AMS-204: Introduction to Statistical Analysis
Fall 2021

Raquel Prado

raquel@soe.ucsc.edu

Introduction: R, RStudio, LaTeX

- **R** is a statistical computing environment for statistical computation & graphics. It is an interpreted computer language

- **R** is free and available at http://www.r-project.org/

- **R** is available for unix/linux, Windows, and Mac platforms

- **RStudio** is an integrated development environment (IDE) for **R** available at http://www.rstudio.com

Introduction: R, RStudio, LaTeX

- **R** is a "dialect" of the **S** programming language developed at Bell Labs by John Chambers and others starting in 1976 as an internal statistical analysis environment.
- StatSci (later Insightful Corp) developed a commercial implementation called **S-plus** in 1988.
- **R** was created by Ross Ihaka and Robert Gentleman in the Department of Statistics at the University of Auckland in 1991.
- The R Core Group, which controls the source code for **R**, was formed in 1997.

R: Advantages & Disadvantages

Advantages

- Flexible and easy to expand
- Powerful graphics
- Intuitive syntax
- Object oriented
- Lots of packages available; you can develop your own :-)

Disadvantages

- Step learning curve
- Memory hog
- Relatively slow

Introduction: R, RStudio, LaTeX

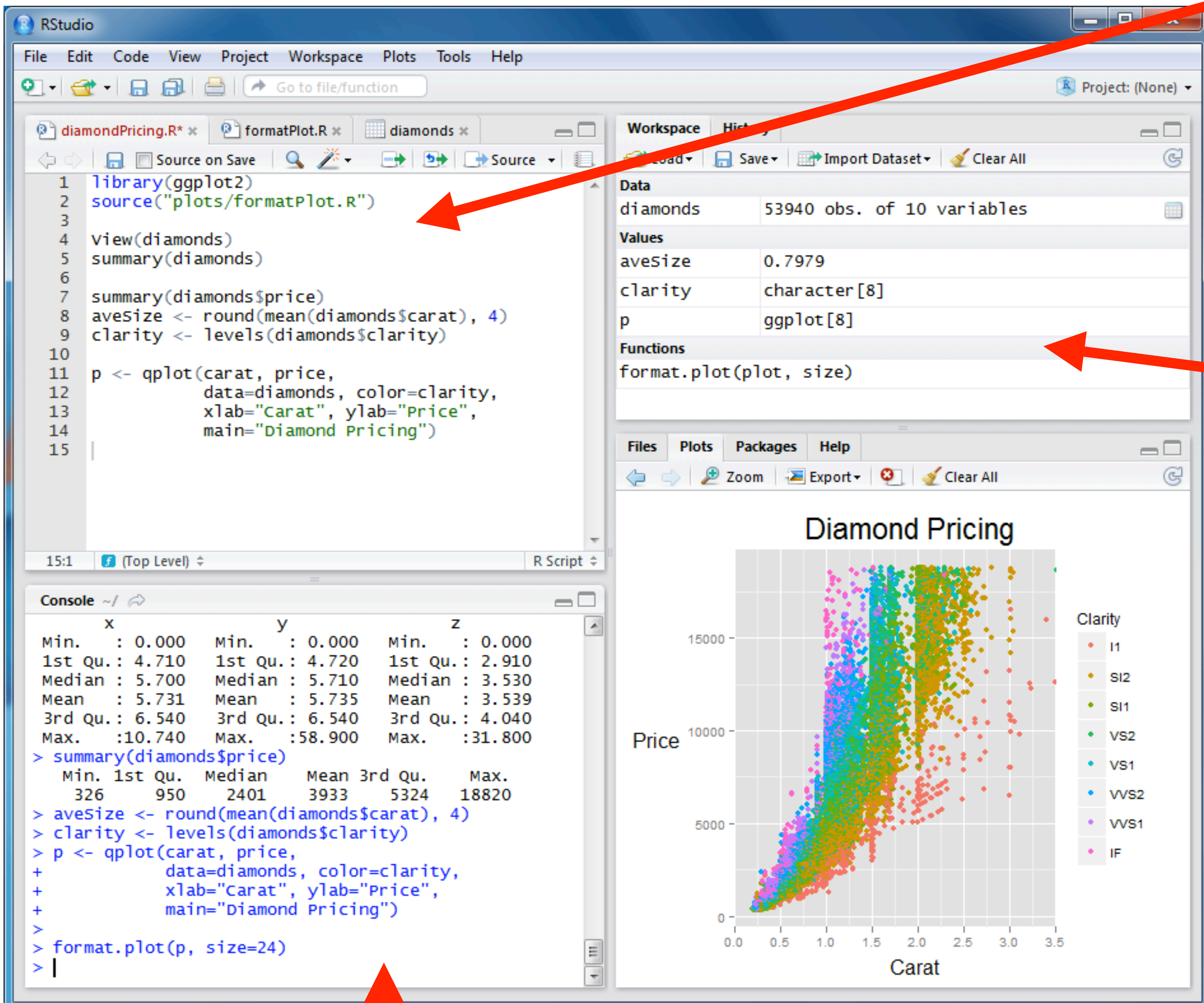
- In a windowing system (Windows, Macintosh, X,…) users interact with R through the R console

- **R** commands can be typed in the console or scripts can be used

- **RStudio** is command based but has a lot of additional functionality. You have to install **R** first.

- In **RStudio** you see 2-4 regions in the interface…

Introduction: R, RStudio, LaTeX



File (R script, R Markdown, LaTeX file, etc)

Objects

Output

Console

Introduction: R, RStudio, LaTeX

- **LaTeX** is a free, high quality typesetting system available at http://www.latex-project.org

- **RStudio** allows you to open and compile **LaTeX** files

- **R/RStudio** includes **Sweave, knitr and R Markdown**

- **Sweave** is based on **LaTeX** base format and **knitr** is an **R** package that adds features to **Sweave**

- You can set your own preferences for LaTeX, **Sweave** options, and editor options in **RStudio**

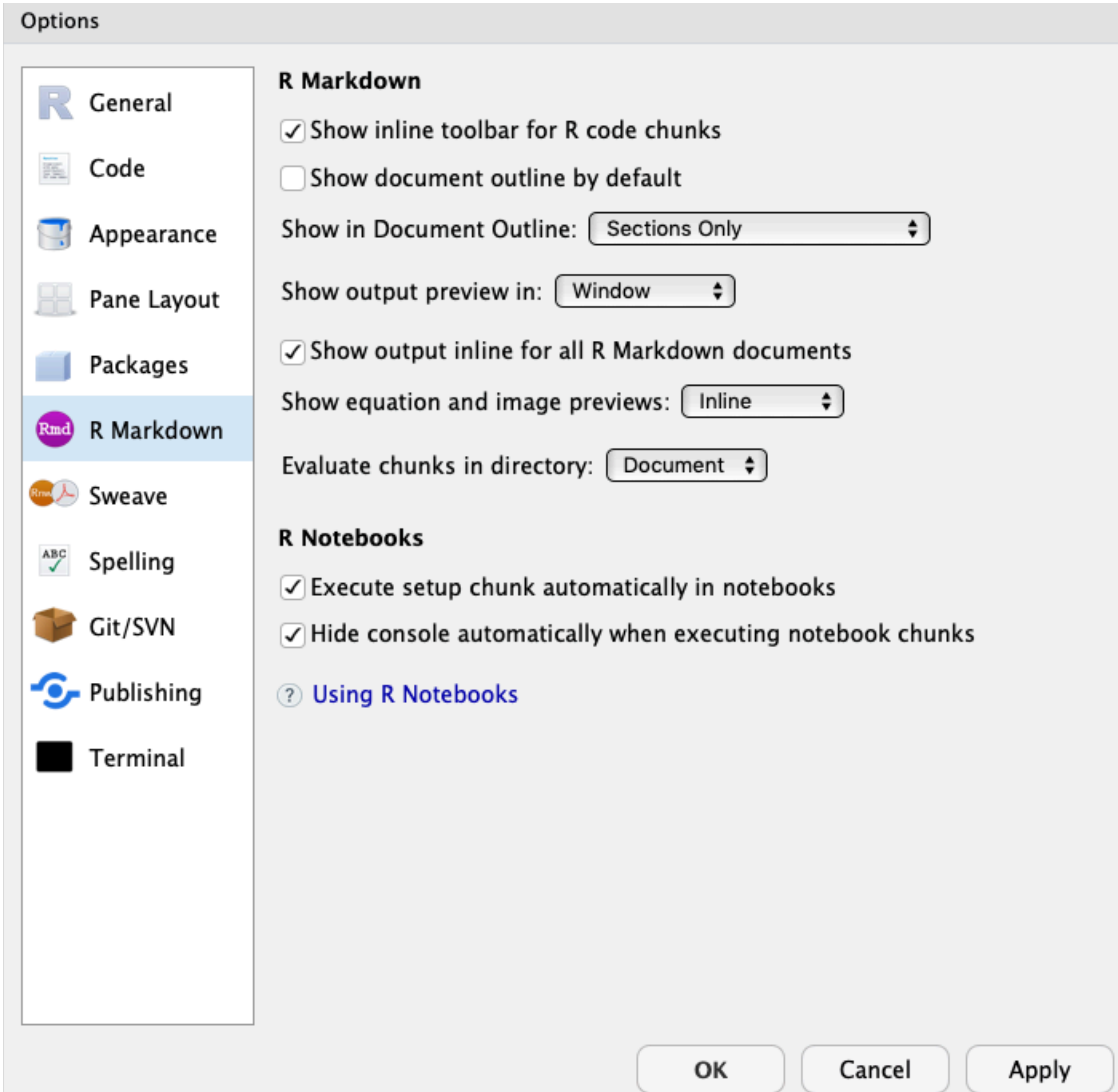
Introduction: R, RStudio, LaTeX

- **R Markdown** is based on markdown
- Both **R Markdown** and **Sweave** allow you to combine **R** code with LaTeX and allow several outputs including LaTeX, pdf, html and docx, among other
- You can also set your preferences for LaTeX within **R Markdown**

Online reference for R Markdown: Xie, Allaire, Grolemund (2019) R Markdown: The Definite Guide

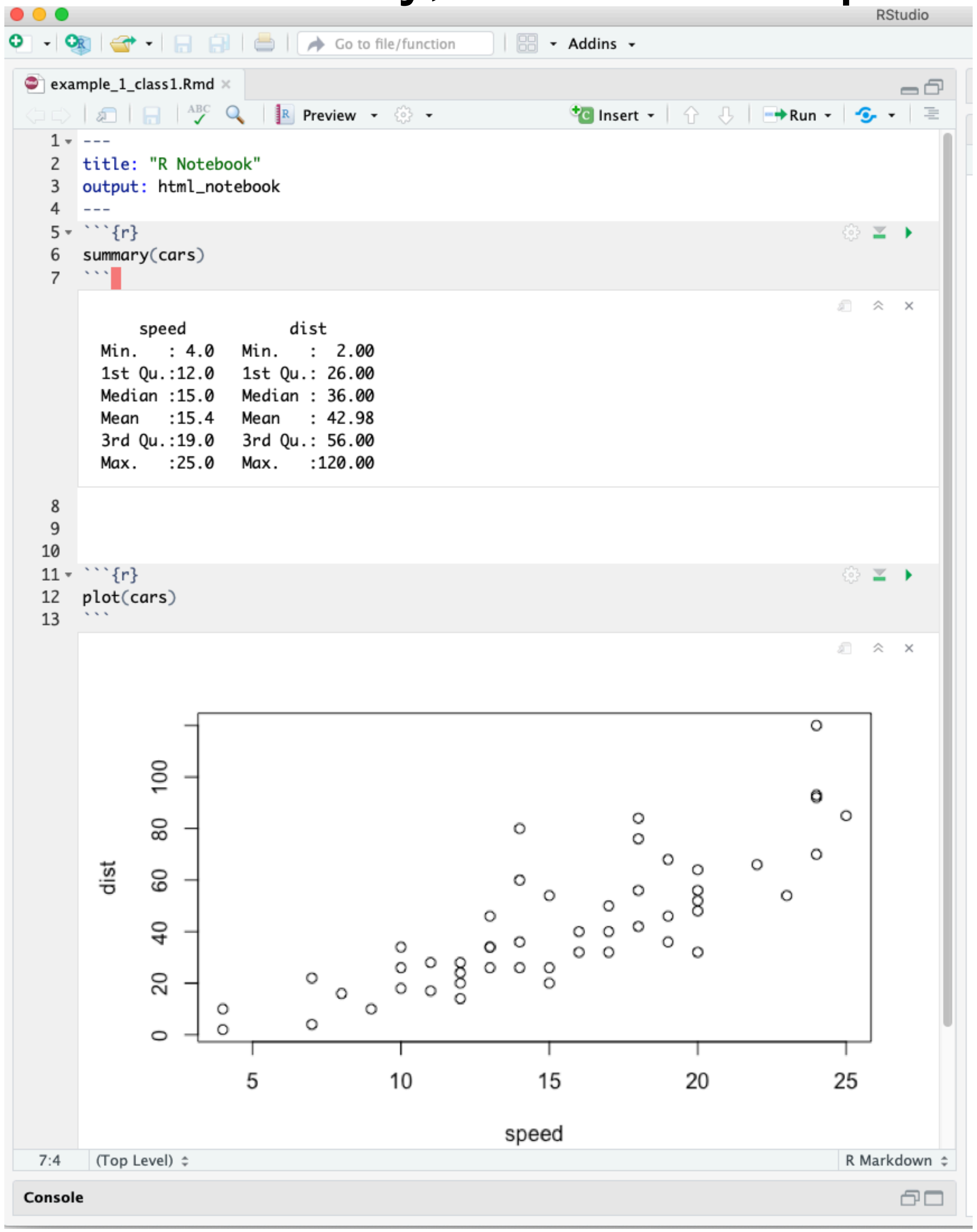
https://bookdown.org/yihui/rmarkdown/

Introduction: R, RStudio, LaTeX



Introduction: R, RStudio, LaTeX

- **Notebooks**: R Markdown document with chunks of code that can be executed independently and interactively, and with output appearing beneath the input.



Let's look at some examples…

Introduction: R Markdown, LaTeX

- **Notebook example**: `example_1_class1.Rmd`
- **R Markdown example, pdf**: `example_2_class1.Rmd`
- **Final report example with ASA double column style:**

1. Create a `.Rmd` file, e.g., use the `final_report.Rmd` file and download `asaproc.cls` into the same directory where the `.Rmd` file is

2. In the `final_report.Rmd` file note: you can change the LaTeX options; all the figures will go into the subdirectory /Fig/; you can change the options for the figures; bibliography can also follow a particular style

3. A `final_report.tex` file will be created. You can modify this file and compile the final pdf version outside RStudio using pdflatex

Introduction: R Markdown, LaTeX

- You can also just work with the LaTeX file and import all the figures you save from your RStudio session
- Use pdflatex of `template.tex` as an example

- **R** packages can be easily installed using the function `install.packages`
- You can get help/documentation for any **R** function (e.g., `lm`) by typing:

```
>help(lm)
```

or

```
>?lm
```

Getting Help in **R**

- Official Manuals:
  - An Introduction to R
  - R Data Import/Export
  - Writing R Extensions
  - R Installation and Administration

- Some good books:
  - Venables, William N., and Brian D. Ripley. *Modern applied statistics with S-PLUS*. Springer Science & Business Media, 2013.
  - Wickham, Hadley. *Advanced R*. CRC Press, 2014.
  - Other books in the *Use R!* sequence.
  - The R inferno!

Some recommendations:

- Make sure you document your analysis:
  - Save the commands you use for your analysis in a separate file. **Rstudio** has its own editor (or you can use your own)
    File -> New File -> R script
  - Comment (use #) and indent your code.
  - Limit the width of your code.
  - In most cases, it pays off in the long run if you save each figure in a multi-figure panel separately.
  - Use names for objects and graphs that are recognizable

- At its most basic, **R** can be used as a calculator:

```
> 5 + 7
[1] 12
```

- Example: Try to compute the natural log of 10 and the tangent of $\pi/4$

```
> log(10)
[1] 2.302585

> tan(pi/4)
[1] 1
```

Introduction: R basics

- Values can be stored into objects, the right-to-left assignment operators are the left arrow `<-` and the equal sign `=`

  `> a<-3` #Creates a scalar 3, assigns it to `a`

  `> x=c(10,1,1)` #Creates a vector (10,1,1), assigns it to `x`

  `> y=rpois(500, lambda=.6)`

  #Assigns result of `rpois` function to `y`. Note that the equal
  #sign inside the parentheses is not an assignment operator;
  #it passes the value of an argument (lambda) to the
  #function `rpois`

Another example:

```
> temps=c(51,60,62)
> 5*(temps-32)/9 #temperatures in Celsius
```

- **R** is case sensitive!
- You can do multiple assignments in **R**, e.g., `a<-b<-5`, but it is not recommended
- Never use the name of an **R** function as an object name (e.g., `mean`)

To see a list of R objects in the current environment

```
> ls()
[1] "a"  "b"
```

To remove an object (e.g., to free memory),

```
> rm(a) # Unusually, rm(“a”), is also valid
> ls()
[1] "b"
```

To remove all objects,

```
> rm(list=ls())
> ls()
character(0)
```

Introduction: R basics

- Basic object classes in **R**:
  Character
  Numeric (real numbers) (Default type for numbers!!)
  Integer
  Complex
  Logical (TRUE/FALSE)
  Factors

- Examples of objects in **R**:
  Vectors
  Matrices and Arrays
  Data Frames
  Lists

- Objects may have attributes (use the function `attr`)

Vectors: a vector contains a finite sequence of values of a single type.

- To create a vector with given entries:

```
> x = c(1,3,4,9,5)
> x
[1] 1 3 4 9 5
> x=c("CA","OR","WA","NE")
> x
[1] "CA" "OR" "WA" "NE"
```

- To create an empty vector of a given length:

```
> x = vector(“logical”, 5)
> x
[1] FALSE FALSE FALSE FALSE FALSE
```

```
> y = vector("numeric", 8)
> y
[1] 0 0 0 0 0 0 0 0
```

- To create a vector of length 0, a shortcut to

```
>x = vector("numeric", 0)
```

is

```
>x = numeric()
```

- You can create vectors that contain linear sequences:

```
> x = seq(1,8, by=2)
> x
[1] 1 3 5 7
> x = 3:8
> x
[1] 3 4 5 6 7 8
```

- You can also create vectors with repeated elements:

```
> x = rep(TRUE, 6)
> x
[1] TRUE TRUE TRUE TRUE TRUE TRUE
> y = rep(seq(1,3), 2)
> y
[1] 1 2 3 1 2 3
> y = rep(seq(1,3), each=2)
> y
[1] 1 1 2 2 3 3
```

- All elements of a vector must belong to the same class (integers, numeric, logical, etc)

- To check the class of the elements of a vector:

```
> x = rep(TRUE, 6)
> is.numeric(x)
[1] FALSE
> is.logical(x)
[1] TRUE
```

- If you mix classes, automatic coercions will happen

```
> x =c("CA",1,TRUE)
> is.numeric(x)
[1] FALSE
```

```
> is.logical(x)
[1] FALSE
> is.character(x)
[1] TRUE
> x = c(1.5, 3.7, 2, TRUE)
> is.numeric(x)
[1] TRUE
```

- To check the length of a vector:

```
> x = c(0, 1, 2, 0, 4, 1)
> length(x)
[1] 6
```

- You can force coercions, but some do not work well …

```
> x = c(0, 1, 2, 0, 4, 1)
> is.numeric(x)
[1] TRUE
> as.logical(x)
[1] FALSE TRUE TRUE FALSE TRUE TRUE
> y = c(TRUE, FALSE, TRUE, TRUE)
> as.numeric(y)
[1] 1 0 1 1
```

```
> z = c("CA","NE","OR")
> as.numeric(z)
[1] NA NA NA
Warning message: NAs introduced by coercion
```

- Missing values (NA) and not-a-number (NaN) are special values that might result from undefined operations.

```
> x = c(NA, 4, 3, NA, 1)
> is.numeric(x)
[1] TRUE
> is.na(x)
[1] TRUE FALSE FALSE TRUE FALSE
> 0/0
[1] NaN
```

- **R** also has a special "number" for infinity, `Inf`

```
> 3/0
[1] Inf
> 4 + Inf
[1] Inf
> 3/Inf
[1] 0
> Inf/Inf
[1] NaN
> 0/0
[1] NaN
> 1/Inf
[1] 0
```

- You can sort and rank the elements of a vector:

```
> x = c(1.3, -2.2, 1.15, 0.23, -1.1)
> sort(x)
[1] -2.20 -1.10  0.23  1.15  1.3
> x[order(x)]
[1] -2.20 -1.10  0.23  1.15  1.3
> order(x)
[1] 2 5 4 3 1
> rank(x)
[1] 5 1 4 3 2
```

- Many other functions: `sum(x), prod(x), which(x<0)`…

Summary statistic	How to compute in R
Number of observations	`length(x)`
Mean	`mean(x)`
Median	`median(x)`
Maximum	`max(x)`
Minimum	`min(x)`
Midrange	`(max(x)+min(x))/2`
Range	`max(x)-min(x)`
10% quantile	`quantile(x, 0.10)`
Variance	`var(x)`
Standard Deviation	`sqrt(var(x))`, `sd(x)`
Logical operators	**How to compute in R**
All entries true?	`all(x)`
At least one entry true?	`any(x)`

- **R** likes vectors. By default operators are "vectorized", i.e., applied to each element of the vector:

```
> x = c(3,-1,2,4)
> x + 3
[1]  6  2  5  7
> y = c(2,1,4,3)
> x*y
[1] 6 -1 8 12 (Not a scalar or matrix!)
> x >= 3
[1] TRUE FALSE FALSE TRUE
> x == -1
[1] FALSE TRUE FALSE FALSE
```

…be careful, `x<-1` and `x< -1` are different!

- If the length of two vectors/matrices involved in operations do not match, the values of the smaller one are recycled

```
> x = c(0,1,2,3)
> y = c(2,1)
> x + y
[1] 2 2 4 4 #As if y = c(2,1,2,1)
```

- No warning about recycling except when dimensions are not multiple of each other

```
> x = c(0,1,2,3)
> y = c(2,1,4)
> x + y
[1] 2 2 6 5 #As if y = c(2,1,4,2)
Warning message: In x + y : longer object
length is not a multiple of shorter
object length
```

- You can access specific elements of a vector. An option is to specify entries using a numeric vector of indexes:

```
> x = c(3,-1,2,4,7,-1)
> x[c(1,3,6)]
[1] 3  2 -1
> x[2:5]
[1] -1  2  4  7
```

- Another option is to use a logical vector (careful, if shorter than vector being sub-set, it is recycled and no error is generated):

```
> x = c(3,-1,2,4,7,-1)
> x[x!= -1]
[1] 3 2 4 7
> x[c(TRUE, FALSE, TRUE)]
[1] 3 2 4 -1
```

- When multiple conditions need to be checked you can use logical operators:

```
> x = c(3,-1,2,4,7,-1)
> x[x==-1 | x==4 | x==5]
[1] -1  4 -1
> x[x <= 7 & x>3]
[1] 4 7
```

- When multiple conditions or operators are required there are streamlined options:

```
> x = c(3,-1,2,4,7,-1)
> x[is.element(x, c(-1,4,5))]
[1] -1  4 -1
> x[x %in% c(-1,4,5)] #Alternative syntax
[1] -1  4 -1
```

- Set operators can be used to define the second argument:

```
> x = c(3,-1,2,4,7,-1)
> x[is.element(x, union(c(-1,4,5),
c(2,-1)))]
[1] -1  2  4 -1
> x[is.element(x, intersect(c(-1,4,5),
c(2,-1)))]
[1] -1 -1
```

- The `[ ]` operator can also be used to remove elements from a vector:

```
> x = c(3,-1,2,4,7,1)
> x[-c(1,5)]
[1] -1 2 4 1
```

- To add new elements to a vector we can either use the concatenation operator, or assign an extra element

```
> x = c(3,-1,2,4,7,1)
> x = c(x, 2)
> x
[1] 3 -1 2 4 7 1 2
> x[8] = -4
[1] 3 -1 2 4 7 1 2 -4
```

A matrix in **R** is a 2-dimensional array of values of a single type

- To define an empty one:

```
> x = matrix(0, nrow=2, ncol=3)
> x
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
```

- If the first argument is a vector, the matrix is filled with it (by default column-wise), and values recycled.

```
> x = matrix(c(1,2,3,4,5,6), nrow=2,
ncol=3)
> x
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

- The default direction in which the array is filled (by columns) can be changed:

```
> x = matrix(c(1,2,3,4,5,6), nrow=2,
ncol=3, byrow=T)
> x
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

- You can attach names to the rows and columns of the matrix:

```
> x = matrix(c(1,2,3,4,5,6), nrow=2,
ncol=3, dimnames=list(c("Row1",
"Row2"),c("Col1", "Col2", "Col3")))
```

```
> x
      [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> y = rownames(x)
> y
[1] “Row1” “Row2”
> colnames(x) = c(“AA”, “BB”, “CC”)
> x
     AA BB CC
Row1  1  3  5
Row2  2  4  6
```

- You can also create matrices by "binding" vectors together:

```
> x = c(1,2,3,4,5,6)
> y = c(11,12,13,14,15,16)
> z1 = rbind(x,y)
> z1
  [,1] [,2] [,3] [,4] [,5] [,6]
x    1    2    3    4    5    6
y   11   12   13   14   15   16
```

- If you want to generate a 6 by 2 matrix instead, use `cbind`.

- Matrices are sub-set in the same way as vectors:

```
> S = matrix(c(1,0.5,1.3,0.4,-1,-0.2,0.5,
-1,0.6), 3, 3)
> S[1:2,c(1,3)]
     [,1] [,2]
[1,]  1.0  0.5
[2,]  0.5 -1.0
```

- You can apply functions to rows and columns (more on this later):

```
> S = matrix(c(1,0.5,1.3,0.4,-1,-0.2,0.5,-1,
0.6), 3, 3)
> apply(S, 2, sum)
[1,]  2.8 -0.8  0.1
```

Retain second dimension (columns), collapse over all others

Function to be applied

- You can access matrices as if they were vectors

```
> S=matrix(c(1,0.5,1.3,0.4,-1,-0.2,0.5,-1,
0.6), 3, 3)
> S
     [,1] [,2] [,3]
[1,]  1.0  0.4  0.5
[2,]  0.5 -1.0 -1.0
[3,]  1.3 -0.2  0.6
> S[6]
[1,] -0.2
> S[6:8]
[1,] -0.2  0.5 -1.0
```

- To figure out the size of a matrix:

```
> dim(S)
[1] 3 3
> length(S)
[1] 9
```

- Unlike Matlab, **R** “drops” unnecessary dimensions by default. This can be problematic, but can be avoided:

```
> S[,3]
[1]  0.5 -1.0  0.6
> S[,3,drop=FALSE]
     [,1]
[1,]  0.5
[2,] -1.0
[3,]  0.6
```

- Linear algebra operations are easy in **R:**

```
> S = matrix(c(1,0.4,0.5,-1), 2, 2)
> x = c(0, 1)
> S%*%x     # matrix product
     [,1]
[1,]  0.5
[2,] -1.0
> eigen(S) # Eigendecomposition of S
$values
[1]  1.095445 -1.095445

$vectors
          [,1]       [,2]
[1,] 0.9822637 -0.2320969
[2,] 0.1875046  0.9726927
```

```
> S = matrix(c(1,0.4,0.5,-1), 2, 2)
> x = c(0, 1)
> t(S)                  # Transpose
     [,1] [,2]
[1,]  1.0  0.4
[2,]  0.5 -1.0
> solve(S)              # Inverse
          [,1]       [,2]
[1,] 0.8333333  0.4166667
[2,] 0.3333333 -0.8333333
> solve(S,x)            # Solution of Sb=x
[1]  0.4166667 -0.8333333
```