POL 345: Quantitative Analysis and Politics

Precept Handout 4

Week 5 (Verzani Chapter 6: 6.2)

Remember to complete the entire handout and submit the precept questions to the Blackboard DropBox 24 hours before precept. In this handout, we cover the following new materials:

- Using `table()`, `prop.table()`, and `addmargins()` to summarize multivariate data
- Using loops `for(i in X){}` to repeat operations and functions
- Using `print()` to print objects in the **R** console
- Using `cat()` to concatenate (i.e. paste) a set of texts and/or objects together and print them
- Generating "container" vectors using `rep(NA , n.obs)`
- Evaluating conditional statements within loops using `if(){}`, `if(){}else{}`, and `if(){}else if(){}else{}`

1 Two-Way Tables

Another way to summarize the relationship between two variables is the use of two-way tables.

- The function `table(X, Y)` will create a two-way table using two variables `X` and `Y`.

```
> ## Minimum wage data from precept 2
> njmin <- read.table("njmin.txt", header=T)
> ## convert chain variable to factor before summarizing
> njmin$chain <- as.factor(njmin$chain)
> ## create a factor variable for state
> njmin$state <- as.factor(ifelse(njmin$location == "PA", "PA", "NJ"))
> ## two-way table
> chains <- table(njmin$state, njmin$chain)
> chains

   burgerking kfc roys wendys
NJ        118  65   73     35
PA         31  10   15     11
```

- The function `prop.table()` will convert a table to a table with proportions.

```
> prop.table(chains)

   burgerking       kfc      roys    wendys
NJ  0.3296089 0.1815642 0.2039106 0.0977654
PA  0.0865922 0.0279330 0.0418994 0.0307263
```

- The function `addmargins()` will append the sums for both rows and columns onto our tables. The function `addmargins()` may be used with both `table()` and `prop.table()`, generating useful summaries of the data by category.

```
> addmargins(chains)

    burgerking kfc roys wendys Sum
NJ         118  65   73     35 291
PA          31  10   15     11  67
Sum        149  75   88     46 358
```

```
> addmargins(prop.table(chains))

    burgerking       kfc      roys    wendys       Sum
NJ   0.3296089 0.1815642 0.2039106 0.0977654 0.8128492
PA   0.0865922 0.0279330 0.0418994 0.0307263 0.1871508
Sum  0.4162011 0.2094972 0.2458101 0.1284916 1.0000000
```

2 Loops

In many situations, we want to repeat the same calculations with different inputs. Loops allow you to avoid writing many similar code chunks. The function `for(i in X)` will create a loop in your programming code where `i` is a counter and `X` is a vector for the counter. That is, the following syntax,

```
for (i in X) {
  command1...
  command2...
  ...
}
```

will execute the code chunk, `command1...  command2...  ...`, the same number of times as the length of `X` vector while setting the counter `i` to each element of `X`. You can have as many commands and lines in a loop as you like. Additionally, you may use any letter to denote the counter. Comments can be written into the loop as with any other code chunk in **R**. Braces `{` and `}` are used to denote the beginning and end of your loops. A simple example is given below:

```
> for (j in 3:5){
+   x <- j*2
+   print(j)  # simple way of printing out
+   cat(j, "times 2 is equal to", x, "\n") # nicer print out
+ }

[1] 3
3 times 2 is equal to 6
[1] 4
4 times 2 is equal to 8
[1] 5
5 times 2 is equal to 10
```

where the function `cat()` will concatenate (i.e. paste) a set of texts and/or objects together (each should be separated by a comma) and then print the information to the **R** console. Note that `\n` changes a line. The function `print()` is similar and prints out any single object but less flexible than `cat()`.

- It is a good idea to indent the code chunk so that the loop can be easily detected. Built in **R** script editor and most text editors (when the settings are set to **R**) will automatically indent code chunks within a loop.
- How to debug the code that involves loops? Since a loop simply executes the same commands many times, one should first check whether the commands that go inside of the loop can be executed without any error. In the above example, one may simply try the following command before constructing the loop,

  ```
  > j <- 4
  > x <- j*2
  > cat(j, "times 2 is equal to", x, "\n")
  ```

```
4 times 2 is equal to 8
```

- When using loop, we often want to store the results from each iteration. The function `rep(X, Y)` can be used to create a vector of length `Y` with each item equal to `X`. We often use this function to generate "container" vectors to store each evaluation of the loop.

```
> Z <- rep(NA, 10)
> Z

 [1] NA NA NA NA NA NA NA NA NA NA

> for (j in 1:10){
+   Z[j] <- j*2
+ }
> Z

 [1]  2  4  6  8 10 12 14 16 18 20
```

- We now provide a more interesting example using the dataset **justices.RData** from Blackboard. This data set was used in Precept Handout 2. We determine the median ideal point by Supreme Court term.

```
> load("justices.RData")
> ## create a vector of the Supreme Court Terms
> SCterms <- unique(justices$term)
> ## create a container vector to store the evaluation of each loop
> median.ip <- rep(NA, length(SCterms))
> names(median.ip) <- SCterms # label each element
> for(i in 1:length(SCterms)){
+   X <- justices[justices$term == SCterms[i], ] # subset to SC term
+   median.ip[i] <- median(X$idealpt)
+ }
> summary(median.ip)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-0.974   0.216   0.555   0.407   0.711   1.120
```

Recall that the same calculation can be more efficiently done by `tapply()`.

```
> summary(tapply(justices$idealpt, justices$term, median))

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-0.974   0.216   0.555   0.407   0.711   1.120
```

Clearly, in this case, a loop is inefficient (though it may be more intuitive) and so `tapply()` should be used. But the example illustrates the idea of loops. Here is another example:

```
> median.names <- rep(NA, length(SCterms))
> names(median.names) <- SCterms
> for (i in 1:length(SCterms)){
+   median.names[i] <- justices$justice[justices$term == SCterms[i] &
+                                       justices$idealpt == median.ip[i]]
+ }
> ## number of terms a justice served as the median justice
> table(median.names)

median.names
      Black    Blackmun     Brennan      Burton       Clark Frankfurter
          3           2           1           3           6           3
   Goldberg      Harlan     Kennedy    Marshall    O'Connor      Powell
          2           1           9           2           8           3
       Reed      Souter     Stewart       White
          3           2           3          13
```

3 Conditional Statements

Earlier in the semester, we learned the `ifelse()` function. Here we consider another form of conditional statements. Again, it is important to indent the code such that code chunks within each conditional statement can be easily detected.

- The following syntax

  ```
  if (X) {
    command1...
    command2...
    ...
  }
  ```

  will execute the code chunk, `command1...` `command2...` if the condition `X` is met. If the condition is not met, then it will not execute that code chunk.

- Conditional statements can be used within a loop. Here is an example where we print out the names of presidents.

  ```
  > pres <- "Nobody"
  > for (i in 1:nrow(justices)) {
  +   if (pres != justices$pres[i]) {
  +     cat("In", justices$term[i], as.character(justices$pres[i]),
  +         "became the president.\n")
  +     pres <- justices$pres[i]
  +   }
  + }
  ```

```
In 1945 Truman became the president.
In 1953 Eisenhower became the president.
In 1961 Kennedy became the president.
In 1964 Johnson became the president.
In 1969 Nixon became the president.
In 1975 Ford became the president.
In 1977 Carter became the president.
In 1981 Reagan became the president.
In 1989 Bush became the president.
In 1993 Clinton became the president.
In 2001 WBush became the president.
```

- `if(){}else{}` statements allow for greater flexibility by incorporating commands to be executed if the conditional statement is `FALSE`. The following syntax will execute the code chunk, `command1... command2...` if `X` is true and the code chunk, `command3... command4...` if `X` is false.

```
if (X) {
  command1...
  command2...
  ...
} else {
  command3...
  command4...
  ...
}
```

- One can construct even more complicated conditional statements using `else if () { }` function. For example, the following syntax will execute the code, `command1... command2... ...`, if the condition `X` is met. If `X` is not met but `Y` is met, then the code `command3... command4... ...`, will be executed. Otherwise, the code, `command5... command6... ...`, will be executed. `else if` can be repeated many times or you may just have `if` and `else` as above.

```
if (X) {
  command1...
  command2...
  ...
} else if (Y) {
  command3...
  command4...
  ...
} else {
  command5...
  command6...
  ...
}
```

- Another way to construct complicated conditional statements is to nest multiple `if ()` statements. Remember that you should always use appropriate indentation so that the nesting structure of conditional statements are clear to readers of your code. For example,

```
if (X) {
  command1...
  if (Y) {
    command2...
    ...
  }
} else {
  command3...
  if (Z) {
    command4...
    ...
  } else {
    command5...
    ...
  }
}
```

- We return to the minimum wage example and explore the relationship between minimum wage increase and employment. Again, we begin by calculating the difference in the full time employment between before and after the minimum wage increase for each store in both states.

```
> njmin <- read.table("njmin2.txt", header = TRUE)
> njmin$diff <- njmin$fullAfter - njmin$fullBefore # difference
```

We now create a new factor variable which is equal to `BigIncrease` (`BigDecrease`) if the difference for a store is positive (negative) and larger than one standard deviation of the full time employment before the minimum wage increase. This variable equals `SomeIncrease` (`SomeDecrease`) if the difference is above half standard deviation but below one standard deviation. Finally, if the difference is less than half standard deviation, the variable will be equal to `LittleChange`.

```
> sds <- tapply(njmin$fullBefore, njmin$state, sd) # sd for each state
> njmin$Change <- rep(NA, nrow(njmin)) # create container variable
> for(i in 1:nrow(njmin)){
+   if (njmin$state[i] == "NJ") { # choose appropriate sd
+     sd.state <- sds[1]
+   } else {
+     sd.state <- sds[2]
+   }
+   if (njmin$diff[i] > sd.state) {
+     njmin$Change[i] <- "BigIncrease"
+   } else if (njmin$diff[i] < -sd.state) {
+     njmin$Change[i] <- "BigDecrease"
+   } else if (njmin$diff[i] > sd.state/2) {
+     njmin$Change[i] <- "SmallIncrease"
```

```
+   } else if (njmin$diff[i] < -sd.state/2) {
+     njmin$Change[i] <- "SmallDecrease"
+   } else {
+     njmin$Change[i] <- "LittleChange"
+   }
+ }
```

We create two-way tables of proportions with margins where rows correspond to states.

```
> njmin$Change <- as.factor(njmin$Change)
> addmargins(table(njmin$state, njmin$Change))
```

```
     BigDecrease BigIncrease LittleChange SmallDecrease SmallIncrease
  NJ          42          53          130            29            37
  PA          15           8           38             2             4
  Sum         57          61          168            31            41

     Sum
  NJ 291
  PA  67
  Sum 358
```

For both states, the majority of restaurants had little change in employment.

4 Precept Questions

In preparation for precept, please answer the following questions based on the 2008 Presidential Election data, **e08.RData**, available on Blackboard. The file contains results from state polls during the 2008 race, as well as the election results in each state. The variables in the data set are as follows:

`State`	State in which poll was conducted
`state.abbv`	State abbreviation
`Pollster`	Organization conducting poll
`DaysToElection`	Days to the election
`Dem`	Predicted support for Obama
`GOP`	Predicted support for McCain
`EV`	State electoral votes

1. We want to know how well the state polls conducted just before the election did at predicting the margin of victory in those states. Begin by reading the data into **R**. Create a variable in the data set for Obama's margin over McCain. Next, create a variable for the election results by state. Additionally, create a data set containing only pre-election polls (strip out the election results by subsetting the data to include all data except the day of the election).

2. For each state, generate a poll prediction for the predicted margin of victory for Obama using only the latest polls from the state (i.e.: the mean of all polls taken in the state on the day closest to the election; note that this day may differ among states). Determine how accurate the poll predictions generated above were by subtracting the poll predictions from the election results of each state. Summarize the findings.