STAT 3355 Introduction to Data Analysis

Lecture 02: R Basics I

Created by Dr. Qiwei Li
Presented by Dr. Octavious Smiley

Department of Mathematical Sciences
The University of Texas at Dallas

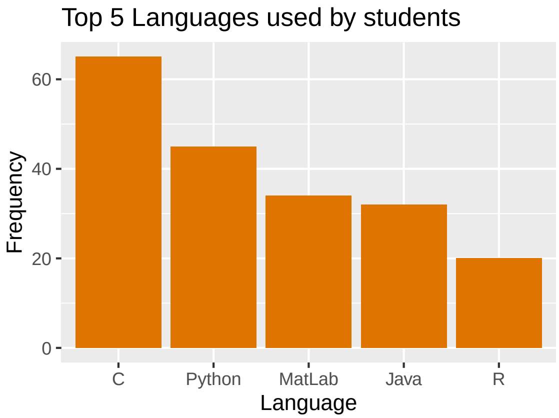
Learning Goals

- What is R and RStudio
- Use R as a calculator
  - Basic mathematical operations
  - Basic logical operations
- Write simple functions in R

What is R

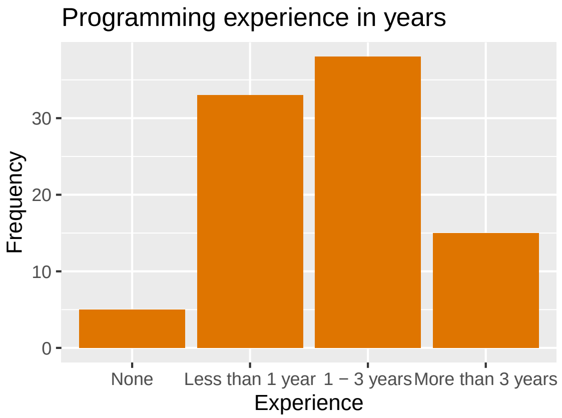
- A programming language and free software environment for statistical computing and graphics supported by the R Foundation
  - Designed by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand
  - First appeared in August, 1993
  - Written in `C`, `Fortran`, and `R` itself
  - Ranked 13-th in the TIOBE index now, a measure of popularity of programming languages

A New Language



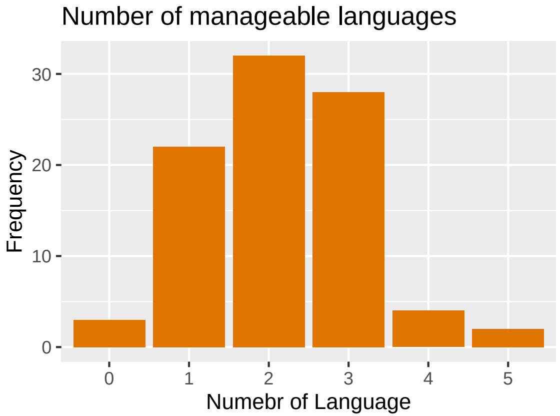
- Only 20 students have programmed in R before!

More about You

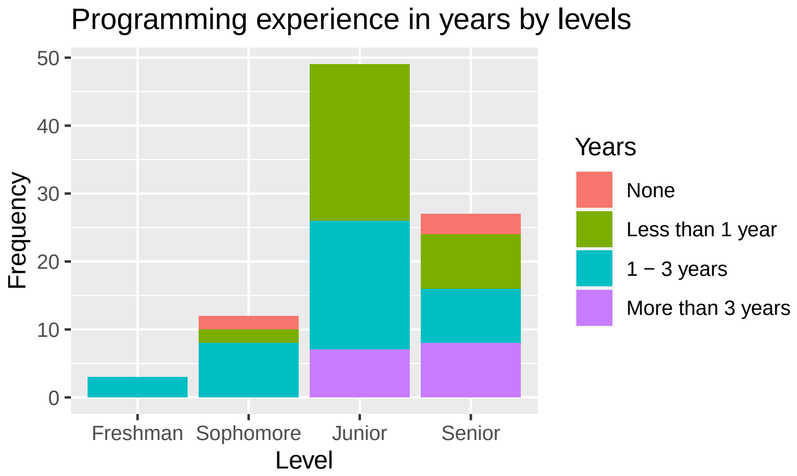


- Only 5 students have no any programming experience before

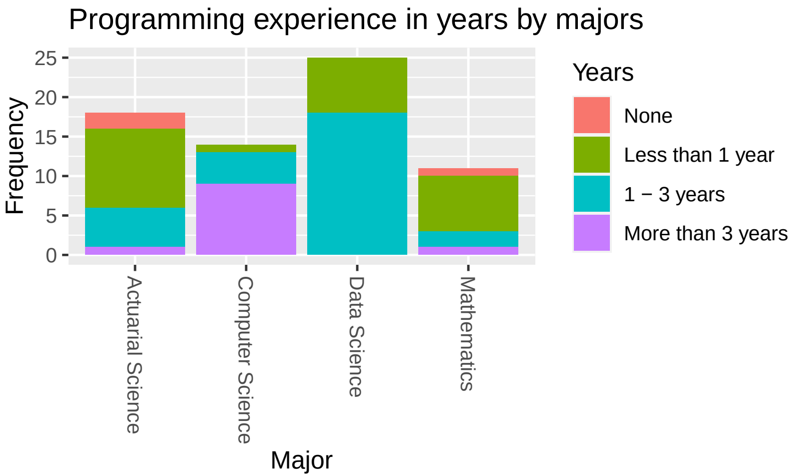
More about You



More about You

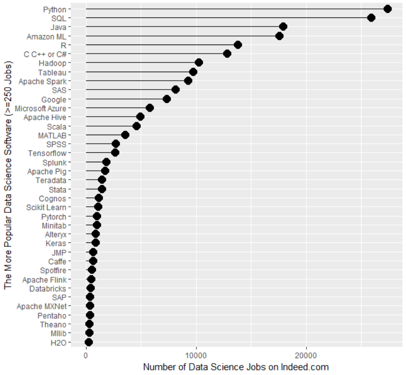


More about You



The Rising of R

- Indeed.com is the biggest job site in the U.S.
- Number of data sciences jobs posted between 2017 and 2019 for the popular software



What is RStudio

- Rstudio is an integrated development environment (IDE) for `R` to develop statistical programs
  - Developed by RSutdio, Inc. founded by Joesph Allaire
  - Hadley Wickham is the chief scientist
  - First appeared in February, 2011
  - Written in `C++`, `Java`, and `JavaScript`
  - Must be used alongside `R`

Analogy

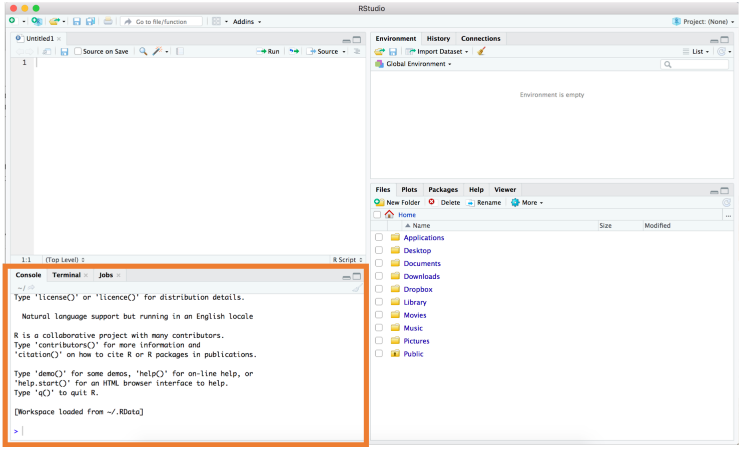
- R is an airplane
  - We can use R to go places
  - We gain skills and confidence with practice

Analogy

- RStudio is an airport
  - Provides support that makes our fly easier!
  - Not a great environment if we learn R without RStudio

RStudio

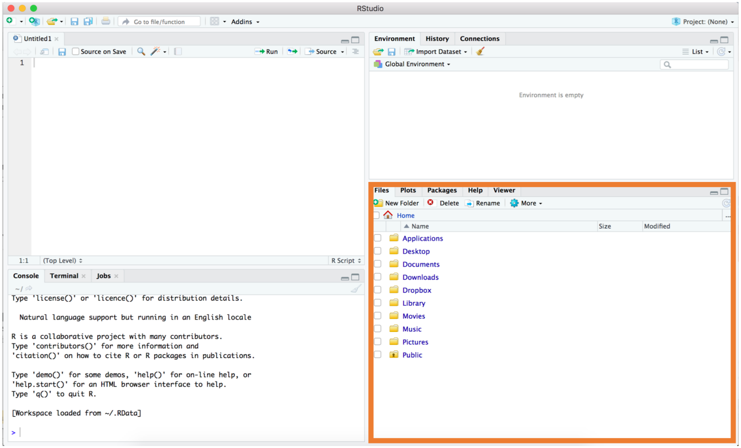
- Console: Run code here



```
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

>
```

RStudio

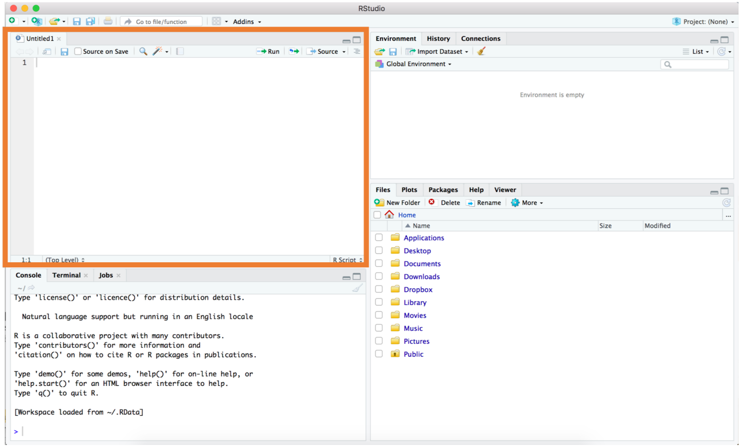
- Output: View plots and help files here



```
~/
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

>
```

RStudio

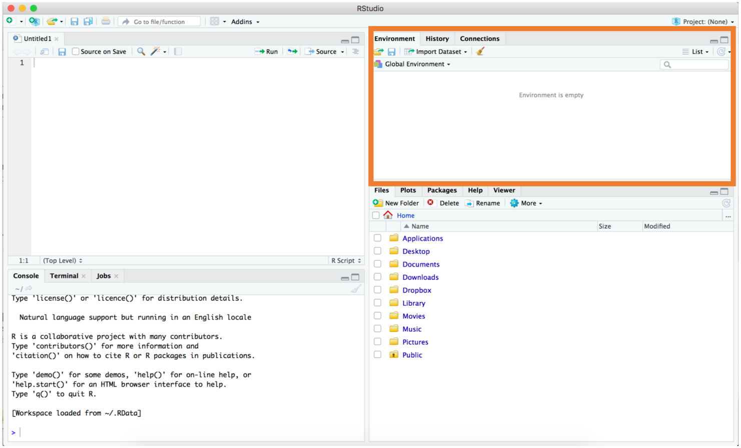
- Editor: Save code here



```
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

>
```

RStudio

- Environment: View environment and history here



```
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

>
```

Arithmetic Operators

- Suppose we have two arbitrary numbers, denoted by x and y
- Addition $x + y$:

  x `+` y

- Subtraction $x - y$:

  x `-` y

- Multiplication $x \times y$:

  x `*` y

- Division $x \div y$:

  x `/` y

- Exponentiation x^y: x`^`y or x`**`y

Arithmetic Operators

- Examples

```
2 + 3

2 * 3

2^3

2 + 3 - 4
2 + 3 * 4
```

- The answer to each question is printed starting with a `[1]`, which make sense once data vectors are explained

Arithmetic Operators

- Priority rule
  - `^` > `%%` > `*` = `/` > `+` = `-`
  - If =, then from left to right
- For overwrite the rule, use parentheses `()`
  - Always use `()` to prioritize operations

Arithmetic Operators

- Examples

```
# Are the following expressions the same?
2 + 3 - 4 * 5 / 100^1/2
(2 + 3) - 4 * 5 / 100^(1/2)
```

- All text in the line following the comment character # is treated as a comment for comprehension

Your Turn

- Rewrite each of the following R expressions as a math expression
  - `2 + 3 - 4 / 5 * 6`
  - `2 + 3 - 4 / 5 ** 6`
  - `2 / 3 / 4 / 5 / 6`

Your Turn

- Solutions
  - $2 + 3 - 4/5 * 6 \quad \rightarrow \quad (2 + 3) - \left(\frac{4}{5} \times 6\right)$
  - $2 + 3 - 4/5 * *6 \quad \rightarrow \quad (2 + 3) - \left(\frac{4}{5^6}\right)$
  - $2/3/4/5/6 \quad \rightarrow \quad \frac{2}{3 \times 4 \times 5 \times 6}$

Function Operators

- Support we have number x
- Absolute value $|x|$: `abs(`x`)`
- Square root $\sqrt{x}$: `sqrt(`x`)`
- Trigonometric: `sin(`x`)`, `cos(`x`)`, `tan(`x`)`
- Factorial $x!$: `factorial(`x`)`
- Natural exponentiation: `exp(`x`)`
- Logarithm: `log(`x`)`, `log2(`x`)`, and `log10(`x`)`
- Rounding: `round(`x`)`, `floor(`x`)`, and `ceiling(`x`)`

Function Operators

- Examples

```
# Absolute
abs(-10 + 2)

# Sqaure root
sqrt(4)
4^(1/2)
sqrt(-4)
squareroot(4)
```

- Warnings and errors: When R finds a command that it doesn't understand
  - Read the warning and error message carefully
  - Ask R for help by using `help()` function
  - What else?

Function Operators

The internet will make those bad words go away
Essential
Googling the Error Message
O RLY?
The Practical Developer
@ThePracticalDev

Function Operators

- Required vs. optional arguments
- Logarithm: `log(`x`, base = `y`)`
- Rounding value: `round(`x`, digits = `y`)`, where $y \in \mathbb{N}$
- Examples

```
log(100)
log(100, base = 2)

round(pi)
floor(pi)
ceiling(log2(100))
```

Function Operators

- Priority rule
  - From inside to outside
- All functions are used in a similar manner
  - Followed by a pair of parentheses `()`
  - Required arguments, e.g. `round(10.567)`
  - Optional arguments with default values, e.g. `round(10.567, digit = 0)`
  - Functions in existing `R` packages or create you own

Your Turn

- Use R to find the natural constant e and round it to 3 digits
- Use R to find the Archimedes' constant π and round it to 3 digits

Your Turn

- Solutions

```
# The natural constant
round(exp(1), 3)

# The Archimedes' constant
round(pi, 3)
round(2 * asin(1), 3)
```

Analogy

- Functions are like pets

  - They don't come unless we call them by name (spelled properly)
  - They have a mouth that likes to be fed (the parentheses)
  - They will complain if they are not fed properly (warnings and errors)
  - They are always our friends and make our lives better

Functions

- Basic structure
  - Function name
  - Input arguments
  - Body (the code)
  - Output (final result)
- Example:

```
this.is.my.first.function = function(x, y) {
    return(x + y)
}
```

- Curly braces {} means "do it together"

Functions

- Example:
```
this.is.my.first.function = function(x, y) {
    return(x + y)
}
```
- Name
  - “Must” start with a letter
  - Cannot contain any mathematical and logical operators
  - Case matters
  - Abbreviated but as long as clear

Functions

- Example:
```
fun.1 = function(x, y) {
    return(x + y)
}
```
- Input arguments
  - First matches name, and then position
  - Position based matching is ok for required arguments of those common functions
  - Use default values if missing
```
fun.1 = function(x, y = 1) {
    return(x + y)
}
```

Functions

- Example:
```
fun.1 = function(x, y) {
    return(x + y)
}
```
- Body
    - The body could be hundred lines
    - Check each step as you go
    - Don't try and do too much at once!
    - "Wrap it up" as a function once everything works

Functions

- Example:
```
fun.1 = function(x, y) {
    return(x + y)
}
```
- Output
  - `return()`
  - `print()`
  - `plot()`
  - Simultaneously

Your Turn

- Use R to write a function of the distance between two numbers

$$|a - b|$$

Your Turn

- Solutions

```
dist = function(a, b) {
  return(abs(a - b))
}
```

Functions

- Why we need functions?
  - Reusable
  - Recursion

Logical Values

- Dialogue with `R`
  - Numeric response
  - Graphic response
  - True-or-false answers
- Logical values
  - True: `TRUE` or `T`
  - False: `FALSE` or `F`

Logical Operators

- Between two arbitrary numbers x and y
  - Less than: x `<` y
  - Less than or equal to: x `<=` y
  - Greater than: x `>` y
  - Greater than or equal to: x `>=` y
  - Equal to: x `==` y (not `=`, will introduce `=` in next class)
  - Not equal to: x `!=` y

Logical Operators

- Examples

```
# Comparison between numeric values
pi > 3
1 + 1 == 2
sin(pi/4) == sqrt(2)/2
```

Logical Operators

- Between two logical values x and y
  - Not: `!`x
  - And: x `&` y or x `&&` y
  - Or: x `|` y or x `||` y

Logical Operators

- Examples

```
# Comparsion between logical values
!TRUE
TRUE && TRUE
TRUE && FALSE
F | T
1 + 1 == 2 && 2 + 1 == 3

# Any nonzero numeric value is treated as
   TRUE
104 && 2 < 4
```

Condition Statements

- Decision making is an important part of all programming
- Fed by a logical value
- Three forms
  - If only: `if(){}`
  - If else: `if(){} else{}`
  - Ladder: `if(){} else if(){} ... else if(){} else{}`

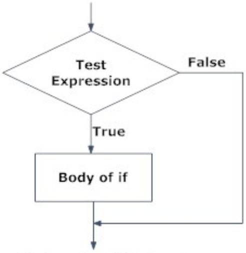


Fig: Operation of if statement

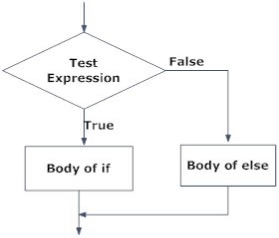


Fig: Operation of if...else statement

Condition Statements

- Examples

```
if(pi == 3.14){
  print("pi.is.3.14")
}

if(round(pi, digit = 2) == 3.14){
  print("pi.is.3.14")
}

if(3 > 0){
  print("3.is.larger.than.0")
} else {
  print("3.is.not.larger.than.0")
}
```

Your Turn

- Use R to write a function that indicate if a number is even or not

Your Turn

- Solution

```
is.even = function(x) {
  if (x %% 2 == 0) {
    print("x.is.an.even.number")
  } else {
    print("x.is.not.an.even.number")
  }
}
```

Your Turn

- Use R to get the i-th Fibonacci number
    - Fibonacci numbers: $0, 1, 1, 2, 3, 5, 8, 13, \ldots$

Your Turn

- Solution

```
Fibo = function(i) {
  if (i <= 2) {
    return (i - 1)
  } else {
    return (Fibo(i - 1) + Fibo(i - 2))
  }
}
```

After-class Reading

- *Using R for Introductory Statistics (1st Ed.)* by John Verzani
- Chapter 1 Data
  - Section 1.2 Some `R` essentials
    - Subsection 1.2.1 Starting `R`
    - Subsection 1.2.2 Using `R` as a calculator