Introduction to Data

Some define *Statistics* as the field that focuses on turning information into knowledge. The first step in that process is to summarize and describe the raw information—the data. In this lab, you will gain insight into public health by generating simple graphical and numerical summaries of a data set collected by the Centers for Disease Control and Prevention (CDC). As this is a large data set, along the way you'll also learn the indispensable skills of data processing and subsetting.[1]

Getting Started

The Behavioral Risk Factor Surveillance System (BRFSS) is an annual telephone survey of 350,000 people in the United States. As its name implies, the BRFSS is designed to identify risk factors in the adult population and report emerging health trends. For example, respondents are asked about their diet and weekly physical activity, their HIV/AIDS status, possible tobacco use, and even their level of health care coverage. The BRFSS website (**http://www.cdc.gov/ brfss**) contains a complete description of the survey, including the research questions that motivate the study and many interesting results derived from the data.

We will focus on a random sample of 20,000 people from the BRFSS survey conducted in 2000. There are over 200 variables in this data set, but we will work with a small subset.

- If you are using SAS University Edition, you need to ensure that interactive mode is turned off. To do this, click the button to the right of **Sign Out** in the upper right corner of the window and then click **Preferences**. In the Preferences window, on the General tab, the bottom check box (located next to the text **Start new programs in interactive mode**) should ***not*** be selected. If the box is selected, you need to clear it and save your change.

We begin by loading the data set of 20,000 observations into SAS. After launching SAS, submit the following code:

```
filename cdc url 'http://www.openintro.org/stat/data/cdc.csv';

proc import datafile=cdc
            out=work.cdc
            dbms=csv
            replace;
            getnames=yes;
run;
```

The data set **cdc** is created from a comma-separated file (.CSV) in the **work** library. PROC IMPORT creates a version of the data in a format that SAS DATA steps and procedures (PROCs) can use. If you view the data set, it will be represented by rows and columns. Rows represent *cases* or *observations* and columns represent *variables*.

To view the names of the variables other information about the data set, use this code:

```
proc contents data=work.cdc short;
run;
```

The last table in the output shows the list of variables, along with their positions in the data set. Each one of these variables corresponds to a question that was asked in the survey. For example, for **genhlth**, respondents were asked to evaluate their general health, responding either excellent, very good, good, fair, or poor. The **exerany** variable indicates whether the respondent exercised in the past month (1) or did not (0). Likewise, **hlthplan** indicates whether the respondent had some form of health coverage (1) or did not (0). The **smoke100** variable indicates whether the respondent had smoked at least 100 cigarettes in his or her lifetime. The other variables record the respondents' height in inches and weight in pounds, as well as their desired weight (**wtdesire**), age in years, and gender. The **VAR1** variable corresponds to a row number column in the .CSV file. It can be used as an ID variable.

Exercise 1: How many cases are there in this data set? How many variables? For each variable, identify its data type (for example, categorical, numeric).

We can have a look at the first few rows of our data with this code:

```
proc print data=work.cdc (obs=10);
run;
```

You could also look at ***all*** of the data set at once by excluding the OBS=10 option, but that might be unwise here. We know that **cdc** has 20,000 rows, so viewing the entire data set would mean flooding your screen. It's better to take small peeks at the data.

Summaries and Tables

The BRFSS questionnaire is a massive trove of information. A good first step in any analysis is to distill all of that information into a few summary statistics and graphics. As a simple example, PROC UNIVARIATE returns a numerical summary, including a wide variety of statistics for numeric variables. These statistics include, but are not limited to, mean, variance, standard deviation, minimum, maximum, and extreme observations. For **weight**, the code is as follows:

```
proc univariate data=work.cdc;
   var weight;
run;
```

The VAR statement can accept a list of variables, each of which will be analyzed separately.

It makes sense to describe a quantitative variable such as **weight** in terms of these statistics, but what about categorical data? We would instead consider the sample frequency distribution. There are many ways to do this in SAS. One way that can create frequency tables as well as plots is to use PROC FREQ (commonly pronounced like "FREAK"). The TABLES statement lists the categorical variables that will be analyzed. The PROC provides the count of the number of times each kind of response was given. For example, to see the number of people who have smoked 100 cigarettes in their lifetime, use this code:

```
proc freq data=work.cdc;
   tables weight;
run;
```

PROC FREQ provides both counts and relative frequencies. Relative frequencies are percents of the entire sample. The Percent column shows the relative frequencies of both categories of the **smoke100** variable. In addition, there are columns of cumulative frequencies in the PROC FREQ output. These values accumulate from top row to the bottom row. Therefore, the bottom row always reports the final tally of all nonmissing observations in the Cumulative Frequency column.

In order to graphically display the frequencies as a bar chart, we can add the option PLOTS=FREQPLOT to the TABLES statement after a slash /. The graph option SCALE=PERCENT requests that the y-axis represent the relative frequency, rather than the frequency count.

```
proc freq data=work.cdc;
   tables smoke100 / plots=freqplot(scale=percent);
run;
```

Exercise 2: Create a numerical summary for **height** and **age**, and compute the interquartile range for each. Compute the relative frequency distribution for **gender** and **exerany**. How many males are in the sample? What proportion of the sample reports being in excellent health?

PROC FREQ can also be used to create multi-way frequency tables. The code for such a table in the TABLES statement would list each variable in the table separated by an asterisk *. The last variable in such a table list would set the column variable and the second-to-last variable would list the row variable. If there are more than two variables for a table, the other variables would be for stratification. For example, to examine which participants have smoked across each gender, we could use the following code:

```
proc freq data=work.cdc;
   tables gender*smoke100;
run;
```

Here we see column labels of 0 and 1. Recall that 1 indicates that a respondent has smoked at least 100 cigarettes. The rows refer to **gender**. To create a mosaic plot of this table, we would add a PLOTS= option and resubmit the code:

```
proc freq data=work.cdc;
   tables gender*smoke100 / plots=mosaicplot;
run;
```

Exercise 3: What does the mosaic plot reveal about smoking habits and gender?

Interlude: How SAS Processes Data

We mentioned that SAS stores data in SAS data sets. In order to understand how SAS processes data, you need to know a little more about SAS data sets. SAS data sets have two portions: a descriptor portion, which contains the attribute information about the data in the SAS data set, and a data portion, which contains the data values in the form of a rectangular table that consists of observations (a different respondent) and variables (the columns; the first being **genhlth**, the second **exerany**, and so on).

We can see the first 10 values in the data portion by submitting this code:

```
proc print data=work.cdc (obs=10);
run;
```

We can see the descriptor portion of the information about the size of the data set, the number of observations and variables, as well as additional information such as the names, types, and lengths of the variables by submitting this code:

```
proc contents data=work.cdc;
run;
```

SAS variables are one of two types: character or numeric. Character variables can contain any letter, number, or symbol that can be typed on the keyboard. They vary from 1 to 32,767 bytes in length. Numeric variables can contain only numbers (0-9), a decimal point (.), a negative sign (-), and the letter E to indicate scientific notation. All SAS numeric are stored in 8 bytes by default. SAS date values are stored as the number of days since January 1, 1960, so you can manipulate date values, too.

SAS steps are sequences of one or more SAS statements. SAS has two kinds of steps: DATA and PROC. *Procedures* (PROC steps) are canned routines that read SAS data sets and generally produce a report as output. *DATA steps* read data to create SAS data sets as output and can modify existing variables or create new variables.

SAS statements usually start with an identifying keyword and always end with a semicolon. The DATA step starts with a DATA statement and ends with a RUN statement.

```
data work.newcdc;
   set work.cdc;
   <more SAS statements>
run;
```

The DATA step names the new SAS data set (**newcdc**), identifies the input data (**cdc**), processes the data, and writes the processed data to the SAS data set as a new observation.

SAS steps process the data in two phases: compile and execute. During compilation, the SAS statements are checked for syntax errors and the English-like program code is translated to machine code. When a RUN statement or the start of another DATA or PROC step (step boundary) is encountered, compilation stops, and the machine code is submitted for execution.

SAS processes data one step at a time. In this case, the DATA step processes **cdc** completely to create **newcdc**, and then reads **newcdc** to execute the PROC PRINT step.

```
data work.newcdc;
   set work.cdc;
   if gender="m" and age>30;
run;

proc print data=work.newcdc (obs=10);
run;
```

Here is a portion of the **work.newcdc** data set:

Obs	genhlth	exerany	hlthplan	smoke100	height	weight	wtdesire	age	gender
1	good	0	1	0	70	175	175	77	m
2	very good	1	1	0	71	194	185	31	m
3	very good	0	1	0	67	170	160	45	m
4	good	1	1	0	70	180	170	44	m
5	excellent	1	1	1	69	186	175	46	m
6	fair	1	1	1	69	168	148	62	m
7	excellent	1	1	1	70	170	170	69	m
8	good	1	1	1	73	185	175	79	m
9	good	0	0	1	67	156	150	47	m
10	fair	0	1	1	71	185	185	76	m

SAS DATA steps process data sequentially from top to bottom by default. The SET statement reads the first observation in **cdc** into memory. The subsequent statements in the DATA step are executed sequentially. And when the RUN statement at the bottom of the DATA step is executed, the current values of the variables in memory are written out as the first observation in the **newcdc** data set.

Control returns to the top of the DATA step. The second observation is read from **cdc** into memory, subsequent statements are executed, and the current values of the variables are written out as the second observation, and so forth. Execution continues until the last observation in **newcdc** is read and processed. Control then returns to the compiler, which processes the next step's statements. This compile and execute stepwise program flow continues until all SAS statements have been executed and the end of the program file is reached.

You can create new variables or change the values of existing variables by using an assignment statement in a DATA step:

```
wtkilos=weight*0.453592;
```

You name the new or existing variable (**wtdiff**), type an equal sign (**=**), and then type the expression you want evaluated to populate the variable (**weight-wtdesire**).

There are many SAS functions (mini-routines) to use in expressions that will calculate statistics:

```
wtdiff=sum(weight,-wtdesire);
```

Or manipulate character values:

```
genhlth=propcase(genhlth);
```

Or manipulate SAS date values:

```
now=today();
```

A partial listing of the resulting **cdc** data set has these values:

Obs	VAR1	genhlth	exerany	hlthplan	smoke100	height	weight	wtdesire	age	gender	wtkilos	wtdiff	now
1	1	Good	0	1	0	70	175	175	77	m	79.3786	0	19915
2	2	Good	0	1	1	64	125	115	33	f	56.6990	10	19915
3	3	Good	1	1	1	60	105	105	49	f	47.6272	0	19915
4	4	Good	1	1	0	66	132	124	42	f	59.8741	8	19915
5	5	Very Good	0	1	0	61	150	130	55	f	68.0388	20	19915

A Little More about Subsetting

Subsetting refers to extracting all the observations in a data set that have specific characteristics. We accomplish this through the SAS WHERE and subsetting IF statements. First, consider expressions like

```
where gender="m";
```

or

```
if age>30;
```

These statements produce either TRUE or FALSE values. There is one value for each observation, where TRUE indicates that the person was male (via the WHERE statement) or older than 30 (subsetting IF statement).

You can use subsetting in the DATA step and most PROC steps. For example, this SAS DATA step reads the **oldcdc** data set, subsets the incoming observations for just the men in the sample, and creates a new data set named **cdc**.

```
data work.newcdc;
   set work.cdc;
   where gender="m";
   <more SAS statements>
run;
```

This new data set name is specified in the DATA step. The source data set is specified in the SET statement. The new data set contains all the same variables but just under half the rows. It is also possible to tell SAS to keep only specific variables. For now, the important thing is that we can carve up the data based on values of one or more variables.

A word of caution with the WHERE statement: You can only have one WHERE statement per step. However, you can test several of conditions by stringing them together with AND and OR operators. The AND indicates that each condition on both sides of the AND must be true in order for the observation to be selected for the subset.

```
where gender="M" and age>30;
```

will give you the data for men over the age of 30.

The OR indicates that one or the other condition on the side of the OR must be true in order for the observation to be selected for the subset. It doesn't matter which one.

The statement

```
if gender="M" or age>30;
```

will include observations of people who are men or anyone over the age of 30 in the subset.

```
data work.newcdc;
   set work.cdc;
   if gender="m" and age>30;
run;
```

Running PROC PRINT on the first few observations shows that only males over the age of 30 are in the new data set, **newcdc**.

```
proc print data=work.newcdc (obs=10);
run;
```

Obs	VAR1	genhlth	exerany	hlthplan	smoke100	height	weight	wtdesire	age	gender
1	1	good	0	1	0	70	175	175	77	m
2	7	very good	1	1	0	71	194	185	31	m
3	8	very good	0	1	0	67	170	160	45	m
4	10	good	1	1	0	70	180	170	44	m
5	11	excellent	1	1	1	69	186	175	46	m
6	12	fair	1	1	1	69	168	148	62	m
7	14	excellent	1	1	1	70	170	170	69	m
8	16	good	1	1	1	73	185	175	79	m
9	17	good	0	0	1	67	156	150	47	m
10	18	fair	0	1	1	71	185	185	76	m

In principle, you can use as many AND and OR clauses as you want when forming a subset. ANDs are evaluated before ORs, but you can use parentheses to control the order of processing.

The statement

```
if gender="m" or (gender="f" and age>30);
```

will give you the data for all men and women over the age of 30.

Why are there two methods of subsetting data in SAS? It's partly due to the difference between the subsetting IF and WHERE statements. In order to use a WHERE statement, the variables you are testing must already exist in all the SAS data sets you are reading. In order to test a new variable created in the current step (such as the result of an assignment statement in the DATA step), you must use a subsetting IF statement.

Exercise 4: Create a new data set named **under23smoke** that contains all observations of respondents under the age of 23 that have smoked 100 cigarettes in their lifetime. Write the programming statements you used to create the new data set as the answer to this exercise.

Quantitative Data

We've already looked at categorical data such as **smoke100** and **gender**, so now let's turn our attention to quantitative data. Two common ways to visualize quantitative data are with box plots and histograms. There are several PROCs in SAS that can produce these plots, but let's use the PROC that we've already been using in this lesson: PROC UNIVARIATE. We can construct a panel of plots, including a box-and-whisker plot and a histogram, as well as a normal probability plot (which will be described in a later lesson) by adding one word to the PROC UNIVARIATE code that you have already seen.

```
ods graphics;
proc univariate data=work.cdc plot ;
   var weight;
run;
```

The PLOTS option is all that is needed.

- If ODS graphics are not turned on, the plots produced will be very different from those that are produced here.

You can analyze the locations of the components of the box by examining the quantiles table in the output.

Confirm that the median and upper and lower quartiles reported in the numerical summary match those in the graph. The purpose of a box plot is to provide a thumbnail sketch of a variable for the purpose of comparing across several categories. For example, we can, compare the heights of men and women using this code:

```
ods graphics;
proc univariate data=work.cdc plots;
   class gender;
   var Weight;
run;
```

The notation here is new. The CLASS statement has the effect of subsetting the output of the PROC without having to first subset the data in a DATA step. A similar subsetting could be performed using a BY statement, but that would require that the input data set first be sorted using PROC SORT.

```
proc sort data=work.cdc;
   by gender;
run;

ods graphics;
proc univariate data=work.cdc plots;
   by gender;
   var Weight;
run;
```

Run the previous code and see how the output differs from that obtained using the CLASS statement.

Next let's consider a new variable that doesn't show up directly in this data set: body mass index (BMI). BMI is a weight to height ratio and can be calculated as.

$$BMI = 703 \times weight(lb)/height(in)^2$$

The following DATA step first creates a new data set with a new variable named **bmi**, and then PROC SGPLOT produces the box-and-whisker plots this time. Separate plots are created for each value of the variable **genhlth**.

```
data work.cdcbmi;
   set work.cdc;
   bmi=(weight/height**2) * 703;
run;
```

Notice that the statement that creates the **bmi** variable above is just some arithmetic, but it's applied to all 20,000 numbers in the **cdcbmi** data set. That is, for each of the 20,000 participants, we take their weight, divide by their height-squared, and then multiply by 703. The result is a new data set with 20,000 **bmi** values, one for each respondent. This is one reason why we like SAS: it lets us perform computations like this using very simple expressions.

```
proc sgplot data=work.cdcbmi;
   vbox bmi / group=genhlth;
run;
```

The SGPLOT procedure is an alternative method to producing box-and-whisker plots. PROC SGPLOT can be used to create all kinds of statistical plots, including scatter plots, histograms, and dot plots. However, here we will stick to box-and-whisker plots. You have the option in PROC SGPLT to orient your box-and-whisker plots either vertically or horizontally. The HBOX statement requests horizontal plots.

In SAS, options in statements generally appear after a slash /. The GROUP= option in the VBOX statement requests that side-by-side box plots be created, subsetting by levels of the **genhlth** variable.

Exercise 5: What does this box plot show? Pick another categorical variable from the data set and see how it relates to BMI. List the variable you chose, why you might think it would have a relationship to BMI, and indicate what the figure seems to suggest.

Finally, let's make some histograms. We can look at the histogram for the age of our respondents with the HISTOGRAM statement in PROC UNIVARIATE. As mentioned, PROC SGPLOT can also be used to produce histograms, but PROC UNIVARIATE has a lot more options for dressing up the plot with statistical information. A generic request for a histogram in PROC UNIVARIATE looks like this:

```
proc univariate data=work.cdcbmi;
   var bmi;
   histogram bmi;
run;
```

Histograms are generally a very good way to see the shape of a single distribution, but that shape can change depending on how the data is split between the different bins. You can control the number of bins by adding an argument to the command. In the next program, we create a histogram of **bmi** with 50 bins. That is accomplished using the NMIDPOINTS= option in the HISTOGRAM statement.

```
proc univariate data=work.cdcbmi;
   var bmi;
   histogram bmi / nmidpoints=50;
run;
```

We can also request that some summary statistics are printed within the graph itself using an INSET statement, which must immediately follow the HISTOGRAM statement. The INSET requests the mean, standard deviation, and median, but many other statistics can be included using appropriate keywords. In addition, we can position the inset anywhere around the plot using a POSITION option in the INSET statement. You can choose geographical directional indicators.

```
ods graphics;
proc univariate data=work.cdc;
   var age;
   histogram age / nmidpoints=50;
   inset mean std median / position=NE;
run;
```

How do these histograms compare?

At this point, we've done a good first pass at analyzing the information in the BRFSS questionnaire. We've found an interesting association between smoking and gender, and we can say something about the relationship between people's assessment of their general health and their own BMI. We've also picked up essential computing tools (summary statistics, subsetting, and plots) that will serve us well throughout this course.