

# Chapter 2

## Introduction to SAS

SAS stands for “Statistical Analysis System.” Even though it runs on linux and Windows PCs as well as on bigger computers, it is truly the last of the great old mainframe statistical packages<sup>1</sup>. The first beta release was in 1971, and the SAS Institute, Inc. was spun off from the University of North Carolina in 1976, the year after Bill Gates dropped out of Harvard. This is a serious pedigree, and it has both advantages and disadvantages.

The advantages are that the number of statistical procedures SAS can do is truly staggering, and the most commonly used ones have been tested so many times by so many people that their correctness and numerical efficiency are beyond any question. For the purposes of this course, there are no bugs. The disadvantages of SAS are all related to the fact that it was *designed* to run in a batch-oriented mainframe environment. So, for example, the SAS Institute has tried hard to make SAS an “interactive” program, but as of January 2016, the interface is still basically file and text oriented, not graphical.

### 2.1 The Four Main File Types

A typical SAS job will involve four main types of file.

- **The Raw Data File:** A file consisting of rows and columns of numbers; or maybe some of the columns have letters (character data) instead of numbers. The rows represent observations and the columns represent variables, as described at the beginning of Section 1.1. In the first example we will consider below, the raw data file is a plain text file called `studentsleep.data.txt`.

In recent years it has become common for scientists to record their data using Microsoft Excel, so that real (not textbook) data sets will often be in Excel spread-

---

<sup>1</sup>This discussion refers to the core applications that are used to conduct traditional statistical analysis: Base SAS, SAS/STAT and SAS/ETS (Econometrics and Time Series). SAS also sells a variety of other software products. They are almost all tools for data extraction, processing and analysis, so they fall under the heading of Statistics broadly defined. However, the details are so shrouded in marketing and corporate IT jargon that you would need specialized (and expensive) training to understand what they do, and even then I assume the details are proprietary. This is a strategy that works well for the SAS Institute.

sheets. The best arrangement is for rows to be cases and columns to be variables. SAS can read data directly from an Excel spreadsheet; this is illustrated for Student's sleep data. Data sets coming from corporations and other organizations may be in Excel format, or they may be in a relational database produced by software such as Microsoft Access. Databases can be imported using `proc sql` (Structured Query Language).

- **The Program File:** The program file consists of commands that the SAS software tries to follow. You create this file with a text editor, either an external editor like Notepad, or a built-in editor. The program file contains a reference to the raw data file (in the `infile` statement), so SAS knows where to find the data. In the first example we will consider below, the program file is called `sleep1.sas`. SAS expects program files to have the extension `.sas`, and you should always follow this convention.
- **The Log File:** This file is produced by every SAS run, whether it is successful or unsuccessful. It contains a listing of the command file, as well any error messages or warnings. The name of the log file is automatically generated by SAS; It will be something like `reading1.log` or `reading1-log.html`.
- **The Output File:** The output file contains the output of the statistical procedures requested in the program file. Output files have names like `reading1-Results.pdf`, `reading1-Results.rtf`, or `reading1-Results.html`. A successful SAS run will almost always produce an output file. The absence of an output file indicates that there was at least one fatal error. The presence of an output file does not mean there were no errors; it just means that SAS was able to do *some* of what you asked it to do. Even if there are errors, the output file will usually not contain any error messages; they will be in the log file.

## 2.2 SAS University Edition

The SAS Institute make a great deal of money selling software licences to corporations, universities, government agencies, and to a lesser extent, individuals. Perhaps under pressure from the free R statistical software, they have recently been offering their core product free of charge to anyone with a university email address. It's called SAS University Edition. It's so well-designed and so convenient that it's difficult to imagine a professor choosing any other version of SAS for a statistics class. Here's the link:

[http://www.sas.com/en\\_us/software/university-edition.html](http://www.sas.com/en_us/software/university-edition.html)

Regardless of operating system, SAS University Edition lives in a virtual `linux` machine.<sup>2</sup> In addition to having SAS installed, the `linux` machine is a Web server. But the web pages

---

<sup>2</sup>A virtual computer is a set of software instructions that act like a complete, separate computer. So, for example, you could have a software version of the original IBM PC with the DOS operating system running on a modern laptop. Virtual machines are great for preserving legacy data and software, experimenting with viruses, and many other uses. In the bad old days, all the hardware in a virtual

it hosts are not available to the entire internet. They are available only to you. Rather than having a proper IP address, the virtual `linux` machine has a `localhost` address: `http://localhost:10080`. With SAS running in the virtual machine, you point your browser to this address. It looks like you are on the Internet, but really you are on a network located within your computer. It's entirely local, and would work at the bottom of a coal mine.

The browser interface (actually a website located on the virtual `linux` machine) is called SAS Studio. It's really nice, with tabs rather than separate windows for the program, log and output files. You can print files from the browser, or save output in `pdf`, `rtf` or `html` format. Because you are interacting with SAS indirectly through Web pages, the operating system on your computer does not matter much, if at all. If you are running Firefox on a Windows PC and I am running Safari on a Mac, the only differences we will experience are differences between Firefox and Safari. It's truly platform independent.

You get your data into SAS via a shared folder – shared between your computer and the virtual `linux` machine. In the `infile` statement of your SAS job, begin the name of the data file with `"/folders/myfolders/"` That's the path to the shared folder on the virtual `linux` machine. The shared folder on *your* machine can be anywhere. When you create the shared folder on your machine, make sure the spelling and capitalization of the folder names is exactly according to instructions. On your machine, the shared folder must be called `SASUniversityEdition`, with a sub-folder called `myfolders`. Sub-folders inside the folder `myfolders` are okay.

## 2.3 Example 1: Student's sleep data

### 2.3.1 The raw data file

The following illustrates a simple SAS run. The first step was to get the raw data file. It's a classic: the data that Student (William Gossett) used to illustrate the *t*-test in the 1908 *Biometrika* paper where he first reported it [22]. These data are given in Gossett's paper. I created a plain-text version of the raw data file called `studentsleep.data.txt` by typing the numbers into a text editor and dragging the file to the `myfolders` sub-folder of the shared folder `SASUniversityEdition`. Here's the data file. Take a look.

Patient	Drug 1	Drug 2
1	0.7	1.9
2	-1.6	0.8
3	-0.2	1.1
4	-1.2	0.1
5	-0.1	-0.1
6	3.4	4.4
7	3.7	5.5

---

machine was represented by software instructions, and they were *slow*. Now they can use the hardware of the host computer more directly, and there's not much of a performance hit.

8	0.8	1.6
9	0.0	4.6
10	2.0	3.4

Actually, it's so obvious that you should look at your data that it is seldom mentioned. But experienced data analysts always do it — or else they assume everything is okay and get a bitter lesson in something they already knew. This is so important that it gets the formal status of a **data analysis hint**.

**Data Analysis Hint 2** *Always look at your raw data file. If the data file is big, do it anyway. At least scroll through it, looking for anything strange. Check the values of all the variables for a few cases. Do they make sense? If you have obtained the data file from somewhere, along with a description of what's in it, never believe that the description you have been given is completely accurate.*

The file `studentsleep.data.txt` contains two variables for ten patients suffering from insomnia. Notice the variable names on the first line. Some software (like R) can use this information. As far as I know, SAS cannot. Furthermore, if SAS tries to read the data and encounters characters where it expects numbers, the results are unpleasant. One solution is to edit the raw data file and get rid of the labels, but actually labels like this can be useful. We'll get SAS to skip the first line, and start reading data from line two.

Each variable is actually a difference, representing how much *extra* sleep a patient got when taking a sleeping pill. Drug 1 is Dextro-hyoscyamine hydrobomide, while Drug 2 is Laevo-hyoscyamine hydrobomide. We want to know whether each drug is effective, and also which drug is more effective. Following Gosset, we'll use one-sample *t*-tests to decide whether each drug is effective; since these one-sample *t*-tests are carried out on differences, they are matched *t*-tests. We'll also compute a matched *t*-test comparing Drug 1 and Drug 2. Notice that this is a within-cases design.

To analyze the data with SAS, we need to create another plain text file containing the SAS program. SAS Studio has a nice built-in editor, and you can compose the whole SAS program with that. Or, you can do the first draft using an external text editor, drag it to `myfolders`, and then edit it there using the built-in SAS editor. If you do it this way, just make sure the program file has the extension `.sas`. For Student's sleep data, my program is called `sleep1.sas`.

### 2.3.2 Structure of the Program File

A SAS program file is composed of units called *data steps* and *proc steps*. The typical SAS program has one `data` step and at least one `proc` step, though other structures are possible.

- Most SAS commands belong either in `data` step or in a `proc` step; they will generate errors if they are used in the wrong kind of step.
- Some statements, like the `title` and `options` commands, exist outside of the `data` and `proc` steps, but there are relatively few of these.

**The Data Step** The `data` step takes care of data acquisition and modification. It almost always includes a reference to at least one raw data file, telling SAS where to look for the data. It specifies variable names and labels, and provides instructions about how to read the data; for example, the data might be read from fixed column locations. Variables from the raw data file can be modified, and new variables can be created.

Each data step creates a **SAS data table**, a file consisting of the data (after modifications and additions), labels, and so on. Statistical procedures operate on SAS data tables, so you must create a SAS data table before you can start computing any statistics.

A SAS data table is written in a binary format that is very convenient for SAS to process, but is not readable by humans. In the old days, SAS data tables were written to temporary scratch files on the computer's hard drive; these days, they may be maintained in RAM if they are small enough. In any case, the default is that a SAS data table disappears after the job has run. If the data step is executed again in a later run, the SAS data set is re-created.

Actually, it is possible to save a SAS data table on disk for later use. We won't do this here, but it makes sense when the amount of processing in a data step is large relative to the speed of the computer. As an extreme example, one of my colleagues uses SAS to analyze data from Ontario hospital admissions; the data files have millions of cases. Typically, it takes around 20 hours of CPU time on a very strong `unix` machine just to read the data and create a SAS data table. The resulting file, hundreds of gigabytes in size, is saved to disk, and then it takes just a few minutes to carry out each analysis. You wouldn't want to try this on a PC.

SAS data tables are not always created by SAS data steps. Some statistical procedures can create SAS data tables, too. For example, `proc standard` can take an ordinary SAS data table as input, and produce an output data table that has all the original variables, and also some of the variables converted to  $z$ -scores (by subtracting off the mean and dividing by the standard deviation). `Proc reg` (the main multiple regression procedure) can produce a SAS data table containing residuals for plotting and use in further analysis; there are many other examples.

**The proc Step** "Proc" is short for procedure. Most procedures are statistical procedures; the most noticeable exception is `proc format`, which is used to provide labels for the values of categorical variables. The `proc` step is where you specify a statistical procedure that you want to carry out. A statistical procedure in the `proc` step will take a SAS data table as input, and write the results (summary statistics, values of test statistics,  $p$ -values, and so on) to the output file. The typical SAS program includes one `data` step and several `proc` steps, because it is common to produce a variety of data displays, descriptive statistics and significance tests in a single run.

### 2.3.3 `sleep1.sas`

Now we will look at `sleep1.sas` in some detail. This program is very simple; it has just one data step and two `proc` steps.

```

/* sleep1.sas */
title "t-tests on Student's Sleep data";

data bedtime;
  infile '/folders/myfolders/studentsleep.data.txt' firstobs=2; /* Skip the header */
  input patient xsleep1 xsleep2;
  sleepdif = xsleep2-xsleep1; /* Create a new variable */

proc print;
  var patient xsleep1 xsleep2 sleepdif;

proc means n mean stddev t probt;
  var xsleep1 xsleep2 sleepdif;

```

Here are some detailed comments about `sleep1.sas`.

- The first line is a comment. Anything between a `/*` and `*/` is a comment, and will be listed on the log file but otherwise ignored by SAS. Comments can appear anywhere in a program. You are not required to use comments, but it's a good idea. The most common error associated with comments is to forget to end them with `*/`. In the case of `sleep1.sas`, leaving off the `*/` (or typing `/*` again by mistake) would cause the whole program to be treated as a comment. It would generate no errors, and no output — because as far as SAS would be concerned, you never requested any. A longer program would eventually exceed the default length of a comment (it's some large number of characters) and SAS would end the “comment” for you. At exactly that point (probably in the middle of a command) SAS would begin parsing the program. Almost certainly, the first thing it examined would be a fragment of a legal command, and this would cause an error. The log file would say that the command caused an error, and not much else. It would be *very* confusing, because probably the command would be okay, and there would be no indication that SAS was only looking at part of it.
- The next two lines (the `options` statement and the `title` statement) exist outside the `proc` step and outside the `data` step. This is fairly rare.
- All SAS statements end with a semi-colon (`;`). SAS statements can extend for several physical lines in the program file. Spacing, indentation, breaking up a statement into several lines of text — these are all for the convenience of the human reader, and are not part of the SAS syntax.
- By far the most common error in SAS programming is to forget the semi-colon. When this happens, SAS tries to interpret the following statement as part of the one you forgot to end. This often causes not one error, but a cascading sequence of errors. The rule is, *if you have an error and you do not immediately understand what it is, look for a missing semi-colon*. It will probably be *before* the portion of the program that (according to SAS) caused the first error.

- Cascading errors are not caused just by the dreaded missing semi-colon. They are common in SAS; for example, a runaway comment statement can easily cause a chain reaction of errors (if the program is long enough for it to cause any error messages at all). *If you have a lot of errors in your log file, fix the first one and re-run the job; and don't waste time trying to figure out the others.* Some or all of them may well disappear.
- **title** This is optional, but recommended. The material between the quotes will appear at the top of each page. This can be a lifesaver when you are searching through a stack of old printouts for something you did a year or two ago.
- **data bedtime;** This begins the data step, specifying that the name of the SAS data set being created is "bedtime." The names of data sets are arbitrary, but you should make them informative. They should begin with letters.
- **infile** Specifies the name of the raw data file. It must begin with `/folders/myfolders/`, the path to the shared folder in the virtual linux machine.
- **firstobs=2** Begin reading the data on line two, skipping the variable names. You can skip any number of lines this way, so a data file could potentially begin with a long description of how the data were collected.
- **input** Gives the names of the variables.
  - Variable names should begin with a letter. Avoid special characters like \$ or #. The variable names will be used to specify statistical procedures requested in the `proc` step. They should be meaningful (related to what the variable *is*), and easy to remember.
  - This is almost the simplest possible form of the `input` statement. It can be very powerful; for example, you can read data from different locations and in different orders, depending on the value of a variable you've just read, and so on. It can get complicated, but if the data file has a simple structure, the input statement can be simple too.
- **sleepdif = xsleep2-xsleep1;** Create a new variable, representing how much more sleep the patient got with Drug 2, compared to Drug 1. This calculation is performed for each case in the data file. Notice that the new variable `sleepdif` does *not* appear in the `input` statement. When some variables are to be created from others, it is a very good idea to do the computation within SAS. This makes raw data files smaller and more manageable, and also makes it easier to correct or re-define the computed variables.
- **proc print;** Now the first `proc` step begins. All we are doing is to list the data to make sure we have computed `sleepdif` correctly. This is actually a good thing to do whenever you compute a new variable. Of course you never (or very seldom) make hard copy of the complete output of `proc print`, because it can be very long. Once you're confident the data are what you think, delete the `proc print`.

- `var patient xsleep1 xsleep2 sleepdif`; List the variables you want to print. The word “`var`” is obligatory, and is among a fairly large number of names reserved by the SAS system. If you tried to name one of your variables `var`, it wouldn’t let you.
- `proc means`; This is the second `proc` step. `Proc means` is most often used to produce simple summary statistics for quantitative variables. The words `n mean stddev t probt` are optional, and specify that we want to see the following for each variable specified: the sample size, mean, standard deviation,  $t$ -test for testing whether the mean is different from zero, and the two-tailed  $p$ -value for the  $t$ -test. These are the paired  $t$ -tests we want. With just `proc means`; and not the option, we would get the default statistics:  $n$ , mean, standard deviation, minimum and maximum. These last two statistics are very useful, because they can alert you to outliers and errors in the data.
- `var` is obligatory. It is followed by a list of the variables for which you want to see means and other statistics.

### 2.3.4 sleep1.log

Log files are not very interesting when everything is okay, but here is an example anyway. Notice that in addition to a variety of technical information (where the files are, how long each step took, and so on), it contains a listing of the SAS program — in this case, `sleep1.sas`. If there were syntax errors in the program, this is where the error messages would appear.

```

1          OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
55
56          /* sleep1.sas */
57          title "t-tests on Student's Sleep data";
58
59          data mylatebedtime;
60              infile '/folders/myfolders/studentsleep.data.txt' firstobs=2; /* Skip t
61              input patient xsleep1 xsleep2;
62              sleepdif = xsleep2-xsleep1; /* Create a new variable */
63

```

NOTE: The infile `'/folders/myfolders/studentsleep.data.txt'` is:  
 Filename=`/folders/myfolders/studentsleep.data.txt`,  
 Owner Name=`root`,Group Name=`vboxsf`,  
 Access Permission=`--rwxrwx---`,  
 Last Modified=`05Jan2016:14:26:25`,  
 File Size (bytes)=`544`



NOTE: 10 records were read from the infile '/folders/myfolders/studentsleep.data.txt'.  
 The minimum record length was 47.  
 The maximum record length was 47.

NOTE: The data set WORK.MYLATEBEDTIME has 10 observations and 4 variables.

NOTE: DATA statement used (Total process time):

```
real time          0.01 seconds
cpu time           0.01 seconds
```

```
64      proc print;

65          var patient xsleep1 xsleep2 sleepdif;
66
```

NOTE: There were 10 observations read from the data set WORK.MYLATEBEDTIME.

NOTE: PROCEDURE PRINT used (Total process time):

```
real time          0.04 seconds
cpu time           0.05 seconds
```

```
67      proc means n mean stddev t probt;

68          var xsleep1 xsleep2 sleepdif;
69
70      OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
82
```

### 2.3.5 Output file

Here is the output file. Notice that the title specified in the `title` statement appears at the top. Then we get statistical output — in this case, the listing of raw data and table of means and *t*-tests.

```

                                t-tests on Student's Sleep data
Obs    patient    xsleep1    xsleep2    sleepdif
  1         1         0.7         1.9         1.2
  2         2        -1.6         0.8         2.4
  3         3        -0.2         1.1         1.3
  4         4        -1.2         0.1         1.3
  5         5        -0.1        -0.1         0.0
  6         6         3.4         4.4         1.0
  7         7         3.7         5.5         1.8
  8         8         0.8         1.6         0.8
```

9	9	0.0	4.6	4.6
10	10	2.0	3.4	1.4

t-tests on Student's Sleep data

2

The MEANS Procedure

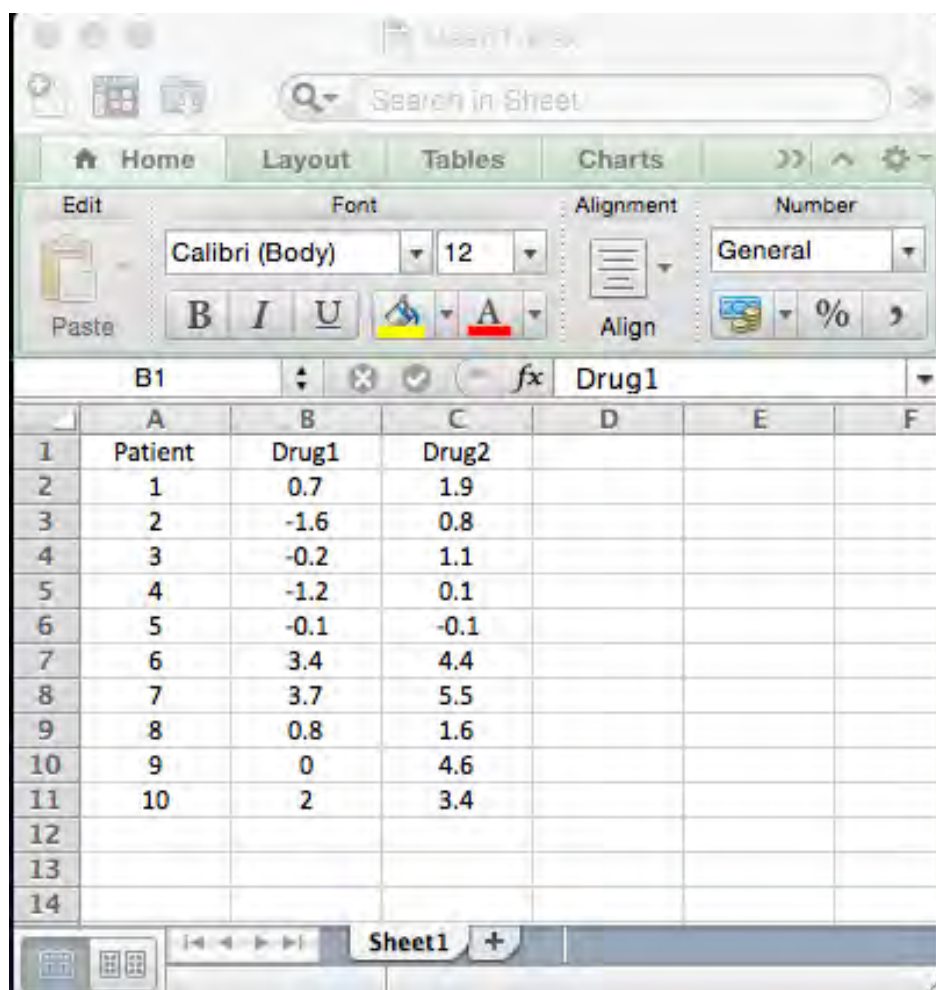
Variable	N	Mean	Std Dev	t Value	Pr >  t
xsleep1	10	0.7500000	1.7890097	1.33	0.2176
xsleep2	10	2.3300000	2.0022487	3.68	0.0051
sleepdif	10	1.5800000	1.2299955	4.06	0.0028

The output is pretty self-explanatory. The  $t$ -tests do not provide convincing evidence that Drug 1 was effective. They suggest that Drug 2 was effective, and better than Drug 1.

### 2.3.6 Reading from an Excel spreadsheet

For convenience (my convenience), most of the data files used in this textbook are in plain text format. I have had most of them for quite a while. Data collected more recently tend to be in Microsoft Excel spreadsheets. Whether you find this repulsive or not, it is a fact of life. The following will serve as a model for reading data directly from an Excel spreadsheet.

I pasted Student's sleep data into a spreadsheet called `sleep1.xlsx`. Here it is. Notice that the column names should be valid SAS names, with no embedded blanks. When the file type is `xlsx` watch out for leading and trailing blanks too. If you ignore this advice, SAS will convert the blanks to underscore characters (`_`)<sup>3</sup>, and you will need to look carefully at your log file to see what the variable names are.



The screenshot shows a Microsoft Excel spreadsheet with the following data:

	A	B	C	D	E	F
1	Patient	Drug1	Drug2			
2	1	0.7	1.9			
3	2	-1.6	0.8			
4	3	-0.2	1.1			
5	4	-1.2	0.1			
6	5	-0.1	-0.1			
7	6	3.4	4.4			
8	7	3.7	5.5			
9	8	0.8	1.6			
10	9	0	4.6			
11	10	2	3.4			
12						
13						
14						

<sup>3</sup>This is true as of SAS Version 9.4

Here's the SAS program.

```

/* sleep1c.sas */
title "t-tests on Student's Sleep data";
title2 'Read data from Excel Spreadsheet';

proc import datafile="/folders/myfolders/sleep1.xlsx"
            out=sleepy dbms=xlsx replace;
            getnames=yes;
/* Input data file is sleep1.xlsx
   Ouput data table is called sleepy
   dbms=xls The input file is an Excel spreadsheet.
           Necessary to read an Excel spreadsheet directly under unix/linux
           Works in PC environment too except for Excel 4.0 spreadsheets
           If there are multiple sheets, use sheet="sheet1" or something.
   replace If the data table already exists, replace it. Use this!
   getnames=yes Use column names as variable names. Beware of
           leading and trailing blanks */

/* proc print; */

data sleepy2;
  set sleepy; /* Now sleepy2=sleepy */
  sleepdif = Drug2-Drug1; /* Create a new variable */

proc print;
  var patient Drug1 drug2 sleepdif;

proc means n mean stddev t probt;
  var drug1 drug2 sleepdif;

```

After the title, the first part of the program is a `proc import`, which imports the data into SAS. The code is thoroughly commented, but here are some details anyway.

- `proc import`
  - `out=sleepy` creates a new data table called `sleepy`.
  - `dbms=xlsx` specifies that it's an `xlsx` spreadsheet. This specification is necessary to read an Excel spreadsheet directly under `unix/linux`. According to the manuals, it works in a Windows environment too except for Excel 4.0 spreadsheets. If you are reading a spreadsheet in the older `xls` format, just replace `xlsx` with `xls` throughout.
  - `replace`: If the data table already exists, replace it. Always use this! If you do not, any corrections you make to the spreadsheet will be ignored.

- `getnames=yes`: Use column names as variable names. Beware of leading and trailing blanks.
- `proc print`; This is commented out. It was used to verify that the data were imported correctly. This is highly recommended. You will ultimately save time by cross-checking everything you can.
- `data sleepy2`; This data step creates a new data table called `sleepy2`. The `proc import` created the data table `sleepy`, but you can't get at it directly to do anything else. The solution is to put the contents of `sleepy` into a new data table and modify that.
  - `set sleepy`; This brings the contents of `sleepy` into `sleepy2`.
  - `sleepdif = Drug2-Drug1`; This creates the new variable `sleepdif`. Now it's possible to compute more new variables, add labels and do all the other things you'd do in a `data` step.

The rest is the same as the original example, except that I played with the capitalization of variable names to remind you that SAS is not very case sensitive.

## 2.4 SAS Example Two: The statclass data

These data come from a statistics class taught many years ago. Students took eight quizzes, turned in nine computer assignments, and also took a midterm and final exam. The data file also includes gender and ethnic background; these last two variables are just guesses by the professor, and there is no way to tell how accurate they were. The data file looks like this. There are 21 columns and 62 rows of data; columns are not aligned and there are no column headers. Here are the first few lines.

```

1 2 9 1 7 8 4 3 5 2 6 10 10 10 5 0 0 0 0 55 43
0 2 10 10 5 9 10 8 6 8 10 10 8 9 9 9 9 10 10 66 79
1 2 10 10 5 10 10 10 9 8 10 10 10 10 10 10 9 10 10 94 67
1 2 10 10 8 9 10 7 10 9 10 10 10 9 10 10 9 10 10 81 65
0 1 10 1 0 0 8 6 5 2 10 9 0 0 10 6 0 5 0 54 .
1 1 10 6 7 9 8 8 5 7 10 9 10 9 5 6 4 8 10 57 52
0 1 0 0 9 9 10 5 2 2 8 7 7 10 10 6 3 7 10 49 .
0 1 10 9 5 8 9 8 5 6 8 7 5 6 10 6 5 9 9 77 64
0 1 10 8 6 8 9 5 3 6 9 9 6 9 10 6 5 7 10 65 42
1 1 10 5 6 7 10 4 6 0 10 9 10 9 10 6 7 8 10 73 .
0 1 9 0 4 6 10 5 3 3 10 8 10 5 10 10 9 9 10 71 37

:

```

Notice the periods at the ends of lines 5, 7 and 10. The period is the SAS *missing value code*. These people did not show up for the final exam. They may have taken a makeup exam, but if so their scores did not make it into this data file. When a case has a missing value recorded for a variable, SAS automatically excludes that case from any statistical calculation involving the variable. If a new variable is being created based on the value of a variable with a missing value, the new variable will usually have a missing value for that case too.

Here is the SAS program `statmarks1.sas`. It reads and labels the data, and then does a variety of significance tests. They are all elementary except the last one, which illustrates testing for one set of explanatory variables controlling for another set in multiple regression.

```

                                /* statmarks1.sas */
title 'Grades from STA3000 at Roosevelt University:  Fall, 1957';
title2 'Illustrate Elementary Tests';

proc format; /* Used to label values of the categorical variables */
  value sexfmt      0 = 'Male'    1 = 'Female';
  value ethfmt      1 = 'Chinese'
                  2 = 'European'
                  3 = 'Other' ;

data grades;
  infile '/folders/myfolders/statclass1.data.txt';
  input sex ethnic quiz1-quiz8 comp1-comp9 midterm final;
  /* Drop lowest score for quiz & computer */
  quizave = ( sum(of quiz1-quiz8) - min(of quiz1-quiz8) ) / 7;
  compave = ( sum(of comp1-comp9) - min(of comp1-comp9) ) / 8;
  label ethnic = 'Apparent ethnic background (ancestry)'
        quizave = 'Quiz Average (drop lowest)'
        compave = 'Computer Average (drop lowest)';
  mark = .3*quizave*10 + .1*compave*10 + .3*midterm + .3*final;
  label mark = 'Final Mark';
  diff = quiz8-quiz1; /* To illustrate matched t-test */
  label diff = 'Quiz 8 minus Quiz 1';
  mark2 = round(mark);
  /* Bump up at grade boundaries */
  if mark2=89 then mark2=90;
  if mark2=79 then mark2=80;
  if mark2=69 then mark2=70;
  if mark2=59 then mark2=60;
  /* Assign letter grade */
  if mark2=. then grade='Incomplete';
  else if mark2 ge 90 then grade = 'A';
  else if 80 le mark2 le 89 then grade='B';

```

```
        else if 70 le mark2 le 79 then grade='C';
        else if 60 le mark2 le 69 then grade='D';
        else grade='F';

format sex sexfmt.;          /* Associates sex & ethnic */
format ethnic ethfmt.;      /* with formats defined above */

proc freq;
    title3 'Frequency distributions of the categorical variables';
    tables sex ethnic grade;

proc means;
    title3 'Means and SDs of quantitative variables';
    var quiz1 -- mark;        /* single dash only works with numbered
                               lists, like quiz1-quiz8 */

proc ttest;
    title3 'Independent t-test';
    class sex;
    var mark;

proc means n mean std t probt;
    title3 'Matched t-test: Quiz 1 versus 8';
    var quiz1 quiz8 diff;

proc glm;
    title3 'One-way anova';
    class ethnic;
    model mark = ethnic;
    means ethnic;
    means ethnic / Tukey Bon Scheffe;

proc freq;
    title3 'Chi-squared Test of Independence';
    tables sex*ethnic sex*grade ethnic*grade / chisq;
proc freq; /* Added after seeing warning from chisq test above */
    title3 'Chi-squared Test of Independence: Version 2';
    tables sex*ethnic grade*(sex ethnic) / norow nopercnt chisq expected;

proc corr;
    title3 'Correlation Matrix';
    var final midterm quizave compave;

proc plot;
    title3 'Scatterplot';
    plot final*midterm; /* Really should do all combinations */

proc reg;
    title3 'Simple regression';
    model final=midterm;
/* Predict final exam score from midterm, quiz & computer */
```

```
proc reg simple;
  title3 'Multiple Regression';
  model final = midterm quizave compave / ss1;
  smalstuf: test quizave = 0, compave = 0;
```

Noteworthy features of this program include

- **options:** Already discussed in connection with `sleep1.sas`.
- **title2:** Subtitle
- **proc format:** This is a non-statistical procedure – a rarity in the SAS language. It is the way SAS takes care of labelling categorical variables when the categories are coded as numbers. `proc format` defines *printing formats*. For any variable associated with the printing format named `sexfmt`, any time it would print the value “0” (in a table or something) it instead prints the string “Male.” The associations between variables and printing formats are accomplished in the `format` statement at the end of the data step. The names of formats have a period at the end to distinguish them from variable names. Of course formats must be defined before they can be associated with variables. This is why `proc format` precedes the data step.
- **quiz1-quiz8:** One may refer to a *range* of variables ending with consecutive numbers using a minus sign. In the `input` statement, a range can be defined (named) this way. It saves typing and is easy to read.
- Creating new variables with assignment statements. The variables `quizave`, `compave` and `mark` are not in the original data file. They are created here, and they are appended to the end of the SAS data set in order of creation. Variables like this should never be in the raw data file.

**Data Analysis Hint 3** *When variables are exact mathematical functions of other variables, always create them in the data step rather than including them in the raw data file. It saves data entry, and makes the data file smaller and easier to read. If you want to try out a different definition of the variable, it's easy to change a few statements in the data step.*

- **sum(of quiz1-quiz8):** Without the word “of,” the minus sign is ambiguous. In the SAS language, `sum(quiz1-quiz8)` is the sum of a single number, the difference between `quiz1` and `quiz8`.
- **format sex sexfmt.;** Associates the variable `sex` with its printing format. In questionnaire studies where a large number of items have the same potential responses (like a scale from 1 = Strongly Agree to 7=Strongly Disagree), it is common to associate a long list of variables with a single printing format.



- `quiz1 -- mark` in the first `proc means`: A double dash refers to a list of variables *in the order of their creation* in the `data` step. Single dashes are for numerical order, while double dashes are for order of creation; it's very handy.
- Title inside a procedure labels just that procedure.
- `proc means n mean std t` A matched t-test is just a single-variable t-test carried out on differences, testing whether the mean difference is equal to zero.
- `proc glm`
  - `class` Tells SAS that the explanatory variable `ethnic` is categorical.
  - `model` Response variable(s) = explanatory variable(s)
  - `means ethnic`: Mean of `mark` separately for each value of `ethnic`.
  - `means ethnic / Tukey Bon Scheffe`: Post hoc tests (multiple comparisons, probing, follow-ups). Used if the overall  $F$ -test is significant, to see which means are different from which other means.
- `chisq` option on `proc freq`: Gives a large collection of chisquare tests. The first one is the familiar Pearson chisquare test of independence (the one comparing observed and expected frequencies).
- `tables sex*ethnic / norow nopercent chisq expected`; In this second version of the crosstab produced `proc freq`, we suppress the row and total percentages, and look at the expected frequencies because SAS warned us that some of them were too small. SAS issues a warning if any expected frequency is below 5; this is the old-fashioned rule of thumb. But it has been known for some time that Type I error rates are affected mostly by expected frequencies smaller than one, not five — so I wanted to take a look.
- `proc corr` After `var`, list the variables you want to see in a correlation matrix.
- `proc plot; plot final*midterm`; Scatterplot: First variable named goes on the  $y$  axis.
- `proc reg; model` Response variable(s) = explanatory variable(s) again
- `simple` option on `proc reg` gives simple descriptive statistics. This last procedure is an example of multiple regression, and we will return to it later once we have more background.

## The output file

```
-----
Grades from STA3000 at Roosevelt University:  Fall, 1957          1
  Illustrate Elementary Tests
  Frequency distributions of the categorical variables
```

## The FREQ Procedure

sex	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Male	39	62.90	39	62.90
Female	23	37.10	62	100.00

## Apparent ethnic background (ancestry)

ethnic	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Chinese	41	66.13	41	66.13
European	15	24.19	56	90.32
Other	6	9.68	62	100.00

grade	Frequency	Percent	Cumulative Frequency	Cumulative Percent
A	3	4.84	3	4.84
B	6	9.68	9	14.52
C	18	29.03	27	43.55
D	21	33.87	48	77.42
F	10	16.13	58	93.55
Incomplete	4	6.45	62	100.00

```
-----
Grades from STA3000 at Roosevelt University:  Fall, 1957          2
  Illustrate Elementary Tests
  Means and SDs of quantitative variables
```

## The MEANS Procedure

Variable	Label	N	Mean	Std Dev
quiz1		62	9.0967742	2.2739413
quiz2		62	5.8870968	3.2294995
quiz3		62	6.0483871	2.3707744
quiz4		62	7.7258065	2.1590022
quiz5		62	9.0645161	1.4471109
quiz6		62	7.1612903	1.9264641
quiz7		62	5.7903226	2.1204477
quiz8		62	6.3064516	2.3787909
comp1		62	9.1451613	1.1430011
comp2		62	8.8225806	1.7604414
comp3		62	8.3387097	2.5020880
comp4		62	7.8548387	3.2180168
comp5		62	9.4354839	1.7237109
comp6		62	7.8548387	2.4350364
comp7		62	6.6451613	2.7526248
comp8		62	8.8225806	1.6745363
comp9		62	8.2419355	3.7050497
midterm		62	70.1935484	13.6235557
final		58	50.3103448	17.2496701

quizave	Quiz Average (drop lowest)	62	7.6751152	1.1266917
compave	Computer Average (drop lowest)	62	8.8346774	1.1204997
mark	Final Mark	58	68.4830049	10.3902874

Variable	Label	Minimum	Maximum
quiz1		0	10.0000000
quiz2		0	10.0000000
quiz3		0	10.0000000
quiz4		0	10.0000000
quiz5		4.0000000	10.0000000
quiz6		3.0000000	10.0000000
quiz7		0	10.0000000
quiz8		0	10.0000000
comp1		6.0000000	10.0000000
comp2		0	10.0000000
comp3		0	10.0000000
comp4		0	10.0000000
comp5		0	10.0000000
comp6		0	10.0000000
comp7		0	10.0000000
comp8		0	10.0000000
comp9		0	10.0000000
midterm		44.0000000	103.0000000
final		15.0000000	89.0000000
quizave	Quiz Average (drop lowest)	4.5714286	9.7142857
compave	Computer Average (drop lowest)	5.0000000	10.0000000
mark	Final Mark	48.4821429	95.4571429

Grades from STA3000 at Roosevelt University: Fall, 1957 3  
 Illustrate Elementary Tests  
 Independent t-test

The TTEST Procedure

Statistics

Variable	sex	N	Lower CL Mean	Upper CL Mean	Lower CL Std Dev	Upper CL Std Dev
mark	Male	36	65.604	68.57	71.535	7.1093
mark	Female	22	62.647	68.341	74.036	9.8809
mark	Diff (1-2)		-5.454	0.2284	5.9108	8.8495

Statistics

Variable	sex	Upper CL Std Dev	Std Err	Minimum	Maximum
mark	Male	11.434	1.4609	54.057	89.932
mark	Female	18.354	2.7382	48.482	95.457
mark	Diff (1-2)	12.859	2.8366		

T-Tests

Variable	Method	Variances	DF	t Value	Pr >  t
mark	Pooled	Equal	56	0.08	0.9361
mark	Satterthwaite	Unequal	33.1	0.07	0.9418

## Equality of Variances

Variable	Method	Num DF	Den DF	F Value	Pr > F
mark	Folded F	21	35	2.15	0.0443

-----

Grades from STA3000 at Roosevelt University: Fall, 1957 4  
 Illustrate Elementary Tests  
 Matched t-test: Quiz 1 versus 8

## The MEANS Procedure

Variable	Label	N	Mean	Std Dev	t Value
quiz1		62	9.0967742	2.2739413	31.50
quiz8		62	6.3064516	2.3787909	20.87
diff	Quiz 8 minus Quiz 1	62	-2.7903226	3.1578011	-6.96

Variable	Label	Pr >  t
quiz1		<.0001
quiz8		<.0001
diff	Quiz 8 minus Quiz 1	<.0001

-----

Grades from STA3000 at Roosevelt University: Fall, 1957 5  
 Illustrate Elementary Tests  
 One-way anova

## The GLM Procedure

## Class Level Information

Class	Levels	Values
ethnic	3	Chinese European Other

Number of Observations Read	62
Number of Observations Used	58

-----

Grades from STA3000 at Roosevelt University: Fall, 1957 6  
 Illustrate Elementary Tests  
 One-way anova

## The GLM Procedure

Dependent Variable: mark Final Mark

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	1238.960134	619.480067	6.93	0.0021
Error	55	4914.649951	89.357272		
Corrected Total	57	6153.610084			

	R-Square	Coeff Var	Root MSE	mark Mean		
	0.201339	13.80328	9.452898	68.48300		
Source	DF	Type I SS	Mean Square	F Value	Pr > F	
ethnic	2	1238.960134	619.480067	6.93	0.0021	
Source	DF	Type III SS	Mean Square	F Value	Pr > F	
ethnic	2	1238.960134	619.480067	6.93	0.0021	

-----

Grades from STA3000 at Roosevelt University: Fall, 1957 7  
 Illustrate Elementary Tests  
 One-way anova

The GLM Procedure

Level of ethnic	N	-----mark----- Mean	Std Dev
Chinese	37	65.2688224	7.9262171
European	15	76.0142857	11.2351562
Other	6	69.4755952	13.3097753

-----

Grades from STA3000 at Roosevelt University: Fall, 1957 8  
 Illustrate Elementary Tests  
 One-way anova

The GLM Procedure

Tukey's Studentized Range (HSD) Test for mark

NOTE: This test controls the Type I experimentwise error rate.

Alpha	0.05
Error Degrees of Freedom	55
Error Mean Square	89.35727
Critical Value of Studentized Range	3.40649

Comparisons significant at the 0.05 level are indicated by \*\*\*.

ethnic Comparison	Difference Between Means	Simultaneous 95% Confidence Limits
European - Other	6.539	-4.460 17.538
European - Chinese	10.745	3.776 17.715 ***
Other - European	-6.539	-17.538 4.460
Other - Chinese	4.207	-5.814 14.228
Chinese - European	-10.745	-17.715 -3.776 ***
Chinese - Other	-4.207	-14.228 5.814

---

Grades from STA3000 at Roosevelt University: Fall, 1957 9  
 Illustrate Elementary Tests  
 One-way anova

The GLM Procedure

Bonferroni (Dunn) t Tests for mark

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than Tukey's for all pairwise comparisons.

Alpha	0.05
Error Degrees of Freedom	55
Error Mean Square	89.35727
Critical Value of t	2.46941

Comparisons significant at the 0.05 level are indicated by \*\*\*.

ethnic Comparison	Difference Between Means	Simultaneous 95% Confidence Limits
European - Other	6.539	-4.737 17.814
European - Chinese	10.745	3.600 17.891 ***
Other - European	-6.539	-17.814 4.737
Other - Chinese	4.207	-6.067 14.480
Chinese - European	-10.745	-17.891 -3.600 ***
Chinese - Other	-4.207	-14.480 6.067

---

Grades from STA3000 at Roosevelt University: Fall, 1957 10  
 Illustrate Elementary Tests  
 One-way anova

The GLM Procedure

Scheffe's Test for mark

NOTE: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than Tukey's for all pairwise comparisons.

Alpha	0.05
Error Degrees of Freedom	55
Error Mean Square	89.35727
Critical Value of F	3.16499

Comparisons significant at the 0.05 level are indicated by \*\*\*.

ethnic Comparison	Difference Between Means	Simultaneous 95% Confidence Limits
European - Other	6.539	-4.950 18.027
European - Chinese	10.745	3.466 18.025 ***
Other - European	-6.539	-18.027 4.950

Other	- Chinese	4.207	-6.260	14.674	
Chinese	- European	-10.745	-18.025	-3.466	***
Chinese	- Other	-4.207	-14.674	6.260	

-----

Grades from STA3000 at Roosevelt University: Fall, 1957 11  
 Illustrate Elementary Tests  
 Chi-squared Test of Independence

The FREQ Procedure

Table of sex by ethnic

sex ethnic(Apparent ethnic background (ancestry))

Frequency				
Percent				
Row Pct				
Col Pct	Chinese	European	Other	Total
Male	27	7	5	39
	43.55	11.29	8.06	62.90
	69.23	17.95	12.82	
	65.85	46.67	83.33	
Female	14	8	1	23
	22.58	12.90	1.61	37.10
	60.87	34.78	4.35	
	34.15	53.33	16.67	
Total	41	15	6	62
	66.13	24.19	9.68	100.00

Statistics for Table of sex by ethnic

Statistic	DF	Value	Prob
Chi-Square	2	2.9208	0.2321
Likelihood Ratio Chi-Square	2	2.9956	0.2236
Mantel-Haenszel Chi-Square	1	0.0000	0.9949
Phi Coefficient		0.2170	
Contingency Coefficient		0.2121	
Cramer's V		0.2170	

WARNING: 33% of the cells have expected counts less than 5. Chi-Square may not be a valid test.

Sample Size = 62

-----

Grades from STA3000 at Roosevelt University: Fall, 1957 12  
 Illustrate Elementary Tests  
 Chi-squared Test of Independence

The FREQ Procedure

Table of sex by grade

sex	grade							Total
Frequency								
Percent								
Row Pct								
Col Pct	A	B	C	D	F	Incomplete		
Male	1	3	13	14	5	3	39	
	1.61	4.84	20.97	22.58	8.06	4.84	62.90	
	2.56	7.69	33.33	35.90	12.82	7.69		
	33.33	50.00	72.22	66.67	50.00	75.00		
Female	2	3	5	7	5	1	23	
	3.23	4.84	8.06	11.29	8.06	1.61	37.10	
	8.70	13.04	21.74	30.43	21.74	4.35		
	66.67	50.00	27.78	33.33	50.00	25.00		
Total	3	6	18	21	10	4	62	
	4.84	9.68	29.03	33.87	16.13	6.45	100.00	

Statistics for Table of sex by grade

Statistic	DF	Value	Prob
Chi-Square	5	3.3139	0.6517
Likelihood Ratio Chi-Square	5	3.2717	0.6582
Mantel-Haenszel Chi-Square	1	0.2342	0.6284
Phi Coefficient		0.2312	
Contingency Coefficient		0.2253	
Cramer's V		0.2312	

WARNING: 58% of the cells have expected counts less than 5. Chi-Square may not be a valid test.

Sample Size = 62



-----

Grades from STA3000 at Roosevelt University: Fall, 1957 13  
 Illustrate Elementary Tests  
 Chi-squared Test of Independence

The FREQ Procedure

Table of ethnic by grade

ethnic(Apparent ethnic background (ancestry))		grade						
Frequency								
Percent								
Row Pct								
Col Pct	A	B	C	D	F	Incomple	Total	
						te		
Chinese	0	2	11	17	7	4	41	
	0.00	3.23	17.74	27.42	11.29	6.45	66.13	
	0.00	4.88	26.83	41.46	17.07	9.76		
	0.00	33.33	61.11	80.95	70.00	100.00		
European	2	4	5	3	1	0	15	
	3.23	6.45	8.06	4.84	1.61	0.00	24.19	
	13.33	26.67	33.33	20.00	6.67	0.00		
	66.67	66.67	27.78	14.29	10.00	0.00		
Other	1	0	2	1	2	0	6	
	1.61	0.00	3.23	1.61	3.23	0.00	9.68	
	16.67	0.00	33.33	16.67	33.33	0.00		
	33.33	0.00	11.11	4.76	20.00	0.00		
Total	3	6	18	21	10	4	62	
	4.84	9.68	29.03	33.87	16.13	6.45	100.00	

Statistics for Table of ethnic by grade

Statistic	DF	Value	Prob
Chi-Square	10	18.2676	0.0506
Likelihood Ratio Chi-Square	10	19.6338	0.0329
Mantel-Haenszel Chi-Square	1	5.6222	0.0177
Phi Coefficient		0.5428	
Contingency Coefficient		0.4771	
Cramer's V		0.3838	

WARNING: 78% of the cells have expected counts less than 5. Chi-Square may not be a valid test.

Sample Size = 62

-----

Grades from STA3000 at Roosevelt University: Fall, 1957 14  
 Illustrate Elementary Tests  
 Chi-squared Test of Independence: Version 2

The FREQ Procedure

Table of sex by ethnic

sex ethnic(Apparent ethnic background (ancestry))

Frequency				
Expected				
Col Pct	Chinese	European	Other	Total
-----+-----+-----+-----+				
Male	27	7	5	39
	25.79	9.4355	3.7742	
	65.85	46.67	83.33	
-----+-----+-----+-----+				
Female	14	8	1	23
	15.21	5.5645	2.2258	
	34.15	53.33	16.67	
-----+-----+-----+-----+				
Total	41	15	6	62

Statistics for Table of sex by ethnic

Statistic	DF	Value	Prob
Chi-Square	2	2.9208	0.2321
Likelihood Ratio Chi-Square	2	2.9956	0.2236
Mantel-Haenszel Chi-Square	1	0.0000	0.9949
Phi Coefficient		0.2170	
Contingency Coefficient		0.2121	
Cramer's V		0.2170	

WARNING: 33% of the cells have expected counts less than 5. Chi-Square may not be a valid test.

Sample Size = 62

-----

Grades from STA3000 at Roosevelt University: Fall, 1957 15  
 Illustrate Elementary Tests  
 Chi-squared Test of Independence: Version 2

The FREQ Procedure

Table of grade by sex

grade	sex		
Frequency			
Expected			
Col Pct	Male	Female	Total
A	1	2	3
	1.8871	1.1129	
	2.56	8.70	
B	3	3	6
	3.7742	2.2258	
	7.69	13.04	
C	13	5	18
	11.323	6.6774	
	33.33	21.74	
D	14	7	21
	13.21	7.7903	
	35.90	30.43	
F	5	5	10
	6.2903	3.7097	
	12.82	21.74	
Incomplete	3	1	4
	2.5161	1.4839	
	7.69	4.35	
Total	39	23	62

Statistics for Table of grade by sex

Statistic	DF	Value	Prob
Chi-Square	5	3.3139	0.6517
Likelihood Ratio Chi-Square	5	3.2717	0.6582
Mantel-Haenszel Chi-Square	1	0.2342	0.6284
Phi Coefficient		0.2312	
Contingency Coefficient		0.2253	
Cramer's V		0.2312	

WARNING: 58% of the cells have expected counts less than 5. Chi-Square may not be a valid test.

Sample Size = 62

-----  
 Grades from STA3000 at Roosevelt University: Fall, 1957 16  
 Illustrate Elementary Tests  
 Chi-squared Test of Independence: Version 2

The FREQ Procedure

Table of grade by ethnic

grade	ethnic(Apparent ethnic background (ancestry))			
Frequency	Chinese	European	Other	Total
Expected				
Col Pct				
A	0	2	1	3
	1.9839	0.7258	0.2903	
	0.00	13.33	16.67	
B	2	4	0	6
	3.9677	1.4516	0.5806	
	4.88	26.67	0.00	
C	11	5	2	18
	11.903	4.3548	1.7419	
	26.83	33.33	33.33	
D	17	3	1	21
	13.887	5.0806	2.0323	
	41.46	20.00	16.67	
F	7	1	2	10
	6.6129	2.4194	0.9677	
	17.07	6.67	33.33	
Incomplete	4	0	0	4
	2.6452	0.9677	0.3871	
	9.76	0.00	0.00	
Total	41	15	6	62

Statistics for Table of grade by ethnic

Statistic	DF	Value	Prob
Chi-Square	10	18.2676	0.0506
Likelihood Ratio Chi-Square	10	19.6338	0.0329
Mantel-Haenszel Chi-Square	1	5.6222	0.0177
Phi Coefficient		0.5428	
Contingency Coefficient		0.4771	
Cramer's V		0.3838	

WARNING: 78% of the cells have expected counts less than 5. Chi-Square may not be a valid test.

Sample Size = 62

```

-----
Grades from STA3000 at Roosevelt University: Fall, 1957      17
  Illustrate Elementary Tests
    Correlation Matrix

  The CORR Procedure

  4 Variables:   final   midterm  quizave  compave

Simple Statistics

Variable      N      Mean      Std Dev      Sum      Minimum      Maximum
final         58     50.31034    17.24967     2918     15.00000     89.00000
midterm       62     70.19355    13.62356     4352     44.00000    103.00000
quizave       62      7.67512     1.12669    475.85714     4.57143     9.71429
compave       62      8.83468     1.12050    547.75000     5.00000    10.00000

Simple Statistics

Variable  Label
final
midterm
quizave  Quiz Average (drop lowest)
compave  Computer Average (drop lowest)

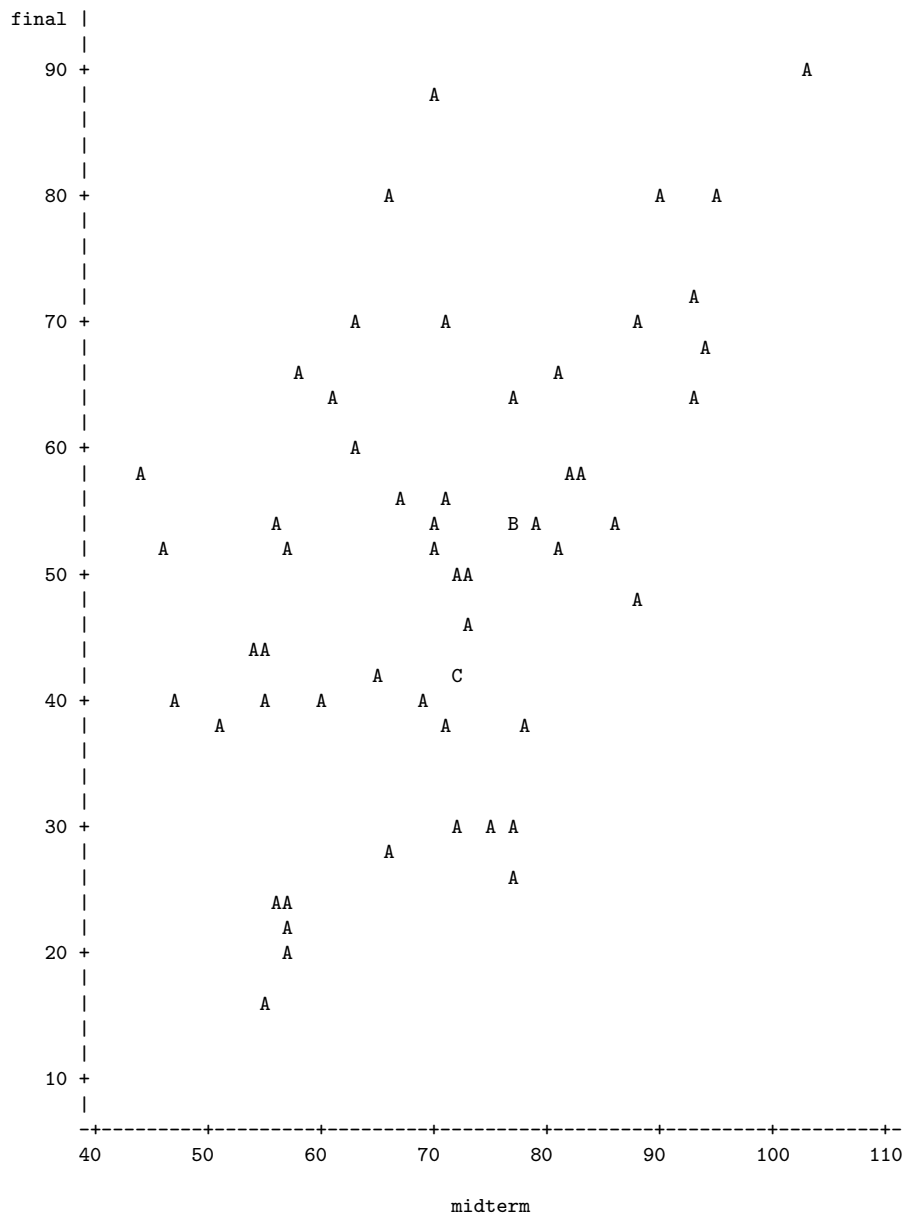
Pearson Correlation Coefficients
Prob > |r| under H0: Rho=0
Number of Observations

              final   midterm   quizave   compave
final
              1.00000   0.47963   0.41871   0.06060
              58       58       58       58
midterm
              0.47963   1.00000   0.59294   0.41277
              0.0001   62       <.0001    0.0009
              58       62       62       62
quizave
Quiz Average (drop lowest)
              0.41871   0.59294   1.00000   0.52649
              0.0011   <.0001    62       <.0001
              58       62       62       62
compave
Computer Average (drop lowest)
              0.06060   0.41277   0.52649   1.00000
              0.6513   0.0009   <.0001    62
              58       62       62
-----

```

Grades from STA3000 at Roosevelt University: Fall, 1957 18  
 Illustrate Elementary Tests  
 Scatterplot

Plot of final\*midterm. Legend: A = 1 obs, B = 2 obs, etc.



NOTE: 4 obs had missing values.

-----  
 Grades from STA3000 at Roosevelt University: Fall, 1957 19  
 Illustrate Elementary Tests  
 Simple regression

The REG Procedure  
 Model: MODEL1  
 Dependent Variable: final

Number of Observations Read 62  
 Number of Observations Used 58  
 Number of Observations with Missing Values 4

## Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	1	3901.64751	3901.64751	16.73	0.0001
Error	56	13059	233.19226		
Corrected Total	57	16960			

Root MSE 15.27063 R-Square 0.2300  
 Dependent Mean 50.31034 Adj R-Sq 0.2163  
 Coeff Var 30.35287

## Parameter Estimates

Variable	DF	Parameter Estimate	Standard Error	t Value	Pr >  t
Intercept	1	6.88931	10.80304	0.64	0.5263
midterm	1	0.61605	0.15061	4.09	0.0001

-----  
 Grades from STA3000 at Roosevelt University: Fall, 1957 20  
 Illustrate Elementary Tests  
 Multiple Regression

The REG Procedure

Number of Observations Read 62  
 Number of Observations Used 58  
 Number of Observations with Missing Values 4

## Descriptive Statistics

Variable	Sum	Mean	Uncorrected SS	Variance	Standard Deviation
Intercept	58.00000	1.00000	58.00000	0	0
midterm	4088.00000	70.48276	298414	180.35935	13.42979
quizave	451.57143	7.78571	3576.51020	1.06498	1.03198
compave	515.50000	8.88793	4641.50000	1.04862	1.02402
final	2918.00000	50.31034	163766	297.55112	17.24967

## Descriptive Statistics

Variable	Label
Intercept	Intercept
midterm	
quizave	Quiz Average (drop lowest)
compave	Computer Average (drop lowest)
final	

-----

Grades from STA3000 at Roosevelt University: Fall, 1957 21  
 Illustrate Elementary Tests  
 Multiple Regression

The REG Procedure  
 Model: MODEL1  
 Dependent Variable: final

Number of Observations Read	62
Number of Observations Used	58
Number of Observations with Missing Values	4

## Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	3	4995.04770	1665.01590	7.51	0.0003
Error	54	11965	221.58085		
Corrected Total	57	16960			

Root MSE	14.88559	R-Square	0.2945
Dependent Mean	50.31034	Adj R-Sq	0.2553
Coeff Var	29.58754		

## Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error
Intercept	Intercept	1	9.01839	19.02591
midterm		1	0.50057	0.18178
quizave	Quiz Average (drop lowest)	1	4.80199	2.46469
compave	Computer Average (drop lowest)	1	-3.53028	2.17562

## Parameter Estimates

Variable	Label	DF	t Value	Pr >  t	Type I SS
Intercept	Intercept	1	0.47	0.6374	146806
midterm		1	2.75	0.0080	3901.64751
quizave	Quiz Average (drop lowest)	1	1.95	0.0566	509.97483
compave	Computer Average (drop lowest)	1	-1.62	0.1105	583.42537

-----



```

Grades from STA3000 at Roosevelt University:  Fall, 1957      22
      Illustrate Elementary Tests
      Multiple Regression

      The REG Procedure
      Model: MODEL1

      Test smalstuf Results for Dependent Variable final

Source          DF          Mean
                DF          Square    F Value    Pr > F
Numerator        2          546.70010    2.47    0.0943
Denominator     54          221.58085

```

**Data in fixed columns** When the data values have at least one space between them, the variables are recorded in the same order for each case, and missing values are indicated by periods, the default version of the `input` statement (list input) does the job perfectly. It is a bonus that the variables need not always be separated by the same number of spaces for each case. Also, there can be more than one line of data for each case, and in fact there need not even be the same number of data lines for all the cases, just as long as there are the same number of variables.

Another common situation is for the data to be lined up in fixed columns, with blanks for missing values. Sometimes, especially when there are many variables, the data are *packed* together, without spaces between values. For example, the Minnesota Multiphasic Personality Inventory (MMPI) consists of over 300 questions, all to be answered True or False. It would be quite natural to code 1=True and 0=False, and pack the data together. There would still be quite a few data lines for each case.

Here is the beginning of the file `statclass2.dat`. It is the same as `statclass1.dat`, except that the data are packed together. Most of the blanks occur because two columns are reserved for the marks on quizzes and computer assignments, because 10 out of 10 is possible. Three columns are reserved for the midterm and final scores, because 100% is possible. For all variables, missing values are represented by blanks. That is, if the field occupied by a variable is completely blank, it's a missing value.

```

12 9 1 7 8 4 3 5 2 6101010 5 0 0 0 0 55 43
021010 5 910 8 6 81010 8 9 9 9 91010 66 79
121010 5101010 9 8101010101010 91010 94 67
121010 8 910 710 9101010 91010 91010 81 65
0110 1 0 0 8 6 5 210 9 0 010 6 0 5 0 54
1110 6 7 9 8 8 5 710 910 9 5 6 4 810 57 52
01 0 0 9 910 5 2 2 8 7 71010 6 3 710 49
0110 9 5 8 9 8 5 6 8 7 5 610 6 5 9 9 77 64
0110 8 6 8 9 5 3 6 9 9 6 910 6 5 710 65 42
1110 5 6 710 4 6 010 910 910 6 7 810 73
01 9 0 4 610 5 3 310 810 51010 9 910 71 37
:

```

Now we will take a look at `statread.sas`. It contains just the `proc format` and the `data` step; There are no statistical procedures. This file will be read by programs that invoke statistical procedures, as you will see.

```

                                /* statread.sas
Read the statclass data in fixed format, define and label variables. Use
with      %include '/folders/myfolders/statread.sas';    */

title 'Grades from STA3000 at Roosevelt University:  Fall, 1957';

proc format; /* Used to label values of the categorical variables */
    value sexfmt    0 = 'Male'    1 = 'Female';
    value ethfmt    1 = 'Chinese'
                  2 = 'European'
                  3 = 'Other' ;

data grades;
    infile '/folders/myfolders/statclass2.data' missover;
    input (sex ethnic) (1.)
          (quiz1-quiz8 comp1-comp9) (2.)
          (midterm final) (3.);
    /* Drop lowest score for quiz & computer */
    quizave = ( sum(of quiz1-quiz8) - min(of quiz1-quiz8) ) / 7;
    compave = ( sum(of comp1-comp9) - min(of comp1-comp9) ) / 8;
    label ethnic = 'Apparent ethnic background (ancestry)'
           quizave = 'Quiz Average (drop lowest)'
           compave = 'Computer Average (drop lowest)';
    mark = .3*quizave*10 + .1*compave*10 + .3*midterm + .3*final;
    label mark = 'Final Mark';
    diff = quiz8-quiz1; /* To illustrate matched t-test */
    label diff = 'Quiz 8 minus Quiz 1';
    mark2 = round(mark);
    /* Bump up at grade boundaries */
    if mark2=89 then mark2=90;
    if mark2=79 then mark2=80;
    if mark2=69 then mark2=70;
    if mark2=59 then mark2=60;
    /* Assign letter grade */
    if mark2=. then grade='Incomplete';
    else if mark2 ge 90 then grade = 'A';
    else if 80 le mark2 le 89 then grade='B';
    else if 70 le mark2 le 79 then grade='C';
    else if 60 le mark2 le 69 then grade='D';

```

```

        else grade='F';
    format sex sexfmt.;          /* Associates sex & ethnic */
    format ethnic ethfmt.;      /* with formats defined above */

/*****/

```

The data step in `statread.sas` differs from the one in `statmarks1.sas` in only two respects. First, the `missover` option on the `infile` statement causes blanks to be read as missing values even if they occur at the end of a line and the line just ends rather than being filled in with space characters. That is, such lines are shorter than the others in the file, and when SAS *over-*reads the end of the line, it sets all the variables it would have read to missing. This is what we want, so you should always use the `missover` option when missing values are represented by blanks.

The other difference between this data step and the one in `statmarks1.sas` is in the `input` statement. Here, we are using *formatted* input. `sex` and `ethnic` each occupy 1 column. `quiz1-quiz8` and `comp1-comp9` each occupy 2 columns. `midterm` and `final` each occupy 3 columns. You can supply a list of formats for each list of variables in parentheses, but if the number of formats is less than the number of variables, they are re-used. That's what's happening in the present case. It is also possible to specify the exact column location in which each variable resides. The `input` statement is very rich and powerful.

The program `statread.sas` reads and defines the data, but it requests no statistical output; `statdescribe.sas` pulls in `statread.sas` using a `%include` statement, and produces basic descriptive statistics. Significance tests would be produced by other short programs.

Keeping the data definition in a separate file and using `%include` (the only part of the powerful *SAS macro language* presented here) is often a good strategy, because most data analysis projects involve a substantial number of statistical procedures. It is common to have maybe twenty program files that carry out various analyses. You *could* have the data step at the beginning of each program, but in many cases the data step is long. And, what happens when (inevitably) you want to make a change in the data step and re-run your analyses? You find yourself making the same change in twenty files. Probably you will forget to change some of them, and the result is a big mess. If you keep your data definition in just one place, you only have to edit it once, and a lot of problems are avoided.

```

                                /* statdescribe.sas */
%include '/folders/myfolders/statread.sas';
title2 'Basic Descriptive Statistics';

proc freq;
    title3 'Frequency distributions of the categorical variables';
    tables sex ethnic grade;

```

```

proc means n mean std;
  title3 'Means and SDs of quantitative variables';
  var quiz1 -- mark2;          /* single dash only works with numbered
                               lists, like quiz1-quiz8    */
proc univariate normal; /* the normal option gives a test for normality */
  title3 'Detailed look at mark and bumped mark (mark2)';
  var mark mark2;

```

## 2.5 SAS Example Three: The Math data

The Math data come from a large multi-campus North American university. These are real data, and a fairly complete analysis will be spread throughout parts of this book. The objective is to illustrate some principles of data analysis that have practical importance, but are not exactly part of Statistics.

The Math study came about because some professors and administrators at one of the campuses wanted to predict performance in first-year calculus so they could give better advice to students. For this purpose, one of the professors made up a 20-question multiple choice test; nine questions were on pre-calculus material, and eleven questions were based on the local curriculum in high school calculus. The main question was whether this diagnostic test was useful. That is, if you knew what courses the students took in high school and how well they did, would your predictions be more accurate if you also had their scores on the diagnostic test? And is so, *how much* more accurate would the predictions be?

To find out, all the students who signed up for first-year calculus at one of the campuses were asked to take the diagnostic test in the week before classes started. Most of them (a total of ) did so. At the end of the school year their calculus marks were recorded. This this mark, a number from zero to one hundred, was the main dependent variable.

But of course not all students remained in the class; some withdrew, and some disappeared in other ways. The reasons for their disappearance were varied, and not part of the data set. Obviously, predictions of numerical grade can only be based on students who stayed in the course until the end, and any advice given to students about marks would have to start out with something like “Assuming you stay in the course until the end, our best guess of your mark is . . .” So a second, very important response variable was simply whether the student passed the course, Yes or No. Another potentially useful possibility would be Pass-Fail-Disappear, a categorical response variable with three categories.

The diagnostic test provides at least two explanatory variables: number of pre-calculus questions correct, and number of calculus questions correct. In addition, high school transcripts were available. It is important to recognize that the information in these transcripts was not in a form that could be used directly in statistical analysis. Each transcript was a sizable plain text file — actually, the disk image of old fashioned line printer output, designed to be printed on big sheets of paper 132 characters wide. There was a cumulative high school grade point average for most students, and also a mark

in an upper level high school English course because it was required for admission to the university. In addition, most students in the sample had taken high school Calculus. Beyond that, they had mostly taken different courses from one another, including similar courses with names that were quite different, and different courses with names that were quite similar. Courses were listed in the order taken. Some students had withdrawn from certain courses more than once before completing them for credit, and some took the same course for credit more than once in an attempt to improve their mark. The second mark was usually higher, but not always.

The point of all this is that while eventually we will analyze a nice orderly data file with rows corresponding to cases and columns corresponding to variables, data do not naturally come that way, most of the time. As mentioned in Data Analysis Hint 1 on page 10, the row-by-column arrangement is something that is imposed on the data by the researchers who gather or analyze the data.

Typically, this process involves a large number of semi-arbitrary but critically important decisions. In the math study, the number of variables that *might* have been extracted from the high school transcripts is difficult even to estimate. For example, *number* of math courses taken was an obvious possibility, but it was eliminated on the basis of preliminary analysis. Many other choices were made, and the details are largely undocumented and forgotten<sup>4</sup>. In the end, the following variables were recorded for each student who took the diagnostic test.

- 
- 
- 
- 

## 2.6 SAS Reference Materials

This course is trying to teach you SAS by example, without full explanation, and certainly without discussion of all the options. If you need more detail, the SAS Institute provides online documentation at <http://support.sas.com/documentation>. Most of the standard statistical procedures you are likely to use are under “SAS/STAT.” For information about the data step (for example, reading a complex data set), choose “Base SAS Software” and then either “SAS Language Reference: Concepts” or “SAS Language Reference: Dictionary.” The SAS Institute also publishes hard copy manuals, but most students will prefer the online version.

Note that this is reference material. The SAS Institute also publishes a variety of manual-like books that are intended to be more instructional, most of them geared to

---

<sup>4</sup>This may be too bad, but it is typical of most research. On the positive side, it will be described shortly how the data were randomly divided into two sub-samples, an exploratory sample and a confirmatory sample. All the semi-arbitrary decisions were based on the exploratory sample *only*

specific statistical topics (like *The SAS system for multiple regression* and *The SAS system for linear models*). These are more readable than the reference manuals, though it helps to have a real textbook on the topic to fill in the gaps.

A better place to start learning about SAS is a wonderful book by Cody and Smith [5] entitled *Applied statistics and the SAS programming language*. They do a really good job of presenting and documenting the language of the data step, and they also cover a set of statistical procedures ranging from elementary to moderately advanced. If you had to own just one SAS book, this would be it.

If you consult *any* SAS book or manual, you'll need to translate and filter out some details. Here is the main case. Many of the examples you see in Cody and Smith's book and elsewhere will not have separate files for the raw data and the program. They include the raw data in the program file in the data step, after a `datalines` or `cards` statement. Here is an example from page 3 of [5].

```
data test;
  input subject 1-2 gender $ 4 exam1 6-8 exam2 10-12 hwgrade $ 14;
  datalines;
10 M 80 84 A
 7 M 85 89 A
 4 F 90 86 B
20 M 82 85 B
25 F 94 94 A
14 F 88 84 C
;
proc means data=test;
run;
```

Having the raw data and the SAS code together in one display is so attractive for small datasets that most textbook writers cannot resist it. But think how unpleasant it would be if you had 10,000 lines of data. The way we would do this example is to have the data file (named, say, `example1.dat`) in a separate file. The data file would look like this.

```
10 M 80 84 A
 7 M 85 89 A
 4 F 90 86 B
20 M 82 85 B
25 F 94 94 A
14 F 88 84 C
```

and the program file would look like this.

```
data test;
```

```
infile '/folders/myfolders/example1.dat'; /* Read data from example1.dat */
input subject 1-2 gender $ 4 Exam1 6-8 exam2 10-12 hwgrade $ 14;
proc means data=test;
```

Using this as an example, you should be able to translate any textbook example into the program-file data-file format used in this book.