R Data Visualization Cookbook

Atmajitsinh Gohil

What this book will do for you...

- Generate various plots in R using the basic R plotting techniques
- Utilize R packages to add context and meaning to your data
- Design interactive visualizations and integrate them on your website or blog
- Communicate using visualization techniques, optimal for the underlying data being used as input
- Create presentations and learn the basics of creating apps in R for your audience
- Introduce users to basic R functions and data manipulation techniques while creating meaningful visualizations
- Add elements, text, animation, and colors to your plot to make sense of data

Inside the Cookbook...

- A straightforward and easy-to-follow format
- A selection of the most important tasks and problems
- Carefully organized instructions for solving the problem efficiently
- Clear explanations of what you did
- Apply the solution to other situations

R is an open source language for data analysis and graphics. It is platform-independent and allows users to load various packages as well as develop their own packages to interpret data better.

This book is packed with practical recipes, designed to provide you with all the guidance needed to get to grips with data visualization with R. It starts off with the basics of R plots and an introduction to heat maps and customizing them, before gradually taking you through creating interactive maps using the googleVis package, generating choropleth maps and contouring maps, bubble plots, and pie charts. You will then learn how to animate 2D and 3D plots in R. By the end of the book, you will be equipped with the key techniques to create impressive data visualizations with professional efficiency and precision.

Over 80 recipes to analyze data and create stunning visualizations with R


Atmajitsinh Gohil

7.50 x 9.25
225 mm x 191 mm

7.50 x 9.25
225 mm x 191 mm

Content Type: Black & White
Paper Type: White
Page Count: 292
File Type: InDesign
Request ID: CS1330826
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 7 'Data in Higher Dimensions'
- A synopsis of the book’s content
- More information on R Data Visualization Cookbook

About the Author

Atmajitsinh Gohil works as a senior consultant at a consultancy firm in New York City. After graduating, he worked in the financial industry as a Fixed Income Analyst. He writes about data manipulation, data exploration, visualization, and basic R plotting functions on his blog at http://datavisualizationineconomics.blogspot.com.

He has a master's degree in financial economics from the State University of New York (SUNY), Buffalo. He also graduated with a master of arts degree in economics from University of Pune, India. He loves to read blogs on data visualization and loves to go out on hikes in his free time.

This book would not have been possible without the help from numerous data visualizers and data scientists around the globe who bring into existence new and innovative ways to transform data into beautiful stories. I would like to sincerely thank the developers of R and R packages who have contributed so generously to the growing R open source community.

I would like to thank Jer Thorpe and Hans Rosling for their inspiring Ted videos on data visualization.

I would also like to thank all the economists and statisticians who have so often inspired me.

I would like to thank my publisher, Packt Publishing, for giving me the opportunity to work on this book. I would also like to thank all the technical reviewers and content development editors at Packt Publishing for their informative comments and suggestions.

Finally, I would like to thank my amazing family and magnificent friends for always encouraging and supporting me.
R Data Visualization Cookbook

Our ability to generate data has improved tremendously with the advent of technology. The data generated has become more complex with the passage of time. The complexity in data forces us to develop new tools and methods to analyze it, interpret it, and communicate with the data. Data visualization empowers us with the necessary skills required to convey the meaning of underlying data. Data visualization is a remarkable intersection of data, science, and art, and this makes it hard to define visualization in a formal way; a simple Google search will prove me right. The Merriam-Webster dictionary defines visualization as "formation of mental visual images". In reality, the term visualization goes beyond the limits of providing visual images by assisting humans in data recording, revealing pattern, exploration of data, and spreading information in a meaningful way.

Jer Thorpe in an interview with Mashable.com (http://mashable.com/2012/12/11/data-visualization-jer-thorp/) introduces the idea of humanizing data:

"...And I think that there's a huge possibility for humans, society as a whole—if we could share that data more usefully, for science and for the construction of cities, and for all these kinds of things, then it becomes much more useful. So in my work, I'm really thinking about how we can give people glimpses into that type of future. Giving people an opportunity to think about data ownership or giving people a visualization so that they can see the kinds of things that can be done with data".

R is an open source platform used to analyze data. It has been widely used as a statistical tool in the past. An individual does not necessarily have to be a programmer to use R. A beginner can use basic R functionalities to manipulate and extract data and create very simple and quick visualizations using the basic graphic tools. An intermediate R user can implement interactive visualizations, perform predictive modeling, or even create animated applications using packages developed by the R community. R will present you with the tools you need to process, manipulate, and communicate with your data, and it is not just limited to statistical analysis.

In this book, you will learn how to generate basic visualizations, understand the limitations and advantages of using certain visualizations, develop interactive visualizations and applications, understand various data exploratory functions in R, and finally learn ways of presenting the data to our audience. This book is aimed at beginners and intermediate users of R who would like to go a step further in using their complex data to convey a very convincing story to their audience.
What This Book Covers

Chapter 1, *A Simple Guide to R*, is a quick tutorial on getting started with R. You will learn how to install packages, access help on R, construct and edit matrices, create and manipulate data frames, and write and save plots.

Chapter 2, *Basic and Interactive Plots*, introduces some of the basic R plots, such as scatter, line, and bar charts. We will also discuss the basic elements of interactive plots using the googleVis package in R. This chapter is a great resource for understanding the basic R plotting techniques.

Chapter 3, *Heat Maps and Dendrograms*, starts with a simple introduction to dendrograms and further introduces the concept of clustering techniques. The second half of this chapter discusses heat maps and integrating heat maps with dendrograms to get a more complete picture.

Chapter 4, *Maps*, discusses the importance of spatial data and various techniques used to visualize geographic data in R. You will learn how to generate static as well as interactive maps in R. The chapter discusses the topic of shape files and how to use them to generate a cartogram.

Chapter 5, *The Pie Chart and Its Alternatives*, is a detailed discussion on how to generate pie charts in R. You will also learn about the various criticisms of pie charts and how the pie chart is transformed to overcome them. The chapter also provides you with various alternatives used by data scientists and visualization artists to overcome the limitation of a pie chart.

Chapter 6, *Adding the Third Dimension*, dives into constructing 3D plots. This chapter also introduces packages such as rgl and animation, which are used to create interactive 3D plots.

Chapter 7, *Data in Higher Dimensions*, demonstrates the use of visualizations that are used to display data in higher dimension. You will learn the techniques to generate sunflower plots, hexbin plots, Chernoff faces, and so on. This chapter also discusses the usefulness of network, radial, and coxcomb plots, which have been widely used in news.
Chapter 8, Visualizing Continuous Data, illustrates the use of visualizations to display time series data. The chapter also discusses some general concepts related to visualizing correlations, the shape of the distribution, and detection of outliers using box and whisker plots.

Chapter 9, Visualizing Text and XKCD-style Plots, illustrates the use of text in creating effective visualizations. This chapter focuses mainly on techniques to create word clouds, phase tree, and comparison clouds in R. You will also learn how to use the XKCD package to introduce humor in visualizations.

Chapter 10, Creating Applications in R, shows you the techniques to create presentations and R markdown documents for publishing on a blog or a website. The chapter further discusses the XML package used to extract and visualize data as well as using shiny package used to create interactive applications.
In this chapter, we will cover the following recipes:

- Constructing a sunflower plot
- Creating a hexbin plot
- Generating interactive calendar maps
- Creating Chernoff faces in R
- Constructing a coxcomb plot in R
- Constructing network plots
- Constructing a radial plot
- Generating a very basic pyramid plot

**Introduction**

Most of the visualizations studied so far have been widely observed in media, magazines, websites, or academic journals. Many of the recipes discussed in this chapter relate to visualizing data in higher dimensions or multivariate data. We might not have encountered some of the visualizations discussed in this chapter due to their limitations, but this does not imply that we cannot utilize them to convey the right information.
In this chapter, we will introduce plots such as sunflower plots, hexbins, calendar maps, coxcombs, and Chernoff faces, which are rarely used but are great tools to explore and present data. We will also explore network plots, pyramid plots, and radial plots, which have been utilized to convey information in a meaningful way.

Note that, while running the code, you might encounter issues where the plots do not look right or they are generated over an existing plot. The best way to avoid this is by clearing the plot window using the `clear all` tab and typing `plot.new()` in the R console window before running any code.

### Constructing a sunflower plot

Sunflower plot, as the name suggests, looks like a sunflower drawn in a 2D space. The sunflower plots are used as variants of scatter plots to display bivariate distribution. When the density of data increases in a particular region of a plot, it becomes hard to read. Each petal in a sunflower plot represents an observation; hence, sunflower plots can deal with high-density data. Hexbin plots, discussed later in the chapter, are also an alternative to resolving the issue of overlapping observations in a scatter plot. Dupont and Plummer Jr. (2003) provide an insightful discussion on the advantages of using a sunflower plot over scatter plots in case of high-density datasets.

Sunflower plots are available with the basic R plotting package, and we can learn more about their various arguments by typing `?sunflowerplot` in the R Console window.
Getting ready

The plot is constructed using the Galton data available with the HistData package in R. Galton data comprises two variables: average height of the father and mother, and the height of the child.

How to do it...

We load the Galton data by installing the package and load it in R by using the install.packages() and library() functions in R, respectively:

```r
install.packages("HistData")
library(HistData)
```

To examine the headers and first six observations, we use the head() function or View(head()) function. It is a good practice to partially view the data before starting to plot it; the head() function allows us to accomplish this in R:

```r
head(Galton)
View(head(Galton,8))
```

To construct a sunflower plot in R, we utilize the sunflowerplot() function:

```r
sunflowerplot(Galton$parent,Galton$child,col = "blue",
seg.col = "red", xlab = "Parent", ylab ="Child")
```

How it works...

The library() and head() functions are self-explanatory. The number 8 in the View() function is the number of data lines to display. The data to be displayed on the x-axis is passed as the first argument in the sunflowerplot() function. The second argument consists of data to be displayed on the y-axis. Note the use of the $ sign; as discussed previously, the $ sign is placed after the referencing dataset, which is followed by the $ sign (data$column name).

The col argument is used to specify the color of the plot and the seg.col argument applies color to the leaves. If the data is a single dot, it is just one observation; if the data has two lines, it represents two points; data with three lines represents three points, and so on. The sunflowerplot() function comes with additional arguments that can be explored by typing ?sunflowerplot in the R console window.
Creating a hexbin plot

Hexbin plots can be viewed as an alternative to scatter plots. The hexagon-shaped bins were introduced to plot densely packed sunflower plots. They can be used to plot scatter plots with high-density data. We will use the `hexbin` package available in R to plot hexagon-shaped bins on a plot instead of a sunflower.

Getting ready

To generate a hexbin plot, we will use the `hexbin` package in R along with the generic `rnorm()` function.

How to do it...

The `install.packages()` and `library()` functions assist in installing the package as well as loading it in R:

```r
install.packages("hexbin")
library(hexbin)
```
For the purpose of this recipe, we will generate a fake dataset. The `set.seed(356)` code sets the state of random number generation in R:

```r
set.seed(356)
```

We will implement the `rnorm()` function to generate 1000 normally distributed random numbers:

```r
x = rnorm(1000)
y = rnorm(1000)
```

In order to generate a hexbin plot, we will have to define a hexbin object using the `hexbin()` function:

```r
bins = hexbin(x, y)
```

Now, we can simply use the generic R `plot()` function to create a hexbin plot:

```r
plot(bins)
plot(bins, border = TRUE)
plot(bins, border = "red")
smb = smooth.hexbin(bins)
plot(smb)
```

### How it works...

The `rnorm()` function consists mainly of three arguments. The first argument is `n`, which represents the number of data points to be generated; the second and third arguments are the mean and standard deviation respectively. By default, R will assume a mean of 0 and a standard deviation of 1 if these arguments are missing. To learn more about all the different distributions available in R, type `?distributions` in the R console window.

One important ingredient required to generate a hexbin plot is generating the bins. We use the `hexbins()` function, available in the hexbin package, to accomplish our task. The first and second arguments of the `hexbin()` function are the values generated for variables `x` and `y`. R will calculate the hexbins and store them as objects under the name `bins`. Please refer to the hexbin manual available on the CRAN website at [http://cran.r-project.org/web/packages/hexbin/hexbin.pdf](http://cran.r-project.org/web/packages/hexbin/hexbin.pdf) to understand the extra arguments related to the `hexbin()` function.

Finally, we can generate a very simple hexbin plot by passing the `bins` object in the generic `plot()` function in R. We can also apply bin smoothing to our data by creating an object using the `smooth.hexbin(bins)` function and subsequently passing it in the `plot(smb)` function. Smoothing our data assists in better interpretation of data, especially in cases where the data is too large.
See also

- It is possible to implement the hexbin plot using the **ggplot2** package. Please refer to the documentation at [http://docs.ggplot2.org/current/stat_binhex.html](http://docs.ggplot2.org/current/stat_binhex.html) to understand its implementation.

Generating interactive calendar maps

Calendar maps are used to display continuous data over a period of time. Calendar plots have been used to display data on a daily or monthly basis, where each square represents a data point.

In the past, Nathan Yau, in his fatal car crashes, has used calendar heat maps to study the safest time of the year to travel. Using stock prices as input in our calendar map assists us in conducting event study or holiday and weekend effects. We can also apply calendar maps to study airline delays during the days of the month or crime rates during specific times of the year.

In this recipe, we will use the calendar plot to study the daily stock price movement of Facebook over 2013. In order to extract the stock prices from Yahoo Finance, we will use the **quantmod** package and construct a calendar plot using the **googleVis** package.
Getting ready

To generate an interactive calendar map, we will install the following packages:

- The quantmod package to download the stock prices
- The googleVis package to generate an interactive calendar plot

How to do it...

To generate an interactive calendar plot in R, we will install as well as load the quantmod and googleVis packages in R by typing the following lines in the R console window:

```r
install.packages(c("quantmod","googleVis"))
library(quantmod)
library(googleVis)
```

To extract prices, we will first create a character vector in R using the `c()` notation and use the `getSymbols()` function to extract the prices of Facebook stock using its ticker `FB`:

```r
prices = c("FB")
getSymbols(prices, src = "yahoo", from =as.Date("2013-01-01"),
to = as.Date("2013-12-31"))
```

To implement a calendar plot using the googleVis package, we need the data in a data frame format. Hence, we use the `data.frame()` function as shown:

```r
data =data.frame(date = as.Date(index(FB)),
open = as.numeric(FB$FB.Open))
```

Finally, we plot the calendar plot in R by creating a calendar object using the `gvisCalendar()` function and display the plot using the `plot()` function:

```r
fb <- gvisCalendar(data, datevar="date",
numvar="open", options=list(title="Daily Prices of Facebook stock",height=320,width = 1000, calendar="{yearLabel: {color: '#FF0000'},cellSize: 14,cellColor:{strokeWidth: 0.5}, monthOutlineColor:{stroke :'white',strokeWidth: 5}}"))
plot(fb)
```

How it works...

The first argument in the `getSymbols()` function is the list of tickers. In our recipe, we are only using the Facebook prices, but we can pass multiple tickers as well using the `prices= c("FB","AMZN","MSFT")` code. The `src` argument is used to define the source to pull the data. In order to learn more about various other acceptable values, please type `?getSymbols` in the R console window. The `from` and `to` arguments are used to define a date range. The `as.Date()` function is a basic R function used to define dates in R.
Data in Higher Dimensions

By default, the data downloaded using the `getSymbols()` function is in the XTS format. We could also download the data as "zoo", "ts", `data.frame`, or in a time series format. All we need to do is include one more additional argument `return.class = ts` (for time series) under the `getSymbols()` function.

The downloaded data has columns: close, volume, high, low, and so on. We will utilize opening prices and date columns. Note that, in order to plot a calendar plot, we require data in the `data.frame` format. We can extract the dates and opening prices columns from the XTS data file using the `data.frame()` function.

We would like to extract two columns from the FB dataset: date and opening prices. Hence, we pass date as the first argument in the `data.frame()` function and the second argument is `FB.open`, which is just the opening price column from the FB data. Note the use of `as.Date(index(FB))` and `as.numeric(FB$FB.Open)`.

The `gvisCalendar()` function allows us to construct the plot. The first argument in the function is our dataset generated using the `data.frame()` function. The `datevar` as well as `numvar` arguments are used to define the columns to be used to construct the plot. The plot can be customized further by specifying various options such as, width, height, labels, and so on, under the `options = list()` argument. A list of all the available options is specified at [http://cran.r-project.org/web/packages/googleVis/googleVis.pdf](http://cran.r-project.org/web/packages/googleVis/googleVis.pdf) and [https://developers.google.com/chart/interactive/docs/gallery/calendar](https://developers.google.com/chart/interactive/docs/gallery/calendar).

The `plot()` function can be called to generate the plot in a new browser window. Note that, at the time of writing this chapter, `googleVis` provided limited flexibility to change the color of the cells in a calendar plot. This might change in future releases.
Creating Chernoff faces in R

One of the alternative methods to visualize multivariate data is using Chernoff faces. Each variable in the dataset is used to represent a feature of the face. Chernoff used 18 variables to represent different facial features such as head, nose, eyes, eyebrows, mouth, and ears. Kosara (http://eagereyes.org/criticism/chernoff-faces) discusses the limitation of using Chernoff faces at length.

In order to construct Chernoff faces, I have downloaded some specific macroeconomic variables for a set of countries from the World Bank website.

![A comparative view using Chernoff faces](source:www.worldbank.org)

**Getting ready**

We will implement Chernoff faces in R by using the `faces()` function available under the `aplpack` package in R.
Data in Higher Dimensions

How to do it...

To create Chernoff faces, we would require to install the `aplpack` package in R and load it in our active R session using the `install.packages()` and `library()` functions:

```r
install.packages("aplpack")
library(aplpack)
```

We can load the data in R using the `read.csv()` function. Note that R will search for the datafile in your current directory. Hence, if the data is not present in your current directory, you would have to change your current directory in R using the `setwd()` function:

```r
data1 = read.csv("worlddecon.csv", header = TRUE, sep =",")
```

It is a good practice to view the data once it is imported in R in order to examine the format and column headers. We can explore the data using the `head()` function:

```r
head(data1)
```

Finally, we generate Chernoff faces using the `faces()` function available with the `aplpack` package:

```r
faces(data1[1:10,3:9], labels = data1$Code, main = "A comparative view using Chernoff faces")
```

How it works...

The first argument in the `faces()` function is the dataset that is imported using the `read.csv()` function. The `labels` argument instructs R to use the specified column as labels. The documentation available on the CRAN website lists many other options that we can implement.

Please note that I have used the first 10 rows to plot the faces. If you use the entire dataset, you might get an error in R. This error can be avoided by simply expanding the R Studio’s plot window. When R plots the faces, it will also plot the column headings corresponding to the facial expression. This information can be used as a legend to add more information about our visualization.

One of the criticisms regarding the use of Chernoff faces is that it is very hard to read facial expressions, especially when they are not very extreme. With regard to our data, we observe that Singapore (SGP) has bigger hair, implying high exports compared to other countries. Also, when considering debt, we observe that Great Britain (GBR) has lower debt compared to Japan (JPN), which is evident from the length of the face.
Constructing a coxcomb plot in R

Coxcomb plots or Polar diagrams were developed by Florence Nightingale to show that most of the deaths of British soldiers were due to sickness rather than actual wounds during the Crimea War. Coxcomb plots are usually viewed as variants of pie charts.

According to Wikipedia, if the count of deaths in each month for a year is to be plotted, then there will be 12 sectors (one per month), all with the same angle of 30 degrees each. The radius of each sector would be proportional to the square root of the death count for the month, so the area of a sector represents the number of deaths in a month. To construct the coxcomb plot, we will use the same dataset that was used by Florence Nightingale.

Getting ready

In order to construct a coxcomb plot, we will utilize the HistData and plotrix packages.

How to do it...

We will install and load the packages in R using the install.packages() and library() functions:

```
install.packages(c("HistData","plotrix"))
library(HistData)
library(plotrix)
```
Data in Higher Dimensions

The data used in this recipe is available with the HistData package. We can load the Nightingale data in R by typing the following code in R:

```r
data = Nightingale[13:24,]
```

To generate the coxcomb plot in R, we will use the `radial.pie()` function available in the plotrix package. The labels for the plot are generated by constructing a character vector, `month`:

```r
month = c("Apr 1855","May","Jun","Jul","Aug","Sep","Oct",
           "Nov","Dec","Jan 1856","Feb","Mar")
radial.pie(data$Disease,labels=month, boxed.radial = FALSE,show.grid = TRUE,sector.colors =c(rep("#60B1FF",12)),
           grid.col= "white",mar=c(2,10,2,10),show.grid.labels = 0,
           axis = FALSE, label.prop= .9)
```

**How it works...**

The HistData library consists of many historical datasets; we will use this package to load the Nightingale dataset in R. We are employing two steps in one line of code. As we only require the data for 1855, we will use `[ ]` to extract the data from Nightingale as `Nightingale[13:24,]` and store it as `data`.

If you go through the `code.txt` file for this chapter, you will observe two additional ways to generate the radial pie. We can manipulate the arguments within the `radial.pie()` function to customize our plot as per our requirements.

The first argument in the `radial.pie()` function is the data that needs to be visualized, and the second argument corresponds to the labels. The `boxed.radial` argument is used to create a box around the radial values. We have displayed the grid around the plot by using the `show.grid = TRUE` argument but have colored it white using the `grid.col` argument.

The `sector.colors` argument allows us to color each sector of the plot differently. However, we would like to apply the same color to all the sectors and hence we use the `rep()` function within the `sector.color` argument.

The `show.grid.labels` argument allows us to display labels in the plot. The `label.prop` argument allows us to position the label; we can change the value and observe the difference. Readers should refer to the plotrix documentation on the CRAN website at [http://cran.r-project.org/web/packages/plotrix/plotrix.pdf](http://cran.r-project.org/web/packages/plotrix/plotrix.pdf) for all the extra arguments that can be used along with the `radial.pie()` function.
See also

- Spiechart (http://spatial.ly/2014/01/coxcomb-plots/) does a very interesting implementation of spiechart and coxcomb plots. The plot has been generated using the ggplot2 package. The author also provides R code for this.
- The Worth a Thousand Words article in The Economist discusses the coxcomb plot. It can be accessed at http://www.economist.com/node/10278643.
- An animated coxcomb plot is shown at http://understandinguncertainty.org/coxcombs.

Constructing network plots

One of the first visualizations that introduced me to network plots was Visualizing Friendships. Network plots are not just limited to social networks but are observed in finance to study the linkages between Markets; they have been implemented in medicine to study the spread of viruses; and they have also been used to study social dynamics of groups, such as a network of friends. Network is an actively researched field and lots of books and articles have been published on it.

In this recipe, we will study the basics of creating a network plot using a random dataset.
Data in Higher Dimensions

Getting ready

We will construct a network plot using the igraph package in R.

How to do it...

To construct a network plot, we will install the igraph package and also load it in our active R session by typing the following lines of code:

```r
install.packages("igraph")
library(igraph)
```

Next, we generate fake data and import it in R using the `read.csv()` function:

```r
net = read.csv("network.csv", sep = ",", header = TRUE)
```

We will now create a network graph object using the `graph.data.frame()` function:

```r
g = graph.data.frame(net)
```

The network object can be used to plot the network graph by calling the simple R plotting function:

```r
plot(g)
```

```r
V(g)$label = LETTERS[1:6]
plot(g, vertex.size = 25, edge.arrow.size = 0.8)
plot(g, vertex.size = 25, edge.arrow.size = 0.8, edge.curved = TRUE, layout = layout.circle)
```

How it works...

In order to create a network graph, we have generated a fake dataset and stored it as a .csv file. Once we have loaded the data in R, we can generate a network object by passing our data as an argument in the `graph.data.frame()` function. A very quick and dirty way to visualize a network is by simply passing the network object in the `plot(g)` function.

The \( E(g) \) and \( V(g) \) functions will print a list of all the edges and vertices in R. All the vertex and edges options can be declared in the basic `plot()` function, as described in the previous code. The entire list of options is available in the igraph library document available on the CRAN website at [http://cran.r-project.org/web/packages/igraph/igraph.pdf](http://cran.r-project.org/web/packages/igraph/igraph.pdf).
There's more...

Readers who are interested in creating an interactive network plot can achieve this by using the `tkplot(g)` function in R. The function allows users to interactively change the layout as well as select edges and vertices.

See also

- Financial and Macroeconomic Connectedness provides a way to replicate the network using gephi and the R package. It can be accessed at http://financialconnectedness.org/Stock.html#.
- I personally prefer gephi to generate network plots. The gephi package is an open source package used to plot networks. Users can easily generate, edit, and perform basic calculations. You can refer to https://gephi.github.io/.

Constructing a radial plot

The main idea behind a radial plot is to project the data as a distance from the centre in a circular form. Radial plots are not observed very often but are good tools to visualize monthly time series data. In this recipe, we will use oil prices in USA as an example to construct the radial plot.
Data in Higher Dimensions

Getting ready
In order to create a radial plot, we need to load the plotrix package.

How to do it...
We can install the package as well as load the library in our active R session by typing the following lines in the R console window:

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

The data consists of 21 years of monthly data for oil prices in USA. We import our data in R using the `read.csv()` function. Note that R will search for the file in our current R directory:

```r
oil = read.csv("oil.csv")
```

We can now construct a radial plot by implementing it in R via the `radial.plot()` function:

```r
radial.plot(oil[,21],rp.type="p",lwd =3,line.col="blue",
labels=oil$Month, clockwise = TRUE, start = 1.5,
grid.unit = c("\$"), main = "Oil prices in 2013")
```

How it works...
The first argument in the radial plot is the data. Note that square brackets in the code `oil[,21]` instruct R to plot all the rows and column 21. The argument `rp.type` assigns the elements to our plot and the `P` argument creates a polygon. The `rp.type` argument also accepts `l` and `s` to plot radial lines and symbols respectively. The `labels` argument is used to assign the labels to our plot; in this recipe, the labels are months.

The `start` argument allows us to change the orientation of the labels. By default, January is the month drawn at 3 o'clock but, using the `start` argument, we can change it so that January begins at 12 o'clock.

The `radial.plot()` function has many other options to learn about. For these options, users should refer to the manual available on the CRAN website. I have used some of the options in the code file to display the use of various arguments in the `radial.plot()` function.
There's more...

If we would like to plot all the data points, we do not need to run each line and change the value under \texttt{oil[,13]} to \texttt{oil[,20]}. In R, we can write a loop that goes to every column and plots the image. The following lines of code are written to demonstrate the looping capability in R:

```r
plot.new()
radial.plot(oil[,2], rp.type="p", lwd =3, line.col="blue",
labels=oil$Month, clockwise = TRUE, start = 1.5, grid.unit =
c("$"), main = "Oil prices cycles 1994-2013",
radial.lim = c(0,4))
for(i in 3:21){
  radial.plot(oil[,i], rp.type="p", lwd =3, line.col="blue",
  labels=oil$Month, clockwise = TRUE, start = 1.5, add = TRUE) }
```

We have started the code by creating a base radial plot on our plotting area. In this recipe, we would like to plot one series over the other and hence we use a loop. Note that we have used the \texttt{add = TRUE} and \texttt{radial.lim} arguments. If we omit the \texttt{radial.lim} argument, the plot area will not automatically adjust and hence the lines in a radial plot will be all over the plotting region. We determine the limit by using the summary function and observing the max values of each series.

To learn more on how to write a loop, please refer to the recipe \textit{Basic loops in R} in Chapter 1, \textit{A Simple Guide to R}.

The plot generated does not allow us to understand the cycles. We have used a loop statement to show how new plots can be added to an existing plot. Nathan Yau uses radial plots to visualize cyclical data such as flight data.

See also

- The plotrix manual at \url{http://cran.r-project.org/web/packages/plotrix/plotrix.pdf}.
- I was unable to cover two other very important plots: StarPie and diamond. Each of these plots is discussed in detail in the plotrix manual. The StarPie plot can be used to compare similarity or dissimilarity between two different datasets. The diamond plot is very similar to radar plots, which we often encounter in business presentations.
**Generating a very basic pyramid plot**

You might have seen these plots in news or journal articles and wondered how to create them quickly. This recipe will help you accomplish this task. Pyramid plots are horizontal bar plots and they are often used to display gender differences in a dataset. I have created this plot based on a New York Times infographic discussing the deaths by different types of cancer among men and women. The data was extracted from the Centers of Disease Control and Prevention website.

![Pyramid Plot Example]

**Getting ready**

We require the following packages:

- plotrix
- RColorBrewer

**How to do it...**

The `install.packages()` and `library()` functions can be used to install and load the packages in R:

```r
install.packages(c("plotrix","RColorBrewer"))
library("plotrix")
library("RColorBrewer")
```

The `read.csv()` function is used to load the CSV file in R:

```r
data = read.csv("cancer.csv", sep = ",", header = TRUE)
```
Next, we can create a pyramid plot in R using the `pyramid.plot()` function and add the legends using the `legend()` argument:

```r
pyramid.plot(data$Men_g, data$Women_g, labels=data$Causes, unit = NA, gap = 60000, laxlab=c(0,100000,150000,200000), raxlab=c(0,100000,150000,200000), top.labels = c("Male", "Cancer", "Female"), lxcol = "+99d8c9", rxcol = "#bcbddc")
pyramid.plot(data$Men_d, data$Women_d, labels=data$Causes, unit = NA, gap = 60000, laxlab=c(0,100000,150000,200000), raxlab=c(0,100000,150000,200000), lxcol = "#2ca25f", rxcol = "+8856a7", add = TRUE, space = 0.5)
legend("topleft", fill = c("#99d8c9","#2ca25f"), legend=c("New Cases","Death"), border = FALSE, bty= "n")
legend("topright", fill = c("#bcbddc","#8856a7"), legend=c("New Cases","Death"), border = FALSE, bty = "n")
```

**How it works...**

Our dataset comprises two sets of data. One set titled `_g` consists of all the cases that were diagnosed with cancer in 2012, whereas columns titled `_d` are individuals that died due to cancer. We have utilized both sets of data.

The `pyramid.plot()` function takes the data to be plotted on the left and on the right as its first two arguments. In order to fix the scaling on the plot we have used the `raxlab` and `laxlab` arguments. The argument `top.labels` is used to title each side of the plot. The arguments `rxcol` and `lxcol` are used to color the bars in our plot. Note that we have used the colors from the `RColorBrewer` package.

Finally, we state `add = true` to allow R to plot the second set of data on top of the first set. If the colors are not transparent, you might not see the effect clearly. We have also included the `gap` argument in the second `pyramid.plot()` function in order to make the plot look like the New York Times Visualization.

We add the legends using the `legend()` function. We will require two sets of legends to appear on each side of the plot; hence, we pass the `legend()` function twice. The first argument in the `legend()` function is the position of the legend. The `fill` argument allows the boxes to be filled with colors and the `legend` argument generates the labels to be applied. To learn more about legends, readers should type `?legend()` in the R console window.

**See also**

- Centers for Disease Control and Prevention, which is a great source for data related to health, at [http://www.cdc.gov/](http://www.cdc.gov/)
Where to buy this book

You can buy R Data Visualization Cookbook from the Packt Publishing website. Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers. Click here for ordering and shipping details.