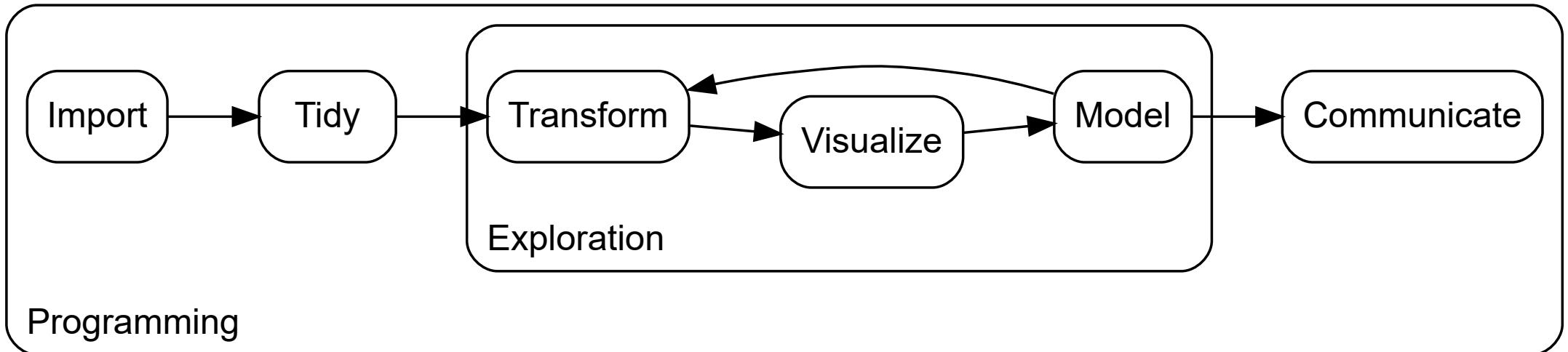


Data Analysis using R

Importing

Sven Werenbeck-Ueding

21.09.2023



Source: Wickham and Grolemund (2016)

Popular file formats

Excel Spreadsheet (.xlsx)

- Format for storing human-readable, rectangular data
- One or more sheets containing rectangular data
- Older spreadsheet file format: .xls

A	B	C	D	E	F	G	H	I	J	K
1 Encuesta Nacional de Empleo/Encuesta Nacional de Ocupación y Empleo (Q3 2003 to Q3 2013)										
2										
3										
4	id	migrate	female	age	municipal	year	quarter	marital_st	empl_stat	educ
5	189889	No	Female	50	2004	2004	Q2	Single	Unemploy	12
6	189889	No	Female	50	2004	2004	Q3	Single	Unemploy	12
7	189889	No	Female	50	2004	2004	Q4	Single	Unemploy	12
8	189889	No	Female	50	2004	2005	Q1	Widowed	Unemploy	12
9	189890	No	Male	26	2004	2005	Q3	Married	Full-time	10
10	189890	No	Male	26	2004	2005	Q4	Married	Full-time	10
11	189890	No	Male	26	2004	2006	Q1	Married	Full-time	10
12	189890	No	Male	26	2004	2006	Q2	Married	Part-time	10
13	189891	No	Male	36	2004	2006	Q4	Married	Part-time	6
14	189891	No	Male	36	2004	2007	Q1	Married	Full-time	6
15	189891	No	Male	36	2004	2007	Q2	Married	Full-time	6
16	189891	No	Male	36	2004	2007	Q3	Married	Part-time	6
17	189894	No	Female	33	2004	2010	Q3	Married	Full-time	9
18	189894	No	Female	33	2004	2010	Q4	Married	Full-time	9
19	189894	No	Female	33	2004	2011	Q1	Married	Part-time	9
20	189894	No	Female	33	2004	2011	Q2	Married	Full-time	9
21	189895	No	Male	32	2004	2011	Q4	Married	Full-time	9
22	189895	No	Male	32	2004	2012	Q1	Married	Full-time	9
23	189895	No	Male	32	2004	2012	Q2	Married	Full-time	9
24	189895	No	Male	32	2004	2012	Q3	Married	Part-time	9

Popular file formats

Comma Separated Values (.csv)

- Format for storing human-readable, rectangular data
- Values are most commonly separated by , or ;
- Each line corresponds to a row
- First line usually contains column names

```
Encuesta Nacional de Empleo/Encuesta Nacional de Ocupación y Empleo (Q3 2003 to Q3 2013,,,,,,,,,,  
,,,,,,,,,,  
id,migrate,female,age,municipality,year,quarter,marital_status,empl_status,educ,income  
189889,No,Female,50,2004,2004,Q2,Single,Unemployed,12,0  
189889,No,Female,50,2004,2004,Q3,Single,Unemployed,12,0  
189889,No,Female,50,2004,2004,Q4,Single,Unemployed,12,0  
189889,No,Female,50,2004,2005,Q1,Widowed,Unemployed,12,0  
189890,No,Male,26,2004,2005,Q3,Married,Full-time,10,NA  
189890,No,Male,26,2004,2005,Q4,Married,Full-time,10,NA  
189890,No,Male,26,2004,2006,Q1,Married,Full-time,10,NA  
189890,No,Male,26,2004,2006,Q2,Married,Part-time,10,NA  
189891,No,Male,36,2004,2006,Q4,Married,Part-time,6,3440  
189891,No,Male,36,2004,2007,Q1,Married,Full-time,6,3440  
189891,No,Male,36,2004,2007,Q2,Married,Full-time,6,3440  
189891,No,Male,36,2004,2007,Q3,Married,Part-time,6,3440  
189894,No,Female,33,2004,2010,Q3,Married,Full-time,9,559  
189894,No,Female,33,2004,2010,Q4,Married,Full-time,9,559  
189894,No,Female,33,2004,2011,Q1,Married,Part-time,9,559  
189894,No,Female,33,2004,2011,Q2,Married,Full-time,9,559  
189895,No,Male,32,2004,2011,Q4,Married,Full-time,9,5590  
189895,No,Male,32,2004,2012,Q1,Married,Full-time,9,5590  
189895,No,Male,32,2004,2012,Q2,Married,Full-time,9,5590  
189895,No,Male,32,2004,2012,Q3,Married,Part-time,9,5590  
189896,No,Male,51,2004,2013,Q1,Widowed,Full-time,6,2580  
189896,No,Male,51,2004,2013,Q2,Single,Full-time,6,2580
```

Example Data

- Mexican survey data used in [Feigenberg \(2020a\)](#) and provided by [Feigenberg \(2020b\)](#)
 - Encuesta Nacional de Ocupación y Empleo (ENOE)
 - Q3 2003 to Q3 2013
 - Quarterly rotating panel: Households included for 5 quarters
 - Records whether any household member leaves to the US
 - Potential migrants are restricted from ages 15 to 65
 - Explanatory variables: age, gender, marital status and education for all household members
- In folder `data/raw/` of the online repository in csv and excel format
- Data was altered to satisfy needs for this course

Importing Data



readxl

The `readxl` package makes it easy to get data out of Excel and into R. [...] `readxl` has no external dependencies, so it's easy to install and use on all operating systems. It is designed to work with tabular data. `readxl` supports both the legacy `.xls` format and the modern xml-based `.xlsx` format.

[Wickham and Bryan \(2022\)](#)



readxl is part of the tidyverse but not part of its core packages!

Load the package to use it:

```
#install.packages("readxl")
library(readxl)
```

read_excel()

```
?read_excel
```

The `read_excel()` function is capable of reading data from `.xls` and `.xlsx` files. If the file format is known, it is recommended to directly use `read_xls()` or `read_xlsx()`, respectively, to prevent R from guessing.

read_excel()

```
?read_excel
```

Function call and arguments:

```
read_excel(  
  path,  
  sheet = NULL,  
  range = NULL,  
  col_names = TRUE,  
  col_types = NULL,  
  na = "",  
  skip = 0,  
  n_max = Inf,  
  ...  
)
```

path

The path to the file you want to read

read_excel()

```
?read_excel
```

Function call and arguments:

```
read_excel(  
  path,  
  sheet = NULL,  
  range = NULL,  
  col_names = TRUE,  
  col_types = NULL,  
  na = "",  
  skip = 0,  
  n_max = Inf,  
  ...  
)
```

sheet

String or integer specifying the sheet name or position to read; defaults to the first sheet

Ignored if the sheet is specified via range

read_excel()

```
?read_excel
```

Function call and arguments:

```
read_excel(  
  path,  
  sheet = NULL,  
  range = NULL,  
  col_names = TRUE,  
  col_types = NULL,  
  na = "",  
  skip = 0,  
  n_max = Inf,  
  ...  
)
```

range

Excel expression for the cell range to read from, e. g. "B3:D87". Can also be used to specify the sheet name like "Budget!B2:G14".

read_excel()

```
?read_excel
```

Function call and arguments:

```
read_excel(  
  path,  
  sheet = NULL,  
  range = NULL,  
  col_names = TRUE,  
  col_types = NULL,  
  na = "",  
  skip = 0,  
  n_max = Inf,  
  ...  
)
```

col_names

- Either TRUE/FALSE or a character vector of column names
- If TRUE, the default, the first row is used as column names
- If FALSE, column names are generated in this fashion: X1, X2, \dots

read_excel()

```
?read_excel
```

Function call and arguments:

```
read_excel(  
  path,  
  sheet = NULL,  
  range = NULL,  
  col_names = TRUE,  
  col_types = NULL,  
  na = "",  
  skip = 0,  
  n_max = Inf,  
  ...  
)
```

col_types

read_excel()

```
?read_excel
```

Function call and arguments:

```
read_excel(  
  path,  
  sheet = NULL,  
  range = NULL,  
  col_names = TRUE,  
  col_types = NULL,  
  na = "",  
  skip = 0,  
  n_max = Inf,  
  ...  
)
```

na

Character vector to interpret as missing values;
defaults to empty cells, i. e. **""**

read_excel()

```
?read_excel
```

Function call and arguments:

```
read_excel(  
  path,  
  sheet = NULL,  
  range = NULL,  
  col_names = TRUE,  
  col_types = NULL,  
  na = "",  
  skip = 0,  
  n_max = Inf,  
  ...  
)
```

skip

Number of rows to skip before reading the data;
defaults to 0

Ignored if range is given

read_excel()

```
?read_excel
```

Function call and arguments:

```
read_excel(  
  path,  
  sheet = NULL,  
  range = NULL,  
  col_names = TRUE,  
  col_types = NULL,  
  na = "",  
  skip = 0,  
  n_max = Inf,  
  ...  
)
```

n_max

Sets the maximum number of rows to read; defaults to **Inf**

Ignored if **range** is given

Let's try this out!

Code Message Output

- Import raw data on migration choice of Mexicans and their socioeconomic characteristics
- Data sets can be found in the data folder of the repository

```
read_excel("data/raw/enoe/enoe.xlsx")
```

Let's try this out!

Code	Message	Output
<pre>## New names: ## • ` ` -> `...2` ## • ` ` -> `...3` ## • ` ` -> `...4` ## • ` ` -> `...5` ## • ` ` -> `...6` ## • ` ` -> `...7` ## • ` ` -> `...8` ## • ` ` -> `...9` ## • ` ` -> `...10` ## • ` ` -> `...11` ## • ` ` -> `...12`</pre>		

Let's try this out!

Code	Message	Output
<pre>## # A tibble: 165,460 × 12 ## Encuesta ...¹ ...2 ...3 ...4 ...5 ...6 ...7 ...8 ...9 ...10 ...11 ...12 ## <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> ## 1 <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> ## 2 <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> ## 3 id migr... age muni... fence year quar... sex mari... empl... educ inco... ## 4 189889 No 50 2004 0 2004 Q3 Fema... Sing... Unem... 12 0 ## 5 189889 No 50 2004 0 2004 Q4 Fema... Sing... Unem... 12 0 ## 6 189889 No 50 2004 0 2005 Q1 Fema... Sing... Unem... 12 0 ## 7 189890 No 26 2004 0 2005 Q4 Male Marr... Full... 10 <NA> ## 8 189890 No 26 2004 0 2006 Q1 Male Marr... Full... 10 <NA> ## 9 189890 No 26 2004 0 2006 Q2 Male N/A Full... 10 <NA> ## 10 189891 No 36 2004 0 2006 Q4 Male Marr... Full... 6 3440 ## # ... with 165,450 more rows, and abbreviated variable name ## # `¹`Encuesta Nacional de Empleo/Encuesta Nacional de Ocupación y Empleo (Q3 2003 to Q3 2013`</pre>		



What happened?

Let's take a look at the excel file:

Encuesta Nacional de Empleo/Encuesta Nacional de Ocupación y Empleo (Q3 2003 to Q3 2013)											
1	id	migrate	female	age	municipali	year	quarter	marital_st	empl_stat	educ	income
2	189889	No	Female	50	2004	2004	Q2	Single	Unemploy	12	0
3	189889	No	Female	50	2004	2004	Q3	Single	Unemploy	12	0
4	189889	No	Female	50	2004	2004	Q4	Single	Unemploy	12	0
5	189889	No	Female	50	2004	2005	Q1	Widowed	Unemploy	12	0
6	189890	No	Male	26	2004	2005	Q3	Married	Full-time	10	
7	189890	No	Male	26	2004	2005	Q4	Married	Full-time	10	
8	189890	No	Male	26	2004	2006	Q1	Married	Full-time	10	
9	189890	No	Male	26	2004	2006	Q2	Married	Part-time	10	
10	189891	No	Male	36	2004	2006	Q4	Married	Part-time	6	3440
11	189891	No	Male	36	2004	2007	Q1	Married	Full-time	6	3440
12	189891	No	Male	36	2004	2007	Q2	Married	Full-time	6	3440
13	189891	No	Male	36	2004	2007	Q3	N/A	Part-time	6	3440
14	189891	No	Male	36	2004	2010	Q3	Married	Full-time	9	559
15	189891	No	Male	36	2004	2010	Q4	Married	Full-time	9	559
16	189894	No	Female	33	2004	2011	Q1	Married	Part-time	9	559
17	189894	No	Female	33	2004	2011	Q2	Married	Full-time	9	559
18	189894	No	Female	33	2004	2011	Q3	Married	Full-time	9	559
19	189894	No	Female	33	2004	2011	Q4	Married	Full-time	9	559
20	189894	No	Female	33	2004	2012	Q1	Married	Part-time	9	559
21	189895	No	Male	32	2004	2011	Q4	Married	Full-time	9	5590
22	189895	No	Male	32	2004	2012	Q1	Married	Full-time	9	5590
23	189895	No	Male	32	2004	2012	Q2	Married	Full-time	9	5590
24	189895	No	Male	32	2004	2012	Q3	Married	Part-time	9	5590
25	189896	No	Male	51	2004	2013	Q1	Widowed	Full-time	6	2580
26	189896	No	Male	51	2004	2013	Q2	Single	Full-time	6	2580
27	189896	No	Male	51	2004	2013	Q3	Single	Full-time	6	2580
28	189897	No	Male	23	2004	2004	Q2	Widowed	Part-time	8	2580
29	189897	No	Male	23	2004	2004	Q3	Single	Part-time	8	2580
30	189897	No	Male	23	2004	2004	Q4	Single	Full-time	8	2580
31	189897	No	Male	23	2004	2005	Q1	Widowed	Full-time	8	2580
32	189898	No	Female	28	2004	2005	Q3	Married	Unemploy	9	0
33	189898	No	Female	28	2004	2005	Q4	Married	Unemploy	9	0

The first few rows contain some information about the spreadsheet itself. This is **very** common with excel files from government authorities!

We can use the `skip` argument to start reading after the third line:

```
read_excel("data/raw/enoe/enoe.xlsx",  
          skip = 3)
```

What happened?

Let's take a look at the excel file:

Encuesta Nacional de Empleo/Encuesta Nacional de Ocupación y Empleo (Q3 2003 to Q3 2013)										
A	B	C	D	E	F	G	H	I	J	K
1	Encuesta Nacional de Empleo/Encuesta Nacional de Ocupación y Empleo (Q3 2003 to Q3 2013)									
2										
3										
4	id	migrate	female	age	municipali	year	quarter	marital_st	empl_stati	educ
5	189889	No	Female	50	2004	2004	Q2	Single	Unemploy	12
6	189889	No	Female	50	2004	2004	Q3	Single	Unemploy	12
7	189889	No	Female	50	2004	2004	Q4	Single	Unemploy	12
8	189889	No	Female	50	2004	2005	Q1	Widowed	Unemploy	12
9	189890	No	Male	26	2004	2005	Q3	Married	Full-time	10
10	189890	No	Male	26	2004	2005	Q4	Married	Full-time	10
11	189890	No	Male	26	2004	2006	Q1	Married	Full-time	10
12	189890	No	Male	26	2004	2006	Q2	Married	Part-time	10
13	189891	No	Male	36	2004	2006	Q4	Married	Part-time	6
14	189891	No	Male	36	2004	2007	Q1	Married	Full-time	6
15	189891	No	Male	36	2004	2007	Q2	Married	Full-time	6
16	189891	No	Male	36	2004	2007	Q3	N/A	Part-time	6
17	189894	No	Female	33	2004	2010	Q3	Married	Full-time	9
18	189894	No	Female	33	2004	2010	Q4	Married	Full-time	9
19	189894	No	Female	33	2004	2011	Q1	Married	Part-time	9
20	189894	No	Female	33	2004	2011	Q2	Married	Full-time	9
21	189895	No	Male	32	2004	2011	Q4	Married	Full-time	9
22	189895	No	Male	32	2004	2012	Q1	Married	Full-time	9
23	189895	No	Male	32	2004	2012	Q2	Married	Full-time	9
24	189895	No	Male	32	2004	2012	Q3	Married	Part-time	9
25	189896	No	Male	51	2004	2013	Q1	Widowed	Full-time	6
26	189896	No	Male	51	2004	2013	Q2	Single	Full-time	6
27	189896	No	Male	51	2004	2013	Q3	Single	Full-time	6
28	189897	No	Male	23	2004	2004	Q2	Widowed	Part-time	8
29	189897	No	Male	23	2004	2004	Q3	Single	Part-time	8
30	189897	No	Male	23	2004	2004	Q4	Single	Full-time	8
31	189897	No	Male	23	2004	2005	Q1	Widowed	Full-time	8
32	189898	No	Female	28	2004	2005	Q3	Married	Unemploy	9
33	189898	No	Female	28	2004	2005	Q4	Married	Unemploy	9

There seems to be some inconsistencies with the naming of missing values...

Use the `na` argument to treat empty cells and N/A as missing values (NA):

```
read_excel("data/raw/enoe/enoe.xlsx",  
          skip = 3,  
          na = c("", "N/A"))
```

Another try...

Code Message Output

```
read_excel("data/raw/enoe/enoe.xlsx",
           skip = 3, # Skip the first three lines
           na = c("", "N/A")) # Interpret empty cells and cells containing "N/A" as missing
```

Another try...

Code Message Output

Another try...

Code	Message	Output
<pre>## # A tibble: 165,457 × 12 ## id migrate age municipality¹ fence year quarter sex marital_status² empl_status³ educ ## <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> ## 1 189889 No 50 2004 0 2004 Q3 Fema... Single Unempl... 12 ## 2 189889 No 50 2004 0 2004 Q4 Fema... Single Unempl... 12 ## 3 189889 No 50 2004 0 2005 Q1 Fema... Single Unempl... 12 ## 4 189890 No 26 2004 0 2005 Q4 Male Married Full-t... 10 ## 5 189890 No 26 2004 0 2006 Q1 Male Married Full-t... 10 ## 6 189890 No 26 2004 0 2006 Q2 Male <NA> Full-t... 10 ## 7 189891 No 36 2004 0 2006 Q4 Male Married Full-t... 6 ## 8 189891 No 36 2004 0 2007 Q1 Male Married Full-t... 6 ## 9 189891 No 36 2004 0 2007 Q2 Male Married Full-t... 6 ## 10 189894 No 33 2004 1 2010 Q3 <NA> Married Full-t... 9 ## # ... with 165,447 more rows, 1 more variable: income <chr>, and abbreviated ## # variable names ¹municipality, ²marital_status, ³empl_status</pre>		



readr

The goal of `readr` is to provide a fast and friendly way to read rectangular data from delimited files, such as comma-separated values (CSV) and tab-separated values (TSV). It is designed to parse many types of data found in the wild, while providing an informative problem report when parsing leads to unexpected results.

Wickham, Hester, and Bryan (2022)

read_csv()

```
?read_csv
```

read_csv() function is a special case of **read_delim()** with the **separator set to ,** used for reading files with comma separated values. The related **read_csv2()**, in contrast, **assumes a ; as separator** for values and interprets **,** as decimal point (which is common in some European countries).

- Considerably faster than the base R solution `read.csv()`
- Consistent parameter naming (`col_names` and `col_type` instead of `header` and `colClasses`)
- Automatically parses common date formats, but leaves strings unaltered
- Progression bar for big data sets

read_csv()

```
?read_csv
```

Function call and arguments:

```
read_csv(  
  file,  
  col_names = TRUE,  
  col_types = NULL,  
  col_select = NULL,  
  na = c("", "NA"),  
  skip = 0,  
  n_max = Inf,  
  ...  
)
```

file

The path to the file you want to read

read_csv()

```
?read_csv
```

Function call and arguments:

```
read_csv(  
  file,  
  col_names = TRUE,  
  col_types = NULL,  
  col_select = NULL,  
  na = c("", "NA"),  
  skip = 0,  
  n_max = Inf,  
  ...  
)
```

col_names

- Either TRUE/FALSE or a character vector of column names
- If TRUE, the default, the first row is used as column names
- If FALSE, column names are generated in this fashion: X1, X2, \dots

read_csv()

```
?read_csv
```

Function call and arguments:

```
read_csv(  
  file,  
  col_names = TRUE,  
  col_types = NULL,  
  col_select = NULL,  
  na = c("", "NA"),  
  skip = 0,  
  n_max = Inf,  
  ...  
)
```

col_types

read_csv()

```
?read_csv
```

Function call and arguments:

```
read_csv(  
  file,  
  col_names = TRUE,  
  col_types = NULL,  
  col_select = NULL,  
  na = c("", "NA"),  
  skip = 0,  
  n_max = Inf,  
  ...  
)
```

col_select

Character vector containing columns to include in the resulting data set; defaults to NULL

read_csv()

```
?read_csv
```

Function call and arguments:

```
read_csv(  
  file,  
  col_names = TRUE,  
  col_types = NULL,  
  col_select = NULL,  
  na = c("", "NA"),  
  skip = 0,  
  n_max = Inf,  
  ...  
)
```

na

Character vector to interpret as missing values;
defaults to c("", "NA")

read_csv()

```
?read_csv
```

Function call and arguments:

```
read_csv(  
  file,  
  col_names = TRUE,  
  col_types = NULL,  
  col_select = NULL,  
  na = c("", "NA"),  
  skip = 0,  
  n_max = Inf,  
  ...  
)
```

skip

Number of lines to skip before reading the data;
defaults to 0

read_csv()

```
?read_csv
```

Function call and arguments:

```
read_csv(  
  file,  
  col_names = TRUE,  
  col_types = NULL,  
  col_select = NULL,  
  na = c("", "NA"),  
  skip = 0,  
  n_max = Inf,  
  ...  
)
```

n_max

Sets the maximum number of lines to read; defaults to Inf

Import the ENE/ENOE Data Set from CSV

Code Message Output

```
read_csv("data/raw/enoe/enoe.csv")
```

Import the ENE/ENOE Data Set from CSV

Code	Message	Output
<pre>## New names: ## • ` ` -> `...2` ## • ` ` -> `...3` ## • ` ` -> `...4` ## • ` ` -> `...5` ## • ` ` -> `...6` ## • ` ` -> `...7` ## • ` ` -> `...8` ## • ` ` -> `...9` ## • ` ` -> `...10` ## • ` ` -> `...11` ## • ` ` -> `...12`</pre> <pre>## Warning: One or more parsing issues, call `problems()` on your data frame for details, ## e.g.: ## dat <- vroom(...) ## problems(dat)</pre>		

Import the ENE/ENOE Data Set from CSV

Code	Message	Output
<pre>## Warning: One or more parsing issues, call `problems()` on your data frame for details, ## e.g.: ## dat <- vroom(...) ## problems(dat) ## # A tibble: 165,460 × 12 ## Encuesta ...¹ ...2 ...3 ...4 ...5 ...6 ...7 ...8 ...9 ...10 ...11 ...12 ## <dbl> <chr> ## 1 NA <NA> ## 2 NA <NA> ## 3 NA migr... age muni... fence year quar... sex mari... empl... educ inco... ## 4 189889 No 50 2004 0 2004 Q3 Fema... Sing... Unem... 12 0 ## 5 189889 No 50 2004 0 2004 Q4 Fema... Sing... Unem... 12 0 ## 6 189889 No 50 2004 0 2005 Q1 Fema... Sing... Unem... 12 0 ## 7 189890 No 26 2004 0 2005 Q4 Male Marr... Full... 10 <NA> ## 8 189890 No 26 2004 0 2006 Q1 Male Marr... Full... 10 <NA> ## 9 189890 No 26 2004 0 2006 Q2 Male N/A Full... 10 <NA></pre>		



Similar problems as before!

- Skip first three lines
- Set first line (after skipping three lines) as column names
- Interpret N/A as missing

Import ENE/ENOE Data with Function Arguments

Code	Message	Output
------	---------	--------

```
read_csv("data/raw/enoe/enoe.csv",
         skip = 3,
         col_names = TRUE, # The default
         na = c("", "N/A"))
```

Import ENE/ENOE Data with Function Arguments

Code	Message	Output
<pre>## Rows: 165457 Columns: 12 ## — Column specification — ## Delimiter: "," ## chr (11): migrate, age, municipality, fence, year, quarter, sex, marital_sta... ## dbl (1): id ## ## i Use `spec()` to retrieve the full column specification for this data. ## i Specify the column types or set `show_col_types = FALSE` to quiet this message.</pre>		

Import ENE/ENOE Data with Function Arguments

Code	Message	Output
<pre>## # A tibble: 165,457 × 12 ## id migrate age municipi...¹ fence year quarter sex marit...² empl_...³ educ ## <dbl> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> ## 1 189889 No 50 2004 0 2004 Q3 Fema... Single Unempl... 12 ## 2 189889 No 50 2004 0 2004 Q4 Fema... Single Unempl... 12 ## 3 189889 No 50 2004 0 2005 Q1 Fema... Single Unempl... 12 ## 4 189890 No 26 2004 0 2005 Q4 Male Married Full-t... 10 ## 5 189890 No 26 2004 0 2006 Q1 Male Married Full-t... 10 ## 6 189890 No 26 2004 0 2006 Q2 Male <NA> Full-t... 10 ## 7 189891 No 36 2004 0 2006 Q4 Male Married Full-t... 6 ## 8 189891 No 36 2004 0 2007 Q1 Male Married Full-t... 6 ## 9 189891 No 36 2004 0 2007 Q2 Male Married Full-t... 6 ## 10 189894 No 33 2004 1 2010 Q3 NA Married Full-t... 9 ## # ... with 165,447 more rows, 1 more variable: income <chr>, and abbreviated ## # variable names ¹municipality, ²marital_status, ³empl_status</pre>		

Data Types

Vectors in R

- Vectors can be distinguished in atomic vectors and lists
 - Atomic vector: All elements must have the same type
 - Lists: Elements can have different types
- If you are already familiar with R, you probably have encountered vectors on several occasions, e. g. the sequence `1:10` is an atomic vector containing all integers between 1 and 10

```
1:10
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

```
typeof(1:10) # Check the type of the integer vector 1:10
```

```
## [1] "integer"
```

Primary Types of Atomic Vectors

Type	Description	How to check
Logical	TRUE/FALSE or their abbreviations T/F. Simple calculations can be conducted on vectors of this type, e. g. <code>sum(T, T, F, T)</code> gives 3 because TRUE is interpreted as 1 and FALSE as 0.	<code>is.logical()</code>
Character	Strings surrounded by " or '	<code>is.character()</code>
Double	Numerical values with decimals. Special values are Inf, -Inf and NaN.	<code>is.double()</code>
Integer	Numerical values that cannot contain fractional values. Must be followed by L	<code>is.integer()</code>



Both, Integers and doubles, are numerical values!

`is.numeric()` returns TRUE for values that are either integers *or* doubles.

Factors: Representation of Categorical Data

- Factors contain predefined values only
- Integer vector with attributes `class` ("factor") and `levels` (set of values)

```
# Create a factor vector with two levels c("Employed", "Unemployed")
fct <- factor(c("Employed", "Employed", "Unemployed", "Employed"))

fct
```

```
## [1] Employed    Employed    Unemployed Employed
## Levels: Employed Unemployed
```

```
# Take a look at the vector
str(fct)
```

```
##  Factor w/ 2 levels "Employed","Unemployed": 1 1 2 1
```

Factors: Representation of Categorical Data

```
# Check the vector type
typeof(fct)
```

```
## [1] "integer"
```

```
# Get the vector attributes
attributes(fct)
```

```
## $levels
## [1] "Employed"    "Unemployed"
##
## $class
## [1] "factor"
```

```
# To get the levels of a vector, use the short-hand levels() function instead
levels(fct)
```

```
## [1] "Employed"    "Unemployed"
```

Lists

```
# Construct a list
our_list <- list(1:10, # Integer sequence
                  seq(0, 1, by = 0.25), # Double sequence
                  c("This", "is", "a", "character", "vector"), # Character vector
                  factor(c("Employed", "Employed", "Unemployed", "Employed"))) # Factor vector

typeof(our_list)

## [1] "list"

# Take a look at the list structure
str(our_list)

## List of 4
## $ : int [1:10] 1 2 3 4 5 6 7 8 9 10
## $ : num [1:5] 0 0.25 0.5 0.75 1
## $ : chr [1:5] "This" "is" "a" "character" ...
## $ : Factor w/ 2 levels "Employed","Unemployed": 1 1 2 1
```

Lists



Lists may contain more complex objects than atomic vectors, such as lists.

```
list(1:10,  
  seq(0, 1, by = 0.25),  
  list("a", 1:2))
```

```
## [[1]]  
## [1] 1 2 3 4 5 6 7 8 9 10  
##  
## [[2]]  
## [1] 0.00 0.25 0.50 0.75 1.00  
##  
## [[3]]  
## [[3]][[1]]  
## [1] "a"  
##  
## [[3]][[2]]  
## [1] 1 2
```

Data Frames

- Data sets are usually represented as `data.frame` objects in R
- A `data.frame` is essentially a named list of vectors with equal length

```
# Construct a data frame
df <- data.frame(income = c(0, 500, 3000),
                  empl_status = factor(c("Unemployed", "Employed", "Employed")))
```

Data Frames

```
# Check the type
typeof(df)

## [1] "list"

# Get the attributes
attributes(df)

## $names
## [1] "income"      "empl_status"
##
## $class
## [1] "data.frame"
##
## $row.names
## [1] 1 2 3
```



tibble

A `tibble`, or `tbl_df`, is a modern reimagining of the `data.frame`, keeping what time has proven to be effective, and throwing out what is not. Tibbles are `data.frames` that are lazy and surly: they do less (i.e. they don't change variable names or types, and don't do partial matching) and complain more (e.g. when a variable does not exist). This forces you to confront problems earlier, typically leading to cleaner, more expressive code. Tibbles also have an enhanced `print()` method which makes them easier to use with large datasets containing complex objects.

Müller and Wickham (2022)

Data Frames vs. Tibbles

Tibbles' enhanced `print()` function shows only the first 10 rows and displays information on the data structure:

```
# Construct a data frame
df <- data.frame(
  income = c(0, 500, 3000),
  empl_status = factor(
    c("Unemployed", "Employed", "Employed")
  )
)
```

```
# Print the data frame
df
```

```
##   income empl_status
## 1      0  Unemployed
## 2    500   Employed
## 3   3000   Employed
```

```
# Construct a tibble
tbl <- tibble(
  income = c(0, 500, 3000),
  empl_status = factor(
    c("Unemployed", "Employed", "Employed")
  )
)
```

```
# Print the tibble
tbl
```

```
## # A tibble: 3 × 2
##   income empl_status
##     <dbl> <fct>
## 1      0 Unemployed
## 2    500 Employed
## 3   3000 Employed
```

Data Frames vs. Tibbles

When subsetting, tibble gives a warning if the column does not exist:

```
df$gender
```

```
## NULL
```

```
tbl$gender
```

```
## Warning: Unknown or uninitialized column: `gender`  
## NULL
```

Importing Data as Tibble

`read_excel()` and `read_csv()` automatically create a `tibble` object:

```
data <- read_csv("data/raw/enoe/enoe.csv", skip = 3, col_names = TRUE, na = c("", "N/A"))

typeof(data)
```

```
## Rows: 165457 Columns: 12
## — Column specification ——————
## Delimiter: ","
## chr (11): migrate, age, municipality, fence, year, quarter, sex, marital_sta...
## dbl (1): id
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## [1] "spec_tbl_df" "tbl_df"        "tbl"           "data.frame"
```



The class of a `tibble` also contains "data.frame". Hence, functions applied to `data.frame` objects can also be applied to `tibbles`.

Further Literature

- Further information on tibbles can be found in chapter 10 of [Wickham and Grolemund \(2016\)](#) and the package documentation (see [Müller and Wickham \(2022\)](#)).
- Chapter 3 of [Wickham \(2019\)](#) discusses the technical properties of vectors in R more deeply.

Exporting Data

write_csv()

```
?write_csv
```

`write_csv()` function is a special case of `write_delim()` with the **separator set to ,**. Again, there is a related `write_csv2()` which sets the separator to ;.

- Considerably faster than the base R solution `write.csv()`
- Does not include row names as a column in the written file
- Progression bar for big data sets

write_csv()

```
?write_csv
```

Function call and arguments:

```
write_csv(  
  x,  
  file,  
  na = "NA",  
  append = FALSE,  
  col_names = !append,  
  ...  
)
```

x

A `data.frame` or `tibble` to write to `.csv`

write_csv()

```
?write_csv
```

Function call and arguments:

```
write_csv(  
  x,  
  file,  
  na = "NA",  
  append = FALSE,  
  col_names = !append,  
  ...  
)
```

file

File (path) to write to

write_csv()

```
?write_csv
```

Function call and arguments:

```
write_csv(  
  x,  
  file,  
  na = "NA",  
  append = FALSE,  
  col_names = !append,  
  ...  
)
```

na

String used for missing values; defaults to "NA"

write_csv()

```
?write_csv
```

Function call and arguments:

```
write_csv(  
  x,  
  file,  
  na = "NA",  
  append = FALSE,  
  col_names = !append,  
  ...  
)
```

append

If FALSE, the default, the existing will be overwritten.
If TRUE, it will be appended to the existing file.

write_csv()

```
?write_csv
```

Function call and arguments:

```
write_csv(  
  x,  
  file,  
  na = "NA",  
  append = FALSE,  
  col_names = !append,  
  ...  
)
```

col_names

- If FALSE, column names will not be included at the top of the file.
- If TRUE, column names will be included.
- The default is to take the opposite value given to argument append

Export the ENE/ENOE Data Set to CSV

Code Output

```
data <- read_csv("data/raw/enoe/enoe.csv", skip = 3, col_names = TRUE, na = c("", "N/A"))

write_csv(data, "data/raw/enoe.csv")
```

Export the ENE/ENOE Data Set to CSV

Code Output

The exported data is stored in csv format in the raw folder of the data folder:

```
list.files("data/raw/")

## [1] "enoe"           "enoe.csv"        "fence_construction"
```

References

- Feigenberg, B. (2020a). "Fenced Out: The Impact of Border Construction on US-Mexico Migration". In: *American Economic Journal: Applied Economics* 12.3, pp. 106-39. DOI: [10.1257/app.20170231](https://doi.org/10.1257/app.20170231). URL: <https://www.aeaweb.org/articles?id=10.1257/app.20170231>.
- Feigenberg, B. (2020b). *Replication package for: Fenced Out: The Impact of Border Construction on US-Mexico Migration*. <https://www.aeaweb.org/journals/dataset?id=10.1257/app.20170231>, Last accessed on 2022-12-29.
- Müller, K. and H. Wickham (2022). *tibble: Simple Data Frames*. <https://tibble.tidyverse.org/>, <https://github.com/tidyverse/tibble>.
- Wickham, H. (2019). *Advanced R*. 2nd. Chapman & Hall/CRC. URL: <http://adv-r.had.co.nz/>.
- Wickham, H. and J. Bryan (2022). *readxl: Read Excel Files*. <https://readxl.tidyverse.org>, <https://github.com/tidyverse/readxl>.
- Wickham, H. and G. Grolemund (2016). *R for data science. import, tidy, transform, visualize, and model data*. O'Reilly. URL: <https://r4ds.had.co.nz/>.
- Wickham, H., J. Hester, and J. Bryan (2022). *readr: Read Rectangular Text Data*. <https://readr.tidyverse.org>, <https://github.com/tidyverse/readr>.