STAT 331 - Tutorial 1

1 Some R Basics

1.1 Vectors and assignment

- Use the function **c( )** to set up a numeric vector named y, say, consisting of five numbers, namely 10.4, 5.6, 3.1, 6.4 and 21.7,

  > y <- c(10.4, 5.6, 3.1, 6.4, 21.7)

  Notice the assignment operator ("<-"), which consists of the two characters "<" (less than) and "-" (minus) occurring strictly side-by-side and it "points" to the object receiving the value of the expression. In most contexts the "="operator can be used as an alternative.

- Assignment can also be made using the function **assign( )**.

  > assign("y", c(10.4, 5.6, 3.1, 6.4, 21.7)

1.2 Vector arithmetic

- Vectors can be used in arithmetic expressions, in which case the operations are performed element by element.

- The elementary arithmetic operators are the usual: $+$, $-$, $*$, $/$ and $\hat{}$ for raising to a power.

  > 1/y
  [1] 0.09615385 0.17857143 0.32258065 0.15625000 0.04608295

- The common arithmetic functions are available:
  log( ), exp( ), sqrt( ), max( ), min( ), sum(), prod( ), mean( ), var( ), etc.
  For example, sample variance of y can be calculated by

  > sum((y-mean(y))^2)/(length(y)-1)

1.3 Logical vectors

- The logical operators are <, <=, >, >=, == for exact equality and != for inequality.

- In addition if c1 and c2 are logical expressions, then c1 & c2 is their intersection ("and"), c1 | c2 is their union ("or"), and !c1 is the negation of c1.

- The elements of a logical vector can have the values TRUE, FALSE.

```
> temp <- y > 10
> temp
[1] TRUE FALSE FALSE FALSE TRUE
```

1.4 Arrays and matrices

- The function **array(data_vector, dim_vector)** can be used to generate matrices.

```
> x <- array(1:20, dim=c(5,2))
> x
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10
```

- Equivalently we can use
  > x <- matrix(1:20, nrow=5, ncol=2).
- Transpose of Matrix: the function **t( )**
- Matrix multiplication: the operator %∗%.
  > x %∗% t(x)
  is the matrix product.
  > x ∗ x
  is the matrix of element by element products
- Forming matrices using function **cbind( )** and **rbind( )**. Roughly **cbind( )** forms matrices by binding together matrices horizontally, or column-wise, and **rbind( )** vertically, or row-wise.

```
> cbind(y,x)
[1,] 10.4   1   6
[2,]  5.6   2   7
[3,]  3.1   3   8
[4,]  6.4   4   9
[5,] 21.7   5  10
```

1.5 Reading data from files

- Use the function **read.table( )**
  The **read.table( )** function will let you read in any type of delimited ASCII file (text file) that contain rectangular data. It can read in both numeric and character values. This is by far the easiest and most reliable method of entering data into R. To read an entire data frame directly, the external file will normally have a special form.

- The first line of the file should have a name for each variable in the data file.
- Each additional line of the file has the values for each variable.

For example, the first few lines of a file to be read as a data frame might look as:

Price	Floor	Area	Room	Age	Cent.heat
52.00	111.0	830	5	6.2	no
54.75	128.0	710	5	7.5	no
57.5	101.0	1000	5	4.2	no
...					

The function **read.table( )** can then be used to read the data frame directly.

```
> HousePrice <- read.table("houses.data", header=TRUE)
```

The default delimiter in **read.table( )** is the space delimiter. The **sep** argument is used to specify the separator/delimiter (e.g. sep=";").

- Accessing builtin datasets
  Around 100 datasets are supplied with R (in package datasets), and others are available in packages (including the recommended packages supplied with R). To see the list of datasets currently available use

```
> data( )
```

- Editing data
  When invoked on a data frame or matrix, edit brings up a separate spreadsheet-like environment for editing. This is useful for making small changes once a data set has been read. The command

```
> HousePriceNew <- edit(HousePrice)
```

will allow you to edit your data set **HousePrice**, and on completion the changed object is assigned to **HousePriceNew**.

1.6 Linear Models

- The basic function for fitting ordinary linear models is **lm( )**, and a streamlined version of the call is as follows:

```
> fitted.model <- lm(formula, data = data.frame)
```

For example

```
> fm <- lm(y ~ x1 + x2, data = production)
```

would fit a multiple regression model of **y** on **x1** and **x2** (with implicit intercept term). The

important (but technically optional) **parameter data = production** specifies that any variables needed to construct the model should come first from the production data frame. The value of **lm( )** is a fitted model object (technically a list of results of class "lm") which is saved in **fm**.

- Generic functions for extracting model information
  Information about the fitted model can be displayed, extracted, plotted and so on by using generic functions that orient themselves to objects of class "lm". A brief description of the most commonly used ones is given below.

  - **anova(object1, object2)**
    Compare a submodel with an outer model and produce an analysis of variance table.
  - **coef(object)**
    Extract the regression coefficient (matrix).
  - **deviance(object)**
    Residual sum of squares.
  - **formula(object)**
    Extract the model formula.
  - **plot(object)**
    Produce four plots, showing residuals, fitted values and some diagnostics.
  - **predict(object, newdata=dataname)**
    The data frame supplied must have variables specified with the same labels as the original. The value is a vector or matrix of predicted values corresponding to the variable values in **dataname**.
  - **print(object)**
    Print a concise version of the object. Most often used implicitly.
  - **resid(object)**
    Extract the (matrix of) residuals.
  - **step(object)**
    Select a suitable model by adding or dropping terms and preserving hierarchies. The model with the smallest value of AIC (Akaike's Information Criterion) discovered in the stepwise search is returned.
  - **summary(object)**
    Print a comprehensive summary of the results of the regression analysis.
  - **vcov(object)**
    Returns the variance-covariance matrix of the main parameters of a fitted model object.

2. Exercise: Fitting Simple Linear Regression Model

Suppose we would like to investigate the relationship between systolic blood pressure and gestational age among the low birth weight infants. The data file "*lowbwt.txt*" contains information for the sample of 100 low birth weight infants born in Boston, Massachusetts. Measurement of systolic blood pressure are saved under the variable name *sbp*, and values of gestational age under the name *gestage*. The data file is available on the LEARN course website.

You are provided with the following R code and output:

```
############################################################
# 2.1 Read data into R                                     #
############################################################

workdat <- read.table("lowbwt.txt", header=T)

x <- workdat$gestage
> x
  [1] 29 31 33 31 30 25 27 29 28 29 26 30 29 29 29 29 29 33 33 29 28 30 27 33 32 28 29
 [28] 28 29 30 31 30 31 29 27 27 27 32 31 28 30 29 28 31 27 25 30 28 28 25 23 27 28 27
 [55] 27 26 25 23 26 24 29 29 27 30 30 32 33 27 31 26 27 27 35 28 30 31 30 27 25 25 26
 [82] 29 29 34 30 29 33 30 29 24 33 25 32 31 31 31 29 32 33 28

y <- workdat$sbp
> y
  [1] 43 51 42 39 48 31 31 40 57 64 46 47 63 56 49 87 46 66 42 52 51 47 54 64 37 36 45
 [28] 39 29 61 53 64 35 34 62 59 36 47 45 62 75 44 39 48 43 19 63 42 44 25 26 27 35 40
 [55] 44 66 59 24 40 49 53 45 50 64 48 48 58 67 40 48 36 44 53 45 54 44 42 50 48 29 30
 [82] 36 44 46 51 51 43 48 52 43 42 48 49 62 45 51 52 47 40 50

n <- length(x)
> n
[1] 100

############################################################
# 2.2 Fitting a Simple Linear Regression Model using Direct Calculation      #
############################################################

############################
# Calculate LSE of b0 and b1 #
############################
b1 <- (sum(x*y)-n*mean(x)*mean(y))/(sum(x^2)-n*(mean(x))^2)
> b1
[1] 1.26438
```



```
b0 <- mean(y)-b1*mean(x)
> b0
[1] 10.55207b1
```

```
###################################
# Calculate Fitted Values and Residuals     #
###################################

yhat <- b0+b1*x
> yhat
  [1] 47.21908 49.74784 52.27660 49.74784 48.48346 42.16156 44.69032 47.21908 45.95470
 [10] 47.21908 43.42594 48.48346 47.21908 47.21908 47.21908 47.21908 47.21908 52.27660
 [19] 52.27660 47.21908 45.95470 48.48346 44.69032 52.27660 51.01222 45.95470 47.21908
 [28] 45.95470 47.21908 48.48346 49.74784 48.48346 49.74784 47.21908 44.69032 44.69032
 [37] 44.69032 51.01222 49.74784 45.95470 48.48346 47.21908 45.95470 49.74784 44.69032
 [46] 42.16156 48.48346 45.95470 45.95470 42.16156 39.63280 44.69032 45.95470 44.69032
 [55] 44.69032 43.42594 42.16156 39.63280 43.42594 40.89718 47.21908 47.21908 44.69032
 [64] 48.48346 48.48346 51.01222 52.27660 44.69032 49.74784 43.42594 44.69032 44.69032
 [73] 54.80536 45.95470 48.48346 49.74784 48.48346 44.69032 42.16156 42.16156 43.42594
 [82] 47.21908 47.21908 53.54098 48.48346 47.21908 52.27660 48.48346 47.21908 40.89718
 [91] 52.27660 42.16156 51.01222 49.74784 49.74784 49.74784 47.21908 51.01222 52.27660
[100] 45.95470

resid <- y-yhat
> resid
  [1]  -4.2190818   1.2521587 -10.2766008 -10.7478413  -0.4834615 -11.1615628 -13.6903223
  [8]  -7.2190818  11.0452980  16.7809182   2.5740575  -1.4834615  15.7809182   8.7809182
 [15]   1.7809182  39.7809182  -1.2190818  13.7233992 -10.2766008   4.7809182   5.0452980
 [22]  -1.4834615   9.3096777  11.7233992 -14.0122210  -9.9547020  -2.2190818  -6.9547020
 [29] -18.2190818  12.5165385   3.2521587  15.5165385 -14.7478413 -13.2190818  17.3096777
 [36]  14.3096777  -8.6903223  -4.0122210  -4.7478413  16.0452980  26.5165385  -3.2190818
 [43]  -6.9547020  -1.7478413  -1.6903223 -23.1615628  14.5165385  -3.9547020  -1.9547020
 [50] -17.1615628 -13.6328033 -17.6903223 -10.9547020  -4.6903223  -0.6903223  22.5740575
 [57]  16.8384372 -15.6328033  -3.4259425   8.1028170   5.7809182  -2.2190818   5.3096777
 [64]  15.5165385  -0.4834615  -3.0122210   5.7233992  22.3096777  -9.7478413   4.5740575
 [71]  -8.6903223  -0.6903223  -1.8053603  -0.9547020   5.5165385  -5.7478413  -6.4834615
 [78]   5.3096777   5.8384372 -13.1615628 -13.4259425 -11.2190818  -3.2190818  -7.5409805
 [85]   2.5165385   3.7809182  -9.2766008  -0.4834615   4.7809182   2.1028170 -10.2766008
 [92]   5.8384372  -2.0122210  12.2521587  -4.7478413   1.2521587   4.7809182  -4.0122210
 [99] -12.2766008   4.0452980

############################
# Check Residuals                  #
############################

> sum(resid)
[1] 6.181722e-13
> sum(resid*x)
[1] 4.639844e-12
> sum(resid*yhat)
[1] 1.291056e-11
```

```
##########################
# Estimate sigma^2       #
##########################

s2 <- sum(resid^2)/(n-2)
> s2
[1] 120.9893

##########################
#Test H_0:  b1=0         #
##########################

sxx <- sum(x^2) - n*(mean(x))^2
se.b1 <- sqrt(s2/sxx)
> se.b1
[1] 0.4362311

tstats <- b1/se.b1
> tstats
[1] 2.898417

pvalue <- 2*(1-pt(tstats, n-2))
> pvalue
[1] 0.004628003

##########################
#Analysis of Variance    #
##########################

SST <- var(y)*(n-1)
> SST
[1] 12873.36

SSE <- sum(resid^2)
> SSE
[1] 11856.95

SSR <- sum((yhat-mean(y))^2)
> SSR
[1] 1016.41

> SSE+SSR
[1] 12873.36

Fstats <- (SSR/1)/(SSE/(n-2))
> Fstats
[1] 8.400823

> tstats^2
[1] 8.400823
```

```
pvalueF <- 1 - pf(Fstats, 1, n-2)
> pvalueF
[1] 0.004628003

Rsquare <- SSR/SST
> Rsquare
[1] 0.07895449

r <- cor(x, y)
> r^2
[1] 0.07895449

#################################################################
# 2.3 Fitting a Simple Linear Regression Model using lm( ) function             #
#################################################################

fm <- lm( sbp ~ gestage, data = workdat )
> summary(fm)

Call:
lm(formula = sbp ~ gestage, data = workdat)

Residuals:
    Min      1Q  Median      3Q     Max
-23.162  -7.828  -1.483   5.568  39.781

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  10.5521    12.6506   0.834  0.40625
gestage       1.2644     0.4362   2.898  0.00463 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 11 on 98 degrees of freedom
Multiple R-squared: 0.07895,   Adjusted R-squared: 0.06956
F-statistic: 8.401 on 1 and 98 DF,  p-value: 0.004628

#################################################################
# 2.4 Confidence Intervals and Prediction                                       #
#################################################################

###############################
# 95% CI for b1                          #
###############################
> 1.2644+qt(0.025, n-2)*0.4362
[1] 0.3987753

> 1.2644-qt(0.025, n-2)*0.4362
[1] 2.130025
```

```
#################################################################
# Suppose we randomly select a new child from the population of low birth  #
# weight infant with gestational age 31 weeks.                                #
#################################################################

###############################
# Predicted sbp for this child          #
###############################
yhat.p <- beta.fit[1] + beta.fit[2]*31
> yhat.p
   49.74784

###############################
# 95% CI for Prediction                 #
###############################
sigma2hat <- deviance(fm)/(n-2)
> sqrt(sigma2hat)
[1] 10.99951

se.yhat.p <- sqrt( (1+1/n+(31-mean(x))^2/sxx)*sigma2hat)

> yhat.p+qt(0.025, n-2)*se.yhat.p
   27.73488

> yhat.p-qt(0.025, n-2)*se.yhat.p
    71.7608

#########################################################################
#Question: Does the least squares regression model seem to fit the observed data? #
#########################################################################
```