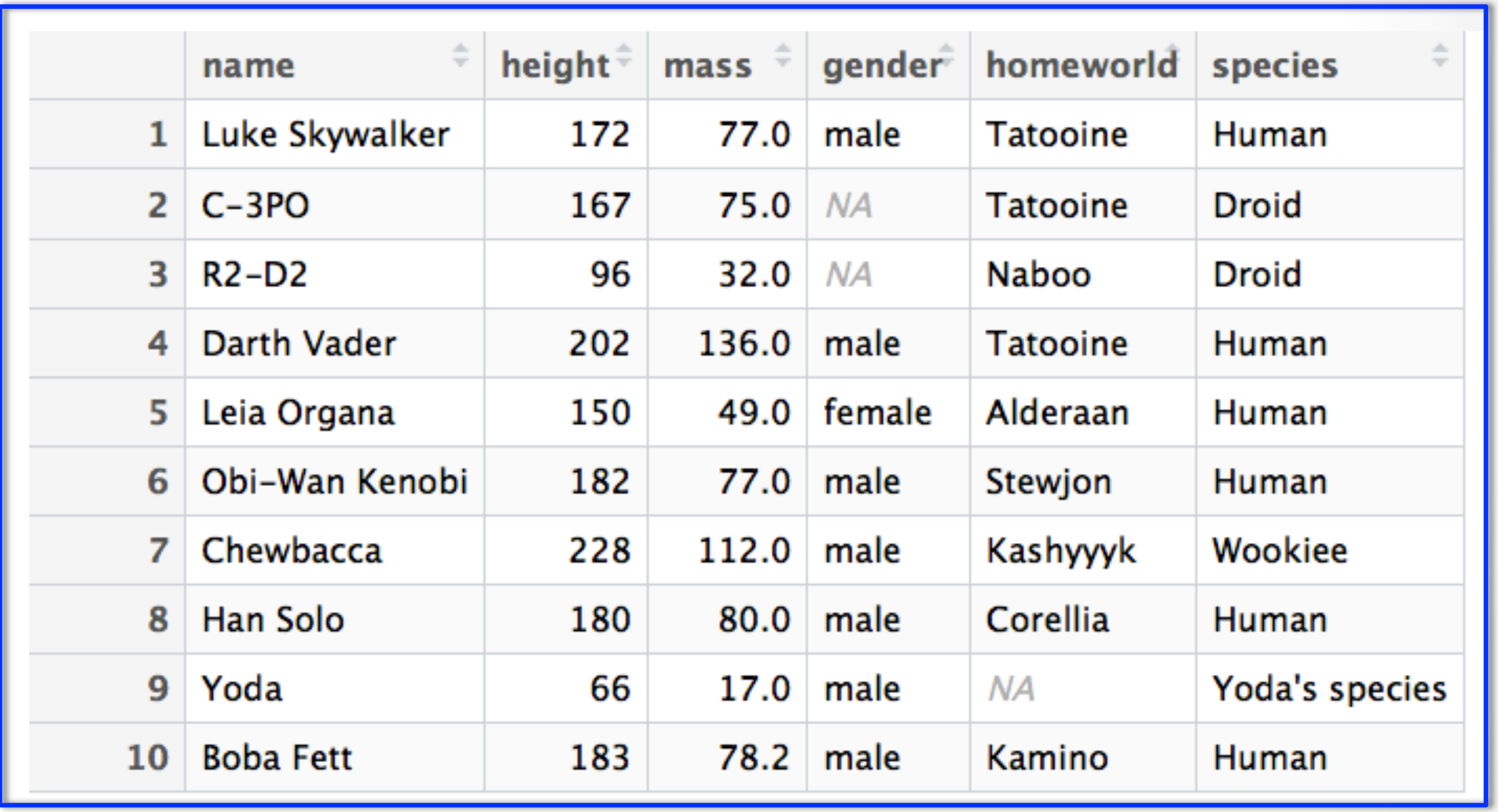
- A basic concept in (statistical) programming is called an **object**.
- An object allows you to store a value or a thing in R.
- You use the object's name to easily access this value or thing.

Data Frames



	name	height	mass	gender	homeworld	species
1	Luke Skywalker	172	77.0	male	Tatooine	Human
2	C-3PO	167	75.0	NA	Tatooine	Droid
3	R2-D2	96	32.0	NA	Naboo	Droid
4	Darth Vader	202	136.0	male	Tatooine	Human
5	Leia Organa	150	49.0	female	Alderaan	Human
6	Obi-Wan Kenobi	182	77.0	male	Stewjon	Human
7	Chewbacca	228	112.0	male	Kashyyyk	Wookiee
8	Han Solo	180	80.0	male	Corellia	Human
9	Yoda	66	17.0	male	NA	Yoda's species
10	Boba Fett	183	78.2	male	Kamino	Human

Data Frames



	name	height	mass	gender	homeworld	species
1	Luke Skywalker	172	77.0	male	Tatooine	Human
2	C-3PO	167	75.0	NA	Tatooine	Droid
3	R2-D2	96	32.0	NA	Naboo	Droid
4	Darth Vader	202	136.0	male	Tatooine	Human
5	Leia Organa	150	49.0	female	Alderaan	Human
6	Obi-Wan Kenobi	182	77.0	male	Stewjon	Human
7	Chewbacca	228	112.0	male	Kashyyyk	Wookiee
8	Han Solo	180	80.0	male	Corellia	Human
9	Yoda	66	17.0	male	NA	Yoda's species
10	Boba Fett	183	78.2	male	Kamino	Human

Object

```
pi
[1] 3.141593
```

Referencing objects

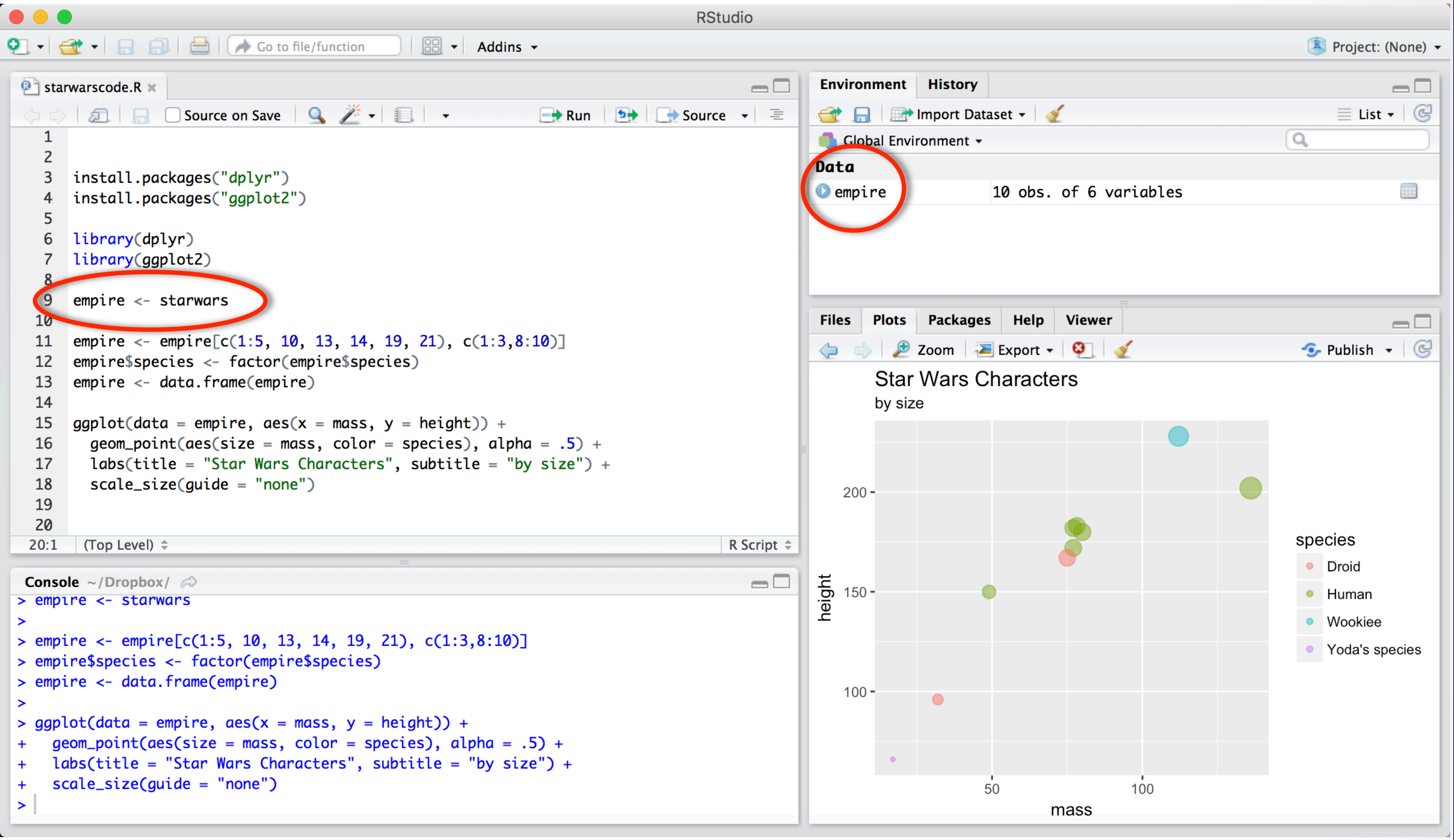
If you want to use an object later on, you have to name it.

```
my_object = thingtoassign
```

```
my_object <- thingtoassign
```

```
Ex) pi <- 3.141593
```

WE STRONGLY RECOMMEND USING THE SECOND WAY!!!!!



```r


install.packages("dplyr")
install.packages("ggplot2")

library(dplyr)
library(ggplot2)

empire <- starwars

empire <- empire[c(1:5, 10, 13, 14, 19, 21), c(1:3,8:10)]
empire$species <- factor(empire$species)
empire <- data.frame(empire)

ggplot(data = empire, aes(x = mass, y = height)) +
  geom_point(aes(size = mass, color = species), alpha = .5) +
  labs(title = "Star Wars Characters", subtitle = "by size") +
  scale_size(guide = "none")


```

20:1 (Top Level) R Script

Console ~/Dropbox/

```
> empire <- starwars
>
> empire <- empire[c(1:5, 10, 13, 14, 19, 21), c(1:3,8:10)]
> empire$species <- factor(empire$species)
> empire <- data.frame(empire)
>
> ggplot(data = empire, aes(x = mass, y = height)) +
+   geom_point(aes(size = mass, color = species), alpha = .5) +
+   labs(title = "Star Wars Characters", subtitle = "by size") +
+   scale_size(guide = "none")
>
```

Data

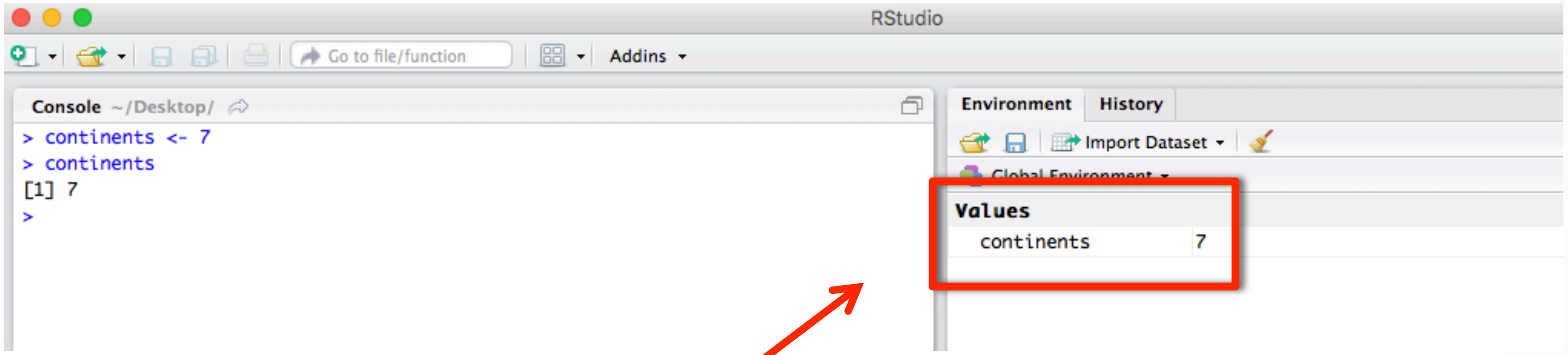
empire	10 obs. of 6 variables

- **Objects** can be used later!
- You can see them in your **Global Environment.**

Exercise

Try creating your own object!

1. In the console, assign the number `7` to `continents`
2. Then type `continents` in the console to print out the assigned value



```
> continents <- 7
> continents
[1] 7
>
```

Values	
continents	7

Look!
Your environment has stored your object!

BONUS: What is the difference between typing in the console or in a script file?

Exercise

Now, let's try adding together two objects.

In your **script**:

- Assign the value of `5` to an object called `chickens`.
- Assign the value of `4` to an object called `cows`.
- Give the command `chickens + cows` to find out how many animals you have!

  - ***BONUS: where is your answer stored?***

Basic Object Classes

Numeric: Decimals (3.141593)

Integer: Natural numbers (0, 1, 2…)

Character: Text or string characters

- Always inside quotations.
- **Factors** (or categories)

Logical: Boolean values (True or False)

- No quotations.
- 2 possible values: **TRUE** or **FALSE**

E	L	G	A	R	■	F	L	A	B	■	B	A	I	L
N	A	O	M	I	■	R	E	D	O	■	O	N	C	E
A	N	G	I	O	■	O	O	L	O	N	G	T	E	A
C	A	R	E	T	S	■	N	I	L	E	■	■	■	■
T	I	E	■	G	N	U	■	B	E	W	I	T	C	H
■	■	A	G	E	O	L	D	■	A	S	M	A	R	A
A	S	T	R	A	■	C	O	R	N	■	P	L	O	D
P	A	G	E	R	■	E	M	U	■	G	U	E	S	S
A	B	U	T	■	W	R	E	N	■	E	R	N	S	T
C	I	N	E	M	A	■	S	E	S	T	E	T	■	■
E	N	S	L	A	V	E	■	S	E	A	■	A	D	M
■	■	■	■	S	E	R	E	■	P	L	A	G	U	E
G	A	T	E	C	R	A	S	H	■	I	D	E	N	T
O	R	A	L	■	E	T	T	A	■	F	E	N	C	E
P	I	N	K	■	D	O	O	M	■	E	S	T	E	R

Factors

Character objects

- Character strings represent distinct groups.
    - Control, Treatment
    - Male, Female

Basic Data Class

To check what data class your object is, you can type `class()` into the console.

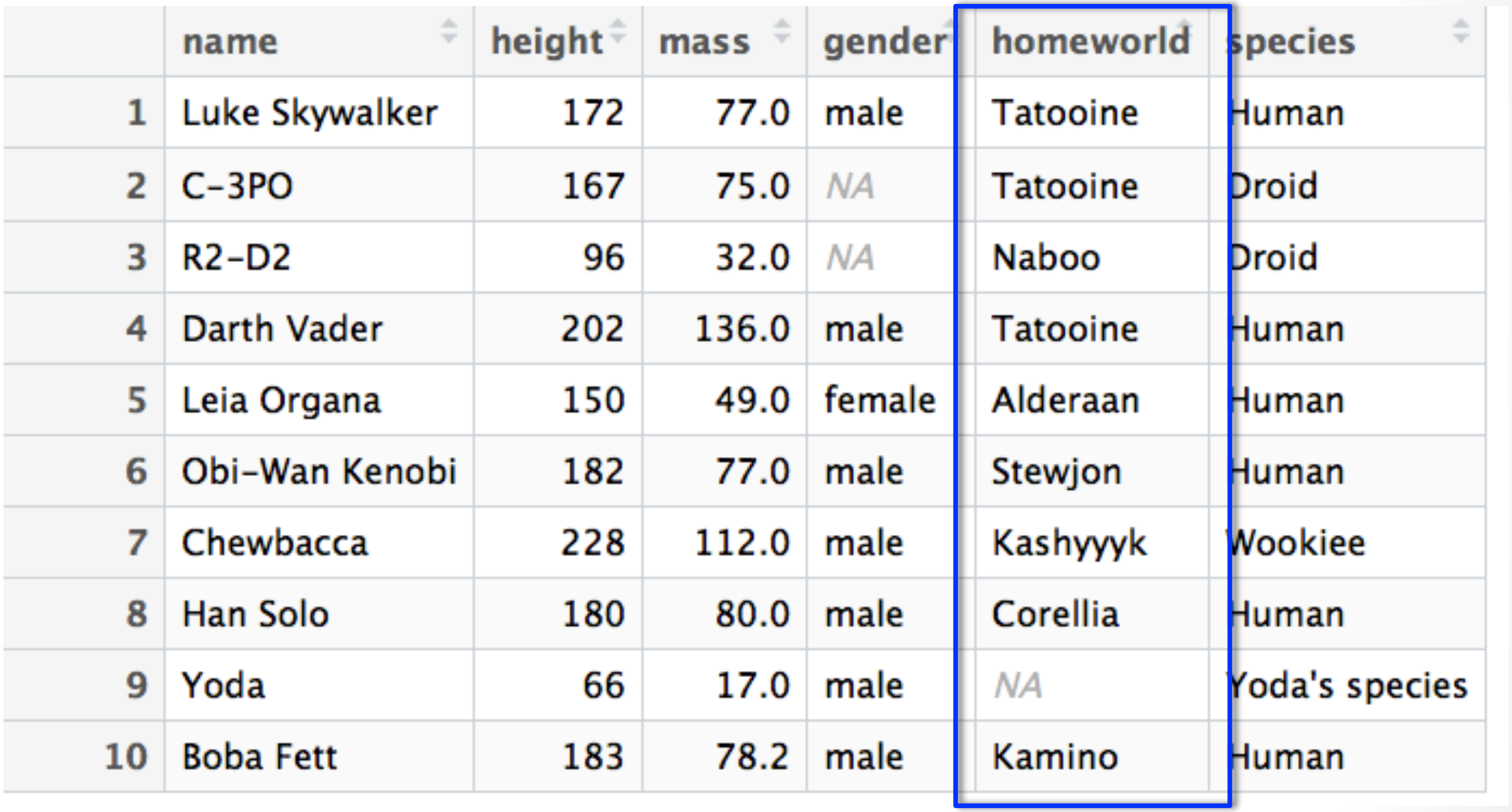
```
class(countries)
[1] “numeric”
```

Vectors

A group of objects is a **vector.**

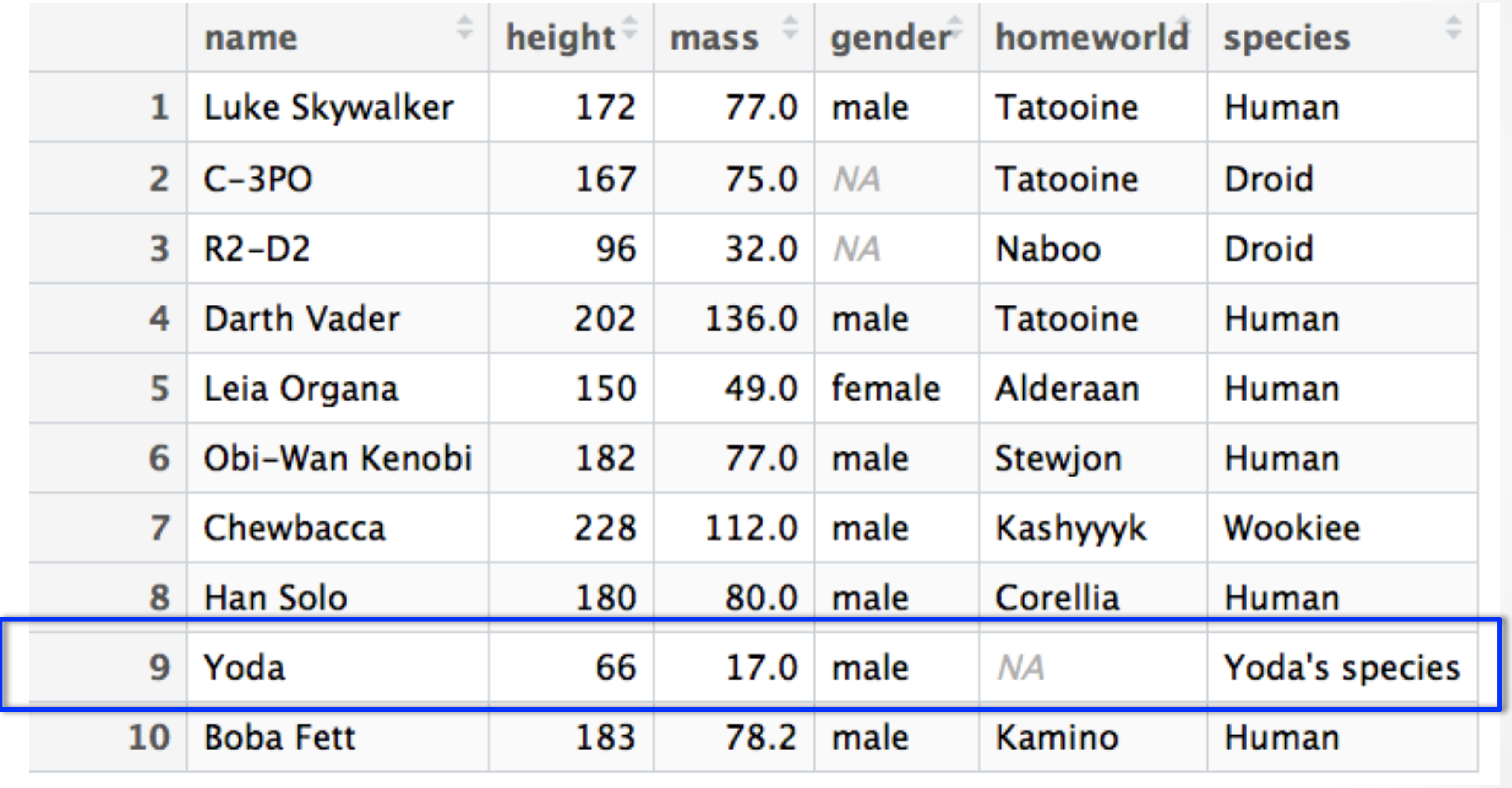
Vectors are ONE-DIMENSIONAL.

Vectors



	name	height	mass	gender	homeworld	species
1	Luke Skywalker	172	77.0	male	Tatooine	Human
2	C-3PO	167	75.0	NA	Tatooine	Droid
3	R2-D2	96	32.0	NA	Naboo	Droid
4	Darth Vader	202	136.0	male	Tatooine	Human
5	Leia Organa	150	49.0	female	Alderaan	Human
6	Obi-Wan Kenobi	182	77.0	male	Stewjon	Human
7	Chewbacca	228	112.0	male	Kashyyyk	Wookiee
8	Han Solo	180	80.0	male	Corellia	Human
9	Yoda	66	17.0	male	NA	Yoda's species
10	Boba Fett	183	78.2	male	Kamino	Human

Vectors



	name	height	mass	gender	homeworld	species
1	Luke Skywalker	172	77.0	male	Tatooine	Human
2	C-3PO	167	75.0	NA	Tatooine	Droid
3	R2-D2	96	32.0	NA	Naboo	Droid
4	Darth Vader	202	136.0	male	Tatooine	Human
5	Leia Organa	150	49.0	female	Alderaan	Human
6	Obi-Wan Kenobi	182	77.0	male	Stewjon	Human
7	Chewbacca	228	112.0	male	Kashyyyk	Wookiee
8	Han Solo	180	80.0	male	Corellia	Human
9	Yoda	66	17.0	male	NA	Yoda's species
10	Boba Fett	183	78.2	male	Kamino	Human

Indexing

Having a group of objects is great, but sometimes you only want one or a few of those objects.

How do we ACCESS our data?

Indexing a vector

To index a vector

- `[ ]`

For example, how many countries has the **3rd** person been to?

- `countries[3]`

Indexing

You can select multiple objects within a vector

To select objects that are sequential (in a row):

- `countries[3:5]`
- You can think of `:` as "through"
  - `[3:5]` = "three ***<u>through</u>*** five"

To select objects that are not in a row:

- `countries[c(1,2,5)]`

Indexing

You can go crazy and combine these!

- `countries[c(1:3, 5)]`

Indexing

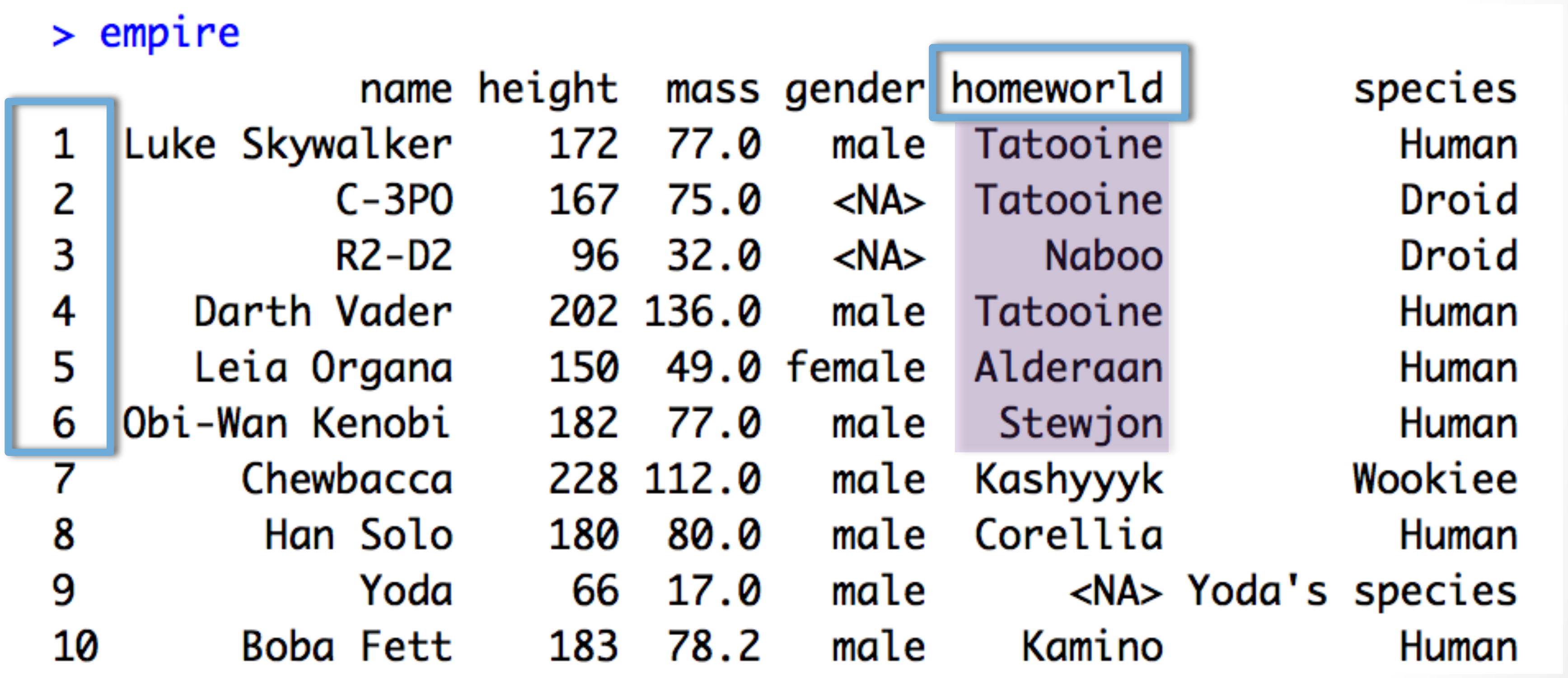
Data frames can be indexed just like vectors

EXCEPT: data frames have **2** dimensions

- Rows and columns

Indexing data frames

```
data.frame[rows, columns]
Ex: empire[1:6,5]
```

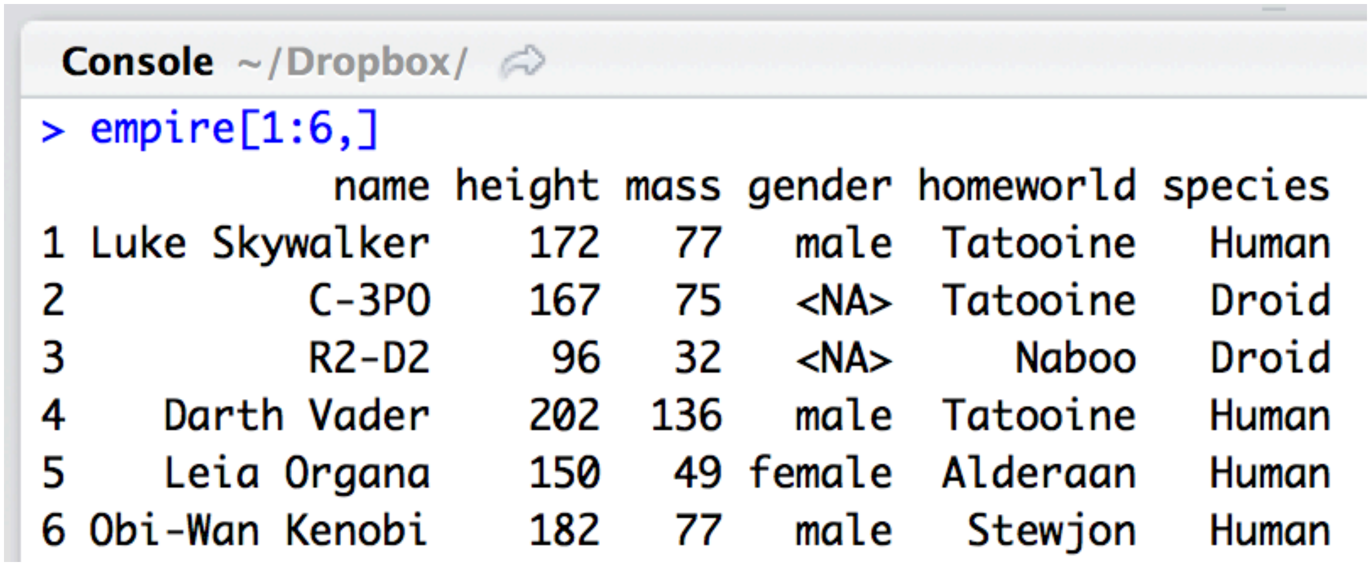


```
> empire
            name height  mass gender homeworld       species
1 Luke Skywalker    172  77.0   male  Tatooine         Human
2          C-3PO    167  75.0   <NA>  Tatooine         Droid
3          R2-D2     96  32.0   <NA>     Naboo         Droid
4    Darth Vader    202 136.0   male  Tatooine         Human
5    Leia Organa    150  49.0 female  Alderaan         Human
6 Obi-Wan Kenobi    182  77.0   male   Stewjon         Human
7      Chewbacca    228 112.0   male  Kashyyyk       Wookiee
8       Han Solo    180  80.0   male  Corellia         Human
9           Yoda     66  17.0   male      <NA> Yoda's species
10     Boba Fett    183  78.2   male    Kamino         Human
```

Indexing data frames

```
empire[1:6,]
```

**If you want all of something, leave it blank.



Console ~/Dropbox/

```
> empire[1:6,]
            name height mass gender homeworld species
1 Luke Skywalker    172   77   male  Tatooine   Human
2          C-3PO    167   75   <NA>  Tatooine   Droid
3          R2-D2     96   32   <NA>     Naboo   Droid
4    Darth Vader    202  136   male  Tatooine   Human
5    Leia Organa    150   49 female  Alderaan   Human
6 Obi-Wan Kenobi    182   77   male   Stewjon   Human
```

Finding Your Data

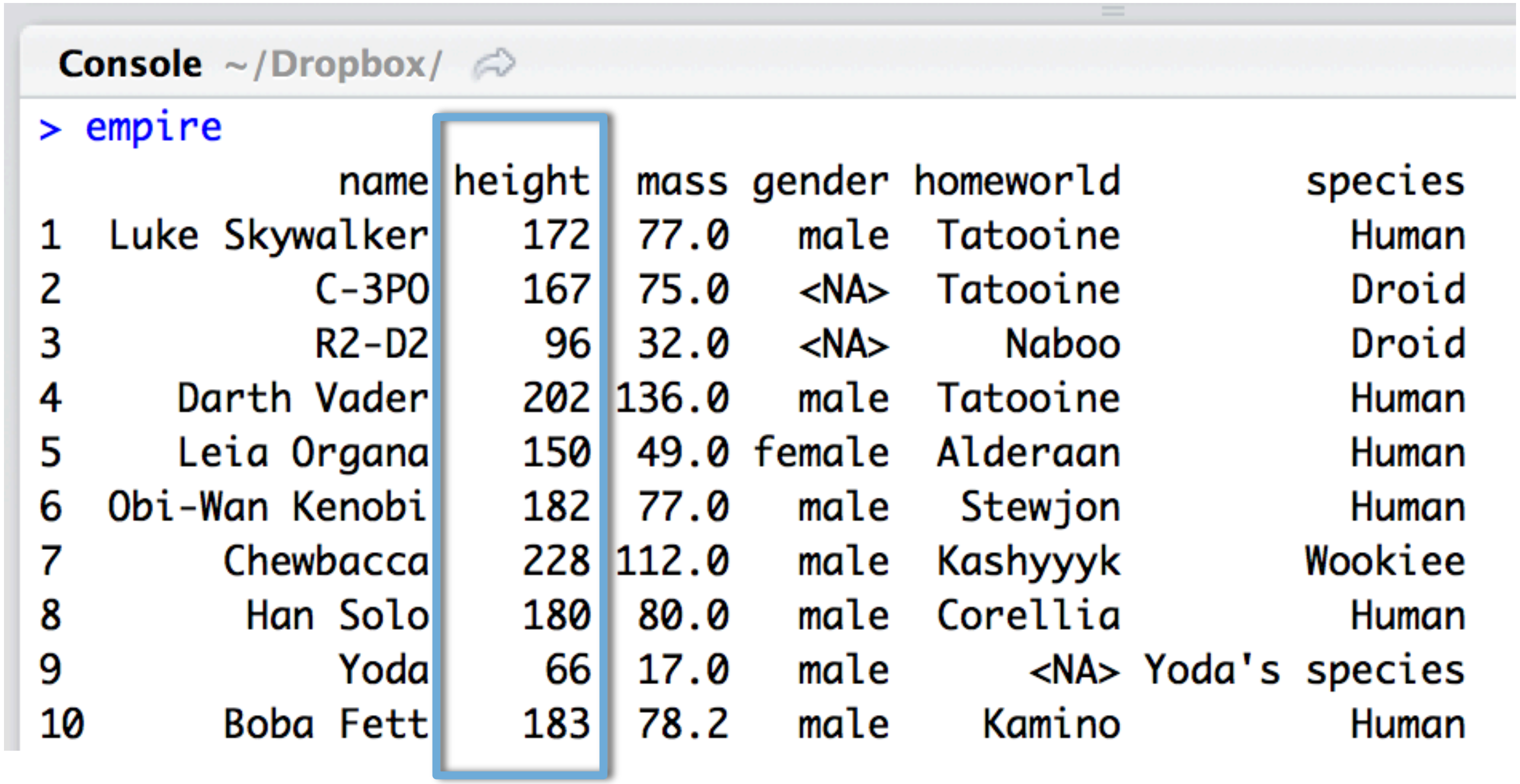
Most of the time, you're not working with an entire data.frame.

In a data.frame, our **columns** have names—we can use these names instead of memorizing what the column number is!

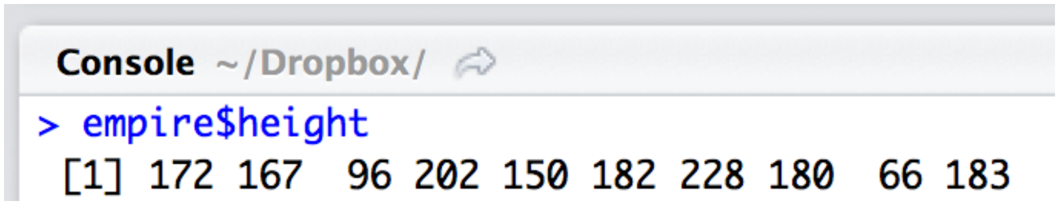
You can access just one column at a time by using $

Finding Your Data

$ means you are accessing just one column within your data frame



```
> empire
              name height  mass gender homeworld       species
1   Luke Skywalker    172  77.0   male  Tatooine         Human
2            C-3PO    167  75.0   <NA>  Tatooine         Droid
3            R2-D2     96  32.0   <NA>     Naboo         Droid
4      Darth Vader    202 136.0   male  Tatooine         Human
5      Leia Organa    150  49.0 female  Alderaan         Human
6   Obi-Wan Kenobi    182  77.0   male   Stewjon         Human
7        Chewbacca    228 112.0   male  Kashyyyk       Wookiee
8         Han Solo    180  80.0   male  Corellia         Human
9             Yoda     66  17.0   male      <NA> Yoda's species
10       Boba Fett    183  78.2   male    Kamino         Human
```



```
> empire$height
 [1] 172 167  96 202 150 182 228 180  66 183
```

ACTING ON VARIABLES

Acting on variables

Actions

- Operators
- Functions

Operators

An **operator** is a simple calculation.

Operators

+	addition
-	subtraction
*	multiplication
/	division
^	taking powers

Order of Operations

Important note: Order of operations matters!

```
(8-4)/2
[1] 2
```

```
8-(4/2)
[1] 6
```

PEMDAS, anyone?

Logical Operators

==	equality
!=	inequality
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

Logical Operators

- Return a value of **TRUE** or **FALSE**
- `workshop$personType == “ant”`

Example

- Test whether `countries` is greater than 3.

Logical Operators

Has anyone in our data.frame visited more than 3 countries?

```
workshop$countries >  3
```

Has anyone in our data.frame visited exactly 3 countries?

```
workshop$countries == 3
```

```
workshop$countries  = 3
```

→ WARNING: a SINGLE equals sign will change your data!

ACTING ON VARIABLES: FUNCTIONS

Functions

- Sometimes, you want to do more than add or multiply variables.

- To perform more complicated actions, use *functions.*
  - Functions are commands that describe, manipulate or analyze objects.

Functions have three parts

1. Function name
   - *Ex:* `log(10)`
     ```
     [1] 2.302
     ```

2. Arguments
   - *Ex:* `log(10)`
     ```
     [1] 2.302
     ```

3. Output
   - *Ex:* `log(10)`
     ```
     [1] 2.302
     ```

Each function has one and only one name.

Functions have three parts

1. Function name
   - *Ex:* `log(10)`

     `[1] 2.302`

2. Arguments
   - *Ex:* `log(10)`

     `[1] 2.302`

3. Output
   - *Ex:* `log(10)`

     `[1] 2.302`

One argument is always specified: the input. This is the object that the function acts on.

Other arguments control **how** the function acts. For example, do you want the natural log? Or log base 10?

Each function has defaults for its arguments. You should know what those are and how to change them.

Functions have three parts

1. Function name
   - *Ex:* `log(10)`

     `[1] 2.302`

2. Arguments
   - *Ex:* `log(10)`

     `[1] 2.302`

3. Output
   - *Ex:* `log(10)`

     `[1] 2.302`

Output can be a:

number/integer
a TRUE/FALSE statement
a character value
all of the above

Output can be a:

single value
vector
data frame
matrix
list

You can store the output by assigning it to another object!

Mathematical functions

`sqrt()`	square root
`round()`	round a number
`log()`	logarithm
`exp()`	exponentiation
`abs()`	absolute value

Example

1. Find the `square root` of `85`.
2. Take the `log` of `100`.

Example

```
sqrt(85)
[1] 9.219544
log(100)
[1] 4.60517
```

Functions you'll use a lot!

`c()` – combine or concatenate

`class()` – check the class of an object

`factor()` – change a character vector into a factor vector (is there meaning? Ex: Treatment vs. Control, Male vs. Female, Session 1 vs. Session 2)

`table()` – really nice for getting quick counts (Ex: how many males and females are there?)

Exercise

Get the `mean()` of the everyone’s # of `countries` visited in our `workshop` data.frame.

What is the name of the function?

What is the input argument?

What is the output?

Multiple arguments

Most functions take more than one argument.

Separate arguments with commas.

```
round (x = 5.86921, digits = 3)
[1] 5.869
```

Number that needs to be rounded.

Multiple arguments

Most functions take more than one argument.

Separate arguments with commas.

```
round (x = 5.86921, digits = 3)
[1] 5.869
```

Number of digits
to round to.

Arguments have Names

Most arguments in functions have names.

USE THE NAMES!!!

```
round (x = 5.86921, digits = 3)
[1] 5.869
```

Exercise

1. Use the `seq()` function to list numbers `0` to `100`.

   Arguments:
   - `from` = starting value of sequence
   - `to` = end value of sequence

2. Use the `seq()` function to list numbers `0` to `100`, by intervals of `10`.

   Arguments:
   - `from` = starting value of sequence
   - `to` = end value of sequence
   - `by` = increment of the sequence

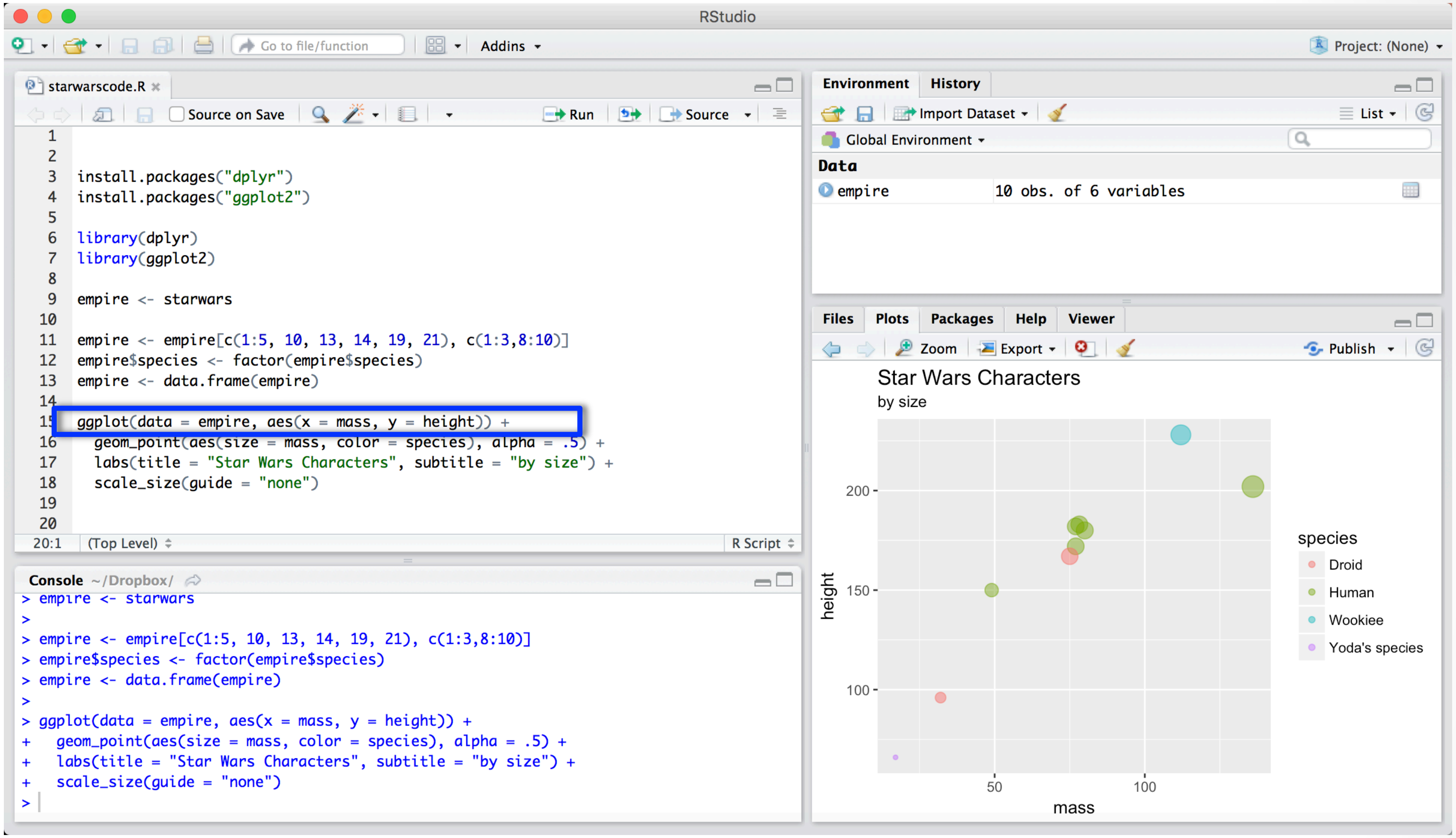
Exercise

1. Use the `seq()` function to list numbers 0 to 100

    - `seq(from = 0, to = 100)`

2. Use the `seq()` function to list numbers 0 to 100, by intervals of 10

    - `seq(from = 0, to = 100, by = 10)`



```
install.packages("dplyr")
install.packages("ggplot2")

library(dplyr)
library(ggplot2)

empire <- starwars

empire <- empire[c(1:5, 10, 13, 14, 19, 21), c(1:3,8:10)]
empire$species <- factor(empire$species)
empire <- data.frame(empire)

ggplot(data = empire, aes(x = mass, y = height)) +
  geom_point(aes(size = mass, color = species), alpha = .5) +
  labs(title = "Star Wars Characters", subtitle = "by size") +
  scale_size(guide = "none")
```

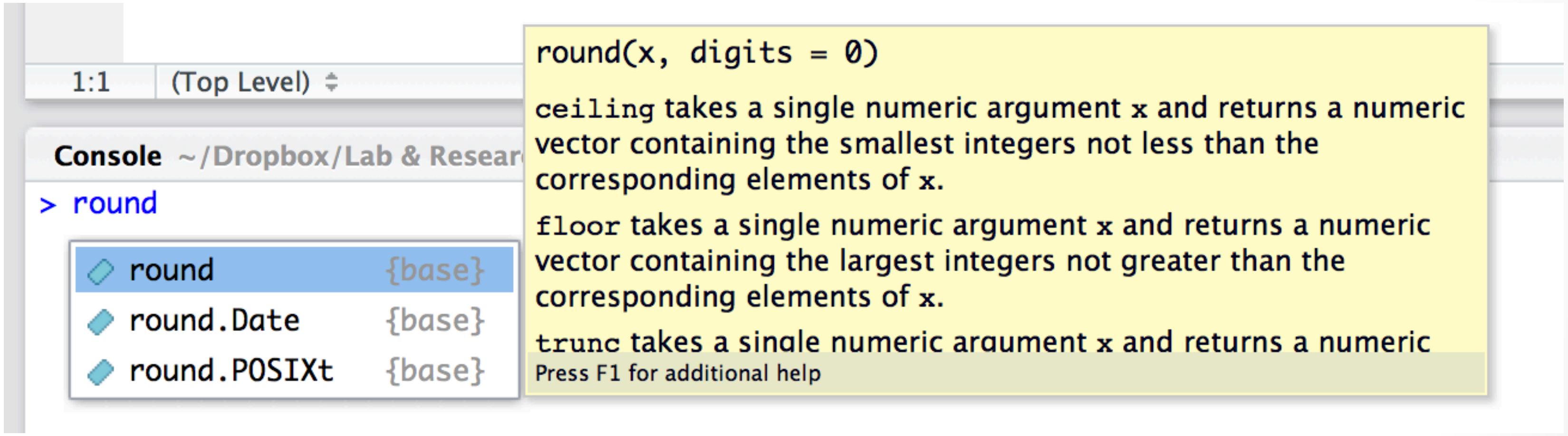
Environment — Global Environment — Data: empire 10 obs. of 6 variables

```
> empire <- starwars
>
> empire <- empire[c(1:5, 10, 13, 14, 19, 21), c(1:3,8:10)]
> empire$species <- factor(empire$species)
> empire <- data.frame(empire)
>
> ggplot(data = empire, aes(x = mass, y = height)) +
+   geom_point(aes(size = mass, color = species), alpha = .5) +
+   labs(title = "Star Wars Characters", subtitle = "by size") +
+   scale_size(guide = "none")
>
```

Great, but how do I know what the arguments are for a function?

Two ways:

1) In RStudio, press the **tab** key to see names of arguments and descriptions.



```
1:1    (Top Level) ÷

Console ~/Dropbox/Lab & Resear
> round

round        {base}
round.Date   {base}
round.POSIXt {base}
```

```
round(x, digits = 0)
```

`ceiling` takes a single numeric argument `x` and returns a numeric vector containing the smallest integers not less than the corresponding elements of `x`.

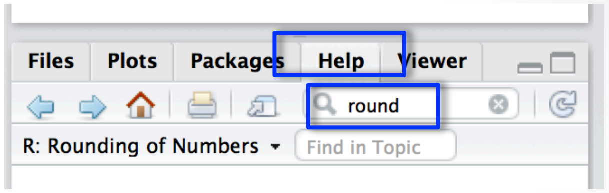
`floor` takes a single numeric argument `x` and returns a numeric vector containing the largest integers not greater than the corresponding elements of `x`.

`trunc` takes a single numeric argument `x` and returns a numeric

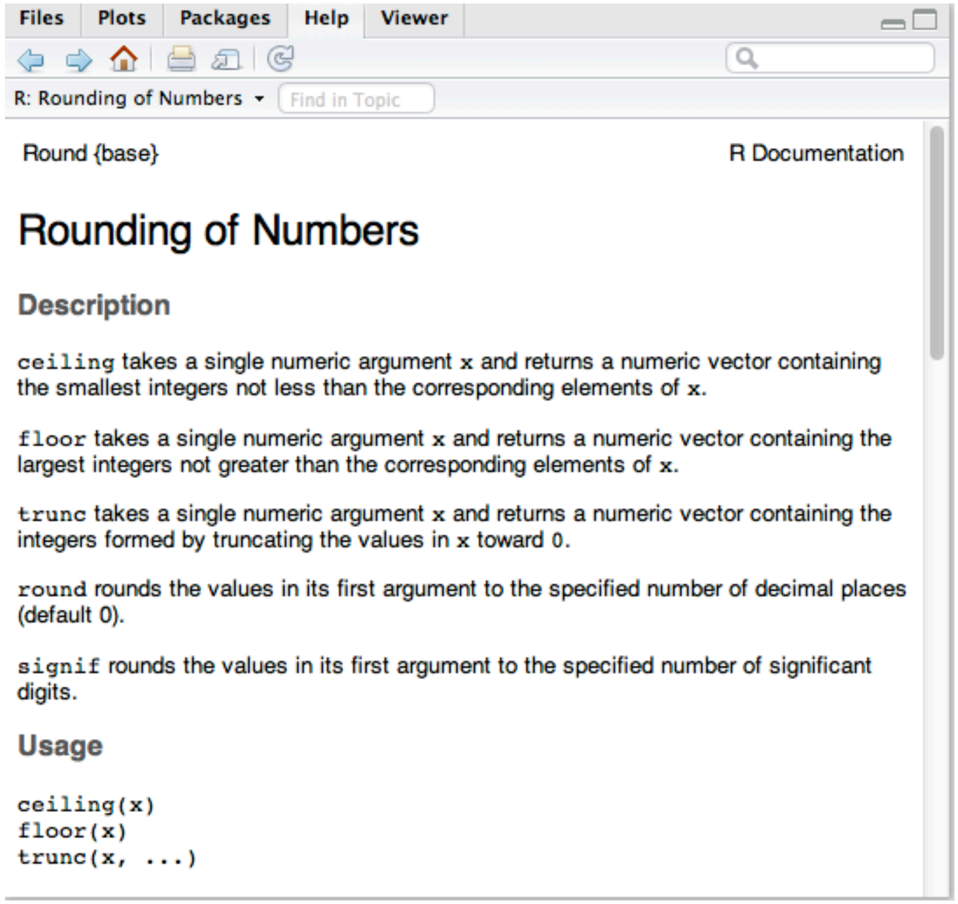
Press F1 for additional help

2) Look in the R documentation!

- Go to Help tab



- Or just type `?round` into the console



Round {base}	R Documentation

Rounding of Numbers

Description

`ceiling` takes a single numeric argument `x` and returns a numeric vector containing the smallest integers not less than the corresponding elements of `x`.

`floor` takes a single numeric argument `x` and returns a numeric vector containing the largest integers not greater than the corresponding elements of `x`.

`trunc` takes a single numeric argument `x` and returns a numeric vector containing the integers formed by truncating the values in `x` toward `0`.

`round` rounds the values in its first argument to the specified number of decimal places (default 0).

`signif` rounds the values in its first argument to the specified number of significant digits.

Usage

```
ceiling(x)
floor(x)
trunc(x, ...)
```

Exercise

1. Look up documentation for `scale` and `plot`.

2. Using the `height` variable from our `empire` data.frame, make a new variable called `height_z`, using `scale`.

3. Do the same thing for `mass`.

4. Combine `height_z` vector and `mass_z` vector into a new data.frame called `empire_z`.

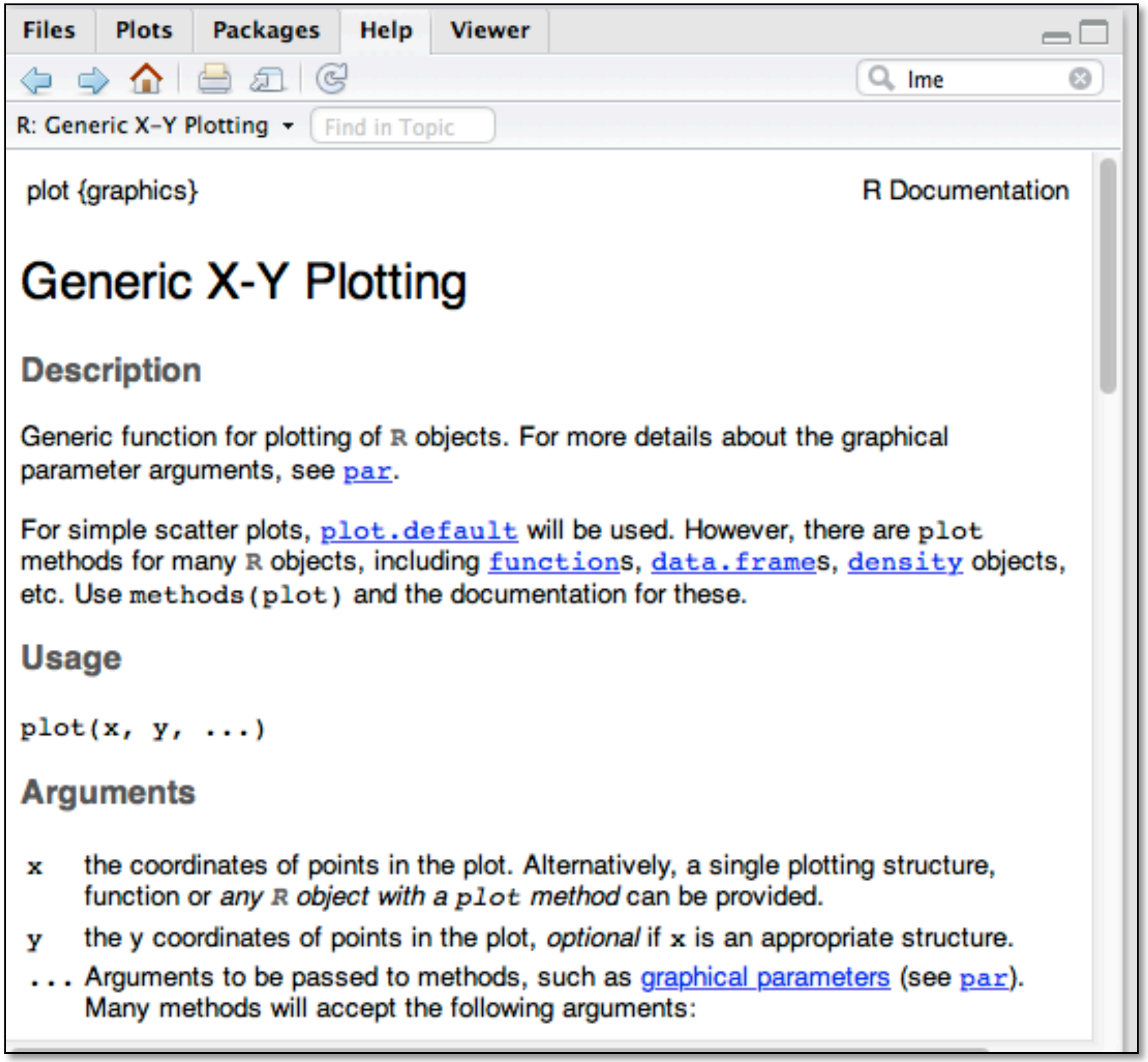
5. Make a scatter plot of standardized height (*hint: y axis*) by standardized mass, using the `plot` function.

6. Add a title to your plot.

7. Add labels to the x and y axes.

What happens if you add `type = “l”` ?

What is the default for type?



plot {graphics} R Documentation

Generic X-Y Plotting

Description

Generic function for plotting of `R` objects. For more details about the graphical parameter arguments, see `par`.

For simple scatter plots, `plot.default` will be used. However, there are `plot` methods for many `R` objects, including `function`s, `data.frame`s, `density` objects, etc. Use `methods(plot)` and the documentation for these.

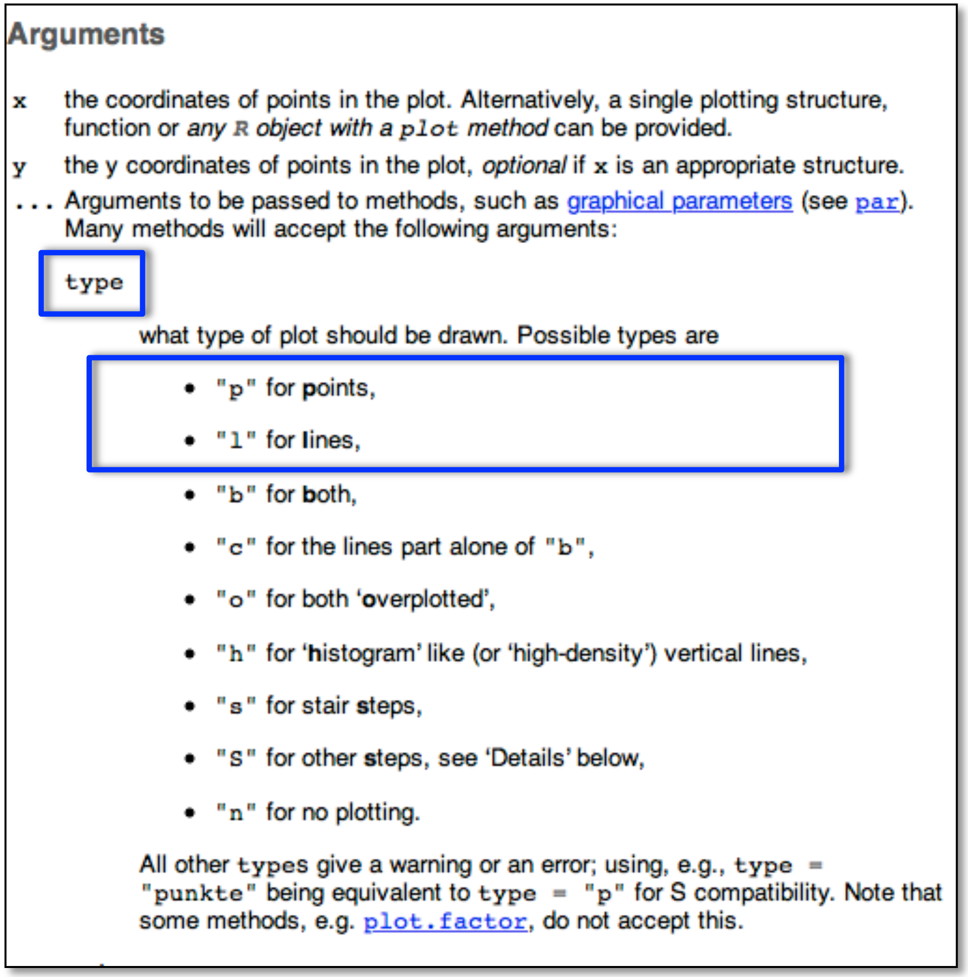
Usage

```
plot(x, y, ...)
```

Arguments

`x`	the coordinates of points in the plot. Alternatively, a single plotting structure, function or *any `R` object with a `plot` method* can be provided.
`y`	the y coordinates of points in the plot, *optional* if `x` is an appropriate structure.
`...`	Arguments to be passed to methods, such as graphical parameters (see `par`). Many methods will accept the following arguments:

Back to the Documentation!



Arguments

`x` the coordinates of points in the plot. Alternatively, a single plotting structure, function or *any `R` object with a `plot` method* can be provided.

`y` the y coordinates of points in the plot, *optional* if `x` is an appropriate structure.

`...` Arguments to be passed to methods, such as graphical parameters (see `par`). Many methods will accept the following arguments:

`type`

what type of plot should be drawn. Possible types are

- `"p"` for **p**oints,
- `"l"` for **l**ines,
- `"b"` for **b**oth,
- `"c"` for the lines part alone of `"b"`,
- `"o"` for both '**o**verplotted',
- `"h"` for '**h**istogram' like (or 'high-density') vertical lines,
- `"s"` for stair **s**teps,
- `"S"` for other **s**teps, see 'Details' below,
- `"n"` for no plotting.

All other `types` give a warning or an error; using, e.g., `type = "punkte"` being equivalent to `type = "p"` for S compatibility. Note that some methods, e.g. `plot.factor`, do not accept this.