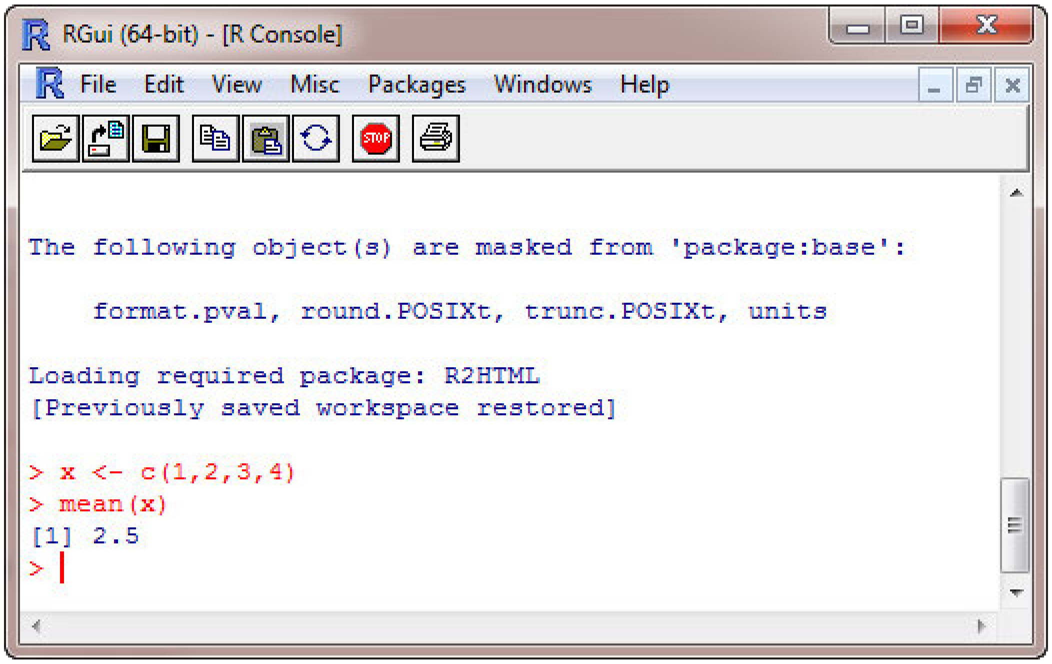
Appendix

Here is a more complete listing of the R code needed in an elementary statistics course.

Section 1: R environment

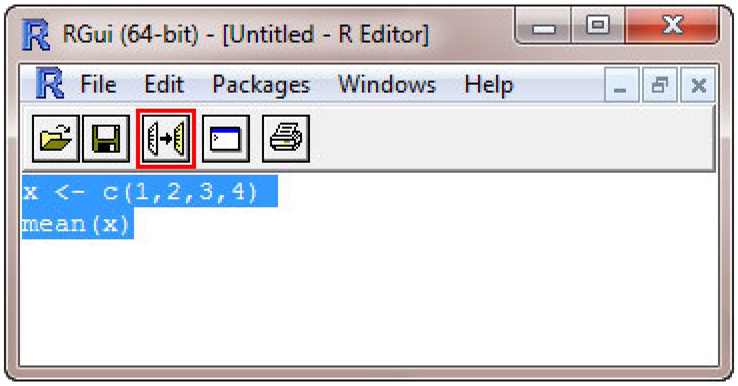
You can download R free of charge from the website cran.r-project.org/. This website also offers manuals, many additional packages and even a journal. The default environment for R is the command line.



```
The following object(s) are masked from 'package:base':

    format.pval, round.POSIXt, trunc.POSIXt, units

Loading required package: R2HTML
[Previously saved workspace restored]

> x <- c(1,2,3,4)
> mean(x)
[1] 2.5
> 
```

Although primitive by modern standards, this interface has some nice features. For example, you can use the arrow keys to scroll through the previously entered commands. This makes it easy to bring down a previous command and fix it or modify it to accomplish a similar task.

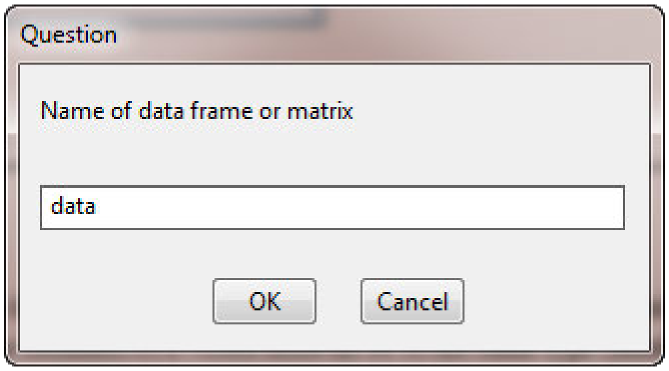
R provides a primitive script editor. Go to File/New script to open the editor and enter the commands. Then press the Run line or selection button. Scripts can be saved to your computer to reduce the typing burden.



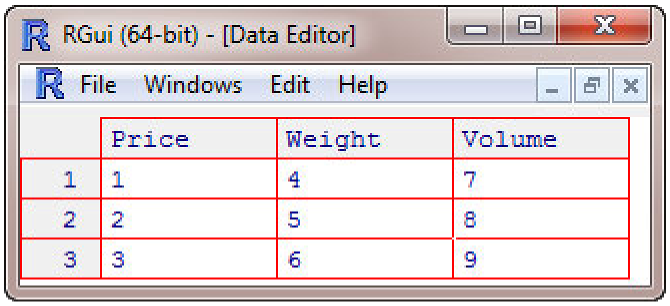
```
x <- c(1,2,3,4)
mean(x)
```

R also provides a data editor which can be used to edit dataframes or matrices. To create a data frame, enter the following command to create an empty data frame. Then go to Edit/Data editor and provide the name for your data frame. You can then provide names for the variables and add or delete data.

```
data <- data.frame( )
```



data



	Price	Weight	Volume
1	1	4	7
2	2	5	8
3	3	6	9

Section 2: Data structures

R supports multiple data structures including vectors, lists, matrices, factors and data frames.

Vectors

Vectors serve as a container for holding data and can be created using the **c** (combine) command. All of the elements in a vector must be of the same type (numerical, string, logical).

```
Calories = c(611, 725, 705, 777, 606)
Meat = c("Turkey", "Turkey", "Ham", "Beef", "Beef")
```

To view the contents of these vectors, or any other variables, simply type in their names.

```
Calories
Meat
```

Lists

Lists may contain data of differing types and allow us to create heterogeneous data structures. Note the use of the assignment operator which assigns the list to the variable *u*.

```
u <- list(firstname ="Fred", children=3, ages=c(4,7,9))
```

The elements of a list can be accessed using the dollar sign ($), which serves as an extraction operator.   If working with a data structure created by R or another user, consider using the names command to see which elements are available for extraction.

```
names(u)
u$children
```

Matrices

To create a matrix in R, we must provide a list of elements that will be used to fill the matrix and the dimensions of the matrix. By default, R fills matrices by column (this default behavior can be changed).

```
m <- matrix(c(1,2,3,4),2,2)
```

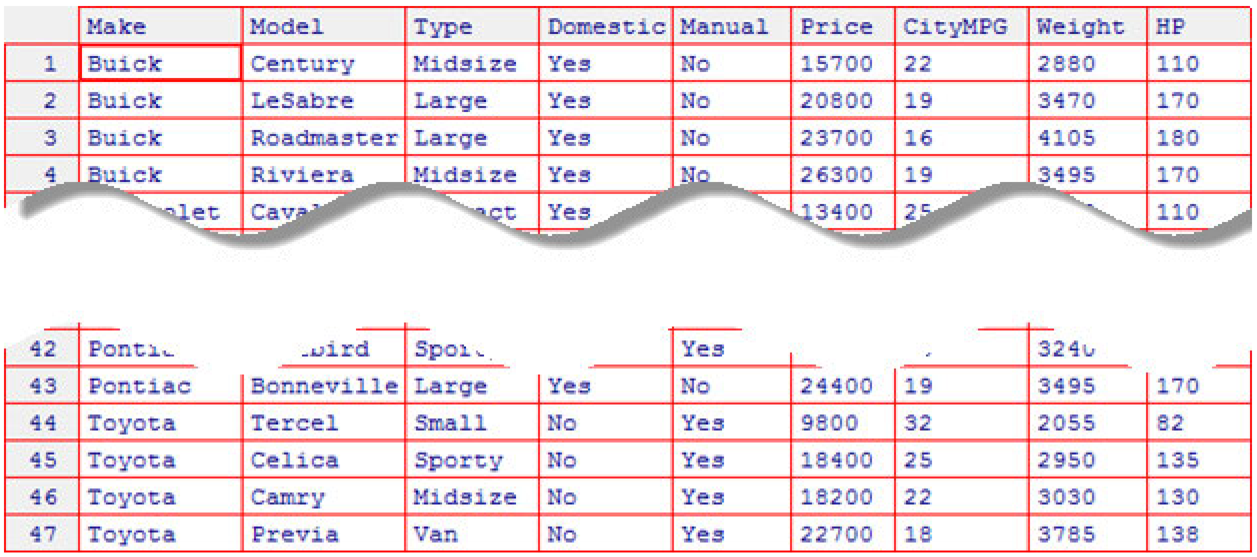
Factors

We can represent categorical variables using factors which store the values of the categorical variable as a vector of integers in the range [1...*k*] where *k* is the number of unique values of the categorical variable. Factors use less memory than character vectors and can be very helpful when using very large data sets.

```
colors <- factor(c("brown", "blue", "brown", "green", "blue"))
levels(colors)
```

Data frames

A data frame is similar to a matrix in that it has a two dimensional structure of rows and columns. Data frames are represented internally in R using a list of vectors representing the columns of the data frame. The primary difference between a matrix and a data frame is that the columns of a data frame are allowed to be of different types (numerical, string, logical). This introduces a level of heterogeneity and usefulness that cannot be achieved using matrices. The data frame shown below contains 8 variables and 47 cars. Each variable is stored in a separate column and the information about each car is stored in a separate row.



	Make	Model	Type	Domestic	Manual	Price	CityMPG	Weight	HP
1	Buick	Century	Midsize	Yes	No	15700	22	2880	110
2	Buick	LeSabre	Large	Yes	No	20800	19	3470	170
3	Buick	Roadmaster	Large	Yes	No	23700	16	4105	180
4	Buick	Riviera	Midsize	Yes	No	26300	19	3495	170
	olet	Cava	act	Yes		13400	25		110
42	Ponti	bird	Spo		Yes			324	
43	Pontiac	Bonneville	Large	Yes	No	24400	19	3495	170
44	Toyota	Tercel	Small	No	Yes	9800	32	2055	82
45	Toyota	Celica	Sporty	No	Yes	18400	25	2950	135
46	Toyota	Camry	Midsize	No	Yes	18200	22	3030	130
47	Toyota	Previa	Van	No	Yes	22700	18	3785	138

The R code for the cars data frame shown above is given on the next page. Copy and paste this data frame into R for use with the examples provided later in this Appendix. Normally, this amount of data would not be entered using copy and paste. A better method would be to use .CSV files as explained in Section 4.

```
cars <- data.frame(

Make = c("Buick", "Buick", "Buick", "Buick", "Chevrolet", "Chevrolet",
"Chevrolet", "Chevrolet", "Chevrolet", "Chevrolet", "Chevrolet",
"Dodge", "Dodge", "Dodge", "Dodge", "Dodge", "Dodge", "Ford", "Ford",
"Ford", "Ford", "Ford", "Ford", "Ford", "Hyundai", "Hyundai",
"Hyundai", "Hyundai", "Mazda", "Mazda", "Mazda", "Mazda", "Mazda",
"Nissan", "Nissan", "Nissan", "Nissan", "Oldsmobile", "Oldsmobile",
"Pontiac", "Pontiac", "Pontiac", "Pontiac", "Toyota", "Toyota",
"Toyota", "Toyota"),

Model = c("Century", "LeSabre", "Roadmaster", "Riviera", "Cavalier",
"Corsica", "Camaro", "Lumina", "Astro", "Caprice", "Corvette", "Colt",
"Shadow", "Spirit", "Caravan", "Dynasty", "Stealth", "Festiva",
"Escort", "Tempo", "Mustang", "Probe", "Aerostar", "Taurus", "Excel",
"Elantra", "Scoupe", "Sonata", "323", "Protege", "626", "MPV", "RX-7",
"Sentra", "Altima", "Quest", "Maxima", "Achieva", "Silhouette",
"LeMans", "Sunbird", "Firebird", "Bonneville", "Tercel", "Celica",
"Camry", "Previa"),

Type = c("Midsize", "Large", "Large", "Midsize", "Compact", "Compact",
"Sporty", "Midsize", "Van", "Large", "Sporty", "Small", "Small",
"Compact", "Van", "Midsize", "Sporty", "Small", "Small", "Compact",
"Sporty", "Sporty", "Van", "Midsize", "Small", "Small", "Sporty",
"Midsize", "Small", "Small", "Compact", "Van", "Sporty", "Small",
"Compact", "Van", "Midsize", "Compact", "Van", "Small", "Compact",
"Sporty", "Large", "Small", "Sporty", "Midsize", "Van"),

Domestic = c("Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes",
"Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes",
"Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "No", "No", "No", "No",
"No", "No", "No", "No", "No", "No", "No", "No", "No", "Yes", "Yes",
"Yes", "Yes", "Yes", "Yes", "No", "No", "No", "No"),

Manual = c("No", "No", "No", "No", "Yes", "Yes", "Yes", "No", "No",
"No", "Yes", "Yes", "Yes", "Yes", "No", "No", "Yes", "Yes", "Yes",
"Yes", "Yes", "Yes", "Yes", "No", "Yes", "Yes", "Yes", "Yes", "Yes",
"Yes", "Yes", "No", "Yes", "Yes", "Yes", "No", "No", "No", "No",
"Yes", "Yes", "Yes", "No", "Yes", "Yes", "Yes", "Yes"),
Price = c(15700, 20800, 23700, 26300, 13400, 11400, 15100, 15900,
16600, 18800, 38000, 9200, 11300, 13300, 19000, 15600, 25800, 7400,
10100, 11300, 15900, 14000, 19900, 20200, 8000, 10000, 10000, 13900,
8300, 11600, 16500, 19100, 32500, 11800, 15700, 19100, 21500, 13500,
19500, 9000, 11100, 17700, 24400, 9800, 18400, 18200, 22700),

CityMPG = c(22, 19, 16, 19, 25, 25, 19, 21, 15, 17, 17, 29, 23, 22,
17, 21, 18, 31, 23, 22, 22, 24, 15, 21, 29, 22, 26, 20, 29, 28, 26,
18, 17, 29, 24, 17, 21, 24, 18, 31, 23, 19, 19, 32, 25, 22, 18),

Weight = c(2880, 3470, 4105, 3495, 2490, 2785, 3240, 3195, 4025, 3910,
3380, 2270, 2670, 2970, 3705, 3080, 3805, 1845, 2530, 2690, 2850,
2710, 3735, 3325, 2345, 2620, 2285, 2885, 2325, 2440, 2970, 3735,
2895, 2545, 3050, 4100, 3200, 2910, 3715, 2350, 2575, 3240, 3495,
2055, 2950, 3030, 3785),

HP = c(110, 170, 180, 170, 110, 110, 160, 110, 165, 170, 300, 92, 93,
100, 142, 100, 300, 63, 127, 96, 105, 115, 145, 140, 81, 124, 92, 128,
82, 103, 164, 155, 255, 110, 150, 151, 160, 155, 170, 74, 110, 160,
170, 82, 135, 130, 138))
```

The variables within the data frame can be accessed using the dollar sign ($), which serves as an extraction operator.

```
cars$Price
```

Storing data in a data frame increases the length of the variables and adds to the typing burden. Here are some options for dealing with the lengthy names created by using data frames.

1. **Live with the long names**. For example, to find the average price of the cars use the command.

   ```
   mean(cars$Price)
   ```

2. **Assign a shorter name to the variable**. For example, to find the mean and standard deviation of the price variable use

   ```
   p <- cars$Price
   mean(p)
   sd(p)
   ```

3. **Use the attach command**. This permits us to refer to variables in a data frame without being prefixed by the name of the data frame.

   ```
   attach(cars)
   mean(Price)
   sd(Price)
   ```

   Using the attach command has some disadvantages. It makes it easy to create multiple copies of a variable, which can lead to errors which are difficult to detect. Thus, it is wise to use the detach command after you finish working with a data frame.

   ```
   detach(cars)
   ```

   If you end up making a mess of your namespace, use the following commands to see the names of the objects stored in memory and erase them.

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

4. **Use the with command**. This option avoids the difficulties associated with the attach command but increases the amount of typing required and clutters up the commands.

   ```
   with(cars, mean(Price))
   ```

5. **Specify the data environment**. This option is nice, but is not available for all functions. It is supported by most modeling functions. Here is how to find the least squares line for the price versus the weights of cars.

   ```
   lm(Price ~ Weight, data=cars)
   ```

Section 3: Data input

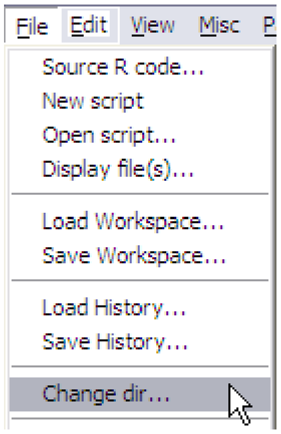
Keyboard (default)

The default behavior for R is to accept input and send output using the keyboard in an interactive session.

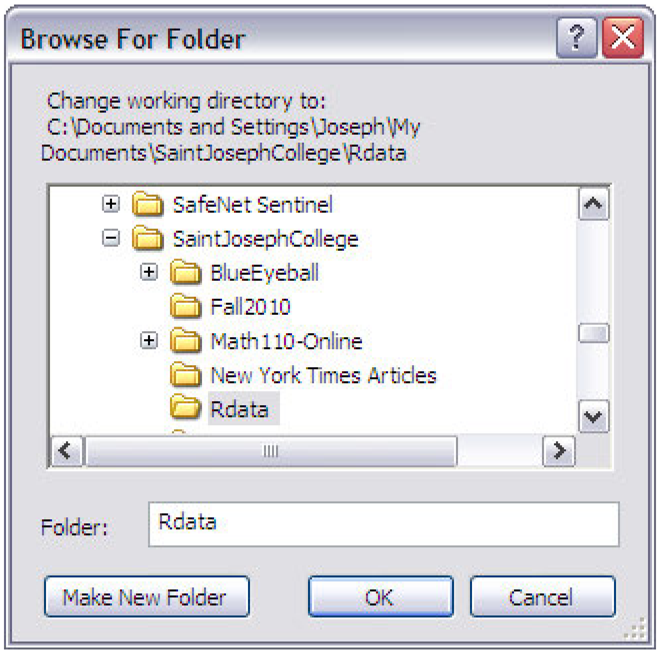
```
> 1+2
```

Set the working directory

If you want to perform input/output operations using files, you will need to tell R where to look. This can be done within each of the input/output commands directly by specifying the full path. However, it is frequently easier to simply tell R which directory to use as the working directory. Then you can simply use filenames for input/output operations, there is no need to specify the complete path. To set the working directory, select **File/Change dir…**. which opens the Browse For Folder dialog box.



Navigate to the folder containing your data and press OK. This will allow R to find your data files, provided that you wisely decide to store your data in the folder you have specified.



Change working directory to:
C:\Documents and Settings\Joseph\My Documents\SaintJosephCollege\Rdata

Folder: Rdata

You can check to make sure you are in the correct folder using the **getwd** (get working directory) command.

```
getwd( )
```

Many data files are large and already available. It would be a waste of time to reenter such data sets using the keyboard. Mercifully, R allows us to import data from files. One of the most commonly used ASCII data formats is comma-separated-values (CSV) format. Files of this type can be imported and exported into and out of many spreadsheets, databases and statistical software packages. For example, I have entered the cars data frame into an Excel spreadsheet and saved it as a CSV file (File/Save As/Save As Type/CSV-comma delimited) with the name **cars.csv**. It is then easy to read the data frame into R using the following command.

```
cars <- read.csv("cars.csv")
```

Section 4: Numerical summaries

Use the **table** command to create a frequency table.

```
table(cars$Make)
```

The **table** command can also be used to create contingency tables.

```
table(cars$Type, cars$Domestic)
```

Use the **mean**, **sd**, **median** and **quantile** commands to compute the mean, standard deviation, median, quartiles and deciles.

```
mean(cars$Price)
sd(cars$Price)
median(cars$Price)
quantile(cars$Price)
quantile(cars$Price, probs=seq(0,1,0.1))
```

If you need to compute numerical summaries by factor, consider using the aggregate command. Here is how to compute the means for the prices of cars by type.

```
aggregate(cars$Price, by=list(cars$Type), FUN=mean)
```

The output for this command is not pretty since the column labels are not helpful. To fix this, note that the output of the **aggregate** command is a dataframe and that we can rename the variables within this dataframe using the **names** command.

```
a <- aggregate(cars$Price, by=list(cars$Type), FUN=mean)
names(a) <- c("Type","Mean")
a
```

Here is how to compute the means and standard deviations for the prices of cars by type and whether or not the car is domestic.

```
a <- aggregate(cars$Price, by=list(cars$Type,cars$Domestic),
FUN=function(d) c(length=length(d),mean=mean(d), sd=sd(d)))
names(a) <- c("Type", "Domestic", "cars")
a
```

Section 5: Graphs

Pie charts

One of the most confusing tasks for students and instructors learning a new statistical software package is to determine whether they are working with summarized or raw data. This distinction is important and most statistical software packages including R provided tools for working with both. For summarized data, send the **pie** command a vector containing the counts.

```
pie(c(10, 20, 35, 16))
```

For raw data, first use the **table** command to create a frequency table and then use the **pie** command.

```
pie(table(cars$Type))
```

As you can see, the default pie chart is very basic. Adding labels containing percentages is fairly labor intensive.

```
# t = frequency table for the makes of cars
# p = frequencies expressed as percentages
# l = labels containing the makes and percentages
```

```
t <- table(cars$Type)
p <- round(100*t/sum(t),1)
l <- paste(names(t), " ", p, "%", sep="")
pie(t, labels=l)
```

Bar charts

For summarized data, send the **barplot** command a vector containing the counts or percentages. For raw data, first use the **table** command to create a frequency table and then use the **barplot** command.

```
# summarized data
# bar chart of counts
# bar chart of percentages
# raw data

d <- c(10, 20, 35, 16)
barplot(d)
barplot(d/sum(d))
barplot(table(cars$Type),col="Skyblue")
```

Stem plot

Creating stem plots is simple. The scale parameter can be used to expand the height of the plot.

```
# default stem plot
# double the height

stem(cars$Price)
stem(cars$Price, scale=2)
```

Histograms

Here are some options for creating histograms; default, relative frequency, specifying the number of classes and specifying the class breaks.

```
# default histogram
# relative frequency histogram
# histogram with 4 classes
# histogram with interval [5000, 50000] and a class width of 5000

hist(cars$Price)
hist(cars$Price, prob=T)
hist(cars$Price, breaks=3)
hist(cars$Price, breaks=seq(5000,50000,5000))
```

The default histogram and the permutations provided above are very basic. However, we can use the **hist** command's many optional arguments to achieve greater control over the graph.

```
hist(x, breaks = "Sturges",
     freq = NULL, probability = !freq,
     include.lowest = TRUE, right = TRUE,
     density = NULL, angle = 45, col = NULL, border = NULL,
     main = paste("Histogram of" , xname),
     xlim = range(breaks), ylim = NULL,
     xlab = xname, ylab,
     axes = TRUE, plot = TRUE, labels = FALSE,
     nclass = NULL, warn.unused = TRUE, ...)
```

As you will see, using these options can produce a more attractive (or at least eye catching) graph.

```
hist(cars$Price, breaks=seq(5000,50000,5000),
                   main="Prices of Cars",
                   xlab="Price",
                   ylab="Weight",
                   col=c("Red","Yellow"))
```

Box plots

Use the **boxplot** command to create a box plot.

```
boxplot(cars$Price)
```

When using the **boxplot** command to produce side-by-side boxplots, we need to be mindful of the data structures involved. If the data is stored in separate vectors, send the boxplot command the names of the vectors separated by commas.

```
# vector containing white blood cell counts for males
# vector containing white blood cell counts for females
# side-by-side boxplot

males <- c(5.25, 5.95, 10.05, 5.45, 5.30, 5.55)
females <- c(8.90, 6.50, 9.45, 7.65, 6.40, 5.15, 16.60)
boxplot(males, females, names=c("Males","Females"))
```

If the data is stored in a data frame, use the modeling notation (this notation is explained in the section on correlation and regression).

```
boxplot(cars$Price ~ cars$Domestic, col="Skyblue")
```

Scatterplot

Use the plot command to create a scatterplot. The pch option is used to select the plotting symbol, in this case a filled circle.

```
plot(cars$CityMPG, cars$Weight, pch=19, col="red")
```

Quantile-Quantile plot

Use the **qqnorm** command to create a normal quantile plot. After creating the plot use the **qqline** command to add the theoretical line to the plot on which the data would fall if it were normally distributed. Here is the normal quantile plot that can be used to discuss whether or not the prices of cars follow a normal distribution.

```
p <- cars$Price
z <- (p-mean(p))/sd(p)
qqnorm(z)
qqline(z)
```

Section 6: Correlation and regression

The most basic modeling notation is

```
y ~ x
```

where *x* is the explanatory variable and *y* is the response variable. For higher degree polynomial regression models, the function I( ) is used to tell R to treat the variable “as is” and not to actually compute the quantity.   For example, use the following notation to model quadratic regression.

```
y ~ x + I(x^2)
```

For higher degree polynomial regression, this notation becomes tedious and it may be best to use the poly command. For example, use the following to model cubic regression.

```
y ~ poly(x,3)
```

The number of possibilities increases dramatically when we introduce more variables. Consider the situation where *x* and *y* are explanatory variables and *z* is the response variable. Then the most basic model is the following.

```
z ~ x + y
```

To create interaction terms use the colon (":").   Here is the model which included the simple effects of the factors *x* and *y* and the interaction between them.

```
z ~ x + y + x:y
```

A shorter way to do the above is to use "*", which is shorthand for including the simple effects and the interactions.

```
z ~ x*y
```

Correlation coefficient

Use the **cor** command to compute the correlation coefficient between the city mileage and weight variables.

```
cor(cars$CityMPG, cars$Weight)
```

Linear regression

We can create a scatter plot using the **plot** command.

```
plot(cars$Weight, cars$CityMPG)
```

Use the **lm** (linear models) command to determine the equation of the least squares line. Extract the coefficients using the dollar sign ($) and add the graph of the line to the scatter plot using the **abline** command.

```
m <- lm(cars$CityMPG ~ cars$Weight)
m$coefficients
abline(m)
```

Comments:

1. Use the **names** command to see a complete list of the information produced by the **lm** command.

   ```
   names(m)
   [1] "coefficients"   "residuals"   "effects"   "rank"
   [5] "fitted.values"   "assign"   "qr"   "df.residual"
   [9] "xlevels"   "call"   "terms"   "model"
   ```

2. The **abline** command is a low graphics command that can be used to add lines to an existing graph. For example, **abline(h=4)** adds the horizontal line $y = 4$, **abline(v=3)** adds the vertical line $x = 3$ and **abline(a=1, b=2)** adds a line with y-intercept 1 and slope 2.

The **plot** command can also be used to plot the residuals.

```
plot(cars$Weight, m$residuals)
abline(h=0)
```

Multiple linear regression

Multiple linear regression is a matter of adding more explanatory variables and can also be performed using the **lm** command. Here is how find the multiple linear regression equation for city mileage as a function of weight and horsepower.

```
m <- lm(cars$CityMPG ~ cars$Weight + cars$HP)
coefficients(m)
```

Nonlinear regression

The **lm** command can also perform nonlinear regression. For example, here is how to determine the quadratic model for city mileage as a function of weight.

```
# quadratic regression model
# print a summary of the model
# create scatterplot
# x – values containing the range of car weights
# y – values predicted by the model
# add the least squares parabola to the scatterplot

model <- lm(cars$CityMPG ~ poly(cars$Weight,2))
summary(model)
plot(cars$Weight, cars$CityMPG)
x <- 1800:4200
y <- predict(model, list(cars$Weight=x))
lines(x,y)
```

Section 7: Probability distributions

R provides four functions for working with each of the probability distributions.

1. Compute the values of the probability densities
2. Compute cumulative probabilities
3. Compute quantiles and the
4. Generate random numbers from the distribution.

Binomial distribution

The binomial distribution is defined by

$$\text{binomial}(k,n,p)=\binom{n}{k}p^k(1-p)^{n-k}, \quad k=0,1,2,\ldots,n$$

and the commands for working with the binomial distribution in R are given by

```
# one binomial probability
# several binomial probabilities
# sum of several binomial probabilities
# cumulative binomial probability
# inverse cumulative binomial probability
# random numbers from binomial distribution
```

```
dbinom(k, n, p)
dbinom(k1:k2, n, p)
sum(dbinom(k1:k2, n, p))
pbinom(k, n, p)
qbinom(percentile, n, p)
rbinom(size, n, p)
```

Example: Plot the binomial distribution $x \sim \text{Binomial}(50, 0.30)$

```
plot(dbinom(0:50, 50, 0.30))
```

Geometric distribution

The geometric distribution is defined by

$$\text{geometric}(k, p) = (1-p)^{k-1} \cdot p, \quad k = 0, 1, 2, 3, \ldots$$

and the commands for working with the geometric distribution in R are given by

```
# one geometric probability
# several geometric probabilities
# sum of several geometric probabilities
# cumulative geometric probability
# inverse cumulative geometric probability
# random numbers from geometric distribution

dgeom(k, p)
dgeom(k1:k2, p)
sum(dgeom(k1:k2, p))
pgeom(k, p)
qgeom(percentile, p)
rgeom(size, p)
```

Poisson distribution

The Poisson distribution is defined by

$$\text{Poisson}(k, \lambda) = \frac{\lambda^k e^{-\lambda}}{k!}, \quad k = 0, 1, 2, 3, \ldots$$

and the commands for working with the Poisson distribution in R are given by

```
# one Poisson probability
# several Poisson probabilities
# sum of several Poisson probabilities
# cumulative Poisson probability
# inverse cumulative Poisson probability
# random numbers from Poisson distribution

dpois(k, lambda)
dpois(k1:k2, lambda)
sum(dpois(k1:k2, lambda))
ppois(k, lambda)
qpois(percentile, lambda)
rpois(size, lambda)
```

Normal distribution

The normal distribution is defined by

$$\text{normal}(x,\mu,\sigma)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}, \quad -\infty<x<\infty$$

and the commands for working with the normal distribution in R are given by

```
# normal density function
# cumulative normal distribution
# inverse cumulative normal distribution
# random numbers from normal distribution

dnorm(x, mean, sd)
pnorm(x, mean, sd)
qnorm(percentile, mean, sd)
rnorm(size, mean, sd)
```

Example: (Workshop Statistics) The birth weights of babies are normally distributed with a mean of 3250 grams and a standard deviation of 550 grams. Determine the following.

a. percentage of babies weighing less than 2500 grams
```
pnorm(2500, 3250, 550)
[1] 0.08634102
```
b. percentage of babies weighing more than 4536 grams
```
1 - pnorm(4536, 3250, 550)
[1] 0.009688909
```
c. percentage of babies weighing between 3000 and 4000 grams
```
pnorm(4000, 3250, 550) - pnorm(3000, 3250, 550)
[1] 0.5889408
```
d. weight needed to be in order to be in the lightest 2.5%
```
qnorm(0.025, 3250, 550)
[1] 2172.02
```
e. weight needed to be in the heaviest 10%
```
qnorm(0.90, 3250, 550)
[1] 3954.853
```
f. range of weights needed to be in the middle 90%.
```
qnorm(0.05, 3250, 550)
[1] 2345.331
qnorm(0.95, 3250, 550)
[1] 4154.669
```

Student-t distribution

The Student-t distribution is defined by

$$\text{student}(t,n)=\frac{\Gamma\left(\frac{n+1}{2}\right)}{\Gamma\left(\frac{n}{2}\right)\sqrt{n\pi}}\left(1+\frac{t^2}{n}\right)^{-\left(\frac{n+1}{2}\right)}, \quad -\infty<t<\infty$$

and the commands for working with the Student-t distribution in R are given by

```
# student-t density function
# cumulative student-t distribution
# inverse cumulative student-t distribution
# random numbers from student-t distribution
```

```
dt(x, n)
pt(x, n)
qt(percentile, n)
rt(size, n)
```

Chi-square distribution

The chi-square is defined by

$$\text{chisquare}(x,n) = \frac{1}{2^{\frac{n}{2}}\Gamma\left(\frac{n}{2}\right)} x^{\frac{n-2}{2}} e^{-\frac{x}{2}}, \quad x > 0$$

and the commands for working with the chi-square distribution in R are given by

```
chi-square density function
cumulative chi-square distribution
inverse cumulative chi-square distribution
random numbers from chi-square distribution

dchisq(x, n)
pchisq(x, n)
qchisq(percentile, n)
rchisq(size, n)
```

F-distribution

The F-distribution is defined by

$$F(x,n_1,n_2) = \frac{\Gamma\left(\frac{n_1+n_2}{2}\right)\cdot\left(\frac{n_1}{n_2}\right)^{\frac{n_1}{2}}}{\Gamma\left(\frac{n_1}{2}\right)\Gamma\left(\frac{n_2}{2}\right)} \cdot x^{\frac{n_1}{2}-1} \cdot \left(1+\frac{n_1}{n_2}x\right)^{-\frac{1}{2}(n_1+n_2)}, \quad x > 0$$

and the commands for working with the F-distribution in R are given by

```
f density function
cumulative f distribution
```

```
inverse cumulative f distribution
random numbers from f distribution

df(x, n1, n2)
pf(x, n1, n2)
qf(percentile, n1, n2)
rf(size, n1, n2)
```

Section 8: Sampling

The **sample** command can be used to perform random sampling. The size option specifies the sample size (the default being the size of the population). The replace option determines whether the sample will be drawn with or without replacement (the default value is FALSE). The prob option allows you to specify the probability distribution for selection, the default being that each element has an equal chance of being selected.

```
sample(x, size, replace = FALSE, prob = NULL)
```

Here is how to select a random sample of the city mileage for 20 cars without replacement,

```
sample(cars$CityMPG,20)
```

Section 9: Inferential statistics

Proportions

The **prop.test** command has the ability to create confidence intervals and perform significance tests for proportions. For the one sample confidence intervals and significance tests, enter the number of successes *x* and the number of trials *n*.

```
prop.test(x, n, p = NULL,
alternative = c("two.sided", "less", "greater"),
conf.level = 0.95, correct = TRUE)
```

Example: In a New York Times article (July 30, 2009), 725 people out of 1050 interviewed had concerns that the quality of their health care could decline.

1. Compute a 95% confidence interval for the proportion of Americans who have concerns that the quality of their health care could decline.

2. Perform a 1-sample proportion test to determine if the proportion of Americans who have concerns that the quality of their health care could decline is different from 0.75.

The **prop.test** command computes the $(1-\alpha)\times 100\%$ Wilson confidence interval

$$\frac{\left(\hat{p}+\frac{z_{1-\alpha/2}^2}{2n}\right)\pm z_{1-\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{n}+\frac{z_{1-\alpha/2}^2}{4n^2}}}{1+\frac{z_{1-\alpha/2}^2}{n}}$$

```
prop.test(725, 1050, conf.level=0.95)
```

The Wald confidence interval

$$\hat{p}\pm E, E = z_C\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

used in many elementary statistics textbooks performs poorly (especially when the population proportion is near 0 or 1) and is not included in R. Here is to compute the Wald interval.

```
# sample proportion
# sample size
# confidence level
# standard error
# z-score
# Wald confidence interval

p <- 725/1050
n <- 1050
c <- 0.95
se <- sqrt(p*(1-p)/n)
z <- qnorm((1+c)/2)
c(p-z*se, p+z*se)
```

The **prop.test** command can also be used to perform the one-sample proportion test.

```
prop.test(725,1050, p=0.75, alternative="two.sided")
```

Means

The **t.test** command has the ability to create confidence intervals and perform significance tests for means and for comparing two means.

```
t.test(x, y = NULL,
       alternative = c("two.sided", "less", "greater"),
       mu = 0, paired = FALSE, var.equal = FALSE,
       conf.level = 0.95, ...)
```

Example: Consider the prices of the cars stored in the cars data frame.

1. Compute a 95% confidence interval for the average price of a car.
2. Perform a one-sample t-test to determine if whether or not the average price of a car is different from $15,000.

```
# confidence interval
# one-sample t-test

t.test(cars$Price, conf.level=0.95)
t.test(cars$Price, mu = 20000, alternative = "two.sided")
```

Comparing means

When comparing two means, we need to be mindful of the data structures involved. If the data is stored in separate vectors, send the t-test command the vectors. If the data is stored in a data frame, use modeling notation.

Example: Consider the difference between the white blood cell counts of male and females.

1. Compute a 95% confidence interval for the difference in means of the white blood cell counts of males and females.
2. Perform an independent samples t-test to determine whether or not there is a statistically significant difference between the white blood cell counts of males and females.

```
# white blood cell counts for males
# white blood cell counts for females
# confidence interval
# independent-samples t-test
```

```
males <- c(5.25, 5.95, 10.05, 5.45, 5.30, 5.55)
females <- c(8.90, 6.50, 9.45, 7.65, 6.40, 5.15, 16.60)
t.test(males, females, conf.level = 0.95)
t.test(males, females, alternative = "two.sided")
```

Example: Perform an independent-samples t-test to determine whether or not there is a statistically significant difference in the prices of domestic and foreign cars.

```
# confidence interval
# independent-samples t-test

t.test(cars$Price ~ cars$Domestic, conf.level = 0.95)
t.test(cars$Price ~ cars$Domestic, alternative = "two.sided")
```

The matched pairs confidence interval and t-test are implemented in a similar manner. The key is to set the paired option to true.

Example: Given the weights of 6 people before and after a weight loss program.

1. Compute a 95% confidence interval for the difference.
2. Perform a paired-samples t-test to determine whether or not there is a statistically significant weight loss.

```
# weight before
# weight after
# confidence interval
# matched-pairs t-test

before <- c(155, 168, 172, 180, 174, 142)
after <- c(152, 170, 166, 180, 160, 143)
t.test(before, after, paired=T, conf.level = 0.95)
t.test(before, after, paired=T, alternative = "greater")
```

Chi-square tests

The **chisq.test** command has the ability to perform the chi-square test for goodness of fit and for independence. If *x* is a matrix with one row or column, or if *x* is a vector and *y* is not given, then a goodness-of-fit test is performed. If *x* is a matrix with at least two rows and columns, it is taken as a two-dimensional contingency table and a chi-square test for independence is performed.

```
chisq.test(x, y = NULL, correct = TRUE,
p = rep(1/length(x), length(x)), rescale.p = FALSE,
simulate.p.value = FALSE, B = 2000)
```

Example: A die was cast 300 times and yielded the frequencies shown below. Perform a chi-square test for goodness-of-fit to determine whether or not the die is fair.

Value	1	2	3	4	5	6
Count	43	49	56	45	66	41

```
# Observed counts
# Probabilities
# Chi-square goodness-of-fit

counts <- c(43, 49, 56, 45, 66, 41)
probs <- c(1/6, 1/6, 1/6, 1/6, 1/6, 1/6)
chisq.test(counts, p=probs, correct=F)
```

Note: The continuity correction | O – E – 0.5| has been disabled in order to provide greater consistency with the methods presented in elementary statistics.

Example: A random sample of 1000 adults was classified according to gender and whether or not they were color-blind. Perform a chi-square test for independence to determine whether or not there is a statistically significant association between gender and color blindness.

	Male	**Female**
Normal	442	514
Color blind	38	6

```
# Matrix of observed counts
# Add row and column labels
# Chi-square test for independence

c <- matrix(c(442, 514, 38, 6), nrow=2, byrow=T)
dimnames(c) <- list(c("Normal", "Color-blind"), c("Male", "Female"))
chisq.test(c, correct=F)
```

The chisq.test command (and many other R commands) generates an output object. Use the names command to view the contents of this output.

```
c <- chisq.test(c)
```

```
names(c)
```

```
[1] "statistic" "parameter" "p.value"   "method"    "data.name"
[6] "observed"  "expected"  "residuals" "stdres"
```

Then use the dollar sign ($) to extract the observed and expected counts.

```
c$observed
c$expected
```

If you are implementing the chi-square test for independence on raw data (stored in a data frame), you can use the table command to create the contingency table first.

Example: Perform a chi-square test for independence to determine whether or not there is a statistically significant association between whether or not a car is domestic and whether or not it has a manual transmission.

```
# Create table of observed counts
# Add row and column labels
# Chi-square test for independence
# Print observed counts, chi-square test and expected counts

t <- table(cars$Domestic, cars$Manual)
dimnames(t) <- list(c("Domestic", "Foreign"), c("Manual
transmission","No manual transmission"))
c <- chisq.test(t, correct=F)
t; c; c$expected
```

Analysis of variance

The simplest way to perform a one-way analysis of variance is to use the **oneway.test** command. The advantage of using this command is that it does not assume equal variances. It accomplishes this by applying a Welch correction for nonhomogeneity. To turn off this correction so that the command produces the same results that students obtain by hand, use the option var.equal = T. Another approach is to use the **aov** command, which always assumes equal variances. Here is how to perform a one-way ANOVA to determine whether or not there is a statistically significant difference in the prices of cars by make.

```
oneway.test(cars$Price ~ cars$Make)
summary(aov(cars$Price ~ cars$Make))
```

Perform a two-way ANOVA to determine whether or not there is a statistically significant difference in the prices of cars by the make and type of car.

```
summary(aov(Price ~ Make*Type, data=cars))
```

Interaction plots show the mean (or other summary) of the response variable for two-way combinations of factors and provide a method for illustrating possible interactions. The **interaction.plot** command may be used to create interaction plots.

```
interaction.plot(x.factor, trace.factor, response, fun = mean, type =
c("l", "p", "b", "o", "c"),...)
```

Example: Create an interaction plot of the prices of the cars versus their make and type

```
interaction.plot(cars$Make, cars$Type, cars$Price, fun = mean)
```

Power

Use the R menu **Packages/Install Package(s)…** to install the **pwr** package. Then use **Packages/Load Package…** to load the **pwr** package. To determine the power of the t-tests, use the command. Exactly one of the parameters d, n, power and sig.level must be passed as NULL, and that parameter is determined from the others.

```
pwr.t.test(n = NULL, d = NULL, sig.level = 0.05, power = NULL, type =
c("two.sample", "one.sample", "paired"), alternative = c("two.sided",
"less","greater"))
```

Compute the power of a single sample t-test, use a command of the form.

```
pwr.t.test(n=60, d=0.2, sig.level=0.10, type="one.sample",
alternative="two.sided")
```

To plot a power curve, compute the power values at a sequence of points and use the **plot** command. I have used the command names(dp) to learn that dp$power can be used to access the power values.

```
d <- seq(-1,1,0.02)
dp <- pwr.t.test(d=d,n=60, sig.level=0.10, type="one.sample",
alternative="two.sided")
plot(d, dp$power, type="l")
```

This package contains additional commands for computing the power of other types of tests.

- pwr.p.test: one sample proportion tests
- pwr.2p.test: two proportions (same sample sizes)
- pwr.2p2n.test: two proportions (different sample sizes)
- pwr.t2n.test: two sample t-tests (different sample sizes)
- pwr.chisq.test power calculations for chi-squared tests
- pwr.anova.test: balanced one-way analysis of variance tests

You can learn more about these commands by using the help command. For example, help(pwr.p.test)

Section 10: Closing considerations

Only a small portion of R's capabilities were explored in this appendix. Here are a few brief comments about some of R's additional functionality.

Subsetting

R makes is easy to select the portion of a data set.

```
# price of first five cars
# price of small cars
# horsepower of small cars costing less than $9,000 dollars

cars$Price[1:5]
cars[cars$Type=="Small", "Price"]
cars[(cars$Type=="Small" & cars$Price<9000),"HP"]
```

Object oriented nature of R

Everything in R is an object and R supports an object oriented including encapsulation, polymorphism and inheritance. Here are some R commands that will allow you to learn more about some commonly used objects.

```
# get the names of the variables in the cars data set

names(cars)
```

```
# set the names for every item in a vector or a list

sales <- c(3400, 2500, 1900)
names(sales) <- c("January", "February", "March")
barplot(sales)sales <- c(3400, 2500, 1900)

# determine the class of an object

class(cars)
class(cars$Price)

# display the structure of an object
# more detailed than the class

str(cars)
str(cars$Price)
```

Section 11: Simulation

Simulation of central limit theorem for proportions

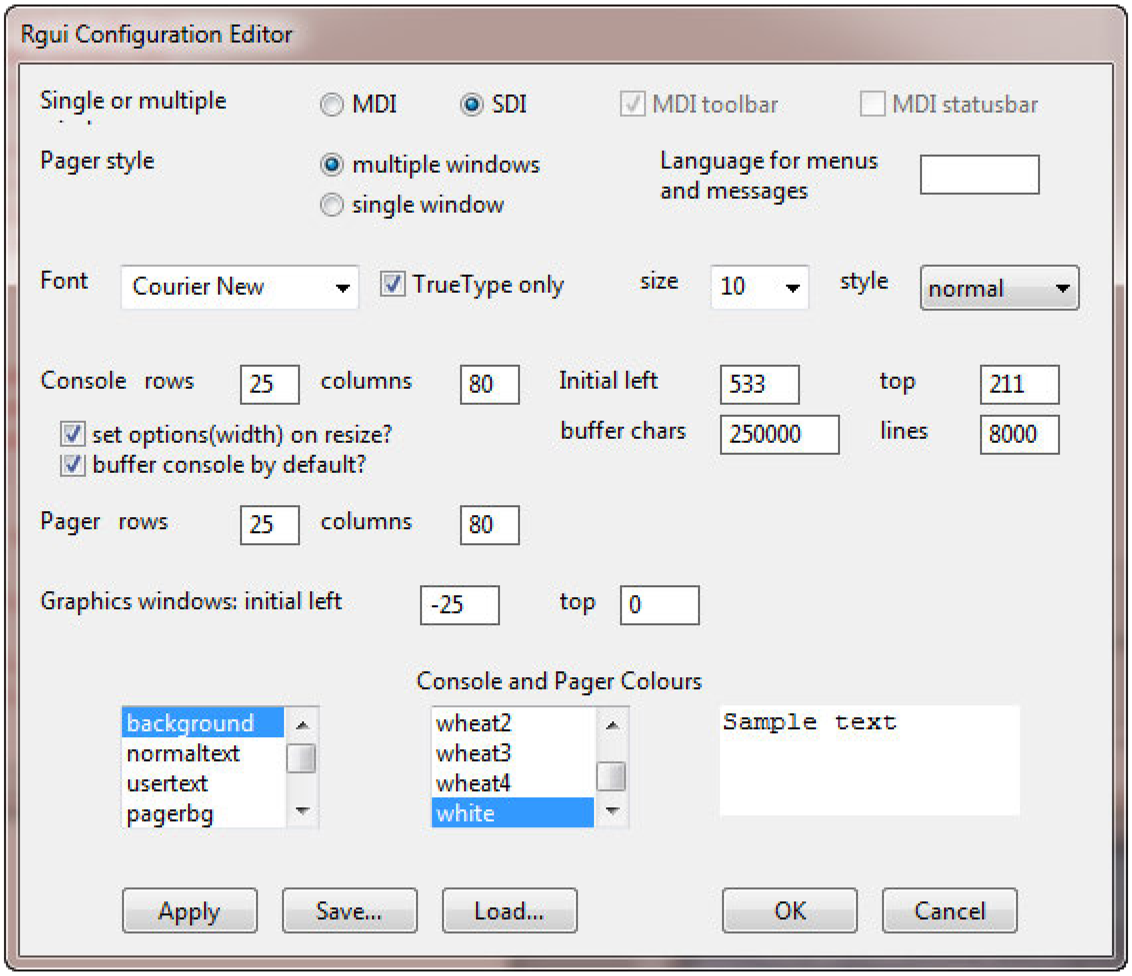
```
# =============================
# sample size
# population proportion
# sample proportions
# rescale (z-score)
# relative frequency histogram
# add standard normal curve
# ============================

n <- 50
p <- 0.20
k <- rbinom(1000,n,p)
s <- (k/n-p)/sqrt(p*(1-p)/n)
hist(s, freq=F, col="Skyblue")
curve(dnorm(x,0,1),-3,3,add=T)
```

Plot the coverage of the Wald confidence interval. This is based on a simulation, not an exact mathematical analysis which tends to produce inequalities.

```
# ================================
# Define Wald confidence interval
# ================================

wald <- function(k,n,c) {
     p <- k/n
     se <- sqrt(p*(1-p)/n)
     zc <- qnorm((1+c)/2)
     return(p+c(-1,1)*zc*se)
}

# ==========================================================

# Plot the coverage probability for Wald confidence interval
# c   : confidence level
# n   : sample size
# p   : vector containing population proportions
# cov : vector containing coverage probabilities
# reps : number of CI’s generated for each value of p
# k : binomial random variable (n, p)
# ci : confidence interval
# count : number of confidence interval containing p
# if statement tests if confidence interval contains p
# ==========================================================

Coverage <- function(c, n) {
     p <- seq(0.005,0.995,0.005)
     cov <- NULL
     reps <- 1000
     for (j in 1:length(p))
     {
     count = 0
       for (i in 1:reps) {
       k <- rbinom(1,n,p[j])
       ci <- wald(k,n,c)
       if (p[j] >= ci[1] & p[j] <= ci[2]) {count = count + 1}
     }
     cov[j] <- count/reps
     }
plot(p,cov,main = "Coverage Probability", type="l")
}
Coverage(0.95,50)
abline(h=0.95)
```

Section 12: R Commander

R Commander is a GUI (graphical user interface) for R and allows you to use R in a manner similar to Minitab, SPSS or other commercial statistical software packages. To function properly under Windows, R Commander requires the single-document interface (SDI) to R. To set this, use **Edit/GUI preferences** to open the Rgui Configuration Editor.



Rgui Configuration Editor

Single or multiple windows: MDI [✓ SDI] ✓ MDI toolbar ☐ MDI statusbar

Pager style: [✓ multiple windows] single window

Language for menus and messages:

Font: Courier New ✓ TrueType only size: 10 style: normal

Console rows: 25 columns: 80 Initial left: 533 top: 211

✓ set options(width) on resize?

✓ buffer console by default?

buffer chars: 250000 lines: 8000

Pager rows: 25 columns: 80

Graphics windows: initial left: -25 top: 0

Console and Pager Colours

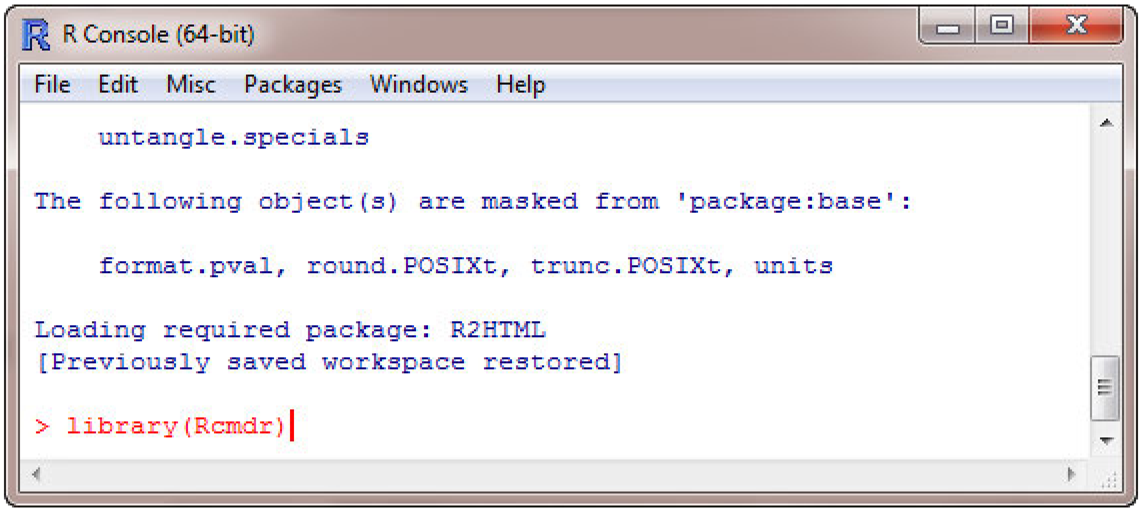
background, normaltext, usertext, pagerbg — selected: background

wheat2, wheat3, wheat4, white — selected: white

```
Sample text
```

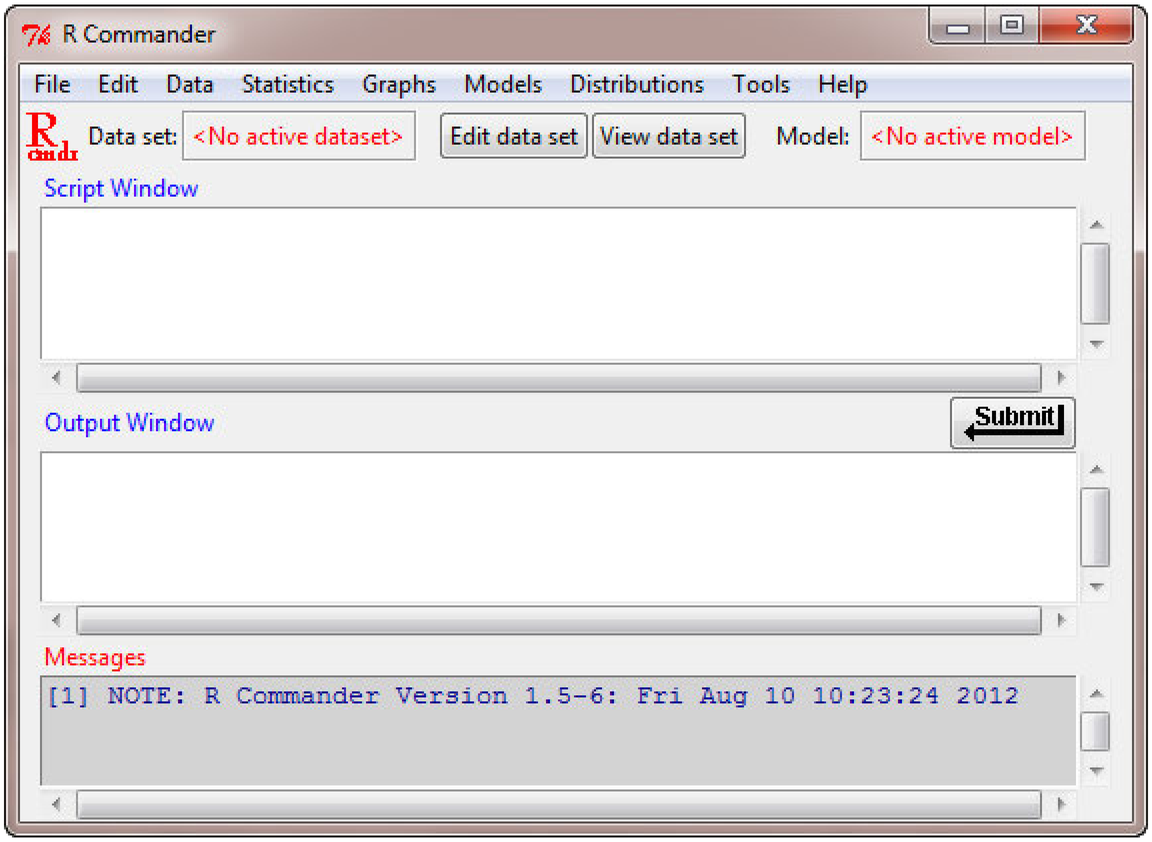
Select **SDI** and then **Save** to save the file RConsole to the etc folder inside the R folder. On my computer the path was **C:\Program Files\R\R-2.11.1-x64\etc**. Then click **Cancel** and close R. When you reopen R, it will open in SDI mode. To check this go to **Edit/GUI preferences** and make sure SDI is selected.

Use the R menu Packages/Install Package(s)… to install the **Rcmdr** package. Then use Packages/Load Package… to load the **Rcmdr** package.



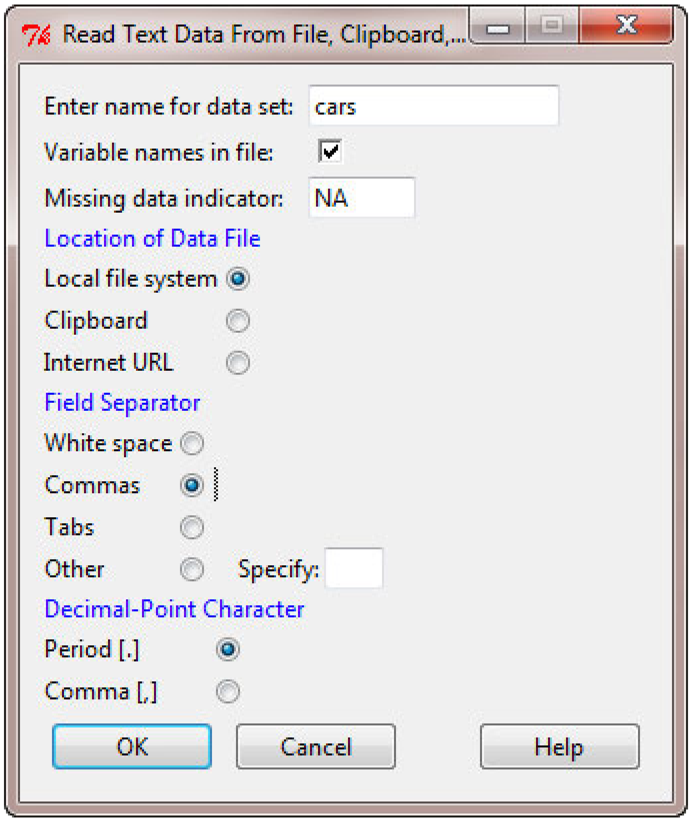
```
    untangle.specials

The following object(s) are masked from 'package:base':

    format.pval, round.POSIXt, trunc.POSIXt, units

Loading required package: R2HTML
[Previously saved workspace restored]

> library(Rcmdr)
```

This will open R Commander.



```
[1] NOTE: R Commander Version 1.5-6: Fri Aug 10 10:23:24 2012
```

As you can see R Commander has menus and functions much like the commercial statistical software packages. In keeping with the command line spirit of R, it will show you the commands which are invoked by making your menu selections. When loading data files into R Commander, it is probably best to use **.csv** files (see Section 5 for a discussion of this file type). I have saved the cars data set as a .csv file. To load it into R Commander, use **Data/Import Data/from text file, clipboard or URL…** Then enter the name of the data set, set the Field Separator to Commas and press OK.


Read Text Data From File, Clipboard,...

Enter name for data set: cars

Variable names in file: [✓]

Missing data indicator: NA

Location of Data File

Local file system [✓] Clipboard Internet URL

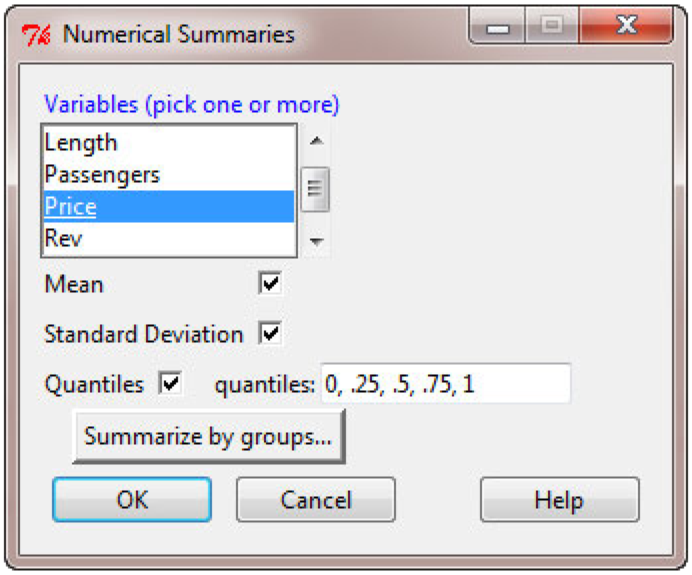
Field Separator

White space [✓ Commas] Tabs Other Specify:

Decimal-Point Character

[✓ Period [.]] Comma [,]

Use the open file dialog box to locate the .csv file on your computer and press Open. The cars data set will now be the active data set. To compute a numerical summary for the prices of cars, use **Statistics/Summaries/Numerical summaries** to open the Numerical summaries dialog box. Select the Price variable and press OK.

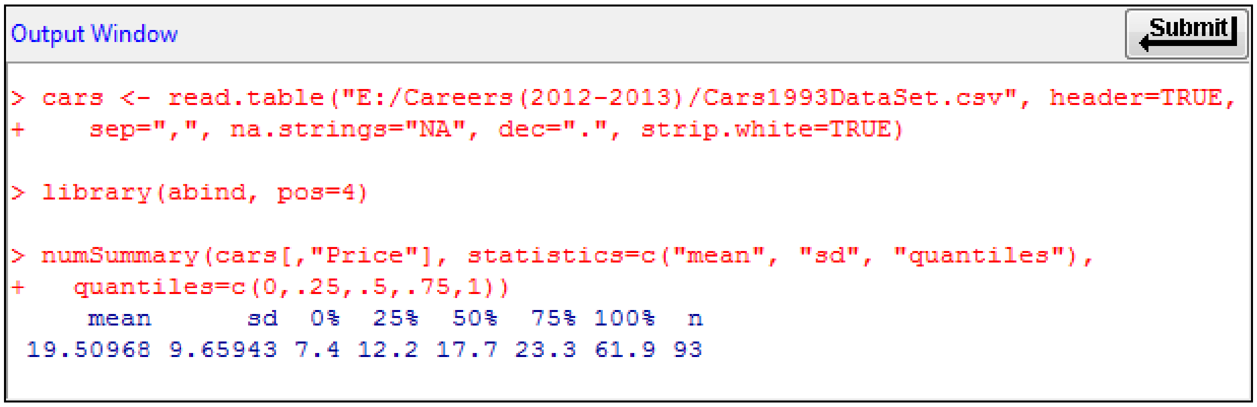


Mean ☑

Standard Deviation ☑

Quantiles ☑ quantiles: 0, .25, .5, .75, 1

A numerical summary for the Price variable will appear in the Output window along with the R code used to produce the summary.



```
> cars <- read.table("E:/Careers(2012-2013)/Cars1993DataSet.csv", header=TRUE,
+    sep=",", na.strings="NA", dec=".", strip.white=TRUE)

> library(abind, pos=4)

> numSummary(cars[,"Price"], statistics=c("mean", "sd", "quantiles"),
+   quantiles=c(0,.25,.5,.75,1))
     mean       sd  0%  25%  50%  75% 100%  n
 19.50968 9.65943 7.4 12.2 17.7 23.3 61.9 93
```