



Loading & Cleaning Data in R

I know the file exists, why doesn't R?

 [steffilazerte](#)
 @steffilazerte@fosstodon.org
 [@steffilazerte](#)
 [steffilazerte.ca](#)

Dr. Steffi LaZerte 
Analysis and Data Tools for Science

	River	Site	Ele	Amo	Wea
1	Grasse	Up stream	Al	0.6055555555555556	sunny
2	Grasse	Mid stream	Al	0.425	snowy
3	Grasse	Down stream	Al	0.1944444444444444	wet
4	Oswegatchie	Up stream	Al	1	cloudy
5	Oswegatchie	Mid stream	Al	0.1611111111111111	cloudy
6	Oswegatchie	Down stream	Al	0.0333333333333333	sunny
7	Raquette	Up stream	Al	0.2916666666666667	sunny
8	Raquette	Mid stream	Al	0.0388888888888889	cloudy
9	Raquette	Down stream	Al	0	sunny
10	St. Regis	Up stream	Al	0.6805555555555556	sunny
11	St. Regis	Mid stream	Al	0.45	snowy
12	St. Regis	Down stream	Al	0.2861111111111111	cloudy
13	Grasse	Up stream	Ba	0.505283381364073	wet
14	Grasse	Mid stream	Ba	0.564841498559078	snowy
15	Grasse	Down stream	Ba	0.523535062439962	cloudy
16	Oswegatchie	Up stream	Ba	0.357348703170029	snowy
17	Oswegatchie	Mid stream	Ba	0.560038424591739	sunny
18	Oswegatchie	Down stream	Ba	1	wet
19	Raquette	Up stream	Ba	0	cloudy
20	Raquette	Mid stream	Ba	0.22478386167147	sunny
21	Raquette	Dow stream	Ba	0.364073006724304	cloudy
22	St. Regis	Up stream	Ba	0.379442843419789	wet
23	St. Regis	Mid stream	Ba	0.296829971181556	snowy
24	St. Regis	Down stream	Ba	0.577329490874159	snowy
25	Grasse	Up stream	Br	0.107142857142857	snowy

Compiled: 2023-04-18

First things first

 Save previous script

 Open New File

(make sure you're in the RStudio Project)

 Write `library(tidyverse)` at the top

 Save this new script

(consider names like `cleaning.R` or `3_loading_and_cleaning.R`)

Side Note

R base vs. tidyverse

R base vs. tidyverse

R base

- Basic R
- Packages are installed and loaded by default
- Base pipe `|>*`



tidyverse

- Collection of 'new' packages developed by a team closely affiliated with RStudio
 - e.g., `ggplot2`, `dplyr`, `tidyr`, `readr`
 - Packages designed to work well together
- Use a slightly different syntax
- tidyverse pipe `%>%` or base pipe `|>*`

Useful to know if functions are
tidyverse or R base



Dealing with data

1. Loading data

- Get your data into R

2. Looking for problems

- Typos
- Incorrectly loaded data

3. Fixing problems

- Corrections
- Renaming

4. Setting formats

- Dates
- Numbers
- Factors

5. Saving your data

Loading Data

Data types: What kind of data do you have?

Specific program files

Type	Extension	R Package	R function
Excel	.xls, .xlsx	<code>readxl</code>	<code>read_excel()</code>
Open Document	.ods	<code>readODS</code>	<code>read_ods()</code>
SPSS	.sav, .zsav, .por	<code>haven</code>	<code>read_spss()</code>
SAS	.sas7bdat	<code>haven</code>	<code>read_sas()</code>
Stata	.dta	<code>haven</code>	<code>read_dta()</code>
Database Files	.dbf	<code>foreign</code>	<code>read.dbf()</code>

Convenient but...

- Can be unreliable
- Can take longer

For files that don't change, better to save as a `*.csv`
(Comma-separated-variables file)

Data types: What kind of data do you have?

General text files

Type	R base	readr package (tidyverse)
Comma separated	<code>read.csv()</code>	<code>read_csv()</code> , <code>read_csv2()</code>
Tab separated	<code>read.delim()</code>	<code>read_tsv()</code>
Space separated	<code>read.table()</code>	<code>read_table()</code>
Fixed-width	<code>read.fwf()</code>	<code>read_fwf()</code>

- **readr** package especially useful for big data sets (fast!)

We'll focus on

- `readxl` package → `read_excel()`
- `readr` package → `read_csv()`, `read_tsv()`

Where is my data?

Common error

```
1 my_data <- read_csv("weather.csv")
```

```
Error: 'weather.csv' does not exist in current working directory ('/home/steffi/Projects/Workshops/Two-day R Workshop/qmd').
```

With no folder (just file name) R expects file to be in **Working directory**

Working directory is:

- Where your RStudio project is
- Your home directory (My Documents, etc.) [If not using RStudio Projects]
- Where you've set it (using `setwd()` or RStudio's Session > Set Working Directory)

Do use Projects in RStudio

Don't use `setwd()`

Where is my data?

A note on file paths (file locations)

```
1 /home
```

- folders separated by /
- `home` is a folder

Where is my data?

A note on file paths (file locations)

```
1 /home/steffi/
```

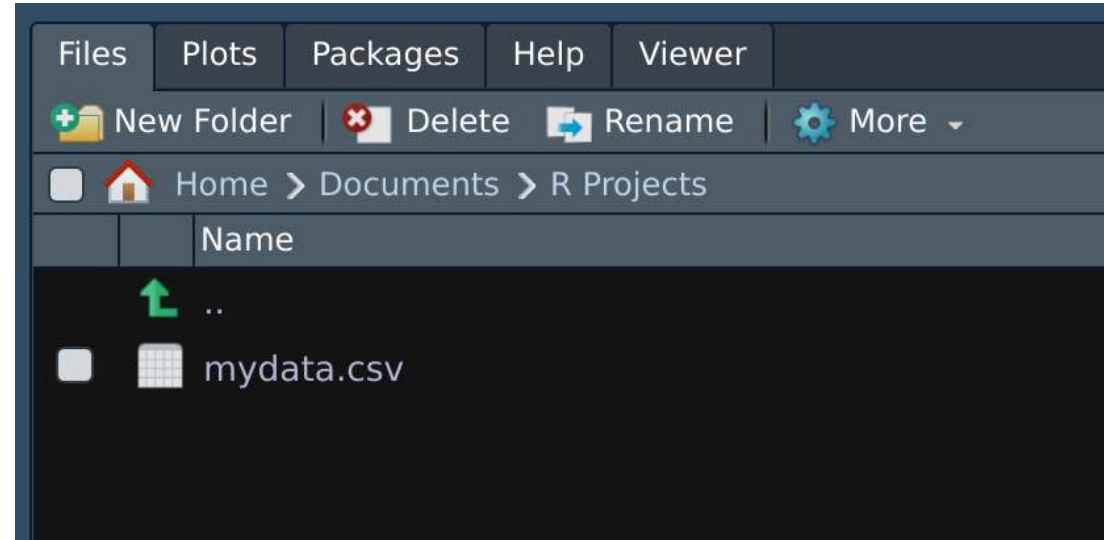
- folders separated by /
- `home` and `steffi` are folders
- `steffi` is a folder inside of `home`

Where is my data?

A note on file paths (file locations)

```
1 /home/steffi/Documents/R Projects/mydata.csv
```

- folders separated by /
- `home`, `steffi`, `Documents`, `R Projects` are folders
- `steffi` is inside of `home`, `Documents` is inside of `steffi`, etc.
- `mydata.csv` is a data file inside `R Projects` folder



RStudio Files Pane

Where is my data?

Absolute Paths

OS	Path
LINUX	/home/steffi/Documents/R Projects/mydata.csv
WINDOWS	C:/Users/steffi/My Documents/R Projects/mydata.csv
MAC	/users/steffi/Documents/R Projects/mydata.csv

Full location, folders and filename

Relative Paths

Path	Where to look
mydata.csv	Here (current directory)
../mydata.csv	Go up one directory (../)
data/mydata.csv	Stay here, go into “data” folder (data/)
../data/mydata.csv	Go up one directory (../), then into “data” folder (data/)

Only *relative* info
Use relative symbols (e.g., `../`)

With RStudio ‘Projects’ only need to use **relative** paths

Keep yourself organized

For simple projects

- Create an ‘RStudio Project’ for each Project (Chapter, Thesis, etc.)
- Create a specific “Data” folder within each project (one per project)

```
1 - Prospect Lake Quality          # Project Folder
2   - prospect_analysis.R
3   - data                        # Data Folder
4     - prospect_data_2017-01-01.csv
5     - prospect_data_2017-02-01.csv
```

- Use **relative** paths to refer to this folder

```
1 d <- read_csv("data/prospect_data_2017-01-01.csv")
```

Let's Load Some Data!

Your turn: Load some data

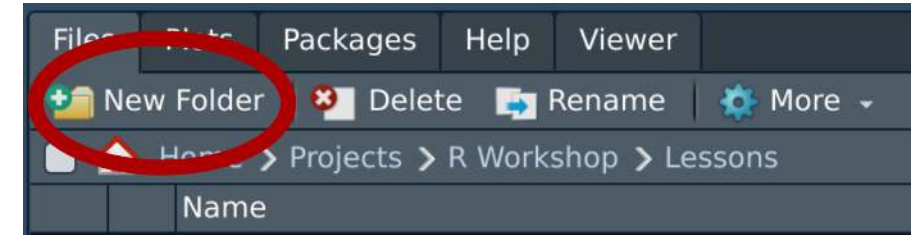
1. Create a 'data' folder in your RStudio project
2. Put `rivers_correct.xlsx` file in the "data" folder
3. Load the package

```
1 library(readxl)
```

4. Read in the Excel file and assign to object `rivers`

```
1 rivers <- read_excel("data/rivers_correct.xlsx")
```

5. Use `head()` and `tail()` functions to look at the data
e.g., `head(rivers)` and `tail(rivers)`
6. Click on the `rivers` object in your "Environment" pane to look at the whole data set



Click on "New Folder"



Use the 'tab' key in RStudio when typing in the file name for auto-complete

Your turn: Load some data

```
1 library(readxl)
2 rivers <- read_excel("data/rivers_correct.xlsx")
```

```
1 head(rivers)

# A tibble: 6 × 5
  `River Name` Site      Ele  Amo      Wea
  <chr>         <chr>    <chr> <chr>    <chr>
1 Grasse       Up stream  Al    0.605555555555556 sunny
2 Grasse       Mid stream Al    0.425      cloudy
3 Grasse       Down stream Al    0.194444444444444 sunny
4 Oswegatchie Up stream  Al    1          cloudy
5 Oswegatchie Mid stream Al    0.161111111111111 snowy
6 Oswegatchie Down stream Al    0.033333333333333 cloudy
```

```
1 tail(rivers)

# A tibble: 6 × 5
  `River Name` Site      Ele  Amo      Wea
  <chr>         <chr>    <chr> <chr>    <chr>
1 Raquette     Up stream  Zr    0.333333333333333 cloudy
2 Raquette     Mid stream Zr    0.111111111111111 snowy
3 Raquette     Down stream Zr    0          sunny
4 St. Regis    Up stream  Zr    0.888888888888889 wet
5 St. Regis    Mid stream Zr    0.777777777777778 sunny
6 St. Regis    Down stream Zr    0.666666666666667 wet
```

	River	Site	Ele	Amo	Wea
1	Grasse	Up stream	Al	0.605555555555556	sunny
2	Grasse	Mid stream	Al	0.425	snowy
3	Grasse	Down stream	Al	0.194444444444444	wet
4	Oswegatchie	Up stream	Al	1	cloudy
5	Oswegatchie	Mid stream	Al	0.161111111111111	cloudy
6	Oswegatchie	Down stream	Al	0.033333333333333	sunny
7	Raquette	Up stream	Al	0.291666666666667	sunny
8	Raquette	Mid stream	Al	0.038888888888889	cloudy
9	Raquette	Down stream	Al	0	sunny
10	St. Regis	Up stream	Al	0.680555555555556	sunny
11	St. Regis	Mid stream	Al	0.45	snowy
12	St. Regis	Down stream	Al	0.286111111111111	cloudy
13	Grasse	Up stream	Ba	0.505283381364073	wet
14	Grasse	Mid stream	Ba	0.564841498559078	snowy
15	Grasse	Down stream	Ba	0.523535062439962	cloudy
16	Oswegatchie	Up stream	Ba	0.357348703170029	snowy
17	Oswegatchie	Mid stream	Ba	0.560038424591739	sunny
18	Oswegatchie	Down stream	Ba	1	wet
19	Raquette	Up stream	Ba	0	cloudy
20	Raquette	Mid stream	Ba	0.22478386167147	sunny
21	Raquette	Dow stream	Ba	0.364073006724304	cloudy
22	St. Regis	Up stream	Ba	0.379442843419789	wet
23	St. Regis	Mid stream	Ba	0.296829971181556	snowy
24	St. Regis	Down stream	Ba	0.577329490874159	snowy
25	Grasse	Up stream	Br	0.107142857142857	snowy

How do I know which function to use?

Look at the file extension:

- `rivers_correct.csv`
- `.csv` → Comma-separated-variables → `read_csv()`

But not always obvious...

How do I know which function to use?

Look at the file: `master_moch.txt`

- Put this file in your `data` folder
- In lower right-hand pane, click on **Files**
 - Click on `data` folder
 - Click on `master_moch.txt`
 - Click “View File” (if asked)

```
ID region hab freq freq.sd p.notes
MCB02 kam 0.5266879074 3.9806600009 3.9806600009 0.4592592593
MCB03 kam -0.9707703735 4.1090031783 4.1090031783 0.5
MCB04 kam -0.9707703735 4.2463067674 4.2463067674 0.5151515152
```

This **does not** read the file into R, but only shows you the contents as text.

Hmm, not comma-separated, maybe tab-separated?

How do I know what to use?

Peak:

- Pick a read function with your best guess (`read_csv()` is a good start)
- Use `n_max` to read only first few rows

```
1 read_csv("data/master_moch.txt", n_max = 3)

# A tibble: 3 × 1
  `ID\tregion\thab\tfreq\tfreq.sd\tp.notes`
  <chr>
1 "MCB02\tkam\t0.5266879074\t3.9806600009\t3.9806600009\t0.4592592593"
2 "MCB03\tkam\t-0.9707703735\t4.1090031783\t4.1090031783\t0.5"
3 "MCB04\tkam\t-0.9707703735\t4.2463067674\t4.2463067674\t0.5151515152"
```

`\t` means tab, so this is tab-separated data

How do I know what to use?

Peak:

- Try again with `read_tsv()`

```
1 read_tsv("data/master_moch.txt", n_max = 3) # note change in function!
```

A tibble: 3 × 6

	ID	region	hab	freq	freq.sd	p.notes
	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>
1	MCB02	kam	0.527	3.98	3.98	0.459
2	MCB03	kam	-0.971	4.11	4.11	0.5
3	MCB04	kam	-0.971	4.25	4.25	0.515

Excellent!

Specifics of loading functions

col_names

- Geolocator data

```
1 my_data <- read_csv("data/geolocators.csv")
2 my_data
```

```
# A tibble: 20 × 2
  `02/05/11 22:29:59` `64`
  <chr>              <dbl>
1 02/05/11 22:31:59    64
2 02/05/11 22:33:59    38
3 02/05/11 22:35:59    38
4 02/05/11 22:37:59    34
5 02/05/11 22:39:59    30
6 02/05/11 22:41:59    34
7 02/05/11 22:43:59    40
8 02/05/11 22:45:59    46
9 02/05/11 22:47:59    48
10 02/05/11 22:49:59    46
# i 10 more rows
```

- `read_csv`, `read_tsv`, etc. assume that the first row contains the column names
- This file doesn't have headers

Oops?

col_names

- Geolocator data

Declare no headings

```
1 my_data <- read_csv("data/geolocators.csv",
2                       col_names = FALSE)
3 my_data
```

```
# A tibble: 21 × 2
   X1                X2
  <chr>            <dbl>
1 02/05/11 22:29:59    64
2 02/05/11 22:31:59    64
3 02/05/11 22:33:59    38
4 02/05/11 22:35:59    38
5 02/05/11 22:37:59    34
6 02/05/11 22:39:59    30
7 02/05/11 22:41:59    34
8 02/05/11 22:43:59    40
9 02/05/11 22:45:59    46
10 02/05/11 22:47:59    48
# i 11 more rows
```

Name headings

```
1 my_data <- read_csv("data/geolocators.csv",
2                       col_names = c("date", "light"))
3 my_data
```

```
# A tibble: 21 × 2
   date                light
  <chr>            <dbl>
1 02/05/11 22:29:59    64
2 02/05/11 22:31:59    64
3 02/05/11 22:33:59    38
4 02/05/11 22:35:59    38
5 02/05/11 22:37:59    34
6 02/05/11 22:39:59    30
7 02/05/11 22:41:59    34
8 02/05/11 22:43:59    40
9 02/05/11 22:45:59    46
10 02/05/11 22:47:59    48
# i 11 more rows
```


skip info rows before data

- Grain size data

```
1 my_data <- read_tsv("data/grain_size.txt")
2 my_data
```



```
# A tibble: 36 × 7
  `DATA DOWNLOAD: 2015-09-23` ...2 ...3 ...4 ...5 ...6 ...7
  <chr> <chr> <chr> <chr> <chr> <chr>
1 SYSTEM 001 <NA> <NA> <NA> <NA> <NA>
2 LOGGER X <NA> <NA> <NA> <NA> <NA>
3 lab_num CSP sample_num depth_lb csa msa fsa
4 3177 CSP01 CSP01-P-1-1 4 13.04 17.37 8.19
5 3178 CSP01 CSP01-P-1-2 12 10.74 16.9 7.92
6 3179 CSP01 CSP01-P-1-3 35 12.11 17.75 6.99
7 3180 CSP01 CSP01-P-1-4 53 17.61 18.16 6.29
8 3181 CSP01 CSP01-P-1-5 83 21.05 18.38 6.26
9 3182 CSP01 CSP01-P-1-6 105 19.02 18.43 6.28
10 3183 CSP08 CSP08-P-1-1 10 11.6 17.14 8.18
# i 26 more rows
```

skip info rows before data

- Grain size data

```
1 my_data <- read_tsv("data/grain_size.txt")
2 my_data
```

Look at the file:

- Click on **Files** tab
- Click on **data** folder
- Click on **grain_size.txt**
- Click “**View file**” (if asked)

```
DATA DOWNLOAD: 2015-09-23
SYSTEM 001
LOGGER X
lab_num CSP sample_num depth_lb csa msa fsa
3177 CSP01 CSP01-P-1-1 4 13.04 17.37 8.19
3178 CSP01 CSP01-P-1-2 12 10.74 16.9 7.92
3179 CSP01 CSP01-P-1-3 35 12.11 17.75 6.99
3180 CSP01 CSP01-P-1-4 53 17.61 18.16 6.29
3181 CSP01 CSP01-P-1-5 83 21.05 18.38 6.26
```

Ah ha!

Metadata was stored at the top of the file

skip info rows before data

- Grain size data
- Add `skip = 3` to skip the first three rows

```
1 my_data <- read_tsv("data/grain_size.txt", skip = 3)
2 my_data
```

```
# A tibble: 33 × 7
  lab_num CSP sample_num depth_lb csa msa fsa
  <dbl> <chr> <chr>      <dbl> <dbl> <dbl> <dbl>
1    3177 CSP01 CSP01-P-1-1         4 13.0 17.4  8.19
2    3178 CSP01 CSP01-P-1-2        12 10.7 16.9  7.92
3    3179 CSP01 CSP01-P-1-3        35 12.1 17.8  6.99
4    3180 CSP01 CSP01-P-1-4        53 17.6 18.2  6.29
5    3181 CSP01 CSP01-P-1-5        83 21.0 18.4  6.26
6    3182 CSP01 CSP01-P-1-6       105 19.0
7    3183 CSP08 CSP08-P-1-1        10 11.6
8    3184 CSP08 CSP08-P-1-2        27 15.4
9    3185 CSP08 CSP08-P-1-3        90 14.9 15.8  7.12
10   3186 CSP02 CSP02-P-1-1         5  8.75  8.64  3.41
# i 23 more rows
```

Much better!

Your turn: Load this data set

Load the telemetry data set: `Sta A Data 2006-11-07.dmp`

1. Look at the file
2. Decide which R function to use based on delimiter (comma, space, or tab?)
3. Any other options need to be specified?

It should look like this:

```
# A tibble: 19 × 7
  StartDate Time      Frequency `Rate/Temp`   Pwr Ant      SD
  <dbl> <time>      <dbl>      <dbl> <dbl> <chr> <dbl>
1  39022 17:15:36    150.      34.8   175 M0      0
2  39022 17:19:14    148.      19.2    72 M0      0
3  39022 17:19:25    148.      19.7   194 M1      0
4  39022 17:20:04    149.      33.8   104 M0      0
5  39022 17:20:17    149.      33.7   152 M1      0
6  39022 17:20:57    150.      34.2   188 M0      0
7  39022 17:22:50    148.       9.8   188 M0      0
# i 12 more rows
```

Too Easy?

Load some of your own tricky data

Your turn: Load this data set

Load the telemetry data set: `Sta A Data 2006-11-07.dmp`

```
1 telemetry <- read_csv("data/Sta A Data 2006-11-07.dmp", skip = 2)
2 telemetry
```

A tibble: 19 × 7

	StartDate	Time	Frequency	`Rate/Temp`	Pwr	Ant	SD
	<dbl>	<time>	<dbl>	<dbl>	<dbl>	<chr>	<dbl>
1	39022	17:15:36	150.	34.8	175	M0	0
2	39022	17:19:14	148.	19.2	72	M0	0
3	39022	17:19:25	148.	19.7	194	M1	0
4	39022	17:20:04	149.	33.8	104	M0	0
5	39022	17:20:17	149.	33.7	152	M1	0
6	39022	17:20:57	150.	34.2	188	M0	0
7	39022	17:22:50	148.	9.8	188	M0	0

i 12 more rows

Looking for problems

Look at the data

- Make sure columns as expected (correctly assigned file format)
- Make sure no extra lines above the data (should we have used a skip?)
- Make sure column names look appropriate

```
1 library(palmerpenguins)
2 penguins

# A tibble: 344 × 8
  species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex    year
  <fct>   <fct>         <dbl>         <dbl>           <int>         <int> <fct> <int>
1 Adelie  Torgersen      39.1           18.7             181           3750 male   2007
2 Adelie  Torgersen      39.5           17.4             186           3800 female 2007
3 Adelie  Torgersen      40.3            18              195           3250 female 2007
4 Adelie  Torgersen      NA              NA              NA            NA <NA>   2007
5 Adelie  Torgersen      36.7           19.3             193           3450 female 2007
6 Adelie  Torgersen      39.3           20.6             190           3650 male   2007
7 Adelie  Torgersen      38.9           17.8             181           3625 female 2007
8 Adelie  Torgersen      39.2           19.6             195           4675 male   2007
9 Adelie  Torgersen      34.1           18.1             193           3475 <NA>   2007
10 Adelie Torgersen      42            20.2             190           4250 <NA>   2007
# i 334 more rows
```

Look at the data

- Did the whole data set load?
- Are there extra blank lines at the end of the data?

```
1 tail(penguins)

# A tibble: 6 × 8
  species    island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex    year
  <fct>      <fct>      <dbl>         <dbl>         <int>         <int> <fct> <int>
1 Chinstrap Dream         45.7           17           195         3650 female 2009
2 Chinstrap Dream         55.8           19.8         207         4000 male   2009
3 Chinstrap Dream         43.5           18.1         202         3400 female 2009
4 Chinstrap Dream         49.6           18.2         193         3775 male   2009
5 Chinstrap Dream         50.8           19           210         4100 male   2009
6 Chinstrap Dream         50.2           18.7         198         3775 female 2009
```


skim()

- Are the formats correct?
 - numbers (**numeric**),
 - text (**character**)
 - date (**date**, **POSIXct**, **datetime**)
 - categories (**factor**)
- Are values appropriate?
 - Should there be **NAs**?
- Are there any typos?
- Number of rows expected?

```
1 library(skimr)
2 skim(penguins)
```

```
— Data Summary —
Name                Values
Number of rows      344
Number of columns    8






Column type frequency:
  factor              3
  numeric             5

Group variables      None

— Variable type: factor
```

	skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
1	species	0	1	FALSE	3	Ade: 152, Gen: 124, Chi: 68
2	island	0	1	FALSE	3	Bis: 168, Dre: 124, Tor: 52
3	sex	11	0.968	FALSE	2	mal: 168, fem: 165

```
— Variable type: numeric
```

	skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
1	bill_length_mm	2	0.994	43.9	5.46	32.1	39.2	44.4	48.5	59.6	
2	bill_depth_mm	2	0.994	17.2	1.97	13.1	15.6	17.3	18.7	21.5	
3	flipper_length_mm	2	0.994	201.	14.1	172	190	197	213	231	
4	body_mass_g	2	0.994	4202.	802.	2700	3550	4050	4750	6300	
5	year	0	1	2008.	0.818	2007	2007	2008	2009	2009	

count()

- Check for sample sizes and potential typos in categorical columns

```
1 count(penguins, species)
```

```
# A tibble: 3 × 2  
  species      n  
  <fct>    <int>  
1 Adelie    152  
2 Chinstrap  68  
3 Gentoo    124
```

```
1 count(penguins, island)
```

```
# A tibble: 3 × 2  
  island      n  
  <fct>    <int>  
1 Biscoe    168  
2 Dream     124  
3 Torgersen  52
```

Example of problematic data

```
1 rivers <- read_csv("data/rivers_correct.csv")
2 rivers
```

```
# A tibble: 300 × 5
  `River Name` Site      Ele    Amo      Wea
  <chr>         <chr>    <chr> <chr>    <chr>
1 Grasse       Up stream Al    0.6055555555555556 sunny
2 Grasse       Mid stream Al    0.425          cloudy
3 Grase        Down stream Al    0.1944444444444444 sunny
4 Oswegatchie  Up stream Al    1              cloudy
5 Oswegatchie  Mid stream Al    0.1611111111111111 snowy
6 Oswegatchie  Down stream Al    0.0333333333333333 cloudy
7 Raquette     Up stream Al    0.2916666666666667 cloudy
8 Raquette     Mid stream Al    0.0388888888888889 sunny
9 Raquette     Down stream Al    0              snowy
10 St. Regis   Up stream Al    0.6805555555555556 wet
# i 290 more rows
```

- Column names are not great (**River Name** not R-friendly) or obvious (what is **Ele**?)
- **Amo** - should be numeric but isn't
- At least one typo in River (**Grase** should be **Grasse**)

Example of problematic data

```
1 skim(rivers)
```

— Variable type: character

	skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
1	River Name	0	1	5	11	0	7	0
2	Site	0	1	9	11	0	3	0
3	Ele	0	1	1	2	0	25	0
4	Amo	12	0.96	1	19	0	198	0
5	Wea	0	1	3	6	0	4	0

- Not much additional info here

Example of problematic data

```
1 count(rivers, `River Name`)
```

```
# A tibble: 7 × 2
```

	`River Name`	n
	<chr>	<int>
1	Grase	1
2	Grasse	73
3	Oswegatchie	75
4	Raquette	74
5	St. Regis	75
6	grasse	1
7	raquette	1

```
1 count(rivers, Site)
```

```
# A tibble: 3 × 2
```

	Site	n
	<chr>	<int>
1	Down stream	100
2	Mid stream	100
3	Up stream	100

Typos in River Name

Fixing problems

Cleaning column names

`clean_names()` from the **janitor** package

```
1 library(janitor)
2 rivers <- clean_names(rivers)
3 rivers
```



```
# A tibble: 300 × 5
  river_name site     ele amo          wea
  <chr>      <chr>   <chr> <chr>      <chr>
1 Grasse    Up stream Al    0.605555555555556 sunny
2 Grasse    Mid stream Al    0.425        cloudy
3 Grasse    Down stream Al    0.194444444444444 sunny
4 Oswegatchie Up stream Al    1            cloudy
5 Oswegatchie Mid stream Al    0.161111111111111 snowy
6 Oswegatchie Down stream Al    0.0333333333333333 cloudy
7 Raquette  Up stream Al    0.291666666666667 cloudy
8 Raquette  Mid stream Al    0.0388888888888889 sunny
9 Raquette  Down stream Al    0            snowy
10 St. Regis Up stream Al    0.680555555555556 wet
# i 290 more rows
```

Cleaning column names

`rename()` * columns

```
1 rivers <- rename(rivers, element = ele, amount = amo)
2 rivers
```

A tibble: 300 × 5

	river_name	site	element	amount	wea
	<chr>	<chr>	<chr>	<chr>	<chr>
1	Grasse	Up stream	Al	0.6055555555555556	sunny
2	Grasse	Mid stream	Al	0.425	cloudy
3	Grase	Down stream	Al	0.1944444444444444	sunny
4	Oswegatchie	Up stream	Al	1	cloudy
5	Oswegatchie	Mid stream	Al	0.1611111111111111	snowy
6	Oswegatchie	Down stream	Al	0.0333333333333333	cloudy
7	Raquette	Up stream	Al	0.2916666666666667	cloudy
8	Raquette	Mid stream	Al	0.0388888888888889	sunny
9	Raquette	Down stream	Al	0	snowy
10	St. Regis	Up stream	Al	0.6805555555555556	wet

i 290 more rows

Subsetting columns

select () * columns you do want

```
1 rivers <- select(rivers, river_name, site, element, amount)
```

OR, unselect () columns you don't want

```
1 rivers <- select(rivers, -wea)
2 rivers
```

```
# A tibble: 300 × 4
  river_name site      element amount
  <chr>      <chr>      <chr>    <chr>
1 Grasse    Up stream    Al      0.6055555555555556
2 Grasse    Mid stream   Al      0.425
3 Grasse    Down stream  Al      0.1944444444444444
4 Oswegatchie Up stream    Al      1
5 Oswegatchie Mid stream   Al      0.1611111111111111
6 Oswegatchie Down stream  Al      0.0333333333333333
7 Raquette  Up stream    Al      0.2916666666666667
8 Raquette  Mid stream   Al      0.0388888888888889
9 Raquette  Down stream  Al      0
10 St. Regis Up stream    Al      0.6805555555555556
# i 290 more rows
```

Cleaning columns

Put it all together

```
1 rivers <- read_csv("data/rivers_correct.csv")
2 rivers <- clean_names(rivers)
3 rivers <- rename(rivers, element = ele, amount = amo)
4 rivers <- select(rivers, -wea)
5 rivers
```

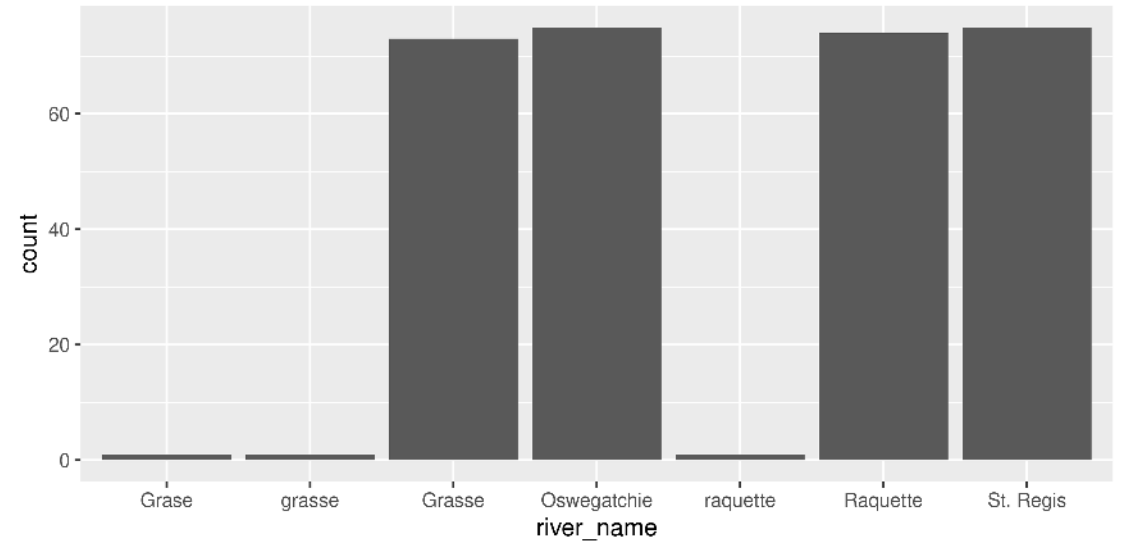
```
# A tibble: 300 × 4
  river_name site      element amount
  <chr>      <chr>      <chr>    <chr>
1 Grasse    Up stream    Al      0.6055555555555556
2 Grasse    Mid stream  Al      0.425
3 Grasse    Down stream Al      0.1944444444444444
4 Oswegatchie Up stream  Al      1
5 Oswegatchie Mid stream  Al      0.1611111111111111
6 Oswegatchie Down stream Al      0.03333333333333333
7 Raquette  Up stream    Al      0.2916666666666667
8 Raquette  Mid stream  Al      0.03888888888888889
9 Raquette  Down stream Al      0
10 St. Regis Up stream    Al      0.6805555555555556
# i 290 more rows
```

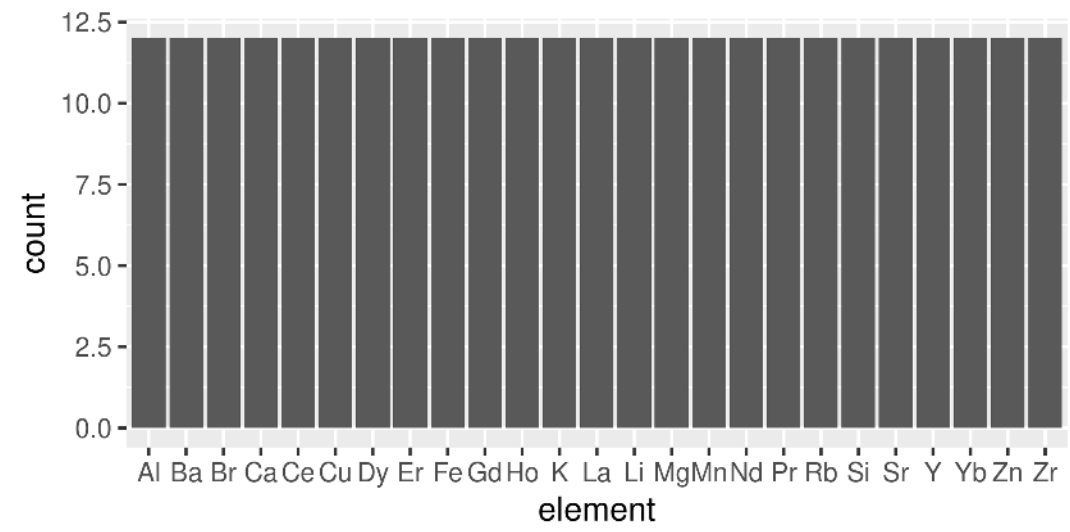
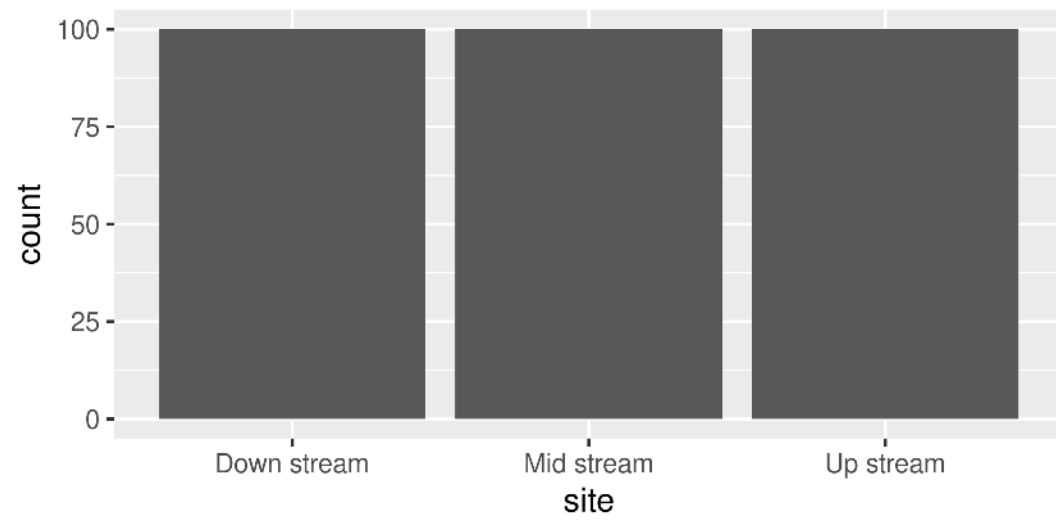
Note how code repeats data frame `rivers...`

Fixing typos

Look for typos (Visually)

```
1 ggplot(data = rivers, aes(x = river_name)) + geom_bar()
2 ggplot(data = rivers, aes(x = site)) + geom_bar()
3 ggplot(data = rivers, aes(x = element)) + geom_bar()
```





Fixing typos

Look for typos with `count()`*

```
1 count(rivers, river_name)
```

```
# A tibble: 7 × 2
  river_name      n
  <chr>         <int>
1 Grase          1
2 Grasse        73
3 Oswegatchie   75
4 Raquette      74
5 St. Regis     75
6 grasse         1
7 raquette       1
```

`filter()`* the data to highlight them

```
1 filter(rivers, river_name == "Grase")
```

```
# A tibble: 1 × 4
  river_name site      element amount
  <chr>      <chr>      <chr>   <chr>
1 Grase      Down stream A1  0.19444444444444444
```

Fixing typos

Replace typos

Combine the `if_else()` / `case_when()` functions with `mutate()` function

`mutate()` creates or changes columns in a data frame:

```
1 mutate(dataframe, column = new_values)
```

`if_else()` tests for a condition, and returns one value if FALSE and another if TRUE

```
1 if_else(condition, value_if_true, value_if_false)
```

`case_when()` tests for multiple conditions, and returns different values depending

```
1 case_when(condition1 ~ value_if_true1,  
2           condition2 ~ value_if_true2,  
3           condition3 ~ value_if_true3,  
4           TRUE ~ default_value)
```

Fixing typos

Replace typos

Combine the `if_else` function with the `mutate()` function

```
1 rivers <- mutate(rivers, river_name = if_else(river_name == "Grase", "Grasse", river_name))
```

Check that it's gone:

```
1 filter(rivers, river_name == "Grase")  
  
# A tibble: 0 × 4  
# i 4 variables: river_name <chr>, site <chr>, element <chr>, amount <chr>
```

Iterative process

- Make some corrections
- Check the data
- Make some more corrections (either add to or modify existing code)

Your Turn: Fix another “Grasse” typo

1. Check the data with `count()`
2. Use `mutate()` and `if_else()` to fix the typo

```
1 rivers <- read_csv("data/rivers_correct.csv")
2 rivers <- clean_names(rivers)
3 rivers <- rename(rivers, element = ele, amount = amo)
4 rivers <- select(rivers, -wea)
5 rivers <- mutate(rivers, river_name = if_else(river_name == "Grase", "Grasse", river_name))
6
7 rivers <- mutate(???, ??? = ???)
```

Too Easy?

Examine and fix problems in your own data

Your Turn: Fix another “Grasse” typo

1. Check the data with `count()`
2. Use `mutate()` and `if_else()` to fix the typo

```
1 rivers <- read_csv("data/rivers_correct.csv")
2 rivers <- clean_names(rivers)
3 rivers <- rename(rivers, element = ele, amount = amo)
4 rivers <- select(rivers, -wea)
5 rivers <- mutate(rivers, river_name = if_else(river_name == "Grase", "Grasse", river_name))
6
7 rivers <- mutate(rivers, river_name = if_else(river_name == "grasse", "Grasse", river_name))
```

Fixing typos

To be more efficient, fix all typos at once

```
1 rivers <- read_csv("data/rivers_correct.csv")
2 rivers <- clean_names(rivers)
3 rivers <- rename(rivers, element = ele, amount = amo)
4 rivers <- select(rivers, -wea)
5 rivers <- mutate(rivers,
6                   river_name = if_else(river_name %in% c("Grase", "grasse"), "Grasse", river_name))
```

== compares one item to one other

%in% compares one item to many different ones

Tangent: tidyverse functions

tidyverse functions

rename(), select(), mutate()

- tidyverse functions always start with the **data**, followed by other arguments
- you can reference any **column** from 'data'

```
1 rivers <- read_csv("data/rivers_correct.csv")
2 rivers <- clean_names(rivers)
3 rivers <- rename(rivers, element = ele, amount = amo)
4 rivers <- select(rivers, -wea)
5 rivers <- mutate(rivers,
6                   if_else(river_name %in% c("Grase", "grasse"), "Grasse", river_name))
```

- **rename()** changes column names
- **select()** chooses columns to keep or to remove (with -)
- **mutate()** changes column contents

Why use tidyverse functions?

Pipes! `|>*` Allow you to string commands together

Instead of:

```
1 rivers <- read_csv("data/rivers_correct.csv")
2 rivers <- clean_names(rivers)
3 rivers <- rename(rivers, element = ele, amount = amo)
4 rivers <- select(rivers, -wea)
5 rivers <- mutate(rivers,
6                   if_else(river_name %in% c("Grase", "grasse"), "Grasse", river_name))
```

We have:

```
1 rivers <- read_csv("data/rivers_correct.csv") |>
2   clean_names() |>
3   rename(element = ele, amount = amo) |>
4   select(-wea) |>
5   mutate(river_name = if_else(river_name %in% c("Grase", "grasse"), "Grasse", river_name))
```

Play around

Take a moment to play with this code in your console

Convert this:

```
1 rivers <- read_csv("data/rivers_correct.csv")
2 rivers <- clean_names(rivers)
3 rivers <- rename(rivers, element = ele, amount = amo)
4 rivers <- select(rivers, -wea)
5 rivers <- mutate(rivers,
6                   river_name = if_else(river_name %in% c("Grase", "grasse"), "Grasse", river_name))
```

To this:

```
1 rivers <- read_csv("data/rivers_correct.csv") |>
2   clean_names() |>
3   rename(element = ele, amount = amo) |>
4   select(-wea) |>
5   mutate(river_name = if_else(river_name %in% c("Grase", "grasse"), "Grasse", river_name))
```

Back to the program!

Your turn: Fix the remaining typo

- Remember this is an iterative process (you may find your self reloading the data often)
- Find the typo (expect `river_name`: Grasse, Oswegatchie, Raquette, St.Regis)
- Add fix to code:

```
1 rivers <- read_csv("data/rivers_correct.csv") |>
2   clean_names() |>
3   rename(element = ele, amount = amo) |>
4   select(-wea) |>
5   mutate(river_name = if_else(river_name %in% c("Grase", "grasse"), "Grasse", river_name))
```

Remember...

Comparing single items

```
1 A == "hello"
2 A %in% "hello"
```

Comparing multiple items

```
1 A %in% c("hello", "bye")
2 # NOT A == c("hello", "bye")
```

Too Easy?

Examine and fix problems in your own data

Your turn: Fix the remaining typo

- Remember this is an iterative process (you may find your self reloading the data often)
- Don't worry about numerical problems for now

All typos fixed

```
1 rivers <- read_csv("data/rivers_correct.csv") |>
2   clean_names() |>
3   rename(element = ele, amount = amo) |>
4   select(-wea) |>
5   mutate(river_name = if_else(river_name %in% c("Grase","grasse"), "Grasse", river_name),
6         river_name = if_else(river_name == "raquette", "Raquette", river_name))
```

Let's combine these with `case_when()`...

Your turn: Fix the remaining typo

- Remember this is an iterative process (you may find your self reloading the data often)
- Don't worry about numerical problems for now

All typos fixed

```
1 rivers <- read_csv("data/rivers_correct.csv") |>
2   clean_names() |>
3   rename(element = ele, amount = amo) |>
4   select(-wea) |>
5   mutate(river_name = case_when(river_name %in% c("Grase", "grasse") ~ "Grasse",
6                                 river_name == "raquette" ~ "Raquette",
7                                 TRUE ~ river_name))
```

Fixing formats

Typos that affect classes (formats)

Look for problems

```
1 rivers

# A tibble: 300 × 4
  river_name site      element amount
  <chr>      <chr>      <chr>    <chr>
1 Grasse    Up stream    Al      0.6055555555555556
2 Grasse    Mid stream   Al      0.425
3 Grasse    Down stream  Al      0.1944444444444444
4 Oswegatchie Up stream    Al      1
5 Oswegatchie Mid stream   Al      0.1611111111111111
6 Oswegatchie Down stream  Al      0.0333333333333333
7 Raquette  Up stream    Al      0.2916666666666667
8 Raquette  Mid stream   Al      0.0388888888888889
9 Raquette  Down stream  Al      0
10 St. Regis Up stream    Al      0.6805555555555556
# i 290 more rows
```

Why all character (chr)?

Changing classes

Function	Input	Output
<code>as.character()</code>	Any vector	Text (Characters)
<code>as.numeric()</code>	Any vector (but returns NAs if not numbers)	Numbers
<code>as.logical()</code>	TRUE, FALSE, T, F, 0 (FALSE), any other number (all TRUE)	TRUE or FALSE
<code>as.factor()</code>	Any vector	Categories

For example...

```
1 a <- c(1, 2, 10)
2 as.character(a)
```

```
[1] "1"  "2"  "10"
```

```
1 as.numeric(a)
```

```
[1] 1  2 10
```

```
1 b <- c("hello", "bye", 1)
2 as.character(b)
```

```
[1] "hello" "bye"   "1"
```

```
1 as.numeric(b)
```

```
[1] NA NA  1
```

We'll deal with dates and times later...

Fixing numerical typos

Find the problem (when we don't know what they are)

- Make a new column and convert `amount` to numbers

```
1 rivers <- mutate(rivers, amount2 = as.numeric(amount))
```

```
Warning: There was 1 warning in `mutate()`.  
! In argument: `amount2 = as.numeric(amount)`.  
Caused by warning:  
! NAs introduced by coercion
```

NAs introduced by coercion

means the function was forced to create NAs.

This warning tells us that some values didn't convert to numbers

Fixing numerical typos

Find the problem (when we don't know what it is)

- Make a new column and convert `amount` to numbers
- Find out where the conversion didn't work

```
1 filter(rivers, !is.na(amount), is.na(amount2))  
  
# A tibble: 1 × 5  
  river_name site      element amount amount2  
  <chr>      <chr>    <chr>   <chr>   <dbl>  
1 Raquette  Mid stream Ca    <0.1    NA
```

- `is.na()` is **TRUE** when the value is missing (**NA**)
- `!` turns a **TRUE** into a **FALSE** (and vice versa)
- This asks, which values are **not missing to begin with** (`!is.na(amount)`) but **are missing after** the conversion (`is.na(amount2)`)

Fixing numerical typos

Find the problem (when we know what it is):

```
1 filter(rivers, amount == "<0.1")
```

A tibble: 1 × 5

	river_name	site	element	amount	amount2
	<chr>	<chr>	<chr>	<chr>	<dbl>
1	Raquette	Mid stream	Ca	<0.1	NA

Fix problem

```
1 rivers <- mutate(rivers, amount = if_else(amount == "<0.1", "0", amount))
```

Correct the class

```
1 rivers <- mutate(rivers, amount = as.numeric(amount))
```

Fixing numerical typos

Last, but not least, check...

```
1 rivers

# A tibble: 300 × 5
  river_name site      element amount amount2
  <chr>      <chr>      <chr>    <dbl>    <dbl>
1 Grasse    Up stream    Al      0.606    0.606
2 Grasse    Mid stream   Al      0.425    0.425
3 Grasse    Down stream  Al      0.194    0.194
4 Oswegatchie Up stream    Al      1        1
5 Oswegatchie Mid stream   Al      0.161    0.161
6 Oswegatchie Down stream  Al      0.0333   0.0333
7 Raquette  Up stream    Al      0.292    0.292
8 Raquette  Mid stream   Al      0.0389   0.0389
9 Raquette  Down stream  Al      0        0
10 St. Regis Up stream    Al      0.681    0.681
# i 290 more rows
```

Put it together...

```
1 rivers <- read_csv("data/rivers_correct.csv") |>
2   clean_names() |>
3   rename(element = ele, amount = amo) |>
4   select(-wea) |>
5   mutate(river_name = case_when(river_name %in% c("Grase", "grasse") ~ "Grasse",
6                                 river_name == "raquette" ~ "Raquette",
7                                 TRUE ~ river_name),
8         amount = if_else(amount == "<0.1", "0", amount),
9         amount = as.numeric(amount))
```

And you have a clean, corrected data frame ready to use

- You have not changed the original data
- You have a **reproducible** record of all corrections
- You can alter these corrections at any time
- You have formatted your data for use in R
- Read these steps line by line to remind yourself what you did

Dates and Times

(Or why does R hate me?)

Dates and Times

- Date/times aren't always recognized as date/times

```
1 geolocators <- read_csv("data/geolocators.csv", col_names = c("time", "light"))
2 geolocators

# A tibble: 21 × 2
   time          light
  <chr>         <dbl>
1 02/05/11 22:29:59    64
2 02/05/11 22:31:59    64
3 02/05/11 22:33:59    38
4 02/05/11 22:35:59    38
5 02/05/11 22:37:59    34
6 02/05/11 22:39:59    30
# i 15 more rows
```

Here `time` column is considered `chr` (character/text)



lubridate package

- Part of `tidyverse`, but needs to be loaded separately
- Great for converting date/time formats

```
1 library(lubridate)
2 geolocators <- mutate(geolocators, time_formatted = dmy_hms(time))
3 geolocators
```

A tibble: 21 × 3

	time <chr>	light <dbl>	time_formatted <dtm>
1	02/05/11 22:29:59	64	2011-05-02 22:29:59
2	02/05/11 22:31:59	64	2011-05-02 22:31:59
3	02/05/11 22:33:59	38	2011-05-02 22:33:59
4	02/05/11 22:35:59	38	2011-05-02 22:35:59
5	02/05/11 22:37:59	34	2011-05-02 22:37:59
6	02/05/11 22:39:59	30	2011-05-02 22:39:59

i 15 more rows

Now `time_formatted` column is considered `dtm` (Date/Time)

lubridate package

Generally, only the order of the **year**, **month**, **day**, **hour**, **minute**, or **second** matters.

date/time	function	class
2018-01-01 13:09:11	<code>ymd_hms()</code>	dtm (POSIXct/POSIXt)
12/20/2019 10:00 PM	<code>mdy_hm()</code>	dtm (POSIXct/POSIXt)
31/01/2000 10 AM	<code>dmy_h()</code>	dtm (POSIXct/POSIXt)
31-01/2000	<code>dmy()</code>	Date

lubridate is smart enough to detect AMs and PMs

Saving data

(For the love of all that is good don't lose that data!!!)

Saving data

Keep yourself organized

- Keep your R-created data in a **different** folder from your 'raw' data *
- If you have a lot going on, split your work into several scripts, and number the both the scripts AND the data sets produced:
- `1_cleaned.csv`
- `2_summarized.csv`
- `3_graphing.csv`

Save your data to file:

```
1 write_csv(rivers, "datasets/rivers_cleaned.csv")
```

Dealing with data

1. Loading data

- Get your data into R

2. Looking for problems

- Typos
- Incorrectly loaded data

3. Fixing problems

- Corrections
- Renaming

4. Setting formats

- Dates
- Numbers
- Factors

5. Saving your data

Wrapping up: Common mistakes

Assuming your data is in one format when it's not

- Print your data to the console and use `skim()` to explore the format of your data
- Use `skim()`, `count()`, `filter()`, `select()`, `ggplot()` to explore the content of your data

Wrapping up: Common mistakes

Confusing pipes with function arguments

- Pipes (`|>` or `%>%`) pass the *output* from one function as *input* to the next function:

```
1 my_data <- my_data |>           # Pass my_data
2   filter(my_column > 5) |>      # Pass my_data, filtered
3   select(my_column, my_second_column)
```

- Arguments may be on different lines, but all part of *one* function

```
1 my_data <- my_data |>           # Pass my_data
2   mutate(my_column1 = if_else(...), # No passing (no pipes!)
3         my_column2 = if_else(...),  # Instead, give 3 arguments to mutate:
4         my_column3 = if_else(...))  # Arguments separated by ",", and surrounded by ( )
```

Wrapping up: Further reading

- R for Data Science
 - [Chapter 5: Transforming data](#)
 - [Chapter 8: RStudio Projects](#)
 - [Chapter 14: Strings](#)
 - [Chapter 15: Factors](#)
 - [Chapter 18: Pipes](#)

