Center for Teaching, Research & Learning
CTRL Lab
American University, Washington, D.C.
http://www.american.edu/ctrl/
202-885-3862

Introduction to R

R is a programming language and an environment for statistical computing and graphics. It is a open source project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R provides a wide variety of statistical (linear and nonlinear modeling, classical statistical tests, time-series analysis, classification, clustering, etc.) and graphical techniques, and is highly extensible.

Course Objective

This course is designed to give a basic understanding of the R language and basic statistical analysis in R.

Learning Outcomes

1. Basic building blocks of an R program
2. Data management, operations, and simple transformations.
3. Data transformation
4. Descriptive Statistics
5. Basic plotting with R
6. Accessing R libraries and installing additional packages
7. Addition information
8. Exporting your data

1. Basic R Windows

- *R Console*: Default window when you open R. This is the command-line interface/output window.
- *R Editor*: Click on **File** > **New Script**. This is the primary script/code window. You run commands in this window by pressing **CTRL + R** or run certain lines of code by highlighting the lines and pressing these commands.
- *Data Editor*: When you've loaded data, type **fix(*name_of_dataset*)** to view and edit your data in this window.

Creating Vectors Within R

Vectors are one-dimensional arrays that can hold numeric, character, or logical data. Vectors can be created by using the arrow function within R.

x1<-2
x_2<-3
X.3<-7

The arrow implies that youa re assigning the variable (x1) with the number 2. One thing to keep in mind is that R's programming language is VERY sensitive, so x1 does not equal X1 or x_1.

Creating vectors/variables with multiple number assingments can be done by using the *c()* function.

X5<-c(1,2,3,4,5,6)

We have now created the variable X5 with multiple data points within it.

We can also create a variable where a set of number repeats, *rep()*, x amount of times.

X6<-rep(c(1,2,3,4),3)

We can also perform matmatical function within R.

2+3^2

x1+x_2^2

y<-x_2^x1

y

We can also list all the varaibles we've created or from a dataset by using the *ls()* function and remove them with the *rm()* function.

ls()

Note: by keeping the space inside the parenthesis blank, it uses our current workspace within the R software. If we used a dataset name within the parenthesis, it would list all the variables within the dataset.

rm(x1)

We can also remove an entire list of varaibles.

rm(list=ls())

2. Data management

R gives its users a wide variety of data management options. R can import data files in text format. It can also import data generated from statistical packages such as S-Plus, Stata, SPSS, SAS, and SYSTAT, in their native format. In addition, you can use standard SQL strings to gain access to relational databases, including MS SQL Server, MySQL, Oracle, IBM-DB2, etc.

Importing data (from a csv file)

You can set your working directory using the *setwd* function. When specifying the path, use either double back slashes or single forward slashes.

setwd("C:/")

or

setwd("C:\\")

You can also check your current working directory by using the *getwd* function.

getwd()

Then we use the *read.csv* function to import data from a comma-delimited text file, *nations.csv*, and create the data frame *data*. For the assignment operator, R will accept either <- or =. As an example data file, we will use the nations.csv data. This file is saved in J:\CLASSES\RSG\Tutorial Data\nations.csv.

As an example data file, we will use the nations.csv data. This file is saved in J:\CLASSES\RSG\Tutorial Data\nations.csv.

We use the *read.csv* function to import data from a comma-delimited text file, *nations.csv*, and create the data frame *data*. For the assignment operator, R accepts <- .
data1<-read.csv("J:\\CLASSES\\RSG\\Tutorial Data\\nations.csv", header=TRUE)

The first argument of the *read.csv* function is your file name with a file path. The second argument indicates whether the variable names are depicted in the first row of the text file (if not set it to FALSE.)

If you would rather click through a window to find your file, you can use the file chooser function within the *read.csv* function.
data<-read.csv(file.choose(),header=TRUE)

This function will allow you to assign the file of your choice from the pop-up window to the data frame name.

Viewing data

At any time, you can view your data using the *fix* function, which will pull up a separate window that will allow you to manipulate your dataset within the dataset itself.
fix(data)

You can also just view the data set in a separate window using the *View()* function. In contrast to the fix() function, you can not manipulate and edit your data set with this function.

You can view the variable names only byusing the *colnames()* or *names()*function.
colnames(data) or **names(data)**
or the **rownames(data)**

We can even pinpoint one observation within the whole dataset using the *data[x,y]*. Notice that we are using brackets [ ] instead of parenthesis ( ). The [ ] tell R that we are specifically looking inside the data frame itself. The *x* represents the row number and *y* the column number.
data[2,3]
Allows us to see only the row.
data[,3]
Allows us to see only the column.
data[2,]

If we want to pull up only one variable, we can just type out the data set and variable name with a *$* separating the two. The "data$" part makes sure to concatenate the new variable "humandev" to the dataset named "data".

data$humandev

We can also use the function *attach* to allow you to directly refer to the variable names available in the data set. The function *names* will list the variable names in the dataset. You will need to attach a data set again if you modify or add a variable winthin the data set.

attach(data)

3. Data Transformation

You can transform you data many different ways within R. One of the common problems is when your numeric data is considered a character string.

One way we can check to see whether our variables are considered numeric or charcters is through the *str()* function.

str(data)

is.numeric(calories)
is.character(humandev)
is.factor(country)

If your variables are imported as characters but should be considered numeric we can use the *as.numeric()* function to convert them.

data$calories2<-as.numeric(calories)
attach(data)

We are revering back to using data$ before the variable name because we are creating a new variable that we wish to be apart of the dataset data. Other wise, we could leave the $ out at it would just save into our current workspace.

4. Descriptive statistics

In this section we analyze our data using descriptive statistics. We begin by using the *summary* function as a catch all for many descriptive statistics. We will also be estimating the mean, median, standard deviation, and variance of a couple of variables within the dataset.

The *summary* is a bit of a catch all fuction for descriptive statiscs. This fuction will automatically produce the min, 1st quartile, median, mean, 3rd quartile, max, and number of missing NA's.

summary(data)
summary(humandev)

We can also calculate individual statisics like the mean, median, standard deviation, variance, maximum, and minimum.

mean(popul)
mean(humandev)

In some cases, the data you are trying to use might have missing values. In this case, you would not be able to use the function mentioned above, unless you explicitly specify to omit the missing values. You can do that by introducing a second argument into the above functions (where "na" and "rm" can be read as "not available" and "remove", respectively).
mean(humandev,na.rm=TRUE)

We can also do the same with…

Median
median(popul)

Standard Deviation
sd(popul)

Variance
var(popul)

Logarithm
log(popul)

Maximum
max(popul)

Minimum
min(popul)

Range
range(popul)

Quantiles (defaults to 0,.25,.5,.75,1)
quantile(popul)

We can also look at the relationship between two variables with corvariance and correlation functions.

Covariance
cov(popgrowt,drate)

Correlation
cor(popgrowt,drate)

Correlation test with confidence intervals, t-values, p-values, df and correlation value.
cor.test(popgrowt,drate)

Frequency and Counts of Variables

Instead of looking at the basic mean, median, and mode, we can also look at the count of a variable by using the table function
table(democrac)
table(democrac, unemploy)

5. Plotting

In this section we will create three different ways of visualizing out data. We will create a boxplot, a histogram, and a scatterplot.

To create a boxplot we can use the *boxplot()* function.
boxplot(drate)
summary(drate)

Histrograms can be created using the *hist()* function.
hist(drate)

We can also modify the amount of columns and main title of the histogram using a few extensions of the function. The *break=* function allows us to dictate the number of bars that appear in the histrogram. The *main=*function allows to to title the main title of the historgram.
hist(drate, breaks=50, main="Frequency of Death Rate")

If you want to know more about the historgram function simple type in *?hist* in to the consule. A pop-up window will appear with a description of the function, its different extensions with explanations, and examples.
?hist

Scatterplots allow us to visualize every point in our dataset and can be made using the *plot()* function.
plot(drate~calories)
Note that when using the tild(~), the first variable will transfer to the y-axis and the second variable the x-axis.
We can also play with some exptensions of the *plot()* function.
plot(drate~calories, col="red", xlab="Caloric Intake",ylab="death rate", pch=2, xlim=c(1500,4250), ylim=c(0,30))
The *col=* allows us to change the color of the datapoints, *xlab=* and *ylab=* relabels the titles of the x and y axis, *pch=* changes the shape of the data points, and *xlim=* and *ylim=* change the limits of the x and y axis.

We can also create a scatterplot matrix with three varaibles instead of two.
pairs(~drate+calories+humandev,main="Simple Scatterplot Matrix",pch=4)
Since we don't have a true y axis, all the variable are on the right of the tild(~) and have plus signs (+) in between them to indicate that they are in addition to the x axis.

6. Libraries

Another benefit to using R is that you're not confined to the basic function of the program itself. With R being an open-source software, statistician and computer programers are constantly creating packages that contain different and more specific functions for different types of analysis or plotting of graphs. For example, laavan is a package for doing Structural Equation Modeling.

To install a package, you will select 'Tools' from the toolbar above. Select 'Install Packages' and type in the package you wish to download.
install.package(lattice)
To make the package you just downloaded active, you need to call it up from they library using the *library()* function.
library(lattice)

Our previous plot was good and accurate, but it the data was obviously not linear.

We can use the *xyplot()* function in the lattice package to create a scatterplot with a lowess line, a non-parametric line.

xyplot(calories~humandev,type=c("smooth", "p"), col="darkblue")

7. Other Additonal Information

If at any time you need help with a function, just type **help(name_of_function)** in the Editor or Console. For example:

help(fix)

To insert comments in your code in the Editor window, use #:

 # Comment code like this

8. Exporting Your Data

The following code allows us to export our data in to a .csv file format. File= denotes the file's name and type. Row.names=TRUE tells R that the first row will be the variable names, sep="," tells R to separate each cell by a common. You can also replace it with a / or ! or any character you wish. In this case, were making a .csv file, so we want to have a comma.

write.table(data,file="nations2.csv", row.names=TRUE, sep=",")

You can also export the files into other software file types (STATA, SAS, etc)

install.packages("foreign")

library(foreign)

write.foreign(data,
"C:\\Users\\rp7436a\\Desktop\\data.sps","C:\\Users\\rp7436a\\Desktop\\data.sps",
package="SPSS")