MEDB 5505, Module01

2025-01-23

Topics to be covered

- What you will learn
  - History of R
  - Installing R
  - Objects in R
  - Anatomy of a small R program
  - Live demonstration
  - Good programming practices
  - Your programming assignment

Special note

- This slide show was created using R.
  - Not complicated
  - But beyond scope of this class
  - Combination of several files in github folder
  - Slightly obsolete explanation on my blog
  - More current explanation on using quarto and reveal.js

Speaker notes

Okay. Let’s get started.

Before I talk about the histormy of R, let me menion one housekeeping item.

This presentation was developed using R. The process is not very complicated, but it is beyond the scope of this class.

I do want to note that I have switched recently from doing presentations using Powerpoint to presentations using reveal.js. I really like reveal.js, for a whole host of reasons that are hard to explain. The only difference that you might notice is that the presentations using reveal.js are html files. That means that you open them up using a web browser rather than Powerpoint.

If you are curious, you can look at the code that I used to develop this PowerPoint presentation. It’s all on my github site.

But don’t feel obligated to look at it. You will not be responsible for any of this in an introductory class.

It may seem a bit weird to have an R program that creates a presentation that talks about a different R program, but it works well for me.

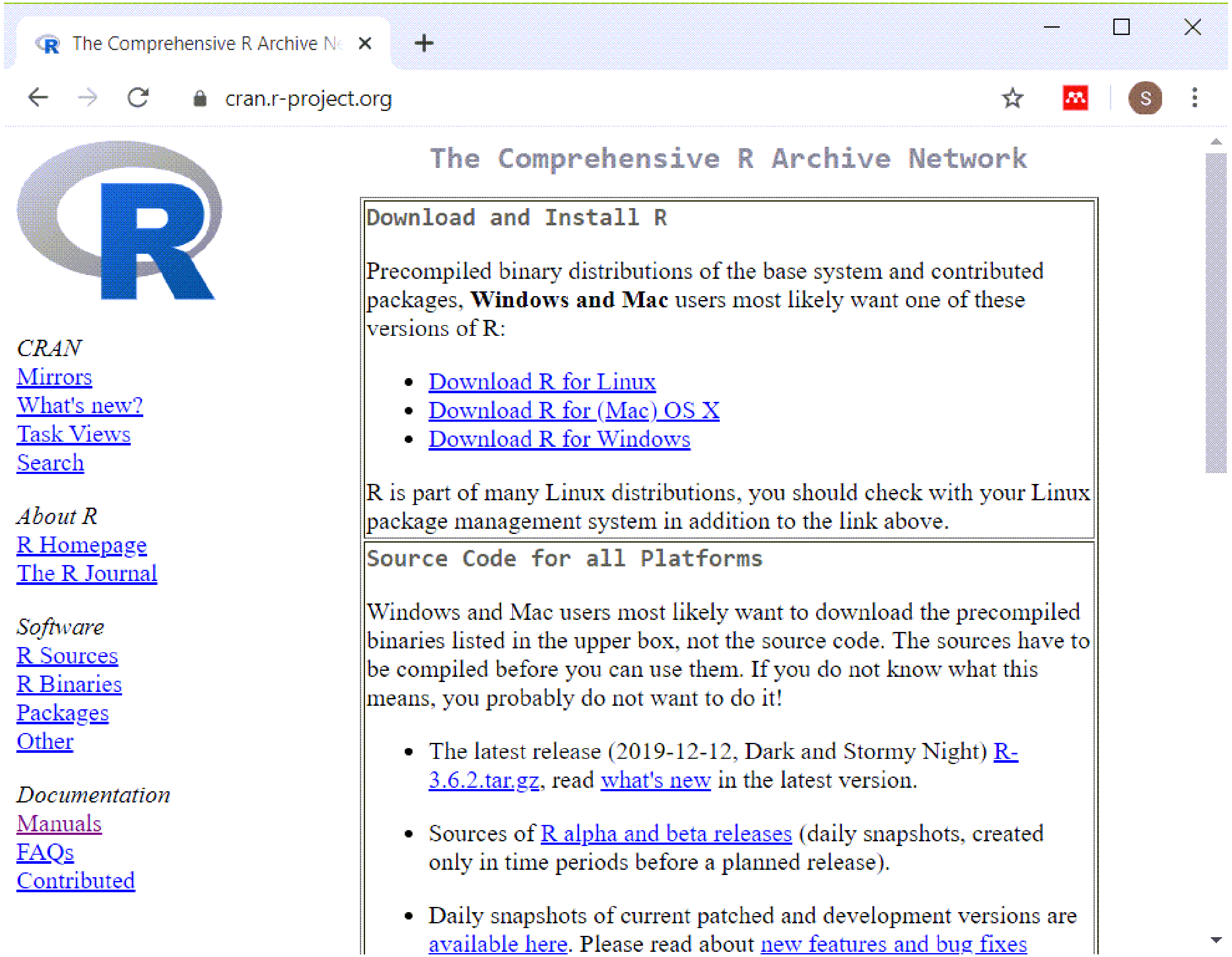
History of R

This portion of the talk will be found in the file history-of-r.pptx.

Break #1

- What you have learned
    - History of R
- What's coming next
    - Installing R

Installing R (https://cran.r-project.org/)



The Comprehensive R Archive Network

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

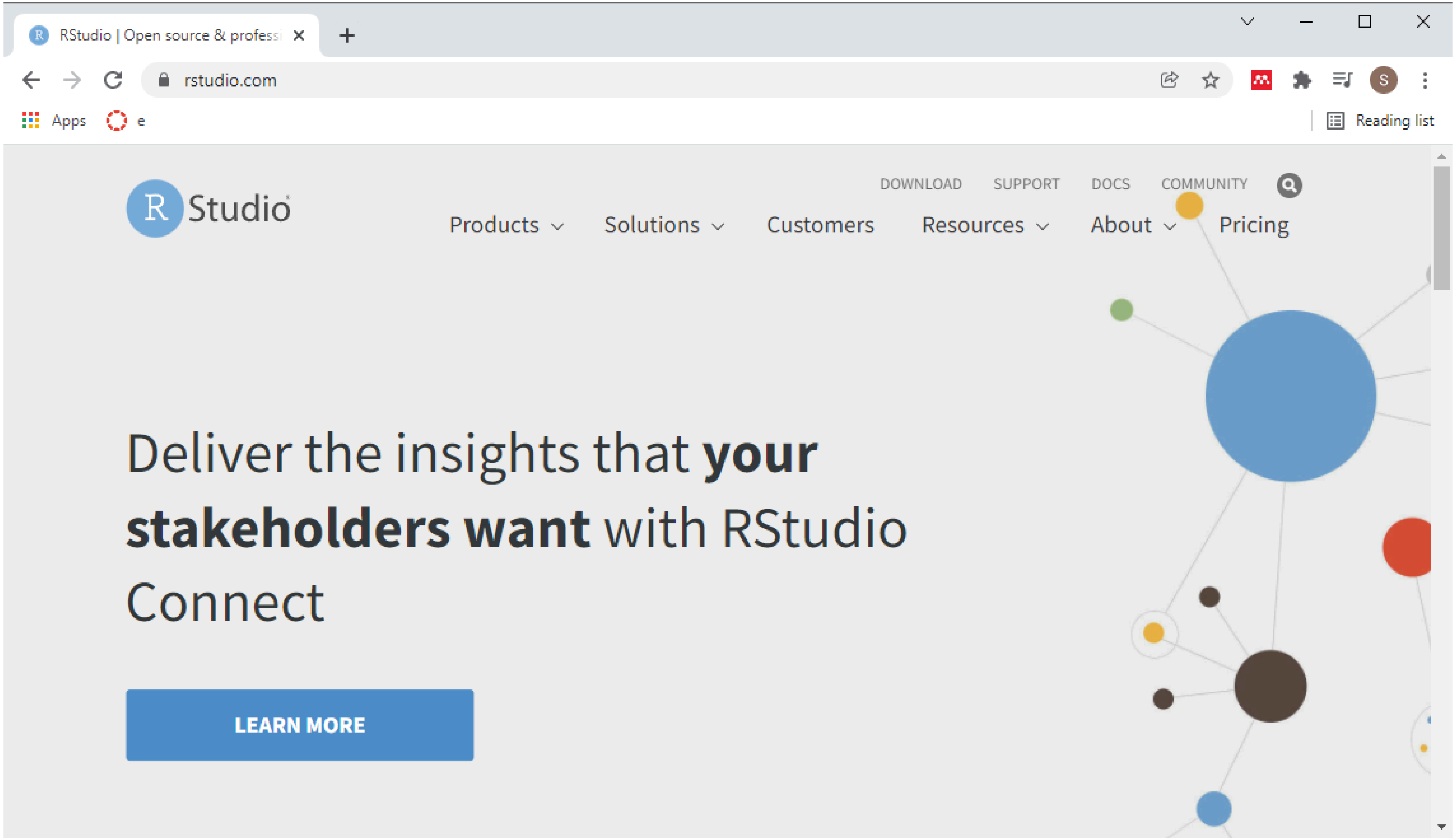
Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2019-12-12, Dark and Stormy Night) R-3.6.2.tar.gz, read what's new in the latest version.
- Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are available here. Please read about new features and bug fixes

Speaker notes

Here is the main page for CRAN. CRAN stands for the Comprehensive R Archive Network. This is where you can download a Linux, Mac, or Windows version of R. Ignore the source code. That is only needed for very advanced applications.

Installing RStudio (https://rstudio.com/)



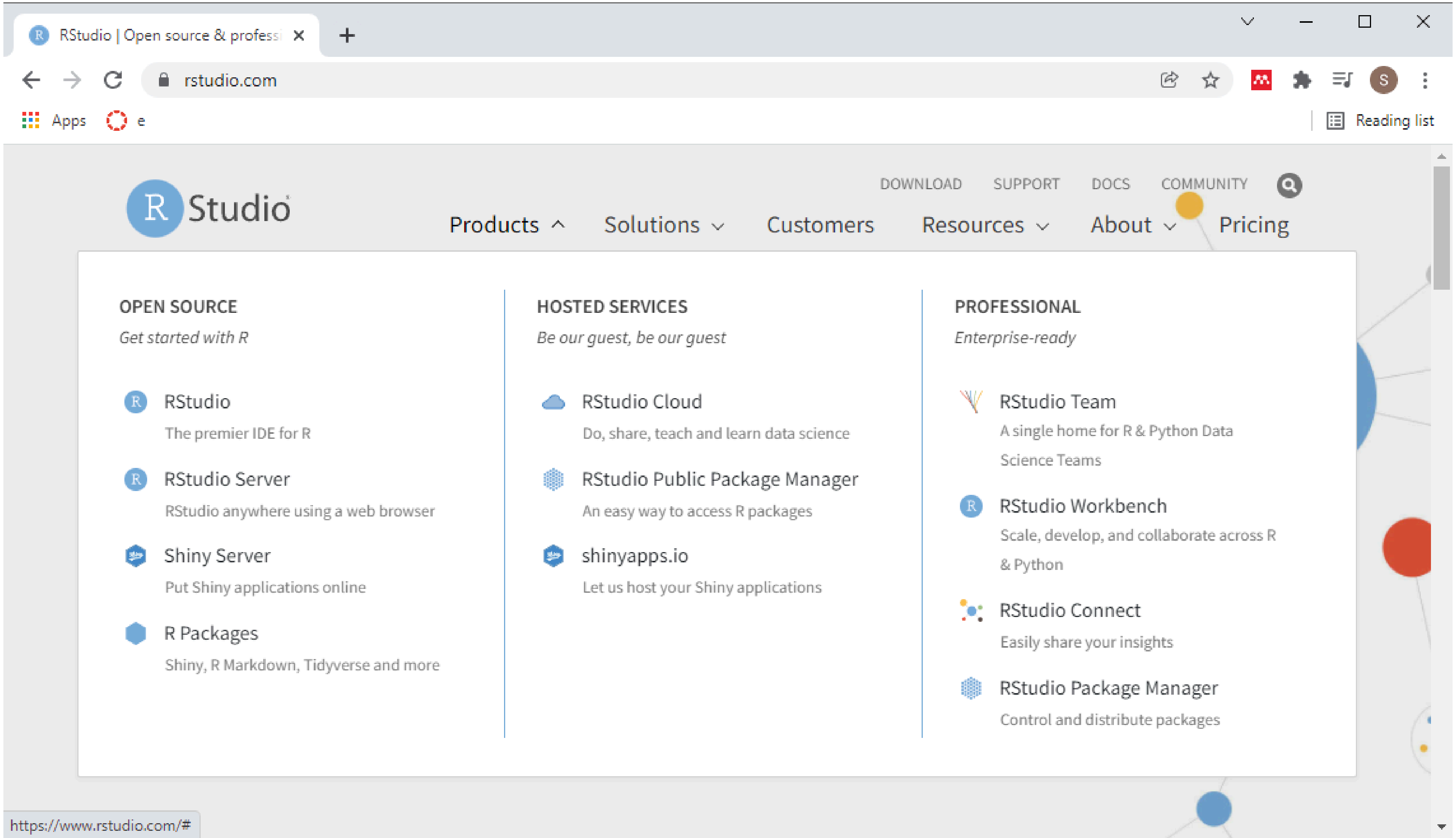
Deliver the insights that **your stakeholders want** with RStudio Connect

Speaker notes

The other thing, a very strong recommendation, is that you should install a package called R Studio. I'll probably talk more about R Studio in a separate handout or video.

Basically it's an integrated development environment that makes it very easy to work with the R programming language. Click on the products link.

Installing RStudio (https://rstudio.com/)



OPEN SOURCE

Get started with R

RStudio
The premier IDE for R

RStudio Server
RStudio anywhere using a web browser

Shiny Server
Put Shiny applications online

R Packages
Shiny, R Markdown, Tidyverse and more

HOSTED SERVICES

Be our guest, be our guest

RStudio Cloud
Do, share, teach and learn data science

RStudio Public Package Manager
An easy way to access R packages

shinyapps.io
Let us host your Shiny applications

PROFESSIONAL

Enterprise-ready

RStudio Team
A single home for R & Python Data Science Teams

RStudio Workbench
Scale, develop, and collaborate across R & Python

RStudio Connect
Easily share your insights

RStudio Package Manager
Control and distribute packages

https://www.rstudio.com/#

Speaker notes

The product you’re looking for is called RStudio. Not RStudio Server or any of the other products. These other products cost a fair amount of money. For a professional organization, they offer a wide range of advanced features. But we don’t need any of those advanced features for this class. So don’t bother with those other products.

Installing R and R Studio

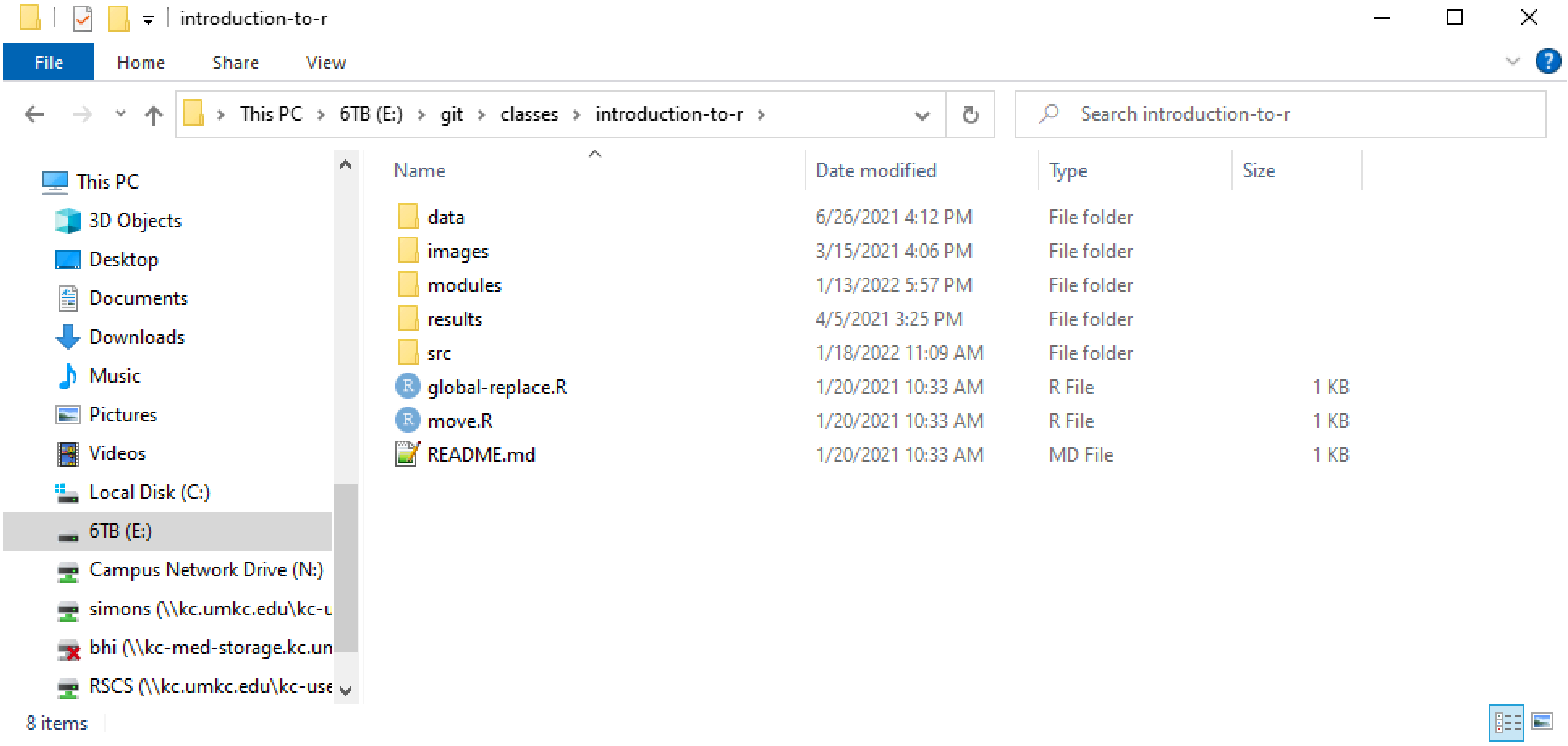
- R is required
- RStudio is strongly recommended
- Do not delay in getting this software installed
- Find me if you have ANY problems

Speaker notes

It should be very easy to install R and RStudio on your computer, but don’t wait. Sometimes installations can get hung up and you won’t be able to make any progress in this class without first getting the software installed.

If you have any problems at all with installation, see me right away. Computers are either our best friend or our worst enemy. It seems like the latter is especially true when you are installing new software.

Recommended directory structure



introduction-to-r
File
Home
Share
View
This PC › 6TB (E:) › git › classes › introduction-to-r ›
Search introduction-to-r
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
6TB (E:)
Campus Network Drive (N:)
simons (\\kc.umkc.edu\kc-u
bhi (\\kc-med-storage.kc.un
RSCS (\\kc.umkc.edu\kc-use
8 items

Name	Date modified	Type	Size
data	6/26/2021 4:12 PM	File folder	
images	3/15/2021 4:06 PM	File folder	
modules	1/13/2022 5:57 PM	File folder	
results	4/5/2021 3:25 PM	File folder	
src	1/18/2022 11:09 AM	File folder	
global-replace.R	1/20/2021 10:33 AM	R File	1 KB
move.R	1/20/2021 10:33 AM	R File	1 KB
README.md	1/20/2021 10:33 AM	MD File	1 KB

Screenshot of the directory structure I use

Speaker notes

Here is the directory structure that I use. You do not have to follow this structure, but it is recommended, not by me, but by the expert programmers at a group known as Software Carpentry. I’ll elaborate in greater detail about this later, but wanted to mention it now. If you are relatively new to programming, you want to start off using good programming practices. A standardized directory structure helps a lot with this especially if you are working with others.

“A place for everything, everything in its place”

- data, for raw/intermediate data files
- doc, for documentation
- images for graphs/illustrations
- results, for program output
- src, for program code
- Other folders as needed

Speaker notes

The quote at the top of the slide is an organizational principle espoused by Benjamin Franklin. If you've seen my office, you'll know that I am probably the last person to lecture you on organization. But I have found that a standardized directory structure has made my life a lot easier.

The data folder contains any raw data files. It's also where I put intermediate files, files that I create and save for later re-use. Some people put intermediate files in the results folder, and that's a fine alternative. Just be sure to be consistent about it.

The doc folder contains any documentation associated with the work. The IRB approved protocol, if I have one, will go here. If I am working with someone and they send me a paper that helps describe the type of data analysis they want, I put it in this folder. I also print key emails from the other team members to pdf format and store them here as well.

If my programs produce any graphs, I will store them in the images folder. I use screenshots of various web pages a lot in my work and I put those here as well.

I usually store program output in the results folder, but not always, for reasons I don't want to get into.

The program code goes in the src folder.

This directory structure has an extra folder, and that's always okay. This folder, modules, contains some of the information on homework assignments, quizzes, and so forth.

I adopted the structure about four years ago. I do not do a lot of collaborative research but I do find myself frequently revisiting a project six months or a year down the road. Having a common directory structure means that I can very quickly and rapidly identify the stuff that was done before and the stuff that needs to change for the new work.

I would strongly encourage you to set up a directory structure like this one.

Break #2

- What you have learned
    - Installing R
- What's coming next
    - Objects in R

Introduction

This is a very brief introduction to the basic objects in R.

```r
R.version.string
```

```
[1] "R version 4.4.1 (2024-06-14 ucrt)"
```

```r
Sys.Date()
```

```
[1] "2025-01-27"
```

Speaker notes

I want to talk about some of the technical underpinnings in R. This is way too technical and way too early to present. I decided to give this lecture here anyway because if you want to become a seasoned R programmer who can write programs from scratch, you need to understand these technical details. For now, if this portion of the lecture seems a bit overwhelming, focus just on the two big things: functions and tibbles. The rest will gradually become more obvious as you see more examples in this class.

Assignment and naming conventions

- Use <- and -> to assign objects to a name
  - Avoid using = for an assignment
- Rules for names
  - Combination of letters and numbers
  - No spaces
  - No symbols other than underscore (_) and dot (.)
  - Cannot start with a number
    - a1 is okay, but 1a is not

Speaker notes

I use the term “object” as a generic name for a variety of things that you need in R, functions, scalars, vectors, etc. You can assign or store objects using the <- or -> symbols. Now R does allow you to use the equal sign to assign objects and this is the common way to assign in other languages, this is discouraged in R. I’m not sure why, but you will see the equal sign used only inside functions.

When you assign an object, you specify a name. That name has to be combination of letters and numbers.

You can’t put spaces in the middle of a name. R will get confused and think of it as two names rather than one.

You can’t use most symbols. A dash inside a variable name for example will cause confusion because R will think that you want to perform a subtraction.

The two symbols that you can use are the underscore and the dot. In fact, you will see these two symbols a lot both in your code and in the various functions that you will use in R. Both the underscore and the dot represent ways to mimic

Recommendations for names, 1

- Avoid generic names (x,y or v1, v2, v3)
- Don't run two words together (writersexchange)
- Use short words separated by underscores (writers_exchange)
  - All lower case
  - No abbreviations
  - Avoid names identical with common functions

Speaker notes

Although you have great latitude in names, I have a few recommendations that will help make your code easier to read and maintain.

First, avoid short generic names. When you assign names of x and y early in the program, you will get confused halfway through and think of x as y or y as x.

Instead use one or two words to describe what you are storing under that name. You can't put spaces between two words, but don't just run the two words together.

There's a story about a group, The Writer's Exchange, that wanted to start a website. So they called it www.writersexchange.com. Then someone notices that you could read it not as writers exchange, but as writer sex change. Not exactly what they wanted.

You should use the underscore to separate two words. So it would be writers underscore exchange.

Use all lower case. Upper case is harder to read. It seems contradictory because DON'T ALL WARNINGS COME IN UPPER CASE? It turns out that upper case is harder to read because all the letters are the same height. With lower case, you have some letters that rise above (such as f and t) and some that stretch out below (such as j and y). These ascenders and descenders provide an extra visual cue.

Now many programmers will avoid all uppercase, but will use an initial capital letter. That's okay, but I find using all lower case makes things easier to remember. Now for headings in tables and titles on graphs, I will still use initial capitals, but only then.

I've also learned to avoid abbreviations. I can never remember whether the abbreviation for height is hgt or ht or even just the single letter h. There's nothing worse than having half of your calculations using hgt and the other half using ht. It's a bit more typing, I'll admit, but I've taken to spelling it out as h-e-i-g-h-t. RStudio and many other integrated programming environments have an autocomplete function that can save you time.

Now avoid using common function names like list and matrix. R will allow this, but your code will be easily misinterpreted by others.

Recommendation for names, 2

- Common alternatives to the underscore separator
  - Short words separated by dots (writers.exchage)
  - Start each word with capital (WritersExchange)
  - **Use dash instead of underscore for file names, chunk names** (writers-exchange)
- What should YOU use
  - Now: Anything is fine, just be consistent
  - Later: See if there is an official or unofficial company standard

Speaker notes

You will see many programmers who use the dot instead of the underscore to separate two or more words in a variable name. To use the earlier example, it would be writers dot exchange. This is fine, but will probably confuse programmers who have first learned how to program in Python, where the dot has a special meaning.

Another common approach is to start each word with a capital letter. Capital W writers capital E exchange.

Now I really like the look of a dash. I know I can’t use it for variable names, but I still use it for file names and chunk names. I know this is inconsistent, but it’s evolved into a habit that I can’t break.

You need to make a choice. For now, any standard is fine. Just be consistent. Later, when you graduate and find yourself in your first job, see if there is a written standard. If not a written standard, then maybe an unofficial standard, which you can infer by looking at what other programmers do.

Functions

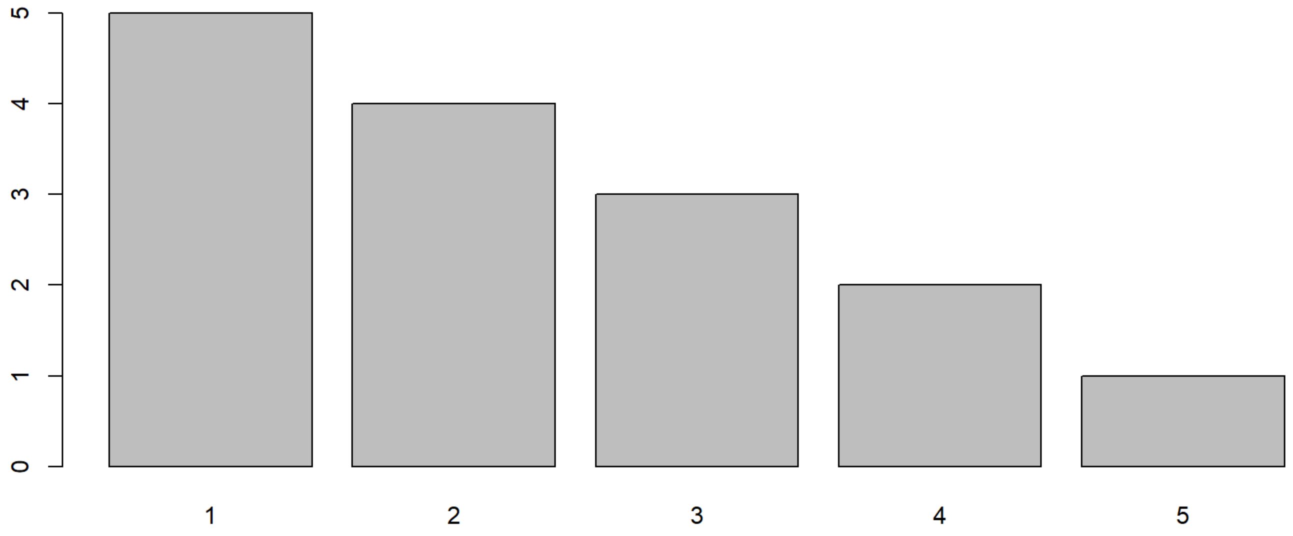
```r
sqrt(3)
```

```
[1] 1.732051
```

```r
sqrt(1:5)
```

```
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
```

```r
barplot(height=5:1, names=1:5)
```



Speaker notes

Functions receive input from one or more objects and produce an object in return. The input values found inside the parentheses of a function are called its arguments.

Functions can be as simple as computing a square root. Many functions are vectorized, meaning that they will compute a function for every element in a vector. Sometimes a function may produce more than just a single object as output. Here’s an example of a function that produces a graph.

Nested functions and pipes

```r
x <- 0.9
y <- asin(sqrt(x))
y
```

```
[1] 1.249046
```

```r
x |>
  sqrt() |>
  asin() -> y
y
```

```
[1] 1.249046
```

Speaker notes

You can nest one function inside another. So asin(sqrt(x)) feeds the variable x into the square root function and then takes the output of the square root function and feeds it into the arcsin function. This result is stored in y.

You can accomplish the same thing using the pipe symbol. Take the variable x and pipe it into the square root function. Then take the output from the square root function and pipe it into the arcsin function. Store the result in y.

While the pipe symbol seems a bit odd in this simple example, you will see examples fairly soon, where you might nest a half dozen functions. The vertical format that pipes produce make the code easier to read. Pipes also make it easier to fix errors.

Named arguments in functions

```
qnorm(p=0.99, mean=100, sd=15)
```

```
[1] 134.8952
```

```
qnorm(0.99, 100, 15)
```

```
[1] 134.8952
```

```
qnorm(0.99)
```

```
[1] 2.326348
```

Speaker notes

Many functions include names for the inputs. The first example shows how to compute the 0.99 quantile of a normal distribution with a mean of 100 and a standard deviation of 15.

Does anyone know what famous measurement has a mean of 100 and a standard deviation of 15? Answer: IQ scores. The 0.99 quantile represents the IQ of a person who is smarter than 99% of the population.

The use of these names is optional in most cases, but it requires you to memorize the proper order.

Many functions will have default values. If you do not specify the mean and sd arguments in the qnorm function, it will use mean=0 and sd=1.

Scalars

```r
scalar_example_1 <- 3
scalar_example_1
```

```
[1] 3
```

```r
scalar_example_2 <- "R"
scalar_example_2
```

```
[1] "R"
```

```r
scalar_example_3 <- "3"
scalar_example_3
```

```
[1] "3"
```

Speaker notes

Single values in R are called scalars. The most common scalars are numbers and strings. Strings are always surrounded by quote marks.

There are other scalars that are important, such as dates

Vectors

```r
vector_example_1 <- c(1, 2, 3)
vector_example_1
```

```
[1] 1 2 3
```

```r
vector_example_2 <- c("a", "b", "c")
vector_example_2
```

```
[1] "a" "b" "c"
```

```r
vector_example_3 <- c("a", 2)
vector_example_3
```

```
[1] "a" "2"
```

Speaker notes

A combination of number or string scalars is called a vector. Use the c function to create vectors.

If you try to mix different scalar types, R will convert them to a common format.

Naming vectors

```r
my_degrees <- c(
  BA=1977,
  MS=1978,
  PhD=1982)
my_degrees
```

```
  BA   MS  PhD 
1977 1978 1982 
```

```r
my_name <- c(
  first_name="Stephen",
  middle_initial="D",
  last_name="Simon")
my_name
```

```
    first_name middle_initial      last_name 
     "Stephen"            "D"        "Simon" 
```

Speaker notes

You can name the individual elements of a vector. This is not done that often, but the same naming concept will be very important for more complex objects.

Matrices using cbind and rbind functions

```r
matrix_example_1 <-
  cbind(
    c(1, 2, 3),
    c(4, 5, 6))
matrix_example_1
```

```
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

```r
matrix_example_2 <-
  rbind(
    c(1, 2, 3),
    c(4, 5, 6))
matrix_example_2
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Speaker notes

A matrix is a combination of vectors, with the important restriction that all the vectors have to be the same type. You can’t mix strings and numbers for example.

You can create a matrix using the cbind or rbind functions.

Matrices using the matrix function

```r
matrix_example_3 <-
  matrix(
    c(1, 2, 3, 4, 5, 6),
    nrow=2,
    ncol=3,
    byrow=TRUE)
matrix_example_3
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Speaker notes

The matrix function will reshape a vector into a matrix. There are lots of options in the matrix function that control the shape of the matrix and the order in which it is filled.

Lists

```r
list_example_1 <-
  list(
    scalar_example_1,
    vector_example_2,
    matrix_example_3)
list_example_1
```

```
[[1]]
[1] 3

[[2]]
[1] "a" "b" "c"

[[3]]
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Speaker notes

A list is a combination of objects of different types. It can include scalars, vectors, matrices, or other types of R objects that we have not seen yet. You can also include one list inside another list.

Lists using names

```r
list_example_2 <-
  list(
    name=my_name,
    degrees=my_degrees,
    age=64)
list_example_2
```

```
$name
    first_name middle_initial      last_name
     "Stephen"            "D"        "Simon"

$degrees
  BA   MS  PhD
1977 1978 1982

$age
[1] 64
```

Speaker notes

A list can contain virtually anything, so using names is almost always recommended.

Data frames

```r
data_frame_example_1 <-
  data.frame(
    vector_example_1,
    vector_example_2)
data_frame_example_1
```

```
  vector_example_1 vector_example_2
1                1                a
2                2                b
3                3                c
```

Speaker notes

A data frame is a combination of vectors. The vectors have to have the same length, but they can be different types.

Naming data frame columns

```r
data_frame_example_2 <-
  data.frame(
    c(1, 2, 3),
    c("a", "b", "c"))
data_frame_example_2
```

```
  c.1..2..3. c..a....b....c..
1          1                a
2          2                b
3          3                c
```

```r
data_frame_example_3 <-
  data.frame(
    small_numbers=c(1, 2, 3),
    early_letters =c("a", "b", "c"))
data_frame_example_3
```

```
  small_numbers early_letters
1             1             a
2             2             b
3             3             c
```

Speaker notes

R will always name the columns in a data frame using common sense rules. Sometimes this leads to poor choices (common sense isn’t always common sense). So it is strongly recommended that you specify names explicitly.

Tibbles

```r
library(tidyverse)

tibble_example_1 <-
  tibble(
    x=c(1, 2, 3),
    y=c("a", "b", "c"))
tibble_example_1
```

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

Speaker notes

A tibble is a slightly modified data frame designed by the tidyverse team. It behaves almost exactly like a data frame. There are other subtle differences in behavior, however, that reduce the risk of unexpected errors. There are a few times when these modifications do not work well with some of the older functions in R. But tibbles work very effectively with the functions of the tidyverse.

Vector or tibble?

```r
sample_vector <- 1:5
sample_vector
```

```
[1] 1 2 3 4 5
```

```r
sample_tibble <- tibble(sample_vector)
sample_tibble
```

```
# A tibble: 5 × 1
  sample_vector
          <int>
1             1
2             2
3             3
4             4
5             5
```

Speaker notes

One of the potential sources for confusion in R is when a single column of data is a vector and when it is a tibble. Note that a vector prints out horizontally and a tibble prints out vertically. Some functions need to see a single column of data as a vector, some need to see a single column of data as a tibble, and some can accept either. It is an unavoidable source of problems.

Break #3

- What you have learned
  - Objects in R
- What's coming next
  - Anatomy of a small R program

Anatomy of a small R program, overview

Speaker notes

This is a full listing of a small program written in Rmarkdown. The font is too small to read. Don’t worry, I just wanted you to see the full picture. I’ll look at small pieces of this program using a readable font size.

YAML header

```
---
title: "Illustrating the structure of an R program"
editor: source
format:
  html:
    embed-resources: true
execute:
  error: true
---
```

Speaker notes

All right, The first thing we have is the header. The header uses a structure called YAML (Y-A-M-L). YAML is a special type of up format that is used by lots of programmers. The header provides a bit of information about the about the program. The default option when you start up a new R Markdown file is to list, the title, author, date, and output format.

I find the word “date” by itself ambiguous, so I add the word “Created” to distinguish the creation date from the last modified date. It is not easy to track the last modified date in RStudio.

Quarto has options for creating web pages (html documents). You can also produce Word documents and PDF documents. I should warn you that creating PDF documents directly is a bit tricky. You need to have a program called LaTeX (pronounced lay-tech) installed first.

You can also produce Powerpoint files, which this is an example of.

The default is HTML and that’s a pretty good format. Quite honestly, I’ve got a lot of mileage out of HTML. It has some limitations compared to other formats, but it is so much simpler to create and maintain the code that produces HTML files.

First comment

```
This program was written by Steve Simon  and created on 2019-01-28 with a
major
revision on 2024-12-27. It is used to illustrate the structure of an R
program.
This program is in the public domain. You can use it any way that you please.
```

Speaker notes

Here’s a brief sentence written as open text. You will be required to provide open text comments on all your homework assignments. Often just a single sentence will do, but do get in the practice of interspersing open text comments throughout your code.

First code chunk

~~~
```{r}
#| label: setup
#| message: false
#| warning: false

R.version.string
Sys.Date()
library(tidyverse)
```
~~~

Speaker notes

In quarto, sections of R code are delineated with three backticks. The backtick is a leftward slanting single quote mark. For most keyboards, you will find the backtick key on the top row, just to the left of the “1” key.

The beginning of a chunk of R code has three backticks, followed by a left curly bracket, the letter “r” and then an optional name. End with a right curly bracket.

This program chunk includes two lines of code.

R.version.string displays the version of R that you are running. You should try to use a version of R at 4.0 or higher. This would only be an issue if you’ve been using R for more than a couple of years and have been lazy about upgrades. If you downloaded R at the beginning of this semester, you will have a version of R beyond 4.0.

The second line of code, Sys.Date(), displays the current date. Sometimes knowing when a program was run can help you decide if you need to run the program again (perhaps because the data has changed, or you are using a new version of R).

After this you see three more backticks, which tells R Markdown that the chunk of code has ended.

I recommend that you try to avoid long stretches of R code. Break it up into segments of no more than 20 lines of code, and often less than this is better. Working with your program in chunks and interspersing free text comments liberally makes it easier to maintain your code.

Second comment

```
Read data from the aids-cases text file. This file is described at

https://github.com/pmean/data/blob/main/files/aids-cases.yaml
```

Speaker notes

Here’s another brief comment. Note that you write an R Markdown a bit differently than most other programs. Most other programs place the comments inside of special delimiters. In SAS, for example, you start a comment with a slash and an asterisk and end it with an asterisk and a slash.

In R Markdown, the comments are not delimited. The code is delimited by three consecutive backticks.

Second code chunk

~~~
```{r}
#| label: read-text-file

aids_cases <- read_csv(
  file="../data/aids-cases.csv",
  col_types="nnn")
glimpse(aids_cases)
```
~~~

Speaker notes

Here’s some more R code. I named the chunk “read-rdata-file”. The load function reads information from a file named ../data.two-small-dataframes.RData.

I have to apologize here because Powerpoint splits the load function onto two consecutive lines, making it a bit harder to read.

The second line of code is the ls function. Just ls with a left and a right parenthesis. This function lists everything that you have either loaded or created within R. In this case, it is just the information that we got from the two-small-dataframes.RData file.

There are two objects, bump and fat. These are actually dataframes, the standard way that R stores data. If you look on my github site, you will find data dictionaries for these two dataframes.

The head function lists the first few lines of ant object in R. In this case, it will show the first six lines of the dataframe bump.

Third comment

```
This is a small dataset with only three variables. Now let's draw a line
graph.
```

Speaker notes

After the second chunk of R code, I add a couple of extra comments.

It’s very simple program but if you can get this program to run, you probably have smooth sailing for the rest of the class. I really believe that installing the software is often tricky. Getting the first program to run is often tricky. But once you get those going then you keep on modifying. Just make tiny changes to the programs you already have. It will either work or there will be an obvious error that you can fix.

The problem is getting the first program will run. Once you get one program that works, the rest is downhill.

I need to emphasize here very strongly if you’re having problems. Come and see me right away. Drop me an email. Try to visit. I do not have official office hours, but I will make appointments at any reasonable time for people to come and talk.

Third code chunk

~~~
```{r}
#| label: line-graph

aids_cases |>
  ggplot() +
    aes(yr, nsw) +
    geom_line()
```
~~~

Speaker notes

This chunk of code produces a simple line graph.

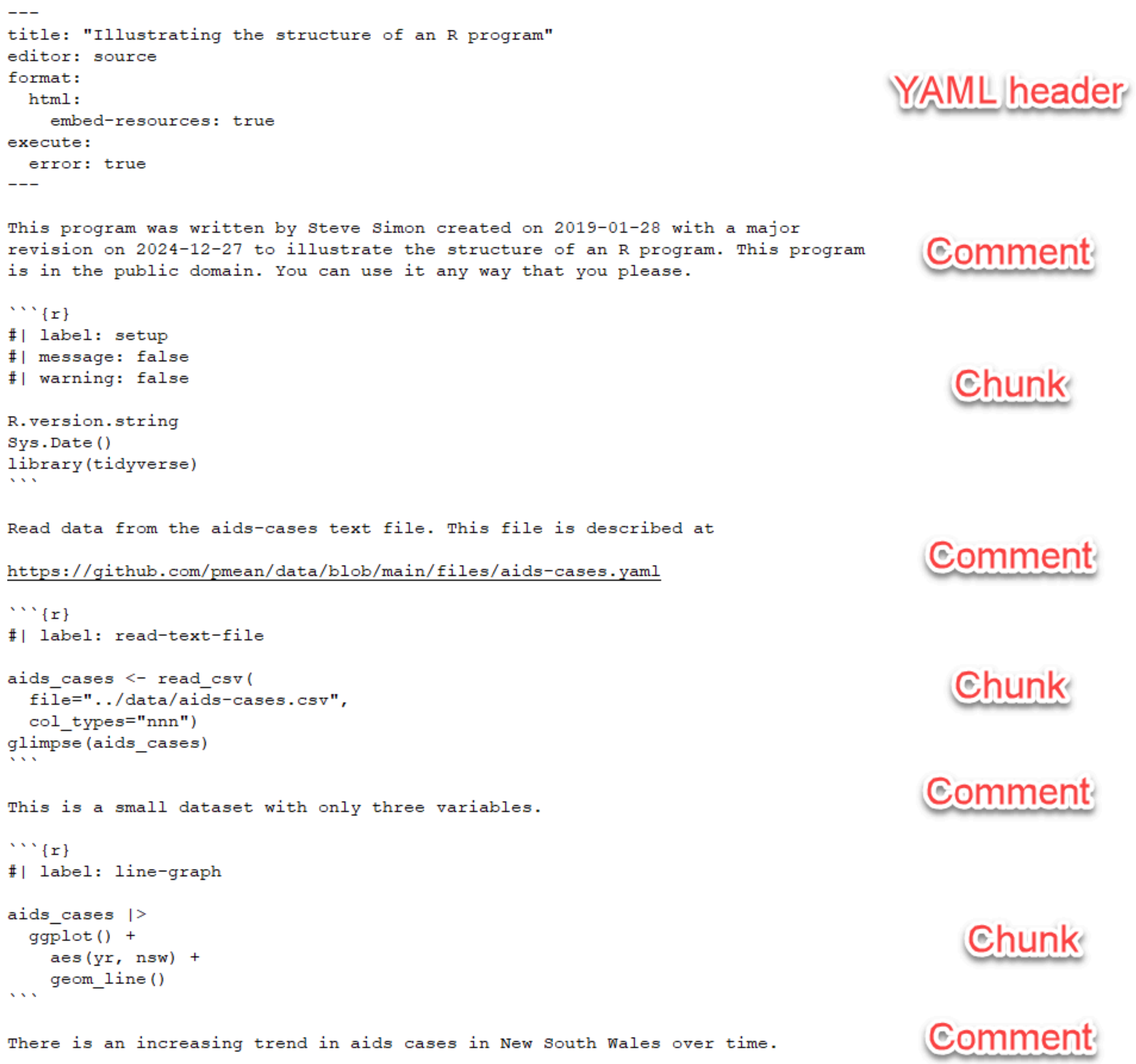
Fourth comment

```
There is an increasing trend in aids cases in New South Wales over time.
```

Speaker notes

The last comment provides a brief interpretation of the graph. You must provide interpretations for every graph and for almost all tables.

Anatomy of a small program, review



~~~
---
title: "Illustrating the structure of an R program"
editor: source
format:
  html:
    embed-resources: true
execute:
  error: true
---

This program was written by Steve Simon created on 2019-01-28 with a major
revision on 2024-12-27 to illustrate the structure of an R program. This program
is in the public domain. You can use it any way that you please.

```{r}
#| label: setup
#| message: false
#| warning: false

R.version.string
Sys.Date()
library(tidyverse)
~~~

Read data from the aids-cases text file. This file is described at

https://github.com/pmean/data/blob/main/files/aids-cases.yaml

```{r}
#| label: read-text-file

aids_cases <- read_csv(
  file="../data/aids-cases.csv",
  col_types="nnn")
glimpse(aids_cases)
```

This is a small dataset with only three variables.

```{r}
#| label: line-graph

aids_cases |>
  ggplot() +
    aes(yr, nsw) +
    geom_line()
```

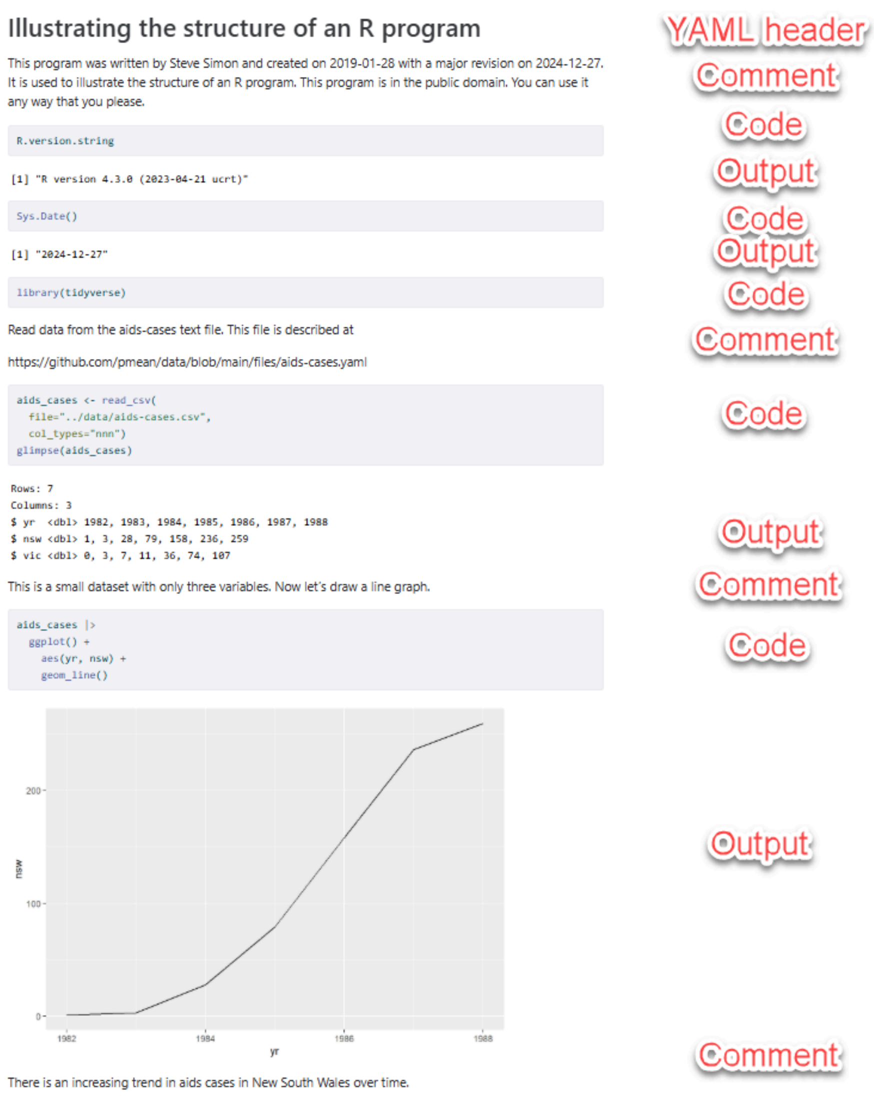
There is an increasing trend in aids cases in New South Wales over time.
```
```

Speaker notes

Here is the full program again. You can see that it starts with a header, and alternates between free text and R code.

I like this structure a lot because it allows you to liberally comment your code. Comments are easy. I’m not the best at using good programming practices, but the one thing I am rabid about is I try to put in a lot of documentation. It takes time, but usually pays off.

Output, overview



Illustrating the structure of an R program

This program was written by Steve Simon and created on 2019-01-28 with a major revision on 2024-12-27. It is used to illustrate the structure of an R program. This program is in the public domain. You can use it any way that you please.

```
R.version.string
```

```
[1] "R version 4.3.0 (2023-04-21 ucrt)"
```

```
Sys.Date()
```

```
[1] "2024-12-27"
```

```
library(tidyverse)
```

Read data from the aids-cases text file. This file is described at

https://github.com/pmean/data/blob/main/files/aids-cases.yaml

```
aids_cases <- read_csv(
  file="../data/aids-cases.csv",
  col_types="nnn")
glimpse(aids_cases)
```

```
Rows: 7
Columns: 3
$ yr  <dbl> 1982, 1983, 1984, 1985, 1986, 1987, 1988
$ nsw <dbl> 1, 3, 28, 79, 158, 236, 259
$ vic <dbl> 0, 3, 7, 11, 36, 74, 107
```

This is a small dataset with only three variables. Now let's draw a line graph.

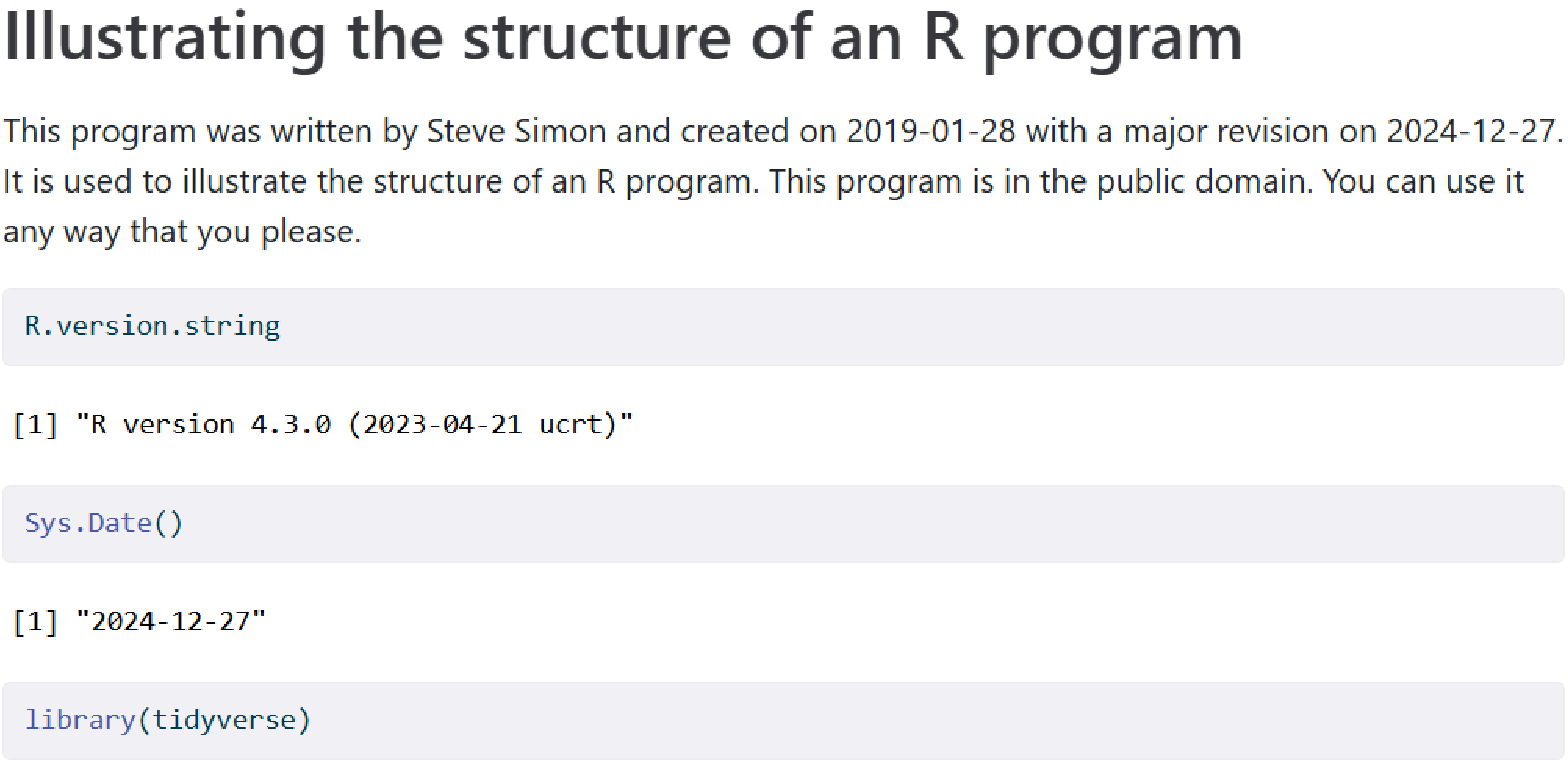
```
aids_cases |>
  ggplot() +
    aes(yr, nsw) +
    geom_line()
```

There is an increasing trend in aids cases in New South Wales over time.

Speaker notes

Here is an overview of the output. The font is too small to read, but I wanted you to see the big picture. The result of all this work is a document that includes information from the YAML header, with comments, code, and output interpresed.

Output, part 1



Illustrating the structure of an R program

This program was written by Steve Simon and created on 2019-01-28 with a major revision on 2024-12-27. It is used to illustrate the structure of an R program. This program is in the public domain. You can use it any way that you please.

```
R.version.string
```

```
[1] "R version 4.3.0 (2023-04-21 ucrt)"
```

```
Sys.Date()
```

```
[1] "2024-12-27"
```

```
library(tidyverse)
```

Speaker notes

The output starts with the title that you specified in the YAML header. Next follows the first comment that you placed at top of the first program chunk. Then the pieces of the program chunk appear with output from each part of the chunk interspersed.

Output, part 2

Read data from the aids-cases text file. This file is described at

https://github.com/pmean/data/blob/main/files/aids-cases.yaml

```
aids_cases <- read_csv(
  file="../data/aids-cases.csv",
  col_types="nnn")
glimpse(aids_cases)
```

```
Rows: 7
Columns: 3
$ yr  <dbl> 1982, 1983, 1984, 1985, 1986, 1987, 1988
$ nsw <dbl> 1, 3, 28, 79, 158, 236, 259
$ vic <dbl> 0, 3, 7, 11, 36, 74, 107
```

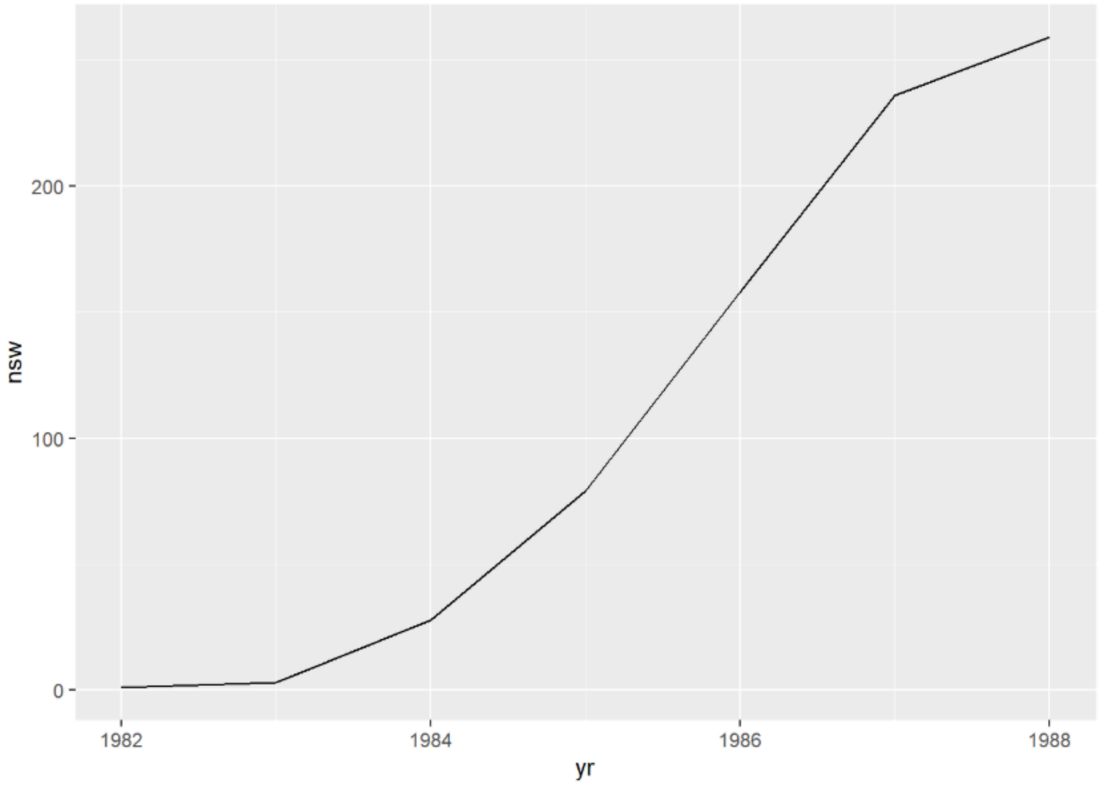
This is a small dataset with only three variables. Now let’s draw a line graph.

Speaker notes

Next is another comment, followed by code, output, and a third comment.

Output, part 3

```r
aids_cases |>
  ggplot() +
    aes(yr, nsw) +
    geom_line()
```



There is an increasing trend in aids cases in New South Wales over time.

Speaker notes

Finally, some more code, a graph, and a final comment.

Suggestions for nice looking comments, 1

- Quarto (and Rmarkdown) use tagged text files
    - Based on Markdown
    - Easy to remember
    - Easy read in its raw form
    - Use any program that edits text files

Speaker notes

Comments in Quarto and its predecessor, Rmarkdown, use a simple format, a tagged text file. These are based on two industry standards, Pandoc and Markdown.

There are lots of tagged text formats. One of the most common is html, the format used to display web pages. Markdown and Pandoc were developed as a simpler alternative to html. It has fewer features, but is simpler and thus easier to remember.

One of the nicest features of Markdown is that it is easy to read in its raw form. If you have ever looked at the raw html code of a web page, it is quite daunting. The tags are big and get in the way of reading the actual content. Markdown, in contrast, adds only a little bit of bulk to your content.

Another big advantage of Markdown is that the files can be edited by a wide range of programs. You are not tied to a specific software system.

Suggestions for nice looking comments, 2

- Interface with Pandoc to convert to (and from)
  - Microsoft Word, Powerpoint
  - Html files
  - PDF files

Speaker notes

Pandoc lives up to the “pan” in its name, which is code for “everything”. Using Markdown syntax, Pandoc can convert markdown files into dozens of different formats. In particular, it can create Word documents, Powerpoint presentations, web pages (html format), and PDF files. I should not that PDF files are a bit tricky because you need some version of Latex (or a new alternative typst) on your system. Instead of going straight to PDF, I recommend that you produce html files and then print that file to a pdf printer.

Suggestions for nice looking comments, 3

- Start line with ## for headlines
- Start lines with -, +, or * for bulleted lists
  - Indent for sub bullets
- Surround text with ** for **bold**
- Surround text with $ for Greek letters (μ) and math symbols ($\sqrt{2}$)
- Use [] for hyperlinks

Many more in quarto guide

Speaker notes

There are lots of little enhancements to your comments. Some that I use quite often are the double pound (##) symbols at the start of the line to make a headline in a larger font. You can make even larger headlines with a single pound, smaller headlines with three or more pounds.

I use bulleted lists a lot and you just have to start each line with a dash, plus, or asterisk to get the lines in a bulleted list. If you need sub bullets, just indent.

Surround one or more words with a double asterisk to get bold text and surround text with single or double dollar signs to get Greek letters like mu or math symbols like the square root sign.

An example of raw Markdown codes

```
## Suggestions for nice looking comments

-   Start line with ## for headlines
-   Start lines with -, +, or * for bulleted lists
    -   Indent for sub bullets
-   Surround text with ** for **bold**
-   Surround text with $ for Greek letters ($\mu$) and math symbols
($\sqrt{2}$)
-   Use [] for hyperlinks

Many more in [quarto guide][ref43]

[ref43]: https://quarto.org/docs/authoring/markdown-basics.html
```

Speaker notes

Here is an example of what some of these Markdown tags look like. The double pound provides the slide title, the dashes provide bullets. Note the indented bullet.

You don’t have to use a lot of these features to make your comments look nice. Just a few headlines here and an occasional bolded phrase will work wonders for you.

You will see how I use these features in various template programs in each module.

Break #4

- What you have learned
    - Anatomy of a small R program
- What’s coming next
    - Live demonstration

Live demonstration of running R

In this segment, you will see a live demonstration running the program simon-5505-01-demo.qmd.

Break #5

- What you have learned
  - Live demonstration
- What’s coming next
  - Good programming practices

General requirements for any program

There are **standards in six areas:**

- Documentation
- Graphs
- Tables
- Readability
- Interpretation
- Conciseness

There **may be times when one or two of these standards do not apply**. Which standards apply and which don’t should be obvious from the nature of the programming assignment. Ask if you are unsure what is required.

Documentation is required!

Documentation should include

- the name of the author (you!),
- the creation date,
- the purpose of your program, and
- any restrictions on use (your choice).
  - Public domain (no restrictions)
  - Specific restrictions on how others can use your program

Graphs cannot rely on default choices, 1

Always modify your graphs. **Do not settle for the default options**.

- Include your name and date on the title of any graph
  - "Steve Simon produced this graph on 2023-09-19."
- Avoid the display of unnecessary decimal places on the axes
- Use comma separators for large numbers
- Replace category codes with descriptive labels

Graphs cannot rely on default choices, 2

- Replace short variable names with longer descriptors
  - Include units of measurement, if needed
- Avoid the gratuitous use of color
  - Unless needed to distinguish between groups
  - Fill boxes and points with white/transparent colors

Tables also need modification

- Round to two or three significant figures
- Use comma separators if numbers are >= 1,000
- Avoid scientific notation (e.g., 1.23E-04)
- Avoid small p-values (e.g., p=0.000)
  - Change to p<0.001
- Suppress the printing of unneeded tables
  - Sometimes difficult

Sometimes default tables/graphs acceptable

- Early assignments may ask for defaults
- Always round and specify units in your interpretations

Your code must be easy to read

- Make liberal use of
  - blank lines
  - line breaks
  - indenting
  - vertical lists

Always include an interpretation

- Use simple evaluative words
  - Young/Elderly
  - Less than half/more than half
  - Almost all/almost none
  - Substantial improvement/roughly comparable
- Depends on context
  - No penalty for subjective judgments

Conciseness

- Do not include analyses that were not asked for
- Avoid displaying excessively large tables
  - This may be difficult for SAS and SPSS

Data dictionary

If you use a **data set that you found on your own** rather than one that your instructors provided, you must include a data dictionary. The elements of a data dictionary should include:

- Source
- Description
- Copyright
- Size
- Variables

Data dictionary: source

- Where did you find the data
  - Website link
  - Formal reference (if available)

Include a complete URL, **except if your data is behind a paywall**. If your data is associated with a peer-reviewed publication, provide a formal reference to that publication.

Data dictionary: Description

Provide a few sentences **explaining the context** of your data. Explain how the data was collected and what it is being used for.

Data dictionary: Copyright

On the Internet, **many datasets do not specify copyright**: the conditions under which you can use the data. If copyright information is available, it might take several forms:

- Public domain
  - Use the data any way you wish
- Copyright
  - Do not use without permission

- Open source license
  - Use the data with just a few restrictions
    - Cite the original source, CC-By
    - No commercial re-use CC-By-NC
    - Do not modify CC-By-ND

Data dictionary: Size

- Number of rows (excluding a header row)
- Number of columns

Data dictionary: Variables

- Name
- Label
- Units of measure

Data dictionary: Variable scale

- Scale
    - Nominal
    - Ordinal
    - Interval
    - Ratio

Data dictionary: Variable range

- Range
  - Non-negative (>= 0)
  - Positive (> 0)
  - Upper bound, if any

Data dictionary: Variable type

- Type
    - Integer
    - Float
    - Character

File details

This file was written by Steve Simon on 2024-12-26. It is in the public domain and you can use it any way you please.

The format function

```r
for (i in seq(2, 10, by=2)) {
    x <- factorial(i)
    y <- format(x, big.mark=",")
  print(glue("{i}! = {y}"))
}
```

```
2! = 2
4! = 24
6! = 720
8! = 40,320
10! = 3,628,800
```

Speaker notes

The format function with the big.mark argument will insert commas for any numbers larger than 999. This is especially important for numbers in the millions and higher.

The round function for large numbers

```r
for (i in seq(2, 10, by=2)) {
    x <- factorial(i)
    y <- round(x, digits=-2)
  print(glue("{i}! is approximately {y}"))
}
```

```
2! is approximately 0
4! is approximately 0
6! is approximately 700
8! is approximately 40300
10! is approximately 3628800
```

Speaker notes

The round function with the argument digits=-2 will round to the nearest hundred value.

The signif function for large numbers

```r
for (i in seq(2, 10, by=2)) {
    x <- factorial(i)
    y <- signif(x, digits=2)
  print(glue("{i}! is approximately {y}"))
}
```

```
2! is approximately 2
4! is approximately 24
6! is approximately 720
8! is approximately 40000
10! is approximately 3600000
```

Speaker notes

Contrast this with the signif function which maintains two significant digits.

The round function for small numbers

```r
for (i in seq(2, 10, by=2)) {
    x <- 0.5^i
    y <- round(x, digits=2)
  print(glue("0.5^{i} is approximately {y}"))
}
```

```
0.5^2 is approximately 0.25
0.5^4 is approximately 0.06
0.5^6 is approximately 0.02
0.5^8 is approximately 0
0.5^10 is approximately 0
```

Speaker notes

The round function with digits=2 will round to the nearest hundredth (0.01).

The signif function for small numbers

```r
for (i in seq(2, 10, by=2)) {
    x <- 0.5^i
    y <- signif(x, 2)
  print(glue("0.5^{i} is approximately {y}"))
}
```

```
0.5^2 is approximately 0.25
0.5^4 is approximately 0.062
0.5^6 is approximately 0.016
0.5^8 is approximately 0.0039
0.5^10 is approximately 0.00098
```

Speaker notes

Contrast this again with the signif function which maintains two significant digits.

Break #6

- What you have learned
  - Good programming practices
- What’s coming next
  - Your programming assignment

Program

- Download the demo program from module01
  - Store it in your src folder
- Modify the file name
  - Use your last name instead of “simon”
  - Change “demo” to “aids-cases”
- Modify the documentation headers
  - Add your name
  - Optional: change the copyright statement

Data

- Download the data file
  - Store it in your data folder
- Refer to the data dictionary, if needed.

Question 1

Calculate the minimum and maximum number of AIDS cases in Victoria from 1982 to 1987. The default format for this table is acceptable. Provide a brief interpretation.

Question 2

Graph the trend in AIDS cases in Victoria from 1982 to 1987.

Use a nice format and provide a brief interpretation.

Important reminder

Keep your output brief. After you have gotten your program to work, remove any code that does not address the questions above. Also, please remove any “Comments on the code” sections. But do provide interpretations when asked.

Grading rubric

You will be evaluated using the general grading rubric for programming assignments.

Your submission

- Save the output in html format
- Convert it to pdf format.
- Make sure that the pdf file includes
  - Your last name
  - The number of this course
  - The number of this module
- Upload the file

If it doesn’t work

Please review the suggestions if you encounter an error page.

File details

This programming assignment was written by Steve Simon on 2024-12-18 and is placed in the public domain.

Summary

- What you have learned
  - History of R
  - Installing R
  - Objects in R
  - Anatomy of a small R program
  - Live demonstration
  - Good programming practices
  - Your programming assignment