# Course:

URL: <https://www.coursera.org/learn/exploratory-data-analysis>

# Week 1:

Edward Tufte – beautiful evidence

1. Show comparisons – evidence is always relative
2. Show causality, mechanism, make an explanation
3. Show multivariate data – identify confounding variables
4. Integration of evidence – make the presentation as information rich as possible
5. Describe evidence with labels, scales, sources etc.
6. Content is king – if not interesting, don’t tell it

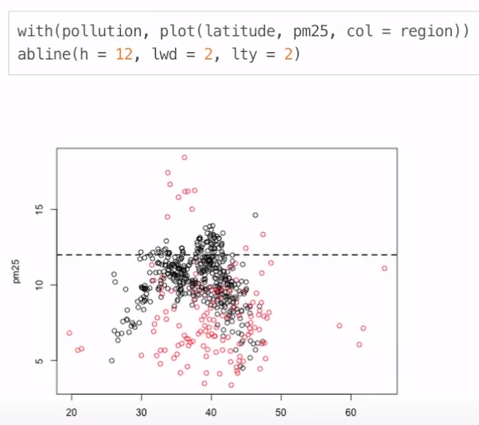
Exploration:

One Dimensional:

* Five number summary – summary()
* Boxplot – boxplot(, col = “blue”) –
  + abline(h=12) for a threshold
* Histogram – hist(, col = “green”) –
  + use rug() to plot all of the points
  + you can change breaks for finer detail, however don’t make the histogram too noisy
* Barplot – barplot(table(), col, main)

Two Dimensional:

* Multiple Boxplot with two variables
* Multiple histograms
* Scatterplot



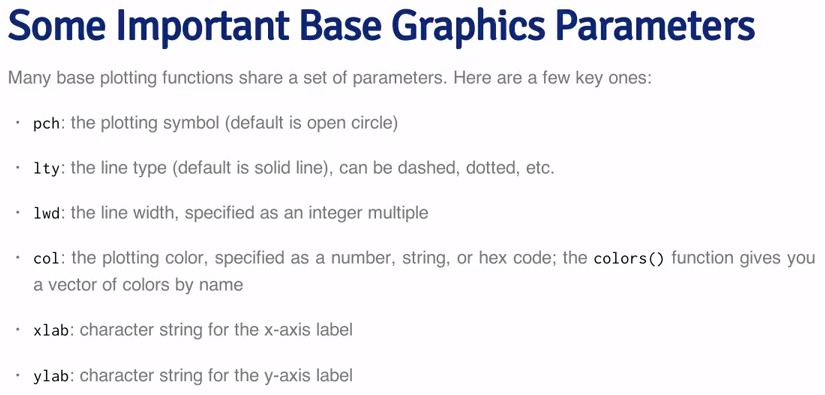
Plotting Systems in R:

* Base plotting system – start blank and layer on from there
  + Plot function to generate and annotation to add on
  + with(cars, plot(speed, dist))
* Lattice system – plots created with a single function call; need to specify a lot of information in the call to the function – good for co-plots; good for defaults
  + xyplot(Life.Exp ~ Income | region, data = state, layout = c(4, 1))
* ggplot2 – good defaults, similar to Lattice
  + qplot(displ, hwy, data = mpg)

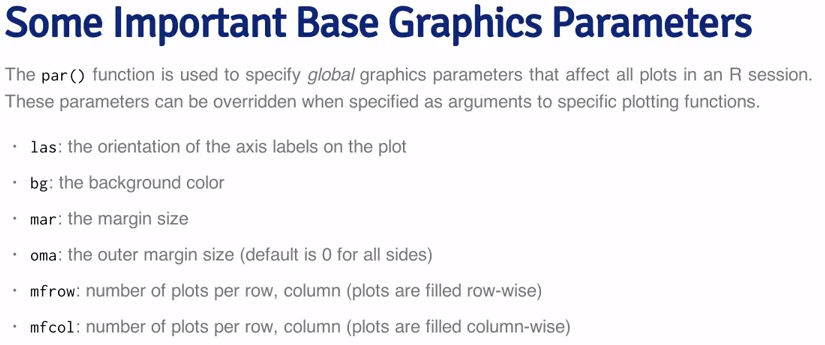
Base Plotting:

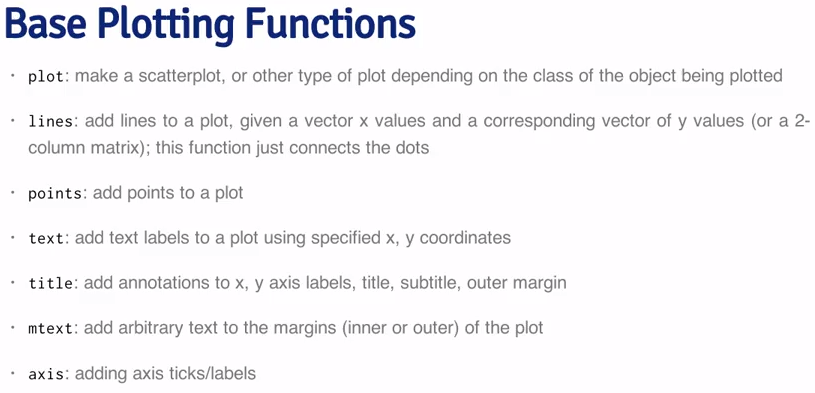
* Histogram – hist()

Base graphics parameters:

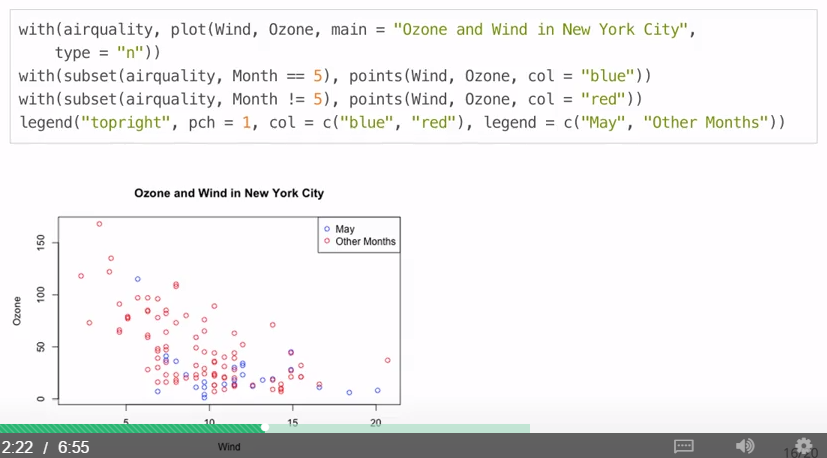


For example: plot(airquality$Ozone, pch = "z", col = "red", xlab = "IIIndex", ylab = "ozone")

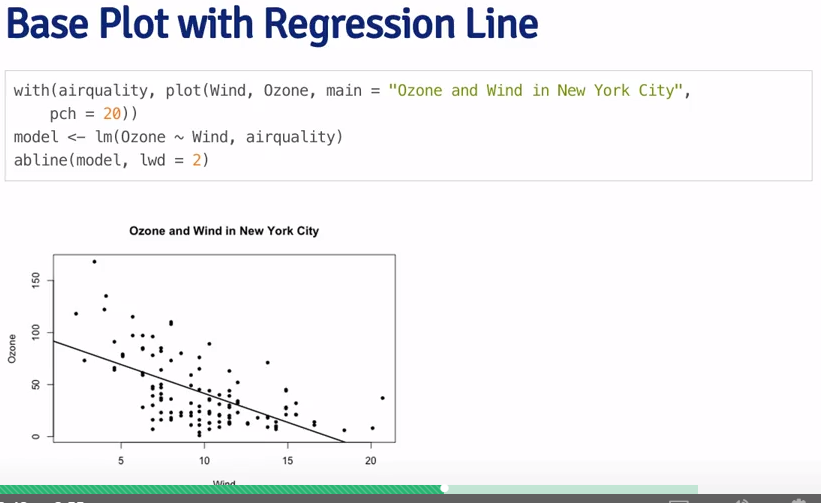


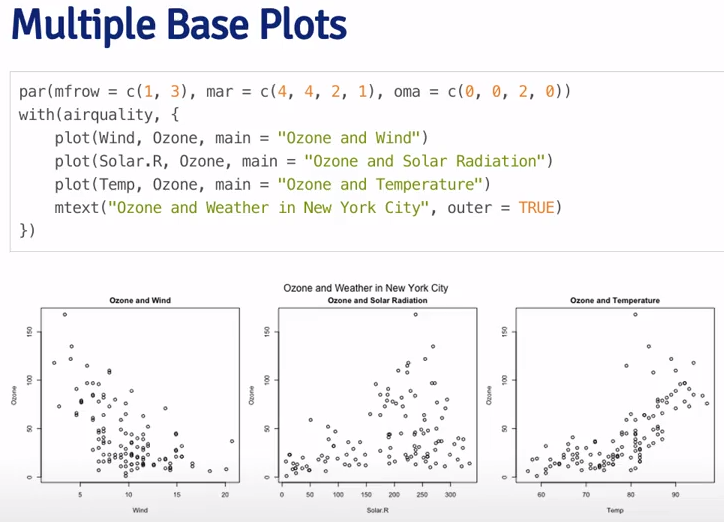


Example:



Adding a regression line:





* Demo: example(points)
* Colours: colours()

Plotting to a file:

* Open file: pdf(file = “”)
* … insert plot commands
* Close graphics device: dev.off()

Graphics devices:

* Open window: windows()
* Current active devices: dev.cur()
* Set the device: dev.set()

Copying plots:

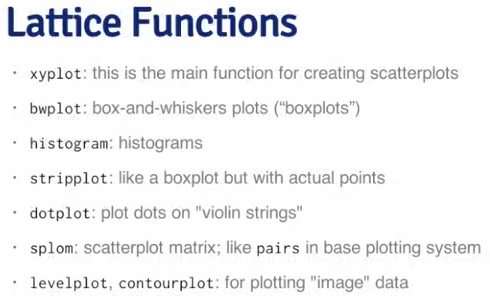
* Copy: dev.copy – from screen to file device
* To pdf: dev.copy2pdf – from screen to pdf

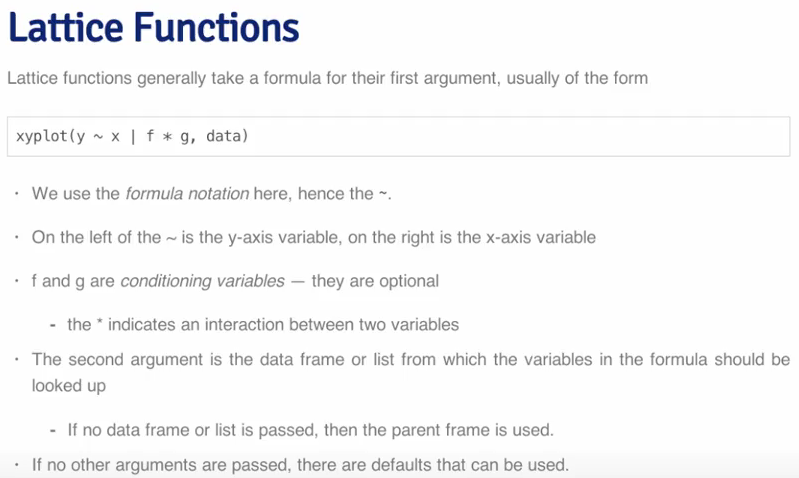
From swirl:

* Save typing dataset multiple times: with(pollution, plot(latitude, pm25))
* List of all devices: ?devices
* Dimensions: dim()
* Range: range()
* Building a legend through concatenation: legend("topright", pch = c(17,8), col = c("blue", "red"), legend = c("May", "Other Months"))

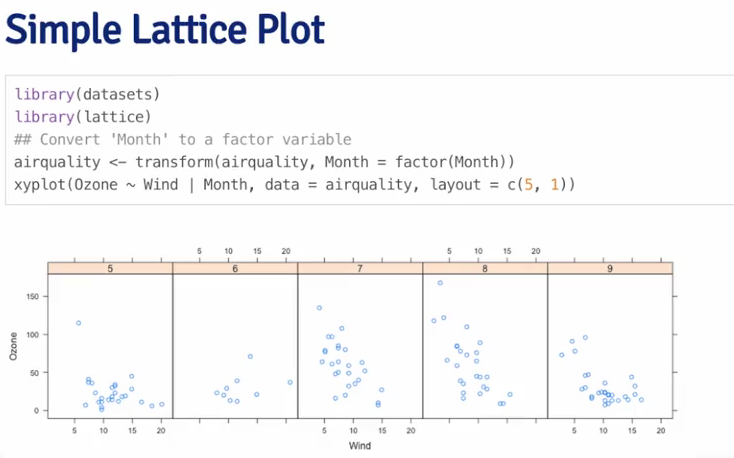
# Week 2:

## Lattice Functions:



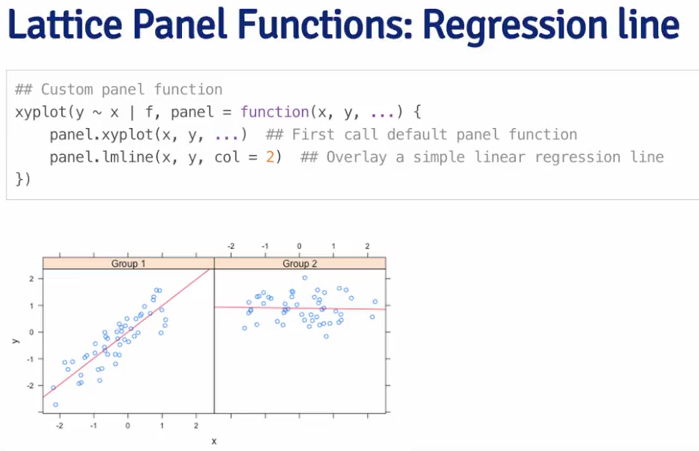


The power is having multi-dimension panels with a single function call:



You can save the plot to an object, or auto-print

More complicated – custom function, call the plot and then add a regression line:



Note: can’t use base plotting annotation in Lattice

Good for many panel plots – for looking at a lot of data

## Ggplot2

library(ggplot2)

str(mpg)

# simplest plot - x, y and data

qplot(displ, hwy, data = mpg)

# modify aesthetics

qplot(displ, hwy, data = mpg, col = drv, geom = c("point", "smooth"))

qplot(displ, hwy, data = mpg, shape = drv)

# single variable histogram

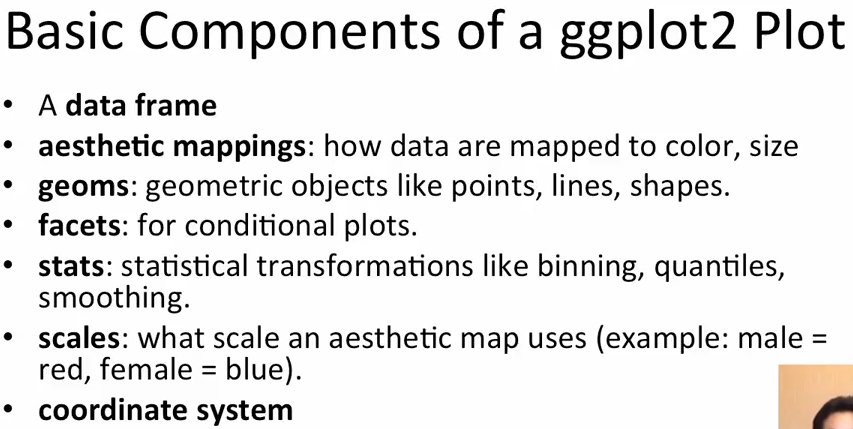
qplot(hwy, data = mpg, fill = drv)

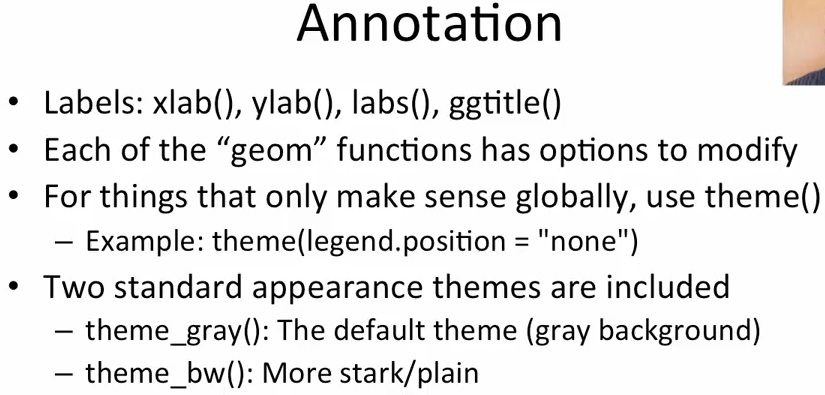
qplot(hwy, data = mpg, col = drv, geom = "density")

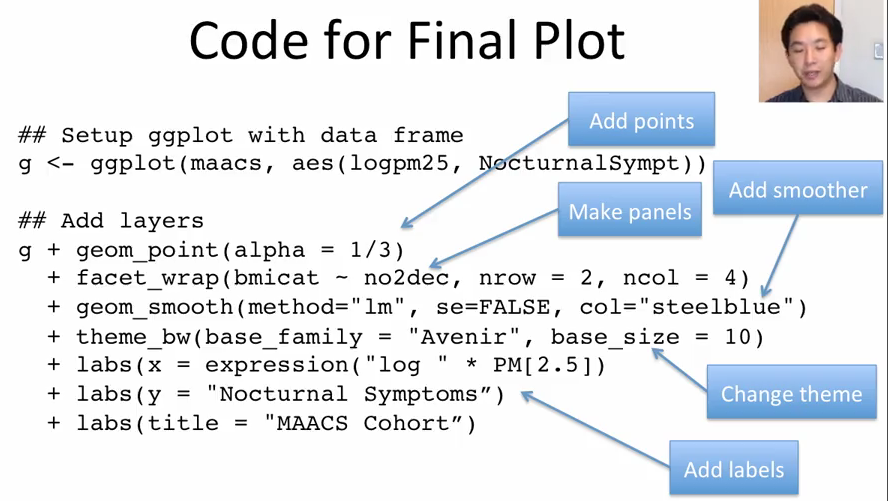
# Facets are like panels in Lattice - use the tilde to indicate number of rows

qplot(displ, hwy, data = mpg, facets = .~drv)

qplot(hwy, data = mpg, facets = drv~.)







# part 2

# qplot doesn't reveal the underlying infrastructure of ggplot, these are basic fundamentals

# verb - noun - adjectives

# you can build layers; plot the data, overlay summary, metadata and annotation

qplot(displ, hwy, data = mpg, facets = .~drv, geom = c("point", "smooth"))

# initial call to ggplot

g <- ggplot(mpg, aes(displ, hwy))

# summary of g

summary(g)

# add geom point layer - with auto print (or save to object too)

g + geom\_point()

# add another layer - smoother

g + geom\_point() + geom\_smooth()

# change default for smooth from "low s" due to noise, to linear regression line (less flexible, but sense of the trend)

g + geom\_point() + geom\_smooth(method = "lm")

# another layer - facets

g + geom\_point() + geom\_smooth(method = "lm") + facet\_grid(. ~drv)

# ggplot requirest that data has good labels as they show on the graphs

# annotation

# points with constant values and alpha transparency

g + geom\_point(color = "steelblue", size = 4, alpha = 1/2)

# points with data variable - you need to use aesthetics function (aes) for this

g + geom\_point(aes(color = drv), size = 4, alpha = 1/2)

# add labels

g + geom\_point(aes(color = drv), size = 4, alpha = 1/2) + labs(title = "play")

# customize the smooth, with no confidence interval

g + geom\_point(aes(color = drv), size = 4, alpha = 1/2) + labs(title = "play") + geom\_smooth(method = "lm", size = 4, linetype = 3, se = F)

# change the theme

g + geom\_point(aes(color = drv), size = 4, alpha = 1/2) + labs(title = "play") + theme\_bw()

# use axis limits to remove outliers, but be careful not to just remove the points - coord\_cartesian

# use the cut() function to cut continuous variables into ranges in order to use as facets

# can store to an object for future use when you can add more layers

h <- g + geom\_point(aes(color = drv), size = 4, alpha = 1/2) + labs(title = "play") + theme\_bw()

summary(h)

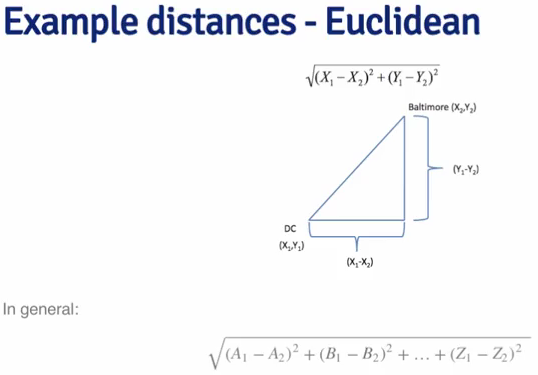
# Week 3:

## Hierarchical Clustering

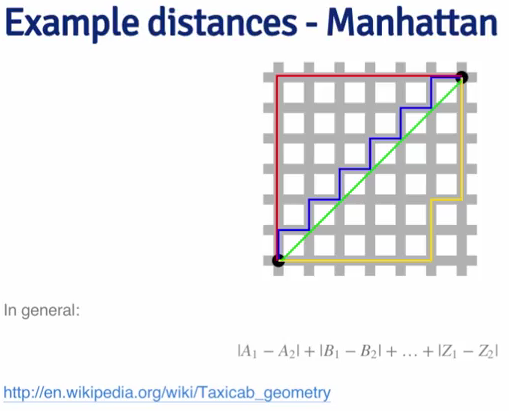
Clustering is organizing things that are close – what defines close, how do we group, how do we visualize, how do we interpret?

Agglomerative approach is about finding the closest points, putting them together into a new point, removing the old and then finding the next closest points… it requires a distance metric and an approach for merging the points. It produces a tree of how things are merged together.

* How do we define close? Distance can be continuous (Euclidean), correlation or binary (Manhattan)
* Euclidean:



* For Euclidean; take the differences, square them, add them together and square root



* Manhattan may be more accurate in some circumstances

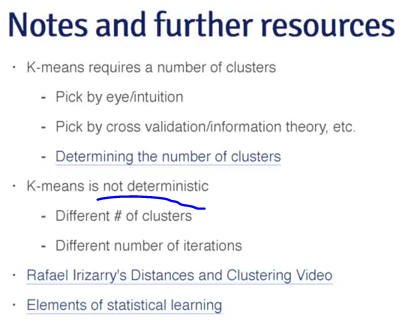
There are different ways of merging points – complete distances, or average

Choosing where to cut is not always obvious… ☺

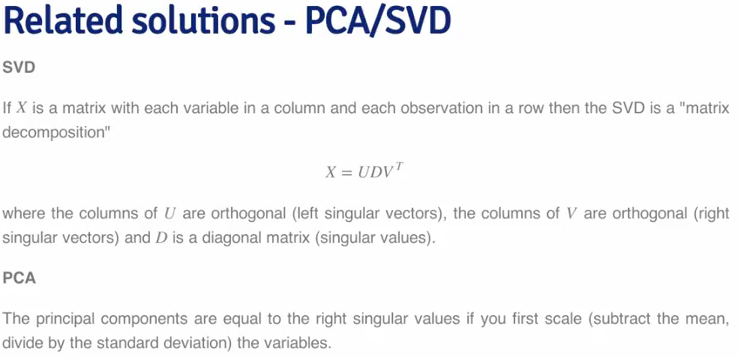
## K-means Clustering

Partitioning a group of points into a set number of clusters, each with a centroid.

1. Each point is assigned to its closest centroid.
2. Recalculate the centroid.
3. Reassign points to the closest centroid.
4. Again, update the centroids.
5. Etc.

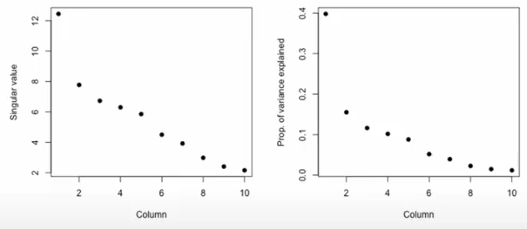


## Principal Component Analysis and Singular Value Decomposition (Dimension Reduction)



Singular Value Decomposition

* Look for the change in the means of the rows and columns
* Variance explained looks at the diagonal matrix, tries to explain how much of the total variance is explained by a value e.g. 40% is explained by a single dimension:

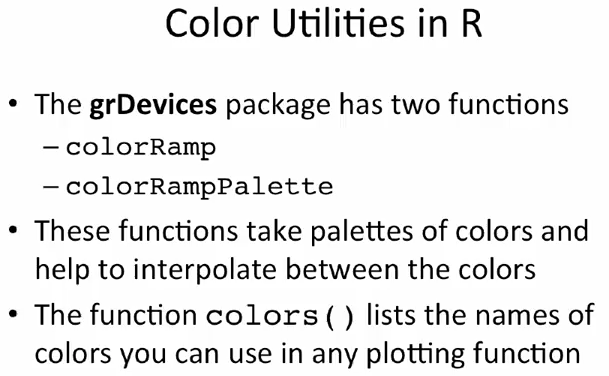


* SVD and Principal Component do the same thing

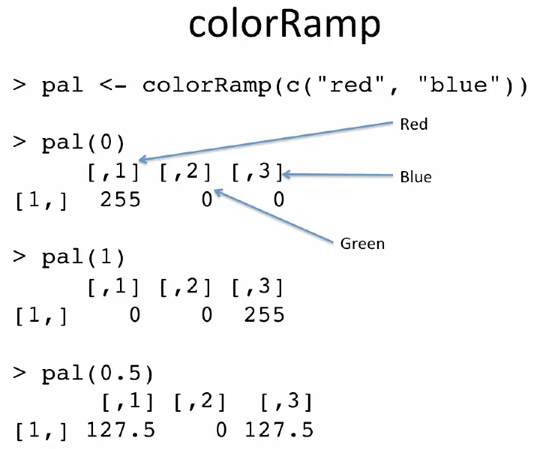
Be careful of missing values – it throws an error

* You can treat these by imputing, based on neighboring values
* You can do this with e.g. images, represented as matrix and convert to a lower rank matrix
  + Run SVD to understand how much variance is explained
  + Then create approximations based on fewer components than the original dataset

## Plotting and Color

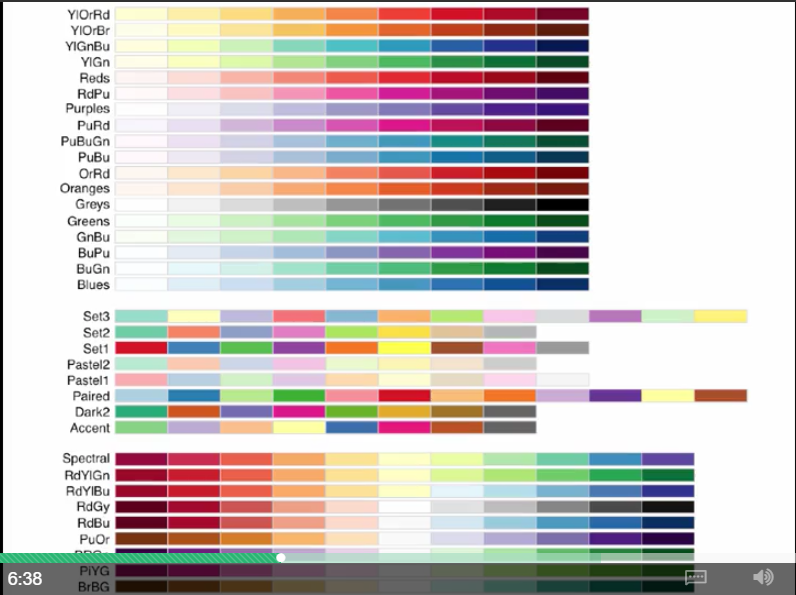


colorRamp returns a function of the colors, the function can be stored in an object:



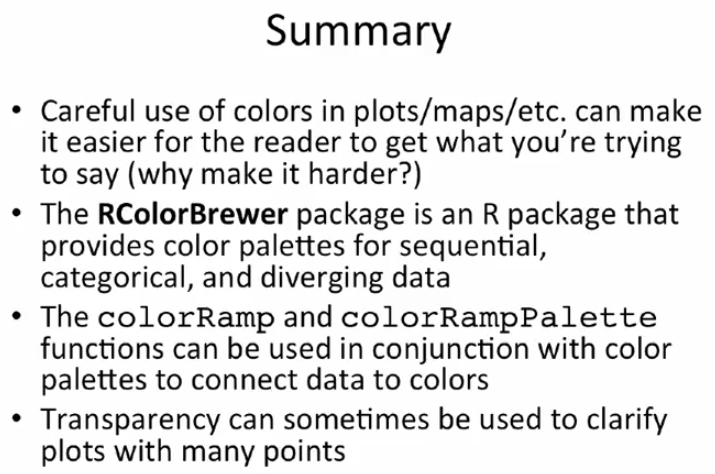
RColorBrewer can be used for creating new colors – these can be passed to ColorRamp etc.

There are 3 categories; sequential (ordered data), diverging (categorical data), and diverging (negative to positive)…



Other functions:

* Function rgb() – use the alpha parameter to specify transparency



# Week 4

## Case Studies

1. Plot
2. Cluster
3. Plot by other variables (max acceleration)
4. Singular value decomposition – see how the singular vectors separate values out
5. Find the maximum contributor – cluster with the which.max contributor
6. K-means clustering – first try with nstart = 1

## Case Study 2: air quality

* Reading the file
  + readLines() is good for reading just one line of a file (e.g. a header)
  + dim() to see how many observations
  + names() to assign the readLines() to the column names
  + make.names() to remove spaces in column names
* Look at the data
  + pull out the column that you’re investigating
  + see what kind of class() it is
  + do a str() to see how many values
  + summary() to see what it looks like and do rudimentary comparisons (e.g. mean)
  + mean(is.na) to determine what % of the dataset is missing – need to think are missing values important?
* Visualize and Negative
  + boxplot(x1, x0) – skew in data can make this difficult
  + use a log boxplot(log10(x1), log10(x0)) to even out the data
  + look for negatives – negative <- x1 < 0 – to create a logical vector and find the mean
  + look for the dates they were negative
* Pick one monitor
  + Subset() on state
  + Paste() to concatenate fields
  + Find the intersection intersect() to find where both exist
  + Split and count number of rows with sapply(split())
  + Plot them, include median
  + Ensure ranges of the y-axis are the same:
    - Calculate the range()
    - Set ylim = range
  + Look for the spread of data – e.g. averages and extreme values are all decreasing
* Average value by State
  + Mean of a vector within sub-groups determined by another vector – use tapply()
  + Create a data frame of the values
  + Merge the data frames
  + Plot and connect with a line
    - Plot – use xlim for years
    - Points