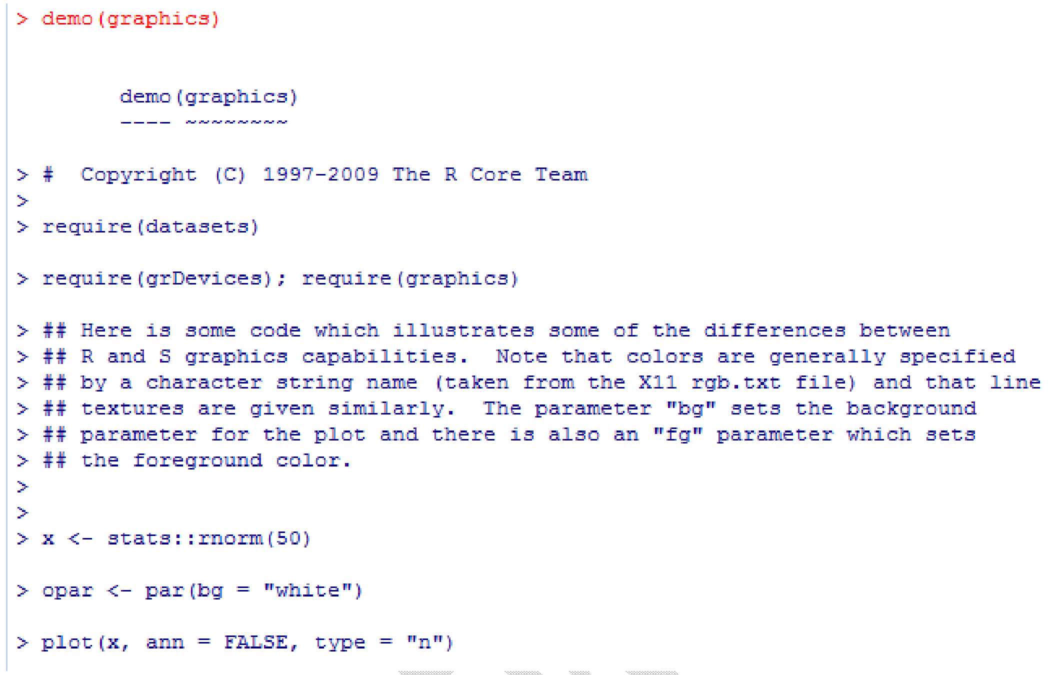
Unit 6 - Graphs

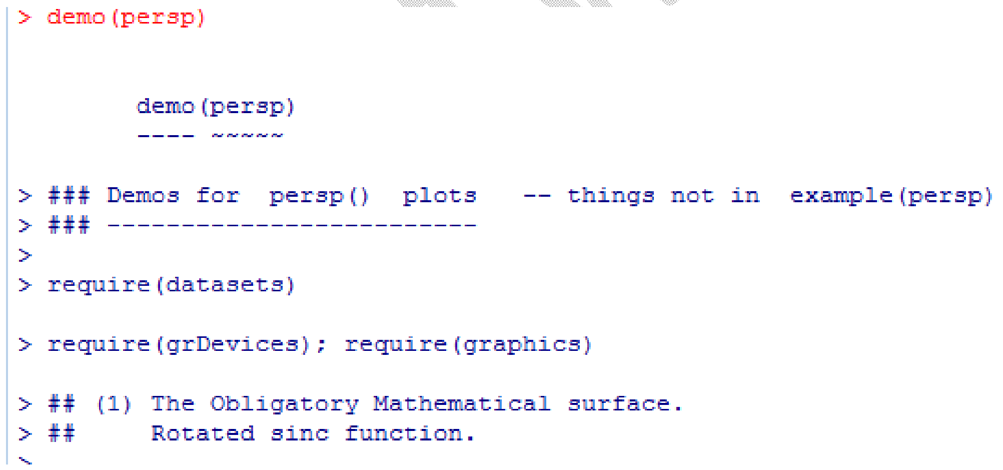
Contents

1. Introduction

- R offers a remarkable variety of graphics.
- In R, graphs are typically created interactively.
- One can type demo(graphics) or demo(persp) to get an idea.



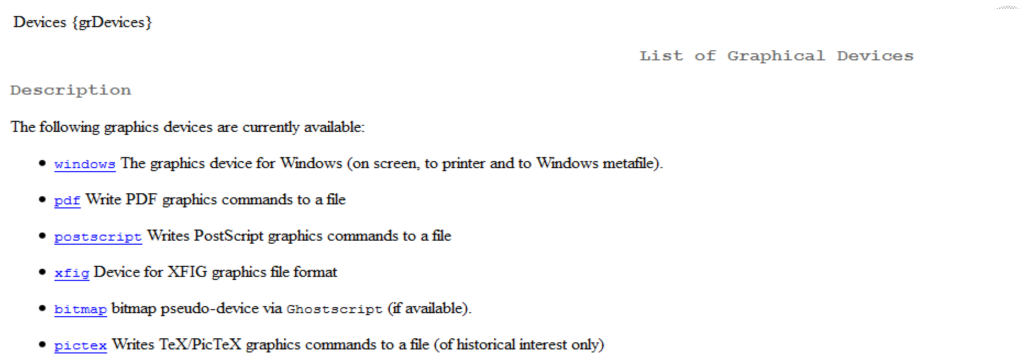
```
> demo(graphics)


        demo(graphics)
        ---- ~~~~~~~~

> #  Copyright (C) 1997-2009 The R Core Team
>
> require(datasets)

> require(grDevices); require(graphics)

> ## Here is some code which illustrates some of the differences between
> ## R and S graphics capabilities.  Note that colors are generally specified
> ## by a character string name (taken from the X11 rgb.txt file) and that line
> ## textures are given similarly.  The parameter "bg" sets the background
> ## parameter for the plot and there is also an "fg" parameter which sets
> ## the foreground color.
>
>
> x <- stats::rnorm(50)

> opar <- par(bg = "white")

> plot(x, ann = FALSE, type = "n")
```



```
> demo(persp)


        demo(persp)
        ---- ~~~~~

> ### Demos for  persp()  plots   -- things not in  example(persp)
> ### -------------------------
>
> require(datasets)

> require(grDevices); require(graphics)

> ## (1) The Obligatory Mathematical surface.
> ##     Rotated sinc function.
>
```

2. Graphical devices

- When a graphical function is executed, R opens a graphical window (X11 for windows) and displays the graphic.
- A graphical device will open with a function depending on the format: postscript(), pt(), pdf(), png().
- The list of available graphical devices can be found with ? Device.



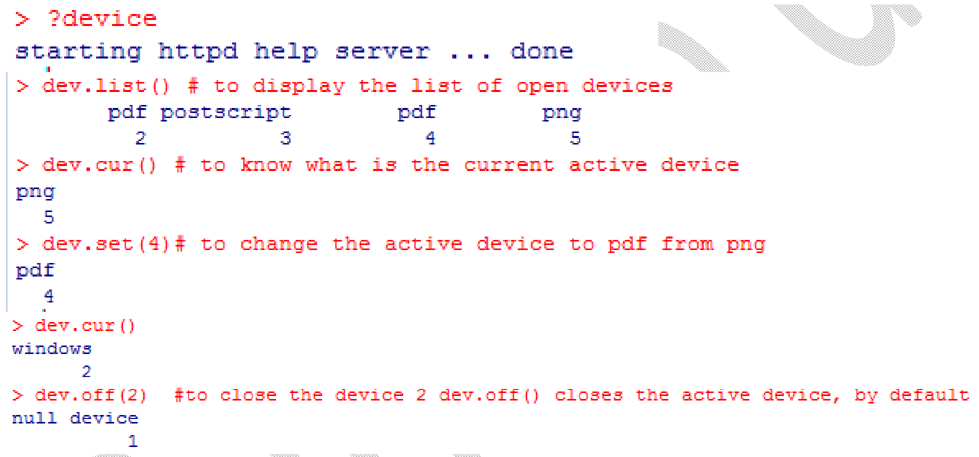
Devices {grDevices}

List of Graphical Devices

Description

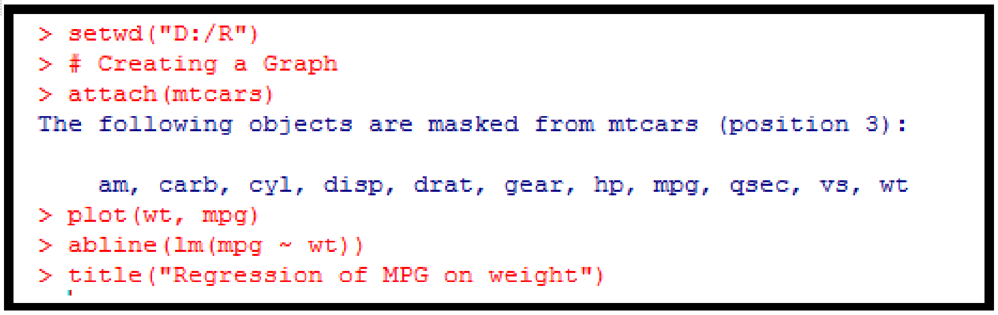
The following graphics devices are currently available:

- windows The graphics device for Windows (on screen, to printer and to Windows metafile).
- pdf Write PDF graphics commands to a file
- postscript Writes PostScript graphics commands to a file
- xfig Device for XFIG graphics file format
- bitmap bitmap pseudo-device via Ghostscript (if available).
- pictex Writes TeX/PicTeX graphics commands to a file (of historical interest only)



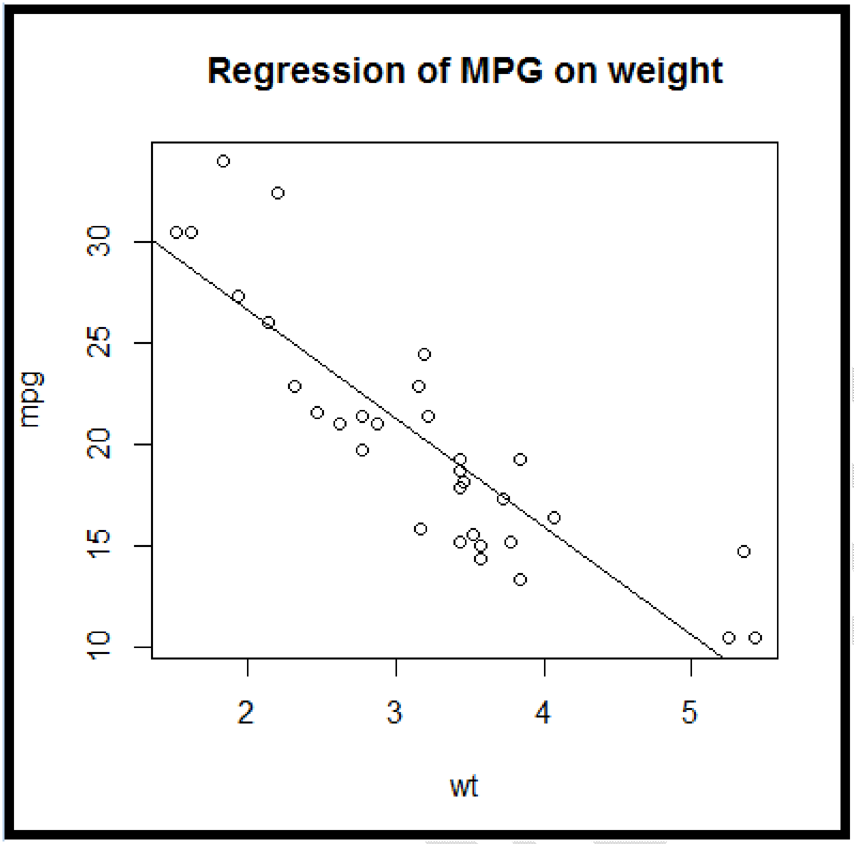
```
> ?device
starting httpd help server ... done
> dev.list() # to display the list of open devices
       pdf postscript        pdf        png
         2          3          4          5
> dev.cur() # to know what is the current active device
png
  5
> dev.set(4)# to change the active device to pdf from png
pdf
  4
> dev.cur()
windows
      2
> dev.off(2)  #to close the device 2 dev.off() closes the active device, by default
null device
          1
```

2.1. Creating a graph



```
> setwd("D:/R")
> # Creating a Graph
> attach(mtcars)
The following objects are masked from mtcars (position 3):

    am, carb, cyl, disp, drat, gear, hp, mpg, qsec, vs, wt
> plot(wt, mpg)
> abline(lm(mpg ~ wt))
> title("Regression of MPG on weight")
```

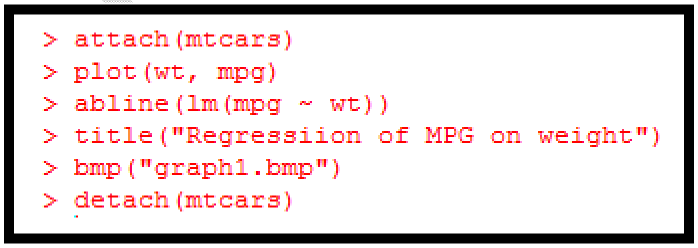
- The plot() function opens a graph window and plots weights vs. miles per gallon.
- The next line of code adds a regression line to this graph.
- The final line adds a title.



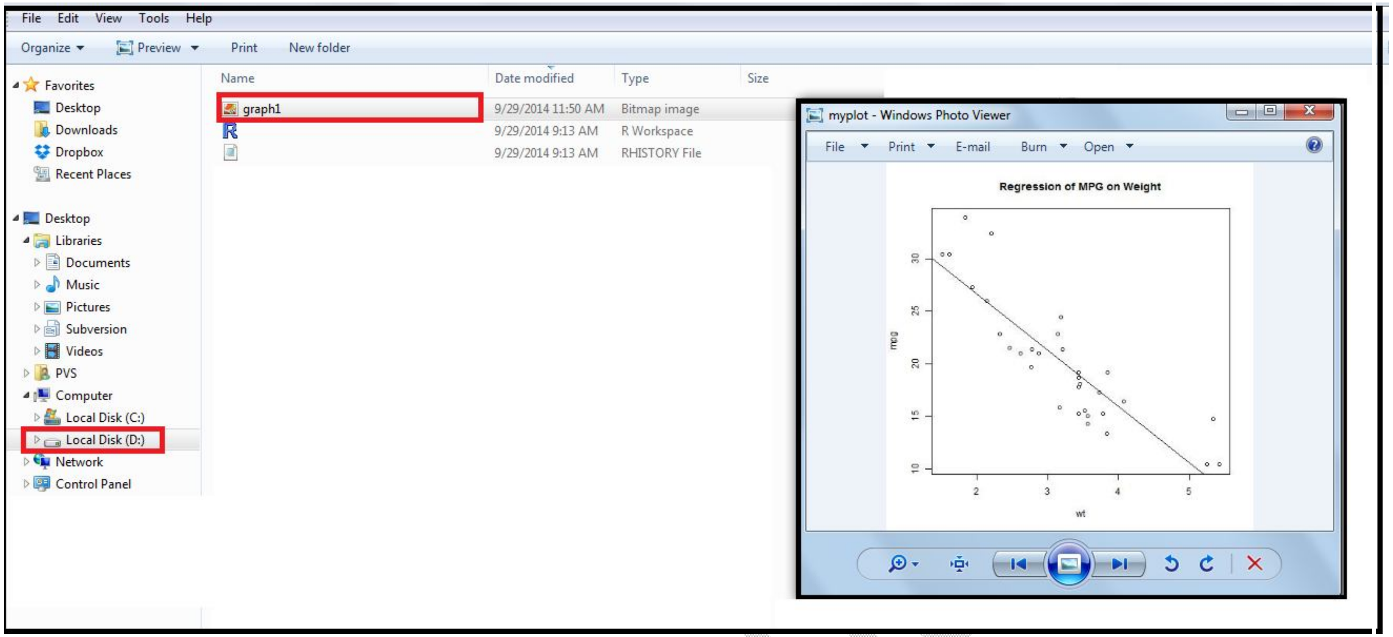
2.2. Saving a graph

- You can save the graph via code using one of the following functions:

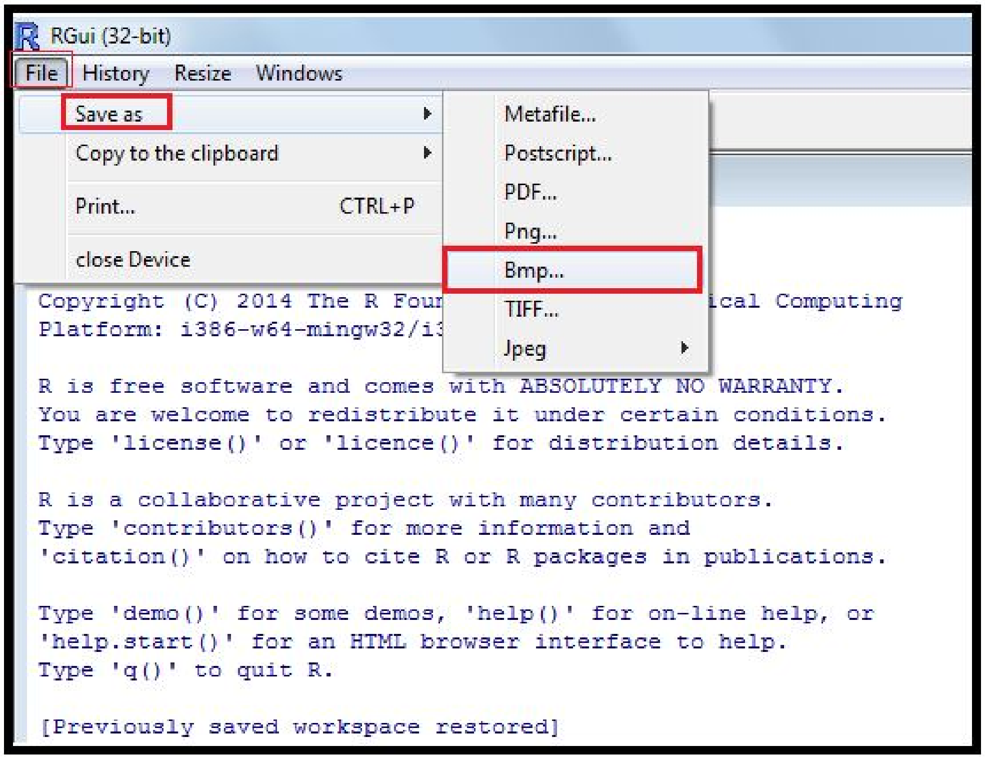
Function	Output to
pdf(“graph1.pdf”)	pdf file
win.metafile(“graph1.wmf”)	windows meta file
png(“graph1.png”)	png file
jpeg(“graph1.jpg”)	jpeg file
bmp(“graph1.bmp”)	bmp file
postscript(“graph1.ps”)	postscript file



```
> attach(mtcars)
> plot(wt, mpg)
> abline(lm(mpg ~ wt))
> title("Regressiion of MPG on weight")
> bmp("graph1.bmp")
> detach(mtcars)
```


File Edit View Tools Help
Organize Preview Print New folder
Favorites
Desktop
Downloads
Dropbox
Recent Places
Desktop
Libraries
Documents
Music
Pictures
Subversion
Videos
PVS
Computer
Local Disk (C:)
Local Disk (D:)
Network
Control Panel
Name Date modified Type Size
graph1 9/29/2014 11:50 AM Bitmap image
9/29/2014 9:13 AM R Workspace
9/29/2014 9:13 AM RHISTORY File
myplot - Windows Photo Viewer
File Print E-mail Burn Open
Regression of MPG on Weight
mpg
wt

- You can save the graph in a variety of formats from the menu **File → Save As**



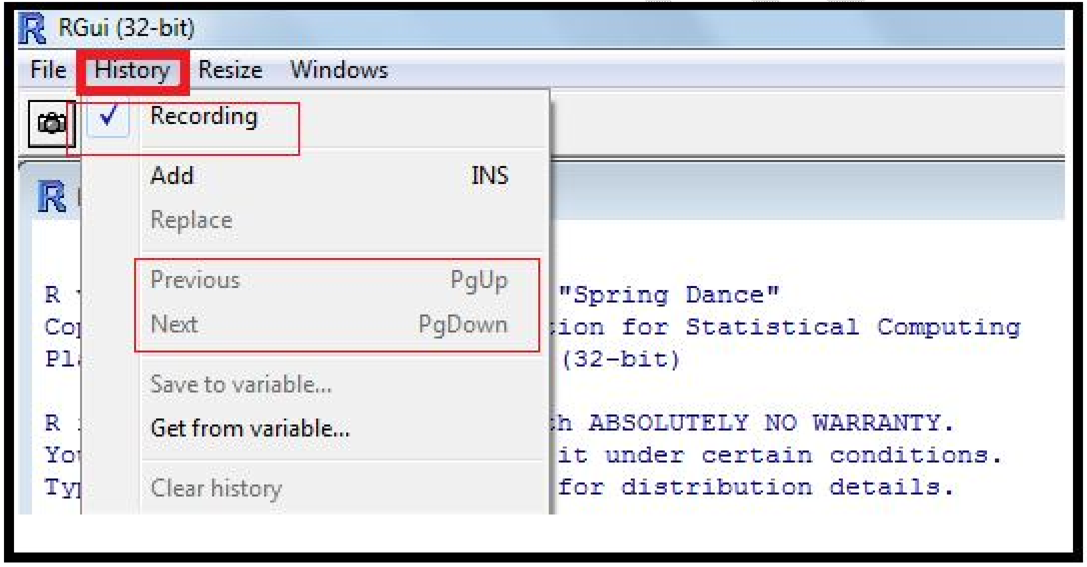
```
Copyright (C) 2014 The R Foun                    ical Computing
Platform: i386-w64-mingw32/i3

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]
```

2.3. Viewing several graphs

- Creating a new graph by issuing a high level plotting command (plot, hist, boxplot, etc.) will typically overwrite a previous graph.
- To avoid this, open a new graph window before a new graph.
- To open a new graph use one of the functions below:

Function	Platform
windows()	Windows
x11()	Unix
quartz()	Mac

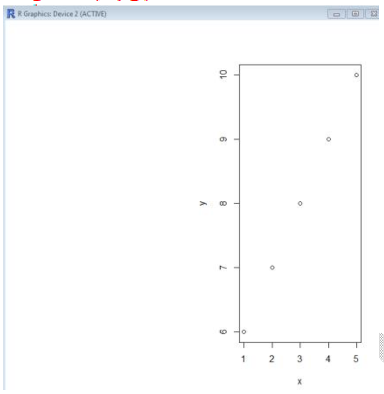
- You can have multiple graph windows open at one time. See **help(dev.cur)** for more details.
- Alternatively, after opening the first graph window, choose **history -> Recording** from the graph window menu. Then you can choose **Previous** and **Next** to step through the graphs you have created.



```
R                          "Spring Dance"
Co                        ion for Statistical Computing
Pl                         (32-bit)

R                         h ABSOLUTELY NO WARRANTY.
Yo                        it under certain conditions.
Ty                        for distribution details.
```

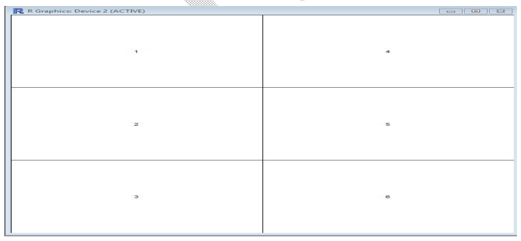
2.1. Partitioning a graphic

```
> split.screen()# details of screen split
[1] FALSE
> split.screen(c(1,2)) # to divides the devide into two parts
[1] 1 2
> split.screen()# screen is split
[1] 1 2
> split.screen()# details of screen split
[1] 1 2
> x
[1] 1 2 3 4 5
> y
[1]  6  7  8  9 10
> screen(2)
> plot(x,y)
```



- After splitting the screen into two, the part can be selected with screen(1) or screen(2).
- A part of the device can itself be divided with split.screen()
- The function layout partitions the active graphic window in several parts where the graphs will be displayed successively.
- To actually visualize the partition created, one can use the function layout.show with the number of sub-windows as argument.

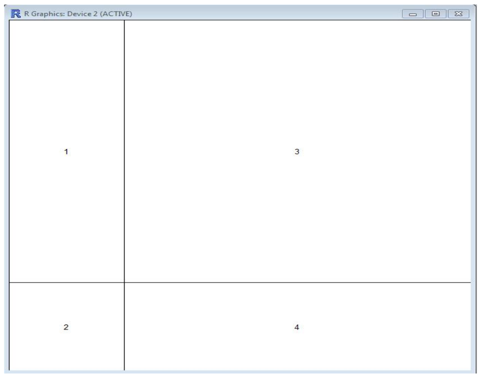
```
> layout(matrix(1:6,3,2))
> layout.show(6)
```



- By default, layout() partitions the device with regular heights and widths: this can be modified with the options widths and heights.

- These dimensions are given relatively.

```
> m<-matrix(1:4,2,2)
> layout(m,widths=c(1,3),heights=c(3,1))
> layout.show(4)
```

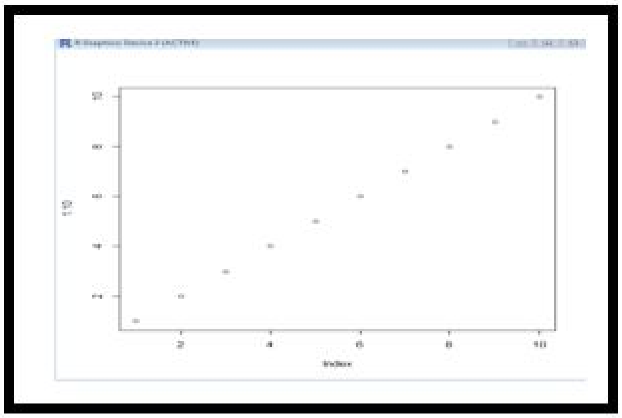


3. Graphical functions

- Built-in functions in R help us to make plots as per the requirement.
- Powerful plotting techniques make R an important tool for statistical applications.
- The plotting commands can be categorized as high-level commands, low-level commands and interactive graphics.

3.1. High-level commands

- High-level commands endue creating new plots with fundamental information like axes, title, labels etc.
- plot(): This is the most frequently used plotting command in R. It is a generic function which creates plots of type equivalent to that of the first argument.
- In the example below, vector data is plotted which is given as the first argument to the plot function.



```
> plot(1:10)
```

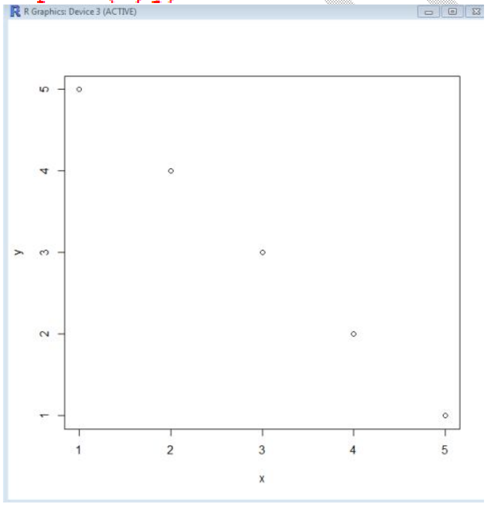
- The other high-level plotting commands in R enable to create variety of plots according to the requirement.
- R provides various high-level plotting functions to create variety of plots including boxplot, pie chart, barplot, and histogram.

Function	Description
plot(x,y)	Bivariate plot of x (on the x-axis) and y (on the y-axis)
boxplot(x)	Box-and whiskers plot
pie(x)	Circular pie chart
barplot(x)	Histogram of the values of x
hist(x)	Histogram of the frequencies of x

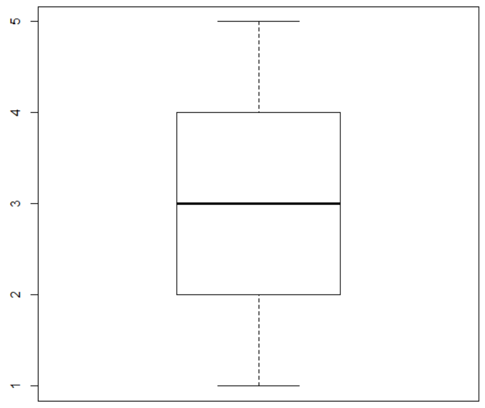
Parameter	Description
add = FALSE	If TRUE, superposes the plot on the previous one, if it exists
axes = TRUE	If FALSE, does not draw the axes and the box

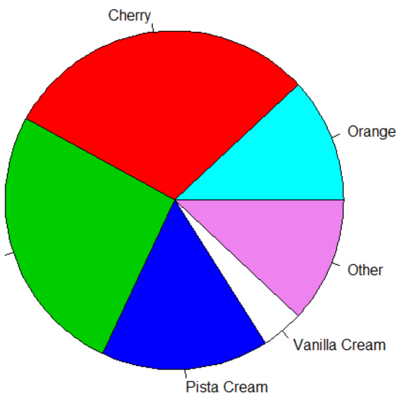
type="p"	Specifies the type of plot, "p": points, "l": lines, "b": points connected by lines, "h":vertical lines, "o":id . But the lines are over the points, "h": vertical lines, "s":steps, the data are represented by the top of the vertical lines, "S":id. But the data are represented by the bottom of the vertical lines
xlin, ylim	Specifies the lower and upper limits of the axes, for example with xlim=c(1,10) or xlim=range(x)
xlab, ylab	Annotates the axes, must be variables of mode character
main=	Main title, must be a variable of mode character
sub=	Sub-title (written in a smaller font)

```
> x<-1:5
> y<-5:1
> plot(x,y)
```

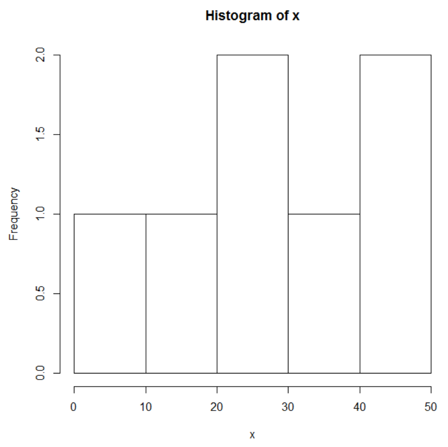


```
> x
[1] 1 2 3 4 5
> boxplot(x)
```

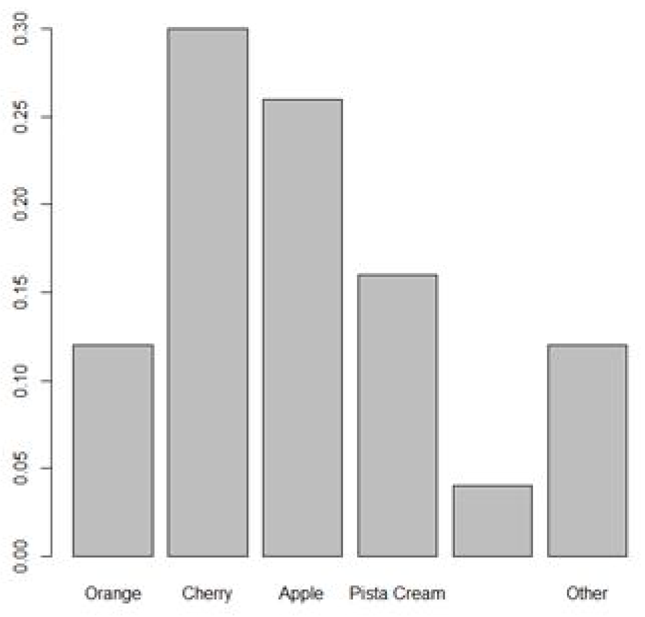




```
> x<-c(9,12,22,23,34,45,49)
> boundaries<-c(0,10,20,30,40,50)
> hist(x,breaks=boundaries)
```



```
> pie.sales<-c(0.12,0.3,0.26,0.16,0.04,0.12)
> names(pie.sales)<-c("Orange","Cherry","Apple","Pista Cream","Vanilla Cream","Other")
> pie(pie.sales,col=c("cyan","red","green3","blue","white","violet"))
```



```
> barplot(pie.sales)
```

3.2. Low - Level commands

- Low-level commands enhance existing plots with features like extra points, labels etc.
- These commands require coordinate positions also, to indicate the position where we have to insert the new plotting elements.
- Some of the low-level plotting commands are listed in the below table.

Function	**Description**
points (x,y)	Add points to the current plot
lines (x,y)	Add connecting lines to the current plot
text (x,y, labels, …)	Add text to the plot at point x and y. Lables is a vector in which labels[i] is at the point (x[i],y[i])
abline (a,b)	Adds a line of slope b and intercept a to the current plot
polygon (x, y,..)	Draws a polygon defined by the vertices (x,y)
legend (x,y, legend, fill=…)	Adds a legend to the current plot at the specified position. fill if specified, this argument will cause boxes filled with the specified colors (or shaded in the specified colors) to appear beside the legend text.
title (main, sub)	Adds a title main to the top of the current plot in a large font and (optionally) a sub-title sub at the bottom in a smaller font
axis (slide)	Adds an axis to the current plot on the slide given by the first argument

3.3. Graphical Parameters

- In addition to the low-level plotting commands, the presentation of graphics can be improved with graphical parameters.
- They can be used as options of graphical functions or with the function par to change permanently the graphical parameters, i.e. the subsequent plots will be drawn with respect to parameters specified by the user.

Parameter	Description
adj	Controls text justification with respect to the left border of the text so that 0 is left-justified, 0.5 – centered, 1 is right-justified, values > 1 move the text further to the left and negative values further to the right; if two values are given (Example: c(0,0)) the second one controls the vertical justification with respect to the text baseline.
bg	Specifies the color of the background For example; bg="red", bg="blue".; the list of 657 available colors is displayed with colors()
bty	A character string which is determined by the type of box which is drawn around the plots. Values are: "o" - default, "l","7","c","u",or "]" the resulting box resembles the corresponding upper case letter. A value of "n" suppresses the box.
Parameter	Description

cex	A value controlling the size of the texts and symbols with respect to the defaiult. cex.axis The magnification to be used for axis annotation relative to the current setting of cex. cex.lab The magnification to be used for x and y labels relative to the current setting of cex. cex.main The magnification to be used for main titles relative to the current setting of cex. cex.sub The magnification to be used for sub-titles relative to the current setting of cex.
col	A specification for the default plotting color. col.axis The color to be used for axis annotation. Defaults to "black". col.lab The color to be used for x and y labels. Defaults to "black". col.main The color to be used for plot main titles. Defaults to "black". col.sub The color to be used for plot sub-titles. Defaults to "black".
font	An integer which controls the style of text (1: Normal, 2: Italics, 3:bold, 4:bold italics) We have font.axis, font.lab, font.main, font.sub
Parameter	Description

las	numeric in {0,1,2,3}; the style of axis labels. 0: always parallel to the axis [default], 1: always horizontal, 2: always perpendicular to the axis, 3: always vertical.
lty	The line type. Line types can either be specified as an integer (0=blank, 1=solid (default), 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash) or as one of the character strings "blank", "solid", "dashed", "dotted", "dotdash", "longdash", or "twodash", where "blank" uses 'invisible lines' (i.e., does not draw them).
lwd	The line width, a positive number, defaulting to 1. The interpretation is device-specific, and some devices do not implement line widths less than one.
mar	A numerical vector of the form c(bottom, left, top, right) which gives the number of lines of margin to be specified on the four sides of the plot. The default is c(5, 4, 4, 2) + 0.1.
Parameter	Description
mfcol, mfrow	A vector of the form c(nr, nc). Subsequent figures will be drawn in an nr-by-nc array on the device by columns (mfcol), or rows (mfrow), respectively.
pch	Either an integer specifying a symbol or a single character to be used as the default in plotting points.
ps	integer; the point size of text (but not symbols).

pty	A character specifying the type of plot region to be used; "s" generates a square plotting region and "m" generates the maximal plotting region.
tck	The length of tick marks as a fraction of the smaller of the width or height of the plotting region. If tck >= 0.5 it is interpreted as a fraction of the relevant side, so if tck = 1 grid lines are drawn. The default setting (tck = NA) is to use tcl = -0.5.
Parameter	Description
tcl	The length of tick marks as a fraction of the height of a line of text. The default value is -0.5; setting tcl = NA sets tck = -0.01 which is S' default.
xaxt	A character which specifies the x axis type. Specifying "n" suppresses plotting of the axis. The standard value is "s"
yaxt	A character which specifies the y axis type. Specifying "n" suppresses plotting.

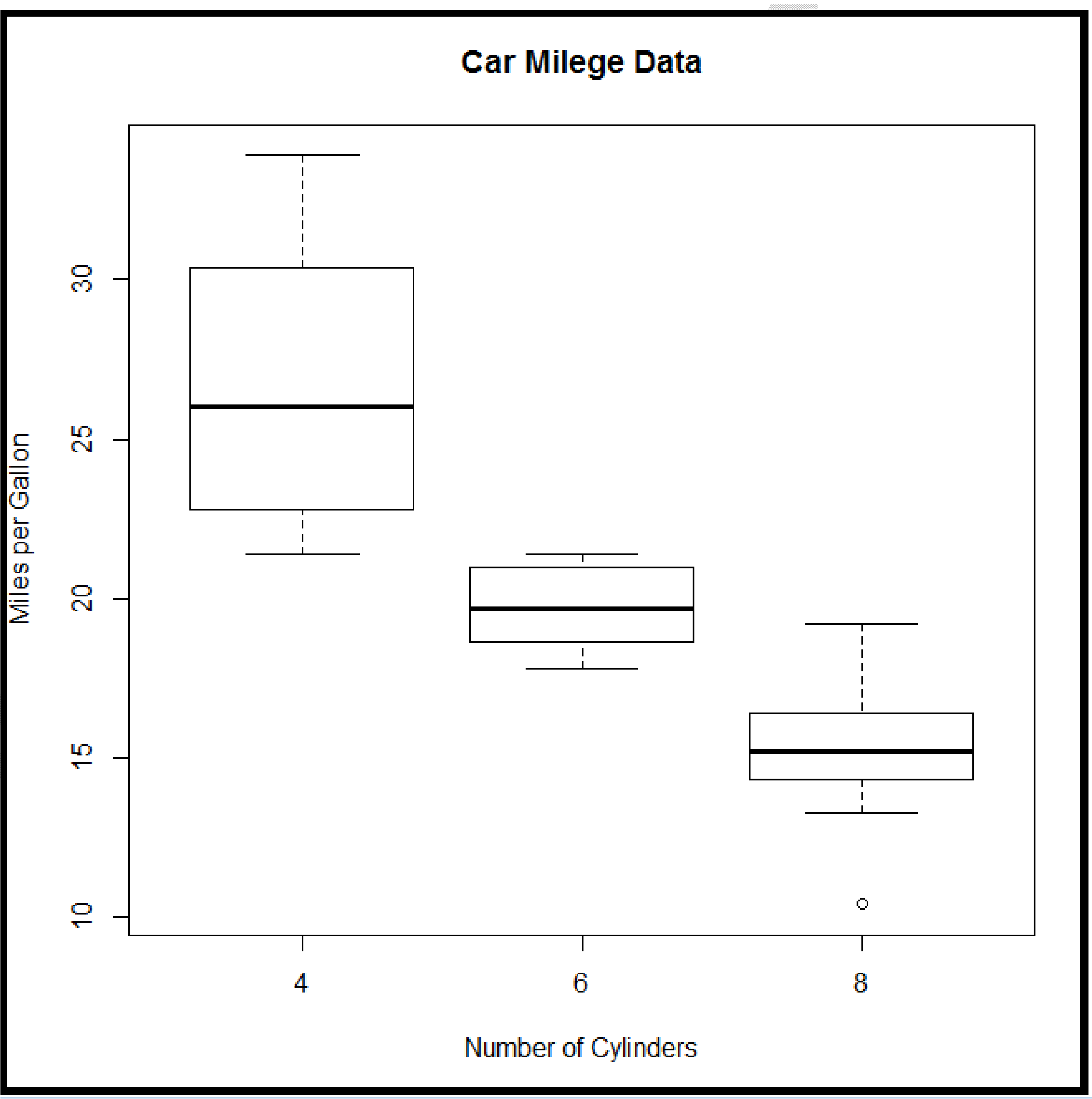
3.4. Examples

3.4.1. Boxplots

- The basic assumption in statistics is that a set of data has a central tendency. That means the number of the data are distributed around some central value. The box of the box-whisker plot takes care of the middle half observations of the data.
- A boxplot splits the data set into quartiles. The body of the boxplot consists of a "box", which goes from the first quartile (Q1) to the third quartile (Q3). Within the box, a vertical line is drawn at the median of the data set (Q2). Two horizontal lines, called whiskers, extend from the front and back of the box. The front whisker goes from Q1 to the smallest non-outlier in the data set, and the back whisker goes from Q3 to the largest non-outlier.
- Boxplots can be created for individual variables or for variables by group. The format is boxplot(x,data=), where x is a formula and data= denotes the data frame providing the data.
- An example of a formula is y-group where a separate boxplot for numeric variable y is generated for each value of group.

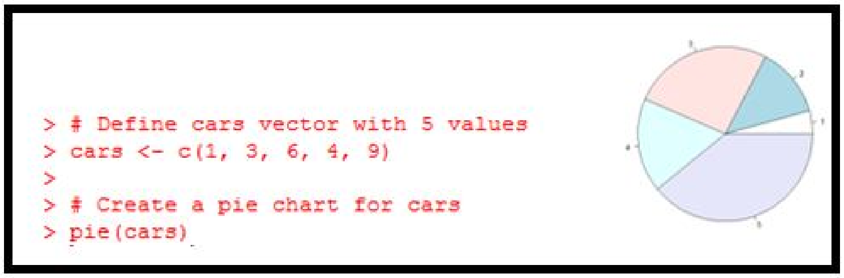
- Add varwidth=TRUE to make boxplot widths proportional to the square root of the sample sizes.
- Add horizontal=TRUE to reverse the axis orientation.

```
> setwd("D:/R")
> # boxplot of MPG by car cylinders
> boxplot(mpg ~ cyl, data=mtcars, main="Car Milege Data",
+ xlab="Number of Cylinders",
+ ylab="Miles per Gallon")
```

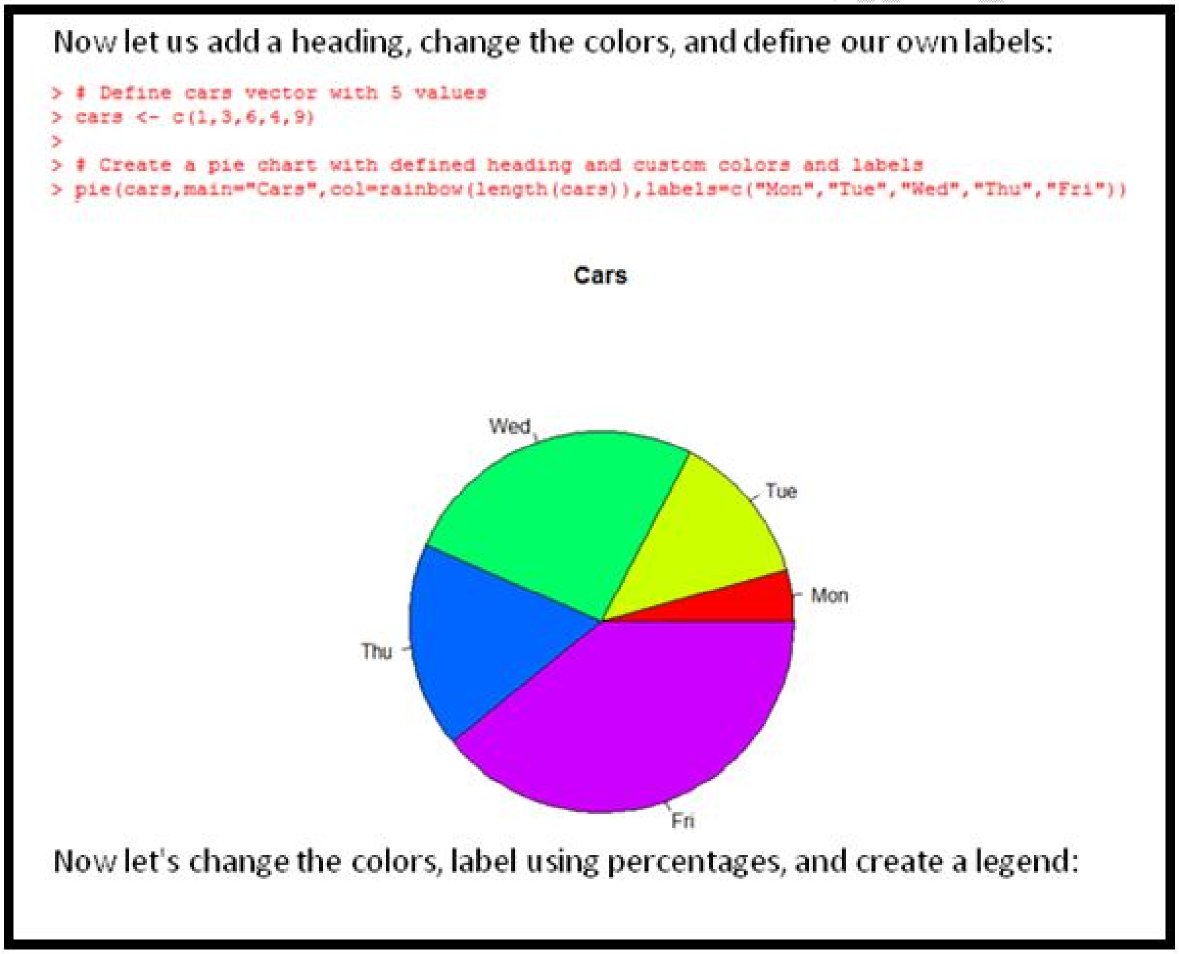


3.4.2. Pie Charts

- A circle graph or pie chart is a way of summarizing a set of categorical data or displaying the different values of a given variable (e.g., percentage distribution).
- This type of chart is a circle divided into a set of series of segments. Each segment represents a particular category. The area of the segment is the same proportion of the circle as the category is of the total data set.



```
> # Define cars vector with 5 values
> cars <- c(1, 3, 6, 4, 9)
>
> # Create a pie chart for cars
> pie(cars)
```

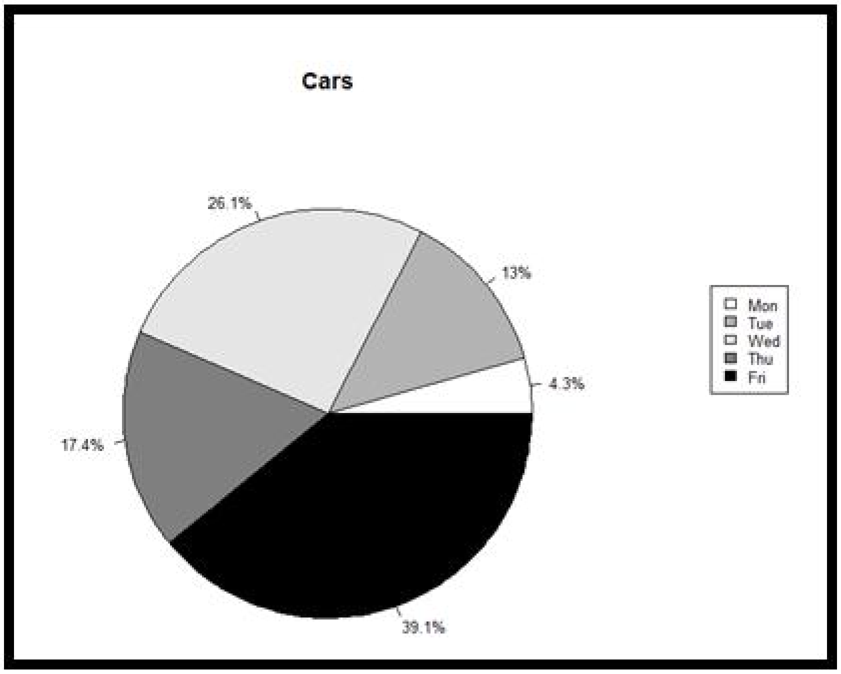


Now let us add a heading, change the colors, and define our own labels:

```
> # Define cars vector with 5 values
> cars <- c(1,3,6,4,9)
>
> # Create a pie chart with defined heading and custom colors and labels
> pie(cars,main="Cars",col=rainbow(length(cars)),labels=c("Mon","Tue","Wed","Thu","Fri"))
```

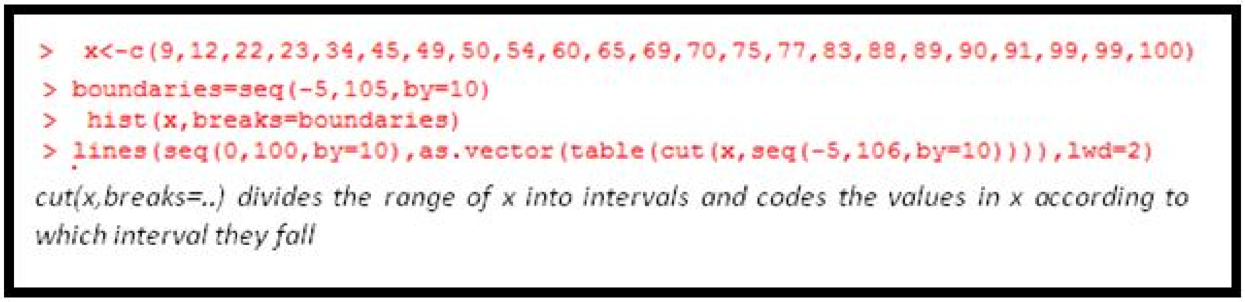
Now let's change the colors, label using percentages, and create a legend:

```
> # Define cars vector with 5 values
> cars <- c(1, 3, 6, 4, 9)
>
> # Define some colors ideal for black & white print
> colors <- c("white","grey70","grey90","grey50","black")
>
> # Calculate the percentage for each day, rounded to one
> # decimal place
> car_labels <- round(cars/sum(cars) * 100, 1)
>
> # Concatenate a '%' char after each value
> car_labels <- paste(car_labels, "%", sep="")
>
> # Create a pie chart with defined heading and custom colors
> # and labels
> pie(cars, main="Cars", col=colors, labels=car_labels,
+    cex=0.8)
>
> # Create a legend at the right
> legend(1.5, 0.5, c("Mon","Tue","Wed","Thu","Fri"), cex=0.8,
+    fill=colors)
>
```



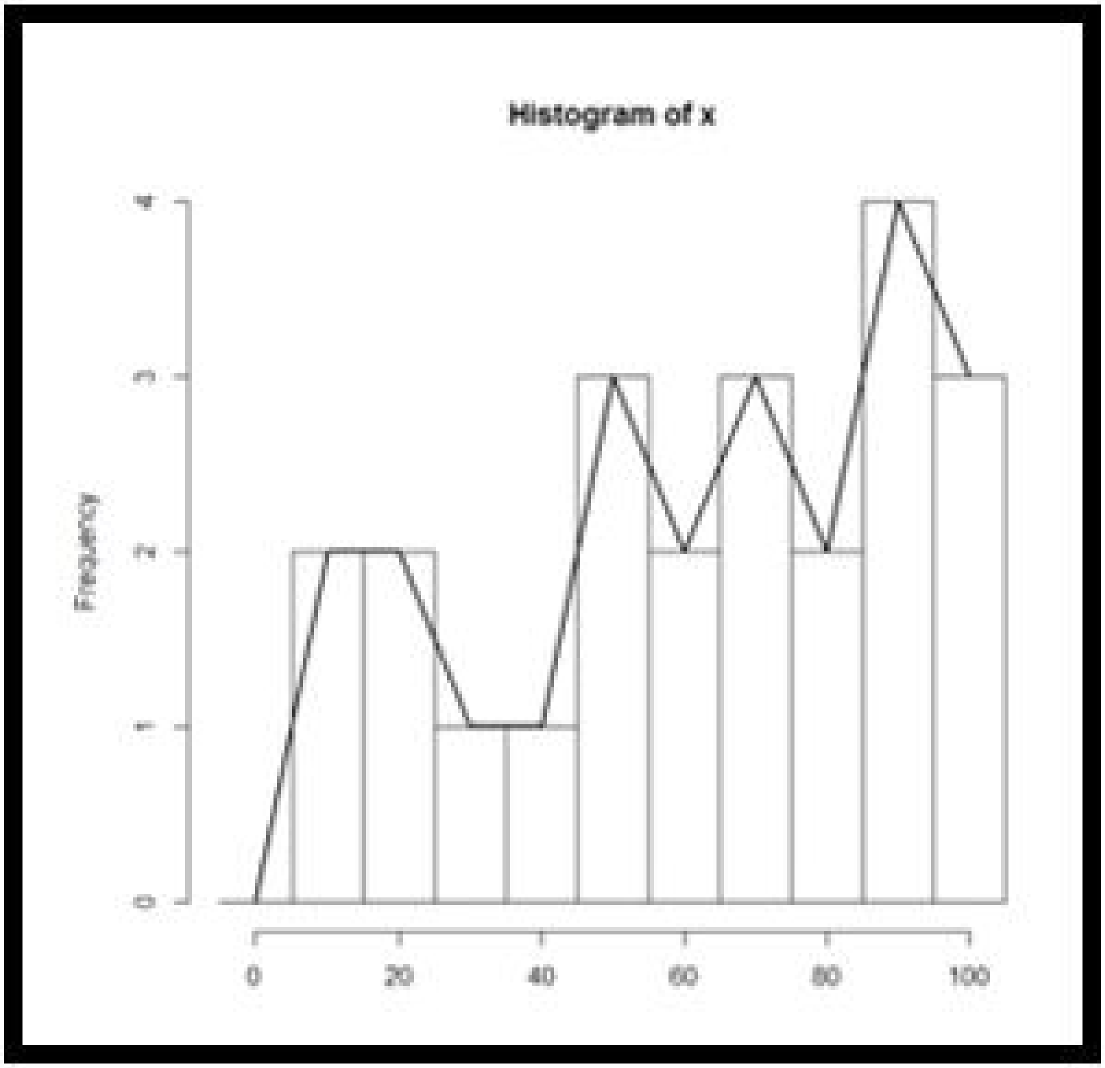
3.4.3. Histograms

- A histogram is a type of graph that has wide applications in statistics. Histograms allow a visual interpretation of numerical data by indicating the number of data points that lie within a range of values, called a class or bin. The frequency of the data that falls in each class is depicted by the use of a bar.

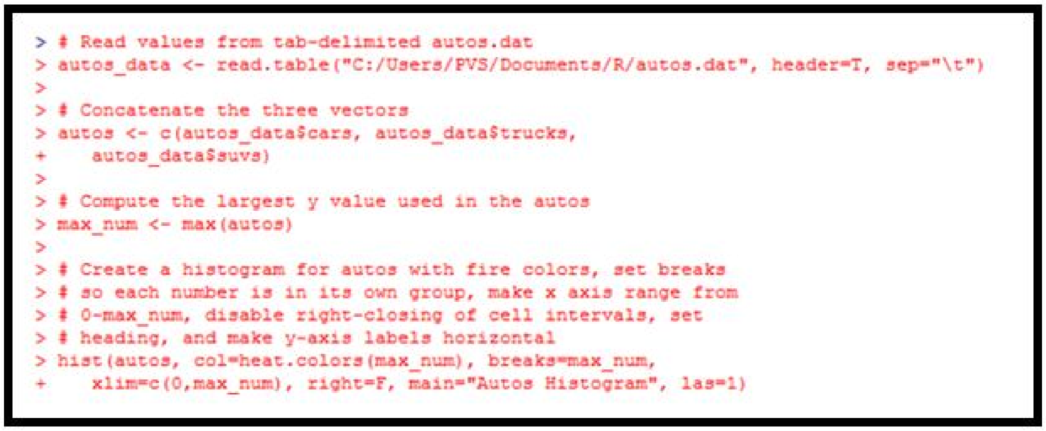


```
>  x<-c(9,12,22,23,34,45,49,50,54,60,65,69,70,75,77,83,88,89,90,91,99,99,100)
> boundaries=seq(-5,105,by=10)
>  hist(x,breaks=boundaries)
> lines(seq(0,100,by=10),as.vector(table(cut(x,seq(-5,106,by=10)))),lwd=2)
```

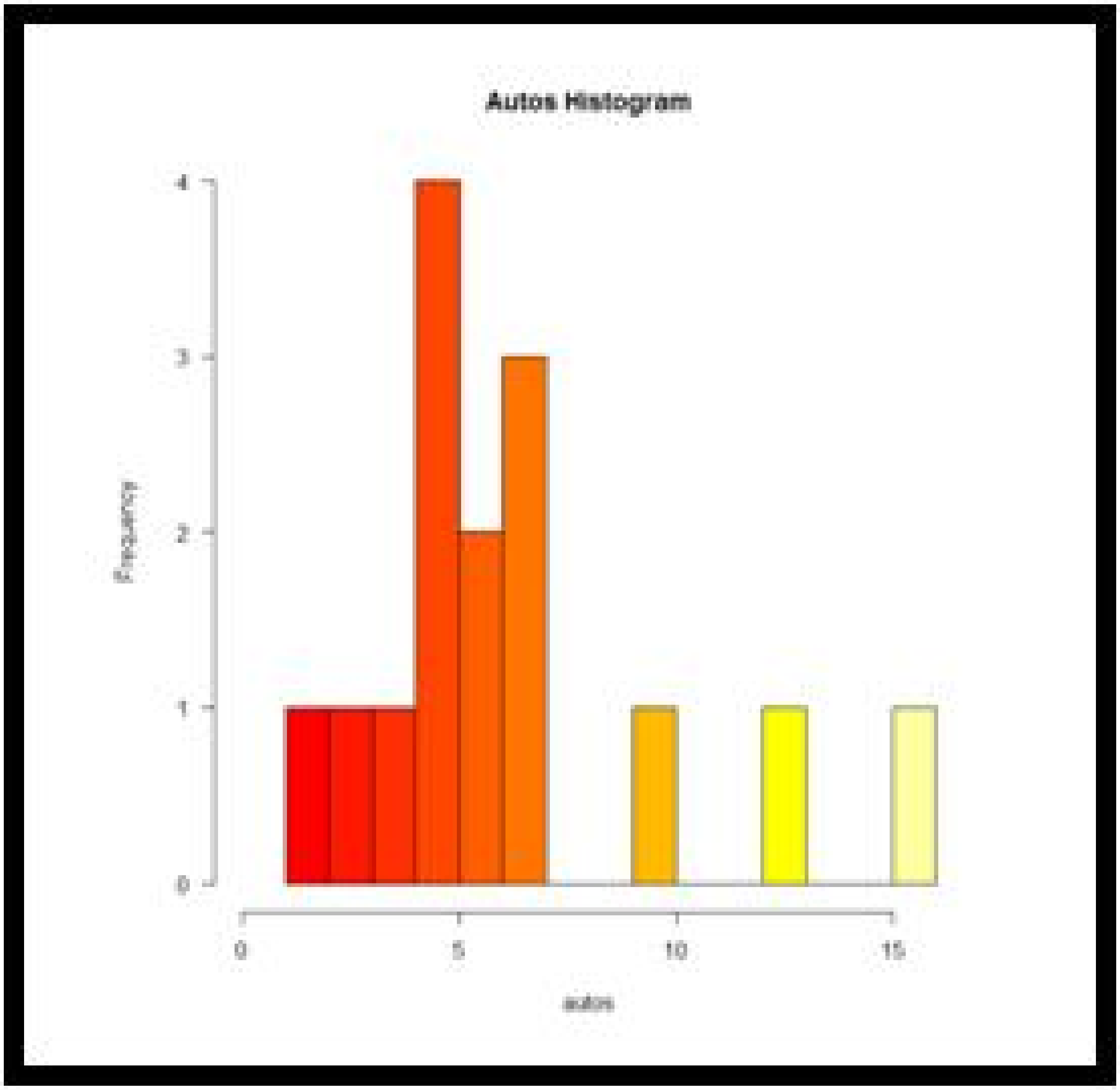
cut(x,breaks=..) divides the range of x into intervals and codes the values in x according to which interval they fall



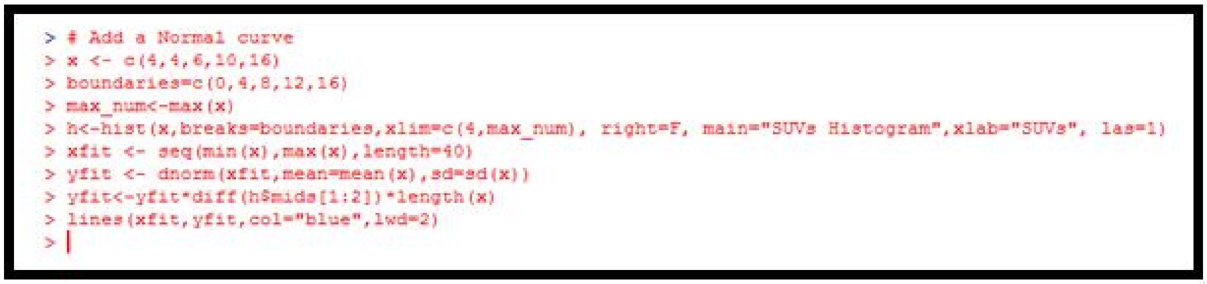
We will try another example.



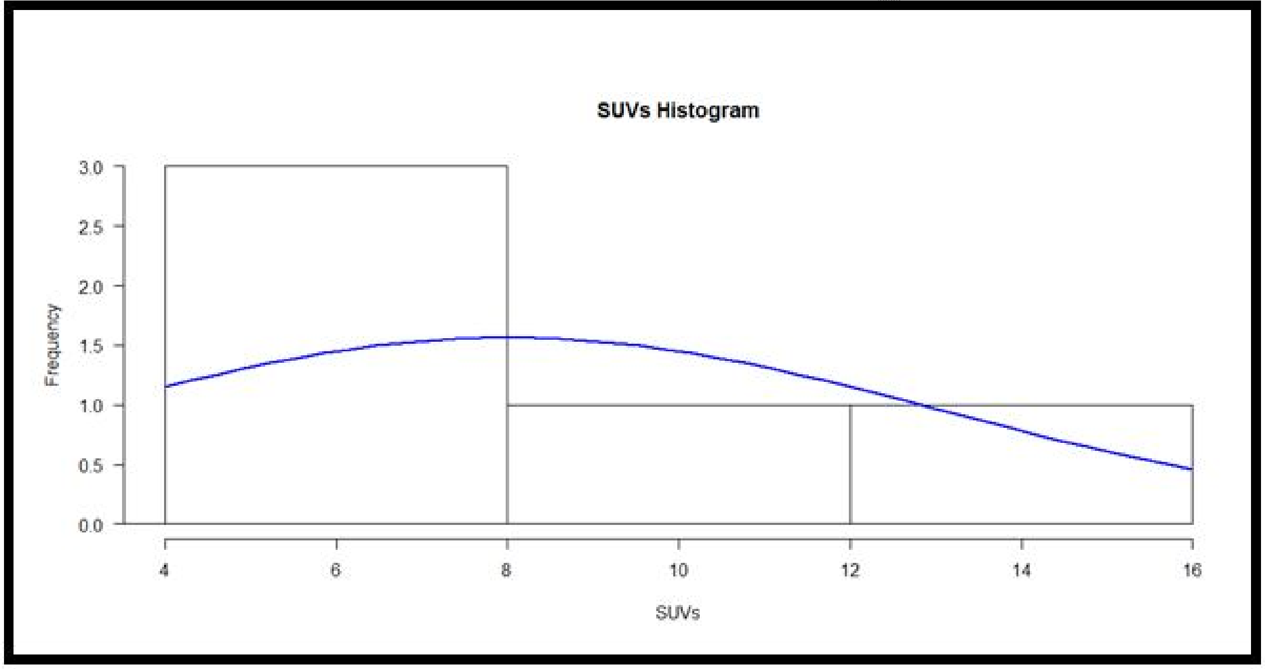
```
> # Read values from tab-delimited autos.dat
> autos_data <- read.table("C:/Users/PVS/Documents/R/autos.dat", header=T, sep="\t")
>
> # Concatenate the three vectors
> autos <- c(autos_data$cars, autos_data$trucks,
+    autos_data$suvs)
>
> # Compute the largest y value used in the autos
> max_num <- max(autos)
>
> # Create a histogram for autos with fire colors, set breaks
> # so each number is in its own group, make x axis range from
> # 0-max_num, disable right-closing of cell intervals, set
> # heading, and make y-axis labels horizontal
> hist(autos, col=heat.colors(max_num), breaks=max_num,
+    xlim=c(0,max_num), right=F, main="Autos Histogram", las=1)
```



We will try another example.

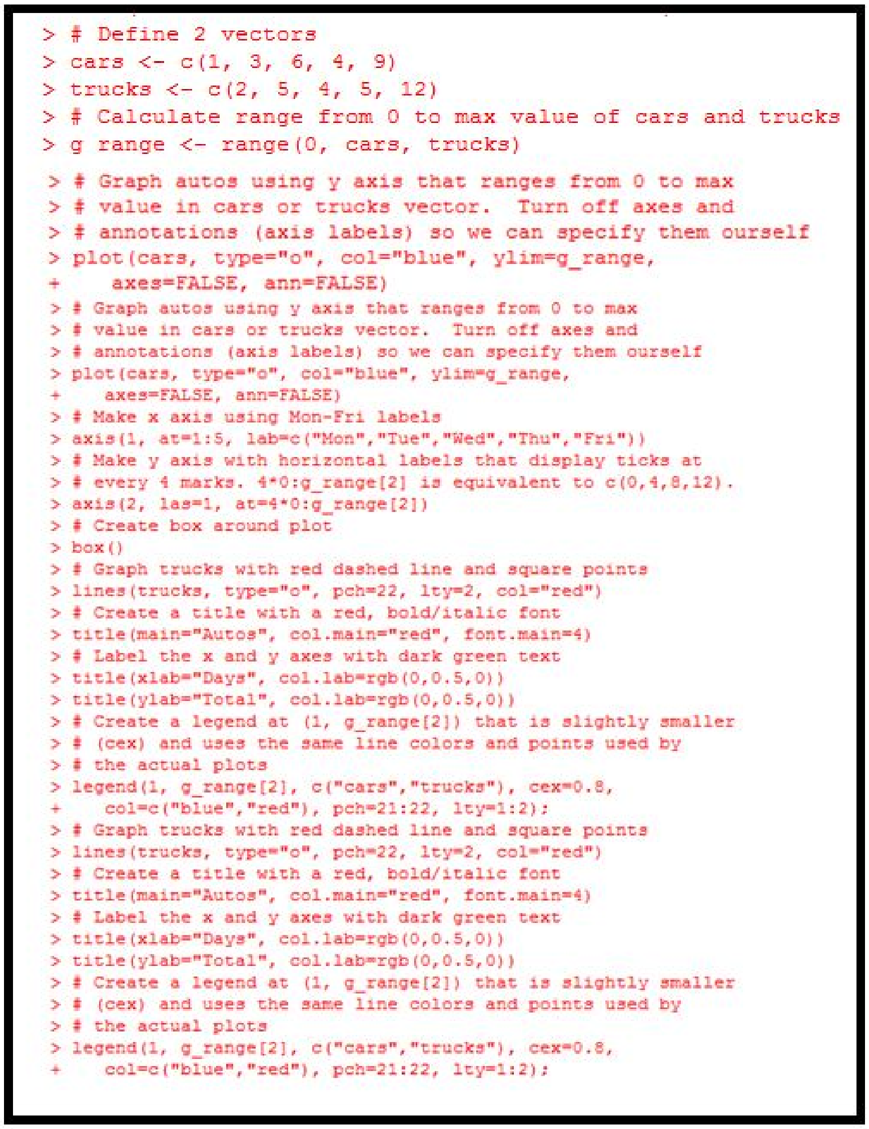


```
> # Add a Normal curve
> x <- c(4,4,6,10,16)
> boundaries=c(0,4,8,12,16)
> max_num<-max(x)
> h<-hist(x,breaks=boundaries,xlim=c(4,max_num), right=F, main="SUVs Histogram",xlab="SUVs", las=1)
> xfit <- seq(min(x),max(x),length=40)
> yfit <- dnorm(xfit,mean=mean(x),sd=sd(x))
> yfit<-yfit*diff(h$mids[1:2])*length(x)
> lines(xfit,yfit,col="blue",lwd=2)
> |
```

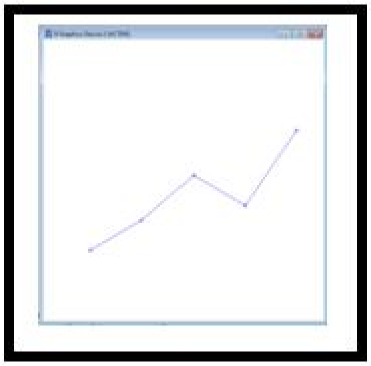


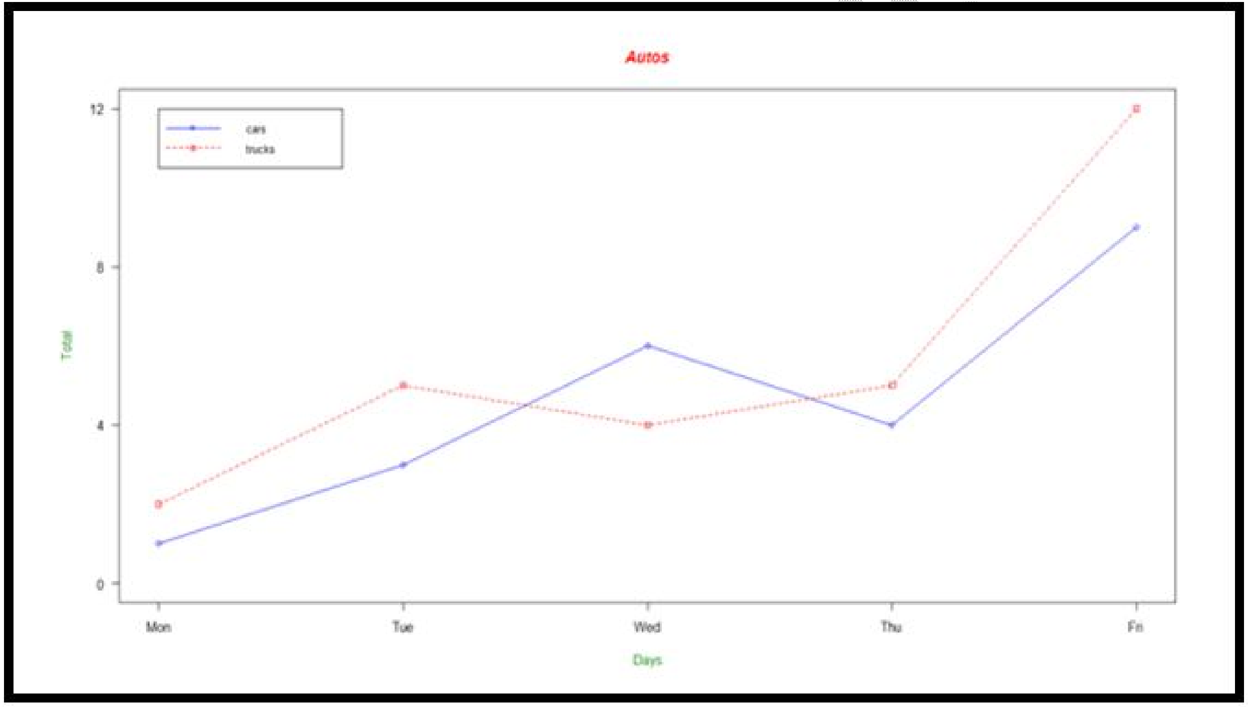
3.4.4. Line Charts

- A line chart or line graph is a type of chart which displays information as a series of data points called 'markers' connected by straight line segments. It is a basic type of chart common in many fields. It is a similar to a scatter plot except that the measurement points are ordered (typically by their x-axis value) and joined with straight line segments.
- Line charts show how a particular data changes at equal intervals of time. A line chart is often used to visualize a trend in data over intervals of time - a time series - thus the line is often drawn chronologically.



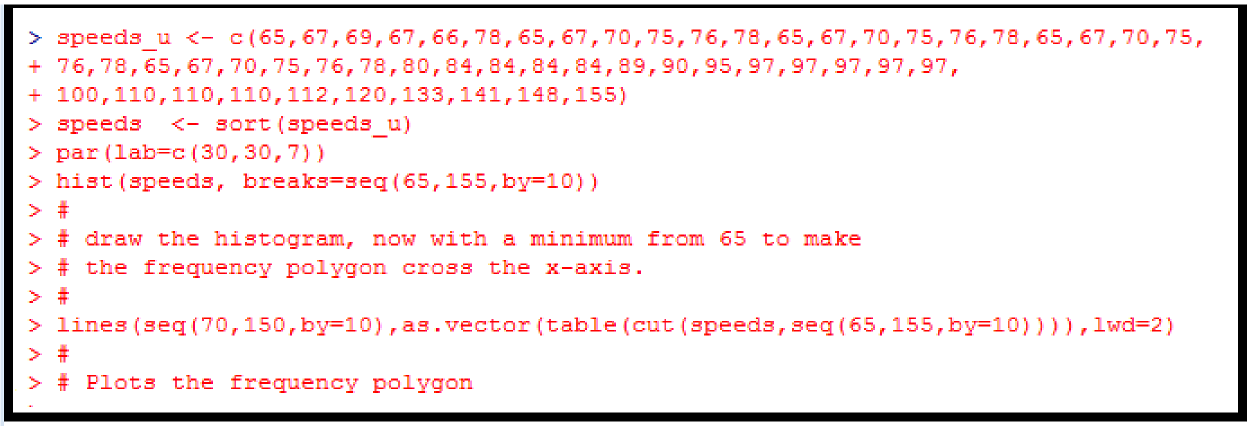
```
> # Define 2 vectors
> cars <- c(1, 3, 6, 4, 9)
> trucks <- c(2, 5, 4, 5, 12)
> # Calculate range from 0 to max value of cars and trucks
> g range <- range(0, cars, trucks)
> # Graph autos using y axis that ranges from 0 to max
> # value in cars or trucks vector.  Turn off axes and
> # annotations (axis labels) so we can specify them ourself
> plot(cars, type="o", col="blue", ylim=g_range,
+     axes=FALSE, ann=FALSE)
> # Graph autos using y axis that ranges from 0 to max
> # value in cars or trucks vector.  Turn off axes and
> # annotations (axis labels) so we can specify them ourself
> plot(cars, type="o", col="blue", ylim=g_range,
+     axes=FALSE, ann=FALSE)
> # Make x axis using Mon-Fri labels
> axis(1, at=1:5, lab=c("Mon","Tue","Wed","Thu","Fri"))
> # Make y axis with horizontal labels that display ticks at
> # every 4 marks. 4*0:g_range[2] is equivalent to c(0,4,8,12).
> axis(2, las=1, at=4*0:g_range[2])
> # Create box around plot
> box()
> # Graph trucks with red dashed line and square points
> lines(trucks, type="o", pch=22, lty=2, col="red")
> # Create a title with a red, bold/italic font
> title(main="Autos", col.main="red", font.main=4)
> # Label the x and y axes with dark green text
> title(xlab="Days", col.lab=rgb(0,0.5,0))
> title(ylab="Total", col.lab=rgb(0,0.5,0))
> # Create a legend at (1, g_range[2]) that is slightly smaller
> # (cex) and uses the same line colors and points used by
> # the actual plots
> legend(1, g_range[2], c("cars","trucks"), cex=0.8,
+    col=c("blue","red"), pch=21:22, lty=1:2);
> # Graph trucks with red dashed line and square points
> lines(trucks, type="o", pch=22, lty=2, col="red")
> # Create a title with a red, bold/italic font
> title(main="Autos", col.main="red", font.main=4)
> # Label the x and y axes with dark green text
> title(xlab="Days", col.lab=rgb(0,0.5,0))
> title(ylab="Total", col.lab=rgb(0,0.5,0))
> # Create a legend at (1, g_range[2]) that is slightly smaller
> # (cex) and uses the same line colors and points used by
> # the actual plots
> legend(1, g_range[2], c("cars","trucks"), cex=0.8,
+    col=c("blue","red"), pch=21:22, lty=1:2);
```



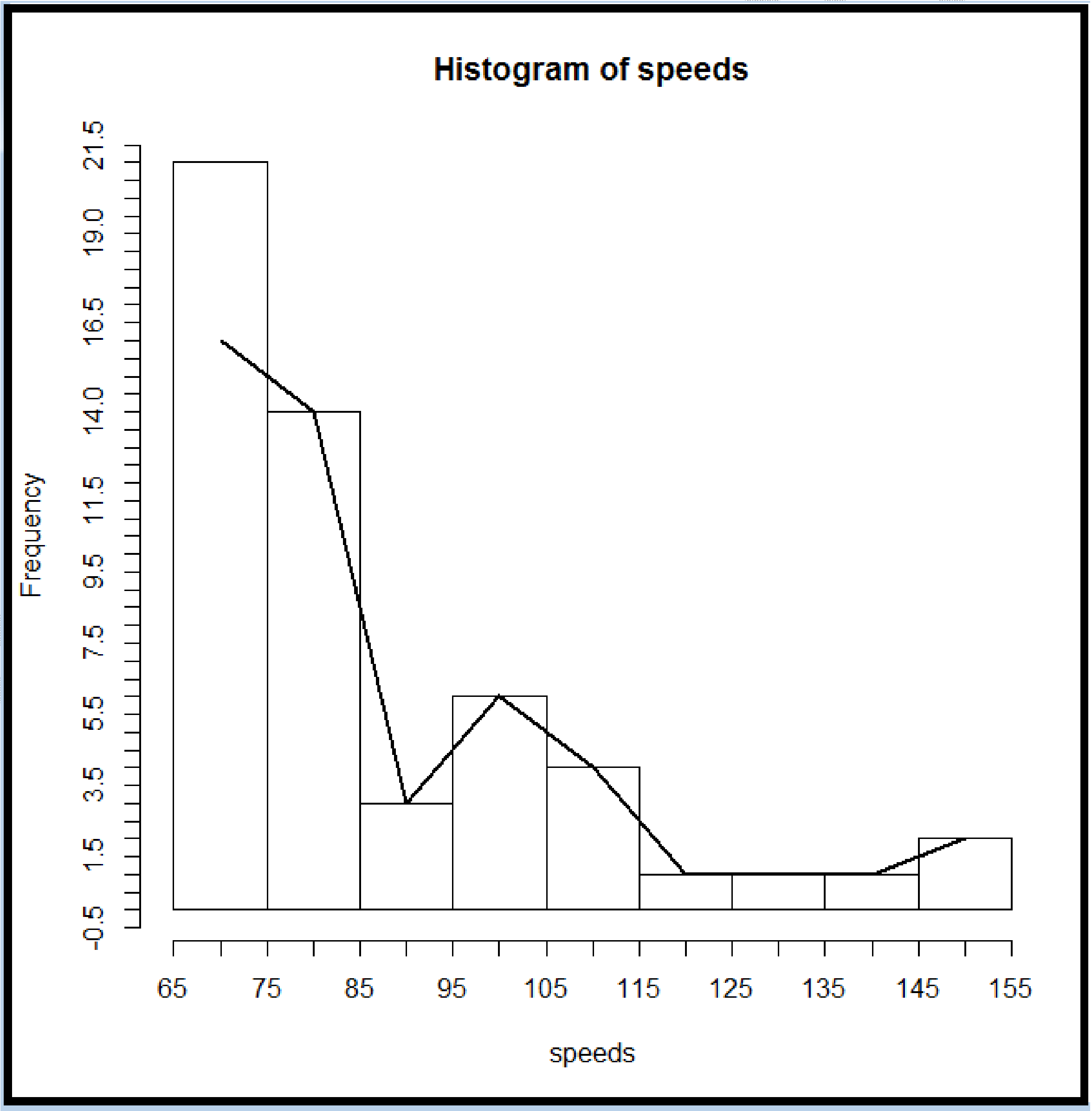


3.4.5. Frequency Polygons

- In a frequency polygon, a line graph is drawn by joining all the midpoints of the top of the bars of a histogram.
- A frequency polygon gives the idea about the shape of the data distribution. The two end points of a frequency polygon always lie on the x-axis.

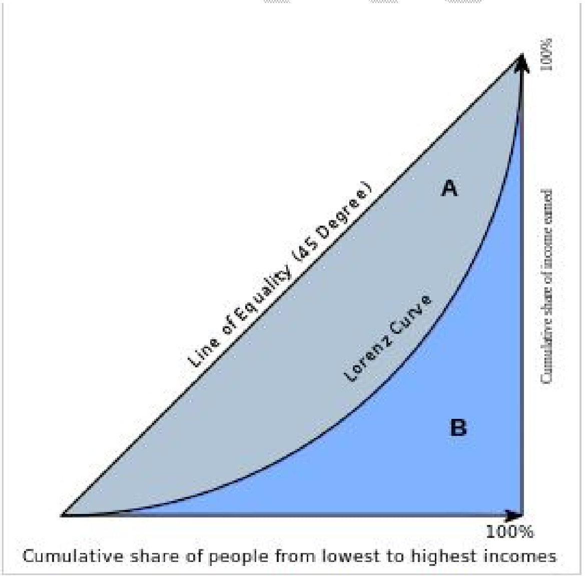


```
> speeds_u <- c(65,67,69,67,66,78,65,67,70,75,76,78,65,67,70,75,76,78,65,67,70,75,
+ 76,78,65,67,70,75,76,78,80,84,84,84,84,89,90,95,97,97,97,97,97,
+ 100,110,110,110,112,120,133,141,148,155)
> speeds  <- sort(speeds_u)
> par(lab=c(30,30,7))
> hist(speeds, breaks=seq(65,155,by=10))
> #
> # draw the histogram, now with a minimum from 65 to make
> # the frequency polygon cross the x-axis.
> #
> lines(seq(70,150,by=10),as.vector(table(cut(speeds,seq(65,155,by=10)))),lwd=2)
> #
> # Plots the frequency polygon
```



3.4.6. Lorenz Curve

- Lorenz Curve is a graphical method of studying dispersion.
- It is a percentage of cumulative curve in which the percentage of cumulative values of one variable is combined with the percentage of cumulative values in the other variable.
- If two curves are drawn on the same graph, it is possible to compare the variability of the distribution.
- The curve further away from the diagonal line measures greater variability.
- The Gini index (a.k.a. Gini coefficient) is usually defined mathematically based on the Lorenz curve, which plots the proportion of the total income of the population (y-axis) that is cumulatively earned by the bottom x% of the population.
- The line at 45 degrees thus represents perfect equality of incomes.
- The Gini index can then be thought of as the ratio of the area that lies between the line of equality and the Lorenz curve (marked A in the diagram) over the total area under the line of equality (marked A and B in the diagram); i.e. G = A / (A+B).


100%
A
Line of Equality (45 Degree)
Lorenz Curve
Cumulative share of income earned
B
100%
Cumulative share of people from lowest to highest incomes

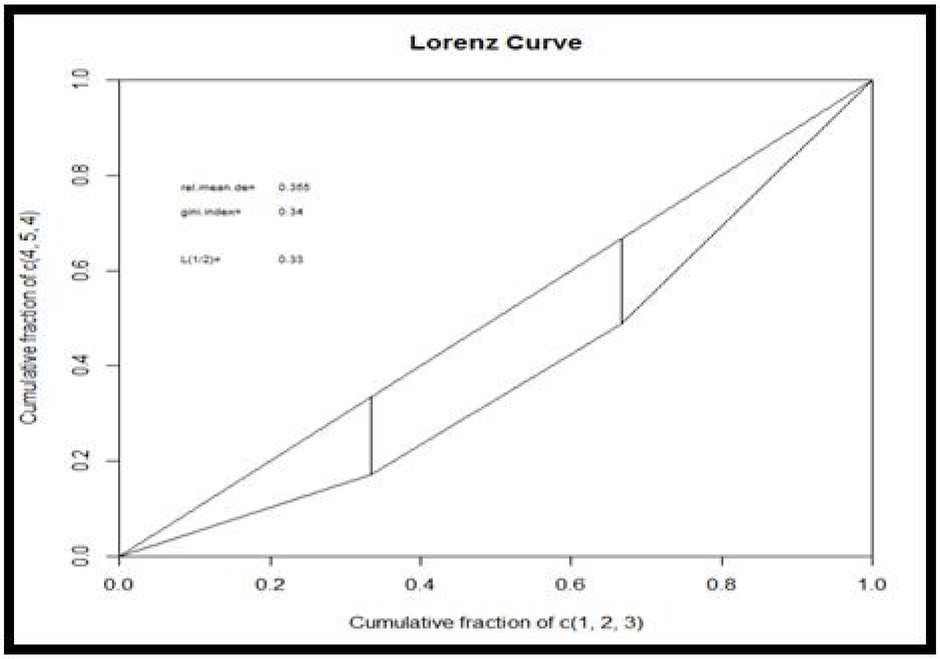
Graphical representation of the Gini coefficient

The graph shows that the Gini coefficient is equal to the area marked *A* divided by the sum of the areas marked *A* and *B*. that is, Gini = *A* / (*A* + *B*). It is also equal to 2**A* due to the fact that *A* + *B* = 0.5 (since the axes scale from 0 to 1).

Reference: Wikipedia The Free Encyclopedia

Let us try the following R code:

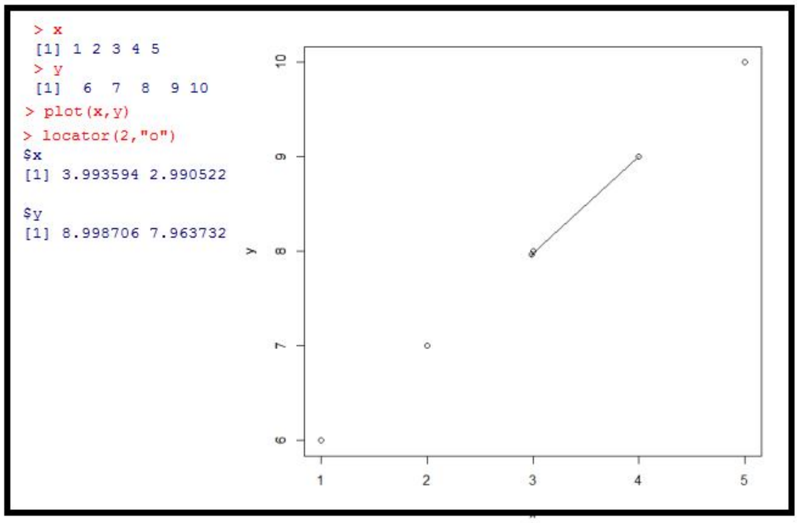
```
> require("lawstat")
> lorenz.curve(c(1,2,3),c(4,5,4))
```



3.5. Interactive graphics

- The interactive graphics facility in R allows the user to extract or add required information to a plot.
- For example, consider locator () function.
- The locator () reads the position of the graphics cursor when the mouse button is pressed. This continues until n locations are selected. Here type argument has to chose any one of the values from "n","p","o". If "p" or "o" the points are plotted; if "l" or "o" they are joined by lines.

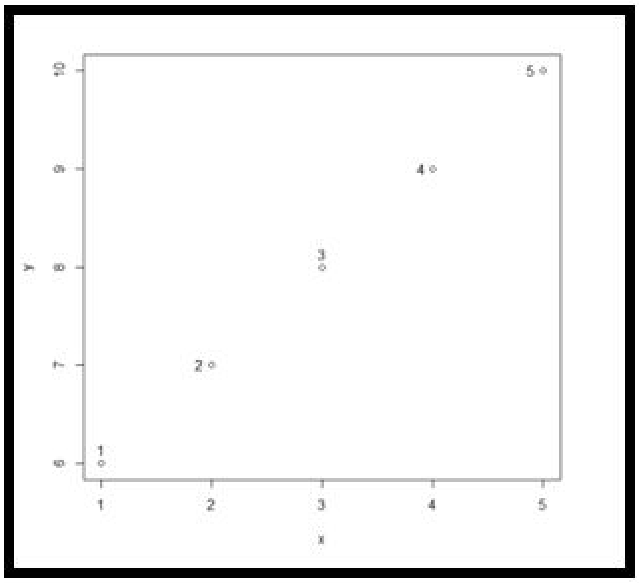
  Syntax is locator(n,type,...).



```
> x
[1] 1 2 3 4 5
> y
[1]  6  7  8  9 10
> plot(x,y)
> locator(2,"o")
$x
[1] 3.993594 2.990522

$y
[1] 8.998706 7.963732
```

- As another example, see the function identify().
- This function reads the position of the graphics pointer when the mouse button is pressed.
- It searches for x and y coordinates closest to the pointer. If this pointer is close enough to the pointer its index will be returned otherwise warning is given. Syntax is identity(x,y,labels).

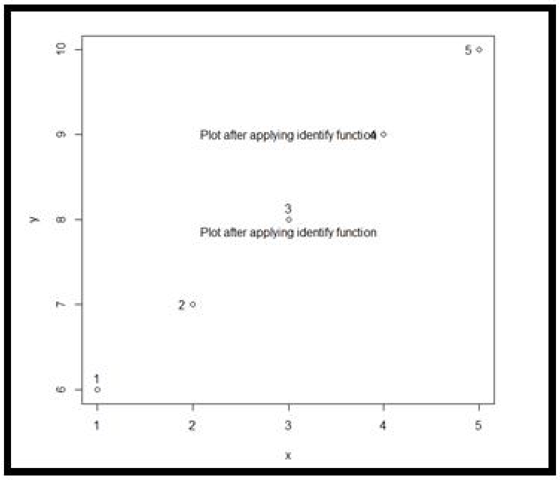
```
> plot(x,y)
> identify(x,y)
```



- See the example given above and the resultant plot after identifying the coordinates.
- If you press the mouse button when the pointer is not close enough to the coordinate, a warning message is displayed. For example, when the pointer is between the coordinates 2 and 3 and the mouse button is pressed, a warning message is displayed.

```
warning: no point within 0.25 inches
```

```
> identify(x,y,"Plot after applying identify function")
```



4. Packages - grid and lattice

- The packages grid and lattice implement the grid and lattice systems.
- Grid is a new graphical mode with its own system of graphical parameters which are distinct from those seen earlier.
- The two main distinctions of grid compared to the base graphics are:
  1. A more flexible to split graphical devices using view points which could be overpalling (graphical objects may even be shared among distinct view points e.g. arrows);
  2. graphical objects (grob) may be modified or removed from a graph without requiring the re-draw all the graph (as must be done with base graphics).
- Grid graphics cannot usually be combined or mixed with base graphics (the gridBase package must be used to do this). However, it is possible to use both graphical modes in the same session on the same graphical device.
- Lattice is essentially the implementation in R of the Trellis graphics of S-PLUS.
- Trellis is an approach for visualizing multivariate data which is particularly appropriate for the exploration of relations or interactions among variables.
- The main idea behind lattice (and Trellis as well) is that of conditional multiple graphs; a bi-variate graph will be split in several graphs with respect to the values of a third variable.
- The function coplot uses a similar approach, but lattice offers wider functionalities.
- Lattice uses the grid graphical mode.
- The lattice package, written by Depayan Sarkar, attempts to improve on base R graphics by providing better defaults and the ability to easily display multivariate relationships.
- In particular, the package supports the creation of trellis - graphs that display a variable or the relationship between variables, conditioned on one or other variables.
- The typical format is graph_type(formula, data=) where graph_type is selected from the listed below:

  formula specifies the variable(s) to display and any conditioning variables.
- For example

i. ~ x | A means display numeric variable x for each level of factor A.

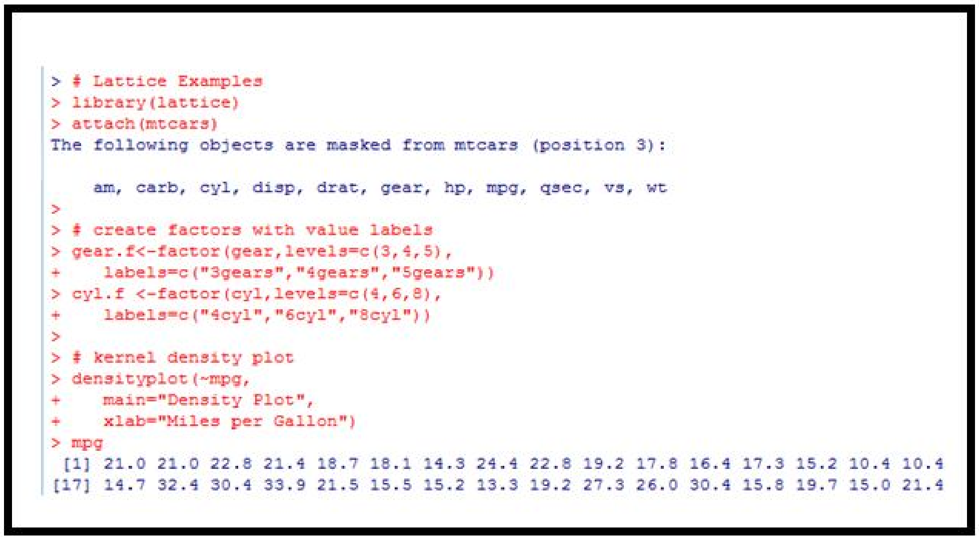
ii. y ~ x | A*B means display the relationship between numeric variables y and x separately for every combination of factor A and B levels.

iii. ~x means display numeric variable x alone.

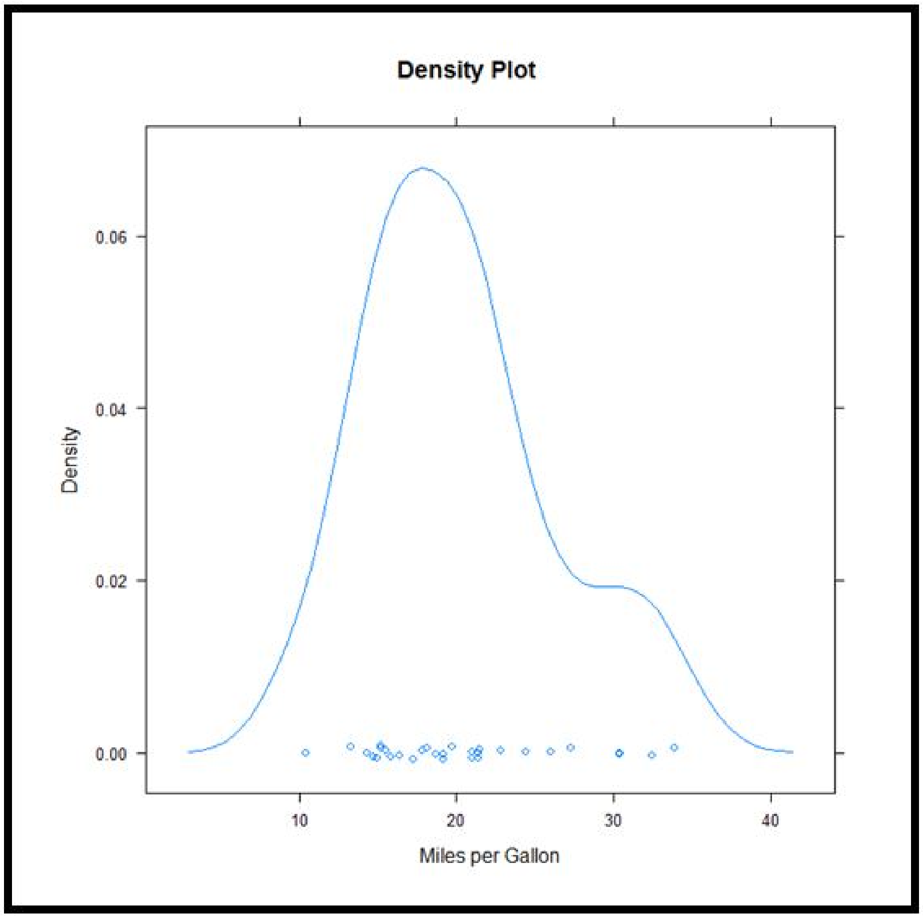
graph_type	description	formula examples
barchart	barchart	x ~ A or A ~ x

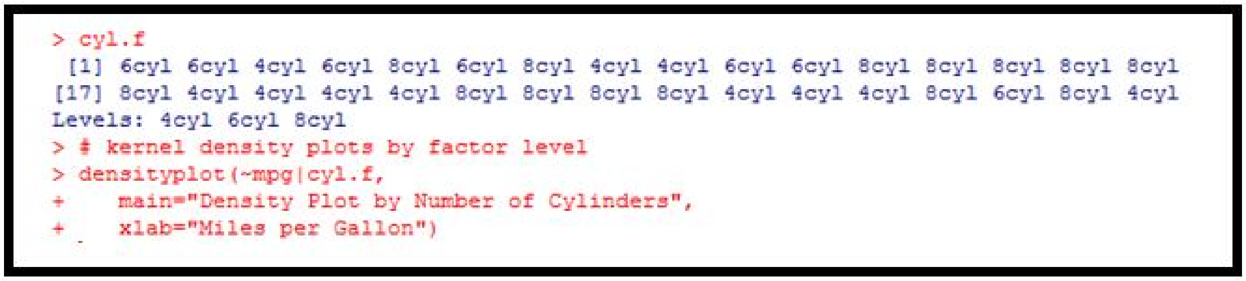
bwplot	boxplot	x ~ A or A ~ x
cloud	3D scatter plot	z ~x * y | A
contourplot	3D contour plot	z ~ x * y
densityplot	kernal density plot	~x | A * B
dotplot	dotplot	~x | A
graph_type	description	formula examples
histogram	Histogram	z ~ y * x
parallel	parallel coordinats plot	data frame
splom	scatter plot matrix	data frame
stripplot	strip plots	A ~x or x ~ A
xyplot	scatterplot	y ~ x | A
wireframe	3D wireframe graph	z ~ y * x

- Here are some examples. They use the car data (mileage, weight, number of gears, number of cylinders, etc.) from the mtcars data frame.

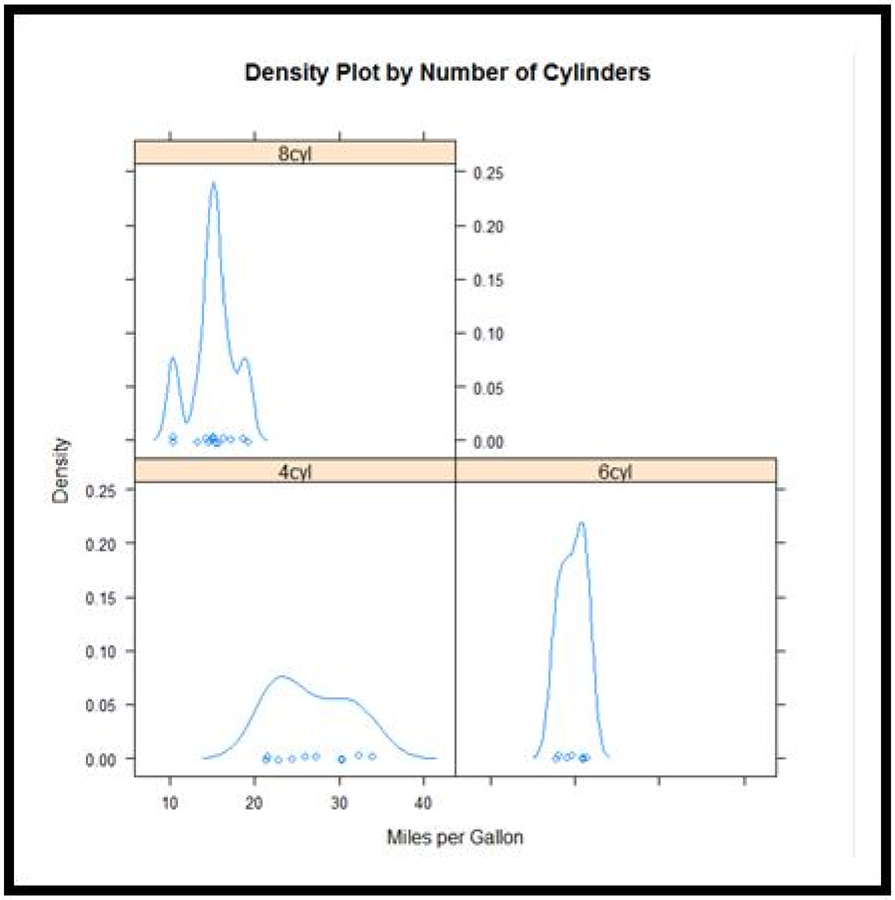


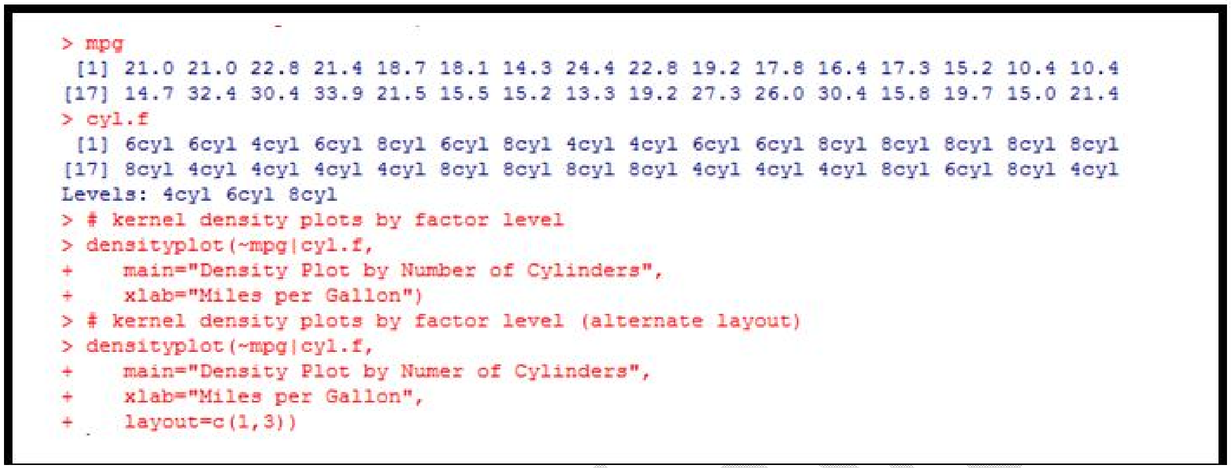
```
> # Lattice Examples
> library(lattice)
> attach(mtcars)
The following objects are masked from mtcars (position 3):

    am, carb, cyl, disp, drat, gear, hp, mpg, qsec, vs, wt

>
> # create factors with value labels
> gear.f<-factor(gear,levels=c(3,4,5),
+    labels=c("3gears","4gears","5gears"))
> cyl.f <-factor(cyl,levels=c(4,6,8),
+    labels=c("4cyl","6cyl","8cyl"))
>
> # kernel density plot
> densityplot(~mpg,
+    main="Density Plot",
+    xlab="Miles per Gallon")
> mpg
 [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4 10.4
[17] 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7 15.0 21.4
```



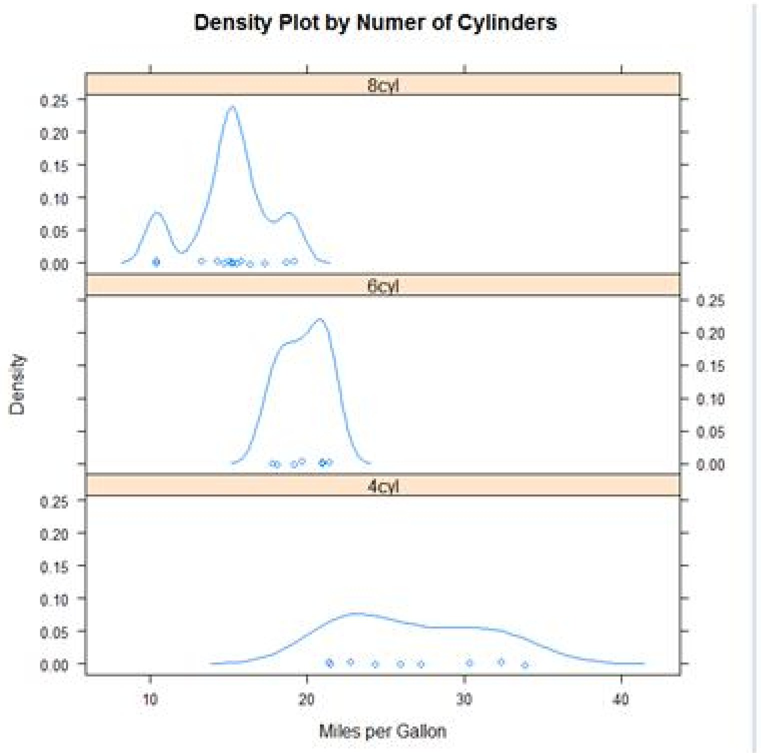


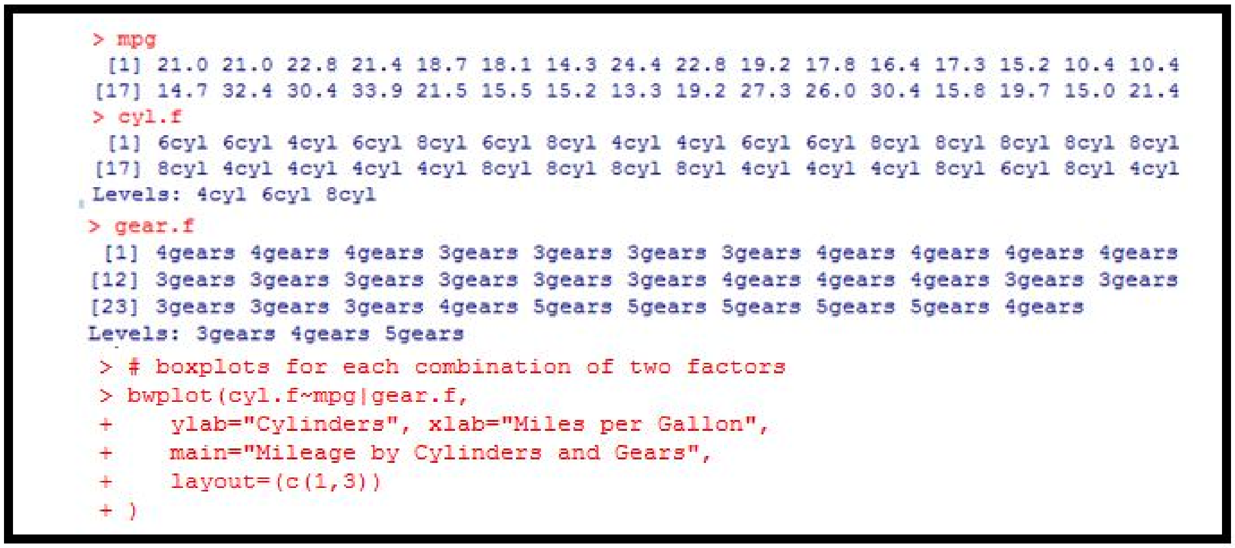
```
> cyl.f
 [1] 6cyl 6cyl 4cyl 6cyl 8cyl 6cyl 8cyl 4cyl 4cyl 6cyl 6cyl 8cyl 8cyl 8cyl 8cyl 8cyl
[17] 8cyl 4cyl 4cyl 4cyl 4cyl 8cyl 8cyl 8cyl 8cyl 4cyl 4cyl 4cyl 8cyl 6cyl 8cyl 4cyl
Levels: 4cyl 6cyl 8cyl
> # kernel density plots by factor level
> densityplot(~mpg|cyl.f,
+     main="Density Plot by Number of Cylinders",
+     xlab="Miles per Gallon")
+
```



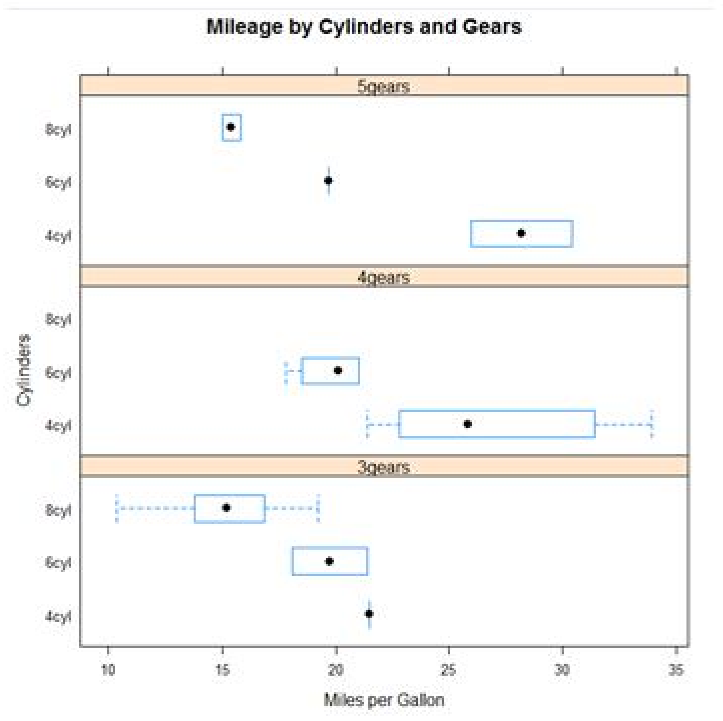


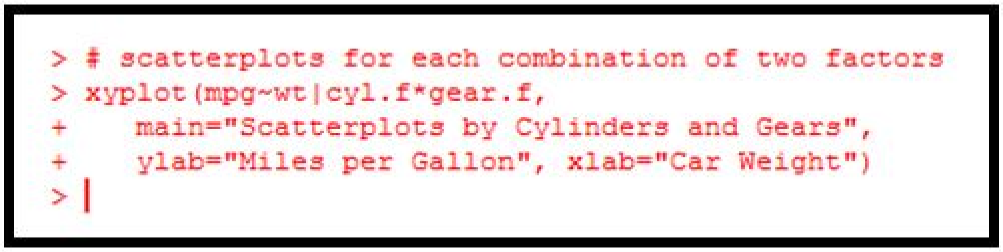
```
> mpg
 [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4 10.4
[17] 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7 15.0 21.4
> cyl.f
 [1] 6cyl 6cyl 4cyl 6cyl 8cyl 6cyl 8cyl 4cyl 4cyl 6cyl 6cyl 8cyl 8cyl 8cyl 8cyl 8cyl
[17] 8cyl 4cyl 4cyl 4cyl 4cyl 8cyl 8cyl 8cyl 8cyl 4cyl 4cyl 4cyl 8cyl 6cyl 8cyl 4cyl
Levels: 4cyl 6cyl 8cyl
> # kernel density plots by factor level
> densityplot(~mpg|cyl.f,
+     main="Density Plot by Number of Cylinders",
+     xlab="Miles per Gallon")
> # kernel density plots by factor level (alternate layout)
> densityplot(~mpg|cyl.f,
+     main="Density Plot by Numer of Cylinders",
+     xlab="Miles per Gallon",
+     layout=c(1,3))
```



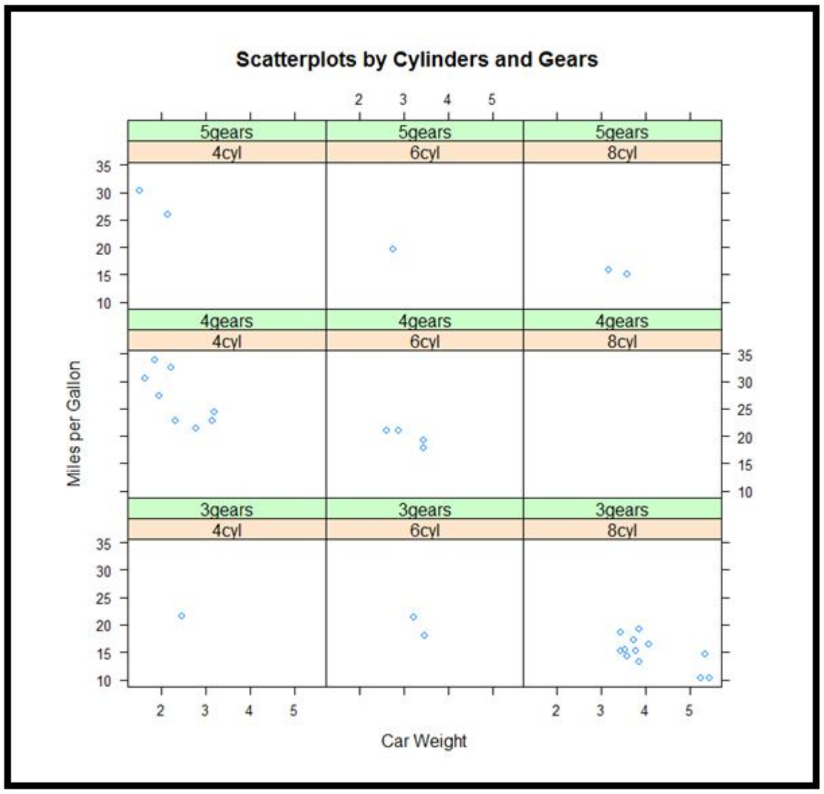


```
> mpg
 [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4 10.4
[17] 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7 15.0 21.4
> cyl.f
 [1] 6cyl 6cyl 4cyl 6cyl 8cyl 6cyl 8cyl 4cyl 4cyl 6cyl 6cyl 8cyl 8cyl 8cyl 8cyl 8cyl
[17] 8cyl 4cyl 4cyl 4cyl 4cyl 8cyl 8cyl 8cyl 8cyl 4cyl 4cyl 4cyl 8cyl 6cyl 8cyl 4cyl
Levels: 4cyl 6cyl 8cyl
> gear.f
 [1] 4gears 4gears 4gears 3gears 3gears 3gears 3gears 4gears 4gears 4gears 4gears
[12] 3gears 3gears 3gears 3gears 3gears 3gears 4gears 4gears 4gears 3gears 3gears
[23] 3gears 3gears 3gears 4gears 5gears 5gears 5gears 5gears 5gears 4gears
Levels: 3gears 4gears 5gears
> # boxplots for each combination of two factors
> bwplot(cyl.f~mpg|gear.f,
+     ylab="Cylinders", xlab="Miles per Gallon",
+     main="Mileage by Cylinders and Gears",
+     layout=(c(1,3))
+ )
```

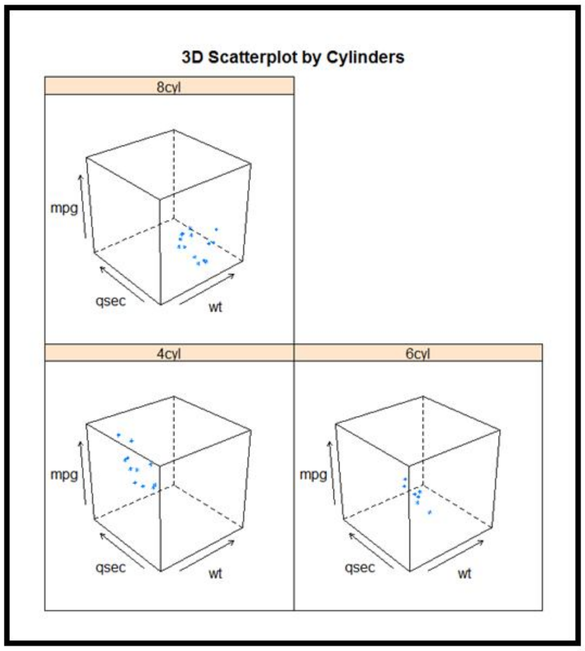




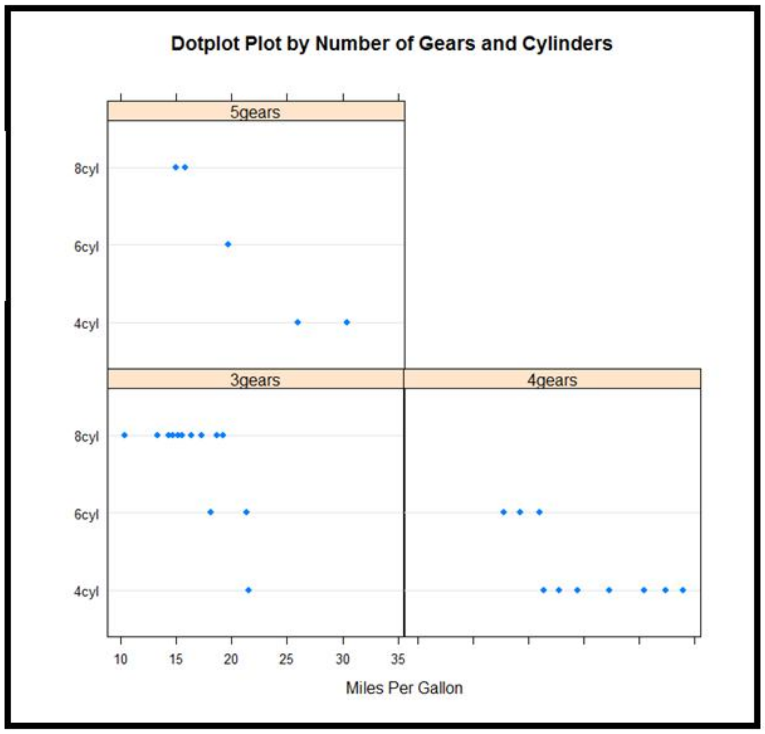
```
> # scatterplots for each combination of two factors
> xyplot(mpg~wt|cyl.f*gear.f,
+     main="Scatterplots by Cylinders and Gears",
+     ylab="Miles per Gallon", xlab="Car Weight")
> |
```



```
> # 3d scatterplot by factor level
> cloud(mpg~wt*qsec|cyl.f,
+     main="3D Scatterplot by Cylinders")
```



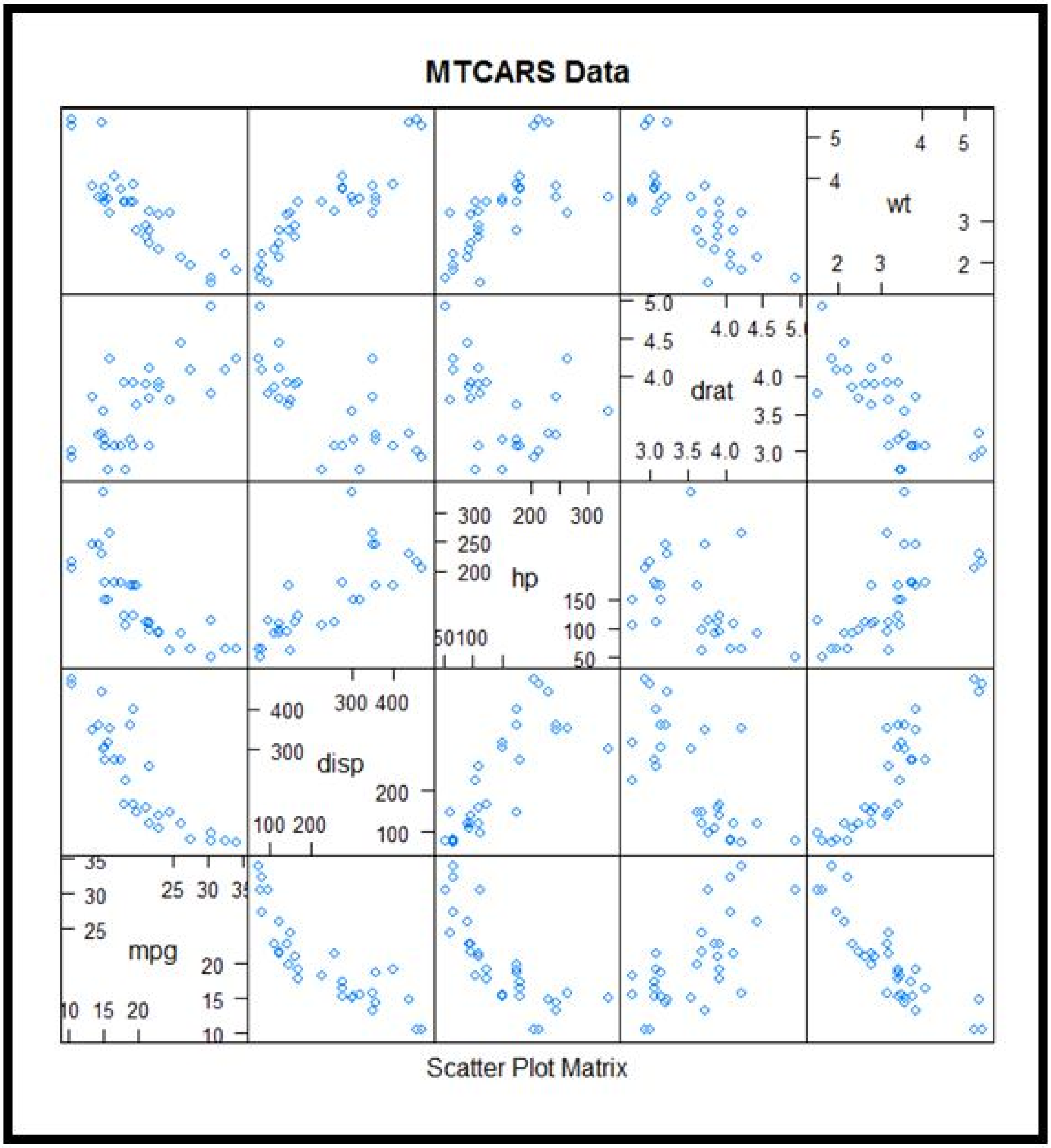
```
> # dotplot for each combination of two factors
> dotplot(cyl.f~mpg|gear.f,
+    main="Dotplot Plot by Number of Gears and Cylinders",
+    xlab="Miles Per Gallon")
```



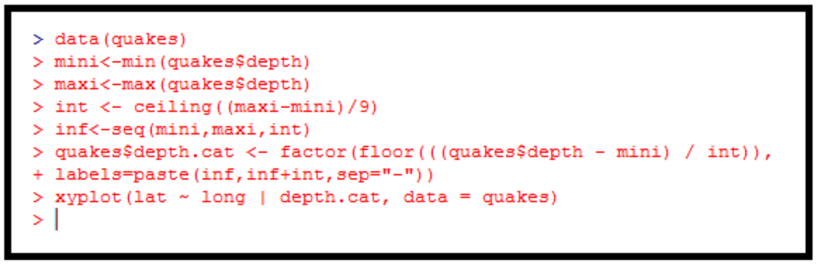
```
> # scatterplot matrix
> splom(mtcars[c(1,3,4,5,6)],
+    main="MTCARS Data")
```

```
                     mpg  disp  hp drat    wt
Mazda RX4           21.0 160.0 110 3.90 2.620
Mazda RX4 Wag       21.0 160.0 110 3.90 2.875
Datsun 710          22.8 108.0  93 3.85 2.320
Hornet 4 Drive      21.4 258.0 110 3.08 3.215
Hornet Sportabout   18.7 360.0 175 3.15 3.440
Valiant             18.1 225.0 105 2.76 3.460
Duster 360          14.3 360.0 245 3.21 3.570
Merc 240D           24.4 146.7  62 3.69 3.190
Merc 230            22.8 140.8  95 3.92 3.150
Merc 280            19.2 167.6 123 3.92 3.440
Merc 280C           17.8 167.6 123 3.92 3.440
Merc 450SE          16.4 275.8 180 3.07 4.070
Merc 450SL          17.3 275.8 180 3.07 3.730
Merc 450SLC         15.2 275.8 180 3.07 3.780
Cadillac Fleetwood  10.4 472.0 205 2.93 5.250
Lincoln Continental 10.4 460.0 215 3.00 5.424
Chrysler Imperial   14.7 440.0 230 3.23 5.345
Fiat 128            32.4  78.7  66 4.08 2.200
Honda Civic         30.4  75.7  52 4.93 1.615
Toyota Corolla      33.9  71.1  65 4.22 1.835
Toyota Corona       21.5 120.1  97 3.70 2.465
Dodge Challenger    15.5 318.0 150 2.76 3.520
AMC Javelin         15.2 304.0 150 3.15 3.435
Camaro Z28          13.3 350.0 245 3.73 3.840
Pontiac Firebird    19.2 400.0 175 3.08 3.845
Fiat X1-9           27.3  79.0  66 4.08 1.935
Porsche 914-2       26.0 120.3  91 4.43 2.140
Lotus Europa        30.4  95.1 113 3.77 1.513
Ford Pantera L      15.8 351.0 264 4.22 3.170
Ferrari Dino        19.7 145.0 175 3.62 2.770
Maserati Bora       15.0 301.0 335 3.54 3.570
Volvo 142E          21.4 121.0 109 4.11 2.780
```

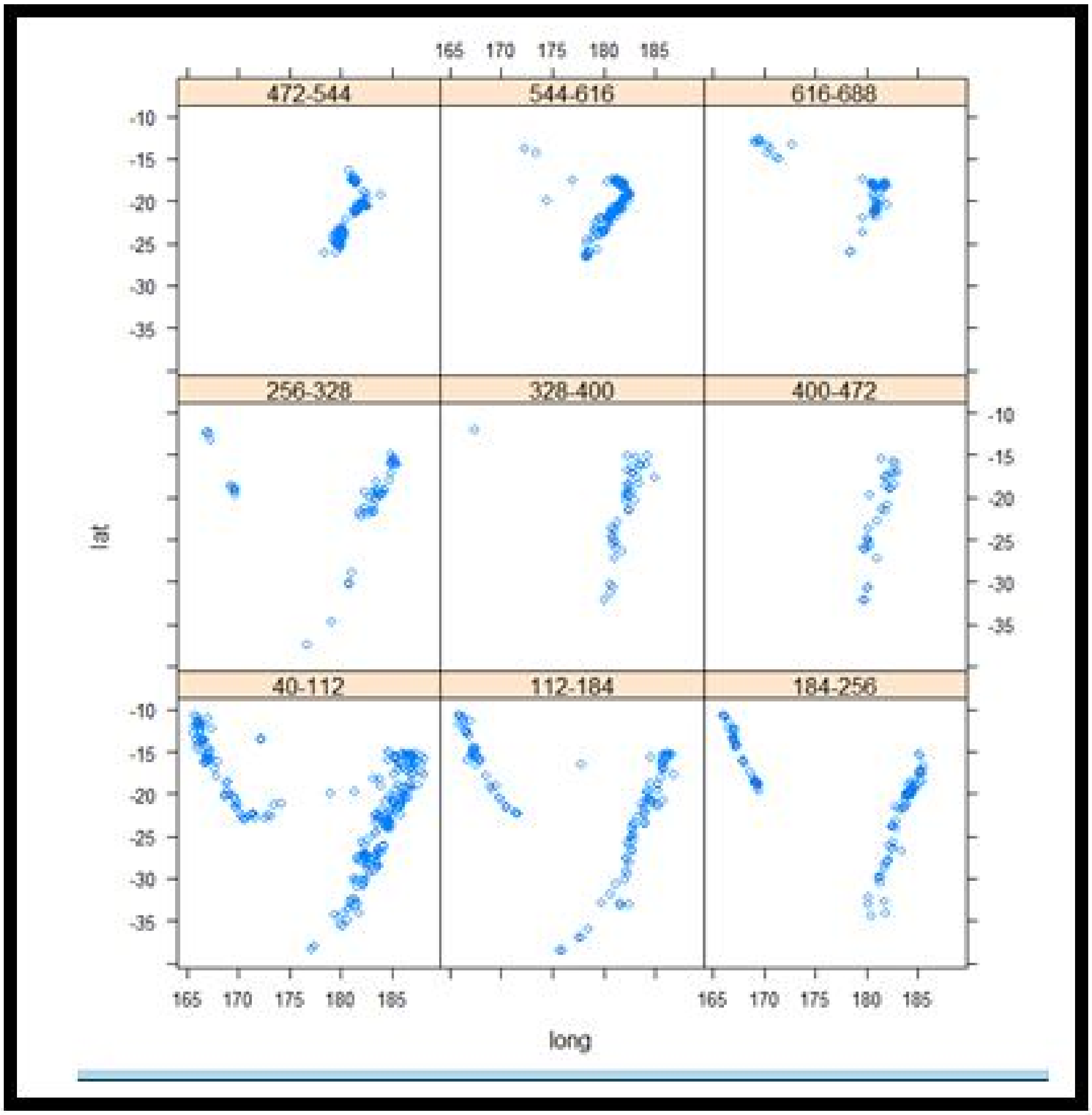
```
> # scatterplot matrix
> splom(mtcars[c(1,3,4,5,6)],
+    main="MTCARS Data")
```


MTCARS Data
wt
drat
hp
disp
mpg
Scatter Plot Matrix

- We shall look at the examples from the help pages of lattice, and use some of the data sets available in R: the locations of 1000 seismic events near the Fiji islands, and some lower measurements made on three species of iris.



```
> data(quakes)
> mini<-min(quakes$depth)
> maxi<-max(quakes$depth)
> int <- ceiling((maxi-mini)/9)
> inf<-seq(mini,maxi,int)
> quakes$depth.cat <- factor(floor(((quakes$depth - mini) / int)),
+ labels=paste(inf,inf+int,sep="-"))
> xyplot(lat ~ long | depth.cat, data = quakes)
> |
```



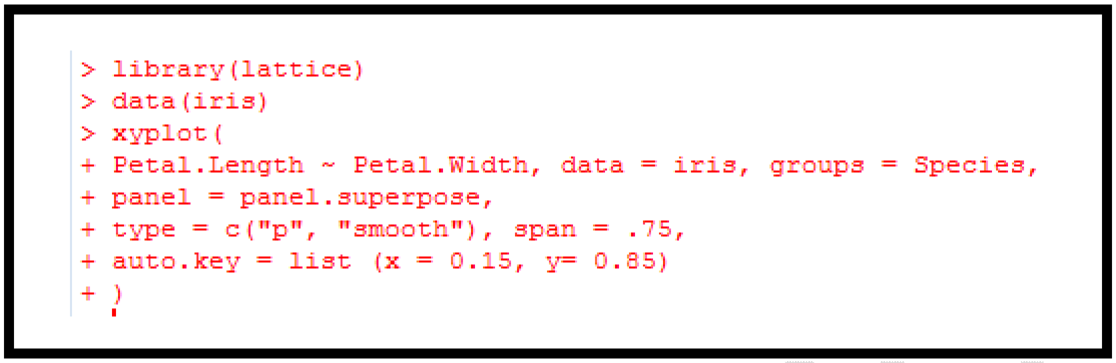
- We shall look at the examples from the help pages of lattice, and use some of the data sets available in R: the locations of 1000 seismic events near the Fiji islands, and some lower measurements made on three species of iris.
- The first command loads the data quakes in memory.

  ```
  data(quakes)
  ```
- The next five commands create a factor by dividing the depth (variable depth) in nine equally ranges intervals: the levels of this factor are labeled with the lower and upper bounds of these intervals.

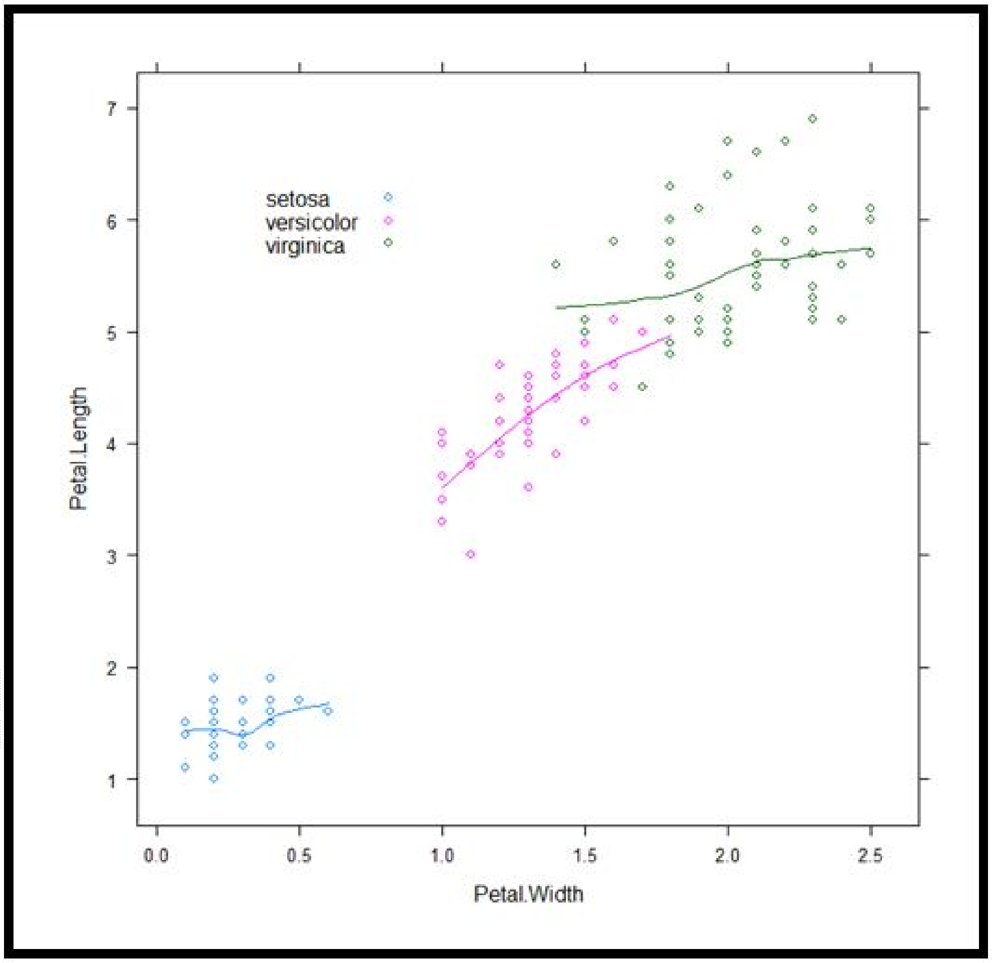
  ```
  mini<-min(quakes$depth)
  maxi<-max(quakes$depth)
  int <- ceiling((maxi-mini)/9)
  inf<-seq(mini,maxi,int)
  quakes$depth.cat <- factor(floor(((quakes$depth - mini) / int)), labels=paste(inf, inf + int, sep="-
  "))
  ```
- It then calls the function xyplot with the appropriate formula and an argument data indicating where xyplot must look for the variables.

  ```
  xyplot(lat ~ long | depth.cat, data = quakes)
  ```
- With the data iris , the overlap among the different species is sufficiently small so they can be plotted on the figure.

- The commands are



```
> library(lattice)
> data(iris)
> xyplot(
+ Petal.Length ~ Petal.Width, data = iris, groups = Species,
+ panel = panel.superpose,
+ type = c("p", "smooth"), span = .75,
+ auto.key = list (x = 0.15, y= 0.85)
+ )
```



- The option groups, as suggested by its name, defines groups that will be used by the other options.

Petal.Length ~ Petal.Width, data = iris, groups = Species,
panel = panel.superpose,

- The option panelwhich defines how the different groups will be represented on the graph: we use here a pre-defined function panel.superpose in order to superpose the groups n the same plot.
- No option is passed to panel.superpose, the default colors will be used to distinguish the groups.

  type = c("p", "smooth"), span = .75,

  The option type, like in plot(), specifies how the data are represented: "p" to draw points, and

  "smooth" to draw a smooth curve which degree of smoothness is specified by span.
  auto.key = list (x = 0.15, y= 0.85)

- The option auto.key adds a legend to the graph: it is only necessary to give, as a list, the coordinates where the legend is to be plotted. Note that here coordinates are relative to the size of the plot (i.e. in [0,1]).