

#1.1_Basics_of_R

```
#=====>
# To set working directory to the source file folder:
# Session -> Set working directory -> To Source file location

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

# Create a vector using concatenation

z = 1 # assign a scalar value to z

v = c(0,1,7,-2) # create a vector v

w = c(1,3,-12,0) # create a vector w

# Try arithmetic operations on the vectors and scalars
z + v
w + v
w*v

# Index an element of a vector
w[2] # second element of w

# Subset a vector
v[1:3] # first 3 elements 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))

# 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

# 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

# Transpose and determinant
t(M1) # transpose of M1
det(M1) # determinant of M1

# Matrix inverse
solve(M1)

#=====>
## Creating, indexing, subsetting and operating lists
# Lists are a more flexible (than vectors and matrices) of storing objects in R

# 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)

# 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

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

## 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;
#what is the function floor() 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

#=====>
## 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 histogram)
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
curve(sin, -4*pi, 4*pi)

#=====>
## 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 less than 3?
2 >= 3 # is 2 greater or equal to 3?

which(w==0) # which element of vector w equals 0?
which(v==12) # which element of vector v equals 12?

any(w< -1) # any element of w less than -1? NB. ensure space between < and -
w %in% v # is there any element of w in v?
all(w==v) # are vectors w and v exactly equal, i.e. element-wise?

#=====>
## if/else statements

# These statements enable us to carry out a task only if conditions are
# satisfied.
i=2
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

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

set.seed(333)

```

```

x = round(rnorm(10),2)
y = ifelse(x>0, 1, -1)
rbind(x,y) #row bind x and y

#=====>
## Loops
# Loops enable a repetition of steps for a given number of times or 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

# Exercise: Can you run a for loop for a product of numbers 1 through 10?

# 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

# 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

#=====>
## 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() is called

# 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

# A function to compute OLS parameters

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)
  m[c(1,2),] = rbind(t(beta),sqrt(diag(varcov)))
  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(tval,pval)
  dimnames(m)[[1]]<- c("estimate", "std. error","t value","p value")
  # label the rows
  return(t(m)) # round final results to 4 decimal places
}

# 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)

# 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 = c(rep(1,4)))
#return log likelihood value for parameter values of 1's

# 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

#=====>

## 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=1)
# Make a density plot
plot(density(x),main = "normal probability density")

# Make 10 000 draws from the beta distribution
set.seed(40)
x = rbeta(1000,shape1 = 1, shape2 = 4)
plot(density(x),main = "beta probability density")

#=====>

```