```
#1.1_Basics_of_R

#==============================================================================
 =>
# To set working directory to the source file folder:
# Session -> Set working directory -> To Source file location
# setwd("~/Dropbox/Prog_Codes/Prog_Workshop/Session_1")
#==============================================================================
 =>
## Creating, indexing, subsetting and operating vectors

# Create a vector using concatenation

z = 1 # assign a scalar value to z
z # print out value of z to the console

v = c(0,1,7,-2) # create a vector v
v # print vector v to console
w = c(1,3,-12,0) # create a vector w
w # print vector w to console

#-----------------------------------------------Ø>
# Arithmetic operations on the vectors and scalars
z + v
w + v
w*v
w/v
# Note: the above operations are element-wise

#----------------------------------------------->
# Index an element of a vector
w[2] # second element of w
w[10] # ??
length(w)
# cannot access an element beyond the length of the vector
#----------------------------------------------->
# Subset a vector
v[1:3] # first 3 elements of v
v[c(2,4)] # 2nd and 4th arguments of v

#==============================================================================
 =>
## Creating, indexing, subsetting and operating matrices

# Create a  2 x 2 matrix m

(m = matrix(c(3,-4.2,-7.1,0.95),nrow=2,ncol=2))
# what does ( ) around code do?
#----------------------------------------------->
# Fill matrix by rows; default is by column
(m = matrix(1:6, nrow=2, byrow=T))
#----------------------------------------------->
# Index 2 row
m[2,]
#----------------------------------------------->
```

```
# Index 3rd column
m[,3]
#------------------------------------------------>
# (2,3)'th element of the matrix m
m[2,3]
#------------------------------------------------>
# Arithmetic operations on Matrices
m-2

m/5
# Note: these operations are element-wise

#------------------------------------------------>
# Matrix multiplication:
set.seed(40); M1 = matrix(runif(9),3,3); M1
set.seed(42); M2 = matrix(runif(9),3,3); M2
# Can you explain the above steps?

# M3 is the product of matrices M1 and M2
M3 = M1 %*% M2
M3 # print M3 to console
#------------------------------------------------>
# Transpose
t(M1) # transpose of M1
#------------------------------------------------>
#determinant
det(M1) # determinant of M1
det(M2)
det(M3)
#------------------------------------------------>
# Matrix inverse
solve(M1)
solve(M2)
solve(M3)
#------------------------------------------------>
# Eigen values
(ev = eigen(M1))
ev$values
ev$vectors

# is M1 positive definite?

#==========================================================================
 =>
## Creating, indexing, subsetting and operating lists
# Lists are a more flexible (than vectors and matrices) of storing objects
 in R
# List can store objects of different types, character, integers, real
 numbers,
# complex numbers, etc.
# create a vector of strings

s = c("Kofi","Kojo","Ziggy")

L = list(sc=z,v1=v,v2=w,m=m,M1=M1,M2=M2,M3=M3,s=s)
```

```
L
# Access elements of list L (either way works)
L$sc
L[[1]]

L$s
L[[8]]

L$M1
L[[5]]

#=============================================================================
=>
## Loading data into R
# Let us load the .csv data from the working directory. This is the mroz
data
# is taken from the Jeffery Wooldridge's textbook website
# Make sure the data set is in the working directory

dat<- read.csv("dat.csv",header = T,sep = " ")
names(dat); dim(dat) # check column names and dimension of matrix
nr = dim(dat)[[1]] # extract number of rows
nc = dim(dat)[[2]] # extract number of columns
nr
nc
#----------------------------------------------->
## Manipulating the data set
# create a matrix of two columns age and experience
xx = as.matrix(cbind(dat$age,dat$experience))

# split data set into two, even and odd-indexed rows
eI = (1:floor(nr/2))*2 # even indices;
head(eI)
tail(eI)

#what are the functions floor(), head(), and tail() doing?

eDat<- cbind(dat$y[eI],xx[eI,]) # subset even-indexed observations of y and
xx
oDat<- cbind(dat$y[-eI],xx[-eI,]) # odd-indexed observations of y and xx
# NB: the negation of an index is all but those observations, odd indices in
# our case
summary(dat$y)
summary(xx)
summary(eDat)
summary(oDat)
# compare even-indexed observations to odd-indexed ones. any differences?

#=============================================================================
=>
## Plotting data in R

# simple scatter plot
plot(dat$nonwife)
#----------------------------------------------->
```

```
# plot a histogram
hist(dat$nonwife)
#------------------------------------------------>
# a histogram with 30 bins
hist(dat$nonwife, breaks=30, main = "Histogram of non-wife income",
     xlab = "Non-wife income in $")
# use "main = ", "xlab = ", "ylab = " to provide main title, label for the
# horizontal axis and for the vertical axis respectively.
#------------------------------------------------>
# a kernel density plot (a continuous version of the histrogram)
plot(density(dat$nonwife),main = "Kernel density plot of non-wife income",
     xlab = "Non-wife income in $")
# only use kernel density plot if your variable is truly continuous
#------------------------------------------------>
# Function plots in R
par(mfrow=c(1,2))
curve(sin,-4*pi,4*pi)
curve(cos,-4*pi,4*pi)
par(mfrow=c(1,1))
#==============================================================================
 =>
## Logicals in R

# Logicals are useful mainly for verifying whether statements are true or
# false

# Examples:
# 1. Verify equality
2 == 3 # is 2 equal to 3? # Note == is logical, = assigns value to the LHS
2 != 3 # is 2 not equal to 3?
2<3 # is 2 less than 3?
2>=3 # is 2 greater or equal to 3?
#------------------------------------------------>
which(w==0) # which element(s) of vector w equals 0?
w[which(w==0)] # extract the such element in w
which(v==12) # which element of vector v equals 12?
which(w%%2==0) #indices of even numbers in w i.e the modulo of which numbers
 =0?
w[which(w%%2==0)] #even numbers in w
w[-which(w%%2==0)] #non-even numbers in w
#or
w[-which(w%%2!=0)]

#------------------------------------------------>
any(w< -1) # any element of w less than -1? NB. ensure space between < and -
w %in% v # which elements of w in v?
all(w==v) # are vectors w and v exactly equal, i.e. element-wise?
w==v # check element-wise equality
#==============================================================================
 =>
## if/else statements

# These statements enable us to carry out a task only if conditions are
#  satisfied.
i=3
```

```
if(w[i]<0){
  print(paste(w[i],"is a negative number"))
}else if(w[i]>0){
  print(paste(w[i],"is a positive number"))
}else{
  print(paste(w[i],"is neither positive nor negative"))
}

# change the value of i to other indices and see what happens

# a programme to print whether number is odd or even
i=3
if(w[i]%%2==0){
  print(paste(w[i],"is an even number"))
}else{
  print(paste(w[i],"is an odd number"))
}

#---------------------------------------------->
# To apply conditional execution to each element of a vector, use the
 function
# ifelse:

set.seed(333)
x = round(rnorm(10),2); head(x)
y = ifelse(x>0, 1, -1); head(y)
rbind(x,y) #row bind x and y


#=============================================================================
 =>
## Loops
# Loops enable a repetition of steps for a given number of times of until
 some
# condition is met

# for loop: suitable for a finite number of steps known before hand
sum = 0 #initialise sum
for (i in 1:10) sum = sum + i
sum
sum(1:10)

pr = 1
for(j in 1:10) pr = pr*j
pr

#---------------------------------------------->
# For a slightly more complicated example, sum over only even numbers:
sum = 0
for (i in 1:10){
  if (i%%2 == 0) sum = sum + i
  }
# the use of curly brackets for a loop is advisable if you have several
 steps
sum
```

```
# Exercise: sum only over odd numbers from 1 through 20

#---------------------------------------------->
# while loop: suitable for a known stopping criterion but not the number of
# steps

# We use a while loop to report how many steps it takes to get to position
# greater than 10, and what that position is given random increments taken
# from the normal distribution mean .5, standard deviation 1.
x=0
n=0
set.seed(333)
# set seed when using random number generation for reproducibility of
 results
while (x <= 10) {
  n=n+1
  x=x+rnorm(1,mean=.5,sd=1)
  }

print(paste ("n = ", n, ", x = ",round(x,2) )) #print out results

# Exercise: Use a while loop to search on the interval [-2,4] for the
 maximum
# of f(x) = -(x-1)^2.
# Use increments of 0.001

#============================================================================
 =>
## User defined functions in R

# Functions in R are key for executing tasks in an orderly way. They take
 input
# and give output.

# The general form of a function definition is
# f = function(x,y,...) expression involving x, y, ...
# The result of the function will be the last evaluated expression, unless
# return(function value) is used

# Here's a simple function that calculates the first three powers of a
 vector
# and arranges the result as a matrix.
#---------------------------------------------->
powers = function(x) {
  matrix(c(x,x^2,x^3),nrow=length(x),ncol=3)
  }
vv = 1:5
powers(vv)
#---------------------------------------------->
# A Cobb-Douglas production function

CDP = function(K,L) (K^0.4) * (L^0.6)
# Example:
CDP(200,40)
```

```
# vary inputs and verify output

# Exercise: code the following function: f(x) = -(x-1)^2. Plot this function
 using
# the curve() function over the interval [-2,4]
#------------------------------------------------>
# A complicated example:
# A function to compute OLS results
OLS<- function(y,x){
  N = length(y) #obtain number of observations
  y = matrix(y,ncol = 1) # a matrix of column length 1
  x = as.matrix(cbind(1,x)) # include 1's for the intercept term
  k=ncol(x) # number of parameters to estimate
  beta = solve(t(x)%*%x)%*%t(x)%*%y # obtain parameters (k x 1 vector)
  res = y - x%*%beta # compute residuals
  df = N - k  #degree of freedom
  sig = sum(res^2)/df #compute sigma squared
  varcov<- sig*solve(t(x)%*%x) # compute variance-covariance matrix

  m = matrix(NA,nrow = 4,ncol = k) # a matrix to store regresion results
  m[c(1,2),] = rbind(t(beta),sqrt(diag(varcov))) # first two rows to store
   parameters
  # and standard errors
  t.stat = m[1,]/m[2,] # compute t statistics
  pval = 2*(1-pt(abs(t.stat),df)) #p values taken from the t distribution
  m[c(3,4), ] <- rbind(t.stat,pval) # store t-stats and p-values in 3rd and
   4th rows
  dimnames(m)[[1]]<- c("estimate", "std. error","t value","p value")
  # label the rows
  return(t(m))
}

# Example:
reg<-OLS(y=dat$nonwife,x=xx)
reg
round(reg,digits = 4) # round to 4 decimal places

# compare to the internal lm() R function
regI<- lm(dat$nonwife~xx)

summary(regI)
reg # for comparison

#------------------------------------------------>
# Write a log-likelihood function for the linear regression model with
# normally distributed errors

like<- function(y,x,pars){
  N = length(y) #obtain number of observations
  y = matrix(y,ncol = 1) # a matrix of column length 1
  x = as.matrix(cbind(1,x)) # include 1's for the intercept term
  k=ncol(x) # number of parameters to estimate
  np = length(pars) #obtain number of parameters (including sigma)
  beta = matrix(pars[-np],ncol = 1) #obtain column vector of parameters
  sig = pars[np]
```

```
  res<- y - x%*%beta
  ll = sum(dnorm(res,sd=sqrt(sig),log = T)) #obtain log joint likelihood
  return(ll)
}
# example:
like(y=dat$nonwife,x=xx,pars = rep(1,4))
#return log likelihood value for parameter values of 1's
like(y=dat$nonwife,x=xx,pars = reg[1,]) #evaluate the likehood at the OLS
 estimates

#---------------------------------------------->
# write and plot a piecewise function

piecefn<- function(x){
  if(x<0){
    y=-x^4
  }else if(x>2){
    y=(x-2)^3
  }else{
    y = 0
  }
  return(y)
}

piecefn=Vectorize(piecefn) # vectorize the function. why?
curve(piecefn,from = -4,to=10) #plot the curve
piecefn(-3)
piecefn(1)
piecefn(2.4)

# Exercise:
# write a function in R that takes integers as input and prints out if the
 number
# is negative, positive or zero.
#

#==========================================================================
 =>

## Generating random numbers in R
# Some times, we may want to obtain draws from a distribution or randomise
# certain operations. This can be done in a number of ways.

(x = runif(10)) # uniformly draw 10 numbers in the default interval [0,1]
# Repeat the above step a number of times. Are the numbers the same in each
 draw?
#---------------------------------------------->
# Now set seed to any number, say 40
set.seed(40) ; (x = runif(10))
# Repeat the above steps. What do you observe?
#---------------------------------------------->
# Make 10 000 draws from the normal distribution, mean 1, standard deviation
 1
set.seed(40) ; x = rnorm(10000,mean=1,sd=2)
# Make a density plot
```

```r
plot(density(x),main = "kernel density plot of x~N(1,2)")
#--------------------------------------------->
# Make 10 000 draws from the beta distribution
set.seed(40)
z = rbeta(1000,shape1 = 1, shape2 = 4)
plot(density(z),main = "kernel density plot of z~beta(1,4)")

#==========================================================================
 =>
```