T01: A brief introduction to R

MATH 4432 Statistical Machine Learning

HUANG XINRUI

MATH, HKUST

2023-09-12

What is it and why should I be using it?

What is R?

R is 'GNU S', a **freely available** language and environment for **statistical computing and graphics** which provides a wide variety of **statistical and graphical techniques** [1] :

- Linear and nonlinear modelling
- Statistical tests
- Time series analysis
- Classification
- Clustering
- ……

R is used among data miners, bioinformaticians and statisticians for **data analysis** and **developing statistical software**. [2]

[1] From CRAN

[2] From Wikipedia

Now let's become a useR!

Download and install R

The Comprehensive R Archive Network (CRAN)

Built in RGui: not recommended

Download and install RStudio

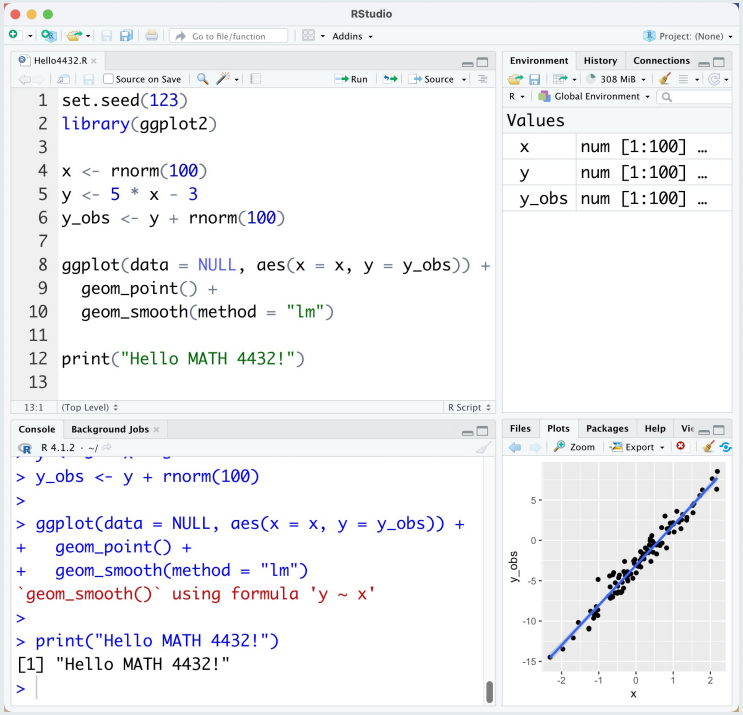
RStudio is an integrated development environment (IDE) for R.

It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management. *

[*] From RStudio

RStudio has 4 main windows ('panes')

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



Now let's have a try!

- Create a new **R Script** (.R) file from the menu `File -> New File -> R Script`
- Type `print("Hello MATH 4432!")`
- Type `print("Hello R!")`
- Click the `Run` option in the top right of the **Source** window

```
print("Hello MATH 4432!")
```

[1] "Hello MATH 4432!"

```
print("Hello R!")
```

[1] "Hello R!"

- Save your **R Script** file as `hello.R`

R Markdown

Why use 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

I recommend you do your homework assignment via a single R Markdown file, in which you can include your solutions, then output a report and submit it on Canvas!

In-class exercise: create your first R Markdown file!

- Create a new **R Markdown** file from the menu `File -> New File -> R Markdown ...`
- Edit **metadata**

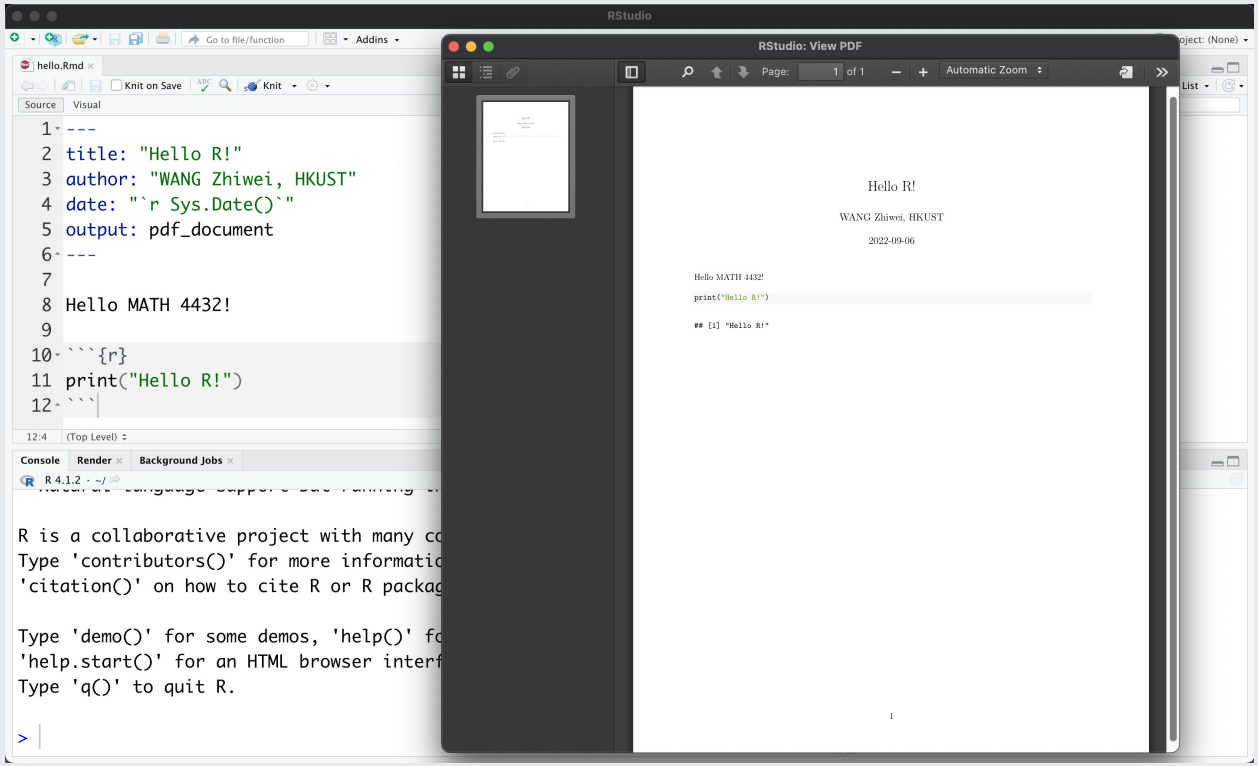
```
---
title: "Hello R!"
author: "HUANG XINRUI, HKUST"
date: "2023-09-12"
output: pdf_document
---
```

- **Text** `Hello MATH 4432!`
- **R code**

````
```{r}
print("Hello R!")
```
````

Output the report

- Click the `Knit` option to produce a PDF file



- Save your Rmd file as `hello.Rmd`

Data types

All data is represented in binary format, by bits (TRUE/FALSE, YES/NO, 1/0)

Booleans

Direct binary values: `TRUE` or `FALSE` in R

```
TRUE
```

[1] TRUE

```
FALSE
```

[1] FALSE

```
T
```

[1] TRUE

```
F
```

[1] FALSE

Integers

The number zero (`0`), a positive natural number (`1, 2, 3, etc.`) or a negative integer with a minus sign (`-1, -2, -3, etc.`), represented by a fixed-length block of bits

```
0L
```

[1] 0

```
1L
```

[1] 1

```
as.integer(-100)
```

[1] -100

Floating point numbers

Positive or negative whole number with a decimal point

```
1
```

[1] 1

```
1.23
```

[1] 1.23

```
5e-8
```

[1] 5e-08

Characters

Fixed-length blocks of bits, with special coding

Strings = sequences of characters

```
"Hello MATH 4432!"
```

[1] "Hello MATH 4432!"

```
"Hello R!"
```

[1] "Hello R!"

```
'HKUST'
```

[1] "HKUST"

Special values

Missing or **ill-defined values** and others

```
NA
```

[1] NA

```
NaN
```

[1] NaN

```
0 / 0
```

[1] NaN

```
1 / 0
```

[1] Inf

Some useful functions about data types

typeof()

Returns the type of the input

```
typeof(TRUE)
```

[1] "logical"

```
typeof(3L)
```

[1] "integer"

```
typeof(3)
```

[1] "double"

```
typeof("HKUST")
```

[1] "character"

`is.foo()`

Return Booleans for whether the argument is of type `foo`

```
is.character(3)
```

[1] FALSE

```
is.character("3")
```

[1] TRUE

```
is.nan(0 / 0)
```

[1] TRUE

```
is.na(3 / 0)
```

[1] FALSE

as.foo()

Tries to “cast” its argument to type `foo` — to translate it sensibly into a `foo`-type value

```
as.character(1 / 3)
```

[1] "0.333333333333333"

```
as.numeric(as.character(1 / 3))
```

[1] 0.3333333

```
3 * as.numeric(as.character(1 / 3))
```

[1] 1

Operators

Arithmetic operators

Binary usual arithmetic operators, take two numbers and give a number

- Addition
- Subtraction
- Multiplication
- Division
- Exponentiation
- .....

```
5 + 3
```

[1] 8

```
5 - 3
```

[1] 2

```
5 * 3
```

[1] 15

```
5 / 3
```

[1] 1.666667

```
5 ^ 3
```

[1] 125

Comparison operators

Binary operators

Take two objects, like numbers, and give a Boolean

```
5 > 3
```

[1] TRUE

```
5 < 3
```

[1] FALSE

```
5 >= 5
```

[1] TRUE

```
5 == 3
```

[1] FALSE

```
5 != 3
```

[1] TRUE

Think about why!

```
0.3 - 0.7 + 0.4 == 0
```

[1] FALSE

```
sqrt(2)^2 == 2
```

[1] FALSE

- Keywords: **floating point precision**, **finite precision**

```
0.3 - 0.7 + 0.4
```

[1] 5.551115e-17

```
sqrt(2)^2 - 2
```

[1] 4.440892e-16

- Try to use `all.equal()` function

```
all.equal(0.3 - 0.7 + 0.4, 0)
```

[1] TRUE

```
all.equal(sqrt(2)^2, 2)
```

[1] TRUE

Boolean operators

! **NOT**, **unary** for arithmetic **negation**

```
! TRUE
```

[1] FALSE

```
! F
```

[1] TRUE

& **AND** (**conjunction**), **binary**

```
TRUE & TRUE
```

[1] TRUE

```
TRUE & FALSE
```

[1] FALSE

| **OR** (**disjunction**), **binary**

```
FALSE | TRUE
```

[1] TRUE

```
FALSE | FALSE
```

[1] FALSE

More complicated cases

- TRUE **AND** TRUE

```
(3 < 5) & (3 * 5 == 15)
```

[1] TRUE

- FALSE **AND** TRUE

```
(3 > 5) & (3 * 5 == 15)
```

[1] FALSE

- FALSE **OR** TRUE

```
(3 > 5) | (3 * 5 == 15)
```

[1] TRUE

Data can have names!

Variables

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

Some built-in variables

```
pi
```

[1] 3.141593

```
cos(pi)
```

[1] -1

```
letters
```

[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t" "u" "v" [23] "w" "x" "y" "z"

Assignment operator

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

```
a <- 1L
a
```

[1] 1

```
is.integer(a)
```

[1] TRUE

```
b = 5
a * b
```

[1] 5

Use descriptive variable names!

- Good: `quota_class <- 40`
- Bad: `qc <- 40`

Data structure

Vectors

A **vector** is a sequence of values, all of the **same type**, including `logical`, `integer`, `numeric`, `character`, etc.

`c()` function returns a vector containing all its arguments **in order**

```
students <- c("Jerry", "Tara", "Tom")
id <- c(1:3)
midterm <- c(80, 96, 75)
```

Type the variable name to display its value

```
students
```

[1] "Jerry" "Tara" "Tom"

```
id
```

[1] 1 2 3

Adding Elements

The function `c()` (for combine) can also be used to add elements to a vector

```
students
```

[1] "Jerry" "Tara" "Tom"

```
students <- c(students, "Zhiwei")
id <- c(id, 4)
midterm <- c(midterm, 59)
```

```
students
```

[1] "Jerry" "Tara" "Tom" "Zhiwei"

```
midterm
```

[1] 80 96 75 59

Indexing

- `vec[1]` is the 1st element, `vec[3]` is the 3rd

```
students[1]
```

[1] "Jerry"

```
students[3]
```

[1] "Tom"

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

[1] "Tara" "Zhiwei"

- `vec[-3]` is a vector containing all but the 3rd element

```
students[-3]
```

[1] "Jerry" "Tara" "Zhiwei"

Named components

You can give **names** to elements or components of vectors

```
students
```

[1] "Jerry" "Tara" "Tom" "Zhiwei"

```
names(midterm) <- students # Assign students' names to the score
```

```
midterm
```

Jerry Tara Tom Zhiwei 80 96 75 59

```
midterm["Zhiwei"] # Get score for Zhiwei
```

Zhiwei 59

```
midterm[c("Tara", "Zhiwei")] # Get scores for 2 students
```

Tara Zhiwei 96 59

Vector arithmetic

Operators apply to vectors **pairwise** or **elementwise**

```
final <- c(84, 90, 78, 61) # Final exam scores
final
```

[1] 84 90 78 61

```
midterm # Midterm exam scores
```

Jerry Tara Tom Zhiwei 80 96 75 59

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

Jerry Tara Tom Zhiwei 82.0 93.0 76.5 60.0

```
bonus <- 5
average + bonus
```

Jerry Tara Tom Zhiwei 87.0 98.0 81.5 65.0

Pairwise comparisons

Is the final score higher than the midterm score?

```
midterm
```

Jerry Tara Tom Zhiwei 80 96 75 59

```
final
```

[1] 84 90 78 61

```
final > midterm
```

Jerry Tara Tom Zhiwei TRUE FALSE TRUE TRUE

Boolean operators can be applied pairwise

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

Jerry Tara Tom Zhiwei FALSE FALSE TRUE TRUE

Functions on vectors

```
sum(vec)      # sum of vec
mean(vec)       # mean of vec
median(vec)     # median of vec
min(vec)      # the smallest element of vec
max(vec)      # the largest element of vec
sd(vec)       # the standard deviation of vec
var(vec)      # the variance of vec
length(vec)     # the number of elements in vec
sort(vec)       # returns the vec in sorted order
order(vec)    # returns the index that sorts the vector vec
unique(vec)     # lists the unique elements of vec
summary(vec) # gives a five-number summary
```

Matrix

As with atomic vectors, the elements of a matrix must be of the **same data type**

```
mat_1 <- matrix(c(1:6), nrow = 3, ncol = 2)
mat_1
```

```
 [,1] [,2]
```

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

Dimension of the matrix

```
dim(mat_1)
```

[1] 3 2

Elements of a matrix can be referenced by specifying the index

```
mat_1[2, 1]
```

[1] 2

Matrix operations

```
mat_2 <- matrix(c(1:6), nrow = 3, ncol = 2)
mat_2
```

```
 [,1] [,2]
```

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

```
mat_1 + mat_2 # Addition
```

```
 [,1] [,2]
```

[1,] 2 8 [2,] 4 10 [3,] 6 12

```
mat_1 - mat_2 # Substraction
```

```
 [,1] [,2]
```

[1,] 0 0 [2,] 0 0 [3,] 0 0

Matrix operations

Elementwise multiplication

```
mat_1 * mat_2
```

```
 [,1] [,2]
```

[1,] 1 16 [2,] 4 25 [3,] 9 36

Multiplication

```
mat_3 <- matrix(c(5:10), nrow = 2, ncol = 3)
mat_4 <- mat_1 %*% mat_3
mat_4
```

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

[1,] 29 39 49 [2,] 40 54 68 [3,] 51 69 87

Matrix operations

Transpose

```
t(mat_4)
```

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

[1,] 29 40 51 [2,] 39 54 69 [3,] 49 68 87

`diag()` function returns a vector containing the elements of the principal diagonal

```
diag(mat_4)
```

[1] 29 54 87

```
diag(t(mat_4))
```

[1] 29 54 87

Data Frame

`data.frame` is a very important data type in R

Creating `data.frame` by hand

```
grade <- data.frame(id = id,
                    midterm = midterm,
                    final = final,
                    average = average)
rownames(grade) <- students
grade
```

Elements of `data.frame` can also be referenced by specifying the row and the column index

```
grade[2, 3]
```

[1] 90

Data Frame

Refer to columns

```
grade[, 4]
```

[1] 82.0 93.0 76.5 60.0

```
grade[, "average"]
```

[1] 82.0 93.0 76.5 60.0

```
grade[["average"]]
```

[1] 82.0 93.0 76.5 60.0

```
grade$average
```

[1] 82.0 93.0 76.5 60.0

Data Frame

Add a new row

```
# new row
Can <- data.frame(id = 5,
                  midterm = 99,
                  final = 99,
                  average = 99)
rownames(Can) <- "Can"

grade <- rbind(grade, Can)
grade
```

Data Frame

Add a new column

```
grade <- cbind(grade,
               gender = c("male", "female", "male", "male", "male"))
grade
```

Transfer `gender` to `factor` mode

```
grade[, "gender"] <- factor(grade[, "gender"], levels = c("female", "ma
grade
```

Data Frame

Actually we usually `data.frame` by `read.csv()` and `read.table()`, i.e. when importing the data into R

```
df <- read.csv("<filename>")
df <- read.table("<filename>")
```

When meeting big data, `fread()` function from `data.table` package will be helpful!

```
dt <- fread("<filename>")
```

Packages

Install and library packages

Many useful R functions come in packages, free libraries of code written by R's active user community.

To install an R package, open an R session and type at the command line

```
install.packages("<package name>")
```

R will download the package from CRAN

Once you have a package installed, you can make its contents available to use in your current R session by running

```
library(<package name>)
```

Recommended packages

`tidyverse`, `stringr`, `ggplot2`, `data.table`, `devtools`, etc.

ggplot2

Base-R | Code | Plot | ggplot2 | Code | Plot

We first use `plot()` function in Base-R

ggplot2

Base-R | Code | Plot | ggplot2 | Code | Plot

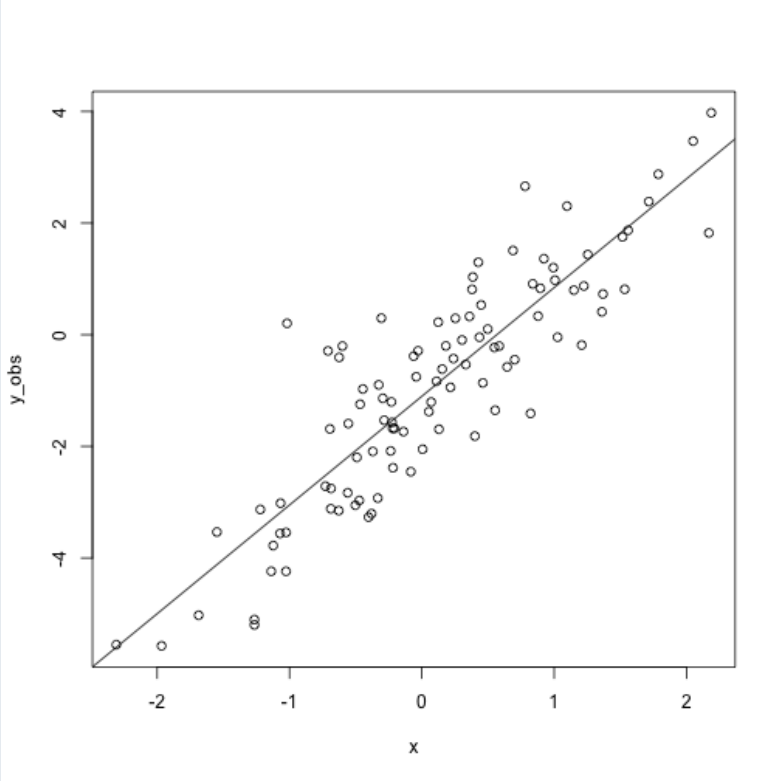
```r
set.seed(123)

# Generate data
x <- rnorm(100) # 100 samples
y <- 2 * x - 1 # Linear function
y_obs <- y + rnorm(100) # Add noise

# Plot
plot(x = x, y = y_obs)
abline(lm(y_obs ~ x))
```

ggplot2

Base-R Code Plot ggplot2 Code Plot



ggplot2

Base-R Code Plot ggplot2 Code Plot

ggplot2 a famous package for making beautiful graphics

ggplot2

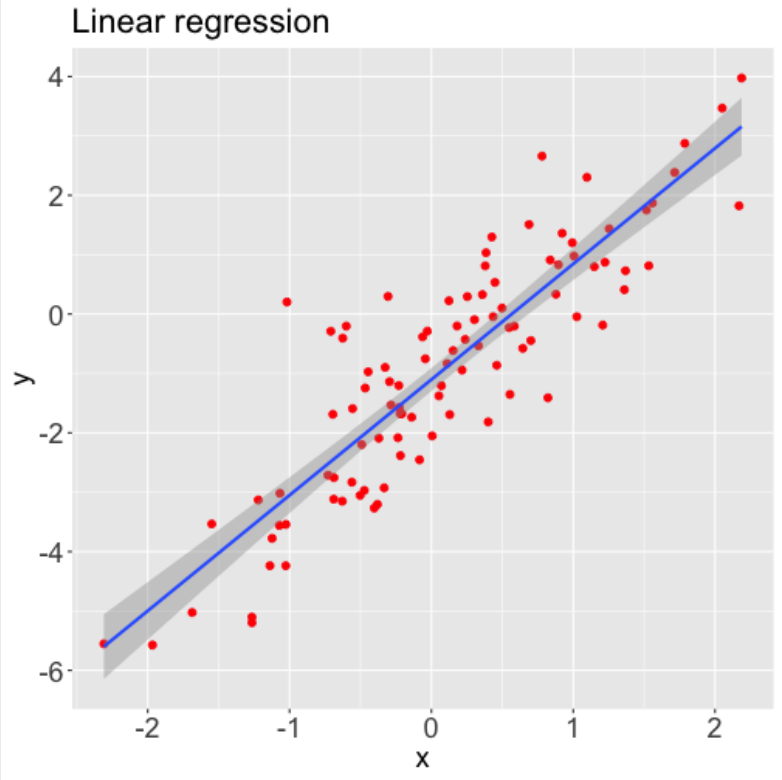
Base-R | Code | Plot | ggplot2 | Code | Plot

```r
# install.packages("ggplot2")
library(ggplot2)
ggplot(data = NULL, aes(x = x, y = y_obs)) +
  geom_point(color = "red", size = 2) +
  geom_smooth(method = "lm") +
  ggtitle("Linear regression") +
  ylab("y") +
  theme(
    text = element_text(size = 18),
    axis.title = element_text(size = 18),
    axis.text.y = element_text(size = 18),
    axis.text.x = element_text(size = 18)
    )
```

ggplot2

Base-R Code Plot ggplot2 Code Plot

```
#> `geom_smooth()` using formula 'y ~ x'
```



leaflet

Leaflet is one of the most popular open-source `JavaScript` libraries for interactive maps.

This R package makes it easy to integrate and control Leaflet maps in R.

```
# install.packages("leaflet")
library(leaflet)
leaflet() %>%
  addTiles() %>%
  setView(lng = 114.2654614760902,
          lat = 22.33639483440875,
          zoom = 14)
```

Helpful resources

References

- 94-842: Programming in R for Analytics, Instructor: Prof. Alexandra Chouldechova, Carnegie Mellon University
- R语言忍者秘笈 (Written in Chinese), 谢益辉, 肖楠
- Programming with R, Software Carpentry

Other resources

- Advanced R, Hadley Wickham (Chief Scientist at RStudio, PBC)
- Yihui Xie's homepage, Yihui Xie (Software Engineer at RStudio, PBC)
- 统计之都 (Capital of Statistics), an online community on statistics in China

Thank you!

Slides created via Yihui Xie's R package **xaringan**.

Theme customized via Garrick Aden-Buie's R package **xaringanthemer**.

Tabbed panels created via Garrick Aden-Buie's R package **xaringanExtra**.

The chakra comes from remark.js, **knitr**, and R Markdown.