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 usable 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: validity, accuracy, completeness, consistency, and uniformity.

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).

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).

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

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).

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).

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.

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 student'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 extract 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. Investigating the data before any manipulation provides you with a general idea about the quality of the data. While this is not a linear process, here is an outline of general steps to take when first investigating data. We provide syntax examples of each step below.

1. Investigate the dataframe
2. Sort, select, and order data to make it easier to view the dataframe
3. Check codebook/documentation
4. Preliminary variable investigations:
    - check for missing observations across all variables
    - identify the "unit of analysis" and which combination of variables uniquely identify observations
    - investigate different variable types: numeric, factor, character, labelled, etc.
    - run some summary statistics. For *numerical* variables: mean, sd, range, quantiles, distribution, outliers, errors. For *categorical* variables: labels, frequencies.

Example of Investigating Data Using NLS72

- Investigate the data frame

```
str(nls_crs) # structure of the data frame

nrow(nls_crs) #num of rows (obs)
ncol(nls_crs) #num of columns
dim(nls_crs) #rows by columns

names(nls_crs) # variable names
glimpse(nls_crs) # prints all columns + type
nls_crs %>% var_label() # variable labels

head(nls_crs) # prints first 6 obs
```

- Sort, select, and order

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

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

- Preliminary investigation of variables

1. continuous variable

```
#Investigate variable course credits
class(nls_crs_v2$crsecred) #  variable class (numeric)
attributes(nls_crs_v2$crsecred)


nls_crs_v2%>% select(crsecred) %>% var_label() #variable label
```

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

#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 greater than or e
```

2. categorical- nominal variable

```
#Investigate varible type of grade
class(nls_crs_v2$gradtype) # variable class (labelled)
nls_crs_v2%>% select(gradtype) %>% var_label() #variable label
attributes(nls_crs_v2$gradtype)


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$transnum) # variable class (numeric)
attributes(nls_crs_v2$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
```

- Check Codebook
  - NLS-72 Postsecondary Education Transcript Study Data File User's Manual
  - NLS-72 Postsecondary Education Transcript Files Supplementary Addendum

2.3 Cleaning Data and Creating New Variables:

Data cleaning involves removing, correcting, or imputing anomalies. This process usually involves the creation of new variables (i.e., analysis variables). To create new variables, you should:

1. Investigate input variable(s)

    - keep survey questionnaire and codebooks on hand to understand how input variables were created
    - explore one-way investigations of input variables
    - explore relationships between input variables
2. Keep input variable(s) the same
3. Create a new analysis variable from input variable(s)
4. Add variable labels, definitions, value labels
5. Run summary statistics for input and new variable to compare and verify new variables were created correctly

In following these steps, the goal is to identify and correct:

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.

Duplicates: data points that are repeated in your data due to a combination of different data sources, multiple submissions, etc.

Type conversion: data that needs to be converted into a different type/class in order to be stored correctly. For example, a date variable should be stored as a date object or a categorical variable should be converted from a string type to a factor class.

Syntax errors: character data that needs some string manipulation in order to be correct or be used in analyses. Such as:

- Removing white spaces: Extra white spaces at the beginning or the end of a string should be removed.
- Padding strings: Strings can be padded with spaces or other characters to a certain width. For example, some numerical codes are often represented with leading or trailing zeros to ensure they always have the same number of digits (e.g., ID variables).
- Fixing typos: strings can be entered in many different ways (e.g., fem, female, f). This can often lead to errors. These inconsistencies can be fixed by mapping each value to the unique expected value via pattern matching or fuzzy matching (i.e., regular expressions).

Standardize: data points that are equivalent but do not have the same standardized format. For example, strings should all either be in lower or upper case or numerical values should all have a certain measurement unit.

Scaling / Transformation: data that need to be transformed in order to fit within a specific scale. For example, variables indicating a percentage should range from 0 to 100 (rather than proportional range 0 to 1), grade point average should range from 0 to 4.0 (if using a 4.0 scale), or using log, square root, or inverse to reduce skewness.

Normalization: data that need to be rescaled into a range from 0 to 1 in order to be normally distributed. For example, test scores data can be normalized based on the mean and standard deviation.

Missing values: data that have missing observations. Missing data are a common problem across data sources. Ignoring missing data or treating them incorrectly due to not understanding rationale(s behind missing data is a fatal error! Missing data are generally treated in three ways:

- Drop missing observations if occurring at random: If missing values for a variable rarely occur and occur at random, sometimes it's easiest to drop missing observations. While this may reduce the total number of observations in your dataset this is sometimes useful when conducting statistical analyses where "filling" missing observations could bias your results.
- Flagged to avoid losing information: sometimes missing data do not occur at random. If so, we usually do not want to drop these observation because the missing data is informative in itself (For example, on a campus climate survey question, all missing values are the result of people from the same race/ethnic group refusing to answer the question). Missing observations for numeric data can be converted to `NA` and can be ignored when calculating any statistical computation or plotting distributions. For categorical data, you can create a separate "Missing" category and assign this category to all missing observations.

- Impute missing data: replace missing data by calculating values based on other observations. For example, you can replace all missing observations with statistical values (e.g., mean, median), or generate random values within 1 or 2 standard deviations of the mean if the data is normally distributed data and randomly missing, or using linear regression to predict the value of the missing observation based on relationships with other variables. Keep in mind these imputations may yield unexpected or biased results.

Outliers: observations in the data that deviate substantially from all other observations. For example, the 2017 average annual household income in Seattle is $100,630 but Bill Gates, an outlier, earned $14.7 billion. "True values until proven otherwise" is a rule of thumb to follow when investigating outliers. Outliers should only be removed with good reason.

In-record & cross-datasets errors: these are data observations that contradict values between columns or across different datasets. For example a student's grade level value is 1st grade but marital status is divorced.

Example of Cleaning Data and Creating Variables Using NLS72

Task: Create a numerical grade variable based on input variables `crsgrada`, `crsgradb`, `gradtype`, and `crsecred_v2`.

Investigate input variable(s): One Way Investigations

```
# crsgrada

  #investigate type, class, and attributes
  typeof(nls_crs_v2$crsgrada) #character variable
  class(nls_crs_v2$crsgrada)
  attributes(nls_crs_v2$crsgrada)

  #one-way investigation
  nls_crs_v2%>% count(crsgrada) #frequency count
  nls_crs_v2%>% filter(crsgrada %in% c("99", "AU", "CR", "I", "NO", "P", "S", "U", "W", "WP")) #printin

  #missing values
  nls_crs_v2%>% count(crsgrada) #per supp addendum doc p.16, value 99 is "unknown" or "missing"


# crsgradb

  #investigate type, class, and attributes
  typeof(nls_crs_v2$crsgradb) #numeric double variable
  class(nls_crs_v2$crsgradb)
  attributes(nls_crs_v2$crsgradb)

  #one-way investigation
  nls_crs_v2 %>% #run some descriptive stats
  summarise_at(
    .vars = vars(crsgradb),
    .funs = funs(min, max, mean, .args=list(na.rm=TRUE))
  )

  #missing values
  nls_crs_v2 %>% filter(crsgradb>4) %>% count(crsgradb) #per supp addendum doc p.17, values 999 and 999
```

```
# gradtype

  #investigate type, class, and attributes
  typeof(nls_crs_v2$gradtype)
  class(nls_crs_v2$gradtype)
  attributes(nls_crs_v2$gradtype)

  #one-way investigation
  nls_crs_v2%>% count(gradtype) #frequency count
  nls_crs_v2%>% filter(gradtype %in% c(1, 2)) #printing some obs for specific values

  #missing values
  nls_crs_v2%>% filter(gradtype %in% c(9)) #per codebook, pg.58 definition, pg. 523 true categories, va



# crsecred

  #investigate type, class, and attributes
  typeof(nls_crs_v2$crsecred)
  class(nls_crs_v2$crsecred)
  attributes(nls_crs_v2$crsecred)


  #one-way investigation
  nls_crs_v2 %>% #run some descriptive stats
  summarise_at(
    .vars = vars(crsgradb),
    .funs = funs(min, max, mean, .args=list(na.rm=TRUE))
  )

  #missing values
  nls_crs_v2 %>% filter(crsgradb>4) %>% count(crsgradb)
  #per codebook pg. 49: 70% of numeric grades were reported on a scale of 0 to 4
  # the remaining were recorded on different scales (e.g., 0-100)
  # no attempts were made to place numeric grades on a common metric
```

Investigate input variable(s): Two Way Investigations

```
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()
```

Keep Input Variable(s), Create Analysis Variables, Treat Missing Values 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
```

```
#Create measure of grade scale that replaces 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 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
```

```
# Create new numeric grade variable based on letter grade (when type of grade is letter) & numeric grad

# 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 numeric 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
```

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

```r
saveRDS(hsls, "hsls.rds") #  could only save 1 R object
save(nls_crs_v2, nls_crs_trans, file = './dataset.RData') # for saving multiple R objects
```

Save Stata dataframe

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

Export to CVS

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