

# Data Cleaning

## 1) General approach to cleaning data

- objective is to use the available data to form an analysis data set, which
  - has measures that correspond as closely as possible to the variables in our conceptual model, hypothesis, or research question
  - can support an analysis of the behaviors and relationships of the conceptual model or hypothesis
  - is representative of the population of interest or a population of interest
  - has measures with good properties (avoids outliers, mistaken reports, etc.)
- raw data from surveys are rarely in a form that permits immediate analysis
- the process of taking the data from its raw form to an analysis form is called “cleaning the data”
- in this class we will
  - talk about some general data checking procedures
  - discuss some SAS commands that are helpful for cleaning and transforming data
  - discuss some SAS commands for documenting measures

## 2) Some practical data cleaning tips

- first and most importantly, make sure that the data have been read in correctly
  - potential problems with column formatted data if the column specifications get mixed up; data can also be only partially downloaded
  - check that the number of observations matches the documentation
  - also check the ranges of your variables match the values from the codebook
- you need to determine how the relevant sample was collected and who was a respondent
  - some data sets are simple and only contain information on respondents
  - other data sets have different respondents at different times or for different data items
  - for example, longitudinal surveys have people who are respondents in some waves but not others
  - you need to find the variables that identify respondents (such as the why non-response variables in the PSID) and make sure that you can reproduce the original sample totals
- a related issue is that you need to familiarize yourself with any “skip patterns” or special universes for data that you may be interested in
  - a “skip pattern” refers to questions in a survey that are only asked conditionally; people who don’t meet the conditions, skip the questions
  - the “universe” for a data item refers to types of study subjects that are actually asked a question or for whom the item is constructed
  - you should check the conditioning variables and verify that you can reproduce the universes for all data items
- “top codes” and other special values in data
  - many data sets reserve special values to represent missing data, refusals, don’t knows, or not being in the data universe

- we usually remember to look for these in categorical data; however, they also sometimes appear in continuous data
- one particular type of special code is a “top code,” which can be used to indicate missing data, values beyond a certain range (for instance incomes or weekly hours above certain thresholds), or other conditions
- for example, some data sets record valid usual weekly work hours as being between 0 and 97, code usual hours in excess of 97 with a 98, and code variable hours (i.e., people who work but have irregular schedules) with a 99
- for privacy reasons, many government data sets recode incomes above a certain amount at either that particular threshold or at the mean of the censored incomes
- as much as is reasonable and possible, you want to check the consistency and reasonableness of the data items
  - to do this you need related or conceptually conditional variables
  - for example, you might compare people’s reported hours of work and earnings with their reported employment status (are there non-workers who are reporting positive hours or earnings)
  - examine the distributions of variables for implausibly high or low values, such as wages that are substantially below the minimum wage or levels of education or experience that exceed people’s ages
  - more generally, look for outliers in the data
- in longitudinal data, look for consistency in distributions and definitions of variables over time
- examine the data for unusual patterns of missing responses
  - for example, an examination of the employment status information in the 2005 PSID individual file revealed that there were a surprisingly high number of missing responses and almost no valid responses for heads and wives; a closer look through the documentation revealed that heads’ and wives’ information was stored in another file and had not been cleaned
  - unusual patterns may indicate a skip pattern that you’ve overlooked or some other misunderstanding of the data
- some analysis variables will need to be constructed
  - you want to apply the same consistency and reasonableness checks to these data as to the raw data
  - because these data are constructed from other underlying data, additional consistency checks are possible
  - for example, if you construct a categorical measure from another underlying measure, you could use PROC FREQ or PROC TABULATE to look at the conditional distributions and make sure that they match up with your intended coding
  - the consistency checks for constructed variables are intended to catch programming mistakes, that is to help debug your programs
  - the more complicated the data construction procedures, the more thorough you need to be in your checking
  - for especially complicated or tricky procedures, selected hand-checking may be necessary; in a hand check, you would actually go line by line through the data and the construction steps, making sure that all of the programming and logic is sound

- compare your data to other data sets

### 3) Conditional processing

- as the preceding discussion suggests, a lot of the processing in data cleaning is conditional
- we have already gone over IF-THEN/ELSE statements; these are the workhorse statements of conditional processing
- an alternative for processing *mutually exclusive conditions involving a single variable or expression* is the SELECT sequence
  - the syntax for the statement is

```
SELECT <(select-expression)>;
      WHEN (when-expression-1, <more when expressions>) statement;
```

...

```
      <WHEN (when-expression-n, <more when expressions>) statement>;
      <OTHERWISE statement>;
      END;
```

- the (*select-expression*) can be any SAS expression that produces a single value, including character values
  - ◊ the select-expression is optional (you don't have to include one)
  - ◊ if an expression is used, it must appear in parentheses
- similarly, the (*when-expression[s]*) can be any SAS expressions, including constants, that produce a single value
  - ◊ when-expressions are not optional, at least one must be included
  - ◊ these expressions also appear in parentheses
  - ◊ if a select expression is provided, SAS compares the when and select expressions
  - ◊ if a select expression is not provided, the when expressions must be logical expressions, and they are evaluated on their own
  - ◊ the when expressions are evaluated sequentially; once a when expression is true and the corresponding statement is executed, SAS steps out of the SELECT sequence
- the OTHERWISE statement is optional and operates like an unconditional ELSE statement
- SAS expects a single statement to follow the WHEN and OTHERWISE conditions
  - ◊ to execute multiple statements within a single condition use the DO-END construct
  - ◊ you can also include a null statement (just a semicolon)
- note the conditions in the SELECT sequence must be exhaustive of the possible conditions; if SAS steps through the entire sequence without finding a true condition, it will issue an error message and stop the DATA step at that point
- SELECT statement is easier to read and more efficient (faster) than a comparable IF-THEN/ELSE if you have many conditions
- examples: suppose that you have a categorical variable *x* that takes on values 1-5
  - ◊ example with a select-expression

```

SELECT (x);
  WHEN (1)      y = x;
  WHEN (2,3,4)  y = x**2;
  OTHERWISE;
END;

```

y will have values if x is less than 5 and will be missing if x

◇ equivalent example without a select-expression

```

SELECT;
  WHEN (x EQ 1)      y = x;
  WHEN (x IN (2,3,4))  y = x**2;
  OTHERWISE;
END;

```

- you might consider using a null statement with the OTHERWISE condition to trap for unexpected results from the expression comparisons (and to keep SAS from stopping in the middle of your DATA step); alternatively you could write an error message to the LOG, e.g.

```
OTHERWISE PUT 'Unexpected SELECT at obs. ' _N_ ', X = ' x;
```

- to debug conditional processes, it is important that you go through them and consider what will happen under all circumstances

#### 4) SAS functions

- SAS has a number of functions that help with manipulating data
- general use
  - functions are most commonly used in assignment statements; however, they can also be used in most places where SAS evaluates expressions
  - the general syntax for a function is *function\_name(argument\_list)*
  - the arguments themselves can be SAS expressions, including functions; multiple arguments are delimited by commas
  - some functions, such as the DATE function, don't use arguments; for these, you would just include parentheses but nothing between them
  - invoking a function will cause a value to be returned
- some categories of functions are listed below along with some specific functions
- arithmetic, mathematical, trigonometric, and hyperbolic
  - ABS(*argument*) returns the absolute value of the argument
  - EXP(*argument*) returns the exponential of the argument
  - LOG(*argument*) returns the natural logarithm of the argument
  - LOG10(*argument*) returns the base-10 logarithm of the argument
  - MAX(*argument\_list*) returns the maximum value from the list
  - MIN(*argument\_list*) returns the minimum value from the list

- MOD(*arg1,arg2*) returns the remainder from *arg1/arg2*
- N(*argument\_list*) returns the number of non-missing values from list
- NMISSED(*argument\_list*) returns the number of missing values from list
- SQRT(*argument*) returns the square root of the argument
- SUM(*argument\_list*) returns the sum of the values from list
- character
  - LEFT(*argument*) left aligns a character expression
  - RIGHT(*argument*) right aligns a character expression
  - SUBSTR(*arg1,pos,n*) extracts *n* characters from *arg1* starting at *pos*
- date and time
  - DATE() returns an integer representing today's date
  - DAY(*argument*) returns the day of the month from a SAS date value
  - MDY(*mm,dd,yyyy*) returns the SAS date value for *mm/dd/yyyy*
  - MONTH(*argument*) returns the month (1-12) from a SAS date value
  - YEAR(*argument*) returns the year from a SAS date value
- geographic
  - FIPSTATE(*argument*) converts FIPS code to state postal abbreviation
  - STFIPS(*argument*) converts the state postal abbreviation to a FIPS code
  - ZIPCITY(*argument*) converts a ZIP code to a city name and postal abbrev.
- probability and quantile (inverse probability)
  - PROBCH(*arg1,df*) chi-square cumulative distribution function
  - PROBF(*arg1,ndf,ddf*) F cumulative distribution function
  - PROBNORM(*argument*) standard normal cumulative distribution function
- random number
  - NORMAL(*seed*) returns a pseudo-random std. normal variable using *seed*
  - RANEXP(*seed*) returns a pseudo-random exponential variable using *seed*
  - RANUNI(*seed*) returns a pseudo-random uniform (0,1) variable using *seed*
- sample statistics
  - CV(*argument\_list*) calculates the coefficient of variation of the arguments
  - MEAN(*argument\_list*) calculates the mean of the arguments
  - STD(*argument\_list*) calculates the standard deviation of the arguments
- truncation
  - CEIL(*argument*) rounds the argument up to the next integer
  - FLOOR(*argument*) rounds the argument down to the next integer
  - INT(*argument*) truncates the argument to an integer
  - ROUND(*argument*) rounds the argument to the nearest integer
- for more information and the complete list of SAS functions, see <http://support.sas.com/documentation/cdl/en/lrdict/59540/HTML/default/a000245852.htm>

## 5) Documenting your data

- programs should be documented with comments
- output should be documented using the TITLE and FOOTNOTE commands
- additionally, variables themselves should be labeled and possibly formatted
- LABEL statement

- SAS restricts the length of a variable name to 32 characters and restricts the types of characters and symbols that can be used in a name
- LABELs allow you to provide more detailed names
- the syntax for the label statement is

```
LABEL    <variable1> = '<label1>'
          <variable2> = '<label2>'
          ...
          <variablen> = '<labeln>';
```

- LABEL statements can be used in DATA steps or in other procedures
  - ◇ if a LABEL statement is used in a DATA step, the LABEL stays with the variable when the SAS data set from the step is created
  - ◇ if a LABEL statement is used in a reporting procedure, such as GPLOT or TABULATE, the LABEL only remains with the variable for that procedure
- to remove a previously applied label, use the command

```
LABEL    <variable1> = ;
```

- it is sometimes necessary to remove or modify a label to get output to print reasonably, e.g., to get tables in a MEANS or TABULATE procedure to fit on a single page