2.2 Introduction to SAS

SAS stands for "Statistical Analysis System." Even though it runs on PCs and Macs as well as on bigger computers, it is truly the last of the great old mainframe statistical packages. The first beta release was in 1971, and the SAS Institute, Inc. was spun off from North Carolina State University 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 is beyond any question. For the purposes of this class, 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 has not really worked. It's as if someone painted an eighteen-wheel transport truck yellow, and called it a school bus. Yes, you can take the children to school in that thing, but would you want to?

2.2.1 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 called `drp.dat`.

- **The Program File**: This is also sometimes called a "command file," because it's usually not much of a program. It consists of commands that the SAS software tries to follow. You create this file with a text editor like `pico` or `emacs`. The command 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 command file is called `reading.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 of 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 combines the first part of the command file's name with the extension `.sas`. So for example, when SAS executes the commands in `reading.sas`, it writes a log file named `reading.log`.

- **The List File**: The list file contains the output of the statistical procedures requested by the command file. The list file has the extension `.lst` — so, for example, running SAS on the command file `reading.sas` will produce `reading.lst` as well as `reading.log`. A successful SAS run will almost always produce a list file. The absence of a list file indicates that there was at least one fatal error. The presence of a list 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 list file will usually not contain any error messages; they will be in the log file.

2.2.2 Running SAS from the Command Line

There are several ways to run SAS. We will run SAS from the unix command line. In my view, this way is simplest and best.

If, by accident or on purpose, you type SAS without a filename, then SAS assumes you want to initiate an interactive session, and it tries to start the SAS Display Manager. If you are logged in through an ordinary telnet or ssh session, SAS terminates with an error: `ERROR: Cannot open X display. Check display name/server access authorization.` SAS assumes you are using the unix X-window graphical interface, so it will not work if your computer is emulating a (semi) dumb terminal. If you are in an X-window session, after a while several windows will open up. The only suggestion I have is this: Make sure the SAS Program Editor window is selected. From the File menu, choose Exit. Whew.

If you choose to ignore this advice and actually try to use the Display Manager, you are on your own. You will have my sympathy, but not my help. The joke about painting the transport truck yellow applies, and the joke is on you.

The following illustrates a simple SAS run from the command line. Initially, there are only two files in the (sub)directory — `reading.sas` (the program file) and `drp.dat` (the raw data file). The command `sas reading` produces two additional files — `reading.log` and `reading.lst`. In this and other examples, the unix prompt is `tuzo.erin` (the name of the unix machine used to produce the examples), followed by a `>` sign.

```
tuzo.erin > ls
drp.dat         reading.sas
tuzo.erin > sas reading
tuzo.erin > ls
drp.dat         reading.log     reading.lst     reading.sas
```

2.2.3 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 the 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 set**, a file consisting of the data (after modifications and additions), labels, and so on. Statistical procedures operate on SAS data sets, so you must create a SAS data set before you can start computing any statistics.

A SAS data set 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 sets were always 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 set 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 set on disk for later use. We won't do this much (there will be just one example), 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 set. 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.

To repeat, SAS data *steps* and SAS data *sets* sound similar, but they are distinct concepts. A SAS data *step* is part of a SAS program; it generates a SAS data *set*, which is a file – usually a temporary file.

SAS data sets are not always created by SAS data steps. Some statistical procedures can create SAS data sets, too. For example, `proc univariate` can take an ordinary SAS data set as input, and produce an output data set 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 set 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 main exception is `proc format`, which is used to provide labels for the values of categorical independent variables. The proc step is where you specify a statistical procedure that you want to carry out. A statistical procedures in the proc step will take a SAS data sets as input, and write the results (summary statistics, values of test statistics, p-values, and so on) to the list 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.

2.2.4 A First Example: `reading.sas`

Earlier, we ran SAS on the file `reading.sas`, producing `reading.log` and `reading.lst`. Now we will look at `reading.sas` in some detail. This program is very simple; it has just one data step and one proc step. More details will be given later, but it's based on a study in which one group of grade school students received a special reading programme, and a control group did not. After a couple of months, all students were given a reading test. We're just going to do an independent groups t-test, but first take a look at the raw data file. You'd do this with the unix `more` command.

Actually, it's so obvious that you should look at your data that nobody ever says it. But experienced data analysts always do it — or else they assume everything is okay and get a bitter lesson in something they already knew. It's so important that it gets the formal status of a **data analysis hint**.

Data Analysis Hint 1 *Always look at your raw data file. It the data file is big, do it anyway. At least page through it a screen at a time, 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.*

Anyway, here is the file `drp.dat`, with the middle cut out to save space.

Treatment	24
Treatment	43
Treatment	58
⋮	⋮
Control	55
Control	28
Control	48
⋮	⋮

Now we can look at `reading.sas`.

```
/****************** reading.sas *********************
*  Simple SAS job to illustrate a two-sample t-test    *
*******************************************************/

options linesize=79 noovp formdlim='_';
title 'More & McCabe (1993) textbook t-test Example 7.8';

data reading;
     infile 'drp.dat';
     input group $ score;
     label group = 'Get Directed Reading Programme?'
           score = 'Degree of Reading Power Test Score';
proc ttest;
     class group;
     var score;
```

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

- The first three lines are 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 `reading.sas`, leaving off the `*/` (or typing
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 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 (for example, see the `label` statement). Spacing, indentation, breaking up s statement into several lines of text – these are all for the convenience of the human reader, and are not part of the SAS syntax.

- 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 tried 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 don't waste time trying to figure out the others.* Some or all of them may well disappear.

- `options linesize=79 noovp formdlim='_';`

  These options are highly recommended. The `linesize=79` option is so highly recommended it's almost obligatory. It causes SAS to write the output 79 columns across, so it can be read on an ordinary terminal screen that's 80 characters across. You specify an output width of 79 characters rather than 80, because SAS uses one column for printer control characters, like page ejects (form feeds).

  If you do not specify `options linesize=79;`, SAS will use its default of 132 characters across, the width of sheet of paper from an obsolete line printer you probably have never seen. Why would the SAS Institute hang on to this default, when changing it to match ordinary letter paper would be so easy? It probably tells you something about the computing environments of some of SAS's large corporate clients.

- The `noovp` option makes the log files more readable if you have errors. When SAS finds an error in your program, it tries to *underline* the word that caused the error. It does this by going back and *overprinting* the offending word with a series of "underscores" (_ characters). On many printers this works, but when you try to look at the log file on a terminal screen (one that is *not* controlled by the SAS Display Manager), what often appears is a mess. The `noovp` option specifies `no overprinting`. It causes the "underlining" to appear on a separate line under the program line with the error. If you're running SAS from the unix command line and looking at your log files with the `more` command (or the `less` command or the `cat` commmand), you will probably find the `noovp` option to be helpful.

- The `formdlim='_'` option specifies a "form delimiter" to replace most form feeds (new physical pages) in the list file. This can save a lot of paper (and page printing charges). You can use any string you want for a form delimiter. The underscore (the one specified here) causes a solid line to be printed instead of going to a new sheet of paper.

- `title` This is optional, but recommended. The material between the single 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 reading;` This begins the data step, specifying the name of the SAS data set that is being created.

- `infile` Specifies the name of the raw data file. The file name, enclosed in single quotes, can be the full unix path to the file, like `/dos/brunner/public/senic.raw`. If you just give the name of the raw data file, as in this example, SAS assumes that the file is in the same directory as the command file.

- `input` Gives the names of the variables.

  - A character variable (the values of `group` are "Treatment' and "Control") must be followed by a dollar sign.
  - Variable names must be eight characters or less, and should begin with a letter. They will be used to request statistical procedures in the `proc` step. They should be meaningful (related to what the variable *is*), and easy to remember.
  - This is almost the simplest 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.

- `label` Provide descriptive labels for the variables; these will be used to label the output, usually in very nice way. Labels can be quite useful, especially when you're trying to recover what you did a while ago. Notice how this statement extends over two physical lines.

- `proc ttest;` Now the proc step begins. This program has only one data step and one proc step. We are requesting a two-sample t-test.

- `class` Specifies the independent variable.
- `var` Specifies the dependent variable(s). You can give a list of dependent variables. A separate univariate test (actually, as you will see, *collection* of tests is performed for each dependent variable.

reading.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, `reading.sas`. If there were syntax errors in the program, this is where the error messages would appear.

```
tuzo.erin > cat reading.log
1                                                       The SAS System                              11:08 Friday, January 2,

NOTE: Copyright (c) 1989-1996 by SAS Institute Inc., Cary, NC, USA.
NOTE: SAS (r) Proprietary Software Release 6.12  TS020
      Licensed to UNIVERSITY OF TORONTO/COMPUTING & COMMUNICATIONS, Site 0008987001.



This message is contained in the SAS news file, and is presented upon
initialization.  Edit the files "news" in the "misc/base" directory to
display site-specific news and information in the program log.
The command line option "-nonews" will prevent this display.



NOTE: AUTOEXEC processing beginning; file is /local/sas612/autoexec.sas.

NOTE: SAS initialization used:
      real time           0.780 seconds
      cpu time            0.152 seconds


NOTE: AUTOEXEC processing completed.

1          /****************** reading.sas *********************
2          *  Simple SAS job to illustrate a two-sample t-test    *
3          ******************************************************/
4
5          options linesize=79 noovp formdlim='_';
6          title 'More & McCabe (1993) textbook t-test Example 7.8';
7          data reading;
8               infile 'drp.dat';
9               input group $ score;
10              label group = 'Get Directed Reading Programme?'
11                    score = 'Degree of Reading Power Test Score';

NOTE: The infile 'drp.dat' is:
      File Name=/res/jbrunner/442s04/notesSAS/drp.dat,
      Owner Name=jbrunner,Group Name=research,
      Access Permission=rw-------,
      File Size (bytes)=660

NOTE: 44 records were read from the infile 'drp.dat'.
      The minimum record length was 14.
      The maximum record length was 14.
NOTE: The data set WORK.READING has 44 observations and 2 variables.
NOTE: DATA statement used:
      real time           0.190 seconds
      cpu time            0.051 seconds


12         proc ttest;
```

```
13              class group;
14              var score;
NOTE: The PROCEDURE TTEST printed page 1.
NOTE: PROCEDURE TTEST used:
      real time           0.030 seconds
      cpu time            0.009 seconds


2                               The SAS System    11:08 Friday, January 2, 2004


NOTE: The SAS System used:
      real time           1.120 seconds
      cpu time            0.233 seconds

NOTE: SAS Institute Inc., SAS Campus Drive, Cary, NC USA 27513-2414
```

reading.lst Here is the list file. Notice that the title specified in the `title` statement appears at the top, along with the time and date the program was executed. Then we get means and standard deviations, and several statistical tests — including the one we wanted. We get other stuff too, whether we want it or not. This is typical of SAS, and most other mainstream statistical packages as well. The default output from any given statistical procedures will contain more information than you wanted, and probably some stuff you don't understand at all. There are usually numerous options that can add *more* information, but almost never options to reduce the default output. So, you just learn what to ignore. It is helpful, but not essential,to have at least a superficial understanding of everything in the default output from procedures you use a lot.

```
---------------------------------------------------------------------------

               More & McCabe (1993) textbook t-test Example 7.8              1
                                          11:08 Friday, January 2, 2004

                                TTEST PROCEDURE

Variable: SCORE        Degree of Reading Power Test Score

GROUP          N                 Mean              Std Dev            Std Error
---------------------------------------------------------------------------
Control       23          41.52173913          17.14873323          3.57575806
Treatmen      21          51.47619048          11.00735685          2.40200219

Variances        T       DF    Prob>|T|
---------------------------------------
Unequal    -2.3109     37.9      0.0264
Equal      -2.2666     42.0      0.0286

For H0: Variances are equal, F' = 2.43    DF = (22,20)    Prob>F' = 0.0507
```

Now here are some comments about `reading.lst`.

- `Variable: SCORE` This tells you what the dependent variable is – particularly useful if you have more than one. Notice the nice use of the variable label that was supplied in the `label` statement.
- `GROUP` The independent variable. Underneath are the values of the independent variable. We also have the sample size n for each group, and the group mean, standard deviation, and also the standard error or the mean ($\frac{s}{\sqrt{n}}$, the estimated standard deviation of the sampling distribution of the sample mean).

- Well actually, if you look carefully, you see that we do *not* quite get the values of the independent variable under `GROUP`. The values of the (alphanumeric, or character-valued) variable `group` are `Control` and `Treatment`, but the printout says "`Treatmen`." This is not a printing error; it is a subtle error in the reading of the data. The default length of an alphanumeric data value is 8 characters, but "Treatment" has 9 characters. So SAS just read the first eight. No error message was generated and no harm was done in this case, but in other circumstances this error can turn a data file into a giant pile of trash, without warning. Later we will see how to override the default and read longer strings if necessary.

- Next we get a table whose first column is entitled "Variances." This gives t statistics for testing equality of means, which was what we are interested in. The traditional t-test assumes equal variances, and it is given in the column entitled "Equal."

  - The value of the test statistic is `-2.2666`.
  - The degrees of freedom $n_1 + n_2 - 2$ is given in the `DF` column.
  - The column `Prob>|T|` gives the two-tailed (two-sided) p-value. It is less than the traditional value of 0.05, so the results are statistically significant.

Sample Question 2.2.1 *What do we conclude from this study? Say something about reading, using non-technical language.*

Answer to Sample Question 2.2.1 *Students who received the Directed Reading Program got higher average reading scores than students in the control condition.*

It's worth emphasizing here that the main objective of doing a statistical analysis is to draw conclusions about the data — or to refrain from drawing such conclusions, for good reasons. The question "What do we conclude from this study?" will always be asked. The right answer will always be either "Nothing; the results were not statistically significant," or else it will be something about reading, or fish, or potatoes, or AIDS, or whatever is being studied. Many students, even when they have been warned, respond with a barrage of statistical terminology. They go on and on about the null hypothesis and Type I error, and usually say nothing that would tell a reasonable person what actually happened in the study. In the working world, a memo filled with such garbage could get you fired. Here, it will get you a zero for the question, even if the technical details you give are correct.

Remember, the purpose of writing up a statistical analysis is not to sound impressive and technical, but to impart information. To say things in a simple way is a virtue. It shows you understand what is going on. Now back to the printout.

- The row entitled "Unequal" gives a sort of t-test that does not assume equal variances. Well, it's not really a t-test, because the test statistic does not really have a t distribution, even when the data are exactly normal. But, the (very unpleasant) distribution of the test statistic is well approximated by a t distribution with the right degrees of freedom — not $n_1 + n_2 - 2$, but something messy that depends on the data. See the odd fractional degrees of freedom? See [2] for details. In any case, it does not matter much in this case, because the p-value is almost the same as the

p-value from the traditional test. They lead to the same conclusions, and there is no problem. What should you do when they disagree? I'd go with the test that makes fewer assumptions.

- Next we see `For H0: Variances are equal` and an F-test. This is the traditional test for whether the variances of two groups are equal, and it's *almost* significant. This test is provided so people can test for differences between variances; if it is significantly different they can use the unequal variance t-test, and otherwise they can use the traditional test. This seems reasonable, except for the following.

  Both the two-sample t-test and the F-test for equality of variances assume that the data are normally distributed. However, the normality assumption does not matter much for the t-test when the sample sizes are large, while for the variance test it matters a *lot*, regardless of how much data you have. When the data are non-normal, the test for variances will be significant more than 5% of the time even when the population variances are equal. If you have equal population variances and a large sample of non-normal data, the F-test for variances could easily be significant, leading you to worry unnecessarily about the validity of the t-test.

2.2.5 Background of the First Example

We don't do statistical analysis in a vacuum. Before proceeding with more computing details, let's find out more about the reading data. This first example is from an introductory text. It's Example 7.8 (p. 534) in More and McCabe's excellent *Introduction to the practice of statistics* [2]. We are interested in analyzing *real* data, not in doing textbook exercises. But we will not turn up our noses just yet, because

Data Analysis Hint 2 *When learning how to carry out a procedure using unfamiliar statistical software, always do a textbook example first, and compare the output to the material in the text. Regardless of what the manual might say, never assume you know what the software is doing until you see an example.*

More and McCabe do a great job of explaining the t-test with unequal variances, something SAS produces (along with usual t-test that assumes equal variances) without being asked when you request a t-test. Besides, the data actually come from someone's Ph.D. thesis, so there is an element of realism. Here is Moore and McCabe's description of the study.

> An educator believes that new directed reading activities in the classroom will help elementary school pupils improve some aspects of their reading ability. She arranges for a third grade class of 21 students to take part in these activities. A control classroom of 23 third graders follows the same curriculum without the activities. At the end of 8 weeks, all students are given a Degree of Reading Power (DRP) test, which measures the aspects of reading ability that the program is designed to improve.

Sample Question 2.2.2 *What's wrong with this study?*