Data Cleaning

# From technically correct data to consistent data

What is consistent data ? It is technically data that are fit for statistical analysis – data which missing values, special values, errors and outliers are either removed, corrected or imputed. The data is consistent with real world constraints and subject matter expertise.

The process towards consistent data can be broken down into 3 steps:

1. Detection of an inconsistency – e.g. age can’t be negative
2. Selection of the field causing inconsistency
3. Correction of the fields

## Detecting and localization of errors

### Missing values

A missing value is represented by NA in R , is a placeholder for a datum that is known but its’ value isn’t. As such, missing values makes it impossible to perform statistical analysis where one or more values in the date are missing.

We can always omit rows which contains missing data, but that depends on many factors, including the percentage of rows which contains missing data.

Sources of errors can include:

* Software driven errors – Excel 2010 silently imputes 0 when you attempt to add one cell with an empty cell
* Confusion of NA in categorical data with unknown – if its categorical data, it should be added as a factor level. For example, consider the categorical variable *place of birth*. An entry unknown can be a factor level data, while NA means we have no information to determine whether the birth place is known or not.

QUESTION 1: Consider the following code in R.

age <- c(23, 16, NA)

mean(age)

**#NOTICE THAT MEAN OF A VECTOR CONTAINING NA WILL RESULT IN NA**

mean(age, na.rm = TRUE)

**#WHAT DOES THE na.rm function do?**

Other functions such as sum, prod, quantile, sd, cor and cov offer this option.

QUESTION 2: Consider the following code in R.

**# READ THE person.txt using read.csv as the object called person**

person <- read.csv( file = "files/unnamed.txt" , header = FALSE , col.names = c("age","height") )

person

**#CHECK FOR THE PRESENCE OF NA IN THE PERSON OBJECT USING THE is.na() function**

**#CHECK FOR WHICH ROWS CONTAINS NA IN THE PERSON OBJECT USING THE complete.cases() function**

(persons\_complete <- na.omit(person))

**#HOW MANY ROWS DOES person and person\_complete contains?**

na.action(persons\_complete)

### Special values

Special values for numerical data includes ±Inf, NA and NaN, and calculations involving these will often result in special values.

Special values are not desirable, and needs to be handled.

Consider the following block of code

is.finite(c(1, Inf, NaN, NA))

**#LET’S CONSIDER A FUNCTION WHICH CHECKS FOR SPECIAL VALUES**

**is.special <- function(x){ if (is.numeric(x)) !is.finite(x) else is.na(x) }**

**sapply(person, is.special)**

### Outliers

An outlier is *an observation (or set of observations) which is inconsistent with that set of data*.

Outliers are not errors- as such it might be detected , but need not always be removed

To test for outliers, we can used the box and whisker plot.

The upper whiskey can be calculated adding the 1.5 times (the coefficient) the interquartile range to the 3rd quartile and rounding to the nearest lower observation, and likewise for the lower whisker

x <- c(1:10, 20, 30)

boxplot.stats(x)$out

**# Why are 20 and 30 detected as outliers?**

boxplot.stats(x, coef = 2)$out

**# What happened above?**

### Obvious Inconsistencies

An obvious inconsistency occurs when a record contains a value or combination of values that cannot fit a real- world scenario. Example, negative values for age, or a man can’t be pregnant etc. and so forth.

These kind of knowledge can be expressed as rules or constraints. In the data editing field, theses rules are know as *edit rules* or *edits* to be precise. Edit rules can be checked using logical indices and recycling, using the *editrules* package.

x\_nonnegative <- x >= 0

**#TO CHECK WHICH ELEMENT OF X OBEY THE ‘NON NEGATIVE’ EDIT RULES**

people <- read.csv("files/people.txt")

people

library(editrules)

(E <- editset(c("age >=0", "age <= 150")))

violatedEdits(E, people)

The editfile() function allows rules to be read in a limited R-syntax, directly from a text file.

# numerical rules

age >= 0

height > 0

age <= 150

age > yearsmarried

… etc

We read the text file in the text above and apply to the people dataset. As the number of rules grow, the full array produced by violatedEdits becomes messy . Thankfully a summary is possible.

E <- editfile("files/edits.txt")

ve <- violatedEdits(E, people)

summary(ve)

plot(ve)

### Error localizations

Edits are frequently interconnected, and changes in one element to prevent an edit from affecting another edit.

Consider the following example

id <- c(2, 5)

people[id, ]

le <- localizeErrors(E, people[id, ], method = "mip")

le$adapt

people[2, "status"] <- "single"

people[5, "height"] <- 7

people[5, "agegroup"] <- "adult"

summary(violatedEdits(E, people[id, ]))

Record 2 violates edits mix6 while record 5 violates num2, cat5, and mix8. We use localizeErrors with mixed-integer programming to minimize the set of variables to adapt.

## Correction

Correction methods aim to fix inconsistent observation by altering invalid values in a record based on information from valid values. This could be a single-step procedure or a two-step procedure (where certain fields are emptied then filled by an imputation step)

### Deductive correction

This is where we deduce the correct data based on certain rules – and can be used to correct errors resulting from rounding errors, sign errors or variable swaps. We can make use of the *deducorrect* package , using functions such as correctRoundings() to fix deviation of one or two measurement units.

Try the following code snippet

library(deducorrect)

e <- editmatrix("x + y == z")

d <- data.frame(x = 100, y = 101, z = 200)

cor <- correctRounding(e, d)

cor$corrected

cor$corrections

**#WHAT HAPPENED HERE?**

e <- editmatrix("x + y == z")

d <- data.frame(x = 100, y = 101, z = 200)

cor <- correctRounding(e, d)

cor$corrected

cor$corrections

**#WHAT WAS THE CORRECTION HERE?**

e <- editmatrix("x + y == z")

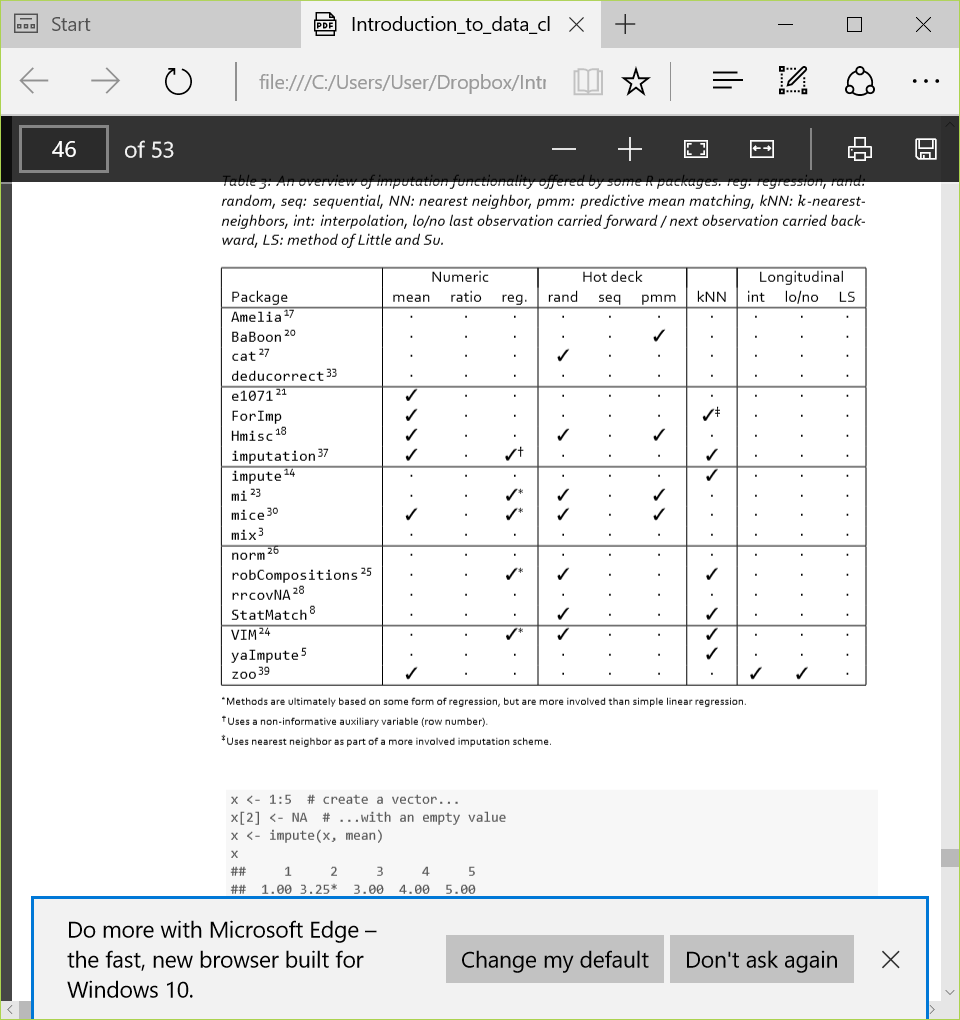
d <- data.frame(x = 123, y = 132, z = 246)

cor <- correctTypos(e, d)

cor$corrected

**#WHAT WAS THE CORRECTION HERE?**

### IMPUTATION



The table above shows the different packages that helps to do imputation.

Try the following block of code, which allows us to impute the **mean** of a object.

library(Hmisc)

x <- 1:5 # create a vector...

x[2] <- NA # ...with an empty value

x <- impute(x, mean)

x

is.imputed(x)

**#REPEAT THIS USING MEDIAN IMPUTATION INSTEAD..LOOK UP THE PARAMETERS FOR THE impute() function**

Another method is to use **(generalized) linear regression models**. Here, we build a linear model for the columns and use the model to predict values for NAs, provided we know the other values.

Try the following block of code

data(iris)

iris$Sepal.Length[1:10] <- NA

model <- lm(Sepal.Length ~ Sepal.Width + Petal.Width, data = iris)

I <- is.na(iris$Sepal.Length)

iris$Sepal.Length[I] <- predict(model, newdata = iris[I, ])

Yet another method is to use **k-NN** to impute the missing values. In 𝑘 nearest neighbor imputation one deﬁnes a distance function 𝑑 (𝑖 ,𝑗) that computes a measure of dissimilarity between records. A missing value is then imputed by ﬁnding ﬁrst the 𝑘 records nearest to the record with one or more missing values. Next, a value is chosen from or computed out of the 𝑘 nearest neighbors. In the case where a value is picked from the 𝑘 nearest neighbors, kNN-imputation is a form of hot-deck imputation – ie missing values are imputed by copying values from similar records in the same dataset.

Try the following block of code

library(VIM)

data(iris)

n <- nrow(iris)

# provide some empty values (10 in each column, randomly)

for (i in 1:ncol(iris)) {

iris[sample(1:n, 10, replace = FALSE), i] <- NA

}

iris

iris2 <- kNN(iris)

any(is.na(iris2))

**#WHAT IS THE FUNCTION OF THE LAST LINE OF CODE IN THIS BLOCK?**

**EXERCISES**

In the following exercises we are going to use the dirty\_iris dataset. You can download this

dataset from

https://raw.github.com/edwindj/datacleaning/master/data/dirty\_iris.csv

Exercise 3.1. Reading and manually checking.

a. View the ﬁle in a text-editor to determine its format and read the ﬁle into R. Make sure that

strings are not converted to factor.

HINT

Assign the URL to an object called fileurl

download.file(fileurl, destfile = "dirty\_iris.csv", mode='wb')

b. Calculate the number and percentage of observations that are complete.

c. Does the data contain other special values? If it does, replace them with NA

Exercise 3.2. Checking with rules

a. Besides missing values, the data set contains errors. We have the following background

knowledge:

– Species should be one of the following values: setosa, versicolor or virginica.

– All measured numerical properties of an iris should be positive.

– The petal length of an iris is at least 2 times its petal width.

– The sepal length of an iris cannot exceed 30 cm.

– The sepals of an iris are longer than its petals.

Deﬁne these rules in a separate text ﬁle and read them into R using editfile (package

editrules). Print the resulting constraint object.

b. Determine how often each rule is broken (violatedEdits). Also summarize and plot the

result.

c. What percentage of the data has no errors?

d. Find out which observations have too long petals using the result of violatedEdits.

e. Find outliers in sepal length using boxplot and boxplot.stats. Retrieve the

corrosponding observations and look at the other values. Any ideas what might have

happened? Set the outliers to NA (or a value that you ﬁnd more appropiate)

Exercise 3.3. Correcting

a. Replace non positive values from Petal.Width with NA using correctWithRules from

the library deducorrect.

b. Replace all erronous values with NA using (the result of) localizeErrors

Exercise 3.4. Imputing

a. Use kNN imputation (VIM) to impute all missing values.

b. Use sequential hotdeck imputation to impute Petal.Width by sorting the dataset on

Species. Compare the imputed Petal.Width with the sequential hotdeck imputation

method. Note the ordering of the data!

c. Do the same but now by sorting the dataset on Species and Sepal.Length.