# Chapter 3

# Data Exploration

This chapter shows examples on data exploration with R. It starts with inspecting the dimensionality, structure and data of an R object, followed by basic statistics and various charts like pie charts and histograms. Exploration of multiple variables are then demonstrated, including grouped distribution, grouped boxplots, scattered plot and pairs plot. After that, examples are given on level plot, contour plot and 3D plot. It also shows how to saving charts into files of various formats.

## 3.1 Have a Look at Data

The `iris` data is used in this chapter for demonstration of data exploration with R. See Section 1.3.1 for details of the `iris` data.

We first check the size and structure of data. The dimension and names of data can be obtained respectively with `dim()` and `names()`. Functions `str()` and `attributes()` return the structure and attributes of data.

```
> dim(iris)

[1] 150   5

> names(iris)

[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"

> str(iris)

'data.frame':        150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...

> attributes(iris)

$names
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"

$row.names
  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19
 [20]  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38
```

```
 [39]  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57
 [58]  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76
 [77]  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95
 [96]  96  97  98  99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114
[115] 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133
[134] 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150

$class
[1] "data.frame"
```

Next, we have a look at the first five rows of data. The first or last rows of data can be retrieved with `head()` or `tail()`.

```
> iris[1:5,]

  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa

> head(iris)

  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa

> tail(iris)

    Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
145          6.7         3.3          5.7         2.5 virginica
146          6.7         3.0          5.2         2.3 virginica
147          6.3         2.5          5.0         1.9 virginica
148          6.5         3.0          5.2         2.0 virginica
149          6.2         3.4          5.4         2.3 virginica
150          5.9         3.0          5.1         1.8 virginica
```

We can also retrieve the values of a single column. For example, the first 10 values of `Sepal.Length` can be fetched with either of the codes below.

```
> iris[1:10, "Sepal.Length"]

 [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9

> iris$Sepal.Length[1:10]

 [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9
```

## 3.2 Explore Individual Variables

Distribution of every numeric variable can be checked with function `summary()`, which returns the minimum, maximum, mean, median, and the first (25%) and third (75%) quartiles. For factors (or categorical variables), it shows the frequency of every level.

```
> summary(iris)
```

```
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

The mean, median and range can also be obtained with functions with `mean()`, `median()` and `range()`. Quartiles and percentiles are supported by function `quantile()` as below.

```
> quantile(iris$Sepal.Length)
```

```
  0%  25%  50%  75% 100%
 4.3  5.1  5.8  6.4  7.9
```

```
> quantile(iris$Sepal.Length, c(.1, .3, .65))
```

```
 10%  30%  65%
4.80 5.27 6.20
```

Then we check the variance of `Sepal.Length` with `var()`, and also check its distribution with histogram and density using functions `hist()` and `density()`.

```
> var(iris$Sepal.Length)
[1] 0.6856935
> hist(iris$Sepal.Length)
```

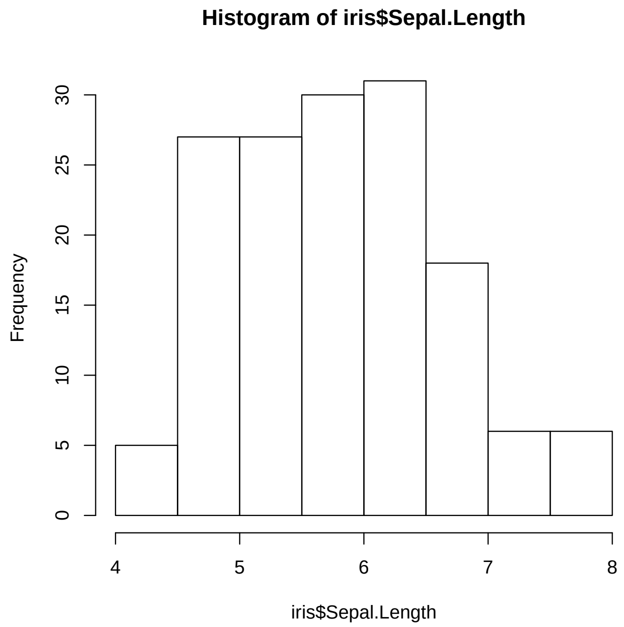


Figure 3.1: Histogram

```
> plot(density(iris$Sepal.Length))
```

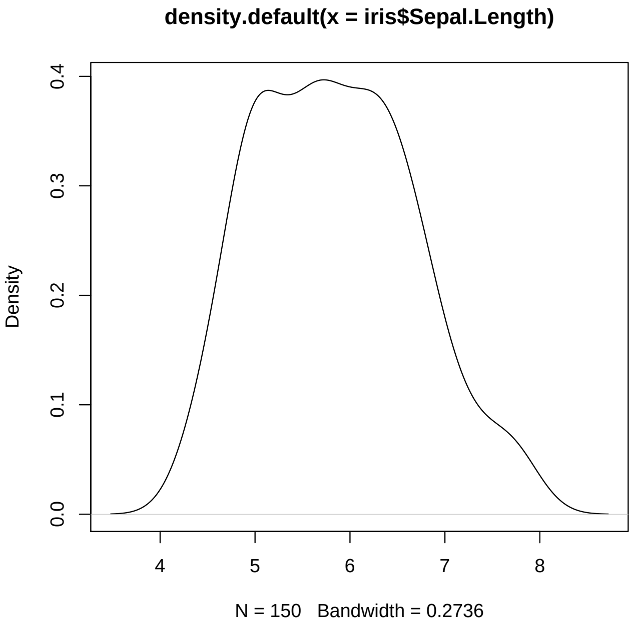


Figure 3.2: Density

The frequency of factors can be calculated with function `table()`, and then plotted as a pie chart with `pie()` or a bar chart with `barplot()`.

```
> table(iris$Species)

    setosa versicolor  virginica
        50         50         50

> pie(table(iris$Species))
```

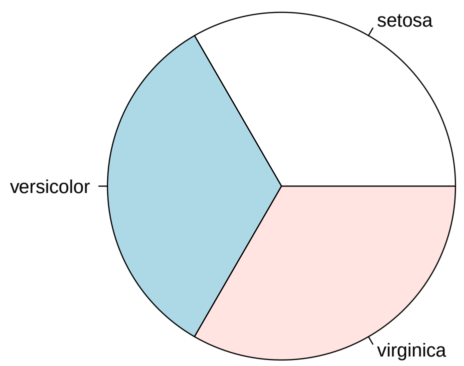


Figure 3.3: Pie Chart

```
> barplot(table(iris$Species))
```

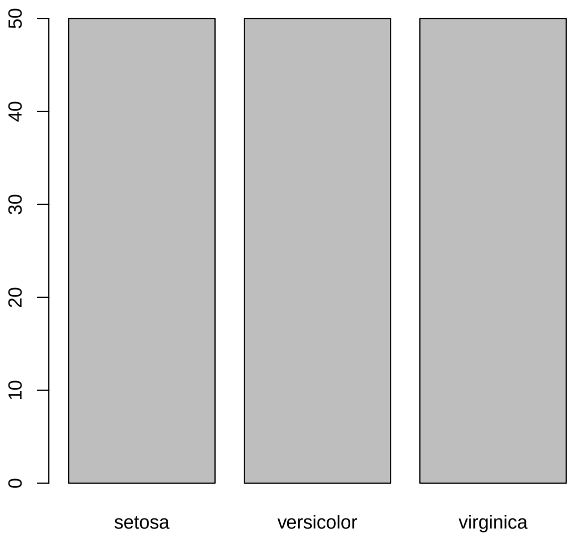


Figure 3.4: Bar Chart

## 3.3 Explore Multiple Variables

After checking the distributions of individual variables, we then investigate the relationships between two variables. Below we calculate covariance and correlation between variables with `cov()` and `cor()`.

```
> cov(iris$Sepal.Length, iris$Petal.Length)
```

```
[1] 1.274315
```

```
> cov(iris[,1:4])
```

```
             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    0.6856935  -0.0424340    1.2743154   0.5162707
Sepal.Width    -0.0424340   0.1899794   -0.3296564  -0.1216394
Petal.Length    1.2743154  -0.3296564    3.1162779   1.2956094
Petal.Width     0.5162707  -0.1216394    1.2956094   0.5810063
```

```
> cor(iris$Sepal.Length, iris$Petal.Length)
```

```
[1] 0.8717538
```

```
> cor(iris[,1:4])
```

```
             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    1.0000000  -0.1175698    0.8717538   0.8179411
Sepal.Width    -0.1175698   1.0000000   -0.4284401  -0.3661259
Petal.Length    0.8717538  -0.4284401    1.0000000   0.9628654
Petal.Width     0.8179411  -0.3661259    0.9628654   1.0000000
```

Next, we compute the stats of `Sepal.Length` of every `Species` with `aggregate()`.

```
> aggregate(Sepal.Length ~ Species, summary, data=iris)
```

```
     Species Sepal.Length.Min. Sepal.Length.1st Qu. Sepal.Length.Median
1     setosa             4.300                4.800               5.000
2 versicolor             4.900                5.600               5.900
3  virginica             4.900                6.225               6.500
  Sepal.Length.Mean Sepal.Length.3rd Qu. Sepal.Length.Max.
1             5.006                5.200             5.800
2             5.936                6.300             7.000
3             6.588                6.900             7.900
```

We then use function `boxplot()` to plot a box plot, also known as box-and-whisker plot, to show the median, first and third quartile of a distribution (i.e., the 50%, 25% and 75% points in cumulative distribution), and outliers. The bar in the middle is the median. The box shows the interquartile range (IQR), which is the range between the 75% and 25% observation.

```
> boxplot(Sepal.Length~Species, data=iris)
```

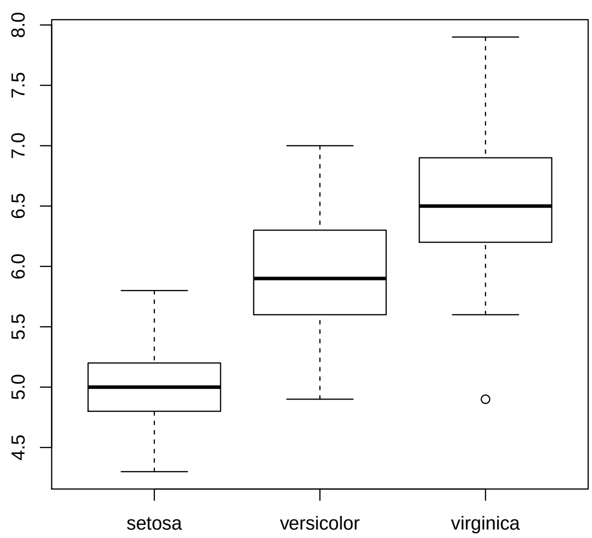


Figure 3.5: Boxplot

A scatter plot can be drawn for two numeric variables with `plot()` as below. Using function `with()`, we don't need to add "`iris$`" before variable names. In the code below, the colors (`col`) and symbols (`pch`) of points are set to `Species`.

```
> with(iris, plot(Sepal.Length, Sepal.Width, col=Species, pch=as.numeric(Species)))
```

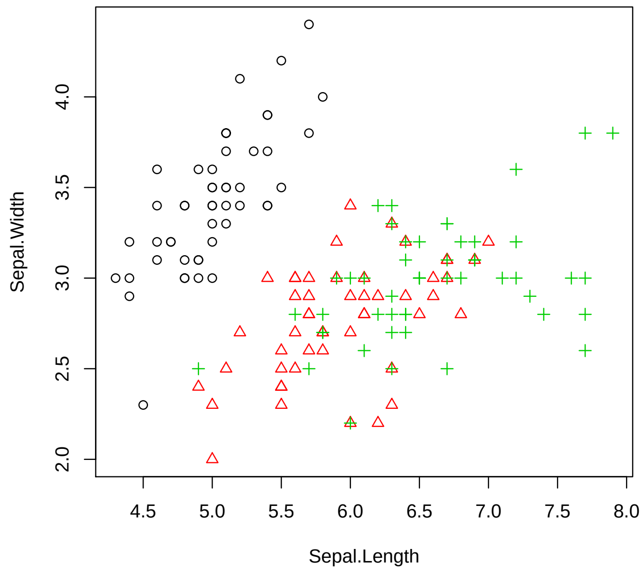


Figure 3.6: Scatter Plot

When there are many points, some of them may overlap. We can use `jitter()` to add a small amount of noise to the data before plotting.

```
> plot(jitter(iris$Sepal.Length), jitter(iris$Sepal.Width))
```

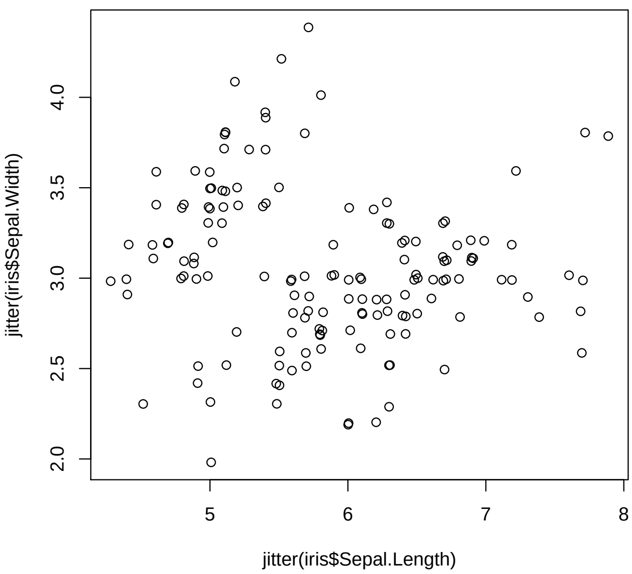


Figure 3.7: Scatter Plot with Jitter

A matrix of scatter plots can be produced with function `pairs()`.

```
> pairs(iris)
```

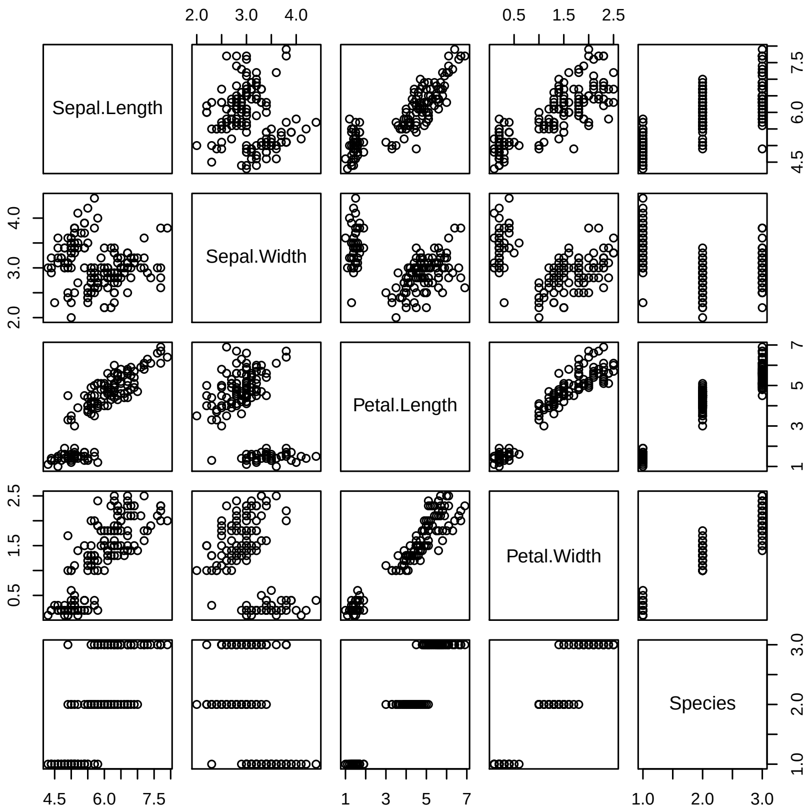


Figure 3.8: A Matrix of Scatter Plots

## 3.4 More Explorations

This section presents some fancy graphs, including 3D plots, level plots, contour plots, interactive plots and parallel coordinates.

A 3D scatter plot can be produced with package *scatterplot3d* [Ligges and Mächler, 2003].

```
> library(scatterplot3d)
> scatterplot3d(iris$Petal.Width, iris$Sepal.Length, iris$Sepal.Width)
```

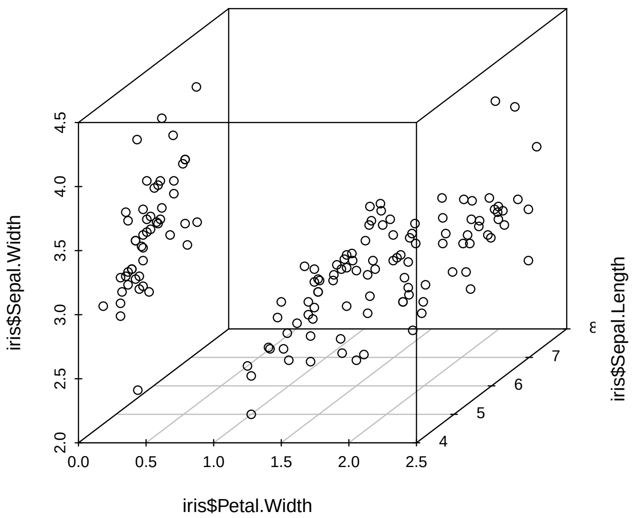


Figure 3.9: 3D Scatter plot

Package *rgl* [Adler and Murdoch, 2012] supports interactive 3D scatter plot with `plot3d()`.

```
> library(rgl)
> plot3d(iris$Petal.Width, iris$Sepal.Length, iris$Sepal.Width)
```

A heat map presents a 2D display of a data matrix, which can be generated with `heatmap()` in R. With the code below, we calculate the similarity between different flowers in the `iris` data with `dist()` and then plot it with a heat map.

```
> distMatrix <- as.matrix(dist(iris[,1:4]))
> heatmap(distMatrix)
```

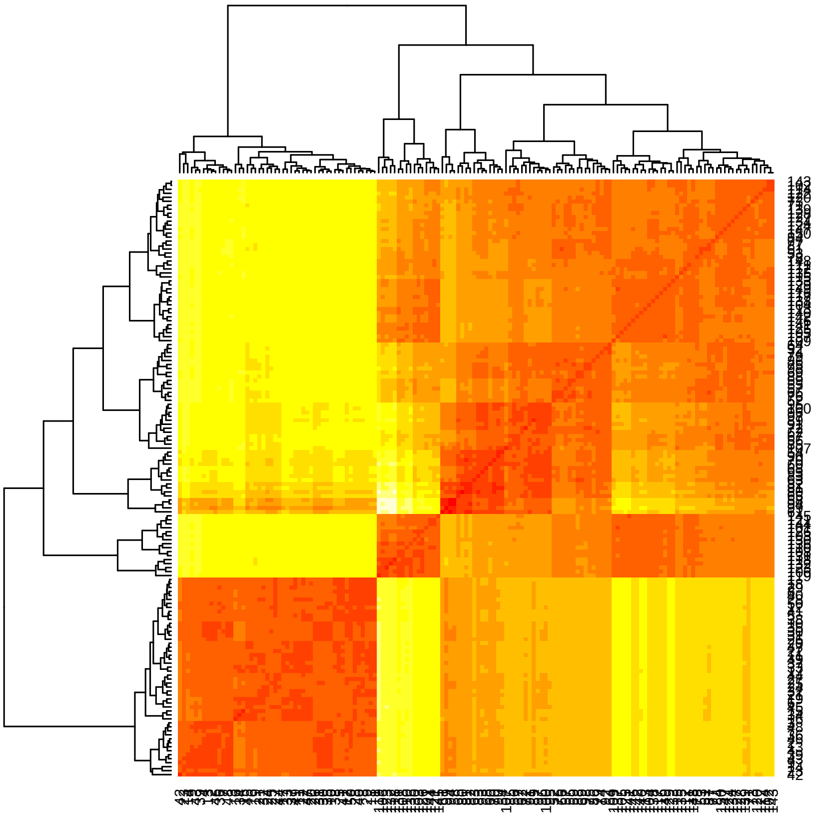

Figure 3.10: Heat Map

A level plot can be produced with function `levelplot()` in package *lattice* [Sarkar, 2008]. Function `grey.colors()` creates a vector of gamma-corrected gray colors. A similar function is rainbow(), which creates a vector of contiguous colors.

```
> library(lattice)
> levelplot(Petal.Width~Sepal.Length*Sepal.Width, iris, cuts=9,
+           col.regions=grey.colors(10)[10:1])
```

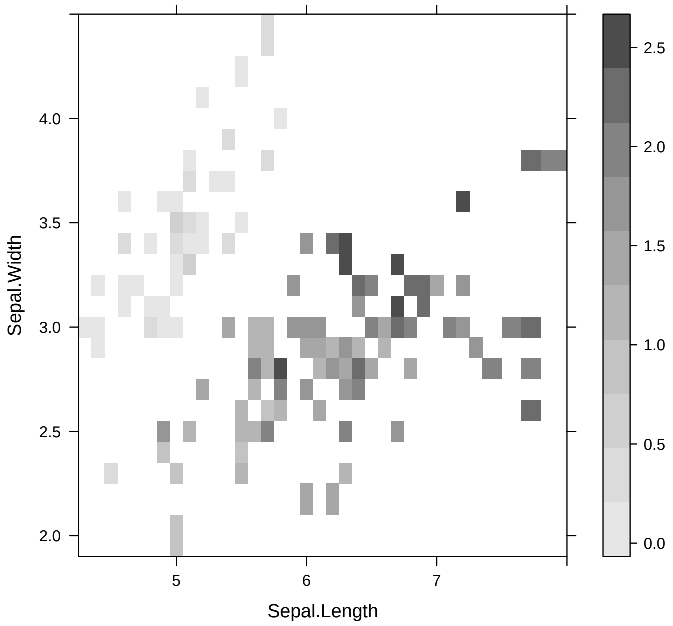


Figure 3.11: Level Plot

Contour plots can be plotted with contour() and filled.contour() in package *graphics*, and with `contourplot()` in package *lattice*.

```
> filled.contour(volcano, color=terrain.colors, asp=1,
+                plot.axes=contour(volcano, add=T))
```

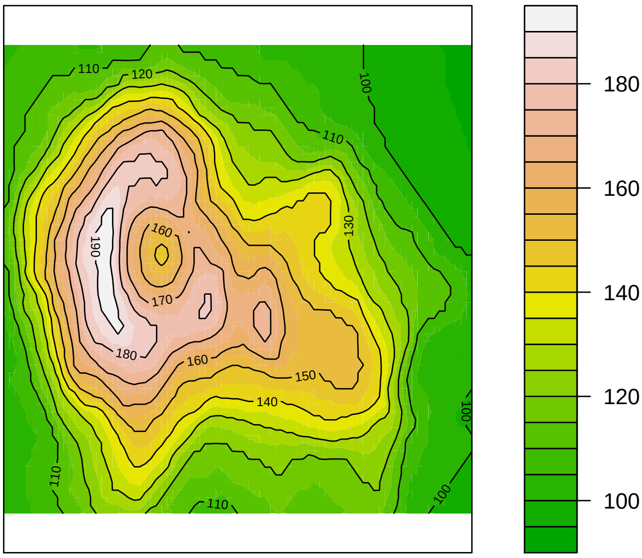


Figure 3.12: Contour

Another way to illustrate a numeric matrix is a 3D surface plot shown as below, which is generated with function `persp()`.

```
> persp(volcano, theta=25, phi=30, expand=0.5, col="lightblue")
```

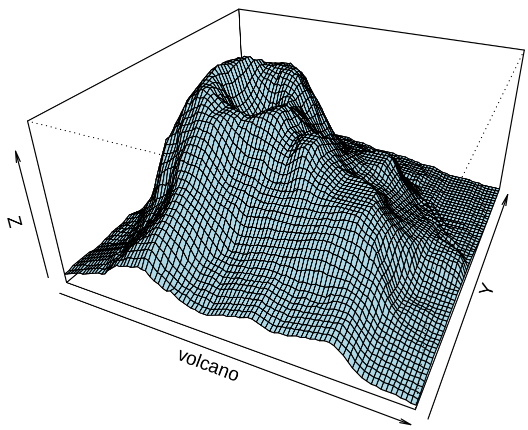


Figure 3.13: 3D Surface

Parallel coordinates provide nice visualization of multiple dimensional data. A parallel coordinates plot can be produced with `parcoord()` in package *MASS*, and with `parallelplot()` in package *lattice*.

```
> library(MASS)
> parcoord(iris[1:4], col=iris$Species)
```

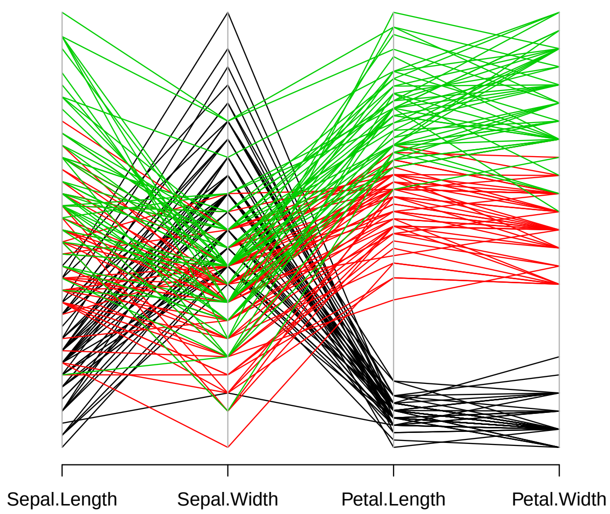


Figure 3.14: Parallel Coordinates

```
> library(lattice)
> parallelplot(~iris[1:4] | Species, data=iris)
```

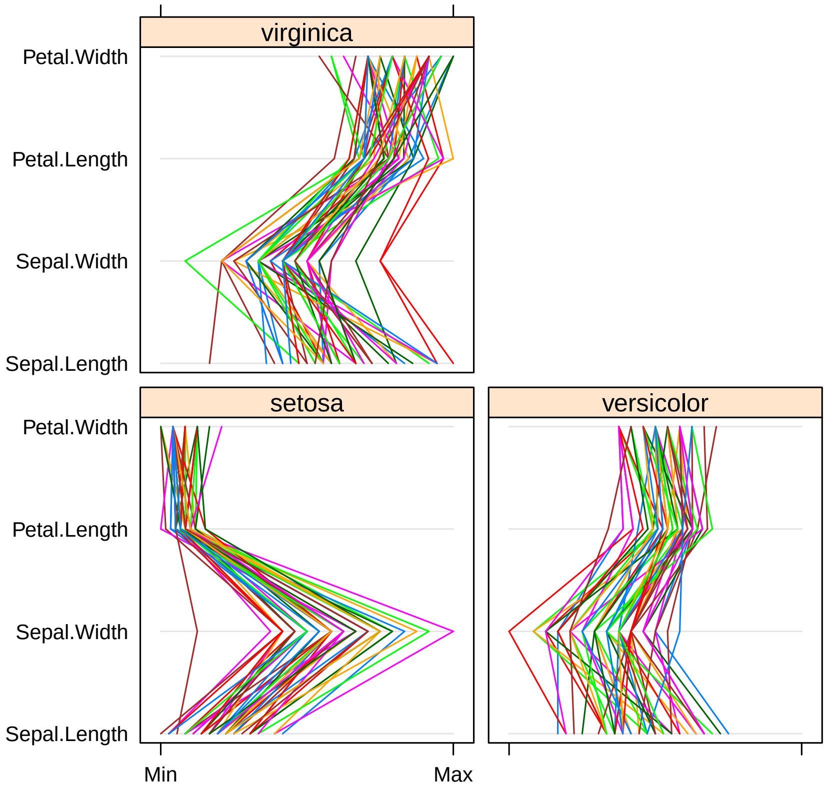


Figure 3.15: Parallel Coordinates with Package *lattice*

Package *ggplot2* [Wickham, 2009] supports complex graphics, which are very useful for exploring data. A simple example is given below. More examples on that package can be found at `http://had.co.nz/ggplot2/`.

```
> library(ggplot2)
> qplot(Sepal.Length, Sepal.Width, data=iris, facets=Species ~.)
```

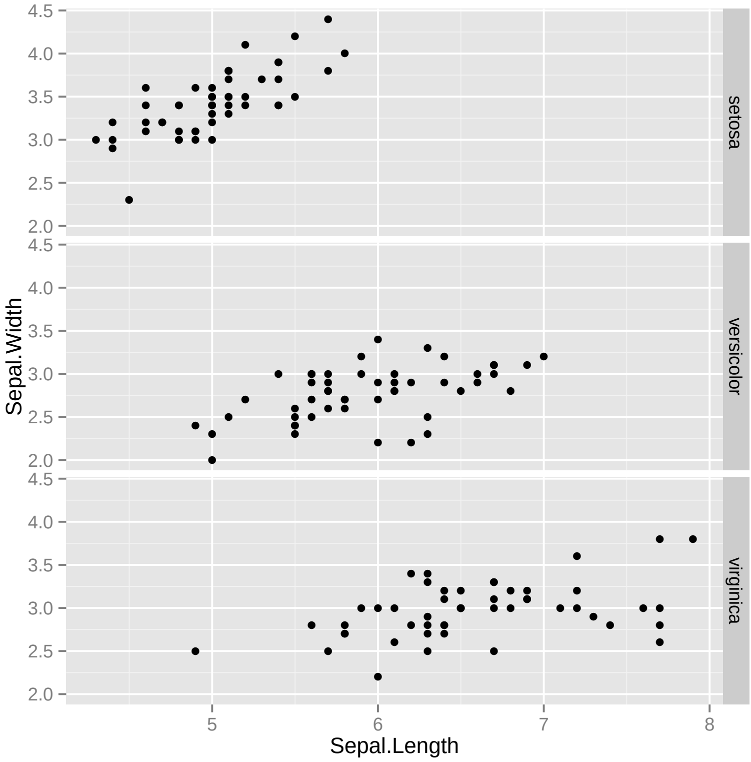


Figure 3.16: Scatter Plot with Package *ggplot2*

## 3.5 Save Charts into Files

If there are many graphs produced in data exploration, a good practice is to save them into files. R provides a variety of functions for that purpose. Below are examples of saving charts into PDF and PS files respectively with `pdf()` and `postscript()`. Picture files of BMP, JPEG, PNG and TIFF formats can be generated respectively with `bmp()`, `jpeg()`, `png()` and `tiff()`. Note that the files (or graphics devices) need be closed with `graphics.off()` or `dev.off()` after plotting.

```
> # save as a PDF file
> pdf("myPlot.pdf")
> x <- 1:50
> plot(x, log(x))
> graphics.off()
> #
> # Save as a postscript file
> postscript("myPlot2.ps")
> x <- -20:20
> plot(x, x^2)
> graphics.off()
```