

Introduction to data cleaning in



Dr N Green
Imperial College London



NEW YORK TIMES
BEST SELLER

—
2 MILLION COPIES
SOLD WORLDWIDE

the life-changing magic of tidying up

the Japanese art of decluttering
and organizing

marie kondo

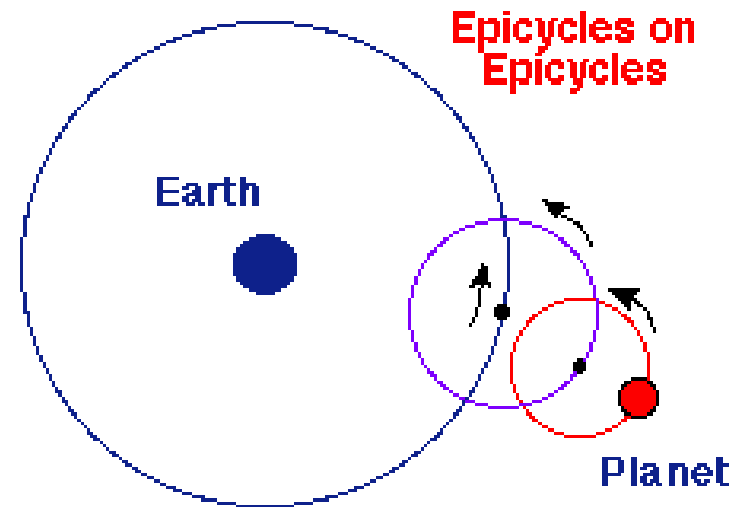
Names for the early steps of “Data Science”

- Data *munging*
- Data *wrangling*
- Web *scraping*
- Data *pulling*
- *Pre-processing*
- Data *cleansing*
- Data *harmonising*
- Data *cleaning*
- <https://www.datacamp.com/community/blog/an-introduction-to-cleaning-data-in-r>

Epicycle of Analysis

(Art of Data Science, Peng)

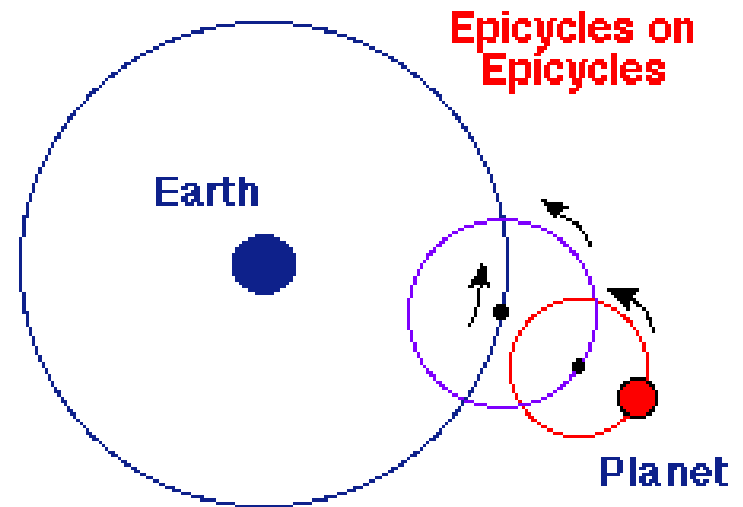
- There are 5 core activities of data analysis:
 - Stating and refining the question
 - Exploring the data (EDA)
 - Building formal statistical models
 - Interpreting the results
 - Communicating the results



Epicycle of Analysis

(Art of Data Science, Peng)

- There are 5 core activities of data analysis:
 - Stating and refining the question
 - Exploring the data (EDA)
 - Building formal statistical models
 - Interpreting the results
 - Communicating the results



EDA

- *“There are no routine statistical questions, only questionable statistical routines.”* —Sir David Cox
- *“Far better an approximate answer to the right question, which is often vague, than an exact answer to the wrong question, which can always be made precise.”* —John Tukey

Exploratory Data Analysis

- (From *Art of Data Science*) Formulate your question:
 - Read in your data
 - Look at the top and the bottom of your data
 - Check your "n"s
 - Validate (with at least one external data source)
 - Make a plot
 - Try the *easy* solution first
 - Follow up...

“Mise en place”!

```
rm(list=ls())
```

```
getwd()
```


```
setwd("<location of your dataset>")
```

- `.Rprofile`: file for customising start-up

OpenRefine

Follow us on: [Github](#) [Twitter](#)

Google Custom Search

A free, open source, powerful tool for working with messy data

Home

Download

Documentation

Community

Post archive

OpenRefine News:
Spring 2016

OpenRefine News:
December 2015

OpenRefine News:
November 2015

OpenRefine News:
October 2015

OpenRefine News:
September 2015

OpenRefine News:
August 2015

OpenRefine News:
July 2015

OpenRefine News:
June 2015

Mapping OpenRefine
Ecosystem

Welcome!


OpenRefine (formerly Google Refine) is a powerful tool for working with messy data: cleaning it; transforming it from one format into another; and extending it with web services and external data.

Please note that since October 2nd, 2012, Google is not actively supporting this project, which has now been rebranded to OpenRefine. Project development, documentation and promotion is now fully supported by volunteers. Find out more about the [history of OpenRefine](#) and how you can help the community.

2017 OpenRefine User Survey

It's been a while since our last user survey (see the result from the 2014 edition), we would like to know who you are, how you use OpenRefine and what your expectations are. So here it is the 2017 edition of the OpenRefine user survey! Thank you for sharing it with your friends, coworker, and communities!

Take the survey



Using OpenRefine - The Book

Using OpenRefine, by Ruben Verborgh and Max De Wilde, offers a great introduction to OpenRefine. Organized by recipes with hands on examples, the book covers the following topics:

1. Import data in various formats
2. Explore datasets in a matter of seconds
3. Apply basic and advanced cell transformations
4. Deal with cells that contain multiple values
5. Create instantaneous links between datasets
6. Filter and partition your data easily with regular expressions
7. Use named-entity extraction on full-text fields to automatically identify topics
8. Perform advanced data operations with the General Refine Expression Language

Introduction to OpenRefine

- <https://github.com/OpenRefine/OpenRefine/wiki/Documentation-For-Users>

Validation

- Check basic characteristics
- Validate with another (external) data source if possible
- Make a simple plot (boxplot, (paired) scatter plot,...)

Reading data from GitHub

```
library(readr)

conception <-
read_csv("https://raw.githubusercontent.com/n8thangreen/ONS/master/conception-under18.csv")
View(conception)
```

Validation type	How it works	Example
Check digit	The last couple of digits can be used as a 'check sum' that can detect if errors have occurred	Bar code readers in shops
Type check	Checks the data is in the right format	Numbers in currency cell must be a monetary value with two decimal points
Length check	Checks the data is an acceptable length	A PIN for online banking needs to be four (or six) characters long
Lookup table	Checks that the value provided matches an item in a set list	A limited set of values, such as the seven days of the week
Presence check	Checks that data has been entered into a field	In most databases a key field cannot be left blank
Range check	Checks that a value falls within the specified range	An online gift certificate purchase must be more than or equal to £5 but less than or equal to £50
Spell check	Looks up words in a dictionary	A search engine often recommends a correct spelling if a word is spelt wrong
Input mask	Checks that data has been entered with the correct amount of characters and/or numbers	National insurance numbers need to be in the format: YY XX XX XX Y, where Y = letter and X = number
Duplicate	Checks that a value has not been repeated	A primary key value can only be entered once

Variables types and indexing

- `numeric` Numeric data (approximations of the real numbers, \mathbb{R})
- `integer` Integer data (whole numbers, \mathbb{Z})
- `factor` Categorical data (simple classifications, like gender)
- `ordered` Ordinal data (ordered classifications, like educational level)
- `character` Character data (strings)
- `raw` Binary data

```
typeof()  # what is it?  
length()  # how long is it? What about two  
dimensional objects?  
attributes()  # does it have any metadata?
```

Example

```
x <- "dataset"  
typeof(x)  
attributes(x)
```

```
y <- 1:10  
typeof(y)  
length(y)  
attributes(y)
```

```
z <- c(1L, 2L, 3L)  
typeof(z)
```

Useful functions

- `head()` – see first 6 rows
- `tail()` – see last 6 rows
- `dim()` – see dimensions
- `nrow()` – number of rows
- `ncol()` – number of columns
- `str()` – structure of each column
- `names()` – will list the names attribute for a data frame (or any object really), which gives the column names.
- `table()`
- `unique()` ; `duplicated()`

String normalisation

```
library(stringr)
str_trim(" hello world ")
## [1] "hello world"
str_trim(" hello world ", side = "left")
## [1] "hello world "
str_trim(" hello world ", side = "right")
## [1] " hello world"

toupper("Hello world")
## [1] "HELLO WORLD"
tolower("Hello World")
## [1] "hello world"
```


String matching (regular expressions)

```
gender <- c("M", "male ", "Female",  
"fem.")  
grepl("m", gender)  
## [1] FALSE TRUE TRUE TRUE  
grep("m", gender)  
## [1] 2 3 4  
grepl("m", gender, ignore.case = TRUE)  
## [1] TRUE TRUE TRUE TRUE  
grepl("m", tolower(gender))  
## [1] TRUE TRUE TRUE TRUE  
grepl("^m", gender, ignore.case = TRUE) ##  
[1] TRUE TRUE FALSE FALSE
```

Running example

```
library("stringr", lib.loc="~/R/win-library/3.2")

conception$Name
conception$Name <- str_trim(conception$Name)
conception$Name <- gsub(pattern = " UA", replacement
= "", conception$Name)
```

Consistent data

- Missing values

```
age <- c(23, 16, NA)
mean(age)
## [1] NA
mean(age, na.rm = TRUE)
## [1] 19.5
```

- Checks

- `is.na()`
- `is.nan()`
- `na.omit()`
- `complete.cases(x)`

Running example

```
apply(conception, 2, function(x) any(is.na(x)))
```

```
data$Name <- toupper(conception$Name)
```

```
chartr("AIai", "@!@!", conception$Name)
```

- Outliers
 - What's realistic? Permissible (integrity)?
(*Outliers: An Evaluation of Methodologies*, Dhiren Ghosh, Andrew Vogt,
https://www.amstat.org/sections/srms/proceedings/y2012/files/304068_72402.pdf)
- What to do?
 - Imputation: substitution mean, logistic regression prediction, matching
 - Truncation, removal, Winsorising, scaling, judgement

Subsetting

```
X[1]
```

```
x[-c(1, 5)]
```

```
x[c(TRUE, TRUE, FALSE, FALSE)]
```

```
x[which(x > 3)]
```

```
z[c("a", "d")]
```

```
df <- data.frame(x=1:3, y=3:1, z=letters[1:3])
```

```
df[df$x == 2, ]
```

The `apply()` family

- `lapply()`
 - Loop over a list and evaluate a function on each element
- `sapply()`
 - Same as `lapply` but try to simplify the result
- `apply()`
 - Apply a function over the margins of an array
- `tapply()`
 - Apply a function over subsets of a vector
- `mapply()`
 - Multivariate version of `lapply`

Examples

```
x <- matrix(rnorm(200), 20, 10)
apply(x, 2, mean)
```

```
x <- list(a = 1:5, b = rnorm(10))
lapply(x, mean)
sapply(x, mean)
```

```
x <- c(rnorm(10), runif(10), rnorm(10, 1))
f <- gl(3, 10)
tapply(x, f, mean)
```

```
rowSums = apply(x, 1, sum)
rowMeans = apply(x, 1, mean)
colSums = apply(x, 2, sum)
colMeans = apply(x, 2, mean)
```


“Tidy” data

- *Happy families are all alike; every unhappy family is unhappy in its own way* [Leo Tolstoy]
- A dataset is said to be *tidy* if it satisfies the following conditions
 - observations are in rows
 - variables are in columns
 - contained in a single dataset.

TB incidence data

```
tb <- read.csv(file =  
"http://stat405.had.co.nz/data/tb.csv",  
header = TRUE, stringsAsFactors = FALSE)
```

- Except for `iso2` and `year`, the rest of the columns headers are actually values of a lurking variable, in fact combination of two lurking variables, `gender` and `age`.)

Reshape(2)

- Create categorical values

```
data$zipGroups <-  
cut(data$zipCode,breaks=quantile(data$zipCode))
```

```
install.packages("Hmisc")  
library(Hmisc)
```

```
data$zipGroups <- cut2(data$zipCode, g=4)  
table(data$zipGroups)
```

Transformations

- absolute value

`abs(x)`

- square root

`sqrt(x)`

- `ceiling(3.4)` is 4

`ceiling(x)`

- `floor(3.4)` is 3

`floor(x)`

- rounding

`round(x, digits=n)`

- `signif(3.475, digits=2)` is 3.5

`signif(x, digits=n)`

- natural logarithm

`log(x)`

- exponentiating x

`exp(x)`

*useful for table formatting

Sorting

```
data <- data.frame( "column1"=sample(1:5),  
  "column2"=sample(6:10), "column3"=sample(11:15) )
```

```
data <- data[sample(1:5),]
```

```
data$column2[c(1,3)]<-NA
```

```
sort(data$column1, decreasing=FALSE)
```

```
sort(data$column2, decreasing=FALSE, na.last=TRUE)
```

```
data[order(data$column1),]
```

Running example

```
conception <- conception[order(conception$Name),]
```

```
conception <- within(conception, new <- `Number of  
Conceptions`*2)
```

```
conception <- conception[order(conception  
$`Conception rate per 1,000 women in age group`),]
```

```
conception <- conception[order(conception  
$`Conception rate per 1,000 women in age group`,  
decreasing = T),]
```