SAS Examples of Chapter 1

1.1 Input and output of datasets of SAS

Before we handle our data under SAS environment, we have to go through some basic concepts about SAS.

Dataset = A collection of observations. Each column of dataset represents a variable which stores observations with the same feature. For example, variable 'blood pressure' in a dataset stores information of blood pressures of all patients and each row of dataset represents an observation of observed data.

Library = A collection of datasets. A library consists of many datasets.

By default, all created datasets in SAS are stored in the library `work`. However, one pitfall of this approach is that the datasets are lost once SAS is shut down. If we want to store the datasets in our computer permanently, we have to create a new library and store them in this library.

Under SAS application environments, there are 5 main windows:

1. `Program Editor` – Edit SAS programs
2. `Log` – Records the running messages of SAS session, which is very helpful for program debugging
3. `Output` – Display output from SAS procedures
4. `Explorer` – Manage SAS datasets or Create new libraries
5. `Result` – Show a tree-like summary of your Output window

1.1.1 Create a new SAS library

To create a new SAS library, the following statement should be used.

```
libname stat3302 'D:\';
```

After running the above statement, a new library `stat3302` is created and all datasets are stored in the path `D:\`;

1.1.2 How can we access datasets in the other SAS libraries?

Generally, datasets stored in the library `work` can be accessed directly. For example,

```
data testdat;
   set testdat;
   ⋮
   run;
```

Then, we can access the dataset `testdat` by inserting any statements before the statement `run`.

Suppose that we have a dataset `testdat` stored in library `stat3302`. Then, we have to use the following statements:

```
data stat3302.testdat;
   set stat3302.testdat;
   ⋮
   run;
```

Note that we have to add the library name as prefix before the name of dataset and they are separated by a dot. Then, we can perform the similar operations by inserting statements before the statement `run`.

1.1.3 How do we input data in SAS?

1.1.3.1 When the number of observations is small,

When the number of observations in the dataset is small, a direct method of creating a new dataset is to include all observations in our SAS program. For example,

```
data homes;
   input price area acres rooms baths;
   datalines;
   179000 3060 0.7500 8 2.0
   126500 1600 0.2600 8 1.5
   134500 2000 0.7000 8 1.0
   ;
   run;
```

After running the above statements, a dataset `homes` which contains five columns and three rows will be created.

1.1.3.2 When the number of observations is large,

However, when the number of observations in the dataset is large, it may not be feasible to include all observations in our SAS program. Nevertheless, there are three alternatives to solve this problem.

A. Use data step to create new dataset.

When an external data file is stored in ASCII format and each column is separated by space, then we can use the following data step to create dataset:

```
data stat3302.testdat;
   infile "C:\TUT01-1.DAT" delimiter=" ";
   input Age Weight Oxygen RunTime;
   run;
```

Obviously, the name of external data file is `TUT01-1.DAT` and we have to use the option `delimiter=" "` in this example. The subsequent statement `input` is to assign the variable names for columns in the dataset `stat3302.testdat`.

Nevertheless, in my experience, when the file is stored in CSV format, the process of data input can be completed successfully in most cases. To change the format of external data file, we can use Excel to read in the data file and save it in CSV `'comma delimited'` format. Then, we can use the data step to input the CSV formatted data file with option `delimiter=","`. A simple example is shown as follows:

```
data stat3302.testdat;
   infile "C:\TUT01-2.CSV" delimiter=",";
   input Age Weight Oxygen RunTime;
   run;
```

BE CAREFUL! We should add the signs $ after the variable names if they are string variables.

B. Use `PROC IMPORT` to create new dataset.

When an external data file is stored in Excel format, then the following statements can be used.

```
proc import datafile="C:\TUT01-3.xls"
   out=stat3302.testdat
   dbms=excel2000
   replace;
   sheet="Sheet1";
   getnames=yes;
   run;
```

When we use the `PROC IMPORT` to create a new dataset, we should be careful about the following options: `out=` represents the name of new dataset; `dbms=` specifies the format of input file; `replace` means overwriting the dataset without warning if it exists already; `sheet=` specifies the name of worksheet to be imported. `getnames=yes` retrieves the variable names from the header of the worksheet. If `getnames=no` is used, each column in the dataset will be labelled by `F1`,...,`F?` consequently.

C. Create a new dataset interactively by import wizard in SAS.

From the main menu,

- ▷ `Select File`
- ▷ Select `File`
- ▷ Select `Import Data`
- ▷ `Select a data source from the list below (Microsoft Excel ... )`
- ▷ Click `Next`
- ▷ `Enter the name of excel file`
- ▷ Click `OK`

- ▷ `Which table do you want to import?` Specify the name of worksheet for import
- ▷ Click `Next`
- ▷ `Enter new member name`. Enter the name of new dataset
- ▷ Click `Finish`

BE CAREFUL! One should check with the new dataset for security.

1.1.3.3 Add new observations at the end of a dataset

Suppose that we want to add new observations at the end of a dataset. Then, we have to create a new dataset to store new observations. As an example,

```
data homes2;
   input price area acres rooms baths;
   datalines;
   164000    1956    0.5000    8    2.5
   146000    2400    0.4000    7    2.5;
   run;
```

Now, we have created a dataset `homes2` which stores new observations. Then, we can use data step to combine the new observations with original dataset.

```
data newhomes
   set homes homes2;
   run;
```

A new dataset `newhomes` is created and stores the observations of dataset `homes2` after those of dataset `homes`.

1.1.4 How do we output dataset in SAS?

A. Use `PROC EXPORT` to output dataset in excel file.

In most cases, we want to export dataset in excel format. Then, we can use the following statements:

```
proc export data=stat3302.testdat
   dbms=excel2000
   outfile="C:\stat3302dat.xls"
   replace;
   sheet="Sheet1";
   run;
```

Basically, options in this procedure are similar to those in `PROC IMPORT`, thus the explanation will not be repeated for convenience.

B. Export dataset interactively by export wizard in SAS.
From the main menu,
▷ Select `File`
▷ Select `Export Data`
▷ Select `library and member`, i.e., the names of library and dataset for export
▷ Click `Next`
▷ `Select a data source from the list below (Microsoft Excel ... )`
▷ Click `Next`
▷ `Enter the name of excel file`
▷ Click `OK`
▷ `Assign a name to the export table`. Specify the name of worksheet in the excel file
▷ Click `Finish`

1.2 How to use SAS procedures?

The syntax of using SAS procedures is given below:

```
proc procedure_names data=dataset_names <options>;
   ... statements ...
   run;
```

Examples of some useful procedures:

```
proc print data=dataset_names ;
   var variable
   run;
proc univariate data=dataset_names ;
   var variable ;
   run;
proc corr data=dataset_names ;
   var variable ;
   run;
proc gplot data=dataset_names ;
   plot y_variable*x_variable ;
   run;
```

1.3 A SAS procedure for interactive data analysis

Once we have run the following SAS statements:

```
proc insight data=dataset_names ;
   run;
```

an pop-up screen will be displayed and we can perform many useful data analyses.

1.4 A SAS procedure for interactive matrix language (IML) programming

Basically, IML (Interactive Matrix Language) is used to perform matrix manipulation and computation. Indeed, it can also be used as a big calculator. Since many intrinsic functions have been provided by IML, many calculations can be simplified by a few program lines. Typically, the structure of IML program is expressed in the following format:

```
proc iml;
   start <subroutine name_1>;
      ... statements ...
   finish <subroutine name_1>;
   ⋮
   start <subroutine name_n>;
      ... statements ...
   finish <subroutine name_n>;
   run;
quit;
```

In the above illustration, a program is divided into n small programs (subroutines) which are specified by different subroutine names. In later example, we will show how to call the subroutines within the main program. Note that we do not need to specify the name of main program. Implicitly, the name of main program is called `main`. Therefore, if we specify the main program in the following forms:

```
start;
  ... statements ...
finish;
```

or

```
start main;
  ... statements ...
finish main;
```

they are identical.

If our programs do not have any subroutines, they can be specified in the following form:

```
proc iml;
   start;
     ... statements ...
   finish;
   run;
quit;
```

or more simply,

```
proc iml;
    ... statements ...
quit;
```

As we can see that if we use the `start` and `finish` statements together, we have to use the `run` statement to execute the main program. Otherwise, no program will be executed although it has been defined already.

To execute the subroutines within the main program, we use the following syntax,

```
run subroutine_name<(optional_parameters)>;
```

Suppose that we want to solve a quadratic equation problem. i.e. given a, b and c, solve $ax^2 + bx + c = 0$ for x. Typically, we have an analytical form for the solution x.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a},$$

for $a \neq 0$.

Now, we formulate the problem by `PROC IML`.

```
proc iml;
   a = 1;
   b = 2;
   c = 1;
   d = b**2 - 4*a*c;
   if (d >= 0) then
      do;
         x1 = (-b - sqrt(d))/(2*a);
         x2 = (-b + sqrt(d))/(2*a);
         print 'The solutions are real.';
         print 'They are :' x1 'and ' x2;
      end;
   else
      do;
         x1 = -b/(2*a);
         x2 = sqrt(abs(d))/(2*a);
         print 'The solutions are imaginary.';
         print 'They are :' x1 '-' x2 'and 'x1 '+' x2;
      end;
quit;
```

After execution of the program, we have the following output:

```
The solutions are real.
          They are :     -1 and     -1
```

In the above example, we solve the problem $x^2 + 2x + 1 = 0$ and the solution should be -1. Nevertheless, the above program can be used to solve the the case of imaginary roots as well because we have considered the case in the program already. (Try $a = 1, b = 1$ and $c = 1$!).

Another example is to produce univariate summary statistics if we are given the observations of two variables:

```
proc iml;
   dat = {1 3,
          2 5,
          6 8};
   xmean = dat[:,];
   xstd = j(ncol(dat),1,0);
   do j = 1 to ncol(dat);
      xstd[j] = sqrt(ssq(dat[,j]-xmean[j])/(nrow(dat)-1));
   end;
   print 'The means are' ((xmean`)`)[format=10.4];
   print 'The standard deviations are' (xstd`)[format=10.4];
quit;
```

After the program is executed, we have the following output:

```
The means are     3.0000     5.3333

The standard deviations are     2.6458     2.5166
```

Although the above program is not written optimally, it is used to demonstrate the usefulness of IML. In fact, this program can also be used to produce the means and standard deviations of matrix `dat` without specifying the number of rows and columns explicitly in the program because we have used the functions `nrow(·)` and `ncol(·)` to take care of it. On the other hand, for mean calculation, we have used the operator `":"` for the matrix `dat`. Note that the resulting matrix `xmean` should be a row vector of order 1×2. Also, for the calculation of standard deviation, we have used the following formula:

$$\sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}}.$$

The function `ssq(·)` is used to calculate the sum of squares for each column of `dat` and `nrow(dat)` will return the value of n.

The next example will show the possibility of multiple regression (which will be discussed in this course) by IML. The main purpose is to demonstrate how to formulate the problem in IML. Suppose that we have two data matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ where

$$\boldsymbol{X} = \begin{pmatrix} 1 & 1 & 3 \\ 1 & 2 & 5 \\ 1 & 6 & 8 \\ 1 & 5 & 2 \end{pmatrix} \text{ and } \boldsymbol{Y} = \begin{pmatrix} 4 \\ 9 \\ 11 \\ 5 \end{pmatrix}.$$

Then, the explicit formula for the estimate of intercept and slope coefficients is

$$\hat{\boldsymbol{B}} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\left(\boldsymbol{X}^T\boldsymbol{Y}\right).$$

and their estimated standard errors are the square roots of diagonal elements of

$$s^2\left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1},$$

where $s^2 = \dfrac{(\boldsymbol{Y}-\boldsymbol{X}\hat{\boldsymbol{B}})^T(\boldsymbol{Y}-\boldsymbol{X}\hat{\boldsymbol{B}})}{4-3}$. Note that the degree of freedom in the denominator is specified explicitly for simplicity.

```
proc iml;
   xdat = {1 1 3,
           1 2 5,
           1 6 8,
           1 5 2};
   ydat = {4,
           9,
           11,
           5};
   bhat = inv(xdat‘*xdat)*xdat‘*ydat;
   s_sq = ((ydat-xdat*bhat)‘*(ydat-xdat*bhat))/(nrow(xdat)-ncol(xdat));
   bse = sqrt(vecdiag(s_sq*inv(xdat‘*xdat)));
   print ’The estimated parameters are’, (bhat‘)[format=10.4];
   print ’The standard errors are’, (bse‘)[format=10.4];
quit;
```

After the program is executed, we have the following output:

```
The estimated parameters are
1.7227     0.1553     1.1075

The standard errors are
2.3508     0.5239     0.4714
```

1.5 Documentation of SAS procedures

The details of all SAS procedures can be found in the SAS Help and Documentation:

- ▷ Select `Help`
- ▷ Select `SAS Help and Documentation`
- ▷ Select `SAS Products`

1.5.1 For reference of PROC PRINT, PROC CORR and PROC UNIVARIATE,

▷ Select `Base SAS`
  ▷ Select `SAS procedures`
  ▷ Select `Procedures`
  Then, we can select the procedure that we want to know.

1.5.2 For reference of PROC GPLOT,

▷ Select `SAS/GRAPH`
  ▷ Select `SAS/GRAPH Reference`
  ▷ Select `SAS/GRAPH Procedures`
  ▷ Select `The GPLOT Procedure`

1.5.3 For reference of PROC IML,

▷ Select `SAS/IML`
  ▷ Select `Language Reference`

1.5.4 For reference of the other SAS procedures using in this course,

▷ Select `SAS/STAT`
  ▷ Select `SAS/STAT User's Guide`
  Then, we can select the procedure that want to know.

1.6 Examples

1.6.1 Example 1

Suppose we are given a dataset of 3 variables.

X_1	X_2	X_3
1	2	3
3	2	1
4	2	1
24	1	0
1	3	8

a. Use procedure `PROC IML` to find the sample mean and sample standard deviation.

b. Use procedure `PROC IML` to find the sample covariance matrix and correlation matrix.

Explanatory Note

#1 Invoke `PROC IML`.

#2 Define a variable $\boldsymbol{x}$ which stores a matrix. The matrix should be embraced by curly braces. Each row is separated by a comma while the elements in each row are separated by space characters. The 5×3 matrix is then defined as

$$\boldsymbol{x} = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 4 & 2 & 1 \\ 24 & 1 & 0 \\ 1 & 3 & 8 \end{pmatrix}.$$

#7 Function `nrow(x)` is used to obtain the number of rows of matrix `x`. In this example, `n` $= 5$.

#8 `x[:,]` returns a row vector whose elements are the sample means of each column of matrix `x`. Specifically, `x[:,]` produces a 1×3 row vector whose the first element is calculated by `(1+3+4+24+1)/5`. Similarly, the second and third elements are `(2+2+2+1+3)/5` and `(3+1+10+8)/5` respectively. As a result, `x[:,]` gives a 1×3 vector

$$\begin{pmatrix} \bar{x}_1 & \bar{x}_2 & \bar{x}_3 \end{pmatrix}.$$

Function `t(x[:,])` transposes the matrix `x[:,]` inside the bracket. Then, expression `mean = t(x[:,])` gives a 3×1 vector

$$\begin{pmatrix} \bar{x}_1 \\ \bar{x}_2 \\ \bar{x}_3 \end{pmatrix}.$$

Alternatively, expression `x[:,]‘` can give the same result.

#9 Compute $\boldsymbol{W} = \boldsymbol{X}'\boldsymbol{X} - n\bar{\boldsymbol{x}}\,\bar{\boldsymbol{x}}'$ and we have

$$\begin{pmatrix} \sum_{i=1}^{n}(x_{i1}-\bar{x}_1)^2 & \sum_{i=1}^{n}(x_{i1}-\bar{x}_1)(x_{i2}-\bar{x}_2) & \sum_{i=1}^{n}(x_{i1}-\bar{x}_1)(x_{i3}-\bar{x}_3) \\ \sum_{i=1}^{n}(x_{i2}-\bar{x}_2)(x_{i1}-\bar{x}_1) & \sum_{i=1}^{n}(x_{i2}-\bar{x}_2)^2 & \sum_{i=1}^{n}(x_{i2}-\bar{x}_2)(x_{i3}-\bar{x}_3) \\ \sum_{i=1}^{n}(x_{i3}-\bar{x}_3)(x_{i1}-\bar{x}_1) & \sum_{i=1}^{n}(x_{i3}-\bar{x}_3)(x_{i2}-\bar{x}_2) & \sum_{i=1}^{n}(x_{i2}-\bar{x}_2)^2 \end{pmatrix}.$$

#10 Compute $\boldsymbol{S} = \boldsymbol{W}/(n-1)$ and we have

$$\begin{pmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{32} & s_{33} \end{pmatrix}.$$

#11 Function `vecdiag(.)` extracts the elements on the principal diagonal of the matrix and arranges them in a column vector. For example, $\boldsymbol{x} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, `vecdiag(x)` gives $\begin{pmatrix} a \\ d \end{pmatrix}$. On the other hand, the function `sqrt(.)` will give a matrix whose elements are square-rooted, i.e. $\boldsymbol{x} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ and `sqrt(x)` gives $\begin{pmatrix} \sqrt{a} & \sqrt{b} \\ \sqrt{c} & \sqrt{d} \end{pmatrix}$ when a, b, c and $d \geq 0$. Thus, `sqrt(vecdiag(cov))` gives a diagonal matrix of standard deviations of X_1, X_2 and X_3, i.e.,

$$\begin{pmatrix} \sqrt{s_{11}} \\ \sqrt{s_{22}} \\ \sqrt{s_{33}} \end{pmatrix}.$$

#12 Compute the matrix $\boldsymbol{D}^{-1}$. Function `diag(.)` creates a diagonal matrix whose elements on the principal diagonal are given by a column vector inside the bracket. Then, `diag(1/std)` gives

$$\begin{pmatrix} \frac{1}{\sqrt{s_{11}}} & 0 & 0 \\ 0 & \frac{1}{\sqrt{s_{22}}} & 0 \\ 0 & 0 & \frac{1}{\sqrt{s_{33}}} \end{pmatrix}.$$

Note that `(1/std)` performs an element by element division.

#13 Compute $\boldsymbol{R} = \boldsymbol{D}^{-1}\boldsymbol{S}\boldsymbol{D}^{-1}$ and we have

$$\begin{pmatrix} 1 & \frac{s_{12}}{\sqrt{s_{11}s_{22}}} & \frac{s_{13}}{\sqrt{s_{11}s_{33}}} \\ \frac{s_{21}}{\sqrt{s_{11}s_{22}}} & 1 & \frac{s_{23}}{\sqrt{s_{22}s_{33}}} \\ \frac{s_{31}}{\sqrt{s_{11}s_{33}}} & \frac{s_{32}}{\sqrt{s_{22}s_{33}}} & 1 \end{pmatrix}.$$

#14 Display the results on the screen.

1.6.2 Example 2

Suppose eight men each received a certain drug. The changes in blood sugar (BS), blood pressure (low and high) are recorded.

Name	AGE	BS	Low_BP	High_BP
Schenk	30	30	−8	−1
Voss	32	90	7	6
Steen	35	−10	−2	4
Thompson	35	35	10	2
Blondel	35	30	−2	5
Plaziat	35	60	0	3
Bright	35	0	−2	4
De_Wit	40	40	1	2

a. Create a temporary SAS data set called `BLOOD` from data lines.

b. Use procedure `PROC PRINT` to display the data.

c. Use procedure `PROC UNIVARIATE` and `PROC CORR` to compute summary statistics.

d. Use procedure `PROC GPLOT` to plot `BS` against `Low_BP`.

e. Invoke the `INSIGHT` procedure to produce a scatter plot matrix.

f. Use `PROC IML` to read the dataset into a data matrix and compute the summary statistics including the means, standard deviations and correlations.

g. Redo part (f) on the reduced dataset consists of all men with age 35.

Explanatory Note

Previously, the statements within `PROC IML` are executed successively. In some cases, we may need to use certain portions of program repeatedly. Therefore, it is helpful to split the program into several small programs (subroutines).

Now, we want to call a the subroutine `summary` performs the same function as in that in Example 1. Then, we have to provide a matrix `x` as a parameter for the subroutine `summary`. Recall that to run a subroutine, the syntax is

```
run subroutine_name <(optional_parameters)>;
```

Thus, for part (f) of the example, we are asked to produce summary statistics for a dataset.

```
1  use blood;
2  read all var {age bs low_bp high_bp} into blood1;
3  close blood;
4  run summary(blood1);
```

#1 Open the dataset `blood` which is created previously.

#2 Read the observations of variables `age`, `bs`, `low_bp` and `high_bp` into matrix `blood1`.

#3 Close the dataset `blood`.

#4 Run the subroutine `summary` and use `blood1` as the input parameter.

For part (g), we are asked to compute summary statistics of observations for all men with age 35. Then, we can use the following statement to read in observations into matrix `blood2`.

```
read all var {age bs low_bp high_bp} where(age=35) into blood2;
```

Note that we have used the statement `where` to specify the condition.

1.6.3 Example 3

Suppose that a dataset `dat1` is created by the following SAS statements:

```
1  data dat1;
2     input Age Weight Oxygen RunTime @@;
3     datalines;
4     44 89.47 44.609 11.37 40 75.07 45.313 10.07
5     44 85.84 54.297 8.65  42 68.15 59.571 8.17
6     38 89.02 49.874  .    47 77.45 44.811 11.63
7     40 75.98 45.681 11.95 43 81.19 49.091 10.85
8     44 81.42 39.442 13.08 38 81.87 60.055 8.63
9     44 73.03 50.541 10.13 45 87.66 37.388 14.03
10    45 66.45 44.754 11.12 47 79.15 47.273 10.60
11    54 83.12 51.855 10.33 49 81.42 49.156 8.95
12    51 69.63 40.836 10.95 51 77.91 46.672 10.00
13    48 91.63 46.774 10.25 49 73.37   .    10.08
14    57 73.37 39.407 12.63 54 79.38 46.080 11.17
15    52 76.32 45.441 9.63  50 70.87 54.625 8.92
16    51 67.25 45.118 11.08 54 91.63 39.203 12.88
17    51 73.71 45.790 10.47 57 59.08 50.545 9.93
18    49 76.32   .      .    48 61.24 47.920 11.50
19    52 82.78 47.467 10.50
20    ;
21    run;
```

The description of this SAS program is given below:

#1 Create a new dataset `dat1` under `library work`.

#2 Read in the data into the corresponding variables `Age`, `Weight`, `Oxygen` and `RunTime`.

#3 Specify the following lines are data.

#4-20 Lines of data and it is end by a semicolon.

#21 Run the data step.

Then, by using the above SAS statements, a new dataset `dat1` is created.

Simple manipulation of SAS dataset

Suppose that we want to use the dataset `dat1` to do further data analysis. Indeed, a new dataset is created for each example after running the SAS statements. Thus, if we do not want to create a new dataset, we can use the same dataset name in the statements `data` as in that in the statement `set`.

Example 3.1: Remove the missing observations. Note that in SAS the symbol `.` represents missing observation by default.

```
1  data dat1a;
2     set dat1;
3     if oxygen = . or runtime = . then delete;
4     run;
```

#1 Create a new dataset `dat1a`.

#2 Read the observations from the dataset `dat1`.

#3 Remove the observations when either `oxygen` or `runtime` is missing.

Note that SAS is case-insensitive to the names of variable and the command. Thus, uppercase or lowercase of commands and variable names would not lead to different results.

Example 3.2: Create a variable `ox2` which is the square of oxygen.

```
1  data dat1b;
2     set dat1;
3     ox2 = oxygen**2;
4     run;
```

#3 Create a variable `ox2` which is the square of `oxygen`.

Example 3.3: Remove the variable `runtime` in the dataset `dat1`.

```
1  data dat1c;
2     set dat1;
3     drop runtime;
4     run;
```

#3 Remove the variable `runtime`.

Example 3.4: Keep the variables `oxygen` and `runtime` in the dataset `dat1` only.

```
data dat1d;
   set dat1;
   keep oxygen runtime;
   run;
```

#3 Keep the variables `oxygen` and `runtime`.

Example 3.5: Change the name of variable `oxygen` to `ox`.

```
data dat1e;
   set dat1;
   rename oxygen=ox;
   run;
```

#3 Rename the variable `oxygen` to `ox`.

Example 3.6: Label the variable `oxygen` by an easier description.

```
data dat1f;
   set dat1;
   label oxygen="Oxygen Consumed";
   run;
```

#3 Label the variable `oxygen` as Oxygen Consumed.

SAS Codes for Example 1

```
proc iml;
   x = { 1 2 3,
         3 2 1,
         4 2 1,
        24 1 0,
         1 3 8 };
   n = nrow(x);              /* number of observations */
   mean = t(x[:,]);          /* vector of column means */
   w = x‘*x-n*mean*mean‘;     /* CSSP matrix */
   cov = w/(n-1);            /* sample cov matrix: unbiased estimate */
   std = sqrt(vecdiag(cov)); /* column vector of standard deviation */
   d = diag(1/std);          /* scaling matrix for cov matrix */
   corr = d*cov*d;           /* correlation matrix */
   print mean std ’ ’ corr [format=5.3],, cov;
   run;
quit;
```

SAS Codes for Example 2

```
/* (a) */
data blood;
   /* the $ sign indicates that name is a character variable */
   input name $ age BS Low_BP High_BP;
   datalines;
   Schenk   30  30 -8 -1
   Voss     32  90  7  6
   Steen    35 -10 -2  4
   Thompson 35  35 10  2
   Blondel  35  30 -2  5
   Plaziat  35  60  0  3
   Bright   35   0 -2  4
   De_Wit   40  40  1  2
   ;
   run;

/* (b) */
proc print data=blood;
   run;

/* (c) */
proc univariate data=blood;
   var age BS Low_BP High_BP;
   run;

proc corr data=blood cov;
   var age BS Low_BP High_BP;
   run;

/* (d) */
proc gplot data=blood;
   plot BS*Low_BP;
   run;

/* (e) */
proc insight data=blood;
   run;

/* (f) and (g) */
proc iml;
   start summary(x);
   n = nrow(x);               /* number of observations */
   mean = t(x[:,]);           /* vector of column means */
   w = x`*x-n*mean*mean`;     /* CSSP matrix */
   cov = w/(n-1);             /* sample cov matrix:unbiased estimate*/
   std = sqrt(vecdiag(cov));  /* column vector of standard deviation*/
   if all(std) = 0 then
      print "Some STD are zeros: perhaps constant columns?",, mean
            std;
   else
      do;
         d = diag(1/std);     /* scaling matrix for cov matrix */
```

```
53          corr = d*cov*d;      /* correlation matrix */
54          print mean std ' ' corr [format=5.3],, cov;
55       end;
56 finish summary;
57 /* End of module */
58
59 /* define a 5 by 3 data matrix */
60 x = { 1 2 3,
61       3 2 1,
62       4 2 1,
63      24 1 0,
64       1 3 8 };
65 /* Compute summary statistics of x */
66 run summary(x);
67
68 /* Import data matrix from a SAS data set */
69 use blood;
70 read all var {age bs low_bp high_bp} into blood1;
71 close blood;
72 run summary(blood1);
73
74 /* Import data matrix from a (reduced) SAS data set */
75 use blood;
76 read all var {age bs low_bp high_bp} where(age=35) into blood2;
77 close blood;
78 run summary(blood2);
79 /* note that the first column causes trouble: a constant */
80
81 /* creates a data matrix without the first column */
82 blood3 = blood2[, 2:4];
83 run summary(blood3);
84 quit;
85 /* exit the procedure IML */
```