

Reproducible Research Course Notes

Xing Su

Contents

Replication	3
Reproducibility	3
Literate/Statistical Programming	4
Structure of Data Analysis	5
Key Challenge in Data Analysis	5
Define The Question	5
Determine the Ideal Data Set	5
Determine What Data You Can Access	5
Obtain The Data	5
Clean The Data	5
Exploratory Data Analysis	6
Statistical Modelling/Prediction	7
Interpret Results	8
Challenge Results	9
Synthesize/Write Up and Results	9
Create Reproducible Code	9
Organizing Data Analysis	10
Coding Standards in R	11
Markdown (documentation)	11
R Markdown	11
Communicating Results	15
Hierarchy of Information - Research Paper	15
Hierarchy of Information - Email Presentation	15
RPods (link)	15
Reproducible Research Checklist	16
Do's	16
Don'ts	17
Replication vs Reproducibility	18
Background and Underlying Trend for Reproducibility	18
Problems with Reproducibility	18
Evidence-based Data Analysis	19

Caching Computations	20
cachier Package	20
Case Study: Air Pollution	24
Analysis: Does Nickel Make PM Toxic?	24
Conclusions and Lessons Learnt	26
Case Study: High Throughput Biology	27
Conclusions and Lessons Learnt	27

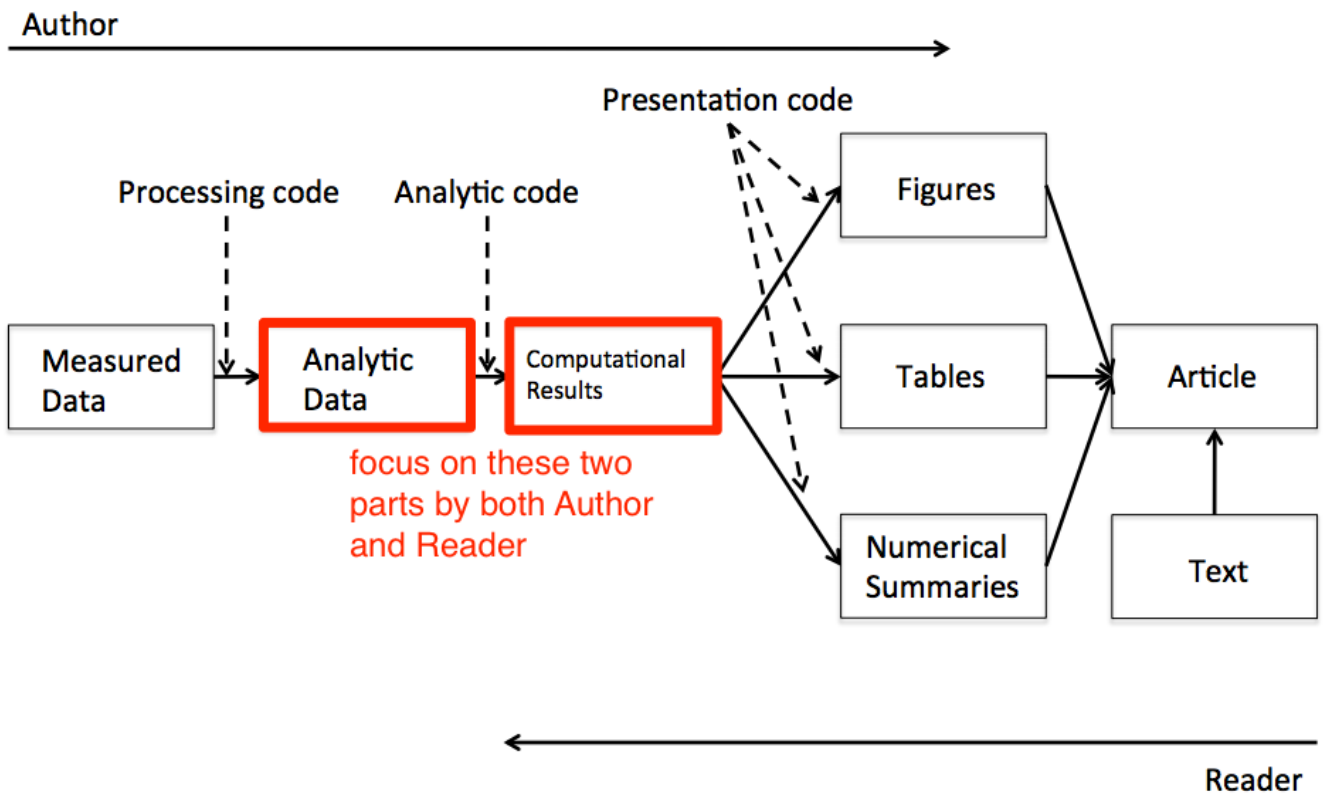
Replication

- ultimate standard for strengthening of scientific evidence
- reproduce findings and conduct studies with independent investigators/data/analytical methods/laboratories/instruments
- particularly important in studies that affect policy
- HOWEVER, some studies can't be replicated (time, opportunity, money, unique)
 - Make research reproducible (making available data/code so others may reproduce analysis)

Reproducibility

- bridges gap between replication and letting the study stand alone
- take data/code and replicate findings → validate their findings
- why we need reproducible research
 - new technologies increase data throughput while adding complexities and dimension to data
 - existing databases merged into bigger collection of data
 - computational power increased
- **Research Pipeline**

Loading required package: pacman



- Institute of Medicine report - Evolution of Translational Omics
 - Data/meta data should be made available
 - Computer code/fully specified computational procedures need to be available
 - Data processing and computational analysis steps described
- **Necessities for Reproducible Research**
 - analytical data (not raw data)
 - analytical code (applied to data, i.e. regression model etc.)
 - documentation of data/code
 - standard means of distribution
- **Challenges**
 - authors make considerable effort to make data/results available
 - readers must download data/results individually and piece it all together
 - readers may not have the same resources as authors (computing clusters, etc.)
 - few tools for readers/authors
 - In reality, authors just put their content online (often disorganized, but there are some central databases for various fields) and readers have to find the data and piece it together

Literate/Statistical Programming

- Notion of an article is a stream of **text** and **code**
- original idea came from *Don Knuth*
- statistical analysis is divided into text and code chunks
 - text explains whats going on
 - code loads data/computes results
 - presentation code formats results (tables, graphics, etc)
- Literate Programming requires
 - **documentation language** - [*weave*] produce human-readable docs (HTML/PDF)
 - **programming language** - [*tangle*] produce machine-readable code
- **Sweave** - original program in R designed to do literate programming
 - LaTeX for documentation, R for programming
 - developed by *Friedrich Leisch* (member of R core)
 - still maintained by R core
 - **limitations**
 - * focus on LaTeX, difficult to learn markup languages
 - * lacks features like caching, multiple plots per chunk, mixing programming languages with many other technical terms
 - * not frequently updated/actively developed
- **knitr** = alternative literate programming package
 - R for programming (can use others as well)
 - LaTeX/HTML/Markdown for documentation
 - developed by Iowa grad student *Yihui Xie*
- **reproducible research** = minimum standard for non-replicable/computationally intensive studies
 - infrastructure still need for creating/distributing reproducible documents

Structure of Data Analysis

Key Challenge in Data Analysis

“Ask yourselves, what problem have you solved, ever, that was worth solving, where you knew all of the given information in advance? Where you didn’t have a surplus of information and have to filter it out, or you had insufficient information and have to go find some?” - *Dan Myer*

Define The Question

- most powerful dimension reduction tool, narrowing down question reduces noise/simply problem
- Science → Data → Applied Statistics → Theoretical Statistics
- properly using the combination of science/data/applied statistics = proper data analysis
- *example*
 - general question: Can I automatically detect emails that are SPAM from those that are not?
 - concrete question: Can I use quantitative characteristics of the emails to classify them as SPAM/HAM?

Determine the Ideal Data Set

- after having the concrete question, find the data set that is suitable for your goal:
 - *Descriptive* = whole population
 - *Exploratory* = random sample with multiple variables measured
 - *Inferential* = drawing conclusion from a sample for a larger population, so choosing the right population, and carefully/randomly sampled
 - *Predictive* = need training and test data sets from the same population to build a model and classifier
 - *Causal* = (if this is modified, then that happens) experimental data from randomized study
 - *Mechanistic* = data from all components of system that you want to describe

Determine What Data You Can Access

- often won’t have access to the ideal data set
- free data from the web, paid data from provider (must respect terms of use)
- if data doesn’t exist, you may need to generate some of the data

Obtain The Data

- try to obtain the raw data and reference the source
- if loading data from internet source, record at the minimum url and time of access

Clean The Data

- raw data need to be processed to be fed into modeling program
- if already processed, need to understand how and understand documentation on how the pre-processing was done
- understanding source of data (i.e. survey - how survey was done?)
- may need to reformat/subsample data → need to record steps
- determine if data is good enough to solve the problem → if not, find other data/quit/change question to avoid misleading conclusions

Exploratory Data Analysis

- for the purpose of the SPAM question, data needs to be split into training and test test (Predictive)

```
# If it isn't installed, install the kernlab package with p_load()  
p_load("kernlab")
```

```
##  
## The downloaded source packages are in  
## 'private/var/folders/1b/8brmpmcd0fqbn4v_cl5qrz80000gn/T/Rtmp1RyS7D/downloaded_packages'
```

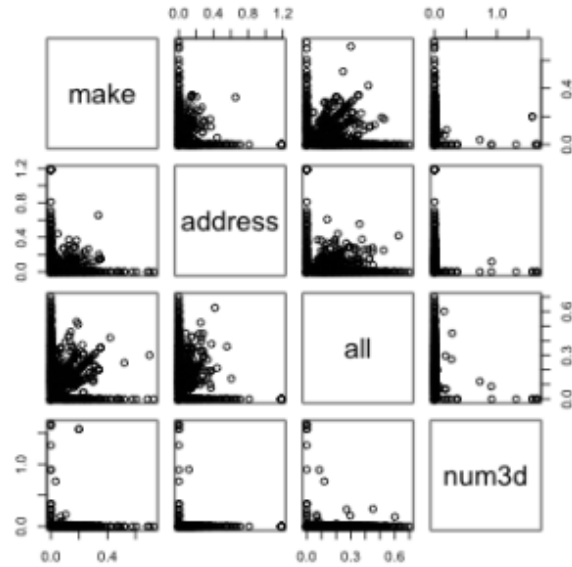
```
##  
## kernlab installed
```

```
data(spam)  
set.seed(3435)  
trainIndicator = rbinom(4601, size = 1, prob = 0.5)  
table(trainIndicator)
```

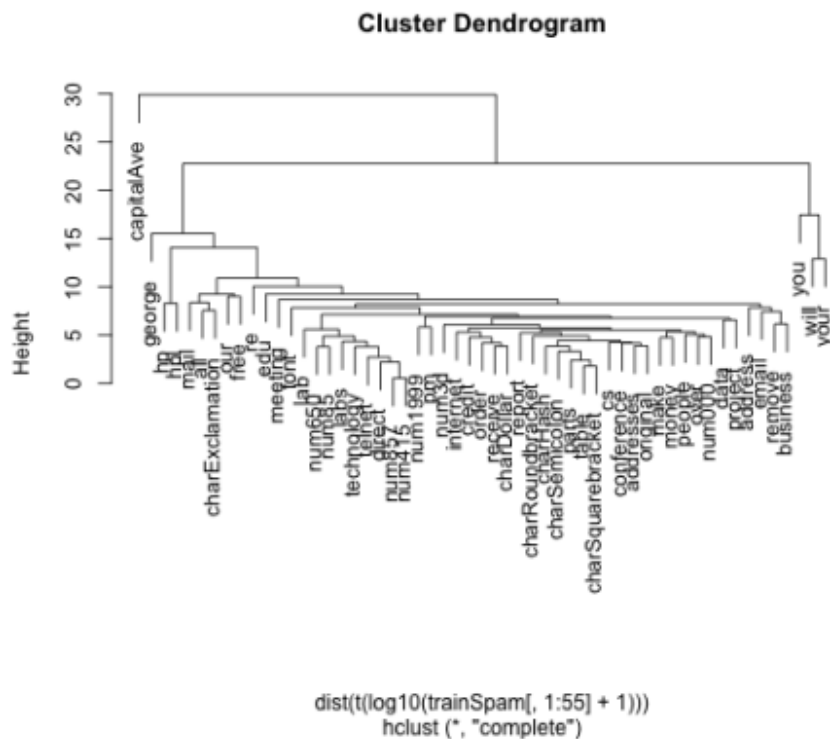
```
## trainIndicator  
##      0      1  
## 2314 2287
```

```
trainSpam = spam[trainIndicator == 1, ]  
testSpam = spam[trainIndicator == 0, ]
```

- look at one/two-dimensional summaries of data, what the distribution looks like, relationships between variables
 - names(), summary(), head()
 - table(trainSpam\$type)
- check for missing data/why there is missing data
- create exploratory plots
 - plot(trainSpam\$capitalAve ~ trainSpam\$type)
 - plot(log10(trainSpam\$capitalAve + 1) ~ trainSpam\$type)
 - * because data is skewed → use log
 - * a lot of zeroes → +1
- perform exploratory analysis (clustering)
 - relationships between predictors
 - plot(log10(trainSpam[, 1:4] + 1))
 - * log transformations diagrams between pairs of variables



- hierarchical cluster analysis (**Dendrogram**) = what words/characteristics tend to cluster together
 - `hCluster = hclust(dist(t(trainSpam[, 1:57])))`
 - * **Note:** Dendrograms can be sensitive to skewness in distribution of individual variables, so maybe useful to transform predictor (see below)
 - `hClusterUpdated = hclust(dist(t(log10(trainSpam[, 1:55] + 1))))`



Statistical Modelling/Prediction

- should be guided by exploratory analysis

- exact methods depend on the question of interest
- account for data transformation/processing
 - go through each variable in the dataset and fit logistic regression to see if we can predict if an email is spam or not by using a single variable

```
trainSpam$numType = as.numeric(trainSpam$type) - 1
costFunction = function(x, y) sum(x != (y > 0.5))
cvError = rep(NA, 55)
p_load("boot")
for (i in 1:55) {
  # creates formula with one variable and the result
  lmFormula = reformulate(names(trainSpam)[i], response = "numType")
  glmFit = glm(lmFormula, family = "binomial", data = trainSpam)
  # cross validated error
  cvError[i] = cv.glm(trainSpam, glmFit, costFunction, 2)$delta[2]
}
# Which predictor has minimum cross-validated error?
names(trainSpam)[which.min(cvError)]
```

```
## [1] "charDollar"
```

- think about measures/sources of uncertainty

```
# Use the best model from the group
predictionModel = glm(numType ~ charDollar, family="binomial", data=trainSpam)
# Get predictions on the test set
predictionTest = predict(predictionModel, testSpam)
predictedSpam = rep("nonspam", dim(testSpam)[1])
# Classify as 'spam' for those with prob > 0.5
predictedSpam[predictionModel$fitted > 0.5] = "spam"
# Classification table
table(predictedSpam, testSpam$type)
```

```
##
## predictedSpam nonspam spam
##      nonspam    1346  458
##      spam       61   449
```

```
# Error rate
(61 + 458)/(1346 + 458 + 61 + 449)
```

```
## [1] 0.2242869
```

Interpret Results

- use appropriate language and not go over what you performed
 - describes, correlates/associated with, lead to/causes, predicts
- give explanation of the results (why certain models predict better than others)
- interpret coefficients

- The fraction of characters that are dollar signs can be used to predict if an email is Spam
 - Anything with more than 6.6% dollar signs is classified as Spam
 - More dollar signs always means more Spam under our prediction
- interpret measures of uncertainty to calibrate your interpretation of results
 - Our test set error rate was 22.4%

Challenge Results

- good to challenge the whole process/all results you found (because somebody else will)
 - question → data source → processing → analysis → conclusion
- challenge measures of uncertainty and choices in what to include in the model
- think about alternative analyses, maybe useful to try in case they produce interesting/better models/predictions

Synthesize/Write Up and Results

- tell coherent story of the most important part of analysis
- lead with question → provide context to better understand framework of analysis
 - Can I use quantitative characteristics of the emails to classify them as SPAM/HAM?
- summarize the analysis into the story
 - Collected data from UCI → created training/test sets
 - Explored relationships
 - Choose logistic model on training set by cross validation
 - Applied to test, 78% test set accuracy
- don't need every analysis performed, only include the ones that are needed for the story or address a challenge
 - Number of dollar signs seems to indicate SPAM, “Make money with Viagra \$ \$ \$ \$!”
 - 78% isn't that great
 - I could use more variables
 - Why logistic regression?
- order analysis according to story rather than chronologically
- include well defined figures to help understanding

Create Reproducible Code

- document your analysis as you do them using markdown or `knitr`
 - preserve any R code or written summary in a single document using `knitr`
- ensure all analysis is reproducible (standard that most data analysis are judged on)

Organizing Data Analysis

- key data analysis files
 - **data:** raw
 - * should be store in analysis folder
 - * if accessed from the web, include in README file the URL, where the data is from, what the data is, brief description of what it is for, date of access
 - * if data is store in Git repository, can use the log to talk about the above info
 - **data:** processed
 - * should be names so that its easy to see what script generated what data
 - * In README, important to document what code files were used to transform raw data to processed data
 - * processed data should be tidy/organized
 - **figures:** exploratory
 - * made during process of analysis, not necessarily all part of final report
 - * don't need to be well formatted/annotated
 - * need to be usable enough so that the author can easily understand what is being presented and how to reproduce it
 - **figures:** final
 - * more polished, better organized, more readable, possibly multiple panels
 - * usually a small subset or original exploratory figures (typical journal article contains 5-6 figures)
 - * clearly labeled and well annotated, axes/color set to make figure clear
 - **R code:** raw/unused scripts
 - * could be less commented, multiple versions
 - * may include analyses that are later discarded
 - * important to name/document appropriately to understand what was performed
 - * placed in a separate part of the data analysis directory (separate from final)
 - **R code:** final scripts
 - * clearly commented - small comments for what/when/why/how, big comment blocks for whole sections to explain what is being done
 - * include processing details for the raw data
 - * should pertain to only analyses that appear in in the final write up
 - **R code:** R markdown files
 - * not necessary or require but useful to summarize parts of analysis
 - * can be used to generate reproducible reports, as it can embed code and text into single document and processing the document into readable webpage/pdf doc
 - * easy to create in Rstudio
 - **text:** README file
 - * explain what is going on in project directory
 - * not necessary if you have R markdown files (don't separate analysis and code - literate programming principle)
 - * should contain step-by-step instructions for how analysis was conducted, what code files are called first, what are used to process the data, what are used to fit models, what are used to generate figures, etc.
 - **text:** text of analysis/report
 - * should include title, introduction/motivation for problem, the method (statistical method), the results (including measures of uncertainty) and conclusions (including pitfalls/problems)

- * coherent story from all analysis performed, but does not need to include all analysis performed but only the most important and relevant parts
- * should include references for statistical methods, software packages, implementation that were used (for reproducibility)
- **Resources**
 - [Project template](#) = a external R package
 - [Managing a statistical analysis project guidelines and best practices](#)

Coding Standards in R

- write code in text editor and save as text file
- indenting (4 space minimum)
- limit the width of your code (80 columns)
- limit the length of functions

Markdown ([documentation](#))

- Markdown = text-to-HTML conversion tool for web writers.
 - simplified version of markup language
 - write using easy-to-read, easy-to-write plain text format
 - any text editor can create markdown document
 - convert the text to structurally valid XHTML/HTML
 - created by John Gruber
- **italics** = `*text*`
- **bold** = `**text**`
- **main heading** = `#Heading`
- **secondary heading** = `##Heading`
- **tertiary main heading** = `###Heading`
- **unordered list** = `- first element`
- **ordered list** = `1. first element`
 - the number in front actually doesn't matter, as long as there is a number there, markdown will be compiled to an ordered list (no need for renumbering)
- **links** = `[text](url)`
 - OR `[text][1]` → later in the document, define all of the links in this format: `[1]: url "text"`
- **new lines** = requires a *double space* after the end of a line

R Markdown

- integration of R code and markdown
 - allows creating documents containing “live” R code
 - R code is evaluated as a part of the processing of the markdown
 - results from R code inserted into markdown document
 - core tool for literate statistical programming
 - **pros**
 - * text/code all in one place in logical order

- * data/results automatically updated to reflect external changes
- * code is live = automatic test when building document
- **cons**
 - * text/code all in one place, can be difficult to read especially if there's a lot of code
 - * can substantially slowdown processing of documents
- **knitr package**
 - written by YihuiXie, built into RStudio
 - support Markdown, LaTeX, HTML as documentation languages
 - Exports PDF/HTML
 - good for manuals, short/medium technical documents, tutorials, periodic reports, data preprocessing documents/summaries
 - not good for long research articles, complex time-consuming computations, precisely formatted documents
 - evaluates R markdown documents and return/records the results, and write out a Markdown files
 - Markdown file can then be converted into HTML using markdown package
 - solidify package converts R markdown into presentation slides
- In RStudio, create new R Markdown files by clicking New → R Markdown
 - ===== indicates title of document (large text)
 - $\$expression\$$ = indicates LaTeX expression/formatting
 - `'text'` = changes text to code format (typewriter font)
 - `"{r name, echo = FALSE, results = hide}..."` = R code chunk
 - * **name** = name of the code chunk
 - * **echo = FALSE** = turns off the echo of the R code chunk, which means display only the result
 - * **Note:** by default code in code chunk is echoed = print code AND results
 - * **results = hide** = hides the results from being placed in the markdown document
 - inline text computations
 - * `'r variable'` = prints the value of that variable directly inline with the text
 - incorporating graphics
 - * `"{r scatterplot, fig.height = 4, fig.width = 6} ... plot() ..."` = inserts a plot into markdown document
 - **scatterplot** = name of this code chunk (can be anything)
 - **fig.height = 4** = adjusts height of the figure, specifying this alone will produce a rectangular plot rather than a square one by default
 - **fig.width = 6** = adjusts width of the figure
 - * knitr produces HTML, with the image embedded in HTML using base64 encoding
 - does not depend on external image files
 - not efficient but highly transportable
 - incorporating tables (xtable package: `install.packages("xtable")`)
 - * **xtable** prints the table in html format, which is better presented than plain text normally

```
p_load("datasets")
p_load("xtable")
fit <- lm(Ozone ~ Wind + Temp + Solar.R, data = airquality)
xt <- xtable(summary(fit))
print(xt, "html")
```

Estimate

Std. Error

t value

$\Pr(>|t|)$

(Intercept)

-64.3421

23.0547

-2.79

0.0062

Wind

-3.3336

0.6544

-5.09

0.0000

Temp

1.6521

0.2535

6.52

0.0000

Solar.R

0.0598

0.0232

2.58

0.0112

- **setting global options**

- “`{r setoptions, echo = FALSE} opts_chunk$set(echo = FALSE, results = "hide")`” = sets the default option to not print the code/results unless otherwise specified

- **common options**

- **output:** `results = "asis" OR "hide"`
 - * `"asis"` = output to stay in original format and not compiled into HTML
- **output:** `echo = TRUE OR FALSE`
- **figures:** `fig.height = numeric`
- **figures:** `fig.width = numeric`

- **caching computations**

- add argument to code chunk: `cache = TRUE`
- computes and stores result of code the first time it is run, and calls the stored result directly from file for each subsequent call
- useful for complex computations
- caveats:

- * if data/code changes, you will need to re-run cached code chunks
 - * dependencies not checked explicitly (changes in other parts of the code → need to re-run the cached code)
 - * if code does something outside of the document (i.e. produce a png file), the operation cannot be cached
- “Knit HTML” button to process the document
 - alternatively, when not in RStudio, the process can be accomplished through the following

```
p_load("knitr")
setwd(<working directory>)
knit2html("document.Rmd")
browseURL("document.html")
```

- processing of `knitr` documents
 - author drafts R Markdown (.Rmd) → `knitr` processes file to Markdown (.md) → `knitr` converts file to HTML
 - **Note:** *author should NOT edit/save the .md or .html document until you are done with the document*

Communicating Results

- when presenting information, it is important to breakdown results of an analysis into different levels of granularity/detail
- below are listed in order of *least* → *most specific*

Hierarchy of Information - Research Paper

- **title/author list** = description of topic covered
- **abstract** = a few hundred words about the problem, motivation, and solution
- **body/results** = detailed methods, results, sensitivity analysis, discussion/implication of findings
- **supplementary material** = granular details about analysis performed and method used
- **code/detail** = material to reproduce analysis/finding

Hierarchy of Information - Email Presentation

- **subject line/sender** = must have, should be concise/descriptive, summarize findings in one sentence if possible
- **email body** = should be 1-2 paragraphs, brief description of problem/context, summarize findings/results
 - If action needs to be taken, suggest some options and make them as concrete as possible
 - If questions need to be addressed, try to make them yes/no
- **attachment(s)** = R Markdown/report containing more details, should still stay concise (not pages of code)
- **links to supplementary materials** = GitHub/web link to code, software, data

RPubs ([link](#))

- built-in service with RStudio and works seamlessly with `knitr`
- after knitting the R Markdown file in HTML, you can click on **Publish** to publish the HTML document on RPub
- **Note:** *all publications to RPub are publicly available immediately*

Reproducible Research Checklist

- **Checklist**

- Are we doing good science?
- Was any part of this analysis done by hand?
 - * If so, are those parts *precisely* document?
 - * Does the documentation match reality?
- Have we taught a computer to do as much as possible (i.e. coded)?
- Are we using a version control system?
- Have we documented our software environment?
- Have we saved any output that we cannot reconstruct from original data + code?
- How far back in the analysis pipeline can we go before our results are no longer (automatically) reproducible?

Do's

- ***start with good science***

- work on interesting problem (to you & other people)
- form coherent/focused question to simplify problem
- collaborate with others to reinforce good practices and habits

- ***teach a computer***

- worthwhile to automate all tasks through script/programming
- code = precise instructions to process/analyze data
- teaching the computer almost guarantee's reproducibility
- `download.file("url", "filename")` = convenient way to download file
 - * full URL specified (instead series of links/clicks)
 - * name of file specified
 - * directory specified
 - * code can be executed in R (as long as link is available)

- ***use version control***

- GitHub/BitBucket are good tools
- helps to slow down
 - * forces the author to think about changes made and commit changes and keep track of analysis performed
- helps to keep track of history/snapshots
- allows reverting to old versions

- ***keep track of software environment***

- some tools/datasets may only work on certain software/environment
 - * software and computing environment are critical to reproducing analysis
 - * everything should be documented
- **computer architecture:** CPU (Intel, AMD, ARM), GPUs, 32 vs 64bit
- **operating system:** Windows, Mac OS, Linux/Unix
- **software toolchain:** compilers, interpreters, command shell, programming languages (C, Perl, Python, etc.), database backends, data analysis software
- **supporting software/infrastructure:** Libraries, R packages, dependencies
- **external dependencies:** web sites, data repositories (data source), remote databases, software repositories

- **version numbers:** ideally, for everything (if available)
- `sessionInfo()` = prints R version, operating system, local, base/attached/utilized packages
- ***set random number generator seed***
 - random number generators produce pseudo-random numbers based on initial seed (usually number/set of numbers)
 - * `set.seed()` can be used to specify seed for random generator in R
 - setting seed allows stream of random numbers to be reproducible
 - whenever you need to generate stream of random numbers for non-trivial purpose (i.e. simulations, Markov Chain Monte Carlo analysis), ***always*** set the seed.
- ***think about entire pipeline***
 - data analysis is lengthy process, important to ensure each piece is reproducible
 - * final product is important, but the process is just as important
 - raw data → processed data → analysis → report
 - the more of the pipeline that is made reproducible, the more credible the results are

Don'ts

- ***do things by hand***
 - may lead to unreproducible results
 - * edit spreadsheets using Excel
 - * remove outliers (without noting criteria)
 - * edit tables/figures
 - * validate/quality control for data
 - downloading data from website (clicking on link)
 - * need lengthy set of instructions to obtain the same data set
 - moving/split/reformat data (no record of what was done)
 - if necessary, manual tasks must be documented precisely (account for people with different background/context)
- ***point and click***
 - graphical user interfaces (GUI) make it easy to process/analyze data
 - * GUIs are intuitive to use but actions are **difficult** to track and for others to reproduce
 - * some GUIs include log files that can be utilized for review
 - any interactive software should be carefully used to ensure all results can be reproduced
 - text editors are usually ok, but when in doubt, document it
- ***save output for convenience***
 - avoid saving data analysis output
 - * tables, summaries, figures, processed, data
 - output saved stand-alone without code/steps on how it is produced is not reproducible
 - when data changes or error is detected in parts of analysis the output is dependent on, the original graph/table will not be updated
 - intermediate files (processed data) are ok to keep but clear/precise documentation must be created
 - should save the **data/code** instead of the output

Replication vs Reproducibility

- **Replication**

- focuses on validity of scientific claim
- if a study states “if X is related to Y” then we ask “is the claim true?”
- need to reproduce results with new investigators, data, analytical methods, laboratories, instruments, etc.
- particularly important in studies that can impact broad policy or regulatory decisions
- stands as **ultimate standard** for strengthening scientific evidence

- **Reproducibility**

- focuses on validity of data analysis
- “Can we trust this analysis?”
- reproduce results new investigators, same data, same methods
- important when replication is impossible
- arguably a **minimum standard** for any scientific study

Background and Underlying Trend for Reproducibility

- some studies cannot be replicated (money/time/unique/opportunistic) → 25 year studies in epidemiology
- technology increases rate of collection/complexity of data
- existing databases merged to become bigger databases (sometimes used off-label) → administrative data used in health studies
- computing power allows for more sophisticated analyses
- “computational” fields are ubiquitous
- all of the above leads to the following
 - analysis are difficult to describe
 - inadequate training in statistics/computing and cause errors to be more easily produced through the pipeline
 - large data throughput, complex analysis, and failure to explain the process can inhibit knowledge transfer
 - results are difficult to replicate (knowledge/cost)
 - complicated analyses are not as easily trusted

Problems with Reproducibility

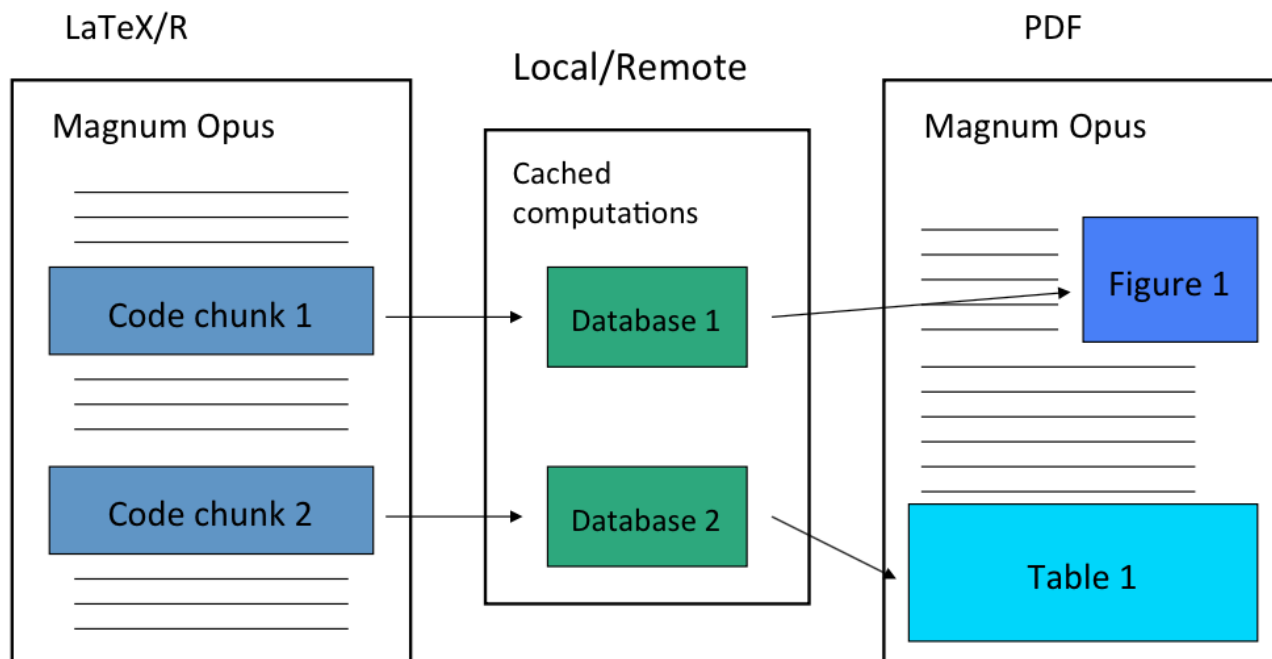
- aiming for reproducibility provides for
 - transparency of data analysis
 - * check validity of analysis → people check each other → “self-correcting” community
 - * re-run the analysis
 - * check the code for bugs/errors/sensitivity
 - * try alternate approaches
 - data/software/methods availability
 - improved transfer of knowledge
 - * focuses on “downstream” aspect of research dissemination (does solve problems with erroneous analysis)
- reproducibility does not guarantee
 - correctness of analysis (reproducible \neq true)
 - trustworthiness of analysis
 - deterrence of bad analysis

Evidence-based Data Analysis

- create analytic pipelines from evidence-based components – standardize it (“[Deterministic Statistical Machine](#)”)
 - apply thoroughly-studied (via statistical research), *mutually-agreed-upon methods* to analyze data whenever possible
 - once pipeline is established, shouldn’t change it (“transparent box”)
 - this reduces the “researcher degrees of freedom” (restricting the ability to alter different parts of analysis pipeline)
 - * analogous to a pre-specified clinical trial protocol
 - *example*: time series of air pollution/health
 - * *key question*: “Are short-term changes in pollution associated with short-term changes in a population health outcome?”
 - * long history of statistical research investigating proper methods of analysis
 - * pipeline
 1. check for outliers, high leverage, overdispersion → skewedness of data
 2. Fill in missing data? → NO! (systematic missing data, imputing would add noise)
 3. model selection: estimate degrees of freedom to adjust for unmeasured confounding variables
 4. multiple lag analysis
 5. sensitivity analysis with respect to unmeasured confounder adjustment/influential points
- this provides standardized, best practices for given scientific areas and questions
- gives reviewers an important tool without dramatically increasing the burden on them
- allows more effort to be focused on improving the quality of “upstream” aspects of scientific research

Caching Computations

- Literate (Statistical) Programming
 - paper/code chunks → database/cached computations → published report



cachier Package

- evaluates code in files and stores intermediate results in a key-value database
- R expressions are given ***SHA-1 hash values*** so changes can be tracked and code can be re-evaluated if necessary
- **cachier packages**
 - used by authors to create data analyses packages for distribution
 - used by readers to clone analysis and inspect/evaluate subset of code/data objects → readers may not have resources/want to see entire analysis
- **conceptual model**
 - dataset/code → source file → result
- **process for authors**
 - **cachier** package parses R source files and creates necessary cache directories/subdirectories

- if expression = never evaluated
 - * evaluate it and store any results R objects in cached data (similar to `cache = TRUE` function for `knitr`)
 - else if cached result exists
 - * lazy load the results from cache database and move on to next expression
 - if expression *does not* create any R objects (nothing to cache)
 - * add expression to list where evaluation needs to be forced
 - write out metadata for this expression to metadata file
 - `cachepackage` function creates `cacher` package storing
 - * source file
 - * cached data objects
 - * metadata
 - package file is zipped and can be distributed
- **process for readers**
 - a journal article can say that “the code and data for this analysis can be found in the cacher package 092dcc7dda4b93e42f23e038a60e1d44dbec7b3f”
 - * the code is the SHA-1 hash code and can be loaded/cloned using the `cacher` package
 - when analysis is cloned, local directories are created, source files/metadata are downloaded
 - * data objects are **NOT** downloaded by default
 - * references to data objects are loaded and corresponding data can be lazy-loaded on demand
 - when user examines it, then it is downloaded/calculated
 - readers are free to examine and evaluate the code
 - * `clone(id = "####")` = loads data from cache
 - using the first 4 characters is generally enough to identify
 - * `showfiles()` = lists R scripts available in cache
 - * `sourcefile("name.R")` = loads cached R file
 - * `code()` = prints the content of the R file line by line
 - * `graphcode()` = plots a graph to demonstrate dependencies/structure of code
 - * `objectcode("object")` = shows lines of code that were used to generate that specific object (tracing all the way back to reading data)
 - * `runcode()` = executes code by loading data from cached database (much faster than regular)
 - by default, expressions that result in object being created are **NOT** run and are loaded from cached databases
 - expressions not resulting in object **ARE** evaluated (i.e. plots)
 - * `checkcode()` = evaluates all expressions from scratch
 - results of evaluation are checked against stored results to see if they are the same
 - setting random number generators are critical for this to work
 - * `checkobjects()` = check for integrity of data objects (i.e. see if there are possible data corruption)
 - * `loadcache()` = loads pointers to data objects in the data base
 - when a specific object is called, cache is then transferred and the data is downloaded
 - **example**

```
p_load("cacher")
clonecache(id = "092dcc7dda4b93e42f23e038a60e1d44dbec7b3f")
clonecache(id = "092d") ## effectively the same as above
```

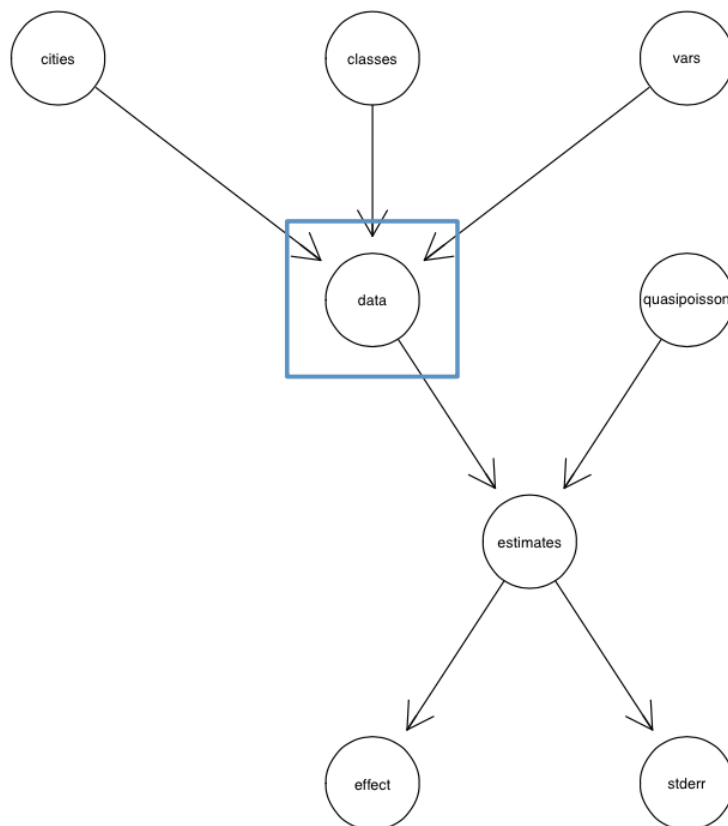
```

# output: created cache directory '.cache'
showfiles() # show files stored in cache
# output: [1] "top20.R"
sourcefile("top20.R") # load R script

code() # examine the content of the code
# output:
# source file: top20.R
# 1 cities <- readLines("citylist.txt")
# 2 classes <- readLines("colClasses.txt")
# 3 vars <- c("date", "dow", "death",
# 4 data <- lapply(cities, function(city) {
# 5 names(data) <- cities
# 6 estimates <- sapply(data, function(city) {
# 7 effect <- weighted.mean(estimates[1,
# 8 stderr <- sqrt(1/sum(1/estimates[2,

graphcode() # generate graph showing structure of code

```



```

objectcode("data")
# output:
# source file: top20.R

```

```

# 1 cities <- readLines("citylist.txt")
# 2 classes <- readLines("colClasses.txt")
# 3 vars <- c("date", "dow", "death", "tmpd", "rmtmpd", "dptp", "rmdptp", "l1pm10tmean")
# 4 data <- lapply(cities, function(city) {
#       filename <- file.path("data", paste(city, "csv", sep = "."))
#       d0 <- read.csv(filename, colClasses = classes, nrow = 5200)
#       d0[, vars]
#   })
# 5 names(data) <- cities

loadcache()
ls()
# output:
# [1] "cities"      "classes"      "data"         "effect"
# [5] "estimates" "stderr"       "vars"

cities
# output:
# / transferring cache db file b8fd490bcf1d48cd06...
# [1] "la"      "ny"      "chic"    "dlft"    "hous"    "phoe"
# [7] "staa"    "sand"    "miam"    "det"     "seat"    "sanb"
# [13] "sanj"    "minn"    "rive"    "phil"    "atla"    "oakl"
# [19] "denv"    "clev"

effect
# output:
# / transferring cache db file 584115c69e5e2a4ae5...
# [1] 0.0002313219

stderr
# output:
# / transferring cache db file 81b6dc23736f3d72c6...
# [1] 0.000052457

```

Case Study: Air Pollution

- reanalysis of data from MMAPS and link with PM chemical constituent data
 - Lippmann *et al.* found strong evidence that Ni modified the short-term effect of PM_{10} across 60 US communities
 - National Morbidity, Mortality, and Air Pollution Study (NMMAPS) = national study of the short-term health effects of ambient air pollution
 - * focused primarily on particulate matter (PM_{10}) and ozone (O_3)
 - * health outcomes included mortality from all causes and hospitalizations for cardiovascular and respiratory diseases
 - * Data made available at the Internet-based Health and Air Pollution Surveillance System (iHAPSS)

Research

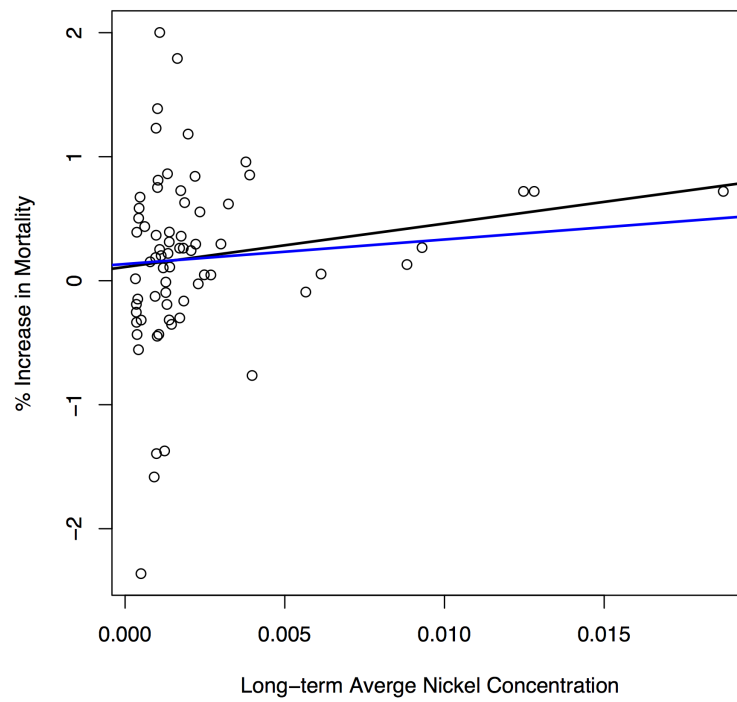
Does the Effect of PM_{10} on Mortality Depend on PM Nickel and Vanadium Content? A Reanalysis of the NMMAPS Data

Francesca Dominici,¹ Roger D. Peng,¹ Keita Ebisu,² Scott L. Zeger,¹ Jonathan M. Samet,³ and Michelle L. Bell²

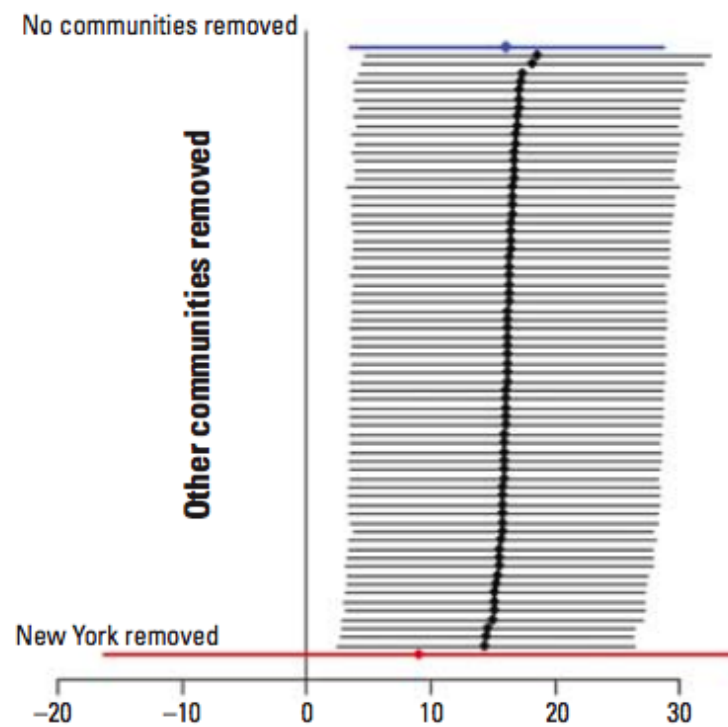
¹Department of Biostatistics, Johns Hopkins Bloomberg School of Public Health, Baltimore, Maryland, USA; ²School of Forestry and Environmental Studies, Yale University, New Haven, Connecticut, USA; ³Department of Epidemiology, Johns Hopkins Bloomberg School of Public Health, Baltimore, Maryland, USA

Analysis: Does Nickel Make PM Toxic?

- Long-term average nickel concentrations appear correlated with PM risk ($p < 0.01 \rightarrow$ statistically significant)
- there appear to be some outliers on the right-hand side (New York City)
 - adjusting the data by removing the New York counties altered the regression line (shown in blue)
 - though the relationship is still positive, regression line no longer statistically significant ($p < 0.31$)



- sensitivity analysis shows that the regression line is particularly sensitive to the New York data



Conclusions and Lessons Learnt

- New York has very high levels of nickel and vanadium, much higher than any other US community
- there is evidence of a positive relationship between Ni concentrations and PM_{10} risk
- strength of this relationship is highly sensitive to the observations from New York City
- most of the information in the data is derived from just 3 observations
- reproducibility of NMMAPS allowed for a secondary analysis (and linking with PM chemical constituent data) investigating a novel hypothesis (Lippmann *et al.*), as well as a critique of that analysis and new findings (Dominici *et al.*)
- original hypothesis not necessarily invalidated, but evidence not as strong as originally suggested (more work should be done)

Case Study: High Throughput Biology

Genomic signatures to guide the use of chemotherapeutics

ature.com/naturemedicine

Anil Potti^{1,2}, Holly K Dressman^{1,3}, Andrea Bild^{1,3}, Richard F Riedel^{1,2}, Gina Chan⁴, Robyn Sayer⁴,
Janiel Cragun⁴, Hope Cottrill⁴, Michael J Kelley², Rebecca Petersen⁵, David Harpole⁵, Jeffrey Marks⁵,
Andrew Berchuck^{1,6}, Geoffrey S Ginsburg^{1,2}, Phillip Febbo¹⁻³, Johnathan Lancaster⁴ &
Joseph R Nevins¹⁻³

- Potti *et al.* published in 2006 that microarray data from cell lines (the NCI60) can be used to define drug response “signatures”, which can be used to predict whether patients will respond
- however, analysis performed were fundamentally flawed as there exist much misclassification and mishandling of data and were unreproducible
- the results of their study, even after being corrected twice, still contained many errors and were unfortunately used as guidelines for clinical trials
- the fiasco with this paper and associated research, though spectacular in its own light, was by no means an unique occurrence

Conclusions and Lessons Learnt

- most common mistakes are simple
 - confounding in the experimental design
 - mixing up the sample labels
 - mixing up the gene labels
 - mixing up the group labels
- most mixups involve simple switches or offsets
 - this simplicity is often hidden due to incomplete documentation
- research papers should include the following (particularly those that would potentially lead to clinical trials)
 - data (with column names)
 - provenance (who owns what data/which sample is which)
 - code
 - descriptions of non-scriptable steps
 - descriptions of planned design (if used)
- **reproducible research** is the key to minimize these errors
 - literate programming
 - reusing templates
 - report structure
 - executive summaries
 - appendices (sessionInfo, saves, file location)