Coursera: Exploratory Data Analysis

Contents

[Principles of Building Analytics Graphics 2](#_Toc419559060)

[Exploratory Graphs 3](#_Toc419559061)

[Characteristics of exploratory graphs 3](#_Toc419559062)

[Example: Air Pollution in the United States 3](#_Toc419559063)

[Simple Summaries of Data 4](#_Toc419559064)

[Summary 5](#_Toc419559065)

[Further Resources 5](#_Toc419559066)

[Plotting Systems in R 5](#_Toc419559067)

[Base Plotting System 5](#_Toc419559068)

[The Lattice System 5](#_Toc419559069)

[The ggplot2 System 5](#_Toc419559070)

[Summary 6](#_Toc419559071)

[Base Plotting System in R 6](#_Toc419559072)

[The Process of Making a Plot 6](#_Toc419559073)

[Base Graphics 7](#_Toc419559074)

[Some Important Base Graphics Parameters 7](#_Toc419559075)

[Base Plotting Functions 7](#_Toc419559076)

[Summary 8](#_Toc419559077)

[Graphics Devices in R 9](#_Toc419559078)

[What is a Graphics Device? 9](#_Toc419559079)

[How Does a Plot Get Created? 9](#_Toc419559080)

[Graphics File Devices 9](#_Toc419559081)

[Multiple Open Graphics Devices 10](#_Toc419559082)

[Copying Plots 10](#_Toc419559083)

[Summary 11](#_Toc419559084)

[Lattice Plotting System 11](#_Toc419559085)

[Lattice Function 11](#_Toc419559086)

[Lattice Behaviour 12](#_Toc419559087)

[Lattice Panel Functions 12](#_Toc419559088)

[Summary 12](#_Toc419559089)

[The ggplot2 Plotting System 13](#_Toc419559090)

[What is ggplot2? 13](#_Toc419559091)

[Grammar of Graphics 13](#_Toc419559092)

[The Basics: qplot() 13](#_Toc419559093)

[Summary of qplot() 13](#_Toc419559094)

[Basic Components of ggplot2 Plot 14](#_Toc419559095)

[Building Plots with ggplot2 14](#_Toc419559096)

[Annotation 14](#_Toc419559097)

[About Axis Limits 15](#_Toc419559098)

[More Complex Examples 15](#_Toc419559099)

[Hierarchical Clustering 15](#_Toc419559100)

[Can we find things that are close together? 15](#_Toc419559101)

[Hierarchical clustering 15](#_Toc419559102)

[How do we define close? 16](#_Toc419559103)

[Hierarchical clustering -dist 16](#_Toc419559104)

[Merging points - complete 16](#_Toc419559105)

[heatmap() 16](#_Toc419559106)

[Notes and further resources 16](#_Toc419559107)

[K-means Clustering 17](#_Toc419559108)

[kmeans() 17](#_Toc419559109)

[Notes and further resources 17](#_Toc419559110)

[Principle Components Analysis and Singular Value Decomposition 18](#_Toc419559111)

[Related problems 18](#_Toc419559112)

[Related solutions - PCA/SVD 18](#_Toc419559113)

# Principles of Building Analytics Graphics

* Trying to tell a story of the data
* Taken from Edward Tuff (Beautiful Evidence, 2006)
* Principle 1: Show comparisons
  + Evidence for a hypothesis is always relative to another competing hypothesis
  + Always ask “Compared to what”
  + E.g. box plot of change in symptom-free days Air Cleaner vs Control (relative to doing nothing air cleaner is showing improvement)
* Principle 2: Show causality, mechanism, explanation, systematic structure
  + What is your causal framework for thinking about a question?
  + E.g. second box-plot with symptom free days next to change in Fine Particulate Matter (less particulate measure explains decrease in symptom free days)
* Principle 3: Show multivariate data
  + Multivariate = more than 2 variables
  + The real world is multivariate
  + Need to “escape flatland”
  + E.g. show PM10 concentration vs mortality split across different systems
* Principle 4: Integrate evidence
  + Completely integrate words, numbers, images, diagrams
  + Data graphics should make use of many modes of data presentation
  + Don’t let the tool drive the analysis
* Principle 5: Describe and document the evidence with appropriate labels, scales, sources, etc.
  + A data graphics should tell a complete story that is credible
* Principle 6: Content is king
  + Analytical presentations ultimately stand or fall depending on the quality, relevance and integrity of the content

# Exploratory Graphs

* To understand data properties
* To find patterns in data
* To suggest modeling strategies
* To “debug” analyses
* To communicate results (not focus of this lecture)

## Characteristics of exploratory graphs

* They are made quickly
* A large number are made
* The goal is for personal understanding
* Axes/legends are generally cleaned up (later)
* Color/size are primarily used for information

## Example: Air Pollution in the United States

* US EPA sets national ambient air quality standards for outdoor air pollution
* For fine particle pollution (PM2.5) the “annual mean, averaged over 3 years”, cannot exceed 12 microgram/m3
* Data on daily PM2.5 are available from the U.S. EPA web site
* Question: are there any counties in the U.S. that exceed that
* Whenever doing exploratory analysis you need to have a at least some kind of background questions

## Simple Summaries of Data

* One dimensions
  + Five-number summary (minimum, 1st quantile, median, 3rd quantile, maximum)
  + Box-plot
  + Histogram
  + Bar plot
* Two dimensions
  + Mutliple/overlayed 1-D plots (Lattice/ggplot2)
  + Scatterplots
  + Smooth scatterplots
* >2 dimensions
  + Overlayed/multiple 2-D plots; coplots
  + Use color, size, shape to add dimensions
  + Spinning plots
  + Actual 3-D plots (not that useful)

|  |  |
| --- | --- |
| Command | Description |
| summary(pollution$pm25) | displays five-number summary + mean for pm25 numeric variable |
| boxplot(pollution$pm25,col="blue") | displays box-plot with blue colour |
| hist(pollution$pm25,col="green")  rug(pollution$pm25) | displays histogram, rug displays actual points underneath |
| hist(pollution$pm25,col="green", breaks = 100) | display histograms with 100 segments |
| boxplot(pollution$pm25,col="blue")  abline(h=12) | displays boxplot and overlays horizontal line at 12 |
| hist(pollution$pm25,col="green")  abline(v=12,lwd=2)  abline(v=median(pollution$pm25),  col="magenta",lwd=4) | displays histogram with overlays of horizontal line at 12 and second at median |
| barplot(table(pollution$region), col="wheat",main="number of counties in each region") | displays bar plot grouped by region |
| boxplot(pm25 ~ region, data = pollution, col="red") | mutltiple boxplots for pm25 by region |
| par(mfrow=c(2,1),mar=c(4,4,2,1))  hist(subset(pollution,region=="east")$pm25  ,col="green")  hist(subset(pollution,region=="west")$pm25 ,col="green") | displays multiple histograms, par sets graphical parameters |
| with(pollution, plot(latitude,pm25))  abline(h=12,lwd=2,lty=2) | displays scatter-plot with latitude on x-axis and pm25 on y-axis |
| with(pollution, plot(latitude,pm25,col=region))  abline(h=12,lwd=2,lty=2) | adds color to make scatter-plot three variable including region |
| par(mfrow = c(1, 2), mar = c(5, 4, 2, 1))  with(subset(pollution, region == "west"), plot(latitude, pm25, main = "West"))  with(subset(pollution, region == "east"), plot(latitude, pm25, main = "East")) | displays two scatter-plots one for each region |
| par(“bg”) | find the default value for global graphics parameter background |

## Summary

* Exploratory plots are “quick and dirty”
* Let you summarize the data (usually graphically) and highly broad features
* Explore basic questions and hypotheses (and perhaps rule them out)
* Suggest modelling strategies for “next step”

## Further Resources

* <http://gallery.r-enthusiasts.com/>
* <http://www.r-bloggers.com/>

# Plotting Systems in R

* R has developed three different plotting systems over the years

## Base Plotting System

* “Artist’s palette” model
* Start with blank canvas and build up from there
* Start with plot function (or similar)
* Use annotation function to add/modify (test, lines, points, axis)
* Plot is just a series of R commands
* Problem if it’s difficult to translate R plot to another person (as it’s just a series of R plots)
* Requires a lot of code if you don’t like defaults

## The Lattice System

* Plots are created with a single function call (xyplot, bwplot, etc.)
* Most useful for conditioning types of plots: Looking at how y changes with x across levels of z
* Things like margins/spacing set automatically because entire plot is specified at once
* Good for putting many many plots on a screen
* Downside:
  + it can be awkward to define entire plot in single function call
  + difficult to annotate

## The ggplot2 System

* Splits the difference between base and lattice in a number of ways
* Automatically deals with spacings, text, titles but also allows you to annotate by “adding” to a plot
* Superficial similarity to lattice but generally easier/more intuitive to use
* Default mode makes many choices for you (but you can still customize to your heart’s desire)

## Summary

* Base: “artists’s pallete” model
* Lattice: Entire plot is specified by one function; conditioning
* ggplot2: Mixes elements of Base and Lattice
* Three models cannot be interchanged

# Base Plotting System in R

* Core plotting and graphics engine in R is encapsulated in the following packages:
  + graphics: contains plotting functions for the “base” graphing systems, including plot, hist, boxplot and many others
  + grDevices: contains all the code implementing the various graphics devices including X11, PDF, PostScript, PNG, etc.
* The lattice plotting system is implemented using the following packages:
  + lattice: contains code for producing Trellis graphics, which are independent of the “base” graphics system; includes functions like xyplot, bwplot, levelplot
  + grid: implements a different graphing system independent of the “base” system; the lattice package builds on top of grid; we seldom call functions from the grid package directly

## The Process of Making a Plot

* When making a plot one must first make a few considerations:
  + Where will the plot be made? On the screen? in a file?
  + How will the plot be used?
    - Is the plot for viewing temporarily on the screen?
    - Will it be presented in a web browser?
    - Will it eventually end up in a paper that might be printed?
    - Are you using it in a presentation?
  + Is there a large amount of data going into the plot? Or is it just a few points?
  + Do you need to be able to dynamically resize the graphic?
  + What graphics system will you use: base, lattice, or ggplot2? These generally cannot be mixed.
  + Base graphics are usually constructed piecemeal, with each aspect of the plot handled separately through a series of function calls; this is often conceptually simpler and allows plotting to mirror the thought process
  + Lattice graphics are usually created in a single function call, so all of the graphics parameters have to specified at once; specifying everything at once allows R to automatically calculate the necessary spacings and font sizes.
  + ggplot2 combines concepts from both base and lattice graphics but uses an independent implementation

## Base Graphics

* Base graphics are used most commonly and are a very powerful system for creating 2-D graphics
* There are two phases to creating a base plot
  + Initializing a new plot
  + Annoting (addint to) an existing plot
* Calling plot(x,y) or hist(x) will launch a graphics device (if one is not already open) and draw a new plot on the device
* If the arguments to plot are not of the same special class, then the default method for plot is called: this function has many arguments, letting you set the title, x-axis label, y-axis label
* The base graphics system has many parameters that can set and tweaked; these parameters are documented in ?par; it wouldn’t hurt to try to memorize this help page!

## Some Important Base Graphics Parameters

Many base plotting functions share a set of parameters. Here are a few key ones:

* pch: the plotting symbol (default is open circle)
* lty: the line type (default is solid line), can be dashed, dotted, etc
* lwd: the line width, specified as an integer multiple
* col: the plotting color, specified as a number, string or hex code; the colors() function gives you a vector of colors by name
* xlab: character string for the x-axis label
* ylab: character string for the y-axis label

The par() function is used to specify global graphics parameters that affect all plots in an R session. These parameters can be overridden when specified as argumetns to specific plotting functions.

* Las: the orientation of the axis labels on the plot
* bg: the background color
* mar: the margin size
* oma: the outer margizen size (default is 0 for all sides)
* mfrow: number of plots per row, column (plots are filled row-wise)
* mfcol: number of plots per row, column (plots are filled column-wise)

## Base Plotting Functions

* plot: makes a scatterplot, or other type of plot depending on the class of the object being plotted
* lines: add lines to a plot, given a vector x values and a corresponding vector of y values (or a 2-column matrix); this function just connects the dots
* points: add points to a plot
* text: add text

Base

|  |  |
| --- | --- |
| Command | Description |
| with(airquality, plot(Wind,Ozone,main="Ozone and Wind in New York City",type="n")) | sets up plot without the data |
| with(subset(airquality,Month==5), points(Wind,Ozone, col="blue")) | Add plot for May as blue |
| with(subset(airquality,Month!=5), points(Wind,Ozone, col="red")) | Add plot for not May as red |
| legend("topright",pch=1, col=c("blue","red"),legend=c("May","Other Months")) | Adds legend to top right |
| plot.new() | Creates a blank plot |
| dev.off() | Clears all plots and parameters |
| with(airquality, plot(Wind,Ozone, main = "Ozone and Wind in New York City",pch=20))  model <- lm(Ozone ~ Wind, airquality)  abline(model,lwd=2) | Creates scatter plot then adds regression line as overlay |
| par(mfrow = c(1, 3), mar = c(4, 4, 2, 1), oma = c(0, 0, 2, 0))  with(airquality, {  plot(Wind, Ozone, main = "Ozone and Wind")  plot(Solar.R, Ozone, main = "Ozone and Solar Radiation")  plot(Temp, Ozone, main = "Ozone and Temperature")  mtext("Ozone and Weather in New York City", outer = TRUE)  }) | create three scatter plots with higher level title |
| example(points) | see example of different points |
| plot(x,y,type="n")  points(x[g == "Male"], y[g == "Male"],col="green")  points(x[g == "Female"], y[g == "Female"],col="pink") | use subsetting and points() to add datasets with different colours |
|  |  |
| image(1:10, 1:40, t(dataMatrix)[,nrow(dataMatrix):1]) | Create an image with 10 elements on the x-axis, 40 elements on the y-axis, and the values creating the colours provided by the 400 element matrix (a heatmap) |
| heatmap(dataMatrix) | Use heatmap which will display dendrites |
|  |  |

## Summary

* Plots in the base plotting system are created by calling successive R functions to “build up” a plot
* Plotting occurs in two stages:
  + Creation of a plot
  + Annotation of a plot (adding lines, points, text, legends)
* The base plotting system is very flexible and offers a high degree of control over plotting

# Graphics Devices in R

## What is a Graphics Device?

* A graphics device is something where you can make a plot appear
  + A window on your computer (screen device)
  + A PDF file (file device)
  + A PNG or JPEG file (file device)
  + A scalable vector graphics (SVG) file (file device)
* When you make a plot in R, it has to be "sent" to a specific graphics device
* The most common place for a plot to be "sent" is the screen device
  + On a Mac the screen device is launched with the quartz()
  + On Windows the screen device is launched with windows()
  + On Unix/Linux the screen device is launched with x11()
* When making a plot you need to consider how the plot will be used to determine what device the plot should be sent to
  + The list of devices is found in ?Devices; there are also devices created by users on CRAN
* For quick visualizations and exploratory analysis, usually you want to use the screen device
  + Functions like plot in base, xyplot in lattice, or qplot in ggplot2 will default to sending a plot to the screen device
  + On a given platform (Mac, Windows, Unix/Linux) there is only one screen device
* For plots that may be printed out or be incorporated into a document (e.g. papers/reports, slide presentations) usually a file device is more appropriate
  + There are many different file devicves to choose from

## How Does a Plot Get Created?

There are two basic approapches to plotting. The first is most common:

1. Call a plotting function like plot, xyplot, qplot
2. The plot appears on the screen

The second approach to plotting is most commonly used for file devices:

1. Explicitly launch a graphics device
2. Call a plotting function to make a plot (Note: if you are using a file device, no plot will appear on the screen)
3. Annotate plot if necessary
4. Explicitly close graphics device with dev.off() (this is very important!)

## Graphics File Devices

There are two basic types of file devices: vector and bitmap devices

Vector formats:

* pdf: useful for line-type graphics, resizes well, usually portable, not efficient if a plot has many objects/points
* svg: XML-based scalable vector graphics; supports animation and interactivity, potentially useful for web-based plots
* win.metafile: Windows metafile format (only on Windows)
* postscript: older format, also resizes well, usually portable, can be used to create encapsulated postscript files, Windows systems usually do not have

Bitmap formats:

* png: bitmapped format, good for line drawings or images with solid colors, uses lossless compression (like the old GIF format) , most web browsers can read this format natively, good for plotting man many many points, does not resize well
* jpeg: good for photographs or natural scense, uses lossy compression, good for plotting many many many ponts, does not resize well, can be read by almost any computer and any web browser, not great for line drawings
* tiff: creates bitmap files in the TIFF format supports lossless compression
* bmp: native Windows bitmapped format

## Multiple Open Graphics Devices

* It is possible to open multiple graphics devices (screen, file or both) for example when viewing multiple plots at once
* Plotting can only occur on one graphics device at a time
* The currently active graphics device can be found by calling dev.cur()
* every open graphics device is assigned an integer >= 2
* You can change the active graphics device with dev.set(<integer>) where <integer> is the number associated with the graphics device you want to switch to

## Copying Plots

Copying a plot to another device can be useful because some plots require a lot of code and it can be a pain to type all that in again for a different device

* dev.copy: copy a plot from one device to another
* dev.copy2pdf: specifically copy a plot to a PDF file

Note: copying a plot is not an exact operation, so the result may be identical to the original

|  |  |
| --- | --- |
| Command | Description |
| pdf(file = "myplot.pdf") | Open PDF device, create myplot.pdf in working directory |
| with(faithful, plot(eruptions, waiting))  title(main="Old Faithful Geyser Data") | creates plot and send to file, annotate plot |
| dev.off() | Closes PDF file device, can now open to see |
| with(faithful, plot(eruptions, waiting))  title(main="Old Faithful Geyser Data")  dev.copy(png,file="geyser.png")  dev.off() | First creates plot in screen device, then copies to PNG device |
|  |  |
|  |  |

## Summary

* Plots must be created on a graphics device
* The default graphics device is almost always the screen device, which is most useful for exploratory analysis
* File devices are useful for creating plots that can be included in other documents or sent to other people
* For file devices, there are vector and bitmap formats
  + Vector formats are good for line drawings and plots with solid colors using a modest number of points
  + Bitmap formats are good for plots with a large number of points, natural scenes or web-based plots

# Lattice Plotting System

Implemented using the following packages:

* lattice: contains code for producing Trellis graphics, which are independent of the “base” graphics system, includes functions like xyplot, bwplot, levelplot
* grid: implements a different graphic system independent of the “base” system; the lattice package build on top of grid
  + We seldom call grid functions directly
* The lattice plotting system does not have a “two-phase” aspect with separate plotting and annotation like in base plotting
* All plotting/annotation is done at once with a single function call

## Lattice Function

* xyplot: main function for creating scatterplots
* bwplot: box-and-whickers plot(“boxplot”)
* histogram: histograms
* stripplot: like a boxplot but with actual points
* dotplot: plot dots on “violin strings”
* splom: scatterplot matrix; like pairs in base plotting system
* levelplot, contourplot: for plotting “image” data

Generally take a formula for their first argument, usually of the form:  
xyplot(y ~ x | f \* g, data)

* We use the formula notation here, hense the ~
* on the left of the ~ is the y-axis variable, on the right is the x-axis variable
* f and g are conditioning variables - they are optional
  + the \* indicates an interaction between two variables
* The second argument is the data frame or list from which the variables in the formula should be looked up
  + If not data frame or list is passed, then the parent frame is used
* If no other arguments are passed, there are defaults that can be used

## Lattice Behaviour

Lattice functions behave differently from base graphics functions in one critical way

* Base graphics functions plot data directly to the graphics device (screen, PDF, etc)
* Lattice graphics functions return an object of class **trellis**
* The print methods for lattice functions actually do the work of plotting the data on the graphics device.
* Lattice functions return “plot objects” that can, in principle, be stored (but it’s usually better to just save the code + data)
* On the command line, trellis objects are auto-printed so that it appears the function is plotting the data

## Lattice Panel Functions

* Lattice functions have a panel function which controls what happens inside each panel of the plot
* The lattice package comes with default panel functions, but you can supply your own if you want to customize what happens in each panel
* Panel functions receive the x/y coordinates of the data points in their panel (along with any optional arguments)

## Summary

* Lattice plots are constructed with a single function call to a core lattice function (e.g. xyplot)
* Apsects like margins and spacing are automatically handled and defaults are usually sufficient
* This lattice system is ideal for creating conditioning plots where you examine the same kind of plot under many different conditions
* Panel functions can be specified/customized to modify what is plotted in each of the plot panels

|  |  |
| --- | --- |
| Command | Description |
| xyplot(Ozone ~ Wind, data = airquality) | create scatter plot on Ozone vs Wind |
| airquality <- transform (airquality, Month = factor(Month))  xyplot(Ozone ~ Wind | Month, data = airquality, layout = c(5,1)) | create multi-panel scatter plot with each Month having one panel |
| p <- xyplot(Ozone ~ Wind, data = airquality)  print(p) | stores graphic (Trellis object) first to p, then prints |
| xyplot(y ~ x | f, panel = function (x, y, ...) {  panel.xyplot(x,y,...)  panel.lmline(x,y,col=2)  }) | Use custom panel function to display an xyplot and linear fit regression line |
|  |  |
|  |  |

# The ggplot2 Plotting System

## What is ggplot2?

* An implementation of the Grammar of Graphics by Leland Wilkinson
* Written by Hadley Wickham (while he was a graduate student at Iowa State)
* A “third” graphics system for R (along with base and lattice)
* Available from CRAN via install.packages()
* Web site: <http://ggplot2.org> (better documentation)
* Grammar of graphics represents and abstractiosn of graphics ideas/objects
* Think “verb”, “noun”, “adjective” for graphics
* Allows for a “theory of

## Grammar of Graphics

“in brief, the grammar tells us that a statistical graphic is a mapping from data to aesthetic attributes (colour, shape, size) of geometric objects (points, lines, bars). The plot may also contain statistical transformations of the data and is drawn in a specific coordinate system” from ggplot2 book

## The Basics: qplot()

* Works must like the plot function in base graphics system
* Looks for data in a data frame, similar to lattice, or in the parent environment
* Plots are made up of aesthetics (size, shape, color) and geoms (points, lines)
* Factors are important for indicating subsets of the data (if they are to have different properties); they should be **labeled**
* The qplot() hides what goes on underneath, which is okay for most operations
* ggplot() is the core function and very flexible for doing things qplot() cannot do

## Summary of qplot()

* The qplot() function is the analog to plot() but with many built-in features
* Syntax somewhere in between base/lattice
* Produces very nice graphics, essentially publication ready (if you like the design)
* Difficult to go against the grain/customize (don’t bother, use full ggplot2 power in that case)

|  |  |
| --- | --- |
| Command | Description |
| qplot(displ, hwy, data = mpg) | display simple qplot with displ as x-axis and hwy as y-axis from mpg data frame |
| qplot(displ,hwy,data = mpg, color = drv) | create groups by drv differentiated by color |
| qplot(displ,hwy,data = mpg, shape = drv) | create groups by drv differentiated by shape |
| qplot(displ, hwy, data = mpg, geom = c(“point”, “smooth”)) | Add statistic summary of data (25th percentile) |
| qplot(hwy,data=mpg,fill=drv) | create histogram on hwy variable, stacked by dr |
| qplot(displ,hwy, data = mpg, facets = . ~ drv) | creates panelled scatterplot by factor drv (column-wise as drv on right side of tilde) |
| qplot(hwy, data = mpg, facets = drv ~ ., binwidth=2) | creates panelled histogram by factor drv (row-wise as drv on left side of tilde); binwidth = 2 doubles size of each histogram |
| qplot(log(hwy), data = mpg, geom = "density", color = drv) | creates density diagram, with log of hwy as variable |
| qplot(displ, hwy, data = mpg, color = drv, geom = c("point", "smooth"), method = "lm") | plots displ by hwy grouped by drv using color, adds geom with method linear model |
| qplot(displ, hwy, data = mpg, size = cyl) | creates scatter with cyl differentiated by size |
| qplot(stepsperday,binwidth=range(stepsperday)[2]/11) + scale\_y\_discrete(breaks = seq(0,10,2)) + geom\_vline( | create a vertical |

## Basic Components of ggplot2 Plot

* A data frame
* aesthetic mappings: how data are mapped to color, size
* geoms: geometric objects like points, lines, shapes
* facets: for conditional plots
* stats: statistical transformations like binning, quantiles, smoothing
* scales: what scale an aesthetic map uses (example: male = red, female = blue)
* coordinate system

## Building Plots with ggplot2

* When building plots in ggplot2 (rather than using qplot) the “artists’s palette” model may be the closest analogy
* Plots are build up in layers
  + Plot the data
  + Overlay a summary
  + Metadata and annotation

## Annotation

* Labels: xlab(), ylab(), labs(), ggtitle()
* Each of the “geom” functions has options to modify
* For things that only make sense globally, use theme()
  + Example: theme(legend.position = "none")
* Two standard appearance themes are included
  + theme\_gray(): the default theme (gray background)
  + theme\_bw(): More stark/plain

|  |  |
| --- | --- |
| Command | Description |
| g <- ggplot(mpg, aes(displ, hwy)) | create ggplot object |
| p <- g + geom\_point() | add points to p (now can print as x/y scatterplot) |
| p <- g + geom\_point() + geom\_smooth() | add points and smoother (loess by default) |
| p <- g + geom\_point() + geom\_smooth(method="lm") | add points and smoother (linear model) |
| p <- g + geom\_point() + facet\_grid(. ~ drv) + geom\_smooth(method="lm") | add points, smoother and group by drv (with horizontal panels) |
| p <- g + geom\_point(color="steelblue" ,size=4,alpha=1/2) | plots with different colour, size |
| p <- g + geom\_point(aes(color=class), size=4,alpha=1/2) | plot assigning color to a data variable (class) instead of constant (steelblue) |
| p <- g + geom\_point(aes(color=class),size=4,alpha=1/2) + labs(title = "Miles per gallon vs displacement") + labs(x = "Engine displacement (litres) ", y = "highway miles per gallon") | plot with custom labels |
| p <- g + geom\_point(aes(color=class),size=4,alpha=1/2) + geom\_smooth(size = 4, linetype = 3, method = "lm", se = FALSE) | plot with modified smoother (se removes confidence intervals) |
| p <- g + geom\_point(aes(color=class),size=4,alpha=1/2) + theme\_bw(base\_family = "Times") | Uses different theme |
| ggsave(file="plot6.png") | save plot on screen to png file |
| facet\_wrap( ~ variable, ncol = 1) | In graphic group by variable and arrange vertically, with variable title on top |
| facet\_grid(variable ~ .) | In graphic group by variable and arrange vertically, with variable title on right |
|  |  |
|  |  |

## About Axis Limits

If you change limit to exclude outlier, then it takes out outlier completely. The solution is to use coord\_cartesian(ylim=c(-3,3)) for example

## More Complex Examples

* How does the relationship between PM2.5 and nocturnal symptoms vary by BMI and NO2?
* Unlike our previous BMI variable, NO2 is continuous
* We need to make N02 categorical so we can condition it in the plotting
  + Use the cut() function for this

# Hierarchical Clustering

Common technique for visualizing high multi-dimensional data.

## Can we find things that are close together?

Clustering organizes things that are **close** into groups

* How do we define close?
* How do we group things?
* How do we visualize the grouping?
* How do we interpret the grouping?

## Hierarchical clustering

* An agglomerative approach
  + Find closest two things
  + Put them together
  + Find next closest
* Requires
  + A defined distance
  + A merging approach
* Produces
  + A tree showing how close things are to each other (sometimes called dendogram)

## How do we define close?

* Most important step
  + Garbage in -> garbage out
* Distance or similarity
  + Continuous - Euclidian distance (i.e. diagonal distance between two cities) [Rafalab lecture](http://rafalab.jhsph.edu/688/lec/lecture5-clustering.pdf)
  + Continuous - correlation similarity
  + Binary -Manhattan distance (i.e. like on a city block grid) [Wikipedia](http://en.wikipedia.org/wiki/Taxicab_geometry)
* Pick a distance/similarity that makes sense for your problem

## Hierarchical clustering -dist

* Important parameters: x, method
* Calculates pair-wise distance between all points in data.frame

## Merging points - complete

* First method is average linkage. The centre of gravity.
* Second method is complete linkage. The farthest two points in a cluster, take the distance between those.

## heatmap()

Uses hierarchical clustering function to visualize groups of data.

## Notes and further resources

* Gives an idea of the relationships between variables/observations
* The picture may be unstable
  + Change a few points
  + Have different missing values
  + Pick a different distance
  + Change the merging strategy
  + Change the scale of points for one variable
* But it is deterministic (no random starting point)
* Choosing where to cut isn’t always obvious
* Should be primarily used for exploration
* [Rafa's Distances and Clustering Video](https://www.youtube.com/watch?v=wQhVWUcXM0A)
* [Elements of statistical learning](http://www-stat.stanford.edu/~tibs/ElemStatLearn/)

|  |  |
| --- | --- |
| Code | Description |
| set.seed(1234)  par(mar=c(0,0,0,0))  x <- rnorm(12,mean=rep(1:3,each=4),sd=.2)  y <- rnorm(12,mean=rep(c(1,2,1),each=4),sd=.2)  plot(x,y,col="blue",pch=19,cex=2)  text(x+.05,y+.05,labels=as.character(1:12)) | For example below |
| dataFrame <- data.frame(x = x, y = y)  dist(dataFrame) | calculates distance between all the different pair-wise rows in data frame (defaults to Euclidean) |
| distxy <- dist(dataFrame)  hClustering <- hclust (distxy)  plot(hClustering) | plots cluster dendogram (doesn’t tell you how many clusters there are) |
| myplclust(hClustering, lab = rep(1:3, each = 4), lab.col = rep(1:3, each =4)) | using myplclust() function to make prettier dendogram (must source custom function) |
| dataFrame <- data.frame (x=x,y=y)  set.seed(143)  > dataMatrix <- as.matrix(dataFrame)[sample(1:12),]  > heatmap(dataMatrix) | Creates heatmap based on hierarchical clustering |
|  |  |

# K-means Clustering

Old technique, but very useful for summarizing high-level dimension data. See hierarchical clustering for requirements on defining close, etc.

* A portioning approach
  + Fix a number of clusters
  + Get “centroids” of each cluster
  + Assign things to closest centroid
  + Recalculate centroids
* Requires
  + A defined distance metric
  + A number of clusters
  + An initial guess as to cluster centroids
* Produces
  + Final estimate of cluster centroids
  + An assignment of each point to clusters

## kmeans()

* important parameters: x, centers, iter-max, nstart

## Notes and further resources

* K-means requires a number of clustesr
  + Pick by eye/intuition
  + Pick by cross validation/information theory etc
  + [Determining the number of clusters](http://en.wikipedia.org/wiki/Determining_the_number_of_clusters_in_a_data_set)
* K-means is not deterministic (if always get quite varying results then likely not stable)
  + Different # of clusters
  + Different number of iterations
* [Rafael Irizarry’s Distance and Clustering Video](https://www.youtube.com/watch?v=wQhVWUcXM0A)

|  |  |
| --- | --- |
| Code | Description |
| set.seed(1234)  par(mar=c(0,0,0,0))  x <- rnorm(12,mean=rep(1:3,each=4),sd=.2)  y <- rnorm(12,mean=rep(c(1,2,1),each=4),sd=.2)  plot(x,y,col="blue",pch=19,cex=2)  text(x+.05,y+.05,labels=as.character(1:12)) | For example below |
| dataFrame <- data.frame(x,y)  kmeansObj <- kmeans(dataFrame, centers = 3) | creates k-means object with three centres |
| par(mar = rep(0.2,4))  plot(x,y, col=kmeansObj$cluster,pch=19,cex=2)  points(kmeansObj$centers, col = 1:3, pch = 3, cex = 3, lwd = 3) | create scatter plot with points colored based on k-means cluster, then add k-means centers |
| set.seed(1234)  dataMatrix <- as.matrix(dataFrame[sample(1:12),])par(mfrow = c(1,2), mar = c(2,4,0.1,0.1))  image(t(dataMatrix)[,nrow(dataMatrix):1], yaxt="n")  image(t(dataMatrix)[,order(kmeansObj$cluster)] ,yaxt="n") | use heat map technique to visual data |
|  |  |
|  |  |

# Principle Components Analysis and Singular Value Decomposition

## Related problems

You have multivariate variables Xi,…. Xn so X1 = (X11, …. X1m)

* Find a new set of multivariate variables that are uncorrelated and explain as much variance as possible (principle components analysis)
* If you put all the variables together in one matrix, find the best matrix created with fewer variables (lower rank) that explains the original data (singular value decomposition)

The first goal is **statistical** and the second goal is **data compression**

## Related solutions - PCA/SVD

**SVD:**

If X is a matrix with each variable in a column and each observation in a row then the SVD is a “matrix decomposition”: X = UDVT

Where the columsn of U are orthogonal (left singular vectors), the columns of V are orthogonal (right singular vectors) and D is a diagonal matrix (singular values)

**PCA:**

Theprinciple components are equal to the right singular values if you first scale (subtract the mean, divide by the standard deviation) the variables.

## Notes and further resources

* Scale matters
* PC’s/SV’s may mix real patterns
* Can be computationally intensive
* [Advanced Data Analysis from an Elementary Point of View](http://www.stat.cmu.edu/~cshalizi/ADAfaEPoV/ADAfaEPoV.pdf)
* [Elements of statistical learning](http://www-stat.stanford.edu/~tibs/ElemStatLearn/)
* Alternatives
  + [Factor analysis](http://en.wikipedia.org/wiki/Factor_analysis)
  + [Independent components analysis](http://en.wikipedia.org/wiki/Independent_component_analysis)
  + [Latent semantic analysis](http://en.wikipedia.org/wiki/Latent_semantic_analysis)

# Plotting and Color in R

* The default color schemes for most plots in R are horrendous
* Recently there have been developments to improve the handling/specifications of colors in plots/graphs/etc
* There are functions in R and in external packages that are very handy

## Color Utilities in R

* the grDevices packages has two functions
  + colorRamp
  + colorRampPalette
* These functions take palettes of colors and help to interpolate between the colors
* The function colors() lists the names of colors you can use in any plotting function
* colorRamp: Take a palette of colors and return a function that takes values between 0 and 1, indicating the extremes of the color palette (e.g. see the ‘gray’ function)
* colorRampPalette: Take a palette of colors and return a function that takes integer arguments and returns a vector of colors interpolating the palette (like heat.colors or topo.colors)

## RColorBrewer Package

* One package on CRAN that contains interesting/useful color palettes
* There are 3 types of palettes
  + Sequential
  + Diverging
  + Qualitative
* Palette information can be used in conjunction with colorRamp() and colorRampPalette()

## Some Other Plotting Notes

* The rgb function can be used to produce any color via red, green, blue proportions
* Color transparency can be added via the alpha parameter to rgb
* The colorspace package can be used for different control over colors

## Summary

* Careful use of colors in plots/maps/etc can make it easier for the reader to get what you’re trying to say (why make it harder?)
* The RColorBrewer package is an R package that provides color palettes for sequential, categorical and diverging data
* The colorRamp and colorRampPalette functions can be used in conjunction with color palettes to connect data to colors
* Transparency can sometimes be used to clarify plots with many points

|  |  |
| --- | --- |
| Command | Description |
| pal <- colorRamp(c("red","blue")) | Create a color ramp palette where pal(0) ~ pal(1) returns color between red and blue |
| pal <- colorRampPalette(c("red","yellow")) | Create a color ramp palette where pal(1) ~ pal(n) returns series of hexadecimal values representing colors between red and yellow |
| library(RColorBrewer)  cols <- brewer.pal(3,"BuGn")  pal <- colorRampPalette(cols)  image(volcano,col=pal(20)) | Create palette based on RColorBrewer blue to green sequence  then create image from volcano dataset using palette |
| x <- rnorm(10000)  y <- rnorm(10000)  smoothScatter(x,y) | creates 2-d histogram (default is lighter blue as you get lower, darker green as you get higher) |
| plot(x,y,col=rgb(0,0,0,0.2),pch=19) | Using transparency to show how points overlap (to demonstrate density) |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |