

Intro2R

R Environment and Syntax

Song, Xiao Ping

xp.song@u.nus.edu

Course materials: <https://github.com/xp-song/Intro2R>

updated 2020-10-09

Before we begin...

1. Navigate to course webpage and read background information
<https://github.com/xp-song/Intro2R>
2. Ensure that you have installed **R** on your computer, followed by **R Studio**
(follow links under 'Instructions' section of webpage)
3. [Download](#) workshop materials
(green button on webpage)

Outline

About

Getting Started

General Syntax

Data Structures

Functions

Useful Resources

What is R?

- Programming language and software environment with a command line interface
- RStudio is often used as a software client
- Both R and RStudio are open source software
- Huge library of packages created by the R community



About this crash course

What it IS

- Designed for those with minimal coding experience
- Give you a taste of what R can do

What it is NOT

- A substitute to practicing the fundamentals of the language
- A lesson in statistics

Outline

About

Getting Started

General Syntax

Data Structures

Functions

Useful Resources

Course materials

Intro2R

*/notes*¹

/data

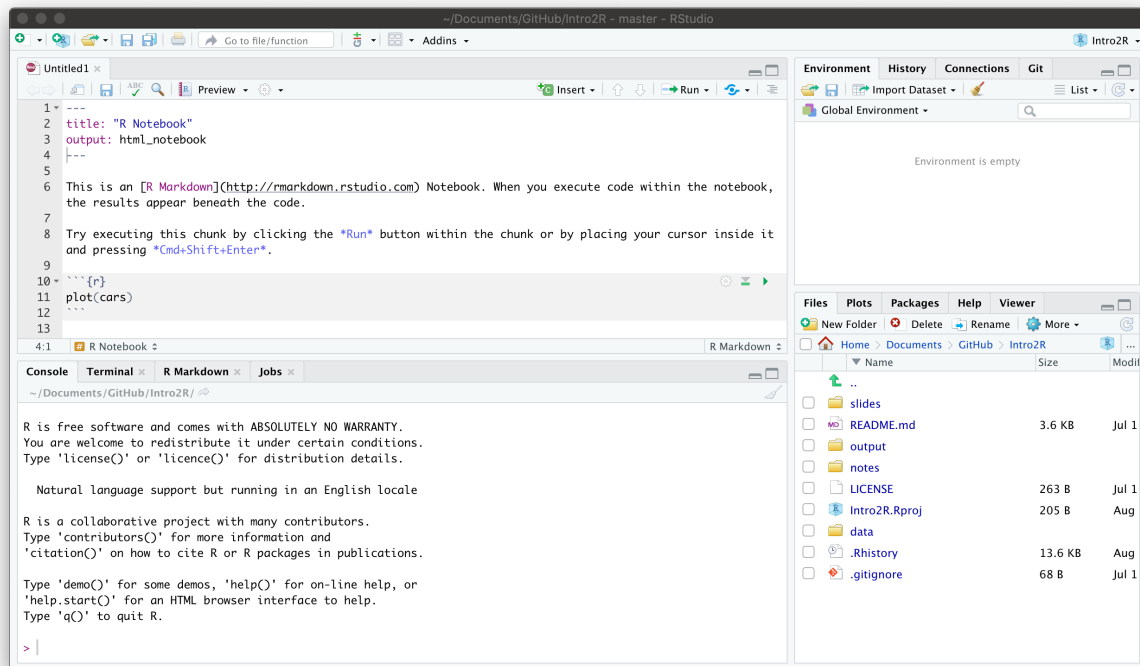
/output

PDF slide decks

Intro2R.Rproj

[1] View in your web browser by opening the *'html'* files

R Studio Client



- **Console:** Command line input/output
- **Script editor:** View/edit files that contain code
- **Environment/History**
- **Files/Plots/Packages/Help/Viewer**

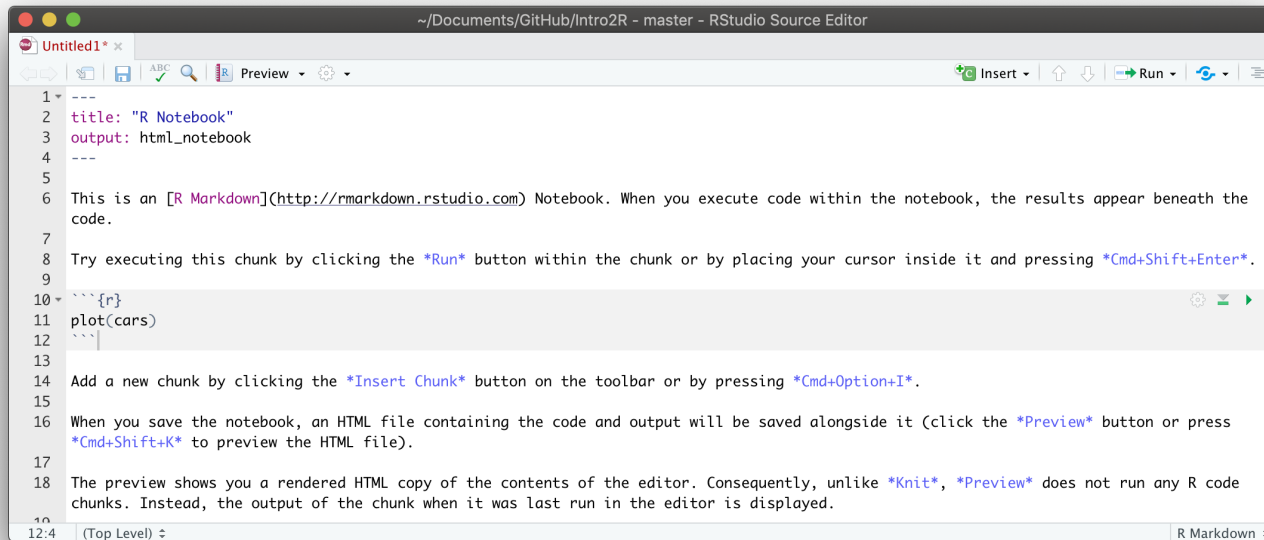
R Notebooks

What are R Notebooks?

- R Notebooks (a.k.a. [R Markdown Notebooks](#)) are files ending with `!Rmd`.
- Compared to basic `!R` scripts, they allows us to:
 - Write normal text alongside code
 - Interact with code within a single document
 - Generate (i.e. *'knit'*) different types of files

Try creating a new R Notebook `File > New File > R Notebook`

R Notebooks

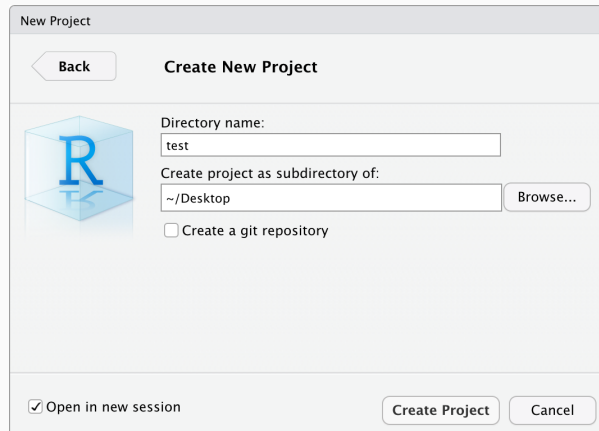


- **Header section:** specify document [parameters](#)
- **Normal text**
- **Code chunk:** write code and specify code [parameters](#)

Save our new file as 'myanalysis.Rmd'

RStudio Projects

Try creating a new RStudio Project `File > New Project > New Directory > New Project`



RStudio Projects

What are RStudio Projects?

- RStudio Projects help organise your work into separate 'R sessions'.
- Each project has it's own workspace a.k.a. 'working directory' (separate configuration, history, etc.)
- The location of the `'.RProj'` file defines the 'working directory'
 - **Type** `getwd()` **in the *console* of our new project**
 - This returns the absolute path to our working directory
e.g. `/Users/<computer_username>/Desktop/test`

RStudio Projects

★ Best Practice

- Use *relative* paths in your script, based on *.RProj* file location

- **Try reading in data in your R Notebook**

```
read.csv("<path to Intro2R folder>/Intro2R/data/ozone_data.csv")
```

```
read.csv("data/ozone_data.csv")
```

- Keep all project items in the working directory

Outline

About

Getting Started

General Syntax

Data Structures

Functions

Useful Resources

General Syntax

Operators

Arithmetic

Solve the following:

$$\frac{(1 + 2) * (4 - 5)}{50}$$

```
(1+2)*(4-5)/50
```

```
## [1] -0.06
```

General Syntax

Operators

Logical

Check if `1e3` is larger or equal to `1*10^3`
What is the output?

```
1e3 >= 1*10^3
```

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


General Syntax

Operators

Variables are named objects used to store data

Variables

- `<-` is used to assign variable names in R (e.g. `x <- 4`)
- Print variables by name (`x` vs. `"x"`)
- Assigning data to an existing variable overwrites it (`x <- 10`)

🌟 Best Practice

- Clear and consistent names
- Avoid numbers/symbols/whitespace

General Syntax

Operators

Variables

Data types and examples:

- Numeric (3.142), Integer (5L)
- Character ("hello")
- Logical (TRUE, FALSE)
- Complex

Let's assign new variables name, age, and weight

Check the data type for each variable using the function

is.numeric(), is.integer(), is.character()

Outline

About

Getting Started

General Syntax

Data Structures

Functions

Useful Resources

Data Structures

Vectors

- Linear collection of data
- Must be of the *same* data type

Assign a *vector* of names to the variable `name`

(use the concatenate function `c()`)

```
name <- c("Me", "Tom", "Dick", "Harry", "Susan") # character vector
```

Assign a *vector* of numbers to the variable `age`

```
age <- c(20, 25, 30, 35, 40) # numeric vector
```

Data Structures

Vectors

- Linear collection of data
- Must be of the *same* data type
- *Operations in R are vectorised*

Subtract 5 from the vector `age`

```
age-5
```

```
## [1] 15 20 25 30 35
```

Add together two vectors

```
age+age
```

```
## [1] 40 50 60 70 80
```

Data Structures

Vectors

- Linear collection of data
- Can contain of different *types* and *structure* of data

Lists

Create a list with a mix of data types and variables

```
myteam <- list(name, age, "Group 1", 2019)
```

Data Structures

Vectors

- Linear collection of data
- Can contain of different *types* and *structure* of data

Lists

```
myteam
```

```
## [[1]]  
## [1] "Me"      "Tom"      "Dick"     "Harry"   "Susan"  
##  
## [[2]]  
## [1] 20 25 30 35 40  
##  
## [[3]]  
## [1] "Group 1"  
##  
## [[4]]  
## [1] 2019
```

Data Structures

Vectors

- A special kind of vector that represents categorical data with discrete levels

Lists

Factors

Let's code the sex of each person in the variable `name`

(use the functions `factor()` and `c()`)

```
sex <- factor(c("M", "M", "M", "M", "F"))
```

```
sex
```

```
## [1] M M M M F
```

```
## Levels: F M
```


Data Structures

Vectors

- A special kind of vector that represents categorical data with discrete levels

Lists

Factors

Let's code the performance of each person in `name`

```
perform <- factor(c("High", "Low", "Med", "Med", "High"))
```

```
perform
```

```
## [1] High Low  Med  Med  High  
## Levels: High Low Med
```

What is wrong with this output?

Data Structures

Vectors

Lists

Factors

Define the order using the `levels=` argument in `factor()`

```
perform <- factor(c("High", "Low", "Med", "Med", "High"),  
                  levels = c("Low", "Med", "High"))  
  
perform
```

```
## [1] High Low  Med  Med  High  
## Levels: Low Med High
```

Data Structures

Vectors

- Tabular data (rows & columns)
- Must be of the *same* data type

Lists

Factors

Create a 4 by 3 matrix of sequential numbers

Use `matrix()` and the `:` operator to create a sequence

Matrices

```
m <- matrix(1:12, nrow = 4)
```

```
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

Data Structures

Vectors

Lists

Factors

Matrices

Dataframes

- Tabular data (rows & columns)
- Rows represent data entries, columns represent different variables

Import the dataset `ozone_data.csv` into your R Notebook using `read.csv()`

```
ozone <- read.csv("data/ozone_data.csv") # column headers in first row
```

Data Structures

Vectors

View the first few rows of ozone

Lists

```
head(ozone) #print first few rows
```

Factors

Check the dimensions of ozone

Matrices

```
dim(ozone)
```

Dataframes

```
## [1] 111 4
```

Data Structures

Vectors

Check the names of `ozone` using `dimnames()`, `rownames()` and `colnames()`

Lists

```
dimnames(ozone)
```

Factors

Matrices

Dataframes

```
## [[1]]
##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13"
##  [14] "14" "15" "16" "17" "18" "19" "20" "21" "22" "23" "24" "25" "26"
##  [27] "27" "28" "29" "30" "31" "32" "33" "34" "35" "36" "37" "38" "39"
##  [40] "40" "41" "42" "43" "44" "45" "46" "47" "48" "49" "50" "51" "52"
##  [53] "53" "54" "55" "56" "57" "58" "59" "60" "61" "62" "63" "64" "65"
##  [66] "66" "67" "68" "69" "70" "71" "72" "73" "74" "75" "76" "77" "78"
##  [79] "79" "80" "81" "82" "83" "84" "85" "86" "87" "88" "89" "90" "91"
##  [92] "92" "93" "94" "95" "96" "97" "98" "99" "100" "101" "102" "103" "104"
## [105] "105" "106" "107" "108" "109" "110" "111"
##
## [[2]]
## [1] "rad"  "temp" "wind" "ozone"
```

Data Structures

Vectors

Extract data by colnames using `$`

(output is a vector)

Lists

```
ozone$temp
```

Factors

```
## [1] 67 72 74 62 65 59 61 69 66 68 58 64 66 57 68 62 59 73 61 61 67 81 79 76 82
```

```
## [28] 82 77 72 65 73 76 84 85 81 83 83 88 92 92 89 73 81 80 81 82 84 87 85 74 86
```

```
## [55] 86 88 86 83 81 81 81 82 89 90 90 86 82 80 77 79 76 78 78 77 72 79 81 86 97
```

```
## [82] 94 91 92 93 93 87 84 80 78 75 73 81 76 77 71 71 78 67 76 68 82 64 71 81 69
```

```
## [109] 75 76 68
```

Matrices

Dataframes

Data Structures

Vectors

Create a dataframe with the vectors `name`, `sex`, `age` **and** `perform`

Lists

```
team_details <- data.frame(name, age, sex, perform)
```

Factors

```
team_details
```

Matrices

Dataframes

Back to operators...

Subsetting in R

What is 5th element in the vector `name`?

```
name[5]
```

```
## [1] "Susan"
```

What is the 4th element of the column `name` in the dataframe `team_details`?

(hint: use `$` to extract columns as vectors)

```
team_details$name[4]
```

```
## [1] "Harry"
```

Back to operators...

Subsetting in R

Extract the element in the 2nd row and 4th col in `team_details`

```
team_details[2,4]
```

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

```
## Levels: Low Med High
```

Extract 2nd row and all cols in `team_details`

```
team_details[2,]
```

Extract the 4th col and all rows except the 2nd in `team_details`

```
team_details[-2,4]
```

```
## [1] High Med  Med  High
```

```
## Levels: Low Med High
```

Back to operators...

Subsetting in R

Extract rows 1 to 3 in `team_details`

```
team_details[1:3,]
```

Extract rows 1 and 3 in `team_details`

```
team_details[c(1,3),]
```

Back to operators...

Subsetting in R: Quick test! ⚡

Extract R's built-in dataset `data(mtcars)`

Extract data on cars with a fuel efficiency of at least 20 mpg, and that are more than 108 hp

- **Hint:** `mtcars[& ,]`
- **Hint:** `mtcars[mtcars$mpg >= 20 & ,]`
- **Answer:** `mtcars[mtcars$mpg >= 20 & mtcars$hp > 108,]`

Outline

About

Getting Started

General Syntax

Data Structures

Functions

Useful Resources

Functions

Overview

Functions have inputs and outputs (look up details with `?`)

E.g. Plot the performance distribution in `team_details`

```
plot(team_details$perform)
```

Functions

Overview

Functions have inputs and outputs (look up details with `?`)

E.g. Find the mean age in `team_details`

```
mean(team_details$age)
```

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

E.g. Find the number of people (rows) in `team_details`

```
nrow(team_details)
```

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

Functions

Overview

User-defined

General structure when defining a function:

```
functionname <- function(inputs){  
  # calculations...  
  output  
}
```

Subsequent calls to the function:

```
functionname(inputs)
```


Functions

Overview

E.g. Load `data/grades.csv` and assign it the name `grades`

User- defined

Functions

Overview

User-
defined

$$\frac{\sum_{i=1}^n \text{gradepoint}_i \times \text{credits}_i}{\sum_{i=1}^n \text{credits}_i}$$

Manually calculate the GPA in R using the formula

```
sum(grades$grade_point * grades$credits) / sum(grades$credits)
```

```
## [1] 3.22
```

Functions

Overview

User-defined

Create a function named `scorer` that:

- Takes a dataframe as input
- Outputs a calculation based on the colnames `grade_point` and `credits`

```
scorer <- function(x){  
  sum(x$grade_point*x$credits) / sum(x$credits)  
}
```

```
scorer(grades) #use function
```

```
## [1] 3.22
```

Functions

Overview

- Loop functions repeat code `i` number of times
- Most common type: `for` loop

User- defined

Loops

Functions

Overview

User-

defined

Loops

Prepare our data inputs to the `for` loop:

Get the grades of other team members within `/data` folder

```
grades_tom <- read.csv("data/grades_tom.csv")
grades_dick <- read.csv("data/grades_dick.csv")
grades_harry <- read.csv("data/grades_harry.csv")
grades_susan <- read.csv("data/grades_susan.csv")
```

Put all these dataframes into a list named `team_grades`

```
team_grades <- list(grades, grades_tom, grades_dick, grades_harry, grades_sus
```

Functions

Overview

User- defined

Loops

For every item (person) in the list `team_grades`, use the function `scorer()` and append results to new column "GPA" in `team_details`

```
for(i in 1:length(team_grades)){  
  team_details$GPA[i] <- scorer(team_grades[[i]])  
}
```

#the named object "i" changes in value with iteration of the loop

Who has the best grades in the team?

Functions

Overview

User-defined

Loops

Loop functions in base R

`lapply(x, FUN)`: Apply a function on each element of `x`, returns a *list*

`apply(x, MARGIN, FUN)`: Apply a function to tabular data by rows (`1`), cols (`2`), or both `c(1,2)`

Find the mean value for *each* numeric column in `team_details`

```
apply(team_details[,c(2,5)], 2, mean) #apply mean() function across columns
```

```
##      age      GPA
## 30.000000 4.122947
```

Functions

Overview

User-
defined

Loops

Quick test! ⚡

Calculate the mean for each numeric variable in `data(mtcars)`

Answer:

```
apply(mtcars, 2, mean)
```

```
##      mpg      cyl      disp      hp      drat      wt      qsec
## 20.090625  6.187500 230.721875 146.687500  3.596563  3.217250 17.848750
##      vs      am      gear      carb
##  0.437500  0.406250  3.687500  2.812500
```


Questions?

[About](#)

[Getting Started](#)

[General Syntax](#)

[Data Structures](#)

[Functions](#)

[Useful Resources](#)

Useful Resources

Online tutorials

- [R for Data Science](#)
- [Quick R](#)
- [R for cats](#) (blog post)
- [swirl](#) (good for practice)
- [R markdown cookbook](#)

Online Q&A

- [Stack Overflow](#)
- [How to ask a good question online](#)
- Remember to check your `sessionInfo()` when troubleshooting!

Others

- [Use R/RStudio from an external drive](#) (if you don't have admin rights to install software)