Chapter 2

Introductory examples

2.1 Introduction

The examples illustrate the use of R for implementing four methods commonly applied in quantitative archaeology applications. They are principal component analysis (PCA), correspondence analysis (CA), cluster analysis, and linear discriminant analysis (LDA). These are all examples of multivariate methods and are often considered to be at the 'complex' or 'advanced' end of the spectrum of quantitative methods used in archaeology. They usually feature in introductory treatments towards the end of the text, if at all.

Apart from introducing R the main aim of this chapter is to illustrate how computationally simple it is to implement such 'advanced' methodologies in R. It will also be argued that the ideas involved are easy to understand; this is discussed in more detail in the chapters devoted to each topic, where some of the finer points of application are covered.

As far as ideas go, given a table of data with n rows and p columns the aim of three of the methods discussed (PCA, CA, LDA) is often to produce a two-dimensional 'map' or bivariate plot of the data that shows (approximately) how similar rows are to each other in terms of the distance between them. Details depend on the type of data available and precise purpose of the analysis and are covered in later chapters. Cluster analysis is also a way of visualizing the similarity between the rows of the table, but uses a different form of graphical representation in the form of a tree-diagram or dendrogram.

The aim in general is to investigate structure in the data, not otherwise easily done for 'large' tables with $p > 3$. In order to concentrate on the practicalities of application, the data sets analyzed below have been chosen because their structure is fairly obvious. Complications arise in practice such as the occurrence of unusual data (outliers), the common need for some form of data pre-treatment (eg., data

transformation), the absence of clear structure, and so on, dealt with in later examples.

2.2 Example – Principal component analysis

The data in Table B.1, from Tubb *et al.* (1980), consist of $n = 48$ rows and $p = 10$ columns to be referred to as *cases* and *variables*. The first nine columns, the data to be analyzed, are concentrations (%) of oxides in specimens of Romano-British pottery. This defines a 48×9 *data matrix*; the tenth column, coding the region of the kiln site where the pottery was found, is used to label plots.

In the original paper several questions were posed.

- Ignoring 'region' is there evidence of chemical grouping in the data?
- If grouping exists can it be associated with region?
- What variables contribute most to group separation, if groups exist?
- Can a subset of variables describe the data well?

With minor complications, to be discussed, these turn out to be fairly simple to answer. Figure 2.1 is a PCA biplot of standardized data based on the oxides (Section 7.2). A single command line was used to obtain the plot

```
biplot(prcomp(tubb.data, scale = TRUE))
```

where `tubb.data` is the name given to the 48×9 data matrix in `R`. The argument `scale = TRUE` standardizes the data so that all variables have zero mean and unit variance, giving them equal weight. Other options are possible (Section 7.2) but this scaling is often desirable (Venables and Ripley, 2002: 303). The appearance of the plot can be customized using other arguments – it is kept simple here.

In mathematical terminology, the data exist in a 9-dimensional space defined by the number of variables. The distance between cases in nine dimensions can be defined mathematically but can't easily be visualized . What PCA is often used for is to transform the data to new variables – *principal components* (PCs) – such that bivariate plots based on the first two of these *approximate* in two dimensions the distances between cases in nine dimensions. The points labeled 1–48 (the row numbers) identify the cases.

It is readily apparent that, apart from a few stray cases, there are three groups in the data, and is easily shown (see Figure 2.2) that these correspond to the three regions. The reasonably tight group to the left corresponds to Region 3, and the central group to Region 1. A biplot also provides information about

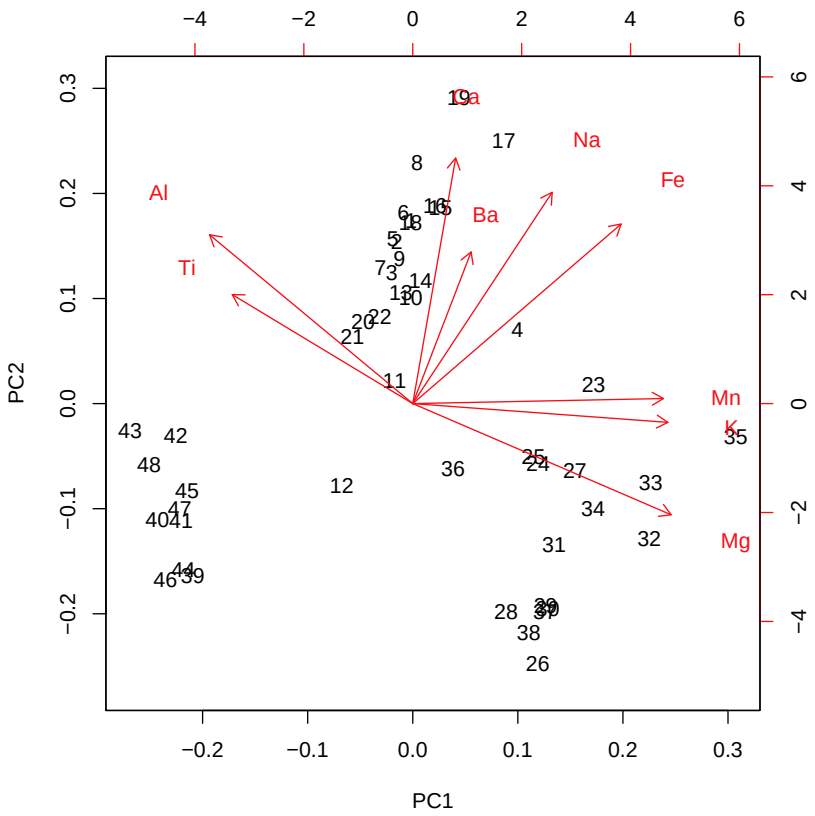

Figure 2.1: *PCA biplot of the standardized chemical data from Table B.1.*

the relationship between the variables. The arrowed lines (vectors) point to the variable markers. Angles between vectors approximate the correlations between variables[1], so we can infer that Al and Ti are strongly positively correlated with each other; negatively correlated with Mg, Mn and K; and poorly correlated with Na. The relative positions of row and column markers suggests that the group to the lower left, compared to other groups, has relatively low values for several variables that plot opposite it. This can, of course, be checked.

The previous code illustrates how simple it is to produce a PCA in a single line of code, and the default output obtained is very informative. There are, however, advantages in breaking the code up, to enhance readability and open up presentational possibilities not otherwise readily available. Thus

[1]Strictly speaking, it's the cosines of angles that approximate correlations.

```
tubb.pca <- prcomp(tubb.data, scale = TRUE)
biplot(tubb.pca)
```

will produce Figure 2.1, but also creates an *object*, `tubb.pca`, that holds information that can be manipulated for presentational and interpretive purposes. This is discussed in detail in Chapter 7, but an illustration of what can be done is shown in Figure 2.2, the code for which is given in Section 2.6. Suppose that only a plot based on cases is required, labeled according to region (remembering that this information is not otherwise used in the PCA). In the figure cases are labeled by number and colored plotting symbols that differentiate between regions.

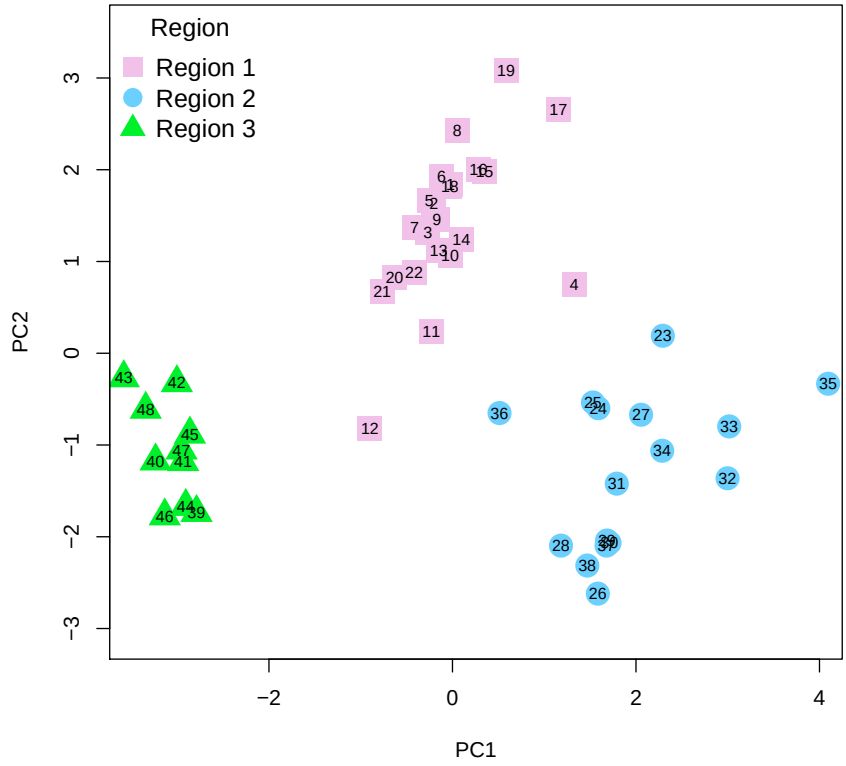

Figure 2.2: *An enhanced PCA row plot of the standardized oxide compositional data from Table B.1.*

An important point to note is that the axes of the plot are equally scaled. This allows the separation between cases to be interpreted as (approximate) distances. Equal scaling is not a feature readily available in some widely used statistical packages. The `biplot` function automatically provides this; for Figure 2.2

the `eqscplot` function produces equal scaling where this is not automatic (Section 2.6.3).

It is as easy to look at these kind of data using PCA as it is to use 'conventional' and 'simpler' graphical methods (which should not be ignored). If needed the PCA can be revisited and refined. Simply scanning the table of data can be informative. This is best done from hard copy, rather than looking at data on a terminal. Looking at the columns of Table B.1 reveals some obvious outliers, almost certainly typos. These are cases 4 (Ti = 0.03), 35 (Mn = 0.394) and 36 (K = 0.81). There is also some evidence of outliers within regions.

In Figure 2.1 the outliers stand a bit apart from the regional groupings to which they belong, with the suggestion of a small tight sub-group in Region 2 to the bottom right. Case 12 seems isolated relative to its regional group, but a chemical reason exists for this (not discussed here). Only a small number of variables are needed to show regional differences. Lacking regional information, sensible univariate or bivariate data inspection would reveal clusters. Two bivariate plots are shown in Figure 2.3 where a clear outlier for the variable K is obvious[2].

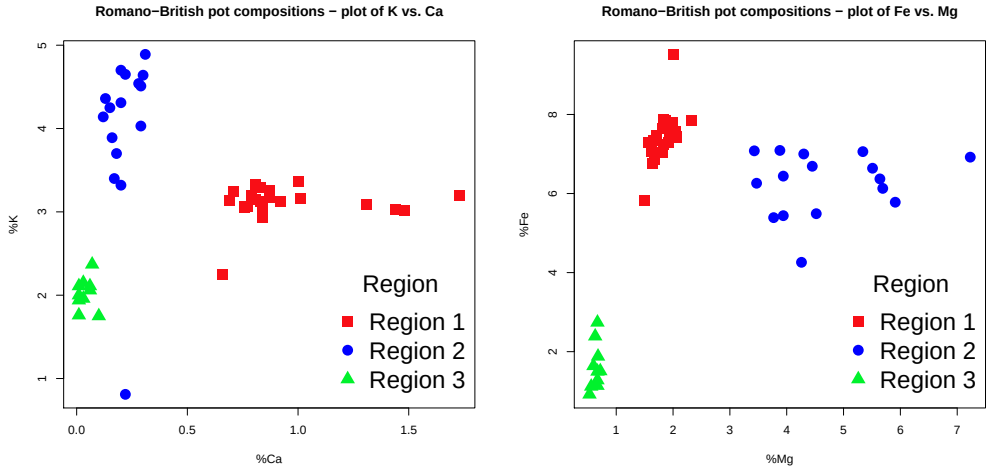

Figure 2.3: *Bivariate plots for selected variables from Table B.1. See the text for an explanation.*

The bivariate plots show that regional clusters are chemically distinct, that only two variables are needed to show this, and that these can be chosen in more than one way. The variable choice can be undertaken in several ways, but inspection

[2]Scatterplot matrices, or pairs plots, can be used to produce displays of all possible bivariate plots. These can be inspected to identify interesting pairs of variables that can be separately plotted in more detail. The `pairs` (Section 5.2) or `scatterplotMatrix` (Section 6.2) functions are available for this kind of analysis.

of Table B.1 is all that is needed here. For example, values for Mg are different between regions; those for Fe separate Region 3 from the other two regions; Ca separates Region 1 from the other two; and so on. In terms of the questions posed at the outset of this analysis intelligent inspection of Table B.1 may be all that is needed, which is what was meant when stated earlier that the data were 'easy to analyze'.

Once groups are established, identifying how they differ is of interest. This is often done by presenting a table of means, standard deviations and other summary statistics. The (arithmetic) `mean` is a *measure of location* usually used with the idea that it is, in some sense, 'typical' of the data. The `median` is an alternative if there are obvious outliers in the data. Both are inappropriate as a measure of what is typical if there is are clear sub-groups within those being summarized. That is, using a simple statistic such as the mean, as with the choices made in PCA, requires a consideration of the validity of the assumptions involved. The `standard deviation` is a `measure of dispersion` or spread often used in conjuction with the mean; the `interquartile range` (IQR) is a measure of dispersion often associated with the median. Table 2.1 shows various measures of typicality and spread for all the data for K, and for the regions. The clear regional differences, and that for other variables, could also be inferred from the biplot of Figure 2.1. The measures of dispersion are greater for Region 2 than the other two regions.

Region	Mean	Mean-	Median	SD	SD-	IQR
Region 1	3.11	-	3.13	0.22	-	0.15
Region 2	4.01	4.22	4.28	0.97	0.48	0.72
Region 3	2.02	-	2.03	0.19	-	0.17
All	3.18	3.22	3.16	0.92	0.86	0.96

Table 2.1: *Summary statistics for K from Table B.1. Mean- and SD- indicate that an outlier, case 36, has been omitted.*

2.3 Example – Correspondence analysis

It is not mandatory, but correspondence analysis (CA) is usually presented as a method appropriate for analyzing two-way cross-tabulations of categorical variables. This includes the special case where the data are recorded as presence or absence, coded as 0–1 (e.g., artifact types and contexts). At the mathematical level there are differences between CA and PCA (Section 9.2) but fundamentally their aims are identical, which is to display the approximate distances between rows, and in the case of CA possibly columns as well, in a 2-dimensional map.

To emphasize the difference in variable types the notation $I \times J$ will be used for the table of data to be analyzed, I and J being the number of categories for the two variables. The data as a whole are often displayed as a biplot - more commonly than with PCA where the emphasis is often on cases. This is not necessarily true of CA, and the roles of the variables can be reversed and a $J \times I$ table analyzed.

A common use of CA is for *seriation* (e.g., Madsen, 1988a; Section 9.5) where it is hoped that an ordering of the rows can be inferred from the row plot and that the order has a chronological interpretation. As an illustration of this kind of application data from McLellan (1979), in a study of the chronology of 'Philistine' burials, are used (Table B.2). Columns in the table correspond to tombs, and rows to counts of 52 different types of pottery found in the tombs. The main interest was on seriating the tombs, with a particular interest in sequencing tombs g to j. A 'horseshoe' shaped pattern to the plots is usually expected in a successful seriation, and the ordering is read around the horseshoe. Archaeological criteria are needed to determine the early and late ends of the sequence. The data have been used in Baxter (2003: 136–8) to illustrate the use of CA for seriation; other purposes to which CA can be put are illustrated in Chapter 9.

A biplot is obtained in a nearly identical fashion to the PCA biplot. With the data held in `burial.data` the `MASS` package needs to be loaded using

```
library(MASS)
```

then

```
biplot(corresp(burial.data, nf = 2))
```

does it. The `corresp` function from the `MASS` package has been used. This needs to be loaded using the `library` function (see Section A.3 for details). The other difference from the PCA analysis is the need for the argument `nf = 2` which specifies the number of 'components' to extract. Two are needed for bivariate plotting with the first two usual, though others can be extracted if wished.

While useful for initial exploratory purposes the resultant biplot is often too crowded to be easily read if there is a large number of rows and/or columns, and it is not shown here. For display purposes separate plots for the rows and columns, presented adjacently, are often to be preferred (see Sections 7.2 and 9.2 for a discussion of plotting issues). This is easily done; for example

```
biplot(corresp(burial, nf = 2), xlabs = rep("", 52))
```

suppresses the printing of labels for the rows. The `rep` function produces blank labels (using `""` for the 52 rows). Column labeling may be suppressed in a similar

way using the `ylabs` argument[3].

Baxter's (2003) Figure 11.1 was produced in the manner just described and revealed an outlying tomb `"c"`, with somewhat larger numbers for four artifact types than other tombs, that cramped the rest of the plot, making it harder to read. As with the PCA analysis there are advantages to breaking down the computations. Specifically, `corresp(burial.data, nf = 2)` creates a CA object allowing access to information more easily manipulated for plotting purposes. This has been done, in much the same way as needed for Figures 2.2 and 2.3 to obtain Figure 2.4, where the outlying tomb has been omitted.

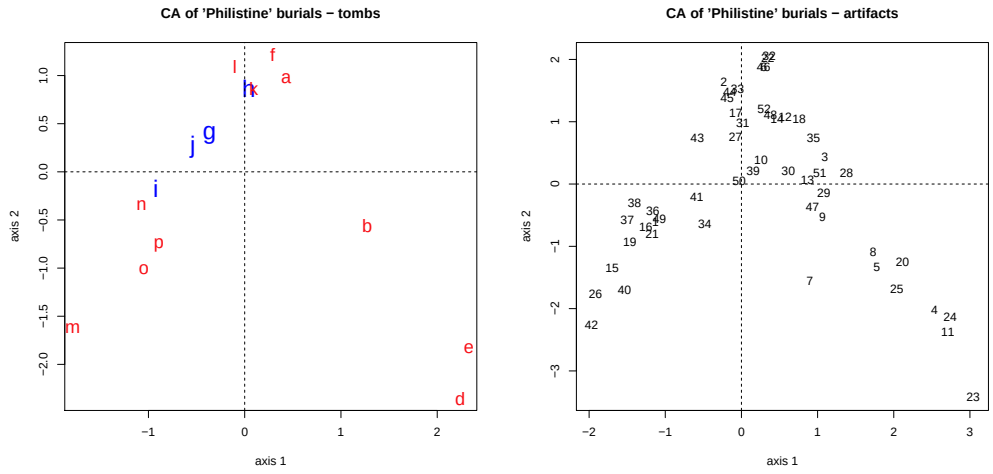

Figure 2.4: *Correspondence analysis plots of the data from Table B.2.*

The plot for tombs shows a good seriation of the data, reflected for the most part in that for the artifact types. Tombs `"g"` to `"j"`, which were of particular interest, are highlighted in blue and with a larger typeface. Their chronological ordering as inferred from the seriation corresponds to that hypothesized by McLellan (1979) using archaeological criteria.

2.4 Example – Cluster analysis

Cluster analysis is a generic term for a wide range of methods that can be implemented in different ways. Given an $n \times p$ matrix of continuous data – the situation assumed here – the common aim is to produce groups, or clusters, of cases such

[3]Several packages contain functions for undertaking CA, which differ in the defaults and the way plots can be labeled. The `ca` function from the `ca` package is used for some of the examples in Chapter 9. In addition to being loaded this package also needs to be imported; see Section A.3 for details.

that cases within a cluster have similar profiles that are distinct from those of clusters.

Obtaining a cluster analysis is straightforward; interpretation not always so. Some of the issues to be taken further in Chapter 10 are raised by the introductory examples presented here. Coding can be reduced to one line as follows.

```
plot(hclust(dist(scale(tubb.data)), method = "ward.D"))
```

The result is shown in Figure 2.5

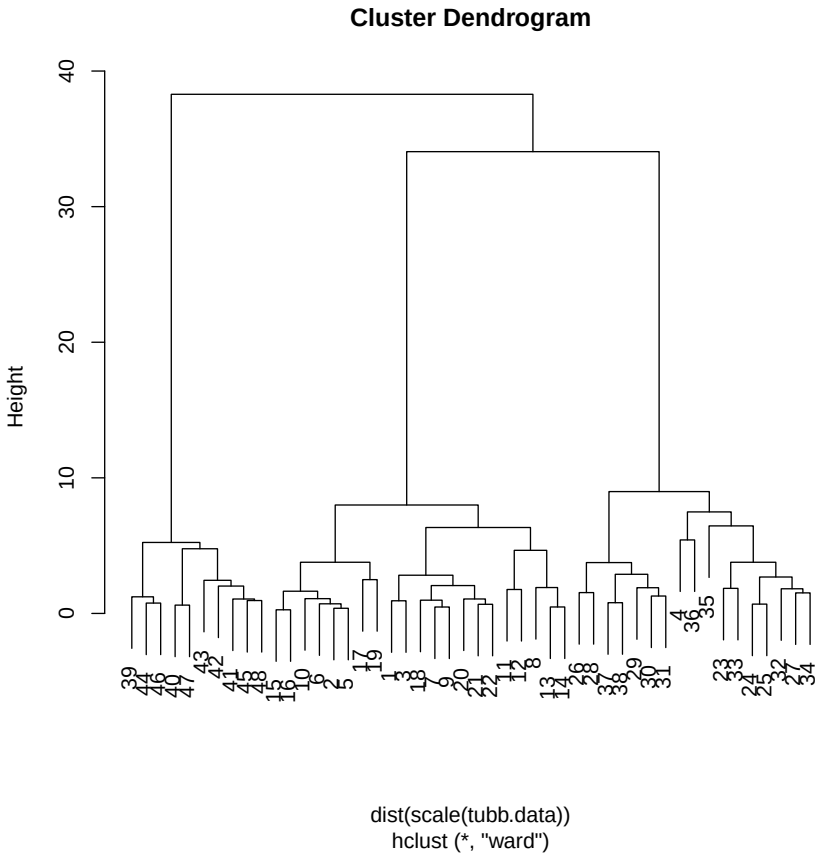

Figure 2.5: *The dendrogram for a Ward's method cluster analysis of the standardized oxide data from Table B.1. This is the default output*

The output produced is a tree-diagram (or *dendrogram*) that can be thought of as consisting of branches and leaves (corresponding to the cases). The idea is to cut the tree at some point to isolate distinct branches whose leaves define the clusters. There are, fairly obviously, three clear clusters in the figure that can be

shown to correspond to the regions. Such identification is not usually as easy when there is less clear structure in the data.

There are three main steps in the methods of cluster analysis most widely used in archaeology. Firstly, the data usually needs to be transformed and this involves the same issues as in PCA (Section 7.2). Here the `scale` function standardizes the data to zero mean and unit variance. Secondly, a measure of (dis)similarity between rows needs to be defined and the `dist` function produces, by default, Euclidean distance[4]. Other choices are possible, but Euclidean distance is much the most common (Section 7.3). Finally, and this is where issues of interpretation arise, a method, or algorithm, for clustering the data needs to be chosen. Hierarchical clustering is common. Cases are initially treated as single clusters and successively merged until a single cluster, of all cases, results. The `hclust` function effects the clustering; there are different algorithms for this that depend on the criterion for merging clusters. The `method = "ward.D"` argument to `hclust` specifies that *Ward's method* is to be used (Section 10.3).

The choice of methods is simply effected using the `method` argument. Choices other than `"ward.D"`, such as `"s"` or `"a"`, produce *single-link* and *average-link* analyses of those illustrated in detail in Chapter 10. These will produce different output from Ward's method, and single-link is illustrated in Figure 2.6. Rather than simply replacing `"ward.D"` with `"s"` the opportunity is used to illustrate the use of the `plot` function, used in conjunction with `hclust`, to show how the default labeling can be 'tidied-up'. The code used is given below.

```
plot(hclust(dist(scale(tubb.data)), method = "s"),
labels = tubb.region, sub = " ", xlab = " ", cex = 0.8,
main = "Single-linkage cluster analysis - Romano-British pot compositions")
```

The `sub = " "` and `xlab = " "` arguments replace the labeling at the bottom of Figure 2.6 with blank space. Replacement text could be added if wished, with the `main` argument showing how a more informative title can be produced. The `labels` argument replaces the default labeling of leaves by row number with regional identifications held in `tubb.region` which needs to be created in advance of analysis. Finally the argument `cex = 0.8` controls the character expansion of the leaf labels, in this case to 0.8 of the default, to remove overlapping.

As far as interpretation goes the appearance of the dendrograms in the two analyses differs, but it is clear that there are three main groups in the single-link analysis, with the revised labeling making it clear that these are regional groups. the main difference is that single-link suggests three outliers (two from Region 2 and one from Region 1) not evident in the Ward's method analysis. This is fairly

[4]Mathematically this is just a generalization of distance as we measure it in two or three dimensions.

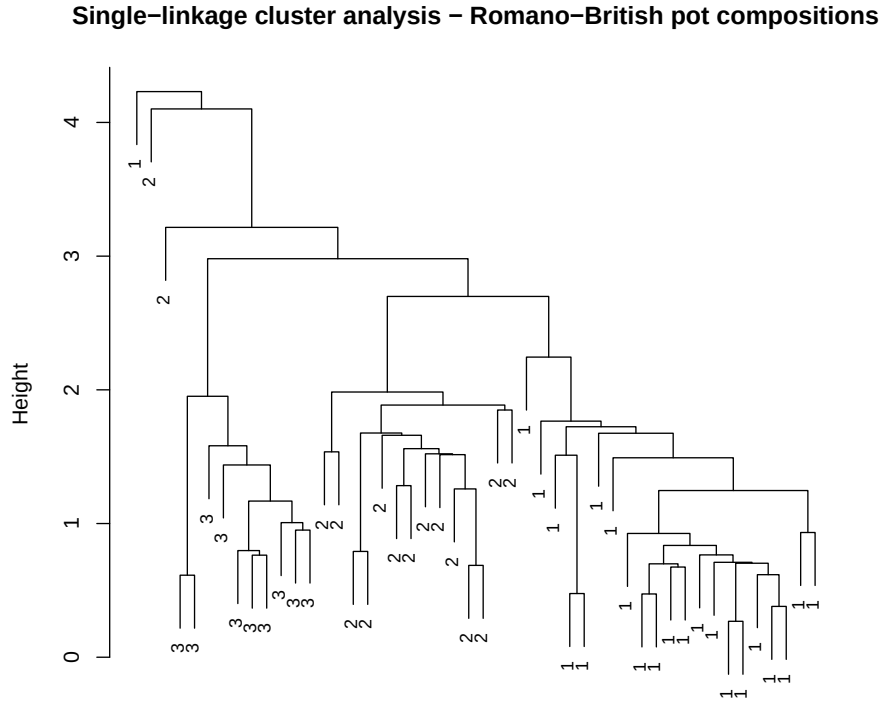

Figure 2.6: *A Single-link cluster analysis of the standardized oxide data from Table B.1.*

characteristic of the way these methods can differ; issues of method choice, cluster validation, and the comparison of analyses are considered in detail in Chapter 10.

2.5 Example – Linear discriminant analysis

The fundamental difference between LDA and PCA is that the former uses the information (or assumptions) about groups in the data in the analysis and, it is hoped, will show much better separation between groups than in the PCA. For what follows it is necessary to load the `MASS` package using `library(MASS)` in order to access the `eqscplot` function, which produces equal scaling of the axes and the `lda` function to carry out the LDA. Once this is done the following one-line command will produce the output of Figure 2.7.

```
eqscplot(predict(lda(tubb.data, tubb.region), dim = 2)$x)
```

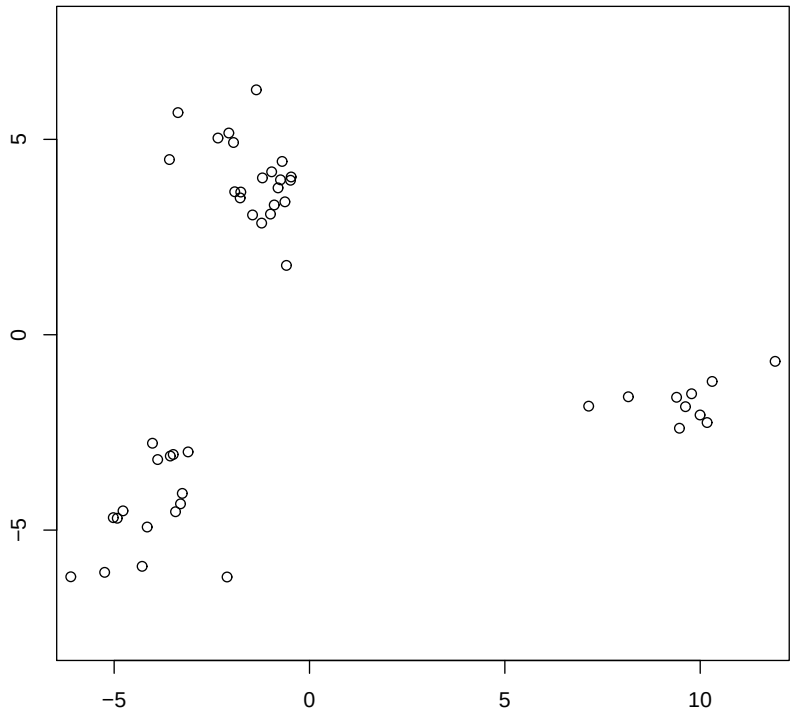

Figure 2.7: *A linear discriminant analysis plot of the data from Table B.1.*

The `lda` function carries out the LDA. The `predict` function with the argument `dim = 2` generates the scores for cases on the first two discriminant functions and the addition of `$x` extracts these for plotting purposes. To use the `eqscplot` function as shown the argument `dim = 2` is needed; the more general case is discussed in Section 2.6.2. More so than the other methods discussed the code benefits from being broken down into more than one line. The plot produced is not entirely satisfactory and would benefit from enhancement. The message is that there are three groups in the data with the separation between them much more evident than for the PCA in Figure 2.2 and this is what we hope to see. It can be assumed, given previous analyses, that the grouping displayed almost certainly corresponds to the regions, but for more general purposes, when the grouping is less clear-cut, we would like as a minimum to label the points to see what group a case is supposed to belong to. How this can be done, along with other aspects of labeling, is discussed at length in Section 2.6.2 as it introduces features of `R` used throughout these notes.

2.6 R notes

2.6.1 Introduction

The chapter concludes, as do other chapters, with notes on the R coding used to obtain the analyses in the chapter where these introduce new features of R. Some features are used on a regular basis and are covered here in the following sub-section to avoid repetition. This includes labeling and presentational options. Some other general features of R that are used regularly are covered in context in other sections. In particular, Section 3.2.2 introduces the idea of user-defined functions, not needed for the present chapter; Appendix A discusses aspects of data entry and accessing user-written packages in more detail than given here.

2.6.2 Aspects of labeling and presentation

The one-line coding used to generate the output for the examples illustrates the ease with which 'advanced' analyses can be undertaken. The plots obtained are useful for initial exploratory purposes; from the point of view of presentation and interpretation some form of enhancement is desirable and sometimes essential. The least satisfactory of the default ouputs presented in the examples was that for LDA, where the regional information is essential for plot construction and ought to be included in the plot in some way. Code for an enhanced version of Figure 2.7, with the associated output, is given below as a peg on which to hang a discussion of aspects of labeling. Without further ado an enhanced version of Figure 2.7 is shown in Figure 2.8 that uses the following code[5].

```
library(MASS)
tubb.lda <- lda(tubb.data, tubb.region)
tubb.ld <- predict(tubb.lda)$x
x1 <- tubb.ld[,1]
x2 <- tubb.ld[,2]

eqscplot(x1, x2, xlab = "first linear discriminant",
ylab = "second linear discriminant", col = Coltubb, pch = Symtubb,
main = "Enhanced LDA - Romano-British pot compositions",
cex = 1.3, cex.axis = 1.2, cex.lab = 1.3, cex.main = 1.3)

legend("topright", c("Region 1", "Region 2", "Region 3"),
col = c("red", "blue", "green2"), pch = c(15,16,17),
bty = "n", title = "Region", cex = 1.4)
```

[5]For display on the page individual directives that can occupy one line if typed at a terminal are sometimes run over two or more lines. Blank lines are used to make it clear where this is occurring. The # symbol comments out what follows it and can be used for annotation.

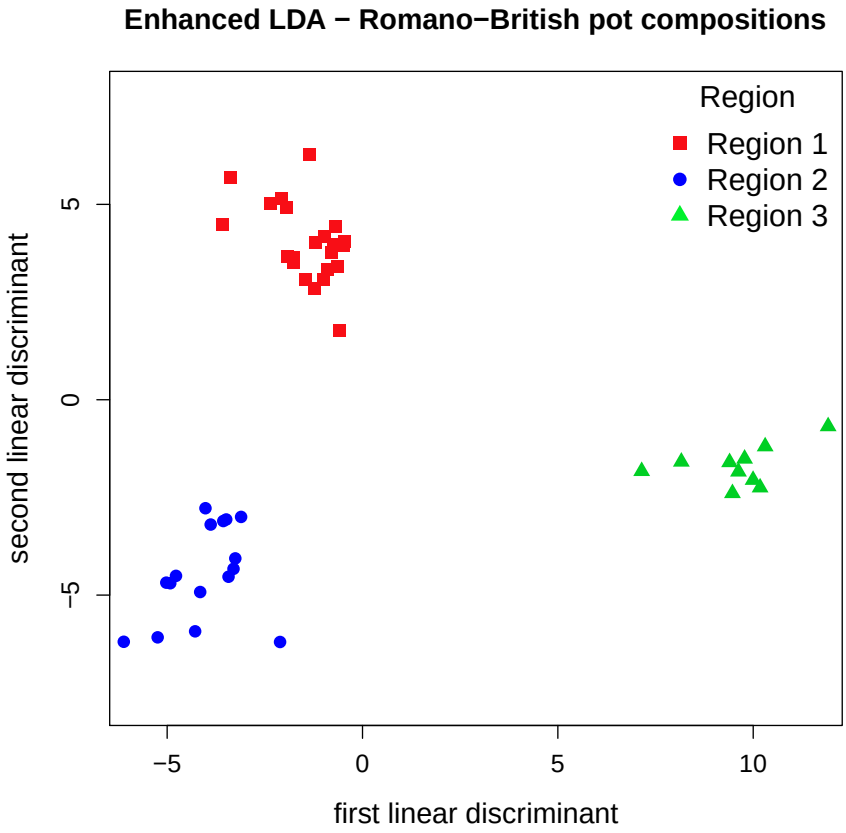

Figure 2.8: *Enhanced linear discriminant analysis plot of the data from Table B.1.*

The original one-line coding has been broken up to make it clearer, and there are slight modifications. The `dim` argument has been omitted from the `predict` function so that `tubb.id` holds information on all the discriminant functions; this requires a slightly different approach from that previously used for plotting. The first two discriminant functions are defined by `x1` and `x2` and these are used in the plotting function[6]. For this particular code to 'work' at least three groups are needed.

The more obvious differences from Figure 2.7 lie in the use of axis labels and titles, the use of different symbols and colors for labeling points, and the provision of a legend. Given the ubiquity of their usage in later examples they are discussed

[6]Note the use of `x1 = tubb.id[ , 1]` etc. to 'pick-out' the discriminant function to use. The `[ , ]` component is used to identify the rows and colums that are included or omitted. Thus, `x[-4, 1:2]` would extract the first two discriminant function omitting case 4. More complicated uses are illustrated in other chapters.

in turn, in some detail.

Labels and titles

Axis labels are produced using the `xlab` and `ylab` arguments in `plot` with the title supplied by the `main` argument. Note that, here and elsewhere, the text must be enclosed in double quotation marks, `" "`. If the content is left blank the axis labels and/or title are also blank. This can be useful for suppressing default labeling. The arguments `cex`, `cex.axis`, `cex.lab` and `cex.main` control the size of the plotting symbols, the size of the numbers on the axes, the size of the axis labels and the size of the main title. What is appropriate will depend on how the output will eventually be presented.

Plotting symbols

In general, points can be plotted using different symbols for individual cases and these are supplied by the plotting character argument, `pch` in the `plot` function (or `eqscplot` if equally scaled axes are needed). The default is to plot using open circles that are otherwise undifferentiated. The argument `pch = Symtubb` was used in the `eqscplot` function above. `Symtubb` is a list of plotting symbols that needs to be created, identified by numbers, of the same length as n. There are 22, 16 and 10 cases for the three regions and the rows are blocked by region. If `Symtubb` is defined as

```
Symtubb <- c(rep(15, 22), rep(16, 16), rep(17, 10))
```

this does the job, where 15, 16 and 17 correspond to solid squares, circles and triangles. The `rep` function replicates its first argument, the second argument defining the number of copies needed. Thus `rep(15, 22)` produces 22 replicates of the number 15; the function `c(...)` combines the arguments given in ... into a list of length 48. The available plotting symbols can be located using judicious Googling and are listed on several websites[7]. They are shown, for convenience of reference, in Figure 2.9. In the code given the character expansion `cex = 1.3` is used to produce more 'visible' symbols on the page than the default.

If only a single plotting symbol is needed, different from the default (e.g., solid circles) then `pch = 16` is sufficient. The graphs in these notes mostly use the symbols 15–17 if three or fewer are needed.

[7] http://research.stowers-institute.org/efg/R/Color/Chart/index.htm for one source for symbols and colors.

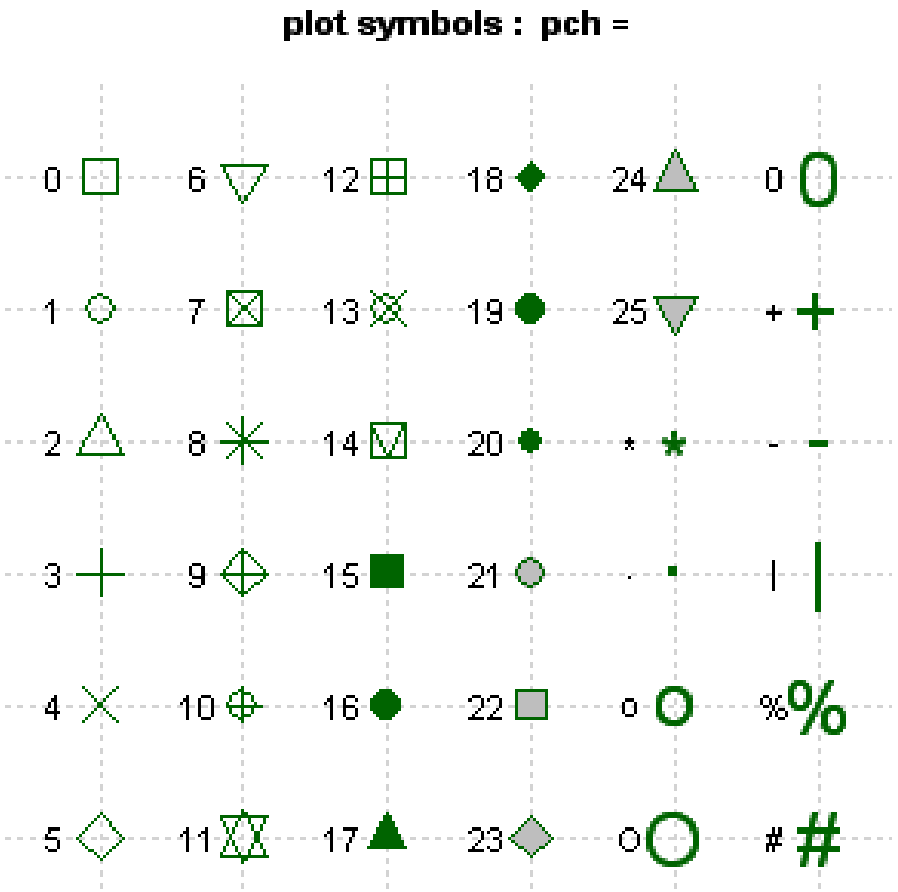

Figure 2.9: *Available symbols for plotting points in* `R`.

Colors

Colors are treated in much the same way as plotting symbols using the `col` argument to the plotting function. The argument `col = Coltubb` specifies the colors used in the example, where `Coltubb` has been defined as

```
Coltubb <- c(rep("red", 22), rep("blue", 16), rep("green2",10))
```

A list of available colors can be obtained in `R` using the functions `colors()` or `colours()`; there are 657 in total. The colors `"red"`, `"blue"` and `"green2"` are numbers 552, 26 and 257 in the list. In the definition of `Coltubb` the text identifiers can be replaced with the numbers as follows, and if wished.

```
Coltubb2 <- c(rep(colors()[552], 22), rep(colors()[26], 16),
              rep(colors()[257],10))
```

This is not as 'neat' as the previous construction. It requires the numeric identifiers and these can be obtained from the listing produced by `colors`. An alternative is to use Figure 2.10 (from the same source as Figure 2.9) to identify color labels that seem suited to the purpose. Apart from displaying the colors on offer this is

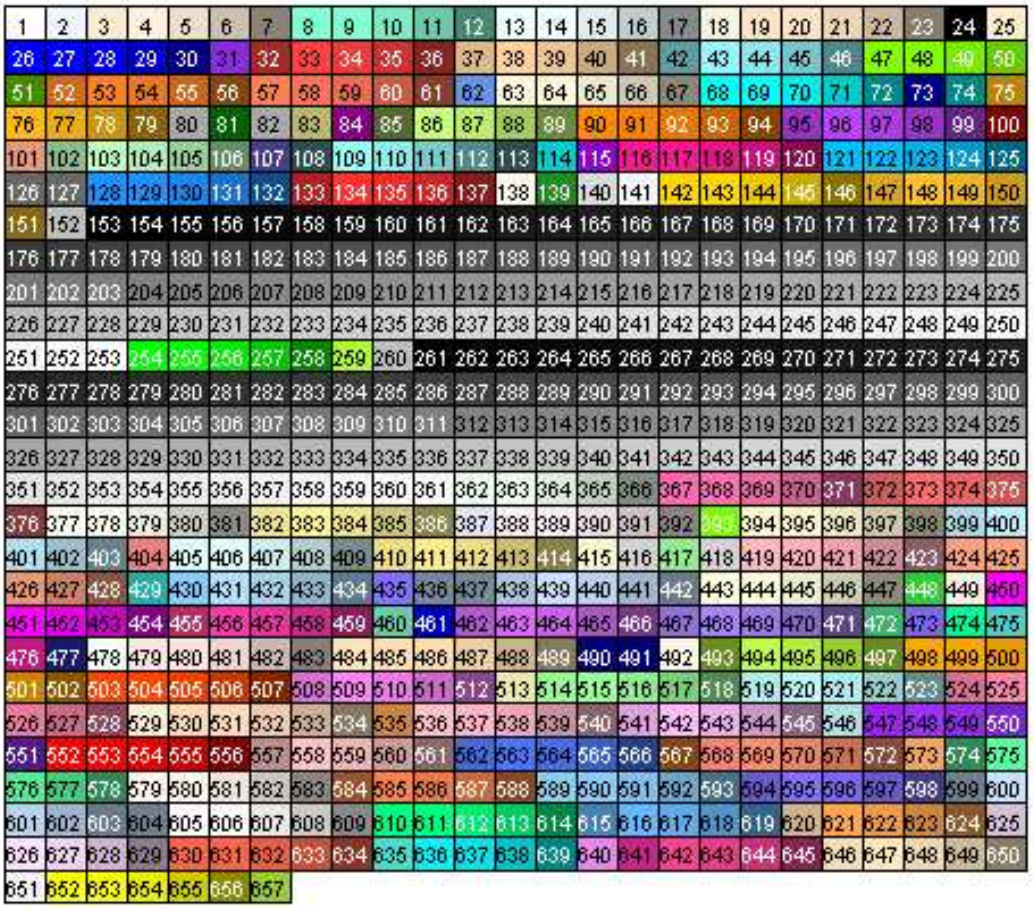

Figure 2.10: *Available colors for plotting in* `R`.

potentially useful for identifying adequately contrasting colors where several are needed – not always straightforward. In these notes, when two or three or colors are needed `"red"`, `"blue"` and `"green2"` are most used, the last tending to display more satisfactorily than `"green"` in many examples. For a single color something like `col = "blue"` will suffice.

Lines

Lines are not needed for Figure 2.8 but it is convenient to discuss them here. The `lty` and `lwd` arguments control the line type and width. The available line types are shown in Figure 2.11, from the same source as Figures 2.9 and 2.10. The default is `lty = 1`, a solid line, with `lwd = 1`; line color can be controlled using the `col` argument. Lines can be added to plots using the `abline` and `lines` functions, for which the same control is available.

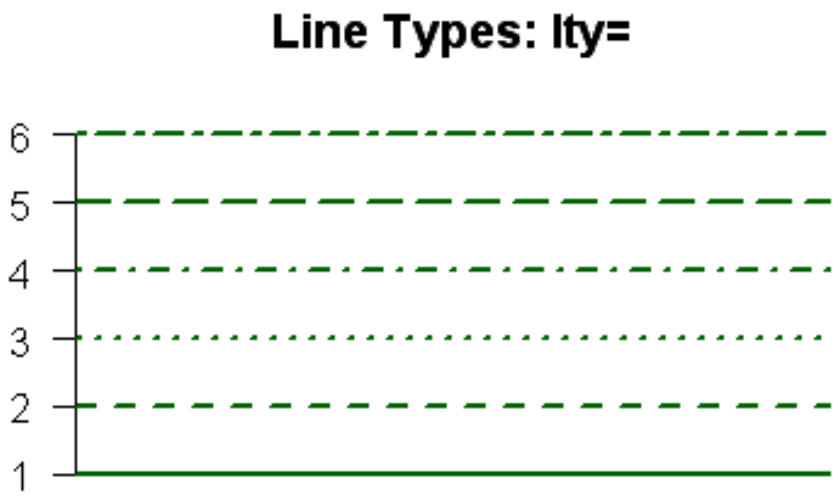

Figure 2.11: *Available lines for plotting in* `R`.

Legends

On first acquaintance the `legend` function can look quite forbidding – see `?legend`. Figure 2.8 is a fairly minimalist example that illustrates some of the more useful features.

In the first argument to the function given in the example code the position of the legend is specified. This can be done by providing the coordinates for the top left-hand corner of the 'box' that encloses the legend, but it is simpler, if space and aesthetics permit, to place the legend in one of the four corners of the figure using one of `"topright"`, as in the example, `"bottomleft"` etc.

The second argument lists the names to be used in the legend. The `col` and `pch` arguments are lists with the same length as that of the legend that indicate the colors and plotting characters associated with each component of the legend; `cex` controls the character expansion used in the legend with `pt.cex` available for addition control over point sizes. The default is to have a visible box enclosing

the legend; `bty = "n"`, as used here, renders this invisible. A title for the legend can be added using the `title` argument. Other features will be introduced, when needed, in later examples.

The code for the examples in this chapter will be provided in detail, where this is not given in the text. In subsequent chapters arguments associated with the labeling and the legend will often be omitted unless it is useful to illustrate features not previously discussed. Thus the code for the example used here might be presented as

```
library(MASS)
tubb.lda <- lda(tubb.data, tubb.region)
tubb.ld <- predict(tubb.lda)$x
x1 <- tubb.ld[,1]
x2 <- tubb.ld[,2]
eqscplot(x1, x2)
```

2.6.3 Code used for analyses in the text

Figure 2.2

The main aim in this plot, apart from serving as a first illustration of plot construction, was to show the disposition of cases corresponding to different regions, while simultaneously labeling points by case number so that potential outliers could be identified. Additionally, the plotting of variable markers provided by the `biplot` default was not of interest, and greater control over labeling than that easily possible with `biplot` was needed.

```
library(MASS)
Coltubb <- c(rep("pink", 22), rep("skyblue", 16), rep("green2",10))
Symtubb <- c(rep(15, 22), rep(16, 16), rep(17, 10))
tubb.pca <- prcomp(tubb.data, scale = TRUE)
tubb.x <- tubb.pca$x
x1 <- tubb.x[,1]; x2 <- tubb.x[,2]

eqscplot(x1, x2, col = Coltubb, pch = Symtubb, xlab = "PC1",
ylab = "PC2", cex = 2.5)

text(x1, x2, 1:dim(tubb.x)[1], cex = 0.75)

legend("topleft", c("Region 1", "Region 2", "Region 3"),
col = c("pink", "skyblue", "green2"), pch = c(15, 16, 17),
title = "Region", bty = "n", cex = 1.2, pt.cex = 2)
```

The `prcomp` function carries out the PCA with the first argument specifying the data to be used, and the `scale = TRUE` argument standardizing the data.

Principal component scores are extracted using `tubb.pca$x` and stored in `tubb.x`. Next, `x1` and `x2` are defined to hold the scores for the first two components needed for plotting. The commands to generate these have been run together on the same line with a semi-colon needed to separate them.

The structure of the `eqscplot` function is as previously illustrated but `Coltubb` is constructed with lighter colors that the previous example, so that case numbers are more easily read when superimposed on them. The `text` function adds the case numbers to the plot. The first two arguments are the variables used and are the same as those used with the `eqscplot` function. The third argument supplies the labels to be used. In this example the function `dim` is the dimension of the data matrix, `tubb.x`, that holds the PC scores. It consists of two elements giving n and p, the number of rows and columns of the data matrix. The first of these is what we need and `dim(tubb.x)[1]` extracts it. It is known that $n = 48$ so the effect is to define the third argument as 1:48 which is a quick way of generating the numbers $(1, 2, \ldots, 48)$. Using 1:48 directly is simpler but less general.

The default text color is `"black"` but it can be changed using the `col` argument if preferred. The `cex` values vary between the `eqscplot` and `text` function. The idea is to arrange things so that the text fits within the symbols, and some experimentation was needed to obtain the effect shown.

Table 2.1

```
K <- tubb.data$K   # tubb.data[ , 6] could also be used
# Create new data omitting an outlier, case 36
K_Out <- K[-36]
tubb.region_Out <- tubb.region[-36]
m <- mean(K)
med <-median(K)
sd <- sd(K) # standard deviation
IQR <- IQR(K) # Inter-Quartile Range
statistics <- c(m, med, sd, IQR)
print(round(statistics, 2))
```

The code can be streamlined by writing it as a function (Section 3.2.2) but is adequate for immediate illustrative purposes. The variable K has a column heading of the same name and is the sixth variable in the data matrix; it can be extracted in either of the ways indicated. The code as given obtains the mean, median, standard deviation and interquartile range (IQR) of the data. The `print` function takes whatever is listed and returns it on the terminal. The `round` function takes the first argument – in this case `statistics`, a list of the summary statistics – and rounds it to the number of decimal places given by the second argument.

To do calculations for the regions replace `K` with `K[tubb.region == 1]`, for example, which extracts the data for Region 1 – note the use of square brackets and the 'double equal' symbol, `==`, to define the subset to be extracted. This can be done for each region in turn.

Figure 2.3 reveals a clear outlier for K in Region 2, and it is easily established that this is case 36 in the data table. The variables `K_Out` and `tubb.region_Out` remove this outlier from the data and regional classification. Only Region 2 is affected by the outlier and only the mean and standard deviation are of interest (the median and IQR will not be much affected, if at all, by the outlier). The mean, for example, can then be obtained by using

$$\texttt{m} < -\texttt{mean}(\texttt{K\_Out}[\texttt{tubb.region\_Out} == 2])$$

.

Figure 2.4

The data of Table B.2, named `burial` here, was analyzed in Baxter (2003: 137) where the third tomb was something of an outlier, plotting sensibly but cramping the rest of the display. In what follows, and for clarity of graphical presentation, `burial1` omits this tomb. The left-hand plot in the figure shows a seriation of the data, and the aim was to highlight tombs `"g"` to `"j"` in which there was a particular interest. This was done by modifying the plotting color and size of the labels for these tombs and introduces features not previously used.

```
library(MASS)     # Needed for corresp and eqscplot
burial1 <- burial[ ,-3]     # Omit tomb "c"
z <- corresp(burial1, nf = 2)

# Extract row and column coordinates for plotting purposes
x1 <- z$rscore[,1]; x2 <- z$rscore[,2]
y1 <- z$cscore[,1]; y2 <- z$cscore[,2]

# Row (artifact) plot
eqscplot(x1, x2, type = "n", xlab = "axis 1", ylab = "axis 2",
main = "CA of 'Philistine' burials - artifacts")

text(x1,x2, 1:52)
abline(h = 0, lty = 2); abline(v = 0, lty = 2)

# Column (tombs) plot
Labs <- letters[-c(3, 17:26)]    # Define text for point labels.
```

```
eqscplot(y1, y2, type = "n", xlab = "axis 1", ylab = "axis 2",
main = "CA of 'Philistine' burials - tombs")

Cex <- rep(1.5, 15); Cex[6:9] <- 2
Colburial <- rep("red", 15); Colburial[6:9] <- "blue"
text(y1,y2, Labs, cex = Cex, col = Colburial)
abline(h = 0, lty = 2); abline(v = 0, lty = 2)
```

Once the coordinates are extracted, as indicated, obtaining the row plot is straightforward. In both `eqscplot` commands, which produces equal scaling of the axes, the argument `type = "n"` produces a blank plot with the axis and titles shown. The `text` function adds text (the point labels) to the plots. For the rows the third argument, `1:52`, defines the row numbers 1 to 52. Labeling for the column plot is a little more involved.

The variables `Cex` and `Colburial` define the character expansion and color to be used, and are then modified to produce a larger size with a different color for tombs (columns) `"g"` to `"j"` (6 to 9 or `6:9` in `burial1`). Note that `cex = Cex` is now specified as has been previously the format for colors and symbols as it is now a variable quantity. Labels for points are defined by `Labs` and are the tomb identifiers `"a"` to `"p"`, omitting `"c"`. These are generated using the `letters` function which generates the 26 lower-case letters of the Roman alphabet (use `LETTERS` for uppercase). The letters from `"q"` onwards are not used and need to be omitted along with `"c"`. In `letters[-c(3, 17:26)]` note the use of `-c(3, 17:26)` to list letters to be omitted, specified by their position in the list generated by letters and that, via the minus sign, omission is intended.

Finally the `abline` function is used to add reference lines to the plots. Here the arguments `h = 0` and `v = 0` add horizontal and vertical lines to the plot. This is particularly useful for biplots presented as separate row and column plots since it aids the visual superimposition of plots when interpretating the results. The `lty = 2` argument specifies the line type `lty` to be used – dashed in this case.