Structure of data analysis in R

Text file data set. You can prepare a text file data set as follows (Available at `e-stat/DALS/brick.txt`):

```
"Brick Weights"
1.099
1.103
...
...
1.132
```

The first line of the file is called a **header**, represents a **variable name** of the data set. To give variable names properly at the first line of the file, you should put it in the double-quotation marks (`"`). And the column directly below the variable name presents the actual data starting from the second line.

Prepare data set. We now read the file "`brick.txt`" into **data frame** in the R programming by using `read.table` function as follows:

```
> BrickData <- read.table("brick.txt", header=T)
```

NOTE: The file "`brick.txt`" must exist at the current working directory. You can always change the working directory by choosing "`Change dir...`" in `[File]` menu. Alternatively you can select the file interactively by using `file.choose()`.

```
> BrickData <- read.table(file.choose(), header=T)
```

"`<-`" in R programming is supposed to play a role of "`=`" as in many other computer languages. Also, R is interactive, known as an "interpretor." For example, if you type "`x <- 3.14`", then you can find that "`x`" has the value 3.14 by simply typing "`x`" (then return).

Declare data frame in use. Before doing anything else, we have to declare the data frame *BrickData* in use by `attach()` function:

```
> attach(BrickData)
```

Now we can use the variable *Brick.Weights*. To see what variables are available, use `names()` function as follows:

```
> names(BrickData)
[1] "Brick.Weights"
```

Note that the original name "Brick Weight" is different from the variable name `Brick.Weight`, where the space was replaced with "`.`"

Summary statistics. To calculate sample statistics, use the `summary` command.

```
> summary(Brick.Weights)
Min. 1st Qu. Median Mean 3rd Qu. Max.
0.874 1.083 1.110 1.111 1.140 1.257
```

Note that the sample size cannot be found from the `summary` command. To find out the data size, use the `length` command.

```
> length(Brick.Weights)
[1] 125
```

Use graphics for data exploration. To draw the histogram, we can use `hist` function as follows:

```
> hist(Brick.Weights)
```

We can change the "number of bands" by assigning the number to `breaks` in `hist` function as follows:

```
> hist(Brick.Weights, breaks=10)
```

We can also add a "color" and a "main title" as follows:

```
> hist(Brick.Weights, breaks=10, col="gray", main="Brick Weights in kg")
```

"`breaks=10`", "`col="gray"`", and "`main="Brick Weights in kg"`" are called *optional arguments*. These optional arguments are not necessarily specified, and are set automatically if they are not.

Use graphics for data exploration. We can draw the boxplot by using `boxplot` function:

```
> boxplot(Brick.Weights)
```

By default boxplots are presented vertically. In order to produce them horizontally we need to specify it by

```
> boxplot(Brick.Weights, horizontal=T)
```

We can also add a "color" and a "main title" as follows:

```
> boxplot(Brick.Weights, col="green", main="Brick Weights in kg")
```

QQ normal plot. If the size n of sample data is small ($n < 30$), the t-test procedure requires that the data be approximately normally distributed; otherwise, the result of t-test may not be reliable. The *quantile-quantile normal plot* (QQ normal plot) is one of the graphical methods to assess the fit of the data to a normal distribution.

```
> qqnorm(Brick.Weights)
```

The values at the x-axis shows the quantiles from the standard normal distribution, and the values at the x-axis correspond the original data. The values between -1.0 and 1.0 in the x-axis consist of approximately 68% of the entire values, which corresponds to the original data between $\mu - \sigma$ and $\mu + \sigma$ in the y-axis if normally distributed. Thus, the straightness of the plot validates goodness of fit, which can be assisted by the display of straight line:

```
> qqline(Brick.Weights, col='green')
```

Statistical inference. The hypothesis testing for population mean μ can be done by the `t.test` command. The `t.test` command calculate the p-value accordingly as

1. $H_A: \mu \neq \mu_0$ (if `alternative="two.sided"` is specified);
2. $H_A: \mu > \mu_0$ (if `alternative="greater"` is specified);
3. $H_A: \mu < \mu_0$ (if `alternative="less"` is specified).

For example, if we construct the hypothesis testing problem

$$H_0: \mu = 1.1 \quad \text{versus} \quad H_A: \mu > 1.1$$

then the `t.test` command must include the options `mu=1.1` and `alternative="greater"`.

Statistical inference. The `t.test` command will return the following output on the display.

```
> t.test(Brick.Weights, mu=1.1, alternative="greater")

        One Sample t-test

data:  Weight
t = 2.2227, df = 124, p-value = 0.01402
alternative hypothesis: true mean is greater than 1.1
95 percent confidence interval:
 1.102680      Inf
sample estimates:
mean of x
 1.110536
```

The above result indicates that (1) t-statistic is 2.2227, (2) p-value is 0.01402, and (3) 95% one-sided confidence interval is $(1.102680, \infty)$. Then, we can reject H_0 with significance level 0.05, but we cannot reject H_0 with significance level 0.01. Thus, the result is modestly significant.

Statistical inference. When you want the 99% two-sided confidence interval instead of the default 95% one-sided confidence interval, we can use the option `conf.level=0.99` together with `alternative="two.sided"` in the `t.test` command.

```
> t.test(Brick.Weights, mu=1.1,
  alternative="two.sided", conf.level=0.99)
  ..........
  ..........
99 percent confidence interval:
 1.098135 1.122937
```

This gave the 99% two-sided confidence interval $(1.098135, 1.122937)$ for the population mean of brick weight.

Clear memory at the end. Typically you start with attaching data frame. When you finished analyzing data, you should detach the data frame.

```
> detach(BrickData)
```

Also, you may want to clear any use of memory in the working environment at the end.

```
> rm(list=ls())
```

However, it should be done only when you are absolutely sure about the completion of data analysis.

Paired or multicolumn data

Paired data. A researcher is interested in how a new class of drug treating a patient actually affects the patient's heart rate reduction. The pairs of heart rate reduction

$$(X_1, Y_1), \ldots, (X_n, Y_n)$$

of n participants under the standard drug and after taking the new drug are measured. The data set of heart rate reductions is prepared in the CSV file `heart.csv`.

```
Patient,StdDrug,NewDrug
1,28.5,34.8
2,26.6,37.3
...
...
40,40.1,40.8
```

Read CSV file into R. We can read it into the data frame *HeartData*.

```
> HeartData <- read.csv("heart.csv")
```

The command `read.csv()` assumes that a header line is present in the CSV file (i.e., header=TRUE is the default). If you do not have a header line in the CSV file, you need to specify "`header=FALSE`" argument.

summary statistics. The `summary` command will show you the variable names and their summary statistics. These variable names are *Patient*, *StdDrug*, and *NewDrug*, as indicated in the output below.

```
> summary(HeartData)
Patient          StdDrug         NewDrug
Min.   : 1.00    Min.   :21.60   Min.   :22.40
1st Qu.:10.75    1st Qu.:27.45   1st Qu.:30.80
Median :20.50    Median :32.00   Median :34.25
Mean   :20.50    Mean   :31.18   Mean   :33.84
3rd Qu.:30.25    3rd Qu.:34.35   3rd Qu.:37.00
Max.   :40.00    Max.   :40.20   Max.   :43.70
```

Note that *Patient* is an identifier of patient and irrelevant to data analysis, and that the summary statistics of *Patient* should not be produced in a report.

Graphical presentations. Here we need the boxplot for each of *StdDrug* and *NewDrug* to compare the two samples graphically. The `boxplot` command will create the two boxplots in one figure.

```
> boxplot(StdDrug, NewDrug)
```

You can change the name of each boxplot by adding "`names`" option.

```
> boxplot(StdDrug, NewDrug, names=c("Standard Drug", "New Drug"))
```

Appearance can be controlled precisely by many options.

```
> boxplot(StdDrug, NewDrug, names=c("Standard Drug", "New Drug"), col="gray", ylab="Hear
rate reductions", main="Boxplots for Heart Rate Reductions")
```

Statistical inference. The paired sample test can be done by the `t.test` command with the option `paired=T`. Suppose that we want to test

$$H_0 : \mu_1 = \mu_2 \quad \text{versus} \quad H_A : \mu_1 < \mu_2$$

where μ_1 and μ_2 are the true means of heart rate reductions with the standard drug and the new drug, respectively. Then the `t.test` command can be used as follows.

```
> t.test(StdDrug, NewDrug, alternative="less", paired=T)

    Paired t-test

data:  StdDrug and NewDrug
t = -4.5016, df = 39, p-value = 2.974e-05
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
  -Inf -1.661287
sample estimates:
mean of the differences
    -2.655
```

Statistical inference. The output indicates that (1) t-statistic is -4.5016, (2) p-value is 2.974×10^{-5}, and (3) the 95% one-sided confidence interval $(-\infty, -1.661287)$ for the difference $(\mu_1 - \mu_2)$. Thus, we can reject H_0 with significance level 0.01. To obtain the 99% two-sided confidence interval, change the alternative hypothesis to `alternative="two.sided"` and add the option "`conf.level=0.99`" as follows.

```
> t.test(StdDrug, NewDrug, alternative="two.sided", paired=T, conf.level=0.99)
```

In practice you should not change the alternative hypothesis based on what you have seen in data (called "data snooping"). But in this particular case the alternative hypothesis was changed in order to obtain a two-sided confidence interval.

Create array variables

Set data into variables In order to test the difference of nerve conductivity speed between healthy persons and patients with nerve disorder, the study considers 12 healthy subjects and 8 subjects with nerve disorder, which is prepared in a form of variable. The first varible, "Healthy," consists of healthy subject data, and the second variable, "Disorder," consists of disordered subject data. In order to set them directly, use `<- c()` where data goes inside the function `c()`.

```
> Healthy <- c(52.2, 53.81, 53.68, 54.47, 54.65, 52.43, 54.43, 54.06,
+ 52.85, 54.12, 54.17, 55.09)
> Disorder <- c(50.68, 47.49, 51.47, 48.47, 52.5, 48.55, 45.96, 50.4)
```

Type variable names to see the content.

```
> Healthy
 [1] 52.20 53.81 53.68 54.47 54.65 52.43 54.43 54.06 52.85
[10] 54.12 54.17 55.09
```

Exploration of data. The function `length()` gives the size of data contained in variable.

```
> length(Healthy)
> length(Disorder)
```

We can also find out their summary statistics (`summary()`) as follows.

```
> summary(Healthy)
> summary(Disorder)
```

The `boxplot` command will be used to compare the two samples.

```
> boxplot(Healthy, Disorder, names=c("Healthy", "Nerve Disorder"), col="gray", ylab="Con
speeds", main="Boxplots for Nerve conductivity speeds")
```

Statistical inference. The `t.test` command can be used again for the two independent sample test. Suppose that our hypothesis testing problem is

$$H_0 : \mu_1 = \mu_2 \quad \text{versus} \quad H_A : \mu_1 > \mu_2$$

where μ_1 and μ_2 are the true means of nerve conductivity speed for healthy subjects and nerve disorder subjects, respectively. Then The `t.test` command generates the following output.

```
> t.test(Healthy, Disorder, alternative="greater")

        Welch Two Sample t-test

data:  Healthy and Disorder
t = 5.3749, df = 8.59, p-value = 0.000262
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 2.884598      Inf
sample estimates:
mean of x mean of y
    53.83     49.44
```

Statistical inference. The result shows that (1) t-statistic is 5.3749, (2) p-value is 0.000262, and (3) the 95% one-sided confidence interval is $(2.884598, \infty)$ for the difference $(\mu_1 - \mu_2)$. Thus, we can reject H_0. If we wish to obtain the 99% two-sided confidence interval, then we use the `t.test` command with "`conf.level=0.99`" as follows.

```
> t.test(Healthy, Disorder, alternative="two.sided", conf.level=0.99)
```

The default procedure corresponds to general t-test. If equal variances are assumed, the pooled t-test should be used. In this case we need to indicate it by the option "`var.equal=T`".

```
> t.test(Healthy, Disorder, alternative="greater", var.equal=T)
```

Linear models

Linear regression model. Suppose that the researcher wants to find how the temperature of factory affects the labor efficiency to unload a truck. We conduct 12 independent experiments with different levels of temperature. The data set consists of the unloading time paired with the respective temperature of the factory.

```
Time, Temp
64, 52
53, 68
...
...
47, 66
```

Data and summary statistics. To read the data set, we use the `read.csv`. Then declare the use of data.

```
> TimeData <- read.csv("time.csv")
> attach(TimeData)
```

To see sample statistics with variable names, we can use the `summary` function. Then, the first line of the output below displays the variable names *Time* and *Temp*.

```
> summary(TimeData)
      Time            Temp
 Min.   :38.00   Min.   :52.00
 1st Qu.:48.75   1st Qu.:63.75
 Median :56.00   Median :69.50
 Mean   :54.92   Mean   :70.42
 3rd Qu.:60.00   3rd Qu.:76.25
 Max.   :68.00   Max.   :88.00
```

Exploratory analysis. The `plot()` function can be used to show the scatter plot of temperature against time.

```
> plot(Temp, Time)
```

The title of graph can be added as follows:

```
> plot(Temp, Time, main="Scatter plot of temperature against time")
```

The `cor()` function calculates the correlation between explanatory variable (x-axis) and dependent variable (y-axis).

```
> cor(Temp, Time)
```

Statistical inference. To fit the data frame *TimeData* into a simple linear model, the `lm` function will be used and the result must be saved as a variable. Then, the `summary` function with the variable produced by the `lm` function can display the result.

```
> TimeLM <- lm(Time ~ Temp)
> summary(TimeLM)

Residuals:
    Min      1Q  Median      3Q     Max
-13.881  -5.394  -0.933   5.375  13.980

Coefficients:
            Estimate  Std.Error t value Pr(>|t|)
(Intercept)  36.1935    16.9515   2.135   0.0585 .
Temp          0.2659     0.2383   1.116   0.2905
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In finding a trend, the result shows that (i) the estimate $\hat{\beta}_1$ of slope is 0.2659, and (ii) the p-value is 0.2905, which is insignificant. Thus, we cannot reject the null hypothesis $H_0 : \beta_1 = 0$, and therefore, there is not sufficient evidence to conclude that the unloading time depends on the temperature. Thus, no relationship has been established between the two variables. As for the intercept, the result shows that (i) the estimate $\hat{\beta}_0$ of intercept is 36.1935, and (ii) the p-value is 0.0585, which is moderately significant. By using the estimates and standard errors we can compute confidence intervals.

```
> confint(outcome, level=0.95)
                2.5 %     97.5 %
```

```
(Intercept) -1.5767841 73.9638157
Temp        -0.2649937  0.7967755
```

Residual analysis. To see the fitted line graphically, we can use the `abline()` function. It adds the fitted line on the scatter plot which was previously drawn.

```
> abline(TimeLM)
```

To assess the fit graphically and examine a possibility of influential points, "residual-fit spread" plot compares the spread of the fitted values with the spread of the residuals. We can create this diagnostic plot by using `plot` function with the model variable name *TimeLM*.

```
> plot(TimeLM, which=1)
```

The plot indicates the 1st and 7th data points as influential points.

Removal of influential points. Observations are arranged in rows, and all the data associated with the 1st observation can be displayed by `TimeData[1,]`. In order to remove data associated with 1st observations, we can create a new data frame `TimeData2` with data without the 1st observations by

```
> TimeData2 <- TimeData[-1,]
```

Then detach the original data frame `TimeData`, and attach the new data frame `TimeData2`.

```
detach(TimeData)
attach(TimeData)
```

After attaching the new data frame we start a new analysis all over again

Prediction by linear model. The `predict()` function is applicable for computation of predicted values corresponding to new observations. Here we wish to predict the unloading times when the temperatures are 60, 70, and 80, respectively. The predicted values are calculated by the linear regression model as follows:

```
new <- data.frame(Temp=c(60, 70, 80))
new
predict(outcome,new,level=0.95,interval="confidence")
predict(outcome,new,level=0.95,interval="prediction")
```

First, a new data frame with the new values of the explanatory variables must be created. It contains a single variable, `Time`, with three new values. It is important that the variable has the same name as the explanatory variable in the original dataset. The type of the intervals, `"confidence"` or `"prediction"`, can be chosen by the option `interval`.

Analysis of variance

Comparison of more than two groups. Hearing aids must be fit individually. A common way to test whether a particular hearing aid is right for a patient is to play a tape on which 25 words are pronounced clearly but at low volume, and ask the patient to repeat the words as heard. Different lists are available that are supposed to be of equal difficulty to understand correctly.

However, a major problem for those wearing hearing aids is that the aids amplify background noise as well as the desired sounds. Are the test lists still equally difficult to understand in the presence of background noise? In this experiment, 24 subjects with normal hearing listened to standard audiology tapes of English words at low volume, with a noisy background. They repeated the words and were scored correct or incorrect in their perception of the words. The order of list presentation was randomized.

Change attribute as factor. The first line of the data file "`hearing.csv`" contains the variable names *SubjectID*, *ListID*, and *Hearing*. The variables *SubjectID* and *ListID* must be **factors**, and the values `S1`, `S2`,..., and `List1`,`List2`,... are called **factor levels** in each of the variables *SubjectID* and *ListID*. To recognize the values as factor levels, we have to explicitly declare it by using `as.factor` function immediately after reading the data file.

```
HearingData <- read.csv("hearing.csv")
HearingData$ListID <- as.factor(HearingData$ListID)
HearingData$SubjectID <- as.factor(HearingData$SubjectID)
```

Or, specify the option "`stringsAsFactors=T`" when `read.csv()` is called.

```
> HearingData <- read.csv("hearing.csv",stringsAsFactors=T)
```

Data and summary statistics.

Declare the use of data.

```
> attach(HearingData)
```

To see sample statistics with variable names, we can use the `summary()` command.

```
> summary(HearingData)
   SubjectID     ListID       Hearing
 S1      : 4   List1:24   Min.   :14.0
 S10     : 4   List2:24   1st Qu.:20.0
 S11     : 4   List3:24   Median :29.0
 S12     : 4   List4:24   Mean   :27.9
 S13     : 4              3rd Qu.:34.0
 S14     : 4              Max.   :44.0
 (Other):72
```

The variables *SubjectID* and *ListID* are recognized as **factors**, and `summary()` shows the size for each factor levels.

Parallel boxplots. We can use the `boxplot` command to create a boxplot for each factor level of different lists.

```
> boxplot(Hearing ~ ListID)
```

We can change the color of box, and add the description of y-axis as follows.

```
> boxplot(Hearing ~ ListID, col="gray", ylab="Scores of hearing")
```

One-way ANOVA. One-way analysis of variance (ANOVA) is carried out by using the `aov` function, and the results must be saved into a variable. To display the ANOVA table, use the `summary` command with the variable produced by the `aov` function.

```
> HearingAnova <- aov(Hearing ~ ListID)
> summary(HearingAnova)
             Df Sum Sq Mean Sq F value Pr(>F)
```

```
ListID       3    429  142.93   2.473 0.0666 .
Residuals   92   5318   57.81
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 '.' 0.1 ' ' 1
```

The result shows that (i) F-statistic is 2.473, and (ii) p-value is 0.0666. Thus, we fail to reject the null hypothesis, and therefore, we can say that there is no effect of differences in lists.

Multiple comparisons. To find out pairwise differences between the treatment means μ_1, μ_2, μ_3, and μ_4, we can use the `TukeyHSD()`.

```
> TukeyHSD(HearingAnova, conf.level=0.95)

$ListID
                  diff        lwr       upr     p adj
List2-List1 -0.5833333  -6.326300 5.1596330 0.9933848
List3-List1 -5.0000000 -10.742966 0.7429663 0.1106952
List4-List1 -3.8333333  -9.576300 1.9096330 0.3059080
List3-List2 -4.4166667 -10.159633 1.3262997 0.1909626
List4-List2 -3.2500000  -8.992966 2.4929663 0.4532014
List4-List3  1.1666667  -4.576300 6.9096330 0.9511824
```

For each pair of groups it provides the difference in the observed means, "`lwr`" giving the lower end point of the interval, "`upr`" giving the upper end point, and "`p adj`" giving the p-value after adjustment for the multiple comparisons.