Chapter 4
Data Visualization

In this chapter, we use a broad range of simulations and hands-on activities to highlight some of the basic data visualization techniques using R. A brief discussion of alternative visualization methods is followed by demonstrations of histograms, density, pie, jitter, bar, line and scatter plots, as well as strategies for displaying trees, more general graphs, and 3D surface plots. Many of these are also used throughout the textbook in the context of addressing the graphical needs of specific case-studies.

It is practically impossible to cover all options of every different visualization routine. Readers are encouraged to experiment with each visualization type, change input data and parameters, explore the function documentation using R-help (e.g., `?plot`), and search online for new `R` visualization packages and new functionality, which are continuously being developed.

We will cover (1) one specific classification of visualization methods, (2) composition (e.g., density, histogram), comparison (e.g., jitter, bar, correlation) and relationship (e.g., line) plots, (3) 2D kernel density and 3D surface plots, and (4) 3D and 4D visualization of solids, (hyper)volumes.

4.1 Common Questions

- What exploratory visualization techniques are available to graphically interrogate my specific data?
- How do we examine paired associations and correlations in a multivariate dataset?

I. D. Dinov, *Data Science and Predictive Analytics*,
https://doi.org/10.1007/978-3-319-72347-1_4

4.2 Classification of Visualization Methods

Scientific data-driven or simulation-driven visualization methods are hard to classify. The following list of criteria can be used to characterize alternative data visualization strategies:

- **Data Type**: structured/unstructured, small/large, complete/incomplete, time/space, ASCII/binary, Euclidean/non-Euclidean, etc.
- **Task type**: Task type is one of the aspects considered in classification of visualization techniques, which provides a means of interaction between the researcher, the data, and the display software/platform
- **Scalability**: Visualization techniques are subject to some limitations, such as the amount of data that a particular technique can exhibit
- **Dimensionality**: Visualization techniques can also be classified according to the number of attributes
- **Positioning and Attributes**: the distribution of attributes on the chart may affect the interpretation of the display representation, e.g., correlation analysis, where the relative distance among the plotted attributes is relevant for observation
- **Investigative Need**: the specific scientific question or exploratory interest may also determine the type of visualization:
  - Examining the composition of the data
  - Exploring the distribution of the data
  - Contrasting or comparing several data elements, relations, association
  - Unsupervised exploratory data mining.

Also, we have the following table for common data visualization methods according to task types (Fig. 4.1):

We introduce common data visualization methods according to this classification criterion, albeit this is not a unique or even broadly agreed upon ontological characterization of exploratory data visualization.

4.3 Composition

In this section, we will see composition plots for different types of variables and data structures.

4.3.1 Histograms and Density Plots

One of the first few graphs we learn is a histogram plot. In R, the command `hist()` is applied to a vector of values and used for plotting histograms. The famous nineteenth century statistician Karl Pearson introduced histograms as graphical

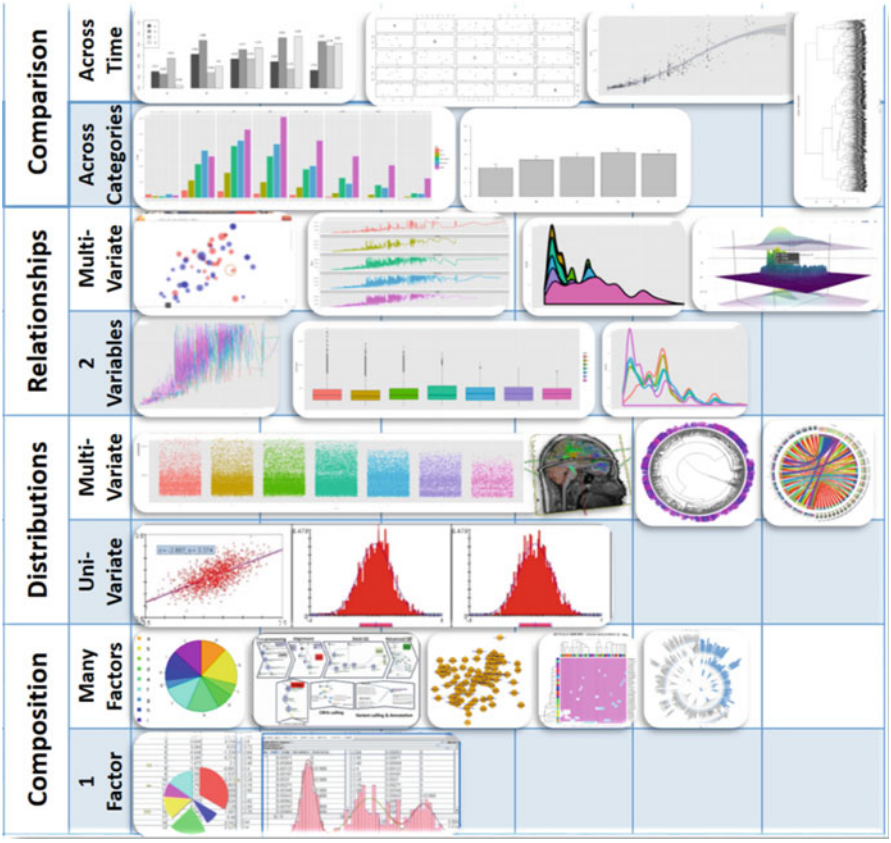

Fig. 4.1 Schematic depiction of a hierarchical classification of different visualization methods

representations of the distribution of a sample of numeric data. The histogram plot uses the data to infer and display the probability distribution of the underlying population that the data is sampled from. Histograms are constructed by selecting a certain number of bins covering the range of values of the observed process. Typically, the number of bins for a data array of size N should be equal to $\sqrt{N}$. These bins form a partition (disjoint and covering sets) of the range. Finally, we compute the relative frequency representing the number of observations that fall within each bin interval. The histogram just plots a piece-wise step-function defined over the union of the bin interfaces whose height equals the observed relative frequencies (Fig. 4.2).

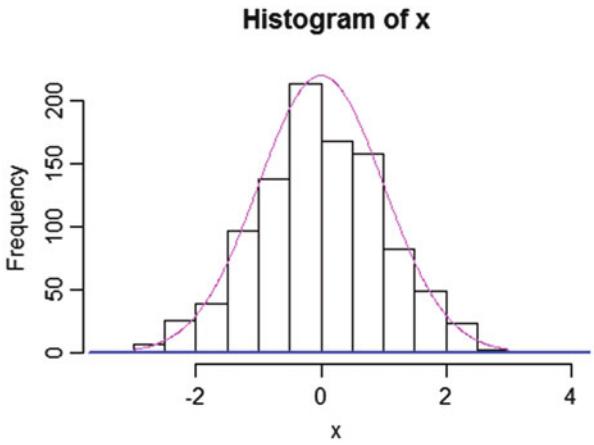

Fig. 4.2 Overlay of Normal distribution histogram and density curve plot

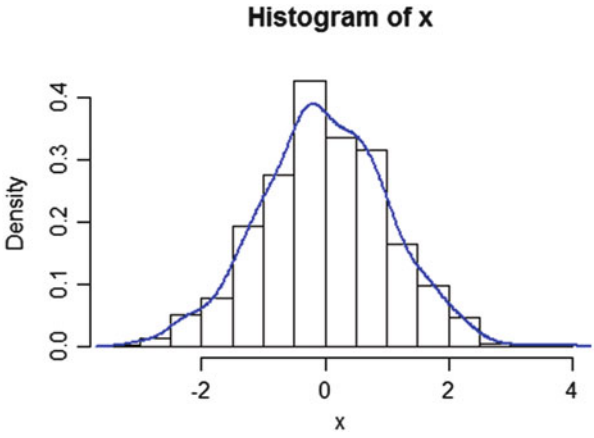

Fig. 4.3 Overlay of histogram plot and a density curve estimate

```
set.seed(1)
x<-rnorm(1000)
hist(x, freq=T, breaks = 10)
lines(density(x), lwd=2, col="blue")
t <- seq(-3, 3, by=0.01)
lines(t, 550*dnorm(t,0,1), col="magenta") # add the theoretical density line
```

Here, `freq=T` shows the frequency for each x value and `breaks` controls for number of bars in our histogram.

The shape of the last histogram we drew is very close to a Normal distribution (because we sampled from this distribution by `rnorm`). We can add a density line to the histogram (Fig. 4.3).

```
hist(x, freq=F, breaks = 10)
lines(density(x), lwd=2, col="blue")
```

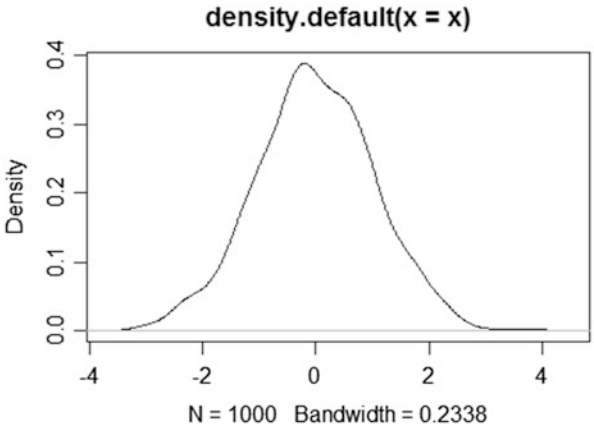

Fig. 4.4 Direct plot of the estimated Normal distribution density curve

We used the option `freq=F` to make the y axis represent the "relative frequency", or "density". We can also use `plot(density(x))` to draw the density plot by itself (Fig. 4.4).

```
plot(density(x))
```

4.3.2 Pie Chart

We are all very familiar with pie charts that show us the components of a big "cake". Although pie charts provide effective simple visualization in certain situations, it may also be difficult to compare segments within a pie chart or across different pie charts. Other plots like bar chart, box or dot plots may be attractive alternatives.

We will use the Letter Frequency Data on SOCR website to illustrate the use of pie charts.

```
library(rvest)

wiki_url <- read_html("http://wiki.socr.umich.edu/index.php/SOCR_LetterFrequ
encyData")
html_nodes(wiki_url, "#content")

## {xml_nodeset (1)}
## [1] <div id="content" class="mw-body-primary" role="main">\n\t<a id="top
...

letter<- html_table(html_nodes(wiki_url, "table")[[1]])
summary(letter)
```

```
##     Letter            English           French            German
##  Length:27          Min.   :0.00000   Min.   :0.00000   Min.   :0.00000
##  Class :character   1st Qu.:0.01000   1st Qu.:0.01000   1st Qu.:0.01000
##  Mode  :character   Median :0.02000   Median :0.03000   Median :0.03000
##                     Mean   :0.03667   Mean   :0.03704   Mean   :0.03741
##                     3rd Qu.:0.06000   3rd Qu.:0.06500   3rd Qu.:0.05500
##                     Max.   :0.13000   Max.   :0.15000   Max.   :0.17000
##     Spanish          Portuguese        Esperanto          Italian
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.01000   1st Qu.:0.00500   1st Qu.:0.01000   1st Qu.:0.00500
##  Median :0.03000   Median :0.03000   Median :0.03000   Median :0.03000
##  Mean   :0.03815   Mean   :0.03778   Mean   :0.03704   Mean   :0.03815
##  3rd Qu.:0.06000   3rd Qu.:0.05000   3rd Qu.:0.06000   3rd Qu.:0.06000
##  Max.   :0.14000   Max.   :0.15000   Max.   :0.12000   Max.   :0.12000
##     Turkish           Swedish           Polish          Toki_Pona
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.01000   1st Qu.:0.01000   1st Qu.:0.01500   1st Qu.:0.00000
##  Median :0.03000   Median :0.03000   Median :0.03000   Median :0.03000
##  Mean   :0.03667   Mean   :0.03704   Mean   :0.03704   Mean   :0.03704
##  3rd Qu.:0.05500   3rd Qu.:0.05500   3rd Qu.:0.04500   3rd Qu.:0.05000
##  Max.   :0.12000   Max.   :0.10000   Max.   :0.20000   Max.   :0.17000
##      Dutch            Avgerage
##  Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.01000   1st Qu.:0.01000
##  Median :0.02000   Median :0.03000
##  Mean   :0.03704   Mean   :0.03741
##  3rd Qu.:0.06000   3rd Qu.:0.06000
##  Max.   :0.19000   Max.   :0.12000
```

We can try to plot the frequency for the first ten English letters. The left hand side shows a table made by the function `legend` (Fig. 4.5).

```
par(mfrow=c(1, 2))
pie(letter$English[1:10], labels=letter$Letter[1:10], col=rainbow(10, start=
0.1, end=0.8), clockwise=TRUE, main="First 10 Letters Pie Chart")
pie(letter$English[1:10], labels=letter$Letter[1:10], col=rainbow(10, start=
0.1, end=0.8), clockwise=TRUE, main="First 10 Letters Pie Chart")
legend("topleft", legend=letter$Letter[1:10], cex=1.3, bty="n", pch=15, pt.c
ex=1.8, col=rainbow(10, start=0.1, end=0.8), ncol=1)
```

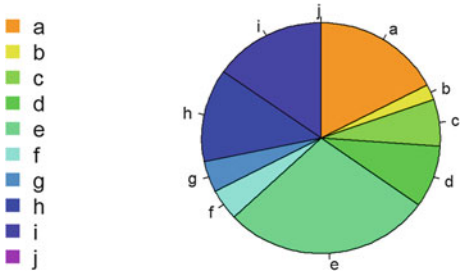

Fig. 4.5 Pie chart showing the frequency of English use of the first ten Latin letters (a–j)

The input type for `pie()` is a vector of non-negative numerical quantities. In the `pie` function, we list the data that we are going to use (positive and numeric), the labels for each of them, and the colors we want to use for each sector. In the `legend` function, we put the location in the first slot and use `legend` as the labels for the colors. Also `cex`, `bty`, `pch`, and `pt.cex` are all graphic parameters that we have talked about in Chap. 2.

More elaborate pie charts, using the Latin letter data, will be demonstrated using `ggplot` below (in the Appendix of this chapter).

4.3.3 Heat Map

Another common data visualization method is the `heat map`. Heat maps can help us visualize intuitively the individual values in a matrix. It is widely used in genetics research and financial applications.

We will illustrate the use of heat maps, based on a neuroimaging genetics case-study data about the association (p-values) of different brain regions of interest (ROIs) and genetic traits (SNPs) for Alzheimer's disease (AD) patients, subjects with mild cognitive impairment (MCI), and normal controls (NC). First, let's import the data into R. The data are 2D arrays where the rows represent different genetic SNPs, columns represent brain ROIs, and the cell values represent the strength of the SNP-ROI association, a probability value (smaller p-values indicate stronger neuroimaging-genetic associations).

```
AD_Data <- read.table("https://umich.instructure.com/files/330387/download?d
ownload_frd=1", header=TRUE, row.names=1,  sep=",", dec=".")
MCI_Data <- read.table("https://umich.instructure.com/files/330390/download?
download_frd=1", header=TRUE, row.names=1,  sep=",", dec=".")
NC_Data <- read.table("https://umich.instructure.com/files/330391/download?d
ownload_frd=1", header=TRUE, row.names=1,  sep=",", dec=".")
```

Then, we load the R packages we need for heat maps (use `install.packages("package name")` first if you have not previously install them).

```
require(graphics)
require(grDevices)
library(gplots)
```

We can convert the datasets into matrices.

```
AD_mat  <- as.matrix(AD_Data); class(AD_mat) <- "numeric"
MCI_mat  <- as.matrix(MCI_Data); class(MCI_mat) <- "numeric"
NC_mat  <- as.matrix(NC_Data); class(NC_mat) <- "numeric"
```

We may also want to set up the row (rc) and column (cc) colors for each cohort.

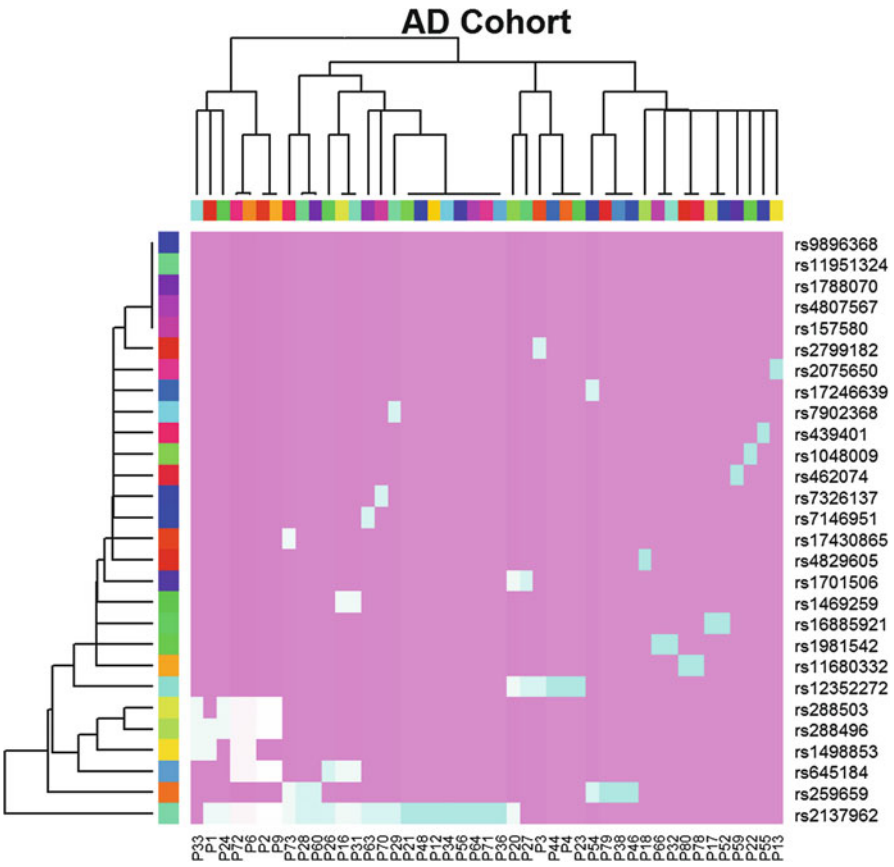

Fig. 4.6 Hierarchically clustered heatmap for the Alzheimer's disease (AD) cohort of the dementia study. The rows indicate the unique SNP reference sequence (rs) IDs and the columns index specific brain regions of interest (ROIs) that are associated with the genomic biomarkers (rows)

```
rcAD <- rainbow(nrow(AD_mat), start = 0, end = 1.0); ccAD<-rainbow(ncol(AD_m
at), start = 0, end = 1.0)
rcMCI <- rainbow(nrow(MCI_mat), start = 0, end=1.0); ccMCI<-rainbow(ncol(MCI
_mat), start=0, end=1.0)
rcNC <- rainbow(nrow(NC_mat), start = 0, end = 1.0); ccNC<-rainbow(ncol(NC_m
at), start = 0, end = 1.0)
```

Finally, we can plot the heat maps by specifying the input type of `heatmap()` to be a numeric matrix (Figs. 4.6, 4.7, and 4.8).

```
hvAD <- heatmap(AD_mat, col=cm.colors(256), scale="column", RowSideColors =
rcAD, ColSideColors = ccAD, margins = c(2, 2), main="AD Cohort")
```

```
hvMCI <- heatmap(MCI_mat, col = cm.colors(256), scale = "column", RowSideCol
ors = rcMCI, ColSideColors = ccMCI, margins = c(2, 2), main="MCI Cohort")
```

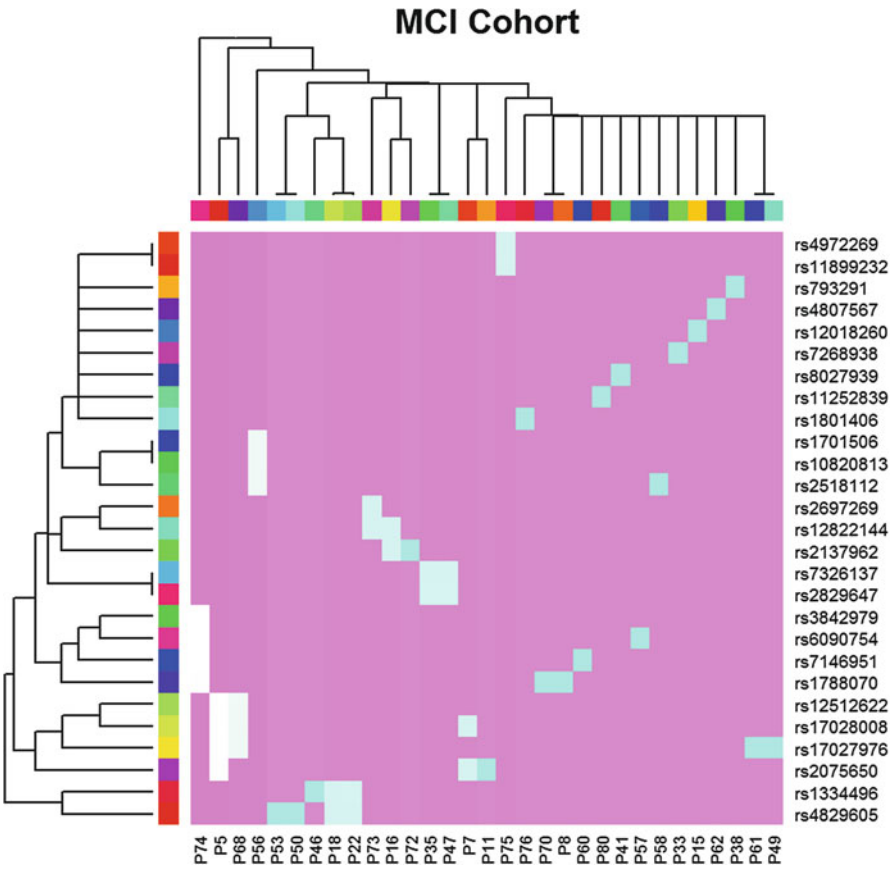

Fig. 4.7 Hierarchically clustered heatmap for the Mild Cognitive Impairment (MCI) cohort

```
hvNC <- heatmap(NC_mat, col=cm.colors(256), scale="column", RowSideColors =
rcNC, ColSideColors = ccNC, margins = c(2, 2), main="NC Cohort")
```

In the `heatmap()` function, the first argument provides the input matrix we want to use. `col` is the color scheme; `scale` is a character indicating if the values should be centered and scaled in either the row direction or the column direction, or none ("row", "column", and "none"); `RowSideColors` and `ColSideColors` creates the color names for horizontal side bars.

The differences between the AD, MCI, and NC heat maps are suggestive of variations of genetic traits or alternative brain regions that may be affected in the three clinically different cohorts.

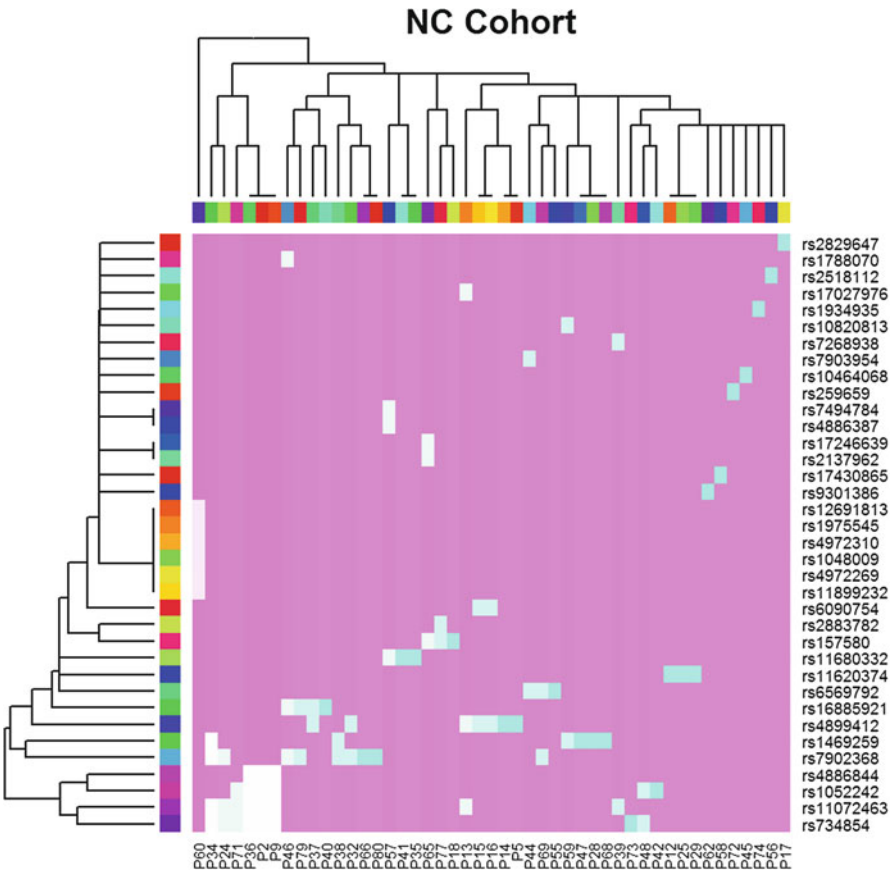

Fig. 4.8 Hierarchically clustered heatmap for the healthy normal controls (NC) cohort

4.4 Comparison

Plots used for comparing different individuals, groups of subjects, or multiple units represent another set of popular exploratory visualization tools.

4.4.1 Paired Scatter Plots

Scatter plots use the 2D Cartesian plane to display a pair of variables. 2D points represent the values of the two variables corresponding to the two coordinate axes. The position of each 2D point on is determined by the values of the first and second variables, which represent the horizontal and vertical axes. If no clear dependent

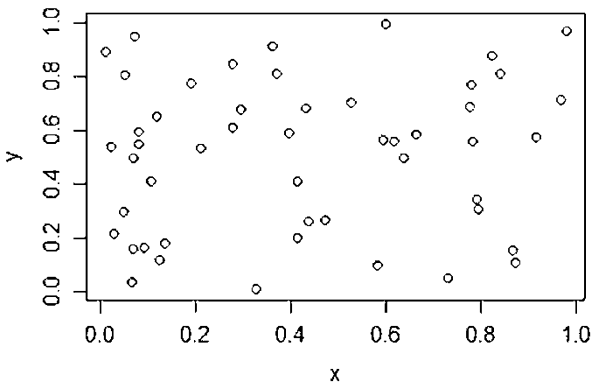

Fig. 4.9 Scatter plot of bivariate uniform process

variable exists, either variable can be plotted on the *X* axis and the corresponding scatter plot will illustrate the degree of correlation (not necessarily causation) between the two variables.

Basic scatter plots can be plotted by function `plot(x, y)` (Fig. 4.9).

```
x<-runif(50)
y<-runif(50)
plot(x, y, main="Scatter Plot")
```

`qplot()` is another way to display elaborate scatter plots. We can manage the colors and sizes of dots. The input type for `qplot()` is a data frame. In the following example, larger *x* will have larger dot sizes. We also grouped the data as ten points per group (Fig. 4.10).

```
library(ggplot2)
cat <- rep(c("A", "B", "C", "D", "E"), 10)
plot.1 <- qplot(x, y, geom="point", size=5*x, color=cat, main="GGplot with R
elative Dot Size and Color")
print(plot.1)
```

Now, let's draw a paired scatter plot with three variables. The input type for `pairs()` function is a matrix or data frame (Fig. 4.11).

```
z<-runif(50)
pairs(data.frame(x, y, z))
```

We can see that variable names are on the diagonal of this scatter plot matrix. Each plot uses the column variable as its X-axis and row variable as its Y-axis.

Let's see a real word data example. First, we can import the Mental Health Services Survey Data into R, which is on the case-studies website.

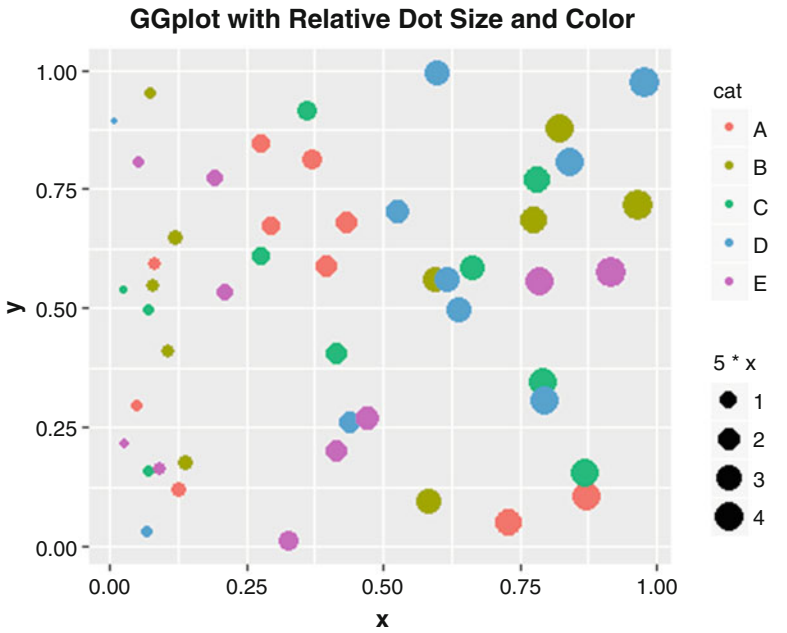

Fig. 4.10 Simulated bubble plot depicting four variable features represented as x and y axes, size and color

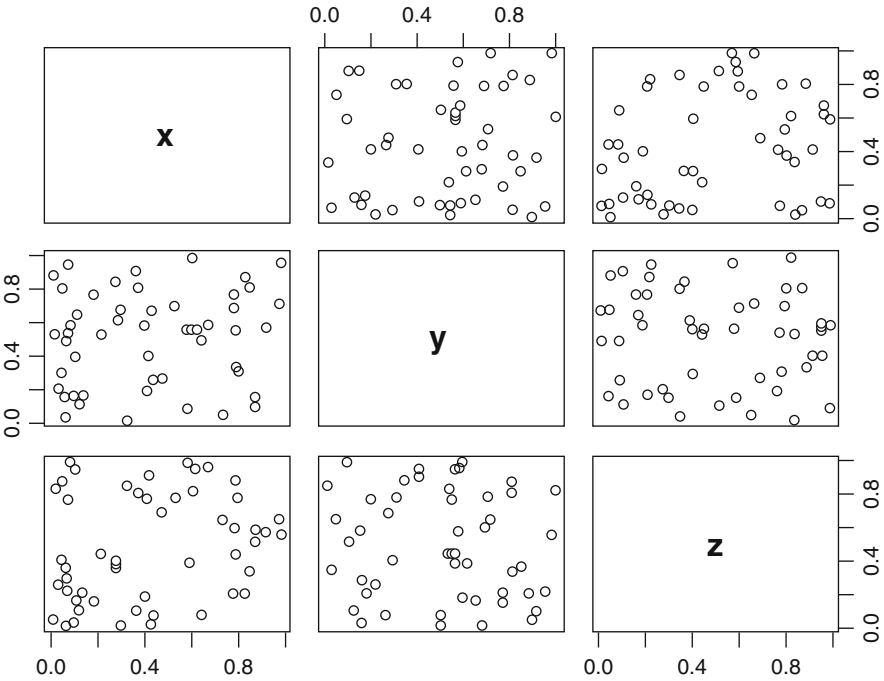

Fig. 4.11 A pairs plot depicts the bivariate relations in multivariate datasets

```
data1 <- read.table('https://umich.instructure.com/files/399128/download?dow
nload_frd=1', header=T)
head(data1)
```

```
##          STFIPS majorfundtype FacilityType Ownership Focus PostTraum GLBT
## 1     southeast             1            5         2     1         0    0
## 2     southeast             3            5         3     1         0    0
## 3     southeast             1            6         2     1         1    1
## 4    greatlakes            NA            2         2     1         0    0
## 5 rockymountain             1            5         2     3         0    0
## 6       mideast            NA            2         2     1         0    0
##   num qual supp
## 1   5   NA   NA
## 2   4   15    4
## 3   9   15   NA
## 4   7   14    6
## 5   9   18   NA
## 6   8   14   NA
```

```
attach(data1)
```

From the `head()` output, we observe that there are a lot of *NA*'s in the dataset. `pairs` automatically deals with this problem (Figs. 4.12 and 4.13).

```
plot(data1[, 9], data1[, 10], pch=20, col="red", main="qual vs supp")
```

```
pairs(data1[, 5:10])
```

Figure 4.12 represents just one of the plots shown in the collage on Fig. 4.13. We can see that `Focus` and `PostTraum` have no relationship - `Focus` can equal to 3 or 1 in either of the `PostTraum` values (0 or 1). On the other hand, larger `supp` tends to correspond to larger `qual` values.

To see this trend we can also make a plot using `qplot` function. This allow us to add a smooth model curve forecasting a possible trend (Fig. 4.14).

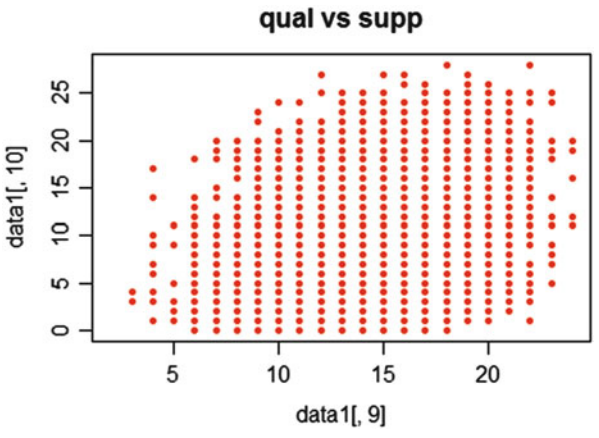

Fig. 4.12 Each of the bivariate plots in a pairs plot collage may be zoomed up and explored further

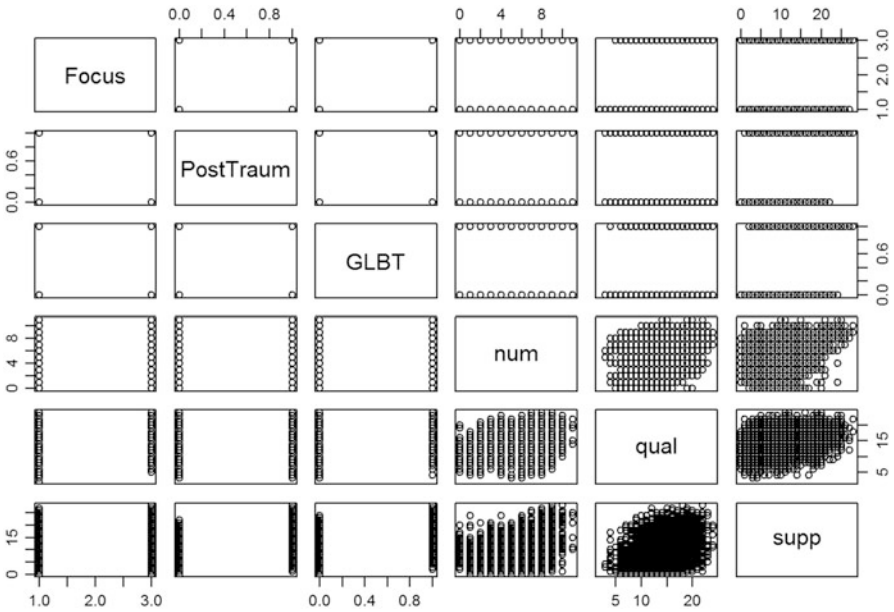

Fig. 4.13 A more elaborate 6D pairs plot showing the type and scale of each variable and their bivariate relations

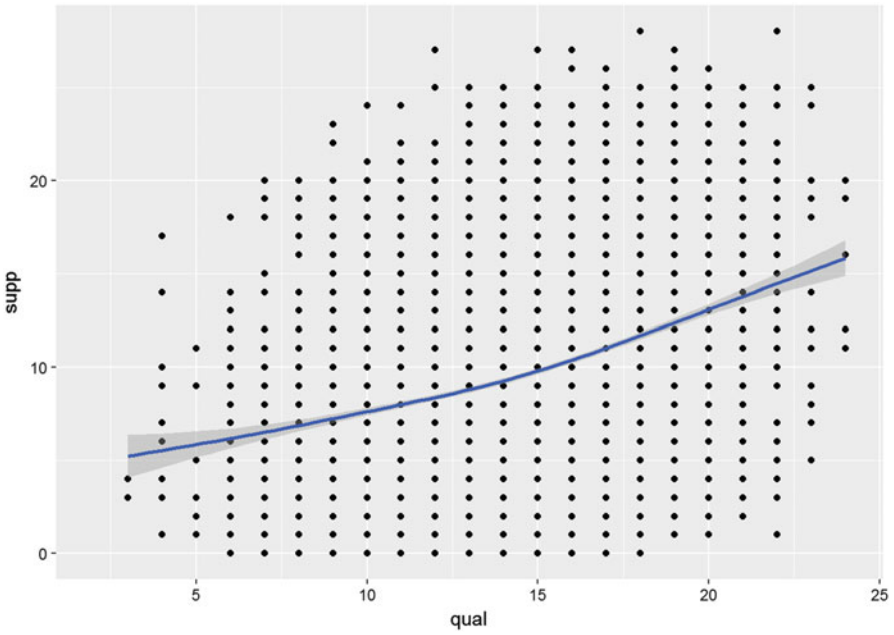

Fig. 4.14 Plotting the bivariate trend along with its confidence limits

```
plot.2 <- qplot(qual, supp, data = data1, geom = c("point", "smooth"))
print(plot.2)
```

You can also use the human height and weight dataset or the knee pain dataset to illustrate some interesting scatter plots.

4.4.2 *Jitter Plot*

Jitter plots can help us deal with the complexity issues when we have many points in the data. The function we will be using is in package `ggplot2` is called `position_jitter()`.

Let's use the earthquake data for this example. We will compare the differences with and without the `position_jitter()` function (Figs. 4.15 and 4.16).

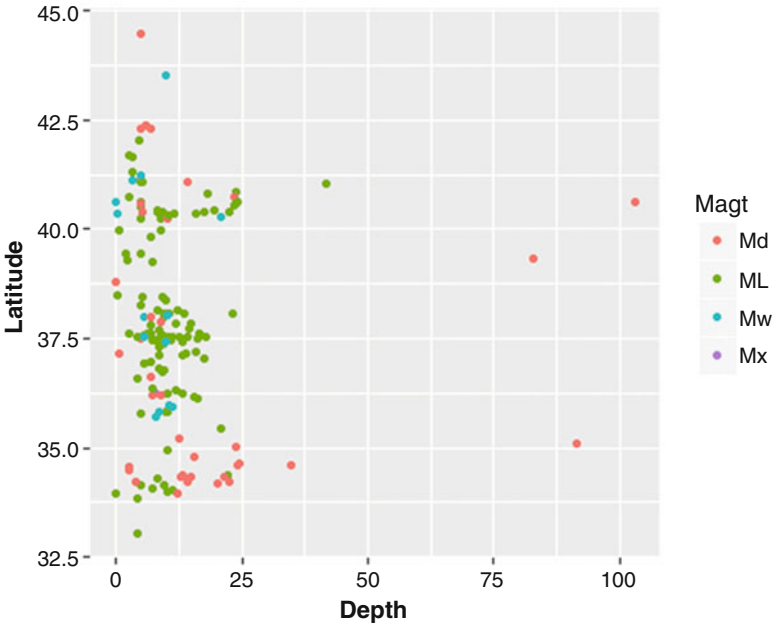

Fig. 4.15 Jitter plot of magnitude type against depth and latitude (Earthquake dataset)

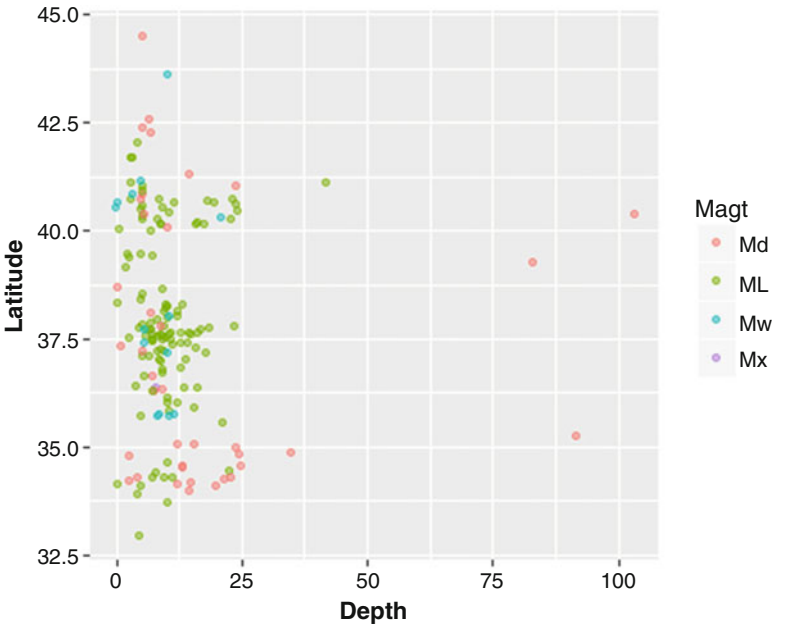

Fig. 4.16 A lower opacity jitter plot of magnitude type against depth and latitude

```
# library("xml2"); library("rvest")
wiki_url <- read_html("http://wiki.socr.umich.edu/index.php/SOCR_Data_Dinov_
021708_Earthquakes")
html_nodes(wiki_url, "#content")

## {xml_nodeset (1)}
## [1] <div id="content" class="mw-body-primary" role="main">\n\t<a id="top
...

earthquake <- html_table(html_nodes(wiki_url, "table")[[2]])

plot6.1<-ggplot(earthquake, aes(Depth, Latitude, group=Magt, color=Magt))+ge
om_point()
plot6.2<-ggplot(earthquake, aes(Depth, Latitude, group=Magt, color=Magt))+ge
om_point(position = position_jitter(w = 0.3, h = 0.3), alpha=0.5)
print(plot6.1)
```

```
print(plot6.2)
```

Note that with option `alpha=0.5` the "crowded" places are darker than the places with only one data point. Sometimes, we need to add text to these points, i.e., add label in `aes` or add `geom_text`. The result may look messy (Fig. 4.17).

```
ggplot(earthquake, aes(Depth, Latitude, group=Magt,
       color=Magt,label=rownames(earthquake)))+
geom_point(position = position_jitter(w = 0.3, h = 0.3), alpha=0.5)+
geom_text()
```

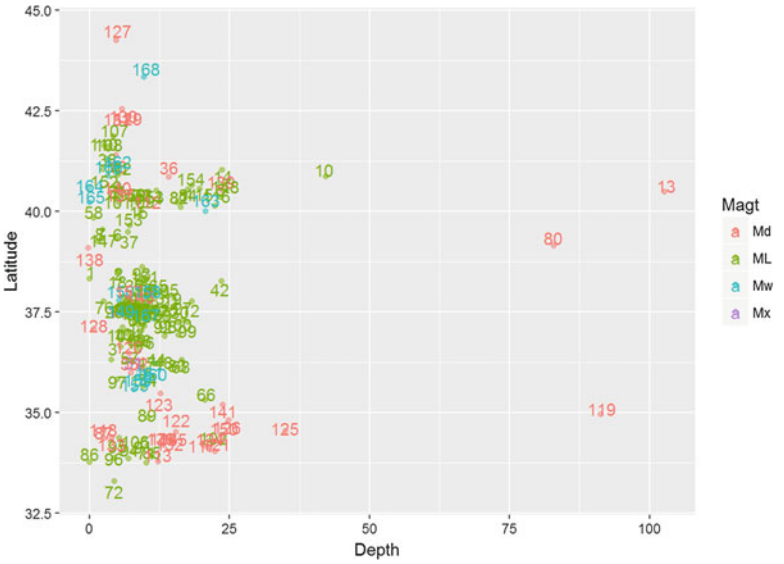

Fig. 4.17 Another version of the jitter plot of magnitude type explicitly listing the Earthquake ID label

Let's try to fix the overlap of points and labels. We need to add `check_overlap` in `geom_text` and adjust the positions of the text labels with respect to the points (Figs. 4.18 and 4.19).

```
ggplot(earthquake, aes(Depth, Latitude, group=Magt, color=Magt,
      label=rownames(earthquake)))+
geom_point(position = position_jitter(w = 0.3, h = 0.3), alpha=0.5)+
geom_text(check_overlap = T,vjust = 0, nudge_y = 0.5, size = 2,angle = 45)
```

```
# Or you can simply use the text to denote the positions of points.
ggplot(earthquake, aes(Depth, Latitude, group=Magt, color=Magt,
label=rownames(earthquake)))+
geom_text(check_overlap = T,vjust = 0, nudge_y = 0, size = 3,angle = 45)
```

4.4.3 Bar Plots

Bar plots, or bar charts, represent group data with rectangular bars. There are many variants of bar charts for comparison among categories. Typically, either horizontal or vertical bars are used where one of the axes shows the compared categories and the other axis represents a discrete value. It's possible, and sometimes desirable, to plot bar graphs including bars clustered by groups.

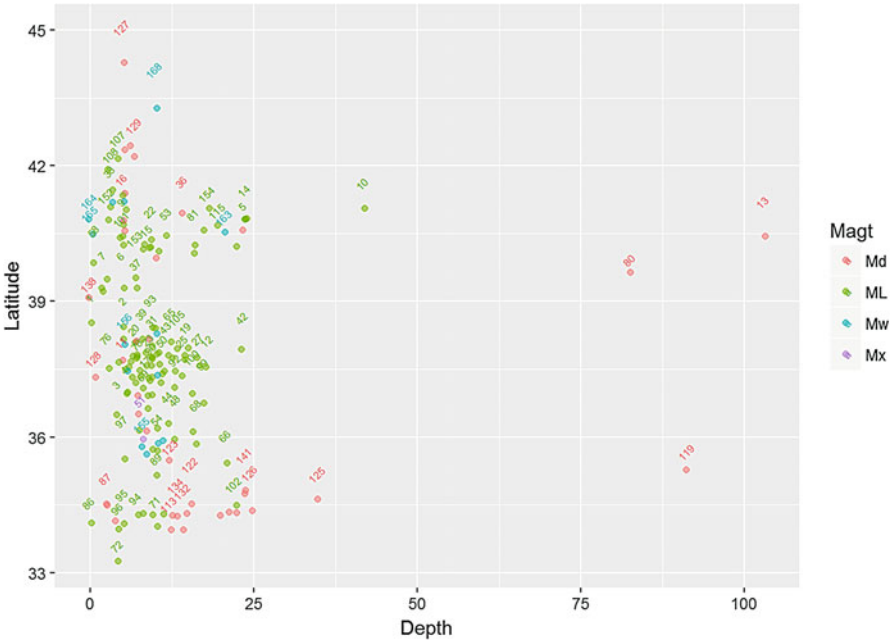

Fig. 4.18 Yet another version of the previous jitter plot illustrating label specifications

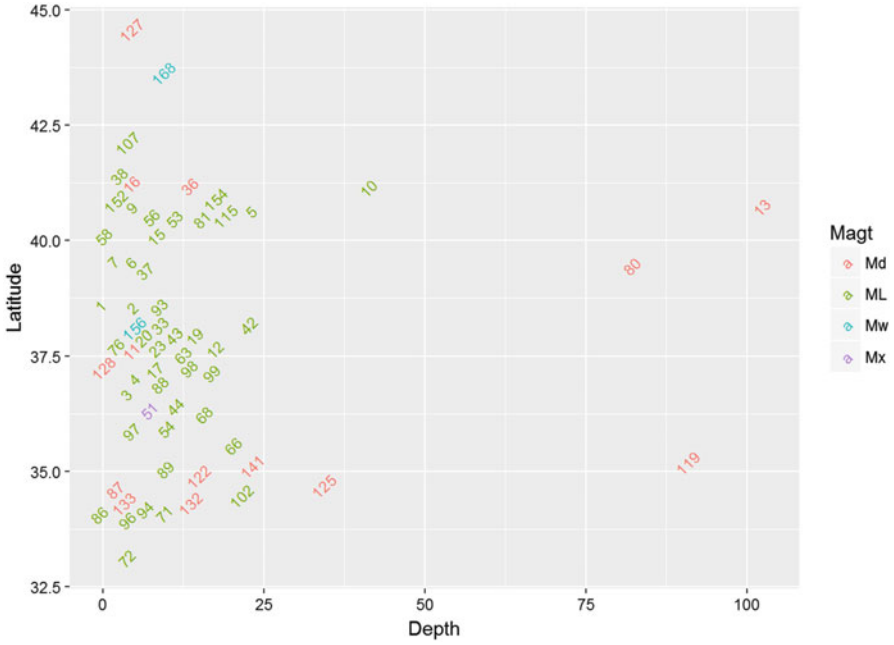

Fig. 4.19 This jitter plot suppresses the scatter point bubbles in favor of ID labels

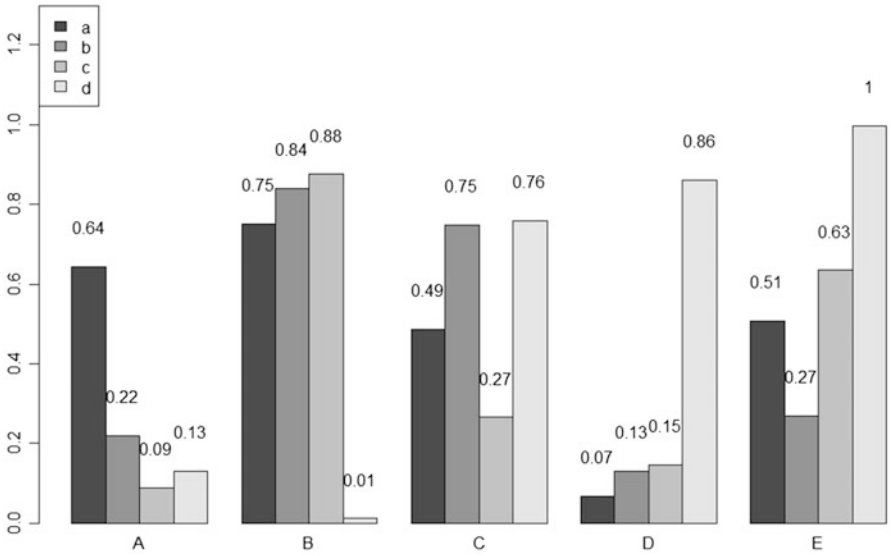

Fig. 4.20 Example of a labeled boxplot using simulated data with grouping categorical labels

In R, we have the `barplot()` function to generate bar plots. The input for `barplot()` is either a vector or a matrix (Fig. 4.20).

```
x <- matrix(runif(50), ncol=5, dimnames=list(letters[1:10], LETTERS[1:5]))
x

##            A          B         C           D         E
## a 0.64397479 0.75069788 0.4859278 0.068299279 0.5069665
## b 0.21981304 0.84028392 0.7489431 0.130542241 0.2694441
## c 0.08903728 0.87540556 0.2656034 0.146773063 0.6346498
## d 0.13075121 0.01106876 0.7586781 0.860316695 0.9976566
## e 0.87938851 0.04156918 0.1960069 0.949276015 0.5050743
## f 0.65204025 0.21135891 0.3774320 0.896443296 0.9332330
## g 0.02814806 0.72618285 0.5603189 0.113651731 0.1912089
## h 0.13106307 0.79411904 0.4526415 0.793385952 0.4847625
## i 0.15759514 0.63369297 0.8861631 0.004317772 0.6341256
## j 0.47347613 0.14976052 0.5887866 0.698139910 0.2023031

barplot(x[1:4, ], ylim=c(0, max(x[1:4, ])+0.3), beside=TRUE, legend.text = l
etters[1:4],
    args.legend = list(x = "topleft"))
text(labels=round(as.vector(as.matrix(x[1:4, ])), 2), x=seq(1.5, 21, by=1) +
rep(c(0, 1, 2, 3, 4), each=4), y=as.vector(as.matrix(x[1:4, ]))+0.1)
```

It may require some creativity to add value labels on each bar. First, let's specify the location on the x-axis `x=seq(1.5, 21, by=1)+rep(c(0, 1, 2, 3, 4), each=4)`. In this example there are 20 bars. The x location for middle of the first bar is 1.5 (there is one empty space before the first bar). The middle of the last bar is

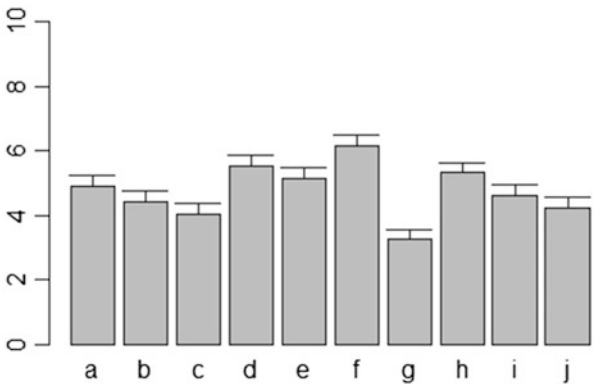

Fig. 4.21 Statistical barplot showing point-estimates and their error limits (simulated data)

24.5. `seq(1.5, 21, by=1)` starts at 1.5 and creates 20 bars that end with `x=21`. Then, we use `rep(c(0, 1, 2, 3, 4), each=4)` to add 0 to the first group, 1 to the second group, and so forth. Thus, we have the desired positions on the x-axis. The y-axis positions are obtained just by adding 0.1 to each bar height.

We can also add standard deviations to the means on the bars. To do this, we need to use the `arrows()` function and the option `angle = 90`, the result is shown on Fig. 4.21.

```
bar <- barplot(m <- rowMeans(x) * 10, ylim=c(0, 10))
stdev <- sd(t(x[1:4, ]))
arrows(bar, m, bar, m + stdev, length=0.15, angle = 90)
```

Let's look at a more complex example. We will utilize the dataset Case_04_ChildTrauma for illustration. This case study examines associations between post-traumatic psychopathology and service utilization by trauma-exposed children.

```
data2 <- read.table('https://umich.instructure.com/files/399129/download?dow
nload_frd=1', header=T)
attach(data2)
head(data2)

##   id sex age ses  race traumatype ptsd dissoc service
## 1  1   1   6   0 black   sexabuse    1      1      17
## 2  2   1  14   0 black   sexabuse    0      0      12
## 3  3   0   6   0 black   sexabuse    0      1       9
## 4  4   0  11   0 black   sexabuse    0      1      11
## 5  5   1   7   0 black   sexabuse    1      1      15
## 6  6   0   9   0 black   sexabuse    1      0       6
```

We have two character variables. Our goal is to draw a bar plot comparing the means of `age` and `service` among different races in this study, and we want to add standard deviation to each bar. The first thing to do is to delete the two character columns. Remember, the input for `barplot()` is a numerical vector or a matrix.

However, we will need race information for the categorical classification. Thus, we will store race in a different variable.

```
data2.sub <- data2[, c(-5, -6)]
data2<-data2[, -6]
```

We are now ready to separate the groups and compute the group means.

```
data2.matrix <- as.data.frame(data2)
Blacks <- data2[which(data2$race=="black"), ]
Other <- data2[which(data2$race=="other"), ]
Hispanic <- data2[which(data2$race=="hispanic"), ]
White <- data2[which(data2$race=="white"), ]

B <- c(mean(Blacks$age), mean(Blacks$service))
O <- c(mean(Other$age), mean(Other$service))
H <- c(mean(Hispanic$age), mean(Hispanic$service))
W <- c(mean(White$age), mean(White$service))

x <- cbind(B, O, H, W)
x

##          B     O    H        W
## [1,] 9.165  9.12 8.67 8.950000
## [2,] 9.930 10.32 9.61 9.911667
```

Now that we have a numerical matrix for the means, we can compute a second order statistics, standard deviation, and plot it along with the means, to illustrate the amount of dispersion for each variable (Fig. 4.22).

```
bar <- barplot(x, ylim=c(0, max(x)+2.0), beside=TRUE,
legend.text = c("age", "service") ,  args.legend = list(x = "right"))
text(labels=round(as.vector(as.matrix(x)), 2),

    x=seq(1.4, 21, by=1.5), #y=as.vector(as.matrix(x[1:2, ]))+0.3)
    y=11.5)

m <- x; stdev <- sd(t(x))
arrows(bar, m, bar, m + stdev, length=0.15, angle = 90)
```

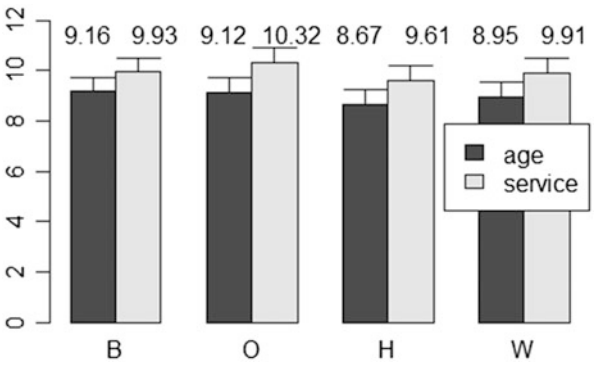

Fig. 4.22 Barplot showing point-estimates and their error limits (Child Trauma dataset)

Here, we want the y margin to be a little higher than the greatest value (`ylim=c(0, max(x)+2.0)`) because we need to leave space for value labels. The plot shows that Hispanic trauma-exposed children may be younger, in terms of average age, and less likely to utilize services like primary care, emergency room, outpatient therapy, outpatient psychiatrist, etc.

Another way to plot bar plots is to use `ggplot()` in the ggplot package. This kind of bar plot is quite different from the one we introduced previously. It displays the counts of character variables rather than the means of numerical variables. It takes the values from a `data.frame`. Unlike `barplot()`, drawing bar plots using `ggplot2` requires that the character variables remain in the original data frame (Fig. 4.23).

```
library(ggplot2)
data2 <- read.table('https://umich.instructure.com/files/399129/download?dow
nload_frd=1', header=T)
bar1 <- ggplot(data2, aes(race, fill=race)) + geom_bar()+
     facet_grid(. ~ traumatype)
print(bar1)
```

This plot helps us compare the occurrence of different types of child-trauma among different races.

4.4.4 Trees and Graphs

In general, a graph is an ordered pair $G = (V, E)$ of vertices (V), i.e., nodes or points, and edges (E), arcs or lines connecting pairs of nodes in V. A tree is a special type of acyclic graph that does not include looping paths. Visualization of graphs is critical in many biosocial and health studies, and we will see menu such examples throughout this textbook.

In Chaps. 10 and 13, we will learn more about how to build tree models and other clustering methods, and in Chap. 23, we will discuss deep learning and neural networks, which have a direct graphical representation.

This section will be focused on displaying tree graphs. We will use a self-efficacy study, 02_Nof1_Data.csv, for this demonstration.

```
data3<- read.table("https://umich.instructure.com/files/330385/download?down
load_frd=1", sep=",", header = TRUE)
head(data3)

##   ID Day Tx SelfEff SelfEff25  WPSS SocSuppt PMss PMss3 PhyAct
## 1  1   1  1      33         8  0.97     5.00 4.03  1.03     53
## 2  1   2  1      33         8 -0.17     3.87 4.03  1.03     73
## 3  1   3  0      33         8  0.81     4.84 4.03  1.03     23
## 4  1   4  0      33         8 -0.41     3.62 4.03  1.03     36
## 5  1   5  1      33         8  0.59     4.62 4.03  1.03     21
## 6  1   6  1      33         8 -1.16     2.87 4.03  1.03      0
```

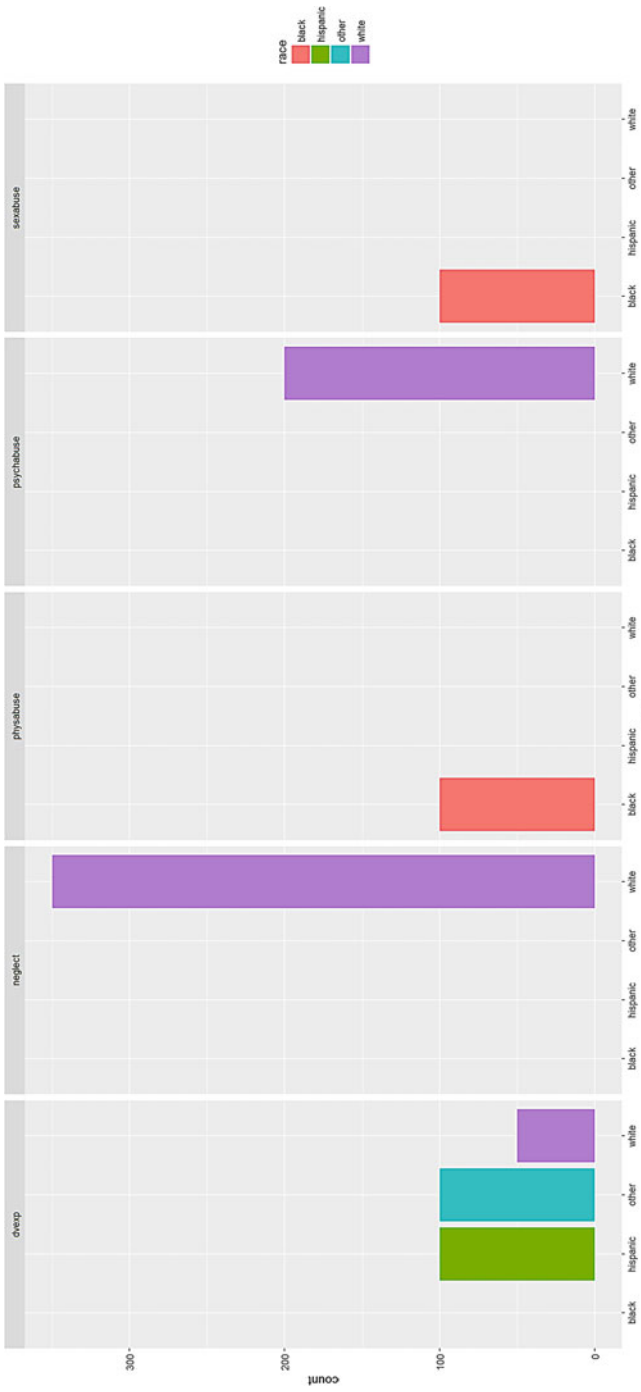

Fig. 4.23 Barplot of counts for different types of child trauma by race (color label)

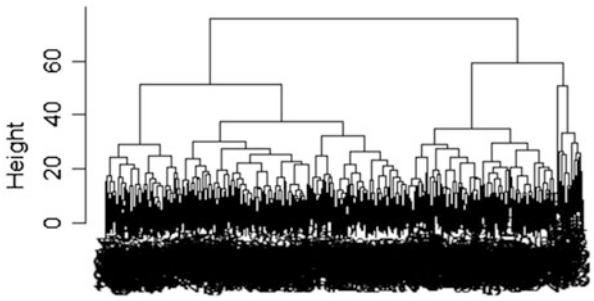

Fig. 4.24 Hierarchical clustering dendrogram of the 900 self-efficacy records of 30 participants including the nine features tracked over a month

We will use `hclust` to build the hierarchical cluster model. `hclust` takes only inputs that have dissimilarity structure as produced by `dist()`. Also, we use the `ave` method for agglomeration, see the tree graph on Fig. 4.24.

```
hc<-hclust(dist(data3), method='ave')
par (mfrow=c(1, 1))
plot(hc)
```

When we specify no limit for the maximum cluster groups, we will get the graph, on Fig. 4.24, which is not easy to interpret. Luckily, `cutree` will help us limit the cluster numbers. `cutree()` takes a `hclust` object and returns a vector of group indicators for all observations.

```
require(graphics)
mem <- cutree(hc, k = 10)

# mem; # to print the hierarchical tree labels for each case
# which(mem==5)  # to identify which cases belong to class/cluster 5
#To see the number of Subjects in which cluster:
# table(cutree(hc, k=5))
```

Usinf a for loop, we can get the mean of each variable within groups.

```
cent <- NULL
for(k in 1:10){
        cent <- rbind(cent, colMeans(data3[mem == k, , drop = FALSE]))
}
```

Now, we can plot a new tree graph with ten groups. Using the `members=table(mem)` option, the matrix is taken to be a dissimilarity matrix between clusters, instead of dissimilarities between singletons, and members represents the number of observations per cluster (Fig. 4.25).

```
hc1 <- hclust(dist(cent), method = "ave", members = table(mem))
plot(hc1, hang = -1, main = "Re-start from 10 clusters")
```

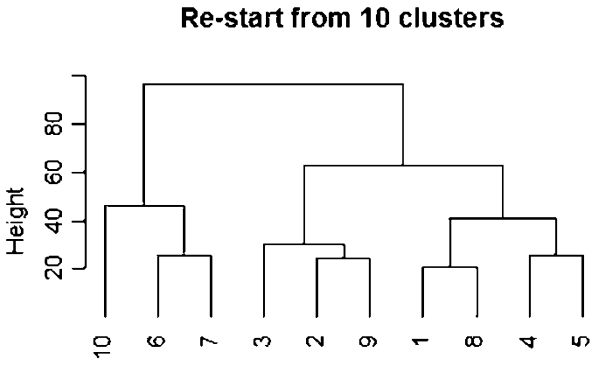

Fig. 4.25 A ten-cluster hierarchical dendrogram of the same dataset as before

4.4.5 Correlation Plots

The `corrplot` package enables the graphical display of correlation matrices, confidence intervals, and other plots showing matrix reordering. There are seven visualization methods (parameter methods) in `corrplot` package, named "circle", "square", "ellipse", "number", "shade", "color", and "pie".

Let's use 03_NC_SNP_ROI_Assoc_P_values.csv again to investigate the associations among SNPs using correlation plots.

The `corrplot()` function we will be using only accepts correlation matrices. So, we need to first obtain the correlation matrix of our data first using the `cor()` function.

```
# install.packages("corrplot")
library(corrplot)
NC_Associations_Data <- read.table("https://umich.instructure.com/files/330391/download?download_frd=1", header=TRUE, row.names=1,  sep=",", dec=".")
M <- cor(NC_Associations_Data)
M[1:10, 1:10]

##              P2          P5          P9         P12         P13
## P2   1.00000000 -0.05976123  0.99999944 -0.05976123  0.21245299
## P5  -0.05976123  1.00000000 -0.05976131 -0.02857143  0.56024640
## P9   0.99999944 -0.05976131  1.00000000 -0.05976131  0.21248635
## P12 -0.05976123 -0.02857143 -0.05976131  1.00000000 -0.05096471
## P13  0.21245299  0.56024640  0.21248635 -0.05096471  1.00000000
## P14 -0.05976123  1.00000000 -0.05976131 -0.02857143  0.56024640
## P15 -0.08574886  0.69821536 -0.08574898 -0.04099594  0.36613665
## P16 -0.08574886  0.69821536 -0.08574898 -0.04099594  0.36613665
## P17 -0.05976123 -0.02857143 -0.05976131 -0.02857143 -0.05096471
## P18 -0.05976123 -0.02857143 -0.05976131 -0.02857143 -0.05096471
##             P14         P15         P16         P17         P18
## P2  -0.05976123 -0.08574886 -0.08574886 -0.05976123 -0.05976123
## P5   1.00000000  0.69821536  0.69821536 -0.02857143 -0.02857143
## P9  -0.05976131 -0.08574898 -0.08574898 -0.05976131 -0.05976131
## P12 -0.02857143 -0.04099594 -0.04099594 -0.02857143 -0.02857143
## P13  0.56024640  0.36613665  0.36613665 -0.05096471 -0.05096471
## P14  1.00000000  0.69821536  0.69821536 -0.02857143 -0.02857143
## P15  0.69821536  1.00000000  1.00000000 -0.04099594 -0.04099594
## P16  0.69821536  1.00000000  1.00000000 -0.04099594 -0.04099594
## P17 -0.02857143 -0.04099594 -0.04099594  1.00000000 -0.02857143
## P18 -0.02857143 -0.04099594 -0.04099594 -0.02857143  1.00000000
```

We will illustrate alternative correlation plots using the `corrplot` function in Figs. 4.26, 4.27, 4.28, 4.29, 4.30, and 4.31.

```
corrplot(M, method = "circle", title = "circle", tl.cex = 0.5, tl.col = 'bla
ck', mar=c(1, 1, 1, 1))
# par specs c(bottom, left, top, right) which gives the margin size
specified in inches
corrplot(M, method = "square", title = "square", tl.cex = 0.5, tl.col =
'black', mar=c(1, 1, 1, 1))
corrplot(M, method = "ellipse", title = "ellipse", tl.cex = 0.5, tl.col =
'black', mar=c(1, 1, 1, 1))
corrplot(M, method = "pie", title = "pie", tl.cex = 0.5, tl.col = 'black',
mar=c(1, 1, 1, 1))
corrplot(M, type = "upper", tl.pos = "td",
         method = "circle", tl.cex = 0.5, tl.col = 'black',
         order = "hclust", diag = FALSE, mar=c(1, 1, 0, 1))
corrplot.mixed(M, number.cex = 0.6, tl.cex = 0.6)
```

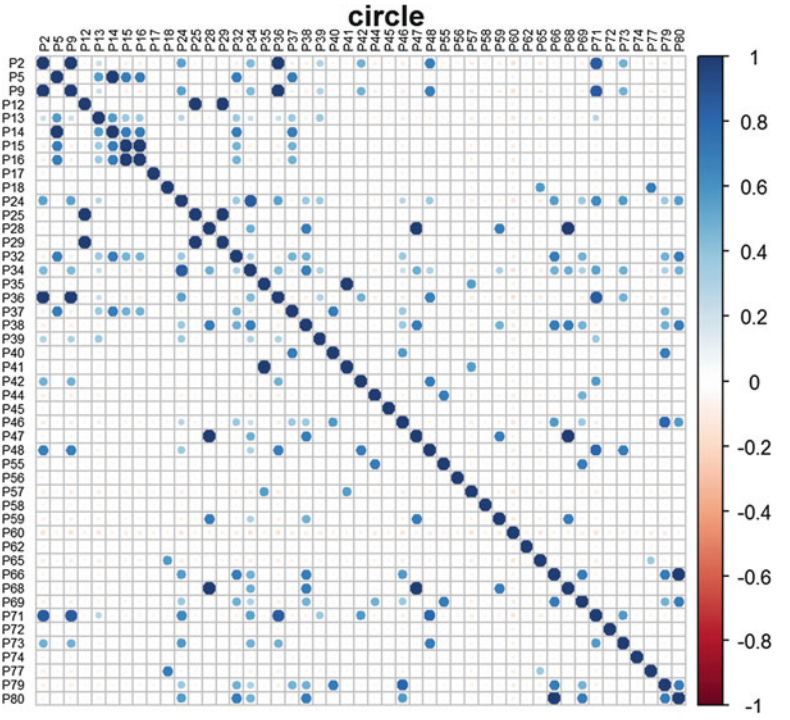

Fig. 4.26 Correlation plot of regional brain volumes of the healthy normal controls using circles

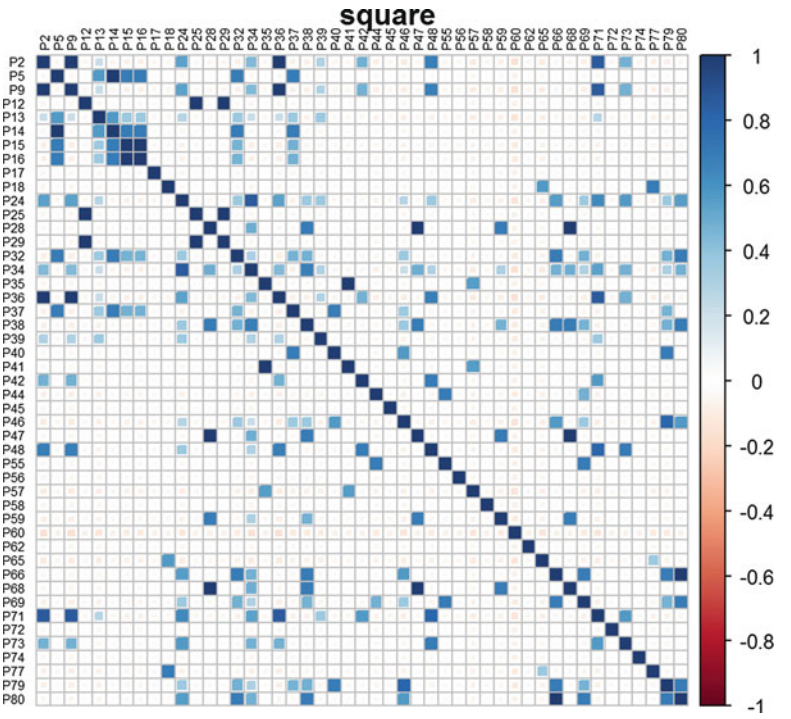

Fig. 4.27 The same correlation plot of regional NC brain volumes using squares

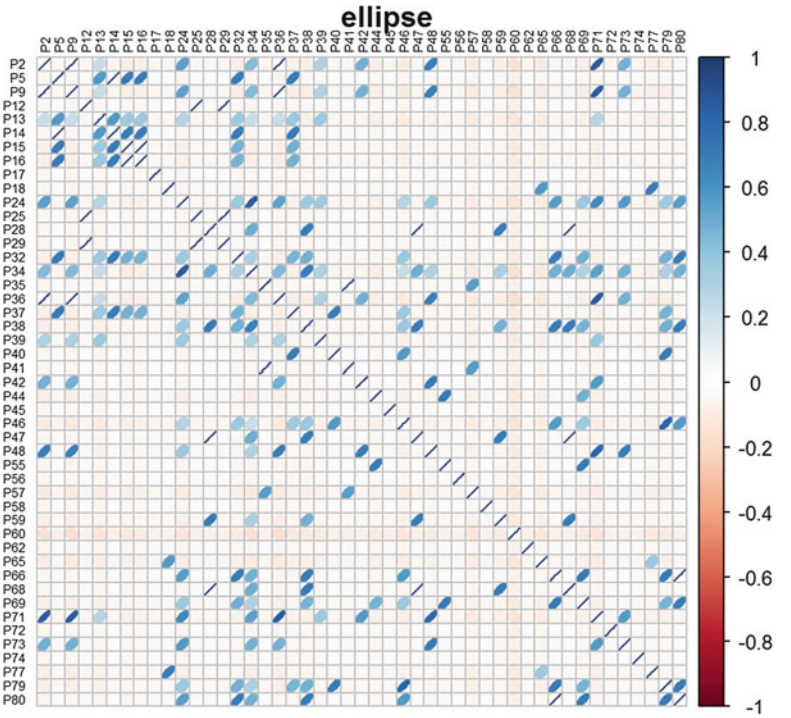

Fig. 4.28 The same correlation plot of regional NC brain volumes using ellipses

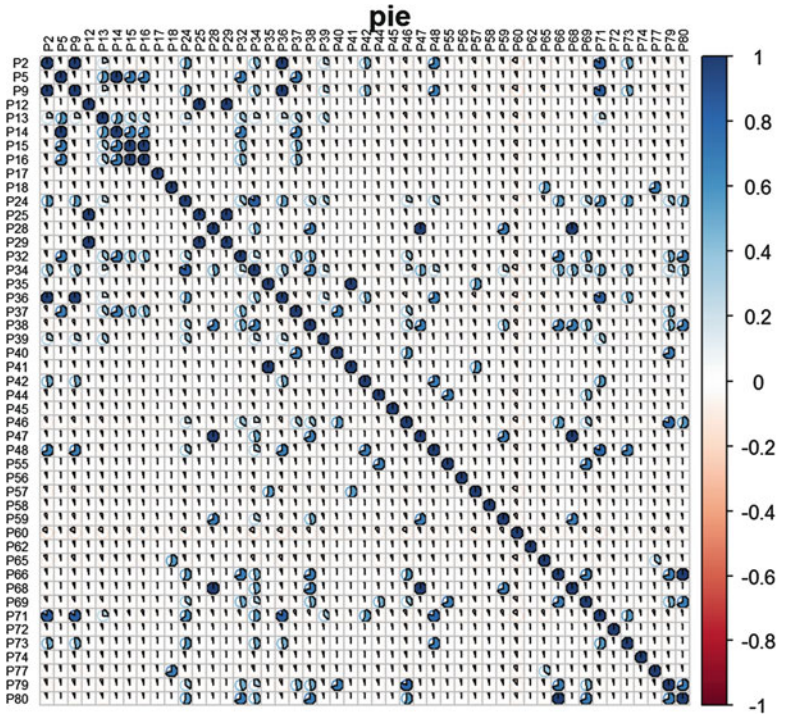

Fig. 4.29 The same correlation plot of regional NC brain volumes using pie segments

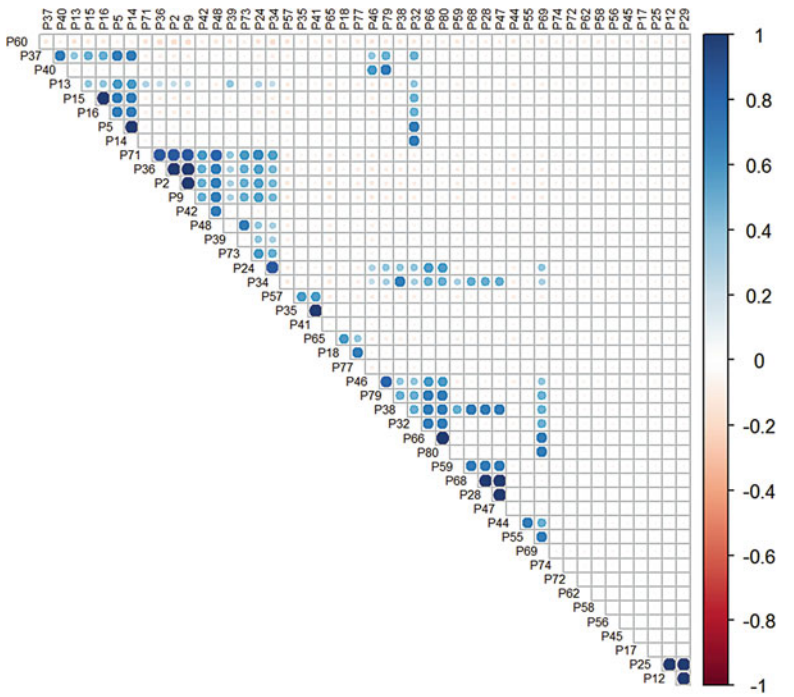

Fig. 4.30 Upper diagonal correlation plot of regional NC brain volumes using circles

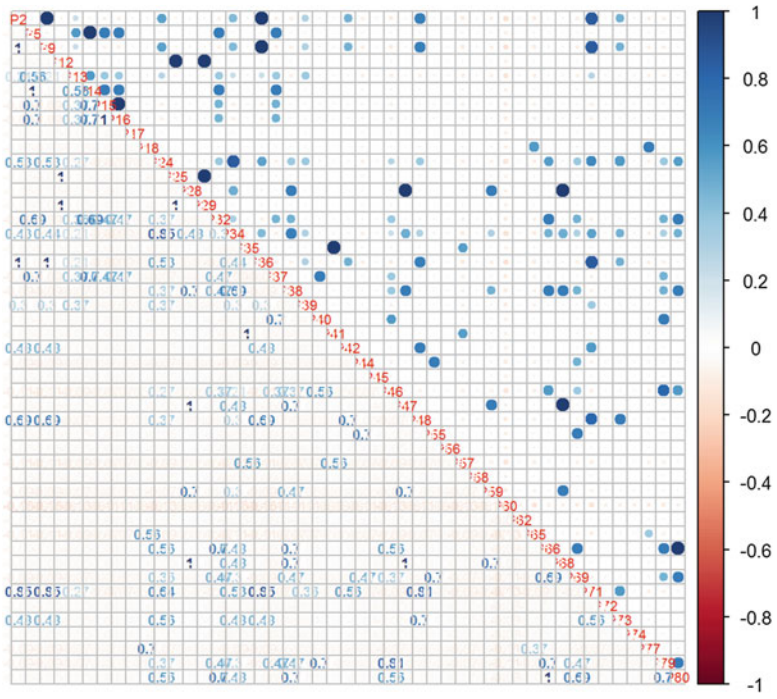

Fig. 4.31 Mixed correlation plot of regional NC brain volumes using circles and numbers

In the figures above, different shades of colors represent low-and-high correlations of the two variables corresponding to the x and y indices.

4.5 Relationships

4.5.1 Line Plots Using `ggplot`

Line charts display a series of data points (e.g., observed intensities (Y) over time (X)) by connecting them with straight-line segments. These can be used to either track temporal changes of a process or compare the trajectories of multiple cases, time series, or subjects over time, space, or state.

In this section, we will utilize the Earthquakes dataset on SOCR website. It records information about earthquakes that occurred between 1969 and 2007 with magnitudes larger than 5 on the Richter scale.

```
# library("xml2"); library("rvest")
wiki_url <- read_html("http://wiki.socr.umich.edu/index.php/SOCR_Data_Dinov_021708_Earthquakes")
html_nodes(wiki_url, "#content")

## {xml_nodeset (1)}
## [1] <div id="content" class="mw-body-primary" role="main">\n\t<a id="top
...

earthquake<- html_table(html_nodes(wiki_url, "table")[[2]])
```

In this dataset, we group the data by `Magt` (magnitude type). We will draw a "Depth vs. Latitude" line plot from this dataset. The function we are using is called `ggplot()` under `ggplot2`. The input type for this function is a data frame and `aes()` specifies aesthetic mappings of how variables in the data are mapped to visual properties (aesthetics) of the `geom` objects, e.g., lines (Fig. 4.32).

```
library(ggplot2)
plot4<-ggplot(earthquake, aes(Depth, Latitude, group=Magt, color=Magt))+

geom_line()
print(plot4)
```

There are two important components in the script. The first part, `ggplot(earthquake, aes(Depth, Latitude, group=Magt, color=Magt))`, specifies the setting of the plot: dataset, group, and color. The second part specifies that we are going to draw lines between data points. In later chapters, we will frequently use package `ggplot2` whose generic structure always involves concatenating function calls like `function1+function2+....`

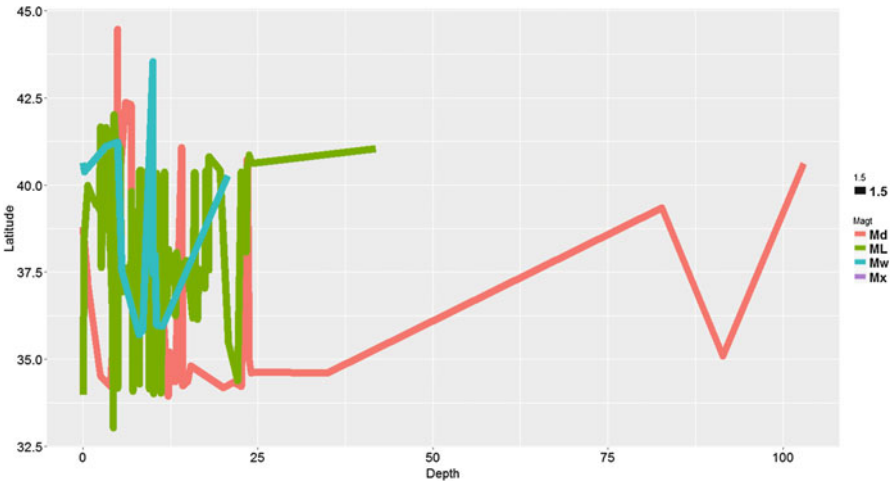

Fig. 4.32 Line plot of Earthquake magnitude type by its ground depth and latitude

4.5.2 Density Plots

We can visualize the distribution of different variables using density plots.

The following segment of R code plots the distribution for latitude among different earthquake magnitude types. Also, it uses the `ggplot()` function combined with `geom_density()` (Fig. 4.33).

```
# library("ggplot2")
plot5<-ggplot(earthquake, aes(Latitude, group=Magt, newsize=2))+
geom_density(aes(color=Magt), size = 2) +
theme(legend.position = 'right',
      legend.text = element_text(color= 'black', size = 12, face = 'bold'),
      legend.key = element_rect(size = 0.5, linetype='solid'),
      legend.key.size = unit(1.5, 'lines'))
print(plot5)
# table(earthquake$Magt) # to see the distribution of magnitude types
```

Note the green `magt` type (Local (ML) earthquakes) peaks at latitude 37.5, which represents 37–38° North, near San Francisco, California.

4.5.3 Distributions

Probability distribution plots depict the characteristics of the underlying process that can be used to contrast and compare the shapes of distributions as proxy of the corresponding natural phenomena. For univariate, bivariate, and multivariate processes, the distribution plots are drawn as curves, surfaces, or manifolds, respectively. These plots may be used to inspect areas under the distribution plot that correspond to either probabilities or data values. The Distributome Cauchy distribution calculator and the SOCR 2D bivariate Normal Distribution plot provide simple examples of distribution plots in 1D and 2D, respectively (Fig. 4.34).

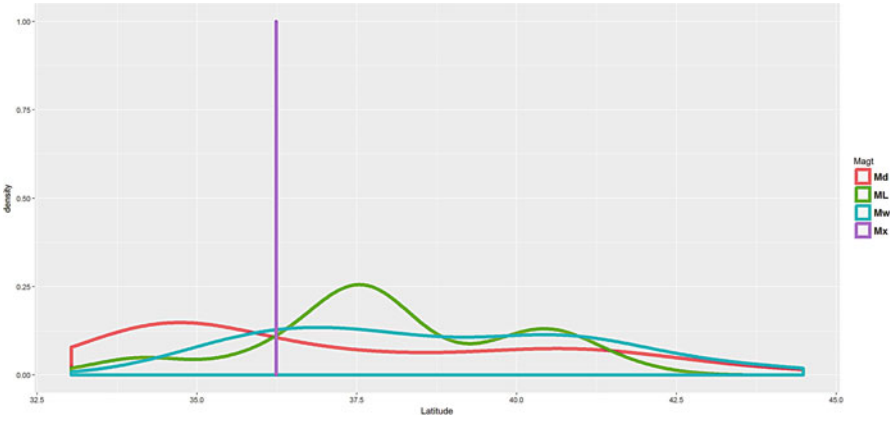

Fig. 4.33 Density plot of Earthquakes according to their magnitude types and latitude location

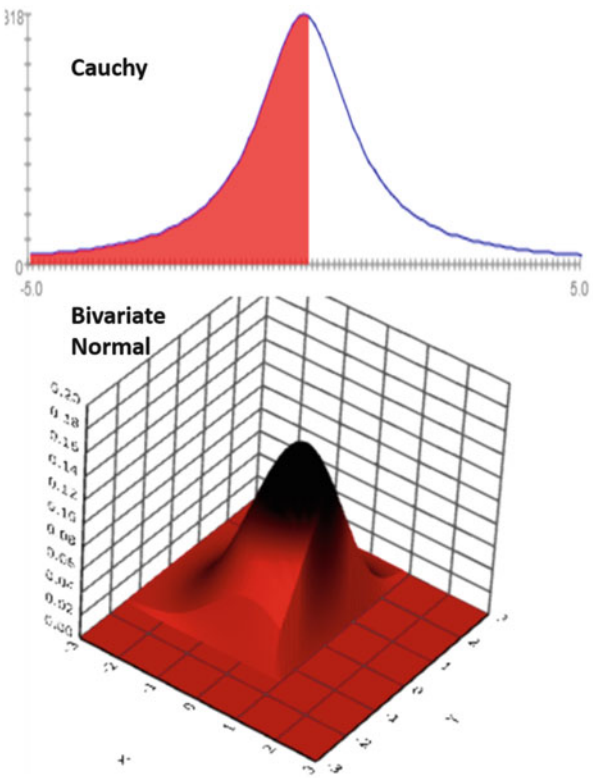

Fig. 4.34 Univariate and bivariate probability distribution calculators (Distributome Project)

4.5.4 2D Kernel Density and 3D Surface Plots

Density estimation is the process of using observed data to compute an estimate of the underlying process' probability density function. There are several approaches to obtain density estimation, but the most basic technique is to use a rescaled histogram.

Plotting 2D Kernel Density and 3D Surface plots is very important and useful in multivariate exploratory data analytics.

We will use `plot_ly()` function under `plotly` package, which requires a data frame input.

To create a surface plot, we use two vectors: x and y with length m and n respectively. We also need a matrix: z of size $m \times n$. This z matrix is created from matrix multiplication between x and y.

To plot the 2D Kernel Density estimation plot we will use the eruptions data from the "Old Faithful" geyser in Yellowstone National Park, Wyoming, stored in R as `geyser`. Also, the `kde2d()` function is needed for 2D kernel density estimation.

```
kd <- with(MASS::geyser, MASS::kde2d(duration, waiting, n = 50))
kd$x[1:5]

## [1] 0.8333333 0.9275510 1.0217687 1.1159864 1.2102041

kd$y[1:5]

## [1] 43.00000 44.32653 45.65306 46.97959 48.30612

kd$z[1:5, 1:5]

##              [,1]         [,2]         [,3]         [,4]         [,5]
## [1,] 9.068691e-13 4.238943e-12 1.839285e-11 7.415672e-11 2.781459e-10
## [2,] 1.814923e-12 8.473636e-12 3.671290e-11 1.477410e-10 5.528260e-10
## [3,] 3.428664e-12 1.599235e-11 6.920273e-11 2.780463e-10 1.038314e-09
## [4,] 6.114498e-12 2.849475e-11 1.231748e-10 4.942437e-10 1.842547e-09
## [5,] 1.029643e-11 4.793481e-11 2.070127e-10 8.297218e-10 3.088867e-09
```

Here `z=t(x)%*%y` and we apply `plot_ly` to the list `kd` via the `with()` function (Fig. 4.35).

```
library(plotly)

with(kd, plot_ly(x=x, y=y, z=z, type="surface"))
```

Note that we used the option `"surface"`.

For 3D surfaces, we have a built-in dataset in R called `volcano`. It records the volcano height at location x, y (longitude, latitude). Because z is always made from x and y, we can simply specify z to get the complete surface plot (Fig. 4.36).

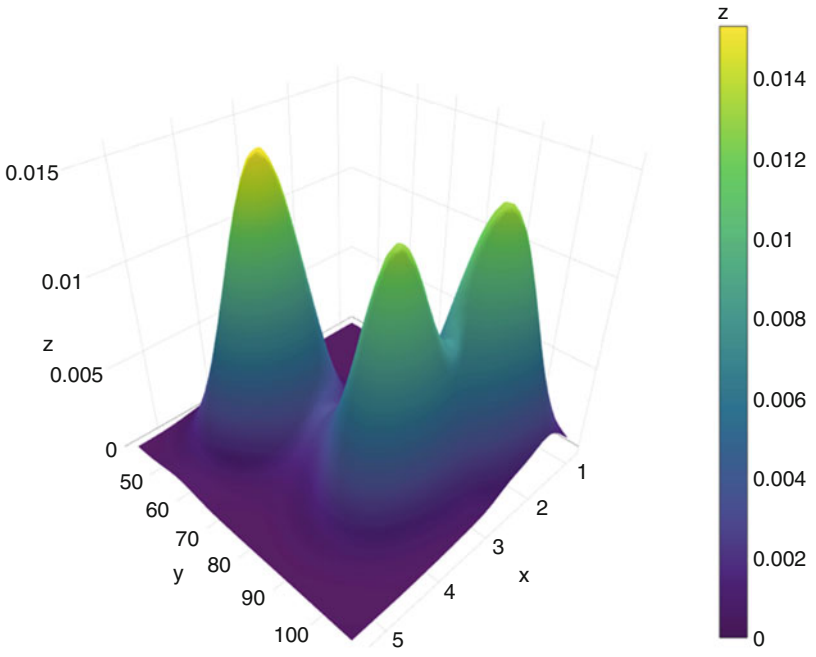

Fig. 4.35 Interactive surface plot of kernel density for the Old Faithful geyser eruptions

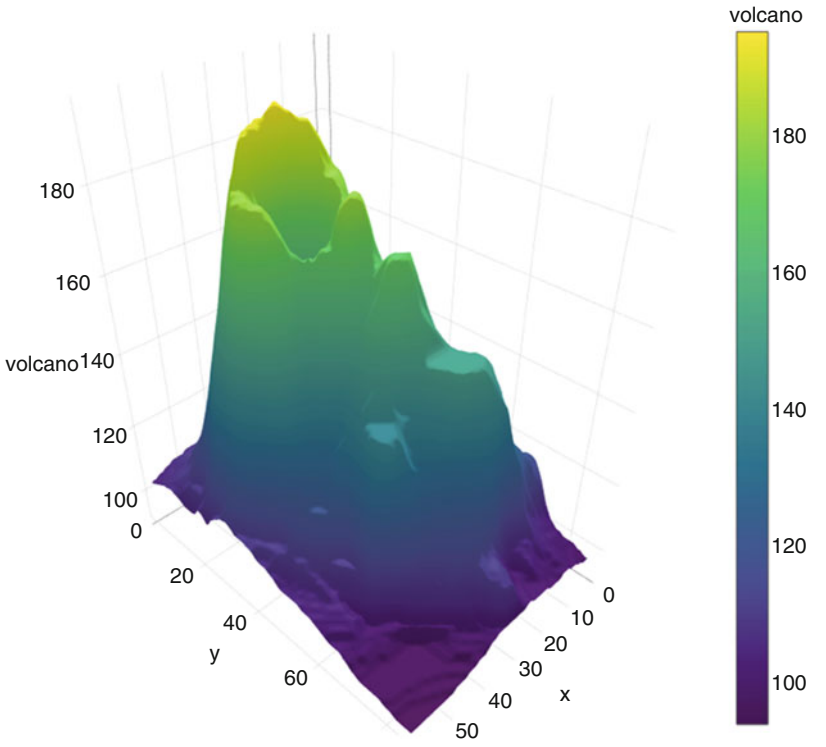

Fig. 4.36 Interactive surface plot of kernel density for the R volcano dataset

```
volcano[1:10, 1:10]
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]  100  100  101  101  101  101  101  100  100   100
##  [2,]  101  101  102  102  102  102  102  101  101   101
##  [3,]  102  102  103  103  103  103  103  102  102   102
##  [4,]  103  103  104  104  104  104  104  103  103   103
##  [5,]  104  104  105  105  105  105  105  104  104   103
##  [6,]  105  105  105  106  106  106  106  105  105   104
##  [7,]  105  106  106  107  107  107  107  106  106   105
##  [8,]  106  107  107  108  108  108  108  107  107   106
##  [9,]  107  108  108  109  109  109  109  108  108   107
## [10,]  108  109  109  110  110  110  110  109  109   108
```

```
plot_ly(z=volcano, type="surface")
```

4.5.5 Multiple 2D Image Surface Plots

Let's look at another example using a 2D brain image (Fig. 4.37).

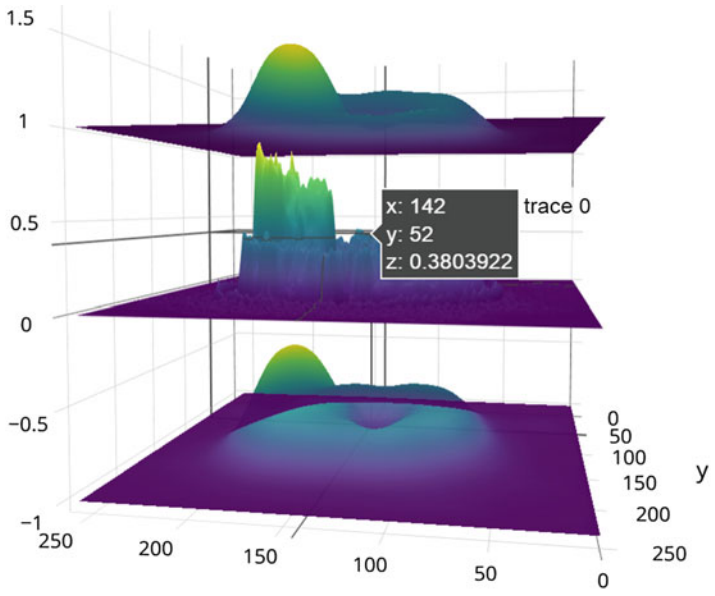

Fig. 4.37 Interactive surface plot of kernel density for the 2D brain imaging data

```
#install.packages("jpeg") ## if necessary
library(jpeg)

# Get an image file downloaded (default: MRI_ImageHematoma.jpg)
img_url <- "https://umich.instructure.com/files/1627149/download?download_fr
d=1"
img_file <- tempfile(); download.file(img_url, img_file, mode="wb")
img <- readJPEG(img_file)
file.info(img_file)

file.remove(img_file) # cleanup

## [1] TRUE

img <- img[, , 1] # extract the first channel (from RGB intensity spectrum)
as a univariate 2D array

# install.packages("spatstat")
# package spatstat has a function blur() that applies a Gaussian blur
library(spatstat)

img_s <- as.matrix(blur(as.im(img), sigma=10)) # the smoothed version of the
image

z2 <- img_s + 1   # abs(rnorm(1, 1, 1)) # Upper confidence surface
z3 <- img_s - 1   # abs(rnorm(1, 1, 1)) # Lower confidence limit

# Plot the image surfaces
p <- plot_ly(z=img, type="surface", showscale=FALSE) %>%
 add_trace(z=z2, type="surface", showscale=FALSE, opacity=0.98) %>%
 add_trace(z=z3, type="surface", showscale=FALSE, opacity=0.98)
p # Plot the mean-surface along with lower and upper confidence services.
```

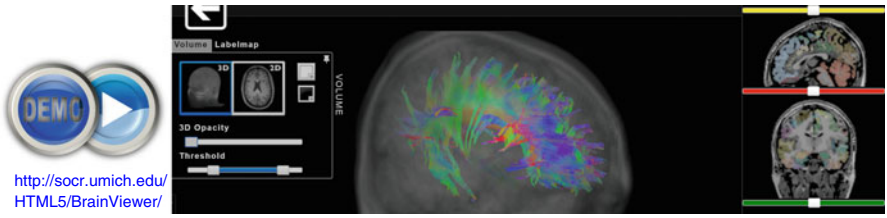

Fig. 4.38 Live demo: interactive brain viewer

The DSPA Online appendix provides additional details on shape representation, modeling, and computing on surfaces and manifolds.

4.5.6 3D and 4D Visualizations

Many datasets have intrinsic multi-dimensional characteristics. For instance, the human body is a 3D solid of matter (three spatial dimensions can be used to describe the position of every component, e.g., sMRI volume) that changes over time (the fourth dimension, e.g., fMRI hypervolumes).

The SOCR BrainViewer shows how to use a web-browser to visualize 2D cross-sections of 3D volumes, display volume-rendering, and show 1D (e.g., 1-manifold curves embedded in 3D) and 2D (e.g., surfaces, shapes) models jointly into the same 3D scene (Fig. 4.38).

We will now illustrate an example of 3D/4D visualization in R using the packages brainR and rgl.

```
# install.packages("brainR") ## if necessary
require(brainR)

# Test data: http://socr.umich.edu/HTML5/BrainViewer/data/TestBrain.nii.gz

brainURL <- "http://socr.umich.edu/HTML5/BrainViewer/data/TestBrain.nii.gz"
brainFile <- file.path(tempdir(), "TestBrain.nii.gz")
download.file(brainURL, dest=brainFile, quiet=TRUE)
brainVolume <- readNIfTI(brainFile, reorient=FALSE)

brainVolDims <- dim(brainVolume); brainVolDims

## [1] 181 217 181

# try different levels at which to construct contour surfaces (10 fast)
# lower values yield smoother surfaces # see ?contour3d
contour3d(brainVolume, level = 20, alpha = 0.1, draw = TRUE)

# multiple levels may be used to show multiple shells
# "activations" or surfaces like hyper-intense white matter
# This will take 1-2 minutes to rend!
contour3d(brainVolume, level = c(10, 120), alpha = c(0.3, 0.5),
        add = TRUE, color=c("yellow", "red"))
```

```
# create text for orientation of right/left
text3d(x=brainVolDims[1]/2, y=brainVolDims[2]/2, z = brainVolDims[3]*0.98,
text="Top")
text3d(x=brainVolDims[1]*0.98, y=brainVolDims[2]/2, z = brainVolDims[3]/2,
text="Right")

### render this on a webpage and view it!
#browseURL(paste("file://",
#        writeWebGL_split(dir= file.path(tempdir(),"webGL"),
#        template = system.file("my_template.html", package="brainR"),
#        width=500), sep=""))
```

For 4D fMRI time-series, we can load the hypervolumes similarly and then display them (Figs. 4.39, 4.40, 4.41, 4.42, 4.43, and 4.44):

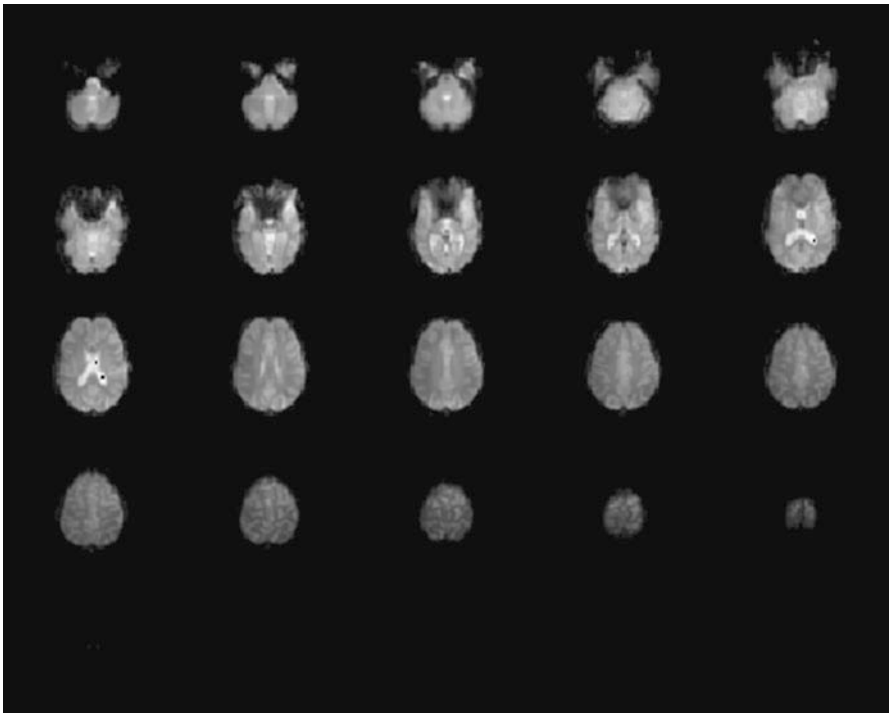

Fig. 4.39 2D cross-sectional (axial) views of the 4D fMRI data

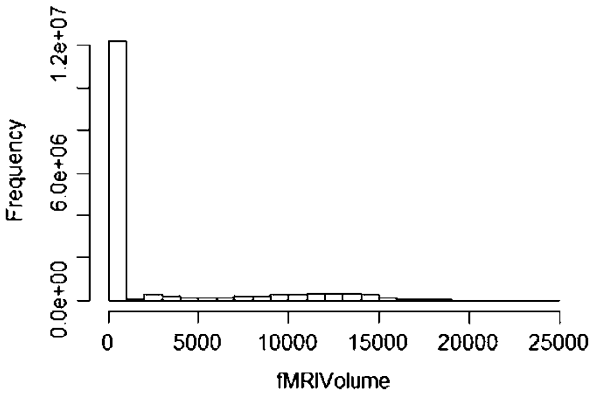

Fig. 4.40 Histogram plot of the fMRI intensities

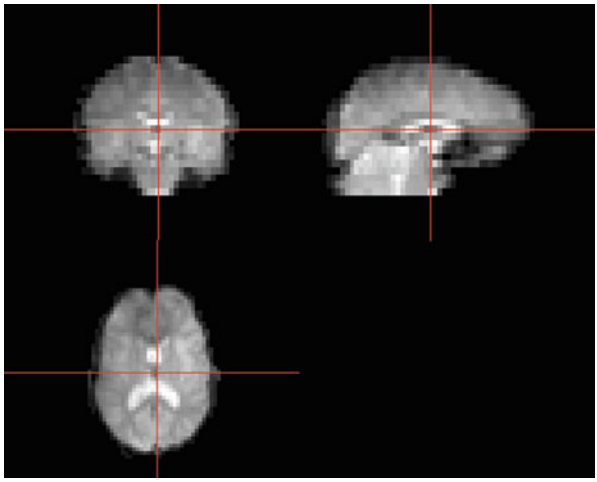

Fig. 4.41 Coronal (top-left), sagittal (to-right), and axial (bottom-left) views of the 4D fMRI data

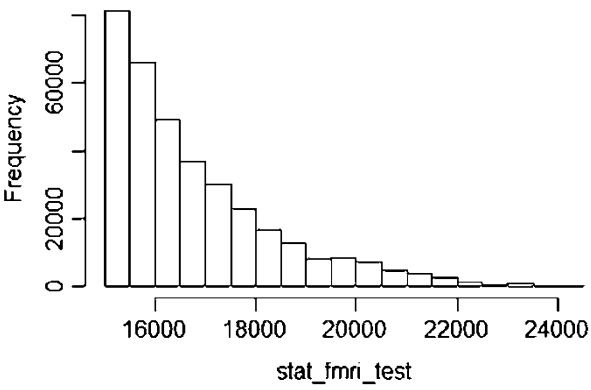

Fig. 4.42 Truncated histogram of the fMRI hyper-volume intensities

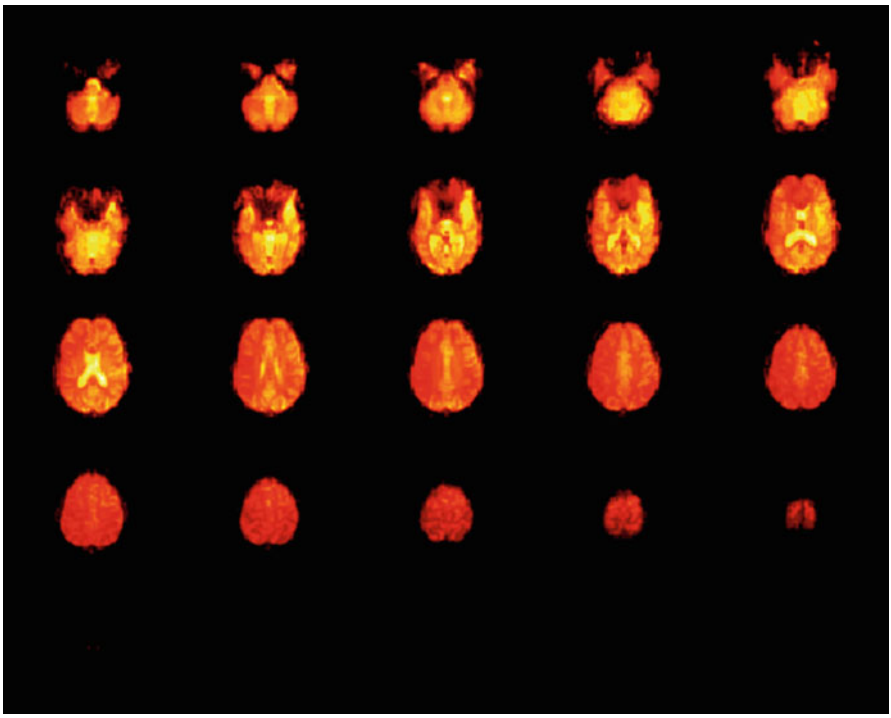

Fig. 4.43 Intensities of the fifth timepoint epoch of the 4D fMRI time series

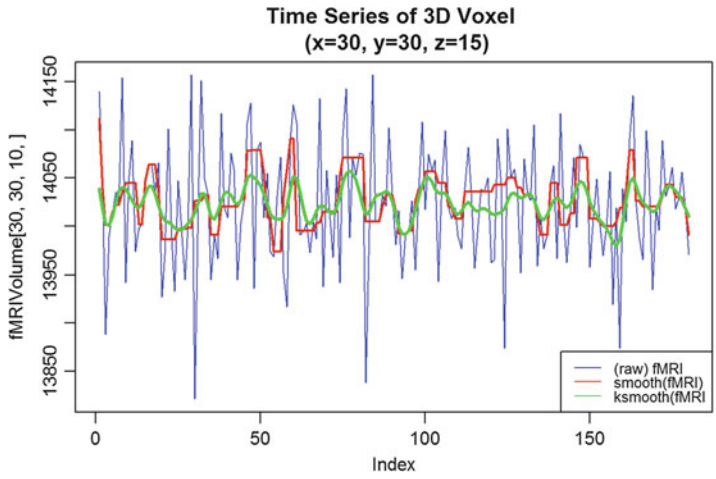

Fig. 4.44 The complete time course of the raw (blue) and two smoothed versions of the fMRI timeseries at one specific voxel location (30, 30, 15)

```
# See examples here: https://cran.r-project.org/web/packages/oro.nifti/vignettes/nifti.pdf
# and here: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0089470

fMRIURL <- "http://socr.umich.edu/HTML5/BrainViewer/data/fMRI_FilteredData_4D.nii.gz"
fMRIFile <- file.path(tempdir(), "fMRI_FilteredData_4D.nii.gz")
download.file(fMRIURL, dest=fMRIFile, quiet=TRUE)
(fMRIVolume <- readNIfTI(fMRIFile, reorient=FALSE))

## NIfTI-1 format
##   Type            : nifti
##   Data Type       : 4 (INT16)
##   Bits per Pixel  : 16
##   Slice Code      : 0 (Unknown)
##   Intent Code     : 0 (None)
##   Qform Code      : 1 (Scanner_Anat)
##   Sform Code      : 0 (Unknown)
##   Dimension       : 64 x 64 x 21 x 180
##   Pixel Dimension : 4 x 4 x 6 x 3
##   Voxel Units     : mm
##   Time Units      : sec

# dimensions: 64 x 64 x 21 x 180 ; 4mm x 4mm x 6mm x 3 sec

fMRIVolDims <- dim(fMRIVolume); fMRIVolDims

## [1]  64  64  21 180

time_dim <- fMRIVolDims[4]; time_dim

## [1] 180

# Plot the 4D array of imaging data in a 5x5 grid of images
# The first three dimensions are spatial locations of the voxel (volume element) and the fourth dimension is time for this functional MRI (fMRI) acquisition.
image(fMRIVolume, zlim=range(fMRIVolume)*0.95)


hist(fMRIVolume)


# Plot an orthographic display of the fMRI data using the axial plane
# containing the left-and-right thalamus to approximately center
# the crosshair vertically

orthographic(fMRIVolume, xyz=c(34,29,10), zlim=range(fMRIVolume)*0.9)


stat_fmri_test <- ifelse(fMRIVolume > 15000, fMRIVolume, NA)
hist(stat_fmri_test)


dim(stat_fmri_test)

## [1]  64  64  21 180

overlay(fMRIVolume, fMRIVolume[,,,5], zlim.x=range(fMRIVolume)*0.95)
```

```
# overlay(fMRIVolume, stat_fmri_test[,,,5], zlim.x=range(fMRIVolume)*0.95)

# To examine the time course of a specific 3D voxel (say the one at x=30, y=
30, z=15):
plot(fMRIVolume[30, 30, 10,], type='l', main="Time Series of 3D Voxel \n (x=
30, y=30, z=15)", col="blue")

x1 <- c(1:180)
y1 <- loess(fMRIVolume[30, 30, 10,]~ x1, family = "gaussian")
lines(x1, smooth(fMRIVolume[30, 30, 10,]), col = "red", lwd = 2)
lines(ksmooth(x1, fMRIVolume[30, 30, 10,], kernel = "normal", bandwidth = 5)
, col = "green", lwd = 3)
```

Chapter 19 provides more details about longitudinal and time-series data analysis.

4.6 Appendix

4.6.1 Hands-on Activity (Health Behavior Risks)

```
# load data CaseStudy09_HealthBehaviorRisks_Data
data_2 <- read.csv("https://umich.instructure.com/files/602090/download?down
load_frd=1", sep=",", header = TRUE)
```

Classify the cases using these variables: "AGE_G" "SEX" "RACEGR3" "IMPEDUC" "IMPMRTL" "EMPLOY1" "INCOMG" "CVDINFR4" "CVDCRHD4" "CVDSTRK3" "DIABETE3" "RFSMOK3" "FRTLT1" "VEGLT1"

```
data.raw <- data_2[, -c(1, 14, 17)]

# Does the classification match either of these:
#   TOTINDA (Leisure time physical activities per month, 1=Yes, 2=No,
9=Don't know/Refused/Missing)
#   RFDRHV4 (Heavy alcohol consumption, 1=No, 2=Yes,
9=Don't know/Refused/Missing)

hc = hclust(dist(data.raw), 'ave')
# the agglomeration method can be specified "ward.D", "ward.D2", "single",
"complete", "average" (= UPGMA), "mcquitty" (= WPGMA), "median" (= WPGMC)
or "centroid" (= UPGMC)
```

Plot a clustering diagram (Fig. 4.45)

```
par (mfrow=c(1, 1))
# very simple dendrogram
plot(hc)
```

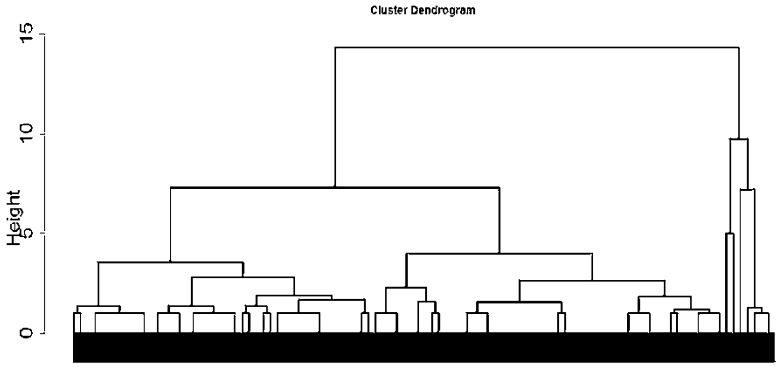

Fig. 4.45 Clustering dendrogram using the Health Behavior Risks case-study

```
summary(data_2$TOTINDA); summary(data_2$RFDRHV4)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    1.00    1.00    1.56    2.00    9.00

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     1.0     1.0     1.0     1.3     1.0     9.0

cutree(hc, k = 2)

##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
…

## [885] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [919] 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [953] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [987] 2 2 2 2 2 2 2 2 2 2 2 2 2

# alternatively specify the height, which is, the value of the criterion ass
ociated with the
# clustering method for the particular agglomeration -- cutree(hc, h= 10)

table(cutree(hc, h= 10)) # cluster distribution

##
##   1   2
## 930  70
```

Let's try to identify the number of cases for varying number of clusters.

```
# To identify the number of cases for varying number of clusters we
# can combine calls to cutree and table in a call to sapply –
# to see the sizes of the clusters for $2\ge k \ge 10$ cluster-solutions:
# numbClusters=4;
myClusters = sapply(2:5, function(numbClusters)table(cutree(hc,
numbClusters)))
names(myClusters) <- paste("Number of Clusters=", 2:5, sep = "")
myClusters
```

```
## $`Number of Clusters=2`
##
##   1   2
## 930  70
##
## $`Number of Clusters=3`
##
##   1   2   3
## 930  50  20
##
## $`Number of Clusters=4`
##
##   1   2   3   4
## 500 430  50  20
##
## $`Number of Clusters=5`
##
##   1   2   3   4   5
## 500 430  10  40  20
```

Next, inspect the cluster labels for all **SubjectIDs**:

```
#To see which SubjectIDs are in which clusters:
table(cutree(hc, k=2))

##
##   1   2
## 930  70

groups.k.2 <- cutree(hc, k = 2)
sapply(unique(groups.k.2), function(g)data_2$ID[groups.k.2 == g])
```

We can examine which TOTINDA (Leisure time physical activities per month, 1 = Yes, 2 = No, 9 = Don't know/Refused/Missing) and which RFDRHV4 are in which clusters (Fig. 4.46):

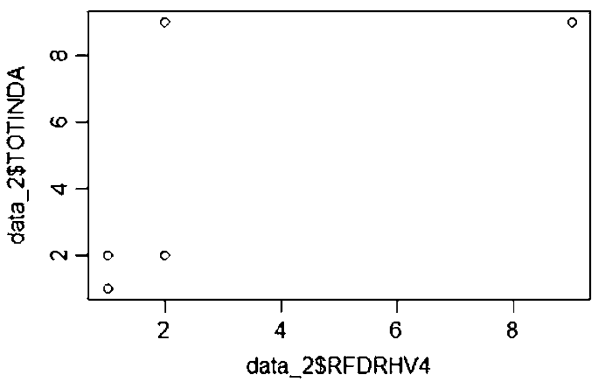

Fig. 4.46 Scatterplot between two variables in the Health Behavior Risks case-study

```
groups.k.3 <- cutree(hc, k = 3)
sapply(unique(groups.k.3), function(g)data_2$TOTINDA [groups.k.3 == g])

## [[1]]
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##

…

## [911] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##
## [[2]]
## [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 9 9 9 9 9
## [36] 9 9 9 9 9 9 9 9 9 9 9 9 9 9
##
## [[3]]
## [1] 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

sapply(unique(groups.k.3), function(g)data_2$RFDRHV4[groups.k.3 == g])

## [[1]]
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  …
## [911] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##
## [[2]]
## [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [36] 2 2 2 2 2 9 9 9 9 9 9 9 9 9
##
## [[3]]
## [1] 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

# Perhaps there are intrinsically 3 groups here e.g., 1, 2 and 9 .
groups.k.3 <- cutree(hc, k = 3)
sapply(unique(groups.k.3), function(g)data_2$TOTINDA [groups.k.3 == g])

## [[1]]
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  …

## [911] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##
## [[2]]
## [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 9 9 9 9 9
## [36] 9 9 9 9 9 9 9 9 9 9 9 9 9 9
##
## [[3]]
## [1] 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
sapply(unique(groups.k.3), function(g)data_2$RFDRHV4 [groups.k.3 == g])

## [[1]]
## [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  …

## [911] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##
## [[2]]
```

```
## [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [36] 2 2 2 2 2 9 9 9 9 9 9 9 9 9 9
##
## [[3]]
## [1] 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

# Note that there is quite a dependence between the outcome variables.
plot(data_2$RFDRHV4, data_2$TOTINDA)
```

```
# drill down deeper
table(groups.k.3, data_2$RFDRHV4)

##
## groups.k.3   1   2   9
##          1 910  20   0
##          2   0  40  10
##          3   0   0  20
```

To characterize the clusters, we can look at cluster summary statistics, like the median, of the variables that were used to perform the cluster analysis. These can be broken down by the groups identified by the cluster analysis. The aggregate function will compute statistics (e.g., median) on many variables simultaneously. Let's examine the median values for each variable we used in the cluster analysis, broken up by cluster groups:

```
aggregate(data_2, list(groups.k.3), median)

##   Group.1    ID AGE_G SEX RACEGR3 IMPEDUC IMPMRTL EMPLOY1 INCOMG CVDINFR4
## 1       1 465.5     5   2       1       5       1       2      4        2
## 2       2 955.5     6   2       4       6       5       8      6        2
## 3       3 990.5     6   2       9       6       6       8      6        2
##   CVDCRHD4 CVDSTRK3 DIABETE3 RFSMOK3 RFDRHV4 FRTLT1 VEGLT1 TOTINDA
## 1      2.0        2        3       1       1      1      1       1
## 2      2.0        2        3       2       2      9      9       2
## 3      4.5        2        4       9       9      9      9       9
```

4.6.2 Additional `ggplot` Examples

Below, we will show additional visualization examples.

Housing Price Data

This example uses the SOCR Home Price Index data of 19 major US cities from 1991 to 2009 (Fig. 4.47).

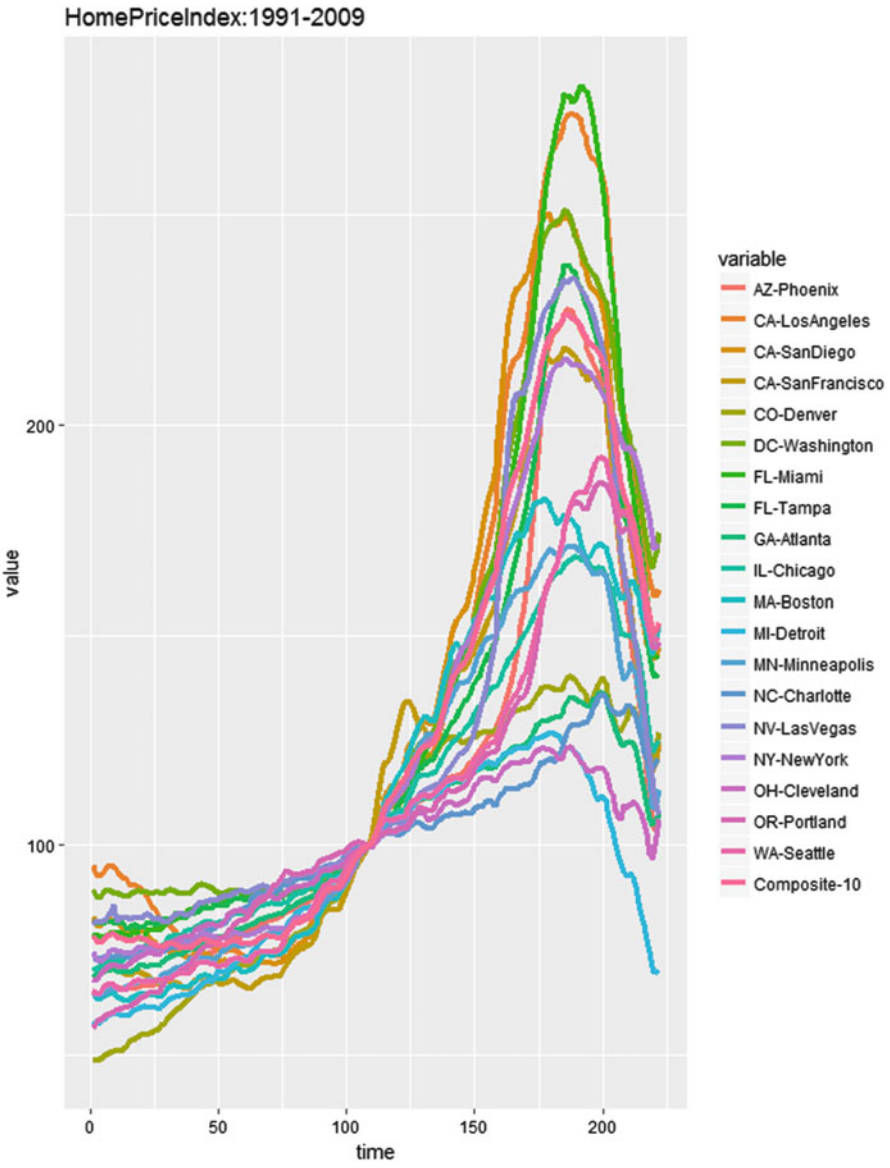

Fig. 4.47 Home price index plot over time

```
library(rvest)
# draw data
wiki_url <- read_html("http://wiki.socr.umich.edu/index.php/SOCR_Data_Dinov_
091609_SnP_HomePriceIndex")
hm_price_index<- html_table(html_nodes(wiki_url, "table")[[1]])
head(hm_price_index)

##   Index Year   Month AZ-Phoenix CA-LosAngeles CA-SanDiego CA-SanFrancisco
## 1     1 1991  January     65.26         95.28       83.13           71.17
## 2     2 1991 February     65.29         94.12       81.87           70.27
## 3     3 1991    March     64.60         92.83       80.89           69.56
## 4     4 1991    April     64.35         92.83       80.73           69.46
## 5     5 1991      May     64.37         93.37       81.41           70.13
## 6     6 1991     June     64.88         94.25       82.20           70.83
##   CO-Denver DC-Washington FL-Miami FL-Tampa GA-Atlanta IL-Chicago
## 1     48.67         89.38    79.08    81.75      69.61      70.04
## 2     48.68         88.80    78.55    81.76      69.17      70.50
## 3     48.85         87.59    78.44    81.43      69.05      70.63
## 4     49.20         87.56    78.55    81.46      69.40      71.09
## 5     49.51         88.61    77.95    81.33      69.69      71.36
## 6     50.09         89.28    78.49    81.77      70.14      71.66
##   MA-Boston MI-Detroit MN-Minneapolis NC-Charlotte NV-LasVegas NY-NewYork
## 1     64.97      58.24          64.21        73.32       80.96      74.59
## 2     64.17      57.76          64.20        73.26       81.58      73.69
## 3     63.57      57.63          64.19        72.75       81.65      72.87
## 4     63.35      57.85          64.30        72.88       81.67      72.29
## 5     63.84      58.36          64.75        73.26       82.02      72.63
## 6     64.25      58.90          64.95        73.49       81.91      73.50
##   OH-Cleveland OR-Portland WA-Seattle Composite-10
## 1        68.24       56.53      65.53        78.53
## 2        67.96       56.94      64.60        77.77
## 3        68.18       58.03      64.47        77.00
## 4        69.10       58.39      65.09        76.86
## 5        69.92       58.90      66.03        77.31
## 6        70.55       59.54      66.68        78.02
hm_price_index <- hm_price_index[, c(-2, -3)]
colnames(hm_price_index)[1] <- c('time')
require(reshape)

hm_index_melted = melt(hm_price_index, id.vars='time') #a common trick for p
lot, wide -> long format
ggplot(data=hm_index_melted, aes(x=time, y=value, color=variable)) +
  geom_line(size=1.5) + ggtitle("HomePriceIndex:1991-2009")
```

Modeling the Home Price Index Data (Fig. 4.48)

```
# Linear regression and predict
hm_price_index$pred = predict(lm(`CA-SanFrancisco` ~ `CA-LosAngeles`,
data=hm_price_index))
ggplot(data=hm_price_index, aes(x = `CA-LosAngeles`)) +
  geom_point(aes(y = `CA-SanFrancisco`)) +
  geom_line(aes(y = pred), color='Magenta', size=2) +
ggtitle("PredictHomeIndex SF - LA")
```

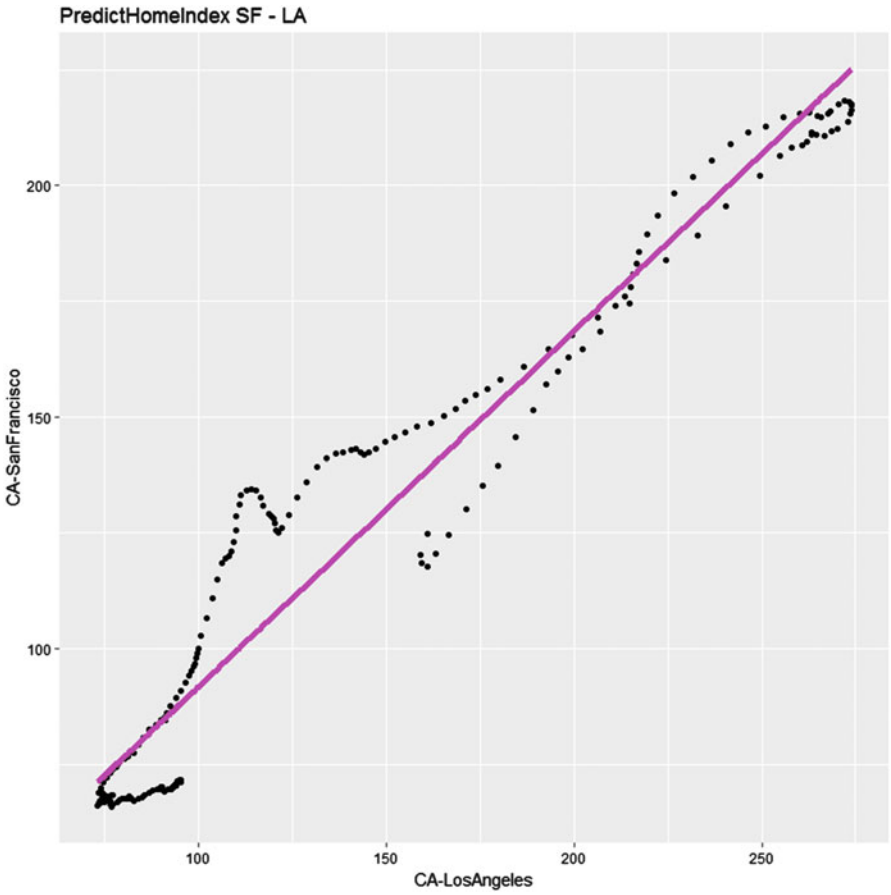

Fig. 4.48 Predicting the San Francisco home process using data from the Los Angeles home sales

We can also use `ggplot` to draw pairs plots (Fig. 4.49).

```
# install.packages("GGally")
require(GGally)

pairs <- hm_price_index[, 10:15]
head(pairs)

##   GA-Atlanta IL-Chicago MA-Boston MI-Detroit MN-Minneapolis NC-Charlotte
## 1      69.61      70.04     64.97      58.24          64.21        73.32
## 2      69.17      70.50     64.17      57.76          64.20        73.26
## 3      69.05      70.63     63.57      57.63          64.19        72.75
## 4      69.40      71.09     63.35      57.85          64.30        72.88
## 5      69.69      71.36     63.84      58.36          64.75        73.26
## 6      70.14      71.66     64.25      58.90          64.95        73.49

colnames(pairs) <- c("Atlanta", "Chicago", "Boston", "Detroit",
"Minneapolis",  "Charlotte")
ggpairs(pairs) # you can define the plot design by claim "upper", "lower",
"diag" etc.
```

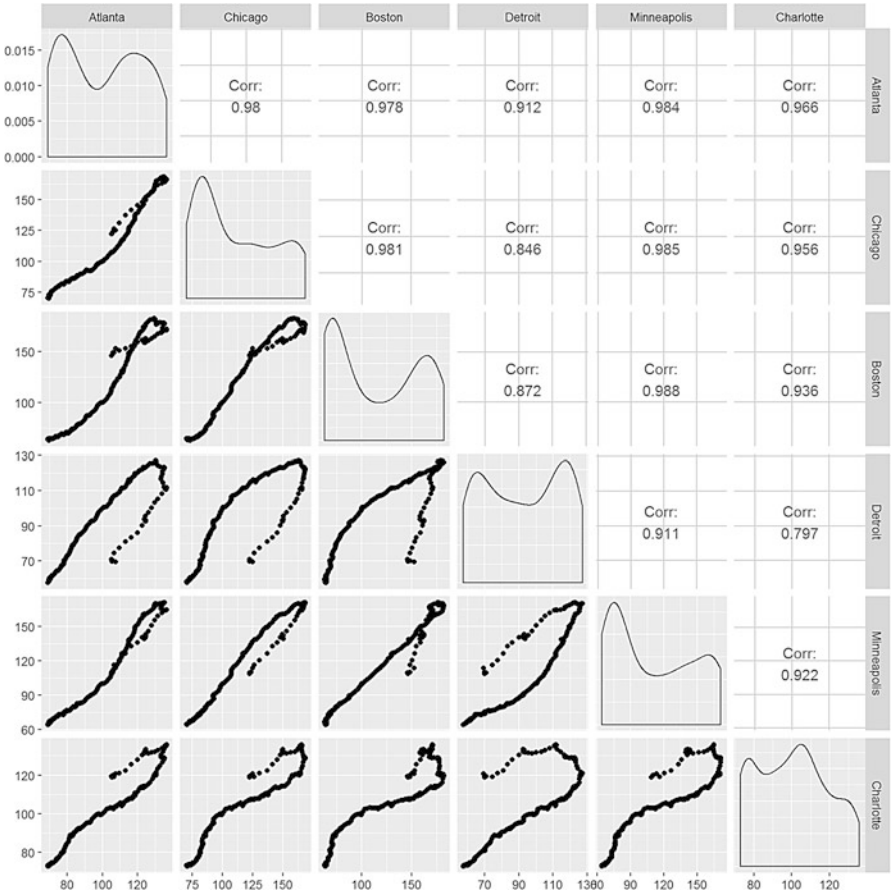

Fig. 4.49 A more elaborate pairs plot of the home price index dataset illustrating the distributions of home prices within a metropolitan area, as well as the paired relations between regions

Map of the Neighborhoods of Los Angeles (LA)

This example interrogates data of 110 LA neighborhoods, which includes measures of education, income, and population demographics.

Here, we select the **Longitude** and **Latitude** as the axes, mark these 110 Neighborhoods according to their population, fill out those points according to the income of each area, and label each neighborhood (Fig. 4.50).

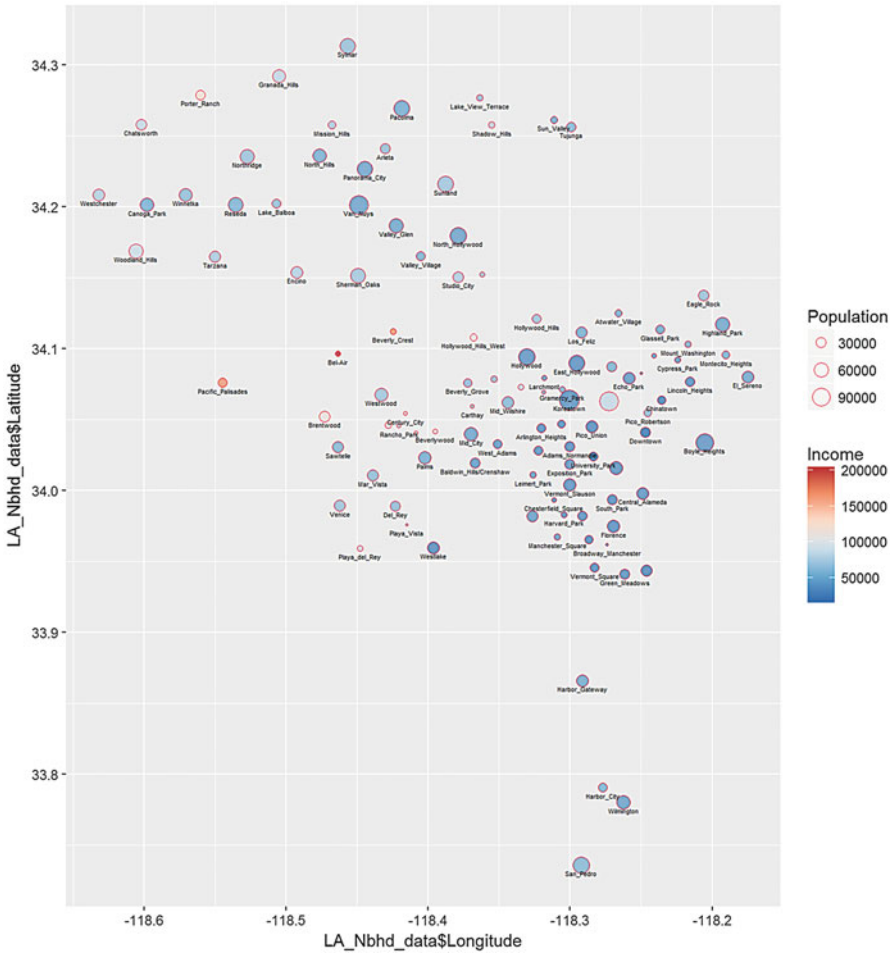

Fig. 4.50 Bubble plot of Los Angeles neighborhood location (longitude vs latitude), population size, and income

```
library(rvest)
require(ggplot2)
#draw data
wiki_url <- read_html("http://wiki.socr.umich.edu/index.php/SOCR_Data_LA_Neighborhoods_Data")
html_nodes(wiki_url, "#content")

## {xml_nodeset (1)}
## [1] <div id="content" class="mw-body-primary" role="main">\n\t<a id="top
...
```

```
LA_Nbhd_data <- html_table(html_nodes(wiki_url, "table")[[2]])
#display several lines of data
head(LA_Nbhd_data);
```

```
##                  LA_Nbhd Income Schools Diversity Age Homes Vets Asian
## 1        Adams_Normandie  29606     691       0.6  26  0.26 0.05  0.05
## 2                 Arleta  65649     719       0.4  29  0.29 0.07  0.11
## 3      Arlington_Heights  31423     687       0.8  31  0.31 0.05  0.13
## 4        Atwater_Village  53872     762       0.9  34  0.34 0.06  0.20
## 5 Baldwin_Hills/Crenshaw  37948     656       0.4  36  0.36 0.10  0.05
## 6                Bel-Air 208861     924       0.2  46  0.46 0.13  0.08
##   Black Latino White Population Area Longitude Latitude
## 1  0.25   0.62  0.06      31068  0.8 -118.3003 34.03097
## 2  0.02   0.72  0.13      31068  3.1 -118.4300 34.24060
## 3  0.25   0.57  0.05      22106  1.0 -118.3201 34.04361
## 4  0.01   0.51  0.22      14888  1.8 -118.2658 34.12491
## 5  0.71   0.17  0.03      30123  3.0 -118.3667 34.01909
## 6  0.01   0.05  0.83       7928  6.6 -118.4636 34.09615
```

```
theme_set(theme_grey())
#treat ggplot as a variable
#When claim "data", we can access its column directly eg"x = Longitude"
plot1 = ggplot(data=LA_Nbhd_data, aes(x=LA_Nbhd_data$Longitude,
y=LA_Nbhd_data$Latitude))
#you can easily add attribute, points, label(eg:text)
plot1 + geom_point(aes(size=Population, fill=LA_Nbhd_data$Income), pch=21,
stroke=0.2, alpha=0.7, color=2)+
  geom_text(aes(label=LA_Nbhd_data$LA_Nbhd), size=1.5, hjust=0.5, vjust=2,
check_overlap = T)+
  scale_size_area() + scale_fill_distiller(limits=c(range(LA_Nbhd_data$Incom
e)), palette='RdBu', na.value='white', name='Income') +
  scale_y_continuous(limits=c(min(LA_Nbhd_data$Latitude), max(LA_Nbhd_data$L
atitude))) +
  coord_fixed(ratio=1) + ggtitle('LA Neughborhoods Scatter Plot (Location,
Population, Income)')
```

Observe that some areas (e.g., Beverly Hills) have disproportionately higher incomes. In addition, it is worth pointing out that the resulting plot resembles this plot of LA County (Fig. 4.51).

Latin Letter Frequency in Different Languages

This example uses `ggplot` to interrogate the SOCR Latin letter frequency data, which includes the frequencies of the 26 common Latin characters in several derivative languages. There is quite a variation between the frequencies of Latin letters in different languages (Figs. 4.52 and 4.53).

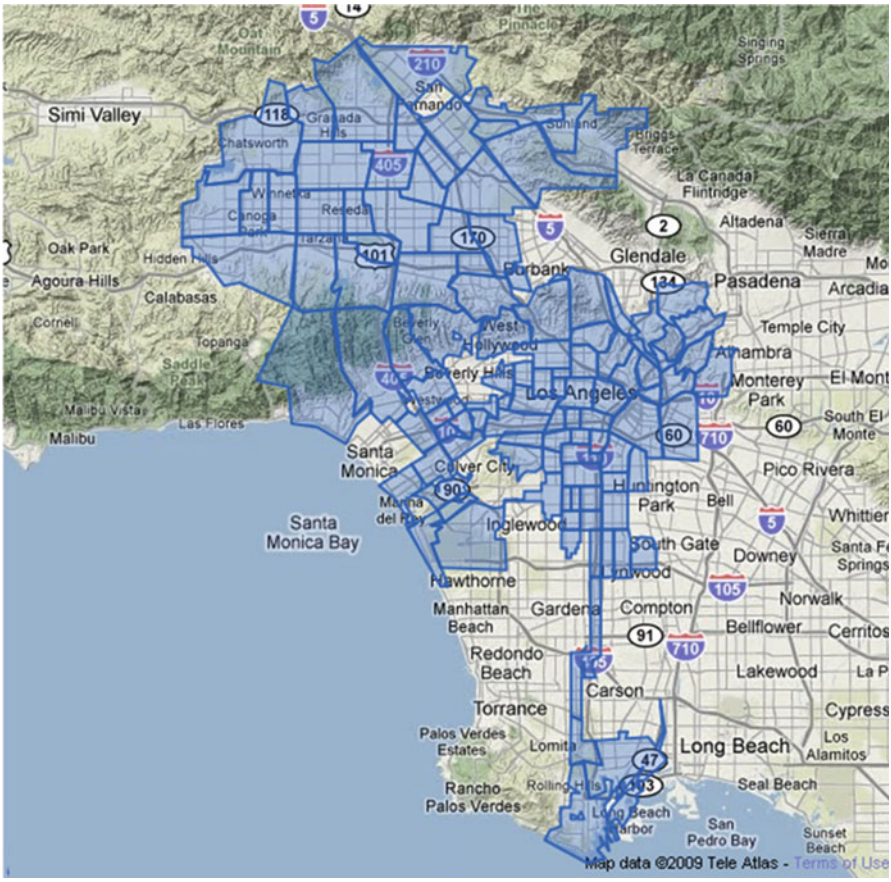

Fig. 4.51 The Los Angeles county map resembles the plot on Fig. 4.50

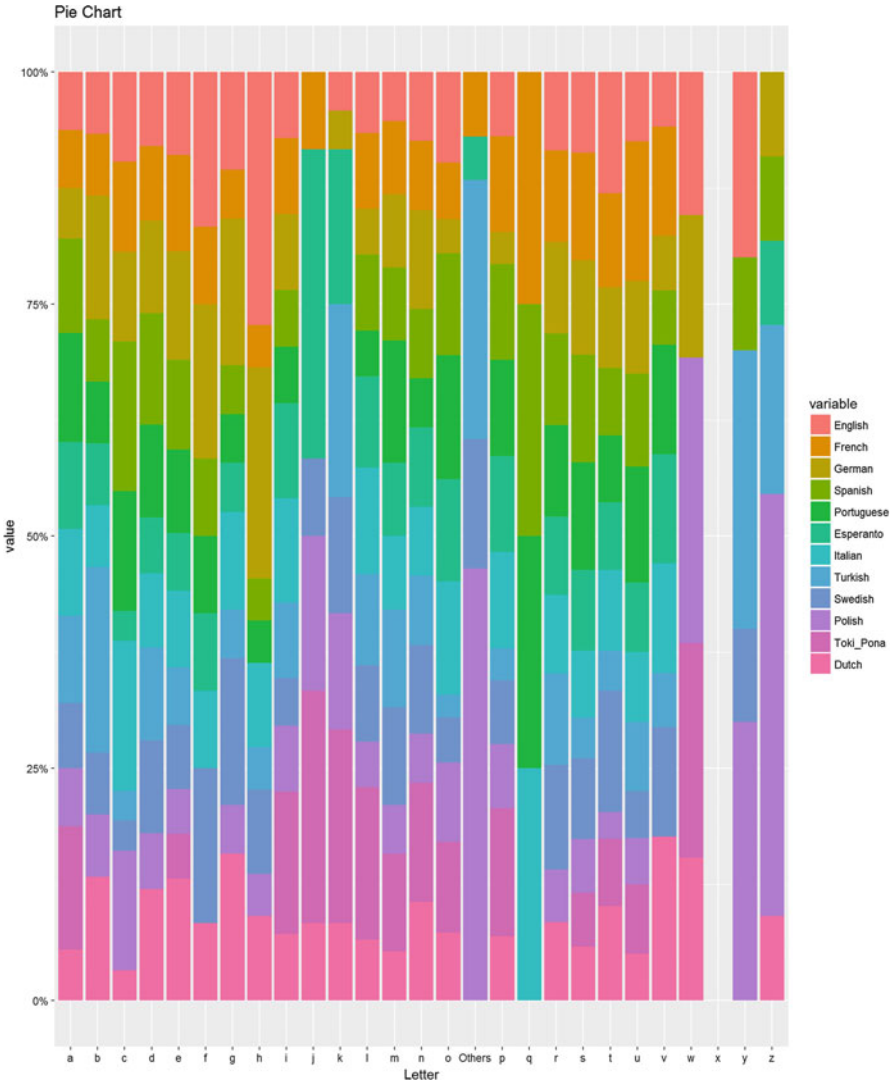

Fig. 4.52 Frequency distributions of Latin letters in several languages

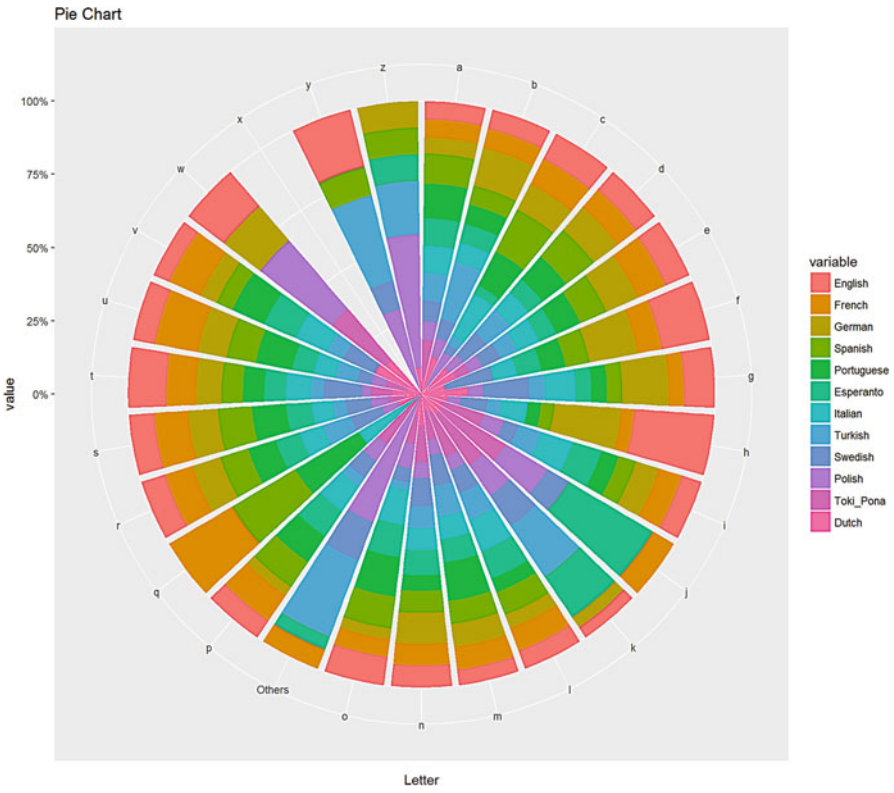

Fig. 4.53 Pie chart similar to the stacked bar chart, Fig. 4.52

```
library(rvest)
wiki_url <- read_html("http://wiki.socr.umich.edu/index.php/SOCR_LetterFrequencyData")
Letter<- html_table(html_nodes(wiki_url, "table")[[1]])
summary(Letter)
```

```
##     Letter             English            French             German
##  Length:27          Min.   :0.00000   Min.   :0.00000   Min.   :0.00000
##  Class :character   1st Qu.:0.01000   1st Qu.:0.01000   1st Qu.:0.01000
##  Mode  :character   Median :0.02000   Median :0.03000   Median :0.03000
##                     Mean   :0.03667   Mean   :0.03704   Mean   :0.03741
##                     3rd Qu.:0.06000   3rd Qu.:0.06500   3rd Qu.:0.05500
##                     Max.   :0.13000   Max.   :0.15000   Max.   :0.17000
##     Spanish          Portuguese        Esperanto          Italian
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.01000   1st Qu.:0.00500   1st Qu.:0.01000   1st Qu.:0.00500
##  Median :0.03000   Median :0.03000   Median :0.03000   Median :0.03000
##  Mean   :0.03815   Mean   :0.03778   Mean   :0.03704   Mean   :0.03815
##  3rd Qu.:0.06000   3rd Qu.:0.05000   3rd Qu.:0.06000   3rd Qu.:0.06000
##  Max.   :0.14000   Max.   :0.15000   Max.   :0.12000   Max.   :0.12000
```

```
##     Turkish           Swedish            Polish          Toki_Pona
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.01000   1st Qu.:0.01000   1st Qu.:0.01500   1st Qu.:0.00000
##  Median :0.03000   Median :0.03000   Median :0.03000   Median :0.03000
##  Mean   :0.03667   Mean   :0.03704   Mean   :0.03704   Mean   :0.03704
##  3rd Qu.:0.05500   3rd Qu.:0.05500   3rd Qu.:0.04500   3rd Qu.:0.05000
##  Max.   :0.12000   Max.   :0.10000   Max.   :0.20000   Max.   :0.17000
##      Dutch            Avgerage
##  Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.01000   1st Qu.:0.01000
##  Median :0.02000   Median :0.03000
##  Mean   :0.03704   Mean   :0.03741
##  3rd Qu.:0.06000   3rd Qu.:0.06000
##  Max.   :0.19000   Max.   :0.12000
```

```
head(Letter)
```

```
##   Letter English French German Spanish Portuguese Esperanto Italian
## 1      a    0.08   0.08   0.07    0.13       0.15      0.12    0.12
## 2      b    0.01   0.01   0.02    0.01       0.01      0.01    0.01
## 3      c    0.03   0.03   0.03    0.05       0.04      0.01    0.05
## 4      d    0.04   0.04   0.05    0.06       0.05      0.03    0.04
## 5      e    0.13   0.15   0.17    0.14       0.13      0.09    0.12
## 6      f    0.02   0.01   0.02    0.01       0.01      0.01    0.01
##   Turkish Swedish Polish Toki_Pona Dutch Avgerage
## 1    0.12    0.09   0.08      0.17  0.07     0.11
## 2    0.03    0.01   0.01      0.00  0.02     0.01
## 3    0.01    0.01   0.04      0.00  0.01     0.03
## 4    0.05    0.05   0.03      0.00  0.06     0.04
## 5    0.09    0.10   0.07      0.07  0.19     0.12
## 6    0.00    0.02   0.00      0.00  0.01     0.01
```

```
sum(letter[, -1]) #reasonable
```

```
## [1] 13.08
```

```
require(reshape)
library(scales)

dtm = melt(Letter[, -14], id.vars = c('Letter'))
p = ggplot(dtm, aes(x = Letter, y = value, fill = variable)) +
  geom_bar(position = "fill", stat = "identity") +
  scale_y_continuous(labels = percent_format())+ggtitle('Pie Chart')
#or exchange
#p = ggplot(dtm, aes(x = variable, y = value, fill = Letter)) +
geom_bar(position = "fill", stat = "identity") +
scale_y_continuous(labels = percent_format())
p
```

```
#gg pie plot actually is stack plot + polar coordinate
p + coord_polar()
```

You can experiment with the SOCR interactive motion chart, see Fig. 4.54.

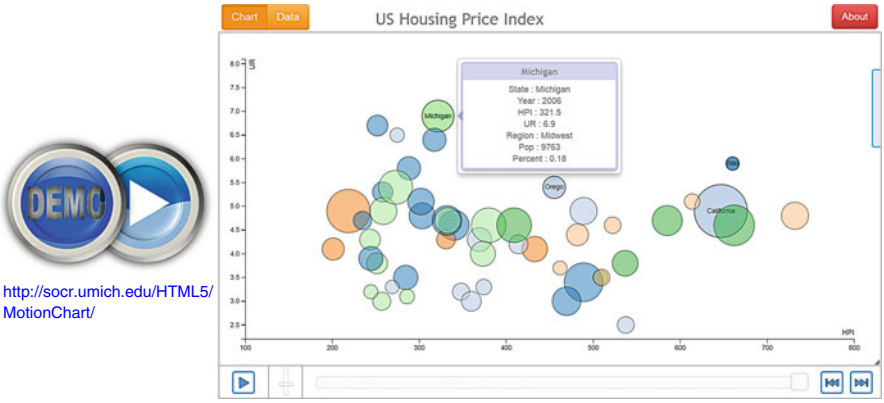

Fig. 4.54 Live demo: 6D SOCR MotionChart

4.7 Assignments 4: Data Visualization

4.7.1 Common Plots

Use the Divorce data (Case Study 01) or the TBI dataset (CaseStudy11_TBI) to generate appropriate visualization of histograms, density plots, pie charts, heatmaps, barplots, and paired correlation plots.

4.7.2 Trees and Graphs

Use the SOCR Resource Hierarchical data (JSON) or the DSPA Dynamic Certificate Map (JSON) to generate some tree/graph displays of the structural information.

The code fragment below shows an example of processing a JSON hierarchy.

```
library(jsonlite)
library(RCurl)
library(data.tree)
url <- "http://socr.umich.edu/html/navigators/D3/xml/SOCR_HyperTree.json"
raw_data <- getURL(url)
document <- fromJSON(raw_data)
tree <- Node$new(document$name)
for(i in seq_len(length(document))) {
  tree$AddChild(document$children$name[[i]])
  for(j in seq_len(length(document$children$children[[i]]))) {
    tree$children[[i]]$AddChild(document$children$children[[i]]$name[[j]])
    for(k in seq_len(length(document$children$children[[i]]$children[[j]]))){
      tree$children[[i]]$children[[j]]$AddChild((document$children$children[[i]]$ch
ildren[[j]]$name[[k]]))
    }
  }
}
suppressMessages(library(igraph))
plot(as.igraph(tree, directed = T, direction = "climb"))

suppressMessages(library(networkD3))
treenetwork <- ToDataFrameNetwork(tree, "name")
simpleNetwork(treenetwork, fontSize = 10)
```

4.7.3 Exploratory Data Analytics (EDA)

- Use SOCR Oil Gas Data to generate plots: (i) read data table, you may need to fill the inconsistent values with NAs; (ii) data preprocessing: select variables, type convert, etc.; (iii) generate two plots: the first plots includes two subplots, consumption plots and production plots; the second figure includes three subplots, for fossil, nuclear and renewable, respectively. To draw the subplots, you can use `facet_grid()`; (iv) all figures should have `year` as x axis; (v) the first figure should include three curves (fossil, nuclear and renewable) for each subplot; the second figure should include two curves (consumption and production) for each subplot.
- Use SOCR Ozone Data to generate a correlation plot with the variables `MTH_1`, `MTH_2`, ..., `MTH_12`. (Hint: you need to obtain the correlation matrix first, then apply the `corrplot` package. Try some alternative methods as well, `circle`, `pie`, `mixed` etc.)
- Use SOCR CA Ozone Data to generate a 3D surface plot (Using variables *Longitude*, *Latitude* and *O3*).
- Generate a sequence of random numbers from student *t* distribution. Draw the sample histogram and compare it with normal distribution. Try different degrees of freedom. What do you find? Does varying the *seed* and regenerating the student *t* sample change that conclusion?
- Use the SOCR Parkinson's Big Meta data (only rows with `time=0`) to generate a *heatmap plot*. Set *RowSideColors*, *ColSideColors* and *rainbow*. (Hint: you may need to select columns, properly convert the data, and normalize it.)
- Use SOCR 2011 US Jobs Ranking draw scatter plot `Overall_Score` vs. `Average_Income(USD)` include title and label the axes. Then try `qplot` for `Overall_Score` vs. `Average_Income(USD)`: (1) fill with the `Stress_Level`; (2) size the points according to `Hiring_Potential`; and (3) label using `Job_Title`.
- Use SOCR Turkiye Student Evaluation Data to generate trees and graphs, using `cutree()` and select any *k* you prefer. (Use variables Q1–Q28).

References

http://www.statmethods.net/graphs/
http://www.springer.com/us/book/9783319497501
www.r-graph-gallery.com