G. A. REYNOLDS

THINK DATA

A CONCEPTUAL INTRODUCTION TO BIG DATA

FOR THE SKEPTICAL, THE PESSIMISTIC, AND THE MILDLY DISTURBED

Remark 1 Two tasks: conceptual and computational. Understand stats first, then learn how to *do* stats using software (statistics machines).

Contents

Introduction

What is data?

Wrong question. The right questions are: what is the role of data? What functions does it serve? How is it used? etc. Our practices involving data tell us what data "is"; there can be no antecedently defined notion of data. (I.e. in spite of the etymology, no data are "given".)

> Myth of the Given
>
> Sellars

Part I

Modeling

Ed. note 0.0.1 *Modeling the world as data; it really should be called "world modeling". E.g. in industry, a "standard data model" is a standard way of representing some domain of interest, such as electrical connectors or transmission lines. TODO: clear, simple examples. Maybe Pipeline Open Data Standard (PODS)? "The PODS Pipeline Data Model provides the database architecture pipeline operators use to store critical information and analysis data about their pipeline systems, and manage this data geospatially in a linear-referenced database which can then be visualized in any GIS platform. The PODS Pipeline Data Model houses the asset information, inspection, integrity management, regulatory compliance, risk analysis, history, and operational data that pipeline companies have deemed mission-critical to the successful management of natural gas and hazardous liquids pipelines."*

TODO: how is this concept of data model related to statistical notions of "model" and modeling?

Part II

Description

Ed. note 0.0.2 *Description of data, not data as description of world - that's modeling*

In principle, every datum must be associated with a complete description that determines both its form and its meaning.

Syntax

Semantics

Type Same as syntax?

Traceability esp. given ability to rename, split, and generally slice and dice, traceability is critical.

Datum: description of types and individuals

Data: description of data collections

Descriptive statistics. The technical statistical side of such description is covered in detail in the companion volume. Here we cover ...?

Quality

How is data quality defined?

Part III

Management

Ed. note 0.0.3 *Here the OAIS model is relevant.*

Acquisition

Conversion

I.e. input adapters, import, reformatting, etc.

Conversion occurs at both ends, not just dissemination.

Normalization: Cleaning, Organization, Imputation, etc.

AKA data munging
ref: the PADS project
Data Management - a comparison of the commands used for common data "management" (actually, munging) tasks in R, SAS, SPSS and Stata.

Misc

Remark 2 examples taken from R, so far

plyr: "plyr is a set of tools for a common set of problems: you need to split up a big data structure into homogeneous pieces, apply a function to each piece and then combine all the results back together"
R Data Manipulation Manipulating Data

Subsetting Data

Subset data in R

Aggregation

Aggregate Data in R Using data.table - here "aggregate" seems to be synonymous with "compute statistic of". Misnomer; an aggregate is not a summary.

Recoding

Recode variables (examples)
New vars, recoding vars, renaming vars
Recoding vars: categorical, continuous, new
Recode data in R: replacement, recoding
Recode one column, output values into another column: transform(), replace()

The Recode Command From the Package Car

New vars

"copy of an existing field. Sometimes you dont want to recode data but instead just want another column containing the same data."

Recode into A New Field Using Data From An Existing field And Criteria from Another Field

Replacing data

"replace the data in an existing field when you want to replace the data for every row (no criteria)." `http://rprogramming.net/recode-data-in-r/`

Renaming vars

Curation

AKA Data Management.
Databases, metadata, ingest/disseminate procs, etc.

Metadata

Provenance

I.e. traceability

Preservation

Discovery

Access

e.g. ACLs; e.g. NORC data enclave

Dissemination

Packaging

Conversion

I.e. export to formats
: Conversion occurs at both ends, not just dissemination.

Presentation

Part IV

Visualization

Data Visualization in Sociology
Visualizing Categorical Data (Friendly, M.)

Part V

Exploration

EDA as involving the combination of statistical description and visualization.

The Split-Apply-Combine Strategy for Data Analysis

Part VI

Tools & Techniques

Choosing your workflow applications

Overview

Standards

OAIS

DDI

SMDX

Languages

- Computing Trends Lead to New Programming Languages
- Twelve New Programming Languages: Is Cloud Responsible?
- The Popularity of Data Analysis Software (2014) A very detailed data-driven analysis of the relative popularity of various languages.

Remark 3 Organize by category?

Standards

Ed. note 0.0.4 *TODO: standards v. data models*

OAIS

Reference Model for an Open Archival Information System (OAIS)

Statistical Standards

- UN Global Inventory of Statistical Standards
- Statistical Standards Program - National Center for Education Statistics
- OECD Glossary of Statistical Terms
- Standards for Statistical Interpretation of Data (British Standards Institute)
- OMB Statistical Programs and Standards

DDI

SDMX

SDMX - Statistical Data and Metadata eXchange

R

R is a very popular and hideous language.

Resources:

- Programming in R
- R Programming Wiki
- Why R is hard to learn

Lispiness

R is lispy, just like javascript is lispy.

"R presents a friendlier interface to programming than Lisp does, at least to someone used to mathematical formulas and C-like control structures, but the engine is really very Lisp-like. R allows direct access to parsed expressions and functions and allows you to alter and subsequently execute them, or create entirely new functions from scratch."http://cran.r-project.org/doc/manuals/R-lang.html#Computing-on-the-language

"There are three kinds of language objects that are available for modification, calls, expressions, and functions."

Meta-types

Ed. note 0.0.5 *Note the weird language. The "language objects" seem to meta-types used in the parse tree, or something like that. Syntactic, meta-objects, not objects "in" the language.*

From http://cran.r-project.org/doc/manuals/R-lang.html#Objects

"There are three types of objects that constitute the R language.[1]

[1] Why call them objects? Why not "types of expression in the language"?

They are calls, expressions, and names...These objects have modes "call", "expression", and "name", respectively. They can be created directly from expressions using the quote mechanism and converted to and from lists by the as.list and as.call functions. Components of the parse tree can be extracted using the standard indexing operations."

That can't be quite right; at any rate, calls, expressions, and names are not defined in the section on basic types. Apparently these are "language objects", or meta-objects rather than objects in the language.

"Parsed expressions are stored in an R object containing the parse tree. A fuller description of such objects can be found in Language objects and Expression objects."http://cran.r-project.org/doc/manuals/R-lang.html#Parser

Language "objects":

Call " The most direct method of obtaining a call object is to use quote with an expression argument, e.g.,

```
> e1 <- quote(2 + 2)
> e2 <- quote(plot(x, y))
```

Then both e1 and e1 have "mode" of "call"; so apparently "call object" is r-speak for "function application expression".

Name "The components of a call object are accessed using a list-like syntax, and may in fact be converted to and from lists using as.list and as.call"

"All the components of the call object have mode "name" in the preceding examples. This is true for identifiers in calls, but the components of a call can also be constantswhich can be of any type, although the first component had better be a function if the call is to be evaluated successfullyor other call objects, corresponding to subexpressions. Objects of mode name can be constructed from character strings using as.name, so one might modify the e2 object as follows"

Expression "An expression contains one or more statements. A statement is a syntactically correct collection of tokens. Expression objects are special language objects which contain parsed but unevaluated R statements. The main difference is that an expression object can contain several such expressions. Another more subtle difference is that objects of type "expression" are only evaluated when explicitly passed to eval, whereas other language objects may get evaluated in some unexpected cases."http://cran.r-project.org/doc/manuals/R-lang.html#Expression-objects

Expression objects "are very similar to lists of call objects."

Metaprogramming

"It is possible for a function to find out how it has been called by looking at the result of sys.call...However, this is not really useful except for debugging ...More often one requires the call with all actual arguments bound to the corresponding formals. To this end, the function match.call is used...The primary use of this technique is to call another function with the same arguments, possibly deleting some and adding others."

"The call can be treated as a list object where the first element is the name of the function and the remaining elements are the actual argument expressions, with the corresponding formal argument names as tags."

"Two further functions exist for the construction of function calls, namely call and do.call."

"It is often useful to be able to manipulate the components of a function or closure. R provides a set of interface functions for this purpose."

body Returns the expression that is the body of the function.

formals Returns a list of the formal arguments to the function. This is a pairlist.

environment Returns the environment associated with the function.

body<- This sets the body of the function to the supplied expression.

formals<- Sets the formal arguments of the function to the supplied list.

environment<- Sets the environment of the function to the specified environment.

"It is also possible to alter the bindings of different variables in the environment of the function, using code along the lines of evalq(x <- 5, environment(f)). It is also possible to convert a function to a list using as.list. The result is the concatenation of the list of formal arguments with the function body. Conversely such a list can be converted to a function using as.function. This functionality is mainly included for S compatibility. Notice that environment information is lost when as.list is used, whereas as.function has an argument that allows the environment to be set."

Evaluation

Lazy Eval

"Promise objects are part of Rs lazy evaluation mechanism. They contain three slots: a value, an expression, and an environment. When a function is called the arguments are matched and then each of the formal arguments is bound to a promise. The expression that was given for that formal argument and a pointer to the environment the function was called from are stored in the promise. Until that argument is accessed there is no value associated with the promise. When the argument is accessed, the stored expression is evaluated in the stored environment, and the result is returned. The result is also saved by the promise. The substitute function will extract the content of the expression slot. This allows the programmer to access either the value or the expression associated with the promise. Within the R language, promise objects are almost only seen implicitly: actual function arguments are of this type. There is also a delayedAssign function that will make a promise out of an expression. There is generally no way in R code to check whether an object is a promise or not, nor is there a way to use R code to determine the environment of a promise."http://cran.r-project.org/doc/manuals/R-lang.html#Promise-objects

"A formal argument is really a promise, an object with three slots, one for the expression that defines it, one for the environment in which to evaluate that expression, and one for the value of that expression once evaluated."

Types

"Symbols refer to R objects. The name of any R object is usually a symbol. Symbols can be created through the functions as.name and quote. Symbols have mode "name", storage mode "symbol", and type "symbol". They can be coerced to and from character strings using as.character and as.name. They naturally appear as atoms of parsed expressions, try e.g. as.list(quote(x + y))."

"In R one can have objects of type "expression". An expression contains one or more statements. A statement is a syntactically correct collection of tokens. Expression objects are special language objects which contain parsed but unevaluated R statements. The main difference is that an expression object can contain several such expressions. Another more subtle difference is that objects of type "expression" are only evaluated when explicitly passed to eval, whereas other language objects may get evaluated in some unexpected cases.

An expression object behaves much like a list and its components should be accessed in the same way as the components of a list."

"In R functions are objects and can be manipulated in much the same way as any other object. Functions (or more precisely, function closures) have three basic components: a formal argument list, a body and an environment...A functions environment is the environment that was active at the time that the function was created. Any symbols bound in that environment are captured and available to the function. This combination of the code of the function and the bindings in its environment is called a function closure, a term from functional programming theory. In this document we generally use the term function, but use closure to emphasize the importance of the attached environment. When a function is called, a new environment (called the evaluation environment) is created, whose enclosure (see Environment objects) is the environment from the function closure. This new environment is initially populated with the unevaluated arguments to the function; as evaluation proceeds, local variables are created within it."

Python

Resources:

- A Roadmap for Rich Scientific Data Structures in Python
- On the growth of R and Python for data science
- Getting Started With Python For Data Science
- Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython (O'Reilly book) "Python for Data Analysis is concerned with the nuts and bolts of manipulating, processing, cleaning, and crunching data in Python. It is also a practical, modern introduction to scientific computing in Python, tailored for data-intensive applications. This is a book about the parts of the Python language and libraries youll need to effectively solve a broad set of data analysis problems. This book is not an exposition on analytical methods using Python as the implementation language."

Numpy

Scipy

IPython

Pandas high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

Ruby

- SciRuby
- StatSample

Java

Resources:

JavaNumerics

Parallel Colt

Colt a set of Open Source Libraries for High Performance Scientific and Technical Computing in Java.

Processing

JFreeChart

Weka a collection of machine learning algorithms for data mining tasks

Clojure

Incanter

Scala

Scala is a java-based language.

Resources:

Scala as a platform for statistical computing and data science good blog post discussing a feature list for statistical computing languages

Brief introduction to Scala and Breeze for statistical computing

Julia

Types and "multiple dispatch" are "the core unifying features of Julia: functions are defined on different combinations of argument types, and applied by dispatching to the most specific matching definition." http://docs.julialang.org/en/latest/manual/introduction/

"Although it seems a simple concept, multiple dispatch on the types of values is perhaps the single most powerful and central feature of the Julia language."http://docs.julialang.org/en/release-0.2/manual/methods/#id2

Other

OCaml

Objective Caml for Scientists

F#

Using F# for Data Science

Experimental/Research

- Chapel
- X10 IBM Research; "Both its modern, type-safe sequential core and simple programming model for concurrency and distribution contribute to making X10 a high-productivity language in the HPC and Big Data spaces."

J

J is a hideous language. But "J is particularly strong in the mathematical, statistical, and logical analysis of data."(http://www.jsoftware.com/) If you can stomach the syntax, not to mention the metalanguage (an "adverb" is unary op? a "monad" is a unary function? really?), not to mention the mental model, you may find it useful.

In J, all data is an array. etc.

Here's an example. The following produces all the factorizations of a number:

```
ext=: [: ~. ,&.> , ;@:(tu&.>)
tu =: ] <@:(/:~)@:*"1 [ ^ </\"1@=@]
af =: ext/ @ q:
```

Holy Brainfuck!!! For comparison, here is Hello World written in that infamous language:

```
++++++++[>++++[>++>+++>+++>+<<<<-]>+>+>->>+[<]<-]>>.>---.+++++++..+++.>>.<-.<.+++.------.--------.>>+.>+
```

The obvious question: is the benefit of becoming fluent in such a syntax worth the cost? More to the point: does it really provide anything that you cannot find in some other less alienating language?

Commercial

Part VII

Appendices

Appendices

Bibliography