Reproducible Research with R

Brooke Anderson

1/25/2021

Overview

Aims for lecture

1. What are “knitted” documents?
2. How do knitted documents work?
3. How can you create knitted documents with R?
4. What is data pre-processing?

Running example

As a running example, we will use data preprocessing with the `xcms` package, available on Bioconductor.

Package description:

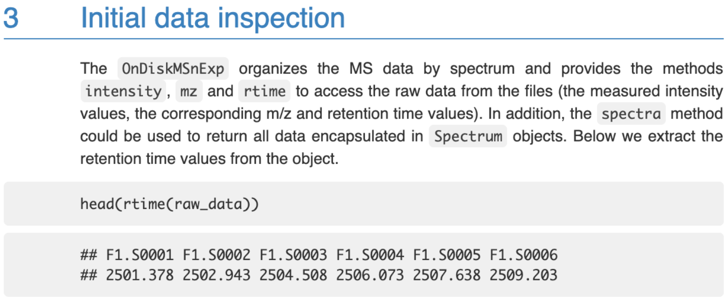
> *"Framework for processing and visualization of chromatographically separated and single-spectra* ***mass spectral data****. Imports from AIA/ANDI NetCDF, mzXML, mzData and mzML files.* ***Preprocesses data*** *for high-throughput, untargeted* ***analyte profiling****."*

Knitted documents

You already use knitted documents!

You have likely already seen and used examples of **knitted documents**.

Many tutorials for R or Python packages are written as knitted documents. For example, here's part of the `xcms` vignette:



3 Initial data inspection

The `OnDiskMSnExp` organizes the MS data by spectrum and provides the methods `intensity`, `mz` and `rtime` to access the raw data from the files (the measured intensity values, the corresponding m/z and retention time values). In addition, the `spectra` method could be used to return all data encapsulated in `Spectrum` objects. Below we extract the retention time values from the object.

```
head(rtime(raw_data))
```

```
## F1.S0001 F1.S0002 F1.S0003 F1.S0004 F1.S0005 F1.S0006
## 2501.378 2502.943 2504.508 2506.073 2507.638 2509.203
```

Definition of knitted documents

The defining characteristic of a knitted document is that it interweaves two elements:

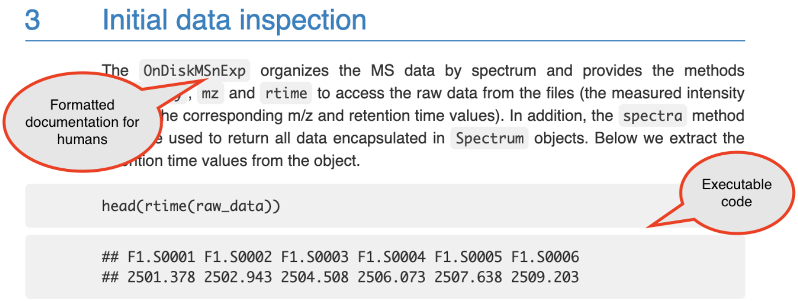
1. Executable code
2. Formatted documentation meant for humans

Definition of knitted documents

The defining characteristic of a knitted document is that it interweaves two elements:

1. Executable code
2. Formatted documentation meant for humans

Example:



3 Initial data inspection

The `OnDiskMSnExp` organizes the MS data by spectrum and provides the methods y , `mz` and `rtime` to access the raw data from the files (the measured intensity he corresponding m/z and retention time values). In addition, the `spectra` method e used to return all data encapsulated in `Spectrum` objects. Below we extract the ention time values from the object.

```
head(rtime(raw_data))
```

```
## F1.S0001 F1.S0002 F1.S0003 F1.S0004 F1.S0005 F1.S0006
## 2501.378 2502.943 2504.508 2506.073 2507.638 2509.203
```

Why use knitted documents?

1. Code is checked every time you render the document (increase *reliability*)
2. Code can be re-run with updated or new datasets (increase **efficiency**)
3. Document is in plain text, so it can be tracked well with version control (increase **transparency**)
4. Code can be clearly and thoroughly documented (increase **reproducibility**)

How knitted documents work

How knitted documents work

1. Knitted documents start as plain text
2. A special section at the start of the document (**preamble**) gives some overall directions about the document
3. Special combinations of characters indicate where the executable code starts
4. Other special combinations show where the regular text starts (and the executable code section ends)
5. Formatting for the rest of the document is specified with a **markup language**
6. You create the final document by **rendering** the plain text document. This process runs through two software programs.
7. The final document is attractive and **read-only**—you should never make edits to this output, only to your initial plain text document.

How knitted documents work

1. Knitted documents start as plain text

For example:

```
# Initial data inspection

The `OnDiskMSExp` organizes the MS data ...
```

How knitted documents work

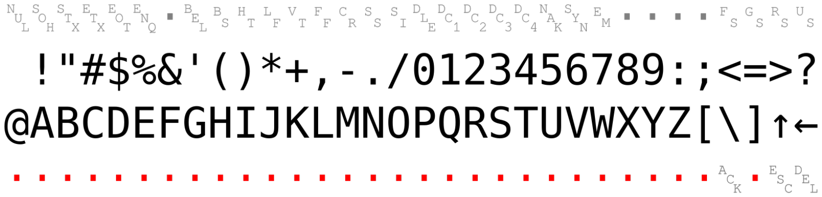
Writing plain text:

- Only use character from the American Standard Code for Information Interchange (ASCII)
- Use a text editor (*not* Word or similar word processing programs, instead RStudio, Notepad, TextEdit, pico, vi/vim, emacs)
- White space is important (empty lines and spaces)
- Flexibility in file extension—choose based on the “knitting” software (for RMarkdown, “.Rmd”)

ASCII

128 characters. Includes:

- Digits 0–9
- Lowercase and uppercase alphabet (a–z, A–Z)
- Some symbols: e.g., ! " , . + - / # * ~
- Some control codes (e.g., new line, tab, ring a bell)



```
NUL SOH STX ETX EOT ENQ ■ BEL BS HT LF VT FF CR SS SI DLE DC1 DC2 DC3 DC4 NAK SYN EM ■ ■ ■ ■ FS GS RS US
 !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑←
■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ACK ■ ESC DEL
```

Image source: https://commons.wikimedia.org/wiki/File:ASCII1963-infobox-paths.svg

White space

To create a section header, you would write:

```
# Initial Data Inspection
```

Meanwhile this:

```
#Initial Data Inspection
```

Would render to:

#Initial Data Inspection

White space

This would create two paragraphs:

```
This is a first paragraph.

This is a second.
```

Meanwhile this would create one:

```
This is a first paragraph.
This is still part of the first paragraph.
```

How knitted documents work

2. A special section at the start of the document (**preamble**) gives some overall directions about the document

In RMarkdown documents, this preamble is specified using **YAML**: YAML Ain't Markup Language.

For example, here is the YAML for this presentation:

```
---
title: "Reproducible Research with R"
author: "Brooke Anderson"
date: "1/25/2021"
output: beamer_presentation
---
```

How knitted documents work

In this preamble, you can specify things using **keys** and **values**.

For example, you can specify the title:

```
title: "Reproducible Research with R"
```

and the type of output:

```
output: beamer_presentation
```

How knitted documents work

There are other types of knitted documents, too—they might use other languages for the preamble and the markup. Examples of other Markup languages include LaTeX and HTML.

There are websites, cheatsheets, and other resources you can use to find out which keywords are available for the preamble in the type of document you're creating, as well as the range of values those keywords can take.

How knitted documents work

3. Special combinations of characters indicate where the executable code starts
4. Other special combinations show where the regular text starts (and the executable code section ends)

How knitted documents work

3. Special combinations of characters indicate where the executable code starts
4. Other special combinations show where the regular text starts (and the executable code section ends)

For example:

~~~
Some text is here. And then some code:

```{r, eval=TRUE}
class_grades <- c(95, 98, 88)
mean(class_grades)
```
~~~

How knitted documents work

This combination indicates the start of executable code:

~~~
```{r}
~~~

How knitted documents work

This combination indicates the start of executable code:

~~~
```{r}
~~~

This combination indicates the start of regular documentation (that is, the end of executable code):

~~~
```
~~~

How knitted documents work

This combination indicates the start of executable code:

~~~
```{r}
~~~

This combination indicates the start of regular documentation (that is, the end of executable code):

~~~
```
~~~

In the starting combination, you can also add some specifications for how you want the code run and showed:

~~~
```{r echo = FALSE, fig.align = "center"}
~~~

How knitted documents work

5. Formatting for the rest of the document is specified with a **markup language**

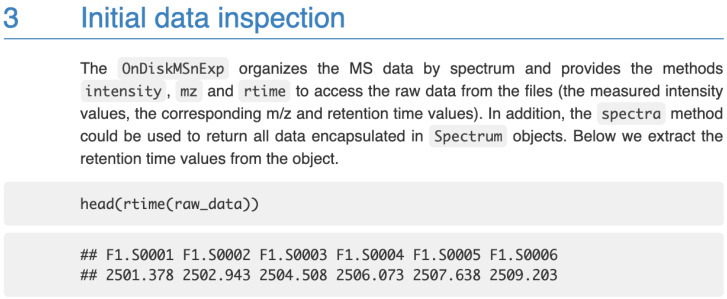
You do not have buttons to click for formatting like bold, italics, font size, and so on. Instead, you use **special characters or character combinations** to specify formatting in the final document.

For example, you'll surround a word or phrase in `**` to make it bold.

To write "**this**" in the final document, you'll write `"**this**"` in the plain text initial document.

How knitted documents work

The start of this document:



3 Initial data inspection

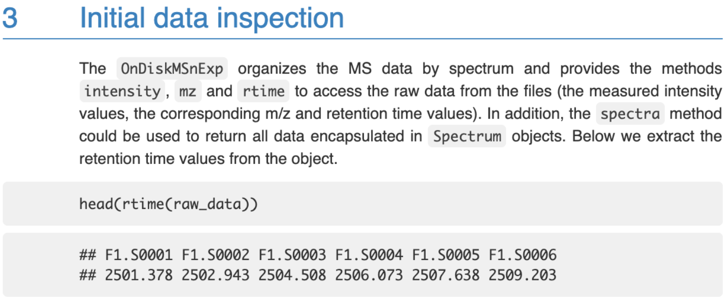
The `OnDiskMSnExp` organizes the MS data by spectrum and provides the methods `intensity`, `mz` and `rtime` to access the raw data from the files (the measured intensity values, the corresponding m/z and retention time values). In addition, the `spectra` method could be used to return all data encapsulated in `Spectrum` objects. Below we extract the retention time values from the object.

```
head(rtime(raw_data))
```

```
## F1.S0001 F1.S0002 F1.S0003 F1.S0004 F1.S0005 F1.S0006
## 2501.378 2502.943 2504.508 2506.073 2507.638 2509.203
```

How knitted documents work

The start of this document:



3 Initial data inspection

The `OnDiskMSnExp` organizes the MS data by spectrum and provides the methods `intensity`, `mz` and `rtime` to access the raw data from the files (the measured intensity values, the corresponding m/z and retention time values). In addition, the `spectra` method could be used to return all data encapsulated in `Spectrum` objects. Below we extract the retention time values from the object.

```
head(rtime(raw_data))
```

```
## F1.S0001 F1.S0002 F1.S0003 F1.S0004 F1.S0005 F1.S0006
## 2501.378 2502.943 2504.508 2506.073 2507.638 2509.203
```

Is written like this:

```
# Initial data inspection

The `OnDiskMSExp` organizes the MS data ...
```

How knitted documents work

Imagine yourself dictating everything to your computer—you have to say not just the words, but the formatting you want as each spot.

Source: The Churchill Project

How knitted documents work

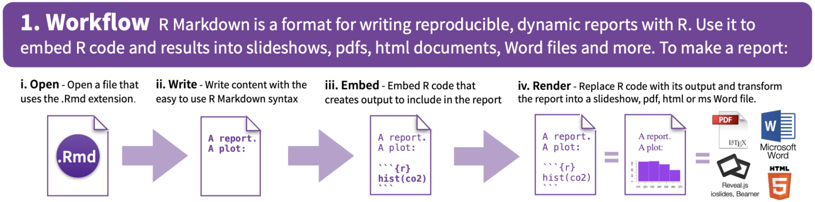
6. You create the final document by **rendering** the plain text document. This process runs through two software programs.
7. The final document is attractive and **read-only**—you should never make edits to this output, only to your initial plain text document.

Creating knitted documents in R

RMarkdown

R has a special format for creating knitted documents, **RMarkdown**.

- **RMarkdown** files are in plain text. They use **YAML** for the preamble and **Markdown** for the primary markup language.
- **Code sections** are marked with ` ```{r} ` at the beginning and ` ``` ` at the end
- **Executable code** can be in R, but also in a number of other languages



1. Workflow R Markdown is a format for writing reproducible, dynamic reports with R. Use it to embed R code and results into slideshows, pdfs, html documents, Word files and more. To make a report:

i. Open - Open a file that uses the .Rmd extension.

ii. Write - Write content with the easy to use R Markdown syntax

iii. Embed - Embed R code that creates output to include in the report

iv. Render - Replace R code with its output and transform the report into a slideshow, pdf, html or ms Word file.

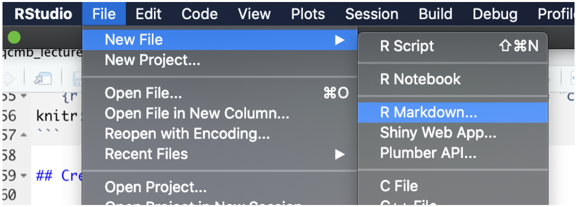
.Rmd
A report.
A plot:
```{r}
hist(co2)
```
=
PDF
LaTeX
Microsoft Word
Reveal.js
ioslides, Beamer
HTML

Source: https://rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf

Creaing an RMarkdown document

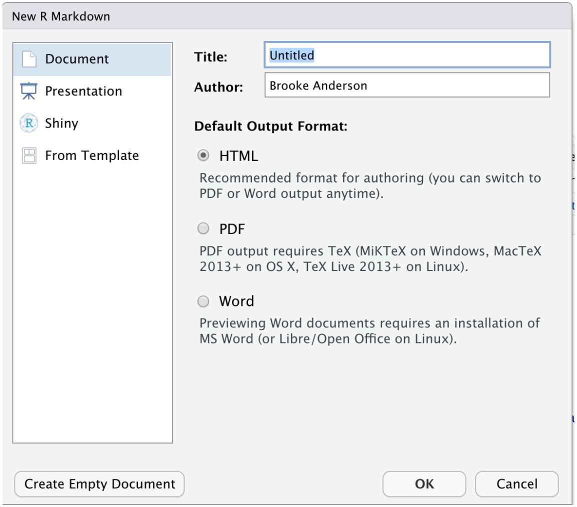
In RStudio, you can use create a number of types of new files through the "File" menu.

To create a new RMarkdown file, choose "New File" -> "RMarkdown"



Creaing an RMarkdown document

This will open a window with some options. You can specify the title of the document, for example, and its output format.


New R Markdown

Title: Untitled

Author: Brooke Anderson

Default Output Format:

[✓ HTML]

Recommended format for authoring (you can switch to PDF or Word output anytime).

PDF

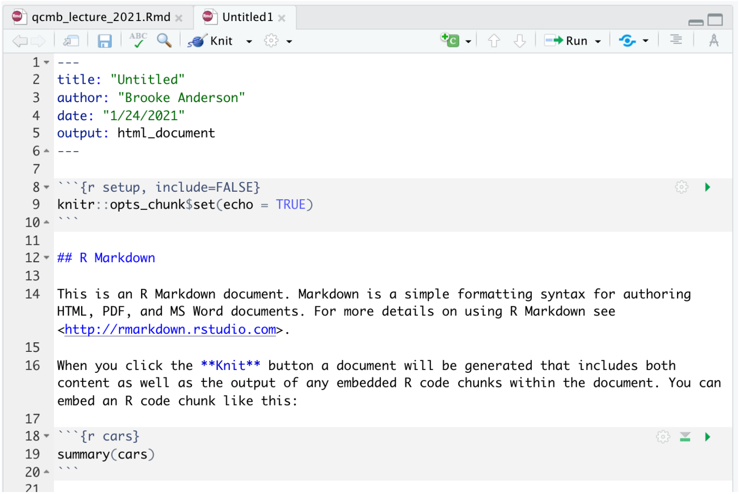
PDF output requires TeX (MiKTeX on Windows, MacTeX 2013+ on OS X, TeX Live 2013+ on Linux).

Word

Previewing Word documents requires an installation of MS Word (or Libre/Open Office on Linux).

Creaing an RMarkdown document

This will open a new document. It won't be blank, though. Instead, it will give an example document that you can test out:



~~~
---
title: "Untitled"
author: "Brooke Anderson"
date: "1/24/2021"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
~~~

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring
HTML, PDF, and MS Word documents. For more details on using R Markdown see
<http://rmarkdown.rstudio.com>.

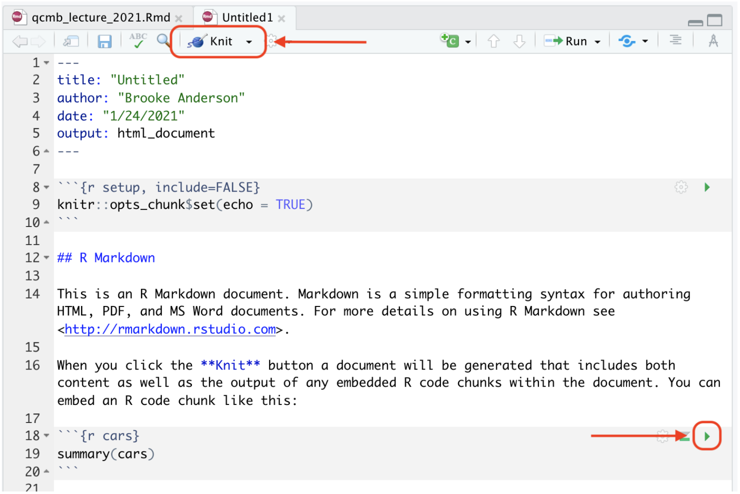
When you click the **Knit** button a document will be generated that includes both
content as well as the output of any embedded R code chunks within the document. You can
embed an R code chunk like this:

```{r cars}
summary(cars)
```

```
```

Creaing an RMarkdown document

You can render the whole document using the **Knit button**. You can run code in specific chunks at the console using a button in the chunk.



~~~
---
title: "Untitled"
author: "Brooke Anderson"
date: "1/24/2021"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
~~~

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring
HTML, PDF, and MS Word documents. For more details on using R Markdown see
<http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both
content as well as the output of any embedded R code chunks within the document. You can
embed an R code chunk like this:

```{r cars}
summary(cars)
```
```
```

Markdown syntax

For the main text, all format is down using **Markdown** syntax.

Some example formatting symbols and conventions:

- ** for bold, * for italics
- # for first-level headers, ## for second-level, and so on
- Double new lines for new paragraphs (blank line between paragraphs)
- Hyphens on new lines for itemized lists

For more, see the RMarkdown Reference Guide: https://rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf

Code chunk options

You can set many options for the code in a specific chunk by setting **chunk options** for that chunk.

- `eval`: Whether to evaluate the code
- `echo`: Whether to print the code
- `message`, `warning`: Whether to print messages and warnings in the document code output

For more, see the RMarkdown Cheatsheet: https://rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf

Useful advanced features in RMarkdown

1. **Include bibliographical references**
2. *Include math / equations*
3. Include code that executes in different languages
4. Shift to LaTeX / HTML if you can't express something in Markdown
5. Can output different formats (articles, presentations, posters)
6. Expands to create larger projects: books (`bookdown`), blogs (`blogdown`), online dashboards (`flexdashboard`)

Including bibliographical references

To include references in RMarkdown documents, you can use something called **BibTeX**.

This has three components:

1. Create a plain text file with listings for each of your references (**BibTeX file**). Save it with the extension `.bib`.
2. In your RMarkdown document, include the filepath to this BibTeX file.
3. In the text of the RMarkdown file, include a key and special character combination anytime you want to reference a paper.

Including bibliographical references

Once you have a BibTeX file, you will include its path in the YAML:

```
---
title: "Reproducible Research with R"
author: "Brooke Anderson"
date: "1/25/2021"
output: beamer_presentation
bibliography: mybibliography.bib
---
```

It is easiest if you save the BibTeX file in the same directory as the RMarkdown document.

Including bibliographical references

You can add references in the text using a key for each paper within special characters (`[@paper1, @paper2]`).

Including bibliographical references

You can add references in the text using a key for each paper within special characters (`[@paper1, @paper2]`).

For example, you would write:

```
This technique follows earlier work [@fox2020,
@anderson2019].
```

To create:

> *"This technique follows earlier work (Fox et al. 2020, Anderson et al. 2019)."*

With full paper details included at the end of the document.

Including bibliographical references

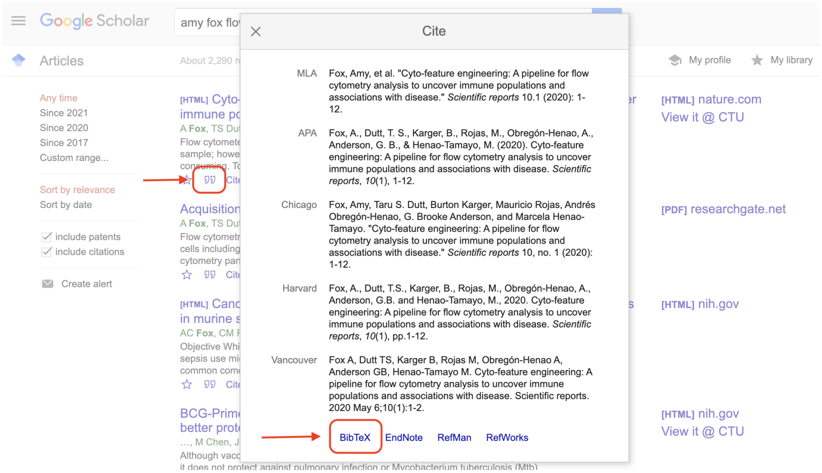
Put all the bibliographical details in the BibTeX file:

You will have a file named something like `mybibliography.bib` with entries for each paper like this:

```
@article{fox2020,
  title={Cyto-feature engineering: A pipeline for flow cytometry
    analysis to uncover immune populations and associations with
    disease},
  author={Fox, Amy and Dutt, Taru S and Karger, Burton and Rojas,
    Mauricio and Obreg{\'o}n-Henao, Andr{\'e}s and
    Anderson, G Brooke and Henao-Tamayo, Marcela},
  journal={Scientific Reports},
  volume={10},
  number={1},
  pages={1--12},
  year={2020}
}
```

Including bibliographical references

You can get these details from Google Scholar:



MLA	Fox, Amy, et al. "Cyto-feature engineering: A pipeline for flow cytometry analysis to uncover immune populations and associations with disease." *Scientific reports* 10.1 (2020): 1-12.
APA	Fox, A., Dutt, T. S., Karger, B., Rojas, M., Obregón-Henao, A., Anderson, G. B., & Henao-Tamayo, M. (2020). Cyto-feature engineering: A pipeline for flow cytometry analysis to uncover immune populations and associations with disease. *Scientific reports*, *10*(1), 1-12.
Chicago	Fox, Amy, Taru S. Dutt, Burton Karger, Mauricio Rojas, Andrés Obregón-Henao, G. Brooke Anderson, and Marcela Henao-Tamayo. "Cyto-feature engineering: A pipeline for flow cytometry analysis to uncover immune populations and associations with disease." *Scientific reports* 10, no. 1 (2020): 1-12.
Harvard	Fox, A., Dutt, T.S., Karger, B., Rojas, M., Obregón-Henao, A., Anderson, G.B. and Henao-Tamayo, M., 2020. Cyto-feature engineering: A pipeline for flow cytometry analysis to uncover immune populations and associations with disease. *Scientific reports*, *10*(1), pp.1-12.
Vancouver	Fox A, Dutt TS, Karger B, Rojas M, Obregón-Henao A, Anderson GB, Henao-Tamayo M. Cyto-feature engineering: A pipeline for flow cytometry analysis to uncover immune populations and associations with disease. Scientific reports. 2020 May 6;10(1):1-2.

BibTeX EndNote RefMan RefWorks

Where to develop RMarkdown skills

- RStudio RMarkdown material: https://rmarkdown.rstudio.com/
  - Tutorials
  - Gallery
  - Advanced articles
- Free online books
  - *RMarkdown: The Definitive Guide*
  - *RMarkdown Cookbook*

Pre-processing for research data

Pre-processing research data

When we take measurements, we do so with the goal of using those **data** (direct measurements of something) to gain **knowledge**.

Sometimes direct measurements line up very closely with a research question (e.g., mortality status of a test subject).

Sometimes data from measurements need a lot of **pre-processing** to use to gain knowledge and test meaningful hypotheses.

Often lots of preprocessing required for data from **complex equipment** that leverage cleverness (in physics or chemistry) to see a new angle, but need more work to interpret the resulting measurements.

LC-MS data

Liquid chromatography–mass spectrometry:

- Often used for chemical analysis, including biochemical molecules, including for metabolomics and proteomics
- Used in academic research, also in industry research (pharmaceutical, for example)
- Leverages principles from chemistry and physics to identify "stuff" in a sample and how much of each type of "stuff"

Preprocessing LC-MS data with code

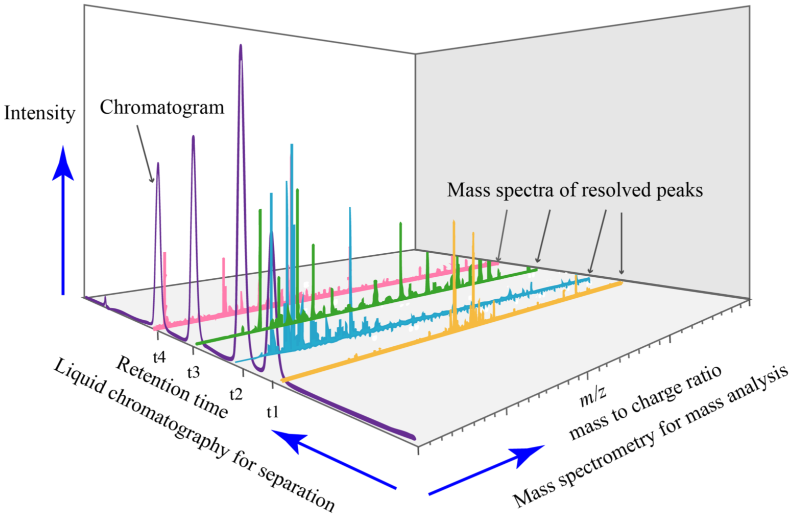

Intensity
Chromatogram
Mass spectra of resolved peaks
t4
t3
t2
t1
Retention time
Liquid chromatography for separation
m/z
mass to charge ratio
Mass spectrometry for mass analysis

Image source: Daniel Norena-Caro

Preprocessing LC-MS data

1. **Import data** from specialized file formats (NetCDF, mzML / mzXML, mzData)
2. **Filter data** (e.g., to a certain range of retention times)
3. **Explore data**
4. Perform **quality control**

Pre-processing research data: GUI vs code

- Complex equipment will often come with its own, or have available through outside vendors, proprietary software
- This is typically based on a **GUI** [more on this]
- You can use this for your preprocessing, but there are some very good reasons not to if you can avoid it

Preprocessing LC-MS data with code

- Complex equipment will often come with its own, or have available through outside vendors, proprietary software
- This is typically based on a **GUI** [more on this]
- You can use this for your preprocessing, but there are some very good reasons to use code instead, when possible.
- Reproducible—by you or by others.
- Much easier to write the Methods section later!

Preprocessing LC-MS data with code

- [More on chromatography]
- [More on mass spec]

Preprocessing LC-MS data with code

Common preprocessing steps:

- Import data (R has *loads* of packages for handling import of many file formats; `mzR` [?] helpful for common LC-MS file formats)
- Filter to a subset range of retention time [add why]
- Ensure files from different samples are comparable
- Identify and remove sample runs that failed or had other major problems
- Line up retention times across samples [?]
- Figure out the typical width of chromatographic peaks in a sample (to decide on parameters for chromatographic peak detection)
- Chromatographic peak detection
- Refine results from peak detection (e.g., diagnose and fix or remove overlapping peaks or incorrectly split peaks)
- Extract specific chromatographic peaks
- Peak alignment / retention time correction (adjusts raw retention times so that they're comparable across

Preprocessing LC-MS data with code

These preprocessing steps all come *before* any data analysis or visualization (other than exploratory data analysis). Data analysis *after* this preprocessing could include things like checking which features [?] / peaks [?] are significantly different between two experimental groups.

Preprocessing choices: GUI or script

You will have choices about how you preprocess the data. For example, when doing chromotographic peak detection, there will be choices in the algorithm that is used (the `centWave` algorithm in `xcms`). These include the expected range of chromatographic peak widths and the deviation of mass to charge values that will are expected, as a maximum, for chromatographic peaks [?].

In the `centWave` function, these two choices can be specified with `peakwidth` and `ppm`, respectively.

Preprocessing choices: GUI or script

- Reproducible—by you or by others.
- Much easier to write the Methods section later!