Six Sigma with R - A Tutorial

An introduction to using R for Six Sigma

John McDonough
Six Sigma projects typically involve a significant amount of statistical analysis. Black Belts commonly use Minitab or JMP to perform these analyses. These products are fairly limited in capability and quite expensive. This document provides an overview of using the very capable open source application R to perform those analyses common in Six Sigma projects.
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>An Example R Session</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>Getting Data into R</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>A Simple Look at Data</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>Customizing Plots</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>Scripts and Functions in R</td>
<td>31</td>
</tr>
<tr>
<td>7</td>
<td>Control Charts</td>
<td>37</td>
</tr>
<tr>
<td>8</td>
<td>Process Capability</td>
<td>45</td>
</tr>
<tr>
<td>9</td>
<td>Hypothesis Testing</td>
<td>47</td>
</tr>
</tbody>
</table>

Preface

1. Document Conventions ................................................................. v
   1.1. Typographic Conventions ....................................................... v
   1.2. Pull-quote Conventions ....................................................... vi
   1.3. Notes and Warnings ............................................................... vii
2. We Need Feedback! ........................................................................ vii

2. An Example R Session

3. Getting Data into R
   3.1. Reading tabular data ............................................................ 5
   3.2. Reading delimited data ......................................................... 5
   3.3. Header Rows and Columns in Tabular Data ................................. 6
   3.4. Using Database Data ............................................................. 7

4. A Simple Look at Data
   4.1. Plotting .................................................................................. 10
   4.2. Simple Statistics ..................................................................... 15
   4.3. Checking for Normality ......................................................... 16

5. Customizing Plots
   5.1. Adjusting the Background Color ............................................. 19
   5.2. Changing the Color of Plot Elements ...................................... 20
   5.3. Annotating Graphs .................................................................... 21
   5.4. Changing the Plotting Character ............................................ 25
   5.5. Graphics Devices ..................................................................... 25
   5.6. Even More Customization ....................................................... 27

6. Scripts and Functions in R
   6.1. Using R in Parallel with an Editor ......................................... 31
   6.2. Scripts with R .......................................................................... 32
   6.3. Functions in R .......................................................................... 34

7. Control Charts
   7.1. The qcc package ....................................................................... 38
   7.2. XBar Charts ............................................................................ 38
   7.3. Tailoring qcc charts ............................................................... 40
   7.4. R Charts .................................................................................. 42
   7.5. Other types of control charts ................................................. 44
     7.5.1. The xbar.one chart ............................................................. 44
     7.5.2. The S chart ....................................................................... 44
     7.5.3. The p chart ....................................................................... 44
     7.5.4. The np chart ..................................................................... 44
     7.5.5. The c chart ....................................................................... 44
     7.5.6. The u chart ....................................................................... 44
   7.6. Autocorrelation ....................................................................... 44

8. Process Capability

9. Hypothesis Testing
10. Gage R&R 49
11. Comparing Groups 51
12. Pareto Charts 53
13. Ishikawa Diagrams 55
14. Regression Modeling 57
15. Logistic Regression 59
16. Experimental Design 61
17. Principal Component Analysis 63
18. Simulation 65
19. Conclusion 67
A. Revision History 69
Index 71
Preface

1. Document Conventions
This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the Liberation Fonts\(^1\) set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

1.1. Typographic Conventions
Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

Mono-spaced Bold

Used to highlight system input, including shell commands, file names and paths. Also used to highlight keycaps and key combinations. For example:

To see the contents of the file `my_next_bestselling_novel` in your current working directory, enter the `cat my_next_bestselling_novel` command at the shell prompt and press `Enter` to execute the command.

The above includes a file name, a shell command and a keycap, all presented in mono-spaced bold and all distinguishable thanks to context.

Key combinations can be distinguished from keycaps by the hyphen connecting each part of a key combination. For example:

Press `Enter` to execute the command.

Press `Ctrl+Alt+F2` to switch to the first virtual terminal. Press `Ctrl+Alt+F1` to return to your X-Windows session.

The first paragraph highlights the particular keycap to press. The second highlights two key combinations (each a set of three keycaps with each set pressed simultaneously).

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in mono-spaced bold. For example:

File-related classes include `filesystem` for file systems, `file` for files, and `dir` for directories. Each class has its own associated set of permissions.

Proportional Bold

\(^1\) https://fedorahosted.org/liberation-fonts/
This denotes words or phrases encountered on a system, including application names; dialog box text; labeled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

Choose **System → Preferences → Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).

To insert a special character into a **gedit** file, choose **Applications → Accessories → Character Map** from the main menu bar. Next, choose **Search → Find…** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit → Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in proportional bold and all distinguishable by context.

**Mono-spaced Bold Italic or Proportional Bold Italic**

Whether mono-spaced bold or proportional bold, the addition of italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

To connect to a remote machine using ssh, type **ssh username@domain.name** at a shell prompt. If the remote machine is **example.com** and your username on that machine is john, type **ssh john@example.com**.

The **mount -o remount file-system** command remounts the named file system. For example, to remount the **/home** file system, the command is **mount -o remount /home**.

To see the version of a currently installed package, use the **rpm -q package** command. It will return a result as follows: **package-version-release**.

Note the words in bold italics above — username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

Publican is a **DocBook** publishing system.

**1.2. Pull-quote Conventions**

Terminal output and source code listings are set off visually from the surrounding text.

Output sent to a terminal is set in **mono-spaced roman** and presented thus:
Source-code listings are also set in **mono-spaced** **roman** but add syntax highlighting as follows:

```java
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
{
    public static void main(String args[])
        throws Exception
    {
        InitialContext iniCtx = new InitialContext();
        Object         ref    = iniCtx.lookup("EchoBean");
        EchoHome       home   = (EchoHome) ref;
        Echo           echo   = home.create();

        System.out.println("Created Echo");
        System.out.println("Echo.echo('Hello') = "+ echo.echo("Hello"));
    }
}
```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

- **Note**

  Notes are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

- **Important**

  Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring a box labeled 'Important' will not cause data loss but may cause irritation and frustration.

- **Warning**

  Warnings should not be ignored. Ignoring warnings will most likely cause data loss.

2. We Need Feedback!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in Bugzilla: [http://bugzilla.redhat.com/bugzilla/](http://bugzilla.redhat.com/bugzilla/) against the product R.
When submitting a bug report, be sure to mention the manual's identifier: *Six_Sigma_with_R_-_A_Tutorial*

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.
Introduction

Six Sigma projects typically involve a significant amount of statistical analysis. Black Belts commonly use Minitab or JMP to perform these analyses. These products are fairly limited in capability and quite expensive. This document provides an overview of using the very capable open source application R to perform those analyses common in Six Sigma projects. R is a free and open source software tool. Often, individuals wishing to perform the types of analysis employed in Six Sigma are frustrated by the limitations of the tools at their disposal, and the cost of more capable tools. R avoids those frustrations.

R is an extremely versatile and powerful statistical tool. However, unlike the commercial desktop tools, R is a command-line application. The command line makes it easier to replicate analyses, make subtle changes to analyses in a controlled fashion, and use revision control tools to manage those analyses. There are a number of graphical user interfaces available for R, however this paper will not address them.

This paper does not attempt to teach the reader to become a Six Sigma Black Belt. Becoming an effective Six Sigma Black Belt requires far more than a little statistics, and should be achieved under the mentorship of an experienced Master Black Belt. Rather, this article discusses how common analyses used in Six Sigma may be performed using R.
An Example R Session

This section is essentially the same as the example in the R introduction shipped with R. It needs to be replaced with an example more germane to Black Belts.

In R, objects may be scalars, vectors or matrices. Since in statistics, vectors and matrices are far more common than scalars, scalars are actually matrices with only one element.

The `rnorm()` function will generate a vector of length n of normally distributed random numbers when invoked with n. If invoked with a vector, it will return a vector of the same length.

```r
> x <- rnorm(50)
> y <- rnorm(x)
```

Typing the name of an object will cause its value to be displayed.

```r
> x
[1] 1.243876970 0.547453690 -1.727531027 -0.436034735 0.065133143
   [6] 0.006555023 0.083376593 -0.244198944 0.552114137 -0.313153977
  [11] -1.121540146 0.270685664 -0.398695870 -0.317617995 -0.092834160
  [16] -1.324416196 -1.23351161 -0.398695870 0.381098559 -0.39624353
  [21] -0.989753419 1.933498742 0.852212695 -0.439848456 -0.787993018
```
A simple scatter plot of the data is easily generated with the `plot()` function:

```
> plot(x,y)
```

![Figure 2.1. Test Plot](image)

Although this plot is fairly crude, the `plot()` function has many options permitting great control over the generated plot.

The objects currently in memory can be viewed simply using the `ls()` function:

```
> ls()
[1] "x"  "y"
```

And similarly, objects no longer needed may be simply removed by means of the `rm()` function:

```
> rm(x, y)
```
Getting Data into R

The first order of business when preparing for any sort of analysis is collecting the data and getting it into a usable form. R can use data in many different forms. Data are commonly available in tabular form, separated either by spaces or tabs, or delimited by some character, typically a comma, from a spreadsheet or similar application.

3.1. Reading tabular data

Tabular data may be read into an array with the `read.table()` function. Suppose, for example, we have a file containing data in columns, each column separated by a number of spaces, such as the following:

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>200</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>301</td>
</tr>
<tr>
<td>3</td>
<td>31</td>
<td>399</td>
</tr>
</tbody>
</table>

Assuming the file has the name `Spaces.txt`, the file can be read into R as follows:

```r
> data <- read.table("Spaces.txt")
> data
   V1 V2  V3
1  1 10 100
2  2 20 200
3  3 30 301
4  4 31 399
```

The data are read into an array we have named `data` and as before, simply by typing the name of the object we can see its value.

Notice that the `<-` symbol is used as an assignment operator. In this case it means take the result from the `read.table()` function and assign it to the object `data`. Since `read.table()` results in an array, `data` is also an array.

The `=` symbol may also be used as an assignment operator, however, it is more easily confused with the equality operator, so it is generally preferable to choose the `<-` symbol.

Had the columns been separated by tabs rather than spaces the result would have been the same.

3.2. Reading delimited data

Data are often available separated by some special character, most commonly a comma. The `read.table()` function can read this data too, however, we must specify the delimiting symbol with the `sep` parameter.

Using the same data but this time formatted with commas:

```
1,10,100
2,20,200
3,30,301
4,31,399
```
Assuming the file has the name `Csv.txt`, the file can be read into R as follows:

```r
> data <- read.table("Csv.txt",sep=",")
> data
V1 V2  V3
1  1 10 100
2  2 20 200
3  3 30 301
4  4 31 399
>
```

Notice that the result is the same. Other applications might use a semicolon, slash or other character as a delimiter; with R we merely specify the delimiter to be able to use the data.

### 3.3. Header Rows and Columns in Tabular Data

By default, R assumes the input file contains only data, and it names each column with the creative names `V1`, `V2`, `V3`, etc. However, your data may contain a row of column names. In this case, you can use the `header` parameter on the `read.table()` function to notify R to treat the first row as column names.

For example, suppose you had the following data:

```
Moe Larry Curly
1 10 100
2 20 200
3 30 301
```

Then a command like:

```r
> myData <- read.table("SpacesH.txt",header=TRUE)
```

would read the data this time using names from the first row:

```r
> myData
     Moe Larry Curly
1  1   10   100
2  2   20   200
3  3   30   301
4  4   31   399
>
```

Similarly, the `row.names` parameter identifies the column which contains the row names. With a file like:

```
When        Moe     Larry   Curly
Today       1       10      100
```

Then a command like:

```r
> myData <- read.table("NamesR.txt",header=TRUE,row.names="When")
```

would read the data this time using names from the row specified:

```r
> myData
     Moe Larry Curly
Today  1   10   100
```
### 3.4. Using Database Data

When multiple people are working on a dataset, individual text data files can become unwieldy. Also, large amounts of data can be difficult to work with in text files. The answer, of course, is to use a database.

R supports connections to most of the popular databases. For each database there is a package that must be used to add the commands for the particular database to R's skill set. For our example, we will use the very popular MySQL database, but connections to other databases, and to ODBC data sources are very similar.

The first step is to include the library for the database:

```r
> library(RMySQL)
```

Next, a connection to the database must be established. This requires providing the database name and optionally the user, password and perhaps even host name:

```r
> con <- dbConnect(dbDriver("MySQL"), dbname = "myDB", 
+                   user="R", password = "mypass" )
```

Once the connection is established, a table may be read not unlike a tabular text file. Notice that we must provide not only the name of the table, but also the connection which identifies the database (we could have connections to multiple databases):

```r
> d1 <- dbReadTable(con,"table3")
> summary(d1)
```

```
  alpha            beta           gamma           delta
Min.   :  5.004 Min.   :  86.85 Min.   :  744.1 Min.   :3.013
1st Qu.:152.649 1st Qu.:1755.42 1st Qu.: 6191.1 1st Qu.:3.125
Median :292.579 Median :3081.68 Median :10907.9 Median :3.349
Mean   :306.351 Mean   :3210.23 Mean   :11417.2 Mean   :3.464
3rd Qu.:485.733 3rd Qu.:4902.08 3rd Qu.:17295.0 3rd Qu.:3.743
Max.   :685.431 Max.   :6962.24 Max.   :24607.4 Max.   :4.882
```
Of course, the database offers additional functionality. Key among the database capabilities is the ability to select a portion of a particular table for analysis. From R, a simple SQL query can be passed to the database:

```r
> d2 <- dbGetQuery(con, paste(
+          "SELECT alpha, beta FROM table3 WHERE gamma<10000"
+          ))
> summary(d2)
    alpha       beta
  Min.   :  5.004  Min.   :  86.85
  1st Qu.: 66.554  1st Qu.: 836.53
  Median :132.986  Median :1489.23
  Mean   :140.027  Mean   :1548.73
  3rd Qu.:214.739  3rd Qu.:2357.75
  Max.   :274.571  Max.   :2783.47
```

Notice that whenever we retrieve data from the database, the columns are named with the database field names, instead of the V1, V2, V3 names.
A Simple Look at Data

When first encountering a set of data, one first wants to get a “feel” for the data. R has a number of ways to do this. We have already seen that, when the dataset is small, an obvious thing to do is simply list the data:

```r
> a <- read.table("ColNames.txt",header=TRUE,row.names=1)
> a
Moe      Larry    Curly
Today     1       10       100
Yesterday 2       20       200
Tomorrow  3       30       301
Someday   4       31       399
```

R includes a number of sample datasets which can be accessed through the `data()` function. Typing `data()` with no arguments will display a list of the available datasets. We will use the `ChickWeight` dataset in the examples in this section.

When there is more data, the `summary()` function can give us a good overview:

```r
> data(ChickWeight)
> summary(ChickWeight)

    weight     Time      Chick     Diet
      Min. : 35.0  Min. : 0.00   13 : 12   1:220
    1st Qu.: 63.0  1st Qu.: 4.00   9 : 12   2:120
   Median :103.0  Median :10.00   20: 12   3:120
   Mean :121.8   Mean :10.72   10 : 12   4:118
   3rd Qu.:163.8  3rd Qu.:16.00 17 : 12
   Max. :373.0   Max. :21.00 (Other):506
```

Notice that the `weight` and `Time` columns are shown differently than the `Chick` and `Diet`. That is because the first two are numerical data, and the second two categorical. For the categorical columns R displays the number of observations for each category.
4.1. Plotting

R can easily give us plots, too. We can plot specific pairs of variables:

```r
> plot(ChickWeight$Time, ChickWeight$weight)
```

results in a simple scatter diagram:

![Scatter Diagram](image_url)

*Figure 4.1. Scatter Diagram*
If there are only a few columns in the dataset, specifying the entire array will produce a grid of scatter plots of each column versus every other:

```r
> plot(ChickWeight)
```

![Multiple Scatter Diagram](image)

*Figure 4.2. Multiple Scatter Diagram*
If the X axis is a categorical, rather than numerical, variable, \texttt{R} will produce a box plot of the data.

\begin{verbatim}
> plot(ChickWeight$Diet,ChickWeight$weight)
\end{verbatim}

yields:

![Box Plot]

\textbf{Figure 4.3. Box Plot}

There is also a \texttt{boxplot()} function with parameters specific to box plots.
It can also be interesting to see how the data is distributed. A frequency histogram is the obvious way to have a quick look:

```r
> hist(ChickWeight$weight)
```

gives a plot like:

![Histogram of ChickWeight$weight](image)

**Figure 4.4. Simple Frequency Histogram**
The histogram can be enhanced in a number of ways. The **breaks** parameter can adjust the number of bars, a **density()** line can be plotted with the **lines()** function, and **rug()** can show individual tic marks along the X axis for each observation.

```r
> hist(ChickWeight$weight, breaks=20)
> lines(density(ChickWeight$weight, bw="SJ"))
> rug(ChickWeight$weight)
```

(although in this particular case the density line is not especially useful.)

![Histogram of ChickWeight$weight](image)

*Figure 4.5. Customized Frequency Histogram*
It may also be useful to examine the cumulative distribution in some cases. The \texttt{ecdf()} (empirical cumulative distribution function) allows us to easily access that view:

```r
> plot(ecdf(ChickWeight$weight), do.points=FALSE, verticals=TRUE)
```

![Empirical Cumulative Distribution](image)

**Figure 4.6. Empirical Cumulative Distribution**

### 4.2. Simple Statistics

\texttt{R} of course provides functions for the more common statistics. These can be assigned to variables, or simply displayed. For example, the mean of the above data:

```r
> mean(ChickWeight$weight)
[1] 121.8183
```

Note that the last two columns are categories, rather than variables, so a message is produced if we attempt to get means for all columns at once:

```r
> mean(ChickWeight)
weight   Time  Chick  Diet
121.8183 10.7179        NA  NA
Warning messages:
1: In mean.default(X[[3L]], ...) :
   argument is not numeric or logical: returning NA
2: In mean.default(X[[4L]], ...) :
   argument is not numeric or logical: returning NA
```

Of course, \texttt{R} has a function for variance:
Chapter 4. A Simple Look at Data

> var(ChickWeight$weight)
[1] 5051.223
>

The `sd()` function computes the standard deviation:

> sd(ChickWeight)
 weight      Time     Chick     Diet
71.071960  6.758400 14.568795  1.162678
>

Skewness can be determined with the `cum3()` function which is part of the `boot` library.

> library(boot)
> cum3(ChickWeight$weight)
[1] 345997.7
>

4.3. Checking for Normality
A distribution-comparison plot gives a quick feel for how close to normal the data are. The values are plotted against expected values were the data normal. A straight line is generally added to the points showing the normal expectation.
This plot is actually two plots, one over the other. The first plots the data, and the second plots the expected normal line:

```r
> qqnorm(ChickWeight$weight)
> qqline(ChickWeight$weight)
```

resulting in:

![Normal Q-Q Plot](image)

**Figure 4.7. Distribution-Comparison Plot**

A more formal test for normality is the Shapiro-Wilk test:

```r
> shapiro.test(ChickWeight$weight)

Shapiro-Wilk normality test

data:  ChickWeight$weight
W = 0.9087, p-value < 2.2e-16
```

Setting up a Kolmogorov-Smirnov test is slightly more involved (and it turns out it doesn't work well for this data):

```r
> ks.test(ChickWeight$weight, "pnorm",
+         mean=mean(ChickWeight$weight),
+         sd=sqrt(var(ChickWeight$weight))
+ )

One-sample Kolmogorov-Smirnov test

data:  ChickWeight$weight
D = 0.1202, p-value = 1.108e-07
alternative hypothesis: two-sided
```
Warning message:
In ks.test(ChickWeight$weight, "pnorm", mean = mean(ChickWeight$weight), :
cannot compute correct p-values with ties
>
Customizing Plots

Six Sigma is all about making the data, and the decisions, visible. Visible usually means graphical, and it is often important to have visually appealing graphics. So far, all the plots we have examined have been dull, black on white plots. While these may be fine for academic papers, they typically aren't the kinds of plots we would like for a Six Sigma storyboard.

However, R provides an almost endless variety in the way plots may be customized. In this section some of the more common customizations are examined.

5.1. Adjusting the Background Color

Most of the plot customizations may be applied directly on the `plot()`, `hist()`, `qqnorm()` or other plotting function. However, the background color can only be adjusted with the `par()` function. (There is a background color parameter on the `plot()` function, but it affects the background color of the symbols, rather than the plot as a whole.)

The `par()` function sets “permanent” plotting parameters. Most of the plot customizations may be applied in the `par()` function and they will apply to all subsequent plots. This includes the `bg` parameter, which is unique in that it cannot be applied to the plotting functions.

The following sequence:

```r
> data(ChickWeight)
> par(bg="aliceblue")
> plot(ChickWeight$Time,ChickWeight$weight)
```

will yeild the following image:

![Figure 5.1. Scatter Plot](image-url)
5.2. Changing the Color of Plot Elements

The color may be adjusted for any part of the plot. For example, if we wanted the plotted points to be a different color:

```
> plot(ChickWeight$Time, ChickWeight$weight,
+      col="steelblue")
```

![Figure 5.2. Scatter Plot](image)
The color of the labels ay also be changed:

```r
> plot(ChickWeight$Time, ChickWeight$weight,
+       col="steelblue", col.lab="cadetblue")
```

Figure 5.3. Scatter Plot

5.3. Annotating Graphs

Graphs have annotation on the axes, may have titles and other text additions. R provides parameters for managing those various annotations.
By default, the variable names are displayed along the axes. In some cases, as in the plots above, the result is not very satisfying. We can provide our own text with the `xlab` and `ylab` parameters, and of course the color may be adjusted as well.

```r
> plot(ChickWeight$Time, ChickWeight$weight,
+      col="steelblue", col.lab="cadetblue",
+      xlab="Time", ylab="Chick Weight"
+      )
```

Colors may be described as standard color names, or as in web colors, may be a # sign followed by six hex digits, as in `#5f9ea0` (the code for Cadet Blue).
A title may also be assigned to the graph:

```r
> plot(ChickWeight$Time, ChickWeight$weight,
+      col="steelblue", col.lab="cadetblue",
+      xlab="Time", ylab="Chick Weight",
+      main="Chick Diet Study", col.main="navy"
+      )
```

**Figure 5.5. Scatter Plot**
A subtitle may also be placed at the bottom of the graph:

```r
> plot(ChickWeight$Time, ChickWeight$weight,
+     col="steelblue", col.lab="cadetblue",
+     xlab="Time", ylab="Chick Weight",
+     main="Chick Diet Study", col.main="navy",
+     sub="Chick weight vs. time with different diets", col.sub="lightsteelblue"
+     )
```

Figure 5.6. Scatter Plot
5.4. Changing the Plotting Character

By default, points are shown as small circles (actually, only true on most devices). However, one may want to select a different character for many reasons. The \texttt{pch} parameter (plot character) controls this.

```r
> plot(ChickWeight$Time, ChickWeight$weight,
+ col="steelblue", col.lab="cadetblue",
+ xlab="Time", ylab="Chick Weight",
+ main="Chick Diet Study", col.main="navy",
+ sub="Chick weight vs. time with different diets", col.sub="lightsteelblue",
+ pch=19
+ )
```

Figure 5.7. Scatter Plot

The plotting symbol selected may be in the range of zero to 127. Symbols from zero to 18 are compatible with the older \texttt{S} language. Symbols 19 to 25 are newer symbols, and may be filled with a background color when \texttt{bg=} is specified. Symbols 32 through 127 are displayed as their ASCII equivalents, and may also be represented as a quoted letter (e.g. \texttt{pch="x"}).

Figure 5.8. Plotting Symbols

5.5. Graphics Devices

When \texttt{R} generates a plot, it writes on some logical "device". There are a number of available devices, and the user may use a variety of these devices in a single session.

By default, the plotted image is drawn on the \texttt{X11()} device (\texttt{window()} on Windows). The user may open a new plotting space, leaving the previous plot open, merely by typing \texttt{X11()}. 


However, there are plenty of options for the new workspace. Most commonly specified are the `width` and `height` parameters. The command:

```r
> X11(width=5,height=4)
```

will open a new workspace 5 inches wide and 4 inches tall. The user may also specify the background and canvas colors, the display on which to place the workspace (assuming the user has multiple logical displays attached), the default font size and font, the window position and title, etc.

However, `X11()` isn't the only device available. Most of the other devices write to files. Some devices of interest are:

- `bmp()`
- `jpeg()`
- `pdf()`
- `png()`
- `postscript()`
- `tiff()`
Sizes for \texttt{bmp()}, \texttt{jpeg()}, \texttt{png()} and \texttt{tiff()} are specified in pixels, while \texttt{pdf()} and \texttt{postscript()} are specified in inches. \texttt{pdf()} and \texttt{postscript()} also have parameters for positioning the plot on standard sized pages, which the other devices lack.

The devices which write to a file each have a default filename, but the user may specify a \texttt{file=} parameter. If the first parameter is not named it is assumed to be a filename:

\begin{verbatim}
> png("myPlot.png",width=640,height=480)
\end{verbatim}

\section*{5.6. Even More Customization}

Additional elements may also be added to the graph. We saw earlier with the \texttt{qqline()} how an additional line could be added to the plot. That was a specialized case, but the \texttt{lines()} and \texttt{points()} functions allow additional lines or points to be added to an existing plot. Of course, plotting characters, colors, etc. may be applied.

Often it is desirable to plot multiple variables on the same plot, and display the variables in different colors. The \texttt{plot()} function can generate the plot window, scales, etc., possibly with one variable, then the \texttt{lines()} or \texttt{points()} function may be used for additional variables.
Consider the following commands (a different internal dataset is used in this case):

```r
data(USArrests)
plot(USArrests$UrbanPop, USArrests$Rape, col="red", pch=16,
     ylab="Rapes (red), Murders (blue) per 100K",
     xlab="Percent Urban Population",
     main="Arrests versus Urban Population",
     col.main="steelblue")
points(USArrests$UrbanPop, USArrests$Murder, col="blue", pch=15)
```

The first call to `plot()` sets up the axes, labels, etc. and provides the first set of data points, while the call to `points()` adds the second set:

![Arrests versus Urban Population](image)

**Figure 5.10. Different Color Data**

At times it can be helpful to generate the plot frame with axes, labels, etc., with no data. In the previous example, we needed to use the `plot()` function for rapes first, because the number of arrests for rape was higher than for murders. Had we plotted murders first, many of the rape points would have fallen off the chart.
Also, we may frequently want to repeatedly generate plots of data across a consistent scale, or, as in the above case, may want to manage the scale manually. By creating short vectors with the minimum and maximum for each scale, and plotting them with a `type="n"`, an empty plot will result:

```r
x=c(0,100)
y=c(0,10)
plot(x,y,type="n")
```

![Figure 5.11. Empty Plot Frame](image)

---

**Draft**

Even More Customization

---

29
Scripts and Functions in R

The command line nature of R lends itself to scripting and customizations through scripts. This chapter highlights some of the kinds of things that may be accomplished with scripts.

6.1. Using R in Parallel with an Editor

One of the simplest, and most convenient, ways to use R interactively is to have an editor open while performing an analysis. Many popular editors and IDEs (emacs, gedit, and geany to name a few) recognize R commands and can provide syntax highlighting which can reduce errors.

Figure 6.1. Using emacs with R
With both windows open, the user may cut and paste between them. Commands may be recalled for editing with the up arrow in R, but this can get unwieldy if there are multiple lines or commands involved. While the above figure shows emacs, many other popular editors have similar features.

![Figure 6.2. gedit](image)

### 6.2. Scripts with R

Often there are sequences of command we execute frequently; perhaps we grab some data from a database, perhaps we look at a particular collection of statistics from whatever data we are evaluating, perhaps we like our plots to look some particular way. We can put those commands into a file and then call that file into R with the `source()` function. This can be a great time saver, and can also assure that we are consistent in how we perform certain analyses.

Imagine a case where data are being routinely entered into a database, perhaps by manufacturing operators, or perhaps automatically by a process. We may wish to periodically review that data, for example each morning.

In that case, we need to connect to the database, extract some data from the database, perhaps view some scatter plots, and then depending on what we see perhaps dive deeper into the data. In a case like this, we might begin by executing the following sequence of commands:

```r
library(RMySQL)
con <- dbConnect(dbDriver("MySQL"), dbname = "myDB",
    user="R", password = "mypass", host="cimbaoth" )
d1 <- dbReadTable(con,"table2")
d4 <- dbGetQuery(con,paste("SELECT alpha, beta FROM table3 WHERE gamma<10000")
    )
X11(width=3, height=3,
    xop=30, ypos=50)
plot(d1$alpha,d1$beta,
    main="Alpha-Beta",
    xlab="Alpha",
    ylab="Beta",
    pch=24,
)```

```
This might get tedious to do every morning. However, if we put those commands into a file, they may be recalled with the `source()` function. In this case notice that we have opened two plot windows so we can view the scatter diagrams. Since we are only interested in an overview, we have specified smaller plot windows, and we have adjusted the position of those windows so we don’t need to move windows around to view both plots.

If the name of the file was `getData.R`, the result might look something like the following:

![Figure 6.3. Executing a script](image)

Notice that when the script has completed the data is available for further analysis.

A script isn’t only useful for initialization. Consider a situation where we would like to look at plots in some particular way, perhaps over a variety of data sets. If we put our plot command in a script with our defaults, and perhaps some easily remembered names for the variables, we could re-use all that typing.
Chapter 6. Scripts and Functions in R

If we had a script like the following

```r
X11(height=4.0,width=7.0)
par(bg="lemonchiffon")
plot(x,y,
    col="darkred",
    fg="peru",
    bg="darkgoldenrod1",
    col.axis="saddlebrown",
    type="b",
    pch=23,
    cex=2,
    lwd=2
)
```

then we could assign `x` and `y` to the variables we are currently examining, and easily see the plot how we would like.

![Figure 6.4. Plotting with a script](image)

### 6.3. Functions in R

We have already examined dozens of functions that are included in R, but it is also possible to create custom functions. Adding customized functions to the repertoire is arguably the most useful feature of scripts.
For those that aren't familiar with programming languages such as Python or C, creating your own function may seem a bit daunting at first. But actually, it is quite straightforward.

First, you give your function a name, and then you assign (with the <- assignment operator) it to the function function(). The function() declaration is then followed by the statements you wish to execute, surrounded by curly-braces. Something like the following:

```r
myFunction <- function( x )
{
  plot(x)
}
```

The parentheses following function list parameters you wish to pass into the function. The indentation isn't required, it simply makes it a little easier to read.

You may then call the function like any other R function:

```r
myFunction(ChickWeight$weight)
```

You may pass a number of parameters, separated by commas. For example, we might extend the above function to show a title.

```r
myFunction <- function( x, title )
{
  plot(x,main=title)
}
```

You may also add a default value to your parameters. For example, if you wished to show a default title for your plot:

```r
myFunction <- function( x, title="My Plot" )
{
  plot(x,main=title)
}
```

the calling the function like:

```r
myFunction(ChickWeight$weight)
```

would yield a plot with the title "My Plot", but you have the option of providing your own title with something like:

```r
myFunction(ChickWeight$weight,title="Chick Weights")
```

Of course, once the function has been defined, we might use it a number of times. And, we would probably be a little more inclined to tweak our plot (or whatever) a little more, since we only have to type once to use the features many times.
Consider the following file, **SimpleFun.R**:

```r
myplot <- function( v, title=NULL )
{
  par(bg="#4a0062")
  plot(v, main=title,col="white",col.main="khaki",pch=15,
       col.lab="hotpink",col.axis="orchid",fg="purple",
       bty="L",type="b")
}
```

The file can be included with `source()` once, and then the function `myplot()` called many times:

![Figure 6.5. Using a simple custom function](image)

Figure 6.5. Using a simple custom function
Control Charts

During the Measure phase, one of the first things the Back Belt wants to do is to determine whether the process is in control with respect to the major 'Y'. The primary tool for this is a control chart. In many cases, the process may already keep control charts; many do. But there are large number of way in which control charts are produced, and a great many pitfalls, so the Black Belt would be well advised to examine the procedures used for the control chart and ensure they are appropriate for his purposes.

The simplest control chart consists of a simple plot of the observed variable versus time, with the control limits marked on the chart, and sometimes, the specification limit.

The control limits are typically set at +/- three standard deviations. It is important to remember that the control limits should not be recalculated each time the control chart is redrawn. Rather, they should be set once, and then changed because of a change in the process.

Presuming we have a series of observations, v, with v$V1 representing some observation number, time, etc., and v$V2 the value of the observation, the control limits may be easily calculated.

```r
> lcl <- mean(v$V2) - 3.0 * sd(v$V2)
> ucl <- mean(v$V2) + 3.0 * sd(v$V2)
```

Because there are a number of lines on the control chart, generating the chart involves a number of steps. First, we will generate the plot frame. We would like the plot frame to cover all the data, and allow the control limit lines to be displayed. This value could be set at four standard deviations which will almost always cover the data, or set manually. Depending on the use, the Black Belt may also prefer to set the height of the plot frame manually, keeping it consistent so that operators are accustomed to looking at the same chart.

```r
> ymin <- mean(v$V2) - 4.0 * sd(v$V2)
> ymax <- mean(v$V2) + 4.0 * sd(v$V2)
> xgr <- c(0,dim(v)[1])
> ygr <- c(ymin,ymax)
```

The X range is calculated based on the number of observations in the data set. The plot frame may then be drawn, in this case, on a window that is selected to be wider than it is high:

```r
> X11(height=4,width=8)
plot(xgr,ygr,type="n",xlab="Observation",ylab="Density",
     main="Density Control Chart",col.main="darkblue")
```

The parameter type="n" causes the plot frame to be drawn and no points plotted.
Finally, we can draw the control limits, center line, and the actual observations on the chart:

```r
> lines(xgr,c(lcl,lcl),col="#bfbfcf",lty=2)
> lines(xgr,c(ucl,ucl),col="#bfbfcf",lty=2)

m <- (ucl + lcl) / 2.0
lines(xgr,c(m,m))
lines(v,col="firebrick",lwd=2)
```

Figure 7.1. Example Control Chart

### 7.1. The qcc package

R has many, many add-on packages available, some of which we have already mentioned. The `qcc` package contains many functions useful in Six Sigma, especially around control charts.

The `qcc` package provides:
- Shewhart charts; XBar, R, S, p, np, c, u
- Cusum charts
- EWMA charts
- Operating characteristic curves
- Process capability analysis
- Pareto charts
- Ishikawa diagrams

### 7.2. XBar Charts

Often, control charts are produced by taking several samples from the process and plotting the mean (XBar) and range (R) of the samples. The `qcc` package expects a data frame containing samples in the rows and the individual observations in the columns.
Frequently, however, all the observations are in one column with some sort of sample identifier in another column. \texttt{qcc} provides a function to deal with this; \texttt{qcc.groups()}. \texttt{qcc.groups()} accepts two vectors as input and returns a data frame properly formatted for \texttt{qcc}. The first vector contains the individual observations and the second, the sample identifier.

The dataset \texttt{pistonrings} included in \texttt{qcc} has three columns labeled diameter, sample and trial. There are 40 values for sample with five observations each. A Shewhart XBar chart may be drawn from that dataset as follows:

```r
library(qcc)
data(pistonrings)
attach(pistonrings)
Diameter <- qcc.groups(diameter,sample)
obj <- qcc(Diameter)
```

resulting in the following graph:

![xbar Chart for Diameter](image)

**Figure 7.2. Default XBar Chart**

Notice that points beyond the control limits are drawn in red. Points that participate in a run of data violating any of the seven Shewhart rules are marked in orange.
Chapter 7. Control Charts

7.3. Tailoring qcc charts

qcc() does not accept most of the common plot() parameters. Many parameters can be passed in through the par() function. The various qcc functions create plots in multiple steps, however, so not all the parameters have the expected effect. qcc provides a special object, qcc.options() for some of the customizations that are likely to be needed.

The parameters to the qcc() function allow some control over annotation, but mostly affect the calculation. Below are some of the more commonly used parameters. See help(qcc) for a complete list:

- **type** - Type of plot to produce: “xbar”, “R”, “S”, “xbar.one”, “p”, “np”, “c”, “u”, “g”.
- **center** - Value specifying the process target
- **std.dev** - Standard deviation to be used
- **limits** - A two-value vector specifying the control limits
- **data.name** - String specifying display name for the variable to be plotted. If not provided is taken from the object given as data.
- **labels** - A character vector of labels for each group
- **nsigmas** - The number of sigmas to use for calculating control limits
- **confidence.level** - An alternative way to specify control limits
- **title** - Title string for the plot
- **xlab, ylab** - X-axis, Y-axis label

Parameters to the par() function that are likely to be useful are:

- **fg** - Foreground color for the plot. This affects the lines, points, inner box and tic marks.
- **col** - Color - the same as fg, except that it does not affect the tic marks. If issued before fg, will be overridden.
- **col.axis** - Color to be used for the numbering along the axes
- **col.main** - Color to be used for the plot title
- **col.lab** - Color to be used for the annotation along the axes
- **lwd** - Line width
- **lty** - Line type. (0=blank, 1=solid (default), 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash) or as a character string (“blank”, “solid”, “dashed”, “dotted”, “dotdash”, “longdash”, “twodash”).
- **bty** - Box type. Note that a box is always drawn around the plot area using the selected lty, lwd and fg. This box will be drawn later using the col and lwd. For many selections, this second box will not be visible.
- **cex.axis, cex.lab, cex.main** - Multiplier to the character size of the axis numbers, labels, and plot title respectively.

qcc.options() can accept the the following parameters which affect the appearance of the plots:

- **bg.margin** - background color used to draw the margin of the charts.
- **bg.figure** - background color used to draw the figure of the charts.
- **beyond.limits$pch** - plotting character used to highlight points beyond control limits.
- **beyond.limits$col** - color used to highlight points beyond control limits.
- **cex** - character expansion used to draw plot annotations (labels, title, tickmarks, etc.).
- **cex.stats** - character expansion used to draw text at the bottom of control charts.
- **font.stats** - font used to draw text at the bottom of control charts.
- **violating.runs$pch** - plotting character used to highlight points violating runs.
- **violating.runs$col** - color used to highlight points violating runs.

And the following parameters affect the calculation:
• **exp.R.unscaled** - a vector specifying, for each sample size, the expected value of the relative range (i.e. \( R/\sigma \)) for a normal distribution. This appears as d2 on most tables containing factors for the construction of control charts.

• **se.R.unscaled** - a vector specifying, for each sample size, the standard error of the relative range (i.e. \( R/\sigma \)) for a normal distribution. This appears as d3 on most tables containing factors for the construction of control charts.

• **run.length** - the maximum value of a run before to signal a point as out of control.

The above parameters can provide considerable control over the appearance and calculations used in the control chart. By way of example, the following:

```r
library(qcc)
data(pistonrings)
attach(pistonrings)
Diameter <- qcc.groups(diameter,sample)
qcc.options(bg.figure="goldenrod", bg.margin="lightgoldenrod")
par( fg="red", col="darkred", col.axis="sienna" )
par( col.main="darkolivegreen", col.lab="saddlebrown" )
par( lwd=2, lty=3, bty="l" )
par(cex.axis=1.2, cex.lab=1.4, cex.main=2)
obj <- qcc(Diameter[16:40,,], limits=c(73.99,74.02), data.name="Ring Diameter",
xlab="Sample ID", ylab="Diameter")
```

produces the following control chart:

![xbar Chart for Ring Diameter](image)

**Figure 7.3. Customized Control Chart**
7.4. R Charts

XBar charts are often plotted alongside R charts, which plot the range of each sample, rather than the mean. In some cases, R charts are used alone. `qcc()` can display an R chart merely by adding `type="R"` to the parameters:

![R Chart for Diameter](image)

Number of groups = 40  
Center = 0.023425  
StdDev = 0.0107094  
LCL = 0  
UCL = 0.04953145  
Number beyond limits = 0  
Number violating runs = 0

Figure 7.4. R Chart
Often the experimenter wishes to periodically review the control charts, and typically wishes to see the XBar and R charts next to each other. As shown earlier, a little scripting to properly position the plots on the screen can make this a simple task. The following lines:

```r
library(qcc)
data(pistonrings)
attach(pistonrings)
Diameter <- qcc.groups(diameter,sample)
X11(width=8,height=3,xpos=200,ypos=420)
obj <- qcc(Diameter,type="R",add.stats=FALSE)
X11(width=8,height=4,xpos=200,ypos=40)
obj <- qcc(Diameter,type="xbar")
```

will yeild a pair of plots. Note that the specific values chosen depend on screen size and resolution so the analyst will need to experiment with the position and size of the windows.

![Figure 7.5. XBar and R Charts](image-url)
Chapter 7. Control Charts

7.5. Other types of control charts

qcc() can produce a number of other types of control charts for specialized circumstances. Like the R chart, these alternative control chart types are invoked by adding the `type` parameter to the qcc() invocation.

7.5.1. The xbar.one chart

In a continuous process it doesn't always make sense to group observations into samples. When `type="xbar.one"` is specified, qcc() expects a single vector of observations.

7.5.2. The S chart

Sometimes it is desirable to follow the sample variation by standard deviation rather than (or in addition to) the range. `type="S"` will create a chart of sample standard deviations.

7.5.3. The p chart

When `type="p"` is specified, the proportion of nonconforming units is plotted. The control limits are calculated based on the binomial distribution.

7.5.4. The np chart

When `type="np"` is specified, the number of nonconforming units is plotted. The control limits are calculated based on the binomial distribution.

7.5.5. The c chart

The number of defectives per unit are plotted for `type="c"`. This chart assumes that defects of the quality attribute are rare, and the control limits are computed based on the Poisson distribution.

7.5.6. The u chart

When `type="u"` is specified, the average number of defectives per unit is plotted. The Poisson distribution is used to compute control limits, but, unlike the c chart, this chart does not require a constant number of units.

7.6. Autocorrelation

7.7. EMWA Charts

7.8. Cusum charts
Process Capability
Hypothesis Testing
Gage R&R
Comparing Groups
Pareto Charts
Ishikawa Diagrams

Figure 13.1. Example Ishikawa Diagram
Regression Modeling
Logistic Regression
Experimental Design
Principal Component Analysis
Simulation
Conclusion
# Appendix A. Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Author</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Sat Jul 23 2011</td>
<td>John McDonough</td>
<td><a href="mailto:jjmcd@fedoraproject.org">jjmcd@fedoraproject.org</a></td>
</tr>
</tbody>
</table>
| 3        | Tue Jul 19 2011 | John McDonough | jjmcd@fedoraproject.org | Various typo corrections  
|          |             |                      | Additional content                        |
| 2        | Tue Jul 5 2011  | John McDonough | jjmcd@fedoraproject.org | Convert to book |
| 1        | Mon Jul 4 2011  | John McDonough | jjmcd@fedoraproject.org | Initial chapters |
| 0        | Sat Dec 25 2010 | John McDonough | jjmcd@fedoraproject.org | Initial creation of book by publican |
Index

B
Background Color, 19
bg, 19, 25
bmp(), 26
boot, 15
Box Plot, 12
breaks, 14

C
Cause and Effect
  Diagram, 55
ChickWeight, 9
col, 20
col.lab, 21, 22
col.main, 23
col.sub, 24
cum3(), 15
Cumulative Frequency, 15

D
Data
  Database, 7
  Delimited, 5
  Tabular, 5
data(), 9, 27
Dataset
  ChickWeight, 9
  USArrests, 27
dbConnect(), 7
dbGetQuery(), 7
dbReadTable(), 7
density(), 14
Distribution-Comparison Plot, 16

E
edcf(), 15

F
feedback
  contact information for this manual, vii
Fishbone
  Diagram, 55

H
height, 26
hist(), 13, 19

Histogram, 13

I
Ishikawa
  Diagram, 55

J
jpeg(), 26

K
Kolmogorov-Smirnov, 17

L
lines(), 14, 27

M
main, 23
Mean, 15
mean(), 15
MySQL, 7

N
Normality, 16

O
ODBC, 7

P
par(), 19
pch, 25, 25
pdf(), 26
plot(), 10, 19, 27
png(), 26
points(), 27
postscript(), 26

Q
qqline(), 16, 27
qqnorm(), 16, 19

R
read.table(), 5
RMySQL, 7
rug(), 14

S
S, 25
Scatter Diagram, 10
<table>
<thead>
<tr>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>sd(), 15</td>
</tr>
<tr>
<td>Shapiro-Wilk, 17</td>
</tr>
<tr>
<td>Skewness, 15</td>
</tr>
<tr>
<td>Standard Deviation, 15</td>
</tr>
<tr>
<td>sub, 24</td>
</tr>
<tr>
<td>summary(), 7, 9</td>
</tr>
<tr>
<td><strong>T</strong></td>
</tr>
<tr>
<td>tiff(), 26</td>
</tr>
<tr>
<td>type, 28</td>
</tr>
<tr>
<td><strong>V</strong></td>
</tr>
<tr>
<td>var(), 15</td>
</tr>
<tr>
<td>Variance, 15</td>
</tr>
<tr>
<td><strong>W</strong></td>
</tr>
<tr>
<td>width, 26</td>
</tr>
<tr>
<td><strong>X</strong></td>
</tr>
<tr>
<td>X11(), 25</td>
</tr>
<tr>
<td>xlab, 22</td>
</tr>
<tr>
<td><strong>Y</strong></td>
</tr>
<tr>
<td>ylab, 22</td>
</tr>
</tbody>
</table>