```
##############################################################################
## Math Prefresher, Fall 2011
## Jennifer Pan
##############################################################################

## This file walks you through some basic things with R
## Thanks to Patrick Lam, Maya Sen, and Iain Osgood for letting me borrow
## their materials

## To download R, please go to http://www.r-project.org/
## and follow the directions. You can also access R through
## any of the computers in the HMDC lab and also some of the FAS
## computers (for example, in the science center).

############################################################
## PREAMBLE: Script and Console
############################################################

## R works by
## 1. typing code in the console (next to the red 'greater than'
## sign)
## 2. OR by typing on a script (like this one) and sending the lines of
## code to the console.

## You can use the R Editor (thge R GUI, pronounced "gooey")
## or you can use from a variety of free R Editors, like Tinn R.
## Unless you are working on very simple calculations,
## you should always save your code as a script using a ".R" file extension.

## Every line that begins with '#' is ignored by the console.
## So you can add comments to your code without generating syntax errors.
## In fact, your code should be heavily commented so that you can understand
## what you've done if you had to come back to it later.


############################################################
## R help!
############################################################

## Use the "help" command; it's your friend.

?mean

## or

help(mean)


############################################################
## R as a calculator
############################################################

2 + 18

50821/6

21^4

log(4)/9^2


## R is an object-oriented programming language
## Store an object for later retrieval by using " <- " as assignment operator
```

```
a <- 2 + 18
a

b <- 50821/6
b

my.name <- "Jen"
my.name


## Use the ls() command to see what objects are currently stored
## in the R environment

ls()

## Use the rm() command to get rid of a partcular object stored

rm(my.name)
ls()

## Use the rm(list = ls()) command to get rid of all objects stored

rm(list = ls())
ls()



############################################################
## Objects: Basics
############################################################

###### Storing objects (again)

## R can store objects, and you can call these up later.
## As noted above objects are defined using " <- "

prefresher <- "fun"
prefresher

## Don't name your objects things like "mean" or "sum"
## or "7" since those are things that R already has pre-packaged.

###### Vectors

## All objects consist of one or more vectors.
## Vector is a combination of elements (i.e. numbers, words)
## Vectors can be created using c(), :, seq(), rep()

one.to.five <- c(1,2,3,4,5)
one.to.five

today <- c("Prefresher","is","fun")
today

one.to.ten <- 1:10
one.to.ten

two.to.ten <- seq(from = 2, to = 10, by = 1)
        # need to fill in three arguments for seq()
two.to.ten

five.fours <- rep(4, times=5)
```

```
five.fours


############################################################
## Objects: Data Types
############################################################

## All elements in a vector must be of the same data type!
bad.vec <- c(1, "fun") # will change 1 to a character
bad.vec

## There are 3 data types: numeric, character, logic

###### Numeric: numbers, e.g., 2, 4, 10.987

four <- 4
four

is.numeric(four)
        # check if an object is the numeric data type

as.numeric("4")
        # change to numeric data type (here from character to numeric)

###### Character: words for phrases must be in " ", e.g., "fun" "10"

president <- "Barack Obama"
president

is.character(president)
        # check if an object is the character data type

as.character(4)
        # change to character data type (here from numeric to character)

###### Logic: TRUE (T) or FALSE (F)

is.character(president)
is.numeric(four)
        # from before

logical.vec <- c(is.character(president), is.numeric(four))
        # combine with c() into a vector
logical.vec

is.logical(logical.vec)
        # check if an object is the logic data type

## logic data type can also be represented with 1's (TRUE) and 0's (FALSE)
as.numeric(logical.vec)


############################################################
## Objects: Object Classes
############################################################

## We know now what an object is
## We know that object can contain three types of data
## We've played around with vectors, which are strings of stuff
## In addition to vectors, there are 4 other object classes:
## Matrix, Array, Dataframe, List

###### Matrix: two-dimensional (row x column) object
```

```
a.matrix <- matrix(c(1, 2, 3, 4), nrow = 2, ncol = 2)
        # arguments include the data, number of rows, number of columsn
        # by default, data is filled in by column a.matrix

## All elements in a matrix must be of the same data type

###### Array: three-dimensional (row x column x height) object

## Simiarly, all elements in an array must be of the same data type

an.array <- array(0, dim = c(2, 2, 3))
        # arguments include the data, and dimensions (row, column, height)
an.array

###### Dataframe: a data frame is also a two-dimensional (row x column) object

## Each column must be of the same data type
## But data type may vary by column

## Regression and other statistical functions usually use dataframes
## Use as.data.frame()to convert matrices to dataframes

as.data.frame(a.matrix)

###### List: a set of objects, each object can be any data class

list(five.fours, a.matrix, an.array)


############################################################
## Playing around with Objects
############################################################

###### Combining

## Combine vectors or lists with c(), for concatenate

vec1 <- c(4, 6, 9)
vec2 <- 10:15
comb.vec <- c(vec1, vec2)
comb.vec

## Combine matrices or dataframes with other matrices,
## dataframes, or vectors, use cbind()or rbind()

a.matrix <- matrix(0, nrow = 2, ncol = 3)
a.matrix        #2 by 3 matrix of 0's

rbind(a.matrix, 1:3)    #add a row with 1, 2, 3

cbind(a.matrix, 4:5)    #add a column with 4,5

## Note: dimensions much match
cbind(a.matrix, 4:6)    #doesn't work

###### Do math on numeric objects

vec1*3

vec1+vec2       # notice what this did, since vectors are different lengths

a.matrix + 2
```

```
###### Give names to elements, rows, columns within objects

## Giving name makes it easier to isolate the things you want
## Use names() to name vectors, dataframes and lists
## Use rownames()and colnames() to name rows and columns in matrices and dataframes
## Use dimnames() to name height in arrays

leaders <- c("Obama", "Merkel", "Hu")
leaders

names(leaders) <- c("US", "Germany", "China")
leaders # vector with names

age <- c(49,56,67)
dataset <- data.frame(cbind(leaders, age))
dataset

###### Subsetting (use [] to pick out elements of an object)

## Recall: vec1 is c(4,6,9); vec2 is 10:15

vec1[1]         # first element
vec2[6]         # sixth element
vec2[-1]                # everything but the first element
vec2[c(2,4)]    # second and fourth element
vec2[c(2:4)]    # second to fourth element

## You can also replace a particular element of a vector
vec1[3] <- 500
vec1

## Recall dataset, a dataframe, with leaders and ages
## We can subset this data.frame

dataset

dataset[1,]             # first row
dataset[-1,]    # all but first row
dataset[,1]             # first column
dataset[,"age"] # calls column by name
dataset$leaders # retrieve columns, only for dataframes

## Similarly, you can replace a particular element of a dataframe

dem <- c(1,1,1)

dataset <- cbind(dataset, dem)
        # add a column to the dataset
dataset

dataset[3, "dem"] <- 0
        # make China a 0 (third row, "dem" column)
dataset

## We can also subset using logical statements

dataset$dem == 1
        # look at which elements of the "dem" columns are 1's

dataset[dataset$dem == 1, ]
        # take out the rows where "dem" is 1
```

```
## Other operators include: ==, !=, >, <, >=, <=, & for logical statements


############################################################
## Basic Functions
############################################################

## R has many preprogrammed functions that manipulate objects.
## To use a function, you type the function name followed by the
## arguments in parentheses

## Recall:
vec1 <- c(4, 6, 9)
vec2 <- 10:15

sum(vec1)       # sum of all elements of vec1
sum(vec1,vec2)  # sum of all elements vec1 and vec2

prod(vec1)      # product of all elements of vec1

max(vec2)       # max element in vec2
min(vec2)       # min element in vec2

mean(vec1)      #mean of vec1
median(vec2)    #median of vec2

length(vec2)  #number of elements, useful to calculate sample size

unique(vec2)  #lists unique values (gets rid of repeats)
sort(vec2)  #puts elements in order from smallest to largest, shows value
order(vec2)  #puts elements in order from smallest to largest, shows element#

## We can do additional analysis with the following functions:

sd(vec2)
var(vec2)
quantile(vec2)

## As you probably now realize, c(), rep(), seq() are all functions

## You can store the output of a function, as a new object.

sample.size <- length(vec2)
sample.size

## Use help to to learn the arguments for a particular function
?sort

############################################################
## Making R interact with your computer and the WWW
############################################################

## Working directory is the "folder" where R loads and saves data

getwd()

## To change the working directory, use "setwd()"

setwd("C:/Users/Jpan/Documents/Grad school/2010-2011/PreFresher/Materials/R")

## R can load all kinds of data easily

ls()
```

```
rm(list = ls())


## 1. From working directory, saved in a previous R sessions as "*.RData"

load("cambridge.RData")
ls()

## 2. From working directory, saved as a text document "*.txt"

lalonde <- read.table("lalonde.txt")
ls()

## 3. From working directory, saved as a csv document "*.csv"

pakistan <- read.csv("pakistan.csv")
ls()

## 4. From someone else's website, saved as a text document

nes <- read.table("http://www.people.fas.harvard.edu/~blackwel/nes.dat")
nes


############################################################
## R packages
############################################################

## People write software for R all the time.
## These are called "packages" and add to R's functionality

## Useful packages are:
## foreign -- permits you to load data formatted for other software (e.g., stata)
## xtables -- helps you write up tables in LaTeX code
## car -- has many datasets and linear regresison functions
## zelig -- many useful functions, as well as datasets
## more packages are at http://cran.r-project.org/web/packages/

## A great website with info on cutting-edge stuff that
## people are writing is http://www.r-bloggers.com/

install.packages("foreign")
        # this must be done only once

library(foreign)
     # this must be done every time you use the foreign library

## Once you install and load foreign, you can get data such a .dta files
## From working directory, saved as a STATA document "*.dta"

ccarddata <- read.dta("ccarddata.dta")
ls()


############################################################
## Getting a feel for the data
############################################################

## There are a number of really useful commands that will
## quickly give you a feel for any data:

## Earlier we loaded cambridge.RData, within it is a dataset called "loans"
ls()
```

```
class(loans)    # tells you the object class
head(loans)             # first few observations, top of dataset
dim(loans)              # the dimensions of the dataset (rows by columns)
nrow(loans)             # number of rows
ncol(loans)             # number of columns
names(loans)    # column (generally variable) names
summary(loans)  # summary statistics

## We can call up individual columns (variables) like before

loans$amount

## We can grab certain rows

loans[c(105,216,307,415,430),]

## And certain columns

loans[,c("hisp","income")]
head(loans[,c("hisp","income")]) #can put head() on the outside

## We can subset data using logical operators

men <- loans[loans$sex == "Male",]
women <- loans[loans$sex == "Female",]  ## or loans$gender != "Male"

mean(men$income)
mean(women$income)

## There are some additional functions for dataframes and matrices

colMeans(loans)
        # doesn't work because all values have to be numeric

head(loans)
        # sex is currently not numeric, but we can make it numeric

## Let's use the function without analyzing sex (the 5th column)
colMeans(loans[,-5])
rowMeans(loans[,-5])
colSums(loans[,-5])
rowSums(loans[,-5])


## Let's re-code Male as 0 and Female as 1
loans$sex[loans$sex == "Male"] <- 0
loans$sex[loans$sex == "Female"] <- 1

head(loans)  #now gender is in terms of 0's and 1's



######################################################################
## Creating and Saving Figures
######################################################################

load("cambridge.RData")

## You can make a histogram, basic

hist(loans$rate)
```

```
## You can make it pretty

hist(loans$rate, col = "gold", xlim = c(3,10),
        xlab = "Interest rate", ylab = "# applicants",
        main = "Cambridge Mortgage Rates 2006")

## You can add a vertical line at the mean:

abline(v = mean(loans$rate), col = "blue", lty="dashed", lwd=2)

## You can add a legend

legend(x="topright", legend=c("mean"), col="blue", lty="dashed", lwd=2)

## To save the file:
## Note: Close the window before running

pdf(file= "LoanHistogram.pdf")
hist(loans$rate, col = "gold", xlim = c(3,10),
        xlab = "Interest rate", ylab = "# applicants",
        main = "Cambridge Mortgage Rates 2006")
abline(v = mean(loans$rate), col = "blue", lty="dashed", lwd=2)
legend(x="topright", legend=c("mean"), col="blue", lty="dashed", lwd=2)
dev.off()

## We can also do different kinds of figures

## Scatterplot
plot(x=loans$income, y=loans$amount, xlab = "Income", ylab = "Amount",
        xlim=c(0,600), main = "Scatterplot")

## Density Plot
plot(density(loans$rate), xlab = "Interest rate", ylab = "Density",
        main = "Density plot, cambridge mortgage rates 2006")

## We can multiple plots with par()
par(mfrow = c(1, 2))
 plot(x=loans$income, y=loans$amount, xlim=c(0,600),main = "", xlab = "", ylab = "")
 plot(density(loans$rate), main = "", xlab = "", ylab = "")


######################################################################
## Plots Continued
######################################################################

## load the ccarddata dataset
library(foreign)
ccarddata <- read.dta("ccarddata.dta")

## scatterplot of the age and credit card expenditure with best fit line
plot(x=ccarddata$age, y=ccarddata$credit_card_expend,
  xlab="age", ylab="Credit Card Expenditure", main="Scatterplot")
lines(lowess(x=ccarddata$age, y=ccarddata$credit_card_expend, f=0.5), col="purple")

## same scatterplot, but identify the income of points
plot(x=ccarddata$age, y=ccarddata$credit_card_expend,
  xlab="age", ylab="Credit Card Expenditure", main="Scatterplot")
identify(x=ccarddata$age, y=ccarddata$credit_card_expend,label=ccarddata$income)

## scatterplot of rent and own in different colors with separate best fit lines
#sub-set the data
c.own <- ccarddata[ccarddata$own_rent == 1,]
summary(c.own)
```

```
c.rent <- ccarddata[ccarddata$own_rent == 0,]
summary(c.rent)

#make the plot
plot(x=c.own$age, y=c.own$credit_card_expend, col="green", pch=19)
points(x=c.rent$age, y=c.rent$credit_card_expend, col="yellow", pch=19)
lines(lowess(x=c.own$age, y=c.own$credit_card_expend, f=1),
  col="green", lwd=2)
lines(lowess(x=c.rent$age, y=c.rent$credit_card_expend, f=1),
  col="yellow", lwd=2)


######################################################################
## Creating Tables
######################################################################

## The two main functions for creating Latex tables in R are xtable()
## and latex().  Neither function is in the base package, so first you need to
## install one of the relevant libraries.

## To use xtable():

install.packages("xtable")
library(xtable)

## The easiest way to use a function like xtable() is to collect
## everything you want in your table into a matrix.
## matrix(), as.matrix(), cbind() and rbind() are useful here.

## Let's make an example matrix:

example <- matrix(data = 1:12, nrow = 4)
example

## I can add row and column names to my matrix with

rownames(example) <- c("Row 1", "Second row", "Mumero tres", "Mr. 4")
colnames(example) <- c("A variable", "Some means", "Something Else")

## Now we can generate the code to write the matrix as a latex table

xtable(example)


############################################################
## Writing Functions
############################################################

## You can write your own functions in R using the "function" command
## Functions can be really complicated or really simple

## my.function <- function(x,y,z){  # tells R that this is a function
## out <- crazy function stuff  # the meat of the function (you usually "tab" this)
## return(out)                          # returns the output of the function
## }                                            # closes the function up


## Example 1) This function will take three numbers as arguments;
## it will add the first two and divide the sum by the third

my.function <- function(x,y,z){
  out <- (x + y)/z
  return(out)
```

```
}

## Now we call our function with either of the following:

my.function(x = 5, y = 10, z = 3)
my.function(5, 10, 3)

## Example 2) This function deletes the first
## p percent of observations from a dataframe

load("cambridge.RData")
dim(loans)      #929 rows in original data

trim.func <- function(x, p = .1){   # trims first 10% of observations
  n <- nrow(x) # number of observations
  trim.number <- round(p*n) # number to delete (rounded)
  trimmed.data <- x[-c(1:trim.number),] # deletes from top
  return(trimmed.data)
}

trimmed.loans <- trim.func(x=loans)
dim(trimmed.loans)      #836 rows remaining


############################################################
## Some More Helpful functions
############################################################

###### apply() function

## The apply() function takes a function typically for vectors
## and applies it on each row or column of a matrix, dataframe, or array.

load("cambridge.RData")

loan.medians <- apply(loans, MARGIN = 2, FUN = median)
        # MARGIN argument gets 1 for row and 2 for column
loan.medians

###### sample() function

## The sample() function allows for sampling from a vector

population <- loans$amount
samp.w.rep <- sample(population, size = 10, replace = T)
samp.w.rep

samp.wo.rep <- sample(population, size = 10, replace = F)
samp.wo.rep

###### for loop

## A for loop is a way of repeating a process a vast
## number of times.  For each iteration of the loop
## R keeps an index number stored which can be handy.

## A for loop is computationally intensive, so use as last resort
## Many things can be done using apply() instead

holder <- c()
holder

for(i in 1:100){
```

```
  draw <- rnorm(1, mean = 0, sd = 1)
        # here we are drawing one observation from a normal distribution
        # with mean zero and standard dev 1
  holder[i] <- draw
}

holder



############################################################
## Matrix operations in R
############################################################

## R has many canned functions for matrix operations

## Create two example matrices
mat1 <- matrix(3:6, nrow=2, ncol=2, byrow=T)
mat2 <- matrix(1:4, nrow=2, ncol=2)

## Transpose the matrix
t(mat1)

## Obtain the diagonal elements of a matrix
diag(mat1)

## Find the determinant of the matrix
det(mat1)

## Find the inverse of the matrix
solve(mat1)

## Perform matrix multiplication
mat1 %*% mat2


#####################################################################
## More resources
#####################################################################

## Stackoverflow is a great place to search for answers to questions
## http://cran.r-project.org/doc/contrib/usingR.pdf (the Maindoland intro)
## http://www.r-bloggers.com/ (the R bloggers website)
```