Chapter 2

First set of tools: SAS running under unix (including linux)

The SAS language is the same regardless of what hardware you use or what operating system is running on the hardware. SAS programs are simple text files that can be transported from one machine to another with minimal difficulty. In this course, everything will be illustrated with SAS running under the unix operating system, but it's not a problem even if the next place you go only has PCs. The adjustment to SAS-PC should be fast and fairly painless.

2.1 Unix

Unix is a line-oriented operating system. Well, there's X-windows (a graphical shell that runs on top of unix), but we won't bother with it. Basically, you type a command, press Enter, and unix does something for (or to) you. It may help to think of unix as DOS on steroids, if you remember DOS. The table below has all the unix commands you will need for this course. Throughout, *fname* stands for the name of a file.

A Minimal Set of unix Commands

exit Logs you off the system: ALWAYS log off before leaving!

passwd Lets you change your password. Recommended.

man *command name* Online help: explains *command name*, (like `man sort`).

ls Lists names of the files in your directory.

less *fname* Displays fname on screen, one page at a time. Spacebar for next page, `q` to quit.

lpr *fname* Prints hard copy on a laser printer. lpr stands for line printer. These physical devices no longer exist in most installations.

rm *fname* Removes *fname*, erasing it forever.

cp *fname1 fname2* Makes a copy of *fname1*. The new copy is named *fname2*.

mv *fname1 fname2* Moves (renames) *fname1*

emacs *fname* Starts the `emacs` text editor, editing *fname* (can be new file).

R Gets you into the `R` implementation of the S environment.

sas *fname* Executes `SAS` commands in the file ***fname*`.sas`**, yielding ***fname*`.log`** and (if no fatal errors) ***fname*`.lst`**.

ps Shows active processes

kill -9 # Kills process (job) number `#`. Sometimes you must do this when you can't log off because there are stopped jobs. Use `ps` to see the job numbers.

mail yourname@yourisp.com < *fname* Email a file to yourself. Very handy for getting files to your home computer for printing.

curl *URL* > *fname* A *URL* is a Web address. This command is intended to help you get a copy of the source code of Web pages. But when the web page contains just a data file, as it sometimes does in this course, this is a great way to get a copy of the data. Copy the *URL* from your browser, like this.

```
curl http://fisher.utstat.toronto.edu/~brunner/429f07/code_n_data/drp.dat > drp.dat
```

This really is a minimal set of commands. The unix operating system is extremely powerful, and has an enormous number of commands. You can't really see the power from the minimal set above, but you can see the main drawback from the standpoint of the new user. Commands tend to be terse, consisting of just a few keystrokes. They make sense once you are familiar with them (like `ls` for listing the files in a directory, or `rm` for remove), but they are hard to guess. The `man` command (short for manual) gives very

accurate information, but you have to know the name of the command before you can use `man` to find out about it.

Just for future reference, here are a few more commands that you may find useful, or otherwise appealing.

A Few More unix Commands

mkdir *dirname* Makes a new sub-directory (like a folder) named *dirname*. You can have sub-directories within sub-directories; it's a good way to organize your work.

cp *fname dirname* Copies the file *fname* into the directory *dirname*.

cd *dirname* Short for Change Directory. Takes you to the sub-directory *dirname*.

cd .. Moves you up a directory level.

cd Moves you to your main directory from wherever you are.

ls > *fname* Sends the output of the `ls` command to the file *fname* instead of to the screen.

cat *fname* Lists the whole file on your screen, not one page at a time. It goes by very fast, but usually you can scroll back up to see the entire file, if it's not too long.

cat *fname1 fname2* > *fname3* Concatenates *fname1* and *fname2* (sticks them together) and re-directs the output to *fname3*

grep ERROR cartoon1.log Searches for the string ERROR in the file cartoon1.log. Echos each line containing the string. Silent if ERROR does not occur. Case sensitive.

alias chk "grep ERROR *.log ; grep WARN *.log" Makes a new command called `chk`. It checks for the string ERROR and the string WARN in any log file.

cal Displays a calendar for this month

cal 1 3002 Displays a calendar for January 3002.

unset noclobber Are you tired of being asked if you really want to remove or overwrite a file?

rm *fname1 fname2* Remove both

rm -f *fname* Remove without asking for confirmation, this time only.

alias rm "rm -f" `rm` now means `rm -f`.

rm -r *dirname* Remove the directory, and everything in it recursively.

R –vanilla < *fname1* > *fname2* Execute the S language commands in *fname1*, sending output to *fname2*. Run in "plain vanilla" mode.

Printing files at home This is a question that always comes up. Almost surely, the printer connected to your printer at home is not directly connected to the university network. If you want to do something like print your SAS output at home, you have to transfer the file on the unix machine to the hard drive of your home computer, and print it from there. One way is to use some kind of `sftp` (secure file transfer protocol) tool to get the file in question onto your hard drive. For short files, you can also use the `less` or `cat` command to list the file on your screen, select it with your mouse, copy it, paste it to a word processing document, and print it from there. It is a good idea to use a fixed-width font like Courier, and not the Times or Times Roman font. Everything will be lined up better.

Perhaps easiest of all is to email yourself the file. This is illustrated in the first set of unix commands. To repeat, `mail yourname@yourisp.com < fname`.

2.2 Introduction to SAS

SAS stands for "Statistical Analysis System." Even though it runs on PCs 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 the interface still basically file and text oriented, not graphical.

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 `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 `reading1.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 `.log`. So for example, when SAS executes the commands in `reading1.sas`, it writes a log file named `reading1.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 `reading1.sas` will produce `reading1.lst` as well as `reading1.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. In this text, all the examples will be run from the unix command line. In my view, this way is simplest and also the best way to start. Also, it is by far the easiest way to use SAS from home, assuming that SAS is running on a remote server and not your home computer.

The following illustrates a simple SAS run. The unix prompt is `YesMaster >`, indicating that unix is waiting for a command. You unix prompt will probably be different. Initially, there are just two files in the directory, the program file `reading1.sas` and the raw data file `drp.dat`. First we see what files we have with `ls`, then run SAS, and then `ls` agaain to see that SAS has created two additional files.

```
YesMaster > ls
drp.dat        reading1.sas
YesMaster > sas reading1
YesMaster > ls
drp.dat        reading1.log   reading1.lst   reading1.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 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 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 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 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 standard` 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 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 procedures in the `proc` step will take a SAS data set 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: `reading1.sas`

Earlier, we ran SAS on the file `reading1.sas`, producing `reading1.log` and `reading1.lst`. Now we will look at `reading1.sas` in some detail. This program is very simple; it has just one data step and two proc steps. 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 get basic descriptive statistics (not even a t-test), but first take a look at the raw data file. You'd do this with the unix `less` command.

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 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 and end cut out to save space.

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

Now we can look at `reading1.sas`.

```
/****************** reading1.sas *********************/

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 freq;
     tables group;
proc means;
     var score;
```

Here are some detailed comments about `reading1.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 `reading1.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.

- 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 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 `less` command or the `cat` command, 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 that the name of the SAS data set being created is "reading." The names of data sets are arbitrary, but you should make them informative.

- `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 remember what you did a while ago. Notice how this statement extends over two physical lines.

- `proc freq;` Now the first `proc` step begins. We want a frequency distribution of `group`, to see how many students are in each group. It is always a good idea to look at frequency distributions of your categorical variables, including quantitative variables taking on just a few values.

- `tables` is obligatory. It is followed by a list of variables for which you want to see tables. Proc freq can also give you *joint* frequency distributions, or cross-tabulations, along with chisquare tests for association between categorical variables.

- `proc means;` This is the second `proc` step. We want the mean, standard deviation and so on for reading score. It is always a good idea to look at means and standard deviations of your quantitative variables. By default, `proc means` gives you the minimim and maximum too, which 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.

reading1.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, `reading1.sas`. If there were syntax errors in the program, this is where the error messages would appear. The `less` command lets you look at a file one page at a time. Press the space bar for the next page, or `q` to quit.

```
YesMaster > less reading1.log
1                                                          The SAS System
                    09:27 Saturday, October 27, 2057

NOTE: Copyright (c) 1999-2001 by SAS Institute Inc., Cary, NC, USA.
NOTE: SAS (r) Proprietary Software Release 8.2 (TS2M0)
      Licensed to UNIVERSITY OF TORONTO/COMPUTING & COMMUNICATIONS, Site 0008987
001.
NOTE: This session is executing on the SunOS 5.9 platform.



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: SAS initialization used:
      real time           0.02 seconds
      cpu time            0.03 seconds

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

NOTE: The infile 'drp.dat' is:
      File Name=/u/brunner/442s08/text/drp.dat,
```

```
      Owner Name=brunner,Group Name=dos,
      Access Permission=rw-r--r--,
      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           1.44 seconds
      cpu time            0.02 seconds


11         proc freq;
12              tables group;

NOTE: There were 44 observations read from the data set WORK.READING.
NOTE: The PROCEDURE FREQ printed page 1.
NOTE: PROCEDURE FREQ used:
      real time           1.50 seconds
      cpu time            0.02 seconds


13         proc means;
14              var score;
15
^L2                                The SAS System

NOTE: There were 44 observations read from the data set WORK.READING.
NOTE: The PROCEDURE MEANS printed page 2.
NOTE: PROCEDURE MEANS used:
      real time           0.22 seconds
      cpu time            0.02 seconds


NOTE: SAS Institute Inc., SAS Campus Drive, Cary, NC USA 27513-2414
NOTE: The SAS System used:
      real time           4.79 seconds
      cpu time            0.09 seconds
```

reading1.lst Here is the list file. Notice that the title specified in the `title` statement appears at the top. Then we get statistical output — in this case, the frequency distribution and table of means *etc.*.

```
YesMaster > less reading1.lst

---------------------------------------------------------------------------

              More & McCabe (1993) textbook t-test Example 7.8              1

                            The FREQ Procedure

                      Get Directed Reading Programme?

                                                 Cumulative    Cumulative
         group       Frequency     Percent     Frequency      Percent
         ------------------------------------------------------------
         Control            23       52.27            23        52.27
         Treatmen           21       47.73            44       100.00

---------------------------------------------------------------------------

              More & McCabe (1993) textbook t-test Example 7.8              2

                            The MEANS Procedure

          Analysis Variable : score Degree of Reading Power Test Score

        N            Mean         Std Dev         Minimum         Maximum
       ---------------------------------------------------------------------
       44      46.2727273      15.2351546      10.0000000      85.0000000
       ---------------------------------------------------------------------
YesMaster >
YesMaster > mail jerry@cia.gov < reading1.log
YesMaster > mail jerry@cia.gov < reading1.lst
YesMaster > exit
```

The output is pretty self-explanatory, except the last bit. Once you finish running a SAS job, your log and list files reside on a hard drive attached to unix machine, not your home computer. If you are using a computer in a computer lab on campus, you can probably print to a printer in the lab with the `lpr` command, like `lpr reading1.lst`. `lpr` is short for line printer; you have never seen one.

To print from home, it is easiest to email yourself a copy of the files you want to print. At the unix prompt, type `mail`, then your email address, then a $<$ sign, and then the name of the file you want to mail. The less than sign is unix redirection. It says send the file that way – that is, use the file as input to the mail command.

Where did the data file and the program file come from? In the preceding example, we started with `drp.dat` and `reading1.sas` already in existence. But they didn't come from nowhere. Initially, there is nothing at all in the directory. Your first job is to create a copy of the raw data file. One way is to type it in, but we will assume that the data file already exists, and you just need to upload it. In this course, all the data files will be on the Web, and you will use the unix `curl` command as illustrated below.

Curl is a cute way to "C" the "URL." Get the unix prompt in one window (probably using `PuTTY`), and open a Web browser in another window. Navigate to the data file, and then select the entire Web address (URL). Choose Copy from the Edit menu, and then click on the window with the unix prompt.

At the unix prompt, you type `curl`, then a space, then paste in the URL. In `PuTTY`, you paste by clicking the right mouse button. Then you type another space, a greater than sign (>) and the name of the new file you want to create, which is usually just the name of the data file, like `poverty.data`. Then press Enter, and in a flash the data file is in your directory. This process is illustrated below.

```
YesMaster > ls
YesMaster > curl http://fisher.utstat.toronto.edu/~brunner/429f07/code_n_data/
text/drp.dat > drp.dat

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   660  100   660    0     0  85859      0 --:--:-- --:--:-- --:--:--     0

YesMaster > ls
drp.dat
```

The URL was so long that the `curl` command wrapped to the next line. This usually happens; just keep typing. Once you have your data file, take a look at it a page at a time by typing `less drp.dat`. Space bar for another page, `q` for quit. We'll examine the data file in the next section.

Next one needs to create a SAS program by typing it in. The `emacs` text editor is recommended. At the unix prompt, you type `emacs`, then a space, then the name of a file. You will go into a full-screen editing mode. It's somewhat like Notepad in Windows, but much more powerful and with no mouse. If the file already exists in the directory where you are, you are editing that file. If no file of that name exists yet, you have created it, but it's empty until you start typing.

My suggestion is that when you're still new to this, you go to a quiet place and write out your SAS program on a piece of paper before sitting down at the computer, using examples from lecture and this document as models.

```
YesMaster > emacs reading1.sas
```

(Now you type in your program and then exit emacs)

```
YesMaster > ls
drp.dat        reading1.sas
```

Now you're ready to go. Here is an outline of the whole process.

- Get a copy of the data file with `curl`.
- *Look at the data file* with `less` or `cat`.
- Edit the data file with `emacs` if necessary.
- Create the program file by typing it into `emacs`. It is a good idea to copy-paste bits of my code or your own code from earlier jobs, but of course you are not allowed to look at your classmates' work at all, much less copy it.
- Run SAS.
- Look at the log file. If there are errors or warnings, edit the program or the data file and run SAS again. Repeat as necessary.
- When there are no more errors or warnings, look at the list file. Some mistakes are apparent in the list file (statistical output), but do not cause error or warning messages. If necessary, edit the program or the data file and run SAS again.
- When everything is okay, email the log and list files to yourself and print them.

2.2.5 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 not aligned. Here are the first few lines.