

Overview

1. Data frames
2. Importing and exporting data
3. Lists
4. Subsetting


Data frames

Data frames

A data frame is a tabular object consisting of **rows** and **columns**.

Row names

Column names



	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	110	3.9	2.6
Mazda RX4 Wag	21.0	6	160	110	3.9	2.9
Datsun 710	22.8	4	108	93	3.9	2.3
Hornet 4 Drive	21.4	6	258	110	3.1	3.2
Hornet Sportabout	18.7	8	360	175	3.2	3.4
Valiant	18.1	6	225	105	2.8	3.5
Duster 360	14.3	8	360	245	3.2	3.6
Merc 240D	24.4	4	147	62	3.7	3.2
Merc 230	22.8	4	141	95	3.9	3.2

This is similar to Stata or SPSS datasets, or Excel spreadsheets.

Data frames

We can create data frames with the `data.frame` function:

```
# Define three vectors
> x <- sample(1:100, 5)
> y <- letters[1:5]
> z <- rnorm(5)

# Combine them to create a data frame
> data.frame(x, y, z)
```

	x	y	z
1	41	a	0.41417411
2	88	b	-0.61963691
3	21	c	0.12795071
4	10	d	-1.20726869
5	37	e	-0.07773731

You can define the vectors and data frame together:

```
> data.frame(var1 = c(1, 2, 3, 4, 5),  
>             var2 = seq(1, 5),  
>             var3 = 1:5)
```

var1	var2	var3
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5

Most of the time, we're **importing** data from other sources (e.g., from SPSS, Stat, Excel, or CSV).

(More on that in a moment).

5 things you should know about data frames

1. All variables must be of **equal length**.

5 things you should know about data frames

1. All variables must be of **equal length**.

```
> data.frame(1:5, 1:4)
```

```
Error in data.frame(1:5, 1:4)
```

```
arguments imply differing number of rows: 5, 4
```

5 things you should know about data frames

1. All variables must be of **equal length**.
2. Variables can be of **different types**. (Unlike a matrix).

5 things you should know about data frames

1. All variables must be of **equal length**.
2. Variables can be of **different types**. (Unlike a matrix).
3. A data frame has row and column **names**. These can be accessed (and changed) with `rownames` and `names`.

5 things you should know about data frames

1. All variables must be of **equal length**.
2. Variables can be of **different types**. (Unlike a matrix).
3. A data frame has row and column **names**. These can be accessed (and changed) with `rownames` and `names`.

```
> mydata <- data.frame(v1 = 1:4,  
>                      v2 = c("Strongly agree",  
>                             "Agree",  
>                             "Disagree",  
>                             "Strongly agree"))  
> names(mydata)  
[1] "v1"  "v2"  
>  
> names(mydata) <- c("id", "response")
```

5 things you should know about data frames

1. All variables must be of **equal length**.
2. Variables can be of **different types**. (Unlike a matrix).
3. A data frame has row and column **names**. These can be accessed (and changed) with `rownames` and `names`.
4. You can **count** the number of rows/columns with `nrow` and `ncol`.

5 things you should know about data frames

1. All variables must be of **equal length**.
2. Variables can be of **different types**. (Unlike a matrix).
3. A data frame has row and column **names**. These can be accessed (and changed) with `rownames` and `names`.
4. You can **count** the number of rows/columns with `nrow` and `ncol`.

```
> nrow(mydata)
```

```
[1] 2
```

```
> ncol(mydata)
```

```
[1] 4
```

5 things you should know about data frames

1. All variables must be of **equal length**.
2. Variables can be of **different types**. (Unlike a matrix).
3. A data frame has row and column **names**. These can be accessed (and changed) with `rownames` and `names`.
4. You can **count** the number of rows/columns with `nrow` and `ncol`.
5. You can view the top/bottom of a data frame with `head` and `tail`.

5 things you should know about data frames

1. All variables must be of **equal length**.
2. Variables can be of **different types**. (Unlike a matrix).
3. A data frame has row and column **names**. These can be accessed (and changed) with `rownames` and `names`.
4. You can **count** the number of rows/columns with `nrow` and `ncol`.
5. You can view the top/bottom of a data frame with `head` and `tail`.

```
head(mydata)      # Returns 6 rows by default  
tail(mydata)  
head(mydata, 20)  # But we can request more.
```

These functions (`head/tail`) can be applied to other objects.

Importing data

Importing data

We often need to work with data stored as:

.csv	Comma- or tab-separated values (CSV, TSV)
.xlsx	Microsoft Excel files
.sav	SPSS datasets
.dta	Stata datasets

(Any others?).

readr



CSV and TSV

readxl



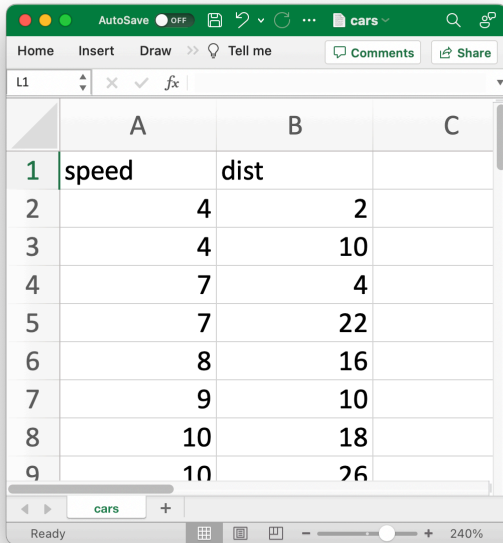
Microsoft Excel

haven



Stata, SPSS, SAS

Importing CSVs with `readr`



The screenshot shows a Google Sheets interface with a spreadsheet titled "cars". The spreadsheet contains a table with 9 rows and 3 columns. The columns are labeled "speed", "dist", and an unlabeled column. The rows contain numerical data representing car performance metrics. The interface includes a top menu bar with "Home", "Insert", and "Draw" options, a search bar, and a "Tell me" button. The bottom status bar shows "Ready" and a zoom level of 240%.

	A	B	C
1	speed	dist	
2	4	2	
3	4	10	
4	7	4	
5	7	22	
6	8	16	
7	9	10	
8	10	18	
9	10	26	

```
> library(readr)
> cars <- read_csv("cars.csv")
```

```
Rows: 50 Columns: 2
```

```
-- Column specification -----
```

```
Delimiter: ","
```

```
dbl (2): speed, dist
```

Use `spec()` to retrieve the full column specification for this data.

```
> head(cars)
```

	speed	dist
1	4	2
2	4	10
3	7	4
4	7	22
5	8	16
6	9	10

It's not always so easy...

```
le_cols <- cols(`Country Name` = col_character(),  
               `Country Code` = col_character(),  
               `Series Name` = col_character(),  
               `Series Code` = col_character(),  
               `2019 [YR2019]` = col_double())  
  
life_expectancy <- read_csv("life_expectancy.csv",  
                           n_max = 49,  
                           na = "..",  
                           col_types = le_cols)
```

Importing Excel files with `readxl`

```
> library(readxl)
> cars <- read_xlsx("cars.xlsx")
```

Like `read_csv`, column types will be guessed. This isn't always helpful. Some useful arguments:

<code>sheet</code>	Specify a sheet (e.g., <code>1</code> or <code>"Sheet1"</code>).
<code>range</code>	Specify a range (e.g., <code>B3:D87</code>).
<code>col_types</code>	Specify the column types.
<code>na</code>	Specify which values should be interpreted as <code>NA</code> .
<code>skip</code>	Skip the first few rows (e.g., <code>skip = 5</code>).
<code>n_max</code>	Specify how many rows to import.

Importing Stata/SPSS/SAS files with haven

```
> library(haven)
> df_stata <- read_dta("stata.dta")
> df_spss <- read_sav("spss.sav")
> df_sas <- read_sav("sas.sas7bdat")
```

Importing Stata/SPSS/SAS files with `haven`

```
> library(haven)
> df_stata <- read_dta("stata.dta")
> df_spss <- read_sav("spss.sav")
> df_sas <- read_sav("sas.sas7bdat")
```

We can also import from a URL:

```
> df_stata <- read_dta(
>   "http://www.stata-press.com/data/r17/auto.dta"
> )
```


To export, find the corresponding `write_*` function:

- `write_csv`
- `write_xlsx` (from the `writexl` package).
- `write_dta`, `write_sav`

To learn more:

<https://r4ds.hadley.nz/import.html>

And then:

<https://readr.tidyverse.org>

<https://readxl.tidyverse.org>

<https://haven.tidyverse.org>

<https://docs.ropensci.org/writexl/>



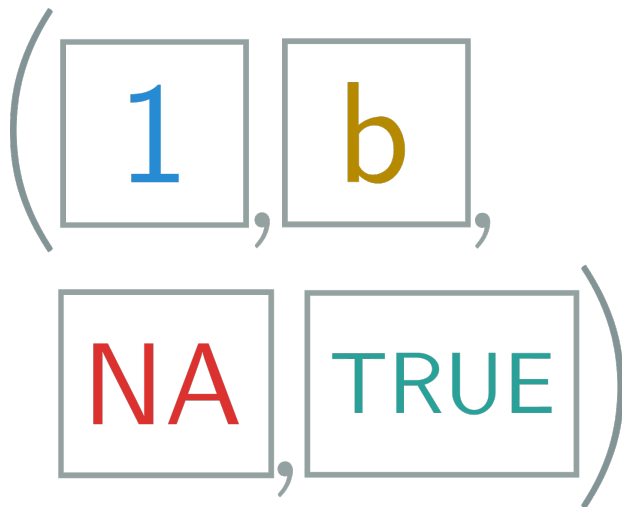
Lists

Lists

- A `list` can contain other objects (of any type).

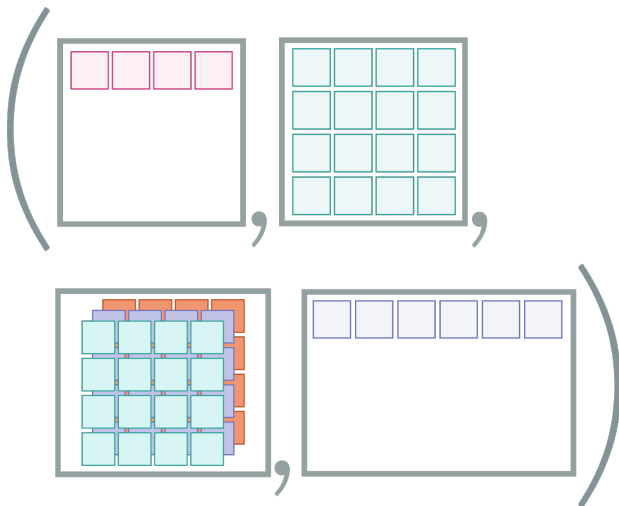
Lists

- A `list` can contain other objects (of any type).



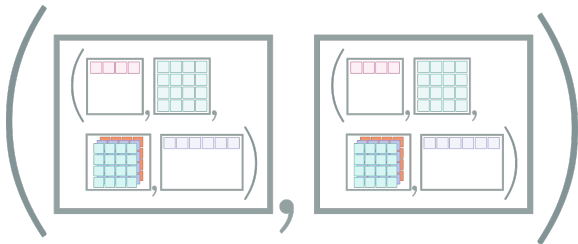
Lists

- A **list** can contain other objects (of any type).



Lists

- A **list** can contain other objects (of any type).



Even other lists... 🤯

```
# Create some objects
```

```
x <- "A string."
```

```
y <- seq(1, 100, 2)
```

```
z <- data.frame(a = rnorm(10),  
                y = 1:10)
```

```
# Character
```

```
# Vector
```

```
# Data frame
```

```
# Create a list
```

```
my_list <- list(x, y, z)
```



```
# Create some objects
```

```
x <- "A string."
```

```
# Character
```

```
y <- seq(1, 100, 2)
```

```
# Vector
```

```
z <- data.frame(a = rnorm(10),  
                y = 1:10)
```

```
# Data frame
```

```
> my_list
```

```
[[1]]
```

```
[1] "A string."
```

```
[[2]]
```

```
[1] 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
```

```
[29] 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89
```

```
[[3]]
```

```
      rnorm.100. X1.100
```

```
1      0.455326375      1
```

```
2      0.692059671      2
```

```
3      0.326601040      3
```

```
4      0.840223512      4
```

```
5     -0.069551338      5
```

Subsetting

Subsetting

- Subsetting is how we **extract** things from objects.

Subsetting

- Subsetting is how we **extract** things from objects.
- For example, you have a vector of 4 integers, and you want the 3rd item.



Subsetting

- Subsetting is how we **extract** things from objects.
- For example, you have a vector of 4 integers, and you want the 3rd item.



- Or you want to extract particular rows or columns from a `data.frame`.

Subsetting

- Subsetting is how we **extract** things from objects.
- For example, you have a vector of 4 integers, and you want the 3rd item.



- Or you want to extract particular rows or columns from a `data.frame`.
- You will use this **all the time**.

Subsetting

- Subsetting is how we **extract** things from objects.
- For example, you have a vector of 4 integers, and you want the 3rd item.



- Or you want to extract particular rows or columns from a `data.frame`.
- You will use this **all the time**.
- Helpful to combine with `str`.
 - i) Check object structure with `str`;
 - ii) Select required elements with subsetting.

Subsetting vectors

We can select elements by position with square brackets (e.g. `x[4]`).

```
x <- c("A", "B", "C", "D", "E", "F")
```

```
x[1]           # 1st element
```

```
x[3]           # 3rd element
```

```
x[length(x)]  # Last element
```


Subsetting vectors

We can select elements by position with square brackets (e.g. `x[4]`).

```
x <- c("A", "B", "C", "D", "E", "F")
```

```
x[1]           # 1st element
```

```
x[3]           # 3rd element
```

```
x[length(x)]  # Last element
```

We can use a vector to select **multiple items**.

```
x[c(1, 2)]     # 1st & 2nd element
```

```
x[1:5]         # First 5 elements
```

```
x[seq(1, 6, 2)] # Odd elements
```

We can omit items by negating the index.

```
x[-1]           # All except 1st  
x[-(1:3)]       # 4th to 6th only  
x[-c(5, 6)]     # All except 5/6th
```

Subsetting matrices

This is the same, but now we have **two dimensions**.

```
mat <- matrix(1:20, ncol = 5)
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
```

We can select single elements by **row** and **column** position: `mat[row, col]`

```
mat[1, 1]      # Row 1, column 1.
mat[1, 2]      # Row 1, column 2.
```

As before, we can select **multiple** elements and **omit** elements with negation.

```
mat[1, c(1, 3)] # Row 1, columns 1 & 3.  
mat[1:3, 2:4]   # Rows 1-3, columns 2-4.  
mat[-1, -1]     # Omit first row/column.
```

If we omit an index, that entire row/column is returned.

```
mat[1, ]         # Row 1, all columns.  
mat[, 2:3]       # All rows, columns 2-3.
```

Subsetting data frames

We're using the [starwars](#) dataset from tidyverse.

```
> data(starwars)
```

```
> head(starwars)
```

	name	height	mass	hair_color	skin_color	eye_color
Luke Skywalker		172	77	blond	fair	
	C-3PO	167	75	<NA>	gold	yellow
	R2-D2	96	32	<NA>	white, blue	
	Darth Vader	202	136	none	white	yellow

birth_year	gender	homeworld	species
19.0	male	Tatooine	Human
112.0	<NA>	Tatooine	Droid
33.0	<NA>	Naboo	Droid
41.9	male	Tatooine	Human

```
# We can select columns by name:
```

```
starwars$height
```

```
[1] 172 167  96 202 150 178 165  97 183 182 180 172 178 179 175 170 173  
[17] 170 180  66 170 183 200 190 177 175 180 150 178 172 176 170 178 179
```

```
# We can select columns by name:
```

```
starwars$height
```

```
[1] 172 167  96 202 150 178 165  97 183 182 188  
[17] 170 180  66 170 183 200 190 177 175 180 150
```

```
# Or, alternatively:
```

```
starwars[["height"]]
```

```
starwars[[2]]
```



```
# We can select columns by name:
```

```
starwars$height
```

```
[1] 172 167  96 202 150 178 165  97 183 182 188  
[17] 170 180  66 170 183 200 190 177 175 180 150
```

```
# Or, alternatively:
```

```
starwars[["height"]]
```

```
# Or by position:
```

```
starwars[[2]]
```

```
# We can select multiple columns:
```

```
starwars[, c("height", "species")]
```

```
starwars[, c(1, 2)]
```

Subsetting lists

A list can contain any number of other objects (even other lists). We need some way of referring the **elements** of a list as well as the **contents** of these elements.

```
> x <- list(1:3, "a", 4:6)
> x
[[1]]
[1] 1 2 3

[[2]]
[1] "a"

[[3]]
[1] 4 5 6

> x[1]
[[1]]
[1] 1 2 3

> x[[1]]
[1] 1 2 3
```

This also applies to data frames:

Select the **contents** of the height column:

```
starwars[["height"]]  
starwars$height  
starwars[[2]]
```

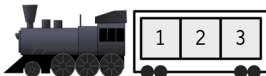
x[[1]]

1	2	3
---	---	---

Select the **entire column**, returning a data frame:

```
starwars["height"]  
starwars[, "height"]  
starwars[2]  
starwars[, 2]
```

x[1]



Subsetting named lists

We can create a list with named elements.

```
> x <- list(first = seq(0, 50, 5),  
>           second = c("A", "string", "vector"),  
>           third  = rnorm(10))  
> x  
$first  
[1] 0 5 10 15 20 25 30 35 40 45 50  
  
$second  
[1] "A" "string" "vector"  
  
$third  
[1] -0.11864667 1.20817170 -0.59353976 1.91463235  
[7] -0.53772881 0.06389441 0.91135953 -0.91389382
```

We can then select items by name:

```
> x$first
```

```
[1]  0  5 10 15 20 25 30 35 40 45 50
```

```
> x["third"]
```

```
[1] -0.11864667  1.20817170 -0.59353976  1.91463235
```

```
[7] -0.53772881  0.06389441  0.91135953 -0.91389382
```

We can then select items by name:

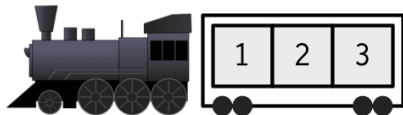
```
> x$first  
[1] 0 5 10 15 20 25 30 35 40 45 50  
  
> x["third"]  
[1] -0.11864667 1.20817170 -0.59353976 1.91463235  
[7] -0.53772881 0.06389441 0.91135953 -0.91389382
```

Again, notice the difference between `[[` and `:`:

```
> x[["third"]]  
[1] -0.11864667 1.208171  
[7] -0.53772881 0.063894
```



```
> x["third"]  
$third  
[1] -0.11864667 1.208171  
[7] -0.53772881 0.063894
```



Materials on subsetting are adapted from Chapter 4 of *Advanced R*, available online at:

<https://adv-r.hadley.nz/subsetting.html>