ETC5523: Communicating with Data

Week 2

Introduction to Web and Data Technologies

Lecturer: *Emi Tanaka*
Department of Econometrics and Business Statistics
ETC5523.Clayton-x@monash.edu
11th August 2020

Housekeeping

- Take-home assessment will be released **Thu 20 Aug 7PM** on Moodle.
- More details on Moodle.

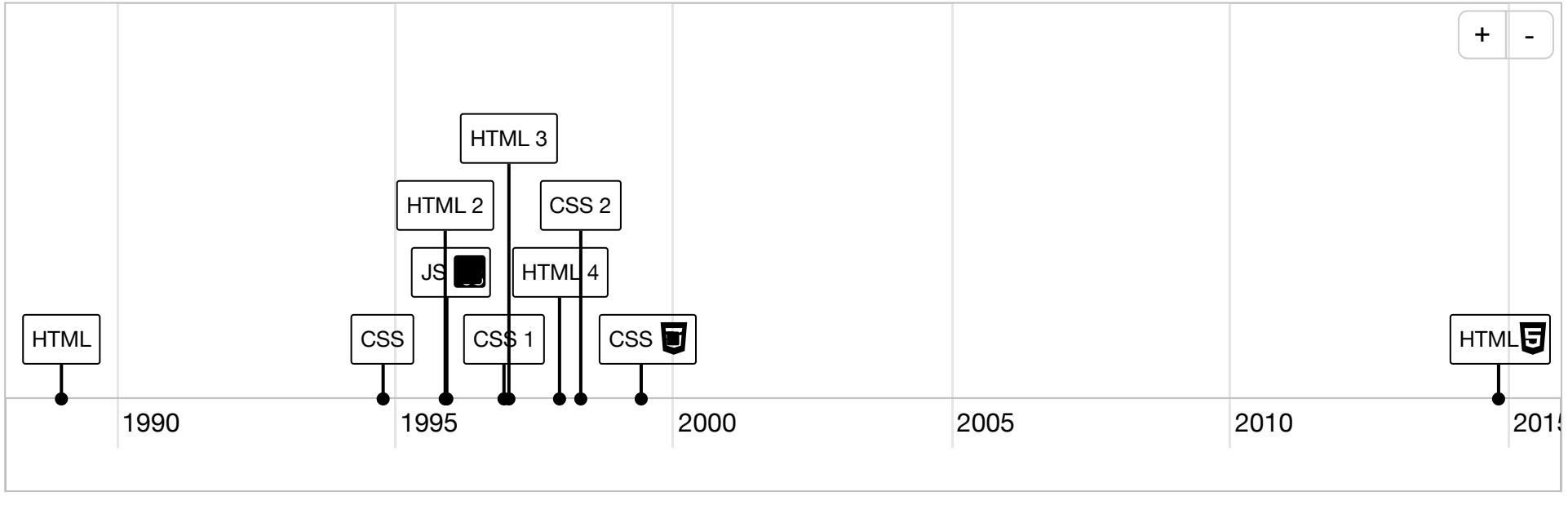
Introduction to Web Technologies

World Wide Web (WWW)

- WWW (or the **Web**) is the information system where documents (web pages) are identified by Uniform Resource Locators (**URL**s)
- A web page consists of:
  - **HTML** provides the basic structure of the web page
  - **CSS** controls the look of the web page (optional)
  - **JS** is a programming language that can modify the behaviour of elements of the web page (optional)
- Keep in mind that you are not a web developer (or maybe you are) so you don't need to know these in-depth, but you require some knowledge in web technologies if you want to do some low-level customisation of HTML documents.

🔨 Web Documents are Handy

- HTML documents are really handy for including interactive elements and supported in almost all computer devices.
- Naturally, this ties in well with interactive data visualisation.
- Below is an interactive timeline visualisation of historical developments of HTML/CSS/JS:



So what exactly is HTML, CSS, and JS?

03:00

Hypertext Markup Language (HTML)

- HTML files have the extension `.html`.
- HTML files are often rendered using a web browser via an URL.
- HTML files are just text files that follows a special syntax that alerts web browsers how to render it.

`lecture-02-simple-example.html`

As seen via a *web browser*

ETC5523: Communicating with Data

Lecturers

- Emi Tanaka (Chief Examiner)
- Stuart Lee

Tutor

- Mitchell O'Hara-Wild

As seen via a *text editor*

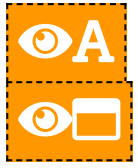
HTML Structure

```
<!DOCTYPE html>

<html>
<!--This is a comment and ignored by web client.-->
<head>
  <!--This section contains web page metadata.-->
  <title>Communicating with Data</title>
  <meta name="author" content="Emi Tanaka">
  <link rel="stylesheet" href="css/styles.css">
</head>

<body>
<!--This section contains what you want to display on your web page.-->
<h1>I'm a first level header</h1>
<p>This is a paragraph.</p>
</body>
</html>
```

HTML Syntax



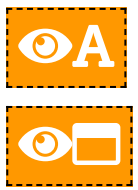
```
<span style="color:blue;">Author content</span>
```

Author content

The breakdown of this syntax:

start tag:	`<span style="color:blue;">Author content</span>`
end tag:	`<span style="color:blue;">Author content</span>`
content:	`<span style="color:blue;">Author content</span>`
element name:	`<span style="color:blue;">Author content</span>`
attribute:	`<span style="color:blue;">Author content</span>`
attribute name:	`<span style="color:blue;">Author content</span>`
attribute value:	`<span style="color:blue;">Author content</span>`

Not all HTML tags have an end tag:



```
<img height="40px" src="https://tinyurl.com/rlogo-svg">
```

Some HTML elements

block element:	`<div>content</div>`
inline element:	`<span>content</span>`
paragraph:	`<p>content</p>`
header level 1:	`<h1>content</h1>`
header level 2:	`<h2>content</h2>` (note: only up to 6 levels)
italic:	`<i>content</i>`
emphasised text:	`<em>content</em>`
bold:	`<b>content</b>`
strong importance:	`<strong>content</strong>`
link:	`<a href="https://cwd.numbat.space/">content</a>`
insert new line:	` `
unordered list:	`<ul>`

```
<li>item 1</li>
<li>item 2</li>
</ul>
```

How these are rendered to the browser depends on the browser default style values, style attribute or CSS...

Cascading Style Sheet (CSS)

- CSS files have the extension `.css` and styles also XHTML, plain XML, SVG and XUL.
- There are 3 ways to style elements in HTML:
  - **inline** by using the `style` attribute inside HTML start tag:

```
<h1 style="color:blue;">Blue Header</h1>
```

  - **externally** by using the `<link>` element:

```
<link rel="stylesheet" href="styles.css">
```

  - **internally** by defining within `<style>` element:

```
<style type="text/css">
h1 { color: blue; }
</style>
```

- By convention, the `<style>` and `<link>` elements tend to go into the `<head>` section of the HTML document.

CSS Syntax

```
<style type="text/css">
 h1 { color: blue; }
</style>
<h1>This is a header</h1>
```

This is a header

The breakdown of the CSS syntax:

selector: `h1 { color: blue; }`

property: `h1 { color: blue; }`

property name: `h1 { color: blue; }`

property value: `h1 { color: blue; }`

You may have multiple properties for a single selector:

```
h1 {
  color: blue;
  font-size: 16pt;
}
```

Some CSS properties

```
<div> Sample text </div>
```

background color:	`div { background-color: yellow; }`	Sample text
text color:	`div { color: purple; }`	Sample text
border:	`div { border: 1px dashed brown; }`	Sample text
left border only:	`div { border-left: 10px solid pink; }`	Sample text
text size:	`div { font-size: 10pt; }`	Sample text
padding:	`div { background-color: yellow; padding: 10px; }`	Sample text
margin:	`div { background-color: yellow; margin: 10px; }`	Sample text
horizontally center text:	`div { background-color: yellow; padding-top: 20px; text-align: center; }`	Sample text
font family:	`div { font-family: Marker Felt, times; }`	Sample text
strike:	`div { text-decoration: line-through; }`	~~Sample text~~
underline:	`div { text-decoration: underline; }`	Sample text
opacity:	`div { opacity: 0.3 }`	Sample text

CSS Selector

⚠ Unlike `class`, you can only have one `id` value and must be unique in the whole HTML document.

	ents
	> elements
`div, p`	selects all `<div>` and `<p>` elements
`div p`	selects all `<p>` within `<div>`
`div > p`	selects all `<p>` one level deep in `<div>`
`div + p`	selects all `<p>` immediately after a `<div>`
`div ~ p`	selects all `<p>` preceded by a `<div>`
`.classname`	selects all elements with the attribute `class="classname"`.
`.c1.c2`	selects all elements with *both* `c1` and `c2` within its class attribute.
`.c1 .c2`	selects all elements with class `c2` that is a descendant of an element with class `c1`.
`#p1`	selects all elements with the attribute `id="p1"`.

```
<body>
<h1>This is a sample html</h1>
 <blockquote>
 <p>Maybe stories are just data with a soul.</p>
 <footer>—Brene Brown</footer>
 </blockquote>

 <div id="p1" class="parent">
 Hmm
 <p>Hi!</p>
 How are you?
 <div class="child nice">
   <p>Hello!</p>
 </div>
 </div>

 <p>Hous

 lt;div
 <p>Hi!</p>
 &<blockquote class="child rebel">
```

You cannot use CSS to select a parent or ancestor. You will need to use JS...

JS JavaScript (JS) LIVE DEMO

- JS is a programming language (not to be confused with Java) and enable interactive components in HTML documents.
- You can insert JS into a HTML document in two ways:
  - **internally** by defining within `<script>` element:

```
<script>
document.getElementById("p1").innerHTML = "content";
</script>
```

  - **externally** by using the `src` attribute to refer to the external file:

```
<script src="myjava.js"></script>
```

- You are *not* expected to be able to do any JS in this course.

Inserting CSS in Rmd documents

- There is a `css` engine:

````
```{css, echo = FALSE}
h1 { color: red; }
```
````

- This inserts the following output into the document:

```
<style type="text/css">
h1 { color: red; }
</style>
```

- If the output is a HTML document then the defined styles will apply to the output document.

- If you have an external file, say `styles.css`, that you define the styles, then most HTML outputs will support this with YAML argument `css`

```
output:
  html_document:
    css: ["styles.css"]
```

or say

```
output:
  bookdown::html_document2:
    css: ["styles.css", "custom.css"]
```

Inserting JS into Rmd documents

- There is a `js` engine:

````
```{js, echo = FALSE}
document.getElementById("p1").innerHTML = "content";
```
````

- This inserts the following output into the document:

```
<script type="text/javascript">
document.getElementById("p1").innerHTML = "content";
</script>
```

- If you have an external file, say `myjava.js`, then you can dire insert this in the body of the R file as:

```
<script src="myjava.js"></script
```

- If you need to insert at a spec location within the document you can use `includes`:

```
output:
  html_document:
    includes:
      in_header: ["header.html"]
      before_body: ["before_body.ht
      after_body: ["after_body.html
```

Community Web Enhancements

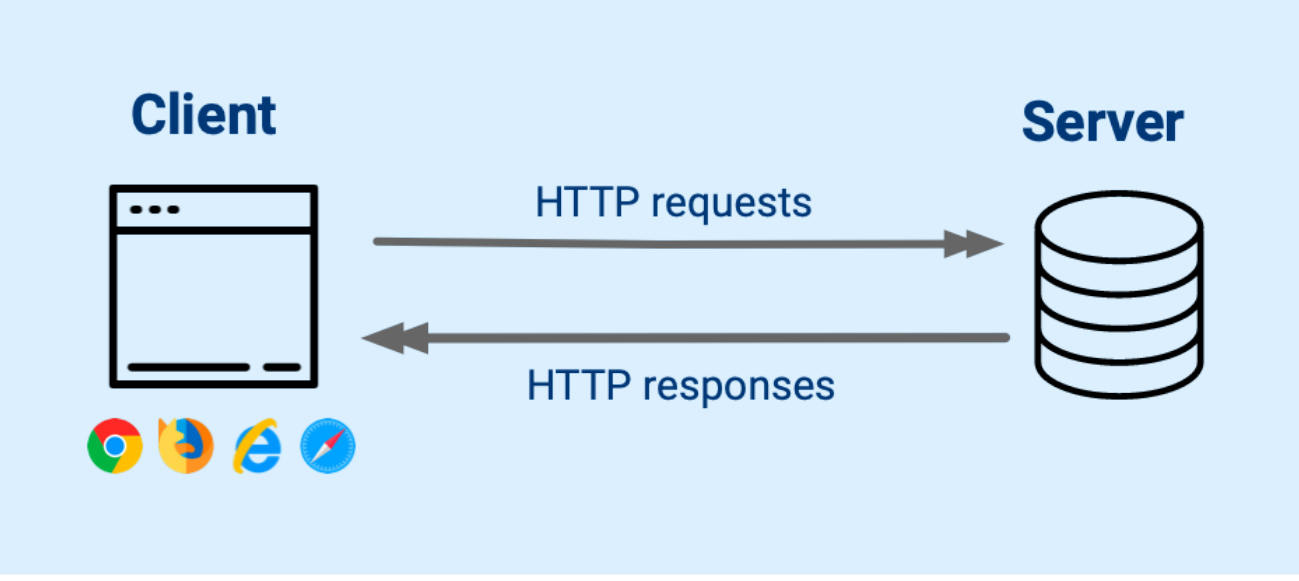
- Bootstrap is a free open-source CSS and JS that is widely adopted. Included in the default `rmarkdown::html_document`.
- jQuery is a widely used JS library for object selection and manipulation. Included in the default `rmarkdown::html_document`.
- MathJax is a JS library for displaying mathematics.

  Included in the default `rmarkdown::html_document`.
- Font Awesome inserts icons to web using their CSS.

  Sometimes included in certain Rmd HTML documents.
- animate.css is a large collection of CSS animations (powering the bouncing animation for the top icon)
- D3.js is one of the most popular JS library for interactive data visualisation.

☁️ Communication in the Web

- Hypertext Transfer Protocol (**HTTP**) functions as the communicator in the Web
- HTTPS is the secure version of HTTP where communications are encrypted



- ⚠️ Different clients may work differently! E.g. Internet Explorer and Chrome may render the same web page differently.

Web Standard

- There are 3 major groups that govern the standard for the Web:
  - World Wide Web Consortium (**W3C**) formed in 1994 maintains the CSS specifications
  - Web Hypertext Application Technology Working Group (**WHATWG**) formed in 2004 and is the publisher of the HTML and DOM standards
  - **TC39 technical committee** of Ecma International, renamed from European Computer Manufacturers Association (ECMA) in 1994, maintains the standards for JS
- These groups consist of Mozilla, Apple, Google, Microsoft and other invited members.

https://developer.mozilla.org/

- Documentation of web standards maintained by the community.
- Includes status of use:

> **Obsolete**
>
> This feature is obsolete. Although it may still work in some browsers, its use is discouraged since it could be removed at any time. Try to avoid using it.

- As well as its compatibility with web clients:

Writing HTML LIVE DEMO

You can play at https://htmltidy.net/

 Interactive Cheatsheets LIVE DEMO

HTML Cheatsheet

https://htmlcheatsheet.com/

CSS Cheatsheet

https://htmlcheatsheet.com/css/

JS Cheatsheet

https://htmlcheatsheet.com/js/

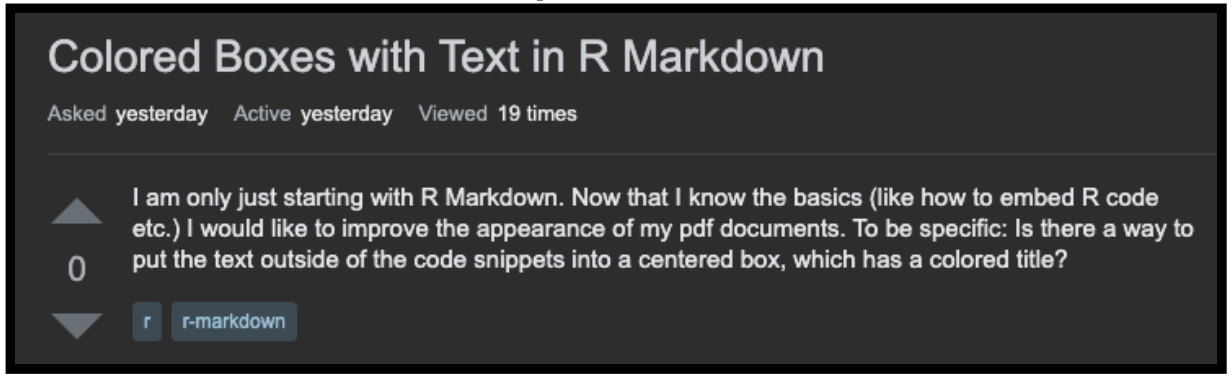
🔍 Inspect Element LIVE DEMO

Below GIF shows interactive use of Inspect Element (or called Inspect) available from the menu in most web browsers when you right click on the web page: (note: this is for inspection and doesn't actual make changes to the files)

Communicating your problem

SOS Asking for help Part 1/2

- Asking for help, requires you to communicate what your problem is to another party.
- How you communicate your problem, can assist you greatly in getting the answer to your problem.
- What do you think about the question below?



- What is the author's problem?
- This is a real question from stackoverflow so feel free to answer it there if you know the answer.

Asking for help Part 2/2

- How about the question on the right?
- This is also a question from stackoveflow.
- What makes it *hard* for people to answer the question?
- What would make it *easier* for people to answer the question?
- In this example, the author provides the data but it requires work for others to read this data in.

☝ Tips for communicating your problem and getting help

- If you want to receive help, you generally want to **make it easier for others to see your problem and what you expect**.
- If you have data, is it in a form **easy for others to read the data** in? If your data is big, could you **cull your data** to the minimum amount needed to communicate your problem? If you cannot supply data due to privacy issues, etc, you can still supply a dummy data.
- If your problem is code related, could you **cull your code** to the minimum amount required to reproduce your problem? Did you include code to load relevant libraries or packages?
- If your question is software related, did you supply the **code to reproduce your example**, your **software version(s)** and your **operating system**?

Session Information

You can easily get the session information in R using `sessioninfo::session_info()`. Scroll to see the packages used to make these slides.

```
sessioninfo::session_info()
```

```
## ─ Session info ─────────────────────────────────────────────────────────────
##  setting  value
##  version  R version 4.0.1 (2020-06-06)
##  os       macOS Catalina 10.15.6
##  system   x86_64, darwin17.0
##  ui       X11
##  language (EN)
##  collate  en_AU.UTF-8
##  ctype    en_AU.UTF-8
##  tz       Australia/Melbourne
##  date     2020-08-11
##
## ─ Packages ─────────────────────────────────────────────────────────────────
```

 Reproducible Example with `reprex` LIVE DEMO

- Copy your **minimum reproducible example** then run

```
reprex::reprex(si = TRUE)
```

- Note that `si` argument is deprecated in favour of `session_info` so in future you need to use `session_info = TRUE` instead of `si = TRUE`.
- Once you run the above command, your clipboard contains the formatted code and output for you to paste into places like GitHub issues, Stackoverflow and forums powered by Discourse, e.g. RStudio Community.
- For general code questions, I suggest that you post to the community forums rather than Moodle.

Introduction to Data Technologies

What are these?

Do you have a device to read the information from these?

Image by George Chernilevsky taken from Wikipedia.

💾 Data Storage: Computer memory

- The **bit** (or binary digit) is the smallest unit of data in computer memory.
- **A single bit contains a boolean value**, commonly represented as 0/1, true/false, yes/no, and on/off.
- A collection of 8 bits is called a **byte**.
- A **file** points to a block of computer memory.
- A **file format** signals a way to interpret the information stored in computer memory.
- ⚠ **Backup**: if the hardware that you store your data is damaged, then you may lose your data.
- ⚠ **Prosperity**: the hardware you use to store your data may make it hard to extract information from in the future.

📁 File formats: Plain text formats Part 1/2

- The simplest way to store data is in a plain text format.
- Some **metadata** may be at the beginning of the file.
- Plain text format typically arrange data in rows where each row stores values of variables corresponding to that observational unit.
- **Delimited formats** have values separated by a special character, called **delimiter**. The most common types are **csv** and **tsv**.

comma-separated values

```
car, speed, dist
1, 4, 2
2, 4, 10
3, 7, 4
```

tab-separated values

```
car   speed   dist
1   4   2
2   4   10
3   7   4
```

delimiter is ;

```
car;speed;dist
1;4;2
2;4;10
3;7;4
```

File formats: Plain text formats Part 2/2

- **Fixed-width formats** have values at fixed positions within every row.
- E.g., below we are told that *variable 1* at positions 1 to 10, *variable 2* at positions 11 to 18, *variable 3* at positions 19 to 22, *variable 4* at position 23. What is the values of variable 3 and 4 for the 4th observational unit?

```
2020/01/01 Sunny   2  5
2020/01/02 Cloudy  10 3
2020/01/03 Raining 10 2
2020/01/04 Raining 1002
```

- White spaces are often trimmed for delimited formats and fixed-width formats.

File formats: Binary formats

- A computer can interpret a byte as a **binary number**, e.g.
  - `00000001` represents $2^0 = 1$ and
  - `00000011` represents $2^1 + 2^0 = 3$.
- With this a single byte can store integers from 0 to 255.
- In a plain text format, each digit is stored by a byte so a value like 23 requires 2 bytes; one to store 2 and the other to store 3.
- A binary representation requires only one byte as `00010111` $(2^4 + 2^2 + 2^1 + 2^0 = 23)$.
- In practice, 4 bytes are often used to store integers with one bit for the sign that allows to store a value from -2,147,483,647 to 2,147,483,647 $(\pm 2^{31} - 1)$.

📁 File formats: Binary formats in R

- In R, every element in a vector of integers (written by appending an integer with a L, e.g. 1L) occupies 4 bytes and an every element in a numerical vector occupies 8 bytes. See also Paul Murrell's notes.
- Files with extension `.RData` or its shorthand form `.rda` is a binary format designed for storing selected objects in the workspace that can be loaded back into R.

```
a <- 1L:3L; b <- c(4, 5.2, 3)
save(list = c("a", "b"), file = "data.rda") # to save to binary format
load("data.rda") # to load into R
```

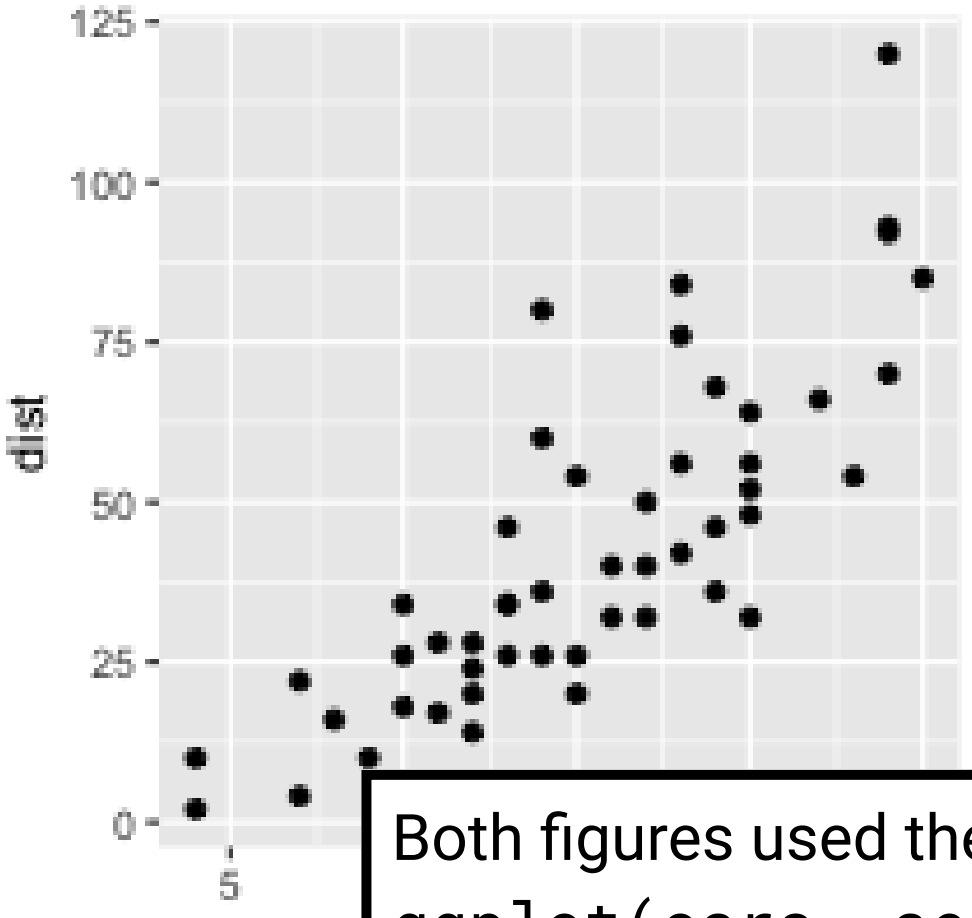
- The `.rds` files are also binary format designed for usage with R but stores only a single object.

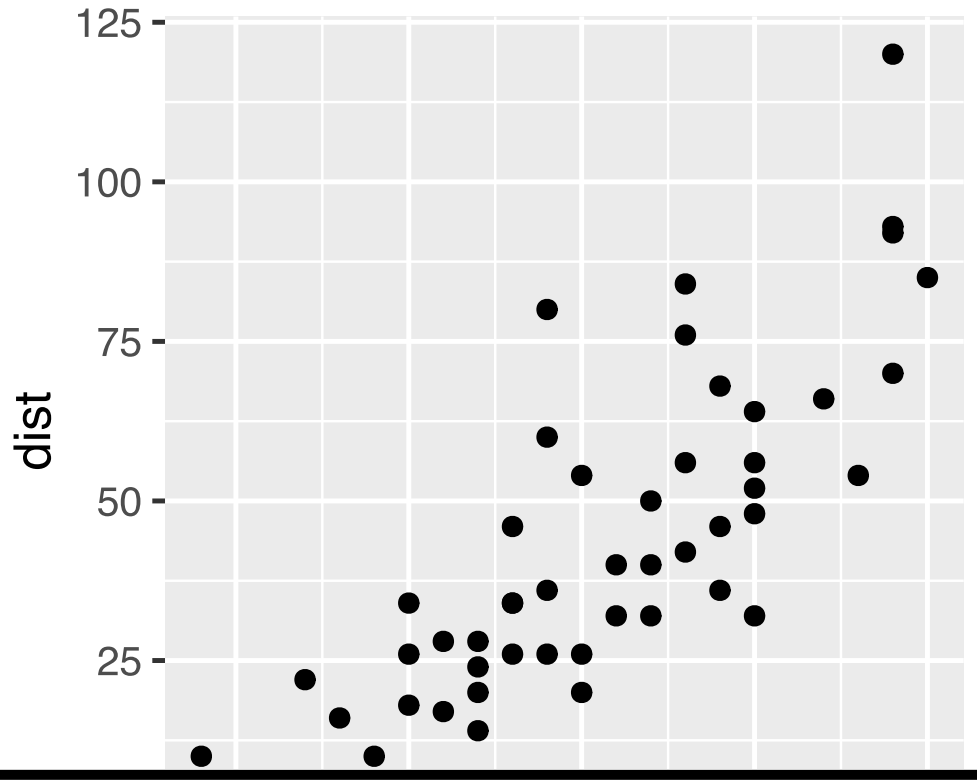
```
saveRDS(a, file = "a.rds"); a <- readRDS("a.rds")
```

📁 File formats: Proprietary formats

- Some data are stored in **proprietary formats** which means that the data are stored in a particular encoding scheme, designed so that decoding and reading the data is accomplished by particular software or hardware.
- `.rda` and `.rds` are some examples, but common ones include `.xlsx` for Microsoft Excel, `.sav` for SPSS, `.sas7bdat` for SAS, and `.dta` for Stata.
- You can use R packages, like `foreign`, `readxl`, and `haven`, to read these data formats into R.

💭 What's wrong with this figure?





Both figures used the same code below to produce the figure:

```
ggplot(cars, aes(speed, dist)) + geom_point()
```

but I saved one to a **png** file and the other as a **svg** file.

 Communicating Figures

- There are two main formats for graphics:
  - **Raster formats** contain description of each pixel. Common formats are:
    - `jpg` (or `jpeg`) uses a lossy data compression that results in some loss of information but usually a small file size.
    - `png` uses a lossless data compression and works well if the image has uniform colors.
  - **Vector formats** contain a geometric description and hence render smoothly at any display size. Common formats include `svg`, `pdf` and `eps`. E.g. svg and its source.
- A vector format scales well to any display size, however, the file size may become prohibitively big when there are many geometric objects (i.e. displaying many data).

That's it!

Lecturer: Emi Tanaka

Department of Econometrics and Business Statistics

ETC5523.Clayton-x@monash.edu