```
/*************************************************************************
                    Week 04. Read Data File with Commonly Methods
     Instructor: C. Peng
           Date: 02/10/2021
         Topics: 1. Handling challenging source data
                    (text data via INPUT or INFILE-INPUT Statements)
                 2. Read DELIMITERED data files
                 3. Read data using INPUT WIZARD
                 4. Three basic types data formats: Excel, CSV and Text
                 5. Loading SAS dataset to SAS
                 6. EXPORT SAS data to CSV
*************************************************************************/

DM 'CLEAR LOG';
DM 'CLEAR OUT';

OPTIONS PS=90 LS=70 NOCENTER NONUMBER NODATE;

/** library on the RAMCLOUD - sychromizing files between my computer and
    my OneDrive on the RamCloud                                          **/
LIBNAME w04 "C:\STA311\w04";

/*NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
 Topic 1: Reading challenging data: external and in-line
          Challenges: missing values, character variable's
                      string value contains blanks, special
                      formatted numerical values, etc.
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN*/

/* Missing values at the end of the record: MISSOVER!!
   INFILE statement's MISSOVER option to read records when
   some of the values are missing at the end of a record  */

DATA reading;                          /* temporary SAS data  */
      INFILE DATALINES  MISSOVER;      /* INFILE-DATALINES read in-line data */
      INPUT Name $ Week1-Week5;        /* shortcut of patterned variable names */
      /* Caution: records 3 and 4 have missing values at the end of the records.  */
      DATALINES;
Robin 3 2 5 1 6
Jack 4 4 4 3 4
Tim 3 0 0
```

```
Martin 1 0 1 1
Caroline 2 3 4 5 6
;
RUN;

PROC PRINT DATA = reading;
      TITLE 'Summer Reading Program';
      VAR Name Week1-Week5;    /* Caution: in proc print, the variable type
                                  should NOT be specified. Otherwise, an error
                                  will be written to the log               */
RUN;



/* Missing values in the middle or at the end of the record: DSD!!
   INFILE statement's DSD option to read records when some of the values
   are missing in the middle or at the beginning of a record          */

DATA survey;
    LENGTH Name $ 9;                  /* Katherine has 9 characters!   */
      INFILE DATALINES DLM=',' DSD;   /* 1. in-line data is commoa delimited
                                         2. missing values occured in the
                                            middle of the records */
      INPUT Name $ (Q1-Q5) ($);      /* Pay attention to the character
                                        variables with patterned names! */
      DATALINES;
Robert,,A,C,A,D
William,B,C,A,D,A
Linda,C,B,,A,C
Lisa,D,D,D,C,A
Katherine,A,B,C,D,A
RUN;

PROC PRINT DATA = survey;
      TITLE 'Survey Results';
      VAR Name Q1-Q5;           /* Requested character variables without the
                                   specification of var type!             */
RUN;

/* The following program illustrates that the DSD option can also be used
      i) when there is a missing value at the beginning of a record, and
```

```
     ii) when the data are delimited by blanks (in conjunction with the DLM= option): */

DATA survey;
    LENGTH Name $ 9;
      infile DATALINES DLM=' ' DSD;
      input Name $ (Q1-Q5) ($);
      DATALINES;
Robert  A C A D
William B C A D A
Linda C B  A C
 D D D C A
Katherine A B C D A
RUN;

PROC PRINT data = survey;
      title 'Survey Results';
      var Name Q1-Q5;
RUN;


/** Modifiers (&) and (:) - modifier list input for some challenges in text data files:
    1. The ampersand (&) modifier allows you to read character values that
       contain embedded blanks.

       CAUTION: The ampersand (&) that follows the city variable in the INPUT
       the statement tells SAS that the city values may contain one or more
       SINGLE embedded blanks. Because the ampersand modifier is used, SAS
       will read the city value UNTIL TWO OR MORE CONSECUTIVE blanks are
       encountered.

       That is a very important point!!!!

       When you use ampersand modified list input, the values that you are
       reading in must be separated by two or more consecutive blanks.

       You cannot use any other delimiter to indicate the end of each field.

    2. The colon (:) modifier allows you to read nonstandard data values and
       character values that are longer than eight characters, but which have no embedded blanks.
**/
```

```
DATA citypops;
   INFILE DATALINES FIRSTOBS = 2;
   INPUT city pop2000;   /* This will not correctly!
                            (1) city is character.
                            (2) Some of its value involves embedded blank.
                            (3) variable length is is also an issue     */
   DATALINES;
City  Yr2000Popn
New York  8,008,278
Los Angeles  3,694,820
Chicago  2,896,016
Houston  1,953,631
Philadelphia  1,517,550
Phoenix  1,321,045
San Antonio  1,144,646
San Diego  1,223,400
Dallas  1,188,580
San Jose  894,943
;
RUN;

PROC PRINT data = citypops;
   title 'The citypops data set';
RUN;;


/** The issues in the above code were fixed in the following code **/
DATA citypops;
      INFILE DATALINES FIRSTOBS = 2;
      LENGTH city $ 12;
      INPUT city & pop2000;
      DATALINES;
City  Yr2000Popn
New York  8008278
Los Angeles  3694820
Chicago  2896016
Houston  1953631
Philadelphia  1517550
Phoenix  1321045
San Antonio  1144646
San Diego  1223400
```

```
Dallas  1188580
San Jose  894943
;
RUN;

PROC PRINT data = citypops;
      TITLE 'The citypops data set: with &';
      format pop2000 comma10.;
RUN;

/*****************************************************************************
  The colon (:) modifier allows us to use list input to read nonstandard data
  values and character values that are longer than eight characters, but which
  contain no embedded blanks. The colon (:) indicates that values are read
  until a blank (or other delimiters) is encountered, and then an informat is
  applied. If an informat for reading character values is specified, the w value
  specifies the variable's length, overriding the default length of 8.
*****************************************************************************/

DATA citypops_colon;
      INFILE DATALINES FIRSTOBS = 2;
      INPUT city & $12. pop2000 : comma.; /*  & --> will take care of one or ore SINGLE blanks
                                                  in the first character variable.
                                             $12. -->  take cares if variable type and LENGTH,
                                                  caution: (.) is required!
                                             : --> tells SAS to read until a blank (or other
                                                  delimiter) is encountered, and then an
                                                  informat is applied. Here, comma. is the informat.

                                                  Caution: Since in this example, the INFORMAT is
                                                  comma, so the delimiter should not be a comma!
                                          */
DATALINES;
City  Yr2000Popn
New York  8,008,278
Los A ngeles  3,694,820
Chicago  2,896,016
Houston  1,953,631
Philadelphia  1,517,550
Phoenix  1,321,045
San Antonio  1,144,646
```

```
San Diego  1,223,400
Dallas  1,188,580
San Jose  894,943
;
RUN;

PROC PRINT DATA = citypops;
      TITLE 'The citypops data set: Both (&) and (:)';
      FORMAT pop2000 comma10.;
RUN;

/*NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
 Topic 2: Read externals data with common formats: Excel, CSV, Text,
          and data from other applications such as SPSS using IMPORT WIZARD
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN*/
/*----------------------------------------------------------------------------
1. Please download the famous iris data in three formats: xlsx, CSV, and txt
   and save in a folder in a local drive that your sas can access.
2. please also download SPSS data with extension .sav and save it to the same
   folder.

Once you save the four data sets in a folder that SAS can access, we demonstrate
the steps for loading these data to SAS create SAS data sets.

SAS INPUT WIZARD will also generate SAS code for you so you can use it for
other applications.

----------------------------------------------------------------------------*/




/*NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
 Topic 3: Read externals data with common formats: Excel, CSV, Text,
          and data from other applications such as SPSS using PROC IMPORT
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN*/

PROC IMPORT OUT= WORK.Iris_TEMPLATE02    /* SAS file in the temporary library  */
            DATAFILE= "C:\STA311\w04\w04-iris-xlsx.xlsx"  /* source xlsx file  */
            DBMS=EXCEL REPLACE;    /*File type, replace if already exists*/
            RANGE="'w04-iris$'";   /*Indicates name of the sheet within Excel workbook
```

```
                                        Note that sheet names can only be 31 characters long.  */

            GETNAMES=YES; /*Indicates the first row contains variable names. The default setting
                           and SAS will automatically use the first row of data as variable names.
                           If the first row of your sheet does not contain variable names use the
getnames=no.*/
            MIXED=YES; /*Indicates both numeric and character variables in the data set. SAS uses the
                         first eight rows of data to determine whether the variable should be read as
                         character or numeric.  The default setting mixed=no assumes that each variable
                         is either all character or all numeric. If you have a variable with both
                         character and numeric values or a variable with missing values use mixed=yes
                         statement to be sure SAS will read it correctly.                              */
            SCANTEXT=YES; /*Tells SAS to scan column to determine length of text*/
            USEDATE=YES; /*Tells SAS to assign DATE format to date data*/
            SCANTIME=YES; /* Tells SAS to assign TIME format to time data */
RUN;

PROC PRINT;
RUN;

PROC CONTENTS DATA = Iris_TEMPLATE02;
RUN;



/**    Make sure SAS can find the CSV data file    **/
PROC IMPORT DATAFILE = "C:\STA311\w04\w04-iris-csv.csv"
    OUT = IRISCSV       /* name of the SAS data set to send to the tempoary library  */
    DBMS = CSV          /* Database management system: CSV */
      GETNAMES = YES;   /* first row is the not a record. */
 ;
RUN;

PROC IMPORT DATAFILE = "C:\STA311\w04\w04-iris-text.txt"
    OUT = TEXTIRIS         /*  */
    DBMS = CSV  REPLACE;
    GETNAMES = YES;
RUN;
```

```
PROC IMPORT OUT= WORK.demographics
            DATAFILE= "C:\STA311\w04\w04-Demographics.sav"
            DBMS=SPSS REPLACE;
RUN;



/*NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
 Topic 4:  Export SAS data set to a CSV format data
           Why CSV?
           The CSV (short for "comma-separated values") is one of the most
           flexible formats. CSV is a bit faster, smaller in size, very easy
           to handle (even in Excel) and many existing applications understand
           it, it is a widely used standard. It is still the first choice in
           many situations.

          This is why I only use an example to demonstrate this programmatic
          approach to export CSV data!
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN*/

PROC EXPORT DATA= IRIS_template   /* Export the SAS file from the permanebt lbrary and
                                    save it as a CSV file in the next folder. */
            OUTFILE= "C:\STA311\w04\outcsv_mycsv.csv"
            DBMS=CSV REPLACE;  /* REPLACE will overwrite the existing csv file the same name. */
     PUTNAMES=YES;             /* carry names of the variables in the SAS data file */
RUN;

/**    Why do we need PROC IMPORT/EXPORT while we have INPUT WIZARD?    **/

/*NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
 Topic 5:  Loading SAS Data set to SAS for DATA MANAGEMENT
           We only focus on the syntax and some basic controls this time.
           We will use this SET statement frequently beyond this point.

           Basic Syntax:

           DATA newdata;
           SET oldSASdataset;
             more statements to modify the oldSASdataset;
           RUN;
```

```
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN*/

    /****  A simple example: Citypops.sas7bdat is in the temporary library ***/

  PROC PRINT DATA = Citypops;
  TITLE "Old City Population Data Set";
  RUN;

  DATA NewCityPops;
  SET CityPops;
      New_popn = ROUND(pop2000 * 1.0001, 1); /* add a new variable reflecting
                                                 the new population size.
                                                */
  RUN;

  /**   ROUND() is a SAS built-in function: the following examples illustrate
                how to use it:
                       round(1234.56789, 100)-----> 1200;
                       round(1234.56789, 10) -----> 1230;
                       round(1234.56789, 1)  -----> 1235;
                       round(1234.56789, .1) -----> 1234.6;
                       round(1234.56789, .01)-----> 1234.57;
                       round(1234.56789, .001)----> 1234.568;
                       round(1234.56789, .0001)---> 1234.5679;
                       round(1234.56789, .00001)--> 1234.56789;
                       round(1234.56789, .1111)---> 1234.5432;        **/

PROC PRINT DATA = NewCityPops;
TITLE "New City Population Data";
RUN;

TITLE " ";  /* clear the title for next output  */
```