General R Information: Sections 1-6

How to Save R “Programs” AND How to Create Data Files for R

How to Get Data into R

How to Find, Install and Load R Packages

How to Generate Random Samples and Determine Their Frequency Distributions

How to See Whether a Specific Value Occurs in a Data Set

How to Extract Particular Data Items or Sequences of Them

Section 1: How to Save R "Programs" AND How to Create Data Files for R

How to Save R Programs

After you have been working interactively in R, copy and paste the session into NotePad. Edit out everything but the "good" commands – that is, delete all of the > prompt symbols, the incorrect commands and their error messages, and the output. Save the remaining list of good commands as a text file.

When you open R again, you can copy/paste this file into R right after the > prompt appears. R will run all of the commands at once as a "program" and reproduce all the prior output.

Note: You can put comment lines in your R code so that when you go back to it later, you know what the code is supposed to do. To make a comment line, just put a hashtag symbol immediately after the prompt symbol. For example, you could write:

```
> #This is a comment line    (and R would just ignore it)
```

How to Create Data Files for R

Here are two options.

1. Type the data into a NotePad text file. The first column should be ID values for the data items, and the first row should be headers. Save as a text file.

2. Enter the data in a spreadsheet such as Excel because it is easier to edit it there. But then you have to save it as a text file that R can use. To do so:
   - make sure the spreadsheet file only has one worksheet
   - when you save it, you will want to save it twice.

Save it once as a regular spreadsheet file, in case you want to edit some more. Save it the second time as file type "text (tab delimited)" and that will turn it into a text file with spaces between the columns. This is the form that you will use in R.

To make it more readable once it is a text file, you may want to add more spaces in order to make the columns straight. If so, use the space bar; DO NOT USE the tab key. However, straightening out the columns is optional; R will not care one way or the other.

A Few Words about Data

When you are working in Excel, it is natural to leave spaces in labels such as column headings, or in other data that is not numeric. It also seems natural to include dollar signs if the data is about currency, commas in long numbers, and similar notations. Likewise, it seems natural to use numbers as identifiers.

DO NOT DO ANY OF THOSE THINGS! SEE EXAMPLES BELOW.

They will cause various kinds of errors when R tries to read or work with the data set!

Examples

Instead of: Use: Reason:

Student ID	Student.ID	R would interpret the first version as two items, not one as you intended it to be.
$400,000	400000	The $ and the comma make it non-numeric, so that it cannot be used for calculations in R.
23 *(as an ID#)*	N23	R will incorrectly interpret the “23” as an actual number and not an ID number.

There are other ways to address these problems, but the simplest thing is to plan ahead when you set up your data and avoid them entirely.

Section 2: How to Get Data into R

(Uses data file: Hospitals.txt)

The prompt in R is the > symbol. When you see this symbol, you can type a command. Your first problem is to get some data into R so that you can work on it. So first read the data table into R. Assuming you have column headings in your data set, the form of the command is:

```
> Data = read.table ("Drive:/Folder/File.txt", header = TRUE)
```

For example, if the file "Hospitals.txt" is stored in a folder called "Data Files" on drive E, the command is:

```
> Data = read.table ("E:/Data Files/Hospitals.txt", header = TRUE)
```

You can choose the name that you want R to use for the whole data set and put it on the left side of the equal sign, right after the prompt symbol. You can either give it a meaningful name (for example, you could call it Hospital.Name) or just call it a generic name like "Data." The line above uses the generic name.

Once you have read the data into R, it needs to be "attached." THIS IS ESSENTIAL! Otherwise R cannot work with the data set. You specify "attach" followed by the name that you gave the data set in parentheses.

```
> attach (Data)
```

To display the data once it has been read into R, type the name that you gave the data set. For the rest of this section, assume that you gave it the generic name: Data.

```
> Data
```

The resulting output is as shown.

-- The first column is a record counter generated by R; it is NOT part of the data file and is not an ID number.
-- The second column is the name of the hospital; in this example, these are made-up names using Greek letters. In a real data set, you would have real names of real hospitals.
-- The third is the count of its capacity in number of inpatient beds.

	Hospital	No.Beds
1	Alpha	787
2	Beta	356
3	Gamma	190
4	Delta	1252
5	Epsilon	767
6	Sigma	264
7	Theta	457
8	Kappa	154
9	Lambda	333
10	Omega	525

Section 3: How to Find, Install and Load R Packages

1. How to Find R Packages

Frequently, you will know what you want R to do but you may not know whether R has a ready-made command to do it. Or you may know such a command exists, but you may not know the exact syntax. You can search R documentation for the base package by typing a double question mark (??) after the prompt (>), and then the name of the process you want to perform. If R can match exactly what you type, it will show you documentation. Otherwise (maybe more often than not), you will have to try something else.

In that case, do an internet search using your preferred search engine. For that, you will not need to know the exact R command, but you can type a brief description of what you want to do. For example, you might search online for the following: "Test single variance in R."

Your search engine will usually find references for you, which you can use to figure out the command that you need. The query above will fairly quickly lead you to the command "varTest," which is in the package "EnvStats."

That brings you to the point where you have to find out whether or not you already have that package. If you do, you just need to load it. If you don't, you need to install it and then load it. Once the package is loaded, you can use the command. Note that, if you stop work and close R, you will need to reload the package whenever you want it again. You should not need to reinstall it however.

2. How to See Whether or Not You Already Have a Particular Package

On the menu bar, choose "Packages" and then "Load Package." You will get a long list of packages in alphabetical order. Scroll down the list until you get to the point where the package you want should be in the list. If it shows up, you have it and can skip the "install" step and go directly to "load" step. If it doesn't show up, you have to "install" the package.

3. How to Install an R Package

Back on the menu bar, choose "Packages" and then "Install package." You will get a list of CRAN mirrors; these are repositories of base R and R libraries. The list is alphabetical, so the U.S. repositories are near the end of the list. Choose one of them. Note: In at least one recent version of R, this does not work properly and all you get is a one line of output about utility programs. If that happens, you can accomplish the same thing by typing "install.packages( )" after the > prompt.

After you choose a CRAN mirror, you will get an alphabetical list of packages. Find the one you want and click on it. The installation process will run automatically.

4. How to Load an R Package (Once it has been installed)

If you got here directly from step 2, you have already located the package. So you just click on it and wait for R to load it. If you had to do step 3, now repeat step 2 and then click on the name of the package. Either way, when the loading is finished, you will see the R prompt (>) again and can now use the command that you wanted.

Section 4: How to Generate Random Samples and Determine Their Frequency Distributions

(Uses no data files)

The "sample( )" command generates random whole numbers. It requires three items inside the parentheses: (1) the interval for the random values, written in the form "lower:upper ", (2) the number of values to be generated – that is, the sample size, and (3) a subcommand to indicate whether the sample is to be done with or without replacement.

Here are three examples.

EXAMPLE A

This example generates a random sample of 100 zeros and ones, with replacement. It represents the experiment of tossing a coin, with the notation 0 = Tails and 1 = Heads. The phrase "with replacement" means that values can be repeated within the sample. In this case, they have to be since only two values are possible and there are 100 repetitions.

```
> s1 = sample (0:1, 100, replace=TRUE)
> s1
```

The resulting output changes each time the command is run because a new sample is generated. A typical example is shown below.

```
1 0 1 1 0 1 0 1 0 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 1 0 0 1 1 0 1 1 1 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0
1 1 1 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 1 1 0 1 1 1 1 1 0 0 0 1 1 0 0 0 0 0 0 1 1 1 0
```

You can then get a frequency distribution for the sample by using the "ftable( )" command. The name that you gave the sample, in this case s1, goes in the parentheses. You can also obtain a histogram with "hist( )" and the name of the sample in parentheses.

```
> ftable (s1)
```

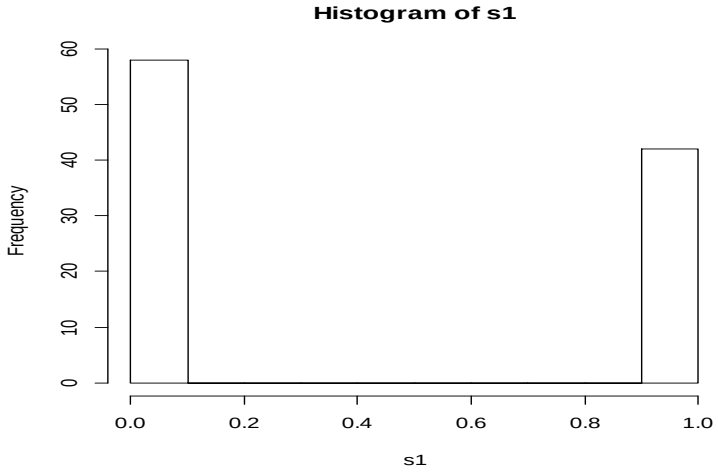
The resulting output follows; it indicates that there were 58 "tails" and 42 "heads" in the sample.

```
s1   0   1
    58  42
```

You can also generate a histogram of the sample.

```
>  hist (s1)
```

You then get the following graph.



EXAMPLE B

The second example generates a random sample of 100 values 1 through 6, with replacement. This represents the experiment of rolling a fair die, one hundred times.

```
> s2 = sample (1:6, 100, replace=TRUE)
> s2
```

The resulting output changes each time the command is run because a new sample is generated. A typical example is shown below.

```
1 1 6 1 5 6 4 6 5 1 1 6 3 3 5 1 2 4 2 5 1 6 2 5 6 4 4 3 5 4 2 2 4 3 4 1 6  4 2 5 1 4 4 1 3 3 3 1 6 4 2 2 4 5 2 3 3 3
2 5 1 5 5 4 4 3 1 4 2 4 5 4 6 4 6 6 6 6 5 6 2 5 6 1 2 6 3 6 1 2 5 6 4 3 3 6 6 1 1 4
```

You can then get a frequency distribution and a histogram if you choose.

```
> ftable (s2)
```

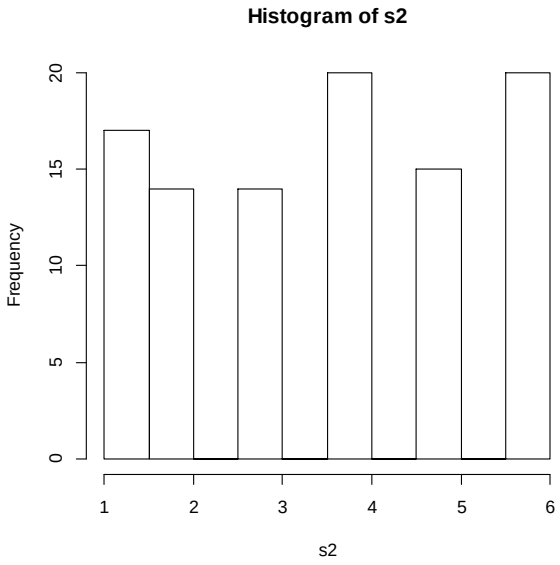
The resulting output is given below.

```
s2      1    2    3    4    5    6
       17   14   14   20   15   20
```

This output indicates that you “rolled” 17 ones, 14 twos, 14 threes, 20 fours, 15 fives and 20 sixes in the sample.

```
> hist (s2)
```

The following graph results.



EXAMPLE C

This last example illustrates sampling without replacement. That means a value cannot be repeated within the sample, so your sample size cannot exceed the number of items in the set from which you are sampling.

You can picture this example as if you had the numbers 1 through 5 written on pieces of paper and placed in a box. You then draw three of them at random.

```
> s3 = sample (1:5, 3, replace=FALSE)
> s3
```

The resulting output changes each time the command is run because a new sample is generated. A typical example is shown below.

```
4 1  5
```

Note that you if you try to generate a sample without replacement, and the sample size is bigger than the number of original items, you will get an error message. You can see what this looks like if you change the "3" in the sample command for s3 to any number larger than five.

Section 5: How to See Whether a Specific Value Occurs in a Data Set
(Uses data file: AnxietyRepeat.txt)

This example uses the data set AnxietyRepeat.txt. This data set contains scores on an Anxiety Test, repeated three times on thirty-six (fictional) students. The test is first given during their first term in college (labelled Fall1), repeated second term (labelled Spr1) and then again in their third term (labelled Fall2).

First read in the data table giving the anxiety scores and attach it. Then display the data set.

```
> Data = read.table ("E:/Data Files/AnxietyRepeat.txt", header=TRUE)
> attach (Data)
> Data
```

A partial display of the output is as follows; the spacing lines after every three scores have been inserted here for ease of reading.

```
     ID  Test.Session  Anx.Score
1    S1       Fall1         25
2    S1       Spr1          22
3    S1       Fall2         33

4    S2       Fall1         11
5    S2       Spr1           5
6    S2       Fall2         20
:
103  S35      Fall1         22
104  S35      Spr1          27
105  S35      Fall2         17

106  S36      Fall1         24
107  S36      Spr1          30
108  S36      Fall2         37
```

For instance, suppose you want to see whether or not anyone has a score of 22.

```
> which (Anx.Score == 22)
```

The output is:

```
 2  60  70 103
```

This tells you that the value “22” occurred in the list of scores at positions 2, 60, 70 and 103. You can see the first and the last of these in the partial data set displayed above.

Similarly, suppose you want to see whether or not anyone has a score of 9.

```
> which (Anx.Score == 9)
```

The output is a message:

```
integer(0)
```

This message doesn't seem to tell you much, but it means that the value you wanted does not occur in the data. That is, there is no score of "9" in this set of scores.

Section 6: How to Extract Particular Data Items or Sequences of Them

(Uses data file: AnxietyRepeat.txt)

The example uses the data set AnxietyRepeat.txt. This data set contains scores on an Anxiety Test, repeated three times on thirty-six (fictional) students. The test is given during their first term in college (labelled Fall1), repeated second term (labelled Spr1) and again in their third term (labelled Fall2).

First read in the data table giving the anxiety scores and attach it. Then display the data set.

```
> Data = read.table ("E:/Data Files/AnxietyRepeat.txt", header = TRUE)
> attach (Data)
> Data
```

A partial display of the output is as follows; the spacing lines after every three scores have been inserted here for ease of reading.

```
    ID  Test.Session  Anx.Score
1   S1     Fall1         25
2   S1     Spr1          22
3   S1     Fall2         33

4   S2     Fall1         11
5   S2     Spr1           5
6   S2     Fall2         20
:
:
103 S35    Fall1         22
104 S35    Spr1          27
105 S35    Fall2         17

106 S36    Fall1         24
107 S36    Spr1          30
108 S36    Fall2         37
```

Here are some examples of picking out specific data items by specifying their locations in the list.

EXAMPLE A Suppose you want to see only the scores for the student whose ID is S1; that is, the first three items in the list. All you need to do for this is to specify that you want values from Anx.Score, positions 1 through 3. The following command will do this.

```
> Student1 = Anx.Score [1:3]
```

The output is:

```
25  22  33
```

This tells you that the first three scores (the scores for student S1) are 25, 22 and 33. You can confirm this by looking at the partial display of the data set above.

EXAMPLE B Suppose you want to see only this first student's second score, you would specify position 2 only, as follows.

```
> Student1.Score2 = Anx.Score [2]
```

The output is:

```
22
```

EXAMPLE C Continuing with this idea, suppose that the first half of the scores were from students in one major and the second half were from students in a different major. If you want to split the whole list into two separate ones, one for each major, you would type:

```
> Major1.Scores = Anx.Score [1:54]
> Major2.Scores = Anx.Score [55:108]
```

If you then display them, the Major1.Scores returned are:

```
25 22 33 11  5 20  8 31 13 17 13 18  7 20 13 24 36 43 20 15 27 27 31 15 29 50 15 18 14 24 24 18 30 14 37
35 12 26 29 45 19 40  6  5 13 10 12 18 16 14 17 42 39 17
```

And the Major2.Scores returned are:

```
20 33 19  1 26 22 13  7 13 13 39 20 15 23 13 22 27 10 27  5  5 27 33 18 18 42 12 25 15  5 23 11 21 14 37
27 23 36 30  6 8 28 31 18 34 11 32 18 22 27 17 24 30 37
```

You can then work with the scores for each major as separate data sets (named Major1.Scores and Major2.Scores, respectively) if you wish.

EXAMPLE D Now, suppose you want to obtain the sample means for each test session. First you need to split the appropriate scores out of the list by session. Looking at the data set displayed at the beginning of this section, you will notice that R has assigned counter values to the lines of data, appearing on the far left. These counters indicate that:

the Fall1 scores are in positions 1, 4, 7, ….,
the Spr1 scores are in positions 2, 5, 8, …,
the Fall2 scores are in positions 3, 6, 9, … .

You need to make R generate these position sequences and then use them to pick the corresponding anxiety scores out of the last column.

First you need to generate the sequences for these positions. If you name these F1, S1 and F2, respectively, the R code will be as follows. You give R the starting point, the ending point, and the step size for each.

```
> F1 = seq (from=1, to=108, by=3)
> S1 = seq (from=2, to=108, by=3)
> F2 = seq (from=3, to = 108, by=3)
```

You can see these sequences if you then type:

```
> F1, S1, F2
```

If you check the resulting output, you will see that the sequences give the lists of positions (not scores) you want.

```
1  4  7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88
91  94  97 100 103 106          ← This is the first sequence of positions that you want.

2  5  8 11 14 17 20 23 26 29 32 35 38 41 44 47 50 53 56 59 62 65 68 71 74 77 80 83 86 89
92  95  98 101 104 107          ← This is the second sequence of positions that you want.

3  6  9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60 63 66 69 72 75 78 81 84 87 90
93  96  99 102 105 108          ← And this is the last sequence of positions that you want.
```

You can then make R pick out the anxiety scores in these specific sets of positions. The code below picks out the scores from the positions listed in F1 and assigns them to a variable called F1.Scores, and does the other two similarly.

```
> F1.Scores = Anx.Score [F1]
> S1.Scores = Anx.Score [S1]
> F2.Scores = Anx.Score [F2]
```

You can view the separate lists of scores if you wish just by typing the variable names. For instance, typing the name F1.Scores will show you the list below. You can check it against the original data set to see that it is the right list. The others would be done similarly.

```
25 11 8 17 7 24 20 27 29 18 24 14 12 45  6 10 16 42 20 1 13 13 15 22 27 27 18 25 23 14 23 6 31 11 22 24
```

COMMENT: You can then work with the three individual sets of scores. For instance, you can find the means and variances for the anxiety scores of each test session separately. Here are two lines of code to find the mean and variance of the F1.Scores

```
> mean (F1.Scores)
> var (F1.Scores)
```

The resulting output is:

```
19.16667          ← This is the mean of the F1.Scores in the sample.
90.77143          ← This is the variance of the F1.Scores in the sample.
```

The other two sets of scores can be handled similarly. At that point, you will have obtained basic summary information for each test session.