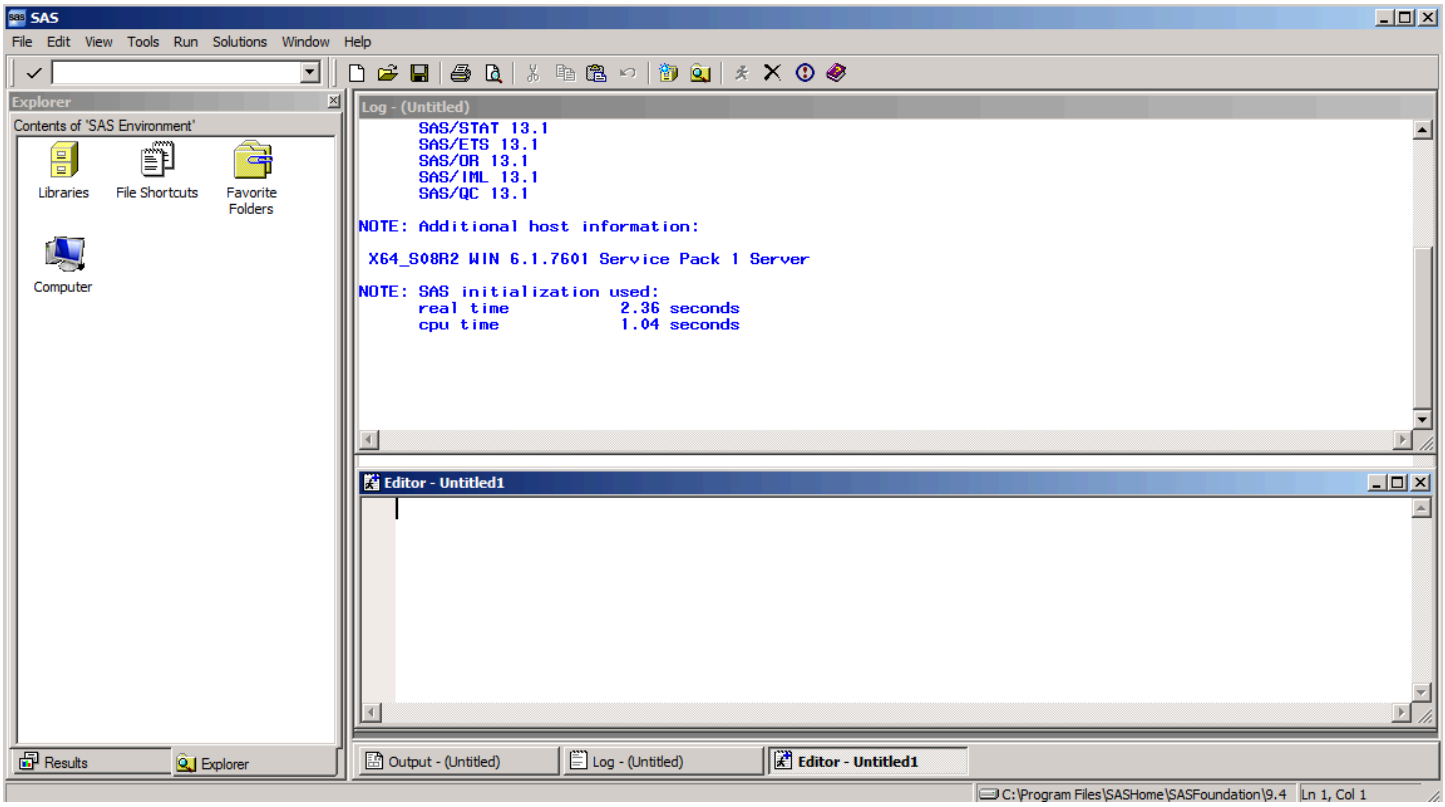
Introduction to SAS

Table of Contents

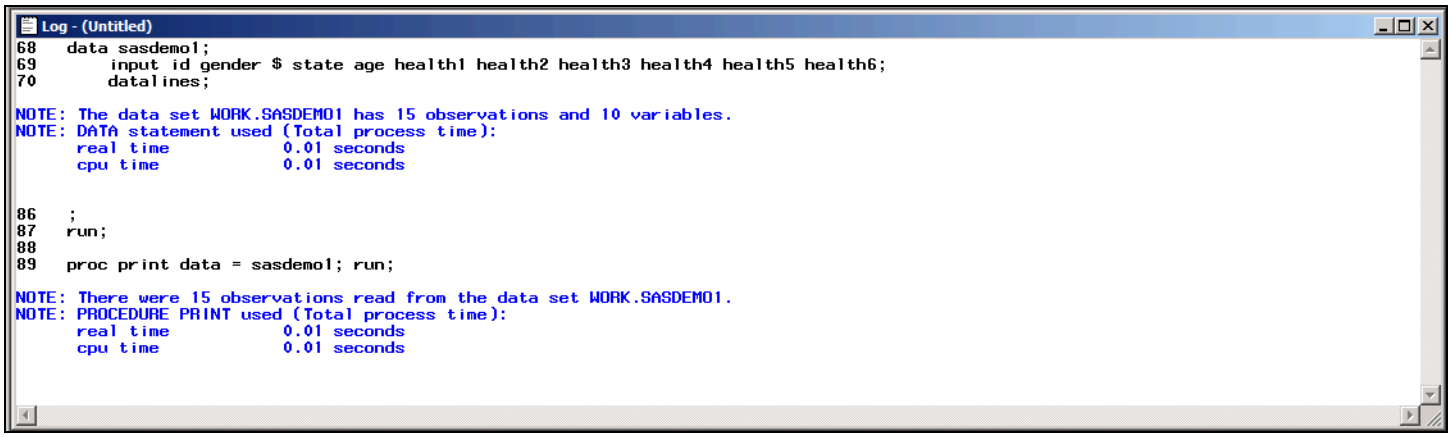
SAS Environment

SAS Environment

- When you open up Base SAS, you see three different boxes
- The Log is where the code is processed and SAS describes any errors in your code
- The Editor is where you write and save SAS Syntax
- The Explorer box shows shortcuts to SAS Libraries, File Shortcuts, Favorite Folders and Computer
- The Explorer also toggles to the Results window which acts as a table of contents when you start creating output



Log



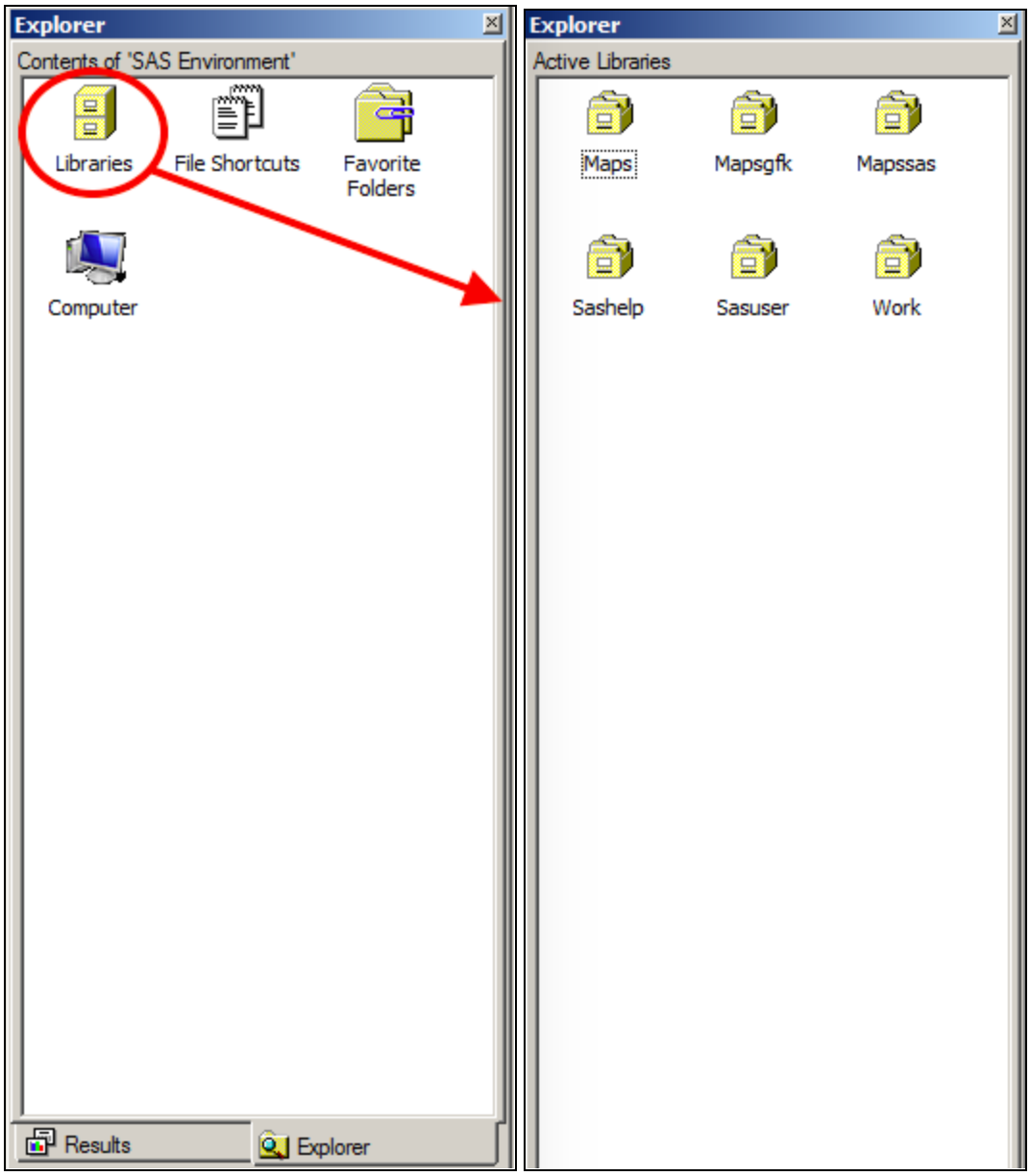
Editor

Syntax *

```
/***************
 Entering Data
****************/
data sasdemo1;
    input id gender $ state age health1 health2 health3 health4 health5 health6;
    datalines;
1   M   1   51  1   4   2   1   4   5
2   F   3   -1  2   3   3   2   3   4
3   F   1   29  5   2   4   2   1   3
4   M   1   21  5   1   5   4   2   1
5   M   2   56  2   4   2   4   3   3
6   M   3   72  1   5   4   2   4   5
7   F   3   46  2   5   3   1   3   4
8   M   2   33  5   2   4   5   2   1
9   F   2   36  3   3   4   5   2   2
10  M   1   -1  3   3   3   4   2   4
11  F   2   41  2   4   3   3   3   3
12  F   1   57  1   4   2   1   5   5
13  M   2   30  3   2   3   4   1   3
14  F   1   48  5   3   3   4   2   2
15  M   3   32  4   2   4   3   2   2
;
```

Libraries

- A SAS Library is a collection of SAS recognized files such as datasets or formats
- There are six default libraries when opening SAS: Maps, Mapsgfk, Mapsas, Sashelp, Sasuser and Work
- The Work library is the default library and is also temporary, meaning that the files will be cleared once the SAS session is terminated
- To create your own library use the `libname` statement
    - `libname desk "C:\Users\NYU User\Desktop\";`
- If you are using NYU's Virtual Computing Lab (VCL), your path will be similar to the one below, where `NetID` is your NetID
    - `libname desk "\\apporto.com\dfs\Users\NetID\Desktop\";`
- Now when you want to specify a file in a specific library, you write the following: `libraryname.datasetname`

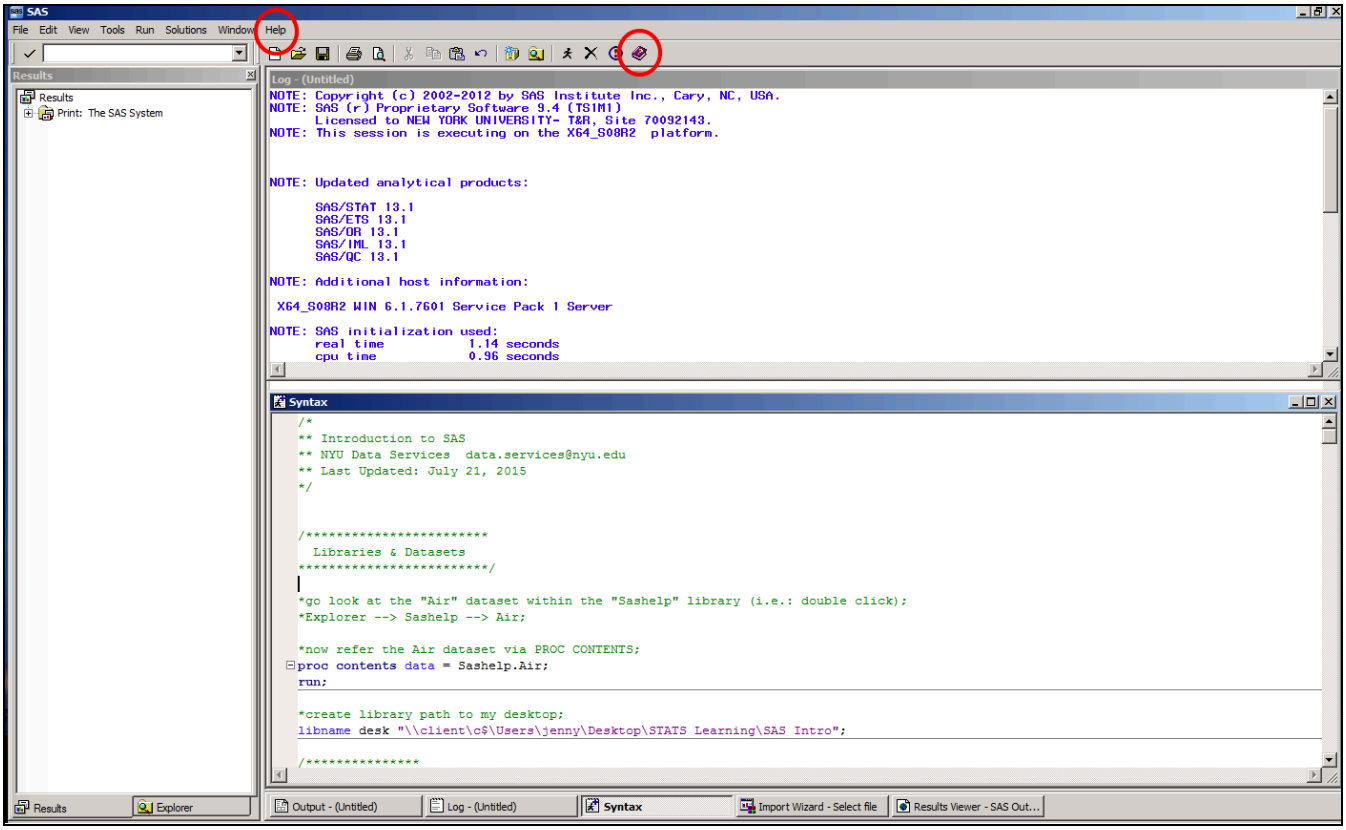


SAS Syntax Basics

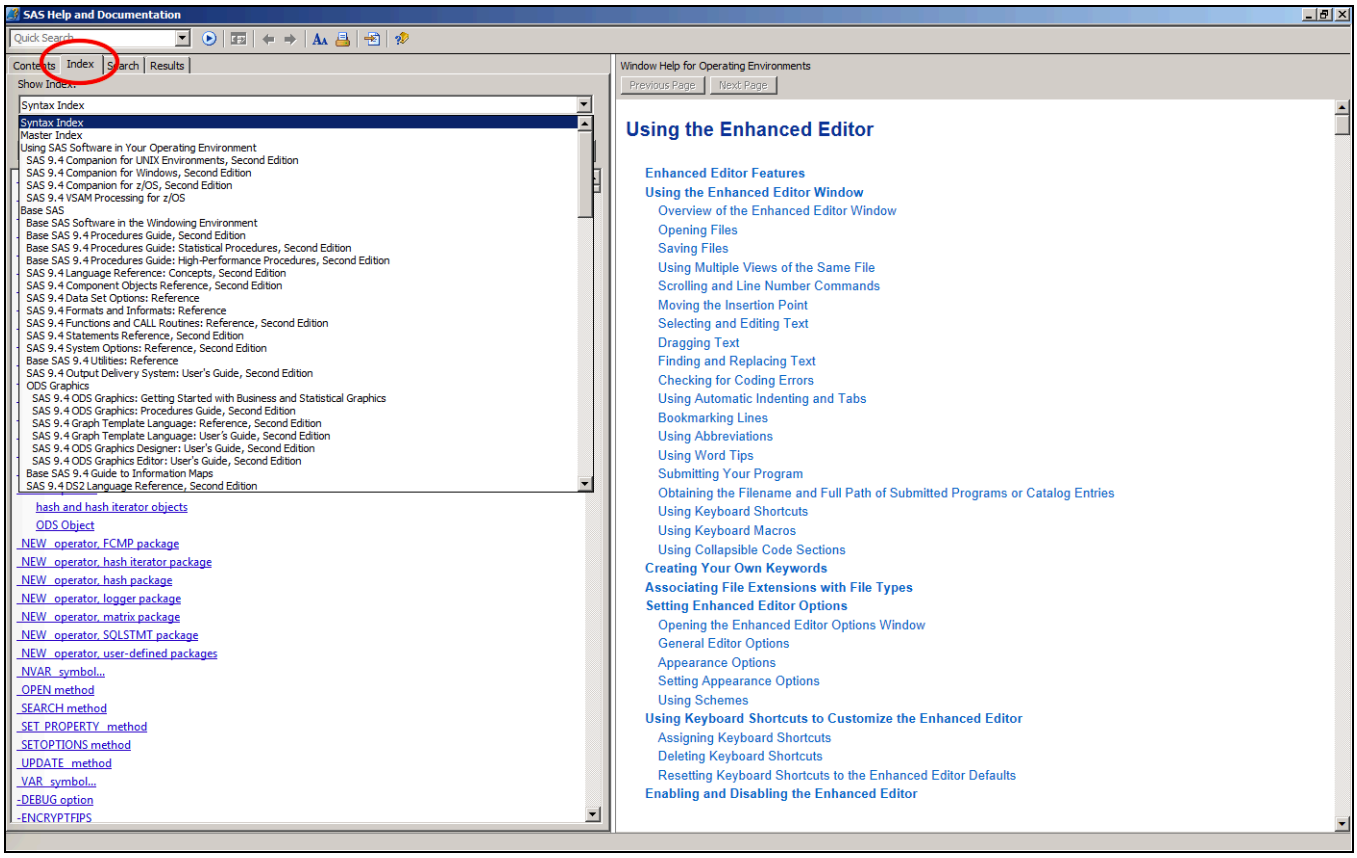
- There are two main groups of SAS statements
  - `data` steps
    - start with the word **`data`**
    - usually deal with managing data (read in data, create variables, drop variables, adding labels, merging datasets
    - data steps produce data sets
  - `proc` steps
    - short for procedure
    - starts with **`proc`** and the second word reveals the type of procedure (ex. means, ttest, regression, import, cluster etc…)
    - SAS proc steps generally create output/results
  - `proc` steps and `data` steps each have several statements inside, forming paragraphs
  - Each statement then has a list of options which can be added to customize output
- Each SAS statement ends with a semicolon **`;`**
- Each **`proc`**/**`data`** step ends with the word `run;` or `quit;`
- To run the SAS code, select the lines to run and press F8 or use the Run button 
- SAS code is case insensitive
- Start a line with an `*` and end with a `;` to write a comment

How to Get Help in SAS

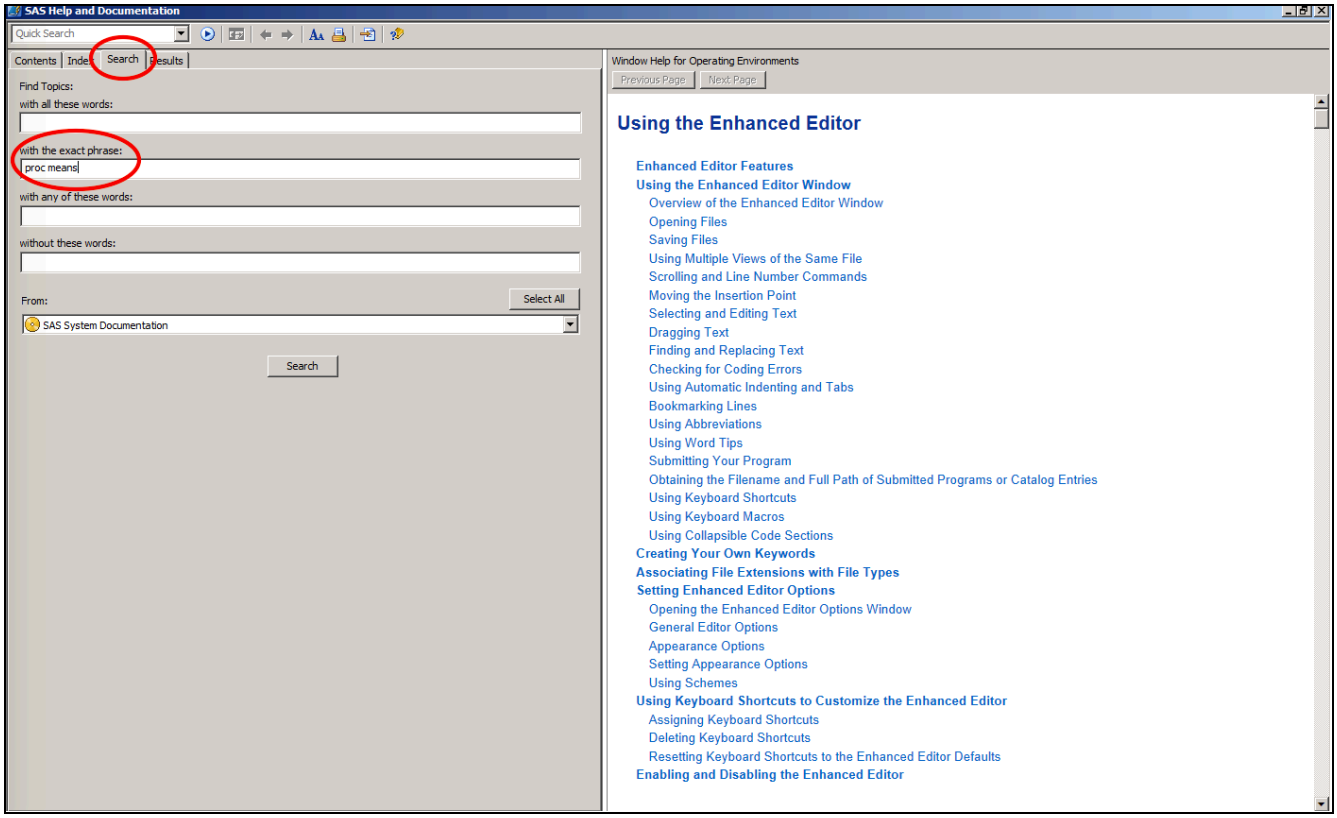
Step 1. Go to the *Help* tab, or click on the *Help Button* in the toolbar



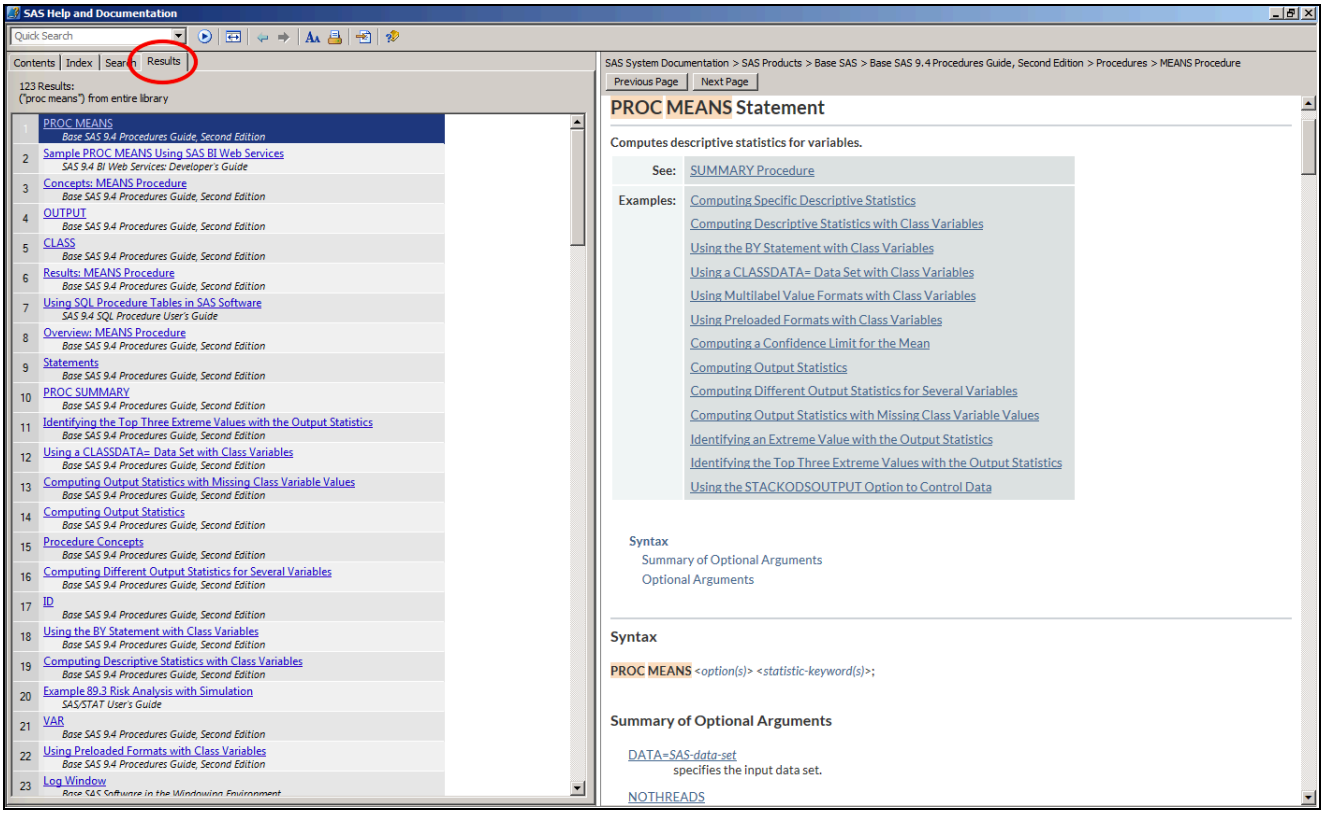
Step 2.1. As the Help window pops up, you can search for the topic of your interest by *Index*



Step 2.2. ...Or go to the *Search* tab and enter the keywords (e.g., PROC MEANS). Click on the Search button at bottom to proceed to search results.



Step 3. You will be led to the search results that match your keywords. Click on any of the search results, and the window on the right will show the details accordingly.



Importing Data

Manual Data Entry

- Use the `input` statement to define variables and use informats to help SAS read in those variables
- Use the `datalines` statement to enter rows of data
-
```
data desk.sasdemo1;
    input id gender $ state age health1 health2 health3 health4 health5
    health6;
    datalines;
    1  M  1  51  1  4  2  1  4  5
    2  F  3  -1  2  3  3  2  3  4
    3  F  1  29  5  2  4  2  1  3
    4  M  1  21  5  1  5  4  2  1
    5  M  2  56  2  4  2  4  3  3
    6  M  3  72  1  5  4  2  4  5
    7  F  3  46  2  5  3  1  3  4
    8  M  2  33  5  2  4  5  2  1
    9  F  2  36  3  3  4  5  2  2
    10 M  1  -1  3  3  3  4  2  4
    11 F  2  41  2  4  3  3  3  3
    12 F  1  57  1  4  2  1  5  5
    13 M  2  30  3  2  3  4  1  3
    14 F  1  48  5  3  3  4  2  2
    15 M  3  32  4  2  4  3  2  2
    ;
run;
```
- In this examples, the `$` is used after the variable gender in the input statement, to tell SAS that gender is a character variable

Using the Data Step

- Instead of using `datalines` to manually enter data, we can enter data from a file using the `infile` statement in the **`data`** step
-
```
data desk.sasdemo2;
    input id gender $ state age health1 health2 health3 health4 health5
    health6;
    infile "C:\Users\NYU User\Desktop\Dataset.csv" delimiter = ","
    firstobs = 2;
run;
```

- In the infile statement, we will use the `delimiter = ","` option to specify that the file is comma delimited and the `firstobs = 2` option to tell SAS that the first row of data appears on the second row of the file

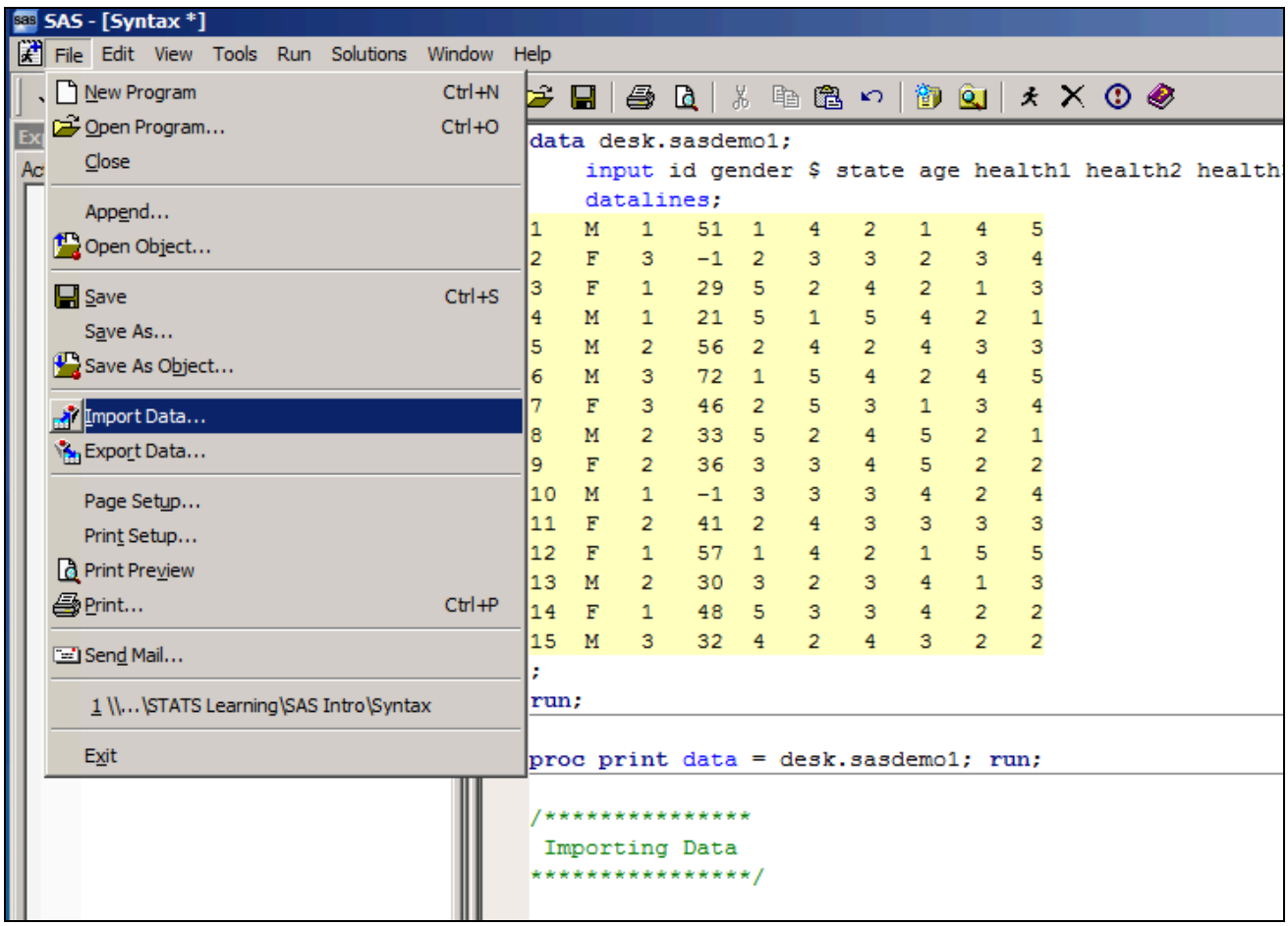
Proc Import

- Using **proc import** can make reading in data even easier
-
```
proc import datafile = "C:\Users\NYU User\Desktop\Dataset.csv"
   out = desk.healthSurvey dbms = csv;
      getnames = yes;
      datarow = 2;
run;
```

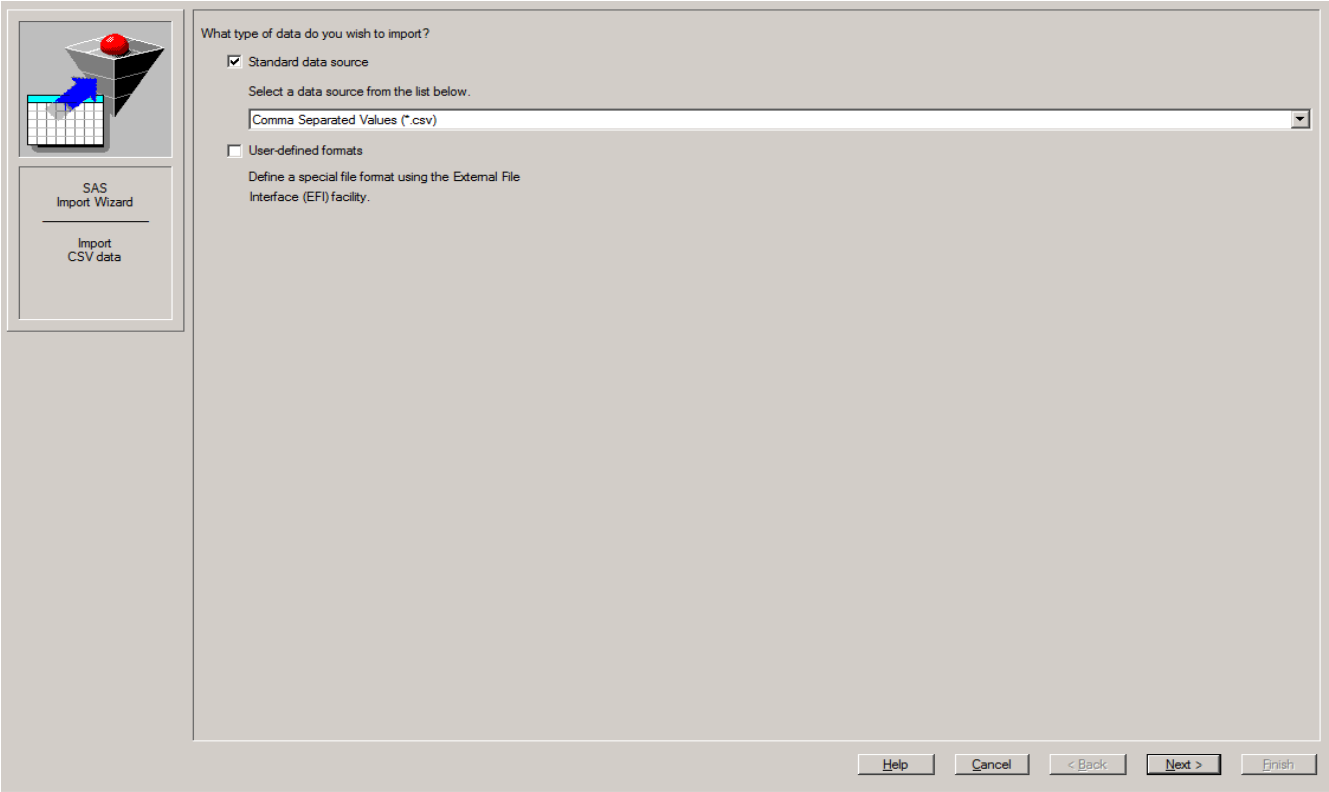
- The `datafile` option in **proc import** specifies the location of the file
- The `out = desk.healthSurvey` option tells SAS to save the imported file in the `desk` library as `healthSurvey`
- The `dbms` option tells SAS the datasource type, in this case the file is a csv
- The `getnames = yes` statement specifies that the first row of the file is the variable names
- The `datarow = 2` option tells SAS that the first row of data appears on the second row of the file
- More information about **proc import**

Import Wizard

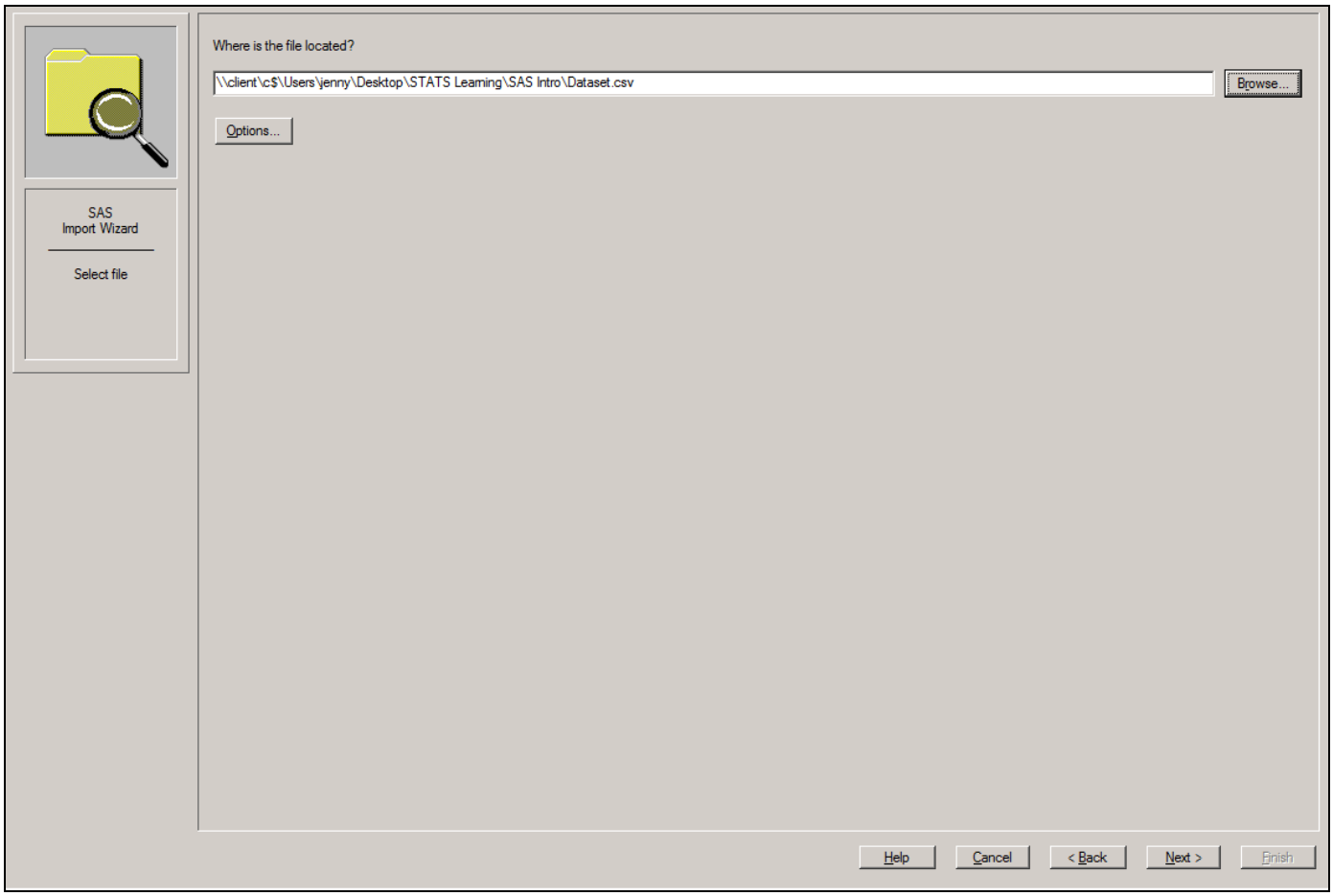
Step 1. Go to File → Import Data.



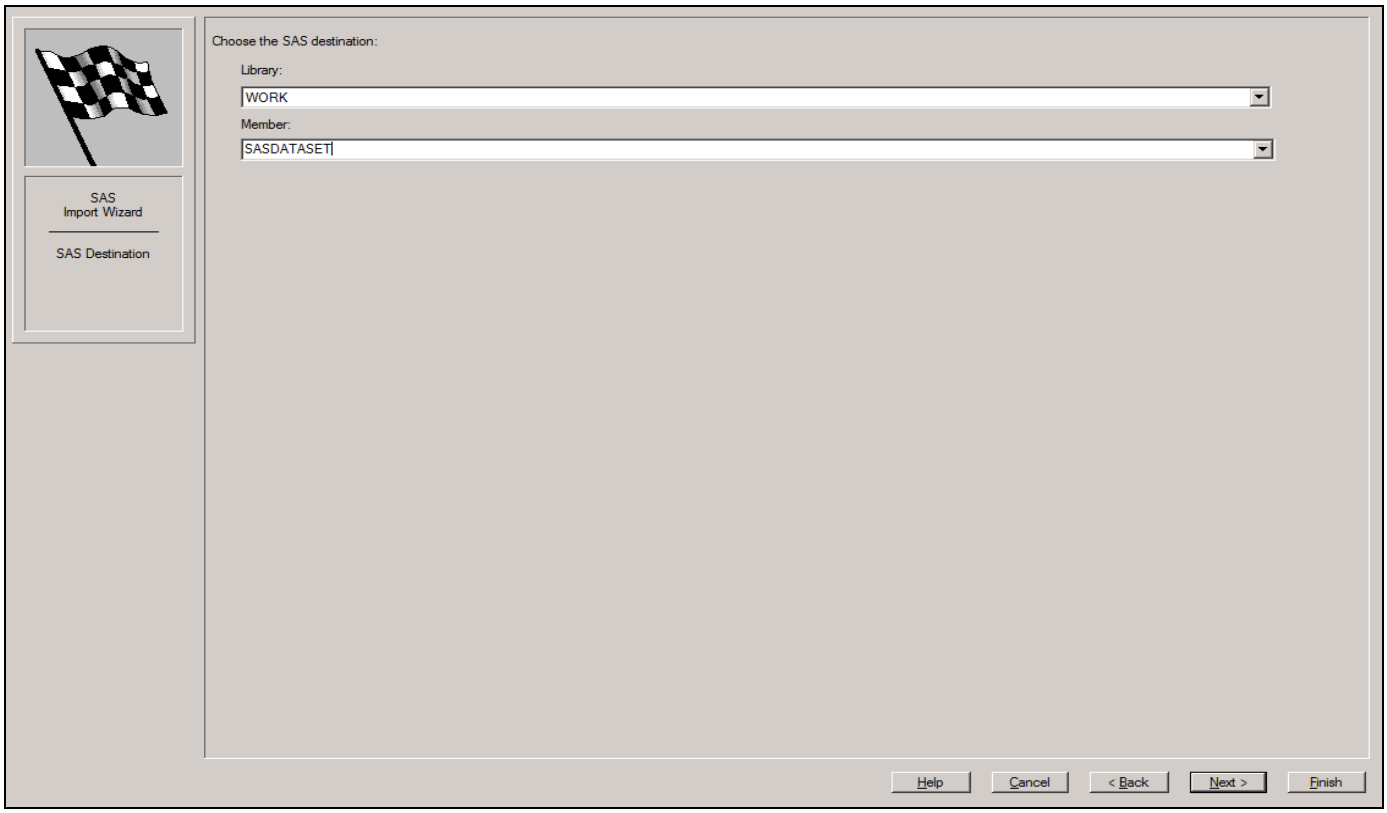
Step 2. Select the type of data you would like to import (e.g., csv, xls), and click on *Next*.



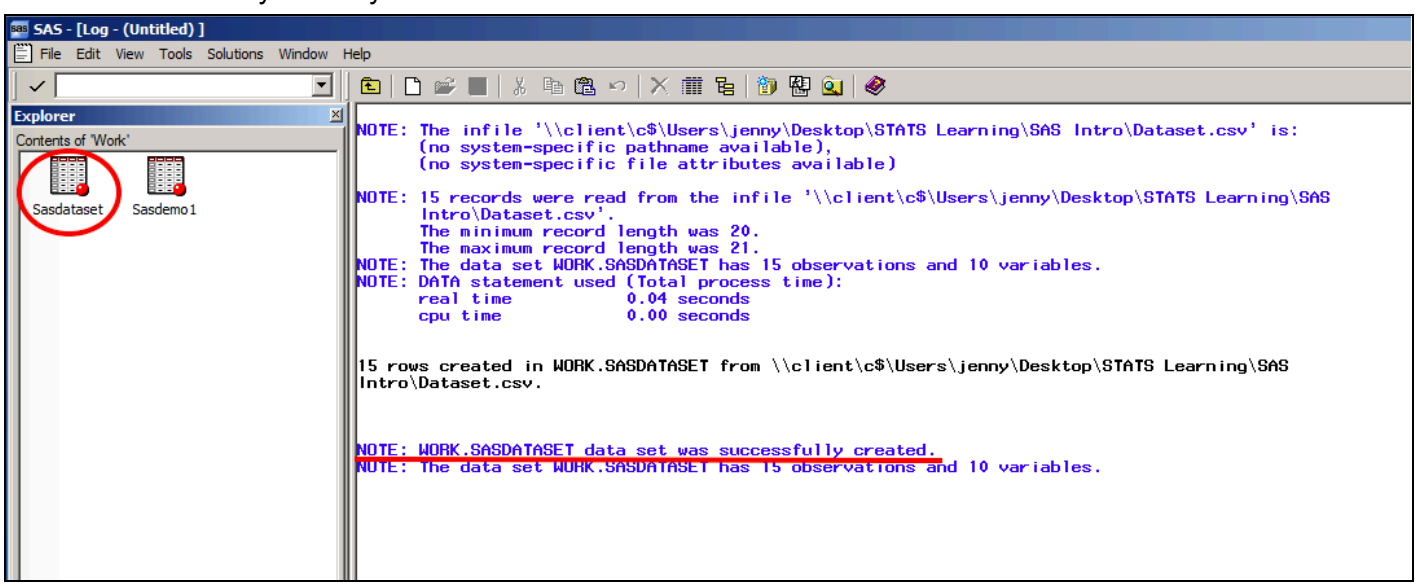
Step 3. Locate the file either by browsing or by directly typing the path to the file. Click on *Next* to proceed.



Step 4. Select the *Library* where you'd like to store the file. Name your file under *Member*, and click on *Finish*.



Step 5. Now the file has been successfully imported to SAS. You can see the message in Log and find the dataset in the Library where you stored the dataset.



Displaying Dataset Information

proc contents

- `proc contents` will report information about the dataset to the output window, including variable names, formats, labels and other useful information.
-
```
proc contents data = desk.healthSurvey;
run;
```
- For more information about **proc contents**

proc print

- A `proc print` will print the dataset to the output window
-
```
proc print data = desk.healthSurvey;
run;
```

- Like all SAS procs, the results of `proc print` can be completely customized. For example a list of variables can be given to the `var` statement to only print those variables. The `noobs` option can be used in the proc print statement to specify to not print the observation number.
-
```
proc print data = desk.healthSurvey noobs;
     var age health1 health2;
run;
```

- For more information about **proc print**

Variable Management

Create New Variables

- To create a new variable in the dataset, place the new variable name on the left of the equal sign and the formula on the right hand side
-
```
data desk.healthSurvey;
     set desk.healthSurvey;
     healthSum1 = (health1+health2+health3+health4+health5+health6);
     healthSum2 = sum(health1, health2, health3, health4, health5,
     health6);
     healthSum3 = sum(of health1-health6);
run;
```

Keep/Drop Variables

- Specify the variables you would like to keep after the `keep` statement in the `data` step
-
```
data test;
     set desk.healthSurvey;
     keep id gender age healthSum1;
run;
```

- Specify the variables you would like to drop after the `drop` statement in the `data` step
-
```
data test;
     set desk.healthSurvey;
     drop health1 health2 health3 health4 health5 health6;
run;
```

Recoding Variables

- `if`/`then` statements allow values in a variable to be changed based on a condition
- Use `if`/`then` statements to replace all “-1” values within age to be missing
-
```
data desk.healthSurvey;
      set desk.healthSurvey;
      if age = -1 then age = .;
run;
```

- The `if`/`then` statement can also be written as the following:
```
if age = -1 then age = missing;
```

- `if`/`then` statements can also be used to create a new variable based off of values of an existing variable
-
```
data desk.healthSurvey;
      set desk.healthSurvey;
      agegroup = .;
      if (0 < age <= 30) then agegroup = 1;
      else if (30 < age <= 50) then agegroup = 2;
      else if (age > 50) then agegroup = 3;
run;
```

- One way to change the character gender variable into a new numeric dummy, is to use `if`/`then`and `else if`/`then` statements. After the 1 and 0 values are defined, the character variable gender is dropped and the new numeric variable for gender is renamed ‘gender.’
-
```
data desk.healthSurvey;
      set desk.healthSurvey;
      temp = .;
      if gender = "M" then temp = 1;
      else if gender = "F" then temp = 0;
      drop gender;
      rename temp = gender;
run;
```

- Reverse coding

```
*reverse code health2, health5 and health6;
data desk.sasdemo_temp;
    set desk.sasdemo;
    *health2;
    if health2 = 1 then health2 = 5;
    else if health2 = 2 then health2 = 4;
    else if health2 = 3 then health2 = 3;
    else if  health2 = 4 then health2 = 2;
    else if health2 = 5 then health2 = 1;
    *health5;
    if health5 = 1 then health5 = 5;
    else if health5 = 2 then health5 = 4;
    else if health5 = 3 then health5 = 3;
    else if health5 = 4 then health5 = 2;
    else if health5 = 5 then health5 = 1;
    *health6;
    if health6 = 1 then health6 = 5;
    else if health6 = 2 then health6 = 4;
    else if health6 = 3 then health6 = 3;
    else if health6 = 4 then health6 = 2;
    else if health6 = 5 then health6 = 1;
run;

proc print data = desk.sasdemo_temp;
run;
```

```
*same as above but use an array this time;
data desk.sasdemo_temp2;
    set desk.sasdemo;
    array a[3] health2 health5 health6;
    do i = 1 to 3;
        if a[i] = 1 then a[i] = 5;
        else if  a[i] = 2 then a[i] = 4;
        else if a[i] = 3 then a[i] = 3;
        else if a[i] = 4 then a[i] = 2;
        else if a[i] = 5 then a[i] = 1;
    end;
    drop i; *running a do loop, creates a temporary variable i;
run;
```

```
*same as above but use a formula this time;
data desk.sasdemo;
    set desk.sasdemo;
    array a[3] health2 health5 health6;
    do i = 1 to 3;
        a[i] = 6 - a[i];
    end;
    drop i;
run;

proc print data = desk.sasdemo;
run;
```

Formats

- A format changes the way data is displayed
- The first step to using a format is to create a format using **proc format**
- Many formats can be defined in a single **proc format** with several value statements
-
```
proc format;
      value agegroupF
            1 = "30 and younger"
            2 = "31-50"
            3 = "51 and older";
run;
```
- The formats can either be used in a particular proc which will only change the data for that proc, or the formats can be used in a data step, which will allow the data to always have the formats attached.
-
```
proc print data = desk.healthSurvey;
      format agegroup agegroupF.;
run;
```
-
```
data desk.healthSurvey;
      format agegroup agegroupF.;
run;
```
- For more information about **proc format**

Informats

- **SAS Formats and Informats**
- **SAS Informats by Category**

Descriptive Statistics

- **proc means** will create a table of simple descriptive statistics for one or more variables
- Specify the variable(s) in the var statement
-
```
proc means data = desk.healthSurvey;
      var HealthSum1 age;
run;
```

- Use a where statement to run descriptive statistics on a subset of the data
-
```
proc means data = desk.healthSurvey;
      var HealthSum1;
      where age > 45;
run;
```

- Use a by statement to split the output into groups of a categorical variable. However, every time a by statement is used, the dataset must be sorted using **proc sort**.
-
```
proc sort data = desk.healthSurvey;
      by gender;
run;

proc means data = desk.healthSurvey;
      var HealthSum1;
      by gender;
run;
```

- **proc univariate** will provide a broader range of descriptive statistics
- This procedure will only take one variable in the var statement
-
```
proc univariate data = desk.healthSurvey;
      var HealthSum1;
run;
```

- For more information about **proc means** or **proc univariate**

Frequency Tables

- Use **proc freq** to create frequency tables and cross tabulations
- To create a simple frequency table, specify the categorical variable after the tables statement
-
```
proc freq data = desk.healthSurvey;
      tables gender;
run;
```

- To create a cross tabulation, specify two categorical variables after the tables statement, separated by an asterisk
-
```
proc freq data = desk.healthSurvey;
      tables gender*state;
run;
```

- The table can be cleaned up by adding options in the `tables` statement. The `norow` option will remove column percents, the `nocol` option will remove column percentages and the `nopct` option will remove total percentages.
- ```
  proc freq data = desk.healthSurvey;
        tables gender*state / norow nocol nopct;
  run;
  ```

- For more information about **proc freq**

Graphs

Boxplot

- **proc boxplot** can be used to create a boxplot
- Using the `plot` statement with the dependent variable `*` grouping variable
- ```
  proc boxplot data = desk.healthSurvey;
        plot HealthSum1*gender;
  run;
  ```
- For more information about **proc boxplot**

Scatter plot

- There are several ways to create a simple scatter plot in SAS. One way is to use the `plot` statement in **proc gplot**. After the `plot` statement comes the dependent variable `*` independent variable.
- ```
  proc gplot data = desk.healthSurvey;
        plot HealthSum1*age;
  run;
  ```
- For more information about **proc gplot**

- **proc sgplot** can also be used to create scatter plots. Using the `scatter` statement in **proc sgplot** allows the user to define the dependent and independent variables. A `group` option can be used to color code the points based on a categorical variable.
- ```
  proc sgplot data = desk.healthSurvey;
        scatter y = HealthSum1 x = age / group = gender;
  run;
  ```

- Alternatively, by using the `reg` statement, a regression line will be added to the scatter plot.
- ```
  proc sgplot data = desk.healthSurvey;
        reg y = HealthSum1 x = age / group = gender;
  run;
  ```
- For more information about **proc sgplot**

- **proc sgscatter** can also be used to create a scatter plot. The notation is similar to **proc sgplot** but the `compare` statement is used.
- ```
  proc sgscatter data = desk.healthSurvey;
        compare y = HealthSum1 x = age  / group = gender;
  run;
  ```
- For more information about **proc sgscatter**

Basic Analysis

T-Test

- To run a t-test, use **`proc ttest`**
- The variable in the `var` statement is the dependent variable and the variable in the `class` statement is the grouping variable. In this case the results for a two sample t-test will be produced.
-
```
proc ttest data = desk.healthSurvey;
      class gender;
      var HealthSum1;
run;
```
- For more information about **`proc ttest`**

Chi-Square Test

- To run a chi-square test, first start with **`proc freq`**
- After defining the specified cross tab in the `tables` statement, use a `chisq` option to produce a chi-square test. The `exact` option produces a Fisher's exact test.
-
```
proc freq data = desk.healthSurvey;
      tables gender*state / chisq exact;
run;
```
- For more information about **`proc freq`**

Regression

- To run a linear regression, use **`proc reg`**
- Use the `model` statement to define the model, where the dependent variable appears to the right of the equal sign and the independent variables appear on the left of the equal sign separated by spaces
-
```
proc reg data = desk.healthSurvey;
      model HealthSum1 = age gender;
run;
```
- For more information about **`proc reg`**