Empirical Reasoning Center
R Faculty Workshop (Summer 2016)
Session 3

1 User-defined Functions

One advantage of R is its ability to incorporate user-defined functions. Like the built-in functions above, user-defined functions can perform complex (or, not so complex) sets of operations many times (e.g., calculating means or estimating a parameter that requires you to hand-code an estimator). Functions allow you to write code once and implement it multiple times by calling the function. User-defined functions are especially useful because they save coding time and mitigate the risk of coding errors or bugs. You can use a function over and over again, so be sure to verify that your code works properly and correctly provides the output you want.

User-defined functions are composed of three key elements:

1. input objects, known as arguments
2. an operation or set of operations
3. a returned object or set of objects

Note that objects in the function are local to the function. Functions can take or return objects of any data type.

1.1 Defining Functions

Defining functions is relatively straightforward. Use the assignment operator to name your function, add the arguments and the operations, and return the output object(s). You can use built-in functions within your user-defined functions. Indeed, many functions in R actually are functions of functions. Basic function structure and syntax are provided below. Note the use of curly braces.

```
# some_function <- function(arg1, arg2, ...){
#
#       statements / operations
#
#       return(object)
#
# }
```

For example, you can define a function that adds two vectors.

```
rm(list=ls(all=TRUE))

library(dplyr)

setwd("/Users/patriciakirkland/Dropbox/Empipirical Reasoning Center/R Workshop")
```

```
addVectors <- function(a,b) {

    out.vec <- a + b

    return(out.vec)
```

```
}
```

1.2 Invoking user-defined functions

Once defined, your function will work like built-in R functions. Note that the objects you pass through a user-defined function do not have to have the same name as the argument—see the example below.

```
a <- 1:10
b <- -1:-10

addVectors(a, b)

##  [1] 0 0 0 0 0 0 0 0 0 0

x <- 1:10
y <- -1:-10

addVectors(x, y)

##  [1] 0 0 0 0 0 0 0 0 0 0

z <- addVectors(a=x, b=y)
```

Below is a much more complicated function for estimating cluster-robust standard errors. Many, many R users rely on this user-defined function (which they probably found on the internet).

```
cluster_se   <- function(dat,fm, cluster){
    require(sandwich, quietly = TRUE)
    require(lmtest, quietly = TRUE)
    M <- length(unique(cluster))
    N <- length(cluster)
    K <- fm$rank
    dfc <- (M/(M-1))*((N-1)/(N-K))
    uj  <- apply(estfun(fm),2, function(x) tapply(x, cluster, sum));
    vcovCL <- dfc*sandwich(fm, meat=crossprod(uj)/N)
    coeftest(fm, vcovCL) }
```

An example with the municipal finance data...

First, load the data and fit an OLS model

```
load("muni_finance_data_cleaned.RData")

fit_3 <- lm(Total.Expenditure.PC ~ Total.Taxes.PC
            + Population
            + Census.Region
            , data=COG.fips)
summary(fit_3)

##
## Call:
## lm(formula = Total.Expenditure.PC ~ Total.Taxes.PC + Population +
```

```
##     Census.Region, data = COG.fips)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8993 -0.5034 -0.2145  0.2300  8.6835
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)             5.931e-01  2.244e-02  26.435  < 2e-16 ***
## Total.Taxes.PC          1.650e+00  2.407e-02  68.530  < 2e-16 ***
## Population              4.173e-07  3.096e-08  13.478  < 2e-16 ***
## Census.RegionNortheast  3.747e-01  3.688e-02  10.158  < 2e-16 ***
## Census.RegionSouth      3.094e-01  2.681e-02  11.543  < 2e-16 ***
## Census.RegionWest      -1.303e-01  2.688e-02  -4.849 1.27e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8901 on 7707 degrees of freedom
## Multiple R-squared:  0.4898,Adjusted R-squared:  0.4895
## F-statistic:  1480 on 5 and 7707 DF,  p-value: < 2.2e-16
```

To get heteroskedasticity-robust standard errors, you can use the built-in `coeftest()` function.

```
library(lmtest)

## Loading required package:  zoo
##
## Attaching package:  'zoo'
## The following objects are masked from 'package:base':
##
##    as.Date, as.Date.numeric

library(sandwich)

# heteroskedasticity-robust standard errors
coeftest(fit_3, vcov=vcovHC(fit_3, type="HC1"))

##
## t test of coefficients:
##
##                           Estimate  Std. Error t value  Pr(>|t|)
## (Intercept)             5.9308e-01  2.1572e-02 27.4937 < 2.2e-16 ***
## Total.Taxes.PC          1.6497e+00  3.4732e-02 47.4977 < 2.2e-16 ***
## Population              4.1734e-07  3.7661e-08 11.0814 < 2.2e-16 ***
## Census.RegionNortheast  3.7468e-01  4.0670e-02  9.2125 < 2.2e-16 ***
## Census.RegionSouth      3.0940e-01  2.6809e-02 11.5408 < 2.2e-16 ***
## Census.RegionWest      -1.3031e-01  2.0485e-02 -6.3611 2.118e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

However, you could also define your own function to estimate robust standard errors.

```r
robust_se <- function(regmodel){
    require(sandwich, quietly = TRUE)
    require(lmtest, quietly = TRUE)
    coeftest(regmodel, vcov=vcovHC(regmodel, type="HC1"))
}
```

Examples

```r
# robust SEs with our user-defined function
robust_se(fit_3)

##
## t test of coefficients:
##
##                             Estimate  Std. Error t value  Pr(>|t|)
## (Intercept)               5.9308e-01  2.1572e-02 27.4937 < 2.2e-16 ***
## Total.Taxes.PC            1.6497e+00  3.4732e-02 47.4977 < 2.2e-16 ***
## Population                4.1734e-07  3.7661e-08 11.0814 < 2.2e-16 ***
## Census.RegionNortheast    3.7468e-01  4.0670e-02  9.2125 < 2.2e-16 ***
## Census.RegionSouth        3.0940e-01  2.6809e-02 11.5408 < 2.2e-16 ***
## Census.RegionWest        -1.3031e-01  2.0485e-02 -6.3611 2.118e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
# cluster-robust standard errors
cluster_se(COG.fips, fit_3, COG.fips$fipsid)

##
## t test of coefficients:
##
##                             Estimate  Std. Error t value  Pr(>|t|)
## (Intercept)               5.9308e-01  5.0197e-02 11.8152 < 2.2e-16 ***
## Total.Taxes.PC            1.6497e+00  7.8029e-02 21.1417 < 2.2e-16 ***
## Population                4.1734e-07  9.9522e-08  4.1934 2.779e-05 ***
## Census.RegionNortheast    3.7468e-01  9.9119e-02  3.7801  0.000158 ***
## Census.RegionSouth        3.0940e-01  6.5415e-02  4.7298 2.287e-06 ***
## Census.RegionWest        -1.3031e-01  5.0369e-02 -2.5870  0.009699 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

1.3 Source files

One very convenient feature of R is that you can use `source()` files so that you do not have to include every user-defined function you want to use in a script. If there are user-defined functions that you use often, you can create an R script that includes one or more user-defined function(s), and call the functions using the `source()` function. Many R users make one source file that they use in every script, while others make a source file for each project, assignment, or class.

Below is an example to illustrate. Start by clearing the workspace to remove the functions we defined above. You can use the `source()` function at the start of a script when you clear the workspace, set your working directory, and load packages.

```r
## clear the workspace
rm(list=ls(all=TRUE))

setwd("/Users/patriciakirkland/Dropbox/Empiprical Reasoning Center/R Workshop")

source("ERC R Workshop Source.R")
```

For this example, start by loading data and running a regression model.

```r
load("muni_finance_data_cleaned.RData")

fit_3 <- lm(Total.Expenditure.PC ~ Total.Taxes.PC
            + Population
            + Census.Region, data=COG.fips)
summary(fit_3)

##
## Call:
## lm(formula = Total.Expenditure.PC ~ Total.Taxes.PC + Population +
##     Census.Region, data = COG.fips)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8993 -0.5034 -0.2145  0.2300  8.6835
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)               5.931e-01  2.244e-02  26.435  < 2e-16 ***
## Total.Taxes.PC            1.650e+00  2.407e-02  68.530  < 2e-16 ***
## Population                4.173e-07  3.096e-08  13.478  < 2e-16 ***
## Census.RegionNortheast    3.747e-01  3.688e-02  10.158  < 2e-16 ***
## Census.RegionSouth        3.094e-01  2.681e-02  11.543  < 2e-16 ***
## Census.RegionWest        -1.303e-01  2.688e-02  -4.849 1.27e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8901 on 7707 degrees of freedom
## Multiple R-squared:  0.4898,Adjusted R-squared:  0.4895
## F-statistic:  1480 on 5 and 7707 DF,  p-value: < 2.2e-16
```

```r
# robust SEs with our user-defined function (from source file)
robust_se(fit_3)

##
## t test of coefficients:
##
##                              Estimate  Std. Error t value  Pr(>|t|)
## (Intercept)                5.9308e-01  2.1572e-02 27.4937 < 2.2e-16 ***
## Total.Taxes.PC             1.6497e+00  3.4732e-02 47.4977 < 2.2e-16 ***
## Population                 4.1734e-07  3.7661e-08 11.0814 < 2.2e-16 ***
## Census.RegionNortheast     3.7468e-01  4.0670e-02  9.2125 < 2.2e-16 ***
```

```
## Census.RegionSouth      3.0940e-01  2.6809e-02 11.5408 < 2.2e-16 ***
## Census.RegionWest      -1.3031e-01  2.0485e-02 -6.3611 2.118e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# cluster-robust standard errors (from source file)
cluster_se(COG.fips, fit_3, COG.fips$fipsid)

##
## t test of coefficients:
##
##                          Estimate  Std. Error t value  Pr(>|t|)
## (Intercept)             5.9308e-01  5.0197e-02 11.8152 < 2.2e-16 ***
## Total.Taxes.PC          1.6497e+00  7.8029e-02 21.1417 < 2.2e-16 ***
## Population              4.1734e-07  9.9522e-08  4.1934 2.779e-05 ***
## Census.RegionNortheast  3.7468e-01  9.9119e-02  3.7801  0.000158 ***
## Census.RegionSouth      3.0940e-01  6.5415e-02  4.7298 2.287e-06 ***
## Census.RegionWest      -1.3031e-01  5.0369e-02 -2.5870  0.009699 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

2 Automating tasks— an example

This is just one example of how you might use R to automate tasks. Other tasks you could automate include querying databases (e.g. MS Access files), scraping web pages, and collecting data from social media sites. Of course you can also take a similar approach to running simulations, randomization inference, running multiple model specifications, and many other tasks.

```
rm(list=ls(all=TRUE))

library(dplyr)

setwd("/Users/patriciakirkland/Dropbox/Empipirical Reasoning Center/R Workshop")
```

In this example, you will use the `paste()` function and a loop to automate assembly of a dataset. This example uses Census of Government municipal finance data. The historic database comes in a series of text files with three files for each year. One way to build a time-series cross-sectional dataset is to read in the files for each year and bind them by column. The next step is to append each year.

The code below automates this process.

```
#### build annual .csv files from COG text files
directory <- "/Users/patriciakirkland/Dropbox/Census of Governments/_IndFin_1967-2007"
year <- 2000:2007

COG.muni <- data.frame()

for(j in year){
```

```r
i <- substr(as.character(j), 3, 4)

COG.a <- read.csv(paste0(directory, "/", "IndFin", formatC(i, width = 2, flag = "0"), "a",
                         ".txt"))

COG.b <- read.csv(paste0(directory, "/", "IndFin", formatC(i, width = 2, flag = "0"), "b",
                         ".txt"))

COG.c <- read.csv(paste0(directory, "/", "IndFin", formatC(i, width = 2, flag = "0"), "c",
                         ".txt"))

COGmerge <- left_join(COG.a, COG.b)

COGmerge <- left_join(COGmerge, COG.c)

COG.muni.temp <- subset(COGmerge, Type.Code == 2)

COG.muni <- rbind(COG.muni, subset(COGmerge, Type.Code == 2))

}

## Joining, by = c("SortCode", "Year4", "ID")
## Joining, by = c("SortCode", "Year4", "ID")
## Joining, by = c("SortCode", "Year4", "ID")
## Joining, by = c("SortCode", "Year4", "ID")
## Joining, by = c("SortCode", "Year4", "ID")
## Joining, by = c("SortCode", "Year4", "ID")
## Joining, by = c("SortCode", "Year4", "ID")
## Joining, by = c("SortCode", "Year4", "ID")
## Joining, by = c("SortCode", "Year4", "ID")
## Joining, by = c("SortCode", "Year4", "ID")
## Joining, by = c("SortCode", "Year4", "ID")
## Joining, by = c("SortCode", "Year4", "ID")
## Joining, by = c("SortCode", "Year4", "ID")
## Joining, by = c("SortCode", "Year4", "ID")
## Joining, by = c("SortCode", "Year4", "ID")
## Joining, by = c("SortCode", "Year4", "ID")
```