INFO I-415
Lab 1

NB: Part of lab 2 is built on lab 1. You may use the same file to continue working on lab 2.

Indexing Data

We often wish to examine part of a set of data. Suppose that our data is stored in the matrix `A`.

```
> A <- matrix(1:16, 4, 4)
> A
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
```

Then, typing

```
> A[2, 3]
[1] 10
```

will select the element corresponding to the second row and the third column. The first number after the open-bracket symbol `[` always refers to the row, and the second number always refers to the column. We can also select multiple rows and columns at a time, by providing vectors as the indices.

```
> A[c(1, 3), c(2, 4)]
     [,1] [,2]
[1,]    5   13
[2,]    7   15
> A[1:3, 2:4]
     [,1] [,2] [,3]
[1,]    5    9   13
[2,]    6   10   14
[3,]    7   11   15
> A[1:2, ]
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
```

```
> A[, 1:2]
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
```

The last two examples include either no index for the columns or no index for the rows. These indicate that R should include all columns or all rows, respectively. R treats a single row or column of a matrix as a vector.

```
> A[1, ]
[1]  1  5  9 13
```

The use of a negative sign - in the index tells R to keep all rows or columns except those indicated in the index.

```
> A[-c(1, 3), ]
     [,1] [,2] [,3] [,4]
[1,]    2    6   10   14
[2,]    4    8   12   16
> A[-c(1, 3), -c(1, 3, 4)]
[1] 6 8
```

The dim() function outputs the number of rows followed by the number of columns of a given matrix.

```
> dim(A)
[1] 4 4
```

Loading Data

For most analyses, the first step involves importing a data set into R. The read.table() function is one of the primary ways to do this. The help file contains details about how to use this function. We can use the function write.table() to export data.

Before attempting to load a data set, we must make sure that R knows to search for the data in the proper directory. For example, on a Windows system one could select the directory using the Change dir... option under the File menu. However, the details of how to do this depend on the operating system (e.g. Windows, Mac, Unix) that is being used, and so we do not give further details here.

We begin by loading in the Auto data set. This data is part of the ISLR2 library, discussed in Chapter 3. To illustrate the read.table() function, we load it now from a text file, Auto.data, which you can find on the textbook website. The following command will load the Auto.data file into R and store it as an object called Auto, in a format referred to as a *data frame*. Once the data has been loaded, the View() function can be used to view it in a spreadsheet-like window.[1] The head() function can also be used to view the first few rows of the data.

```
> Auto <- read.table("Auto.data")
> View(Auto)
> head(Auto)
    V1         V2           V3          V4     V5
1  mpg cylinders displacement horsepower weight
2 18.0         8        307.0      130.0  3504.
3 15.0         8        350.0      165.0  3693.
4 18.0         8        318.0      150.0  3436.
5 16.0         8        304.0      150.0  3433.
6 17.0         8        302.0      140.0  3449.
            V6   V7     V8                          V9
1 acceleration year origin                        name
2         12.0   70      1 chevrolet chevelle malibu
3         11.5   70      1          buick skylark 320
4         11.0   70      1         plymouth satellite
5         12.0   70      1              amc rebel sst
6         10.5   70      1                ford torino
```

Note that Auto.data is simply a text file, which you could alternatively open on your computer using a standard text editor. It is often a good idea to view a data set using a text editor or other software such as Excel before loading it into R.

This particular data set has not been loaded correctly, because R has assumed that the variable names are part of the data and so has included them in the first row. The data set also includes a number of missing observations, indicated by a question mark ?. Missing values are a common occurrence in real data sets. Using the option `header = T` (or `header = TRUE`) in the `read.table()` function tells R that the first line of the file contains the variable names, and using the option `na.strings` tells R that any time it sees a particular character or set of characters (such as a question mark), it should be treated as a missing element of the data matrix.

```
> Auto <- read.table("Auto.data", header = T, na.strings = "?",
     stringsAsFactors = T)
> View(Auto)
```

The `stringsAsFactors = T` argument tells R that any variable containing character strings should be interpreted as a qualitative variable, and that each distinct character string represents a distinct level for that qualitative variable. An easy way to load data from Excel into R is to save it as a csv (comma-separated values) file, and then use the `read.csv()` function.

```
> Auto <- read.csv("Auto.csv", na.strings = "?",
    stringsAsFactors = T)
> View(Auto)
```

[1]This function can sometimes be a bit finicky. If you have trouble using it, then try the `head()` function instead.

```
> dim(Auto)
[1] 397 9
> Auto[1:4, ]
```

The `dim()` function tells us that the data has 397 observations, or rows, and nine variables, or columns. There are various ways to deal with the missing data. In this case, only five of the rows contain missing observations, and so we choose to use the `na.omit()` function to simply remove these rows.

```
> Auto <- na.omit(Auto)
> dim(Auto)
[1] 392   9
```

Once the data are loaded correctly, we can use `names()` to check the variable names.

```
> names(Auto)
[1] "mpg"          "cylinders"    "displacement" "horsepower"
[5] "weight"       "acceleration" "year"         "origin"
[9] "name"
```

Additional Graphical and Numerical Summaries

We can use the `plot()` function to produce *scatterplots* of the quantitative variables. However, simply typing the variable names will produce an error message, because `R` does not know to look in the `Auto` data set for those variables.

```
> plot(cylinders, mpg)
Error in plot(cylinders, mpg) : object 'cylinders' not found
```

To refer to a variable, we must type the data set and the variable name joined with a `$` symbol. Alternatively, we can use the `attach()` function in order to tell `R` to make the variables in this data frame available by name.

```
> plot(Auto$cylinders, Auto$mpg)
> attach(Auto)
> plot(cylinders, mpg)
```

The `cylinders` variable is stored as a numeric vector, so `R` has treated it as quantitative. However, since there are only a small number of possible values for `cylinders`, one may prefer to treat it as a qualitative variable. The `as.factor()` function converts quantitative variables into qualitative variables.

```
> cylinders <- as.factor(cylinders)
```

If the variable plotted on the x-axis is qualitative, then *boxplots* will automatically be produced by the `plot()` function. As usual, a number of options can be specified in order to customize the plots.

```
> plot(cylinders, mpg)
> plot(cylinders, mpg, col = "red")
> plot(cylinders, mpg, col = "red", varwidth = T)
```

```
> plot(cylinders, mpg, col = "red", varwidth = T,
    horizontal = T)
> plot(cylinders, mpg, col = "red", varwidth = T,
    xlab = "cylinders", ylab = "MPG")
```

The `hist()` function can be used to plot a *histogram*. Note that `col = 2` has the same effect as `col = "red"`.

```
> hist(mpg)
> hist(mpg, col = 2)
> hist(mpg, col = 2, breaks = 15)
```

The `pairs()` function creates a *scatterplot matrix*, i.e. a scatterplot for every pair of variables. We can also produce scatterplots for just a subset of the variables.

```
> pairs(Auto)
> pairs(
    ~ mpg + displacement + horsepower + weight + acceleration,
    data = Auto
  )
```

In conjunction with the `plot()` function, `identify()` provides a useful interactive method for identifying the value of a particular variable for points on a plot. We pass in three arguments to `identify()`: the x-axis variable, the y-axis variable, and the variable whose values we would like to see printed for each point. Then clicking one or more points in the plot and hitting Escape will cause `R` to print the values of the variable of interest. The numbers printed under the `identify()` function correspond to the rows for the selected points.

```
> plot(horsepower, mpg)
> identify(horsepower, mpg, name)
```

The `summary()` function produces a numerical summary of each variable in a particular data set.

```
> summary(Auto)
      mpg          cylinders      displacement
 Min.   : 9.00   Min.   :3.000   Min.   : 68.0
 1st Qu.:17.00   1st Qu.:4.000   1st Qu.:105.0
 Median :22.75   Median :4.000   Median :151.0
 Mean   :23.45   Mean   :5.472   Mean   :194.4
 3rd Qu.:29.00   3rd Qu.:8.000   3rd Qu.:275.8
 Max.   :46.60   Max.   :8.000   Max.   :455.0

   horsepower       weight      acceleration
 Min.   : 46.0   Min.   :1613   Min.   : 8.00
 1st Qu.: 75.0   1st Qu.:2225   1st Qu.:13.78
```

```
 Median : 93.5   Median :2804   Median :15.50
 Mean   :104.5   Mean   :2978   Mean   :15.54
 3rd Qu.:126.0   3rd Qu.:3615   3rd Qu.:17.02
 Max.   :230.0   Max.   :5140   Max.   :24.80

      year            origin                      name
 Min.   :70.00   Min.   :1.000   amc matador       :  5
 1st Qu.:73.00   1st Qu.:1.000   ford pinto        :  5
 Median :76.00   Median :1.000   toyota corolla    :  5
 Mean   :75.98   Mean   :1.577   amc gremlin       :  4
 3rd Qu.:79.00   3rd Qu.:2.000   amc hornet        :  4
 Max.   :82.00   Max.   :3.000   chevrolet chevette:  4
                                 (Other)           :365
```

For qualitative variables such as `name`, `R` will list the number of observations that fall in each category. We can also produce a summary of just a single variable.

```
> summary(mpg)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   9.00   17.00   22.75   23.45   29.00   46.60
```

Once we have finished using `R`, we type `q()` in order to shut it down, or quit. When exiting `R`, we have the option to save the current *workspace* so that all objects (such as data sets) that we have created in this `R` session will be available next time. Before exiting `R`, we may want to save a record of all of the commands that we typed in the most recent session; this can be accomplished using the `savehistory()` function. Next time we enter `R`, we can load that history using the `loadhistory()` function, if we wish.