Course_Notes

Ruhika Chatterjee

2024-12-07

Notes taken from Johns Hopkins University Coursera course series Data Science Specialization.

Intro Information

Every Data Science Project starts with a question that is to be answered with data

- That means that forming the question is an important first step in the process.
- The second step is finding or generating the data you're going to use.
- With the question solidified and data in hand, the data are then analyzed first by exploring the data and then often by modeling the data.
- After drawing conclusions, the project has to be communicated to others.
- Most projects build off someone else's work. It's really important to give credit.

R project: Data, Scripts, Output

Troubleshooting

- Check error messages and outputs
- Talk to your rubber duck
- https://www.r-project.org/help.html
- Use help() function and ? command (i.e. help(lm) or help("lm"), or ?lm or ¿'lm")
- Use Google or search.r-project.org
- Check forums StackOverflow and CrossValidated

How to effectively ask questions on forums

- The question you are trying to answer
- How you approached the problem, what steps you have already taken

- What steps will reproduce the problem (including sample data!)
- What was the expected output
- What you saw instead (including any error messages you received!)
- What troubleshooting steps you have already tried
- Details about your set-up, eg: OS, Rversion, packages
- Be specific in the title of your questions!
- Read the forum posting guidelines, ask your question on an appropriate forum!
- Be explicit, detailed, courteous, and succinct
- Follow up on the post OR post the solution

Good to know: Know Thy System

- How much memory is available?
- Applications in use?
- Other users in the system?
- OS? 32 or 64-bit?
- memory used for numeric data frame = 2 * rows * columns * 8byte/numeric / 2^20 bytes/MB / 2^10 GB/MB

Data Science Essentials

Types of Questions

- Descriptive: describe or summarize a set of data by generating simple summaries about the samples and their measurements, usually measures of central tendency (eg: mean, median, mode) or measures of variability (eg: range, standard deviations or variance). No interpretation involved.
- Exploratory: examine or explore the data and find relationships that weren't previously known, analyzing correlative (not causative) relationships. Allows formulation of hypotheses and can drive the design of future studies and data collection.
- Inferential: use a relatively small sample of data to infer or say something about the population at large through statistical modelling. Finding population-wide estimate and uncertainty using a representative sample.
- Predictive: use current data to make predictions about future data, or the likelihood of future outcomes, through past patterns of relevant variables. Best models involve more data and simple models, noting that prediction is based on correlative relationship.

- Causal: see what happens to one variable when we manipulate another variable, looking at the cause and effect of a relationship. Usually uses results of randomized studies that were designed to identify causation, using aggregate results (i.e. causation may not apply to everyone). Data collection is a challenge.

- Mechanistic: understand the exact changes in variables that lead to exact changes in other variables. In deterministic relationships for simple situations or in those that are nicely modeled by deterministic equations (i.e. only noise in measurement error). Cannot use for inference, only really used in physical or engineering sciences.

Experimental Design

- Organizing an experiment so that you have the correct and sufficient data. to clearly and effectively answer your data science question. Ask question, create setup, identify problems or sources of error, collect data.

- Keep track of independent and dependent variables, decide what variables you will measure, and which you will manipulate to effect changes in other measured variables.

- Develop your hypothesis, decide sample size.

- Determine confounders, control for it by measuring confounding variables or fixing the variable. Consider control groups, including blinding to combat placebo effect; randomization to remove confounder bias and systemic error; and replication to measure variability accurately and establish significance. These strategies remove the effect of confounding.

- Share data and code.

- Beware of p-hacking. p-value: value that tells you the probability that the results of your experiment were observed by chance, where significance is usually set below 0.05. p-hacking involves testing hypotheses till one reveals a statistically significant p-value.

Big Data

Difficulty in volume, velocity, and variety. Also move from structured data (long tables, spreadsheets, or databases with columns and rows of information that you can sum or average or analyse however you like within those confines) to unstructured data. Big data is big, changing, from many sources, and messy. But volume of data can counteract the messiness, changing data allows real-time analysis, can answer more type of questions, can uncover hidden correlations.

Intro to R

- CRAN: https://cran.r-project.org/

- RStudio: https://posit.co/products/open-source/rstudio/

History of R

- R is a dialect of S - language developed by John Chambers at Bell Labs for statistical analysis (lots of corporate acquisitions and versions)

- Developed to allow users to progress to programmers
- R developed in 1991 at Aukland, New Zealand by Ihaka and Gentleman.
- Similar syntax and superficial semantics to S, run on any standard OS, frequent releases. Lean base software, sophisticated graphics, interactive with powerful programming language, active community.
- Free and freedom to run program as desired, study and adapt program, redistribute copies, and improve program.
- Drawbacks: based on 40 year old tech, lacking dynamic/3D graphics, functionality based on consumer demand and user contribution, objects stored on physical memory, not always ideal.
- To understand computations in R, two slogans are helpful: 1. Everything that exists is an object. 2. Everything that happens is a function call.

Helpful Books

- Chambers (2008). *Software for Data Analysis*, Springer.
- Chambers (1998). *Programming with Data*, Springer.
- Venables & Ripley (2002). *Modern Applied Statistics with S*, Springer.
- Venables & Ripley (2000). *S Programming*, Springer.
- Pinheiro & Bates (2000). *Mixed-Effects Models in S and S-PLUS*, Springer.
- Murrell (2005). *R Graphics*, Chapman & Hall/CRC Press.
- Springer's series *Use R!*

R Markdown

- https://rmarkdown.rstudio.com/
- **bold** and *italics*
- Parameter {r Markdown, echo = False} to not print code
- Ctrl+Alt+I (Windows) to initiate code block
- note: add 2 spaces after bullet text

```
#install.packages("rmarkdown")
```

Coding Standards

- Write text in text editor
- Indent your code (4-8)

- Limit width of code (80 cols)
- Limit length of function (1 activity, single page)

Debugging

Problem

- message: no stop, message function.
- warning: unexpected, not fatal, end of function, warning function.
- error: fatal, stop function.
- Condition: classes listed above or creatable, generic.

```
log(-1) # returns NaN and warning about args
```

```
## Warning in log(-1): NaNs produced
```

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

```
printmessage <- function(x) {
  if (x > 0) print("x > 0")
  else print(" x <= 0")
  invisible(x) # return no print
}
#printmessage(NA) # comparison not possible, error
```

How to know something is wrong?

- What was the input? How was the function called?
- What was expected? i.e. output, message, other result
- What was done?
- What was the difference between expected and actual results?
- Were the expectations correct?
- Is the problem reproducible?

Tools

Use in commandline and in code ####traceback - print function call stack after an error. Call immediately after error. Useful when functions call functions.
- lm(y - x)
- traceback()
- prints list of function calls, error is in the top function

debug

- flags function fro debug mode to step through execution line-by-line
- debug(lm)
- lm(y - x)
- prints function code then browser prompt (workspace in function environment).
- Press n for next to iterate through the lines. Can call dubugger on functions inside debugger.

browser

- suspends execution of function wherever called and enters debug mode

trace

- insert debugging code into function in specific places

recover

- modify error behavior to browse function call stack. Set global option for all command-line calls
- options(error = recover)
- Prints call stack options, select level to browse environment of function

Version Control

- GitHub Repository Instruction: https://docs.github.com/en/get-started/start-your-journey/hello-world
- Git Download: https://git-scm.com/downloads

To configure Git in Git Bash

- git config –global user.name "Jane Doe"
- git config –global user.email janedoe@gmail.com
- git config –list confirm changes
- exit exit Terminal

Link

- Link RStudio to Git: Tools > Global Options > Git/SVN then confirm directory
- Link to GitHub: "Create RSA Key", close, "View public key", in GitHub Account Settings > "SSH and GPG keys" > "New SSH key", paste, title

Linking Repositories

- To Link Directly: Create repository in GitHub, copy URL, create new project in RStudio (Version control), Git as VC software, paste URL
- Stage, commit, push (with a message explaining what changed, why and by whom) Save, Git Tab check file as "Staged", click Commit, Commit message, Commit, Push
- Clone Repository: RStudio File > New Project > Version Control, Git, URL, location, create

Add existing Project to GitHub

in Git Bash

- cd ~/dir/name/of/path/to/file #navigate Terminal
- git init #initializes as git repo
- git add . #adds directory files to repo
- git commit -m "Initial commit" #commit

in GitHub: create repo (same name, no .readme, .gitignore, license), select "Push an existing repository from the command line", copy the code, reopen all

Swirl

https://swirlstats.com/

```
# install.packages("swirl", repos = "http://cran.us.r-project.org")
# install_from_swirl("R Programming") # install course
# library("swirl") # load interactive
# Swirl() # launch interactive
# help.start() # help
# play() # to leave lesson
# nxt() # to return to lesson
# Esc to return to R prompt
# bye() # exit and save
# skip() # skip current question
# main() # swirl main menu
# info() # display options
```

Scoping

```
search() # provides list for environments
```

```
## [1] ".GlobalEnv"        "package:stats"     "package:graphics"
## [4] "package:grDevices" "package:utils"     "package:datasets"
## [7] "package:methods"   "Autoloads"         "package:base"
```

- Global environment is the workspace.
- Scoping begins in global environment ends in base package, order depends on configuration of setup (loading package), inserts in slot 2 in reverse order.
- Separate namespace for function and non-function objects.
- Functions: Scoping in R is lexical/static scoping (also Python, Lisp). Local variables assigned in the body of a function. Formal arguments defined in args passed. Free variable not explicitly defined, in R looks in environment where function was defined then parent environments till reach global environment/namespace of package, then down from top-level environment to empty environment. Finally throws error if nothing found.
- Dynamic scoping looks in calling environment first.
- Environment: collection of symbol-value pairs.
- Every environment (except empty environment) has parent environment (can have multiple children).
- Parent environment: environment in which the function was called.
- Take function and associate with environment is a function closure.
- **THEREFORE**: define free variables in workspace and define function in global environment. EXCEPT when defining functions inside functions.
- R objects must be stored in memory. Functions carry pointer to defining environment.
- Optimization: pass function to functions optim, nlm, optimize. Object function depends on parameters and data.
- **Course 2, Module 2, scoping Rules- Optimization Example**
- Example:

```
make.power <- function(n) {
  pow <- function(x) { # x is free variable
    x^n
  }
  pow # returns function as object
}

cube <- make.power(3)
square <- make.power(2)
cube(3)
```

```
## [1] 27
```

```
square(3)
```

```
## [1] 9
```

R Packages

- Repositories: CRAN, BioConductor (bioinformatics), GitHub
- Search: https://www.rdocumentation.org/
- Base packages: utils, stats, datasets, graphics, grDevices, grid, methods, tools, parallel, compiler, splines, tcltk, stats4.
- Recommended packages: boot, class, cluster, codetools, foreign, KernSmooth, lattice, mgcv, nime, rpart, survival, MASS, spatial, nnet, Matrix.

```
# Install from CRAN:
#   install.packages("ggplot2", repos = "http://cran.us.r-project.org") #install
#   install.packages(c("labeling","tibble"), repos = "http://cran.us.r-project.org") #multiple

# Install from Bioconductor
#   install.packages("BiocManager", repos = "https://bioconductor.org/biocLite.R")
#   BiocManager::install(c("GenomicFeatures", "AnnotationDbi")) #install package

# Install from GitHub (need package, author name)
#   install.packages("devtools", repos = "http://cran.us.r-project.org") #only once
#   library(devtools)
#   install_github("author/package") #installs package

# library(ggplot2)# Load package, careful of dependencies
# installed.packages() #check installed packages
# library() #alternate
# old.packages(repos = "http://cran.us.r-project.org") #check packages to update
# update.packages(repos = "http://cran.us.r-project.org") #update all packages
# install.packages("ggplot2") #to update single package
# detach("package:ggplot2", unload=TRUE) #unload function
# remove.packages("ggtree") #remove package
# help(package = "ggplot2") #package info
# browseVignettes("ggplot2") #extended help files
```

R Profiler and Optimization

- Systematic way to examine time spent in various part of the program. Useful to optimize the code.
- DON'T PREMATURELY OPTIMIZE
- Measure, not guess, data on what needs to be optimized.
- User time: computer experienced, may be greater if multiple cores/processors (accessible in multi-threaded BLAS libraries). Elapsed time: wall-clock time, may be greater if other computing tasks.

```
system.time(read.csv("hw1_data.csv")) # returns seconds to execute, if error then seconds to error. Wra
```

```
##    user  system elapsed
##       0       0       0
```

```r
data(mtcars)
Rprof() # track function call stack at intervals (def = 0.02 sec), time spent in functions.

fit <- lm(mtcars$mpg ~ mtcars$cyl)

Rprof(NULL)

summaryRprof() # makes Rprof readable, tabluates, time in each function
```

```
## $by.self
## [1] self.time  self.pct   total.time total.pct
## <0 rows> (or 0-length row.names)
##
## $by.total
## [1] total.time total.pct  self.time  self.pct
## <0 rows> (or 0-length row.names)
##
## $sample.interval
## [1] 0.02
##
## $sampling.time
## [1] 0
```

```r
# $by.total - divides time spent per function by total run time
# $by.self - same as by.total but first subtracts time spent in function above in call stack. Helps tar

rm(list=ls())
```