

Workshop: Dealing with Data in R

Loading and Cleaning Data in R

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

	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

First things first

Save previous script

Open New File

(make sure you're in the RStudio Project)

Add `library(tidyverse)` to the top

Save this new script

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

R base vs. tidyverse

R base

- R base is basic R
- Most packages used are installed and loaded by default

R base vs. tidyverse

R base

- R base is basic R
- Most packages used are installed and loaded by default

tidyverse

- Collection of 'new' packages developed by a team closely affiliated with RStudio
- Packages designed to work well together
- Use a slightly different syntax
- Among others, includes packages used for data transformations and visualizations:
 - e.g., **ggplot2**, **dplyr**, **tidyr**, **readr**

R base vs. tidyverse

R base

- R base is basic R
- Most packages used are installed and loaded by default

tidyverse

- Collection of 'new' packages developed by a team closely affiliated with RStudio
- Packages designed to work well together
- Use a slightly different syntax
- Among others, includes packages used for data transformations and visualizations:
 - e.g., **ggplot2**, **dplyr**, **tidyr**, **readr**

Can be helpful to understand whether functions are **tidyverse** or R base functions

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

1. Loading Data

Data types: What kind of data do you have?

Specific program files

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

Data types: What kind of data do you have?

Specific program files

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

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>

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!)
- Error/warnings from **readr** are a bit more helpful

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!)
- Error/warnings from **readr** are a bit more helpful

We'll focus on:

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

Where is my data?

Common error

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

```
## Error: 'weather.csv' does not exist in current working directory ('/home/steffi/Projects/R  
Workshop/Lessons').
```

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

Where is my data?

Common error

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

```
## Error: 'weather.csv' does not exist in current working directory ('/home/steffi/Projects/R  
Workshop/Lessons').
```

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)

Where is my data?

Common error

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

```
## Error: 'weather.csv' does not exist in current working directory ('/home/steffi/Projects/R  
Workshop/Lessons').
```

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)

Using Projects in RStudio is a great idea, try to avoid **setwd()**

Where is my data?

A note on file paths (file locations)

```
/home
```

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

Where is my data?

A note on file paths (file locations)

```
/home/steffi/
```

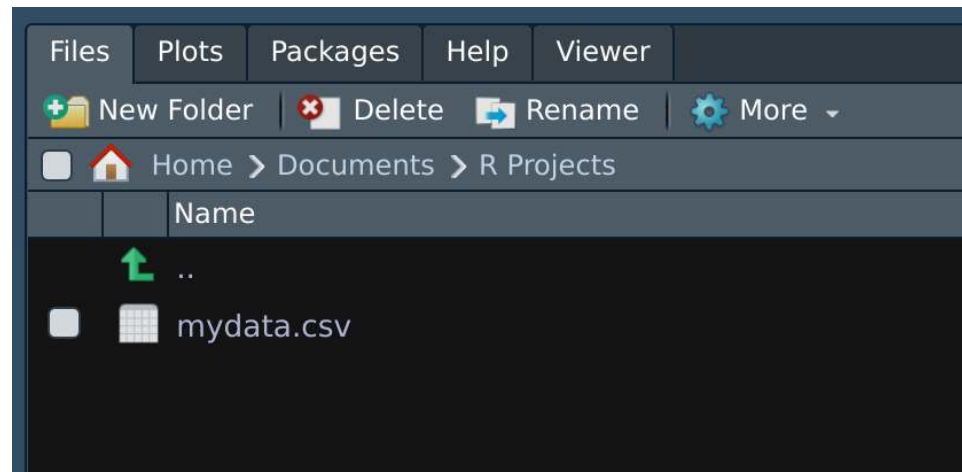
- folders separated by /
- **home** and **steffi** are folders
- **steffi** is a folder inside of **home**

Where is my data?

A note on file paths (file locations)

```
/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



Where is my data?

A note on file paths (file locations)

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

Where is my data?

A note on file paths (file locations)

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 include some folders, and filename.
Use relative symbols (e.g., `../`)

Where is my data?

A note on file paths (file locations)

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

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/)

With RStudio 'Projects' only need to use **relative** paths

Full location, folders and filename

Only include some folders, and filename.
Use relative symbols (e.g., `../`)

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)

```
- Prospect Lake Quality          # Project Folder
  - prospect_analysis.R
- data                            # Data Folder
  - prospect_data_2017-01-01.csv
  - prospect_data_2017-02-01.csv
```

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)

```
- Prospect Lake Quality          # Project Folder
  - prospect_analysis.R
- data                            # Data Folder
  - prospect_data_2017-01-01.csv
  - prospect_data_2017-02-01.csv
```

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

```
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

```
library(readxl)
```

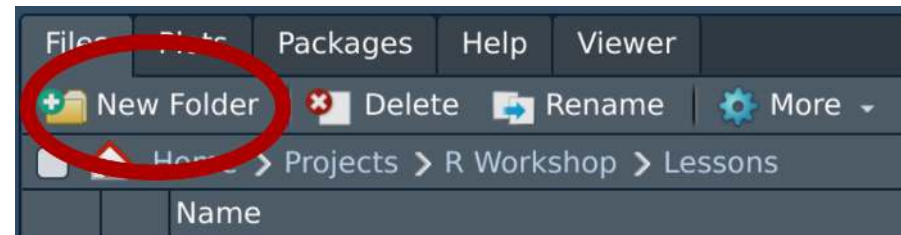
4. Read in the Excel file and assign to object **rivers**

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

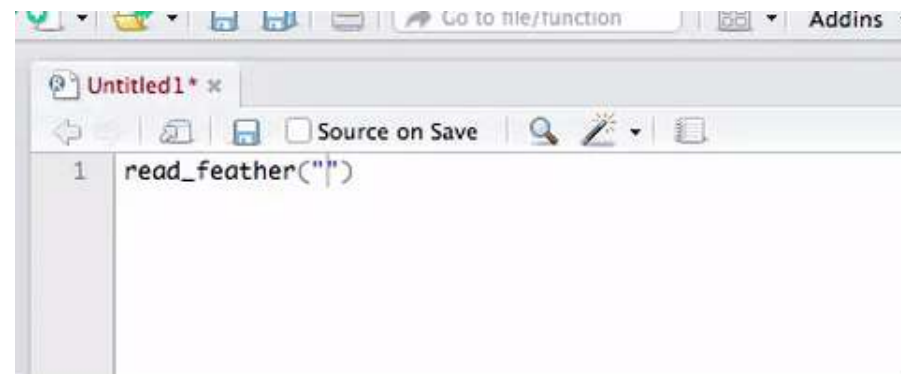
5. Use **head()** and **tail()** functions to look at the data

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

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

```
head(rivers)
```

```
## # A tibble: 6 × 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      sunny
## 3 Grasse       Down stream Al    0.1944444444444444 cloudy
## 4 Oswegatchie Up stream Al    1          wet
## 5 Oswegatchie Mid stream Al    0.1611111111111111 snowy
## 6 Oswegatchie Down stream Al    0.0333333333333333 cloudy
```

```
tail(rivers)
```

```
## # A tibble: 6 × 5
##   `River Name` Site      Ele   Amo      Wea
##   <chr>         <chr>   <chr> <chr>   <chr>
## 1 Raquette     Up stream Zr    0.3333333333333333 cloudy
## 2 Raquette     Mid stream Zr    0.1111111111111111 cloudy
## 3 Raquette     Down stream Zr    0          wet
```

	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

How do I know which function to use?

Look at the file extension:

- [rivers_correct.csv](#)
- `.csv` = Comma-separated-variables = `read_csv()`

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.

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)

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

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.

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

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

```
## # A tibble: 3 × 1
```

```
##   `ID\tregion\tthab\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()`

```
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

1. col_names

- Geolocator data

```
my_data <- read_csv("data/geolocators.csv")  
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  
## # ... with 10 more rows
```

Oops?

1. col_names

- Geolocator data

```
my_data <- read_csv("data/geolocators.csv")
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
## # ... with 10 more rows
```

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

Oops?

1. col_names

Declare no headings

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

```
## # ... with 11 more rows
```

1. col_names

Declare no headings

```
my_data <- read_csv("data/geolocators.csv",  
                    col_names = FALSE)  
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
##	# ... with 11 more rows	

Name headings

```
my_data <- read_csv("data/geolocators.csv",  
                    col_names = c("date", "light"))  
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
##	# ... with 11 more rows	

2. skip info rows before data

- Grain size data

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

```
## # A tibble: 36 × 7
##   `DATA DOWNLOAD: 2015-09-23` ...2 ...3 ...4 ...5 ...6 ...7
##   <chr> <chr> <chr> <chr> <chr> <chr> <chr>
## 1 SYSTEM 001 <NA> <NA> <NA> <NA> <NA> <NA>
## 2 LOGGER X <NA> <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
## # ... with 26 more rows
```

2. skip info rows before data

- Grain size data

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

Look at the file:

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

2. skip info rows before data

- Grain size data

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

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

2. skip info rows before data

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

```
my_data <- read_tsv("data/grain_size.txt", skip = 3)
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  18.4  6.28
## 7   3183 CSP08 CSP08-P-1-1        10 11.6  17.1  8.18
## 8   3184 CSP08 CSP08-P-1-2        27 15.4  16.2  6.76
## 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
## # ... with 23 more rows
```

Your turn: Load this data set

Try loading 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?

Extra Challenge
Load some of your own
tricky data

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
## # ... with 12 more rows
```

Your turn: Load this data set

Try loading the telemetry data set: [Sta A Data 2006-11-07.dmp](#)

```
telemetry <- read_csv("data/Sta A Data 2006-11-07.dmp", skip = 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
## # ... with 12 more rows
```

2. 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

```
library(palmerpenguins)
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
## # ... with 334 more rows
```

Look at the data

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

```
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 column formats correct?
 - i.e., numbers (**numeric**), text (**character**), date (**date**, **POSIXct**, **datetime**), categories (**factor**)
- Are numeric values appropriate?
 - Should there be **NAs**?
- Are there any typos in categorical columns?
- Are there as many rows as you expected?

```
library(skimr)
skim(penguins)
```

```
## — Data Summary —————
##                               Values
## Name                          penguins
## Number of rows                 344
## Number of columns              8
## -----
## Column type frequency:
##   factor                       3
##   numeric                      5
## -----
```

skim()

```
##
## — 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

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

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

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

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

Example of problematic data

```
rivers <- read_csv("data/rivers_correct.csv")
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          sunny
## 3 Grase        Down stream Al    0.194444444444444 cloudy
## 4 Oswegatchie Up stream  Al    1              wet
## 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 cloudy
## 9 Raquette     Down stream Al    0              snowy
## 10 St. Regis   Up stream  Al    0.680555555555556 wet
## # ... with 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

```
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

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

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

```
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 both categorical columns

3. Fixing problems

Cleaning column names

clean_names()

```
library(janitor)
rivers <- clean_names(rivers)
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      sunny
## 3 Grasse     Down stream Al    0.1944444444444444 cloudy
## 4 Oswegatchie Up stream Al    1          wet
## 5 Oswegatchie Mid stream Al    0.1611111111111111 snowy
## 6 Oswegatchie Down stream Al    0.03333333333333333 cloudy
## 7 Raquette   Up stream Al    0.2916666666666667 cloudy
## 8 Raquette   Mid stream Al    0.03888888888888889 cloudy
## 9 Raquette   Down stream Al    0          snowy
## 10 St. Regis  Up stream Al    0.6805555555555556 wet
## # ... with 290 more rows
```

Cleaning column names

rename() columns

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

```
## # A tibble: 300 × 5
##   river_name  site      element amount      wea
##   <chr>      <chr>    <chr>  <chr>      <chr>
## 1 Grasse     Up stream Al      0.605555555555556 sunny
## 2 Grasse     Mid stream Al      0.425          sunny
## 3 Grasse     Down stream Al      0.194444444444444 cloudy
## 4 Oswegatchie Up stream Al      1              wet
## 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 cloudy
## 9 Raquette   Down stream Al      0              snowy
## 10 St. Regis  Up stream Al      0.680555555555556 wet
## # ... with 290 more rows
```

Subsetting columns

select() columns you do want

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


Subsetting columns

`select()` columns you do want

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

OR, `unselect()` columns you don't want

```
rivers <- select(rivers, -wea)  
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      0.19444444444444444  
## 4 Oswegatchie Up stream Al      1  
## 5 Oswegatchie Mid stream Al      0.16111111111111111  
## 6 Oswegatchie Down stream Al      0.033333333333333333  
## 7 Raquette   Up stream Al      0.2916666666666667  
## 8 Raquette   Mid stream Al      0.03888888888888889  
## 9 Raquette   Down stream Al      0
```

Cleaning columns

Put it all together

```
rivers <- read_csv("data/rivers_correct.csv")
rivers <- clean_names(rivers)
rivers <- rename(rivers, element = ele, amount = amo)
rivers <- select(rivers, -wea)
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      0.19444444444444444
## 4 Oswegatchie Up stream Al      1
## 5 Oswegatchie Mid stream Al      0.16111111111111111
## 6 Oswegatchie Down stream Al      0.033333333333333333
## 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
```

Cleaning columns

Put it all together

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

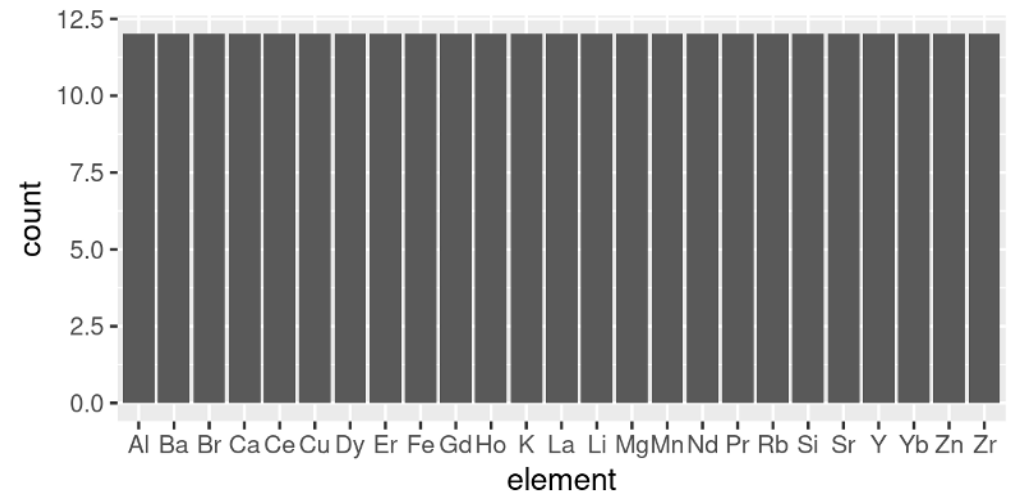
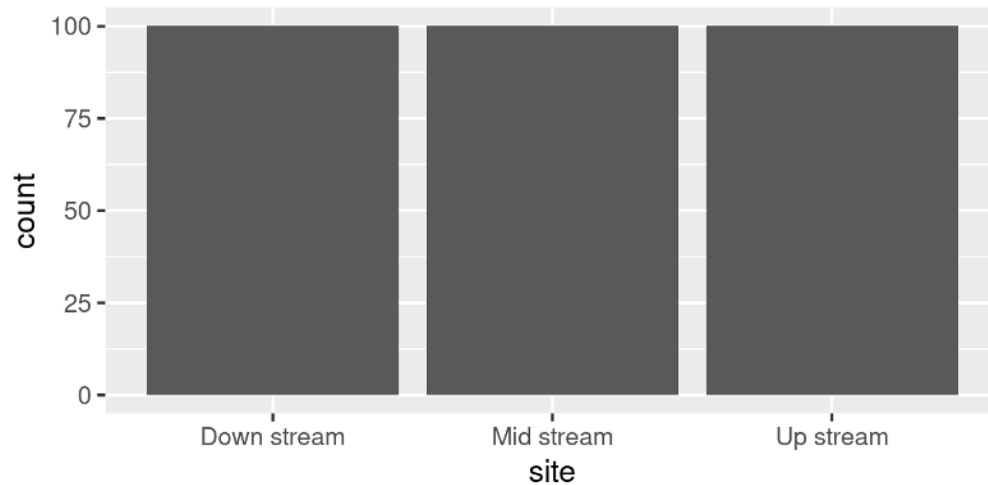
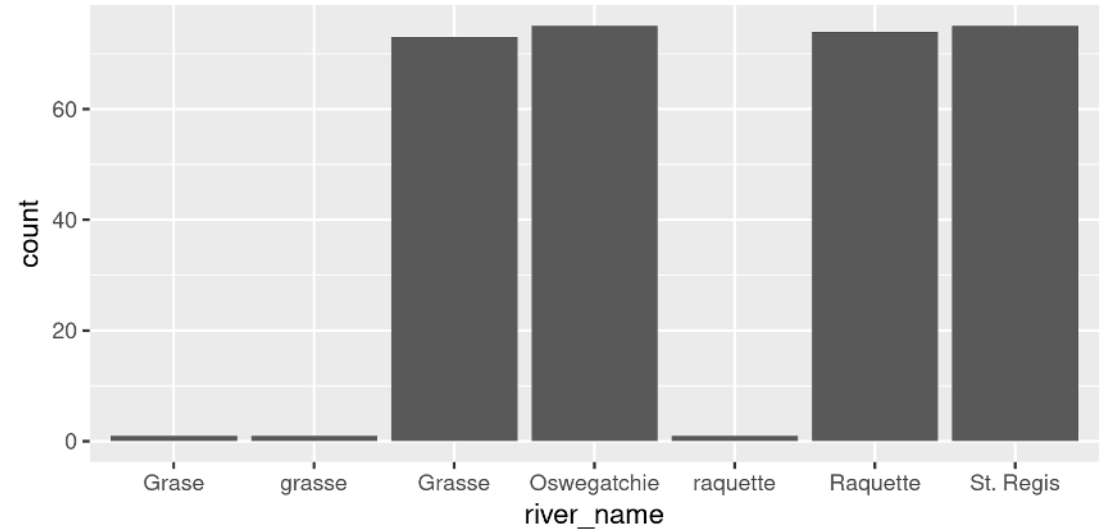
Note repeated data frame
rivers

```
## # 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      0.19444444444444444
## 4 Oswegatchie Up stream Al      1
## 5 Oswegatchie Mid stream Al      0.16111111111111111
## 6 Oswegatchie Down stream Al      0.033333333333333333
## 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
```

Fixing typos

Look for typos (Visually)

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



Fixing typos

Look for typos with `count()`

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

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

`filter()` the data to highlight them

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

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

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

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

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

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

```
case_when(condition1 ~ value_if_true1,  
          condition2 ~ value_if_true2,  
          condition3 ~ value_if_true3,  
          TRUE ~ default_value)
```

Fixing typos

Replace typos

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

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

Check that it's gone:

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

```
## # A tibble: 0 × 4
```

```
## # ... with 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 one of the "Grasse" typos

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

Extra Challenge
Examine and fix problems
in your own data

```
rivers <- read_csv("data/rivers_correct.csv")
rivers <- clean_names(rivers)
rivers <- rename(rivers, element = ele, amount = amo)
rivers <- select(rivers, -wea)
rivers <- mutate(rivers, river_name = if_else(river_name == "Grase", "Grasse", river_name))
```

```
rivers <- mutate(???, ??? = ???)
```

Your Turn: Fix another "Grasse" typo

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

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

Fixing typos

To be more efficient, fix all typos at once

```
rivers <- read_csv("data/rivers_correct.csv")
rivers <- clean_names(rivers)
rivers <- rename(rivers, element = ele, amount = amo)
rivers <- select(rivers, -wea)
rivers <- mutate(rivers,
  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

`rename()`, `select()`, `mutate()`

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

```
rivers <- read_csv("data/rivers_correct.csv")
rivers <- clean_names(rivers)
rivers <- rename(rivers, element = ele, amount = amo)
rivers <- select(rivers, -wea)
rivers <- mutate(rivers,
                 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

Tangent: Why use tidyverse functions?

Pipes! %>% Allow you to string commands together

Instead of:

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

Tangent: Why use tidyverse functions?

Pipes! %>% Allow you to string commands together

Instead of:

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

We have:

```
rivers <- read_csv("data/rivers_correct.csv") %>%
  clean_names() %>%
  rename(element = ele, amount = amo) %>%
  select(-wea) %>%
  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:

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

To this:

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

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:

```
ivers <- read_csv("data/ivers_correct.csv") %>%  
  clean_names() %>%  
  rename(element = ele, amount = amo) %>%  
  select(-wea) %>%  
  mutate(river_name = if_else(river_name %in% c("Grase", "grasse"), "Grasse", river_name))
```

Extra Challenge
Examine and fix problems
in your own data

Remember...

Comparing single items

```
A == "hello"  
A %in% "hello"
```

Comparing multiple items

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


Your turn: Fix the remaining typos

- 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

```
ivers <- read_csv("data/ivers_correct.csv") %>%
  clean_names() %>%
  rename(element = ele, amount = amo) %>%
  select(-wea) %>%
  mutate(river_name = if_else(river_name %in% c("Grase", "grasse"), "Grasse", river_name),
         river_name = if_else(river_name == "raquette", "Raquette", river_name))
```

Your turn: Fix the remaining typos

- 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

```
ivers <- read_csv("data/ivers_correct.csv") %>%  
  clean_names() %>%  
  rename(element = ele, amount = amo) %>%  
  select(-wea) %>%  
  mutate(river_name = if_else(river_name %in% c("Grase", "grasse"), "Grasse", river_name),  
         river_name = if_else(river_name == "raquette", "Raquette", river_name))
```

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

Your turn: Fix the remaining typos

- 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

```

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

```

4. Fixing formats

Typos that affect classes (formats)

Look for problems

```
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
## # ... with 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

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...

```
a <- c(1, 2, 10)
```

```
as.character(a)
```

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

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

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

```
b <- c("hello", "bye", 1)
```

```
as.character(b)
```

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

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

```
## Warning: NAs introduced by coercion
```

```
## [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

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

```
## Warning in mask$eval_all_mutate(quo): 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

```
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):

```
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
```

Fixing numerical typos

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

```
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

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

Fixing numerical typos

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

```
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

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

Correct the class

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

Fixing numerical typos

Last, but not least, check...

```
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
## # ... with 290 more rows
```

Put it together...

```
rivers <- read_csv("data/rivers_correct.csv") %>%
  clean_names() %>%
  rename(element = ele, amount = amo) %>%
  select(-wea) %>%
  mutate(river_name = case_when(river_name %in% c("Grase", "grasse") ~ "Grasse",
                                river_name == "raquette" ~ "Raquette",
                                TRUE ~ river_name),
         amount = if_else(amount == "<0.1", "0", amount),
         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

Dates and Times

(Or why does R hate me?)

Dates and Times

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

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

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

```
## # A tibble: 21 × 3
##   time                light time_formatted
##   <chr>                <dbl> <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
## # ... with 15 more rows
```

lubridate package

Generally, only the order of the **y**ear, **m**onth, **d**ay, **h**our, **m**inute, or **s**econd 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

Note: R *generally* requires that times have dates (**datetime/POSIXct**), but dates don't have to have times (**Date**)

5. Saving 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**

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:

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

* I usually have a **data** folder and then both **raw** and **datasets** folders inside of that

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

```
my_data <- my_data %>%           # Pass my_data
  filter(my_column > 5) %>%     # Pass my_data, filtered
  select(my_column, my_second_column)
```

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

```
my_data <- my_data %>%           # Pass my_data
  mutate(my_column1 = replace(...), # No passing (no pipes!)
         my_column2 = replace(...), # Instead, give 3 arguments to mutate:
         my_column3 = replace(...)) # 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](#)