12

Generating Data

Stata can generate data in a number of ways. The simplest is by use of the `generate` command. It can generate data in loops and through the use of matrix operations as well.

Generating data is important to both R and Stata users. As we have seen, many R functions are controlled by numeric, character, or logical vectors. You can generate those vectors using the methods in this chapter, making quick work of otherwise tedious tasks.

The following are some of the ways we have used vectors as arguments to R functions.

- To create variable names that follow a pattern like q1, q2,...using the `paste` function For an example, see Chap 7.
- To create a set of index values to select variables, as with `mydata[ ,3:6]`. Here the colon operator generates the simple values 3, 4, 5, 6. The methods below can generate a much wider range of patterns.
- To generate sets of column widths when reading data from fixed-width text files. Our example used a very small vector of value widths, but the methods in this chapter could generate long sets of patterns to read hundreds of variables with a single command.

Generating data is also helpful in more general ways for situations similar to those in other statistics packages.

- Generating data allows you to demonstrate various analytic methods, as we are doing in this book. It is not easy to find one data set to use for all of the methods you might like to use.
- Generating data lets you create very small examples that can help you debug an otherwise complex problem. Debugging a problem on a large data set often introduces added complexities and slows down each new attempted execution. When you can demonstrate a problem with the smallest possible set of data, it helps you focus on the exact nature of the

problem. This is usually the best way to report a problem when requesting technical assistance. If possible, provide a small generated example when you ask questions to the R-help e-mail support list.

- When you are designing an experiment, the levels of the experimental variables usually follow simple repetitive patterns. You can generate those and then add the measured outcome values to it manually. With such a nice neat data set to start with, it is tempting to collect data in that order. However, it is important to collect it in random order whenever possible so that factors such as human fatigue or machine wear do not bias the results of your study.

Some of our data generation examples use R's random number generator. It will give a different result each time you use it unless you use the `set.seed` function before each function that generates random numbers.

12.1 Generating Numeric Sequences

R generates data using specialized functions. We have used the simplest one: the colon operator. We can generate a simple sequence with

```
> 1:10

 [1]  1  2  3  4  5  6  7  8  9 10
```

You can store the results of any of our data generation example in a vector using the assignment operator. So we can create the id variable we used with

```
> id <- 1:8

> id

[1] 1 2 3 4 5 6 7 8
```

The `seq` function generates sequences like this too, and it offers more flexibility. The following is an example:

```
> seq(from=1, to=10, by=1)

 [1]  1  2  3  4  5  6  7  8  9 10
```

The `seq` function call above has three arguments.

1. The `from` argument tells it where to begin the sequence.
2. The `to` argument tells it where to stop.
3. The `by` argument tells it the increments to use between each number.

The following is an example that goes from 5 to 50 in increments of 5.

```
> seq(from=5, to=50, by=5)

 [1]  5 10 15 20 25 30 35 40 45 50
```

Of course, you do not need to name the arguments if you use them in order. So you can do the above example using this form too.

```
> seq(5, 50, 5)

 [1]  5 10 15 20 25 30 35 40 45 50
```

12.2 Generating Factors

The `gl` function *g*enerates *l*evels of factors. The following is an example that generates the series 1,2,1,2. . . :

```
> gl(n=2, k=1, length=8)

[1] 1 2 1 2 1 2 1 2

Levels: 1 2
```

The `gl` function call above has three arguments.

1. The `n` argument tells it how many levels your factor will have.
2. The `k` argument tells it how many of each level to repeat before incrementing to the next value.
3. The `length` argument is the total number of values generated. Although this would usually be divisible by n*k, it does not have to be.

To generate our gender variable, we just need to change k to be 4.

```
> gl(n=2, k=4, length=8)

[1] 1 1 1 1 2 2 2 2

Levels: 1 2
```

There is also an optional `label` argument. Here we use it to generate workshop and gender, complete with value labels.

```
> workshop <- gl(n=2, k=1, length=8, label=c("R","Stata") )

> workshop

[1] R   Stata R   Stata R   Stata R   Stata
Levels: R Stata
```

```
> gender <- gl( n=2, k=4, length=8, label=c("f","m") )

> gender

[1] f f f f m m m m
Levels: f m
```

12.3 Generating Repetitious Patterns (Not Factors)

When you need to generate repetitious sequences of values, you are often creating levels of factors, which is covered in the previous section. However, sometimes you need similar patterns that are numeric, not factors. The `rep` function generates these.

```
> gender <- rep(1:2, each=4, times=1)

> gender

[1] 1 1 1 1 2 2 2 2
```

The call to the `rep` function above has three simple arguments.

1. The set of numbers to repeat. We have used the colon operator to generate the values 1 and 2. You could use the `c` function here to list any set of numbers you need.
2. The `each` argument tells it often to repeat each number in the set. Here we need four of each number.
3. The `times` argument tells it the number of times to repeat the (`set` by `each`) combination. For this example, we only needed one.

Note that while we are generating the gender variable as an easy example, `rep` did not create gender as a factor. To make it one, you would have to use the `factor` function. You could instead generate it directly as a factor using the more appropriate `gl` function.

Next, we generate the workshop variable by repeating each number in the 1:2 sequence only one time each but repeat that set four times.

```
> workshop <- rep(1:2, each=1, times=4)

> workshop

[1] 1 2 1 2 1 2 1 2
```

By comparing the way we generated gender and workshop, the meaning of the `rep` function's arguments should become clear.

Now we come to the only example that we actually needed for the `rep` function in this book. To generate a constant! We use a variation of this in Chapter 15, *Traditional Graphics*, to generate a set of zeros to use on a histogram.

```
> myZeros <- rep(0, each=8)

> myZeros

[1] 0 0 0 0 0 0 0 0
```

12.4 Generating Integer Measures

R's ability to generate samples of integers is easy to use and is in some respects similar to that of Stata. In other respects, as we will discover, they are quite dissimilar. You provide a list of possible values and then use the `sample` function to generate your data. First, we put the Likert scale values 1, 2, 3, 4, and 5 into myValues.

```
> myValues <- c(1, 2, 3, 4, 5)
```

Next, we set the random number seed using the `set.seed` function, so you can see the same result when you run it.

```
> set.seed(1234) #Set random number seed.
```

Finally, we generate a random sample size of 1,000 from myValues using the `sample` function. We are sampling with replacement so we can use the five values repeatedly.

```
> q1 <- sample(myValues, size=1000, replace=TRUE)
```

Now we can check its mean and standard deviation.

```
> mean(q1)

[1] 3.029

> sd(q1)

[1] 1.412854
```

To generate a sample using the same numbers but with a roughly normal distribution, we can change the values to have more as you reach the center.

```
> myValues <- c(1, 2, 2, 3, 3, 3, 4, 4, 5)
> set.seed(1234)
> q2 <- sample( myValues, 1000, replace = TRUE)

> mean(q2)
[1] 3.012

> sd(q2)
[1] 1.169283
```

You can see from the bar plots in Fig. 12.1 that our first variable follows a uniform distribution (left) and our second one follows a normal distribution (right). Do not worry about how we created the plot; we will cover than that in Chapter 15, "Traditional Graphics."

We could have done the latter example more precisely by generating 1,000 samples from a normal distribution and then chopping it into 5 equally spaced groups. We will cover generating samples from continuous samples in the next section.

```
> barplot( table(q1) )
> barplot( table(q2) )
```

If you would like to generate two Likert-scale variables that have a mean difference, you can do so by providing them with different sets of values from which to sample. In the example below, we nest the call to the `c` function, to generate a vector of values, within the call to the `sample` function. Notice that the vector for q1 has no values greater than 3 and q2 has none less than 3. This difference will create the mean difference. Here we are only asking for a sample size of 8.

```
> set.seed(1234)
```

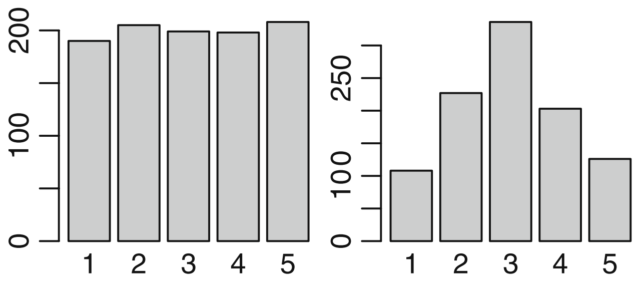

Fig. 12.1. Barplots showing the distributions of our generated integer variables.

```
> q1 <- sample( c(1, 2, 2, 3), size=8, replace = TRUE)

> mean(q1)
[1] 1.75

> set.seed(1234)

> q2 <- sample( c(3, 4, 4, 5), size=8, replace = TRUE)

> mean(q2)
[1] 3.75
```

12.5 Generating Continuous Measures

You can generate continuous random values from a uniform distribution using the `runif` function.

```
> set.seed(1234)

> x1 <- runif(n=1000)

> mean(x1)
[1] 0.5072735

> sd(x1)
[1] 0.2912082
```

where the `n` argument is the number of values to generate. You can also provide `min` and `max` arguments to set the lowest and highest possible values, respectively. So you might generate 1,000 pseudo-test-scores that range from 60 to 100 with

```
> set.seed(1234)

> x2 <- runif(n=1000, min=60, max=100)

> mean(x2)
[1] 80.29094

> sd(x2)
[1] 11.64833
```

Normal distributions with a mean of 0 and standard deviation of 1 have many uses. You can use the `rnorm` function to generate 1,000 values from such a distribution with

```
> set.seed(1234)

> x3 <- rnorm(n=1000)

> mean(x3)
[1] -0.0265972

> sd(x3)
[1] 0.9973377
```

You can specify other means and standard deviations as in the following example:

```
> set.seed(1234)

> x4 <- rnorm(n=1000, mean=70, sd=5)

> mean(x4)
[1] 69.86701

> sd(x4)
[1] 4.986689
```

We can use the hist function to see what two of these distributions look like, see Fig. 12.2.

```
> hist(x2)
> hist(x4)
```

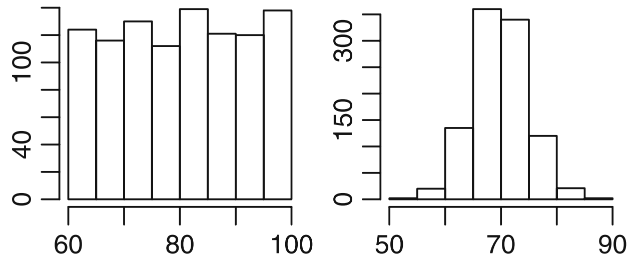

Fig. 12.2. Histograms showing the distributions of our continuous generated data.

12.6 Generating a Data Frame

Putting all of the above ideas together, we can use the following function calls to create a data frame similar to our practice data set, with a couple of test scores added. We are not bothering to set the random number generator seed, so each time you run this, you will get different results.

```
> id <- 1:8

> workshop <- gl( n=2, k=1,
+   length=8, label=c("R","Stata") )

> gender <-   gl( n=2, k=4,
+   length=8, label=c("f","m")   )

> q1 <- sample( c(1, 2, 2, 3), 8, replace = TRUE)
> q2 <- sample( c(3, 4, 4, 5), 8, replace = TRUE)
> q3 <- sample( c(1, 2, 2, 3), 8, replace = TRUE)
> q4 <- sample( c(3, 4, 4, 5), 8, replace = TRUE)

> pretest   <- rnorm( n=8, mean=70, sd=5)
> posttest  <- rnorm( n=8, mean=80, sd=5)

> myGenerated <- data.frame(id, gender, workshop,
+   q1, q2, q3, q4, pretest, posttest)

> myGenerated
  id gender workshop   q1 q2 q3 q4  pretest posttest
1  1      f        R    1  5  2  3 67.77482 77.95827
2  2      f      Stata  1  4  2  5 58.28944 78.11115
3  3      f        R    2  3  2  4 68.60809 86.64183
4  4      f      Stata  2  5  2  4 64.09098 83.58218
5  5      m        R    1  5  3  4 70.16563 78.83855
6  6      m      Stata  2  3  3  3 65.81141 73.86887
7  7      m        R    2  4  3  4 69.41194 85.23769
8  8      m      Stata  1  4  1  4 66.29239 72.81796
```

A more complex example is presented in Appendix D.

12.7 Example Programs for Generating Data

12.7.1 Stata Program for Generating Data

```
* Filename: GenerateData.do
```

```
clear
set obs 8

* ID
gen byte id = _n

* workshop
gen byte workshop = mod(_n, 2)
replace workshop = 2 if workshop==0

* gender
gen gender = int(runiform()+.5)
label define gender 0 "f" 1 "m"
label values gender gender


* Q variables
forvalues i = 1/8 {
forvalues j = 1/4 {
 cap gen q`j' = round(runiform() * `i' *`j')+1
  }
}

* Pretest and Posttest
gen pretest  = int(normal(runiform())*100)
gen posttest = int(normal(runiform())*100)

list
```

12.7.2 R Program for Generating Data

```
# Filename: GenerateData.R

# Simple sequences.

1:10
seq(from=1,to=10,by=1)
seq(from=5,to=50,by=5)
seq(5,50,5)

# Generating our ID variable
id <- 1:8
id
```

```
# gl function Generates Levels.

gl(n=2,k=1,length=8)
gl(n=2,k=5,length=8)

#Adding labels.

workshop <- gl(n=2,k=1,length=8,label=c("R","Stata"))
workshop

gender <- gl( n=2,k=4,length=8, label=c("f","m") )
gender

# The rep function.

# Simple sequences.

1:10
seq(from=1,to=10,by=1)
seq(from=5,to=50,by=5)
seq(5,50,5)

# Generating our ID variable
id <- 1:8
id

# gl function Generates Levels.

gl(n=2,k=1,length=8)
gl(n=2,k=5,length=8)

#Adding labels.

workshop <- gl(n=2,k=1,length=8,label=c("R","Stata"))
workshop

gender <- gl( n=2,k=4,length=8, label=c("f","m") )
gender

# Generating uniformly distributed Likert data

myValues <- c(1, 2, 3, 4, 5)
set.seed(1234)


# Simple sequences.
```

```
1:10
seq(from=1, to=10, by=1)
seq(from=5, to=50, by=5)
seq(5, 50, 5)

# Generating our ID variable
id <- 1:8
id

# gl function Generates Levels.

gl(n=2,k=1,length=8)
gl(n=2,k=5,length=8)

#Adding labels.

workshop <- gl(n=2,k=1,length=8,label=c("R","Stata"))
workshop

gender <- gl( n=2,k=4,length=8, label=c("f","m") )
gender

# Generating repetitious Patterns (Not Factors).

gender <- rep(1:2, each=4, times=1)
gender

workshop <- rep(1:2, each=1, times=4)
workshop

myZeros <- rep(0, each=8)
myZeros

# Generating uniformly distributed Likert data

myValues <- c(1, 2, 3, 4, 5)
set.seed(1234)
q1 <- sample( myValues, size=1000, replace = TRUE)
mean(q1)
sd(q1)

# Generating normally distributed Likert data

myValues <- c(1, 2, 2, 3, 3, 3, 4, 4, 5)
```

```
set.seed(1234)
q2 <- sample( myValues , size=1000, replace = TRUE)
mean(q2)
sd(q2)

# Plot details in Traditional Graphics chapter.
par( mar=c(2, 2, 2, 1)+0.1 )
par( mfrow=c(1, 2) )
barplot( table(q1) )
barplot( table(q2) )
# par( mfrow=c(1, 1) ) #Sets back to 1 plot per page.
# par( mar=c(5, 4, 4, 2)+0.1 )

# Two Likert scales with mean difference
set.seed(1234)
q1 <- sample( c(1, 2, 2, 3), size=8, replace = TRUE)
mean(q1)

set.seed(1234)
q2 <- sample( c(3, 4, 4, 5), size=8, replace = TRUE)
mean(q2)

# Generating continuous data

# From uniform distribution.
# mean=0.5
set.seed(1234)
x1 <- runif(n=1000)
mean(x1)
sd(x1)

# From a uniform distribution
# between 60 and 100
set.seed(1234)
x2 <- runif(n=1000, min=60, max=100)
mean(x2)
sd(x2)

# From a normal distribution.

set.seed(1234)
x3 <- rnorm(n=1000)
mean(x3)
sd(x3)
```

```
set.seed(1234)
x4 <- rnorm(n=1000, mean=70, sd=5)
mean(x4)
sd(x4)

# Plot details are in Traditional Graphics chapter.
par( mar=c(2,2,2,1)+0.1 )
par( mfrow=c(1,2) )
hist(x2)
hist(x4)
# par( mfrow=c(1,1) ) #Sets back to 1 plot per page.
# par( mar=c(5,4,4,2)+0.1 )

# Generating a Data Frame.

id <- 1:8
workshop <- gl( n=2, k=1,
  length=8, label=c("R","Stata") )
gender <-   gl( n=2, k=4,
  length=8, label=c("f","m")   )
q1 <- sample( c(1, 2, 2, 3), 8, replace = TRUE)
q2 <- sample( c(3, 4, 4, 5), 8, replace = TRUE)
q3 <- sample( c(1, 2, 2, 3), 8, replace = TRUE)
q4 <- sample( c(3, 4, 4, 5), 8, replace = TRUE)
pretest   <- rnorm( n=8, mean=70, sd=5)
posttest  <- rnorm( n=8, mean=80, sd=5)

myGenerated <- data.frame(id, gender, workshop,
  q1, q2, q3, q4, pretest, posttest)
myGenerated
```