

# Lecture 1 Introduction and Basics

Prof. Alexandra Chouldechova

Fall 2020

What are we trying to accomplish?

*The sample analysis was shown only in class and is not viewable in this version of the notes.*

# Agenda

- ▶ Course overview
- ▶ Introduction to R, RStudio and R Markdown
- ▶ Programming basics

## How this class will work

- ▶ No programming knowledge presumed
- ▶ Some stats knowledge presumed. E.g.:
  - ▶ Hypothesis testing (t-tests, confidence intervals)
  - ▶ Linear regression
- ▶ Synchronous attendance is encouraged, but not required
- ▶ Class will be very cumulative

# Mechanics

- ▶ Two 80 minute lectures a week:
  - ▶ First 60-70 minutes: concepts, methods, examples
  - ▶ Last 10-20 minutes: short labs
- ▶ Class participation (10%)
- ▶ Quizzes (10%)
- ▶ Weekly homework (40%)
- ▶ Final project (2.5 weeks) (40%)
  - ▶ **Disclaimer:** To pass the class, you must achieve a passing score on the final project (at least 21 / 40)

# Mechanics

- ▶ **Class participation (10%)**
  - ▶ **Labs:** Each lecture has an accompanying lab assignment.
  - ▶ Course website shows how participation grade will be calculated
- ▶ **Quizzes (10%)**
  - ▶ 4 quizzes in the second half of term. Dates TBA.
- ▶ **Homework assignments (40%)**
  - ▶ There will be 5 weekly HW assignments
  - ▶ Single *lowest* HW score will be dropped
  - ▶ HW assigned on Wednesdays, **due Wednesdays at 1:30PM ET**
  - ▶ Late homework **will not be accepted for credit**
- ▶ **Final project (40%)**
  - ▶ You will write a report analysing a policy question using a publicly available data set

## Course resources

- ▶ Assignments, office hours, class notes, grading policies, useful references on R:  
<http://www.andrew.cmu.edu/~achoulde/94842/>
- ▶ Canvas for **gradebook** and for **turning in homework**
- ▶ Piazza for **discussion forum** (embedded in Canvas)
  - ▶ Please **post class/homework related question on Piazza** instead of emailing the teaching staff
- ▶ Check the class website for everything else
- ▶ No required textbook, but I highly recommend:
  - ▶ Garrett Golemund and Hadley Wickham, R for Data Science

# Goal of this class

## **This class will teach you to use R to:**

- ▶ Generate graphical and tabular data summaries
- ▶ Efficiently manipulate data using **tidyverse** libraries
- ▶ Perform statistical analyses (e.g., hypothesis testing, regression modeling)
- ▶ Produce *reproducible* statistical reports using R Markdown
- ▶ Near the end of class we'll also preview how to integrate R with other tools (e.g., databases, web, etc.)

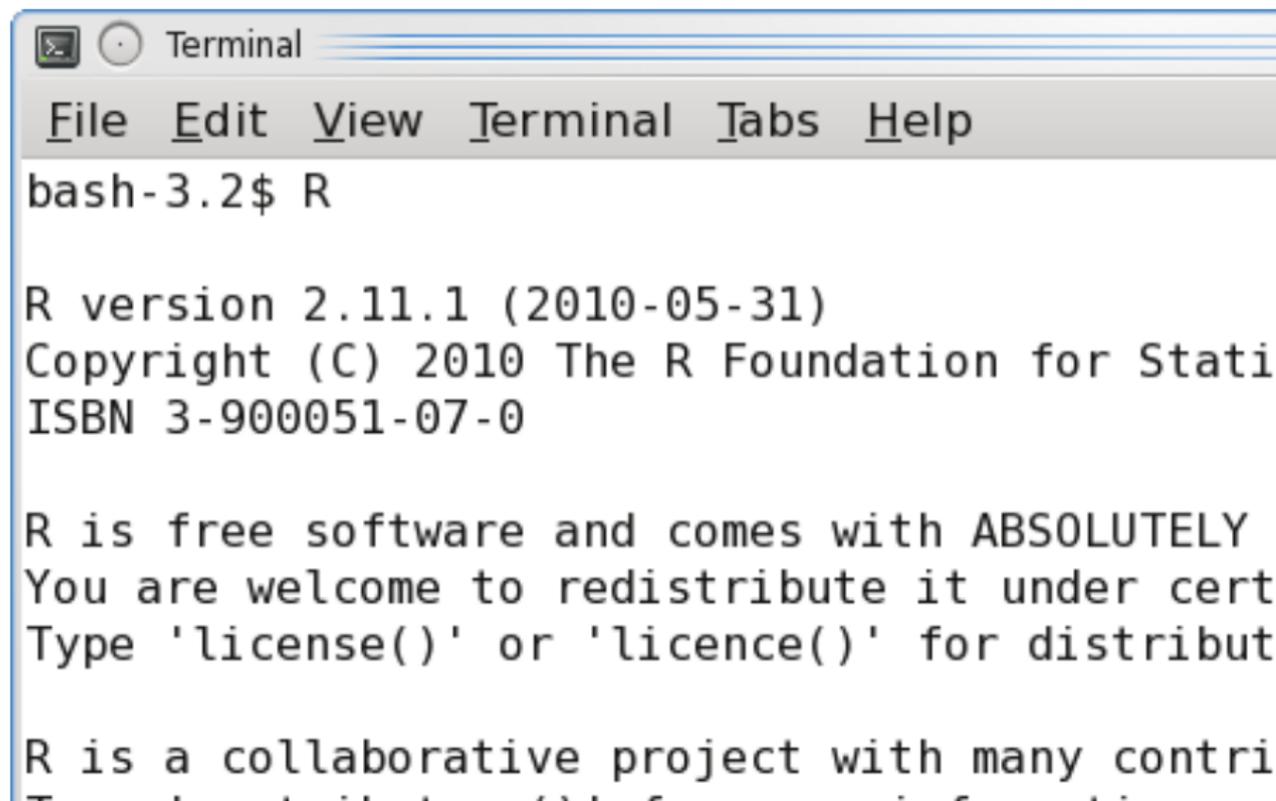
# Why R?

- ▶ Free (open-source)
- ▶ Programming language (not point-and-click)
- ▶ Excellent graphics
- ▶ Offers broadest range of statistical tools
- ▶ Easy to generate reproducible reports
- ▶ Easy to integrate with other tools

## The R Console

Basic interaction with R is through typing in the **console**

This is the **terminal** or **command-line** interface

A screenshot of a terminal window titled "Terminal". The window has a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal content shows the command "bash-3.2\$ R" followed by the R version information: "R version 2.11.1 (2010-05-31)", "Copyright (C) 2010 The R Foundation for Statistical Computing", and "ISBN 3-900051-07-0". Below this is a paragraph of text: "R is free software and comes with ABSOLUTELY NO WARRANTY. You are welcome to redistribute it under certain conditions. Type 'license()' or 'licence()' for distribution details." The bottom of the image shows the start of another paragraph: "R is a collaborative project with many contributors..."

```
Terminal
File Edit View Terminal Tabs Help
bash-3.2$ R
R version 2.11.1 (2010-05-31)
Copyright (C) 2010 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
```

# The R Console

- ▶ You type in commands, R gives back answers (or errors)
  - ▶ Menus and other graphical interfaces are extras built on top of the console
  - ▶ We will use **RStudio** in this class
1. Download R: <http://lib.stat.cmu.edu/R/CRAN>
  2. Then download RStudio: <http://www.rstudio.com/>

# RStudio is an IDE for R

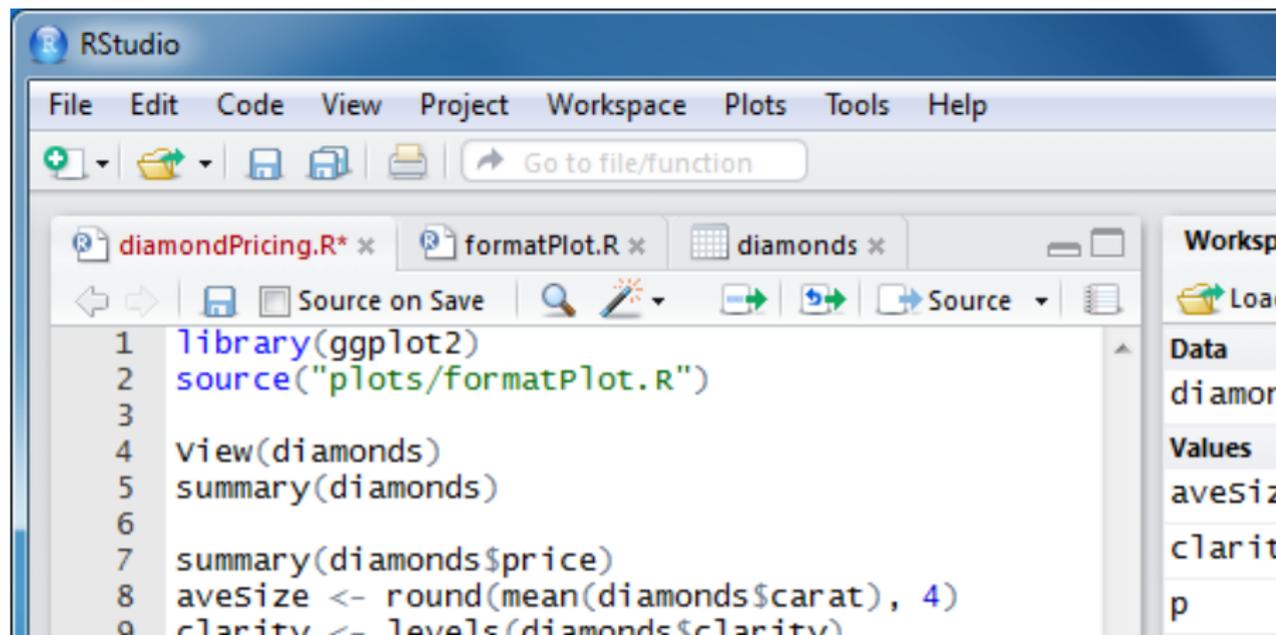
RStudio has 4 main windows ('panes'):

- ▶ Source
- ▶ Console
- ▶ Workspace/History
- ▶ Files/Plots/Packages/Help

# RStudio is an IDE for R

RStudio has 4 main windows (aka 'panes'):

- ▶ Source
- ▶ Console
- ▶ Workspace/History
- ▶ Files/Plots/Packages/Help



## RStudio: Panes overview

1. **Source** pane: create a file that you can save and run later
2. **Console** pane: type or paste in commands to get output from R
3. **Workspace/History** pane: see a list of variables or previous commands
4. **Files/Plots/Packages/Help** pane: see plots, help pages, and other items in this window.

## Console pane

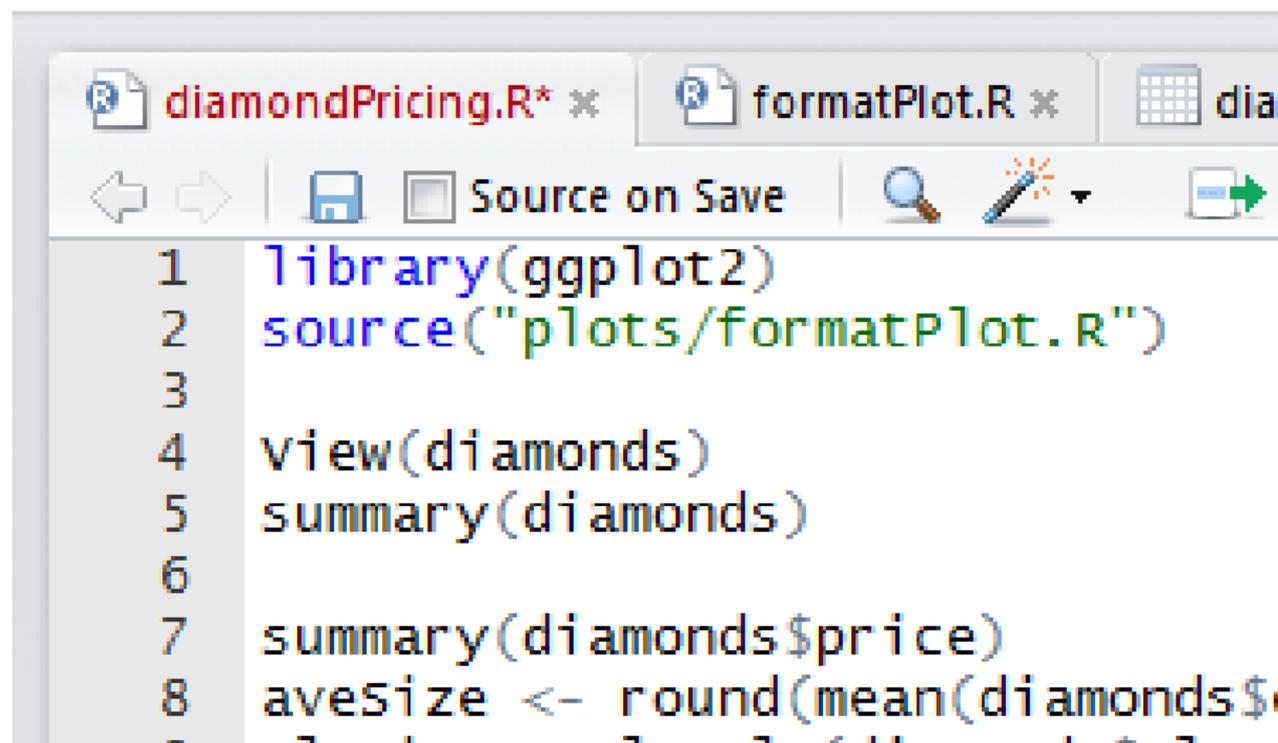
- ▶ Use the **Console** pane to type or paste commands to get output from R
- ▶ To look up the help file for a function or data set, type `?function` into the Console
  - ▶ E.g., try typing in `?mean`
- ▶ Use the tab key to auto-complete function and object names

Console ~/ 

	x		y
Min.	: 0.000	Min.	: 0.000
1st Qu.:	4.710	1st Qu.:	4.720
Median	: 5.700	Median	: 5.710
Mean	: 5.731	Mean	: 5.735
3rd Qu.:	6.540	3rd Qu.:	6.540

## Source pane

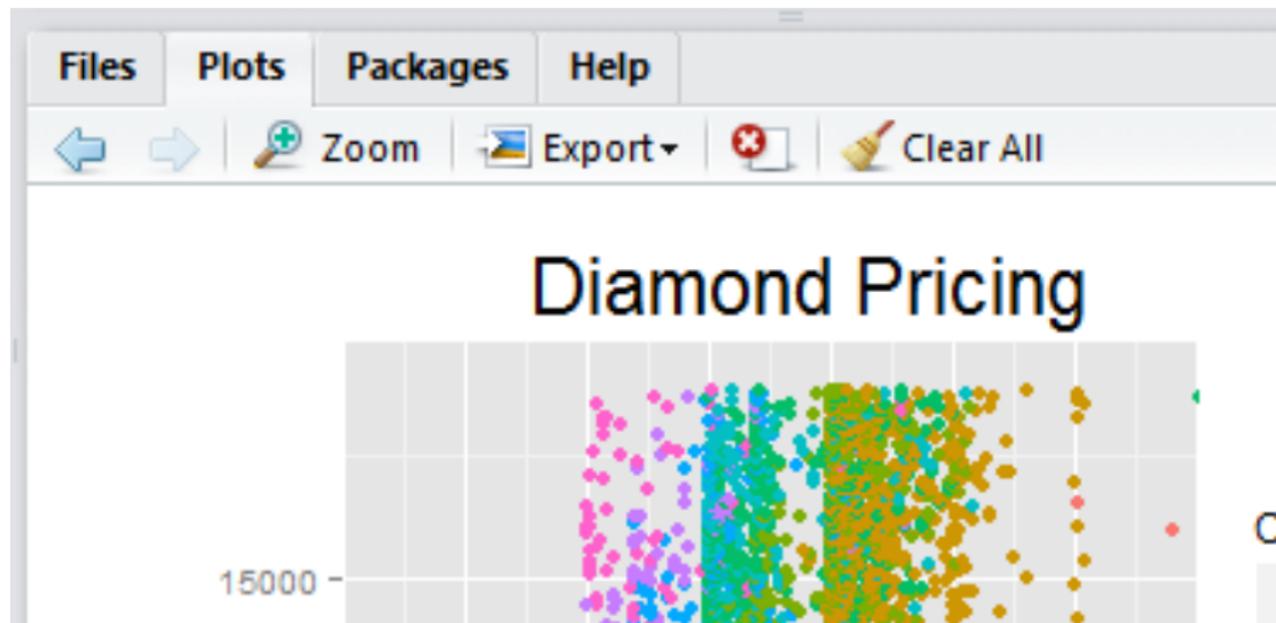
- ▶ Use the **Source** pane to create and edit R and Rmd files
- ▶ The menu bar of this pane contains handy shortcuts for sending code to the **Console** for evaluation



```
1 library(ggplot2)
2 source("plots/formatPlot.R")
3
4 view(diamonds)
5 summary(diamonds)
6
7 summary(diamonds$price)
8 aveSize <- round(mean(diamonds$
```

## Files/Plots/Packages/Help pane

- ▶ By default, any figures you produce in R will be displayed in the **Plots** tab
  - ▶ Menu bar allows you to Zoom, Export, and Navigate back to older plots
- ▶ When you request a help file (e.g., ?mean), the documentation will appear in the **Help** tab



## RStudio: Source and Console panes

**Source window:** Create a file here, so that you can save and run it later (or turn in as homework)

Any non-command line should start with a #



```
hw1.R *
Source on Sav
1 #
2 # David Choi
3 # HW 1
4 #
5
6 # Here are the
7 x = sqrt(5)
8 y = sqrt(2)
9 z = x + y
10 z
11
```

**Console:** Type or paste

# Variables

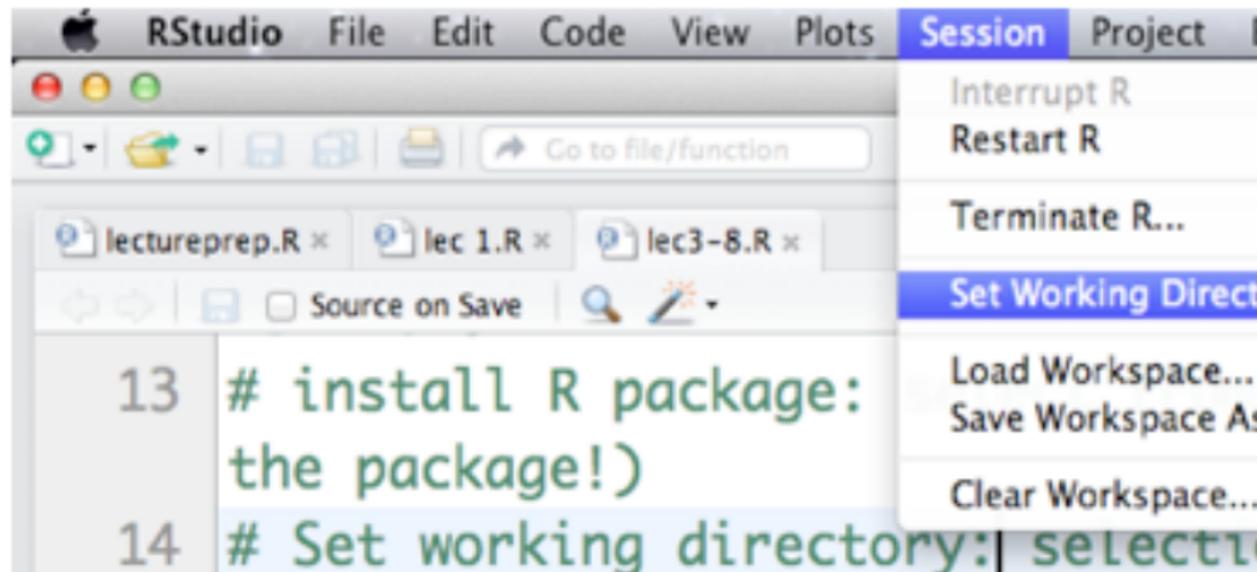
- Save the results of a command in memory by **giving it a name:**

`z` uses `x` and `y`: \_\_\_\_\_

Values are not linked;

# Two helpful menu

- Set the current directory:



The screenshot shows the RStudio application window. The menu bar includes Apple logo, RStudio, File, Edit, Code, View, Plots, Session, and Project. The Session menu is open, showing options: Interrupt R, Restart R, Terminate R..., Set Working Directory (highlighted), Load Workspace..., Save Workspace As..., and Clear Workspace... The code editor shows two lines of R code:

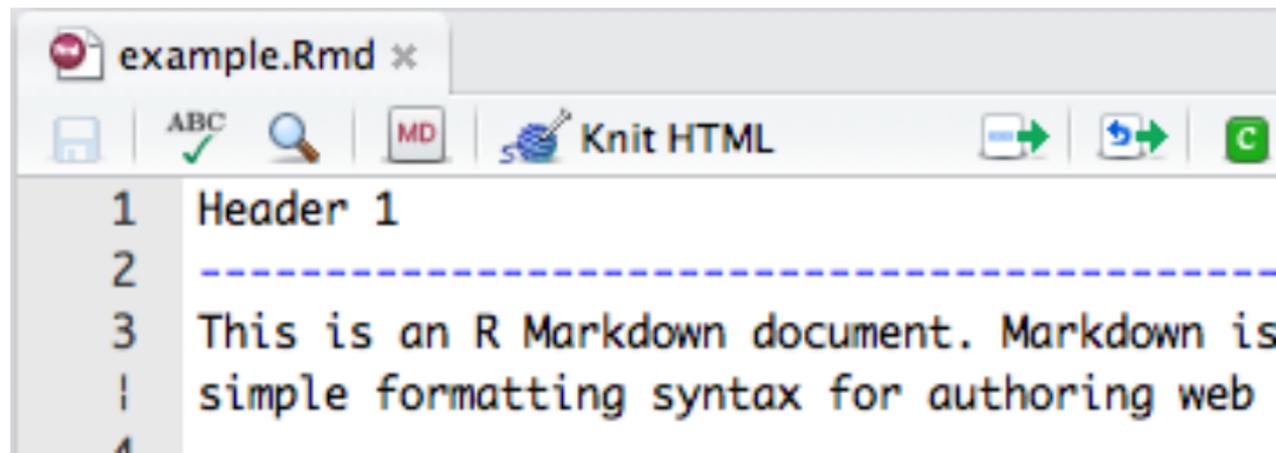
```
13 # install R package:  
    # the package!)  
14 # Set working directory: selecti
```

# R Markdown

- ▶ R Markdown allows the user to integrate R code into a report
- ▶ When data changes or code changes, so does the report
- ▶ No more need to copy-and-paste graphics, tables, or numbers
- ▶ Creates **reproducible** reports
  - ▶ Anyone who has your R Markdown (.Rmd) file and input data can re-run your analysis and get the exact same results (tables, figures, summaries)
- ▶ Can output report in HTML (default), Microsoft Word, or PDF

## R Markdown

- ▶ This example shows an **R Markdown** (.Rmd) file opened in the Source pane of RStudio.
- ▶ To turn an Rmd file into a report, click the **Knit HTML** button in the Source pane menu bar
- ▶ The results will appear in a **Preview window**, as shown on the right
- ▶ You can knit into html (default), MS Word, and pdf format
- ▶ These lecture slides are also created in RStudio (using ioslides as the output format)

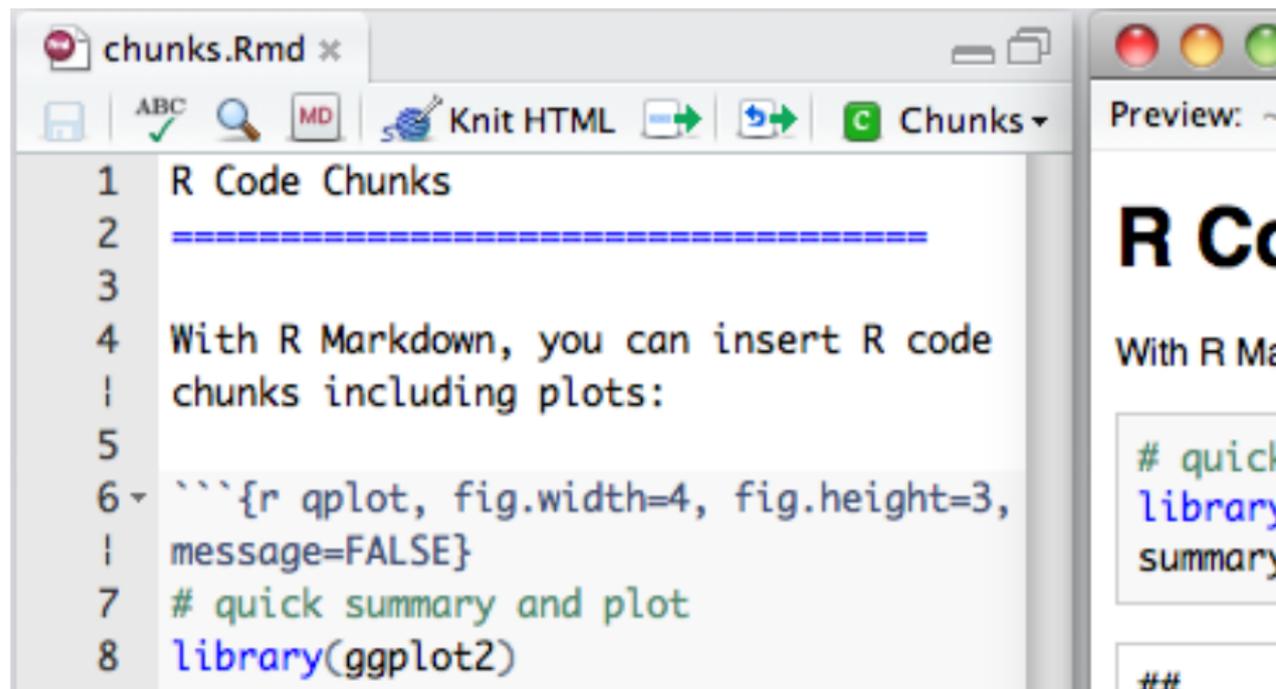


The screenshot shows the RStudio Source pane with a file named 'example.Rmd' open. The menu bar includes icons for saving, undo, redo, and a 'Knit HTML' button. The code content is as follows:

```
1 Header 1
2 -----
3 This is an R Markdown document. Markdown is
4 | simple formatting syntax for authoring web
```

## R Markdown

- ▶ To integrate R output into your report, you need to use R code chunks
- ▶ All of the code that appears in between the “triple back-ticks” gets executed when you Knit



The screenshot shows a software interface for editing R Markdown files. The main window displays a file named 'chunks.Rmd' with the following content:

```
1 R Code Chunks
2 =====
3
4 With R Markdown, you can insert R code
5 | chunks including plots:
6 | ````{r qplot, fig.width=4, fig.height=3,
7 | message=FALSE}
8 | # quick summary and plot
   | library(ggplot2)
```

The right-hand side of the image shows a 'Preview' window. It displays the rendered HTML output of the code above, including the heading 'R Code Chunks', a horizontal line, and the text 'With R Markdown, you can insert R code chunks including plots:'. Below this, a code block is shown with the R code: `# quick summary and plot` and `library(ggplot2)`.

## In-class exercise: Hello world!

1. Open **RStudio** on your machine
2. File > New File > R Markdown ...
3. Change `summary(cars)` in the first code block to `print("Hello world!")`
4. Click Knit HTML to produce an HTML file.
5. Save your Rmd file as `helloworld.Rmd`  
*All of your Homework assignments and many of your Labs will take the form of a single Rmd file, which you will edit to include your solutions and then submit on Canvas*

## Basics: the class in a nutshell

- ▶ Everything we'll do comes down to applying **functions** to **data**

- ▶ **Data:** things like 7, "seven", 7.000, the matrix  $\begin{bmatrix} 7 & 7 & 7 \\ 7 & 7 & 7 \end{bmatrix}$

- ▶ **Functions:** things like `log`, `+` (two arguments), `<` (two), `mod` (two), `mean` (one)

*A function is a machine which turns input objects (**arguments**) into an output object (**return value**), possibly with **side effects**, according to a definite rule*

# Data building blocks

You'll encounter different kinds of data types

- ▶ **Booleans** Direct binary values: TRUE or FALSE in R
- ▶ **Integers**: whole numbers (positive, negative or zero)
- ▶ **Characters** fixed-length blocks of bits, with special coding; **strings** = sequences of characters
- ▶ **Floating point numbers**: a fraction (with a finite number of bits) times an exponent, like  $1.87 \times 10^6$
- ▶ **Missing or ill-defined values**: NA, NaN, etc.

## Operators (functions)

You can use R as a very, very fancy calculator

---

Command	Description
<code>+, -, *, \</code>	add, subtract, multiply, divide
<code>^</code>	raise to the power of
<code>%%</code>	remainder after division (ex: $8 \% 3 = 2$ )
<code>( )</code>	change the order of operations
<code>log()</code> , <code>exp()</code>	logarithms and exponents (ex: $\log(10) = 2.30$ )
<code>sqrt()</code>	square root
<code>round()</code>	round to the nearest whole number (ex: <code>round(2.3)</code> )
<code>floor()</code> , <code>ceiling()</code>	round down or round up
<code>abs()</code>	absolute value

---

```
7 + 5 # Addition
```

```
## [1] 12
```

```
7 - 5 # Subtraction
```

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

```
7 * 5 # Multiplication
```

```
## [1] 35
```

```
7 ^ 5 # Exponentiation
```

```
## [1] 16807
```

```
7 / 5 # Division
```

```
## [1] 1.4
```

```
7 %% 5 # Modulus
```

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

```
7 %/% 5 # Integer division
```

```
## [1] 1
```

## Operators cont'd.

**Comparisons** are also binary operators; they take two objects, like numbers, and give a Boolean

```
7 > 5
```

```
## [1] TRUE
```

```
7 < 5
```

```
## [1] FALSE
```

```
7 >= 7
```

```
## [1] TRUE
```

```
7 <= 5
```

```
## [1] FALSE
```

```
7 == 5
```

```
## [1] FALSE
```

```
7 != 5
```

```
## [1] TRUE
```

## Boolean operators

Basically “and” and “or”:

```
(5 > 7) & (6*7 == 42)
```

```
## [1] FALSE
```

```
(5 > 7) | (6*7 == 42)
```

```
## [1] TRUE
```

(will see special doubled forms, && and ||, later)

## More types

- ▶ `typeof()` function returns the type
- ▶ `is.foo()` functions return Booleans for whether the argument is of type `foo`
- ▶ `as.foo()` (tries to) “cast” its argument to type `foo` — to translate it sensibly into a `foo`-type value

**Special case:** `as.factor()` will be important later for telling R when numbers are actually encodings and not numeric values. (E.g., 1 = High school grad; 2 = College grad; 3 = Postgrad) `##`

```
typeof(7)
```

```
## [1] "double"
```

```
is.numeric(7)
```

```
## [1] TRUE
```

```
is.na(7)
```

```
is.character(7)
```

```
## [1] FALSE
```

```
is.character("7")
```

```
## [1] TRUE
```

```
is.character("seven")
```

```
## [1] TRUE
```

```
is.na("seven")
```

```
## [1] FALSE
```

# Variables

We can give names to data objects; these give us **variables**

A few variables are built in:

```
pi
```

```
## [1] 3.141593
```

Variables can be arguments to functions or operators, just like constants:

```
pi*10
```

```
## [1] 31.41593
```

```
cos(pi)
```

```
## [1] -1
```

## Assignment operator

Most variables are created with the **assignment operator**, `<-` or `=`

```
time.factor <- 12  
time.factor
```

```
## [1] 12
```

```
time.in.years = 2.5  
time.in.years * time.factor
```

```
## [1] 30
```

The assignment operator also changes values:

```
time.in.months <- time.in.years * time.factor  
time.in.months
```

```
## [1] 30
```

```
time.in.months <- 45  
time.in.months
```

```
## [1] 45
```

- ▶ Using names and variables makes code: easier to design, easier to debug, less prone to bugs, easier to improve, and easier for others to read
- ▶ Avoid “magic constants”; use named variables
- ▶ Use descriptive variable names
  - ▶ Good: `num.students <- 35`
  - ▶ Bad: `ns <- 35`

# The workspace

What names have you defined values for?

```
ls()
```

```
## [1] "time.factor"      "time.in.months"  "time.in.years"
```

Getting rid of variables:

```
rm("time.in.months")
```

```
ls()
```

```
## [1] "time.factor"      "time.in.years"
```

## First data structure: vectors

- ▶ Group related data values into one object, a **data structure**
- ▶ A **vector** is a sequence of values, all of the same type
- ▶ `c()` function returns a vector containing all its arguments in order

```
students <- c("Sean", "Louisa", "Frank", "Farhad", "Li")  
midterm <- c(80, 90, 93, 82, 95)
```

- ▶ Typing the variable name at the prompt causes it to display

```
students
```

```
## [1] "Sean" "Louisa" "Frank" "Farhad" "Li"
```

# Indexing

- ▶ `vec[1]` is the first element, `vec[4]` is the 4th element of `vec`

```
students
```

```
## [1] "Sean" "Louisa" "Frank" "Farhad" "Li"
```

```
students[4]
```

```
## [1] "Farhad"
```

- ▶ `vec[-4]` is a vector containing all but the fourth element

```
students[-4]
```

```
## [1] "Sean" "Louisa" "Frank" "Li"
```

## Vector arithmetic

Operators apply to vectors “pairwise” or “elementwise”:

```
final <- c(78, 84, 95, 82, 91) # Final exam scores  
midterm # Midterm exam scores
```

```
## [1] 80 90 93 82 95
```

```
midterm + final # Sum of midterm and final scores
```

```
## [1] 158 174 188 164 186
```

```
(midterm + final)/2 # Average exam score
```

```
## [1] 79 87 94 82 93
```

```
course.grades <- 0.4*midterm + 0.6*final # Final course grades  
course.grades
```

```
## [1] 78.8 86.4 94.2 82.0 92.6
```

## Pairwise comparisons

Is the final score higher than the midterm score?

```
midterm
```

```
## [1] 80 90 93 82 95
```

```
final
```

```
## [1] 78 84 95 82 91
```

```
final > midterm
```

```
## [1] FALSE FALSE TRUE FALSE FALSE
```

Boolean operators can be applied elementwise:

```
(final < midterm) & (midterm > 80)
```

```
## [1] FALSE TRUE FALSE FALSE TRUE
```

## Functions on vectors

---

Command	Description
<code>sum(vec)</code>	sums up all the elements of <code>vec</code>
<code>mean(vec)</code>	mean of <code>vec</code>
<code>median(vec)</code>	median of <code>vec</code>
<code>min(vec), max(vec)</code>	the largest or smallest element of <code>vec</code>
<code>sd(vec), var(vec)</code>	the standard deviation and variance of <code>vec</code>
<code>length(vec)</code>	the number of elements in <code>vec</code>
<code>pmax(vec1, vec2), pmin(vec1, vec2)</code>	example: <code>pmax(quiz1, quiz2)</code> returns the higher of quiz 1 and quiz 2 for each student
<code>sort(vec)</code>	returns the <code>vec</code> in sorted order
<code>order(vec)</code>	returns the index that sorts the vector <code>vec</code>
<code>unique(vec)</code>	lists the unique elements of <code>vec</code>
<code>summary(vec)</code>	gives a five-number summary
<code>any(vec), all(vec)</code>	useful on Boolean vectors

---

## Functions on vectors

```
course.grades
```

```
## [1] 78.8 86.4 94.2 82.0 92.6
```

```
mean(course.grades) # mean grade
```

```
## [1] 86.8
```

```
median(course.grades)
```

```
## [1] 86.4
```

```
sd(course.grades) # grade standard deviation
```

```
## [1] 6.625708
```

## More functions on vectors

```
sort(course.grades)
```

```
## [1] 78.8 82.0 86.4 92.6 94.2
```

```
max(course.grades) # highest course grade
```

```
## [1] 94.2
```

```
min(course.grades) # lowest course grade
```

```
## [1] 78.8
```

## Referencing elements of vectors

```
students
```

```
## [1] "Sean" "Louisa" "Frank" "Farhad" "Li"
```

Vector of indices:

```
students[c(2,4)]
```

```
## [1] "Louisa" "Farhad"
```

Vector of negative indices

```
students[c(-1,-3)]
```

```
## [1] "Louisa" "Farhad" "Li"
```

## More referencing

which() returns the TRUE indexes of a Boolean vector:

```
course.grades
```

```
## [1] 78.8 86.4 94.2 82.0 92.6
```

```
a.threshold <- 90 # A grade = 90% or higher  
course.grades >= a.threshold # vector of booleans
```

```
## [1] FALSE FALSE TRUE FALSE TRUE
```

```
a.students <- which(course.grades >= a.threshold) # Applying  
a.students
```

```
## [1] 3 5
```

```
students[a.students] # Names of A students
```

```
## [1] "Frank" "Li"
```

## Named components

You can give names to elements or components of vectors

```
students
```

```
## [1] "Sean" "Louisa" "Frank" "Farhad" "Li"
```

```
names(course.grades) <- students # Assign names to the grades  
names(course.grades)
```

```
## [1] "Sean" "Louisa" "Frank" "Farhad" "Li"
```

```
course.grades[c("Sean", "Frank", "Li")] # Get final grades
```

```
## Sean Frank Li  
## 78.8 94.2 92.6
```

Note the labels in what R prints; these are not actually part of the value

## Useful RStudio tips

---

Keystroke	Description
<tab>	autocompletes commands and filenames, and lists arguments for functions. Highly useful!
<up>	cycle through previous commands in the console prompt
<ctrl-up>	lists history of previous commands matching an unfinished one
<ctrl-enter>	paste current line from source window to console. Good for trying things out ideas from a source file.
<ESC>	as mentioned, abort an unfinished command and get out of the + prompt

---

### “Homework” 0: Course survey

- ▶ You'll receive an announcement providing a link to a Google