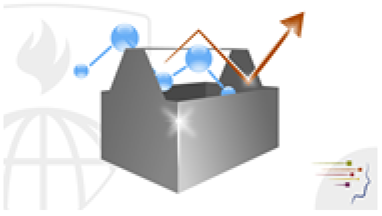
Coursera Johns Hopkins Specialization in Data Science course dependency information

There are nine courses in the sequence plus a capstone project course. For the courses, we consider two forms of dependency

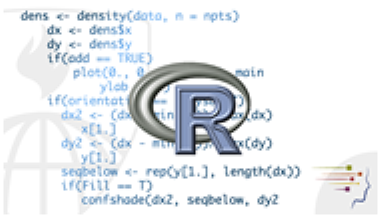
Hard dependency: Students will be *required* to know material from the prerequisite course. Taking the dependent course simultaneously will be challenging and only possible for highly motivated students willing to work ahead of the course schedule for the prerequisite. Taking hard dependent courses out of order is not possible unless the student *already knows* the material covered in the prerequisite course.

Soft dependency: Knowledge of material from the prerequisite course is recommended and useful. Concurrently taking the prerequisite course and the dependent course is possible. It is not recommended to take them out of order, but would be possible for highly motivated students willing to self teach components of the prerequisite course as needed.



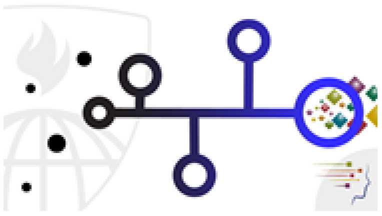
The Data Scientist's Toolbox

This is the primary introductory course for the specialization. It should be taken first and has no prerequisite courses. Students should be computer literate, have programmed in at least one computer language and be motivated self learners.



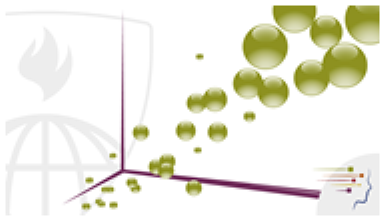
R Programming

This is the most crucial course for the remainder of the specialization. It is **softly dependent on The Data Scientist's Toolbox**. It should be taken before the remaining courses in the series.



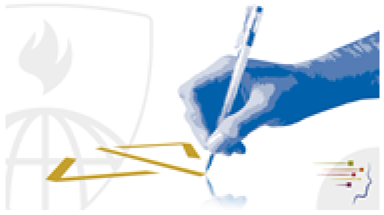
Getting and Cleaning Data

This course has **hard dependencies on R Programming and The Data Scientist's Toolbox.**



Exploratory Data Analysis

This course has **hard dependencies on R Programming and The Data Scientist's Toolbox.**

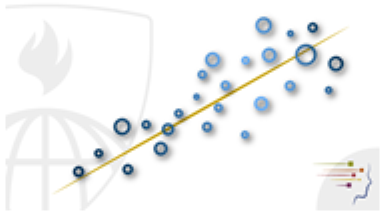


Reproducible Research

This course has **hard dependencies on R Programming and The Data Scientist's Toolbox.**

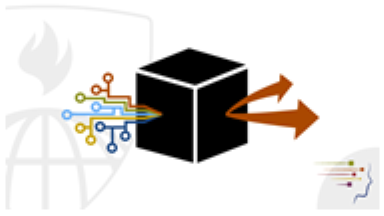
Statistical Inference

This course has **hard dependencies on R Programming and The Data Scientist's Toolbox.** In addition, students will need basic (non calculus) mathematics skills.



Regression Models

This course has **hard dependencies on R Programming**, **The Data Scientist's Toolbox and Statistical Inference.**



Practical Machine Learning

This course has **hard dependencies on R Programming**, **The Data Scientist's Toolbox and Regression Models.** It has a **soft dependency on Exploratory Data Analysis.**

Developing Data Products

This course has **hard dependencies on R Programming**, **The Data Scientist's Toolbox and Reproducible Research.** It has a soft dependency of **Exploratory Data Analysis.**

Student Handbook on

Referencing

September 2010 v1

Johns Hopkins Bloomberg School of Public Health
615 N Wolfe Street, Baltimore, MD 21205
This document is subject to revision.
For the latest version, please see:
https://my.jhsph.edu/Offices/academicaffairs/Research/Pages/default.aspx

Inside This Handbook

What is Plagiarism?

A Simple Definition

Plagiarism constitutes the majority of academic ethics violations at the School. Plagiarism is defined in the Student Policy and Procedure Memorandum on Academic Ethics as:

> "…taking for one's own use the words, ideas, concepts or data of another without proper attribution. Plagiarism includes both direct use or paraphrasing of the words, thoughts, or concepts of another without proper attribution. Proper attribution includes: (1) use of quotation marks or single-spacing and indentation for words or phrases directly taken from another source, accompanied by proper reference to that source and (2) proper reference to any source from which ideas, concepts, or data are taken even if the exact words are not reproduced." (The Johns Hopkins Bloomberg School of Public Health, Policy and Procedures Memorandum Students-1 Academic Ethics; October 2006)

Accurately and appropriately citing your sources is your best defense against any allegations of plagiarism. Ignorance of proper referencing standards or the failure to apply these standards for any reason is not a valid defense. The purpose of this handbook is to provide you with an overview of the school's standards and expectations regarding referencing and citation.

Plagiarism is a violation of academic integrity.

Why Do We Reference Sources?

Four Good Reasons

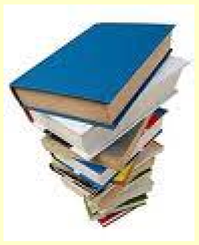
Acknowledging how the scholarship of others has contributed to your work is necessary to maintain both academic and professional integrity. According to Turabian (2007, 133), referencing:

Drawing from credible sources leads to increased credibility of your ideas.

Properly attributes words and ideas to their owners. People deserve acknowledgment for their words and ideas. Proper referencing assures that you sufficiently provide this acknowledgment. Failure to acknowledge the ways in which others have contributed to your work is analogous to stealing in the academic realm.

Enhances the credibility of your arguments. It is not only important to be accurate in the content of your work, but also to correctly indicate from where the content came. Proper referencing of content allows readers to judge the quality of your sources that have informed your work. Drawing from credible sources leads to increased credibility of your ideas.

Provides readers with a background into your area of interest. By providing readers with a complete and accurate portrayal of the sources you have consulted in your work, you provide them with insight into the range of sources that deal with your topic area and how your work is linked to the published literature in your area of interest.



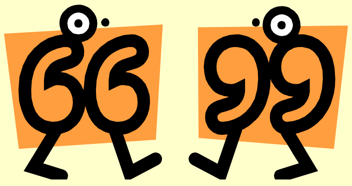
Advances your field of inquiry. Referencing provides readers with information to pursue other avenues of investigation in the same field. Researchers often formulate future endeavors based on previous scholarship, and your work combined with the references you used to develop your work may serve as a catalyst for others to explore further issues in the field that need to be addressed.

In addition to these reasons, as mentioned above, proper referencing is your best defense against charges of plagiarism.

When should we cite sources?

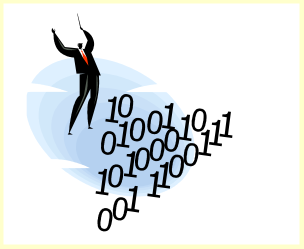
Citation is used to distinguish your words and ideas from the ideas and words of another. You should provide a citation in your academic work in four situations:

When you quote a source. You should clearly indicate words taken verbatim from another source by (1) placing quotation marks around the quoted material or using a block indent for longer quotes, AND (2) providing a citation for the quote. Providing only a citation for a quote without placing quotation marks or a block indent around the quoted words is not sufficient.



When you paraphrase a source. When you are using content and ideas from another source but placing them in your own words, you should cite the source.

When you summarize information from a source. When you condense the ideas from a source into a summary, you should cite the source.



When you use facts or data in your work. Facts that constitute "common knowledge" do not have to be cited. All other facts or data in a paper should be referenced. It's not always clear what facts constitute "common" knowledge", so if you are unsure it's best to consult your TA or professor. When in doubt, cite the information. Information that typically requires citation include: statistics, descriptions of specific methods or events, technical definitions, data results from experiments, and the opinions, arguments or reasoning of experts.

If you are in doubt about whether you should cite a source, the safest thing to do is to cite it.

What are some general rules for citation?

Over the years, various disciplines have developed different citation practices. In a multi-disciplinary field such as public health, you will find that different departments within our School may follow different styles of citation. Regardless of which citation style you decide to use, there are some common guidelines that you should follow.

Adhere to a bibliography style or a reference list style: In bibliography style, you indicate source content by placing a superscript number at the end of the sentence. You then provide a citation to the source through a footnote or through an endnote that corresponds to the superscript number. In reference list style, you indicate a source by placing a parenthetical notation that identifies the source at the end of the sentence. Depending on the particular citation style, relevant identifying information could include author, year of publication, and page number. Most bibliography and reference list styles have a bibliography or a reference list, which consists of a compilation of sources consulted with more detailed identifying information at the end of the document. See examples of both styles below.

REFERENCE LIST: CHICAGO STYLE

In text parenthetical reference: Developing cultural competency is important for lawyers and expert witnesses involved in capital defense cases (Perlin and McClain 2009, 257).

Corresponding reference list entry:
Perlin, Michael L. and Valerie McClain. 2009. "WHERE SOULS ARE FORGOTTEN: Cultural Competencies, Forensic Evaluations, and International Human Rights." *Psychology, Public Policy, and Law* 15: 257-277.

BIBLIOGRAPHY: CHICAGO STYLE

In text footnote: Developing cultural competency is important for lawyers and expert witnesses involved in capital defense cases.[1]

Footnote:
[1]Michael L. Perlin and Valerie McClain. "WHERE SOULS ARE FORGOTTEN: Cultural Competencies, Forensic Evaluations, and International Human Rights." *Psychology, Public Policy, and Law* 15 (2009): 257.

Corresponding bibliography entry:
Perlin, Michael L. and Valerie McClain. "WHERE SOULS ARE FORGOTTEN: Cultural Competencies, Forensic Evaluations, and International Human Rights." *Psychology, Public Policy, and Law* 15 (2009): 257-277.

Identifying source information: All citation styles provide enough information to allow the reader to locate the source of the information. Traditionally this information includes author, title, page numbers, and publication information.

Consistency: Once you have chosen a citation style to use, you should use the style accurately and consistently throughout the work.

Electronic sources and citation: Check with your particular citation style guide to properly cite electronic sources in your work. Links to web sites alone within the document are not a sufficient way to cite sources on the web. Most styles require at a minimum, url, access date and author or sponsor. **Please be aware that frequent citing of unpublished electronic sources, even if done accurately, may not earn high marks in your academic evaluation. Professors pay attention to the quality of the sources cited when evaluating student work. Wikipedia, for example, is rarely an acceptable source in academic writing. The highest quality sources are derived from peer reviewed academic and scholarly works.**

What are some commonly used citation styles?

Two citation styles which students commonly use are:

APA (American Psychological Association) Style
http://www.apastyle.org/

and

Chicago Style http://www.chicagomanualofstyle.org

While the APA is primarily geared toward writing for publication purposes, the Chicago style was adapted for use by student researchers by Kate Turabian. (See Turabian, Kate L. *A Manual for Writers of Research Papers, Theses, and Dissertations*. Revised by Booth, Wayne, Gregory Columb, and Joseph Williams. 7th ed. Chicago: University of Chicago Press, 2007.)

The Resources section at the end of this handbook has information on other styles. If you are unsure whether a citation style is acceptable, you should check with your professor.

Tools to Make Referencing Easier

Following the proper rules of citation can be tedious and time-consuming. Fortunately, there are tools available to make referencing less burdensome. Three of the most popular programs available to help manage references are Endnote, Reference Manager, and RefWorks.

The Johns Hopkins University has obtained a license to allow students to use RefWorks, and free instruction on how to use the program is available at the Welch Library. For more information, visit: http://www.welch.jhu.edu/welch_tutorials/RefWorks.cfm

Microsoft Word® makes citation easier through the Citations and Bibliography section of the References tab. Formatting assistance for several citation styles, including APA and Chicago, is available. For specific help on how to create a bibliography in Word, see the Microsoft office help site. http://office.microsoft.com/en-us/word-help/create-a-bibliography-HA010368774.aspx?CTT=1

Can I receive assistance with my writing?

Our diverse and multi-cultural faculty and student body offer a rich learning environment that allows our students to learn public health concepts from a global perspective. Interaction and feedback between and among students and faculty members is encouraged both inside and outside of the classroom.

When completing specific academic assignments, however, you should exercise caution and awareness when seeking assistance from others. **Unless your instructor has indicated otherwise, all class assignments, including homework assignments and take-home exams, are to be done individually.** This usually encompasses all phases of completing the assignment, including brainstorming ideas, developing your argument, writing drafts, and your final work product. Requesting others to contribute to this process without approval by the instructor may lead to charges of cheating. **Allowing others to edit your work, even if they are providing minor editing for grammatical and spelling errors, is prohibited without specific instructor approval.** Faculty are aware that English is a second language for many students, and base their evaluation of academic work on the content of the material rather than on grammar and spelling.

There are a several university resources for international students and others who would like to improve their writing skills. The Welch Center has classes on writing. Information about times can be found at http://www.welch.jhu.edu/classes/free.cfm. Additional writing classes are available through the School and through the Professional Development Office: http://www.jhsph.edu/student_affairs/writing.html

Resources on citation and writing can be found at the end of this handbook. In addition, there are several plagiarism detection software programs that can serve as a tool to prevent plagiarism. If you are unsure if you have referenced appropriately, you can run your work through one of these programs. The University of Maryland offers a free plagiarism detection program on their website http://www.dustball.com/cs/plagiarism.checker/

Common Mistakes

- ✓ Submitting a “References” or “Sources” section at the end of an assignment without including endnotes/footnotes or parenthetical citation in appropriate places in the body of the work.
- ✓ Citing online sources with only a URL within the body of the work. Most citation styles require other information, such as access date and/or authors.
- ✓ Failing to put direct quotations in quotation marks, or indent quotations to make it clear that you used actual words from a text. This error is especially insidious. It is easy to cut and paste from online sources, but it is also easy to detect this.
- ✓ Failing to be consistent in your citation style throughout the assignment.
- ✓ Taking inadequate notes on sources consulted during the research process, which leads to inadequate referencing. Sloppiness is not an excuse for plagiarism.

Sloppiness and ignorance are not acceptable excuses for plagiarism.

Strategies for Avoiding Plagiarism

Familiarize yourself with the rules of citation.

Students are expected to be knowledgeable about how to correctly cite and reference sources in their academic work. This handbook provides a general overview of citation and resources for further reading.

Remember that the purpose of citation is to properly acknowledge your sources and to provide enough information to allow a reader to find the source material from which your information is generated.

Take detailed notes during research process and cite as you write.

Often times students focus on gathering content for their research and forget to pay equal attention to diligently writing down the sources of the content. Writing down proper reference citation, including accurate page numbers and versions, in your note-taking process will assure that you can correctly attribute source material when you begin writing your academic assignment.

In addition, citing as you work makes it less likely that you will forget to cite a source and unintentionally plagiarize. Any work you allow others to see, including draft papers submitted for review, may be assumed to be written with proper citation unless you indicate otherwise, which is yet another reason to cite as you write.

When in doubt, cite.

If you are unclear if a specific phrase is sufficiently unique to necessitate a quotation, or if a fact is "common" knowledge that does not require a citation, you should err on the side of caution and cite.

Plan ahead accordingly to reduce stress and time pressure.

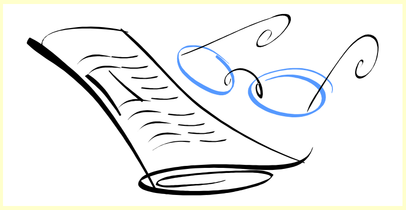
Many times students plagiarize as a shortcut to proper researching and referencing when they are under stress to meet deadlines. If time or stress management is an issue for you, address it as soon as possible to avoid the temptation to commit plagiarism.

The Student Assistance Program can help you with any personal problems you may be facing (443-287-7000; http://www.jhu.edu/~hr1/fasap/BSPHsap.html).

When in doubt, ask your professor for clarification on citing style.

While the professor may indicate which citation style he or she prefers in academic assignments, many times this information is left out of assignment instruction. It is your duty as a student to meet the citation standards required by your professor, so if you have doubts on when and how to cite your sources on assignments, then you should ask your professor for clarification. In addition, unless otherwise indicated by your professor, it is assumed that all classroom assignments and exams must be completed individually.

Resources and Further Reading



For common citation styles:

Turabian style (adapted Chicago style for student researchers):

Turabian, Kate L. *A Manual for Writers of Research Papers, Theses, and Dissertations*. Revised by Booth, Wayne, Gregory Columb, and Joseph Williams. 7th ed. Chicago: University of Chicago Press, 2007.

American Psychological Association (psychology and social sciences):

Publication Manual of the American Psychological Association. 6th ed. Washington, DC: American Psychological Association, 2010

International Committee of Medical Journal Editors (ICMJE) or Vancouver style (requirement for submission to most biomedical journals):

http://www.icmje.org/; see also:

http://www.ncbi.nlm.nih.gov/bookshelf/br.fcgi?book=citmed

Writing Resources:

Welch Library:
http://www.welch.jhu.edu/welch_tutorials/

Listing of Writing and Research Guides:
http://www.jhsph.edu/student_affairs/writing.html

Purdue University Online Writing Lab (OWL) has excellent instruction on writing and citation:
http://owl.english.purdue.edu/

For questions or comments:

Contact: Miye Schakne
Research Compliance Officer

615 N. Wolfe Street, W1033
Baltimore, MD 21205

Phone:
410-502-0433

E-mail:
mschakne@jhu.edu

Overview and History of R

Roger D. Peng, Associate Professor of Biostatistics
Johns Hopkins Bloomberg School of Public Health

What is R?

What is R?

What is R?

R is a dialect of the S language.

What is S?

- S is a language that was developed by John Chambers and others at Bell Labs.
- S was initiated in 1976 as an internal statistical analysis environment—originally implemented as Fortran libraries.
- Early versions of the language did not contain functions for statistical modeling.
- In 1988 the system was rewritten in C and began to resemble the system that we have today (this was Version 3 of the language). The book *Statistical Models in S* by Chambers and Hastie (the white book) documents the statistical analysis functionality.
- Version 4 of the S language was released in 1998 and is the version we use today. The book *Programming with Data* by John Chambers (the green book) documents this version of the language.

Historical Notes

- In 1993 Bell Labs gave StatSci (now Insightful Corp.) an exclusive license to develop and sell the S language.
- In 2004 Insightful purchased the S language from Lucent for $2 million and is the current owner.
- In 2006, Alcatel purchased Lucent Technologies and is now called Alcatel-Lucent.
- Insightful sells its implementation of the S language under the product name S-PLUS and has built a number of fancy features (GUIs, mostly) on top of it—hence the “PLUS”.
- In 2008 Insightful is acquired by TIBCO for $25 million
- The fundamentals of the S language itself has not changed dramatically since 1998.
- In 1998, S won the Association for Computing Machinery’s Software System Award.

S Philosophy

In “Stages in the Evolution of S”, John Chambers writes:

“[W]e wanted users to be able to begin in an interactive environment, where they did not consciously think of themselves as programming. Then as their needs became clearer and their sophistication increased, they should be able to slide gradually into programming, when the language and system aspects would become more important.”

http://www.stat.bell-labs.com/S/history.html

Back to R

- 1991: Created in New Zealand by Ross Ihaka and Robert Gentleman. Their experience developing R is documented in a 1996 *JCGS* paper.
- 1993: First announcement of R to the public.
- 1995: Martin Mächler convinces Ross and Robert to use the GNU General Public License to make R free software.
- 1996: A public mailing list is created (R-help and R-devel)
- 1997: The R Core Group is formed (containing some people associated with S-PLUS). The core group controls the source code for R.
- 2000: R version 1.0.0 is released.
- 2013: R version 3.0.2 is released on December 2013.

Features of R

- Syntax is very similar to S, making it easy for S-PLUS users to switch over.
- Semantics are superficially similar to S, but in reality are quite different (more on that later).
- Runs on almost any standard computing platform/OS (even on the PlayStation 3)
- Frequent releases (annual + bugfix releases); active development.

Features of R (cont'd)

- Quite lean, as far as software goes; functionality is divided into modular packages
- Graphics capabilities very sophisticated and better than most stat packages.
- Useful for interactive work, but contains a powerful programming language for developing new tools (user -> programmer)
- Very active and vibrant user community; R-help and R-devel mailing lists and Stack Overflow

Features of R (cont'd)

It's free! (Both in the sense of beer and in the sense of speech.)

Free Software

With *free software*, you are granted

- The freedom to run the program, for any purpose (freedom 0).
- The freedom to study how the program works, and adapt it to your needs (freedom 1). Access to the source code is a precondition for this.
- The freedom to redistribute copies so you can help your neighbor (freedom 2).
- The freedom to improve the program, and release your improvements to the public, so that the whole community benefits (freedom 3). Access to the source code is a precondition for this.

http://www.fsf.org

Drawbacks of R

- Essentially based on 40 year old technology.
- Little built in support for dynamic or 3-D graphics (but things have improved greatly since the “old days”).
- Functionality is based on consumer demand and user contributions. If no one feels like implementing your favorite method, then it’s *your* job!
    - (Or you need to pay someone to do it)
- Objects must generally be stored in physical memory; but there have been advancements to deal with this too
- Not ideal for all possible situations (but this is a drawback of all software packages).

Design of the R System

The R system is divided into 2 conceptual parts:

1. The “base” R system that you download from CRAN
2. Everything else.

R functionality is divided into a number of *packages*.

- The “base” R system contains, among other things, the **base** package which is required to run R and contains the most fundamental functions.
- The other packages contained in the “base” system include **utils**, **stats**, **datasets**, **graphics**, **grDevices**, **grid**, **methods**, **tools**, **parallel**, **compiler**, **splines**, **tcltk**, **stats4**.
- There are also “Recommend” packages: **boot**, **class**, **cluster**, **codetools**, **foreign**, **KernSmooth**, **lattice**, **mgcv**, **nlme**, **rpart**, **survival**, **MASS**, **spatial**, **nnet**, **Matrix**.

Design of the R System

And there are many other packages available:

- There are about 4000 packages on CRAN that have been developed by users and programmers around the world.
- There are also many packages associated with the Bioconductor project (http://bioconductor.org).
- People often make packages available on their personal websites; there is no reliable way to keep track of how many packages are available in this fashion.

Some R Resources

Available from CRAN (http://cran.r-project.org)

- An Introduction to R
- Writing R Extensions
- R Data Import/Export
- R Installation and Administration (mostly for building R from sources)
- R Internals (not for the faint of heart)

Some Useful Books on S/R

Standard texts

- Chambers (2008). *Software for Data Analysis*, Springer. (your textbook)
- 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.

Other resources

- Springer has a series of books called *Use R!*.
- A longer list of books is at http://www.r-project.org/doc/bib/R-books.html

Getting Help

Roger D. Peng, Associate Professor of Biostatistics
Johns Hopkins Bloomberg School of Public Health

Asking Questions

- Asking questions via email is different from asking questions in person
- People on the other side do not have the background information you have
    - they also don't know you personally (usually)
- Other people are busy; their time is limited
- The instructor (me) is here to help in all circumstances but may not be able to answer all questions!

Finding Answers

- Try to find an answer by searching the archives of the forum you plan to post to.
- Try to find an answer by searching the Web.
- Try to find an answer by reading the manual.
- Try to find an answer by reading a FAQ.
- Try to find an answer by inspection or experimentation.
- Try to find an answer by asking a skilled friend.
- If you're a programmer, try to find an answer by reading the source code.

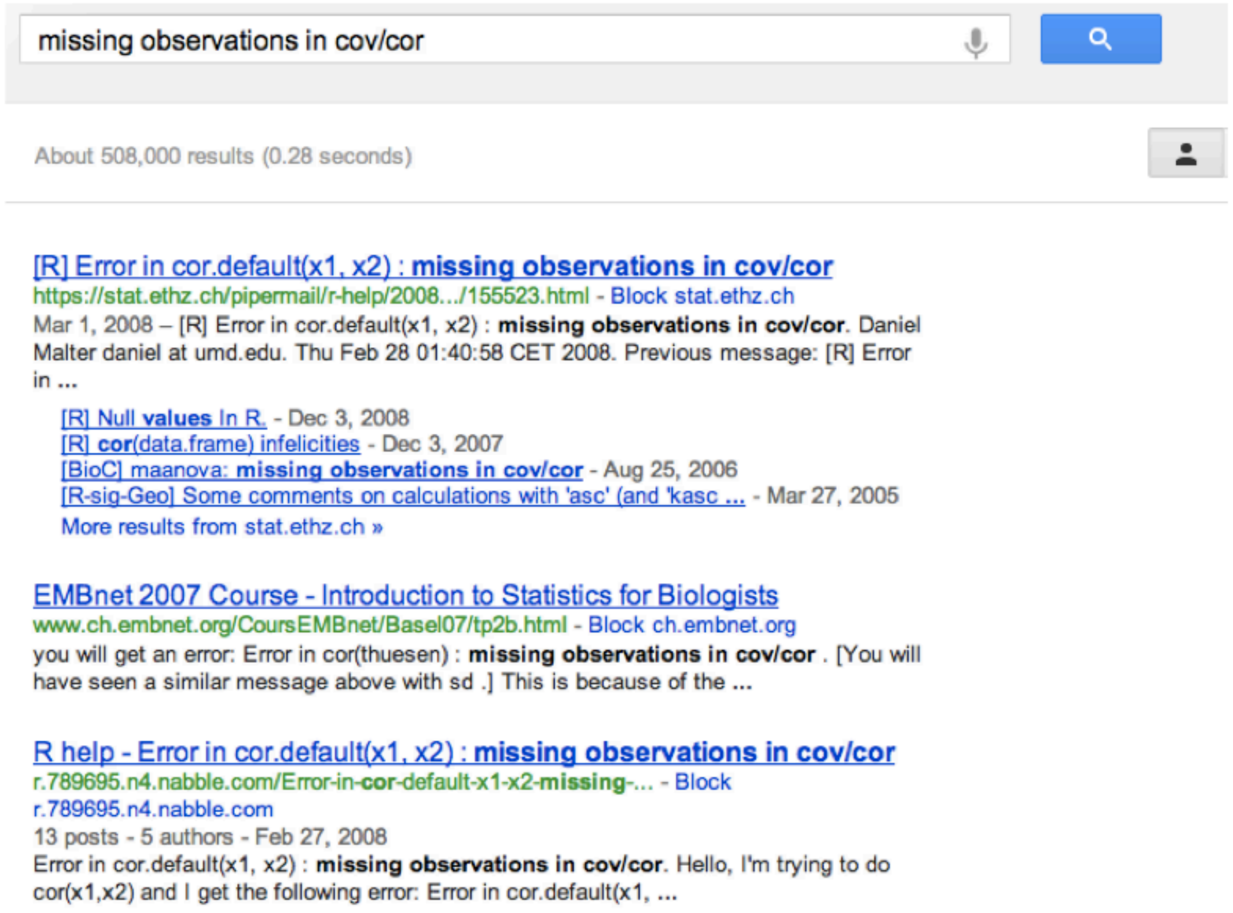
Asking Questions

- It’s important to let other people know that you’ve done all of the previous things already
- If the answer is in the documentation, the answer will be “Read the documentation”
  - one email round wasted

Example: Error Messages

```
> library(datasets)
> data(airquality)
> cor(airquality)
Error in cor(airquality) : missing observations in cov/cor
```

Google is your friend



missing observations in cov/cor

About 508,000 results (0.28 seconds)

[R] Error in cor.default(x1, x2) : **missing observations in cov/cor**
https://stat.ethz.ch/pipermail/r-help/2008.../155523.html - Block stat.ethz.ch
Mar 1, 2008 – [R] Error in cor.default(x1, x2) : **missing observations in cov/cor**. Daniel Malter daniel at umd.edu. Thu Feb 28 01:40:58 CET 2008. Previous message: [R] Error in ...

[R] Null **values** In R. - Dec 3, 2008
[R] **cor**(data.frame) infelicities - Dec 3, 2007
[BioC] maanova: **missing observations in cov/cor** - Aug 25, 2006
[R-sig-Geo] Some comments on calculations with 'asc' (and 'kasc ... - Mar 27, 2005
More results from stat.ethz.ch »

EMBnet 2007 Course - Introduction to Statistics for Biologists
www.ch.embnet.org/CoursEMBnet/Basel07/tp2b.html - Block ch.embnet.org
you will get an error: Error in cor(thuesen) : **missing observations in cov/cor** . [You will have seen a similar message above with sd .] This is because of the ...

R help - Error in cor.default(x1, x2) : **missing observations in cov/cor**
r.789695.n4.nabble.com/Error-in-**cor**-default-x1-x2-**missing**-... - Block r.789695.n4.nabble.com
13 posts - 5 authors - Feb 27, 2008
Error in cor.default(x1, x2) : **missing observations in cov/cor**. Hello, I'm trying to do cor(x1,x2) and I get the following error: Error in cor.default(x1, ...

Asking Questions

- What steps will reproduce the problem?
- What is the expected output?
- What do you see instead?
- What version of the product (e.g. R, packages, etc.) are you using?
- What operating system?
- Additional information

Subject Headers

- Stupid: "Help! Can't fit linear model!"
- Smart: "R 3.0.2 lm() function produces seg fault with large data frame, Mac OS X 10.9.1"
- Smarter: "R 3.0.2 lm() function on Mac OS X 10.9.1 -- seg fault on large data frame"

Do

- Describe the goal, not the step
- Be explicit about your question
- Do provide the minimum amount of information necessary (volume is not precision)
- Be courteous (it never hurts)
- Follow up with the solution (if found)

Don't

- Claim that you've found a bug
- Grovel as a substitute for doing your homework
- Post homework questions on mailing lists (we've seen them all)
- Email multiple mailing lists at once
- Ask others to debug your broken code without giving a hint as to what sort of problem they should be searching for

Case Study: A Recent Post to the R-devel Mailing List

```
Subject: large dataset - confused
Message:
  I'm trying to load a dataset into R, but
    I'm completely lost. This is probably
    due mostly to the fact that I'm a
    complete R newb, but it's got me stuck
    in a research project.
```

Response

```
Yes, you are lost. The R posting guide is
  at http://www.r-project.org/posting-
  guide.html and will point you to the
  right list and also the manuals (at
  e.g. http://cran.r-project.org/
  manuals.html, and one of them seems
  exactly what you need).
```

Analysis: What Went Wrong?

- Question was sent to the wrong mailing list (R-devel instead of R-help)
- Email subject was very vague
- Question was very vague
- Problem was not reproducible
- No evidence of any effort made to solve the problem
- RESULT: Recipe for disaster!

Places to Turn

- Class discussion board; your fellow students
- r-help@r-project.org
- Other project-specific mailing lists (This talk inspired by Eric Raymond's "How to ask questions the smart way")

Entering Input

At the R prompt we type expressions. The `<-` symbol is the assignment operator.

```
> x <- 1
> print(x)
[1] 1
> x
[1] 1
> msg <- "hello"
```

The grammar of the language determines whether an expression is complete or not.

```
> x <-  ## Incomplete expression
```

The # character indicates a comment. Anything to the right of the # (including the # itself) is ignored.

Evaluation

When a complete expression is entered at the prompt, it is evaluated and the result of the evaluated expression is returned. The result may be auto-printed.

```
> x <- 5  ## nothing printed
> x       ## auto-printing occurs
[1] 5
> print(x)  ## explicit printing
[1] 5
```

The `[1]` indicates that `x` is a vector and `5` is the first element.

Printing

```
> x <- 1:20
> x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
[16] 16 17 18 19 20
```

The `:` operator is used to create integer sequences.

Objects

R has five basic or “atomic” classes of objects:

- character
- numeric (real numbers)
- integer
- complex
- logical (True/False)

The most basic object is a vector

- A vector can only contain objects of the same class
- BUT: The one exception is a *list*, which is represented as a vector but can contain objects of different classes (indeed, that’s usually why we use them)

Empty vectors can be created with the `vector()` function.

Numbers

- Numbers in R a generally treated as numeric objects (i.e. double precision real numbers)
- If you explicitly want an integer, you need to specify the `L` suffix
- Ex: Entering `1` gives you a numeric object; entering `1L` explicitly gives you an integer.
- There is also a special number `Inf` which represents infinity; e.g. `1 / 0`; `Inf` can be used in ordinary calculations; e.g. `1 / Inf` is 0
- The value `NaN` represents an undefined value ("not a number"); e.g. 0 / 0; `NaN` can also be thought of as a missing value (more on that later)

Attributes

R objects can have attributes

- names, dimnames
- dimensions (e.g. matrices, arrays)
- class
- length
- other user-defined attributes/metadata

Attributes of an object can be accessed using the `attributes()` function.

Creating Vectors

The `c()` function can be used to create vectors of objects.

```
> x <- c(0.5, 0.6)       ## numeric
> x <- c(TRUE, FALSE)    ## logical
> x <- c(T, F)           ## logical
> x <- c("a", "b", "c")  ## character
> x <- 9:29              ## integer
> x <- c(1+0i, 2+4i)     ## complex
```

Using the `vector()` function

```
> x <- vector("numeric", length = 10)
> x
 [1] 0 0 0 0 0 0 0 0 0 0
```

Mixing Objects

What about the following?

```
> y <- c(1.7, "a")   ## character
> y <- c(TRUE, 2)    ## numeric
> y <- c("a", TRUE)  ## character
```

When different objects are mixed in a vector, *coercion* occurs so that every element in the vector is of the same class.

Explicit Coercion

Objects can be explicitly coerced from one class to another using the `as.*` functions, if available.

```
> x <- 0:6
> class(x)
[1] "integer"
> as.numeric(x)
[1] 0 1 2 3 4 5 6
> as.logical(x)
[1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
> as.character(x)
[1] "0" "1" "2" "3" "4" "5" "6"
```

Explicit Coercion

Nonsensical coercion results in `NA`s.

```
> x <- c("a", "b", "c")
> as.numeric(x)
[1] NA NA NA
Warning message:
NAs introduced by coercion
> as.logical(x)
[1] NA NA NA
> as.complex(x)
[1] NA NA NA
Warning message:
NAs introduced by coercion
```

Matrices

Matrices are vectors with a *dimension* attribute. The dimension attribute is itself an integer vector of length 2 (nrow, ncol)

```
> m <- matrix(nrow = 2, ncol = 3)
> m
     [,1] [,2] [,3]
[1,]   NA   NA   NA
[2,]   NA   NA   NA
> dim(m)
[1] 2 3
> attributes(m)
$dim
[1] 2 3
```

Matrices (cont'd)

Matrices are constructed *column-wise*, so entries can be thought of starting in the “upper left” corner and running down the columns.

```
> m <- matrix(1:6, nrow = 2, ncol = 3)
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Matrices (cont'd)

Matrices can also be created directly from vectors by adding a dimension attribute.

```
> m <- 1:10
> m
[1] 1 2 3 4 5 6 7 8 9 10
> dim(m) <- c(2, 5)
> m
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

cbind-ing and rbind-ing

Matrices can be created by *column-binding* or *row-binding* with `cbind()` and `rbind()`.

```
> x <- 1:3
> y <- 10:12
> cbind(x, y)
     x  y
[1,] 1 10
[2,] 2 11
[3,] 3 12
> rbind(x, y)
  [,1] [,2] [,3]
x    1    2    3
y   10   11   12
```

Lists

Lists are a special type of vector that can contain elements of different classes. Lists are a very important data type in R and you should get to know them well.

```
> x <- list(1, "a", TRUE, 1 + 4i)
> x
[[1]]
[1] 1

[[2]]
[1] "a"

[[3]]
[1] TRUE

[[4]]
[1] 1+4i
```

Matrices

Matrices are vectors with a *dimension* attribute. The dimension attribute is itself an integer vector of length 2 (nrow, ncol)

```
> m <- matrix(nrow = 2, ncol = 3)
> m
     [,1] [,2] [,3]
[1,]   NA   NA   NA
[2,]   NA   NA   NA
> dim(m)
[1] 2 3
> attributes(m)
$dim
[1] 2 3
```

Matrices (cont'd)

Matrices are constructed *column-wise*, so entries can be thought of starting in the “upper left” corner and running down the columns.

```
> m <- matrix(1:6, nrow = 2, ncol = 3)
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Matrices (cont'd)

Matrices can also be created directly from vectors by adding a dimension attribute.

```
> m <- 1:10
> m
[1] 1 2 3 4 5 6 7 8 9 10
> dim(m) <- c(2, 5)
> m
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

cbind-ing and rbind-ing

Matrices can be created by *column-binding* or *row-binding* with `cbind()` and `rbind()`.

```
> x <- 1:3
> y <- 10:12
> cbind(x, y)
     x  y
[1,] 1 10
[2,] 2 11
[3,] 3 12
> rbind(x, y)
  [,1] [,2] [,3]
x    1    2    3
y   10   11   12
```

Factors

Factors are used to represent categorical data. Factors can be unordered or ordered. One can think of a factor as an integer vector where each integer has a *label*.

- Factors are treated specially by modelling functions like `lm()` and `glm()`
- Using factors with labels is *better* than using integers because factors are self-describing; having a variable that has values “Male” and “Female” is better than a variable that has values 1 and 2.

Factors

```
> x <- factor(c("yes", "yes", "no", "yes", "no"))
> x
[1] yes yes no yes no
Levels: no yes
> table(x)
x
no yes
 2   3
> unclass(x)
[1] 2 2 1 2 1
attr(,"levels")
[1] "no"  "yes"
```

Factors

The order of the levels can be set using the `levels` argument to `factor()`. This can be important in linear modelling because the first level is used as the baseline level.

```
> x <- factor(c("yes", "yes", "no", "yes", "no"),
              levels = c("yes", "no"))
> x
[1] yes yes no yes no
Levels: yes no
```

Missing Values

Missing values are denoted by `NA` or `NaN` for undefined mathematical operations.

- `is.na()` is used to test objects if they are `NA`
- `is.nan()` is used to test for `NaN`
- `NA` values have a class also, so there are integer `NA`, character `NA`, etc.
- A `NaN` value is also `NA` but the converse is not true

Missing Values

```
> x <- c(1, 2, NA, 10, 3)
> is.na(x)
[1] FALSE FALSE  TRUE FALSE FALSE
> is.nan(x)
[1] FALSE FALSE FALSE FALSE FALSE
> x <- c(1, 2, NaN, NA, 4)
> is.na(x)
[1] FALSE FALSE  TRUE  TRUE FALSE
> is.nan(x)
[1] FALSE FALSE  TRUE FALSE FALSE
```

Data Frames

Data frames are used to store tabular data

- They are represented as a special type of list where every element of the list has to have the same length
- Each element of the list can be thought of as a column and the length of each element of the list is the number of rows
- Unlike matrices, data frames can store different classes of objects in each column (just like lists); matrices must have every element be the same class
- Data frames also have a special attribute called `row.names`
- Data frames are usually created by calling `read.table()` or `read.csv()`
- Can be converted to a matrix by calling `data.matrix()`

Data Frames

```
> x <- data.frame(foo = 1:4, bar = c(T, T, F, F))
> x
  foo   bar
1   1  TRUE
2   2  TRUE
3   3 FALSE
4   4 FALSE
> nrow(x)
[1] 4
> ncol(x)
[1] 2
```

Names

R objects can also have names, which is very useful for writing readable code and self-describing objects.

```
> x <- 1:3
> names(x)
NULL
> names(x) <- c("foo", "bar", "norf")
> x
foo bar norf
  1   2    3
> names(x)
[1] "foo"  "bar"  "norf"
```

Names

Lists can also have names.

```
> x <- list(a = 1, b = 2, c = 3)
> x
$a
[1] 1

$b
[1] 2

$c
[1] 3
```

Names

And matrices.

```
> m <- matrix(1:4, nrow = 2, ncol = 2)
> dimnames(m) <- list(c("a", "b"), c("c", "d"))
> m
  c d
a 1 3
b 2 4
```

Summary

Data Types

- atomic classes: numeric, logical, character, integer, complex \
- vectors, lists
- factors
- missing values
- data frames
- names

Reading Data

There are a few principal functions reading data into R.

- `read.table`, `read.csv`, for reading tabular data
- `readLines`, for reading lines of a text file
- `source`, for reading in R code files (`inverse` of `dump`)
- `dget`, for reading in R code files (`inverse` of `dput`)
- `load`, for reading in saved workspaces
- `unserialize`, for reading single R objects in binary form

Writing Data

There are analogous functions for writing data to files

- write.table
- writeLines
- dump
- dput
- save
- serialize

Reading Data Files with read.table

The `read.table` function is one of the most commonly used functions for reading data. It has a few important arguments:

- `file`, the name of a file, or a connection
- `header`, logical indicating if the file has a header line
- `sep`, a string indicating how the columns are separated
- `colClasses`, a character vector indicating the class of each column in the dataset
- `nrows`, the number of rows in the dataset
- `comment.char`, a character string indicating the comment character
- `skip`, the number of lines to skip from the beginning
- `stringsAsFactors`, should character variables be coded as factors?

read.table

For small to moderately sized datasets, you can usually call read.table without specifying any other arguments

```
data <- read.table("foo.txt")
```

R will automatically

- skip lines that begin with a #
- figure out how many rows there are (and how much memory needs to be allocated)
- figure what type of variable is in each column of the table Telling R all these things directly makes R run faster and more efficiently.
- `read.csv` is identical to read.table except that the default separator is a comma.

Reading in Larger Datasets with read.table

With much larger datasets, doing the following things will make your life easier and will prevent R from choking.

- Read the help page for read.table, which contains many hints
- Make a rough calculation of the memory required to store your dataset. If the dataset is larger than the amount of RAM on your computer, you can probably stop right here.
- Set `comment.char = ""` if there are no commented lines in your file.

Reading in Larger Datasets with read.table

- Use the `colClasses` argument. Specifying this option instead of using the default can make ’read.table’ run MUCH faster, often twice as fast. In order to use this option, you have to know the class of each column in your data frame. If all of the columns are “numeric”, for example, then you can just set `colClasses = "numeric"`. A quick an dirty way to figure out the classes of each column is the following:

```r
initial <- read.table("datatable.txt", nrows = 100)
classes <- sapply(initial, class)
tabAll <- read.table("datatable.txt",
                     colClasses = classes)
```

- Set `nrows`. This doesn’t make R run faster but it helps with memory usage. A mild overestimate is okay. You can use the Unix tool `wc` to calculate the number of lines in a file.

Know Thy System

In general, when using R with larger datasets, it’s useful to know a few things about your system.

- How much memory is available?
- What other applications are in use?
- Are there other users logged into the same system?
- What operating system?
- Is the OS 32 or 64 bit?

Calculating Memory Requirements

I have a data frame with 1,500,000 rows and 120 columns, all of which are numeric data. Roughly, how much memory is required to store this data frame?

1,500,000 × 120 × 8 bytes/numeric

= 1440000000 bytes

= 1440000000 / 2^{20} bytes/MB

= 1,373.29 MB

= 1.34 GB

Textual Formats

- `dumping` and dputing are useful because the resulting textual format is edit-able, and in the case of corruption, potentially recoverable.
- `Unlike` writing out a table or csv file, `dump` and `dput` preserve the *metadata* (sacrificing some readability), so that another user doesn't have to specify it all over again.
- `Textual` formats can work much better with version control programs like subversion or git which can only track changes meaningfully in text files
- Textual formats can be longer-lived; if there is corruption somewhere in the file, it can be easier to fix the problem
- Textual formats adhere to the "Unix philosophy"
- Downside: The format is not very space-efficient

dput-ting R Objects

Another way to pass data around is by deparsing the R object with dput and reading it back in using `dget`.

```
> y <- data.frame(a = 1, b = "a")
> dput(y)
structure(list(a = 1,
               b = structure(1L, .Label = "a",
                             class = "factor")),
          .Names = c("a", "b"), row.names = c(NA, -1L),
          class = "data.frame")
> dput(y, file = "y.R")
> new.y <- dget("y.R")
> new.y
   a  b
1  1  a
```

Dumping R Objects

Multiple objects can be deparsed using the dump function and read back in using `source`.

```
> x <- "foo"
> y <- data.frame(a = 1, b = "a")
> dump(c("x", "y"), file = "data.R")
> rm(x, y)
> source("data.R")
> y
  a  b
1 1  a
> x
[1] "foo"
```

Interfaces to the Outside World

Data are read in using *connection* interfaces. Connections can be made to files (most common) or to other more exotic things.

- `file`, opens a connection to a file
- `gzfile`, opens a connection to a file compressed with gzip
- `bzfile`, opens a connection to a file compressed with bzip2
- `url`, opens a connection to a webpage

File Connections

```
> str(file)
function (description = "", open = "", blocking = TRUE,
          encoding = getOption("encoding"))
```

- `description` is the name of the file
- `open` is a code indicating
  - "r" read only
  - "w" writing (and initializing a new file)
  - "a" appending
  - "rb", "wb", "ab" reading, writing, or appending in binary mode (Windows)

Connections

In general, connections are powerful tools that let you navigate files or other external objects. In practice, we often don't need to deal with the connection interface directly.

```
con <- file("foo.txt", "r")
data <- read.csv(con)
close(con)
```

is the same as

```
data <- read.csv("foo.txt")
```

Reading Lines of a Text File

```
> con <- gzfile("words.gz")
> x <- readLines(con, 10)
> x
 [1] "1080"      "10-point" "10th"      "11-point"
 [5] "12-point" "16-point" "18-point" "1st"
 [9] "2"         "20-point"
```

`writeLines` takes a character vector and writes each element one line at a time to a text file.

Reading Lines of a Text File

`readLines` can be useful for reading in lines of webpages

```
## This might take time
con <- url("http://www.jhsph.edu", "r")
x <- readLines(con)
> head(x)
[1] "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 Transitional//EN\">"
[2] ""
[3] "<html>"
[4] "<head>"
[5] "\t<meta http-equiv=\"Content-Type\" content=\"text/html;charset=utf-8
```

Subsetting

There are a number of operators that can be used to extract subsets of R objects.

- `[` always returns an object of the same class as the original; can be used to select more than one element (there is one exception)
- `[[` is used to extract elements of a list or a data frame; it can only be used to extract a single element and the class of the returned object will not necessarily be a list or data frame
- `$` is used to extract elements of a list or data frame by name; semantics are similar to that of `[[`.

Subsetting

```
> x <- c("a", "b", "c", "c", "d", "a")
> x[1]
[1] "a"
> x[2]
[1] "b"
> x[1:4]
[1] "a" "b" "c" "c"
> x[x > "a"]
[1] "b" "c" "c" "d"
> u <- x > "a"
> u
[1] FALSE TRUE TRUE TRUE TRUE FALSE
> x[u]
[1] "b" "c" "c" "d"
```

Subsetting Lists

```
> x <- list(foo = 1:4, bar = 0.6)
> x[1]
$foo
[1] 1 2 3 4

> x[[1]]
[1] 1 2 3 4

> x$bar
[1] 0.6
> x[["bar"]]
[1] 0.6
> x["bar"]
$bar
[1] 0.6
```

Subsetting Lists

```
> x <- list(foo = 1:4, bar = 0.6, baz = "hello")
> x[c(1, 3)]
$foo
[1] 1 2 3 4

$baz
[1] "hello"
```

Subsetting Lists

The `[[` operator can be used with *computed* indices; `$` can only be used with literal names.

```
> x <- list(foo = 1:4, bar = 0.6, baz = "hello")
> name <- "foo"
> x[[name]]  ## computed index for ‘foo’
[1] 1 2 3 4
> x$name     ## element ‘name’ doesn’t exist!
NULL
> x$foo
[1] 1 2 3 4  ## element ‘foo’ does exist
```

Subsetting Nested Elements of a List

The `[[` can take an integer sequence.

```
> x <- list(a = list(10, 12, 14), b = c(3.14, 2.81))
> x[[c(1, 3)]]
[1] 14
> x[[1]][[3]]
[1] 14

> x[[c(2, 1)]]
[1] 3.14
```

Subsetting a Matrix

Matrices can be subsetted in the usual way with (*i,j*) type indices.

```
> x <- matrix(1:6, 2, 3)
> x[1, 2]
[1] 3
> x[2, 1]
[1] 2
```

Indices can also be missing.

```
> x[1, ]
[1] 1 3 5
> x[, 2]
[1] 3 4
```

Subsetting a Matrix

By default, when a single element of a matrix is retrieved, it is returned as a vector of length 1 rather than a 1 × 1 matrix. This behavior can be turned off by setting `drop = FALSE`.

```
> x <- matrix(1:6, 2, 3)
> x[1, 2]
[1] 3
> x[1, 2, drop = FALSE]
     [,1]
[1,] 3
```

Subsetting a Matrix

Similarly, subsetting a single column or a single row will give you a vector, not a matrix (by default).

```
> x <- matrix(1:6, 2, 3)
> x[1, ]
[1] 1 3 5
> x[1, , drop = FALSE]
     [,1] [,2] [,3]
[1,]    1    3    5
```

Partial Matching

Partial matching of names is allowed with `[[` and `$`.

```
> x <- list(aardvark = 1:5)
> x$a
[1] 1 2 3 4 5
> x[["a"]]
NULL
> x[["a", exact = FALSE]]
[1] 1 2 3 4 5
```

Removing NA Values

A common task is to remove missing values (`NA`s).

```
> x <- c(1, 2, NA, 4, NA, 5)
> bad <- is.na(x)
> x[!bad]
[1] 1 2 4 5
```

Removing NA Values

What if there are multiple things and you want to take the subset with no missing values?

```
> x <- c(1, 2, NA, 4, NA, 5)
> y <- c("a", "b", NA, "d", NA, "f")
> good <- complete.cases(x, y)
> good
[1]  TRUE  TRUE FALSE  TRUE FALSE  TRUE
> x[good]
[1] 1 2 4 5
> y[good]
[1] "a" "b" "d" "f"
```

Removing NA Values

```
> airquality[1:6, ]
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
> good <- complete.cases(airquality)
> airquality[good, ][1:6, ]
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
7    23     299  8.6   65     5   7
```

Introduction to the R Language

Vectorized Operations

Roger Peng, Associate Professor
Johns Hopkins Bloomberg School of Public Health

Vectorized Operations

Many operations in R are *vectorized* making code more efficient, concise, and easier to read.

```
> x <- 1:4; y <- 6:9
> x + y
[1] 7 9 11 13
> x > 2
[1] FALSE FALSE  TRUE  TRUE
> x >= 2
[1] FALSE  TRUE  TRUE  TRUE
> y == 8
[1] FALSE FALSE TRUE FALSE
> x * y
[1] 6 14 24 36
> x / y
[1] 0.1666667 0.2857143 0.3750000 0.4444444
```

Vectorized Matrix Operations

```
> x <- matrix(1:4, 2, 2); y <- matrix(rep(10, 4), 2, 2)
> x * y       ## element-wise multiplication
     [,1] [,2]
[1,]   10   30
[2,]   20   40
> x / y
     [,1] [,2]
[1,]  0.1  0.3
[2,]  0.2  0.4
> x %*% y     ## true matrix multiplication
     [,1] [,2]
[1,]   40   40
[2,]   60   60
```