Data Cleaning Guidelines

10/08/2019

The purpose of this document is to provide you with guidelines for investigating and cleaning data from secondary sources (e.g., National Center for Education Statistics datasets). Data cleaning refers to the process of ensuring that your data is correct, consistent, and useable for analyses by identifying any errors or missing values in the data and either correcting or deleting them.

1. Criteria for Data Quality:

This document will provide a step-by-step process for performing quality checks. But what are these steps trying to accomplish? What is data quality and how do we measure it? High-quality data need to pass the following set of quality criteria:

1.1 Validity:

The extent to which the data conform to defined rules or constraints.

Data-Type Constraints: values in a particular column must be of a particular datatype (e.g., numeric, character, factor).

Range Constraints: typically, numbers or dates should fall within a certain range. That is, they have minimum and/or maximum permissible values (e.g., age cannot be negative).

Mandatory Constraints: certain columns cannot be empty (e.g., student identification number).

Unique Constraints: a field, or a combination of fields, must be unique across a dataset (e.g., social security number).

Set-Membership constraints: values of a column come from a set of discrete values (e.g., a factor variable for compass directions has predefined values of North, South, East, West).

Foreign-key constraints: as in relational databases, a foreign key column can't have a value that does not exist in the referenced primary key. For example, in our recruiting events database, the "state" column is required to belong to one of the US's defined states or territories and the set of permissible states/territories is recorded in a separate States table.

Regular expression patterns: text fields that have to be in a certain pattern (e.g., phone number).

Cross-field validation: certain conditions that span across multiple fields must hold (e.g., a student's graduation date cannot be earlier than their enrollment date).

1.2 Accuracy:

The degree to which the data is close to the true values of what we are attempting to measure. while defining all valid values allows us to easily identify invalid values, it does not mean values are accurate. For example, a students's grade level may be a valid value, 12th grade, but may not be accurate (student is in 10th grade).

Sometimes, achieving accuracy requires accessing an external source containing the true value. For example, the recruiting database queries the google maps API to ensure the web-scraped street addresses of recruiting events actually exist and/or to exract more accurate addresses.

1.3 Completeness:

The degree to which all required data is known. Missing data happens for many reasons. "Fixing" missing data will nearly always require you to return to data documentation by the original source.

1.4 Consistency:

The degree to which the data is consistent, within the same data set or across multiple datasets. Inconsistency occurs when two values contradict each other (e.g., a valid age for a student may not seem to match grade level: age 5 and 12th grade). Fixing inconsistency can be done by deciding which data were more recent or reliable.

1.5 Uniformity:

The degree to which the data is specified using the same unit of measurement (e.g., distance in miles or kilometers)

2. Workflow for Producing High-Quality Data

Below we will review an iterative sequence of steps to produce high-quality data that incorporate the above criteria. Section 2.1 begins these steps by providing several different options for importing data into R.

2.1 Loading libraries and importing data:

Load libraries

```
library(tidyverse); library(tidyr); library(readr); library(lubridate)
library(stringr); library(kableExtra); library(labelled); library(haven)
library(stargazer); library(Hmisc); library(sjlabelled)
```

Import data example: load local file via relative filepath

```
#Use relative file path to data
getwd() # view working directory
load("data/recruiting/recruit_ps1_small.Rdata") #load data
```

Import data example: download, unzip, and read CSV data

```
# Downloading file from the ipeds website
download.file("https://nces.ed.gov/ipeds/datacenter/data/HD2017.zip",
              destfile = "hd2017", mode = "wb")

# Unzip zip file and keep original name
unzip(zipfile = "hd2017" , unzip = "unzip")

# read csv data
  # reading in CSV file does not always read in columns in correct data type, use colClasses to specify
  # na.string tells R to treat both NA and empty strings in columns of character data to missing
hd <- read.csv("hd2017.csv", colClasses=c(ZIP ="character", UNITID="factor"), na.strings=c("","NA"), enc
```

Import data example: "readxl" package reads in excel spreadsheets

```
library(readxl)
# Specify sheet by its name
excel_example <- read_excel("data/fsa/peps300.xlsx", sheet = "300")

# Specify sheet by its index
excel_example <- read_excel("data/fsa/peps300.xlsx", sheet = 1)
```

Import data example: "haven" package reads in SPSS (read_sav), SAS (read_sas), Stata (read_dta)

```
# Import HSLS data from website
hsls <- read_dta(file="https://github.com/ozanj/rclass/raw/master/data/hsls/hsls_stu_small.dta")
```

```
# Import NLS72 data from website
nls_crs <- read_dta(file="https://github.com/ozanj/rclass/raw/master/data/nls72/nls72petscrs_v2.dta", en
```

2.2 Investigate Data

Investigating data is time-consuming and requires using many different methods to detect unexpected, incorrect, and inconsistent data. Here is an outline of general steps to take when investigating data. We provide syntax examples of each step below.

1. Investigate your dataframe
2. Check codebook/documentation
3. Data profiling:
    - duplicates in rows or columns
    - standard pattern: string or number
    - summary statistics: For *numerical*: mean, sd, range, quantiles, distribution, outliers, errors. For *categorical*: labels, frequencies, errors
    - missing values: skip values or true missing? NA, unknown (may not be missing), out-of-range, 0 (missing or true value?)
    - relationship of multiple columns in the same row or across datasets

Example of workflow using NLS72

- Investigate of data frame

```
names(nls_crs) # variable names
glimpse(nls_crs) # prints all columns + type
str(nls_crs) # structure of the data frame
head(nls_crs) # prints first 6 obs
nls_crs %>% var_label() # variable labels
```

- Check Codebook
    - NLS-72 Postsecondary Education Transcript Study Data File User's Manual
    - NLS-72 Postsecondary Education Transcript Files Supplementary Addendum
- Sort, select, and order

```
#Investigate of variable names, variable label
names(nls_crs)  # variable name
var_label(nls_crs)  # variable label

# Sort observations so transcript number, term number, and course name are in ascending order and assig
nls_crs_v2 <- nls_crs %>% arrange(id, transnum, termnum, crsename)

# Select order variables, drop var cname
nls_crs_v2 <- nls_crs_v2 %>% select(id, transnum, termnum, crsecred, gradtype, crsgrada, crsgradb, crse
```

- Preliminary investigation of variables

1. continuous varible

```
#Investigate variable course credits
class(nls_crs_v2$crsecred) #  variable class (numeric)
nls_crs_v2%>% select(crsecred) %>% var_label() #variable labels
options(tibble.print_min=50)
```

```
nls_crs_v2%>% count(crsecred) # frequency

#Run some descriptive stats: range, mean, sd
nls_crs_v2 %>%
  summarise_at(
    .vars = vars(crsecred),
    .funs = funs(min, max, mean, sd, .args=list(na.rm=TRUE))
)

#investigate high values of crsecred
nls_crs_v2%>% filter(crsecred>=100) %>% count(crsecred) # frequency table of crsecred no less than 100
nls_crs_v2%>% filter(crsecred==999) # printing some observations for specific values of crsecred
nls_crs_v2%>% filter(crsecred>=999) %>% count(crsecred) # frequency table of crsecred no less than 999
```

2. categorical- nominal variable

```
#Investigate varible type of grade
class(nls_crs_v2$gradtype) # variable class (labelled)
#glimpse(nls_crs_v2)
nls_crs_v2%>% select(gradtype) %>% var_label() # variable labels
nls_crs_v2%>% select(gradtype) %>% val_labels() # value labels
nls_crs_v2 %>% count(gradtype) #freq count of values
nls_crs_v2 %>% count(gradtype) %>% as_factor() #freq count with value labels
nls_crs_v2%>% filter(gradtype %in% c("9")) #printing some observations for specific values
```

3. categorical- ordinal variable

```
#Investigate variable transcript number
class(nls_crs_v2$transnum) # variable class (numeric)
nls_crs_v2%>% select(transnum) %>% var_label() #variable labels
nls_crs_v2%>% count(transnum) # frequency

#Check that sum of transnum equals number of rows in dataset
nls_crs_v2 %>%
  group_by(transnum) %>% #grouping by transum
  summarise(count_transum = n()) %>% #count for each value of transum
  ungroup() %>% #ungroup
  mutate(total_obs = sum(count_transum)) #Get the sum of count to check that it equals the number of ob
```

- Investigate the relationship between pairs of variables

```
options(tibble.print_min=50)

#Investigate gradtype, crsgrada, crsgradb

#some tabulations for different values of gradtype and crsgrada
nls_crs_v2 %>% group_by(gradtype) %>% count(crsgrada) # cross tab of vars gradtype & crsgrada
nls_crs_v2 %>% group_by(gradtype) %>% count(crsgrada) %>% as_factor() #cross tab this time show value l

nls_crs_v2%>% filter(gradtype==1) %>% count(crsgrada) # letter grade
nls_crs_v2%>% filter(gradtype==2) %>% count(crsgrada) # numeric grade
nls_crs_v2%>% filter(gradtype==9) %>% count(crsgrada) # missing


#some tabulations for different values of gradtype and crsgradb
nls_crs_v2 %>% group_by(gradtype) %>% count(crsgradb) # cross tab of vars gradtype & crsgradb
```

```
nls_crs_v2 %>% group_by(gradtype) %>% count(crsgradb) %>% as_factor() #cross tab this time show value l
nls_crs_v2 %>% group_by(gradtype) %>%
  summarise_at(.vars = vars(crsgradb),
               .funs = funs(min, max, .args = list(na.rm = TRUE))) %>%
  as_factor()

nls_crs_v2%>% filter(gradtype==1) %>% count(crsgradb) # letter grade
nls_crs_v2%>% filter(gradtype==2) %>% count(crsgradb) # numeric grade
nls_crs_v2%>% filter(gradtype==9) %>% count(crsgradb) # missing

#some tabulations for different values of crsgrada and crsgradb
nls_crs_v2 %>% group_by(crsgrada) %>% filter(crsgrada %in% c("A+", "A", "A-")) %>% count(crsgradb) %>% a
nls_crs_v2 %>% group_by(crsgrada) %>% filter(crsgrada %in% c("A+", "A", "A-")) %>%
  summarise_at(.vars = vars(crsgradb),
               .funs = funs(min, max, .args = list(na.rm = TRUE))) %>%
  as_factor()
```

2.3 Cleaning:

Remove, correct, or impute anomalies.

Steps of creating new variables:

Keep input variables the same

Create a new variable based on the following rules

Add variable labels, definitions, value labels

Run summary statistics for input and new variables, compare and check (https://cran.r-project.org/web/packages/assertthat/assertthat.pdf)

Cleaning rules:

Irrelevant data: data (column-wise or row-wise) that are not actually needed, and don't fit under the context of the problem we're trying to solve. Solution: drop it only if it is unimportant for sure.

Duplicates: data points that are repeated in your dataset. Reason: e.g., combination of different data sources; multiple submissions, etc. Solution: remove it if complete duplicate.

Type conversion: numerical or categorical values can be converted into each other: encode categorical values as numbers, or convert numerical values into categories. Caution: the failure of a conversion indicates incorrect value. *Syntax errors*: a. Remove white spaces: Extra white spaces at the beginning or the end of a string should be removed. b. Pad strings: Strings can be padded with spaces or other characters to a certain width. For example, some numerical codes are often represented with prepending zeros to ensure they always have the same number of digits. c. Fix typos: Strings can be entered in many different ways, e.g., fem, female, feml, but only female is expected. Solution: map each value to the unique expected value; use pattern matching; use fuzzy matching.

Standardize: put each value in the same standardized format. E.g., Strings: all values are either in lower or upper case; numerical values: all values have a certain measurement unit.

Scaling / Transformation: scaling: transform the data to fit within a specific scale. E.g., percentage 0-100, GPA 0-5; use log, square root or inverse to reduce skewness

Normalization: transform the data to normally distribution. E.g., normalization of test scores using mean and sd.

Missing values: Solutions: a. flag (avoid losing information): treat them as null or investigate further looking at the codebook and investigating the data. Missing data is informative in itself especially if not at random.

E.g., a specific group refuses to answer a certain question. Caution: 0 might be missing numeric data. Unknown may not be missing. b. Impute (may yield unexpected or biased results): calculate the missing value based on other observations, e.g., using statistical values (mean when data is not skewed, median when data is skewed); generating random number (normally distributed data, random missing); using linear regression to predict.

Outliers: values that are significantly different from all other observations: lies more than 1.5 * interquartile range away from the Q1 and Q3. Caution: investigating and giving a reason before removing weird, suspicious values, or leave them.

In-record & cross-datasets errors: contradict values between columns or across different datasets.

Sample Code (NLS72)

- Missing values

1. treat anomalies as NA

```
#Create measure of course credits attempted that replaces 999 and 999.999 with missing

#investigate
nls_crs_v2%>% count(crsecred)

nls_crs_v2%>% filter(crsecred==999) # printing some observations for specific values of crsecred
nls_crs_v2%>% filter(crsecred==1000) # printing some observations for specific values of crsecred
nls_crs_v2%>% filter(crsecred>=999) %>% count(crsecred) # printing some observations for specific value

#create new variable
nls_crs_v2<- nls_crs_v2%>%
  mutate(crsecredv2= ifelse(crsecred>=900, NA, crsecred))

#check
  nls_crs_v2%>% filter(crsecred==999) %>% select(crsecred,crsecredv2)
  nls_crs_v2%>% filter(crsecred>999) %>% select(crsecred,crsecredv2)
  nls_crs_v2%>% filter(crsecred>900) %>% count(crsecredv2) # one-way frequency table
  nls_crs_v2%>% filter(crsecred>900) %>% group_by(crsecred) %>% count(crsecredv2) # two-way frequency t
```

2.treat missing as NA

```
#type of grade: replace 9 with missing

#investigate
nls_crs_v2%>% count(gradtype) %>% as_factor()

nls_crs_v2%>% filter(gradtype==9) # printing some observations for specific values of crsecred

#create new variable
nls_crs_v2<- nls_crs_v2%>%
  mutate(gradtypev2= ifelse(gradtype==9, NA, gradtype))

    #add variable label
  nls_crs_v2 <- nls_crs_v2 %>%
  set_variable_labels(
    gradtypev2 = "new type of grade"
  )

   #add value label
  nls_crs_v2 <- nls_crs_v2 %>%
  set_value_labels(gradtypev2 = c("letter grade" = 1,
                                  "numeric grade" = 2)
   )

#check
  nls_crs_v2 %>% filter(gradtype==9) %>% select(gradtype,gradtypev2)
  nls_crs_v2 %>% filter(gradtype==9) %>% count(gradtypev2) # one-way frequency table
  nls_crs_v2 %>% filter(gradtype==9) %>% group_by(gradtype) %>% count(gradtypev2) # two-way frequency t
```

- Create new variable

1. create new numeric grade variable when type of grade is numeric & course credit not missing

```
#investigate values of the numeric grade when gradtype==2 and course credit not missing

  typeof(nls_crs_v2$crsgradb)
  class(nls_crs_v2$crsgradb)

  options(tibble.print_min=300)
  nls_crs_v2%>% filter(gradtype==2, (!is.na(crsecredv2))) %>% count(crsgradb)

#create new variable
  nls_crs_v2<- nls_crs_v2%>% mutate(crsgradbv2= ifelse(crsgradb<=4 & gradtype==2 & (!is.na(crsecredv2))
  #%>%   filter(gradtype==2)

#check
    nls_crs_v2%>% filter(is.na(crsecredv2)) %>% count(crsgradbv2) # course credit is missing
    nls_crs_v2%>% filter(gradtype %in% c(1,9)) %>% count(crsgradbv2) # course credit is missing
    nls_crs_v2%>% filter(crsgradb>4) %>% count(crsgradbv2) # course credit is missing
    nls_crs_v2%>% filter(crsgradb<=4, gradtype==2, (!is.na(crsecredv2))) %>% count(crsgradbv2) # tabula
    nls_crs_v2%>% filter(crsgradb<=4, gradtype==2, (!is.na(crsecredv2))) %>% group_by(crsgradb) %>% cou
    nls_crs_v2%>% filter(crsgradb<=4, gradtype==2, (!is.na(crsecredv2))) %>% mutate(assert=crsgradb==cr
```

2. Create new numeic grade variable based on letter grade (when type of grade is letter) & numeric grade (when type of grade is numeric)

```
# investigate
  nls_crs_v2%>% count(gradtype)
  nls_crs_v2%>% count(gradtype) %>% haven::as_factor()

  nls_crs_v2%>% filter(gradtype==1) %>% count(crsgrada)

  #options(tibble.print_min=200)

#create new variable
  nls_crs_v2<- nls_crs_v2%>%
    mutate(
      numgrade=case_when(
        crsgrada %in% c("A+","A") & gradtype==1 & (!is.na(crsecredv2)) ~ 4,
        crsgrada=="A-" & gradtype==1 & (!is.na(crsecredv2)) ~ 3.7,
        crsgrada=="B+" & gradtype==1 & (!is.na(crsecredv2)) ~ 3.3,
        crsgrada=="B" & gradtype==1 & (!is.na(crsecredv2)) ~ 3,
        crsgrada=="B-" & gradtype==1 & (!is.na(crsecredv2)) ~ 2.7,
        crsgrada=="C+" & gradtype==1 & (!is.na(crsecredv2)) ~ 2.3,
        crsgrada=="C" & gradtype==1 & (!is.na(crsecredv2)) ~ 2,
        crsgrada=="C-" & gradtype==1 & (!is.na(crsecredv2)) ~ 1.7,
        crsgrada=="D+" & gradtype==1 & (!is.na(crsecredv2)) ~ 1.3,
        crsgrada=="D" & gradtype==1 & (!is.na(crsecredv2)) ~ 1,
        crsgrada=="D-" & gradtype==1 & (!is.na(crsecredv2)) ~ 0.7,
        crsgrada %in% c("F","E","WF") & gradtype==1 & (!is.na(crsecredv2)) ~ 0,
        crsgradb<=4 & gradtype==2 & (!is.na(crsecredv2)) ~ crsgradb # use values of numeric var crsgrad
      )
    )
  #add variable label
  nls_crs_v2 <- nls_crs_v2 %>%
  set_variable_labels(
    numgrade = "new numberic grade"
  )

#check
  nls_crs_v2%>% count(numgrade)
  nls_crs_v2%>% filter(is.na(crsecredv2)) %>% count(numgrade) # missing when crsecredv2==NA
  nls_crs_v2%>% filter(gradtype==9) %>% count(numgrade) # missing when grade-type is not "letter"
  nls_crs_v2%>% filter(gradtype==1, (!is.na(crsecredv2))) %>% count(numgrade) # when grade-type=letter 
  nls_crs_v2%>% filter(gradtype==1, (!is.na(crsecredv2))) %>% group_by(crsgrada) %>% count(numgrade)  #

  #check against values of crsgradb
  nls_crs_v2%>% filter(crsgradb>4, gradtype==2, !is.na(crsecredv2)) %>% group_by(crsgradb) %>% count(nu
```

3. Create quality points variable based on the relation: quality points = credits attempted * numeric grade

```
#create new variable
  nls_crs_v2<- nls_crs_v2%>% mutate(qualpts=numgrade*crsecredv2)

    #add variable label
  nls_crs_v2 <- nls_crs_v2 %>%
  set_variable_labels(
    qualpts = "quality points"
  )

#check
  nls_crs_v2 %>%
    count(qualpts)#freq count of new variable
  nls_crs_v2 %>%
    select(id, transnum, numgrade, crsecredv2, qualpts)
  nls_crs_v2%>% filter(is.na(numgrade)) %>% count(qualpts) # missing when numgrade==NA
  nls_crs_v2%>% group_by(numgrade, crsecredv2) %>% count(qualpts) #group by input variables and get a c
  nls_crs_v2%>% filter(numgrade==0 & qualpts!=0) %>% select(numgrade,crsecredv2,qualpts) #If variable w
  nls_crs_v2%>% filter(crsecredv2==0 & qualpts!=0) %>% select(numgrade,crsecredv2,qualpts) #same logic 
```

4. Create institution-level GPA variable based on the relation: institution-level GPA = sum of quality points / sum of total credits

```
# Create institution-level GPA variable and save as a new object

    nls_crs_trans <- nls_crs_v2%>% group_by(id,transnum) %>%
      summarise(
        cred_trans=sum(crsecredv3, na.rm=TRUE), # sum total credits attempted where grade is known
        qualpts_trans=sum(qualpts, na.rm=TRUE) # sum of quality points where grade is known
      ) %>%
      mutate(gpa_trans=qualpts_trans/cred_trans)

    nls_crs_trans
    #options(tibble.print_min=400)

    #add variable label
  nls_crs_trans <- nls_crs_trans %>%
  set_variable_labels(
    gpa_trans = "institution-level GPA"
  )

# check
  nls_crs_trans %>%
    count(gpa_trans) #freq count of new variable
  nls_crs_trans %>%
    filter(is.na(gpa_trans)) #view NA for new variable
  nls_crs_trans %>%
    filter(cred_trans==0 & qualpts_trans!=0) #checking to see if cred_trans equals 0 and qualpts_trans
  nls_crs_trans %>%
    filter(cred_trans==0 & !is.na(gpa_trans)) #checking to see if cred_trans equals 0 and gpa_trans doe
  nls_crs_trans %>%
    filter((qualpts_trans/cred_trans) != gpa_trans)
  nls_crs_trans %>%
    count(gpa_trans) %>%
    filter(n > 1)
```

2.4 Reshape and/or merge data:

reshape:

- from wide to long

gather(data, key, value, . . . , na.rm = FALSE, convert = FALSE)

```
table208_30_tidy <- table208_30 %>%
  gather(`2000`,`2005`,`2009`,`2010`, `2011`,  key = year, value = total_teachers)
```

- from long to wide

spread(data, key, value, fill = NA, convert = FALSE)

```
all_obs_tidy <- all_obs_temp %>%
  spread(key=age_lev, value=efage09)
```

merge:

```
load("jkcf_opeid5_list.RData")
jkcf_opeid5_mrc3 <- jkcf_opeid5_list %>%
  left_join(mrc3_by_cohort, by = "opeid5")

anti_test <- jkcf_opeid5_list %>%
  anti_join(mrc3_by_cohort, by = "opeid5")
```

2.5 Export:

Save RData dataframe

```
saveRDS(hsls, "hsls.rds")
```

Save Stata dataframe

```
write_dta(hsls, path = "../../data/hsls/hsls_sch_small.dta")
```

Export to CVS

```
write_csv(hsls, path = "../../data/hsls/hsls_sch_small.dta")
```

##2.6 Report: A report about the changes made and the quality of the currently stored data is recorded.

location: arithmetic mean, median, mode, interquartile mean

spread: standard deviation, variance range, interquartile range, absolute deviation, mean absolute difference, distance standard deviation

shape: skewness, kurtosis distribution

dependence: correlation coefficient, distance correlation

cross tabulation, compare means (t-test), ANOVA, correlation

visualizations: histogram, frequency distribution table, box plot, bar char, pie chart, scatter plot

##questions: create R data dataframe? rename variable? standardize names?