R Tutorial

Contents

1. R Windows Environment

The R programming language is a free open-source statistical software package. The program can be downloaded at: http://cran.r-project.org. Once installed on your computer, enter into an R session and a console will open where commands are input, as shown in Figure 1.

Commands are typed in line by line, similar to a Texas Instruments calculator. Once a line is entered, R will immediately compile (evaluate) it. For the sake of simplicity, short names will be used (''x,'' ''y,'' etc.) when labeling data objects, but these are just labels and can always be adjusted. [Note: The R help feature will give further details about discussed commands:

```
> ? Command Name
```

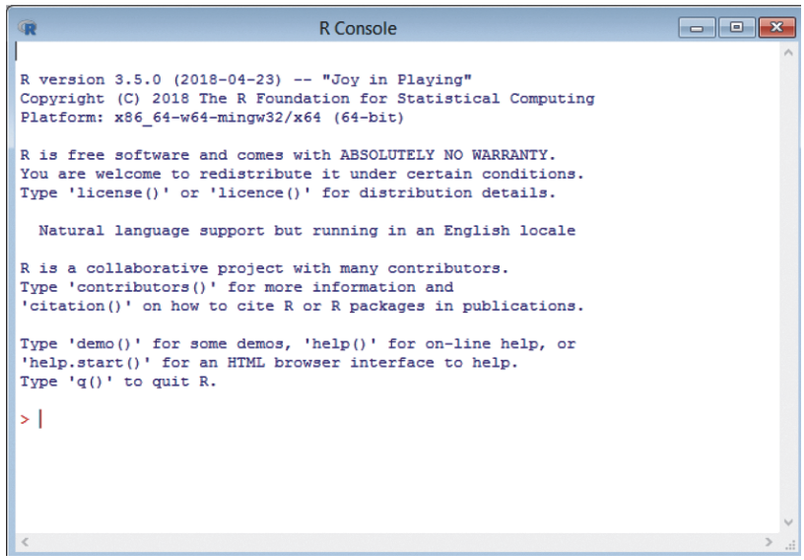
Replace ''`Command Name`'' with the command of interest (e.g., ''? mean''). Additionally, you can go to the CRAN website to find information on additional commands not discussed. A useful starting point for further information is: http://cran.r-project.org/doc/manuals/R-intro.html.]

Important: R commands are case sensitive. If a variable, object, data file, matrix, etc., is named using lower-case (upper-case) letters, then the command must employ lower-case (upper-case) letters.

2. Preparing Data for Analysis in R

In R data is stored in ''objects'' on which analyses are run. Two objects discussed here are vectors and matrices. A vector stores the values for one variable. Matrices store entire tables of data, consisting of multiple variables, and are useful for complicated analyses. It is easier, especially with tables, to store data in a ''.txt'' or ''.csv'' file and import it into R, instead of writing everything out.

Figure 1 R commands for creating vectors



Direct Data Entry: To create a vector of one value, 5, in R and call it "x," use the command

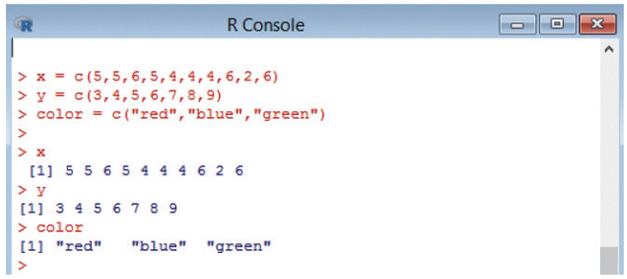
```
> x = 5
```

This tells R to create an object called "x," which equals the value 5. To create a vector of the values {1,2,3,4} in R, use the command:

```
> x = c(1,2,3,4)
```

This creates a vector called "x" that contains the desired values. The command "c()" tells R to create a vector of the values in the parentheses (each value must be separated by a comma). If desired, vectors can contain character values instead of numeric by enclosing the desired characters in quotes. You can see what values are in any vector or matrix by typing out the name and pressing enter/return. See Figure 2 for examples.

Figure 2 R Commands for Creating Data Vectors

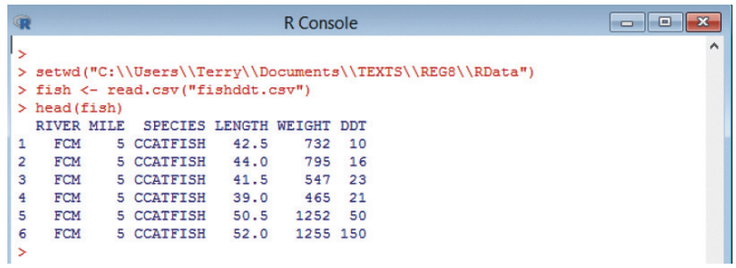


Importing Data:

A vector or matrix can be created by importing the data. To import data from a ".csv" file and save it, first identify the working directory where the file resides. Then use the "setwd" command, with the directory of interest in quotes. See, for example, the 1st command line in Figure 3. Now use the "read.csv" command to identify the file you want to import (e.g., fishddt.csv) in quotes, as shown in the 2nd command of Figure 3. The user-defined name preceding "<-read.csv" specifies that

the data will be saved in a data matrix named "fish" during this R session. From here on out, we will use "fish" in all subsequent R commands that analyze this data. [Note: To view the first 6 observations (rows) in the data matrix named "fish", use the "head(fish)" command, as shown in Figure 3.]

Figure 3 R Commands for Setting the Working Directory and Importing a Text File



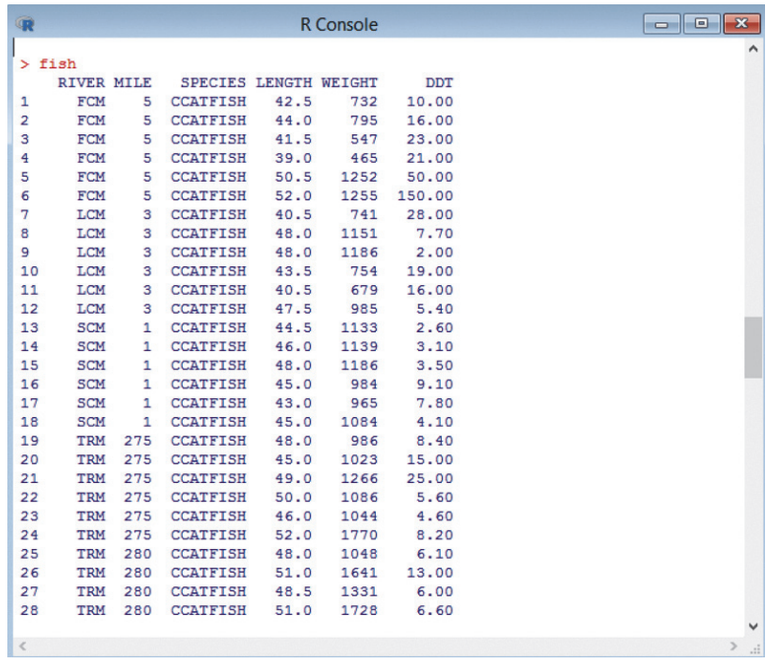
3. Listing Data and Saving Output

The data stored in a working file can be displayed by typing the name of the file into the console and pressing enter:

```
> fish
```

All the data will be listed, as shown in Figure 4.

Figure 4 R Commands for Listing Data



To save a generated output (e.g., a plot) for use in a report, select the window containing the output (i.e., make sure the output panel is selected and not the command console) and go to $File \rightarrow SaveAs$, as shown in Figure 5.

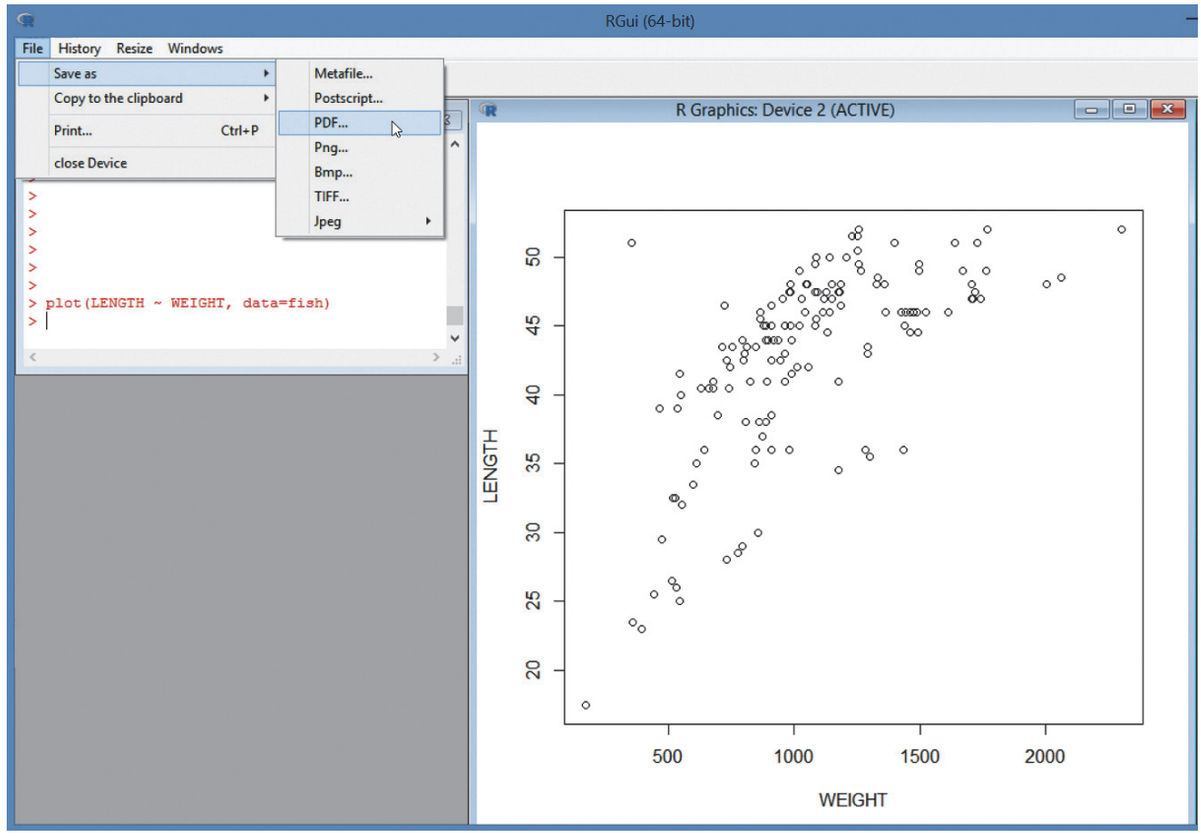


Figure 5 Saving Output in R

4. Graphing Data

To create a frequency histogram of observations for a quantitative variable (e.g., "LENGTH") saved in a data file (e.g., "fish"), use the command:

```
> hist(fish$LENGTH)
```

Note that the data set name is listed first in parentheses, followed by "$" and the variable name. (See Figure 6.)

Note: R will automatically decide how many bins to use for the histogram. To pre-select the number of bins, say "11", use the command:

```
> hist(fish$LENGTH, breaks=11)
```

The number of bins is equal to the number of breaks plus one. R will take the number of breaks requested as a suggestion and may add more bins than asked for. For a relative frequency histogram (with a total area of one under the curve), add in "prob=TRUE":

```
> hist(fish$LENGTH, breaks=11, prob=TRUE)
```

To create a bar chart, for a qualitative variable (e.g., "SPECIES") saved in a data file (e.g., "fish"),' use the command:

```
> barplot(table(fish$SPECIES))
```

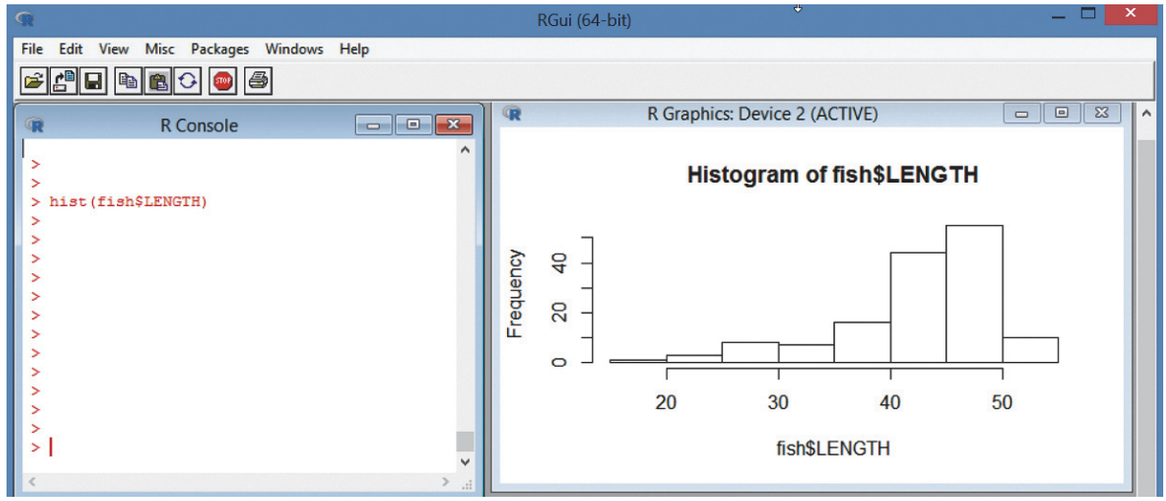


Figure 6 Producing a Histogram in R

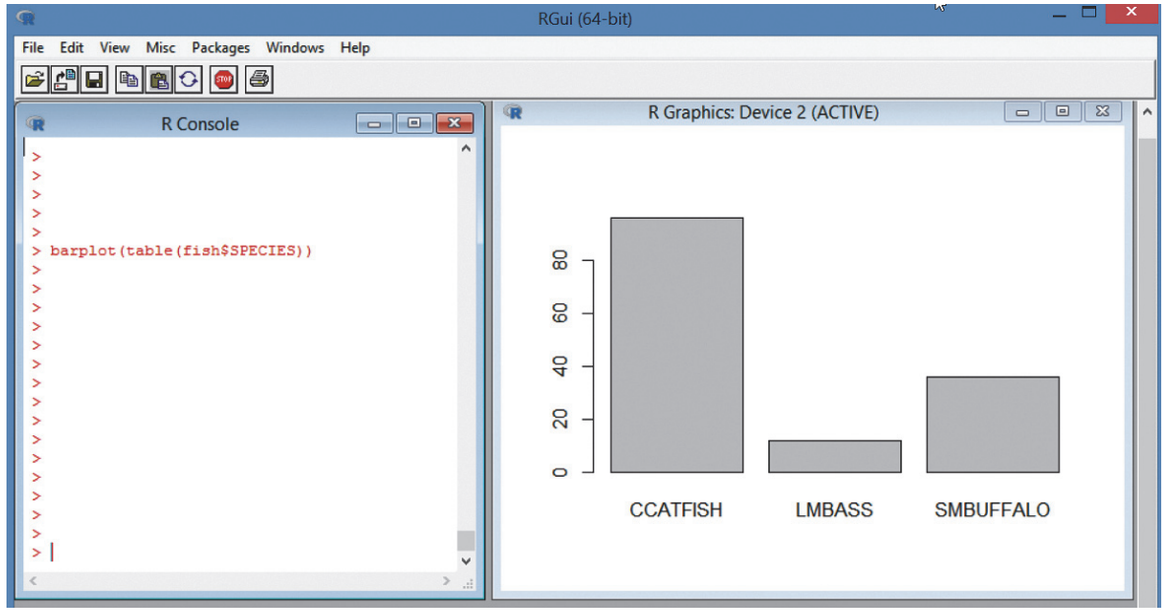


Figure 7 Producing a Bar Graph in R

Again, note that the data set name is listed first in parentheses, followed by "$" and the variable name. (See Figure 7.)

To create a pie chart for a qualitative variable (e.g., "SPECIES") saved in a data file (e.g., "fish"),' first determine the frequency of observations for each level of the variable. Use the command:

```
> summary(fish$SPECIES)
```

Next, create a vector called "counts" that contains the frequency for each level. These counts may be replaced by proportions if desired.

```
> counts <-c(96,12,36)
```

Finally, a basic pie chart can be created using the command:

```
> pie(counts)
```

Add more informative labels to the pie chart using the command:

```
> pie(counts, labels=c(''CCATFISH'',''LMBASS'',''SMBUFFALO''))
```

These commands (and the resulting output) are shown in Figure 8.

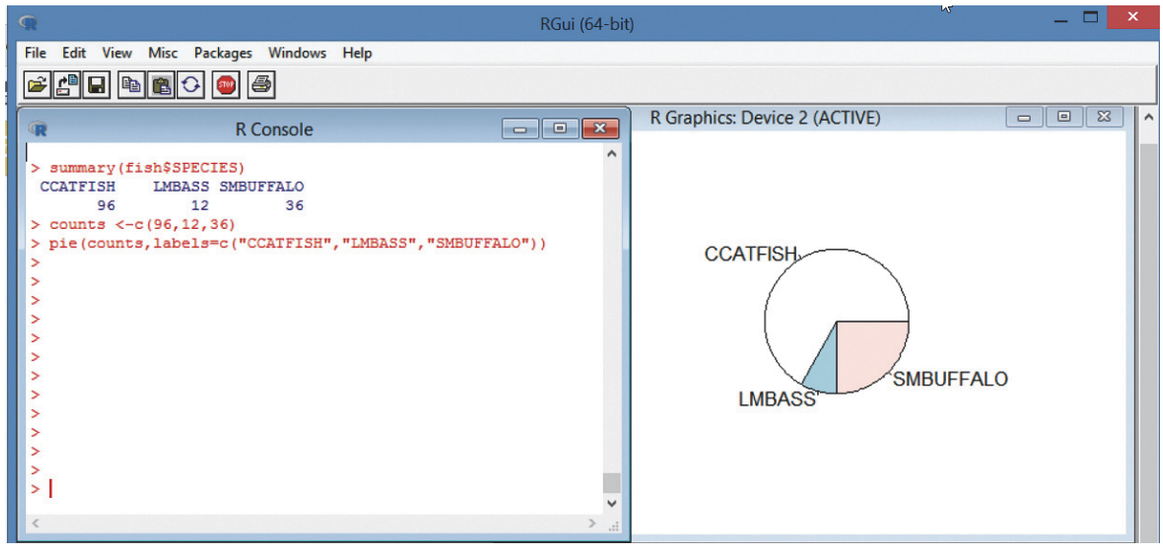


Figure 8 Producing a Pie Chart in R

To construct a boxplot for a quantitative variable (e.g., "LENGTH") saved in a data file (e.g., "fish"),' use the command:

```
> boxplot(fish$LENGTH)
```

To construct a scatterplot of two quantitative variables (e.g., "WEIGHT" and "LENGTH") saved in a data file (e.g., "fish"),' use the command:

```
> plot(fish$WEIGHT, fish$LENGTH)
```

The variable listed first will be plotted on the horizontal axis and the second variable will be on the vertical. Alternatively, you may use the command:

```
> plot(WEIGHT ~ LENGTH, data=fish)
```

With this syntax, the variable to the left of the "~" is plotted on the vertical axis and the variable to the right on the horizontal axis. (See Figure 9.)

5. Descriptive Statistics

There are several R functions for summarizing quantitative variables. Consider a quantitative variable (e.g., "LENGTH") saved in a data file (e.g., "fish"). Below is a list of five useful commands.

Five Number Summary (also calculates the mean):

```
> summary(fish$LENGTH)
```

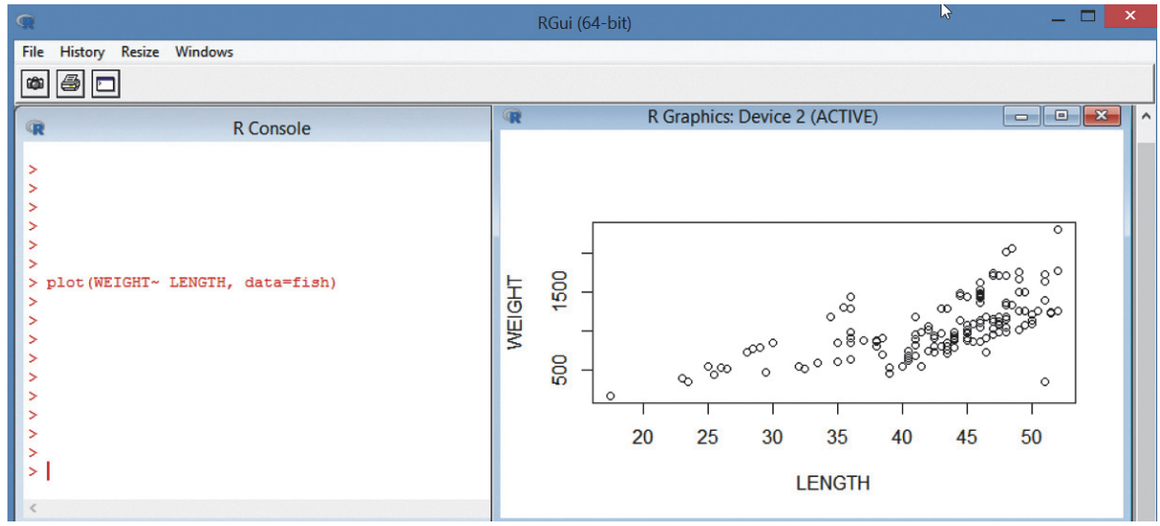


Figure 9 Producing a Scatterplot in R

Quantiles (Percentiles):

```
> quantile(fish$LENGTH, prob=a)
```

Replace "a" with the desired quantile (in decimal form).

Mean:

```
> mean(fish$LENGTH)
```

Standard Deviation:

```
> sd(fish$LENGTH)
```

Variance:

```
> var(fish$LENGTH)
```

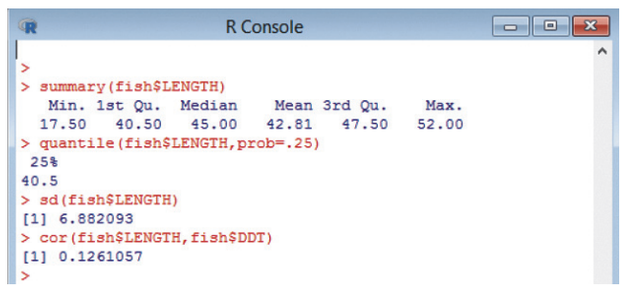
Now consider a 2nd quantitative variable (e.g., "DDT").

Correlation:

```
> cor(fish$LENGTH,fish$DDT)
```

Several of these commands (with results) are shown in Figure 10.

Figure 10 Producing Descriptive Statistics in R



6. Hypothesis Tests on Means

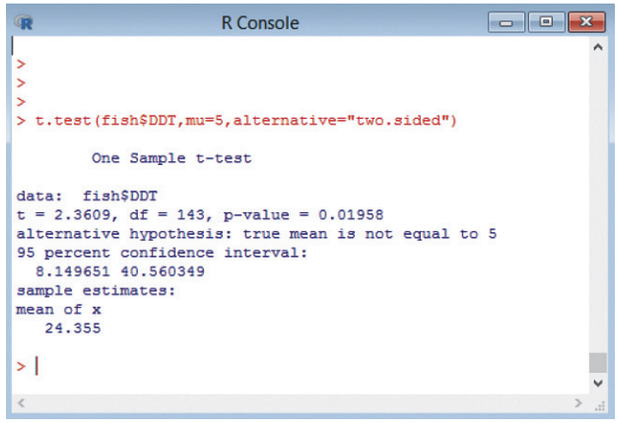
To perform a hypothesis test for a single population mean, use the command "*t*-test." If "x" is the quantitative variable of interest and "`mu_null`" is the value of the mean under the null hypothesis, then use the command:

```
>t.test(x, mu = mu_null, alternative = 'two.sided')
```

The above will return the *p*-value of the test as well as a 95% confidence interval for the mean. If a one-sided test is desired, then replace "two.sided" with "greater" or "less."

Figure 11 shows this command (and results) for the variable "LENGTH" on the "fish" data file.

Figure 11 R Commands for Testing a Single Mean

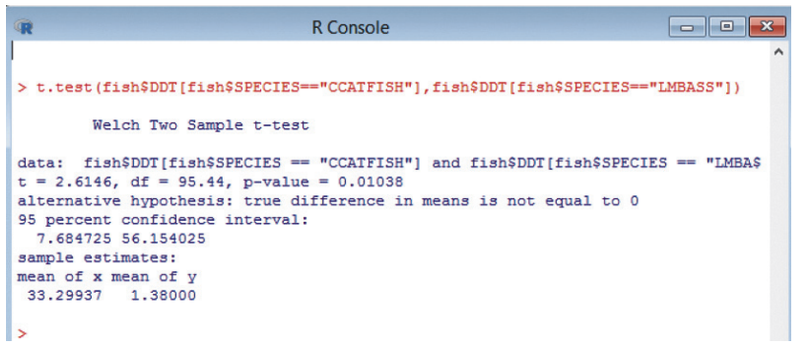


The "t-test" command can also be used to compare two population means. Consider comparing the means of a quantitative variable (e.g., "DDT") for two levels of a qualitative variable (e.g., "SPECIES"), where the data is saved in a file (e.g., "fish"). Use the command:

```
> t.test(fish$DDT[fish$SPECIES==''CCATFISH''],
  fish$DDT[fish$SPECIES==''LMBASS'']) 
```

Note that the two levels of the qualitative variable (SPECIES) are given in quotes after the two equal signs. This command (and results) are shown in Figure 12.

Figure 12 R Commands for Testing/Comparing Two Means



Alternatively, if the qualitative variable of interest in the data file has more than two levels, you can create a subset data file with the two levels you want to compare as follows:

```
> fish2 <-fish[fish$SPECIES %in% c(''CCATFISH'','LMBASS''),]
```

Here, we create a new data file named "fish2" that includes only the values of "CCATFISH" and "LMBASS" for the qualitative variable "SPECIES". Now, we can compare the two means using the command:

```
> t.test(DDT ~ SPECIES,data=fish2)
```

7. Simple Linear Regression and Multiple Regression

Simple Linear Regression:

To perform a simple linear regression relating a quantitative dependent variable (e.g., "DDT") to a quantitative independent variable (e.g., "LENGTH") for data saved in a file (e.g., "fish"), use the command:

```
> lm (DDT ~ LENGTH, data=fish)
```

The above command will give the estimate for the intercept followed by the slope, as shown in Figure 13.

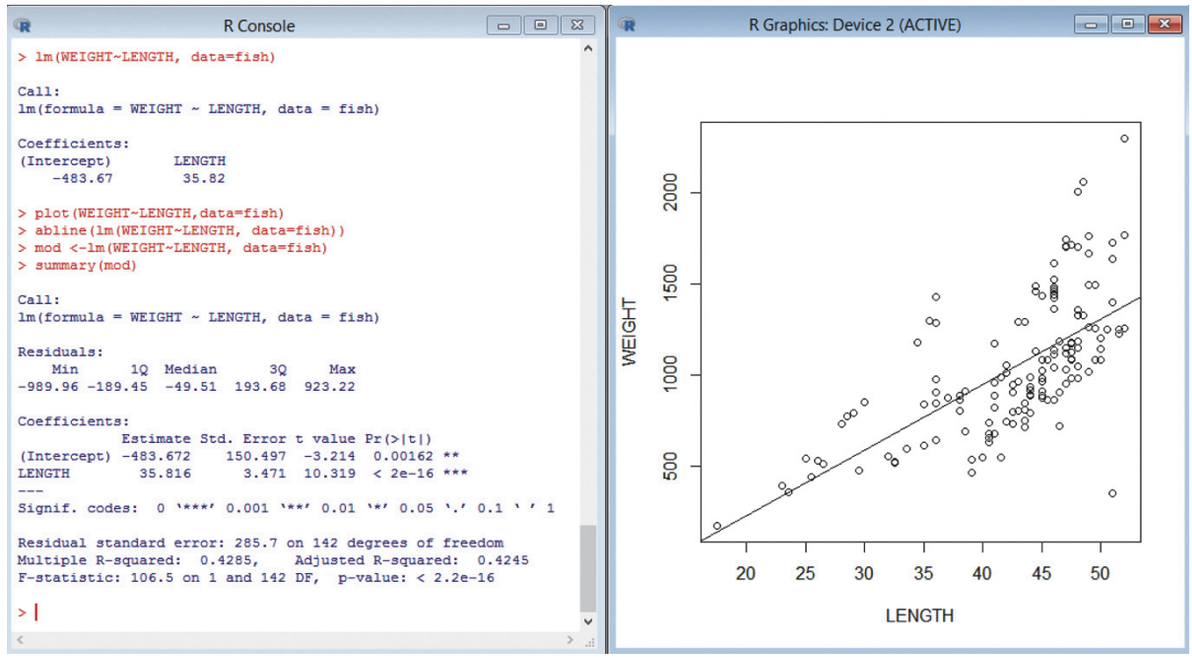


Figure 13 R Commands for Simple Linear Regression

To produce a scatterplot of the data, with the least squares line, use the commands:

```
> plot(WEIGHT ~ LENGTH,data=fish)
> abline(lm(WEIGHT ~ LENGTH, data=fish))
```

Finally, to produce statistics related to model fit (e.g., R^2, t-test, p-value, standard deviation), use the commands:

```
> mod <-lm(WEIGHT ~ LENGTH, data=fish)

> summary(mod)
```

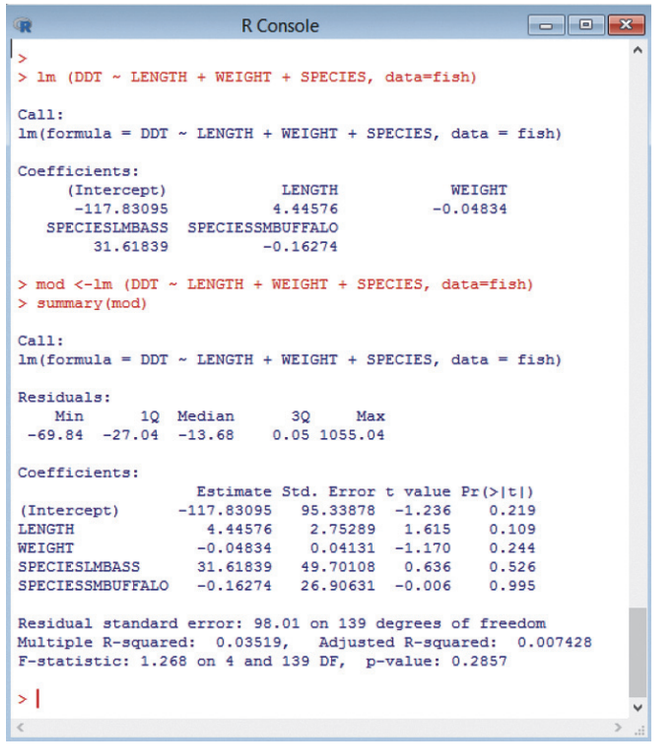
Multiple Regression:

To perform a multiple regression relating a quantitative dependent variable (e.g., "DDT") to both quantitative and qualitative independent variables (e.g., "LENGTH", "WEIGHT", and "SPECIES") for data saved in a file (e.g., "fish"), use the command:

```
> lm (DDT ~ LENGTH + WEIGHT + SPECIES, data=fish)
```

The above command will produce estimates of the betas in the model, as shown in Figure 14.

Figure 14 R Commands for Multiple Regression



Note that R automatically creates dummy variables for the qualitative variable. If you want to include interactions and higher-order terms in the model, first you will need to create these additional columns using commands like this:

```
> fish$LEN.WT <- fish$LENGTH * fish$WEIGHT
```

A new variable, named "LEN.WT", is added to the "fish" data file and this variable is "LENGTH" multiplied by "WEIGHT" (i.e., an interaction term).

To produce statistics related to model fit (e.g., R^2, F-test, p-value, t-tests, standard deviation), and save them in an object called "mod", use the commands:

```
> mod <- lm (DDT ~ LENGTH + WEIGHT + SPECIES, data=fish)

> summary(mod)
```

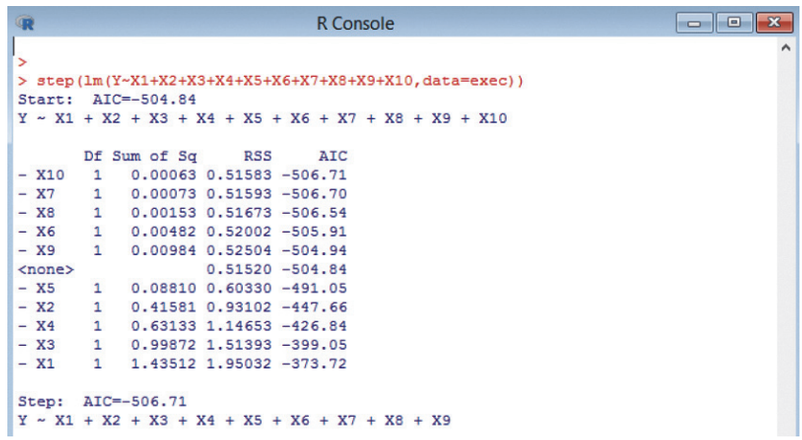
8. Stepwise Regression

Consider a data file called "exec" with 10 potential independent variables (named X1, X2, X3, . . . , X10) and a dependent variable (named Y). To perform a standard stepwise regression, use the command:

```
> step(lm(Y ~ X1+X2+X3+X4+X5+X6+X7+X8+X9+X10), data=exec)
```

This command (and some results) are shown in Figure 15.

Figure 15 R Commands for Stepwise Regression



Note: To run either a forward or backward stepwise regression, add "direction=" to the end of the command, as shown below:

```
> step(lm(Y ~ X1+X2+X3+X4+X5+X6+X7+X8+X9+X10), data=exec),
  direction=''forward'')
> step(lm(Y ~ X1+X2+X3+X4+X5+X6+X7+X8+X9+X10), data=exec),
  direction=''backward'')
```

9. Residual Analysis and Influence Diagnostics

To produce residual plots for the multiple regression after you have saved the regression results in an object called "mod", use the following commands:

Histogram of residuals:	`> hist(resid(mod))`
Normal probability plot of residuals:	`> qqnorm(resid(mod))`
Plot of residuals vs. predicted:	`> plot(resid(mod) ~ fitted(mod))`
Plot of residuals vs. independent variable:	`> plot(resid(mod) ~ fish$WEIGHT)`

To obtain a list of influence diagnostics (e.g., leverage values, Cook's distance, studentized-deleted residuals) for each observation in the data set, use the command:

```
> influence.measures(mod)
```

Two of these commands (and resulting output) are shown in Figure 16.

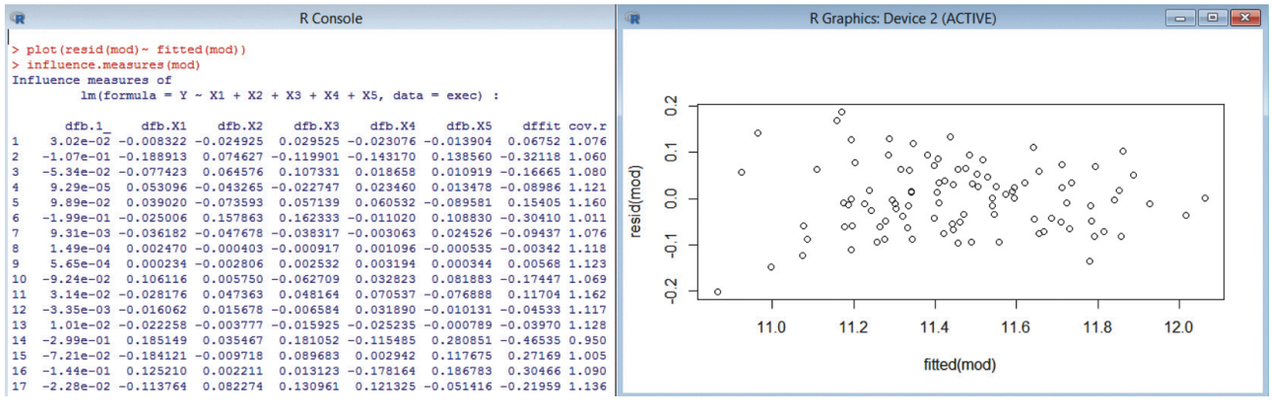


Figure 16 R Commands for Residual Plots and Influence Diagnostics

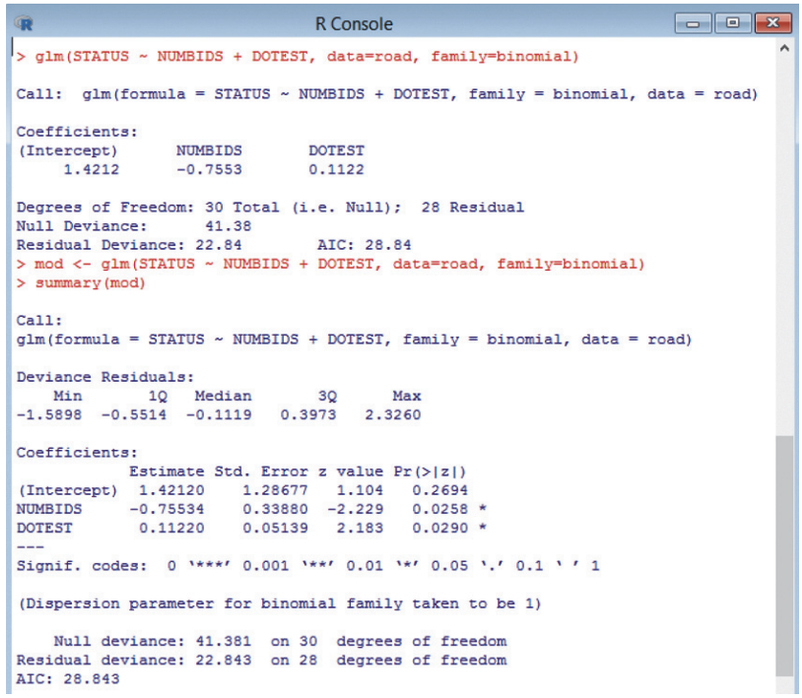
10. Logistic Regression

To perform a logistic regression relating a qualitative dependent variable (e.g., "STATUS") to both quantitative and qualitative independent variables (e.g., "DOTEST" and "NUMBIDS") for data saved in a file (e.g., "road"), use the command:

```
> glm (STATUS ~ NUMBIDS + DOTEST, data=road, family=binomial)
```

The above command will produce estimates of the betas in the model, as shown in Figure 17. (Note: For qualitative independent variables, R will create the appropriate number of dummy variables.)

Figure 17 R Commands for Logistic Regression



To produce statistics related to model fit (e.g., model χ^2-test, parameter tests, p-values), and save them in an object called ''mod'', use the commands:

```
> mod <- glm (STATUS ~ NUMBIDS + DOTEST, data=road,
  family=binomial)
> summary(mod)
```

See Figure 17.

To obtain a list of the predicted probabilities for the observations in the data set, use the command:

```
> fitted.values(mod)
```

To obtain a list of residuals for the observations in the data set, use the command:

```
> residuals(mod)
```

11. Poisson Regression

To perform a Poisson regression relating a count-type dependent variable (e.g., ''Failures'') to both quantitative and qualitative independent variables (e.g., ''Month'' and ''AdjTA'') for data saved in a file (e.g., ''banks''), use the command:

```
> glm (Failures ~ Month + AdjTA, data=banks, family=poisson)
```

The above command will produce estimates of the betas in the model, as shown in Figure 18. (Note: For qualitative independent variables, R will create the appropriate number of dummy variables.)

Figure 18 R Commands for Poisson Regression

R Console

```
> glm(Failures ~ AdjTA + Month, data=banks, family=poisson)

Call:  glm(formula = Failures ~ AdjTA + Month, family = poisson, data = banks)

Coefficients:
(Intercept)        AdjTA     MonthAug     MonthDec     MonthFeb     MonthJan
 11.4465595   -0.0007837   -0.9130949   -0.0461367    0.3033030    0.8216924
   MonthJul     MonthJun     MonthMar     MonthMay     MonthNov     MonthOct
  0.1057088    0.0250779   -0.4039405    0.8536381   -0.7550241    0.7509937
   MonthSep
 -0.9139864

Degrees of Freedom: 47 Total (i.e. Null);  35 Residual
Null Deviance:      58.15
Residual Deviance: 36.52        AIC: 120.4
> mod2 <- glm(Failures ~ AdjTA + Month, data=banks, family=poisson)
> summary(mod2)

Call:
glm(formula = Failures ~ AdjTA + Month, family = poisson, data = banks)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.5241  -0.9576  -0.4243   0.6481   1.3532

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) 11.4465595  4.4986754   2.544  0.01095 *
AdjTA       -0.0007837  0.0003006  -2.607  0.00914 **
MonthAug    -0.9130949  1.1578700  -0.789  0.43035
MonthDec    -0.0461367  0.9274990  -0.050  0.96033
MonthFeb     0.3033030  0.7637681   0.397  0.69128
MonthJan     0.8216924  0.6902572   1.190  0.23388
MonthJul     0.1057088  0.8183230   0.129  0.89722
MonthJun     0.0250779  0.8168083   0.031  0.97551
MonthMar    -0.4039405  0.9128715  -0.442  0.65813
MonthMay     0.8536381  0.6901174   1.237  0.21611
MonthNov    -0.7550241  1.1653072  -0.648  0.51704
MonthOct     0.7509937  0.7392767   1.016  0.30970
MonthSep    -0.9139864  1.1582496  -0.789  0.43005
---
```

To produce statistics related to model fit (e.g., model χ^2-test, parameter tests, p-values), and save them in an object called ‘‘mod2’’, use the commands:

```
> mod2 <- glm (Failures ~ Month + AdjTA, data=banks,
  family=poisson)
> summary(mod2)
```

See Figure 18.

To obtain a list of the predicted counts for the observations in the data set, use the command:

```
> fitted.values(mod)
```

To obtain a list of residuals for the observations in the data set, use the command:

```
> residuals(mod)
```

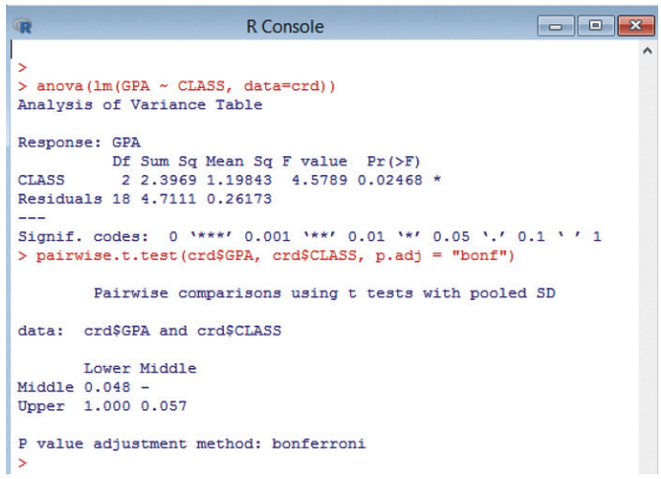
12. One-Way Analysis of Variance

Consider a completely randomized design with a single factor (e.g., ‘‘CLASS’’) and a quantitative response variable (e.g., ‘‘GPA’’). To perform a one-way ANOVA on the data saved in a file (e.g., ‘‘crd’’), use the command:

```
> anova (lm(GPA ~ CLASS, data=crd))
```

The above command will produce an ANOVA table and F-test, as shown in Figure 19.

Figure 19 R Commands for 1-Way ANOVA



To perform multiple comparisons of the factor-level means using Bonferroni’s adjustment method, run the command:

```
> pairwise.t.test (crd ~ GPA, crd$CLASS, p.adj=‘‘bonf’’)
```

Note that within the parentheses, the dependent variable is listed first, followed by the factor variable. See Figure 19.

To perform multiple comparisons of the factor-level means using Tukey's adjustment method, run the following commands:

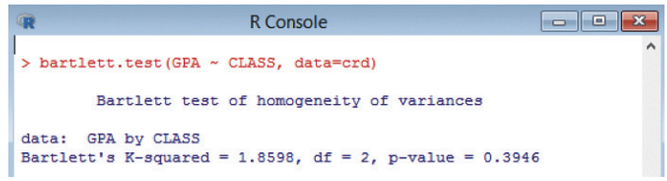
```
> aov (lm(GPA ~ CLASS, data=crd))
> aov2 <- aov(lm(GPA ~ CLASS,data=crd))
> tukey.test <- TukeyHSD(aov2)
> tukey.test
```

To test for the equality of treatment (factor-level) means using Bartlett's test, run the command:

```
> bartlett.test(GPA ~ CLASS, data=crd)
```

See Figure 20.

Figure 20 R Commands for Testing Equality of Variances in a 1-Way ANOVA



To test for the equality of treatment (factor-level) means using Levene's test, run the following commands:

```
> install.packages(''car'')
> library(car)
> leveneTest(crd$GPA, crd$CLASS)
```

Note that within the parentheses, the dependent variable is listed first, followed by the factor variable.

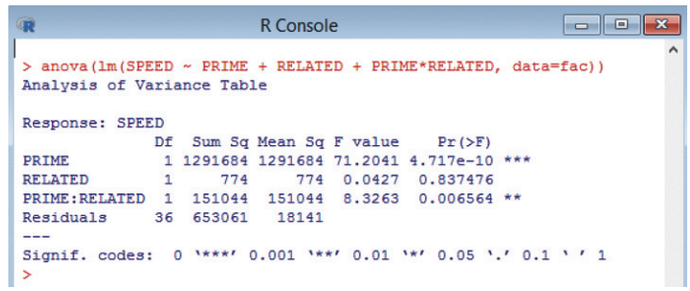
13. Analysis of Variance for Factorial Design

Consider a factorial design with two factors (e.g., "PRIME" and "RELATED") and a quantitative response variable (e.g., "SPEED"). To perform a factorial ANOVA on the data saved in a file (e.g., "fac"), use the command:

```
> anova(lm(SPEED ~ PRIME + RELATED + PRIME*RELATED, data=fac))
```

The command and resulting output are shown in Figure 21.

Figure 21 R Commands for a Factorial ANOVA



Note: A two-way ANOVA table without an interaction term can be generated using the command:

```
> anova(lm(SPEED ~ PRIME + RELATED, data=fac))
```

In general, more factors can be included in the equation by simply inserting the factor name after a "+" sign. If you want to include all possible factor main effects and interaction terms, separate each factor with an asterisk (*) in the equation, as shown in the command below:

```
> anova(lm(SPEED ~ PRIME*RELATED*ACCURACY, data=fac))
```

14. Time Series Forecasting

Consider a time series variable (e.g., "SALES") saved in a data file (e.g., "rev").

Moving Averages:

For moving average smoothing of the variable, first create a vector of weights to use in the smoothing. Let "L" be the length of the moving average. For standard moving average smoothing, each term in the average gets the same weight. The vector of weights, call it "w," can be constructed using the command:

```
> w = rep(1,times=L)/L
```

This command creates a vector of length "L" where each entry has value "1/L." Different weights can be used by manually creating "w" with the desired weights. For a 4-point moving average, L=4 and the command is:

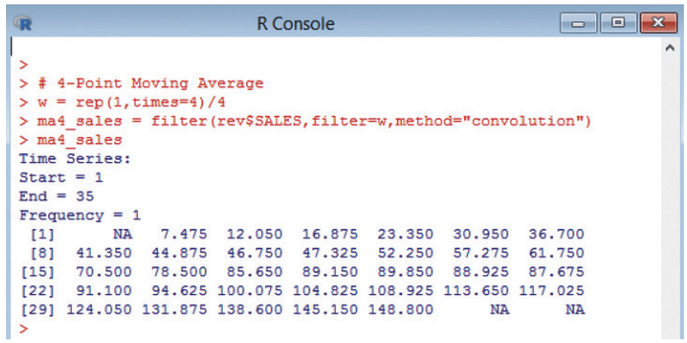
```
> w = rep(1,times=4)/4
```

To smooth the variable "SALES" and store the values in "ma4_sales" use the command:

```
> ma4_sales=filter(rev$SALES,filter=w, method=''convolution'')
```

as shown in Figure 22.

Figure 22 R Commands for Moving Averages



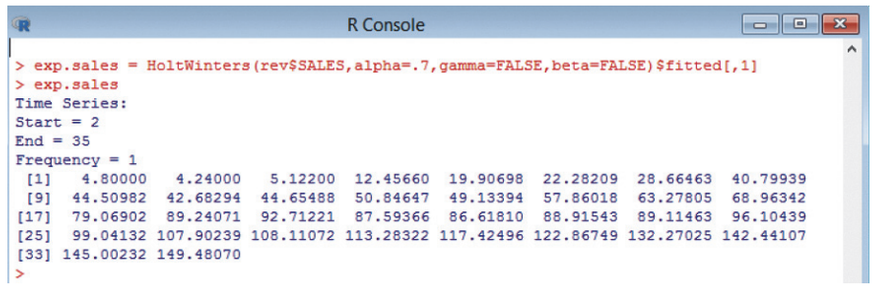
Exponential Smoothing:

For single exponential smoothing, first let "alpha" be the value (e.g., .7) of the exponential smoothing constant. To smooth the time series variable (e.g., "SALES") saved in a data file (e.g., "rev") and store the values in "exp.sales", use the command:

```
> exp.sales = HoltWinters(rev$SALES, alpha=.7, gamma=FALSE,
  beta=FALSE)$fitted[,1]
```

Note: The "[,1]" at the end is included to select the smoothed values. See Figure 23.

Figure 23 R Commands for Exponential Smoothing



For Holt-Winters smoothing, first let "alpha" be the value (e.g., .7) of the exponential smoothing constant and "beta" be the value (e.g., .5) of the trend smoothing constant. To smooth the time series variable (e.g., "SALES") saved in a data file (e.g., "rev") and store the values in "hw.sales", use the command:

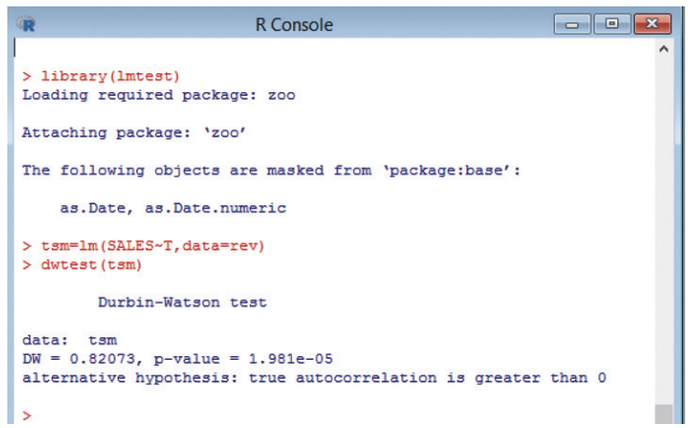
```
> hw.sales = HoltWinters(rev$SALES, alpha=.7, beta=.5,
  gamma=FALSE)$fitted[,1]
```

Durbin-Watson Test:

Consider a time series regression model for a dependent variable (e.g., "SALES") as a function of independent variables (e.g., "T") on a data file (e.g., "rev"). To conduct the Durbin-Watson test for correlated regression errors, run the following commands (as shown in Figure 24):

```
> install.packages(''lmtest'')
> library(lmtest)
> tsm = lm(SALES ~ T, data=rev)
> dwtest(tsm)
```

Figure 24 R Commands for the Durbin-Watson Test



Autoregressive Error Time Series Model:

Consider a time series regression model for a dependent variable (e.g., "SALES") as a function of independent variables (e.g., "T") on a data file (e.g., "rev"). Assume that the error terms in the model follow an AR(1) process. To fit the model, run the following commands (as shown in Figure 25):

```
> install.packages(''nlme'')

> library(nlme)

> gls(SALES ~ T, correlation=corARMA(p=1,q=0), data=rev)
```

Note: In the above command, the value of p specifies the order of the autoregressive portion of the error model; the value of q specifies the order of the moving average portion of the error model.

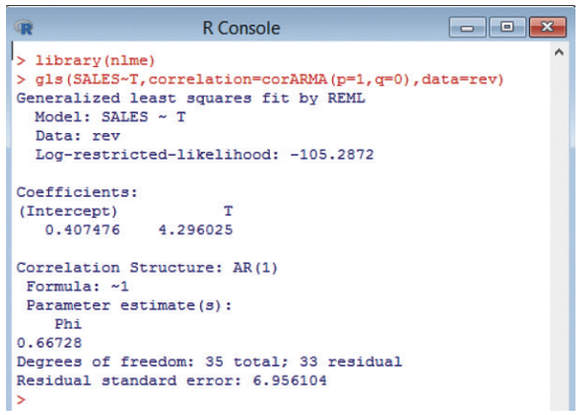


Figure 25 R Commands for the AR(1) Error Regression Model