VISUALIZING COMPLEX DATA USING R

Powerful ideas to unleash your data, create killer visuals and transform your next presentation from good to great using R,

N.D. Lewis

Visualizing Complex Data Using R A pocket guide for ideas to unleash your data, create killer visuals and transform your next presentation from good to great

N.D. Lewis

ISBN-13: 978-1490433547

ISBN-10: 1490433546

DEDICATION

For my lovely Angela.

TABLE OF CONTENTS

PART 1: HOW TO IGNITE THE VISUAL POWER OF YOUR DATA

"The secret to becoming a winner is knowing that you can be whatever you were created to be" – Lewis Timberlake

1. Unleash the visual power of your data

Wish you had fresh ways to present data, explore relationships, visualize high volume datasets and break free from mundane charts and diagrams? Visualizing complex relationships with ease begins here. In this book you will find innovative ideas to unlock the relationships in your own data and create killer visuals to help you transform your next presentation from good to great. This book is for you if you:

- Need to impress your boss with you next presentation!
- Need fresh ideas on the presentation of complex data.
- Want to boost your visual presentation toolkit.
- Wish to capture and connect relationships in your data.
- Uncover and exploit new insights.
- Master new techniques for data presentation.
- Seek real life applications of scientific data visualization.
- Whittle down complexity to its bare bones.
- Stimulate innovation, creativity and insight.
- Intensify knowledge discovery using visual techniques.
- Diagnose, segment, connect or clarify obtuse relationships.
- Need inspiration to ignite your own creativity.

In every chapter, you will discover easy to use, scientifically based tools guaranteed to help you visualize relationships in your data. The pages you are about to read contain the key to unleash the power of your data visually. In this book you will find straight forward practical advice. By applying the insights from each chapter to your own data, you will discover how to create scientific data visualizations easily and quickly, using the free and widely available R computing package. You do not need to have any prior experience using R; Everything is explained clearly in the text.

Altogether you will find in these pages hundreds of ideas, tips and suggestions you can use to boost both your understanding and presentation of complex relationships. If you make use of even 10 percent of these, you will leap light-years beyond others in your ability to explore, comprehend and present the relationships in your own complex data. To assist you in recreating the illustrations used in the text the R computing packaged is used throughout. Illustrations are fully documented with detailed instructions on how to recreate them given at the end of each chapter. We use R, because it is free, easy to learn, extremely popular and runs on a variety of computing platforms. You do not need any prior knowledge of R to use this book. I have found individuals learn R best by ‘doing’, and this book was designed to be used actively. To get started, type in the examples as they appear in the text. You will be amazed at how easy R is to learn by simply ‘doing’!

Scientific Data Visualization

When I read my Master's Degree in computer visualization and PhD in Statistics almost a decade ago, job positions in the field of scientific data visualization were rare, existing mainly in university research departments. This is no longer the case. A new and exciting market for scientific visualization has emerged in recent years. Commercial companies, government agencies and research institutions have all recognized the value in using scientific visualization tools to explore, understand and exploit complex relationships in their data. A casual search of a popular online jobsite using the term "Data visualization" returned over 2,000 active open positions, with many offering basic salaries in excess of $150,000 per year!

All of a sudden hundreds of companies want you to help them visually analyze their data, create new visualizations, and visually display interesting relationships of all kinds. Typically, these positions ask for the ability to:

- Develop data visualizations for exploratory and expository data analysis.
- Create visualizations for public consumption that inform, engage, and educate.
- Research new ways of displaying scientific data.
- Explore data structures and features to discern representation challenges.

Good scientific visualization captures the audience's attention and exposes complex associations which might be easily overlooked by "traditional" non-visual presentational tools. When used effectively scientific visualization has the capacity to achieve two valuable goals. First, it can expose something new about the underlying patterns and relationships contained within the data. Second, it provides a mechanism to obtain a "big picture" perspective, by quickly illustrating exactly how data items are interrelated.

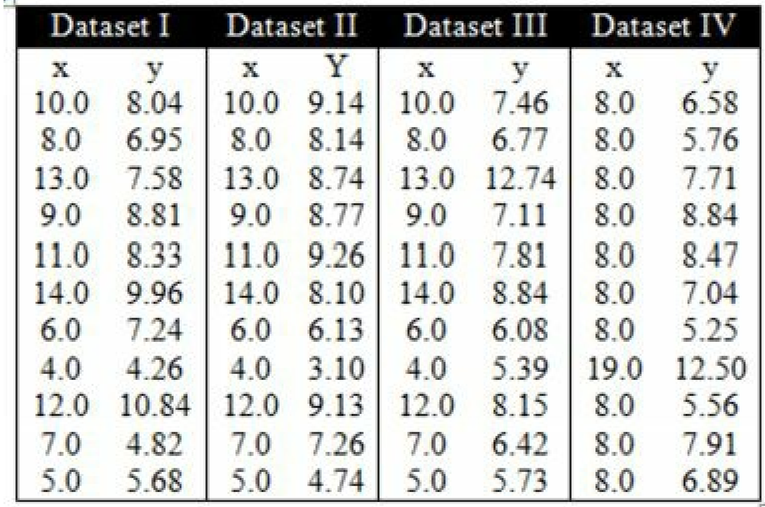
By taking data and turning it into visual content to expose relationships, business leaders, consultants and consumers are more likely to engage with, share and productively exploit it. Understanding relationships and being able to observe them is critical for good decision making. The key in communicating scientific information lies in presenting relationships in an accessible and digestible way. Have no doubt, if you possess the tools to do this you will stand out from the rest of the pack.

However, not all data visualization techniques are created equal. Just as we have paints, pencils, chalk and film to help us capture the world in different mediums, there are multiple ways in which to investigate the same data. In **Visualizing Complex Data using R** we have sorted through hundreds of scientific visualization ideas, and present to you in these pages those which we have found both practicable and informative, and can be implemented by you easily and quickly using R.

The Exceptional Power of Data Visualization

To illustrate the raw power of data visualization, let’s walk through a simple example. Imagine the following situation. You have been asked by your boss to analyze the relationships in a large dataset, perhaps containing thousands or even millions of rows of data. A key customer, product or profit hangs in the balance. Your boss has to make a decision and wants to be informed by the data before making it. You must present your result tomorrow! How are you going to make sense of all that information efficiently?

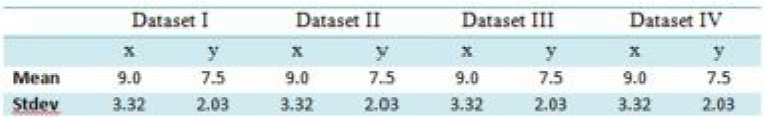
To make things a little easier suppose you have been asked to examine the four datasets presented in Table 1 1[i]. Your boss wants you to identify significant differences between the datasets. What are you to do?



Dataset I		Dataset II		Dataset III		Dataset IV	
x	y	x	Y	x	y	x	y
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89

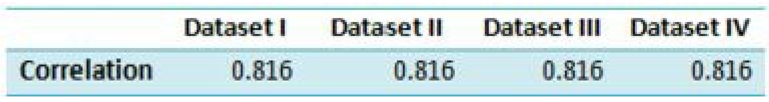
Table 1 1 The four dataset's you need to investigate

Recalling your undergraduate statistics 101 class, you calculate summary statistics. The results are shown in Table 1 2 . There appears to be little difference between the four datasets. However, just to be on the safe side (which is the best place to be with your stress ball of a boss!), you also calculate the correlation coefficient between each x and y variable. The results are shown in Table 1 3 . Again, you see no visible difference between the datasets.



	Dataset I		Dataset II		Dataset III		Dataset IV	
	x	y	x	y	x	y	x	y
Mean	9.0	7.5	9.0	7.5	9.0	7.5	9.0	7.5
Stdev	3.32	2.03	3.32	2.03	3.32	2.03	3.32	2.03

Table 1 2 Summary statistics (mean, standard deviation)



	Dataset I	Dataset II	Dataset III	Dataset IV
Correlation	0.816	0.816	0.816	0.816

Table 1 3 The x,y correlation coefficients for each dataset

If you were to end your analysis with this conclusion you would be making a huge mistake. At best the mistake will make your boss look foolish, cost you your job, your company lost profits and result in a very dissatisfied client. Too see why, take a look at Figure 1 1 , Figure 1 2 , Figure 1 3 and Figure 1 4 . All four diagrams show the scatterplot of x and y (with regression line in red) for each dataset. Clearly the relationship between x and y differs greatly, yet this difference was unobservable in the traditional summary statistics of mean, standard deviation and correlation. If you were to report the results of standard statistical analysis, you would not find much difference between the data. The real magic happens when these datasets are represented visually. All of a sudden, the datasets look very different from each other, and you cannot help but see and understand much more about the underlying relationships. This is the power of visualizing data. ***Scientific data visualization is a collection statistical methods, both quantitative and qualitative, for identifying relationships in data and consolidating them into an illustrative informative summarizing graphic.***

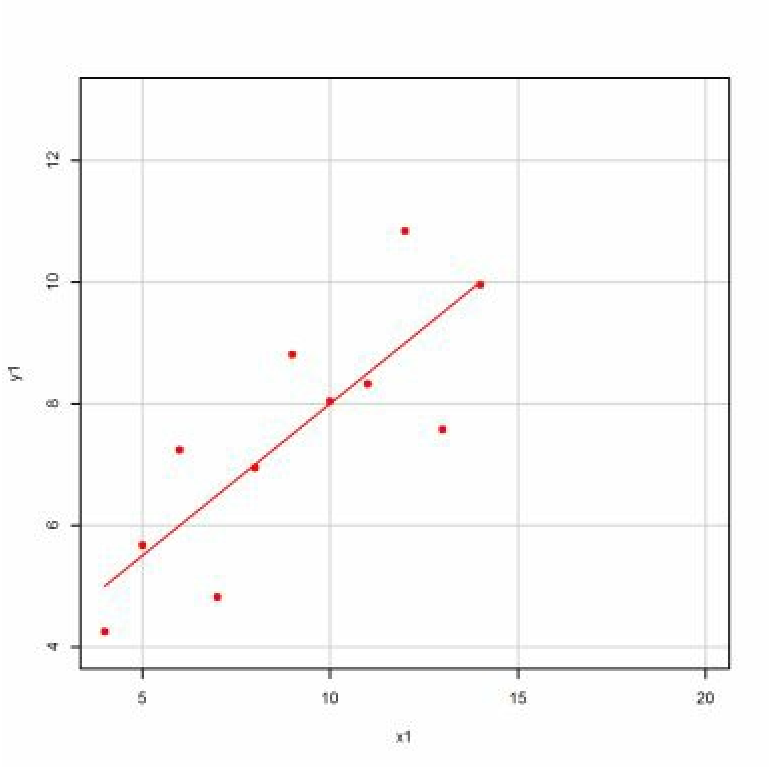


Figure 1 1 Plot of x and y for dataset 1

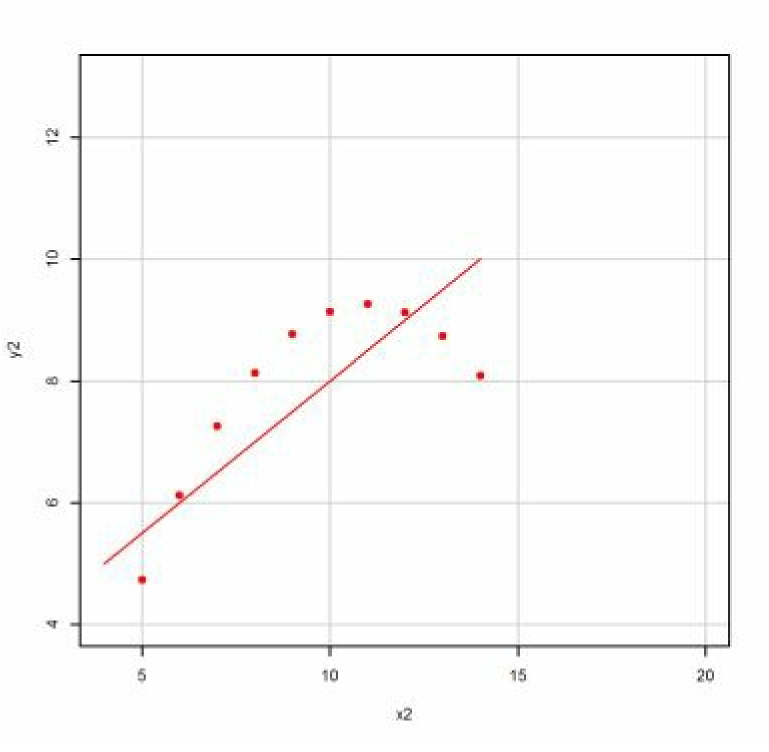


Figure 1 2 Plot of x and y for dataset 2

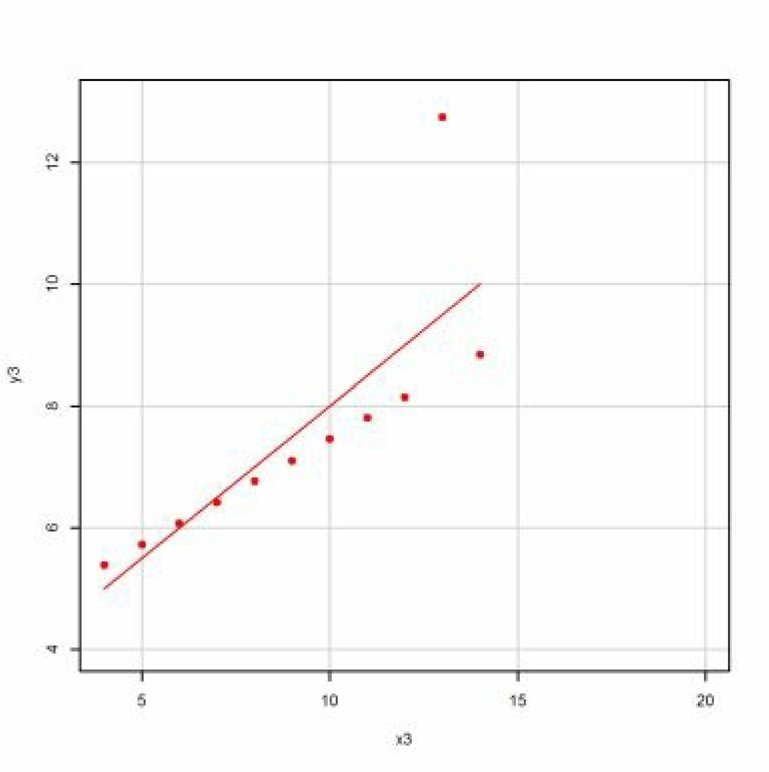


Figure 1 3 Plot of x and y for dataset 3

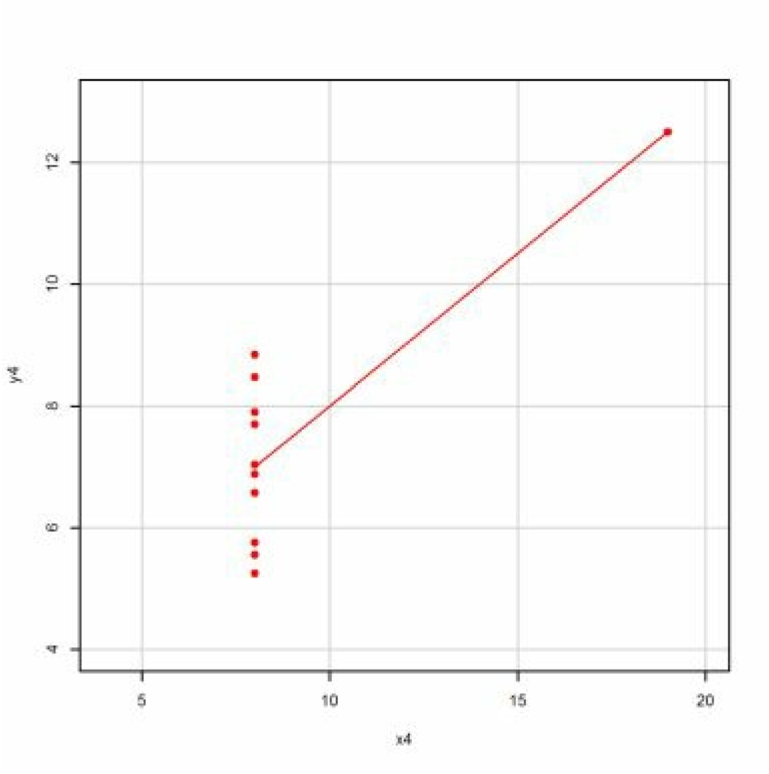


Figure 1 4 Plot of x and y for dataset 4

Totally new to R?

New users to R can use this book easily and without any prior knowledge. This is best achieved by typing in the examples as they are given, reading the comments and experimenting beginning with the visualization tips.

Since R is a software system which gains it power from user developed packages, the entire code, including installation of packages are given for each chart. Installation commands need only be entered if the package is not already installed on your machine. If an R package mentioned in the text is not installed on your machine you can download it by typing:

```
install.packages("package_name")
```

Followed by:

```
require(package_name)
```

For example to download and install the portfolio package you would type in the R console window:

```
install.packages("portfolio")
require(portfolio)
```

Tip: Copies of R and **free** tutorial guides for beginners can be downloaded at http://www.r-project.org/

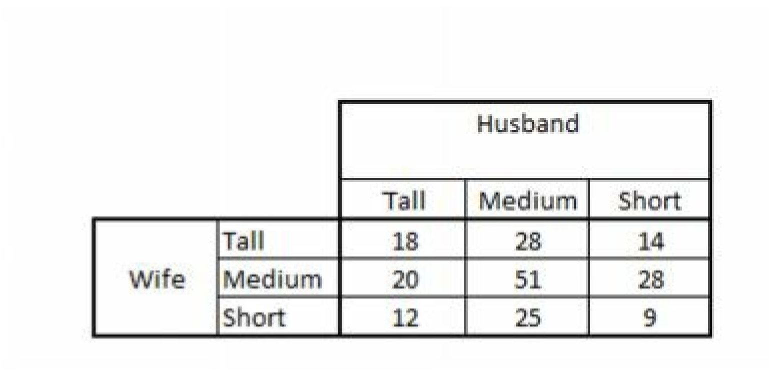
How to get the most from this book

There are at least three ways to use this book to boost your productivity and creativity. First, you can dip into it as an efficient reference tool. Identify the data visualization idea you need and see how to create it quickly. For best results type in the examples given in the text, examine the results, and then adjust for your own use. Second, browse through the numerous illustrations, tips and examples to help stimulate your own creativity. Third, by typing the many illustrations and experimenting, you will strengthen your knowledge and understanding of both scientific data visualization and R.

You don't have to wait until you have read the entire book to incorporate the ideas into your own analysis and presentations. You can experience their marvelous potency for yourself almost immediately. You can go straight to the idea of interest and immediately test, create and exploit it in your own research, analysis and presentations.

Applying the ideas in this book will transform your analysis. If you utilize even one tip or idea from each chapter, you will be far better prepared to win when faced with the challenges and opportunities of the ever expanding deluge of exploitable data. Enough talking, let's get started!

2. Easily illuminate associations in cross tabulations



		Husband		
		Tall	Medium	Short
Wife	Tall	18	28	14
	Medium	20	51	28
	Short	12	25	9

Figure 2 1 Marriage selection in respect to stature

Extracting meaningful relationships from cross tabulation data (also known as contingency tables) is an important task for many data analysts. Visual techniques offer powerful means to illuminate associations in cross tabulations. As an illustration, consider the famous investigation on marriage selection in respect to stature conducted by Sir Francis Galton[ii]. Data was collected on 205 married couples, divided into three height groups – Tall, Medium and Short respectively (medium male measurements being taken as 67 inches to 70 inches). The number of marriages in each possible combination between are presented in the cross tabulation in Figure 2 1 . On rather close inspection you might observe that men and women of contrasted heights, Short and Tall, or Tall and Short, married about as frequently as men and women of similar heights, both Tall or both Short. Of course, when presenting to an eager audience, the effort required to obtain and extract this interpretation from the cross tabulation table may be too great. You need more efficient techniques in your toolkit to display and present important cross tabulation associations. One solution to easilyilluminate associations in cross tabulations is to deploy the association plot illustrated in Figure 2 2 .

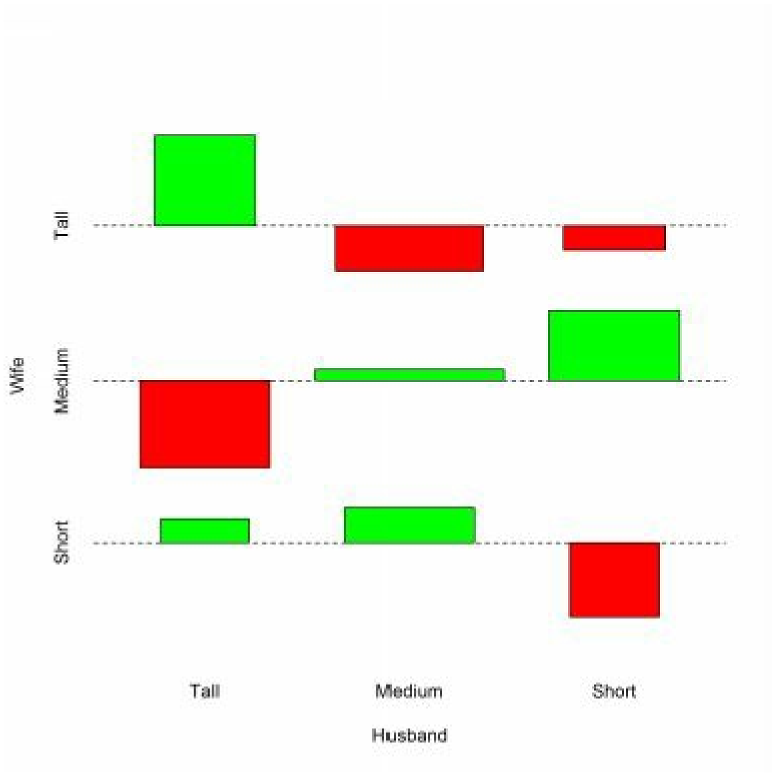


Figure 2 2 Association plot of marriage selection in respect to stature

In this plot, the area of the boxes are proportional to the difference in observed and expected frequencies in the data. The green rectangles above the dashed lines indicate observed frequencies exceeding those expected. The red rectangles below the dashed lines indicate observed frequencies below those expected[iii]. In this example, the green boxes indicate positive association, and the red boxes indicate negative association. The size and shapes of the box indicate how strong the associations are. It is immediately evident that tall people tend to marry tall people at a higher rate than one might expect; However, this is not the case with short people, they tend to marry other short people at lower frequency than might be expected.

Figure 2 2 was created using the assocplot function contained in the R base package. It was built using the following three steps:

Step 1: Download and install the required packages.

```
require{graphics}
```

Step 2: Clean and prepare the data

```
M <- as.table(cbind(c(18,28,14), c(20,51,28) , c(12,25,9)))
dimnames(M) <- list(Husband = c("Tall", "Medium", "Short"), Wife= c("Tall","Medium", "Short"))
```

Step 3: Plot the chart

```
assocplot(M,col = c("green", "red"))
```

Step 1: Installs the required packaged and makes it available to R.

Step 2: Creates a table comparableto that shown in Figure 2 1 .

Step 3: The assocplot function draws the image.

Tip 1: Change the color of the boxes to suite your taste. In Figure 2 2 we used col = c("green", "red") to create green boxes for positive association and red boxes for negative association.

Tip 2: The R package provides multiple ways to achieve a task.It is always useful to have knowledge of an alternative function for the same task. An association plot can be created using the assoc function contained in the vcd package. The result is shown in Figure 2 3 .

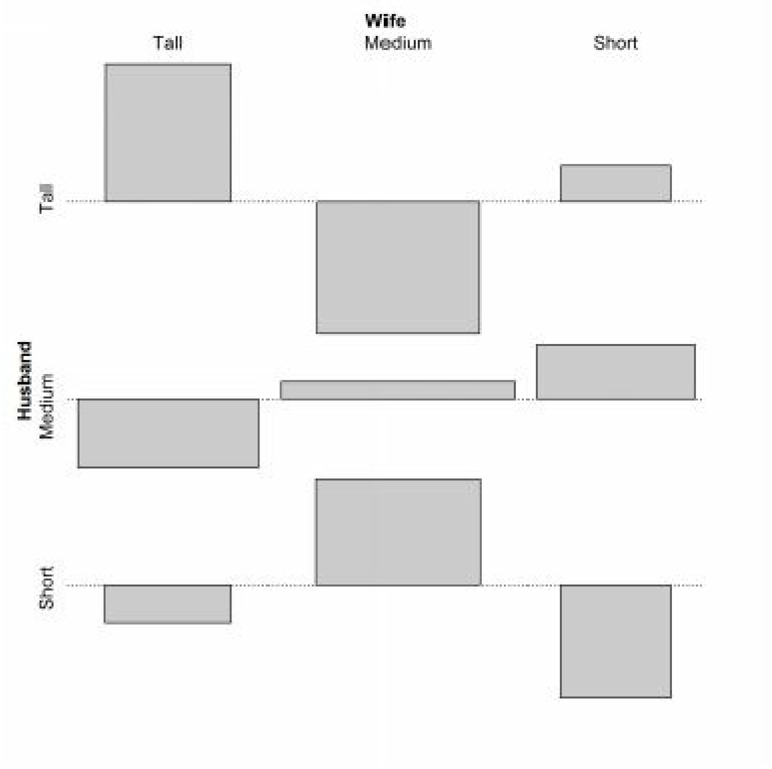


Figure 2 3 Association plot using the assoc function in the vcd package

Figure 2 3 was constructed using the following lines of R code:

```
install.packages(vcd)
library(vcd)
M <- as.table(cbind(c(18,28,14), c(20,51,28) , c(12,25,9)))
dimnames(M) <- list(Husband = c("Tall", "Medium", "Short"), Wife= c("Tall","Medium", "Short"))
assoc(M)
```

Spline Plots

Another visualizationthat is useful for cross tabulations is the spline plot. It shows differing sized boxes associated with the frequency of each factor in a cross tabulation. Figure 2 4 illustrates a spline plot for marriage selection in respect to stature. Since there are three factors (Tall, Medium and Short) the resultant image is composed of nine shaded boxes, sized by frequency.

Smallpox in England: Figure 2 5 reports the small-pox attack rate (percentage) in infected homes of individuals under and over 10 years of age for five English towns. The attack rate for those under ten appears consistently smaller across the towns. A similar observations appears to hold for those vaccinated over ten. These initial observations are more easily observed in the spine plot of Figure 2 6 .

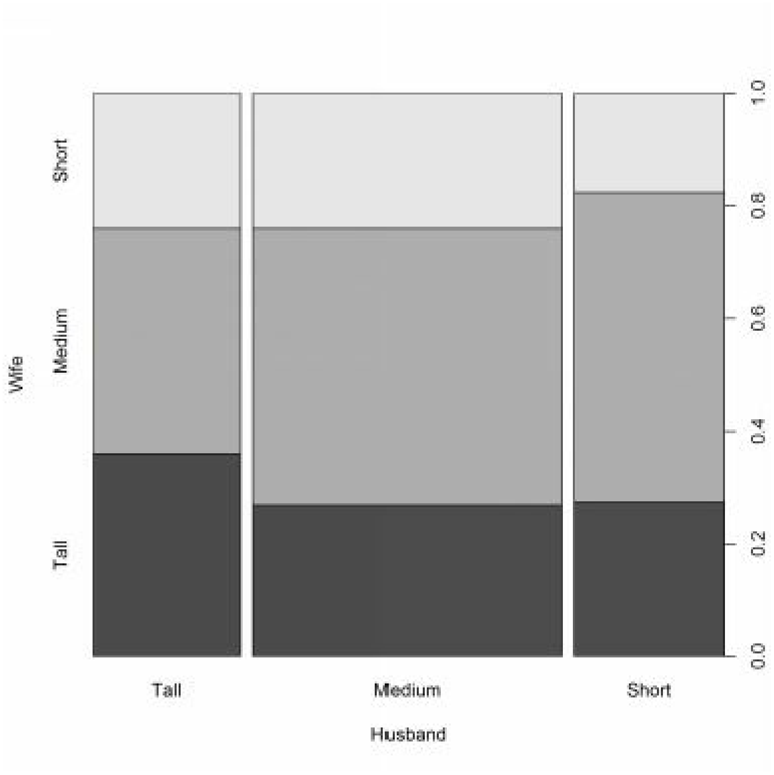
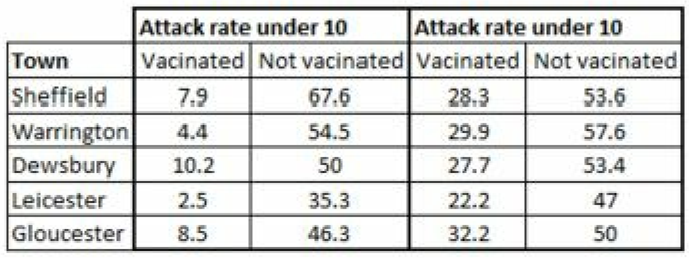


Figure 2 4 Spline plot of marriage selection in respectto stature



	Attack rate under 10		Attack rate under 10	
Town	Vacinated	Not vacinated	Vacinated	Not vacinated
Sheffield	7.9	67.6	28.3	53.6
Warrington	4.4	54.5	29.9	57.6
Dewsbury	10.2	50	27.7	53.4
Leicester	2.5	35.3	22.2	47
Gloucester	8.5	46.3	32.2	50

Figure 2 5 Smallpox in infected homes in five English towns

Figure 2 6 was created using the spineplot function contained in the R base package. It was built using the following three steps:

Step 1: Download and install the required packages.

```
require{graphics}
```

Step 2: Clean and prepare the data

```
r1 = c (7.9,    67.6,   28.3,   53.6)
r2 = c (4.4,    54.5,   29.9,   57.6)
r3 = c (10.2,   50, 27.7,   53.4)
r4 = c (2.5,    35.3,   22.2,   47)
r5 = c (8.5,    46.3,   32.2,   50)
data <- as.table(rbind(r1,r2,r3,r4,r5))

dimnames(data) <- list(Town = c("Sheffield","Warrington", "Dewsbury","Leicester","Gloucester")
, Vaccinated = c("Yes (< 10)","No (<10)", "Yes (> 10)","No (> 10)"))
```

Step 3: Plot the chart

```
spineplot(data)
```

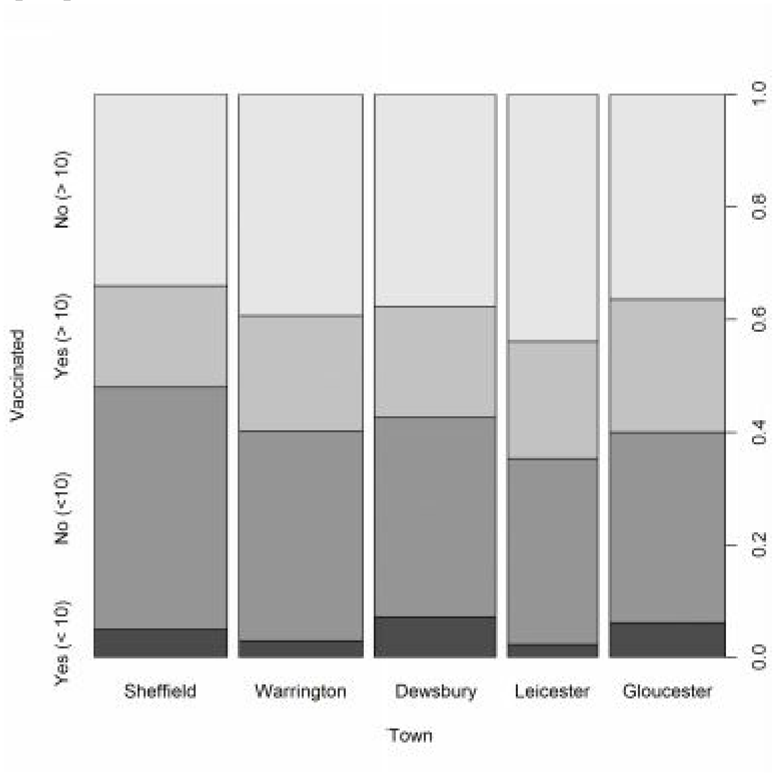


Figure 2 6 Spine plot of smallpox in infected homes in five English towns

Step 1: Installs the required packaged and makes it available to R.

Step 2: Creates a table comparableto that shown in Figure 2 5 .

Step 3: The spineplot function draws the image.

Tip : You can easilychange the color of the individual boxes in the spine plot. For example in Figure 2 7 we used spineplot(data,col= c("darkred","orange","darkgreen","yellow1")) to create dark red, orange, dark green and yellow boxes for vaccinated under 10, not vaccinated under 10, vaccinated over ten and not vaccinated over ten respectively.

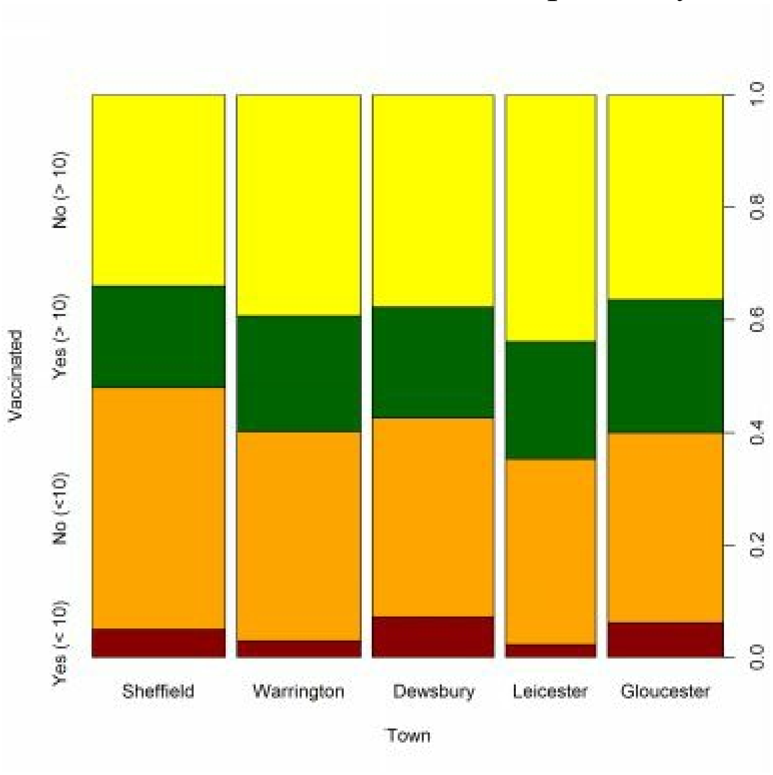


Figure 2 7 Colorful spine plot of smallpox in five English towns

Mosiac Plots

Mosiac plots offer another visualization technique for cross tabulation data. They display relative frequencies in terms of differing sized rectangles. The more frequent a category is observed, the larger is the corresponding rectangle. Figure 2 8 shows the mosaic plot of small-pox data. The mosaic plot shows many things clearly, for instance, that the attack rate is much lower for those vaccinated below the age of ten, and that this rate is relatively lower across all five English towns.

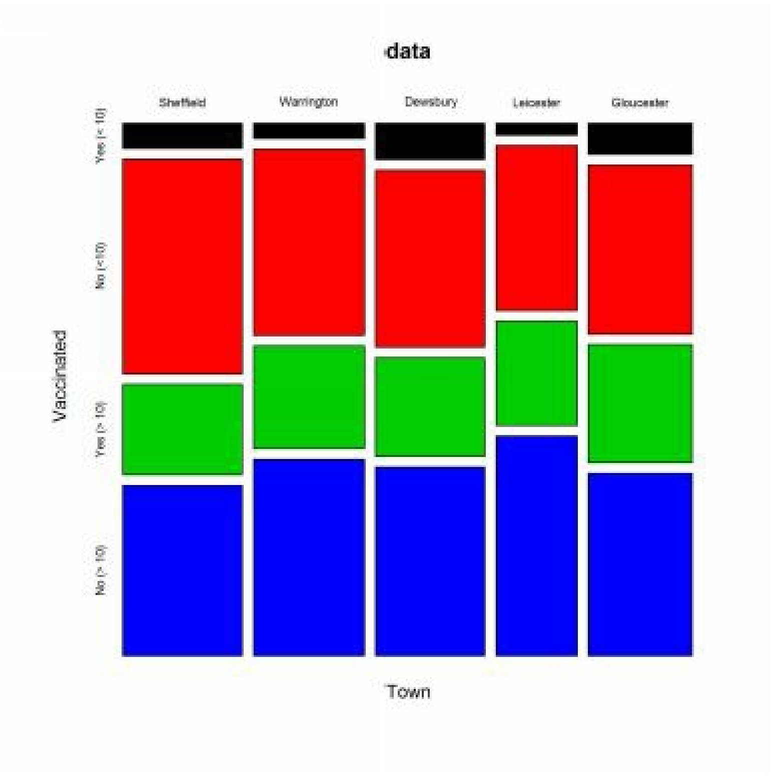


Figure 2 8 Mosiac plot of smallpox in five English towns

Figure 2 8 was created using the mosaicplot function contained in the R base package. It was built using the following three steps:

Step 1: Download and install the required packages.

```
require{graphics}
```

Step 2: Clean and prepare the data

```
r1 = c (7.9,    67.6,   28.3,   53.6)
r2 = c (4.4,    54.5,   29.9,   57.6)
r3 = c (10.2,   50, 27.7,   53.4)
r4 = c (2.5,    35.3,   22.2,   47)
r5 = c (8.5,    46.3,   32.2,   50)
data <- as.table(rbind(r1,r2,r3,r4,r5))

dimnames(data) <- list(Town = c("Sheffield","Warrington", "Dewsbury","Leicester","Gloucester")
, Vaccinated = c("Yes (< 10)","No (<10)", "Yes (> 10)","No (> 10)"))
```

Step 3: Plot the chart

```
mosaicplot(data, color = 1:4)
```

Step 1: Installs the required packaged and makes it available to R.

Step 2: Creates a table comparableto that shown in Figure 2 5 .

Step 3: The `mosaicplot` function draws the image. The colors are specified by the argument `color = 1:4`

Tip 1: Experiment with the color of the individual boxes by setting `color = 2:5` . What do you observe?

Tip 2: Occasional you may need to change the orientation of the text in the image. This can be achieved by using the las argument. For example, `mosaicplot(data, color = 1:4, las = 1)` produces the image in Figure 2 9 .

Tip 3: It is always useful to know of differentfunctions which produce similar types of chart. A mosaic plot can also be produced using the mosaic function in the package vcd.

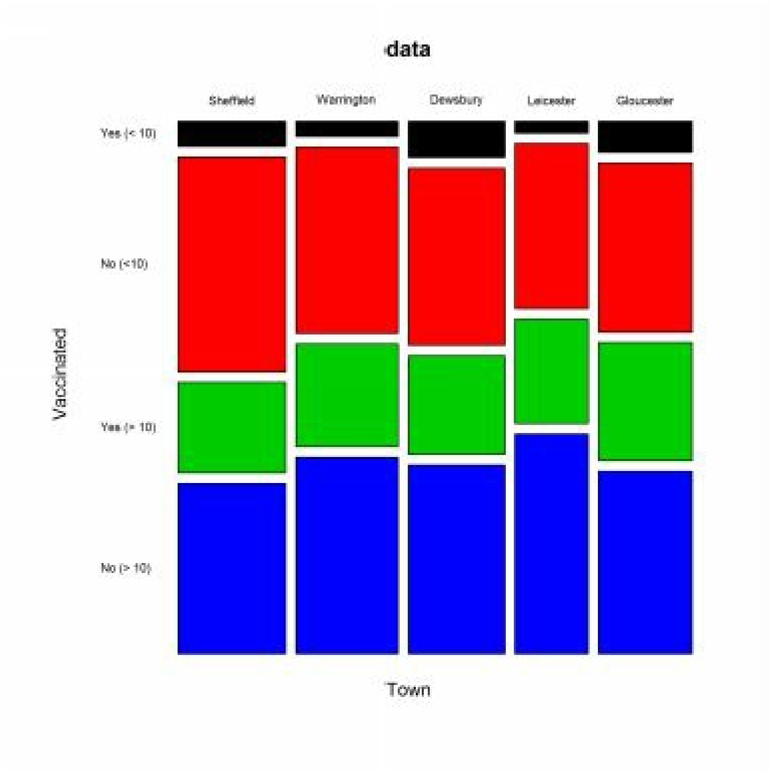


Figure 2 9 Mosiac plot using mosaicplot(data, color = 1:4, las = 1)

Quick reference R code

Figure 5 1 Box plot of the effect of Vitamin C on tooth growth

```
data(ToothGrowth)
boxplot(len ~ dose, data = ToothGrowth,
        boxwex = 0.25, at = 1:3 - 0.2,
        subset = supp == "VC", col = "red",
        xlab = "Vitamin C dose mg",
        ylab = "Tooth length",
        xlim = c(0.5, 3.5), ylim = c(0, 35), yaxs = "i")
boxplot(len ~ dose, data = ToothGrowth, add = TRUE,
        boxwex = 0.25, at = 1:3 + 0.2,
        subset = supp == "OJ", col = "cyan")
legend(2, 9, c("Ascorbic acid (VC)", "Orange Juice Group (OJ)"),
       fill = c("red", "cyan"))
```

Figure 5 2 2d contour plot of the effect of Vitamin C on tooth growth

```
require(reshape)
require(extracat)
data(ToothGrowth)
boxplot2g(ToothGrowth$dose, ToothGrowth$len, ToothGrowth$sup)
```

Figure 5 3 2d contour plot of Iris data

```
install.packages("extracat ")
require(extracat)
attach(iris)
boxplot2g(iris$Sepal.Length,iris$Sepal.Width,iris$Species)
```

Figure 5 4 relationship between tips and total bill given group size

```
install.packages("extracat ")
install.packages("reshape")
require(extracat)
require(reshape)
data(tips)
boxplot2g(tips$total_bill,tips$tip,tips$size)
```

Figure 5 5 relationship between tips and total bill given day of week

```
install.packages("extracat ")
install.packages("reshape")
require(extracat)
require(reshape)
data(tips)
boxplot2g(tips$total_bill,tips$tip,tips$day)
```

Further resources

Further reading on the extracat package can be found in:

Alexander Pilhoefer, Antony Unwin (2013). New Approaches in Visualization of Categorical Data: R Package extracat. Journal of Statistical Software, 53(7), 1-25. URL http://www.jstatsoft.org/v53/i07/.

For further details on the datasets mentioned in this chapter see:

Anderson, Edgar (1935). The irises of the Gaspe Peninsula, Bulletin of the American Iris Society, 59, 2–5.

R Core Team (2013). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Bryant, P. G. and Smith, M (1995) Practical Data Analysis: Case Studies in Business Statistics. Homewood, IL: Richard D. Irwin Publishing.

Wtite you notes below…

3. Explore similarity using trees

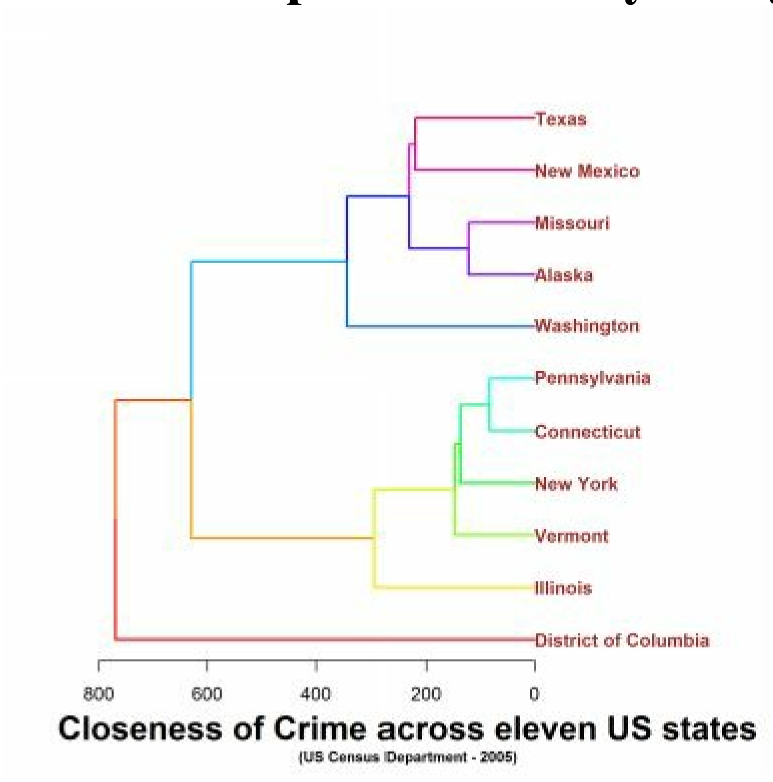


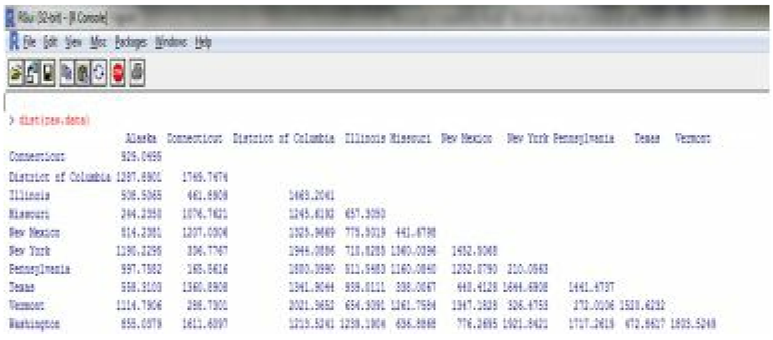
Figure 3 1 Dendrogram of major crime by various US states

A dendrogram (from Greek Dendron meaning tree, and gramma meaning drawing) is a tree diagram used to display the groups formed by hierarchical clustering. Hierarchical clustering is a group of statistical techniques that measure the similarity among a group of entities. These methods start with the calculation of the distances of each entity from all the other entities in a dataset. Groups or clusters are then formed usually by a process of agglomeration or division. With agglomeration algorithms, entities begin in a group of one with close entities merged until the entire dataset is in a single group. With division algorithms, all entities begin in a single group. They are then split into two groups, then each group is split until all entities are in a group of one. Measures of the distance between entities are typically calculated using Euclidean distance. Given p variables $x_1,\ldots,x_p$ measured on n entities Euclidean distance is calculated as:

$$d_{ij} = \sqrt{\left\{\sum_{k=1}^{p}\left[x_{ik} - x_{jk}\right]^2\right\}}$$

Where x_{ik} is the value of variable x_k for entity i and x_{jk} is the value of the same variable for entity j.

Figure 3 1 shows a dendrogram obtained from an average agglomeration method using crime data[iv] from eleven US states. A scale which measures the dissimilarity (distance) has been placed at the bottom of the chart. States which are close in terms of distance have branches coded in similar colors. The distance matrix for this data is shown in the image below. The two closest states were Connecticut and Pennsylvania. These are distance 165 apart. The next closet pair of states is Missouri and Alaska, which are 244 apart. Then come New York and Vermont which are 326 apart. The District of Columbia is the furthest apart or the least similar.

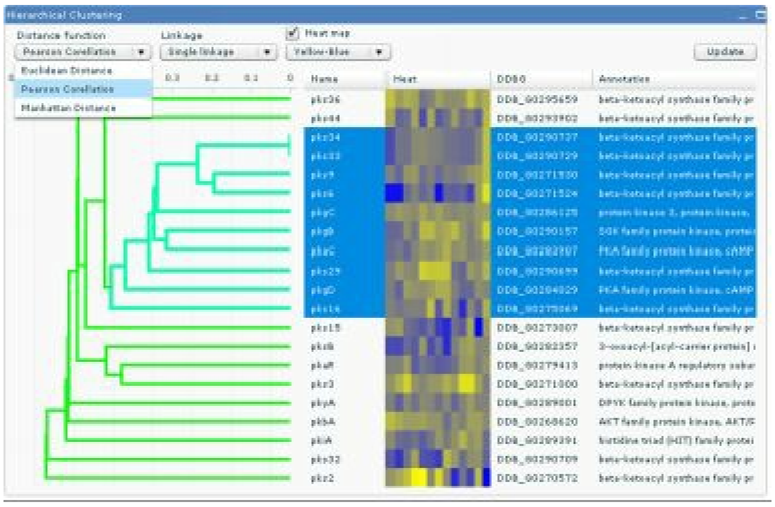

RGui (32-bit) - [R Console]
File Edit View Misc Packages Windows Help

```
> dist(new.data)
                        Alaska Connecticut District of Columbia  Illinois  Missouri New Mexico  New York Pennsylvania     Texas   Vermont
Connecticut           925.0456
District of Columbia 1297.8901   1769.7474
Illinois              508.5065    461.6909            1463.2041
Missouri              244.2358   1076.7621            1245.8192  657.3050
New Mexico            814.2381   1207.0306            1325.9469  775.3019  441.4798
New York             1130.2295    306.7767            1944.0896  718.8283 1360.0396  1452.9068
Pennsylvania          997.7582    165.0616            1880.3890  511.5483 1160.0840  1252.8790  210.0563
Texas                 558.3103   1360.8908            1341.9044  939.0121  338.0067   440.4128 1644.6908    1441.4737
Vermont              1114.7906    286.7301            2021.3652  654.3091 1261.7594  1347.1828  326.4758     272.0006 1520.6232
Washington            855.0078   1611.6097            1215.5241 1238.1904  636.8868   776.2695 1921.9421    1707.2619  472.8827 1893.5248
```

Innovative use of dendrograms

One of the major uses of dendrograms is to discern any sub- sets of observations whose members are both close to one another and/ or isolated from the rest. This ability has proved extremely useful in fields ranging from marketing and fake software detection to understanding genetic diversity in agricultural crops.

dictyExpres: The dictyExpres[v] is an interactive, web-based exploratory data analytics application providing access to over 1,000 Dictyostelium gene expression experiments from Baylor College of Medicine. The dictyExpres application can be used to find clusters of similarly expressed genes via dendrogram visualization. Dendrogram visualization taken directly from the web application is shown in Figure 3 2 .



Name	Heat	DDB0	Annotation
pks36		DDB_G0295659	beta-ketoacyl synthase family pr
pks44		DDB_G0293902	beta-ketoacyl synthase family pr
pks34		DDB_G0290727	beta-ketoacyl synthase family pr
pks33		DDB_G0290729	beta-ketoacyl synthase family pr
pks9		DDB_G0271530	beta-ketoacyl synthase family pr
pks6		DDB_G0271524	beta-ketoacyl synthase family pr
pkgC		DDB_G0286125	protein kinase 3, protein kinase,
pkgB		DDB_G0290157	SGK family protein kinase, protei
pkaC		DDB_G0282707	PKA family protein kinase, cAMP
pks29		DDB_G0290659	beta-ketoacyl synthase family pr
pkgD		DDB_G0284029	PKA family protein kinase, cAMP
pks16		DDB_G0275069	beta-ketoacyl synthase family pr
pks15		DDB_G0273007	beta-ketoacyl synthase family pr
pksB		DDB_G0282357	3-oxoacyl-[acyl-carrier protein] r
pkaR		DDB_G0279413	protein kinase A regulatory subur
pks3		DDB_G0271000	beta-ketoacyl synthase family pr
pkyA		DDB_G0289001	DPYK family protein kinase, prote
pkbA		DDB_G0268620	AKT family protein kinase, AKT/R
pkiA		DDB_G0289391	histidine triad (HIT) family protei
pks32		DDB_G0290709	beta-ketoacyl synthase family pr
pks2		DDB_G0270572	beta-ketoacyl synthase family pr

Figure 3 2 dictyExpres dendrogram visualization

Marketing science key words: Mela et al (2013) identy trends in marketing science through key words used in articles published in the journal "Marketing Science "from its inception in 1982 through 2011 – 30 volumes in total. For each journal article, the collected observations included the authors, volume, issue and date of publication, title, and the sets of key words chosen by the article's author(s). Analysis included 1085 articles by 1051 authors (or co-authors) resulting in a total of 4749 key word choices, of which 2415 were unique. Hierarchical cluster analysis of the twenty most frequently used words resulted in a dendrogram with four main clusters . Cluster 1 was the word Game theory ; Cluster 2 the words Channel, Competition, Competitive, strategy,Pricing and Retailing; Cluster 3 included the words Brand choice, Promotion,Buyer behavior, Econometric models and Choice models; the final cluster included the phrases Customer satisfaction, Price discrimination, Advertising, Bayesian analysis, Hierarchical Bayes, Conjoint analysis, Forecasting, New products and Diffusion. The researchers conclude individual key words were not independent.

Detecting fake software: Fake antivirus software and ransom ware trick internet users into yielding their credit card details or personal banking information. Credit cards are then charged or banking accounts raided. Dietrich et al (2013) use dendrograms to successfully investigate fake antivirus software and ransom ware. Using around 213,671 screenshots of executed malware samples, the investigators find the attacks can be can be grouped into subsets of structurally similar images, reflecting each ransom ware family or attack campaign.

Genetic diversity in Sudanese wheat: Fadoul et al (2013) investigate the genetic diversity among locally cultivated wheat in Sudan. Diversity was estimated by a Random Amplified Polymorphic DNA (RAPD) technique using 21 arbitrary 10-mer and six Dehydration Responsive Elements (DRE) primers. The genetic distance matrix ranged from 0.435 to 0.81 and was used to construct a dendrogram using average linkage between groups. The researchers observed wheat cultivars were clustered into six groups; Cluster A included cultivars Debaira, Nisr and Sasraib, cluster B included Nabta, Wadi Elniel, Argeen and Condor, cluster C included the cultivar Imam, cluster D contained the cultivar Al-neelain, cluster E included Tagana and Bohain and the most distant cultivar (Khalifa) in the dendrogram stands in cluster F. The highest similarity among the wheat cultivars was observed between Debaira and Nisr while the lowest was between Alneelain and Bohain.

Stock market similarity: The Dow Jones Industrial Average lists 30 stocks representative of the American market, the NASDAQ-100 is an index that tracks the 100 largest non-financial companies in the National Association of Securities Dealers Automated Quotations (NASDAQ) market, the FTSE100 is an index of the 100 companies with the largest capitalization traded in the London market, the Deutscher Aktien index (DAX) includes 30 German companies traded in the Frankfurt market, and the SSE-50 lists the 50 major Chinese companies traded in Shanghai. Tapinos and Mendes (2013) investigate the similarity of these markets using dendrogram visualization. They report the SSE-50 as the most dissimilar of all the indices.

Creating Dendrograms in R

Figure 3 1 was created using the ape package. It was constructed using data on major crimes (Murder, Rape, Robbery, Assault, Burglary Larceny, and Motor Theft) recorded for ten US states (Alaska, Connecticut, Illinois, Missouri, New Mexico, New York, Pennsylvania, Texas, Vermont and Washington) and District of Columbia). The data is a subset of a wider sample recorded for all states by the US census department. The dendrogram was built using the following three steps:

Step 1: Download and install the required packages.

```
Install.packages("ape")
require(ape)
```

Step 2: Clean and prepare the data

```
raw.data<-read.csv("http://www.auscov.com/uploads/5/4/2/2/5422365/crime2005.csv",header = TRUE)
rownames(raw.data)=raw.data[,1]
hc <- hclust(dist(raw.data), "average")
data <- as.phylo(hc)
```

Step 3: Plot the chart

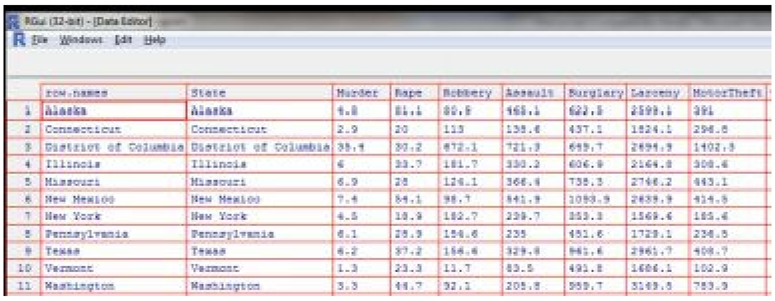
```
plot(data,     edge.color = rainbow(length(data$edge)/2),tip.color="brown",edge.width = 2,font=2)
title("Closeness of Crime across eleven US states",line=-29,cex.main=1.8)
title(" (US Census Department – 2005) ",line=-30,cex.main=0.8,col="grey")
axisPhylo(1, las = 1)
```

Step 1: Installs the required packaged and makes it available to R. If the package is already installed on your machine you do not need to use install.packages . However, the require command is needed at the start of a new R session.

Step 2: Loads the required data into the raw.data dataframe. It contains information on the state name and number of crimes. The data is contained in a csv file on this books website www.auscov.com. To view raw.data in a spreadsheet at the R prompt type:

```
edit(dow.jan.2005)
```

The image below will appear in the R consul:



	row.names	State	Murder	Rape	Robbery	Assault	Burglary	Larceny	MotorTheft
1	Alaska	Alaska	4.8	81.1	80.9	465.1	622.5	2599.1	391
2	Connecticut	Connecticut	2.9	20	113	138.6	437.1	1824.1	296.8
3	District of Columbia	District of Columbia	35.4	30.2	672.1	721.3	649.7	2694.9	1402.3
4	Illinois	Illinois	6	33.7	181.7	330.2	606.9	2164.8	308.6
5	Missouri	Missouri	6.9	28	124.1	366.4	738.3	2746.2	443.1
6	New Mexico	New Mexico	7.4	54.1	98.7	541.9	1093.9	2639.9	414.5
7	New York	New York	4.5	18.9	182.7	239.7	353.3	1569.6	185.6
8	Pennsylvania	Pennsylvania	6.1	28.9	154.6	235	451.6	1729.1	236.5
9	Texas	Texas	6.2	37.2	156.6	329.9	961.6	2961.7	408.7
10	Vermont	Vermont	1.3	23.3	11.7	83.5	491.8	1686.1	102.9
11	Washington	Washington	3.3	44.7	92.1	205.8	959.7	3149.5	783.9

The function hclust performs hierarchical cluster analysis on raw.data using a set of distances calculated by dist(raw.data) using method "average". The result is converted into a tree object of class "phylo" and stored in data . Note class "phylo" is required for printing dendrograms using ape package functionality.

Step 3: The plot function, with a number of arguments, is used to draw the dendrogram. The argument edge.color determines the color of the edges of the tree. To ensure similar colors for close observations we used the rainbow palette. The color of the dendrogram text labels is determined by tip.color. The argument edge.width determines the width or thickness ofthe edges, the larger the thicker the edge. Finally, font is an integer specifying the type of font for the labels: 1 = plain text, 2 = bold, 3 = italic, 4 =bold italic.

Tip 1: There is no generic way to determine the best dendrogram. It is therefore advisable build your cluster with the function hclust using variousmethods and then observe the difference. We used "average" , other arguments to try include "ward", "single", "complete", "mcquitty", "median" or "centroid".

Easily create cladograms,fans,unrooted & radial trees

A super quick way to create a cladogram, fan style plot, unrooted tree or radialtree is to use the type argument in the plot function. Each of the respective plots can be obtained by setting type= "cladogram" , "fan" , "unrooted" or "radial" . The following lines of code illustrate each type.

We begin with a radial tree, shown in Figure 3 3 . It was constructed by typing:

```
plot(data,    edge.color = rainbow(length(data$edge)/2),tip.color="brown",edge.width = 2,font=2,type=
"radial",no.margin=TRUE,cex=0.8)
```

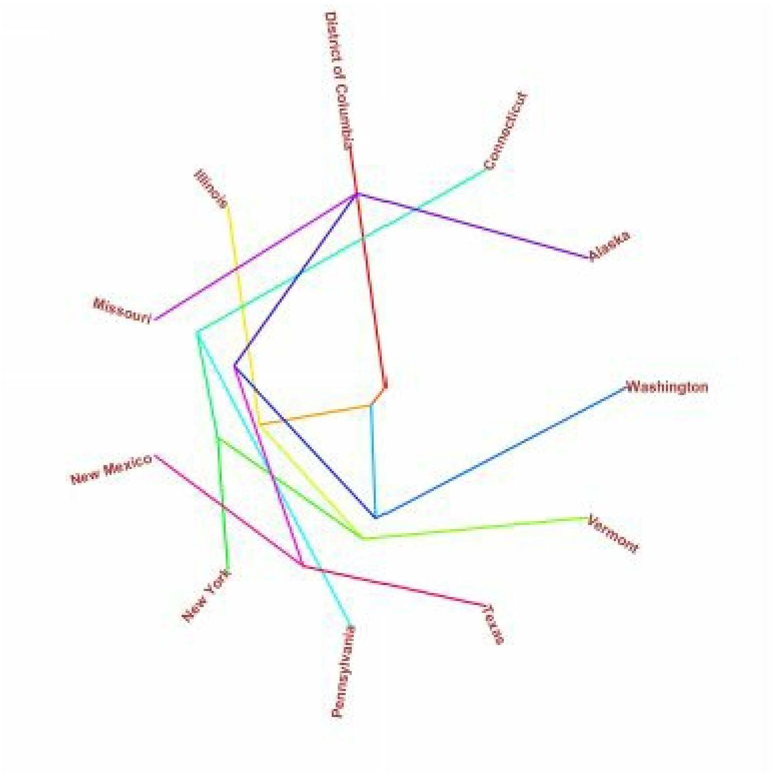


Figure 3 3 Radial tree

The fan tree shown in Figure 3 4 can be easily created by entering:

```
plot(data,    edge.color = rainbow(length(data$edge)/2),tip.color="brown",edge.width = 2,font=2,type=
"fan",no.margin=TRUE)
```

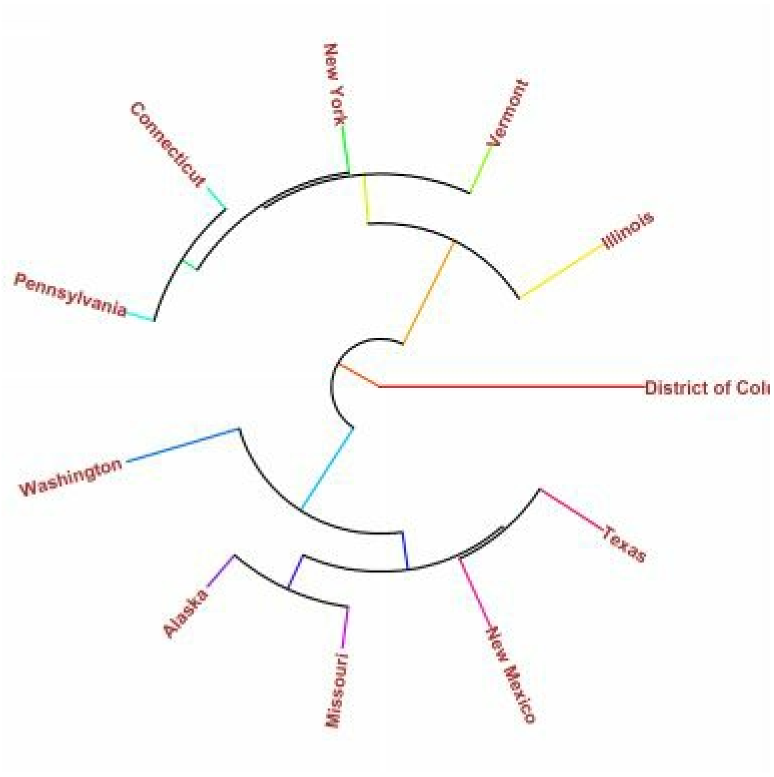


Figure 3 4 Fan tree

Using the same method the cladogram shown in Figure 3 5 was created by entering:

```
plot(data,     edge.color = rainbow(length(data$edge)/2),tip.color=”brown”,edge.width = 2,font=2,type=
“cladogram”,no.margin=TRUE)
```

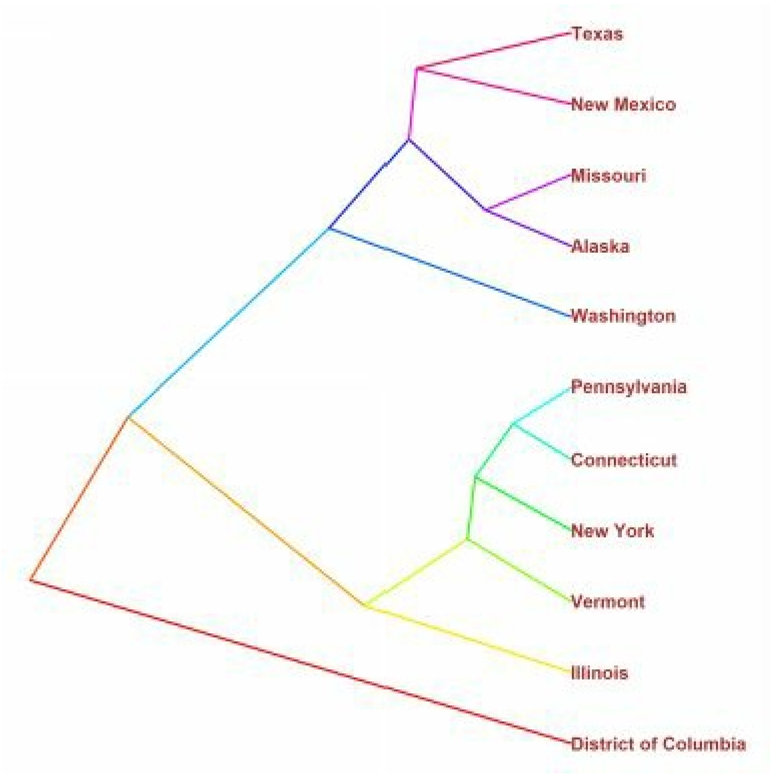


Figure 3 5 Basic cladogram

Finally, the unrooted tree shown in Figure 3 6 was created by typing:

```
plot(data, edge.color = “red”,tip.color=”brown”,edge.width =
2,font=1,type=”unrooted”,cex=.7,no.margin=TRUE)
```

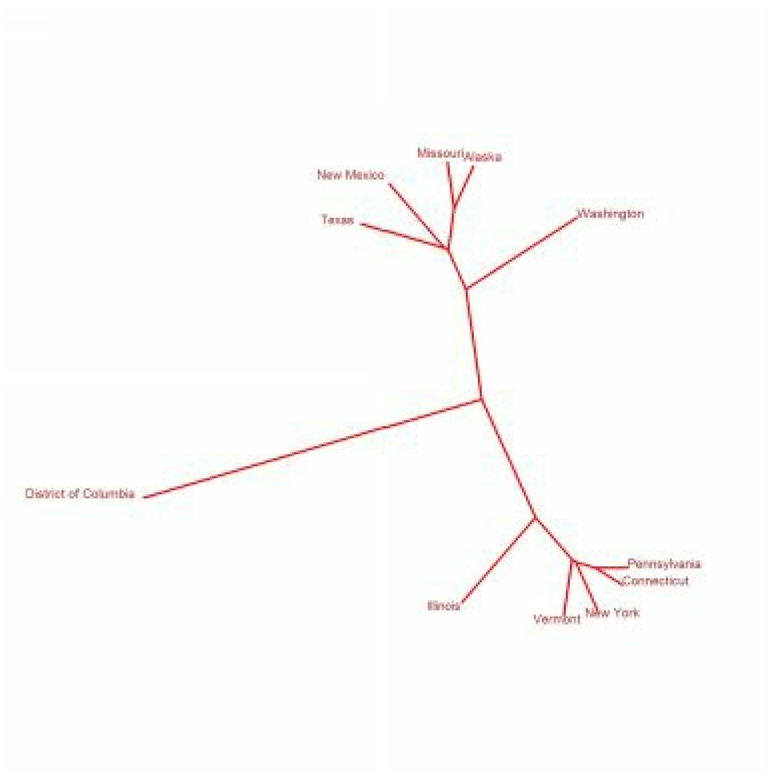


Figure 3 6 Simple unrooted tree

Summary

A dendrogram is a tree diagram used to display the groups formed by hierarchical clustering. One of the major uses of dendrograms is to permit the viewer to perform intuitive visual cluster analysis, that is, to discern any sub-sets of observations whose members are both close to one another and/ or isolated from the rest. This ability has proved extremely useful in fields ranging from marketing, fake software detection to understanding genetic diversity in agricultural crops.Flexible dendrograms can be created rapidly, easily and efficiently using the ape package.

Quick reference R code

Figure 3 1 Dendrogram of major crime by various US states

```
install.packages("ape")
require(ape)
raw.data<-read.csv("http://www.auscov.com/uploads/5/4/2/2/5422365/crime2005.csv",header = TRUE)
rownames(raw.data)=raw.data[,1]
hc <- hclust(dist(raw.data), "average")
data <- as.phylo(hc)
plot(data,     edge.color = rainbow(length(data$edge)/2),tip.color="brown",edge.width = 2,font=2)

title("Closeness of Crime across eleven US states",line=-29,cex.main=1.8)
title(" (US Census Department – 2005) ",line=-30,cex.main=0.8,col="grey")
axisPhylo(1, las = 1)
```

Figure 3 3 Radial tree

```
install.packages("ape")
require(ape)
raw.data<-read.csv("http://www.auscov.com/uploads/5/4/2/2/5422365/crime2005.csv",header = TRUE)
rownames(raw.data)=raw.data[,1]
hc <- hclust(dist(raw.data), "average")
data <- as.phylo(hc)
plot(data,     edge.color = rainbow(length(data$edge)/2),tip.color="brown",edge.width = 2,font=2,type=
"radial",no.margin=TRUE,cex=0.8)
```

Figure 3 4 Fan tree

```
install.packages("ape")
require(ape)
raw.data<-read.csv("http://www.auscov.com/uploads/5/4/2/2/5422365/crime2005.csv",header = TRUE)
rownames(raw.data)=raw.data[,1]
hc <- hclust(dist(raw.data), "average")
data <- as.phylo(hc)
plot(data,     edge.color = rainbow(length(data$edge)/2),tip.color="brown",edge.width = 2,font=2,type=
"fan",no.margin=TRUE)
```

Figure 3 5 Basic cladogram

```
install.packages("ape")
require(ape)
raw.data<-read.csv("http://www.auscov.com/uploads/5/4/2/2/5422365/crime2005.csv",header = TRUE)
rownames(raw.data)=raw.data[,1]
hc <- hclust(dist(raw.data), "average")
data <- as.phylo(hc)
plot(data,     edge.color = rainbow(length(data$edge)/2),tip.color="brown",edge.width = 2,font=2,type=
"cladogram",no.margin=TRUE)
```

Figure 3 6 Simple unrooted tree

```
install.packages("ape")
require(ape)
raw.data<-read.csv("http://www.auscov.com/uploads/5/4/2/2/5422365/crime2005.csv",header = TRUE)
rownames(raw.data)=raw.data[,1]
hc <- hclust(dist(raw.data), "average")
data <- as.phylo(hc)
plot(data, edge.color = "red",tip.color="brown",edge.width =
2,font=1,type="unrooted",cex=.7,no.margin=TRUE)
```

Further resources

Further reading on applications of dendrograms can be found in:

Dietrich, Christian J., Christian Rossow, and Norbert Pohlmann. "Exploiting Visual Appearance to Cluster and Detect Rogue Software." (2013).

Fadoul, Hind E., A. Marmar, and Adil A. El Hussein. "Assessment of genetic diversity among Sudanese wheat cultivars using RAPD markers."(2013)

Mela, Carl F., Jason Roos, and Yiting Deng. "Invited Paper—A Keyword History of Marketing Science." Marketing Science 32.1 (2013): 8-18.

Tapinos, Avraam, and Pedro Mendes. "A Method for Comparing Multivariate Time Series with Different Dimensions." PloS one 8.2 (2013): e54201.

For further details on the ape package see:

Paradis E., Claude J. & Strimmer K. 2004. APE: analyses of phylogenetics and evolution in R language. Bioinformatics 20: 289-290.

-***Wtite you notes below…***

4. How to illuminate significance in hierarchical data

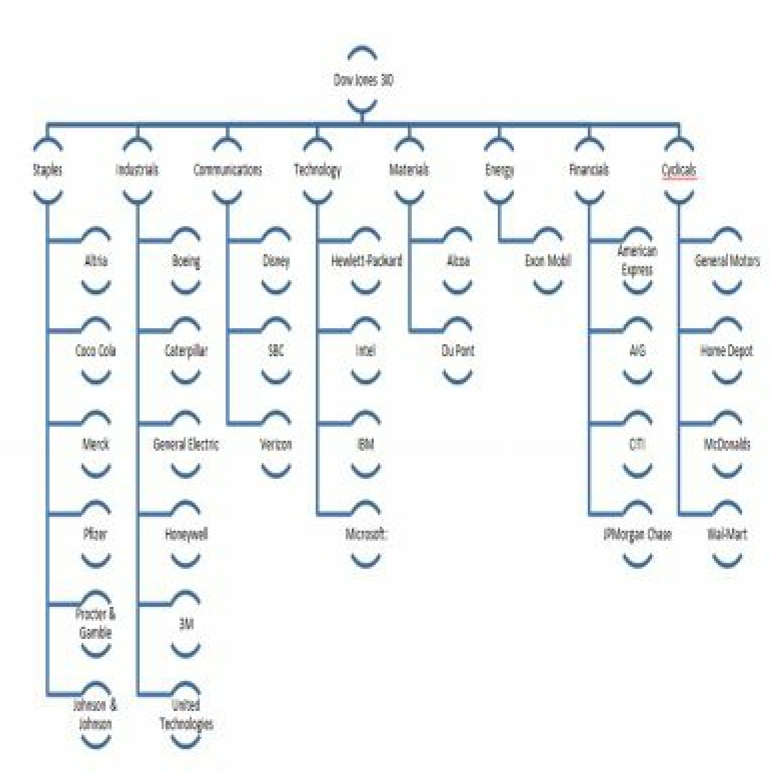


Figure 4 1 Hierarchical (tree-structured) data for the Dow-Jones Index

Treemaps display hierarchical (tree-structured) data in a colored mosaic form containing embedded, rectangular shapes. For example, Figure 4 1 captures the structure of Dow Jones industrial index from sector and stock data. The main branches of the image represent the eight sectors that make up the Dow Jones index (Communications, Cyclicals, Energy, Financials, Industrials Materials, Staples, Technology and Utilities), with individual stocks as leaves or end nodes. In a treemap the branch of tree structured data is given a rectangle, which is then tiled with smaller rectangles representing sub-branches or end leaves - see for example Figure 4 2 . A leaf node's rectangle has an area proportional to a specified dimension of the data. The color and size dimensions of the rectangles allow for the identification of patterns that might be difficult to spot using standard bar charts and graphs. A key advantage of using treemaps for large datasets is that they can legibly display thousands of items simultaneously.

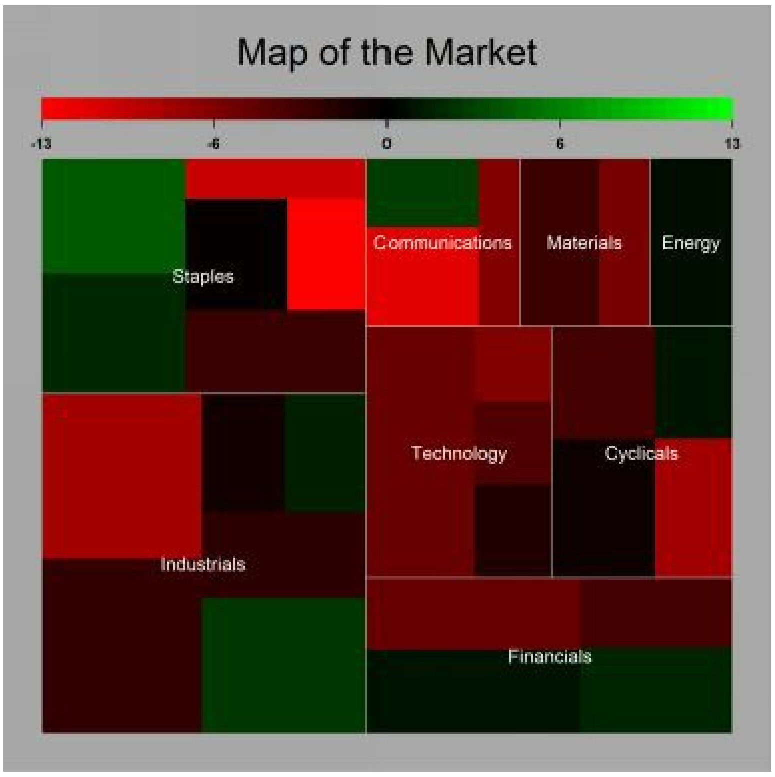


Figure 4 2 Treemap of Dow-Jones Index as at the end of January 2005

Figure 4 2 presents a treemap for the Dow Jones Industrial Average. It represents this data as eight large rectangular regions; one each for each of eight categories making up the Dow Jones Industrial Average. The treemap partitions the image into a collection of box-shaped elements, forming a mosaic. Each inner box represents an individual stock (of which there are thirty in the index) and is sized according it's price. For example, financials consists of four stocks and are represented by four colored rectangular boxes, each with a shading based on the price return for January 2005. Price returns are represented using a red through green scale; the higher the return the greener the box.

From Figure 4 2 the hierarchy of the Dow Jones Industrial Average can be easily seen; as can the relative number of stocks for each sector. The dominant sectors and stocks in terms of price can be quickly identified by the size of rectangle. As can the relative performance of each sector and individual stock.

Innovative use of treemaps

This ability to represent a hierarchy of data makes the treemap visualization both efficient and effective. Originally designed to visualize file space on very large computer-storage disk devices, treemaps are now used in a diverse range of fields including human-gene research, social network analysis and finance. Treemaps are growing rapidly in popularity as data scientists, statisticians and computer analysts discover their power to convey complex information from extremely large data sources. We outline below a number of creative uses for treemaps:

Visualization of news:

Newsmap[vi] visualizes the constantly changing landscape of news. It uses a treemap visualization algorithm to picture the very large volume of information gathered through News aggregators, see Figure 4 3. Interactions between news items are visualized in a dynamic and constantly changing layout. The placement of each news item and the total area it covers depends on the number of news sources that report it.

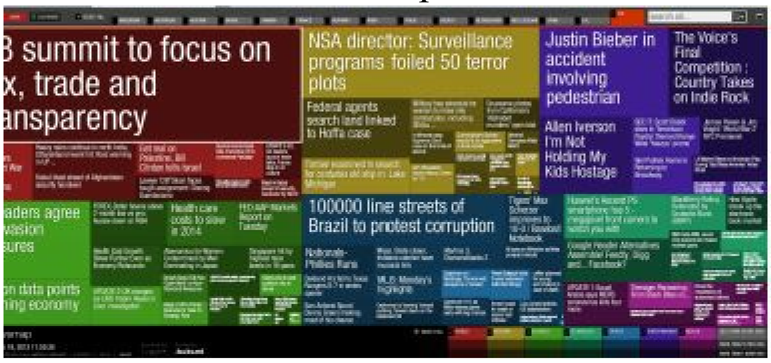

8 summit to focus on x, trade and ansparency
NSA director: Surveillance programs foiled 50 terror plots
Justin Bieber in accident involving pedestrian
The Voice's Final Competition : Country Takes on Indie Rock
Federal agents search land linked to Hoffa case
Allen Iverson I'm Not Holding My Kids Hostage
aders agree vasion sures
Health care costs to slow in 2014
100000 line streets of Brazil to protest corruption

Figure 4 3 Newsmap's treemap visualization of US news

Information visualization in digital Libraries: The exponential growth in digital content in libraries requires powerful tools to allow efficient navigation of content. Wan (21013) explore the role of treemaps in the design of user-friendly interfaces. They develop a visual treemap system to present collections of documents to users.

Workflow management: Oliva, Gustavo, et al (2013) use treemaps to support the management of complex workflow repositories. They are used to depict the change impacts of the complete workflow repository, with rectangles representing a specific section of the workflow process. The researchers report the use of treemaps makes it easy and quick to identify repository sections that require additional attention.

Compounds in traditional Chinese medicines: As part of a study to understand the structural features of natural compounds from traditional Chinese medicines, the scaffold architectures of drug-like compounds in MACCS-II Drug Data

Report (MDDR), non-drug-like compounds in Available Chemical Directory (ACD), and natural compounds in Traditional Chinese Medicine Compound Database (TCMCD) were explored and compared using treemaps by Tian et al (2013). The researchers visualize the structural diversity of the scaffolds within the overall dataset. They observe the distributions of the level 1 scaffolds of MDDR are sparser than those of ACD and TCMCD, indicating a higher structural diversity of the scaffolds in MDDR. They also report, some level 1 scaffolds with the highest frequencies of MDDR are similar to those of TCMCD, indicating that some ring substructures extracted from TCMCD may serve as valuable substructure resource for drug discovery.

Semantic web data search: Non expert users can find it difficult to explore and use semantic web data as the available datasets are often monolithic with files which can often only be explored using complex semantic queries. One solution is to design user interfaces to the data, however these are often very elaborate and difficult to navigate. Treemaps offer a natural solution because they illuminate what kind of entities are contained in a dataset, how they are interrelated, what are the main properties, values and so on. Brunetti et al (2013) exploit these properties to design treemaps to make semantic web data more accessible to non-semantic web users.

How to create a treemap using map.market

The map.market function in the package portfolio offers a simple and often the most convenient approach to rapid treemap construction. Figure 4 2 was created using this function. It was built using the following three steps:

Step 1: Download and install the required packages.

```
install.packages("portfolio")
require(portfolio)
```

Step 2: Clean and prepare the data

```
data(dow.jan.2005)
```

Step 3: Plot the chart

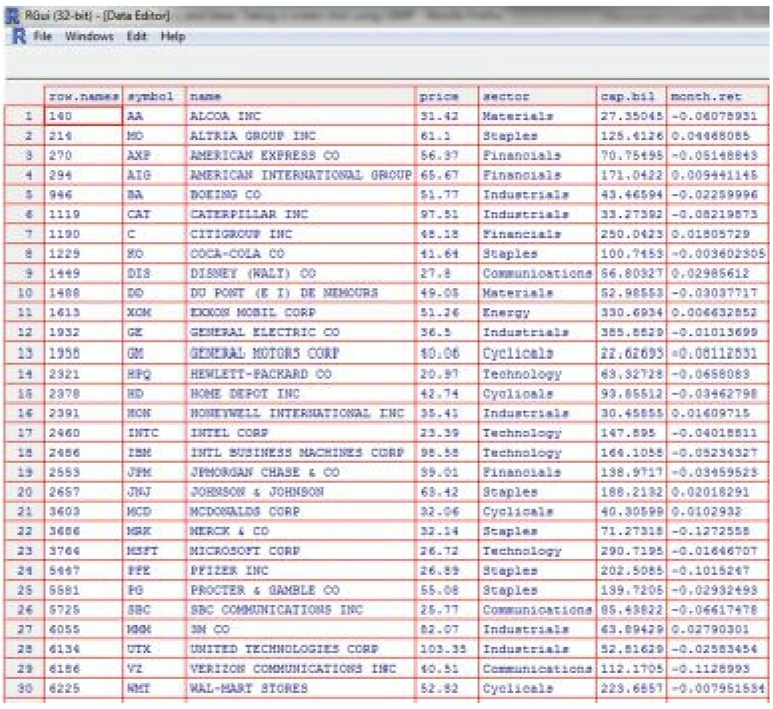
```
map.market(id    = dow.jan.2005$symbol,
          area  = dow.jan.2005$price,
          group = dow.jan.2005$sector,
          color = 100 * dow.jan.2005$month.ret)
```

Step 1: Installs the required packaged and makes it available to R. If the package is already installed on your machine you do not need to use install.packages . However, the require command is needed at the start of a new R session.

Step 2: Loads the dow.jan.2005 dataframe in R. It contains information on the name, stock market symbol, price and business sector of the thirty stocks in the Dow Jones Industrial index as at the end of January 2005. To examine the dataframe in a spreadsheet type format type:

```
edit(dow.jan.2005)
```

The image below will appear in the R consul:



	row.names	symbol	name	price	sector	cap.bil	month.ret
1	140	AA	ALCOA INC	31.42	Materials	27.35045	-0.06078931
2	214	MO	ALTRIA GROUP INC	61.1	Staples	125.4126	0.04468085
3	270	AXP	AMERICAN EXPRESS CO	56.37	Financials	70.75495	-0.05148843
4	294	AIG	AMERICAN INTERNATIONAL GROUP	65.67	Financials	171.0422	0.009441145
5	946	BA	BOEING CO	51.77	Industrials	43.46594	-0.02259996
6	1119	CAT	CATERPILLAR INC	97.51	Industrials	33.27392	-0.08219873
7	1190	C	CITIGROUP INC	48.18	Financials	250.0423	0.01805729
8	1229	KO	COCA-COLA CO	41.64	Staples	100.7453	-0.003602305
9	1449	DIS	DISNEY (WALT) CO	27.8	Communications	56.80327	0.02985612
10	1488	DD	DU PONT (E I) DE NEMOURS	49.05	Materials	52.98553	-0.03037717
11	1613	XOM	EXXON MOBIL CORP	51.26	Energy	330.6934	0.004632852
12	1932	GE	GENERAL ELECTRIC CO	36.5	Industrials	385.8829	-0.01013699
13	1958	GM	GENERAL MOTORS CORP	40.06	Cyclicals	22.62693	-0.08112831
14	2321	HPQ	HEWLETT-PACKARD CO	20.97	Technology	63.32728	-0.0658083
15	2378	HD	HOME DEPOT INC	42.74	Cyclicals	93.85512	-0.03462798
16	2391	HON	HONEYWELL INTERNATIONAL INC	35.41	Industrials	30.45855	0.01609715
17	2460	INTC	INTEL CORP	23.39	Technology	147.895	-0.04018811
18	2486	IBM	INTL BUSINESS MACHINES CORP	98.58	Technology	164.1055	-0.05234327
19	2553	JPM	JPMORGAN CHASE & CO	39.01	Financials	138.9717	-0.03459523
20	2657	JNJ	JOHNSON & JOHNSON	63.42	Staples	188.2132	0.02018291
21	3603	MCD	MCDONALDS CORP	32.06	Cyclicals	40.30599	0.0102932
22	3686	MRK	MERCK & CO	32.14	Staples	71.27318	-0.1272558
23	3764	MSFT	MICROSOFT CORP	26.72	Technology	290.7195	-0.01646707
24	5447	PFE	PFIZER INC	26.89	Staples	202.5085	-0.1015247
25	5581	PG	PROCTER & GAMBLE CO	55.08	Staples	139.7205	-0.02932493
26	5725	SBC	SBC COMMUNICATIONS INC	25.77	Communications	85.43822	-0.04617478
27	6055	MMM	3M CO	82.07	Industrials	63.89429	0.02790301
28	6134	UTX	UNITED TECHNOLOGIES CORP	103.35	Industrials	52.81629	-0.02583454
29	6186	VZ	VERIZON COMMUNICATIONS INC	40.51	Communications	112.1705	-0.1128993
30	6225	WMT	WAL-MART STORES	52.82	Cyclicals	223.6857	-0.007951534

Step 3: The map.market function takes four arguments. The first is the unique identify for each data item. In the case of the dow.jan.2005 dataframe this is the stock identification symbol obtained by dow.jan.2005$symbol. The second argument calculates the area of the rectangles; we used the stock price contained in dow.jan.2005$price as the size variable. The third argument is used to calculate the color of the rectangles; the monthly return of each stock, accessed by dow.jan.2005$month.ret and scaled by 100 was used to color.

Economic wealth & population

The dataframe GNI2010 in the package treemap contains information on population and gross national income for 208 countries in 2010 collected by the world bank.

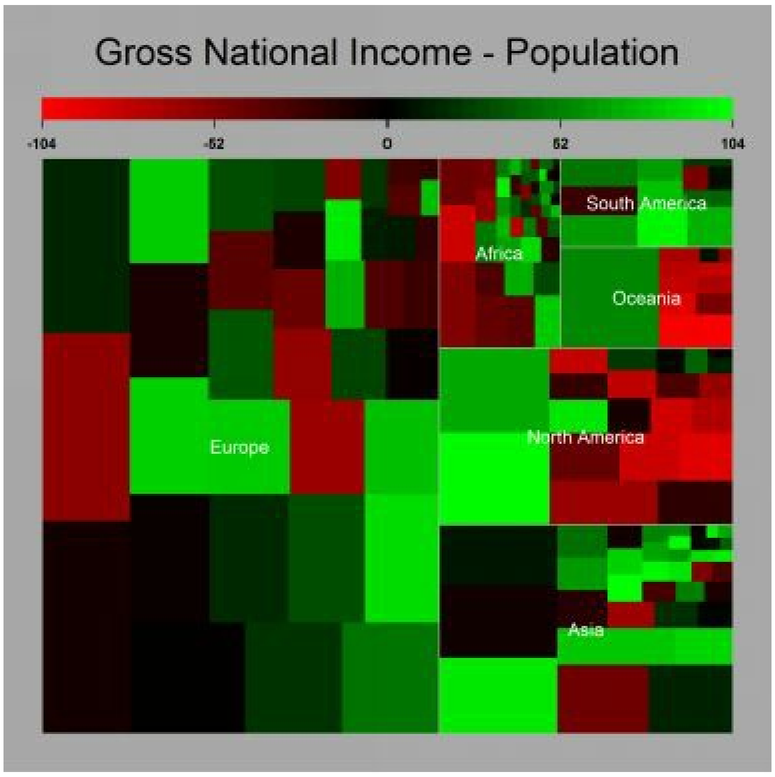


Figure 4 4 Treemap of economic wealth colored by relative population

Figure 4 4 illustrates the treemap obtained using this data. It was created using the following three steps:

Step 1: Download and install the required packages.

```
install.packages("portfolio")
install.packages("treemap")
```

Step 2: Clean and prepare the data

```
require("portfolio")
require("treemap")
data(GNI2010)
z=rank(GNI2010$population)
z=z-104
```

Step 3: Plot the chart

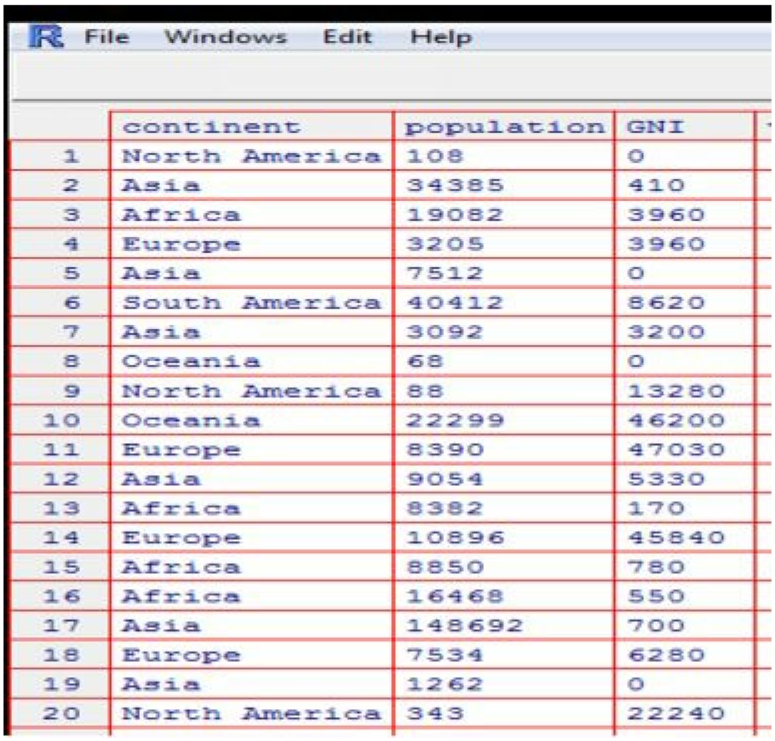
```
map.market(id   = GNI2010$iso3,
        area  = GNI2010$GNI+1,
        group = GNI2010$continent,
        color = z,
        main="Gross National Income - Population")
```

Step 1: Installs the required packaged and makes it available to R. If the package is already installed on your machine you do not need to use install.packages . However, the require command is needed at the start of a new R session.

Step 2: Loads the GNI2010 dataframe into R. It contains information on the country name, unique world bank identifier, population and gross national income. To examine the dataframe in a spreadsheet type format type:

edit(GNI2010)

The image below will appear in the R consul:



	continent	population	GNI
1	North America	108	0
2	Asia	34385	410
3	Africa	19082	3960
4	Europe	3205	3960
5	Asia	7512	0
6	South America	40412	8620
7	Asia	3092	3200
8	Oceania	68	0
9	North America	88	13280
10	Oceania	22299	46200
11	Europe	8390	47030
12	Asia	9054	5330
13	Africa	8382	170
14	Europe	10896	45840
15	Africa	8850	780
16	Africa	16468	550
17	Asia	148692	700
18	Europe	7534	6280
19	Asia	1262	0
20	North America	343	22240

The variable z is used to create a scale for the color of the rectangles. It is based on the rank population size, adjusted so it lies in the range -104 to +104. This adjustment is made to emphasize the visual contrast in colors.

Step 3: The map.market function takes four arguments. The first is the unique identify for each data item. In this case the unique country identifier contained obtained from GNI2010$iso3 . The second argument calculates the area of the rectangles; we used gross national income contained in GNI2010$GNI . The variable z measures the relative rank population size ; it is used to calculate the color of the rectangles. Finally, the title of the chart was created using main="Gross National Income - Population" .

Customizing map.market

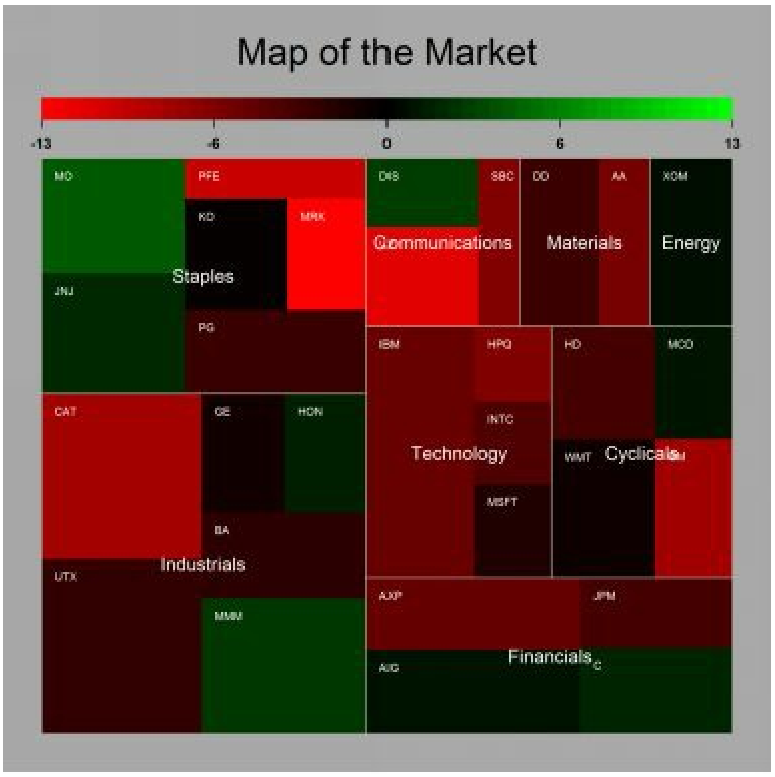


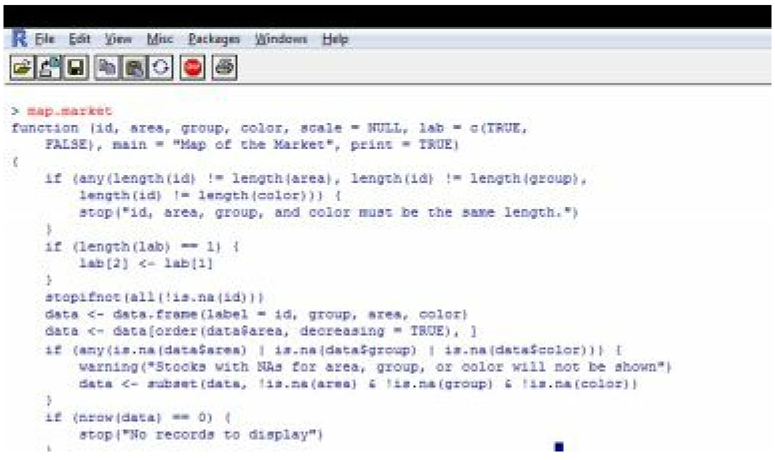
Figure 4 5 Treemap of Dow-Jones Index with stock labels

Tip 1: The most common request is for the individual labels of data elements to be represented on the treemap. This can be achieved by adding the argument `lab=TRUE` to the `map.market` function. For example, to add labels to Figure 4 2 type:

```
data(dow.jan.2005)
map.market(id    = dow.jan.2005$symbol,
        area  = dow.jan.2005$price,
        group = dow.jan.2005$sector,
        color = 100 * dow.jan.2005$month.ret,
        lab=TRUE)
```

The resultant image is shown in Figure 4 5.

Tip 2: It is possible to change the colors used in the `map.market` treemap visualization. This requires a little hacking of the code, although it is quite straightforward. We will change the background from grey to white, have the text appear in blue and use yellow-green colors for positive values and orange - dark red for negative values. To begin enter `map.market` at the R prompt. You should see something along the lines of the image shown below:



```
> map.market
function (id, area, group, color, scale = NULL, lab = c(TRUE,
    FALSE), main = "Map of the Market", print = TRUE)
{
    if (any(length(id) != length(area), length(id) != length(group),
        length(id) != length(color))) {
        stop("id, area, group, and color must be the same length.")
    }
    if (length(lab) == 1) {
        lab[2] <- lab[1]
    }
    stopifnot(all(!is.na(id)))
    data <- data.frame(label = id, group, area, color)
    data <- data[order(data$area, decreasing = TRUE), ]
    if (any(is.na(data$area) | is.na(data$group) | is.na(data$color))) {
        warning("Stocks with NAs for area, group, or color will not be shown")
        data <- subset(data, !is.na(area) & !is.na(group) & !is.na(color))
    }
    if (nrow(data) == 0) {
        stop("No records to display")
```

This is the R code contained in the map.market function. Copy the entire output to a text editor, feel free to use the R text editor or any other that you wish. The first line of code looks something like this:

```
function (id, area, group, color, scale = NULL, lab = c(TRUE,
    FALSE), main = "Map of the Market", print = TRUE)
```

We will create a new function to draw our required treemap. This can be achieved in four steps:

Step 1: Add the term map.market.color<- to the start of the code. Your edited code should now look something like this:

```
map.market.color<-
function (id, area, group, color, scale = NULL, lab = c(TRUE,
    FALSE), main = "Map of the Market", print = TRUE)
```

You have just created a new function called map.market.color.

Step 2: Look for the following lines of code (you should find them around line 165 in your text editor):

```
mapmarket <- gTree(name = "MAPMARKET", children = gList(rectGrob(gp = gpar(col = "dark grey",
      fill = "dark grey"), name = "background"), top.tree,
      map.tree))
```

Change the phase "dark grey" to "white" . The background color has been changed from dark grey to white!

Step 3: Now let's change the color of the text from white to blue. Change "white" to "blue" in the statement (around line 124):

```
label = this.data$label[s], gp = gpar(col = "white")
```

Next, change "white" to "blue" in statement (around line 133):

```
name = "label", gp = gpar(col = "white")
```

Step 4: Finally, identify the statements (around line 44):

```
color.ramp.pos <- colorRamp(c("black", "green"))

   color.ramp.neg <- colorRamp(c("black", "red"))
```

These two lines of code are responsible for the color of positive and negative numbers. Change the lines of code so that they state:

```
color.ramp.pos <- colorRamp(c("yellow", "darkgreen"))
   color.ramp.neg <- colorRamp(c("orange", "darkred"))
```

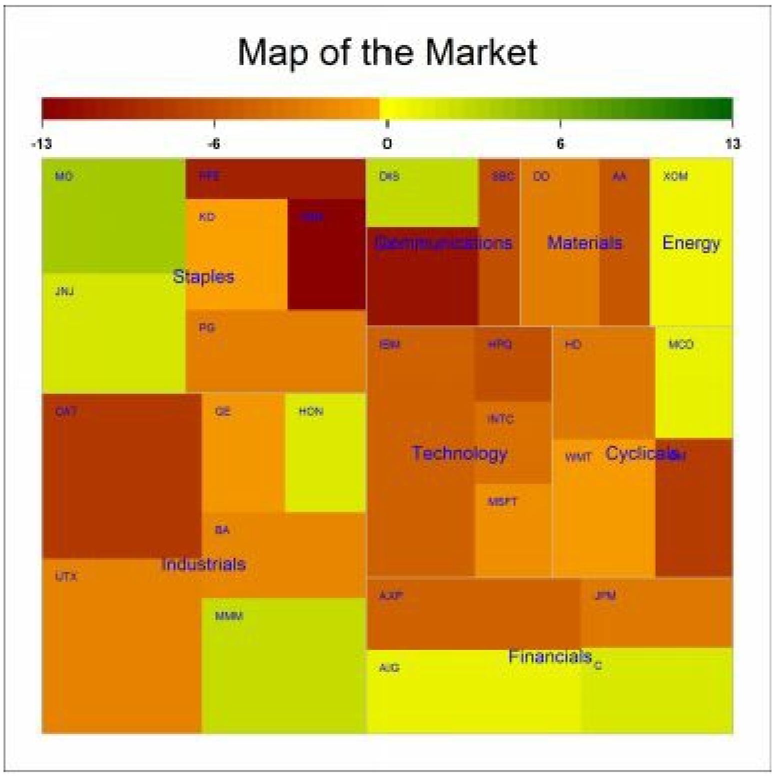


Figure 4 6 Customized map.market treemap

The edited function is complete. Save it and then copy into the R consol. Now you are ready to execute the code. Enter the following into the R consul and you will see the image shown in Figure 4 6:

```
data(dow.jan.2005)
map.market.color (id    = dow.jan.2005$symbol,
         area  = dow.jan.2005$price,
         group = dow.jan.2005$sector,
         color = 100 * dow.jan.2005$month.ret,
         lab=TRUE)
```

How to create a treemap using treemap

The treemap function in the package treemap provides a useful alternative for creating a treemap in R. We illustrate the use of this function using data from the London datastore. This is a free to access website created by the Greater London Authority as a portal for London's data. It is hosted at http://data.london.gov.uk/ with home page similar to that shown below.

GREATERLONDONAUTHORITY
london.gov.uk
london datastore
Search Datastore
RSS feeds
London Dashboard
Datastore
Datasets
Catalogue
Organisations
Categories
Keywords
Search Datasets
Request Dataset
Popular Suggestions
Latest Suggestions
Blog
Inspirational Uses
Get the 2011 Census first results
How is the capital performing? See the London Dashboard
Browse the useful apps and tools created using our data
Get the 2011 Census first results
Looking for the LSEO?
Welcome to the London Datastore
London Skills and Employment Observatory
London Skills & Employment Observatory

Let's make a withdrawal from the store. Our goal is to create a treemap using data on the political control, income and unemployment rate for each London borough. To do this we will use the London boroughs dataset located at: http://data.london.gov.uk/datastore/package/london-borough-profiles . Figure 4 7 illustrates the treemap created using this dataset. It was created using the following three steps:

Step 1: Download and install the required packages.

```
install.packages("treemap")
require("treemap")
```

Step 2: Clean and prepare the data

```
raw.data<-read.csv("http://data.london.gov.uk/datafiles/demographics/london-borough-profiles.csv")
data<-cbind(raw.data[,1:2],as.numeric(as.character(raw.data[,25])),as.numeric(as.character(raw.data[,31]))
data<-data[-1,]
data<-data[-1,]
data<-data[-37,]
data<-data[-36,]
data<-data[-35,]
data<-data[-34,]
data<-data[-33,]
column.names<-c("id","borough","unemp","income","control")
colnames(data)<-column.names
```

Step 3: Plot the chart

```
treemap(data,
      index=c("control", "borough"),
      vSize="income",
      vColor="unemp",
      type="value",
      palette="RdBu",
      title = "London Boroughs: Political Control,Unemployment and Income",
    title.legend = "Unemployment Rate (%)"
      )
```

Explaining each step

Step 1: Installs the required packaged and makes it available to R. The require command is necessary at the start of each new R session.

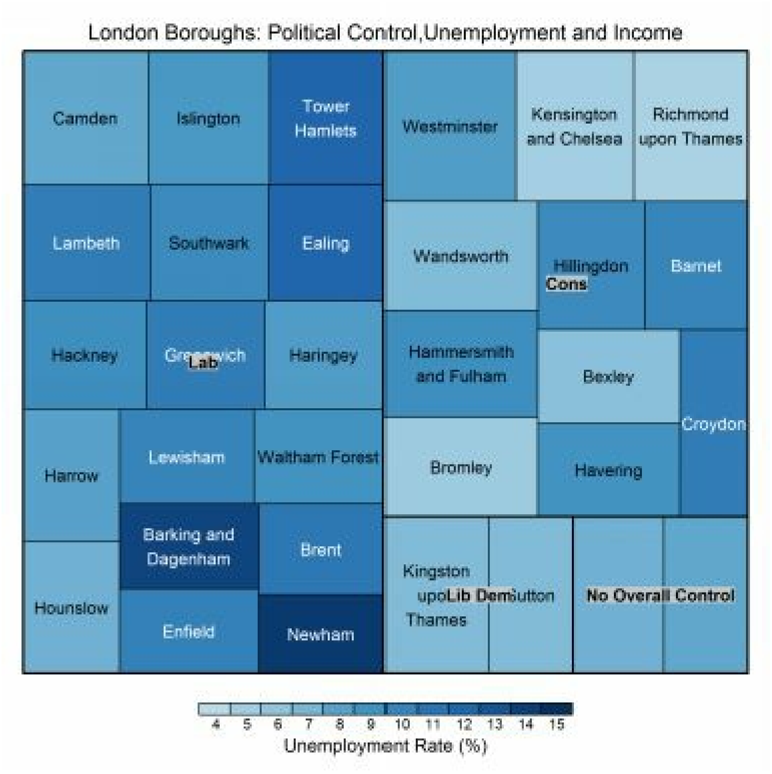


Figure 4 7 Treemap using the treemap function

Step 2: Reads a comma-separated values file (csv)from the London Datastore into the raw.data dataframe. The dataframe consists of 69 columns of demographic, economic and political data for all of the London Boroughs measured in July 2011. A new dataframe named data is created using the cbind function. It contains information on the borough name, unique identificationcode, unemployment rate, income and political control of each borough. We get rid of rows which do not relate to London boroughs using data<-data[-x,], where x is the row to be deleted. Next, new and more intuitive column names are created for the data dataframe.

Step 3: The treemap function takes several important arguments. The argument index indicates the hierarchical structure to be used in the treemap; the first name mentioned is the highest aggregation level, the second name the second-highest aggregation level, and so on. We are interested in political control by borough, and so specify "control" first followed by "borough" .The size of the rectangles is determined by income level using vSize="income" . Color gradation is set by vColor="unemp" using the palette "RdBu". Note the argument type takes a wide variety of character based values. We set type="value" which allows the statement vColor="unemp" to be mapped directly to the "RdBu" color palette.

Summary

The ability to represent a hierarchy of data makes the treemap visualization both efficient and effective. Originally designed to visualize file space on very large computer-storage disk devices, treemaps are now used in a diverse range of fields including human-gene research, social network analysis and finance.The map.market function in the package portfolio offers a simple and often the most convenient approach to rapid treemap construction. The treemap function in the package treemap provides a useful alternative for custom treemaps.

Quick reference R code

Figure 4 2 Treemap of Dow-Jones Index

```
install.packages("portfolio")
require(portfolio)
data(dow.jan.2005)
map.market(id    = dow.jan.2005$symbol,
           area  = dow.jan.2005$price,
           group = dow.jan.2005$sector,
           color = 100 * dow.jan.2005$month.ret)
```

Figure 4 4 Treemap of economic wealth colored by relative population

```
install.packages("portfolio")
install.packages("treemap")
require("portfolio")
require("treemap")
data(GNI2010)
z=rank(GNI2010$population)
z=z-104
map.market(id    = GNI2010$iso3,
           area  = GNI2010$GNI+1,
           group = GNI2010$continent,
           color = z,
           main="Gross National Income - Population")
```

Figure 4 5 Treemap of Dow-Jones Index with stock

```
data(dow.jan.2005)
map.market(id    = dow.jan.2005$symbol,
           area  = dow.jan.2005$price,
           group = dow.jan.2005$sector,
           color = 100 * dow.jan.2005$month.ret,
           lab=TRUE)
```

Figure 4 6 Customized map.market treemap-

```
data(dow.jan.2005)
map.market.color (id    = dow.jan.2005$symbol,
           area  = dow.jan.2005$price,
           group = dow.jan.2005$sector,
           color = 100 * dow.jan.2005$month.ret,
           lab=TRUE)
```

Figure 4 7 Treemap using the treemap function

```
install.packages("treemap")
require("treemap")
raw.data<-read.csv("http://data.london.gov.uk/datafiles/demographics/london-borough-profiles.csv")
data<-cbind(raw.data[,1:2],as.numeric(as.character(raw.data[,25])),as.numeric(as.character(raw.data[,31]))
data<-data[-1,]
data<-data[-1,]
data<-data[-37,]
```

```
data<-data[-36,]
data<-data[-35,]
data<-data[-34,]
data<-data[-33,]
column.names<-c("id","borough","unemp","income","control")
colnames(data)<-column.names
treemap(data,
        index=c("control", "borough"),
        vSize="income",
        vColor="unemp",
        type="value",
        palette="RdBu",
        title = "London Boroughs: Political Control,Unemployment and Income",
     title.legend = "Unemployment Rate (%)"
        )
```

Further resources

Further reading on applications of treemaps can be found in:

Brunetti, Josep Maria, Roberto García, and Sören Auer. "From Overview to Facets and Pivoting for Interactive Exploration of Semantic Web Data." International Journal on Semantic Web and Information Systems (IJSWIS) 9.1 (2013): 1-20.

Oliva, Gustavo, et al. "A Change Impact Analysis Approach for Workflow Repository Management." ICWS-IEEE 20th International Conference on Web Services-2013. 2013.

Tian, Sheng, et al. "Drug-likeness analysis of traditional Chinese medicines: 2. Characterization of scaffold architectures for drug-like compounds, non-drug-like compounds, and natural compounds from traditional Chinese medicines." Journal of cheminformatics 5.1 (2013): 1-14.

Wan, Gang. "Visualizations for digital libraries." Information technology and libraries 25.2 (2013): 88-94.

For further details on the map.market function see:

Jeff Enos, David Kane, with contributions from Daniel Gerlanc and Kyle Campbell (2012). portfolio: Analyzing equity portfolios. R package version 0.4-5. http://CRAN.R-project.org/package=portfolio

For further details on the treemap function see:

Martijn Tennekes (2013). treemap: Treemap visualization. R package version 2.0.1. http://CRAN.R-project.org/package=treemap

Technical details on treemap construction is contained in:

Johnson, B., & Shneiderman, B. (1991) Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In Proceedings of the 2nd International IEEE Visualization Conference, pp. 284-291, San Diego, CA.

Wtite you notes below…

5. A powerful way to simplify bivariate complexity

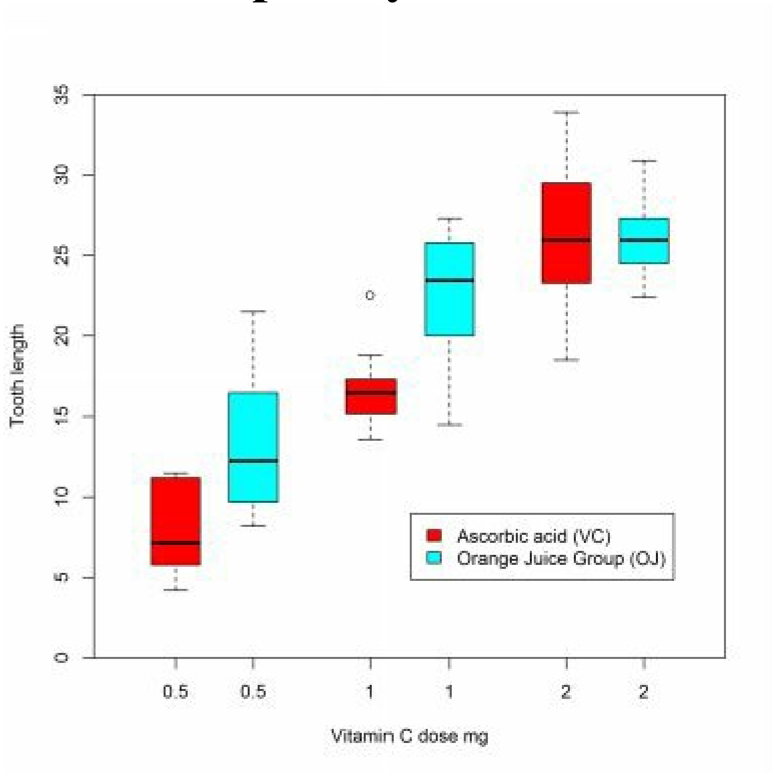


Figure 5 1 Box plot of the effect of Vitamin C on tooth growth

Box plots provide a convenient way of graphically depicting groups of numerical data through their quartiles. Figure 5 1 depicts a typical approach to displaying the relationship between two groups using boxplots. It shows the relationships between the length of odontoblasts (teeth) in each of 10 guinea pigs at each of three dose levels of Vitamin C (0.5, 1, and 2 mg) using two delivery methods (orange juice or ascorbic acid). The relationship between the two delivery methods, dose and tooth growth is evident. Boxplots provide a wonderful tool for displaying this type of data and should form a part of every visual analysts toolkit.

A supplemental technique which can enhance the visual display of box plot type data and assist in exploratory analysis is the two dimensional box plot contours. Two dimensional box plot contours (2d-bc plot) provide a powerful way to visualize bivariate relationships using traditional boxplot metrics. Here is how they work. First, the sample data is centered and standardized with projections onto direction vectors ranging from 0 to 360 degree. Second, the traditional boxplot characteristics (median, box, whisker) are computed for each projection and retransformed into contours using the covariance matrix.

Figure 5 2 presents a typical 2d-bc plot using the same data of the box plots of Figure 5 1. The dose level is captured on the x-axis, with tooth length measured by the y-axis. There appears to be a generally positive relationship across both treatment types (VC and OJ).

One of the major uses of a 2d-bc plot is visualize intuitively relationships between two variables which have a number of categories. Figure 5 3 shows a 2d-bc plot for sepal length (x –axis) and sepal width (y-axis) collected by Anderson (1935) for flowers from each of 3 species of iris (setosa, versicolor, virginica). The relationship between sepal length and width within and between individual flower species is clearly observable.

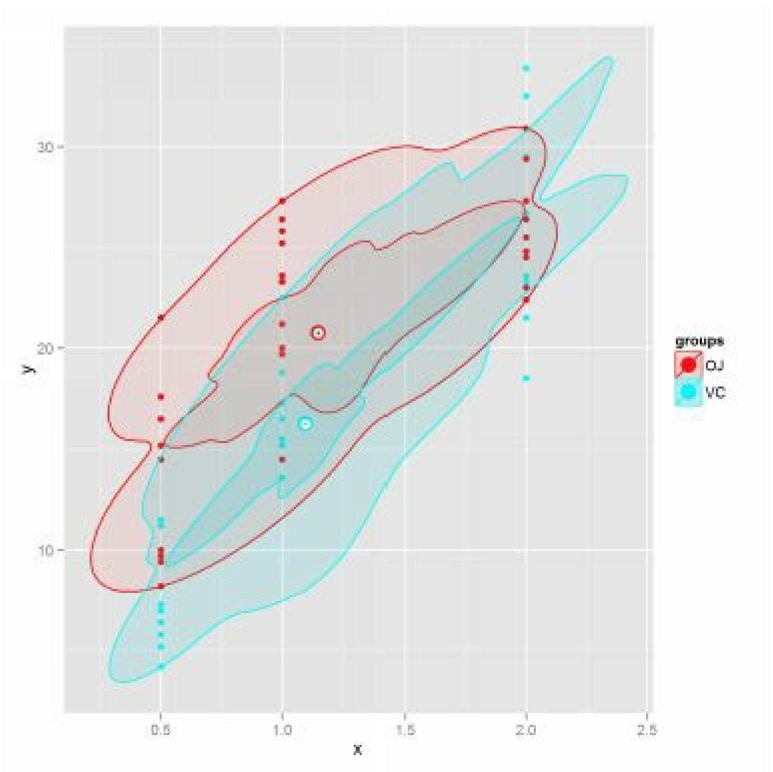


Figure 5 2 2d contour plot of the effect of Vitamin C on tooth growth

The 2d-bc plot offers an extremely useful exploratory data analysis tool as the following example illustrates.

Tipping in Restaurants: Wait table staff make a considerable proportion of their income from tips. The better the service, friendlier the waiter and tastier the food, the higher the expected tip. Bryant & Smith (1995) report on a waiter who recorded information about each tip he received over a period of several months. For each of the 244 tips the waiter recorded information on:

- tip in dollars,
- bill in dollars,
- sex of the bill payer,
- whether there were smokers in the party,
- day of the week,

- time of day, *
- size of the party.

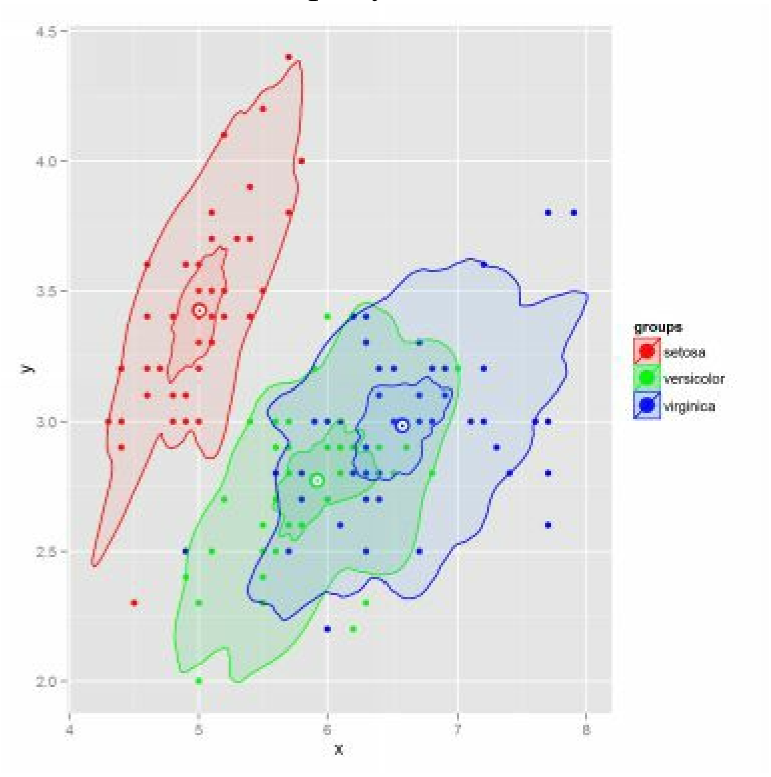


Figure 5 3 2d contour plot of Iris data

Of great interest to managers of restaurants and wait staff is the relationship between the size of the bill and tip in dollars given factors such as sex, day of the week and so on. These relationships can be explored and visualized quickly using a 2d-bc plot, for example in Figure 5 3 . We see clearly the positive relationship between tip (x-axis) and bill size (y-axis). The relationship between these two variables and group size is also evident, as is the relationship between group size and bill size/tips. Thus, the 2d-bc plot provides a powerful way to simplify bivariate group complexity.

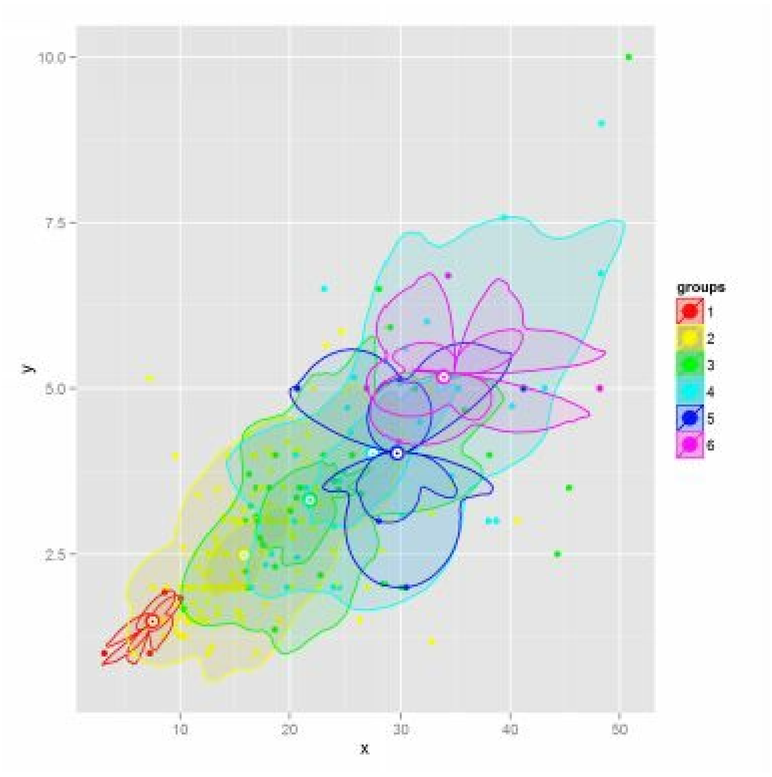


Figure 5 4 relationship between tips and total bill given group size

As a second illustration of this powerful visualizationtool consider the relationship between tips and total bill given the day of the week (the waiter only worked Thursday through Sunday). This relationship is illustrated in*Figure 5 5* . There is a clear positive relationship. It is interesting to note the contours are widest on Sunday, what do you suspect is the cause of this?

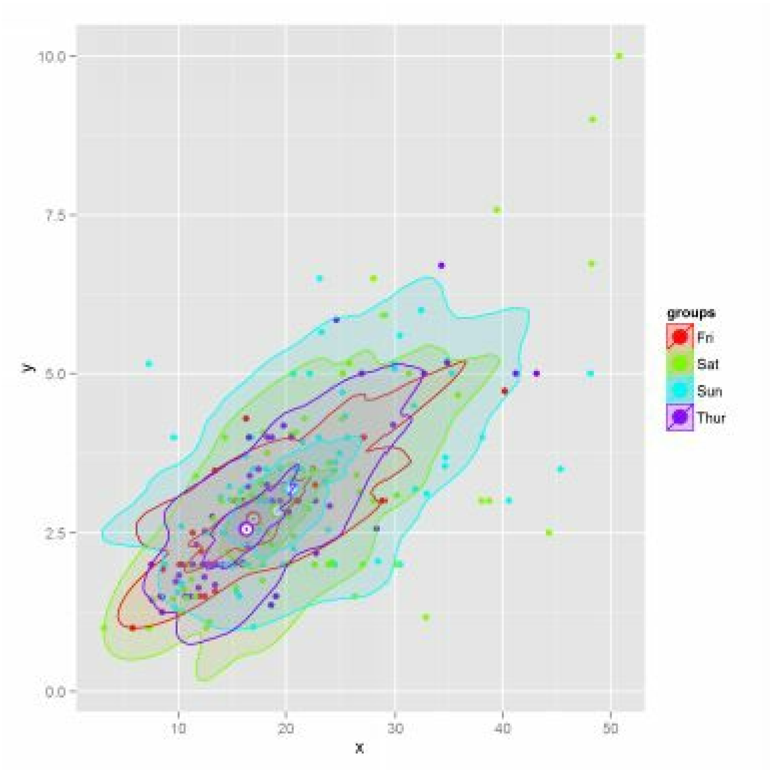


Figure 5 5 relationship between tips and total bill given day of week

Creating 2d-bc plots in R

Figure 5 3 was created using the extracat package. It was built using the following three steps:

Step 1: Download and install the required packages.

```
install.packages("extracat ")-
require(extracat)
```

Step 2: Clean and prepare the data

```
attach(iris)
```

Step 3: Plot the chart

```
boxplot2g(iris$Sepal.Length,iris$Sepal.Width,iris$Species)
```

Step 1: Installs the required packaged and makes it available to R. If the package is already installed on your machine you do not need to use install.packages . However, the require command is needed at the start of a new R session.

Step 2: Loads the required data into the R environment. The iris data is contained in the datasets package with comes pre-installed in R.

Step 3: The boxplot2g function draws the image. The sepal length is plotted on the x-axis, sepal width on the y axis. The third argument passed to the function relates to the groups or categories contained in the data, in this case species (setosa, versicolor, virginica).

Tip : When you have a large number of categories try adding the argument colv = rainbow(n) to boxplot2g. This will use the rainbow pallet to draw the group contours. For example, if you had eight categories you could add colv = rainbow(8) .

Summary

The 2d-bc plot offers a powerful way to visualize the bivariate relationships using traditional boxplot metrics. It provides an intuitive way to visualize relationships between two variables which have a number of categories. The chart can be used to simplify bivariate complexity for presentation purposes or as an data exploration tool, either way it is a- technique you need in your visualization toolbox.

Quick reference R code

Figure 5 1 Box plot of the effect of Vitamin C on tooth growth

```
data(ToothGrowth)
boxplot(len ~ dose, data = ToothGrowth,
        boxwex = 0.25, at = 1:3 - 0.2,
        subset = supp == "VC", col = "red",
        xlab = "Vitamin C dose mg",
        ylab = "Tooth length",
        xlim = c(0.5, 3.5), ylim = c(0, 35), yaxs = "i")
boxplot(len ~ dose, data = ToothGrowth, add = TRUE,
        boxwex = 0.25, at = 1:3 + 0.2,
        subset = supp == "OJ", col = "cyan")
legend(2, 9, c("Ascorbic acid (VC)", "Orange Juice Group (OJ)"),
       fill = c("red", "cyan"))
```

Figure 5 2 2d contour plot of the effect of Vitamin C on tooth growth

```
require(reshape)
require(extracat)
data(ToothGrowth)
boxplot2g(ToothGrowth$dose, ToothGrowth$len, ToothGrowth$sup)
```

Figure 5 3 2d contour plot of Iris data

```
install.packages("extracat ")
require(extracat)
attach(iris)
boxplot2g(iris$Sepal.Length,iris$Sepal.Width,iris$Species)
```

Figure 5 4 relationship between tips and total bill given group size

```
install.packages("extracat ")
install.packages("reshape")
require(extracat)
require(reshape)
data(tips)
boxplot2g(tips$total_bill,tips$tip,tips$size)
```

Figure 5 5 relationship between tips and total bill given day of week

```
install.packages("extracat ")
install.packages("reshape")
require(extracat)
require(reshape)
data(tips)
boxplot2g(tips$total_bill,tips$tip,tips$day)
```

Further resources

Further reading on the extracat package can be found in:

Alexander Pilhoefer, Antony Unwin (2013). New Approaches in Visualization of Categorical Data: R Package extracat. Journal of Statistical Software, 53(7), 1-25. URL http://www.jstatsoft.org/v53/i07/.

For further details on the datasets mentioned in this chapter see:

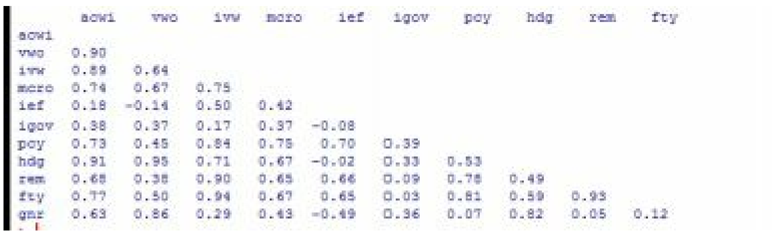
Anderson, Edgar (1935). The irises of the Gaspe Peninsula, Bulletin of the American Iris Society, 59, 2–5.

R Core Team (2013). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Bryant, P. G. and Smith, M (1995) Practical Data Analysis: Case Studies in Business Statistics. Homewood, IL: Richard D. Irwin Publishing.

Wtite you notes below…

6. Network your way to better correlation visualization



```
      acwi   vwo    ivw   mcro    ief   igov    pcy    hdg    rem    fty
acwi
vwo   0.90
ivw   0.89   0.64
mcro  0.74   0.67   0.75
ief   0.18  -0.14   0.50   0.42
igov  0.38   0.37   0.17   0.37  -0.08
pcy   0.73   0.45   0.84   0.75   0.70   0.39
hdg   0.91   0.95   0.71   0.67  -0.02   0.33   0.53
rem   0.68   0.38   0.90   0.65   0.66   0.09   0.78   0.49
fty   0.77   0.50   0.94   0.67   0.65   0.03   0.81   0.59   0.93
gnr   0.63   0.86   0.29   0.43  -0.49   0.36   0.07   0.82   0.05   0.12
```

Figure 6 1 Typical approach used to display correlations

Exploration and display of correlations form an integral part of the day to day duties of the data scientist. Where many variables are concerned, the usual approach is to display a correlation table, typically similar to that shown in Figure 6 1 . The figure shows the correlation of the daily return of eleven exchange traded funds using closing prices over the time period August 23rd 2011 to September 11th 2013. Although the correlations are presented neatly, it is often difficult to discern the true multivariate nature of the relationships. In practice, a researcher might be interested in correlations above a certain absolute value. A popular, although primitive, solution is to highlight such correlations in the above table. A much better idea, is to use a correlation network to display visually the relationship between variables.

Figure 6 2 illustrates a basic correlation network using the exchange traded funds data. It is constructed so that green lines represent a positive correlation between any two variables, and red lines a negative correlation. The thickness of the lines increases with the absolute value of the correlation.

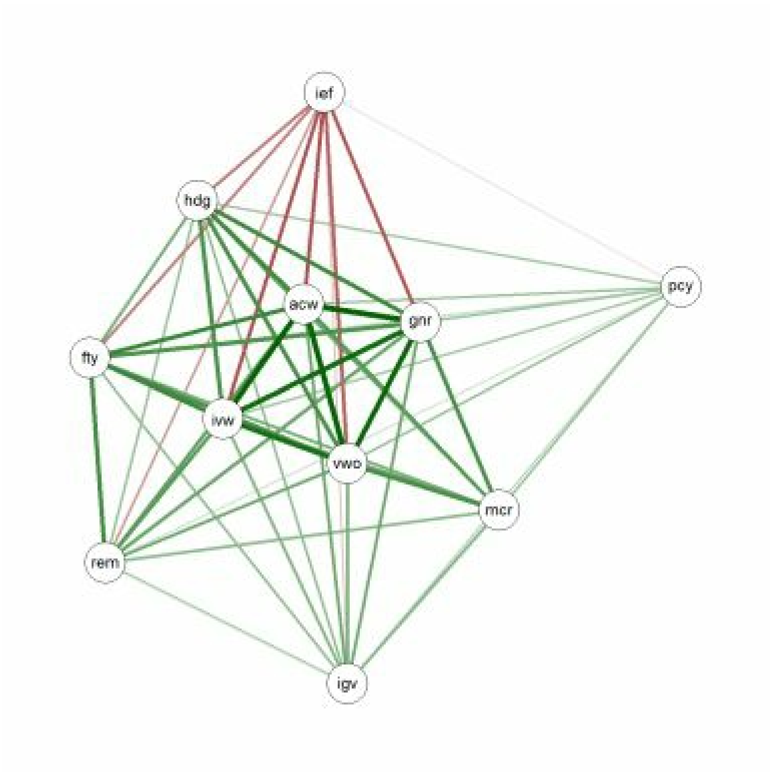


Figure 6 2 Basic correlation networks of popular exchange traded funds

The image captures neatly the multivariate nature of the correlation structure in a more intuitive fashion that a traditional table. Further details on the correlation structure can be revealed by only examining correlations above a specific absolute value. For example, suppose the researcher is interested in absolute correlations greater than 0.5. The resultant correlation network is displayed in Figure 6 3. Notice how the multivariate interrelationships between the variables is immediately evident. A positive core of four exchange traded funds (acw, gnr, vwo and ivw) is apparent, whilst igv and pcy appear uncorrelated with the remaining exchange traded funds (at the cutoff threshold).

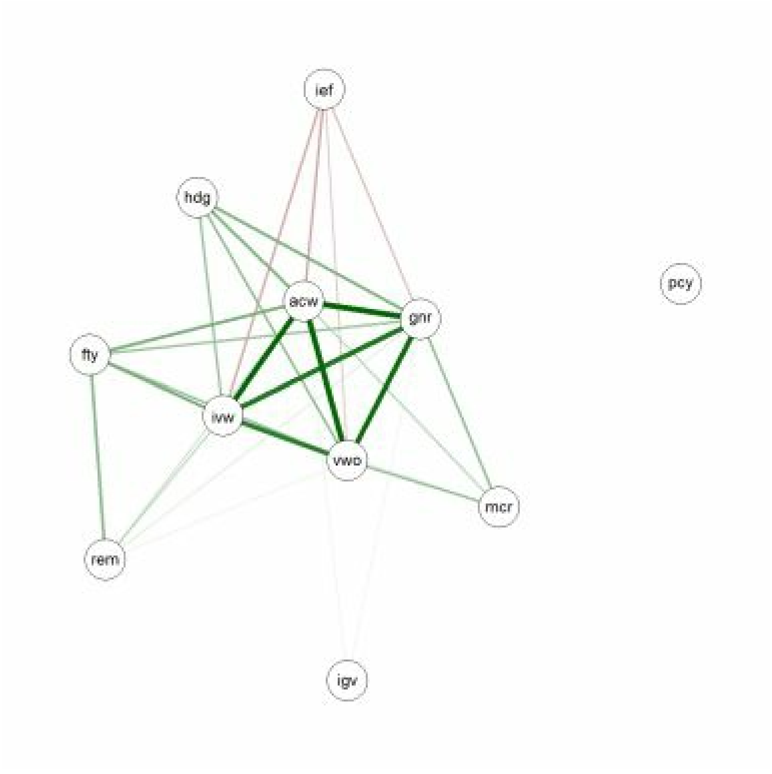


Figure 6 3 Correlations greater than |0.5|

Innovative use of correlation networks

Physiological networks using drugs: Grossman, Adam et al (2013) built correlation networks using physiologic data to investigate changes associated with pressor use in intensive care. The researchers collected 29 physiological variables at one-minute intervals from nineteen trauma patients. They grouped each minute of data as receiving or not receiving pressors. For each group correlation networks of pairs of physiologic variables were built. To visualize drug-associated changes the researchers divided the resultant networks into three components: an unchanging correlation network, a correlation network of connections with changing correlation sign, and a correlation network of connections only present in one group.

Mindfulness training: Taylor, Véronique A., et al (2013) examine the effect of mindfulness training on functional brain connectivity during a restful state. Using data from 13 experienced meditators, 11 beginner meditators and functional magnetic resonance imaging, pairwise correlations were computed between default mode network seed regions' time courses. A correlation network diagram was used to visualize the resultant correlations. The researchers observed relative to beginners, experienced meditators had weaker functional connectivity between default mode network (DMN) brain regions involved in self-referential processing and emotional appraisal. However, they noted experienced meditators had increased connectivity between certain DMN regions, compared to beginner meditators. The researchers conclude meditation training leads to functional connectivity changes between core DMN regions possibly reflecting strengthened present-moment awareness.

Chinese Prices: Gao and Zhong (2013) collect data on 195 Chinese price indices from 2003 to 2011. The relationships between the observations are investigated using a correlation network, see *Figure 6 4* . The researchers choose a correlation of 0.82 as their threshold. Therefore If the correlation coefficient between two price indices is greater than 0.82 there is an edge between them.

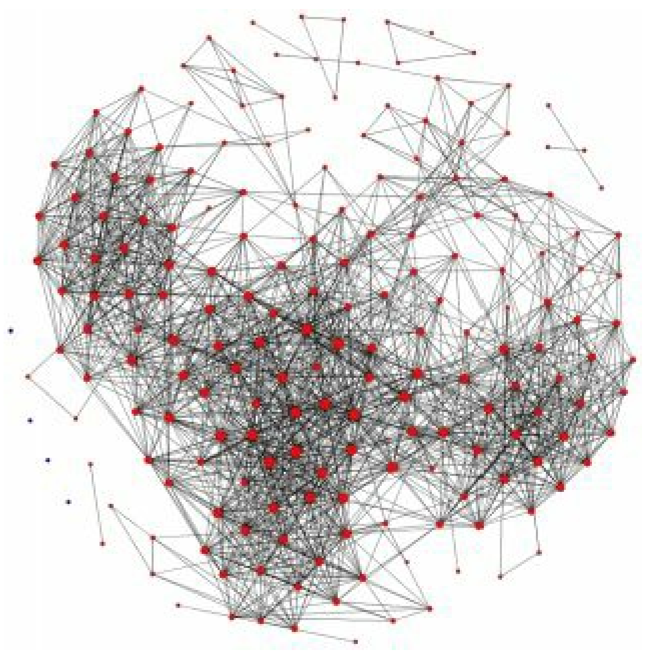


Figure 6 4 Correlation network of Gao and Zhong[vii]

Creating correlation networks in R

Figure 6 1 was created using the qgraph package. It was built using the following three steps:

Step 1: Download and install the required packages.

```
install.packages("PerformanceAnalytics")
install.packages("tseries")
install.packages ("fImport")
install.packages ("qgraph")
library(tseries)
library(PerformanceAnalytics)
library(qgraph)
```

Step 2: Clean and prepare the data

```
days = 750
date= as.Date(c("2013-09-11"))-days

con <- url("http://quote.yahoo.com")
if(!inherits(try(open(con), silent = TRUE), "try-error"))
{
  close(con)
#Note if you change any of the asset classes use search and replace
#don’t do manually as you may miss something

acwi <- get.hist.quote(instrument = "acwi",  start = date ,
compression = "d", quote = "Close")

vwo<- get.hist.quote(instrument = "vwo",  start = date ,
compression = "d", quote = "Close")

iwv<- get.hist.quote(instrument = "ivw",  start = date ,
compression = "d", quote = "Close")


mcro<- get.hist.quote(instrument = "mcro",  start = date ,
compression = "d", quote = "Close")

ief<- get.hist.quote(instrument = "ief",  start = date ,
compression = "d", quote = "Close")

igov<- get.hist.quote(instrument = "igov",  start = date ,
compression = "d", quote = "Close")
-
pcy<- get.hist.quote(instrument = "pcy",  start = date ,
compression = "d", quote = "Close")

hdg<- get.hist.quote(instrument = "hdg",  start = date ,
compression = "d", quote = "Close")

rem<- get.hist.quote(instrument = "rem",  start = date ,
compression = "d", quote = "Close")
fty<- get.hist.quote(instrument = "fty",  start = date ,
compression = "d", quote = "Close")

gnr<- get.hist.quote(instrument = "gnr",  start = date ,
compression = "d", quote = "Close")

}


x <- merge(acwi,vwo,iwv,mcro,ief,igov,pcy,hdg,rem,fty,gnr)

x<- na.omit(x)
colnames(x)<- c("acwi","vwo",  "ivw",  "mcro",  "ief",  "igov",  "pcy", "hdg",  "rem",  "fty",  "gnr"  )


returns =diff(x)/x[-length(x)] # Calculating rates of return from price levels
returns = na.omit(returns)

# calculate correlation matrix
cor=cor(returns)
```

Step 3: Plot the chart

```
qgraph(cor,shape="circle",posCol="darkgreen",negCol="darkred",layout= "spring",vsize=10)
```

Step 1: Installs the required packaged and makes it available to R. If the package is already installed on your machine you do not need to use install.packages . However, the require command is needed at the start of a new R session.

Step 2: Capture the closing price of each exchange traded fund from Yahoo. The data are converted into daily changes and cleaned, followed by the calculation of the correlation matrix.

Step 3: The qgraph function draws the correlation network. It is constructed so that positive correlations are displayed in dark green and negative correlations in dark red.

Tip 1: To add a correlation cutoff value include the comment minimum=cutoff to the qgraph function. For example, Figure 6 3 was built using: qgraph(cor,shape="circle",posCol="darkgreen",negCol="darkred",layout= "spring",vsize=10,minimum=0.5)

Tip 2: The argument vsize changes node size. Try experimenting with various values, for example set it to 20 and then 5. What do you observe?

Tip 3: A circular correlation plot can be obtained by settingthe argument layout= "circular". For example, Figure 6 5 , was constructed using:

```
qgraph(cor,shape="circle",posCol="darkgreen",negCol="darkred",layout= "circular",vsize=10)
```

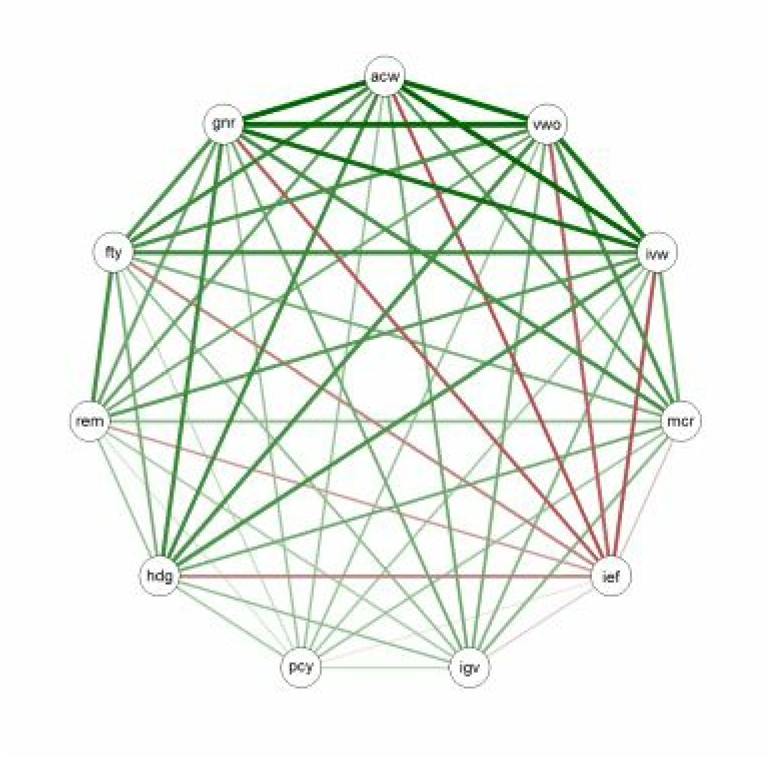


Figure 6 5 Circular correlation network

Quick reference R code

Figure 6 2

```
install.packages("PerformanceAnalytics")
install.packages("tseries")
install.packages ("fImport")
install.packages -("qgraph")
library(tseries)
library(PerformanceAnalytics)
library(qgraph)



days = 750 # number of days to use in analysis approx. 3 years
date= as.Date(c("2013-09-11"))-days

library(tseries)
library(PerformanceAnalytics)

library(qgraph)


####################GET DATA FROM
YAHOO######################################

con <- url("http://quote.yahoo.com")
if(!inherits(try(open(con), silent = TRUE), "try-error"))
{
  close(con)
#Note if you change any of the asset classes use search and replace
#don't do manually as you may miss something

acwi <- get.hist.quote(instrument = "acwi",  start = date ,
compression = "d", quote = "Close")

vwo<- get.hist.quote(instrument = "vwo",  start = date ,
compression = "d", quote = "Close")

iwv<- get.hist.quote(instrument = "ivw",  start = date ,
compression = "d", quote = "Close")


mcro<- get.hist.quote(instrument = "mcro",  start = date ,
compression = "d", quote = "Close")

ief<- get.hist.quote(instrument = "ief",  start = date ,
compression = "d", quote = "Close")

igov<- get.hist.quote(instrument = "igov",  start = date ,
compression = "d", quote = "Close")
-
pcy<- get.hist.quote(instrument = "pcy",  start = date ,
compression = "d", quote = "Close")

hdg<- get.hist.quote(instrument = "hdg",  start = date ,
compression = "d", quote = "Close")

rem<- get.hist.quote(instrument = "rem",  start = date ,
compression = "d", quote = "Close")

fty<- get.hist.quote(instrument = "fty",  start = date ,
compression = "d", quote = "Close")

gnr<- get.hist.quote(instrument = "gnr",  start = date ,
compression = "d", quote = "Close")

}

#### calculate correlations and plot network##################

x <- merge(acwi,vwo,iwv,mcro,ief,igov,pcy,hdg,rem,fty,gnr)

x<- na.omit(x)
colnames(x)<- c("acwi","vwo",  "ivw",  "mcro",  "ief",  "igov",  "pcy", "hdg",  "rem",  "fty",  "gnr"  )


returns =diff(x)/x[-length(x)] # Calculating rates of return from price levels
returns = na.omit(returns)

cor=cor(returns)

qgraph(cor,shape="circle",posCol="darkgreen",negCol="darkred",layout= "spring",vsize=10)
```

Figure 6 3

As above but replace last line with:

```
qgraph(cor,shape="circle",posCol="darkgreen",negCol="darkred",layout= "spring",vsize=10,minimum=0.5)
```

Figure 6 5

As above but replace last line with:

```
qgraph(cor,shape="circle",posCol="darkgreen",negCol="darkred",layout= "circular",vsize=10)
```

Further resources

Further reading on the packages mentioned in this chapter see:

Adrian Trapletti and Kurt Hornik (2013). tseries: Time Series Analysis and Computational Finance. R package version 0.10-32.

Diethelm Wuertz and many others (2013). fImport: Rmetrics - Economic and Financial Data Import. R package version 3000.82. http://CRAN.R-project.org/package=fImport

Peter Carl and Brian G. Peterson (2013). PerformanceAnalytics: Econometric tools for performance and risk analysis.. R package version 1.1.0. http://CRAN.R-project.org/package=PerformanceAnalytics

Sacha Epskamp, Angelique O. J. Cramer, Lourens J. Waldorp, Verena D. Schmittmann, Denny Borsboom (2012). qgraph: Network Visualizations of Relationships in Psychometric Data. Journal of
Statistical Software, 48(4), 1-18. URL http://www.jstatsoft.org/v48/i04/.

For further information on the datasets mentioned in this chapter see:

Gao X, An H, Zhong W (2013) Features of the Correlation Structure of Price Indices. PLoS ONE 8(4): e61091. doi:10.1371/journal.pone.0061091

Grossman, Adam D., et al. "Altering physiological networks using drugs: steps towards personalized physiology." BMC Medical Genomics 6.2 (2013): 1-11.

Taylor, Véronique A., et al. "Impact of meditation training on the default mode network during a restful state." Social cognitive and affective neuroscience 8.1 (2013): 4-14.

Wtite you notes below…

7. How to use a Rose for directional impact

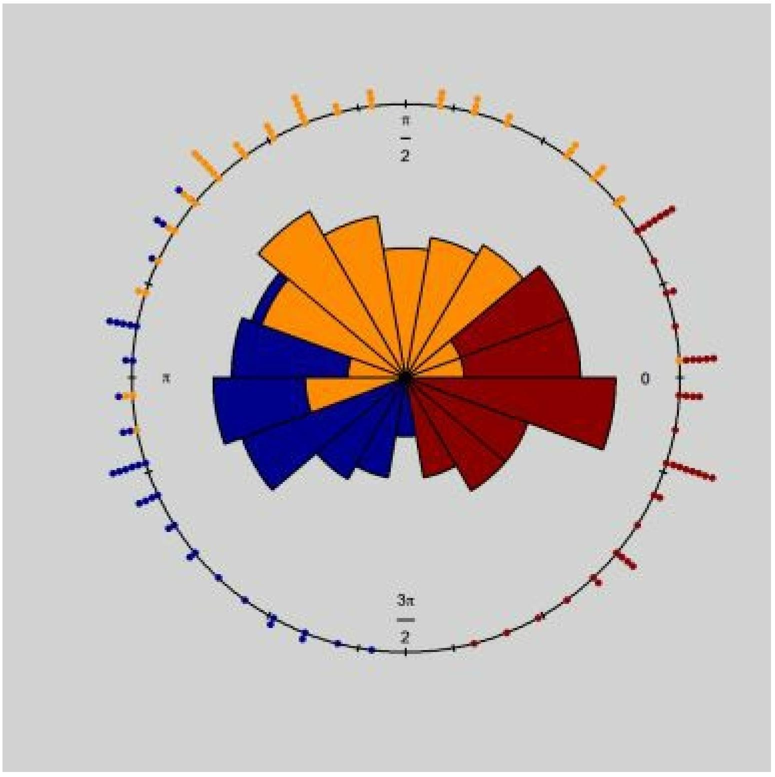


Figure 7 1 Rose diagram of beaver foraging

Directional datasets were first analyzed in the Earth Sciences, Meteorology and Medicine. With the growth in social computing and growing volumes of data from online sources, it will become more frequently observed in nontraditional areas also. The analysis of this type of data poses a number of challenges for the data analyst. For example, the usual statistics such as the arithmetic mean or standard deviation are not rotational invariant. For example, the average direction between 10 degrees and 350 degrees is not given by the arithmetic mean of 180 degrees. However, Rose diagrams offer a powerful solution to the issue of presenting circular or directional data. Figure 7 1 shows the foraging direction of three beavers given in Lewis (2013). Each beaver is represented by a different color (orange, red, blue). The foraging directional preferences of each beaver are clearly discernible from the diagram.

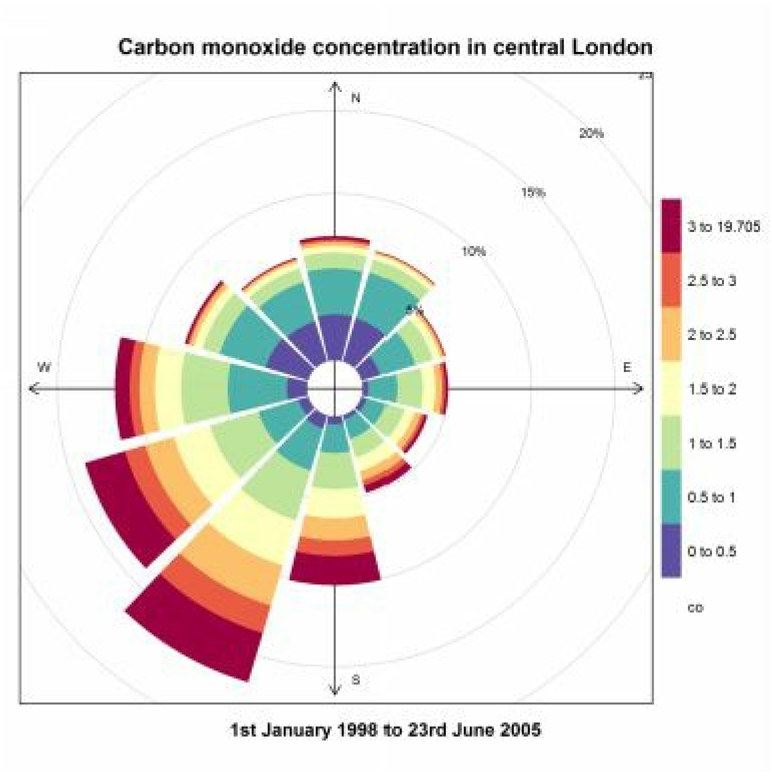


Figure 7 2 Pollution rose of Carbon monoxide concentration at Marylebone

Figure 7 2 illustrates a slight variation of the basic rose diagram know as a pollution rose. It plots the frequency of counts by wind direction for carbon monoxide concentration (parts per million) measured at Marylebone in central London. The petals are similar to the bars in a histogram and represent the frequency of observations. In this image the petals are color coded to capture the extent of concentration. Circular bands are drawn at 5%,10%,15% and 20%. A petal that touches the 5% band represents 5% of the overall number of observations.

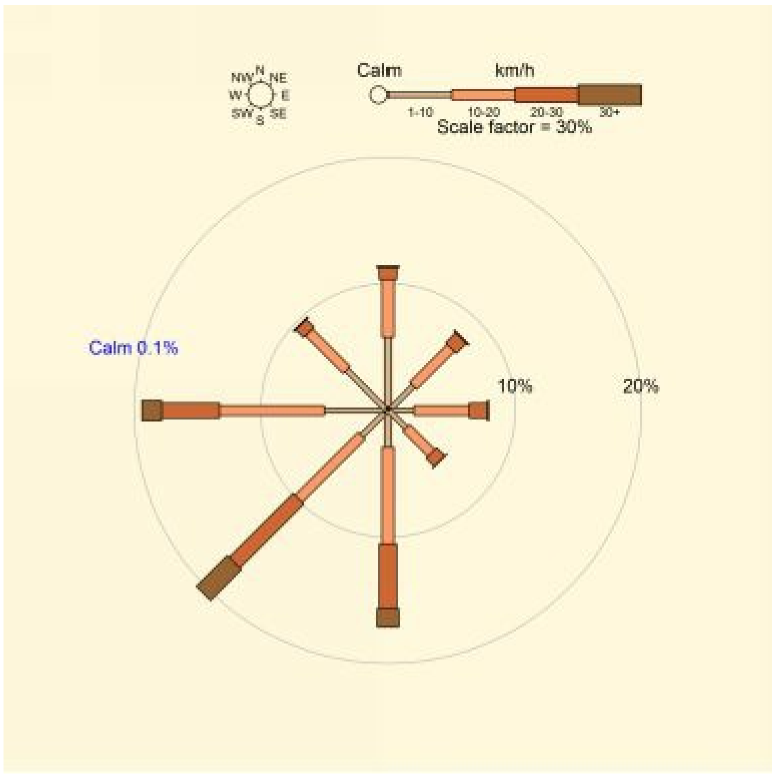
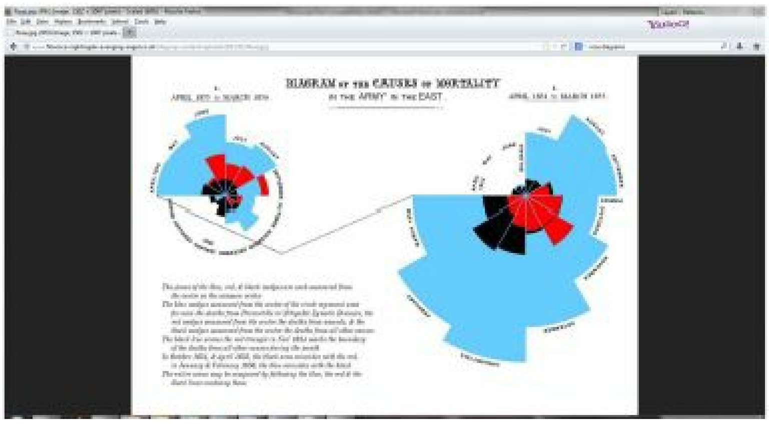


Figure 7 3 Australian wind rose

Figure 7 3 illustrates another variation of the rose diagram - the Australian wind rose. The chart shows a traditional Australian wind rose using data measured at Marylebone, London between 1st January 1998 to 23rd June 2005. Wind directions are divided into eight compass directions. The circles around the image represent the various percentages of occurrence of the winds. For example, the branch to the north west just reaches the 10% ring implying a frequency of 10% blowing from that direction. Calm has no direction.

Innovative use of rose diagrams

Florence Nightingale: A British Social Reformer, Statistician, and founder of modern nursing. She made use of the rose diagram during the late 1850's and early 1860's to convince politicians that deaths from epidemic disease were avoidable. Her evidence presented in the g-graphic below, was compelling[viii].



Fault patterns: Leonard, and Szynkaruk (2013) investigate the fault pattern in areas of the Polish Outer Carpathians. Fault positions were established by photointerpretation where zones of breccia and cataclasites extended on valley slopes as narrow and often dry gullies. The researchers used rose diagrams to visualize the distribution of fault pattern.

Storm surges: The South China Sea (SCS) lies under the southwest-northeast pathway of the seasonal monsoons which dominate the large scale sea level dynamics of region. Tkalich, Pavel, et al (2013) investigate wind-driven sea level anomalies using tide gauge data and satellite altimetry. Climatological wind rose diagrams using data over the period 1984 – 2007 were used as a powerful visualization technique.

Creating rose diagrams in R

Figure 7 1 was created using the circular package. It was built using the following three steps:

Step 1: Download and install the required packages.

```
install.packages("circular")
require(circular)
```

Step 2: Clean and prepare the data

```
set.seed(1234)
beaver.1 <- rvonmises(n=50, mu=circular(0), kappa=3)
beaver.2 <- rvonmises(n=50, mu=circular(pi), kappa=3)
beaver.3 <- rvonmises(n=50, mu=circular(pi/2), kappa=3)
```

Step 3: Plot the chart

```
par(bg="lightgray")
rose.diag(beaver.1, bins=18,axes=TRUE,ticks=TRUE,border="black",col="darkred",tol=0,prop=1.5)
par(new=T)
rose.diag(beaver.2, bins=18,axes=TRUE,ticks=TRUE,border="black",col="darkblue",tol=0,prop=1.5)
par(new=T)
rose.diag(beaver.3, bins=18,axes=TRUE,ticks=TRUE,border="black",col="darkorange",tol=0,prop=1.5)
points(beaver.1, col="darkred", stack=TRUE)
points(beaver.2, col="darkblue", stack=TRUE)
points(beaver.3, col="darkorange", stack=TRUE)
```

Step 1: Installs the required packaged and makes it available to R. If the package is already installed on your machine you do not need to use install.packages . However, the require command is needed at the start of a new R session.

Step 2: Creates the required data. In this case the beaver foraging directions are actually simulated from a Von mises distribution. The argument set.seed(1234) ensures you can replicate exactly the chart as it is presented here. Note the foraging directions are stored in beaver.1 , beaver.2 and beaver.3 respectively

Step 3: The rose.diag function draws the image. The trick is to overlay three rose diagrams. This is achieved using the function par(new=T) before the second and third image are drawn. The actual observations for each beaver are plotted in their respective colors on the diagram using the function points .

Tip: The background color of the image is controlled by par(bg="lightgray"). You can set to an alternative color by specifying your choice. A popular option is par(bg="cornsilk") .

The pollution rose of Figure 7 2 was created using the openair package. It was constructed using the following three steps:

Step 1: Download and install the required packages.

```
install.packages("openair")
require(openair)
```

Step 2: Clean and prepare the data

```
data(mydata)
pollutant <- c("co")# Carbon monoxide data
```

Step 3: Plot the chart

```
title=c("Carbon monoxide concentration in central London")
sub=c("1st January 1998 to 23rd June 2005")
pollutionRose(mydata,paddle=FALSE,pollutant= pollutant,annotate=F, key.footer =   pollutant, key.position
= "right", key = TRUE, breaks = 8,main=title, sub=sub,auto.text=FALSE)
```

Step 1: Installs the required packaged and makes it available to R.

Step 2: Loads the dataframe which amongst other items, contains data on carbon monoxide concentration, in ppm, as a numeric vector. The carbon monoxide data is assigned to the variable `pollutant`.

Step 3: The `pollutionRose` function draws the image.

Tip 1: The location of the key is determined by the argument key.posiition; It can take values "left","right","top" and "bottom". The number of colors (or breaks) represented in the key is determined by the argument breaks; for many datasets values between 6-8 work well.

Tip 2: You can change the plot to a modern wind rose by setting the argument paddle=TRUE.

Tip 3: When substituting your own data note the pollutionRose function requires data frames with a field "date" that can be in either POSIXct or Date format but should be set to the Greenwich Mean Time time-zone. The data should be at least hourly or higher resolution.

Tip 4: The chart used the argument pollutant <- c("co") to access the carbon monoxide observations in the mydata data frame. This data frame contains a number of other air quality observations taken in central London:

1. *"ws" - Wind speed.*
2. *"wd" - Wind direction, in degrees from North.*
3. *"nox"- Oxides of nitrogen concentration.*
4. *"no2" - Nitrogen dioxide concentration.*
5. *"o3"- Ozone concentration.*
6. *"pm10" - Particulate PM10 fraction measurement.*
7. *"so2" - Sulfur dioxide concentration.*
8. *"pm25" - Particulate PM2.5 fraction measurement.*

The Australian wind rose, Figure 7 3 **,** was created using the plotrix package. It was constructed using the following three steps:

Step 1: Download and install the required packages.

```
install.packages("openair")# contains the data
install.packages("plotrix")# used to draw the chart
require(openair)
require(plotrix)
```

Step 2: Clean and prepare the data

```
raw.data<-cbind(mydata$wd,mydata$ws)
```

```
data<-na.omit(raw.data)
table<-bin.wind.records(data[,1],data[,2]*3.6,ndir=8)#
```

Step 3: Plot the chart

```
par(bg="cornsilk")
oz.windrose(table)
```

Step 1: Installs the required packages and makes them available to R.

Step 2: Creates a variable called raw.data which contains data on wind speed and wind direction. Missing observations are removed using na.omit(raw.data). The function bin.wind.records creates a table and covert wind speed to km per hour.

Step 3: The oz.windrose function draws the image.

Tip 1: Data for the oz.windrose function needs to be in a format where the rows represent speed ranges and the columns indicating wind directions. Furthermore, the data should be in percentages such that the sum of all data in your table is equal to 100. This is best achieved using the bin.wind.records function. This function classifies wind direction and speed records into a matrix of percentages of observations in speed and direction bins.Use the argument ndir to change the number of direction bins to be used in the wind rose. The traditional number of bins is ndir=8 .

Tip 2: Use the argument speed.col in the oz.windrose function to change the color of the branches of the wind rose. The default colors, shown in the above chart are obtained by speed.col =c("#dab286","#fe9a66","#ce6733","#986434") . You could also set alternative colors such as speed.col =c("red","green","purple","yellow") .

Tip 3: To remove the legend add the argument show.legend=FALSE to the oz.windrose function.

Summary

Directional datasets were first analyzed in the Earth Sciences, Meteorology and Medicine. Rose diagrams proved an excellent tool for visualizing directional data. Similar to a histogram they provide a quick and simply technique for investigating the distribution of circular datasets. With the growth in social computing and growing volumes of data from online sources, Rose diagrams are becoming more frequently used in a wide range of ‘non-traditional’ areas also.

Quick reference R code

Figure 7 1 Rose diagram of beaver foraging

```
install.packages("circular")
require(circular)
set.seed(1234)
beaver.1 <- rvonmises(n=50, mu=circular(0), kappa=3)
beaver.2 <- rvonmises(n=50, mu=circular(pi), kappa=3)
beaver.3 <- rvonmises(n=50, mu=circular(pi/2), kappa=3)
par(bg="lightgray")
rose.diag(beaver.1, bins=18,axes=TRUE,ticks=TRUE,border="black",col="darkred",tol=0,prop=1.5)
par(new=T)
rose.diag(beaver.2, bins=18,axes=TRUE,ticks=TRUE,border="black",col="darkblue",tol=0,prop=1.5)
par(new=T)
rose.diag(beaver.3, bins=18,axes=TRUE,ticks=TRUE,border="black",col="darkorange",tol=0,prop=1.5)
points(beaver.1, col="darkred", stack=TRUE)
points(beaver.2, col="darkblue", stack=TRUE)
points(beaver.3, col="darkorange", stack=TRUE)
```

Figure 7 2 Pollution rose of Carbon monoxide concentration at Marylebone

```
install.packages("openair")
require(openair)
data(mydata) pollutant <- c("co")title=c("Carbon monoxide concentration in central London")
sub=c("1st January 1998 to 23rd June 2005")
pollutionRose(mydata,paddle=FALSE,pollutant= pollutant,annotate=F, key.footer =     pollutant, key.position
= "right", key = TRUE, breaks = 8,main=title, sub=sub,auto.text=FALSE)
```

Figure 7 3 Australian wind rose

```
install.packages("openair")
install.packages("plotrix")
require(openair)
require(plotrix)
raw.data<-cbind(mydata$wd,mydata$ws)
data<-na.omit(raw.data)# remove missing observations
table<-bin.wind.records(data[,1],data[,2]*3.6,ndir=8)# create table and covert wind speed to km per
hour
par(bg="cornsilk")
oz.windrose(table)
```

Further resources

Further reading on the packages mentioned in this chapter see:

C. Agostinelli and U. Lund (2011). R package 'circular': Circular Statistics (version 0.4-3). URL https://r-forge.r-project.org/projects/circular/

Carslaw, D.C. and K. Ropkins, (2012) openair --- an R package for air quality data analysis. Environmental Modelling & Software. Volume 27-28, 52-61.

Lemon, J. (2006) Plotrix: a package in the red light district of R. R-News, 6(4): 8-12.

For further information on the datasets mentioned in this chapter see:

Lewis, N.D. (2013). The Visual Display of Quantitative Data. Heather Hills Press.

Mastella, Leonard, and Ewa Szynkaruk. "Analysis of the fault pattern in selected areas of the Polish Outer Carpathians." (2013): 263-276.

Tkalich, Pavel, et al. "Storm surges in the Singapore Strait due to winds in the South China Sea." Natural Hazards (2013): 1-18.

Wtite you notes below…

8. Maximize trends in multivariate timeseries data

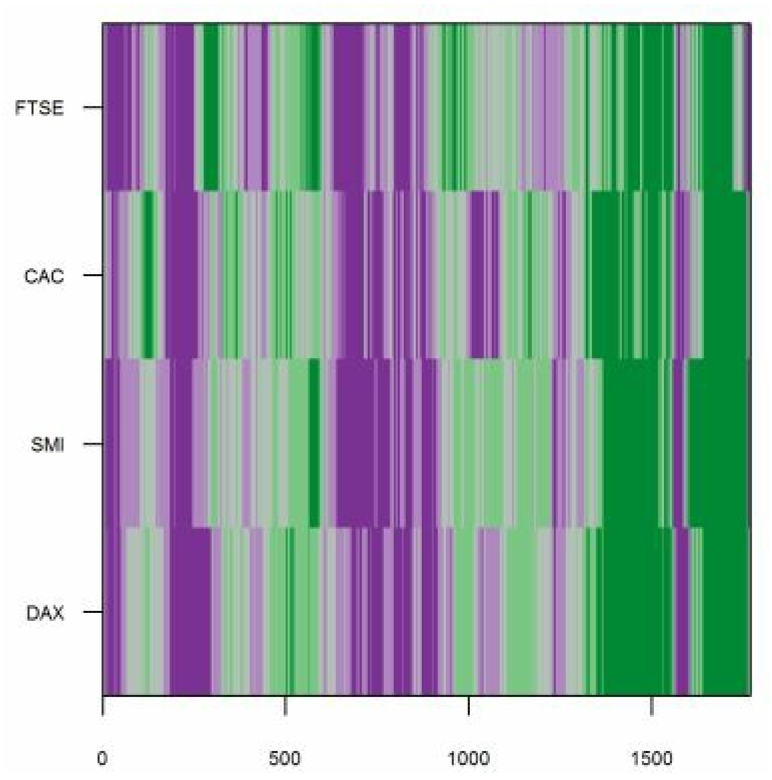


Figure 8 1 Multivariate shaded timeseries plot of European stock markets

Multivariate shaded timeseries plots allow easy identification of relationships between numerous variables and trends in those variables individually and simultaneously across time. Figure 8 1 illustrates such a plot for the 90 day change in stock prices for four European stock market indices (FTSE,CAC, SMI and DAX) using data collected over the period 1991-1998. Notice how the shading of colors ranges from dark purple for negative 90 changes in price, to deep green for very positive changes in price. Observe also how the image allows you to identify the trend in price changes across time and also between stock markets. In this way the multivariate shaded timeseries plot maximizes the visibility of any trends within the sample data.

Innovative use of the multivariate shaded timeseries plot

One of the major uses of a multivariate shaded timeseries plot is to visualize trends between and within variables across time. This is an extremely useful property for exploratory data analysis of datasets which contain large numbers of timeseries data. This chart should be a key technique in your data visualization toolbox.

Sunspots analysis: Sunspots are cool planet-sized areas where intense magnetic loops poke through the Sun's visible surface. They are measured by scientists and used to track solar cycles. Astronomers have been monitoring and collecting data on them since the 1740's. An interesting question is the degree to which sunspot activity changes by month of the year and also across time. This ides is explored in Figure 8 2 , which shows sunspot activity from the mid-18th century to the end of the end of the 20th century. The trend in sunspot activity across time and month is clearly evident, even with this relatively large dataset. The deeper the green the higher is sunspot activity. Low sunspot activity is reflected in deep purple. The relationship between sunspot activity across months and across time is quite striking. The line chart at the bottom of the diagram reports median activity across all months.

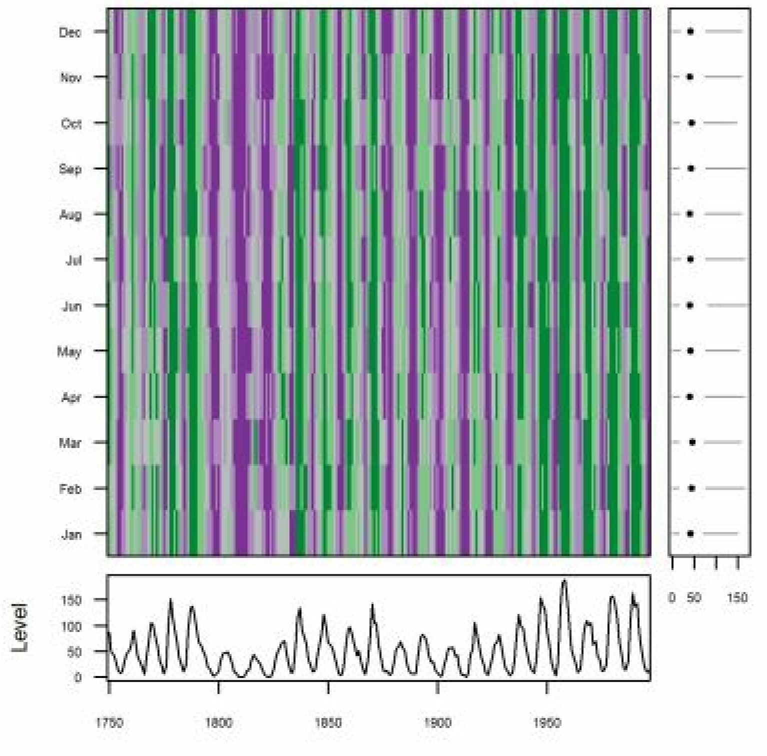


Figure 8 2 Sunspot activity by month of year

Creating multivariate shaded timeseries plots in R

Figure 8 1 was created using the mvtsplot package. It was built using the following three steps:

Step 1: Download and install the required packages.

```
install.packages("mvtsplot")
install.packages("datasets")
require(datasets)
require(mvtsplot)
```

Step 2: Clean and prepare the data

```
data <- diff(EuStockMarkets ,90)
```

Step 3: Plot the chart

```
mvtsplot(data,norm ="internal", levels = 5,margin=FALSE)
```

Step 1: Installs the required packaged and makes it available to R. If the package is already installed on your machine you do not need to use install.packages . However, the require command is needed at the start of a new R session.

Step 2: Loads the required data into the R environment. The EuStockMarkets dataframe is contained in the datasets package/ It is pre-installed in the basic R package. The 90 day change in price level is calculated and stored in the variable called data.

Step 3: The mvtsplot function draws the image.

Tip 1: The parameter levels = 5 patriciansthe data into five levels. Each level is assigned a particular shade of purple or green. Increase the value assigned to levels to increase the range of shading. Figure 8 3 illustrates this with levels = 2 for the Europeanstock market data of Figure 8 1 .

Tip 2: The parameter norm ="internal" ensures each individual time series is categorized into colors based on the its range of values. This implies the same color in two different time series may have a different interpretations. If "global "is specified, then each time series will be categorized based on the range of values for the entire collection of time series.

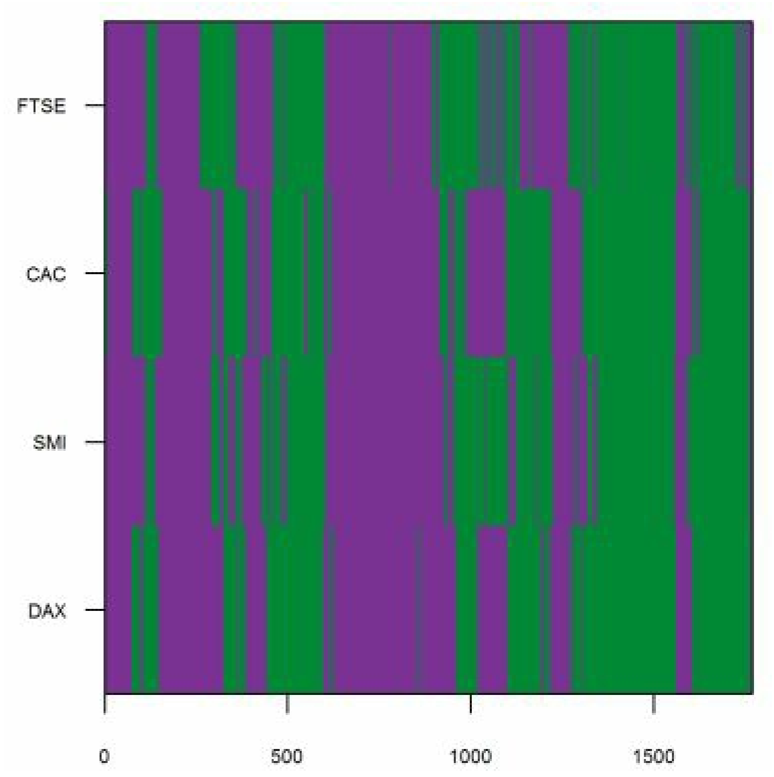


Figure 8 3 Shaded timeseries plot of European stock markets with levels =2

Summary

Multivariate shaded timeseries plots allow easy identification of relationships between numerous variables and trends in those variables individually and simultaneously across time. This is an extremely useful property for exploratory data analysis of datasets which contain large numbers of timeseries data. This chart should be frequently deployed for exploratory analysis and is an important technique in your data visualization and presentation toolkit.

Quick reference R code

Figure 8 1 Multivariate shaded timeseries plot of European stock markets

```
install.packages("mvtsplot")
install.packages("datasets")
require(datasets)
require(mvtsplot)
data <- diff(EuStockMarkets ,90)
mvtsplot(data,norm ="internal", levels = 5,margin=FALSE)
```

Figure 8 2 Sunspot activity by month of year

```
install.packages("mvtsplot")
install.packages("datasets")
require(datasets)
require(mvtsplot)
data<-matrix(data = sunspot.month, ncol = 12, byrow = TRUE)
colnames(data) <- c("Jan","Feb","Mar","Apr","May","Jun","Jul", "Aug","Sep","Oct","Nov","Dec")
rownames(data)<-(1749:1997)
time<-(1749:1997)
mvtsplot(data,xtime = time,norm ="global", levels = 5)
```

Figure 8 3 Shaded timeseries plot of European stock markets with levels =2

```
install.packages("mvtsplot")
install.packages("datasets")
require(datasets)
require(mvtsplot)
data <- diff(EuStockMarkets ,90)
mvtsplot(data,norm ="internal", levels = 2,margin=FALSE)
```

Further resources

Further reading on the mvtsplot package can be found in:

Roger D. Peng (2012). mvtsplot: Multivariate Time Series Plot. R package version 1.0-1. http://CRAN.R-project.org/package=mvtsplot.

For further details on the datasets mentioned in this chapter see:

R Core Team (2013). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Wtite you notes below…

PART 2: FIFTY VISULIZATIONS TO UNLEASH YOUR CREATIVITY

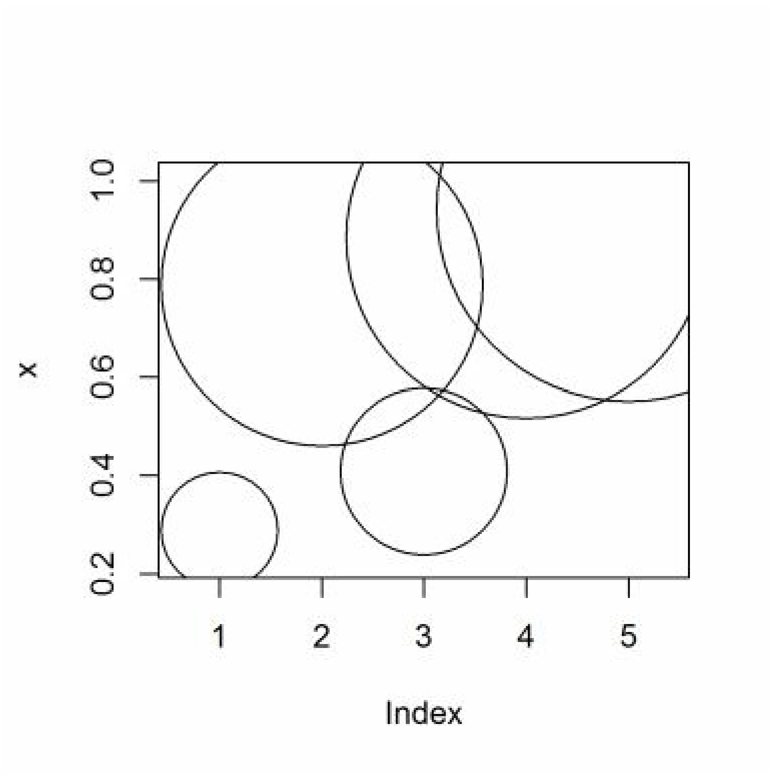
This section of the book shows you how to create fifty scientific data visualizations easily and quickly, using the free and widely available R computing package. It contains no theory, no complex equations, you don't even need to have prior experience. Select the chart of interest, type in the example and you are done. All charts in this section can be reproduced by you in ten minutes or less. Once you are comfortable with a particular data visualization, feel free to adapt it for your own purposes. Tips on how to do this are discussed throughout the text. Each chart is presented alongside step by step instructions on how to quickly construct it. This is followed by practical tips to enhance the visual experience. Finally, at the end of each chart the entire code used to create the chart is given for your reference. If you are in a hurry just copy out this section into R, make changes for your data and you are done!

9. Tips on Color and Shapes in R

Let’s work through an example. If this were a computer programming book, we would begin with a program to produce the statement “Hello World!”; since this is text about data visualization, let us begin with a random circle plot. Here is the code, type it directly into R:.

```
#random circles plot

set.seed(123) # set random number to known value

n = 5# number of circles to draw

x = runif(n) #generate n random numbers

size=abs(x)/sqrt(pi) # set the size of the circles

symbols(x,circles=size) # plot the circles
```

The first line allows you to replicate exactly the image below. It basically starts the random number generator at the same value every time. The second line indicates five circles will be drawn (to draw more set n to a larger number). The third line generates the random numbers. The fifth line sets the size of the circles as a function of the random number value. The final line draws the random circles image onto the R graphics device. You should now see on your R graphics device an image that looks like the one below:



Not that impressive but at least you have drawn something! Let's play a little with it. First let's make the circumference of the circles blue; to do this type:

symbols(x,circles=size,fg="blue")# turn circumference blue

You should now see an image that looks like this:

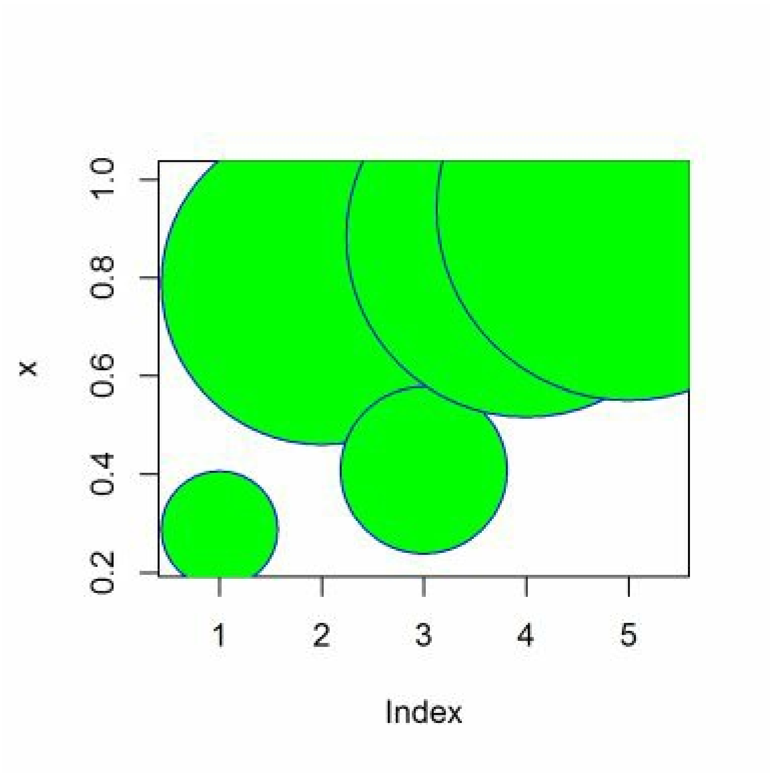
-

The argument fg changes the color of the circumference of the circle. R has a rich array of color options. To get a list of the basic color type colors() into the R consul and press return.

Back to our example. We would like to fill the circles with the color green. This can be accomplished in one line

symbols(x,circles=size,fg="blue",bg="green")#green circles

The resultant image is shown below:

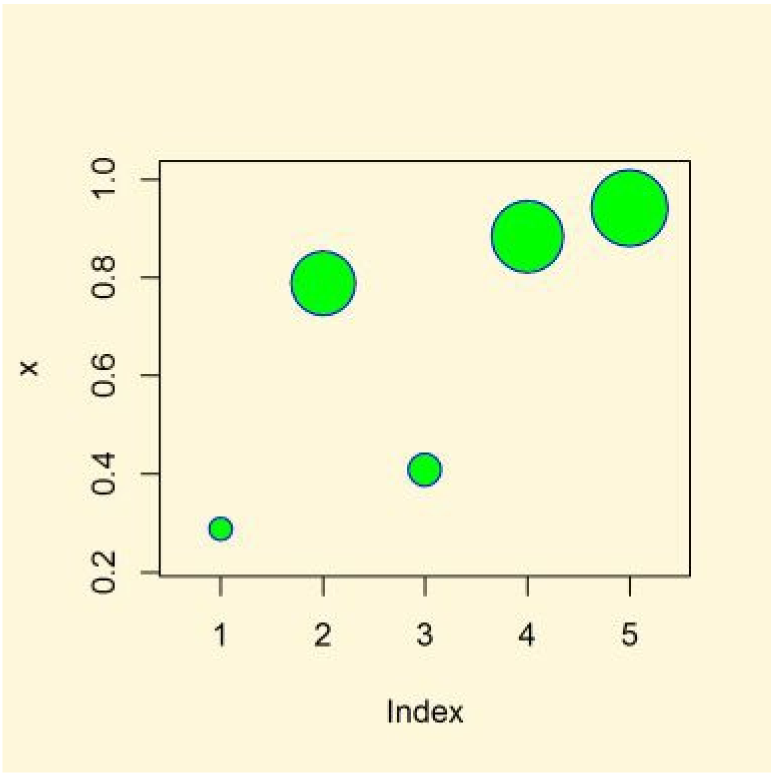


Since the circles are rather large, it might be a good idea to limit their size. This can be done by adding the inches argument to the symbols function. We will set the maximum size to 0.5 of an inch and set the background color of the image to cornsilk. This is achieve using the par() function.

par(bg="cornsilk")# set background color

symbols(x,circles=size,fg="blue",bg="green", inches=0.5)

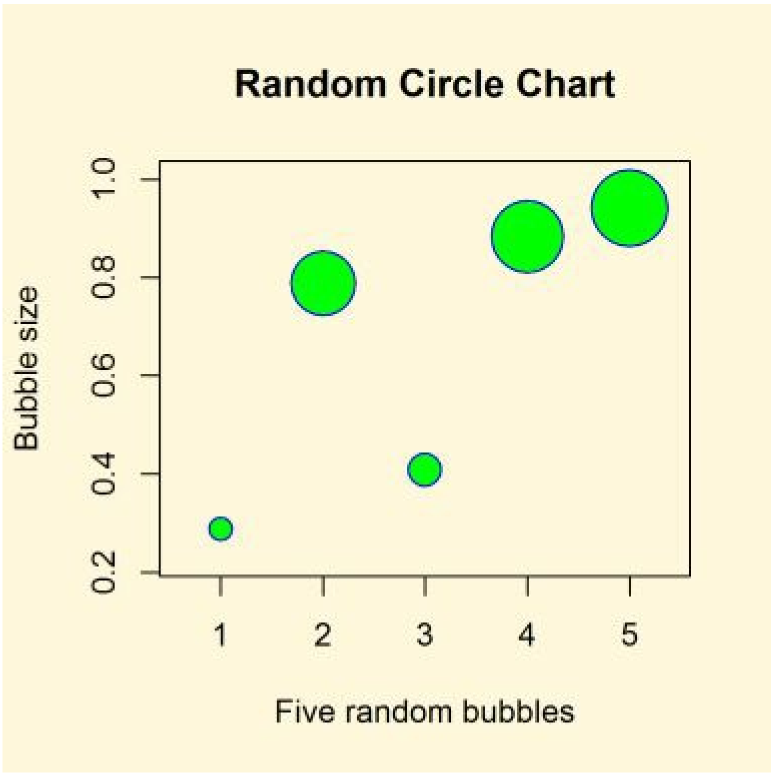
The image below is the result:



Of course, we will need to label the x and y axis and give the image a title. This can achieved by typing one line into R:

symbols(x,circles=size,fg="blue",bg="green", inches=0.2, main="Random Circle Chart", xlab="Five random bubbles",ylab="Bubble size")# random circles chart finished!

The diagram below is the final product:



In reality it only took two lines of code to produce this image:

par(bg="cornsilk")# set background color

symbols(x,circles=size,fg="blue",bg="green", inches=0.2, main="Random Circle Chart", xlab="Five random bubbles",ylab="Bubble size")# draw chart

Not bad for a couple of lines! But we can do even better when we visualize real data in subsequent sections.

Tip 1: To get help on a function, type help(function name) or ? function name. For example to get help on the par function you can type:

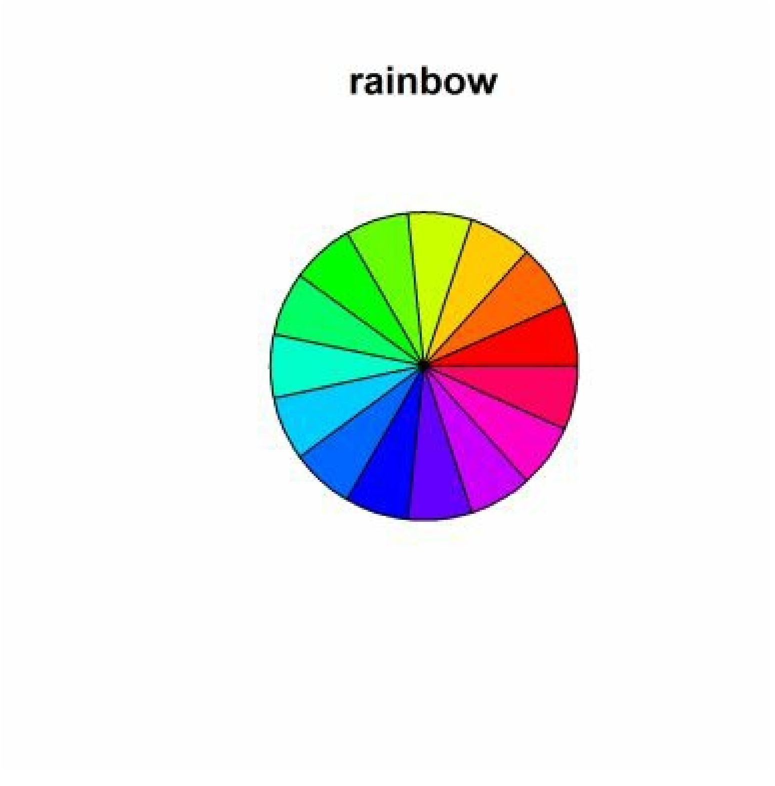
help(par)

or

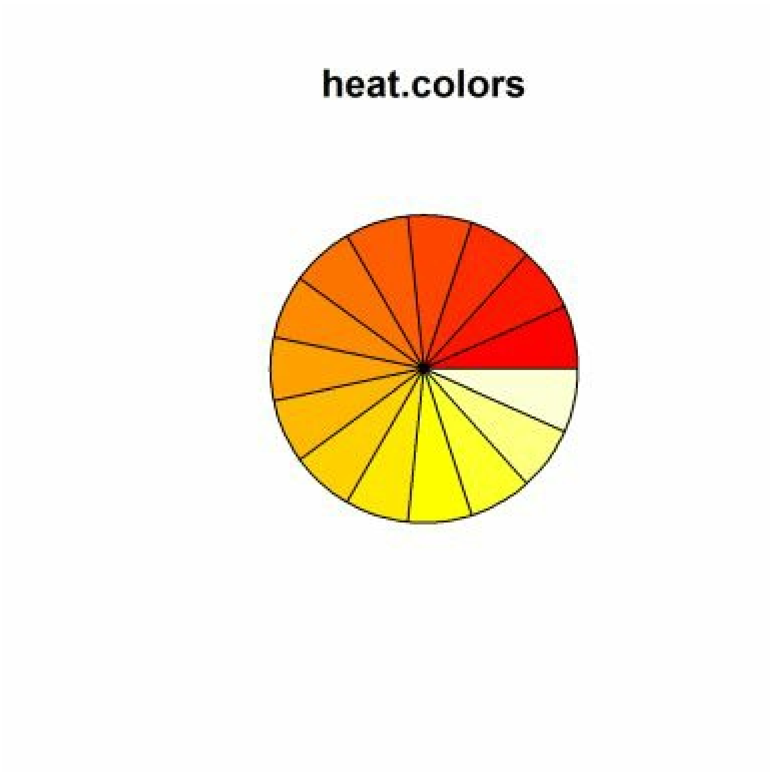
?par

A note on color in R

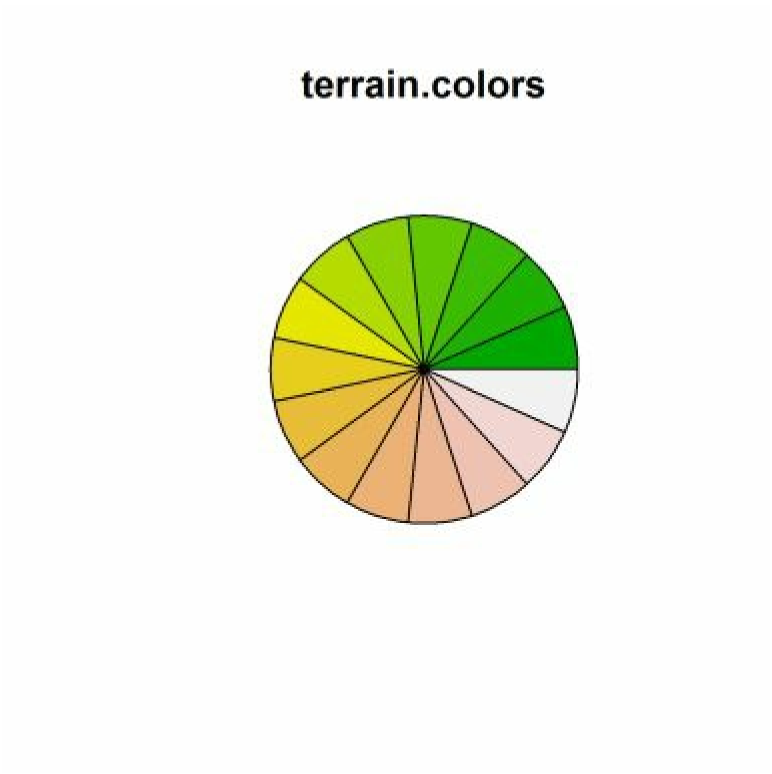
Traditionally scientific charts have used color sparingly. This has begun to change as more and more researchers become aware of both the power of color and the ease by which it can be added to an image. R is particularly rich in color. Let’s try some of the built in palettes right now! We begin with the rainbow palette:



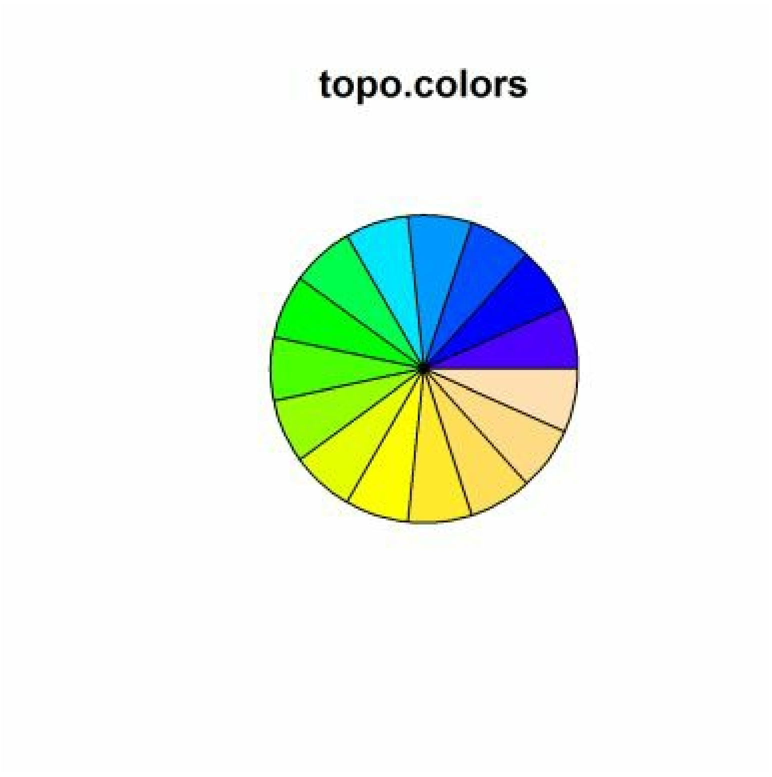
The heat.colors palette is great for heat maps:



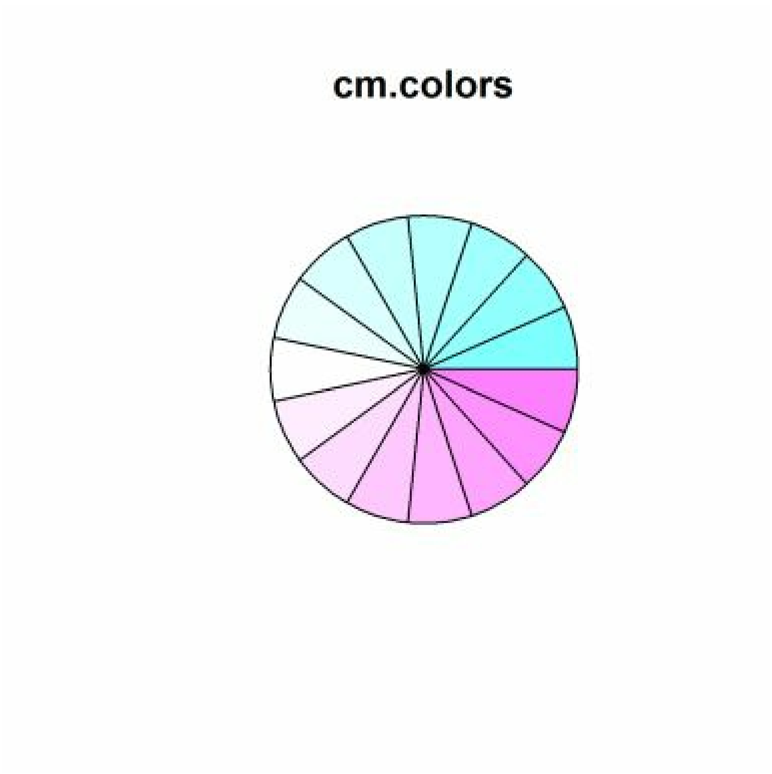
As is the terrain palette for geographical maps and topography:



The topo.colors palette are a little brighter:



You can even cool things down with cm.colors:



We have just scratched the surface. Creative ways to use these and other colors will be explored in the next sections of this text.

Now let’s get started!

10. Bubble plot

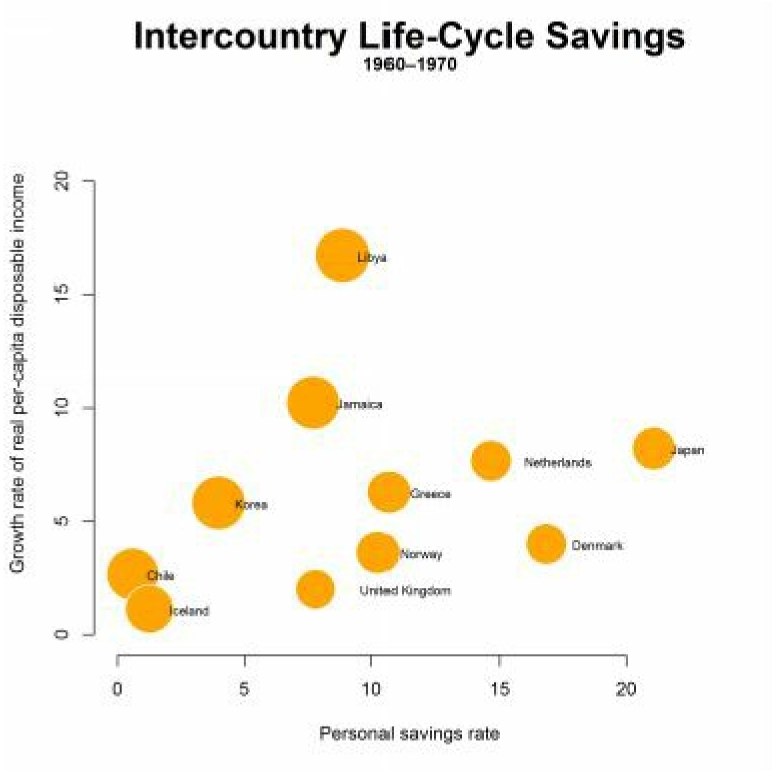


Figure 10 1 Lifecycle savings for a range of countries

How to read the chart

The chart displays the relationship between personal saving (x-axis), growth rate in disposable income (y-axis) and proportion of population less than 15 years old (size of bubble).

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

require(datasets)# contains the data
require (graphics)# used to draw the plot
attach(LifeCycleSavings)

Step 2: Clean and prepare the data

# identify the nations to use in the chart
nations<-c("Chile","Denmark","Greece", "Iceland", "Japan","Korea","Norway","Netherlands","United Kingdom", "Jamaica", "Libya")

```
# store nations data in a data frame called "data"
data<- LifeCycleSavings [match(nations, row.names(LifeCycleSavings)), ]
```
Step 3: Plot the chart
```
par(bg="white")# set the background color
# plot the axes and axes labels
plot(0:23, 0:23, type="n", xlab="Personal savings rate",ylab="Growth rate of real per-capita disposable income",frame.plot=F)
# plot the orange bubbles
symbols(data$sr, data$ddpi, circles = sqrt(data$pop15/pi),
    fg = "white", inches=0.25,bg="orange", add=T)
#add country names
text(data$sr, data$ddpi, rownames(data), cex=0.7,adj=c(-0.5,0.6),col="black")
#add title
title("Intercountry Life-Cycle Savings",col.main="black",line=2, cex.main =2)
title("1960–1970",col.main="black",line=1, cex.main =1)
```

Visualization quick tips

Tip 1: Change the background color of the plot by using the argument par(bg="my color"). The color of the bubbles is controlled by the argument bg in the symbols functions; you can also change the border around the bubbles by setting argument fg="my color".

Tip 2: The radius of the bubbles is controlled by the circles argument in the symbols function. The chart used circles = sqrt(data$pop15/pi). To ensure the bubbles are not to large the maximum size of each bubble was limited by setting inches=0.25. For larger bubbles increase this number, for smaller bubbles decrease the number, the default is 1 inch.

Tip 3: The title function offers considerable flexible in adding text to a chart. The color of the text can be changed by setting col.main ="my color". The argument cex.main controls the size of the text. The bubble chart used a title, with cex.main=2, and a subtitle with a smaller font size set by cex.main =1.

Quick reference R code

```
require(datasets)# contains the data
require (graphics)# used to draw the plot
attach(LifeCycleSavings)
nations<-c("Chile","Denmark","Greece", "Iceland",
"Japan","Korea","Norway","Netherlands","United Kingdom", "Jamaica",
"Libya" )

data<- LifeCycleSavings [match(nations, row.names(LifeCycleSavings)), ]
par(bg="white")
plot(0:23, 0:23, type="n", xlab="Personal savings rate",ylab="Growth rate of
real per-capita disposable income",frame.plot=F)
symbols(data$sr, data$ddpi, circles = sqrt(data$pop15/pi),
    fg = "white", inches=0.25,bg="orange", add=T)
text(data$sr, data$ddpi, rownames(data), cex=0.7,adj=c(-0.5,0.6),col="black")
title("Intercountry Life-Cycle Savings",col.main="black",line=2, cex.main =2)
title("1960–1970",col.main="black",line=1, cex.main =1)
```

11. Multivariate shaded time series plot

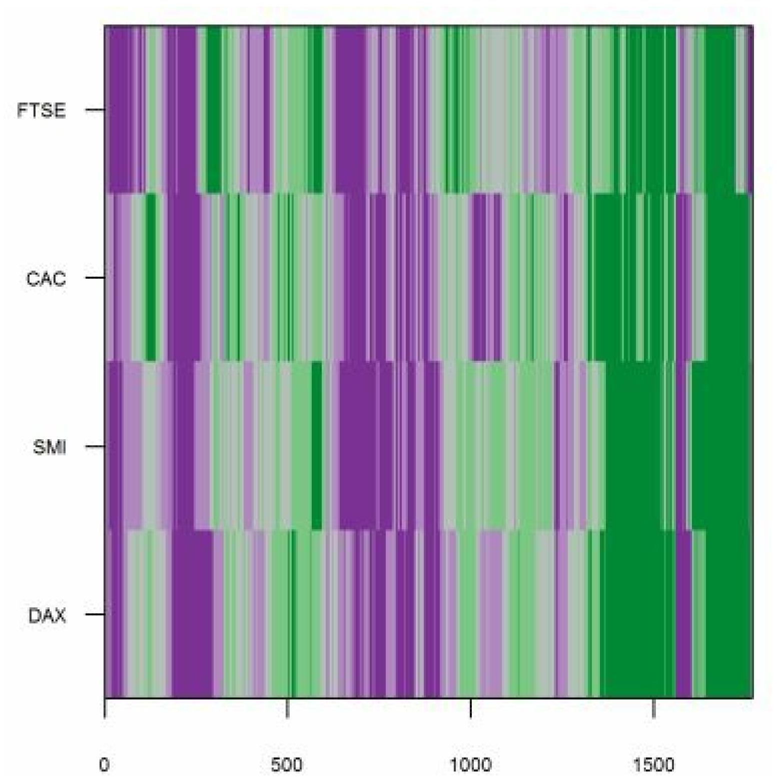


Figure 11 1 The 90 day price change of four European stock market indices

How to read the chart

The chart is constructed from daily data on the four major European stock market indices (FTSE, CAC, SMI and DAX) recorded over the period 1991 to 1998. The panel shows the time series for the 90 day price changes for each market index. The values have been discretized and assigned to five distinct categories. The shading at the extremes represents whether a high 90 day change (green) or low 90 day change (purple) in prices were observed. The deeper the purple, the lower the 90 day price change. The deeper the green, the higher the 90 day price change.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("mvtsplot")# you only need to do this once
install.packages("datasets") # you only need to do this once
require(datasets) # package which contains the data
require(mvtsplot) # loads chart drawing package into R
```

Step 2: Clean and prepare the data

```
data <- diff(EuStockMarkets ,90)
# The data are contained in the EuStockMarkets data frame. The function diff() calculates the 90 day difference. The cleaned observations are stored in the variable called data.
```

Step 3: Plot the chart

```
mvtsplot(data,norm ="internal", levels = 5,margin=FALSE)
```

Visualization quick tips

Tip 1: The mvtsplot function requires data in a matrix format. When using your own data remember to set the argument ncol equal to the number of columns in your data set.

Tip 2: You can supply your own column heading by setting colnames<-(data c("name.1","name.2","name.3",…). Again, make sure you have sufficient column names for the number of columns in your data.

Tip 3: The argument norm in the mvtsplot function addresses whether the categorization of the time series is based on within-series categories (norm = "internal") or using all the data in (norm = "global"). In the chart created in this chapter the categories were the four stock market indices (FTSE, CAC, SMI and DAX) with each categorization created using within-series data. Experiment with both norm = "internal" and norm = "global" to see if trends in the overall dataset appear more clearly.

Tip 4: The argument levels in the mvtsplot function controls how many colors are used in the plot. The default is levels =3. In the chart created in this chapter the values of each time series are divided into five categories ("very low", "low", "medium", "high" and "very high") and plotted using deep purple for very low values and deep green to represent high values. As a rule of thumb smooth data can usually be plotted with a larger number of levels; very noisy or spikey data typically needs to be plotted with fewer levels. Try experimenting with different values to see if trends in the overall dataset appear more clearly.

Quick reference R code

```
install.packages("mvtsplot")
install.packages("datasets")
require(datasets)
require(mvtsplot)
data <- diff(EuStockMarkets ,90
mvtsplot(data,norm ="internal", levels = 5,margin=FALSE)
```

NOTES

Write your notes below…

12. Multivariate marginal shaded time series plot

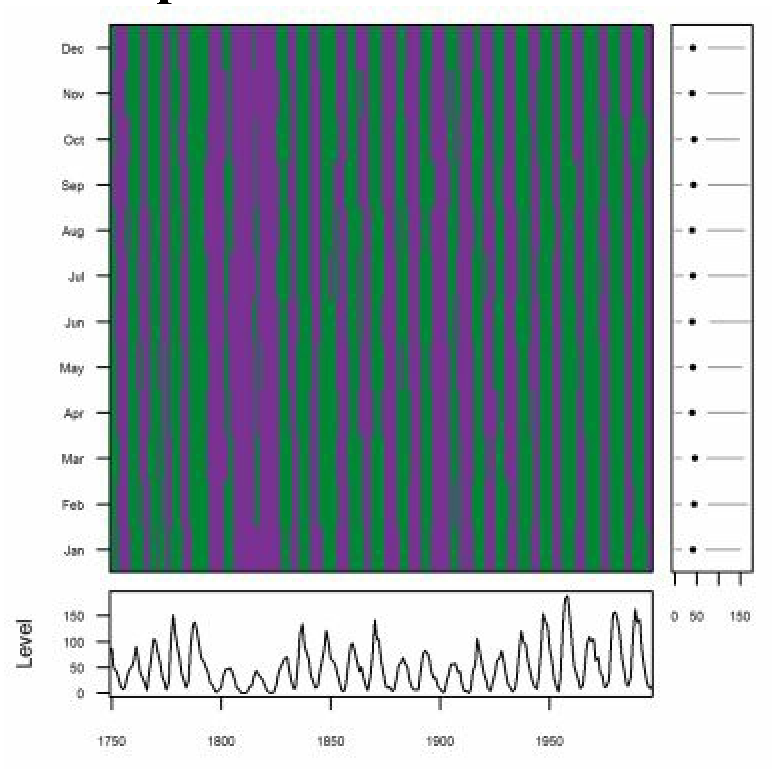


Figure 12 1 Monthly sunspots recorded over the period 1749 to 1997

How to read the chart

The chart is constructed from data on the monthly number of sunspots recorded over the period 1749 to 1997. The main panel shows the time series of sunspots for each month of the year. The values of each time series are discretized and assigned to distinct categories. The shading represents whether a high number (green) or low number (purple) of sunspots were recorded. The right panel contains box plots for each of the twelve months. The median value is represented by a large dot. The lower panel presents a time series plot of the median value of observations for each year.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("mvtsplot")# you only need to do this once
install.packages("datasets") # you only need to do this once
require(datasets) # package which contains the data
require(mvtsplot) # loads chart drawing package into R
```

Step 2: Clean and prepare the data

data<-matrix(data = sunspot.month, ncol = 12, byrow = TRUE)

# The sunspots data is contained in the data-frame sunspot.month. It contains monthly observations from 1749 to 1997. Since there are 12 months in the year the number of columns is set equal to 12 (ncol=12). The parameter byrow is set to TRUE because the data in sunspot.month is stored in rows. The cleaned observations are stored in the variable called data.

colnames(data) <- c("Jan","Feb","Mar","Apr","May","Jun","Jul", "Aug","Sep","Oct","Nov","Dec")# Add column names for each month of the year.

rownames(data)<-(1749:1997) # Data cover the years 1749 to 1997

time<-(1749:1997) # We use this variable to plot years 1749 to 1997 in the lower panel.

Step 3: Plot the chart

mvtsplot(data,xtime = time,norm ="global", levels = 2)

Visualization quick tips

Tip 1: The mvtsplot function requires data in a matrix format. When using your own data remember to set the argument ncol equal to the number of columns in your data set.

Tip 2: You can supply your own column heading by setting colnames<-(data c("name.1","name.2","name.3",…). Again, make sure you have sufficient column names for the number of columns in your data.

Tip 3: The argument norm in the mvtsplot function addresses whether the categorization of the time series is based on within-series categories (norm = "internal") or using all the data in (norm = "global"). In the chart created in this chapter the categories were months of the year using all the data. Experiment with both norm = "internal" and norm = "global" to see if trends in the overall dataset appear more clearly.

Tip 4: The argument levels in the mvtsplot function controls how many colors are used in the plot. The default is levels =3. In this case the values of each time series are divided into three categories ("low", "medium", and "high") and plotted using purple for low values, grey for medium values, and green to represent high values. As a rule of thumb smooth data can usually be plotted with a larger number of levels; very noisy or spikey data typically needs to be plotted with fewer levels. Try experimenting with different values to see if trends in the overall dataset appear more clearly.

Quick reference R code

```
install.packages("mvtsplot")
install.packages("datasets")
require(datasets)
require(mvtsplot)
data<-matrix(data = sunspot.month, ncol = 12, byrow = TRUE)
colnames(data) <- c("Jan","Feb","Mar","Apr","May","Jun","Jul",
"Aug","Sep","Oct","Nov","Dec")
rownames(data)<-(1749:1997)
time<-(1749:1997)
mvtsplot(data,xtime = time,norm ="global", levels = 2)
```

NOTES

Write your notes below…

13. Multivariate split marginal shaded time series plot

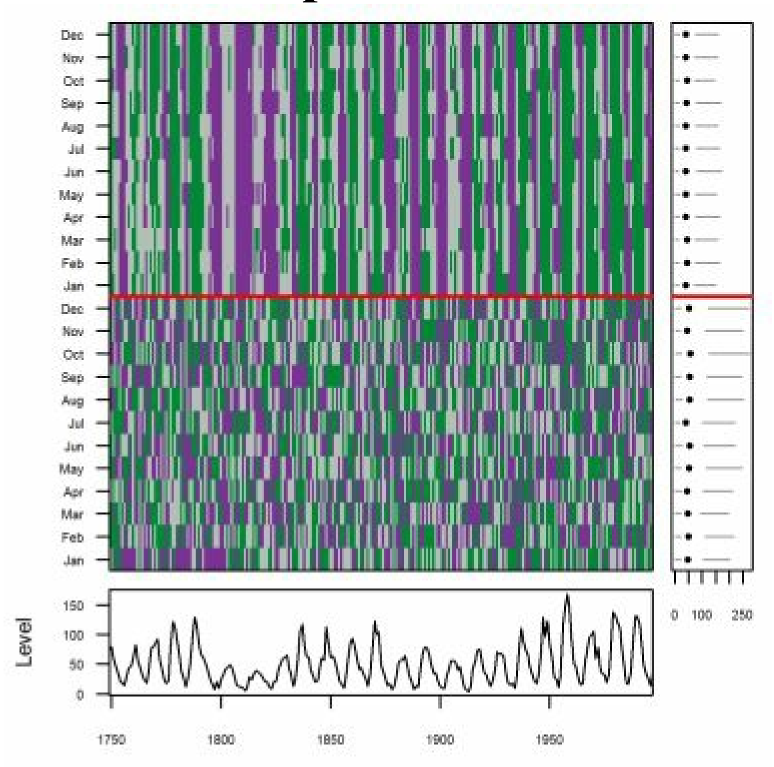


Figure 13 1 Monthly sunspot data versus random log-normal data

How to read the chart

The chart is constructed from data on the monthly number of sunspots recorded over the period 1749 to 1997. The main panel is divided into two sections by a red line. The section above the red line shows the time series of sunspots for each month of the year. The area below the red line presents simulated data of sunspots from a lognormal distribution. The values of both the actual and simulated time series are discretized and assigned to distinct categories. The shading represents whether a high number (green) or low number (purple) of sunspots were recorded. The right panel contains box plots for each series for each month of the year. The median value is represented by a large dot. The lower panel presents a time series plot of the median value of both sets of observations by year.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

install.packages("mvtsplot")# you only need to do this once

```
install.packages("datasets") # you only need to do this once
require(datasets) # package which contains the data
require(mvtsplot) # loads chart drawing package into R
```

Step 2: Clean and prepare the data

```
data<-matrix(data = sunspot.month, ncol = 12, byrow = TRUE)
# transform raw data in to matrix format
colnames(data) <- c("Jan","Feb","Mar","Apr","May","Jun","Jul",
"Aug","Sep","Oct","Nov","Dec")# Add labels for the months of the year.
rownames(data)<-(1749:1997)# Data cover the years 1749 to 1997.
time<-(1749:1997)# Data cover the years 1749 to 1997.
set.seed(1234)# this allows you to replicate the chart.
random.data<- matrix(rlnorm (12*nrow(data),mean=3.9,sd=1.2),
nrow(data),12) # Generate log-normal random variables.
colnames(random.data) <- c("Jan","Feb","Mar","Apr","May" ,"Jun", "Jul",
"Aug","Sep","Oct","Nov","Dec")
# Give the log-normal data columns a heading representing months of the year.
group<-c(0,0,0,0,0,0, 0,0,0,0,0,0,1,1,1,1,1,1, 1,1,1,1,1,1)
# group is used to separate the sunspot data from the random data.
data<-cbind(random.data,data)
# cbind is a function which combines the actual sunspot data with the simulated
data.
```

Step 3: Plot the chart

```
mvtsplot(data,xtime = time, norm ="internal", levels = 3,
group=group,gcol="red")
```

Visualization quick tips

Tip 1: The mvtsplot function requires data in a matrix format. When using your own data remember to set the argument ncol equal to the number of columns in your data set.

Tip 2: You can supply your own column heading by setting colnames<-(data c("name.1","name.2","name.3",…). Again, make sure you have sufficient column names for the number of columns in your data.

Tip 3: The argument norm in the mvtsplot function addresses whether the categorization of the time series is based on within-series categories (norm = "internal") or using all the data in (norm = "global"). Experiment with both norm = "internal" and norm = "global" to see if trends in the overall dataset appear more clearly.

Tip 4: The argument levels in the mvtsplot function controls how many colors are used in the plot. The default is levels =3 which was used in the chart in this chapter. For the default value purple is assigned to low values, grey to medium values, and green to high values. As a rule of thumb smooth data can usually be plotted with a larger number of levels; very noisy or spikey data typically needs to be plotted with fewer levels. Try experimenting with different values to see if trends in the overall dataset appear more clearly.

Quick reference R code

```
install.packages("mvtsplot")
install.packages("datasets")
require(datasets)
require(mvtsplot)
data<-matrix(data = sunspot.month, ncol = 12, byrow = TRUE)
colnames(data) <- c("Jan","Feb","Mar","Apr","May","Jun","Jul",
"Aug","Sep","Oct","Nov","Dec")
rownames(data)<-(1749:1997)
time<-(1749:1997)
set.seed(1234)
random.data<- matrix(rlnorm (12*nrow(data),mean=3.9,sd=1.2),
nrow(data),12)
colnames(random.data) <- c("Jan","Feb","Mar","Apr","May","Jun","Jul",
"Aug","Sep","Oct","Nov","Dec")#
group<-c(0,0,0,0,0,0, 0,0,0,0,0,0,1,1,1,1,1,1, 1,1,1,1,1,1)
```

```
data<-cbind(random.data,data)
mvtsplot(data, xtime = time, norm ="internal", levels = 3, group=group,
gcol="red")
```

NOTES

Write your notes below…

14. Word cloud

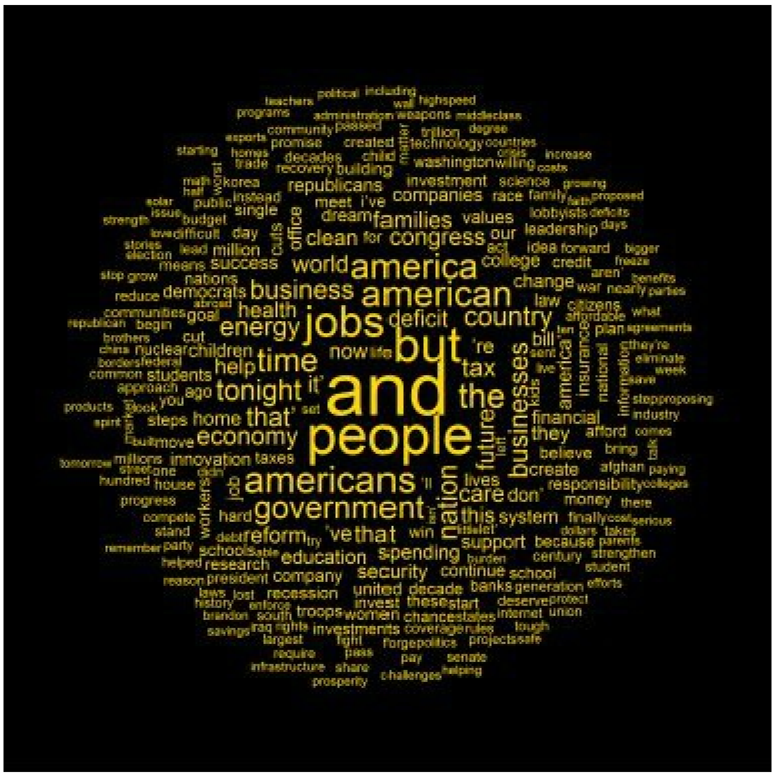


Figure 14 1 United States State of the Union Addresses (2010 and 2011)

How to read the image

The image displays the most frequently used words by President Obama in his State of the Union addresses during 2010 and 2011. Word frequency is visualized by the size of the text; the more frequent the word the-e larger it appears in the image.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("tm") # this package contains the data
install.packages("wordcloud") # this package generates the plot
require(tm)
require(wordcloud)
```

Step 2: Clean and prepare the data

```
data(SOTU)# contains the text of the Presidential addresses.
# we only want the words so we remove punctuation
text <- tm_map(SOTU, removePunctuation)
```

# remove standard stop words. These are standard words which are removed prior to processing.
text <- tm_map(text, function(x)removeWords (x,stopwords()))
# put cleaned data in appropriate format
tdm <- TermDocumentMatrix(text)
m <- as.matrix(tdm)
v <- sort(rowSums(m),decreasing=TRUE)
d <- data.frame(word = names(v),freq=v)
Step 3: Plot the chart
par(bg="black")# set background color
wordcloud(d$word,d$freq, random.order=FALSE,min.freq=5 , color="gold")points(fly.2, plot.info=res, col="darkblue", stack=TRUE)

Visualization quick tips

Tip 1: To create a word cloud your data needs to be in a corpus format. One way to do this is to transform your text into a data frame and then to a corpus (a collection of text documents). Something along the lines of
library(tm)
textframe <- do.call("rbind", lapply(mytext, as.data.frame))
myCorpus <- Corpus(VectorSource(textframe$text))
Note VectorSource requires character vectors.

Tip 2: The argument min.freq controls the minimum word count displayed in the image. Words which occur less than min.freq are excluded from the word cloud.

Tip 3: To limit the maximum number of words in the image add the max.words argument to the wordcloud function i.e. wordcloud(max.word=100,…).

Tip 4: To change the color of the printed text use the argument colors = “my_favorite_color”.

Quick reference R code

```
install.packages("tm") # this package contains the data
install.packages("wordcloud") # this package generates the plot
require(tm)
require(wordcloud)
data(SOTU)
text <- tm_map(SOTU, removePunctuation)
text <- tm_map(text, function(x)removeWords(x,stopwords()))
tdm <- TermDocumentMatrix(text)
        m <- as.matrix(tdm)
        v <- sort(rowSums(m),decreasing=TRUE)
        d <- data.frame(word = names(v),freq=v)
par(bg="black")
wordcloud(d$word,d$freq, random.order=FALSE,min.freq=5 , color="gold")
```

NOTES

Write your notes below…

*

15. Commonality word cloud

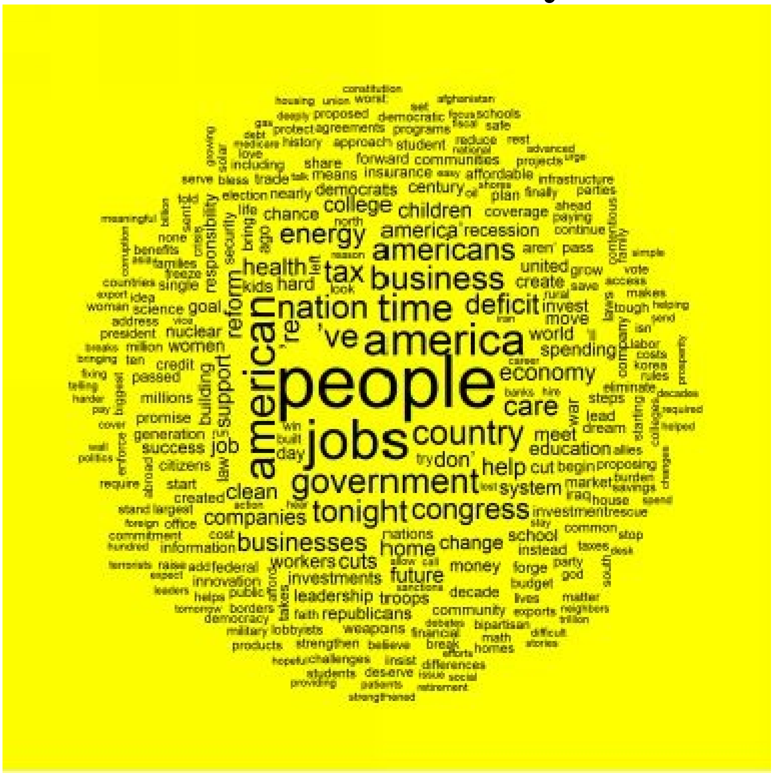


Figure 15 1 United States State of the Union Addresses (2010 and 2011)

How to read the image

The image displays high frequently commonality words used by President Obama in his State of the Union addresses during 2010 and 2011. Word commonality is visualized by the size of the text; the greater the commonality of a word the larger it appears in the image.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("tm") # this package contains the data
install.packages("wordcloud") # this package generates the plot
require(tm)
require(wordcloud)
```

Step 2: Clean and prepare the data

```
data(SOTU)# contains the text of the Presidential addresses.
text <- SOTU
#below the data is cleaned
```

```
text <- tm_map(text, removePunctuation)
text <- tm_map(text, tolower)
text <- tm_map(text, removeNumbers)
# remove standard stop words. These are standard words which are removed prior to processing.
text <- tm_map(text, function(x)removeWords(x,stopwords()))
# put cleaned data in appropriate format
term.matrix <- TermDocumentMatrix(text)
term.matrix <- as.matrix(term.matrix)
```

Step 3: Plot the chart

```
par(bg="yellow")# set chart background color to yellow.
set.seed(123)# use to allow the replication of the image
commonality.cloud(term.matrix,random.order=FALSE)
```

Visualization quick tips

Tip 1: To create a commonality word cloud your data needs to be in a corpus format. One way to do this is to transform your text into a data frame and then to a corpus (a collection of text documents). Your R code with look similar to the text below

```
library(tm)
textframe <- do.call("rbind", lapply(mytext, as.data.frame))
myCorpus <- Corpus(VectorSource(textframe$text))
```

Note VectorSource requires the source be character vectors.

Tip 2: To limit the maximum number of words in the image add the max.words argument to the commonality.cloud function i.e. commonality.cloud (max.word=100,…).

Tip 3: To see the list of stop words used type the following into the R console window: stopwords("en")

Quick reference R code

```
install.packages("tm") # this package contains the data
install.packages("wordcloud") # this package generates the plot
require(tm)
require(wordcloud)
data(SOTU)
text <- SOTU
text <- tm_map(text, removePunctuation)
text <- tm_map(text, tolower)
text <- tm_map(text, removeNumbers)
text <- tm_map(text, function(x)removeWords(x,stopwords()))
term.matrix <- TermDocumentMatrix(text)
term.matrix <- as.matrix(term.matrix)

par(bg="yellow")
set.seed(123)
commonality.cloud(term.matrix,random.order=FALSE)
```

NOTES

Write your notes below…

16. Comparison word cloud

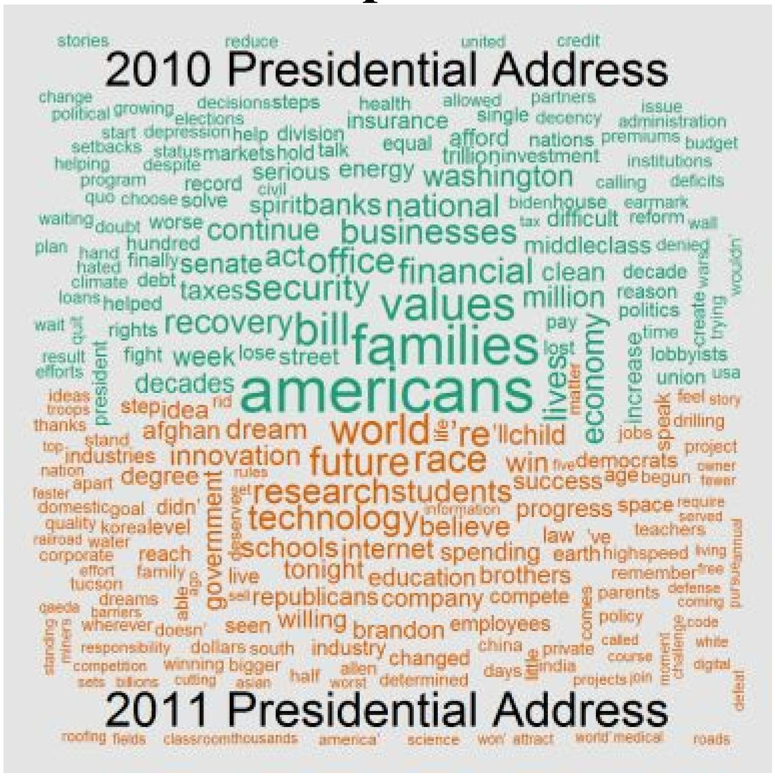


Figure 16 1 United States State of the Union Addresses (2010 and 2011)

How to read the image

The image displays high frequently words used by President Obama in his State of the Union addresses during 2010 and 2011. The top half of the diagram are frequently used word from the 2010 address (green), the bottom half are high frequency words from the 2011 address (burnt orange).

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("tm") # this package contains the data
install.packages("wordcloud") # this package generates the plot
require(tm)
require(wordcloud)
```

Step 2: Clean and prepare the data

```
data(SOTU)# contains the text of the Presidential addresses.
text <- SOTU
#below the data is cleaned
```

```
text <- tm_map(text, removePunctuation)
text <- tm_map(text, tolower)
text <- tm_map(text, removeNumbers)
# remove standard stop words. These are standard words which are removed
prior to processing.
text <- tm_map(text, function(x)removeWords(x,stopwords()))

# put cleaned data in appropriate format
term.matrix <- TermDocumentMatrix(text)
term.matrix <- as.matrix(term.matrix)
colnames(term.matrix) <- c("2010 Presidential Address","2011 Presidential
Address")
```

Step 3: Plot the chart

```
set.seed(1234)
par(bg="gray90")# set background color to gray.
comparison.cloud(term.matrix,max.words=300,random.order=FALSE)
```

Visualization quick tips

Tip 1: To create a comparison word cloud your data needs to be in a corpus format. One way to do this is to transform your text into a data frame and then to a corpus (a collection of text documents). Your R code with look similar to the text below

```
library(tm)
textframe <- do.call("rbind", lapply(mytext, as.data.frame))
myCorpus <- Corpus(VectorSource(textframe$text))
```

Note VectorSource requires that the source is a character vector.

Tip 2: To limit the maximum number of words in the image add the max.words to the comparision.cloud function i.e. comparision.cloud (max.word=100,…).

Tip 3: To see the list of stop words used type the following into the R console window: stopwords("en")

Quick reference R code

```
install.packages("tm") # this package contains the data
install.packages("wordcloud") # this package generates the plot
require(tm)
require(wordcloud)
data(SOTU)
text <- SOTU
text <- tm_map(text, removePunctuation)
text <- tm_map(text, tolower)
text <- tm_map(text, removeNumbers)
text <- tm_map(text, function(x)removeWords(x,stopwords()))
term.matrix <- TermDocumentMatrix(text)
term.matrix <- as.matrix(term.matrix)
colnames(term.matrix) <- c("2010 Presidential Address","2011 Presidential
Address")
set.seed(1234)
par(bg="gray90")
comparison.cloud(term.matrix,max.words=300,random.order=FALSE)
```

NOTES

Write your notes below…

17. State and cities with capital map

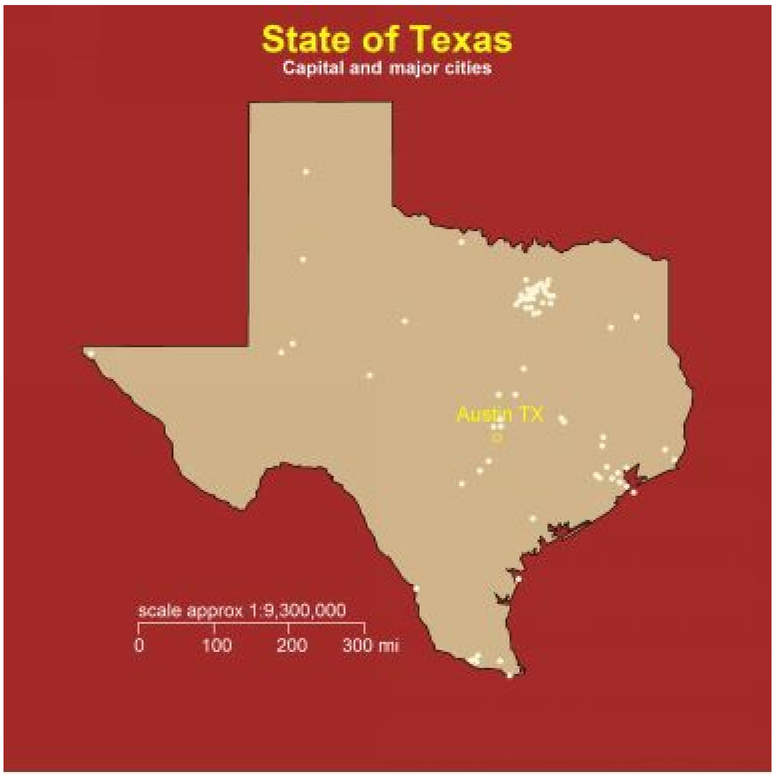


Figure 17 1 Texas cities with scale in miles

How to read the image

The image displays the scale, cities and capital city of the US state of Texas. Major cities are denoted by white dots. Austin, the capital city is highlighted in yellow.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("maps")# this package generates the map
library(maps)
```

Step 2: Clean and prepare the data

```
data(us.cities) # contains the data on US cities.
```

Step 3: Plot the chart

```
# set background color to brown and text to white
par(col="white",bg="brown")
# draw tan colored map of Texas
```

```
map("state", "Texas",col="tan",fill=TRUE)
# add a scale in miles to the plot
map.scale(cex=1,metric=F,relwidth=.3)
# add capital in yellow
map.cities(us.cities, country="TX", capitals=2, par(pch=23,col="yellow"))
#add Texas cities in cornsilk
map.cities(us.cities, country="TX",col="cornsilk",pch=19)
#Add title to map
title="State of Texas"
sub= "Capital and major cities"
title(title,col.main="yellow",line=2, cex.main =2)
title(sub,col.main="white",line=1, cex.main =1)
```

Visualization quick tips

Tip 1: The map function allows you to draw a wide variety of maps for US states and other countries. For example map("state", "iowa") will draw an outline of the state of Iowa. To fill in a state in a solid color use fill=TRUE.

Tip 2: Change the size of the scale by changing the cex argument in map.scale. Values greater 1 increase the size. To use metric distances rather than imperial set the argument metric =T.

Tip 3: You can extend the length of the scale by using the relwidth argument in the map.scale function. The larger the value passed to the argument, the larger the distance measured by the scale.

Quick reference R code

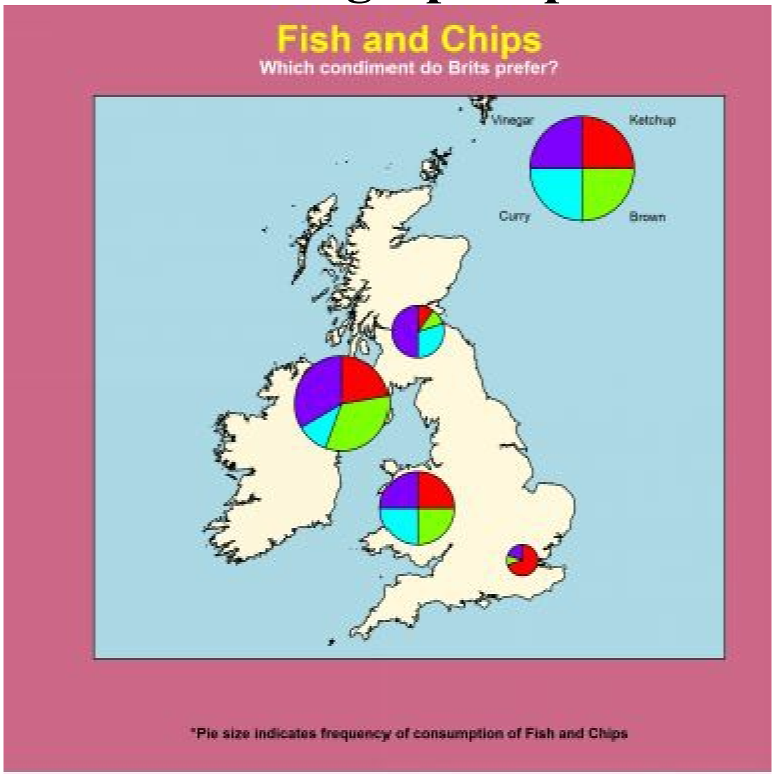
```
install.packages("maps")
library(maps)
data(us.cities)
```

```
par(col="white",bg="brown")
map("state", "Texas",col="tan",fill= TRUE)
map.scale(cex=1,metric=F,relwidth=.3)
map.cities(us.cities, country="TX", capitals=2, par(pch=23,col="yellow"))
map.cities(us.cities, country="TX",col="cornsilk",pch=19)
title="State of Texas"
sub= "Capital and major cities"
title(title,col.main="yellow",line=2, cex.main =2)
title(sub,col.main="white",line=1, cex.main =1)
```

NOTES

Write your notes below…

18. Geographic pie chart map



Fish and Chips
Which condiment do Brits prefer?
Vinegar
Ketchup
Curry
Brown
*Pie size indicates frequency of consumption of Fish and Chips

Figure 18 1 Condiment choice for fish and chips by the British

How to read the image

The image displays the fish and chips condiment choice by the consumers in London, England; Cardiff, Wales; Glasgow, Scotland and Belfast, Northern Ireland. The size of the pie charts reflect the relative consumption by consumers in each city.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("mapplots")#contains the data and draws image
require(mapplots)
```

Step 2: Clean and prepare the data

```
data(coast)# contains the data needed to draw the British Isles
xlim <- c(-10.5,3) # set longitude range on chart
ylim <- c(50,60) #set latitude range on chart
```

Step 3: Plot the chart

```
par(bg="palevioletred3")# set background color
# next we create the outline for the basic plot
```

```
basemap(xlim, ylim, main = "",xlab="",ylab="", axes = FALSE,frame.plot =
FALSE)
# now draw the map of the British isles with color set to cornsilk
draw.shape(coast, col='cornsilk')
# select the colors for the pie charts
col <- rainbow(4)
# add the pie charts
add.pie(z=c(0.7, 0.1, 0, 0.2), x=0, y=51.5,
radius=0.3,labels=NA,col=col)# London, England
add.pie(z=c(0.1, 0.1, 0.3, 0.5), x=-3.5, y=55.87,
radius=0.5,labels=NA, col=col)#Glasgow, Scotland
add.pie(z=c(0.25, 0.25, 0.25, 0.25), x=-3.5, y=52.5,
radius=0.7,labels=NA, col=col) # Cardiff,Wales
add.pie(z=c(0.2, 0.35, 0.1, 0.35), x=-6, y=54.5,
radius=0.9,labels=NA, col=col)# Belfast, Northern Ireland

# add the legend
legend.pie(2,59,labels=c('Ketchup','Brown ','Curry ','Vinegar'), radius=1, bty =
"n", col=col,
cex=0.7, label.dist=1.3)
#add a main title
title("Fish and Chips",col.main="yellow",line=2, cex.main =2)
#add a subtitle
title("Which condiment do Brits prefer?",col.main="white",line=1, cex.main
=1)
# add comments at bottom of the chart
title("*Pie size indicates frequency of consumption of Fish and
Chips",col.main="black",line=-29, cex.main =0.8)
```

Visualization quick tips

Tip 1: You can set a title directly in the basemap function by setting main="my title"; If you would like to have a longitude/ latitude grid around the images set the argument axes = TRUE and axes = FALSE,frame.plot = TRUE. Labels for the x and y axes can be entered by xlab="x axis title",ylab="y axis title".

Tip 2: The map used the argument col <- rainbow(4) to generate four pie chart colors using the rainbow function. A quick and easy alternatives which often looks great is col <- terrain.colors(n), where n is the number of colors you want to generate..

Tip 3: The add.pie function is easy and straightforward to use. The first argument takes the values for the pie chart, the second two arguments the x and y location of the pie chart, and the third argument the size of the pie chart. The label.dist argument is very useful as it specifies the distance between pie chart labels. The images used label.dist=1.3. Try experimenting with larger and smaller values until you find a distance that is visually appealing to you.

Quick reference R code

```
install.packages("mapplots")
require(mapplots)
data(coast)
xlim <- c(-10.5,3)
ylim <- c(50,60)
par(bg="palevioletred3")
basemap(xlim, ylim, main = "",xlab="",ylab="", axes = FALSE,frame.plot =
FALSE)
draw.shape(coast, col='cornsilk')
col <- rainbow(4)
add.pie(z=c(0.7,          0.1,          0,          0.2), x=0, y=51.5,
radius=0.3,labels=NA,col=col)# London, England
add.pie(z=c(0.1,          0.1,          0.3,          0.5), x=-3.5, y=55.87,
radius=0.5,labels=NA, col=col)#Glasgow, Scotland
add.pie(z=c(0.25,          0.25,          0.25,          0.25), x=-3.5, y=52.5,
radius=0.7,labels=NA, col=col) # Cardiff,Wales
add.pie(z=c(0.2,          0.3,          0.1,          0.3), x=-6, y=54.5,
radius=0.9,labels=NA, col=col)# Belfast, Northern Ireland
```

NOTES

Write your notes below…

19. US state income tax map

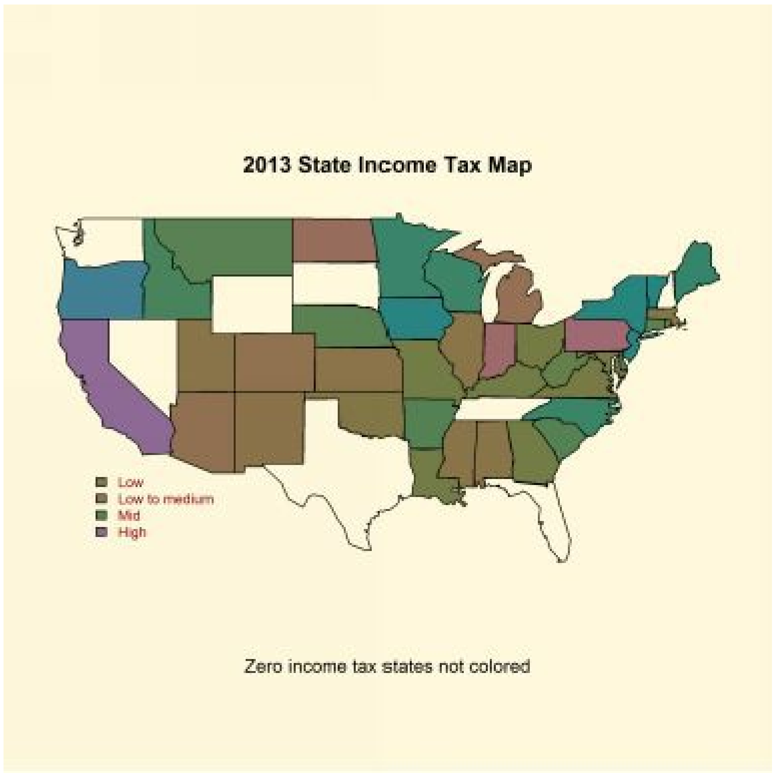


Figure 19 1 State income tax map

How to read the image

The image displays the level of state income tax by state in the United States during 2013. The highest income tax state is California with appears shaded purple. Zero state income tax states, such as Texas, are not colored.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("maps")#used to draw the map
install.packages("plotrix")#used to  color the map
library(maps)
library(plotrix)
```

Step 2: Clean and prepare the data

```
map("usa")#load the US map
data(votes.repub) # this contains names of US States
# enter 2013 highest income tax rates by state
tax=c(5.0          ,NA          ,4.5          ,7.0          ,12.3
,4.6        ,6.7        ,6.8        ,NA         ,6.0        ,11.0
,7.4        ,5.0        ,3.4        ,9.0        ,4.9        ,6.0
,6.0        ,8.0        ,5.8        ,5.3        ,4.3        ,7.9
,5.0        ,6.0        ,6.9        ,6.8        ,NA         ,NA
,9.0        ,4.9        ,8.8        ,7.8        ,4.0        ,5.9
,5.3        ,9.9        ,3.1        ,6.0        ,7.0        ,NA
,NA         ,NA         ,5.0        ,9.0        ,5.8        ,NA
,6.5        ,7.8        ,NA)
```

```
# we don't want to waste time typing in state names, so below is a super-fast way of getting what we want.
temp<-cbind(votes.repub,tax)
# now drop all the other columns and just leave our tax data
data <- temp[,c(32)]
# now match state names for use in the plot
state.to.map <- match.map("state", state.name, exact = F)
# store the result in a variable called x
x <- data[state.to.map]
```

Step 3: Plot the chart

```
# set the colors we will plot
state.col<-color.scale(x,c(0,300),35,50,  color.spec="hcl")
par(bg="cornsilk")#set background color
# plot the map with colors
map("state",fill=TRUE,col=state.col)
#add title and subtitle
title(main="2013 State Income Tax Map",
   sub="Zero income tax states not colored")
#add legend
legend(x="bottomleft", inset=.05,
bty="n", cex=0.75, text.col ="darkred",
c("Low", "Low to medium", "Mid","High"), fill= state.col)
```

Visualization quick tips

Tip 1: For an exact match of state names set exact =TRUE in the match.map function.

Tip 2: Playing with colors is fun. The map used the argument color.scale with color.spec="hcl". Good alternatives are "rgb" and "hsv". The function color.scale calculates a sequence of colors by a linear transformation of the numeric values supplied into the ranges for the three color parameters (the map used c(0,300),35,50). Adjust the value of each parameter to change the visual flavor of the map.

Tip 3: You adjust the location of the legend by specifying two arguments (x and y), these determine the horizontal and vertical location. We used a fast alternative by specifying the location with a shortcut legend("bottomleft",…). You can also specify "topright", "bottomright", "topleft". However, greater precision can be obtained by specifying the x and y parameters separately.

Quick reference R code

```
install.packages("maps")
install.packages("plotrix")
library(maps)
library(plotrix)
map("usa")
data(votes.repub)
tax=c(5.0        ,NA        ,4.5        ,7.0        ,12.3
,4.6        ,6.7        ,6.8        ,NA        ,6.0        ,11.0
,7.4        ,5.0        ,3.4        ,9.0        ,4.9        ,6.0
,6.0        ,8.0        ,5.8        ,5.3        ,4.3        ,7.9
,5.0        ,6.0        ,6.9        ,6.8        ,NA        ,NA
,9.0        ,4.9        ,8.8        ,7.8        ,4.0        ,5.9
,5.3        ,9.9        ,3.1        ,6.0        ,7.0        ,NA
,NA        ,NA        ,5.0        ,9.0        ,5.8        ,NA
,6.5        ,7.8        ,NA)
temp<-cbind(votes.repub,tax)
```

```
# we don’t want to waste time typing in state names, so this is a neat trick
# now drop all the other columns and just leave our tax data
data <- temp[,c(32)]
state.to.map <- match.map("state", state.name, exact = F)
x <- data[state.to.map]
state.col<-color.scale(x,c(0,300),35,50,  color.spec="hcl")
par(bg="cornsilk")
map("state",fill=TRUE,col=state.col)
title(main="2013 State Income Tax Map",
sub="Zero income tax states not colored")
legend(x="bottomleft", inset=.05,
bty="n", cex=0.75, text.col ="darkred",
c("Low", "Low to medium", "Mid","High"), fill= state.col)
```

NOTES

Write your notes below…

20. US Presidential election map

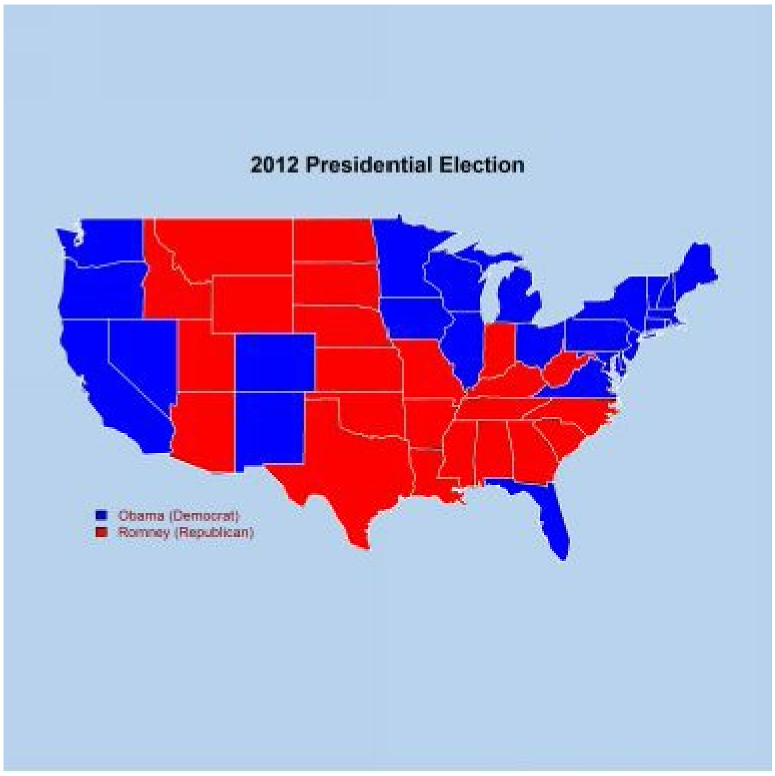


Figure 20 1 Obama v Romney in battle for US presidency

How to read the image

The image displays the outcome of the 2012 US Presidential election. States won by Obama are colored blue, states won by Romney are colored red.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("maps")#used to draw the map
install.packages("plotrix")#used to  color the map
library(maps)
library(plotrix)
```

Step 2: Clean and prepare the data

```
map("usa")#load the US map
data(votes.repub) # this contains names of US States
# enter the election results note 1 equals a republican win
result=c(1          ,1          ,1          ,1          ,0          ,0
,0          ,0          ,0          ,1          ,0          ,1          ,0
,1          ,0          ,1          ,1          ,1          ,0          ,0
,0          ,0          ,0          ,1          ,1          ,1          ,1
,0          ,0          ,0          ,0          ,0          ,1          ,1
,0          ,1          ,0          ,0          ,0          ,1          ,1
,1          ,1          ,1          ,0          ,0          ,0          ,1
,0          ,1)
# we don't want to waste time typing in state names, so below is a super-fast way of getting what we want.
temp<-cbind(votes.repub,result)
# now drop all the other columns and just leave our election data
data <- temp[,c(32)]
data[data == 0] <- NA# replace zeros (Democrat) with NA's
# match state names and store in variable x
state.to.map <- match.map("state", state.name, exact = F)
x <- data[state.to.map]
```

Step 3: Plot the chart

```
# we begin with the Republican color - red
state.col<-color.scale(x, 1,0, 0,  color.spec="rgb")
par(bg="slategray2")# set background color of chart
#draw the map with Republican wins in red
map("state",fill=TRUE,col=state.col, )

# replace Republican with Democrat for next map
data[is.na(data)] <- 0.0
data[data == 1] <- NA
x <- data[state.to.map]
# set color to blue
state.col<-color.scale(x, 0,0, 1,  color.spec="rgb")
#plot Democrat states in blue
map("state",fill=TRUE,col=state.col,add=TRUE, boundary = FALSE)
map("state",fill=F,add=TRUE, boundary = T, col="white")
# add title
```

```
title(main="2012 Presidential Election")
# add legend
legend(x="bottomleft", inset=.05,
bty="n", cex=0.75, text.col ="darkred",
c("Obama (Democrat)", "Romney (Republican)"), fill= c("blue","red"))
```

Visualization quick tips

Tip 1: For an exact match of state names set exact =TRUE in the match.map function.

Tip 2: Playing with colors is fun. The map used the argument color.scale with color.spec="rgb" to generate the red and blue colors of the political parties. Quick and easy alternatives are "hcl" and "hsv".

Tip 3: You adjust the location of the legend by specifying two arguments (x and y), these determine the horizontal and vertical location. We used a fast alternative by specifying the location with a shortcut legend("bottomleft",…). You can also specify "topright", "bottomright", "topleft". However, greater precision can be obtained by specifying the x and y parameters separately.

Quick reference R code

```
install.packages("maps")
install.packages("plotrix")
library(maps)
library(plotrix)

map("usa")
data(votes.repub)
result=c(1          ,1          ,1          ,1          ,0          ,0
,0          ,0          ,0          ,1          ,0          ,1          ,0
,1          ,0          ,1          ,1          ,1          ,0          ,0
,0          ,0          ,0          ,1          ,1          ,1          ,1
,0          ,0          ,0          ,0          ,0          ,1          ,1
,0          ,1          ,0          ,0          ,0          ,1          ,1
,1          ,1          ,1          ,0          ,0          ,0          ,1
,0         ,1)
temp<-cbind(votes.repub,result)
data <- temp[,c(32)]
data[data == 0] <- NA
state.to.map <- match.map("state", state.name, exact = F)
x <- data[state.to.map]
state.col<-color.scale(x, 1,0, 0,  color.spec="rgb")
par(bg="slategray2")
map("state",fill=TRUE,col=state.col, )
data[is.na(data)] <- 0.0
data[data == 1] <- NA
x <- data[state.to.map]
state.col<-color.scale(x, 0,0, 1,  color.spec="rgb")
map("state",fill=TRUE,col=state.col,add=TRUE, boundary = FALSE)
map("state",fill=F,add=TRUE, boundary = T, col="white")
title(main="2012 Presidential Election")
legend(x="bottomleft", inset=.05,
bty="n", cex=0.75, text.col ="darkred",
c("Obama (Democrat)", "Romney (Republican)"), fill= c("blue","red"))
```

NOTES

Write your notes below…

21. Population Cartogram

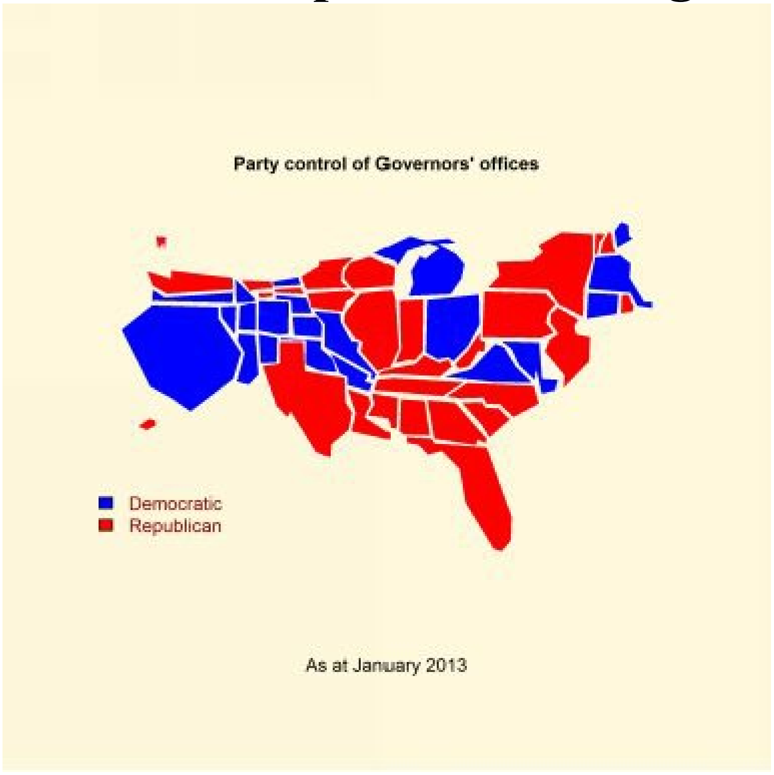


Figure 21 1 Governors' political affiliation adjusted for population size

How to read the image

The image displays US Governors political affiliation by state adjusted by population size. States held by Democratic Governors are blue and Republican red.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("maps")#used to draw the map
library(maps)
```

Step 2: Clean and prepare the data

```
map("usa")#load the US map
data(votes.repub) # this contains names of US States
# enter the election results note 1 equals a republican win
result=c(1          ,1          ,1          ,1          ,0          ,0
,0         ,0         ,0         ,1         ,0         ,1         ,0
,1         ,0         ,1         ,1         ,1         ,0         ,0
,0          ,0          ,0          ,1          ,1          ,1          ,1
,0          ,0          ,0          ,0          ,0          ,1          ,1
,0          ,1          ,0          ,0          ,0          ,1          ,1
,1          ,1          ,1          ,0          ,0          ,0          ,1
,0         ,1)
# we don't want to waste time typing in state names, so below is a super-fast way of getting what we want.
temp<-cbind(votes.repub,result)
# now drop all the other columns and just leave our election data
data <- temp[,c(32)]
data[data == 0] <- NA# replace zeros (Democrat) with NA's
# match state names and store in variable x
state.to.map <- match.map("state", state.name, exact = F)
x <- data[state.to.map]
```

Step 3: Plot the chart

```
# we begin with the Republican color - red
state.col<-color.scale(x, 1,0, 0,  color.spec="rgb")
par(bg="cornsilk")# set background color of chart
#draw the map with Republican states in red
map('state.carto',fill=TRUE,col=state.col, )
# replace Republican with Democrat for next map
data[is.na(data)] <- 0.0
data[data == 1] <- NA
x <- data[state.to.map]
# set color to blue
state.col<-color.scale(x, 0,0, 1,  color.spec="rgb")
#plot Democrat states in blue
map('state.carto',fill=TRUE,col=state.col,add=TRUE, boundary = FALSE)
map('state.carto',fill=F,add=TRUE, boundary = T, col="white")
#add chart title
title(main="       Party       control       of       Governors'       offices       ",
cex.main=1,col.main="black")
```

```
title(    sub="As at January 2013", cex.main=0.3)
#add legend
legend(x="bottomleft", inset=.05,
bty="n", cex=0.75, text.col ="darkred",
c("Democratic)", "Republican"), fill= c("blue","red"))
```

Visualization quick tips

Tip 1: For an exact match of state names set exact =TRUE in the match.map function.

Tip 2: Playing with colors is fun. The map used the argument color.scale with color.spec="rgb" to generate the red and blue colors of the political parties. Quick and easy alternatives are "hcl" and "hsv".

Tip 3: You adjust the location of the legend by specifying two arguments (x and y), these determine the horizontal and vertical location. We used a fast alternative by specifying the location with a shortcut legend("bottomleft",…). You can also specify "topright", "bottomright", "topleft". However, greater precision can be obtained by specifying the x and y parameters separately.

Quick reference R code

```
install.packages("maps")
install.packages("plotrix")
library(maps)
library(plotrix)
map("usa")
data(votes.repub)
result=c(1,          1,          1          ,0          ,0          ,0
,0          ,0          ,1          ,1          ,0          ,1          ,0
,1        ,1        ,1        ,0        ,1        ,1        ,0
,0        ,1        ,0        ,1        ,0        ,0        ,1
,1        ,0        ,1        ,1        ,0        ,1        ,1
,1        ,1        ,0        ,1        ,0        ,1        ,1
,1        ,1        ,1        ,0        ,1        ,0        ,0
,1        ,1)

temp<-cbind(votes.repub,result)
data <- temp[,c(32)]
data[data == 0] <- NA
state.to.map <- match.map("state", state.name, exact = F)
x <- data[state.to.map]
state.col<-color.scale(x, 1,0, 0,  color.spec="rgb")
par(bg=" cornsilk")
map('state.carto',fill=TRUE,col=state.col, )
data[is.na(data)] <- 0.0
data[data == 1] <- NA
x <- data[state.to.map]
state.col<-color.scale(x, 0,0, 1,  color.spec="rgb")
map('state.carto',fill=TRUE,col=state.col,add=TRUE, boundary = FALSE)
map('state.carto',fill=F,add=TRUE, boundary = T, col="white")
title(main="     Party     control     of     Governors'     offices     ",
cex.main=1,col.main="black")
title(    sub="As at January 2013", cex.main=0.3)
legend(x="bottomleft", inset=.05,
bty="n", cex=0.75, text.col ="darkred",
c("Democratic)", "Republican"), fill= c("blue","red"))
```
NOTES

Write your notes below…

22. Visibility map

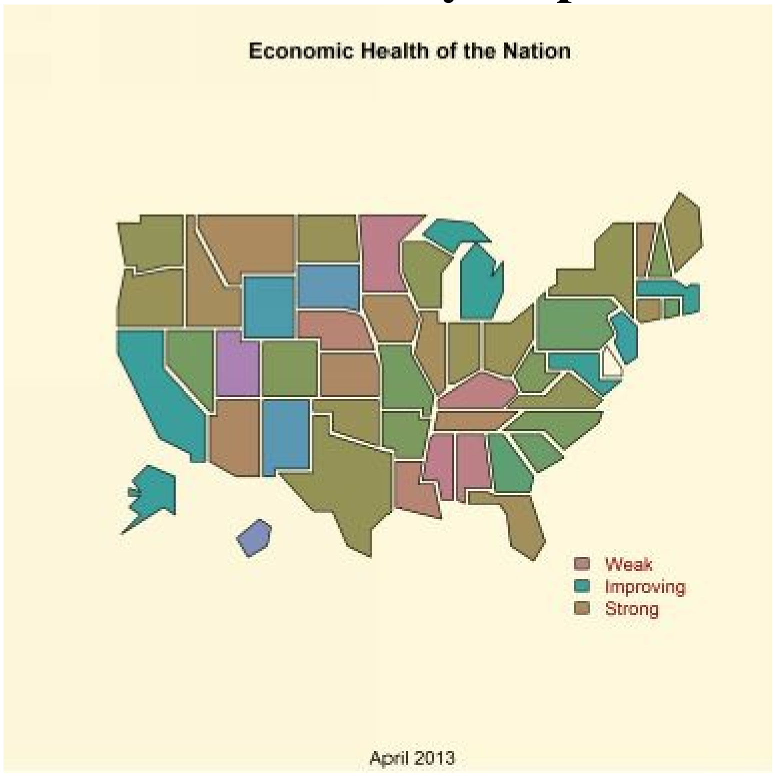


Figure 22 1 Governors' political affiliation adjusted for population size

How to read the image

The image displays economic conditions across the US based on the monthly coincident index for each of the 50 states published by the Federal Reserve Bank of Philadelphia. The image uses a visibility based map of the US which provides simplified state shapes with sufficient areas to allow annotations in even the smaller states.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("maps")#used to draw the map
install.packages("plotrix")#used to  color the map
library(maps)
library(plotrix)
```

Step 2: Clean and prepare the data

```
map("usa")
data(votes.repub)
```

```
# enter the data on economic conditions
conditions=c(129.23,            120.52,            181.44            ,141.03
,158.32            ,181.32            ,156.91            ,146.26
,146.20            ,166.73            ,105.82            ,199.92
,144.80            ,144.44            ,149.63            ,142.42
,141.03            ,130.47            ,135.53            ,149.65
,179.57            ,127.80            ,160.50            ,142.80
,135.51            ,160.73            ,158.23            ,182.76
,191.82            ,151.79            ,159.57            ,150.04
,162.73            ,193.45            ,140.50            ,151.11
,212.39            ,142.99            ,152.15            ,152.43
,158.02            ,153.37            ,188.18            ,199.37
,147.13            ,149.76            ,158.81            ,139.44
,163.91        ,163.22)

# we don't want to waste time typing in state names, so this is a neat trick
temp<-cbind(votes.repub,conditions)
# now drop all the other columns and just leave our tax data
data <- temp[,c(32)]
state.to.map <- match.map("state", state.name, exact = F)
```

Step 3: Plot the chart

```
#set colors to use on the image
state.col<-color.scale(x,c(0,300),40,60,  color.spec="hcl")
par(bg="cornsilk")# set background color
map("state.vbm",fill=TRUE,col=state.col)
#create title and legend
title(main="Economic Health of the Nation",
sub=" Federal Reserve Bank of Philadelphia coincident index – April 2013")
legend(x="bottomright", inset=.05,
bty="n", cex=0.75, text.col ="darkred",
c("Weak", "Improving", "Strong"), fill= state.col)
```

Visualization quick tips

Tip 1: For an exact match of state names set exact =TRUE in the match.map function.

Tip 2: The map used the argument color.scale with color.spec="hcl". Quick and easy alternatives are "rgb" and "hsv".

Tip 3: You adjust the location of the legend by specifying two arguments (x and y), these determine the horizontal location and the vertical location. We used a fast alternative by specifying the location with a shortcut legend("bottomleft", …). You can also specify "topright", "bottomright", "topleft". However, greater precision can be obtained by specifying the x and y parameters separately.

Quick reference R code

```
install.packages("maps")
install.packages("plotrix")
library(maps)
library(plotrix)
map("usa")
data(votes.repub)
conditions=c(129.23,        120.52,        181.44         ,141.03
,158.32          ,181.32          ,156.91          ,146.26
,146.20          ,166.73          ,105.82          ,199.92
,144.80          ,144.44          ,149.63          ,142.42
,141.03          ,130.47          ,135.53          ,149.65
,179.57          ,127.80          ,160.50          ,142.80
,135.51          ,160.73          ,158.23          ,182.76
,191.82          ,151.79          ,159.57          ,150.04
,162.73          ,193.45          ,140.50          ,151.11
,212.39          ,142.99          ,152.15          ,152.43
,158.02          ,153.37          ,188.18          ,199.37
,147.13          ,149.76          ,158.81          ,139.44
,163.91       ,163.22)
temp<-cbind(votes.repub,conditions)
```

```
# we don't want to waste time typing in state names, so this is a neat trick
# now drop all the other columns and just leave our tax data
data <- temp[,c(32)]
state.to.map <- match.map("state", state.name, exact = F)
x <- data[state.to.map]
state.col<-color.scale(x,c(0,300),40,60,  color.spec="hcl")
par(bg="cornsilk")
map("state.vbm",fill=TRUE,col=state.col)
title(main="Economic Health of the Nation",
   sub=" Federal Reserve Bank of Philadelphia coincident index – April 2013")
legend(x="bottomright", inset=.05,
bty="n", cex=0.75, text.col ="darkred",
c("Weak", "Improving", "Strong"), fill= state.col)
```

NOTES

Write your notes below…

23. Geographic map with shape observations

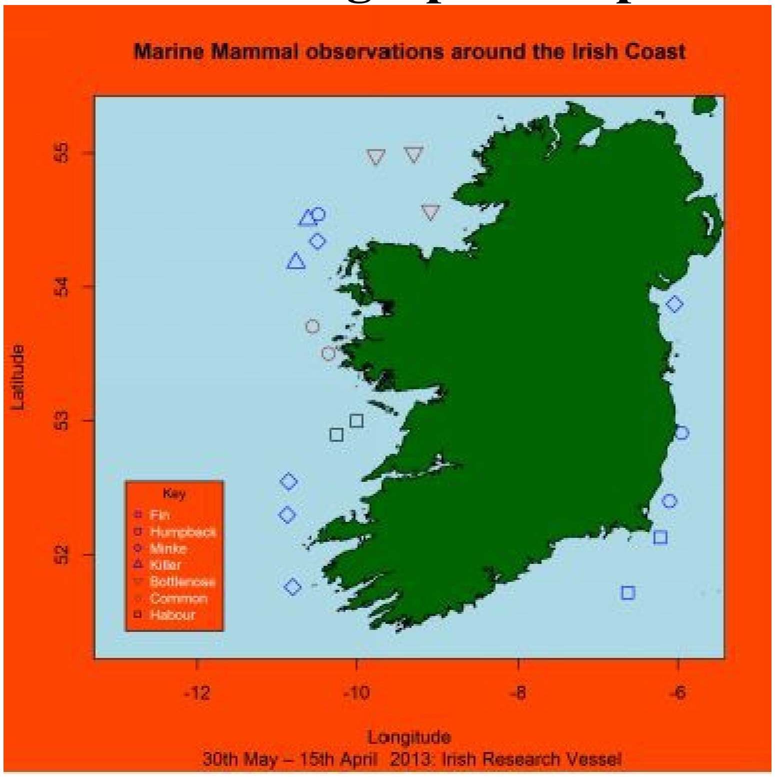

Marine Mammal observations around the Irish Coast
Latitude
55
54
53
52
Key
Fin
Humpback
Minke
Killer
Bottlenose
Common
Habour
-12
-10
-8
-6
Longitude
30th May – 15th April 2013: Irish Research Vessel

Figure 23 1 Latitude & longitude of Irish coast marine mammal observations

How to read the image

The image displays observations on three groups of marine mammal (whales, dolphins and porpoise) around the Irish coast spotted by a research vessel. Each observation is identified by a shape with each mammal group coded by color. The longitude and latitude are captured by the y and x axis.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("mapplots")#used to draw the map
require(mapplots)
```

Step 2: Clean and prepare the data

```
data(coast)# contains the map of the Irish coastline
species=c("Fin","Humpback","Minke","Killer","Bottlenose","Common","Habou
types of marine mammal to be monitored
# below we enter the longitude and latitude of each sighting
#data entered by mammal
```

```
#Fin whale data
fin.x=c(          -5.95,          -6.1,          -10.48)
fin.y=c(          52.91,          52.4,          54.54)
#Humpback whale data
hump.x=c(          -6.63,          -6.22)
hump.y=c(          51.72,          52.13)
#Minke whale data
minke.x=c(          -6.04,          -10.87,          -10.85,          -
10.49,          -10.8)
minke.y=c(          53.87,          52.3,          52.55,          54.34,
51.76)
#Killer whale data
killer.x=c(          -10.61          ,-10.76)
killer.y=c(          54.5,          54.18)

#bottlenose data
bottle.x=c(          -9.76          ,-9.31,-9.10)
bottle.y=c(          54.98,          55.00,54.57)
#common dolphin data
common.x=c(          -10.35,-10.55)
common.y=c(                    53.50,53.70)
#habour dolphin data
habour.x=c(          -10.00,-10.25)
habour.y=c(53.00,52.90)
```

Step 3: Plot the chart

```
par(bg="orangered")# set background color
#draw the region used for the plot to contain the Irish coast
basemap(xlim=c(-13,-5.7), ylim=c(51.5,55.15), main = "", axes = TRUE)
draw.shape(coast, col='darkgreen') #draw the Irish coast

# Fin whale plot
draw.bubble(fin.x, fin.y, z=0.05, maxradius=0.05, pch=21, col="blue")
```

```
# Humpback whale plot
draw.bubble(hump.x, hump.y, z=0.05, maxradius=0.05, pch=22, col="blue")
# Minke whale plot
draw.bubble(minke.x, minke.y, z=0.05, maxradius=0.05, pch=23, col="blue")
# Killer whale plot
draw.bubble(killer.x, killer.y, z=0.05, maxradius=0.05, pch=24, col="blue")
# bottlenose plot
draw.bubble(bottle.x, bottle.y, z=0.05, maxradius=0.05, pch=25, col="brown")
# common plot
draw.bubble(common.x, common.y, z=0.05, maxradius=0.05, pch=21,
col="brown")
# Harbour porpoise plot
draw.bubble(habour.x, habour.y, z=0.05, maxradius=0.05, pch=22,
col="black")

#set the colors for the legend, blue for whales, brown for dolphins
colors=c("blue","blue","blue","blue","brown","brown","black")
shapes=c(21,22,23,24,25,21,22) # shapes used to plot observations
#write legend in bottom left corner of image
legend(x="bottomleft", inset=.05, bty="0", cex=0.75, text.col ="white",
pch=shapes,col=colors,legend=species, title="Key",title.col="black")
#add title to image
title(main="Marine Mammal observations around the Irish Coast",
sub=" 30th May – 15th April  2013: Irish Research Vessel")
```

Visualization quick tips

Tip 1: The draw.bubble function was used to draw the observed marine mammal sighting on the image. The size and radius of each point can be changed by specifying alternative values for the arguments z and maxradius respectively.

Tip 2: To turn off the x and y axis use the argument axes = FALSE in the basemap function.

Tip 3: You adjust the location of the legend by specifying two arguments (x and y), these determine the horizontal location and the vertical location. We used a fast alternative by specifying the location with a shortcut legend("bottomleft", …). You can also specify "topright", "bottomright", "topleft". However, greater precision can be obtained by specifying the x and y parameters separately.

Quick reference R code

```
install.packages("mapplots")
require(mapplots)
data(coast)
species=c("Fin","Humpback","Minke","Killer","Bottlenose","Common","Habou

#Fin whale data
fin.x=c(        -5.95,        -6.1,        -10.48)
fin.y=c(        52.91,        52.4,        54.54)
#Humpback whale data and plot
hump.x=c(        -6.63,        -6.22)
hump.y=c(        51.72,        52.13)
#Minke whale data and plot
minke.x=c(        -6.04,        -10.87,        -10.85,        -
10.49,        -10.8)
minke.y=c(        53.87,        52.3,        52.55,        54.34,
51.76)
#Killer whale data and plot
killer.x=c(        -10.61        ,-10.76)
killer.y=c(        54.5,        54.18)

#bottlenose data and plot
bottle.x=c(        -9.76        ,-9.31,-9.10)
bottle.y=c(        54.98,        55.00,54.57)
```

```
#common dolphin data and plot
common.x=c(              -10.35,-10.55)
common.y=c(                            53.50,53.70)
#habour dolphin data and plot
habour.x=c(              -10.00,-10.25)
habour.y=c(53.00,52.90)

par(bg="orangered")
basemap(xlim=c(-13,-5.7), ylim=c(51.5,55.15), main = "", axes = TRUE)
draw.shape(coast, col='darkgreen')

# Fin whale plot
draw.bubble(fin.x, fin.y, z=0.05, maxradius=0.05, pch=21, col="blue")
# Humpback whale plot
draw.bubble(hump.x, hump.y, z=0.05, maxradius=0.05, pch=22, col="blue")
# Minke whale plot
draw.bubble(minke.x, minke.y, z=0.05, maxradius=0.05, pch=23, col="blue")
# Killer whale plot
draw.bubble(killer.x, killer.y, z=0.05, maxradius=0.05, pch=24, col="blue")
# bottlenose plot
draw.bubble(bottle.x, bottle.y, z=0.05, maxradius=0.05, pch=25, col="brown")
# common plot
draw.bubble(common.x, common.y, z=0.05, maxradius=0.05, pch=21,
col="brown")
# Harbour porpoise plot
draw.bubble(habour.x, habour.y, z=0.05, maxradius=0.05, pch=22,
col="black")


colors=c("blue","blue","blue","blue","brown","brown","black")#legend colors
shapes=c(21,22,23,24,25,21,22)
legend(x="bottomleft", inset=.05, bty="0", cex=0.75, text.col ="white",
pch=shapes,col=colors,legend=species,title="Key",title.col="black")
```

```
title(main="Marine Mammal observations around the Irish Coast",
sub=" 30th May – 15th April  2013: Irish Research Vessel")
```

NOTES

Write your notes below…

24. Two variable 3D scatter plot

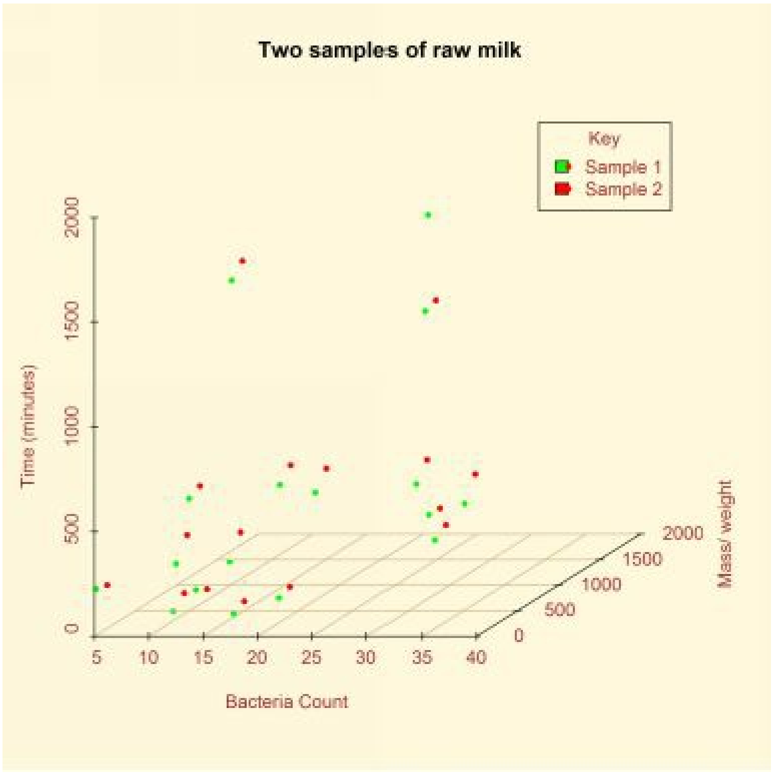


Figure 24 1 Bacteria count, weight over time of two samples of raw milk

How to read the image

The image displays the relationship between bacteria count, weight (mass) and time of two samples of raw milk. Sample one is colored green and sample two colored red.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("scatterplot3d")# this package generates the plot
require("scatterplot3d")
```

Step 2: Clean and prepare the data

```
# Time in minutes
time=c(25,37,49,61,73,85,97,109,121,133,145,157,169,181,193,205,217)
#Sample 1 bacteria count and weight reading
```

```
count.1=c(0,7,9,12,7,30,8,12,33,16,16,30,29,19,28,29,11)
mass.1=c(220,110,210,340,330,560,635,80,605,150,688,422,1511,642,681,196
#Sample 2 bacteria count and weight reading
count.2=c(6,13,15,18,13,36,14,18,39,22,22,36,35,25,34,35,17)
mass.2=c(239,197,213,481,466,591,695,140,746,203,782,494,1562,757,796,20
```

Step 3: Plot the chart

```
# set color of plotted points to green and background color to cornsilk
par(col="green",bg="cornsilk")
#plot the first sample points
scatterplot3d(ylim=c(0,1600),zlim=c(0,2000) , count.1, time, mass.1, angle=22,
pch=20, cex.symbols=1,xlab="", ylab="",
zlab="",grid=FALSE,axis=FALSE,cex.lab=1, cex.axis=1)
# set color of plotted points to red and overlay scatterplot over first sample
points
par(new=TRUE,col="red" )

scatterplot3d(main=" Two samples of raw milk",ylim=c(0,1600),
zlim=c(0,2000), count.2, time, mass.2, angle=22, col.axis="grey5"
,col.grid="tan",xlab=" Bacteria Count", ylab="Mass/ weight" ,zlab="Time
(minutes)", pch=20, cex.symbols=1, box=FALSE, col.lab="brown",cex.lab=1,
cex.axis=1)
```

Visualization quick tips

Tip 1: You can set the limit range of the x,y and z axis by using the arguments xlim, ylim and zlim respectively. So zlim=c(0,2000) sets the range of the z axis between 0 and 2000.

Tip 2: To set the size of the labels, axis and symbols (plotted points) use the argument cex.lab, cex.axis and cex.symbols respectively. Try experimenting with values greater than 1 for much larger labels, symbols or x, y and z axis, values less than 1 produce smaller labels, symbols and x,y,z axis.

Tip 3: Set the color of the grid and axes by using the arguments col.axis and col.grid. You can turn the axis and grid off by setting the arguments axis=FALSE and grid =FALSE.

Quick reference R code

```
install.packages("scatterplot3d")
require("scatterplot3d")
time=c(25,37,49,61,73,85,97,109,121,133,145,157,169,181,193,205,217)
count.1=c(0,7,9,12,7,30,8,12,33,16,16,30,29,19,28,29,11)
mass.1=c(220,110,210,340,330,560,635,80,605,150,688,422,1511,642,681,196
count.2=c(6,13,15,18,13,36,14,18,39,22,22,36,35,25,34,35,17)
mass.2=c(239,197,213,481,466,591,695,140,746,203,782,494,1562,757,796,20
par(col="green",bg="cornsilk")
scatterplot3d(ylim=c(0,1600),zlim=c(0,2000) , count.1, time, mass.1, angle=22,
pch=20, cex.symbols=1,xlab="",ylab="", zlab="",grid=F,axis=F,cex.lab=1,
cex.axis=1)
par(new=TRUE,col="red" )
scatterplot3d(main=" Two samples of raw
milk",ylim=c(0,1600),zlim=c(0,2000), count.2, time, mass.2, angle=22,
col.axis="grey5",col.grid="tan",xlab=" Bacteria Count",ylab="Mass/ weight",
zlab="Time (minutes)", pch=20, cex.symbols=1,box=FALSE,
col.lab="brown",cex.lab=1, cex.axis=0.3)
legend("topright", inset=.05, bty="o", cex=1,
title="Key", c("Sample 1", "Sample 2"), pch=20, fill=c("green",
"red"),text.col="brown")
```

NOTES

Write your notes below…

25. 3D scatter plot

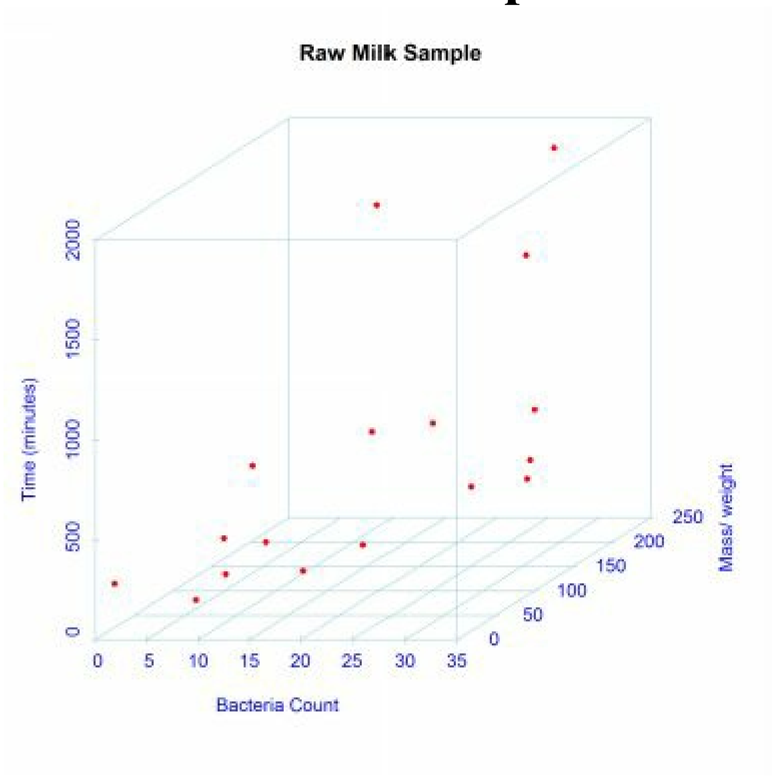


Figure 25 1 Bacteria count, weight over time of a raw milk sample

How to read the image

The image displays the relationship between bacteria count, weight (mass) and time from a sample of raw milk. Sample points are red colored dots.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("scatterplot3d")# this package generates the plot
require("scatterplot3d")
```

Step 2: Clean and prepare the data

```
# Time in minutes
time=c(25,37,49,61,73,85,97,109,121,133,145,157,169,181,193,205,217)
# bacteria count and weight reading
count=c(0,7,9,12,7,30,8,12,33,16,16,30,29,19,28,29,11)
mass=c(220,110,210,340,330,560,635,80,605,150,688,422,1511,642,681,1961,
```

Step 3: Plot the chart

```
# set color of plotted points and background color
par(col="red",bg="white")
# plot 3D scatter plot
scatterplot3d(main="Raw Milk Sample",count, time, mass, angle=22,
col.axis="lightblue",col.grid="lightblue",xlab=" Bacteria Count",ylab="Mass/
weight", zlab="Time (minutes)", pch=20, cex.symbols=1, box=TRUE,
col.lab="blue",cex.lab=1, cex.axis=1)
```

Visualization quick tips

Tip 1: You can set the limit range of the x,y and z axis by using the arguments xlim, ylim and zlim respectively. So zlim=c(0,2000) sets the range of the z axis between 0 and 2000.

Tip 2: To set the size of the labels, axis and symbols (plotted points) use the argument cex.lab, cex.axis and cex.symbols respectively. Try experimenting with values greater than 1 for much larger labels, symbols or x, y and z axis, values less than 1 produce smaller labels, symbols and x,y,z axis.

Tip 3: Set the color of the grid and axes by using the arguments col.axis and col.grid. You can turn the axis and grid off by setting the arguments axis=FALSE and grid =FALSE.

Quick reference R code

```
install.packages("scatterplot3d")
require("scatterplot3d")
time=c(25,37,49,61,73,85,97,109,121,133,145,157,169,181,193,205,217)
count=c(0,7,9,12,7,30,8,12,33,16,16,30,29,19,28,29,11)
mass=c(220,110,210,340,330,560,635,80,605,150,688,422,1511,642,681,1961,
par(col="red",bg="white")
scatterplot3d(main=" Raw Milk Sample",count, time, mass, angle=22,
col.axis="lightblue",col.grid="lightblue",xlab=" Bacteria Count",ylab="Mass/
weight",zlab="Time (minutes)", pch=20, cex.symbols=1,box=TRUE,
col.lab="blue",cex.lab=1, cex.axis=1)
```

NOTES

Write your notes below…

26. 3D cluster scatter plot

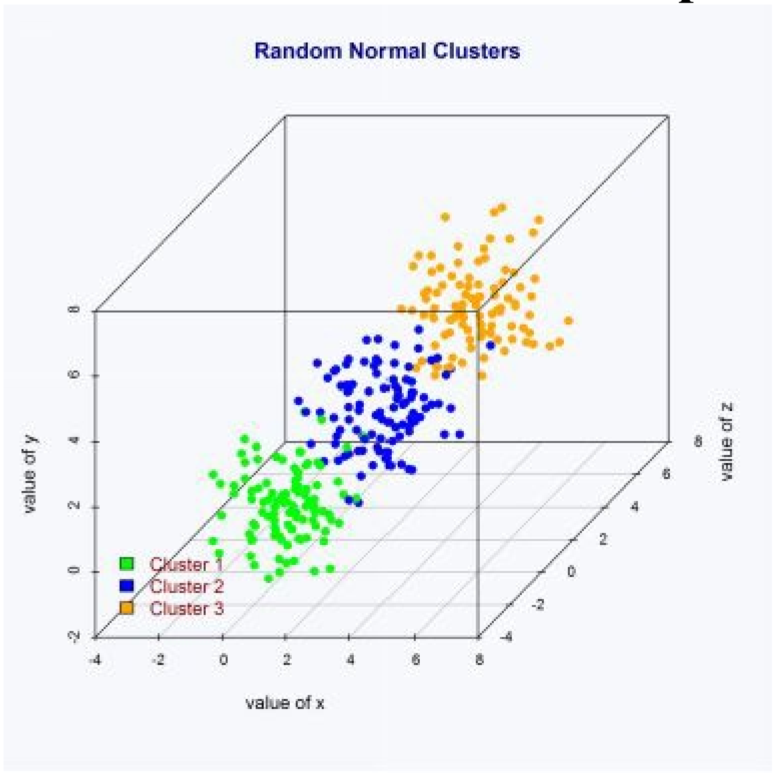


Figure 26 1 Clusters from three samples of normal random variables

How to read the image

The image displays the values from three distinct samples from the normal distribution. Each sample has a different mean, although all three samples have the same variance. Clusters are color coded with increasing mean as you move from the green cluster through the blue cluster, with the orange cluster having the highest mean value.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("scatterplot3d"))# this package generates the plot
require("scatterplot3d")
```

Step 2: Clean and prepare the data

```
set.seed(123)# allows image to be replicated
#simulate 300 observations for group1
sample1<-matrix(rnorm(300,0,1),100,3)
group1<- rep(1,nrow(sample1) )
sample1<-cbind(sample1,group1)
```

```
#simulate 300 observations for group2
sample2<-matrix(rnorm(300,2,1),100,3)
group2<- rep(2,nrow(sample2) )
sample2<-cbind(sample2,group2)
#simulate 300 observations for group3
sample3<-matrix(rnorm(300,4,1),100,3)
group3<- rep(3,nrow(sample3) )
sample3<-cbind(sample3,group3)
#combine the three samples together and add column names
data=rbind(sample1,sample2,sample3)
data<-data.frame(data)
colnames(data)=c("x","y","z","Group")
```

Step 3: Plot the chart

```
#define the colors to be plotted for each group
data$pcolor[data$Group==1] <- "green"
data$pcolor[data$Group ==2] <- "blue"
data$pcolor[data$Group ==3] <- "orange"
par(bg="ghostwhite")#set background color of image
#plot the image and add a legend
with(data, {
image<-scatterplot3d(x,y,z, type="p", pch=19,
color=pcolor,box=TRUE,xlab="value of x",zlab="value of y",ylab="value of
z",angle=45,main="Random Normal Clusters", col.main="darkblue",
col.grid="grey")
  legend("bottomleft", inset=.05,
     bty="n", cex=1, text.col ="darkred", title="", c("Cluster 1", "Cluster 2",
"Cluster 3"), fill=c("green", "blue", "orange"))
})
```

Visualization quick tips

Tip 1: The argument type determines the type of plot. Use type = "p" for points, type = "l" for lines and type = "h" for vertical lines.

Tip 2: Change the angle of the image by using the angle argument. Values in the range of 20 to 45 generally work well. Notice that angle=0 results in a 2-D plot.

Tip 3: Set the color of the grid and axes by using the arguments col.axis and col.grid. You can turn the axis and grid off by setting the arguments axis=FALSE and grid =FALSE.
Tip 4: use the $pcolor argument to set the color of each group of bars.

Quick reference R code

```
install.packages("scatterplot3d")
require(scatterplot3d)
set.seed(123)
sample1<-matrix(rnorm(300,0,1),100,3)
group1<- rep(1,nrow(sample1) )
sample1<-cbind(sample1,group1)

sample2<-matrix(rnorm(300,2,1),100,3)
group2<- rep(2,nrow(sample2) )
sample2<-cbind(sample2,group2)

sample3<-matrix(rnorm(300,4,1),100,3)
group3<- rep(3,nrow(sample3) )
sample3<-cbind(sample3,group3)

data=rbind(sample1,sample2,sample3)
data<-data.frame(data)
colnames(data)=c("x","y","z","Group")

data$pcolor[data$Group==1] <- "green"
data$pcolor[data$Group ==2] <- "blue"
data$pcolor[data$Group ==3] <- "orange"
```

```
par(bg="ghostwhite")
with(data, {
image<-scatterplot3d(x,y,z, type="p", pch=19,
color=pcolor,box=TRUE,xlab="value of x",zlab="value of y",ylab="value of
z",angle=45,main="Random Normal Clusters", col.main="darkblue",
col.grid="grey")
legend("bottomleft", inset=.05,
   bty="n", cex=1, text.col ="darkred",  title="",
   c("Cluster 1", "Cluster 2", "Cluster 3"), fill=c("green", "blue", "orange"))
})
```

NOTES

Write your notes below…

27. Vertical line with points 3D scatter plot

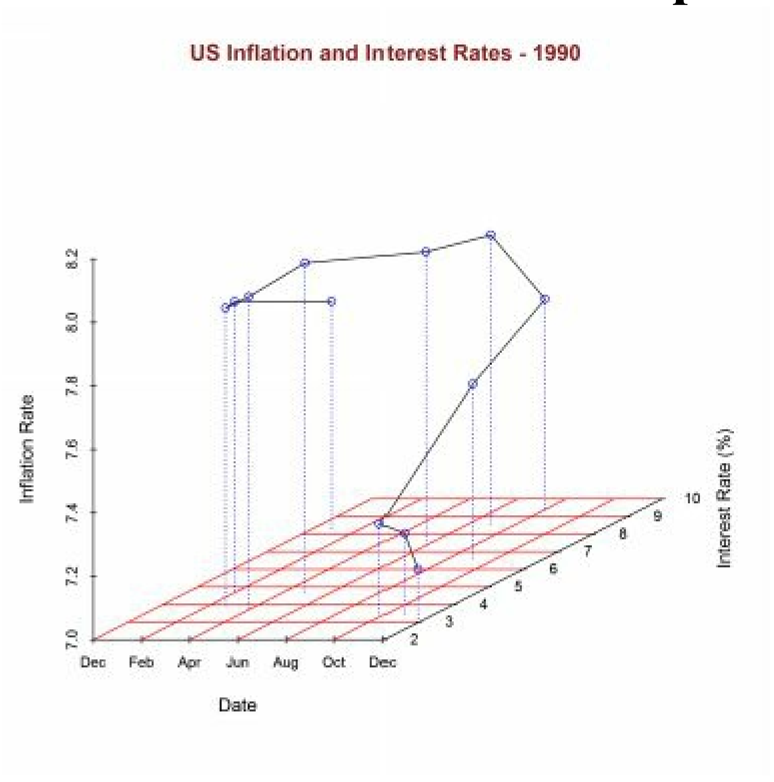


Figure 27 1 Path of US interest rates and inflation during 1990

How to read the image

The figure displays the path of inflation and interest rates in the US during 1990. Each sample point is represented by a vertical dashed line and an open circle. The solid purple line captures the trajectory over time of interest rates and inflation.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("scatterplot3d"))# this package generates the plot
install.packages("Ecdat"))# this package contains the data
require(Ecdat)
require("scatterplot3d")
```

Step 2: Clean and prepare the data

```
# use the Mishkin dataset contained in package Ecdat
data(Mishkin)
#create label for image
```

```
label=c("Dec","Feb","Apr","Jun","Aug","Oct")
# define axis variables
inflation=Mishkin[480:491,2]
rates=Mishkin[480:491,4]
time<-(1:12)# Time in months
```

Step 3: Plot the chart

```
# set color of plotted points to blue and background color to azure
par(bg="burlywood1",col="blue")
scatterplot3d(time,inflation,rates,type="l",box=F, tick.marks=T, label.tick.marks=T, x.ticklabs=label,angle=25,xlab="Date",ylab="Interest Rate (%)", zlab="Inflation Rate")

par(new=TRUE,col="blue" )

scatterplot3d(time,inflation,rates,type="h",box=F, tick.marks=T, label.tick.marks=T, x.ticklabs=label,angle=25,xlab="Date",ylab="Interest Rate (%)", zlab="Inflation Rate", lty.hplot="dotted",col.grid="red",main= "US Inflation and Interest Rates - 1990",col.main="firebrick4")
```

Visualization quick tips

Tip 1: The argument type determines the type of plot. Use type = "p" for points, type = "l" for lines and type = "h" for vertical lines.

Tip 2: The arguments tick.marks and label.tick.marks contain a logical value indicating whether tick marks should be drawn on the plot and whether tick marks should be labeled on the plot respectively.

Tip 3: The argument lty.hplot="dotted" determines the nature of the vertical line. In this case dotted. Other values the argument can take include "blank", "solid", "dashed", "dotdash", "longdash", or "twodash".

Tip 4: Set the color of the grid and axes by using the arguments col.axis and col.grid. You can turn the axis and grid off by setting the arguments axis=FALSE and grid =FALSE.

Quick reference R code

```
install.packages("scatterplot3d")
install.packages("Ecdat")
require(Ecdat)
require("scatterplot3d")
data(Mishkin )
label=c("Dec","Feb","Apr","Jun","Aug","Oct")
inflation=Mishkin[480:491,2]
rates=Mishkin[480:491,4]
time<-(1:12)
par(bg="burlywood1",col="blue")
scatterplot3d(time,inflation,rates,type="l",box=F,                    tick.marks=T,
label.tick.marks=T,  x.ticklabs=label,angle=25,xlab="Date",ylab="Interest  Rate
(%)", zlab="Inflation Rate")
par(new=TRUE,col="blue"  )
scatterplot3d(time,inflation,rates,type="h",box=F, tick.marks=T,
label.tick.marks=T, x.ticklabs=label,angle=25,xlab="Date",ylab="Interest Rate
(%)", zlab="Inflation Rate", lty.hplot="dotted",col.grid="red",main= "US
Inflation and Interest Rates - 1990",col.main="firebrick4")
```

NOTES

Write your notes below…

28. Vertical labeled 3D scatter plot

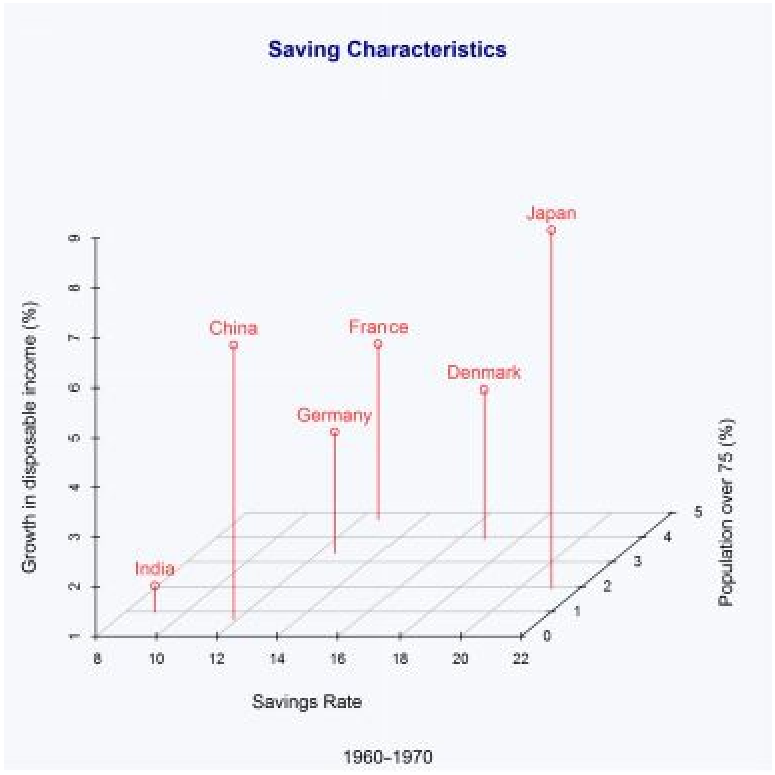


Figure 28 1 Savings characteristics of six nations

How to read the image

The image displays the savings rate, growth in disposable income and population over 75 for six nations averaged over the decade 1960–1970. Each nation is represented by a red vertical dashed line with an open circle.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

install.packages("scatterplot3d"))# this package generates the plot

require("scatterplot3d")

Step 2: Clean and prepare the data

require(datasets)# contains the data frame used

attach(LifeCycleSavings)# contains the data

# we select six nations from the LifeCycleSavings data frame

nations<-rbind(LifeCycleSavings[8,], LifeCycleSavings[11,], LifeCycleSavings[14,], LifeCycleSavings[15,], LifeCycleSavings[20,], LifeCycleSavings[23,])

# axis variables to be used in plot

```
x=nations[,1]
y=nations[,3]
z=nations[,5]
```

Step 3: Plot the chart

```
# set background color and color of plotted points
par(bg="ghostwhite",col="firebrick1")
# details of plot
image <-scatterplot3d(x,y,z,box=FALSE,type="h",angle=45, xlab="Savings Rate", ylab="Population over 75 (%)",zlab="Growth in disposable income (%)",main="Saving Characteristics",sub="1960-1970",col.main="darkblue", col.grid="grey",color="firebrick1")
# get image coordinates to enable plotting to country names
image.coords <- image$xyz.convert(x,y,z)
# add nation names to the vertical line
text(image.coords$x, image.coords$y, labels= row.names (nations) , pos=3, cex=1)
```

Visualization quick tips

Tip 1: The argument type determines the type of plot. Use type = "p" for points, type = "l" for lines and type = "h" for vertical lines.

Tip 2: Change the angle of the image by using the angle argument. Values in the range of 20 to 45 generally work well. Notice that angle=0 results in a 2-D plot.

Tip 3: Add the argument lty.hplot="dotted" to change the nature of the vertical line. Argument can take the form "blank", "solid", "dashed", "dotted", "dotdash", "longdash", or "twodash".

Tip 4: Set the color of the grid and axes by using the arguments col.axis and col.grid. You can turn the axis and grid off by setting the arguments axis=FALSE and grid =FALSE.

Quick reference R code

```
install.packages("scatterplot3d")
require(scatterplot3d)
require(datasets)
attach(LifeCycleSavings)
nations<-rbind( LifeCycleSavings[8,], LifeCycleSavings[11,],
LifeCycleSavings[14,], LifeCycleSavings[15,], LifeCycleSavings[20,],
LifeCycleSavings[23,])
x=nations[,1]
y=nations[,3]
z=nations[,5]
par(bg="ghostwhite",col="firebrick1")
image <-scatterplot3d(x,y,z,box=FALSE,type="h",angle=45, xlab="Savings
Rate",ylab="Population over 75 (%)",zlab="Growth in disposable income
(%)",main="Saving Characteristics",sub="1960-1970",col.main="darkblue",
col.grid="grey",color="firebrick1")
image.coords <- image$xyz.convert(x,y,z)
text(image.coords$x, image.coords$y, labels=row.names(nations),
pos=3, cex=1)
```

NOTES

Write your notes below…

29. 3D grid color bar plot

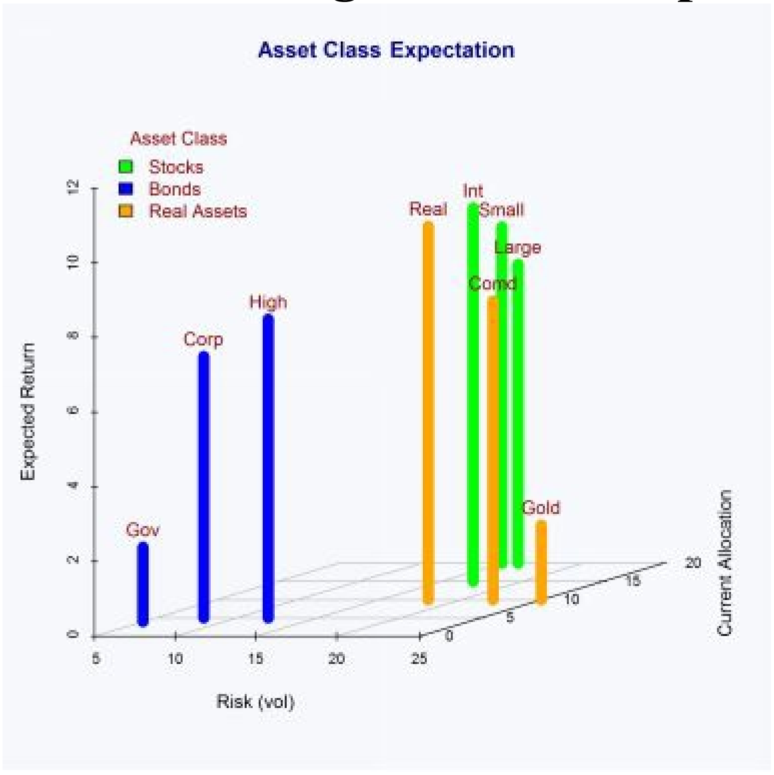


Figure 29 1 Asset class return and risk expectations

How to read the image

The image displays the expected asset class returns and risk for stocks (large cap, small cap, international), bonds (government, corporate, high yield) and real assets (real estate, commodities, gold) alongside the current portfolio asset allocation. Each asset class is color coded, stocks (green), bonds(blue) and real assets (orange). The name of each asset appear at the top of the colored bar.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

install.packages("scatterplot3d"))# this package generates the plot

require("scatterplot3d")

Step 2: Clean and prepare the data

# create a variable for each asset. It contains expected return, expected risk, current portfolio asset allocation and asset group (1 for stocks, 2 for bonds and 3 for real assets)

```
large.cap=c(8,          16,          20,1)
small.cap=c(9,          15,          20,1)
```

```
international=c(10,        17,        15,1)
Corp.bonds=c(        7,        8,        5,2)
Gov.bonds=c(2,        5,        4,2)
High.yield=c(        8,        12,        5,2)
Real.estate=c(        10,        18,        10,3)
Commodities=c(        8,        22,        10,3)
Gold=c(        2,        25,        10,3)
data=rbind(large.cap, small.cap, international, Corp.bonds, Gov.bonds,
High.yield, Real.estate, Commodities, Gold)
```

Add row names and create a data frame then add column names

```
rownames(data)=c("Large","Small","Int","Corp","Gov","High","Real","Comd",'
data<-data.frame(data)
colnames(data)=c("ER","Vol","%","Group")

data<-data.frame(data)
# data used for axis in plot
x<-data[,2]
y<-data[,3]
z<-data[,1]
```

Step 3: Plot the chart

```
#define colors for each group to be plotted on image
data$pcolor[data$Group==1] <- "green"
data$pcolor[data$Group ==2] <- "blue"
data$pcolor[data$Group ==3] <- "orange"
#set background color
par(bg="ghostwhite")
# Create plot
with(data, {
image<-scatterplot3d(x,y,z, type="h", lwd=10, pch=" ", color=pcolor,zlim =
c(0, 12),box=FALSE,xlab="Risk (vol)",zlab="Expected Return",ylab="Current
Allocation",angle=22,main="Asset Class Expectation", col.main="darkblue",
col.grid="grey")
#overlay text names of each asset
image.coords <- image$xyz.convert(x,y,z)
    text(image.coords$x, image.coords$y,
        labels=row.names(data),
        pos=3, cex=1, col="darkred")
#add legend
legend("topleft", inset=.05,
   bty="n", cex=1, text.col ="darkred",
title="Asset Class",
   c("Stocks", "Bonds", "Real Assets"), fill=c("green", "blue", "orange"))
})
```

Visualization quick tips

Tip 1: The argument type determines the type of plot. Use type = "p" for points, type = "l" for lines and type = "h" for vertical lines.

Tip 2: Change the angle of the image by using the angle argument. Values in the range of 20 to 45 generally work well. Notice that angle=0 results in a 2-D plot.

Tip 3: Add the argument lty.hplot="dotted" to change the nature of the vertical line. Argument can take the form "blank", "solid", "dashed", "dotted", "dotdash", "longdash", or "twodash".

Tip 4: Set the color of the grid and axes by using the arguments col.axis and col.grid. You can turn the axis and grid off by setting the arguments axis=FALSE and grid =FALSE.
Tip 5: use the $pcolor argument to set the color of each group of bars.

Quick reference R code

```
install.packages("scatterplot3d")
require(scatterplot3d)

large.cap=c(8,          16,          20,1)
```

```
small.cap=c(9,          15,          20,1)
international=c(10,          17,          15,1)
Corp.bonds=c(          7,          8,          5,2)
Gov.bonds=c(2,          5,          4,2)
High.yield=c(          8,          12,          5,2)
Real.estate=c(          10,          18,          10,3)
Commodities=c(          8,          22,          10,3)
Gold=c(          2,          25,          10,3)

data=rbind(large.cap, small.cap, international, Corp.bonds, Gov.bonds,
High.yield, Real.estate, Commodities, Gold)
rownames(data)=c("Large","Small","Int","Corp","Gov","High","Real","Comd",'
data<-data.frame(data)
colnames(data)=c("ER","Vol","%","Group")

data<-data.frame(data)
x<-data[,2]
y<-data[,3]
z<-data[,1]
data$pcolor[data$Group==1] <- "green"
data$pcolor[data$Group ==2] <- "blue"
data$pcolor[data$Group ==3] <- "orange"

par(bg="ghostwhite")

with(data, {
image<-scatterplot3d(x,y,z, type="h", lwd=10, pch=" ", color=pcolor,zlim =
c(0, 12),box=FALSE,xlab="Risk (vol)",zlab="Expected Return",ylab="Current
Allocation",angle=22,main="Asset Class Expectation", col.main="darkblue",
col.grid="grey")

image.coords <- image$xyz.convert(x,y,z)
   text(image.coords$x, image.coords$y,
          labels=row.names(data),
          pos=3, cex=1, col="darkred")

legend("topleft", inset=.05,
    bty="n", cex=1, text.col ="darkred",

    title="Asset Class",
    c("Stocks", "Bonds", "Real Assets"), fill=c("green", "blue", "orange"))
})
```

NOTES

Write your notes below…

30. Circle plot

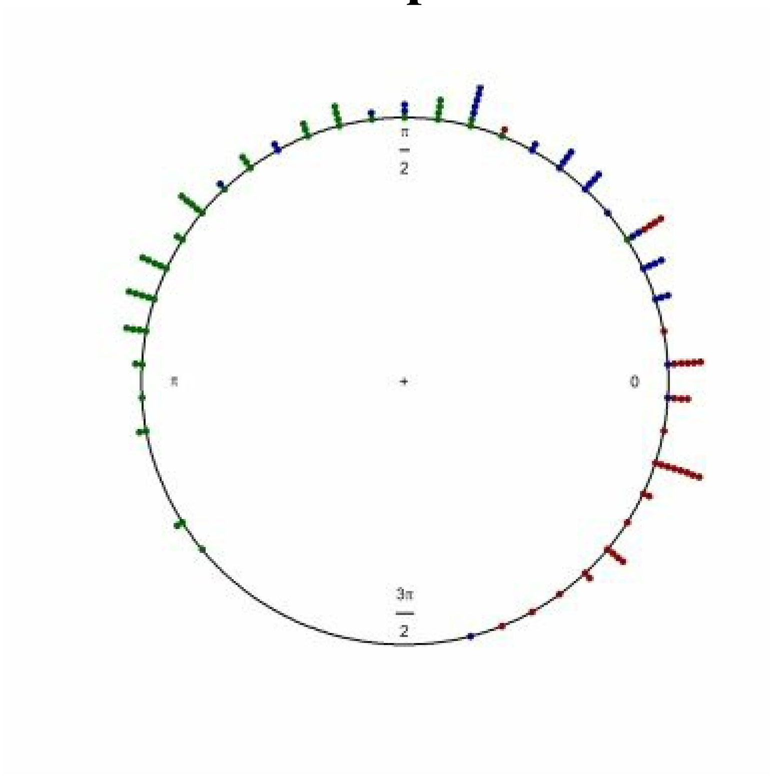


Figure 30 1 Simulated flight direction of three individual files

How to read the chart

The chart displays the simulated flight directions of three flies from a point of rest. The colored dots represent the flight orientation of each fly (red for fly 1, blue for fly 2 and green for fly 3).

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("circular")
require(circular)
# this package generates the plot
require(circular)
```

Step 2: Clean and prepare the data

```
set.seed(1234)# set so results can be replicated
#the code below simulates flight orientation from von mises distribution. We observe 50 flight paths per fly.
fly.1 <- rvonmises(n=50, mu=circular(0), kappa=3)
fly.2 <- rvonmises(n=50, mu=circular(pi/3), kappa=3)
fly.3 <- rvonmises(n=50, mu=circular(pi/1.5), kappa=3)
```

Step 3: Plot the chart

```
res <- plot(fly.1, stack=TRUE, col="darkred") # plot fly 1

#below adds data to chart for fly 2 and fly 3

points(fly.2, plot.info=res, col="darkblue", stack=TRUE)

points(fly.3, plot.info=res, col="darkgreen", stack=TRUE)
```

Visualization quick tips

Tip 1: Data should be in a circular format. This means your data needs to be converted to circular class. To do this try the statement circular (your_data) to coerce data into a circular format.

Tip 2: You can change the color of the points by changing the col argument.

Tip 3: Set stack=FALSE to plot the points around the outer edge of the circle.

Tip 4: To remove the numbers around the edge of the circle add the argument axes=FALSE to the plot function. To add tick marks to the edge of the circle add the argument ticks =TRUE to the plot function.

Quick reference R code

```
install.packages("circular")
require(circular)
data<-circular(wind , zero = pi/2)
par(bg="lightyellow")
rose.diag(data,                    bins=18,axes=TRUE,ticks=TRUE,border="black",
col="darkred",tol=0,prop=1.5)
points(data,stack=TRUE,col="darkgreen",pch=20)
symbols(0, 0, circle = 0.2, inches = FALSE, add = TRUE, fg = 'black')
symbols(0, 0, circle = 0.4, inches = FALSE, add = TRUE, fg = 'black')
symbols(0, 0, circle = 0.6, inches = FALSE, add = TRUE, fg = 'black')
symbols(0, 0, circle = 0.8, inches = FALSE, add = TRUE, fg = 'black')
```

NOTES

Write your notes below…

31. Circular arrows split plot

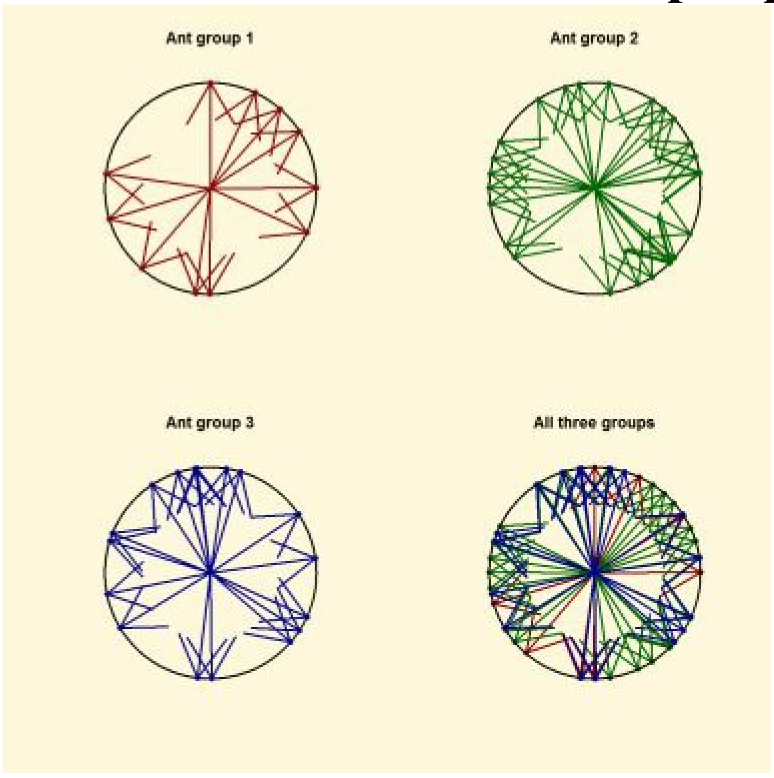


Figure 31 1 Direction of three groups of long legged desert ants

How to read the chart

A arrows chart for circular data shows the orientation of each individual data point by use of an arrow. The chart displays the orientation of three sets of long-legged desert ants after one eye on each ant was 'trained' to learn the ant's home direction, then covered and the other eye uncovered. The charts are orientated so that zero degrees appears at the top.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("circular")
require(circular)
# this package contains the data and generates the plot
require(circular)
```

Step 2: Clean and prepare the data

```
ant.1<-circular(fisherB10$set1,zero=pi/2)
ant.2<-circular(fisherB10$set2,zero=pi/2)
ant.3<-circular(fisherB10$set3,zero=pi/2)
# The data frame fisherB10 contains observations on each of the ant groups. It is converted to an object of class circular and orientated so that zero equals pi/2. This will ensure zero appears at the top of the resultant arrow plots.
```

Step 3: Plot the chart

```
par(bg="cornsilk")# set the background color to cornsilk
# create a 2 by 2 area, each of the four charts is plotted on one area
par(mfrow=c(2,2))
#plot the first chart with arrows
plot(ant.1,col="darkred",main="Ant group 1",axes=FALSE)
arrows.circular (ant.1,col="darkred")
#plot the second chart with arrows
plot(ant.2,col="darkgreen",main="Ant group 2",axes=FALSE)
arrows.circular (ant.2,col="darkgreen")
#plot the third chart with arrows
plot(ant.3,col="darkblue",main="Ant group 3",axes=FALSE)
arrows.circular (ant.3,col="darkblue")
#plot all the data with arrows
plot(ant.1,col="darkred",main="All three groups",axes=FALSE)
arrows.circular (ant.1,col="darkred")
par(new=TRUE) # allows you plot on an existing image
plot(ant.2,col="darkgreen", axes=FALSE)
arrows.circular (ant.2,col="darkgreen")
par(new=TRUE) # allows you plot on an existing image
plot(ant.3,col="darkblue",axes=FALSE)
arrows.circular (ant.3,col="darkblue")
```

Visualization quick tips

Tip 1: The arrows.circular function requires data in a circular format. This means the data needs to be converted to circular class. When using your own data or other datasets you can use the statement circular (your_data) to coerce data into a circular format.

Tip 2: You can include axes on each chart by setting the argument axes=TRUE in the plot function.

Tip 3: Change the background color by setting the argument bg="your chosen color".

Tip 4: To add tick marks around the edge of the circle add the argument ticks =TRUE to the plot function.

Quick reference R code

```
install.packages("circular")
require(circular)
ant.1<-circular(fisherB10$set1,zero=pi/2)
ant.2<-circular(fisherB10$set2,zero=pi/2)
ant.3<-circular(fisherB10$set3,zero=pi/2)
par(bg="cornsilk")
par(mfrow=c(2,2))
plot(ant.1,col="darkred",main="Ant group 1",axes=FALSE)
arrows.circular (ant.1,col="darkred")
plot(ant.2,col="darkgreen",main="Ant group 2",axes=FALSE)
arrows.circular (ant.2,col="darkgreen")
plot(ant.3,col="darkblue",main="Ant group 3",axes=FALSE)
arrows.circular (ant.3,col="darkblue")
plot(ant.1,col="darkred",main="All three groups",axes=FALSE)
arrows.circular (ant.1,col="darkred")
par(new=TRUE)
plot(ant.2,col="darkgreen", axes=FALSE)
arrows.circular (ant.2,col="darkgreen")
par(new=TRUE)
plot(ant.3,col="darkblue",axes=FALSE)
arrows.circular (ant.3,col="darkblue")
```

NOTES

Write your notes below…

32. Rose diagram

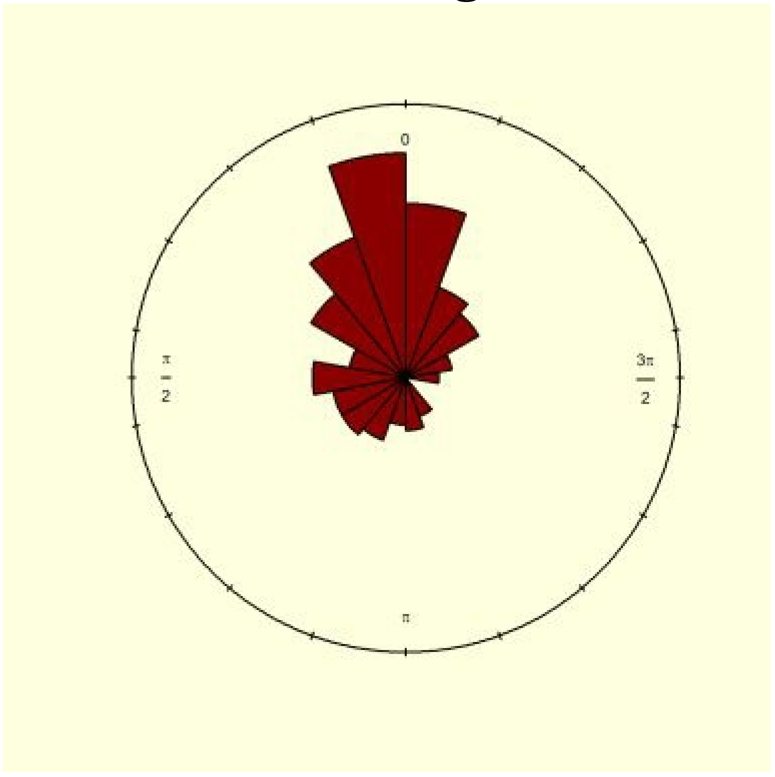


Figure 32 1 Wind direction measurements at Col d al Rosa

How to read the chart

A rose diagram is essentially a histogram for circular data. The chart displays wind direction measurement taken at 15 minute intervals at Col de la Roa in the Italian Alps from January 29, 2001 to March 31, 2001. The dark red bars in the center of the diagram are similar to the bars in a histogram. The image is orientated so that zero degrees appears at the top of the image.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("circular")
require(circular)
# this package contains the data and generates the plot
require(circular)
```

Step 2: Clean and prepare the data

```
data<-circular(wind , zero = pi/2)
# The data frame wind contains the observations. It is converted to an object of
```

class circular and orientated so that zero equals pi/2. This will ensure zero appears at the top of the resultant plot.

Step 3: Plot the chart

```
par(bg="lightyellow")# set the background color to light yellow

rose.diag(data,                    bins=18,axes=TRUE,ticks=TRUE,border="black",
col="darkred",tol=0,prop=1.5)
```

Visualization quick tips

Tip 1: The rose.diag function requires data in a circular format. This means the data needs to be converted to circular class. When using your own data or other datasets you can use the statement circular (your_data) to coerce data into a circular format.

Tip 2: You can change the number of bins by using the arguments bins in rose.diag. The argument takes positive values.

Tip 3: Change the argument col in rose.diag to choose different bin colors.

Tip 4: To remove the numbers around the edge of the circle use axes=FALSE. To remove the tick marks from the edge of the circle use ticks =FALSE.

Quick reference R code

```
install.packages("circular")
require(circular)
data<-circular(wind , zero = pi/2)
par(bg="lightyellow")
rose.diag(data,                    bins=18,axes=TRUE,ticks=TRUE,border="black",
col="darkred",tol=0,prop=1.5)
points(data,stack=TRUE,col="darkgreen",pch=20)
symbols(0, 0, circle = 0.2, inches = FALSE, add = TRUE, fg = 'black')
symbols(0, 0, circle = 0.4, inches = FALSE, add = TRUE, fg = 'black')
symbols(0, 0, circle = 0.6, inches = FALSE, add = TRUE, fg = 'black')
symbols(0, 0, circle = 0.8, inches = FALSE, add = TRUE, fg = 'black')
```

NOTES

Write your notes below…

33. Stacked rose diagram

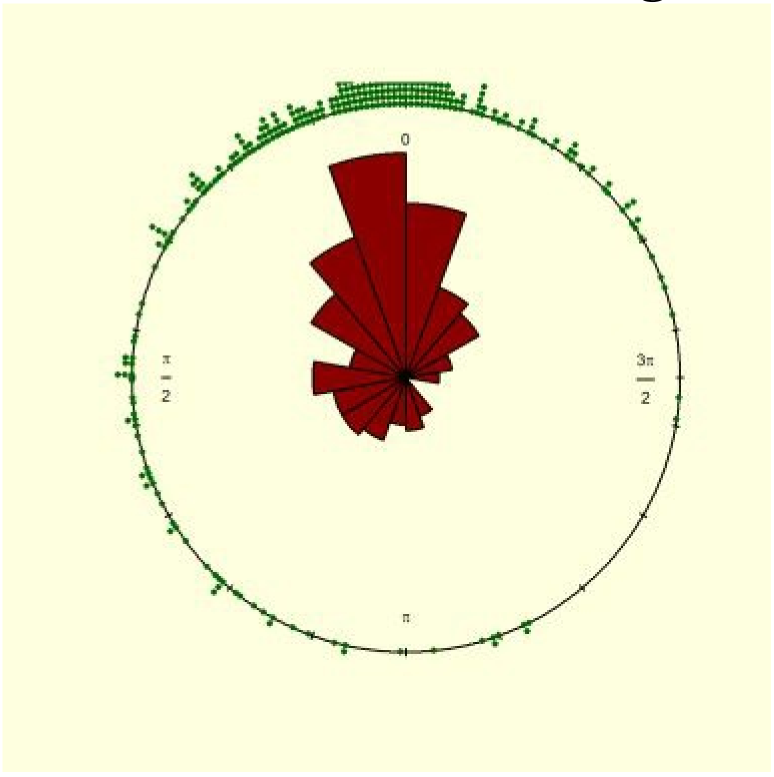


Figure 33 1 Stacked wind direction measurements at Col d al Rosa

How to read the chart

A stacked rose diagram is essentially a histogram for circular data with observations plotted at the edge. The chart displays wind direction measurement taken at 15 minute intervals at Col de la Roa in the Italian Alps from January 29, 2001 to March 31, 2001. The dark red bars in the center of the diagram are similar to the bars in a histogram. The green dots around the edge of the outer circle are the stacked individual observations. The diagram is orientated so that zero degrees appears at the top of the image.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("circular")
require(circular)
# this package contains the data and generates the plot
require(circular)
```

Step 2: Clean and prepare the data

data<-circular(wind , zero = pi/2)
# The data frame wind contains the observations. It is converted to an object of class circular and orientated so that zero equals pi/2. This will ensure zero appears at the top of the resultant plot.

Step 3: Plot the chart

par(bg="lightyellow")# set the background color to light yellow
rose.diag(data, bins=18,axes=TRUE,ticks=TRUE,border="black", col="darkred",tol=0,prop=1.5) # plot the rose diagram
points(data,stack=TRUE,col="darkgreen",pch=20)# add individual observations

Visualization quick tips

Tip 1: The rose.diag function requires data in a circular format. This means the data needs to be converted to circular class. When using your own data or other datasets you can use the statement circular (your_data) to coerce data into a circular format.

Tip 2: You can change the number of bins by using the arguments bins in rose.diag. The argument takes positive values.

Tip 3: Change the argument col in rose.diag to choose different bin colors.

Tip 4: In the points function set Stack =FALSE if you want individual observations plotted around the edge of the outer circle.

Quick reference R code

```
install.packages("circular")
require(circular)
data<-circular(wind , zero = pi/2)
```

```
par(bg="lightyellow")
rose.diag(data,                          bins=18,axes=TRUE,ticks=TRUE,border="black",
col="darkred",tol=0,prop=1.5)
points(data,stack=TRUE,col="darkgreen",pch=20)
```

34. Radial rose diagram

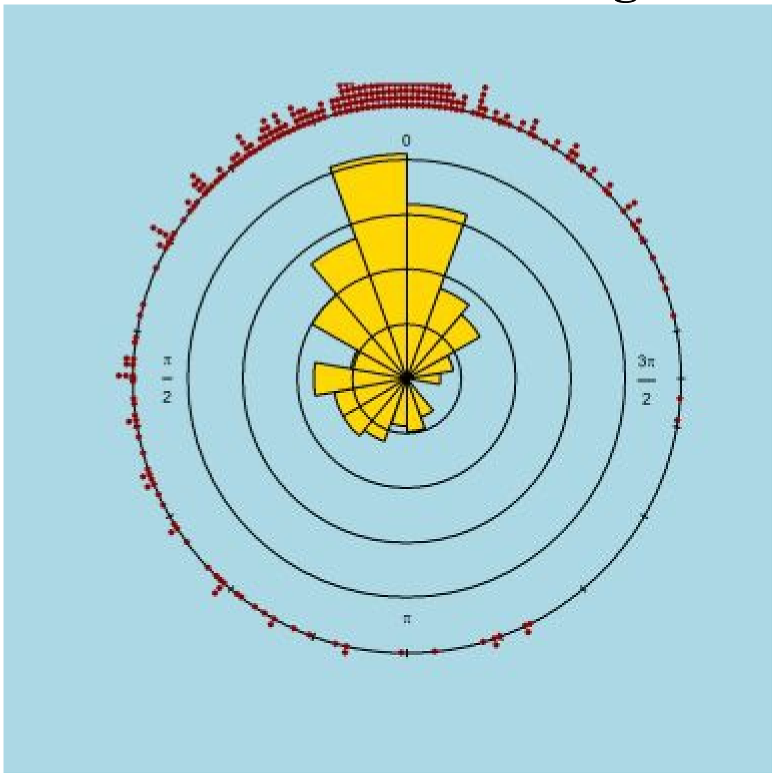


Figure 34 1 Wind direction measurements at Col d al Rosa

How to read the chart

A radial rose diagram is essentially a histogram for circular data with radial lines indicating the frequency of observations. The chart displays wind direction measurement taken at 15 minute intervals at Col de la Roa in the Italian Alps from January 29, 2001 to March 31, 2001. The gold petals in the center of the diagram are similar to the bars in a histogram. The green dots around the edge of the outer circle are the stacked individual observations. The diagram is orientated so that zero degrees appears at the top of the image.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("circular")
require(circular)
# this package contains the data and generates the plot
require(circular)
```

Step 2: Clean and prepare the data

```
data<-circular(wind , zero = pi/2)
```

# The data frame wind contains the observations. It is converted to an object of class circular and orientated so that zero equals pi/2. This will ensure zero appears at the top of the resultant plot.

Step 3: Plot the chart

```
par(bg="lightblue") # set background color to light blue
rose.diag(data,                    bins=18,axes=TRUE,ticks=TRUE,border="black",
col="gold",tol=0,prop=1.5) # function to plot chart
# line below adds individual observations
points(data,stack=TRUE,col="darkred",pch=20)
# lines below add radials
symbols(0, 0, circle = 0.2, inches = FALSE, add = TRUE, fg = 'black')
symbols(0, 0, circle = 0.4, inches = FALSE, add = TRUE, fg = 'black')
symbols(0, 0, circle = 0.6, inches = FALSE, add = TRUE, fg = 'black')
symbols(0, 0, circle = 0.8, inches = FALSE, add = TRUE, fg = 'black')
```

Visualization quick tips

Tip 1: The rose.diag function requires data in a circular format. This means the data needs to be converted to circular class. When using your own data or other datasets you can use the statement circular (your_data) to coerce data into a circular format.

Tip 2: You can change the number of bins by using the arguments bins in rose.diag. The argument takes positive values.

Tip 3: Change the argument col in rose.diag to choose different bin colors. In the points function set Stack =FALSE if you want individual observations plotted around the edge of the outer circle.

Tip 4: You can change the color of the radial lines by setting the argument fg='your chosen color' in the symbols function.

Quick reference R code

```
install.packages("circular") # this package contains the data and generates the
plot
require(circular)
data<-circular(wind , zero = pi/2)

par(bg="lightblue")
rose.diag(data,
bins=18,axes=TRUE,ticks=TRUE,border="black",col="gold",tol=0,prop=1.5)
#
points(data,stack=TRUE,col="darkred",pch=20)
symbols(0, 0, circle = 0.2, inches = FALSE, add = TRUE, fg = 'black')
symbols(0, 0, circle = 0.4, inches = FALSE, add = TRUE, fg = 'black')
symbols(0, 0, circle = 0.6, inches = FALSE, add = TRUE, fg = 'black')
symbols(0, 0, circle = 0.8, inches = FALSE, add = TRUE, fg = 'black')
```

NOTES

Write your notes below…

35. Stacked multivariable rose diagram

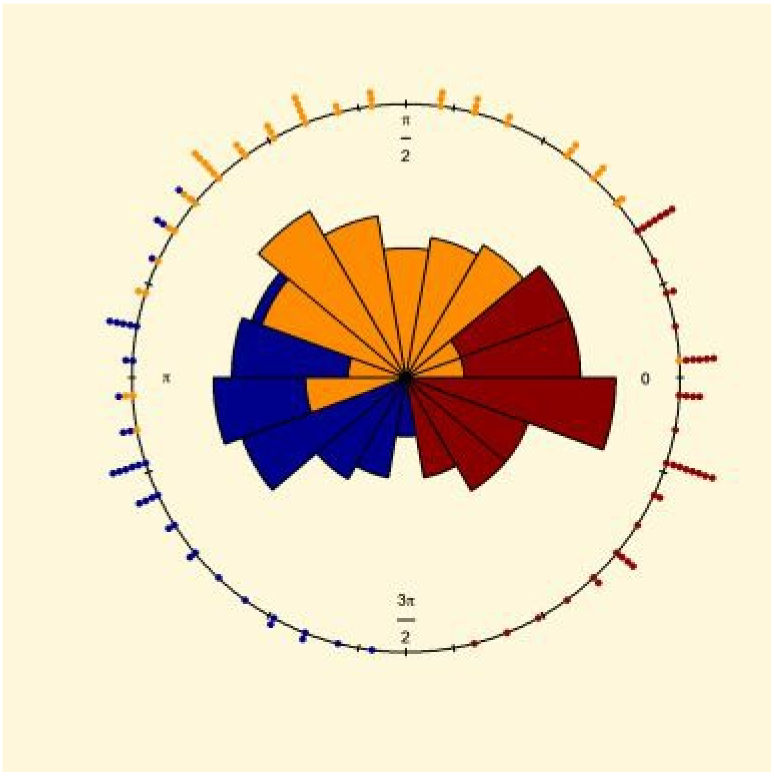


Figure 35 1 Simulated forage orientation of three beavers

How to read the chart

A multivariable rose diagram is essentially a grouped histogram for circular data. The chart displays simulated forage direction of three beavers. The dark red petals in the center of the diagram are similar to the bars in a histogram and represent beaver 1. Beaver 2 is represented by the blue petals, and beaver 3 by the orange petals. The dots around the edge of the outer circle are the color coded stacked observations for each beaver.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

install.packages("circular")
require(circular)
# this package contains the data and generates the plot
require(circular)

Step 2: Clean and prepare the data

set.seed(1234)# allows results to be replicated

```
#below the simulated orientation of the beavers is simulated using a von mises
distribution.
beaver.1 <- rvonmises(n=50, mu=circular(0), kappa=3)
beaver.2 <- rvonmises(n=50, mu=circular(pi), kappa=3)
beaver.3 <- rvonmises(n=50, mu=circular(pi/2), kappa=3)
```

Step 3: Plot the chart

```
par(bg="cornsilk")# set background color
# print rose diagram for beaver 1
rose.diag(beaver.1, bins=18,axes=TRUE,ticks=TRUE,
border="black",col="darkred",tol=0,prop=1.5)
par(new=TRUE) # allow overlay of next image
# print rose diagram for beaver 2
rose.diag(beaver.2, bins=18,axes=TRUE,ticks=TRUE
,border="black",col="darkblue",tol=0,prop=1.5) #
par(new=T) # allow overlay of next image

# print rose diagram for beaver 3
rose.diag(beaver.3, bins=18,axes=TRUE,ticks=TRUE,
border="black",col="darkorange",tol=0,prop=1.5) #
#plot stacked observations for each of the beavers
points(beaver.1, col="darkred", stack=TRUE)
points(beaver.2, col="darkblue", stack=TRUE)
points(beaver.3, col="darkorange", stack=TRUE)
```

Visualization quick tips

Tip 1: The rose.diag function requires data in a circular format. This means the data needs to be converted to circular class. When using your own data or other datasets you can use the statement circular (your_data) to coerce data into a circular format.

Tip 2: You can change the number of bins by using the arguments bins in rose.diag. The argument takes positive values.

Tip 3: Change the argument col in rose.diag to choose different bin colors. In the points function set Stack =FALSE if you want individual observations plotted around the edge of the outer circle.

Tip 4: To create a regular multivariable rose diagram do not enter the last three lines of code.

Quick reference R code

```
install.packages("circular")
require(circular)
set.seed(1234)
beaver.1 <- rvonmises(n=50, mu=circular(0), kappa=3)
beaver.2 <- rvonmises(n=50, mu=circular(pi), kappa=3)
beaver.3 <- rvonmises(n=50, mu=circular(pi/2), kappa=3)
par(bg="cornsilk")
rose.diag(beaver.1,                    bins=18,axes=TRUE,ticks=TRUE,
border="black",col="darkred",tol=0,prop=1.5)
par(new=TRUE)
rose.diag(beaver.2,                    bins=18,axes=TRUE,ticks=TRUE,
border="black",col="darkblue",tol=0,prop=1.5)
par(new=TRUE)
rose.diag(beaver.3,                    bins=18,axes=TRUE,ticks=TRUE,
border="black",col="darkorange",tol=0,prop=1.5)          points(beaver.1,
col="darkred", stack=TRUE)
points(beaver.2, col="darkblue", stack=TRUE)
points(beaver.3, col="darkorange", stack=TRUE)
```

NOTES

Write your notes below…

36. Pollution rose

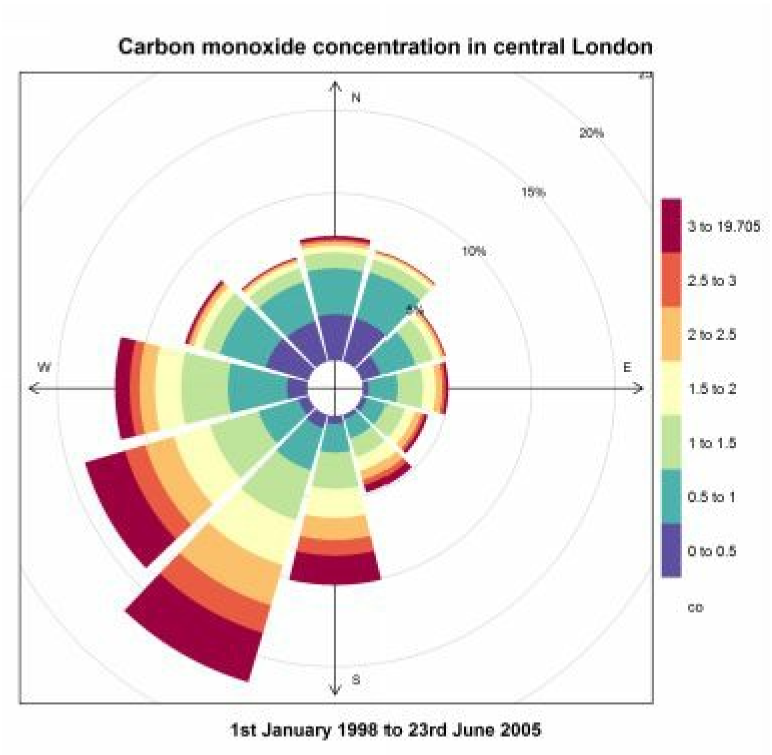


Figure 36 1 Carbon monoxide concentration measured at Marylebone

How to read the chart

The image shows the frequency of counts by wind direction for Carbon monoxide concentration (parts per million) measured at Marylebone, London. The petals are similar to the bars in a histogram and represent the frequency of observations. The petals are color coded to capture the extent of concentration. Circular bands are drawn at 5%,10%,15% and 20%. A petal that touches the 5% band represents 5% of the overall data.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("openair")
require(openair)# this package contains the data and generates the plot
```

Step 2: Clean and prepare the data

```
data(mydata)#data frame contains data on Carbon monoxide concentration
```

Step 3: Plot the chart

```
pollutant <- c("co")# Carbon monoxide observations
# set title and subtitle
title=c("Carbon monoxide concentration in central London")
sub=c("1st January 1998 to 23rd June 2005")

pollutionRose(mydata,paddle=FALSE,pollutant= pollutant,annotate=FALSE,
key.footer = pollutant, key.position = "right", key = TRUE, breaks =
8,main=title, sub=sub,auto.text=FALSE)
```

Visualization quick tips

Tip 1: The location of the key is determined by the argument key.posiition; It can take values “left”,”right”,”top” and “bottom”. The number of colors (or breaks) represented in the key is determined by the argument breaks; for many datasets values between 6-8 work well.

Tip 2: You can change the plot to a modern wind rose by setting the argument paddle=TRUE.

Tip 3: When substituting your own data note the pollutionRose function requires data frames with a field "date" that can be in either POSIXct or Date format but should be set to the Greenwich Mean Time time-zone. The data should be at least hourly or higher resolution.

Tip 4: The chart used the argument pollutant <- c("co") to access the carbon monoxide observations in the mydata data frame. This data frame contains a number of other air quality observations taken in central London:

1. "ws" - Wind speed.
2. "wd" - Wind direction, in degrees from North.
3. "nox"- Oxides of nitrogen concentration.
4. "no2" - Nitrogen dioxide concentration.
5. "o3"- Ozone concentration.
6. "pm10" - Particulate PM10 fraction measurement.
7. "so2" - Sulfur dioxide concentration.

8. "pm25" - Particulate PM2.5 fraction measurement.

Quick reference R code

```
install.packages("openair")
require(openair)
data(mydata)#dataframe contains the data on Carbon monoxide concentration, in ppm, as a numeric vector
pollutant <- c("co")# Carbon monoxide data
title=c("Carbon monoxide concentration in central London")
sub=c("1st January 1998 to 23rd June 2005")
pollutionRose(mydata,paddle=FALSE,pollutant= pollutant,annotate=F, key.footer = pollutant, key.position = "right", key = TRUE, breaks = 8,main=title, sub=sub,auto.text=FALSE)
```

NOTES

Write your notes below…

37. Polar frequency plot

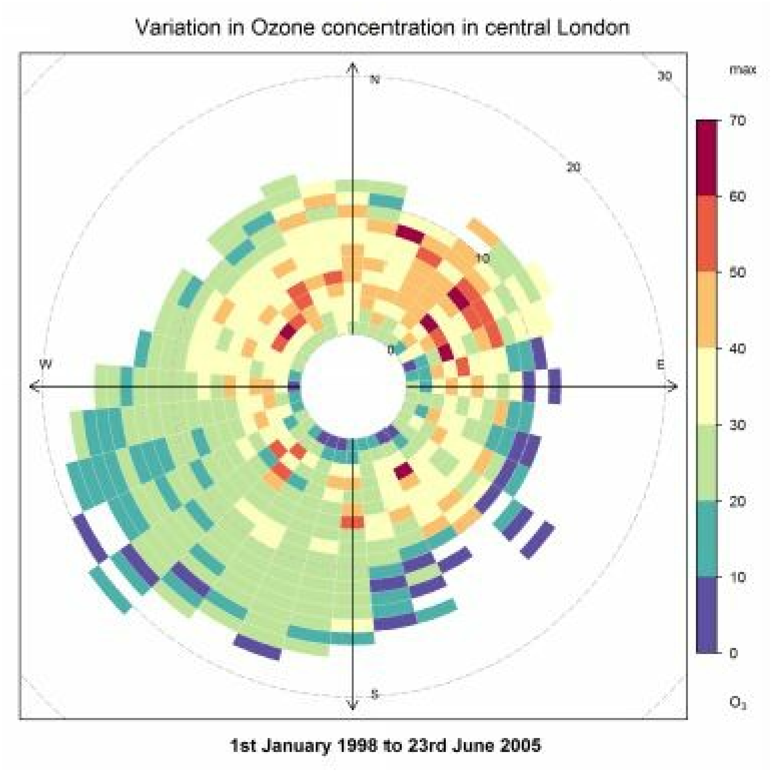


Figure 37 1 Variation in Ozone concentration measured at Marylebone

How to read the chart

The image shows the variation by wind direction in Ozone concentration measured at Marylebone, London. The rectangular colored bars are similar to the bars in a histogram and represent the frequency of observations. The bars are color coded to capture the extent of concentration. Circular bands are drawn at intervals of 10%.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("openair")
require(openair)# this package contains the data and generates the plot
```

Step 2: Clean and prepare the data

```
data(mydata)#dataframe contains data on Carbon monoxide concentration
```

Step 3: Plot the chart

```
pollutant <- c("o3")# Carbon monoxide observations
```

```
# set title and subtitle
title=c("Variation in Ozone concentration in central London")
sub=c("1st January 1998 to 23rd June 2005")
polarFreq (mydata,paddle=FALSE,statistic="max",pollutant=
pollutant,annotate=F, key.footer = pollutant, key.position = "right", key =
TRUE,main=title,
sub=sub,auto.text=TRUE,grid.line=10,border.col="white",breaks = seq(0, 70,
10))
```

Visualization quick tips

Tip 1: The location of the key is determined by the argument key.posiition; It can take values "left","right","top" and "bottom".

Tip 2: The number of colors (or breaks) represented in the key is controlled by the breaks argument in the polarFreq function. The imaged used breaks = seq(0, 70, 10), which generated a series of breaks from 0 to 70 in increments of 10. Try breaks = seq(0, 70, 5) and observe the difference.

Tip 3: When substituting your own data note the polarFreq function requires data frames with a field "date" that can be in either POSIXct or standard R Date format. It should be set to the Greenwich Mean Time time-zone. The data should ideally be at hourly or higher resolution.

Tip 4: The argument grid.line=10 controls the spacing of the dotted grid lines. The argument border.col="white" controls the color of the boarder around each colored rectangular. Set border.col = "transparent" to have an invisible border around each colored rectangle.

Tip 5: The images used the argument pollutant <- c("o3") to access the carbon monoxide observations in the mydata data frame. This data frame contains a number of other air quality observations taken in central London:

9. "ws" - Wind speed.
10. "wd" - Wind direction, in degrees from North.
11. "nox"- Oxides of nitrogen concentration.
12. "no2" - Nitrogen dioxide concentration.
13. "co"- Carbon monoxide concentration.
14. "pm10" - Particulate PM10 fraction measurement.
15. "so2" - Sulfur dioxide concentration.
16. "pm25" - Particulate PM2.5 fraction measurement.

Quick reference R code

```
install.packages("openair")
require(openair)
data(mydata)#dataframe contains the data on Carbon monoxide concentration, in ppm, as a numeric vector
pollutant <- c("o3")# Carbon monoxide data
title=c("Variation in Ozone concentration in central London")
sub=c("1st January 1998 to 23rd June 2005")
polarFreq (mydata,paddle=FALSE,statistic="max",pollutant= pollutant,annotate=F, key.footer = pollutant, key.position = "right", key = TRUE,main=title,
sub=sub,auto.text=TRUE,grid.line=10,border.col="white",breaks = seq(0, 70, 10))
```

NOTES

Write your notes below…

38. Australian wind rose

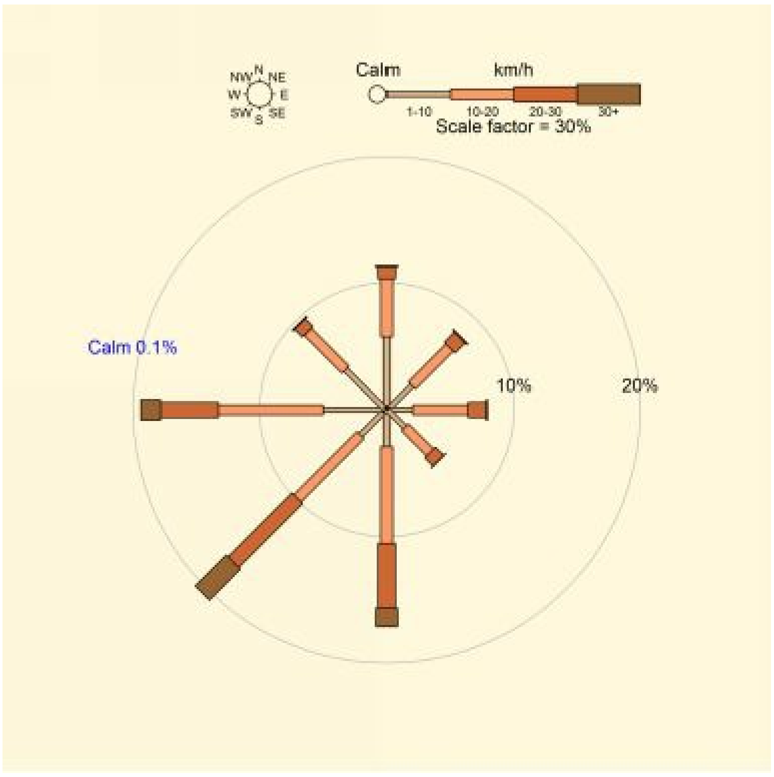


Figure 38 1 Wind direction & wind speed measured at Marylebone, London

How to read the chart

The chart shows a traditional wind rose using data measured at Marylebone, London between 1st January 1998 to 23rd June 2005. Wind directions are divided into eight compass directions. The circles around the image represent the various percentages of occurrence of the winds. For example, the branch to the north west just reaches the 10% ring implying a frequency of 10% blowing from that direction. Calm has no direction.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("openair")# contains the data
install.packages("plotrix")# used to draw the chart
require(openair)
require(plotrix)
```

Step 2: Clean and prepare the data

```
# get the wind speed and direction data
```

```
raw.data<-cbind(mydata$wd,mydata$ws)
# clean the data to remove missing observations
data<-na.omit(raw.data)
# put the contents into a table where the rows represent speed
#ranges and the columns indicating wind directions.
#also convert wind speed from m/s to km per hour
table<-bin.wind.records(data[,1],data[,2]*3.6,ndir=8)
```

Step 3: Plot the chart

```
par(bg="cornsilk")# set background color
oz.windrose(table)
```

Visualization quick tips

Tip 1: Data for the oz.windrose function needs to be in a format where the rows represent speed ranges and the columns indicating wind directions. Furthermore the data should be in percentages such that the sum of all data in your table is equal to 100. This is best achieved using the bin.wind.records function. This function classifies wind direction and speed records into a matrix of percentages of observations in speed and direction bins. Use the argument ndir to change the number of direction bins to be used in the wind rose. The traditional number of bins is ndir=8.

Tip 2: Use the argument speed.col in the oz.windrose function to change the color of the branches of the wind rose. The default colors, shown in the above chart are obtained by speed.col =c("#dab286","#fe9a66","#ce6733","#986434"). You could also set alternative colors such as speed.col =c("red","green","purple","yellow").

Tip 3: To remove the legend add the argument show.legend=FALSE to the oz.windrose function.

Quick reference R code

```
install.packages("openair")
install.packages("plotrix")
require(openair)
require(plotrix)

raw.data<-cbind(mydata$wd,mydata$ws)
data<-na.omit(raw.data)# remove missing observations
table<-bin.wind.records(data[,1],data[,2]*3.6,ndir=8)# create table and covert wind speed to km per hour
par(bg="cornsilk")
oz.windrose(table)
```

NOTES

Write your notes below…

39. Basic heat map

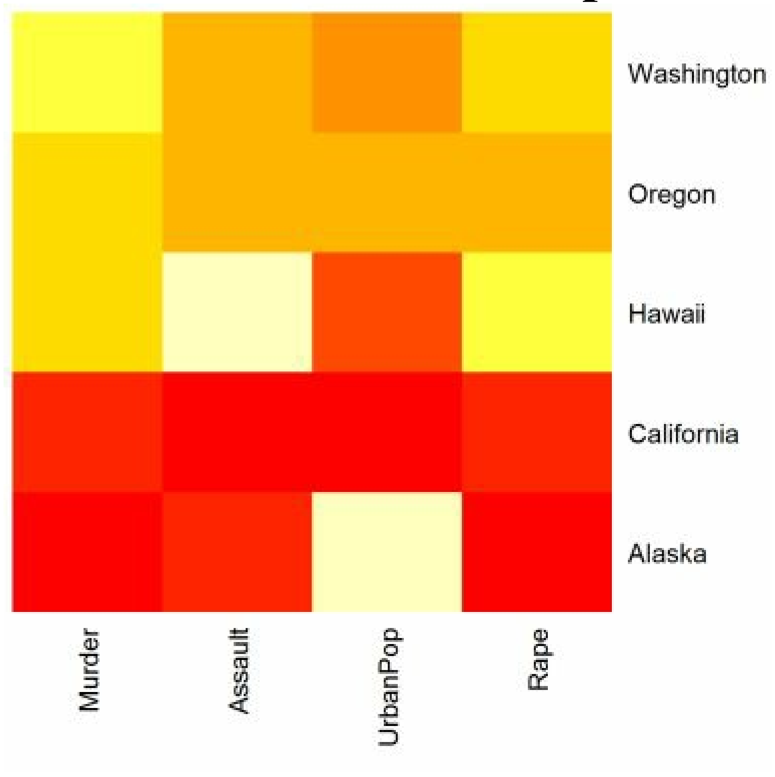


Figure 39 1 Violent crime in Pacific US States in 1973

How to read the chart

The chart is constructed from statistics, in arrests per 100,000 residents for assault, murder, and rape in each of the five Pacific US states in 1973. The shading goes from dark red to very light yellow. The darker the shading the higher the number of violent offenses. The variable UrbanPop on the horizontal axes refers to the size of the urban population, the darker the shading the higher the urban population.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("datasets") # you only need to do this once
require(datasets) # datasets package contains the chart data.
```

Step 2: Clean and prepare the data

```
data<-as.matrix(-rbind(USArrests[2,],USArrests[5,],USArrests[11,],
USArrests[37,],USArrests[47,]))
# use data contained in data frame USArrests. Note the minus sign before the
```

rbind statement. This is set so when the chart is drawn in the next step, the deeper the red (darker color) the higher the variable of interest. Alaska, California, Hawaii, Oregon and Washington are in rows 2,5,11,37 and 47 of USArrests.

Step 3: Plot the chart

```
heatmap(data, Rowv=NA,Colv = NA, scale="column",cexCol=1, cexRow=1
,col= heat.colors(10, alpha = 1) )
```

Visualization quick tips

Tip 1: The heatmap function requires data in a matrix format. When using your own data or other datasets you can use the as.matrix function to coerce data into a matrix.

Tip 2: You can change the size of the text on the column and row by using the arguments cexRow, cexCol respectively. The arguments take positive values. The larger the value, the larger the text. As a rule of thumb set both equal to 1 and then experiment for best effect.

Tip 3: The argument col=heat.colors takes two parameters. The first is the number of colors to use. The second relates to the transparency of the colors. The chart in this chapter used 10 colors. Try experimenting with a lower number. The chart in this chapter set transparency equal to 1 (no transparency). It takes values between 0 and 1. Try experimenting with different values, the visual impact can be quite stunning.

Tip 4: The heat.colors plots the heatmap using shades of red, orange and yellow. Many other options are available. For example terrain.colors(n, alpha = 1), topo.colors(n, alpha = 1) and cm.colors(n, alpha = 1). The parameter n refers to the number of colors and alpha the degree of transparency. As an illustration you might set col= terrain.colors(10, 1). Experiment with color and parameter values to see if trends in the dataset appear more pronounced.

Quick reference R code

```
install.packages("datasets")
require(datasets)
data<-as.matrix(-rbind(USArrests[2,],USArrests[5,],USArrests[11,],
USArrests[37,],USArrests[47,]))
heatmap(data,  Rowv=NA,Colv  =  NA,  scale="column",cexCol=1,  cexRow=1
,col= heat.colors(10, alpha = 1) )
```

NOTES

Write your notes below…

40. Circular data heat map with dendrograms

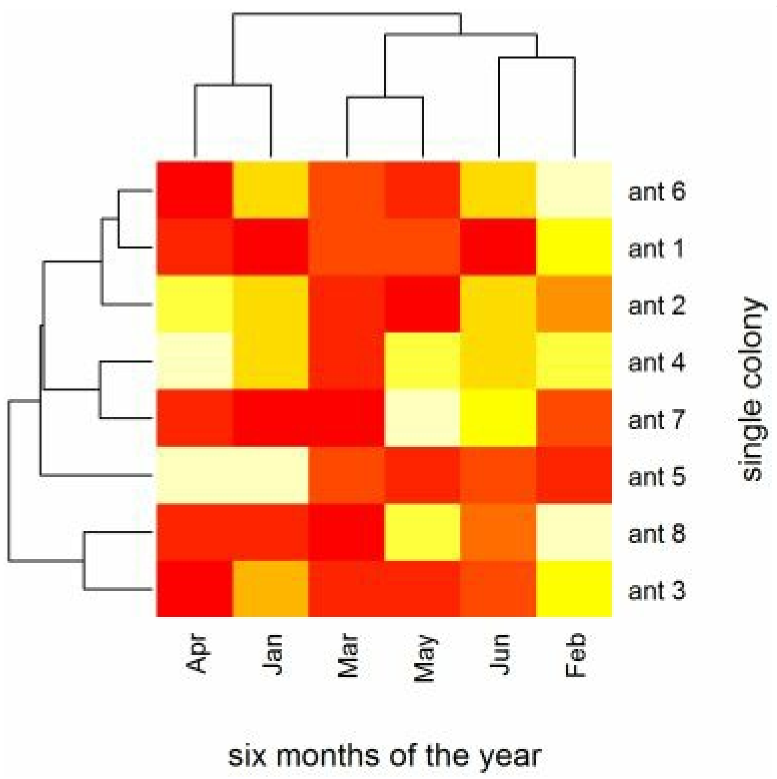


Figure 40 1 Simulated direction of foraging ants at different times of the year

How to read the chart

The chart is constructed from simulated circular data representing the foraging direction of eight ants from a single colony at various times of the year. The shading in the main panel goes from dark red (higher circular values) to very light yellow (lower circular values). The column dendrogram (top) shows the relationship between the various months of the year. The row dendrogram (on left of diagram) shows the relationship between the various ants.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("circular") # you only need to do this once
require(circular) # this package generates data and plot
```

Step 2: Clean and prepare the data

```
set.seed(1234) )# this allows you to replicate the chart
simulated.ant <- circular(matrix(rwrappedstable(48, scale=1, index =1, skewness =0), 8,6) )
```

# a 8 by 6 matrix of simulated observations is stored in the variable simulated.ant. The function rwrappedstable generates observations from a wrapped stable distribution. An index value of 1 simulates from the Cauchy distribution, an index value of 2 simulates from the normal distribution.
colnames(simulated.ant) <- c("Jan","Feb","Mar","Apr","May", "Jun") #create column names
rownames(simulated.ant) <- c("ant 1","ant 2","ant 3","ant 4","ant 5","ant 6","ant 7","ant 8")#create row names

Step 3: Plot the chart
heatmap.circular (simulated.ant,cexCol=1,cexRow=1 ,col= heat.colors(10, alpha = 1),ylab="single colony",xlab="six months of the year")

Visualization quick tips

Tip 1: The heatmap.circular function requires data in a matrix format. When using your own data or other datasets you can use the as.matrix function to coerce data into a matrix.

Tip 2: You can change the size of the text on the column and row by using the arguments cexRow, cexCol respectively. The arguments take positive values. The larger the value, the larger the text. As a rule of thumb set both equal to 1 and then experiment for best effect.

Tip 3: You can insert a title on to the chart by adding the argument main="my title" to the heatmap.circular function. To suppress the row dendrogram add the argument Rowv = NA to the heatmap.circular function. To suppress the column dendrogram add the argument Colv = NA to the heatmap.circular function.

Tip 4: The argument col=heat.colors takes two parameters. The first is the number of colors to use. The second relates to the transparency of the colors. The chart in this chapter used 10 colors. Try experimenting with a lower number of color. The chart in this chapter set transparency equal to 1 (no transparency). It takes values between 0 and 1. Try experimenting with different values, the visual impact can be quite stunning.

Tip 5: The heat.colors function plots shades of red, orange and yellow. Many other options are available. For example terrain.colors(n, alpha = 1), topo.colors(n, alpha = 1) and cm.colors(n, alpha = 1). The parameter n refers to the number of colors and alpha the degree of transparency. As an illustration you might set col= terrain.colors(10, 1). Experiment with color and parameter values to see if trends in the dataset appear more pronounced.

Quick reference R code

```
install.packages("circular") # you only need to do this once
require(circular)
set.seed(1234)
simulated.ant <- circular(matrix(rwrappedstable(48, scale=1, index=1,
skewness =0), 8,6) )
colnames(simulated.ant) <- c("Jan","Feb","Mar","Apr","May","Jun")
rownames(simulated.ant) <- c("ant 1","ant 2","ant 3","ant 4","ant 5","ant 6","ant
7","ant 8")
heatmap.circular (simulated.ant,cexCol=1,cexRow=1 ,col= heat.colors(10,
alpha = 1),ylab="single colony",xlab="six months of the year")
```

NOTES

Write your notes below…

41. Column trace dendrogram heat map

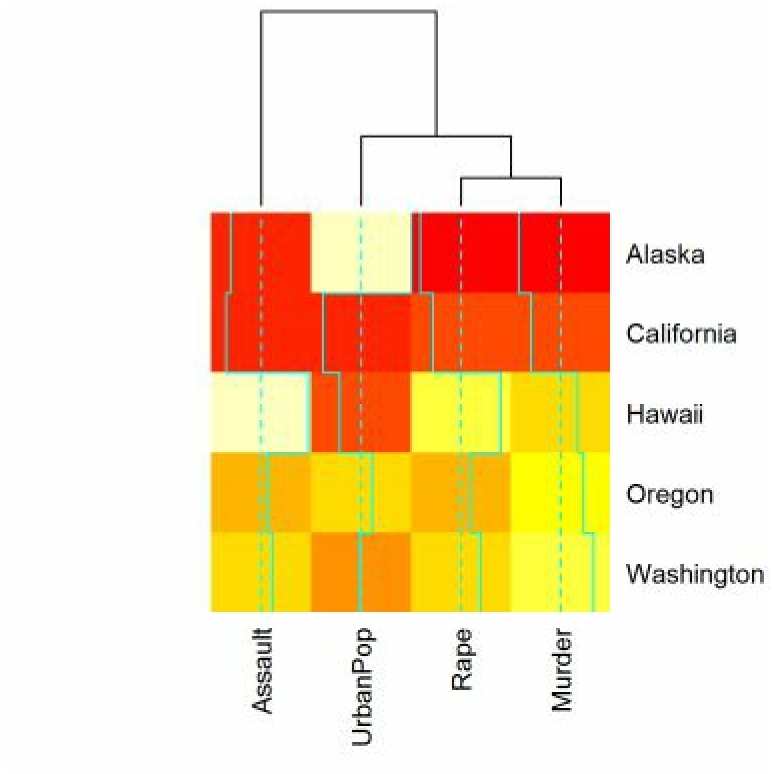


Figure 41 1 Violent crime in Pacific US States in 1973

How to read the chart

The chart is constructed from statistics, in arrests per 100,000 residents for assault, murder, and rape in each of the five Pacific US states in 1973. The shading goes from dark red to very light yellow. The darker the shading the higher the number of violent offenses. The variable UrbanPop on the horizontal axes refers to the size of the urban population, the darker the shading the higher the urban population. The trace (solid line) is drawn down the column's. The distance of the line from the center of each color-cell is proportional to the size of the measurement.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("datasets") # you only need to do this once
install.packages("gplots") # this package generates the plot
require(datasets) # datasets package contains the chart data.
require(gplots)
```

Step 2: Clean and prepare the data

```
data<-as.matrix(-rbind(USArrests[2,],USArrests[5,],USArrests[11,]
,USArrests[37,],USArrests[47,]))
```

# use data contained in data frame USArrests and set so the deeper the red (darker color) the higher the variable of interest. Alaska, California, Hawaii, Oregon and Washington are in rows 2,5,11,37 and 47 of USArrests.

Step 3: Plot the chart

```
heatmap.2(data,scale="column",cexCol=1,cexRow=1,        col=heat.colors(10,
alpha = 1), density.info='none' , Rowv=NA,key=FALSE)
```

Visualization quick tips

Tip 1: The heatmap.2 function requires data in a matrix format. When using your own data or other datasets you can use the as.matrix function to coerce data into a matrix.

Tip 2: You can change the size of the text on the column and row by using the arguments cexRow, cexCol respectively. The arguments take positive values. The larger the value, the larger the text. As a rule of thumb set both equal to 1 and then experiment for best effect.

Tip 3: You can insert a title by adding the argument main="my title" to the heatmap.2 function. To activate the row dendrogram add the argument Rowv = TRUE. To suppress the column dendrogram add the argument Colv = NA.

Tip 4: The argument col=heat.colors takes two parameters. The first is the number of colors to use. The second relates to the transparency of the colors. The chart in this chapter used 10 colors. Try experimenting with a lower number of color. The chart in this chapter set transparency equal to 1 (no transparency). It takes values between 0 and 1.

Tip 5: The heat.colors plots shades of red, orange and yellow. Many other options are available. For example terrain.colors(n, alpha = 1), topo.colors(n, alpha = 1) and cm.colors(n, alpha = 1). The parameter n refers to the number of colors and alpha the degree of transparency. As an illustration you might set col= terrain.colors(10, 1). Experiment with color and parameter values to see if trends in the dataset appear more pronounced.

Tip 6: To add a color key to the chart set the argument key=TRUE. You can set the size of the color key by using the keysize argument. It takes positive values. The larger the value the larger the key. Try experimenting different values. Good starting points are keysize=2 and keysize =1.

Quick reference R code

```
install.packages("datasets") # you only need to do this once
install.packages("gplots") # this package generates the plot
require(datasets) # datasets package contains the chart data.
require(gplots)
data<-as.matrix(-rbind(USArrests[2,],USArrests[5,],USArrests[11,],
USArrests[37,],USArrests[47,]))
heatmap.2(data, scale="column",cexCol=1,cexRow=1,col= heat.colors(10,
alpha = 1), density.info = 'none', Rowv=NA,key=FALSE)
```

NOTES

Write your notes below…

42. Annotated heat map

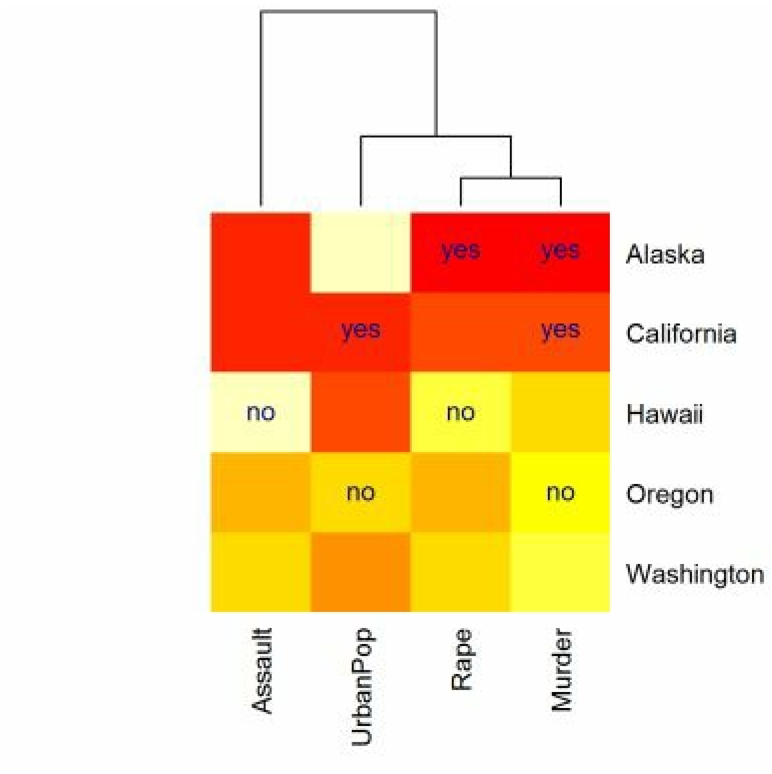


Figure 42 1 Violent crime in Pacific US States in 1973

How to read the chart

The chart is constructed from statistics, in arrests per 100,000 residents for assault, murder, and rape in each of the five Pacific US states in 1973. The shading goes from dark red to very light yellow. The darker the shading the higher the number of violent offenses. The variable UrbanPop on the horizontal axes refers to the size of the urban population, the darker the shading the higher the urban population. The chart has been annotated with comments on those areas where the researcher has sufficient information to recommend policy action (marked “yes”) and those areas where further information is sought (market “no”).

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("datasets") # you only need to do this once
install.packages("gplots") # this package generates the plot
require(datasets) # datasets package contains the chart data.
```

require(gplots)

Step 2: Clean and prepare the data

data<-as.matrix(-rbind(USArrests[2,],USArrests[5,],USArrests[11,], USArrests[37,],USArrests[47,]))

# use data contained in data frame USArrests and set so the deeper the red (darker color) the higher the variable of interest. Alaska, California, Hawaii, Oregon and Washington are in rows 2,5,11,37 and 47 of USArrests.

text<-matrix("",5,4) # Create a blank text matrix
text[1,1]= "yes" # add text comments to appear on heat plot
text[2,1]= "yes"
text[2,3]= "yes"
text[1,4]= "yes"
text[4,1]= "no"
text[3,2]= "no"
text[4,3]= "no"
text[3,4]= "no"

Step 3: Plot the chart

heatmap.2(data, scale="column",cexCol=1,cexRow=1 ,col= heat.colors(10, alpha = 1), density.info='none', cellnote=text, trace="none",notecol="darkblue",Rowv = FALSE,key=FALSE)

Visualization quick tips

Tip 1: The heatmap.2 function requires data in a matrix format. When using your own data or other datasets you can use the as.matrix function to coerce data into a matrix.

Tip 2: You can change the size of the text on the column and row by using the arguments cexRow, cexCol respectively. The arguments take positive values. The larger the value, the larger the text. As a rule of thumb set both equal to 1 and then experiment for best effect.

Tip 3: You can insert a title on to the chart by adding the argument main="my title" to the heatmap.2 function. To activate the row dendrogram add the argument Rowv = TRUE. To suppress the column dendrogram add the argument Colv = NA.

Tip 4: The argument col=heat.colors takes two parameters. The first is the number of colors to use. The second relates to the transparency of the colors. The chart in this chapter used 10 colors. Try experimenting with a lower number of color. The chart in this chapter set transparency equal to 1 (no transparency). It takes values between 0 and 1. Try experimenting with different values, the visual impact can be quite stunning.

Tip 5: The heat.colors plots shades of red, orange and yellow. Many other options are available. For example terrain.colors(n, alpha = 1), topo.colors(n, alpha = 1) and cm.colors(n, alpha = 1). The parameter n refers to the number of colors and alpha the degree of transparency. As an illustration you might set col= terrain.colors(10, 1). Experiment with color and parameter values to see if trends in the dataset appear more pronounced.

Tip 6: To add a color key to the chart set the argument key=TRUE. You can set the size of the color key by using the keysize argument. It takes positive values, larger values increase the size of the key. Try experimenting different values. Good starting points are keysize=2 and keysize =1.

Quick reference R code

```
install.packages("datasets")
install.packages("gplots")
require(datasets)
require(gplots)
data<-as.matrix(-rbind(USArrests[2,],USArrests[5,],USArrests[11,],
USArrests[37,],USArrests[47,]))
text<-matrix("",5,4)
text[1,1]= "yes"
```

```
text[2,1]= "yes"
text[2,3]= "yes"
text[1,4]= "yes"
text[4,1]= "no"
text[3,2]= "no"
text[4,3]= "no"
text[3,4]= "no"
heatmap.2(data, scale="column", cexCol=1,cexRow=1 ,col= heat.colors(10, alpha = 1), density.info='none', cellnote=text, trace="none",notecol="darkblue",Rowv = FALSE,key=FALSE)
```

NOTES

Write your notes below…

43. Correlation matrix

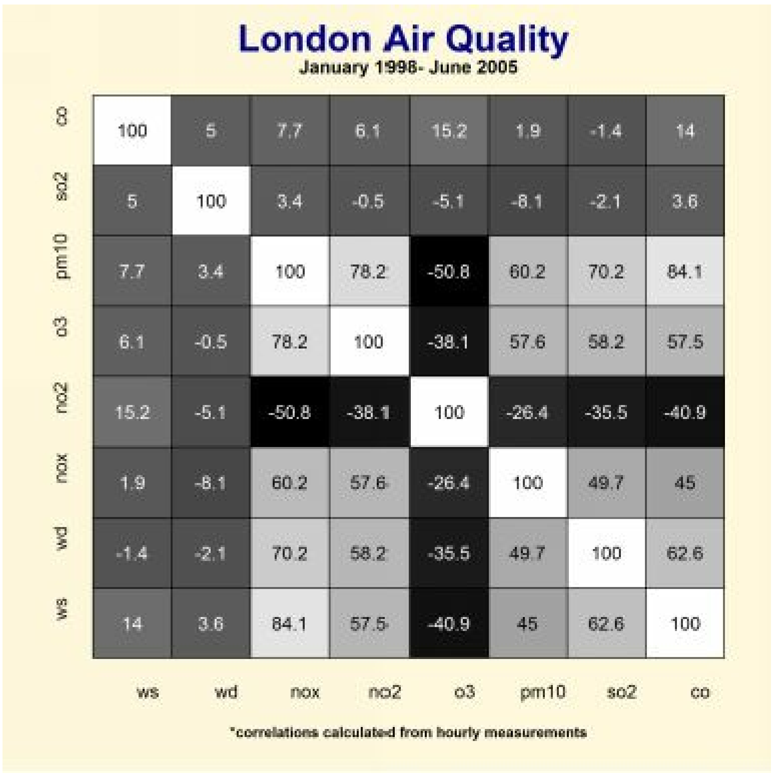


Figure 43 1 Air quality factor correlations measured at Marylebone

How to read the chart

The chart is constructed from the correlation between the eight air quality variables collected at the air quality monitoring station in Marylebone, London. The actual correlations are printed inside each square. The grayscale shading of each square represents the strength of correlation. The scale runs from black (negative 100% correlation) to white (positive 100% correlation).

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("openair")
install.packages("plotrix")
require(openair)#contains the data
require(plotrix)# draws the plot
```

Step 2: Clean and prepare the data

```
# calculate the correlation between indicators
```

```
x<-na.omit(mydata[,2:9])#remove missing values
cor<- cor(x)*100 # make sure correlation is between -100 to +100
```

Step 3: Plot the chart

```
par(bg="cornsilk")# set background color
color2D.matplot(x=round(cor,1), show.values=TRUE,axes=FALSE,
xlab="",ylab="")
#add axis
axis(side=1,at=(0.7:7.7), labels=colnames(cor),cex.axis=1, tick=FALSE)
axis(side=2,at=(0.7:7.7), labels=colnames(cor),cex.axis=1, tick=FALSE)
#add titles
title("London Air Quality ",col.main="darkblue",line=2, cex.main =2)
title("January 1998- June 2005",col.main="black",line=1, cex.main =1)
title("*correlations calculated from hourly
measurements",col.main="black",line=-29, cex.main =0.8)
```

Visualization quick tips

Tip 1: You can replace the squares with hexagons by adding the argument do.hex=TRUE to the plotting function color2D.matplot.

Tip 2: To hide the correlation values set the argument show.values=FALSE. You can also supply your own x and y axis labels by setting xlab="my x axis label",ylab="my y axis label".

Tip 3: To add a graded legend to the chart add the argument show.legend=TRUE to the plotting function color2D.matplot.

Quick reference R code

```
install.packages("openair")
install.packages("plotrix")
require(openair)#contains the data
require(plotrix)# draws the plot
x<-na.omit(mydata[,2:9])#remove missing values
```

```
cor<- cor(x)*100
par(bg="cornsilk")
color2D.matplot(x=round(cor,1),                    show.values=TRUE,axes=FALSE,
xlab="",ylab="")
#create basic plot area
axis(side=1,at=(0.7:7.7), labels=colnames(cor),cex.axis=1, tick=FALSE)
axis(side=2,at=(0.7:7.7), labels=colnames(cor),cex.axis=1, tick=FALSE)
title("London Air Quality ",col.main="darkblue",line=2, cex.main =2)
title("January 1998- June 2005",col.main="black",line=1, cex.main =1)
title("*correlations calculated from hourly
measurements",col.main="black",line=-29, cex.main =0.8)
```

NOTES

Write your notes below…

44. Hinton plot

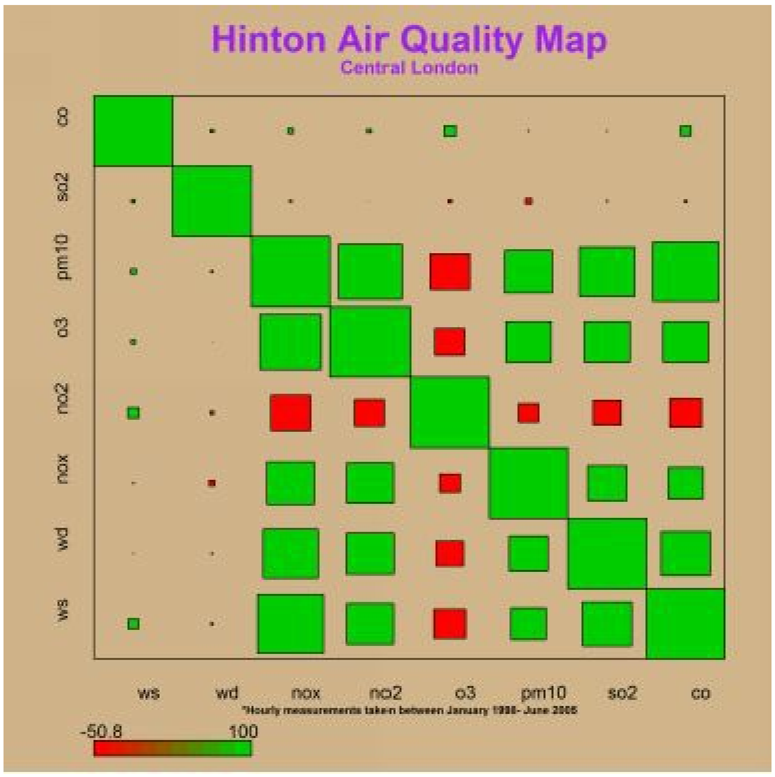


Figure 44 1 Derived Hinton plot from London air quality data

How to read the chart

The chart is constructed from the correlation between eight air quality variables collected at the air quality monitoring station in Marylebone, London. The size of the squares are proportional to the correlation with the sign indicated by the color of the squares.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("openair")
install.packages("plotrix")
require(openair)#contains the data
require(plotrix)# draws the plot
```

Step 2: Clean and prepare the data

```
# calculate the correlation between indicators
x<-na.omit(mydata[,2:9])#remove missing values
```

```
cor<- cor(x)*100 # make sure correlation is between -100 to +100
```

Step 3: Plot the chart

```
par(bg=" tan")# set background color
color2D.matplot(x=round(cor,1), Hinton=T, show.values=F,axes=FALSE,
xlab="",ylab="",show.legend=T,extremes=c(2,3))
#create basic plot area
axis(side=1,at=(0.7:7.7), labels=colnames(cor),cex.axis=1, tick=FALSE)
axis(side=2,at=(0.7:7.7), labels=colnames(cor),cex.axis=1, tick=FALSE)
#add titles
title("Hinton Air Quality Map",col.main="purple",line=2, cex.main =2)
title("Central London",col.main="purple",line=1, cex.main =1)
title("*Hourly measurements taken between January 1998- June
2005",col.main="black",line=-27.9, cex.main =0.6)
```

Visualization quick tips

Tip 1: You can create a correlation matrix by setting Hinton =FALSE in the plotting function color2D.matplot. Replace the squares with hexagons by adding the argument do.hex=TRUE to the plotting function color2D.matplot.

Tip 2: To see the actual variable values set the argument show.values=TRUE. You can also supply your own x and y axis labels by setting xlab="my x axis label",ylab="my y axis label".

Tip 3: To remove the legend set the argument show.legend=FALSE in the plotting function color2D.matplot.

Quick reference R code

```
install.packages("openair")
install.packages("plotrix")
require(openair)#contains the data
require(plotrix)# draws the plot
x<-na.omit(mydata[,2:9])#remove missing values
```

```
cor<- cor(x)*100
par(bg=" tan")
color2D.matplot(x=round(cor,1),     Hinton=T,     show.values=F,axes=FALSE,
xlab="",ylab="",show.legend=T,extremes=c(2,3))
axis(side=1,at=(0.7:7.7), labels=colnames(cor),cex.axis=1, tick=FALSE)
axis(side=2,at=(0.7:7.7), labels=colnames(cor),cex.axis=1, tick=FALSE)
title("Hinton Air Quality Map",col.main="purple",line=2, cex.main =2)
title("Central London",col.main="purple",line=1, cex.main =1)
title("*Hourly measurements taken between January 1998- June
2005",col.main="black",line=-27.9, cex.main =0.6)
```

NOTES

Write your notes below…

45. Battleship plot

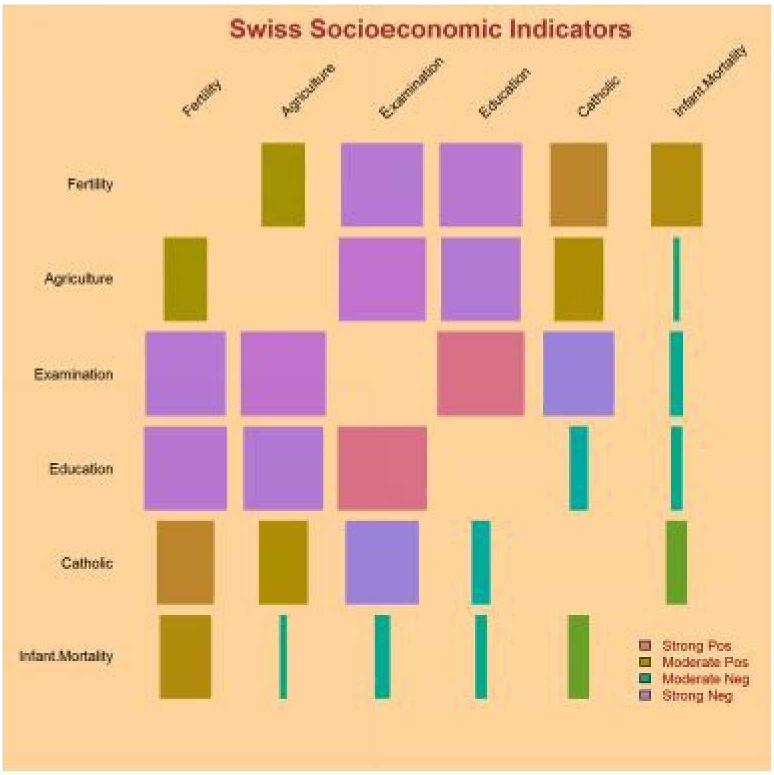


Figure 45 1 Battleship plot of Swiss data collected during 1888

How to read the chart

The chart is constructed from the correlation between the six socioeconomic indicators collected in French-speaking provinces of Switzerland around 1888. The size of the rectangular bars indicate the strength of the correlation. The larger the bar the higher the absolute correlation between two indicators. The bars are color coded to help distinguish positive correlated indicators from negatively correlated indicators.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("plotrix")#package will draw plot
require(plotrix)
require(datasets)# contains the Swiss data
```

Step 2: Clean and prepare the data

```
# calculate the correlation between indicators
data<-cor(swiss)
```

```
#set the diagonal elements of the correlation matrix to zero
data[1,1]=0
data[2,2]=0
data[3,3]=0
data[4,4]=0
data[5,5]=0
data[6,6]=0
```

Step 3: Plot the chart

```
par(bg="burlywood1")# set background color
battleship.plot(data,col=color.scale(-data,c(0,300),70,60, color.spec="hcl"),
cex.labels=0.8,border="burlywood1")
#add title
title("Swiss Socioeconomic Indicators",col.main="brown",line=3.5, cex.main
=1.4)

#add legend
legend(x=5.5,y=1.3, inset=.05, bty="n", cex=0.75, text.col ="darkred",c("Strong
Pos", "Moderate Pos", "Moderate Neg","Strong Neg"), fill= c("#D97088FF",
"#A49100FF", "#00AA83FF", "#B678D5FF"))
```

Visualization quick tips

Tip 1: The argument border="burlywood1" ensures the diagonal elements of the diagram are invisible. If you wish to see them, and have a border around each rectangle set border="your color".

Tip 2: You can change the size of the text on the x and y axis by using the argument cex.labels. The default value is 1, larger values increase the size of the x,y axis labels.

Tip 3: The function title is a simple, fast and flexible way to add a title to a chart. Use the line argument to place the title precisely where you require it.

Quick reference R code

```
install.packages("plotrix")#package will draw plot
require(plotrix)
require(datasets)

data<-cor(swiss)
#set the diagonal elements to zero
data[1,1]=0
data[2,2]=0
data[3,3]=0
data[4,4]=0
data[5,5]=0
data[6,6]=0
par(bg="burlywood1")
battleship.plot(data,col=color.scale(-data,c(0,300),70,60,  color.spec="hcl"),
cex.labels=0.8,border="burlywood1")

title("Swiss Socioeconomic Indicators",col.main="brown",line=3.5, cex.main
=1.4)
legend(x=5.5,y=1.3, inset=.05, bty="n", cex=0.75, text.col ="darkred",c("Strong
Pos", "Moderate Pos", "Moderate Neg","Strong Neg"), fill= c("#D97088FF",
"#A49100FF", "#00AA83FF", "#B678D5FF"))
```

NOTES

Write your notes below…

46. Density rose split plot

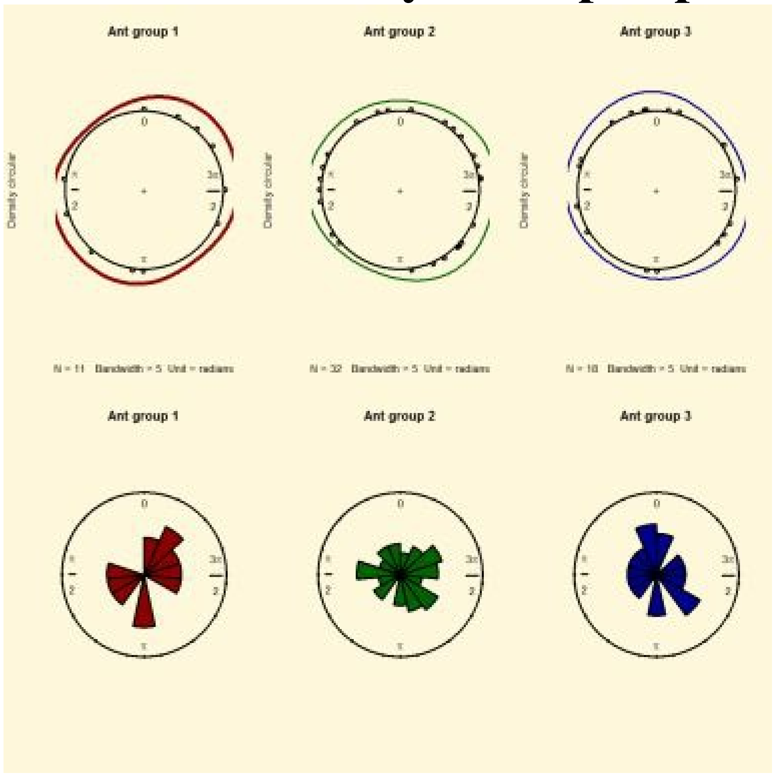


Figure 46 1 Three groups of desert ants –density plot with rose diagram

How to read the chart

The panel displays the circular density and rose diagram for the orientation of three sets of long-legged desert ants after one eye on each ant was 'trained' to learn the ant's home direction, then covered and the other eye uncovered. The top three images show the density fit for each group to the circular distribution. The bottom three images present the rose diagrams for each group of ants. The size of the "petals" indicate the primary orientation of each group of ants.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("circular")
require(circular)# this package contains the data and generates the plot
require(circular)
```

Step 2: Clean and prepare the data

```
ant.1<-circular(fisherB10$set1,zero=pi/2)
ant.2<-circular(fisherB10$set2,zero=pi/2)
```

ant.3<-circular(fisherB10$set3,zero=pi/2)
# The data frame fisherB10 contains observations on each of the ant groups. It is converted to an object of class circular and orientated so that zero equals pi/2. This will ensure zero appears at the top of the resultant arrow plots.

Step 3: Plot the chart
# create a 2 by 3 area, each of the six charts is plotted on one area and set the background color to cornsilk.
par(mfrow=c(2,3), bg="cornsilk")par(bg="cornsilk")
# fit density and then create plot for first group of ants
density.1<- density(ant.1, bw=5)
plot(density.1, points.plot=TRUE, col="darkred",main="Ant group 1",lwd=2)
# fit density and then create plot for second group of ants
density.2<- density(ant.2, bw=5)
plot(density.2, points.plot=TRUE, col="darkgreen",main="Ant group 2")
# fit density and then create plot for third group of ants
density.3<- density(ant.3, bw=5)
plot(density.3, points.plot=TRUE, col="darkblue",main="Ant group 3")
# create and plot the rose diagram for first group of ants
rose.diag(ant.1, bins=15,axes=TRUE, ticks=TRUE ,border= "black",col="darkred",tol=0,prop=1.5, main="Ant group 1")
# create and plot the rose diagram for second group of ants
rose.diag(ant.2, bins=15,axes=TRUE, ticks=TRUE ,border= "black",col="darkgreen",tol=0,prop=1.5, main="Ant group 2")
# create and plot the rose diagram for third group of ants
rose.diag(ant.3, bins=15,axes=TRUE, ticks=TRUE ,border= "black",col="darkblue",tol=0,prop=1.5, main="Ant group 3")

Visualization quick tips

Tip 1: The rose.diag function require data in a circular format. This means the data needs to be converted to circular class. When using your own data or other datasets you can use the statement circular (your_data) to coerce data into a circular format.

Tip 2: You can exclude the axes on each chart by setting the argument axes=FALSE in the plot function.

Tip 3: Change the background color by setting the argument bg="your chosen color".

Tip 4: Change the plot region using par(mfrow=c(x,y)), where x is the number of rows, and y is the number of columns. For example, par(mfrow=c(2,2)) would split the graphic output into four areas and then you could plot a chart to each specific area.

Quick reference R code

```
install.packages("circular") # this package contains the data and generates the
plot
require(circular)
ant.1<-circular(fisherB10$set1,zero=pi/2)
ant.2<-circular(fisherB10$set2,zero=pi/2)
ant.3<-circular(fisherB10$set3,zero=pi/2)
par(mfrow=c(2,3), bg="cornsilk")
density.1<- density(ant.1, bw=5)
plot(density.1, points.plot=TRUE, col="darkred",main="Ant group 1",lwd=2)
density.2<- density(ant.2, bw=5)
plot(density.2, points.plot=TRUE, col="darkgreen",main="Ant group 2")
density.3<- density(ant.3, bw=5)
plot(density.3, points.plot=TRUE, col="darkblue",main="Ant group 3")

rose.diag(ant.1, bins=15,axes=TRUE,ticks=TRUE
,border="black",col="darkred",tol=0,prop=1.5, main="Ant group 1")
rose.diag(ant.2, bins=15,axes=TRUE,ticks=TRUE
,border="black",col="darkgreen",tol=0,prop=1.5, main="Ant group 2")
rose.diag(ant.3, bins=15,axes=TRUE,ticks=TRUE
,border="black",col="darkblue",tol=0,prop=1.5, main="Ant group 3")
```

NOTES

Write your notes below…

47. Cylindrical graded bar plot

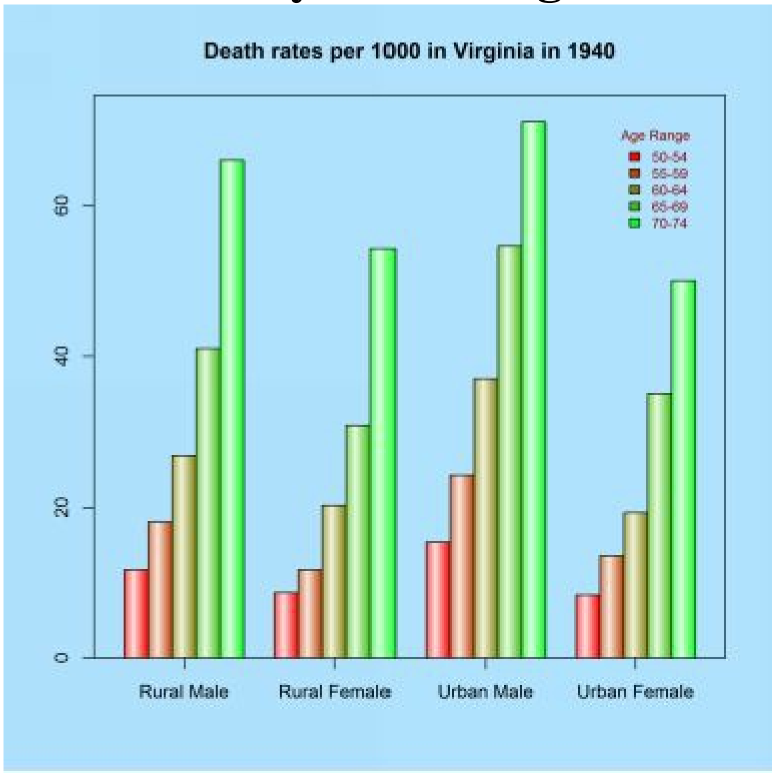


Figure 47 1 Rural and urban death rates in Virginia (1940)

How to read the chart

The chart displays the death rate per 1000 in Virginia during 1940. The data are broken out by gender and by location (urban, rural) The cylindrical bars are shaded by age group.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("plotrix")#used to draw the plot
require(plotrix)
require(datasets)#contains the data used in the example
```

Step 2: Clean and prepare the data

```
age<-c(54,59,64,69,74)#used to color bars
names=colnames(VADeaths) #used to label x-axis
```

Step 3: Plot the chart

```
main= "Death rates per 1000 in Virginia in 1940"# title
par(bg="lightskyblue1") #background color
barp(VADeaths,col=color.scale(age,c(1.1,0),c(0,1.1),c(0,0.1)),cylindrical=TRU
# text for legend
age.range=c("50-54",      "55-59","60-64",      "65-69",      "70-74")
#create legend
legend(x="topright",   inset=.05,   bty="n",   cex=0.75,   text.col   ="darkred",
legend=age.range,   fill=   c("#FF0000FF",   "#BF4006FF",   "#80800DFF",
"#40BF13FF", "#00FF1AFF"), title="Age Range")
```

Visualization quick tips

Tip 1: You can draw standard rectangular bars by setting the argument cylindrical=FALSE.

Tip 2: It is possible to add colors to the legend plot by specifying the color name rather than number, for example fill= c("red", "orange", "purple", "green", "blue")

Tip 3: Change the size of the text in the legend by using the argument cex. It has a default value of 1, larger values increase the text size.

Quick reference R code

```
install.packages("plotrix")
require(plotrix)
require(datasets)
age<-c(54,59,64,69,74)
names=colnames(VADeaths)
main= "Death rates per 1000 in Virginia in 1940"
par(bg="lightskyblue1")
barp(VADeaths,col=color.scale(age,c(1.1,0),c(0,1.1),c(0,0.1)),cylindrical=TRU
age.range=c("50-54",      "55-59","60-64",      "65-69",      "70-74")
```

```
legend(x="topright", inset=.05, bty="n", cex=0.75, text.col ="darkred",
legend=age.range, fill= c("#FF0000FF", "#BF4006FF", "#80800DFF",
"#40BF13FF", "#00FF1AFF"), title="Age Range")
```

NOTES

Write your notes below…

48. Centipede plot

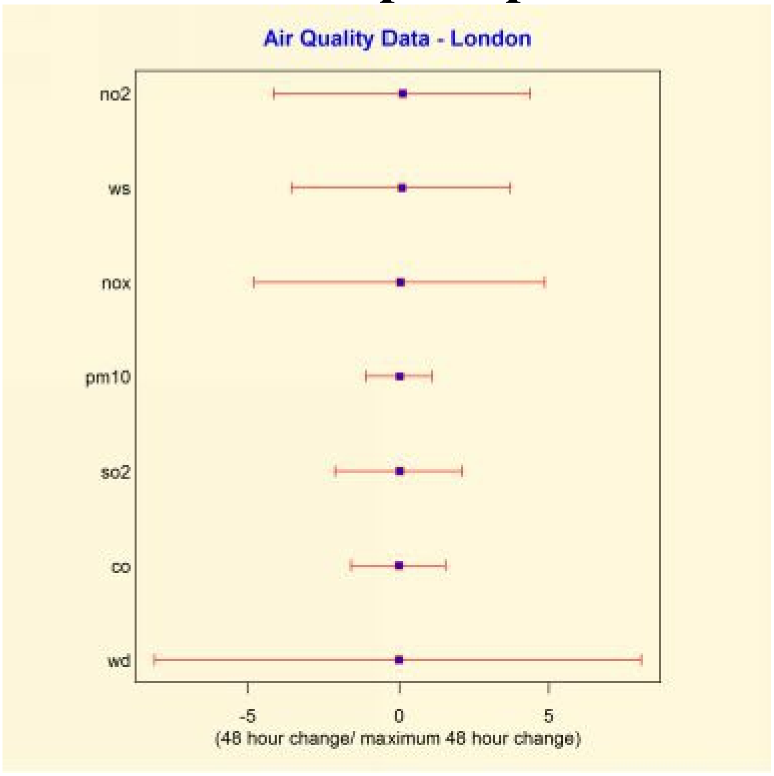


Figure 48 1 Variation in air quality metrics measured at Marylebone

How to read the chart

The chart displays 48 hour variation in various air quality measurements taken in central London from January 1st 1998 to June 23rd 2005. The extent of the red line represents captures the degree of variation of each environmental factor; The blue square in the middle of each line represents the median or average value.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("openair")
install.packages("plotrix")
require(openair)#contains the data
require(plotrix)# draws the plot
```

Step 2: Clean and prepare the data

```
#mydata contains the observations used
x<-na.omit(mydata[,2:9])#remove missing values
```

```
diff=48# used to calculate 48 hour change
#calculate the 48 hour difference of each variable
x1 <- diff(x[,1], k= diff)
x2 <- diff(x[,2], k= diff)
x3 <- diff(x[,3], k= diff)
x4 <- diff(x[,4], k= diff)
x5 <- diff(x[,6], k= diff)
x6 <- diff(x[,7], k= diff)
x7 <- diff(x[,8], k= diff)

#divide 48 hour changes by the maximum 48 hour change
x1 <- x1/max(x1)
x2 <- x2/max(x2)
x3 <- x3/max(x3)
x4 <- x4/max(x4)
x5 <- x5/max(x5)
x6 <- x6/max(x6)
x7 <- x7/max(x7)
#create a list suitable for use in the plot function
temp<-list("",7)
temp[[1]]<-x1
temp[[2]]<-x2
temp[[3]]<-x3
temp[[4]]<-x4
temp[[5]]<-x5
temp[[6]]<-x6
temp[[7]]<-x7
# put data into correct format for plotting
data<-get.segs(temp)
# add column names
colnames(data)=c("ws",  "wd",  "nox",  "no2" , "pm10", "so2",  "co")
```
Step 3: Plot the chart
```
par(bg="cornsilk")# set background color
```

```
centipede.plot(data*10000,sort.segs=TRUE,right.labels=rep("",7),main="Air Quality Data - London",xlab="(48 hour change/ maximum 48 hour change)",pch=22,col="red",bg="blue",col.main="blue")
```

Visualization quick tips

Tip 1: Change the background color of the plot using par(bg="my color") .

Tip 2: The observations are sorted by average value, this is controlled by the argument sort.segs=TRUE. Set to FALSE to plot in the order they appear in the dataset.

Tip 3: Remove the argument right.labels=rep("",7) if you would like to see a count of the number of observations for each variable appear on the right hand side of the graph.

Tip 4: The arguments pch,col,bg, and col.main control the type of shape used to represent the median (use values in the range 19-26), the color of the lines representing variation, color of the shape used to represent the median and the color used in the title.

Quick reference R code

```
install.packages("openair")
install.packages("plotrix")
require(openair)#contains the data
require(plotrix)# draws the plot

x<-na.omit(mydata[,2:9])#remove missing values

diff=48# used to calculate 48 hour change
x1 <- diff(x[,1], k= diff)
x2 <- diff(x[,2], k= diff)
x3 <- diff(x[,3], k= diff)
```

```
x4 <- diff(x[,4], k= diff)
x5 <- diff(x[,6], k= diff)
x6 <- diff(x[,7], k= diff)
x7 <- diff(x[,8], k= diff)

#divide 48 hour changes by the maximum 48 hour change
x1 <- x1/max(x1)
x2 <- x2/max(x2)
x3 <- x3/max(x3)
x4 <- x4/max(x4)
x5 <- x5/max(x5)
x6 <- x6/max(x6)
x7 <- x7/max(x7)

temp<-list("",7)
temp[[1]]<-x1
temp[[2]]<-x2
temp[[3]]<-x3
temp[[4]]<-x4
temp[[5]]<-x5
temp[[6]]<-x6
temp[[7]]<-x7
data<-get.segs(temp)
colnames(data)=c("ws",  "wd",  "nox",  "no2" , "pm10", "so2",  "co")
par(bg="cornsilk")
centipede.plot(data*10000,sort.segs=TRUE,right.labels=rep("",7),main="Air
Quality Data - London",xlab="(48 hour change/ maximum 48 hour
change)",pch=22,col="red",bg="blue",col.main="blue")
```

NOTES

Write your notes below…

49. Circular probability split plot

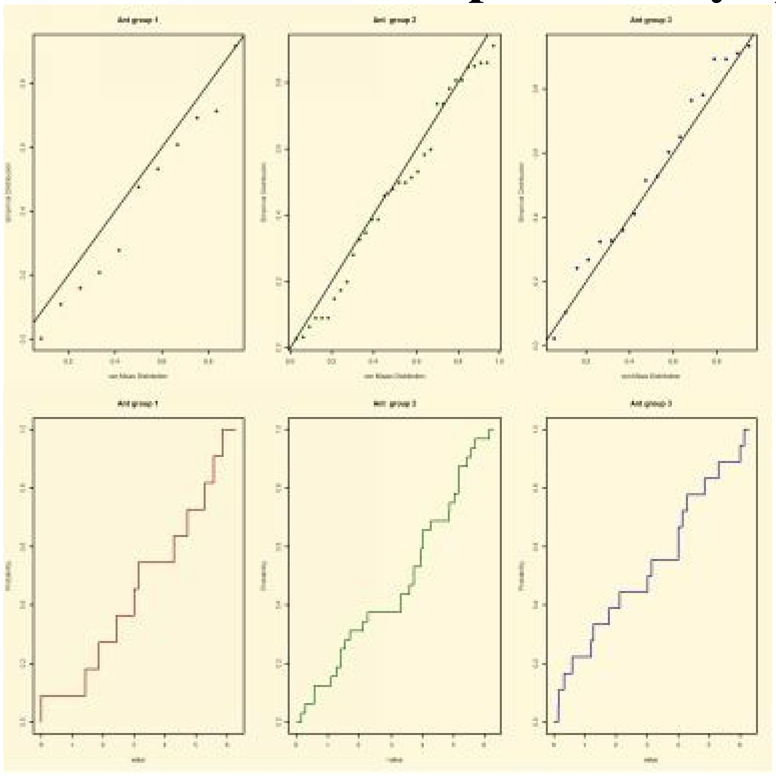


Figure 49 1 Probability and empirical plots for three groups of desert ants

How to read the chart

The chart displays probability and empirical probability plots for the orientation of three sets of long-legged desert ants after one eye on each ant was 'trained' to learn the ant's home direction, then covered and the other eye uncovered. The top three images plot the probability plot for each group relative to the von Misses distribution. The closer the dotted points are the solid black line, the better the observations fit the von Misses distribution. The bottom three charts plot the empirical cumulative probability distribution of the three groups of ants.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("circular")
require(circular)# this package contains the data and generates the plot
require(circular)
```

Step 2: Clean and prepare the data

```
ant.1<-circular(fisherB10$set1,zero=pi/2)
ant.2<-circular(fisherB10$set2,zero=pi/2)
ant.3<-circular(fisherB10$set3,zero=pi/2)
```

# The data frame fisherB10 contains observations on each of the ant groups. It is converted to an object of class circular and orientated so that zero equals pi/2. This will ensure zero appears at the top of the resultant arrow plots.

Step 3: Plot the chart

```
# create a 2 by 3 area, each of the six charts is plotted on one area
par(mfrow=c(2,3))
par(bg="cornsilk")# set the background color to cornsilk
# probability plots of each group of ants
pp.plot(ant.1,col="darkred",pch=19,main="Ant group 1",)
pp.plot(ant.2,col="darkgreen",pch=19,main="Ant group 2")
pp.plot(ant.3,col="darkblue",pch=19,main="Ant group 3")
#cumulative probability plots of each group of ants
plot.edf(ant.1,col="darkred",pch=19,main="Ant group 1", ylab="Probability",
xlab="value")
plot.edf (ant.2,col="darkgreen",pch=19,main="Ant group 2",
ylab="Probability", xlab="value")
plot.edf (ant.3,col="darkblue",pch=19,main="Ant group 3", ylab="Probability",
xlab="value")
```

Visualization quick tips

Tip 1: The pp.plot and plot.edf function require data in a circular format. This means the data needs to be converted to circular class. When using your own data or other datasets you can use the statement circular (your_data) to coerce data into a circular format.

Tip 2: You can include axes on each chart by setting the argument axes=TRUE in the plot function.

Tip 3: Change the background color by setting the argument bg="your chosen color".

Tip 4: Change the plot region using par(mfrow=c(x,y)), where x is the number of rows, and y is the number of columns. For example, par(mfrow=c(2,2)) would split the graphic output into four areas and then you could plot a chart to each specific area.

Quick reference R code

```
install.packages("circular")
require(circular)
ant.1<-circular(fisherB10$set1,zero=pi/2)
ant.2<-circular(fisherB10$set2,zero=pi/2)
ant.3<-circular(fisherB10$set3,zero=pi/2)
par(mfrow=c(2,3))
par(bg="cornsilk")
pp.plot(ant.1,col="darkred",pch=19,main="Ant group 1",)
pp.plot(ant.2,col="darkgreen",pch=19,main="Ant group 2")
pp.plot(ant.3,col="darkblue",pch=19,main="Ant group 3")
plot.edf(ant.1,col="darkred",pch=19,main="Ant group 1", ylab="Probability",
xlab="value")
plot.edf (ant.2,col="darkgreen",pch=19,main="Ant group 2",
ylab="Probability", xlab="value")
plot.edf (ant.3,col="darkblue",pch=19,main="Ant group 3", ylab="Probability",
xlab="value")
```

NOTES

Write your notes below…

50. Wind - pollution stacked bar time series chart

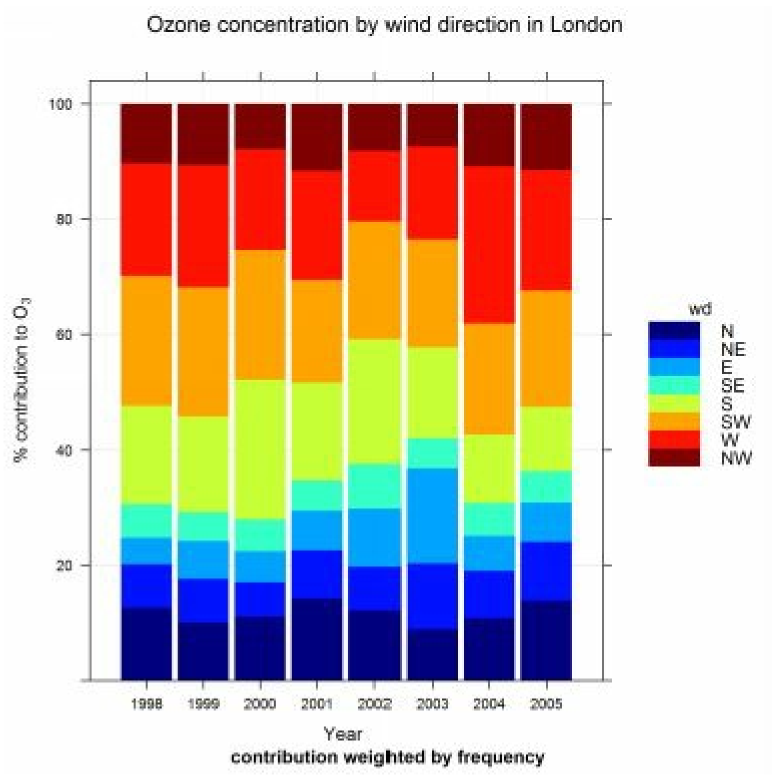


Figure 50 1 Annual ozone concentration by wind direction taken in London

How to read the chart

The chart displays the annual (median) distribution of ozone concentration in central London by direction of the prevailing wind over the years 1998 to 2005. The key on the right side of the graph gives the color codes for wind direction (wd on chart), with dark blue representing "North".

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

install.packages("openair")# this package will draw the plot
require(openair)

Step 2: Clean and prepare the data

data(mydata)#dataframe contains the data on ozone concentrations

Step 3: Plot the chart

```
# text for the title
title="Ozone concentration by wind direction in London"
timeProp(mydata, pollutant="o3", avg.time="year", proportion="wd",normalise
= TRUE ,cols="jet", box.width=.9, key.position="right", statistic =
"frequency",xlab= "Year",main=title,cex.main=1.3)
```

Visualization quick tips

Tip 1: Change the color of the stacked bars by using the argument cols= "my color". Quick and easy alternatives include “default”, “increment” and “heat”.

Tip 2: Set the argument normalise = FALSE to plot the actual frequencies in place of the proportions.

Tip 3: The argument box.width determines the size of the gap between colored bars. It takes a default value of 1 (no gap). Values less than one increase the gap/space between the colored bars.

Tip 4: The argument key.position determines the location of the legend. It can take values "right", "left", "top", "bottom".

Quick reference R code

```
install.packages("openair")
require(openair)
data(mydata)
title="Ozone concentration by wind direction in London"
timeProp(mydata, pollutant="o3", avg.time="year", proportion="wd",normalise
= TRUE ,cols="jet",box.width=.9,key.position="right", statistic =
"frequency",xlab= "Year",main=title,cex.main=1.3)
```

NOTES

Write your notes below…

51. Environmental style time series summary plot

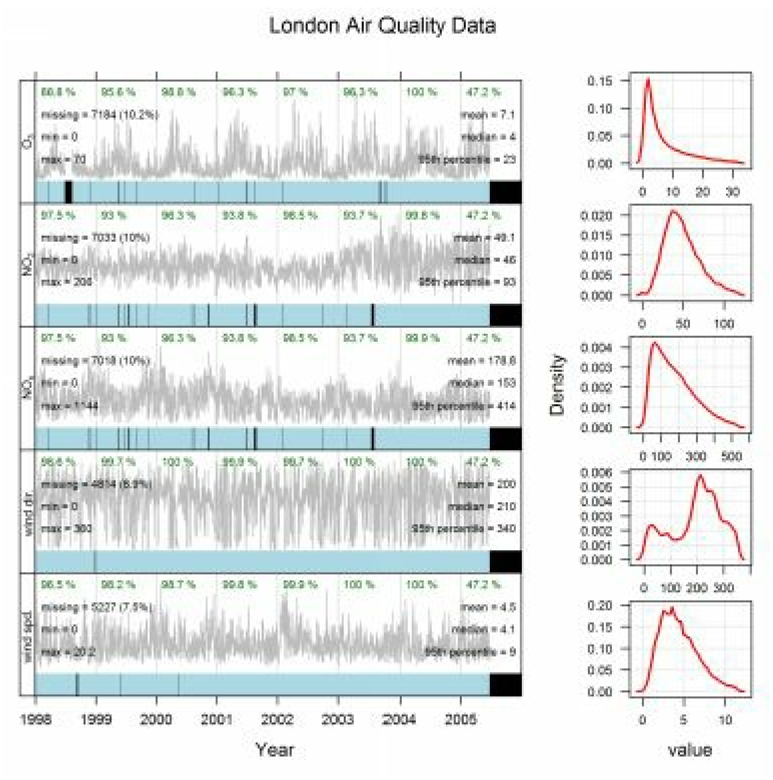


Figure 51 1 Summary plot of air quality data

How to read the chart

The top panel displays ozone concentration (grey line) in central London over the years 1998 to 2005. The cyan bar directly underneath the ozone time series indicates the presence (cyan) or absence (black) of data at a particular point in time. The box to the right of the top panel presents the density curve (red) of ozone. The remaining four panels (nitrogen dioxide concentration, Oxides of nitrogen concentration, wind direction and wind speed) have a similar interpretation.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("openair")# this package will draw the plot
require(openair)
```

Step 2: Clean and prepare the data

```
data(mydata)#dataframe contains the data on ozone concentrations
temp<-mydata
data <- temp[,c(1:6)] # select the data used in chart
```

Step 3: Plot the chart

```
# text for the title
title="London Air Quality Data"
summaryPlot(data,type="density",col.hist="red",col.trend="grey",col.data="ligh
```

Visualization quick tips

Tip 1: Change the color of the line used to draw the density, trend pollution values and present and missing data by setting to "my color" the arguments col.hist,col.trend,col.data and col.mis respectively.

Tip 2: For greater clarity plot fewer time series. This can be achieved by using data <- temp[,c(1:number)], where number is equal to the number of time series you wish to observe. You can plot all of the data in myframe by setting data<-myframe in the above R code.

Tip 3: To use default labels for the x-axis omit the argument xlab. To add custom text to the y-axis use the argument ylab="my text".

Quick reference R code

```
install.packages("openair")
require(openair)
data(mydata)#dataframe contains the data on ozone concentrations
temp<-mydata
data <- temp[,c(1:6)]
title="London Air Quality Data"
summaryPlot(data,type="density",col.hist="red",col.trend="grey",col.data="ligh
```

NOTES

Write your notes below…

52. Bumps chart

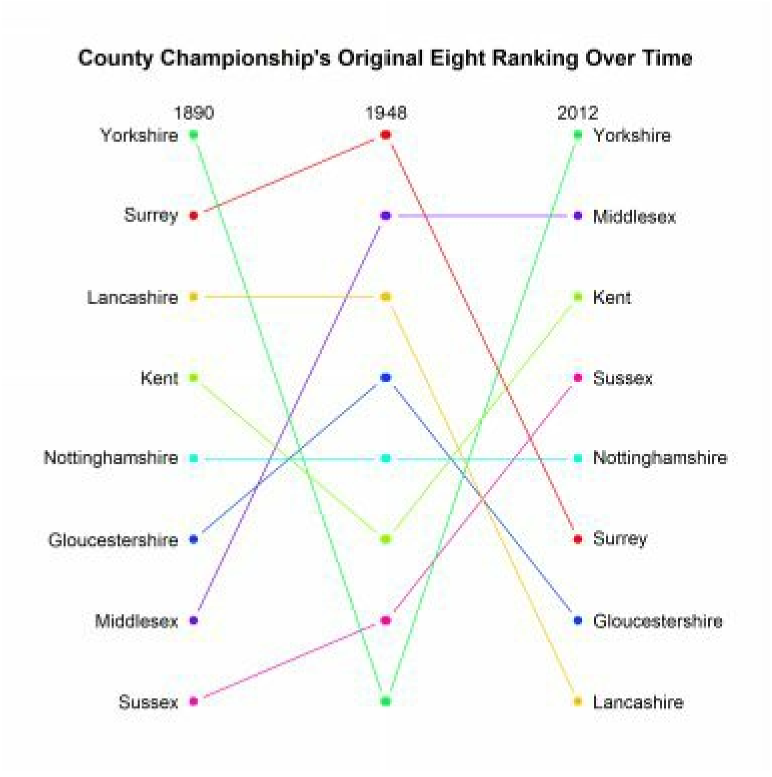


Figure 52 1 Rank position of original county cricket teams over time

How to read the chart

The chart displays the eight counties that took part in the first English cricket country championships beginning in 1890. Surrey won the first championship and are the first team listed under 1890. Sussex scored the lowest number of points in that first season and are therefore placed last for 1890. The lines from each team represent their relative ranking in both 1948 and 2012 using total points earned.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("plotrix")# package will draw plot
require(plotrix)
```

Step 2: Clean and prepare the data

```
#create a matrix of league scores for 1890,1948 and 2012
points<-matrix(c(6,168,139,4,152,106,3,76,170,31,56,194,0,80,163,-
1,128,126,-5,160,172,-10,72,167),ncol=3,byrow=TRUE)
```

```
#add row names
rownames (points) <-c("Surrey","Lancashire","Kent","Yorkshire", "Nottinghamshire","Gloucestershire","Middlesex","Sussex")
#add dates to columns
colnames(points)<-c("1890","1948","2012")
```
Step 3: Plot the chart
```
par(bg="white")# set the background color
bumpchart(points,rank=TRUE,col=rainbow(8),main="County Championship's Original Eight Ranking Over Time")
```

Visualization quick tips

Tip 1: The color of the lines was set using col=rainbow(8), you can also specify the colors of your choice directly by using col=c("your first color","your second color"…,"your final color"). For example, try col=c("red","green","yellow","purple","blue","orange","darkred","white")

Tip 2: The argument rank=TRUE ranks the values before plotting. To turn this option off use rank =FALSE.

Tip 3: Change the background color by setting the argument bg="your chosen color".

Tip 4: The first two arguments in the legend function, determine the horizontal and vertical location of the legend. An alternative, which works in many instances is to specify the location such as legend("topleft",…). You can also specify "topright", "bottomright", "bottomleft".

Quick reference R code

```
install.packages("plotrix")# package will draw plot
require(plotrix)
```

```
points<-matrix(c(6,168,139,4,152,106,3,76,170,31,56,194,0,80,163,-
1,128,126,-5,160,172,-10,72,167),ncol=3,byrow=TRUE)
rownames	(points)	<-c("Surrey","Lancashire","Kent","Yorkshire",
"Nottinghamshire","Gloucestershire","Middlesex","Sussex")
colnames(points)<-c("1890","1948","2012")
par(bg="white")
bumpchart(points,rank=TRUE,col=rainbow(8),main="County Championship's
Original Eight Ranking Over Time")
```

NOTES

Write your notes below…

53. Two ordinate time series stack plot

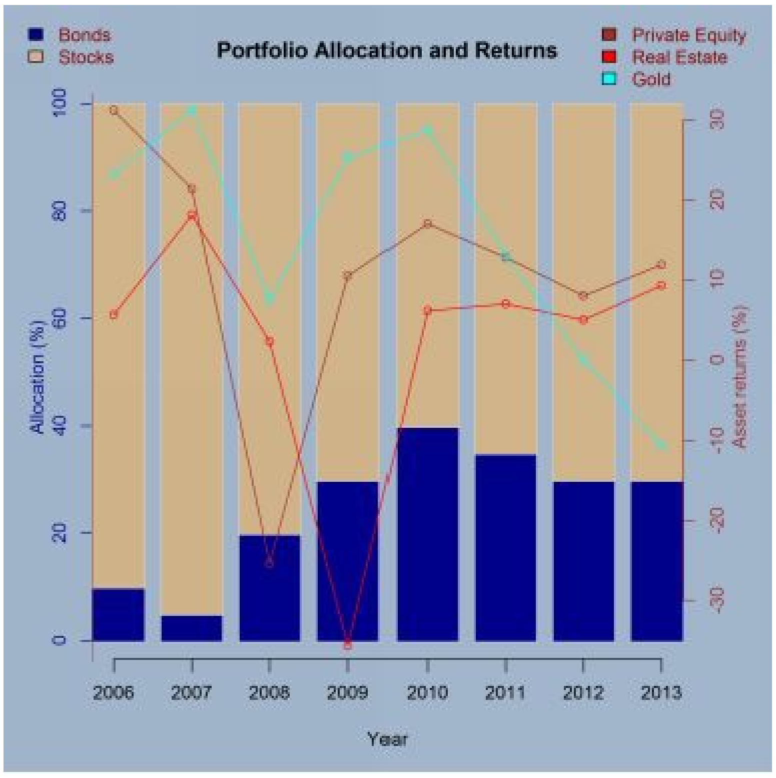


Figure 53 1 Stock – bond allocation with asset class returns

How to read the chart

The chart displays the allocation between bonds (blue) and stocks (tan) for a portfolio over the years 2006 to 2013. The left axis provides the measurements for the allocation. Overlaid on the plot are the annual returns for private equity, real estate and gold with return values measured on the right axis.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("plotrix")# package will draw plot
require(plotrix)
```

Step 2: Clean and prepare the data

```
#enter stock and bond allocation
stock.percent =c(90,95,80,70,60,65,70,70)
bond.percent =100-stock.percent
# enter asset class returns
private_equity=c(31.2,21.4,-25.4,10.5,17,12.8,8,11.9)
```

```
real_estate=c(5.6,18.1,2.3,-35.7,6.1,7,5,9.3)
gold=c(23.2,31.3,7.5,25.3,28.7,13,0,-10.7)
#create a variable to count time for the basic plot
time<-0:7
# combine stocks and bond allocation
allocation <-cbind(bond.percent,stock.percent)
#combine asset class returns
return<-cbind(private_equity, real_estate, gold)
```

Step 3: Plot the chart

```
#set background and axis colors
par(bg=" lightsteelblue3",col=" lightsteelblue3", col.axis=" lightsteelblue3")
#create the basic plot
twoord.stackplot(lx=time, rx=time, ldata=allocation,
            rdata=return,  lcol=c("darkblue", "tan"),
            rcol=c("brown", "red","cyan"), ltype="bar", rtype=c("o","o","o"),
            lylab="Allocation (%)", rylab="Asset returns (%)", xlab="Year",
            incrylim=-2/100,main="Portfolio Allocation and Returns")
par(bg=" lightsteelblue3",col="black", col.axis="black")
#create fancy axis
axis(side=1,at=(0:7),                                          labels=
c("2006","2007","2008","2009","2010","2011","2012","2013"))

par(xpd=TRUE) #extend the area of plotting
par(new=TRUE) #to add new graph "layers"
plot(0:1, 0:1, type="n", xlab="",ylab="", axes=FALSE) #redo the (x,y) range

#asset allocation legend
legend(-0.18, 1.2, inset=.05,
bty="n", cex=1, text.col ="darkred",
c("Bonds", "Stocks"), fill=c("darkblue", "tan"))

#asset classes legend
```

```
legend(0.8, 1.2, inset=.05,
bty="n", cex=1, text.col ="darkred",
c("Private Equity", "Real Estate", "Gold"), fill= c("brown", "red","cyan"))

par(xpd=FALSE, new=FALSE) #default setting
```

Visualization quick tips

Tip 1: Both the ltype and rtype arguments can take various values. The most frequently used are "p" for points, "l" for lines,"b" for both, "c" for the lines part alone of "b", "o" for both 'overplotted', "h","s" for stair steps, "S" for other steps, and "n" for no plotting.

Tip 2: You can change the right hand axis color specifying it as the first value in the rcol argument i.e. rcol =c("my color","next color"…). Similarly, you can change the left hand axis color using the same approach.

Tip 3: Change the background color by setting the argument bg="your chosen color".

Tip 4: The first two argument in the legend function, determine the horizontal and vertical location of the legend. An alternative, which works in many instances is to specify the location such as legend("topleft",…). You can also specify "topright", "bottomright", "bottomleft".

Quick reference R code

```
install.packages("plotrix")
require(plotrix)

stock.percent =c(90,95,80,70,60,65,70,70)
bond.percent =100-stock.percent
private_equity=c(31.2,21.4,-25.4,10.5,17,12.8,8,11.9)
real_estate=c(5.6,18.1,2.3,-35.7,6.1,7,5,9.3)
```

```
gold=c(23.2,31.3,7.5,25.3,28.7,13,0,-10.7)

time<-0:7

allocation <-cbind(bond.percent,stock.percent)

return<-cbind(private_equity, real_estate, gold)

par(bg=" lightsteelblue3",col=" lightsteelblue3", col.axis=" lightsteelblue3")
twoord.stackplot(lx=time, rx=time, ldata=allocation,
            rdata=return,  lcol=c("darkblue", "tan"),
            rcol=c("brown", "red","cyan"), ltype="bar", rtype=c("o","o","o"),
            lylab="Allocation (%)", rylab="Asset returns (%)", xlab="Year",
            incrylim=-2/100,main="Portfolio Allocation and Returns")
par(bg=" lightsteelblue3",col="black", col.axis="black")

axis(side=1,at=(0:7),                                          labels=
c("2006","2007","2008","2009","2010","2011","2012","2013"))

par(xpd=TRUE) #extend the area of plotting
par(new=TRUE) #to add new graph "layers"
plot(0:1, 0:1, type="n", xlab="",ylab="", axes=FALSE) #redo the (x,y) range
legend(-0.18, 1.2, inset=.05,
bty="n", cex=1, text.col ="darkred",
c("Bonds", "Stocks"), fill=c("darkblue", "tan"))
legend(0.8, 1.2, inset=.05,
bty="n", cex=1, text.col ="darkred",
c("Private Equity", "Real Estate", "Gold"), fill= c("brown", "red","cyan"))
par(xpd=FALSE, new=FALSE) #default setting
```

NOTES

Write your notes below…

54. Two ordinate time series plot

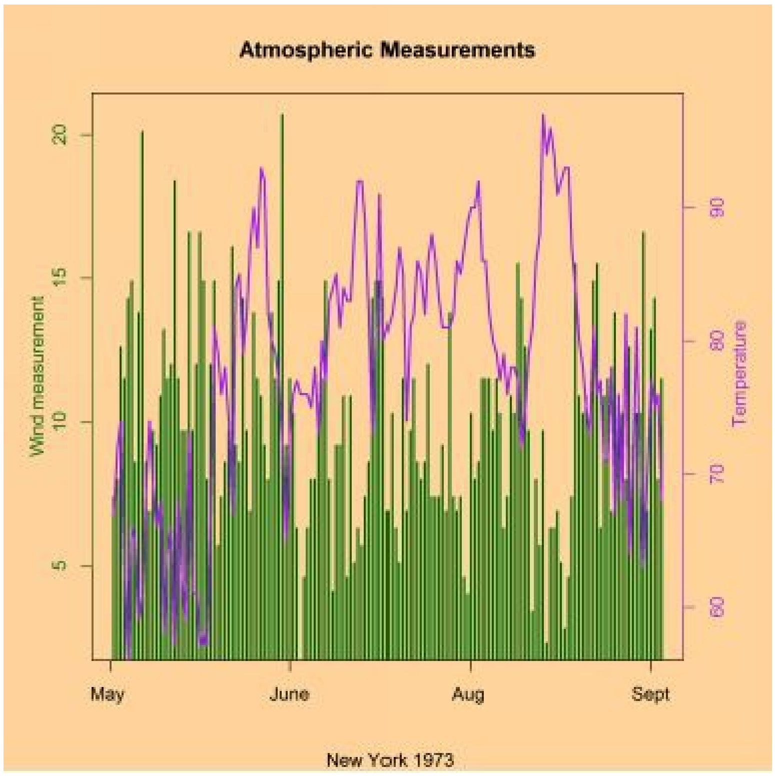


Figure 54 1 Wind and temperature measurements taken in New York city

How to read the chart

The chart displays wind (solid green) and air temperature (purple line) taken in New York during 1973. The left axis provides the measurements for wind, and the right axis the measurements for temperature.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("plotrix")# package will draw plot
require(plotrix)
attach(airquality)#contains the dataset
```

Step 2: Clean and prepare the data

```
x=airquality[,3]#Wind measurement
y= airquality[,4]#temperature
seq<-1:153#used for x axis
labels<-c("May ","June","Aug ","Sept")# dates plotted on image
```

Step 3: Plot the chart

```
par(bg="burlywood1")#set background color
twoord.plot(seq,x,y,type=c("h","l"),rcol="purple",lcol="darkgreen",lwd=2,ylab=
"Wind measurement",rylab=" Temperature",main= "Atmospheric
Measurements",xticklab=labels,xtickpos =c(0,50,100,150),sub=" New York
1973")
```

Visualization quick tips

Tip 1: Change type argument to draw different lines. It takes a range of values. The most frequently used are "p" for points, "l" for lines,"b" for both, "c" for the lines part alone of "b", "o" for both 'overplotted', "h","s" for stair steps, "S" for other steps, and "n" for no plotting.

Tip 2: You can change the right hand axis color by using the argument rcol="my color". Similarly, you can change the left hand axis color by lcol="my color"

Tip 3: Change the background color by setting the argument bg="your chosen color".

Quick reference R code

```
install.packages("plotrix")
require(plotrix)
attach(airquality)
x=airquality[,3]#Wind measurement
y= airquality[,4]#temperature
seq<-1:153
labels<-c("May ","June","Aug ","Sept")
par(bg="burlywood1")
twoord.plot(seq,x,y,type=c("h","l"),rcol="purple",lcol="darkgreen",lwd=2,ylab=
"Wind measurement",rylab=" Temperature",main= "Atmospheric
Measurements",xticklab=labels,xtickpos=c(0,50,100,150),sub=" New York
1973")
```

NOTES

Write your notes below…

55. Chernoff faces

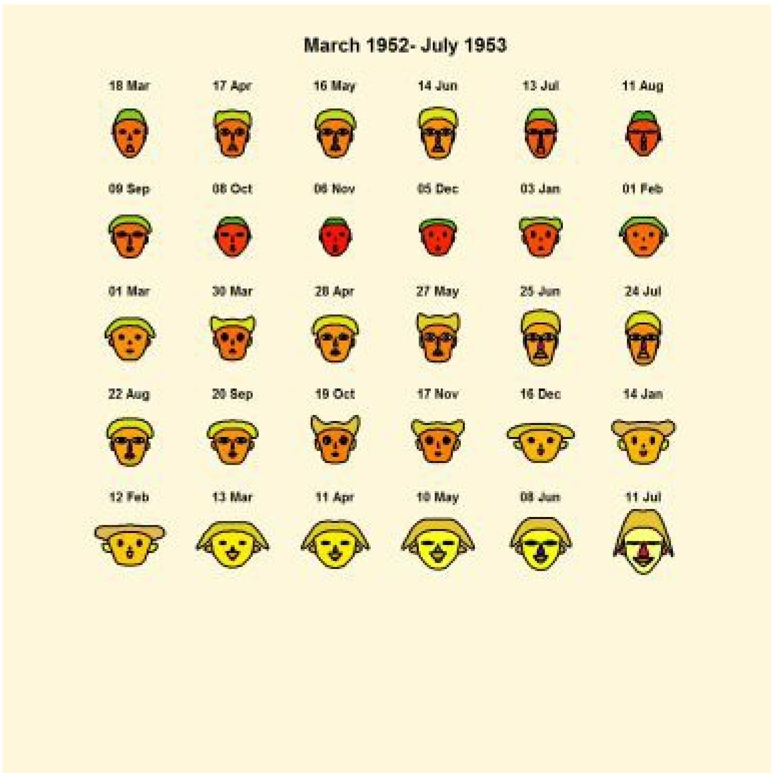


Figure 55 1 Ice cream consumption in the United States

How to read the chart

The image displays Chernoff faces calculated from four variables, consumption of ice cream per head (in pints); average family income per week (in US Dollars); price of ice cream (per pint) and the average temperature (in Fahrenheit). The variables used to define the face were height of face, width of face, structure of face, height of mouth, width of mouth , degree of smiling, height of eyes, width of eyes, height of hair, width of hair, style of hair, height of nose, width of nose, width of ear and height of ear. The value for the height of face was determined from the consumption of ice cream, the value for the structure of face was determined from average family income and so on repeating for the fifteen facial characteristics that define a Chernoff face.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("aplpack")#used to draw image
require("aplpack")
install.packages("Ecdat")# contains the data
```

```
require(Ecdat)
```

Step 2: Clean and prepare the data

```
data(Icecream )#load the date
#add the dates
dates<-c("18 Mar","17 Apr","16 May","14 Jun","13 Jul","11 Aug","09
Sep","08 Oct","06 Nov","05 Dec","03 Jan","01 Feb","01 Mar","30 Mar","28
Apr","27 May","25 Jun","24 Jul","22 Aug","20 Sep","19 Oct","17 Nov","16
Dec","14 Jan","12 Feb","13 Mar","11 Apr","10 May","08 Jun","11 Jul")
```

Step 3: Plot the chart

```
par(bg="cornsilk")# set background color to cornsilk

faces(Icecream,main="March 1952- July 1953",label=dates, face.type=1)
```

Visualization quick tips

Tip 1: To plot outline faces without color set face.type=0. You can also draw Santa Claus faces by setting face.type=2.

Tip 2: Drop the argument labels if you don’t want to have dates above the faces. The dates will be replaced with numbers.

Tip 3: Change the background color by setting the argument bg="your chosen color".

Quick reference R code

```
install.packages("aplpack")
require("aplpack")
install.packages("Ecdat")
require(Ecdat)
data(Icecream )
```

```
dates<-c("18 Mar","17 Apr","16 May","14 Jun","13 Jul","11 Aug","09
Sep","08 Oct","06 Nov","05 Dec","03 Jan","01 Feb","01 Mar","30 Mar","28
Apr","27 May","25 Jun","24 Jul","22 Aug","20 Sep","19 Oct","17 Nov","16
Dec","14 Jan","12 Feb","13 Mar","11 Apr","10 May","08 Jun","11 Jul")
par(bg="cornsilk")
faces(Icecream,main="March 1952- July 1953",label=dates, face.type=1)
```

NOTES

Write your notes below…

56. Textual analysis Chernoff face plot

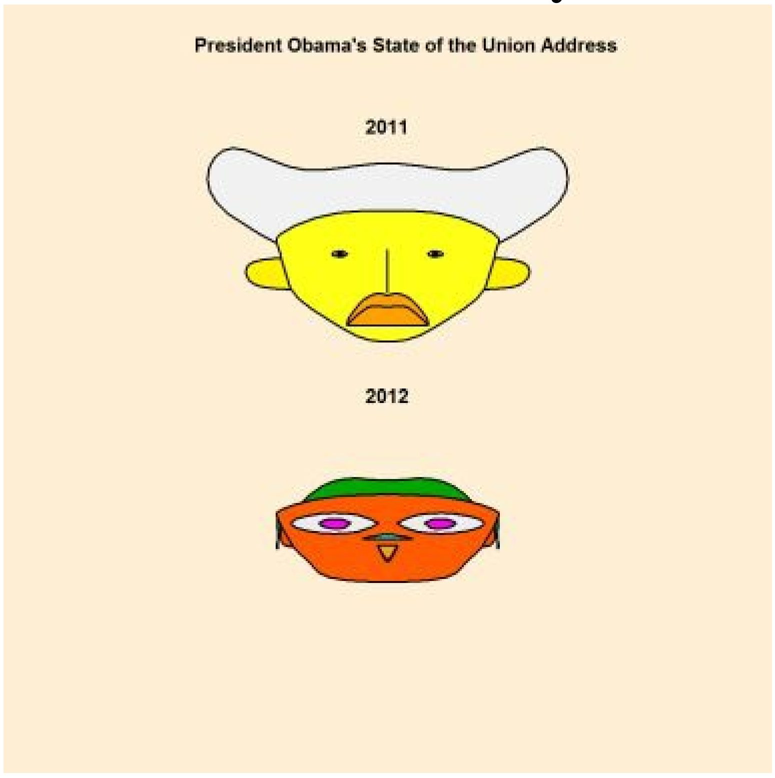


Figure 56 1 State of the Union address using words to define Chernoff faces

How to read the chart

The image displays Chernoff faces calculated from the most frequent words in President Obama’s 2011 and 2012 State of the Union address. The variables used to define the face were height of face, width of face, structure of face, height of mouth, width of mouth , degree of smiling, height of eyes, width of eyes, height of hair, width of hair, style of hair, height of nose, width of nose, width of ear and height of ear. The value for the height of face was determined from the most common word (“America”) , the value for the structure of face was determined by the next most popular word (“People”) and so on.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("aplpack")
require("aplpack")
```

Step 2: Clean and prepare the data

```
# Create variables with count of most popular words
```

```
c.2010<- c(70,33,29,29,20,13,12,11,13,15,13,15,6,11,10)
c.2011<-c(57,33,31,20,14,18,16,16,12,9,10,7,15,9,10)
words<-rbind(c.2010,c.2011)
#Add column names for most common words
colnames(words) <- c("America","People","Jobs","Business", "Time","Government","Nation", "Tonight","Country","Energy", "Taxes","Economy","Futurev","Care","Congress")
rownames(words)<-c("2011","2012")# add year 2011 and 2012
```

Step 3: Plot the chart

```
par(bg="papayawhip")# set the background color
faces(words,face.type=1,main="President Obama's State of the Union Address",ncol.plot=1)
```

Visualization quick tips

Tip 1: To plot outline faces without color set face.type=0. You can also draw Santa Claus faces by setting face.type=2.

Tip 2: Replace ncol.plot with nrow.plot if you wish to plot the faces as rows rather than in a column.

Tip 3: Change the background color by setting the argument bg="your chosen color".

Quick reference R code

```
install.packages("aplpack")
require("aplpack")
c.2010<- c(70,33,29,29,20,13,12,11,13,15,13,15,6,11,10)
c.2011<-c(57,33,31,20,14,18,16,16,12,9,10,7,15,9,10)
words<-rbind(c.2010,c.2011)
colnames(words) <- c("America","People","Jobs","Business", "Time","Government","Nation", "Tonight","Country","Energy",
"Taxes","Economy","Futurev","Care","Congress")
rownames(words)<-c("2011","2012")
par(bg="papayawhip")
faces(words,face.type=1,main="President Obama's State of the Union
Address",ncol.plot=1)
```

NOTES

Write your notes below…

57. Chernoff face time series plot

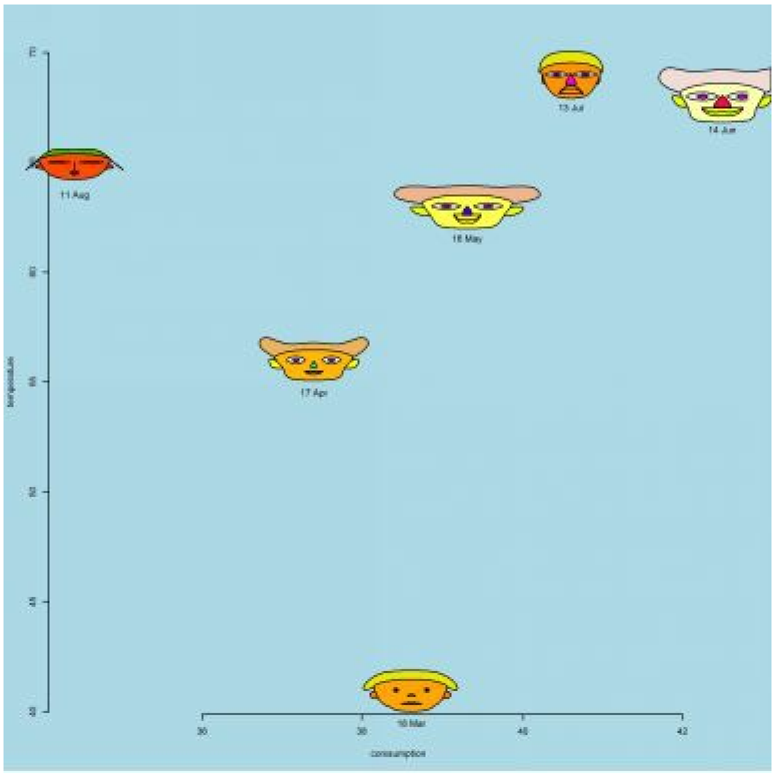


Figure 57 1 Ice cream consumption verses temperature

How to read the chart

The image displays a time series plot of ice cream consumption and temperature using Chernoff faces. The Chernoff faces are calculated from four variables, consumption of ice cream per head (in pints); average family income per week (in US Dollars); price of ice cream (per pint) and the average temperature (in Fahrenheit). The variables used to define the face were height of face, width of face, structure of face, height of mouth, width of mouth , degree of smiling, height of eyes, width of eyes, height of hair, width of hair, style of hair, height of nose, width of nose, width of ear and height of ear. The value for the height of face was determined from the consumption of ice cream, the value for the structure of face was determined from average family income and so on repeating for the fifteen facial characteristics that define a Chernoff face.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("aplpack") #package used to draw image
require("aplpack")
```

```
install.packages("Ecdat") # contains the data
require(Ecdat)
```

Step 2: Clean and prepare the data

```
data(Icecream )# data used in plot
b<- Icecream[1:6,]# We only want first 6 observations
#attach dates to the data
rownames(b)<-c <-c("18 Mar","17 Apr","16 May","14 Jun","13 Jul","11 Aug")
consumption= b[1:6,1] *100
temperature = b[1:6,4]
```

Step 3: Plot the chart

```
#first set up the basic plot area using x and y axis
plot(consumption, temperature,bty="n")
par(bg="lightblue")# set background color
b<-faces(b,plot=FALSE) # create face data frame
plot.faces(b, consumption, temperature,width=2, height=3)
```

Visualization quick tips

Tip 1: To plot outline faces without color set face.type=0. You can also draw Santa Claus faces by setting face.type=2.

Tip 2: Adjust the width and height of the faces in the plot by using the argument width and height. It may take some experimentation to get the size correct when using your own data.

Tip 3: Change the background color by setting the argument bg="your chosen color".

Quick reference R code

```
install.packages("aplpack")

require("aplpack")

install.packages("Ecdat")
```

```
require(Ecdat)
data(Icecream )
b<- Icecream[1:6,]
rownames(b)<-c <-c("18 Mar","17 Apr","16 May","14 Jun","13 Jul","11 Aug")
consumption= b[1:6,1] *100
temperature = b[1:6,4]
plot(consumption, temperature ,bty="n")
par(bg="lightblue")
b<-faces(b,plot=FALSE)
plot.faces(b, consumption, temperature,width=2, height=3)
```

NOTES

Write your notes below…

58. Chernoff face rank order plot

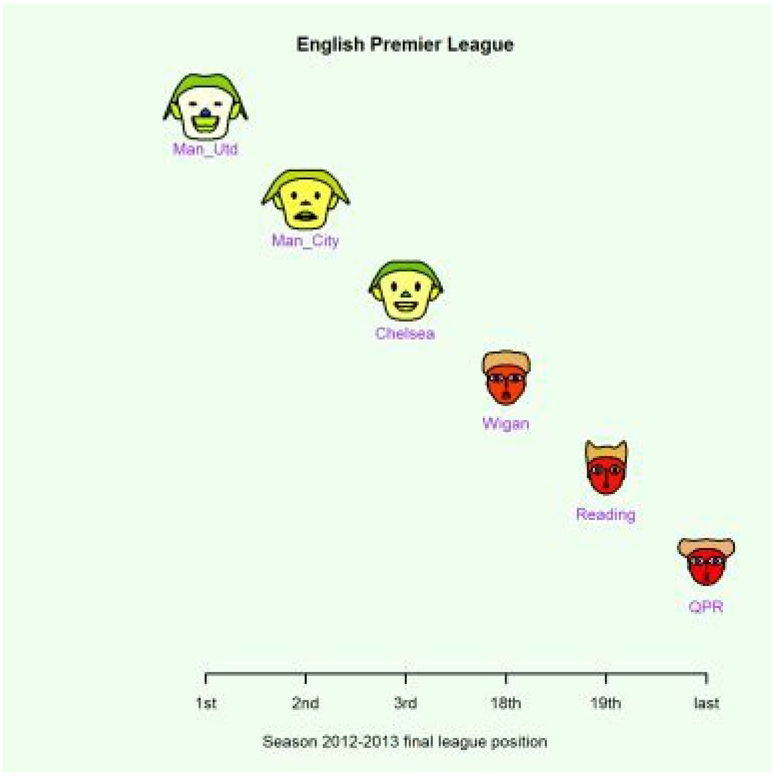


Figure 58 1 Final position of top six and bottom six English football teams

How to- read the chart

The image displays the Chernoff faces of the top three and bottom three English premier league football teams for the season 2012-2013. The Chernoff faces are calculated from the following variables:

1	"height of face "= "Points"
2	"width of face " ="Goal Difference"
3	"structure of face"= "Home wins"
4	"height of mouth " = "Home for"
5	"width of mouth " = "Away wins"
6	"smiling " = "Away for"
7	"height of eyes "= "Home draw"
8	"width of eyes " = "Home loss"
9	"height of hair " = "Home against"
10	"width of hair " = "Away draw"
11	"style of hair " = "Away loss"

12 "height of nose " = "Away against"

13 "width of nose " = "Points"

14 "width of ear " = "Goal Difference"

15 "height of ear " = "Home wins"

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("aplpack")# package used to draw plot
require("aplpack")
```

Step 2: Clean and prepare the data

```
#enter the data for each team
Man_Utd =c(89, 43, 16, 45,
12, 41, 0, 3, 19, 5, 2,
24)
Man_City=c( 78, 32, 14, 41,
9, 25, 3, 2, 15, 6, 4,
19)
Chelsea =c(75, 36, 12, 41, 10,
34, 5, 2, 16, 4, 5, 23)
Wigan=c( 36, -26, 4, 26, 5,
21, 6, 9, 39, 3, 11, 34)
Reading=c (28, -30, 4, 23, 2,
20, 8, 7, 33, 2, 15, 40)
QPR=c( 25, -30, 2, 13, 2,
17, 8, 9, 28, 5, 12, 32)
#combine teams together
names<-rbind(Man_Utd,Man_City,Chelsea,Wigan,Reading,QPR)
#add column names
colnames(names)<-c("Points", "Goal Difference", "Home
wins", "Home for", "Away wins", "Away for",
```

"Home draw", "Home loss", "Home against", "Away draw", "Away loss", "Away against")

Step 3: Plot the chart

#convert names to a face object for plotting
teams<-faces(names,plot.faces=FALSE)
#set background color and color for text associated with faces
par(bg=" honeydew1",col="purple")

#create basic plot area
plot(0:6,0:6,type="n",axes=FALSE,xlab="Season 2012-2013 final league position",ylab="",main="English Premier League")
#set up the custom axis
axis(side=1,at=(1:6), labels=c("1st","2nd","3rd","18th","19th","last"))
#plot the data
plot(teams,x.pos=1:6,y.pos=6:1)

Visualization quick tips

Tip 1: To plot outline faces without color set face.type=0. You can also draw Santa Claus faces by setting face.type=2.

Tip 2: Adjust the width and height of the faces in the plot by using the argument width and height. It may take some experimentation to get the size correct when using your own data.

Tip 3: Change the background color by setting the argument bg="your chosen color".

Quick reference R code

install.packages("aplpack")

require("aplpack")

Man_Utd =c(89, 43, 16, 45,

```
12,        41,        0,        3,        19,        5,        2,
24)

Man_City=c(        78,        32,        14,        41,
9,        25,        3,        2,        15,        6,        4,
19)

Chelsea        =c(75,        36,        12,        41,        10,
34,        5,        2,        16,        4,        5,        23)

Wigan=c(        36,        -26,        4,        26,        5,
21,        6,        9,        39,        3,        11,        34)

Reading=c        (28,        -30,        4,        23,        2,
20,        8,        7,        33,        2,        15,        40)

QPR=c(        25,        -30,        2,        13,        2,
17,        8,        9,        28,        5,        12,        32)

names<-rbind(Man_Utd,Man_City,Chelsea,Wigan,Reading,QPR)

colnames(names)<-c("Points",        "Goal Difference",        "Home
wins",        "Home for",        "Away wins",        "Away for",
"Home draw",        "Home loss",        "Home against",        "Away
draw",        "Away loss",        "Away against")

teams<-faces(names,plot.faces=FALSE)

par(bg=" honeydew1",col="purple")

plot(0:6,0:6,type="n",axes=FALSE,xlab="Season 2012-2013 final league
position",ylab="",main="English Premier League")

axis(side=1,at=(1:6), labels=c("1st","2nd","3rd","18th","19th","last"))

plot(teams,x.pos=1:6,y.pos=6:1)
```

NOTES

Write your notes below…

59. Constant feature Chernoff faces

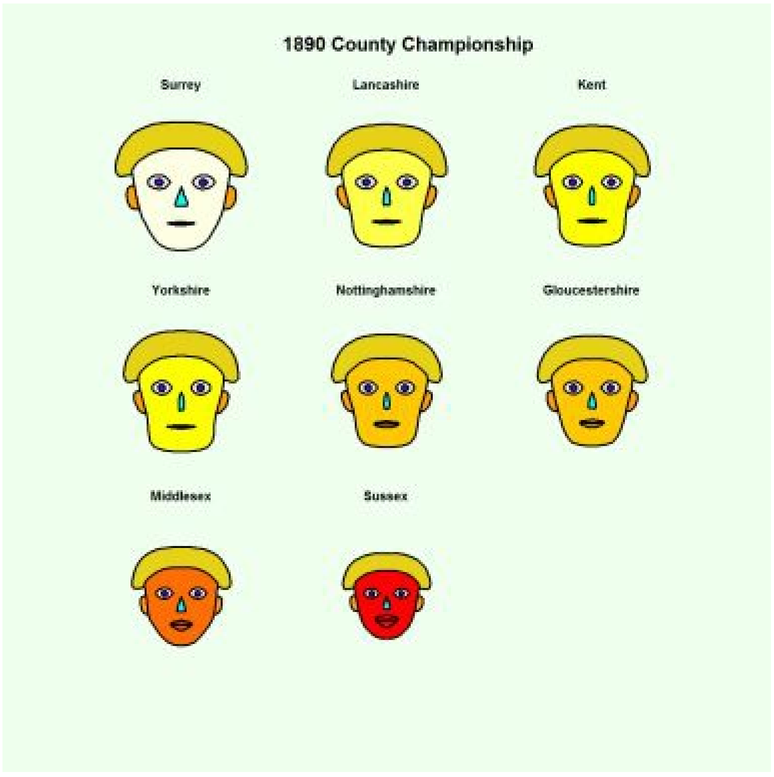


Figure 59 1 Cricket teams playing in the 1890 County Championship

How to read the chart

The image displays constant Chernoff faces calculated from the points scored, number of wins, losses and draws for teams playing in the first English County Championship. The images are ranked from division winners (Surrey) to last place (Sussex).The Chernoff faces are calculated from the following variables:

1 "height of face " = "Points"

2 "width of face "= "Wins"

3 "structure of face" = "Draws"

4 "height of mouth " = "Losses"

The remaining eleven Chernoff face features are given a constant value.

How to create this chart in less than ten minutes

Step 1: Download and install the required packages.

```
install.packages("aplpack")# package used to draw plot
require("aplpack")
```

Step 2: Clean and prepare the data

```
#enter the data for each team
Surrey= c (          6,          9,          2,          3)
Lancashire= c(          4,          7,          4,          3)
Kent          = c(3,          6,          5,          3)
Yorkshire= c (          3,          6,          5,          3)
Nottinghamshire= c (          0,          5,          4,          5)
Gloucestershire= c(          -1,          5,          3,          6)
Middlesex=c (          -5,          3,          1,          8)
Sussex          = c (-10,          1,          0,          11)
#combine teams together
names<-
rbind(Surrey,Lancashire,Kent,Yorkshire,Nottinghamshire,Gloucestershire,Middl
#add column names
colnames(names)<-c("Points",          "Wins",          "Draws",
"Losses")
#create a matrix of constants for the unchanging face features
constant <- matrix(1,nrow=nrow(names),ncol=11)
#combine data on points, wins, draws, losses with constant data
names<-cbind(names,constant)
```

Step 3: Plot the chart

```
par(bg=" honeydew1")#set background color
faces (names, main=" 1890 County Championship", face.type=1)
```

Visualization quick tips

Tip 1: To plot outline faces without color set face.type=0. You can also draw Santa Claus faces by setting face.type=2.

Tip 2: Adjust the width and height of the faces in the plot by using the argument width and height. It may take some experimentation to get the size correct when using your own data.

Tip 3: Change the background color by setting the argument bg="your chosen color".

Quick reference R code

```
install.packages("aplpack")
require("aplpack")
Surrey= c (          6,          9,          2,          3)
Lancashire= c(          4,          7,          4,          3)
Kent          = c(3,          6,          5,          3)
Yorkshire= c (          3,          6,          5,          3)
Nottinghamshire= c (          0,          5,          4,          5)
Gloucestershire= c(          -1,          5,          3,          6)
Middlesex=c (          -5,          3,          1,          8)
Sussex          = c (-10,          1,          0,          11)

names<-rbind(          Surrey,Lancashire,Kent,Yorkshire,Nottinghamshire,
Gloucestershire,Middlesex,Sussex)
colnames(names)<-c("Points",          "Wins",          "Draws",
"Losses")
constant <- matrix(1,nrow=nrow(names),ncol=11)
names<-cbind(names,constant)
par(bg=" honeydew1")
faces (names, main=" 1890 County Championship", face.type=1)
```

NOTES

Write your notes below…

[i] This dataset is rather famous and known as Anscombe's quartet. See Anscombe, F. J. (1973). "Graphs in Statistical Analysis". American Statistician 27 (1): 17–21. JSTOR 2682899.

[ii] See page 206 in Galton, Francis. Natural inheritance. Vol. 42. MacmillaSn, 1889.

[iii] The height of the boxes are proportional to the Pearson residuals with the width proportional to the square roots of the expected frequencies.

[iv] The crime data consisted of the following crimes: Murder, Rape Robbery, Assault, Burglary, Larceny and Motor Theft.

[v] See http://dictyexpress.biolab.si/index.htm

[vi] See www. http://newsmap.jp/#/b,e,m,n,s,t,w/us/view/

[vii] See Gao X, An H, Zhong W (2013) Features of the Correlation Structure of Price Indices. PLoS ONE 8(4): e61091. doi:10.1371/journal.pone.0061091

[viii] Image taken from the website http://www.florence-nightingale-avenging-angel.co.uk/ . Also see the insightful book by Hugh Small entitled Florence Nightingale, Avenging Angel.