Data management

Importing Data

Matilde Trevisani

Entering and cleaning data

There are four basic steps as you prepare to analyze data in R:

1. Identify **where** the data is (If it is on your computer, which directory? If it is online, what is the url?)
2. **Read** data into R (`read.table`, `read.csv`, or `readr` package) using the file path you figured out in step 1
3. **Check** to make sure the data came in correctly (`dim`, `head`, `tail`, `str`)
4. **Clean** the data up

Directories and pathnames

Working directory

You can figure out which directory R is working in by running the command `getwd()`.

```
getwd()
```

```
## [1] "C:/Users/39349/Documents/teaching_2023/ADPI/slides/datamanaging"
```

You can set a new working directory using the `setwd()` function and a **pathname**. For example,

```
setwd("~")
getwd()
```

```
## [1] "C:/Users/39349/Documents"
```

Finally, you can use R Projects from within RStudio. The working directory is the project directory (the directory in which the `.Rproj` file is stored).

File and directory pathnames

Pathnames are the directions for getting to a directory or file stored on your computer.

You can use one of two types of pathnames:

- *Relative pathname*: How to get to the file or directory from your current working directory
- *Absolute pathname*: How to get to the file or directory from anywhere on the computer

> **Forward slashes and backslashes**
>
> Remember to use forward slashes / to separate directories in the path. If you use back slashes \, you will get an error as these are used to escape the following character (e.g., \t is a tab). If you really want back slashes, use two of them \ as the first will escape the second. If you let autocomplete fill in your path, it will do it correctly

Absolute pathnames give the full directions starting all the way at the root directory. For example, the `daily_show_guests.csv` file in the `data` directory has the absolute pathname:

```
"C:\Users\39349\Documents\teaching_2023\ADPI\slides\datamanaging\data"
```

You can use this absolute pathname to read this file in using `read.csv`.

```
file_path <- "C:\\Users\\39349\Documents\\teaching_2023\\ADPI\\slides\datamanaging\\data"
daily_show <- read.csv(file_path, ...)
```

The **relative pathname** depends on your current working directory.

As an example of a relative pathname, say you're working in the directory `datamanaging` and you want to read in the `daily_show_guests.csv` file in the `data` subdirectory. Therefore, the relative pathname would be:

`"data/daily_show_guests.csv"`

You can use this relative pathname to tell R where to find and read in the file:

```
daily_show <- read.csv("data/daily_show_guests.csv", ...)
```

There are some abbreviations for pathnames:

Shorthand	Meaning
`~`	Home directory
`.`	Current working directory
`..`	One directory up from current working directory
`../..`	Two directories up from current working directory

If you are getting errors reading in files, it is often helpful to use `list.files()` to make sure the file in question is in the directory that the relative pathname you are using is directing R to.

```
list.files()
```

Diversion: `paste`

The `paste()` function takes, as inputs, a series of different character strings and pastes them together in a single character string. For example:

```
paste("Sunday", "Monday", "Tuesday")
```

```
## [1] "Sunday Monday Tuesday"
```

The `paste()` function has an option called `sep =` that indicates what is the separator. The default is a space. E.g., if you wanted to paste without spaces, you could use `sep = ""`:

```
paste("Sunday", "Monday", "Tuesday", sep = "")
```

```
## [1] "SundayMondayTuesday"
```

As a shortcut, you could achieve the same thing using the `paste0` function.

```
paste0("Sunday", "Monday", "Tuesday")
```

Reading data into R

Some of the types of data files that R can read in:

- Flat (or plain text) files (files that you can open using a text editor)
- Files from other statistical packages (SAS, Excel, Stata, SPSS)
- Tables on webpages (e.g., the table on ebola outbreaks near the end of this Wikipedia page)
- Data in a database (e.g., MySQL, Oracle)
- Data in JSON and XML formats
- Geographic shapefiles
- Data through APIs [Application Programming Interfaces] (e.g., GoogleMaps, Twitter, many government agencies)

Reading tabular (rectangular) data into R

Tabular data and file format

Tabular data are data that is organized in the form of a **table** with rows and columns. A table often has a **header**, i.e. an additional row that displays variable names.

Tabular data may be stored in files using various formats, spreadsheets, etc.

The most common spreadsheets store data in their own, proprietary file format, e.g. **MS Excel** which produces .xls and .xlsx files. Such formats may be a limitation to data management in R.

Simpler formats such as **plain text files** with .txt or .csv should always be preferred when saving or exporting data from spreadsheets.

Reading local flat files

Flat files basically are files that you can open using a text editor. Most flat files come in two general categories:

1. Fixed width files (fwf)
2. Delimited files
    - Comma-separated values: ".csv"
    - Tab-separated values: ".tab", ".tsv"
    - Other possible delimiters: space, colon, semicolon, pipe ("|")

For example,

```
Course            Number          Day         Time
Intro to Epi      501             M/W/F       9:00-9:50
Advanced Epi      521             T/Th        1:00-2:15
```

```
Course, Number, Day, Time
"Intro to Epi", 501, "M/W/F", "9:00-9:50"
"Advanced Epi", 521, "T/Th", "1:00-2:15"
```

The `read.table` family of functions are part of base R.

If the file is delimited, you can use the `read.table` family of functions. This family of functions includes several specialized functions.

Function	Separator	Decimal point
`read.table`	white space	period
`read.csv`	comma	period
`read.csv2`	semi-colon	comma
`read.delim`	tab	period
`read.delim2`	tab	comma

Some of the interesting parameters with the `read.table` family of functions are:

Option	Description
`sep`	What is the delimiter in the data?
`skip`	How many lines of the start of the file should you skip?
`header`	Does the first line you read give column names?
`as.is`	Should you bring in strings as characters, not factors?
`nrows`	How many rows do you want to read in?
`na.strings`	How are missing values coded?

But, we will use equivalent functions from packages that belong to the *tidyverse* collection

--->

The "tidyverse"

The `readr` package is a member of the **tidyverse** (https://www.tidyverse.org/) of packages.

Most were developed in part or full by Hadley Wickham and others at RStudio.

You can use the `tidyverse` package to download all tidyverse packages at one.

The `read_*` functions

The `read.table` family of functions are part of base R. There is a newer package called `readr` that has a family of `read_*` functions. These functions are very similar, but have some more sensible defaults.

- Work better with large datasets: faster, includes progress bar
- Have more sensible defaults (e.g., characters default to characters, not factors)

Functions in the `read_*` family include:

- `read_csv`, `read_tsv` (specific delimiters)
- `read_delim`, `read_table` (generic)
- `read_fwf`
- `read_log`
- `read_lines`


readr
www.rstudio.com


readxl
www.rstudio.com

readr

Read a flat file into a tibble by `readr` 2.1.5

- `read_table()` - whitespace delimited files
- `read_csv()` - comma delimited files
- `read_csv2()` - semicolon separated files (common in countries where "," is used as the decimal place)
- `read_tsv()` - tab delimited files
- `read_delim()` - reads in files with any delimiter
- `read_fwf()` - fixed width files
- `read_lines` - read some lines

readxl

Read an excel file into a tibble

- `read_excel()` - read xls or xlsx files
- ...

Some tips

It is recommended to systematically inspect data file before importing in R. One way to do it is to open the text file in a text editor (for example RStudio), or to read some of the file into R with `read_lines('file.txt')` (the `n_max` argument is useful if there is lots of data) and determine:

1. which symbol is used as delimiter ("," or ";")
2. which symbol is used as decimal separator ("." , the default, or ",")
3. any extra lines of data that need removing

in order to specify the above inputs, if needed,

1. `delim=`
2. `locale = locale(decimal_mark = ",")`
3. `skip=` (number of lines to skip before reading data.) or `n_max=` (Maximum number of lines to read)

Use `spec()` and `problems()` to verify column specifications and any reading problem.

Reading data

```
nobel <- read_csv(file = "data/nobel.csv")
nobel
```

```
## # A tibble: 935 × 26
##      id firstname      surname    year category affiliation city
##   <dbl> <chr>          <chr>     <dbl> <chr>    <chr>       <chr>
## 1     1 Wilhelm Conrad Röntgen    1901 Physics  Munich Uni… Muni…
## 2     2 Hendrik A.     Lorentz    1902 Physics  Leiden Uni… Leid…
## 3     3 Pieter         Zeeman     1902 Physics  Amsterdam … Amst…
## 4     4 Henri          Becquerel  1903 Physics  École Poly… Paris
## 5     5 Pierre         Curie      1903 Physics  École muni… Paris
## 6     6 Marie          Curie      1903 Physics  <NA>        <NA>
## # ℹ 929 more rows
## # ℹ 19 more variables: country <chr>, born_date <date>,
## #   died_date <date>, gender <chr>, born_city <chr>,
## #   born_country <chr>, born_country_code <chr>,
## #   died_city <chr>, died_country <chr>,
## #   died_country_code <chr>, overall_motivation <chr>,
## #   share <dbl>, motivation <chr>, …
```

Writing data

- Write a file

```
df <- data.frame(
  x = 1:3,
  y = letters[1:3]
)

write_csv(df, file = "data/df.csv")
```

or

```
df <- tribble(
  ~x, ~y,
  1,  "a",
  2,  "b",
  3,  "c"
)

write_csv(df, file = "data/df_2.csv")
```

- Read it back in to inspect

```
read_csv("data/df.csv")
```

```
## # A tibble: 3 × 2
##       x y
##   <dbl> <chr>
## 1     1 a
## 2     2 b
## 3     3 c
```

```
read_csv("data/df_2.csv")
```

```
## # A tibble: 3 × 2
##       x y
##   <dbl> <chr>
## 1     1 a
## 2     2 b
## 3     3 c
```

Variable names

Data with bad names

```
edibnb_badnames <- read_csv("data/edibnb-badnames.csv")
names(edibnb_badnames)
```

```
##  [1] "ID"                  "Price"
##  [3] "neighbourhood"       "accommodates"
##  [5] "Number of bathrooms" "Number of Bedrooms"
##  [7] "n beds"              "Review Scores Rating"
##  [9] "Number of reviews"   "listing_url"
```

... but R doesn't allow spaces in variable names

E.g.,

```
ggplot(edibnb_badnames, aes(x = Number of bathrooms, y = Price)) +
  geom_point()
```

R prints error!

Option 1 - Define column names

```
edibnb_col_names <- read_csv("data/edibnb-badnames.csv",
                             col_names = c("id", "price",
                                           "neighbourhood", "accommodates",
                                           "bathroom", "bedroom",
                                           "bed", "review_scores_rating",
                                           "n_reviews", "url"))

names(edibnb_col_names)
```

```
##  [1] "id"                  "price"
##  [3] "neighbourhood"       "accommodates"
##  [5] "bathroom"            "bedroom"
##  [7] "bed"                 "review_scores_rating"
##  [9] "n_reviews"           "url"
```

Option 2 - Format text to snake_case

```
edibnb_clean_names <- read_csv("data/edibnb-badnames.csv") %>%
  janitor::clean_names()

names(edibnb_clean_names)
```

```
##  [1] "id"                  "price"
##  [3] "neighbourhood"       "accommodates"
##  [5] "number_of_bathrooms" "number_of_bedrooms"
##  [7] "n_beds"              "review_scores_rating"
##  [9] "number_of_reviews"   "listing_url"
```

> **Snake case** is a naming convention that all the words are in lowercase and split by an underscore _ with no spaces in between.

Variable types

Which type is x? Why?

x	y	z
1	a	hi
NA	b	hello
3	Not applicable	9999
4	d	ola
5	e	hola
.	f	whatup
7	g	wassup
8	h	sup
9	i	

```
read_csv("data/df-na.csv")
```

```
## # A tibble: 9 × 3
##   x     y              z
##   <chr> <chr>          <chr>
## 1 1     a              hi
## 2 <NA>  b              hello
## 3 3     Not applicable 9999
## 4 4     d              ola
## 5 5     e              hola
## 6 .     f              whatup
## 7 7     g              wassup
## 8 8     h              sup
## 9 9     i              <NA>
```

Option 1. Explicit NAs

```
read_csv("data/df-na.csv",
         na = c("", "NA", ".", "9999", "Not applicable"))
```

x	y	z
1	a	hi
NA	b	hello
3	Not applicable	9999
4	d	ola
5	e	hola
.	f	whatup
7	g	wassup
8	h	sup
9	i	

```
## # A tibble: 9 × 3
##       x y     z
##   <dbl> <chr> <chr>
## 1     1 a     hi
## 2    NA b     hello
## 3     3 <NA>  <NA>
## 4     4 d     ola
## 5     5 e     hola
## 6    NA f     whatup
## 7     7 g     wassup
## 8     8 h     sup
## 9     9 i     <NA>
```

Option 2. Specify column types

```
read_csv("data/df-na.csv", col_types = list(col_double(),
                                            col_character(),
                                            col_character()))
```

```
## Warning: One or more parsing issues, call `problems()` on your data frame
## for details, e.g.:
##   dat <- vroom(...)
##   problems(dat)
```

```
## # A tibble: 9 × 3
##       x y             z
##   <dbl> <chr>         <chr>
## 1     1 a             hi
## 2    NA b             hello
## 3     3 Not applicable 9999
## 4     4 d             ola
## 5     5 e             hola
## 6    NA f             whatup
## 7     7 g             wassup
## 8     8 h             sup
## 9     9 i             <NA>
```

Use problems() for investigation

```
dat <- read_csv("data/df-na.csv", col_types = list(col_double(), col_character(), col_character()))
```

```
## # A tibble: 9 × 3
##       x y             z
##   <dbl> <chr>         <chr>
## 1     1 a             hi
## 2    NA b             hello
## 3     3 Not applicable 9999
## 4     4 d             ola
## 5     5 e             hola
## 6    NA f             whatup
## 7     7 g             wassup
## 8     8 h             sup
## 9     9 i             <NA>
```

```
problems(dat)
```

```
## # A tibble: 1 × 5
##     row   col expected actual file
##   <int> <int> <chr>    <chr>  <chr>
## 1     7     1 a double .      ""
```

Column types

type function	data type
`col_character()`	character
`col_date()`	date
`col_datetime()`	POSIXct (date-time)
`col_double()`	double (numeric)
`col_factor()`	factor
`col_guess()`	let readr guess (default)
`col_integer()`	integer
`col_logical()`	logical
`col_number()`	numbers mixed with non-number characters
`col_numeric()`	double or integer
`col_skip()`	do not read
`col_time()`	time

Maybe col_number() ?

```
dat <- read_csv("data/df-na.csv", col_types =
                list(col_number(), col_character(), col_character())) %>% print(n = 10)
```

```
## Warning: One or more parsing issues, call `problems()` on your data frame
## for details, e.g.:
##   dat <- vroom(...)
##   problems(dat)

## # A tibble: 9 × 3
##       x y             z
##   <dbl> <chr>         <chr>
## 1     1 a             hi
## 2    NA b             hello
## 3     3 Not applicable 9999
## 4     4 d             ola
## 5     5 e             hola
## 6    NA f             whatup
## 7     7 g             wassup
## 8     8 h             sup
## 9     9 i             <NA>
```

No difference wrt `col_double`.

Turn off some printed output

```
read_csv("data/df-na.csv")
```

```
## Rows: 9 Columns: 3
## ── Column specification ────────────────────────────────────────
## Delimiter: ","
## chr (3): x, y, z
##
## 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.

## # A tibble: 9 × 3
##   x     y              z
##   <chr> <chr>          <chr>
## 1 1     a              hi
## 2 <NA>  b              hello
## 3 3     Not applicable 9999
## 4 4     d              ola
...
```

```
read_csv("data/df-na.csv", show_col_types = F)
```

Case study: Favourite foods

Favourite foods

Student ID	Full Name	favourite.food	mealPlan	AGE	SES
1	Sunil Huffmann	Strawberry yoghurt	Lunch only	4	High
2	Barclay Lynn	French fries	Lunch only	5	Middle
3	Jayendra Lyne	N/A	Breakfast and lunch	7	Low
4	Leon Rossini	Anchovies	Lunch only	99999	Middle
5	Chidiegwu Dunkel	Pizza	Breakfast and lunch	five	High

```
fav_food <- read_excel("data/favourite-food.xlsx")

fav_food
```

```
## # A tibble: 5 × 6
##   `Student ID` `Full Name`    favourite.food mealPlan AGE   SES
##          <dbl> <chr>          <chr>          <chr>    <chr> <chr>
## 1            1 Sunil Huffmann Strawberry yo… Lunch o… 4     High
## 2            2 Barclay Lynn   French fries   Lunch o… 5     Midd…
## 3            3 Jayendra Lyne  N/A            Breakfa… 7     Low
## 4            4 Leon Rossini   Anchovies      Lunch o… 99999 Midd…
## 5            5 Chidiegwu Dun… Pizza          Breakfa… five  High
```

Variable names

Student ID	Full Name	favourite.food	mealPlan	AGE	SES
1	Sunil Huffmann	Strawberry yoghurt	Lunch only	4	High
2	Barclay Lynn	French fries	Lunch only	5	Middle
3	Jayendra Lyne	N/A	Breakfast and lunch	7	Low
4	Leon Rossini	Anchovies	Lunch only	99999	Middle
5	Chidiegwu Dunkel	Pizza	Breakfast and lunch	five	High

```
fav_food <- read_excel("data/favourite-food.xlsx") %>%
  janitor::clean_names()

fav_food
```

```
## # A tibble: 5 × 6
##   student_id full_name       favourite_food meal_plan age   ses
##        <dbl> <chr>           <chr>          <chr>     <chr> <chr>
## 1          1 Sunil Huffmann  Strawberry yo… Lunch on… 4     High
## 2          2 Barclay Lynn    French fries   Lunch on… 5     Midd…
## 3          3 Jayendra Lyne   N/A            Breakfas… 7     Low
## 4          4 Leon Rossini    Anchovies      Lunch on… 99999 Midd…
## 5          5 Chidiegwu Dunk… Pizza          Breakfas… five  High
```

Handling NAs

Student ID	Full Name	favourite.food	mealPlan	AGE	SES
1	Sunil Huffmann	Strawberry yoghurt	Lunch only	4	High
2	Barclay Lynn	French fries	Lunch only	5	Middle
3	Jayendra Lyne	N/A	Breakfast and lunch	7	Low
4	Leon Rossini	Anchovies	Lunch only	99999	Middle
5	Chidiegwu Dunkel	Pizza	Breakfast and lunch	five	High

```
fav_food <- read_excel("data/favourite-food.xlsx",
                       na = c("N/A", "99999")) %>%
  janitor::clean_names()

fav_food
```

```
## # A tibble: 5 × 6
##   student_id full_name       favourite_food meal_plan age   ses
##        <dbl> <chr>           <chr>          <chr>     <chr> <chr>
## 1          1 Sunil Huffmann  Strawberry yo… Lunch on… 4     High
## 2          2 Barclay Lynn    French fries   Lunch on… 5     Midd…
## 3          3 Jayendra Lyne   <NA>           Breakfas… 7     Low
## 4          4 Leon Rossini    Anchovies      Lunch on… <NA>  Midd…
## 5          5 Chidiegwu Dunk… Pizza          Breakfas… five  High
```

Make age numeric

```
fav_food <- fav_food %>%
  mutate(
    age = if_else(age == "five", "5", age),
    age = as.numeric(age)
    )

glimpse(fav_food)
```

	AGE	SES
	4	High
	5	Middle
ch	7	Low
	99999	Middle
ch	five	High

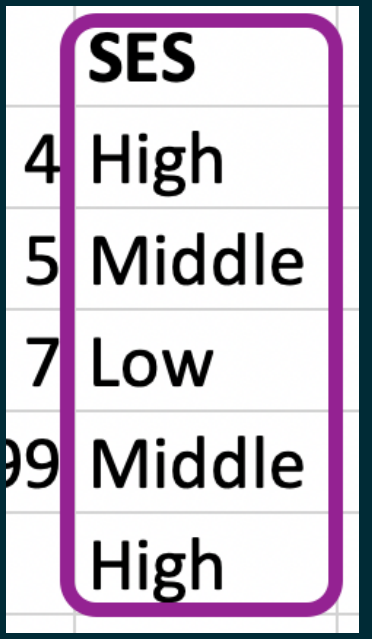
```
## Rows: 5
## Columns: 6
## $ student_id     <dbl> 1, 2, 3, 4, 5
## $ full_name      <chr> "Sunil Huffmann", "Barclay Lynn", "Jayen…
## $ favourite_food <chr> "Strawberry yoghurt", "French fries", NA…
## $ meal_plan      <chr> "Lunch only", "Lunch only", "Breakfast a…
## $ age            <dbl> 4, 5, 7, NA, 5
## $ ses            <chr> "High", "Middle", "Low", "Middle", "High"
```

Socio-economic status

> What order are the levels of `ses` listed in?

```
fav_food %>%
  count(ses)
```

```
## # A tibble: 3 × 2
##   ses        n
##   <chr>  <int>
## 1 High       2
## 2 Low        1
## 3 Middle     2
```



Make `ses` factor

If we use `fct_relevel`, a function (of `forcats`) that makes the old `relevel` more flexible,

```
fav_food <- fav_food %>%
  mutate(ses = fct_relevel(ses, "Low", "Middle
fav_food %>%
  count(ses)
```

```
## # A tibble: 3 × 2
##   ses        n
##   <fct>  <int>
## 1 Low        1
## 2 Middle     2
## 3 High       2
```

We can also use a basic code invoking `factor`,

```
fav_food <- fav_food %>%
  mutate(ses = factor(ses, c("Low", "Middle",
fav_food %>%
  count(ses)
```

```
## # A tibble: 3 × 2
##   ses        n
##   <fct>  <int>
## 1 Low        1
## 2 Middle     2
## 3 High       2
```

Putting it altogether

```
fav_food <- read_excel("data/favourite-food.xlsx", na = c("N/A", "99999")) %>%
  janitor::clean_names() %>%
  mutate(
    age = if_else(age == "five", "5", age),
    age = as.numeric(age),
    ses = fct_relevel(ses, "Low", "Middle", "High")
  )

fav_food
```

```
## # A tibble: 5 × 6
##   student_id full_name       favourite_food meal_plan   age ses
##        <dbl> <chr>           <chr>          <chr>     <dbl> <fct>
## 1          1 Sunil Huffmann  Strawberry yo… Lunch on…     4 High
## 2          2 Barclay Lynn    French fries   Lunch on…     5 Midd…
## 3          3 Jayendra Lyne   <NA>           Breakfas…     7 Low
## 4          4 Leon Rossini    Anchovies      Lunch on…    NA Midd…
## 5          5 Chidiegwu Dunk… Pizza          Breakfas…     5 High
```

Out and back in

```
write_csv(fav_food, file = "data/fav-food-clean.csv")

fav_food_clean <- read_csv("data/fav-food-clean.csv")
```

What happened to `ses` again?

```
fav_food_clean %>%
  count(ses)
```

```
## # A tibble: 3 × 2
##   ses        n
##   <chr>  <int>
## 1 High       2
## 2 Low        1
## 3 Middle     2
```

`read_rds()` and `write_rds()`

- CSVs can be unreliable for saving interim results if there is specific variable type information you want to hold on to.
- An alternative is RDS files, you can read and write them with `read_rds()` and `write_rds()`, respectively.

```
read_rds(path)
write_rds(x, path)
```

Out and back in, take 2

```
write_rds(fav_food, file = "data/fav-food-clean.rds")

fav_food_clean <- read_rds("data/fav-food-clean.rds")

fav_food_clean %>%
  count(ses)
```

```
## # A tibble: 3 × 2
##   ses        n
##   <fct>  <int>
## 1 Low        1
## 2 Middle     2
## 3 High       2
```

Other types of data

Other types of data

- **googlesheets4:** Google Sheets
- **haven**: SPSS, Stata, and SAS files
- **DBI**, along with a database specific backend (e.g. RMySQL, RSQLite, RPostgreSQL etc): allows you to run SQL queries against a database and return a data frame
- **jsonline**: JSON
- **xml2**: xml
- **rvest**: web scraping
- **httr**: web APIs
- **sparklyr**: data loaded into spark