REVOLUTION ANALYTICS

Fundamentals in R: Data Exploration

August 12, 2014

Module 4 - Data Expoloration

In this module we'll cover subsetting, summarizing and visualizing data. We'll cover how to remove duplicates, learning about loops and custom functions in the process. Lastly, we'll see how we can merge two data sets.

The objectives are

- Become comfortable in subsetting using [] and subset()
- Understand various ways to remove duplicates
- Be able to summarize data numerically and visually
- Be able to aggregate and merge data
- Have a basic understanding of how to write your own custom function

Load Your Work

```
load("data/Module3_songData.RData")
```

Subsetting Data

We can *subset* into the data frame and select specific rows & columns

```
songSubset <- songData[1:100, 1:10]
```

```
class(songSubset)
```

```
## [1] "data.frame"
```

```
rm(songSubset)
```

Subsetting Data

… alternatively

```
songSubset2 <- songData[, c(1, 3, 3, 9)]
rm(songSubset2)
```

See

```
?Extract
```

Subsetting Data

… we can *subset* using column names (or row names) instead of column numbers (or row numbers)

```
songSubset3 <- songData[, c("artist.name", "artist.hotttnesss", "artist
    "artist.location", "artist.latitude", "artist.longitude", "song.hot
    "tempo", "duration", "term", "hfRatio")]
```

… or we can *subset* using a vector of logicals. The vector must be the same length as the number of columns (or rows, if subsetting rows)

```
songSubset3 <- songData[, c(FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRU
    TRUE, TRUE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE,
    FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, TRUE, TRUE,
    FALSE, FALSE, TRUE, FALSE, TRUE, TRUE)]
```

Subsetting Data

Instead of typing out the entire vector, we can create it

```
colSelect <- names(songData) %in% c("artist.name", "artist.hotttnesss",
    "artist.location", "artist.latitude", "artist.longitude", "song.hott
    "tempo", "duration", "term", "hfRatio")

colSelect
```

```
##  [1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [23] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE
## [34] FALSE  TRUE  TRUE
```

```
songSubset3 <- songData[, colSelect]
```

Subsetting Data

Do we have any duplicate artists in our subset? If so, which are the 5 most duplicated artists?

```
summary(songSubset3)
```

```
##                artist.name   artist.latitude artist.longitude
##  Mario Rosenstock         :  13   Min.   :-41     Min.   :-162
##  Aerosmith                :  12   1st Qu.: 34     1st Qu.: -93
##  Phil Collins             :  12   Median : 39     Median : -80
##  Sugar Minott             :  12   Mean   : 37     Mean   : -64
##  The Jackson Southernaires:  12   3rd Qu.: 44     3rd Qu.:  -9
...
```

Exercise

Create a new data frame called songAerosmith containing all rows of songSubset3 where the artist name is “Aerosmith”. Your code should return a data frame with 12 rows and 11 columns.

Bonus: Take a look at the documentation for the subset() functions. See if you can do the same using subset()

How many missing observations of song.hottnesss does this subset have?

Solution

```
rowSelect <- songSubset3$artist.name == "Aerosmith"
songAerosmith <- songSubset3[rowSelect, ]
dim(songAerosmith)

## [1] 12 11

# alternatively, using subset() instead of []
songAerosmith <- subset(songSubset3, subset = artist.name == "Aerosmith"
```

Solution

```
summary(songAerosmith)  # identify number of NA's for song.hotttnesss
```

```
##                                    artist.name artist.latitude
##  Aerosmith                                :12   Min.   : NA
##  !!!                                      : 0   1st Qu.: NA
##  (hed) p.e.                               : 0   Median : NA
##  :Blacks On :Blondes                      : 0   Mean   :NaN
##  [kaleidoskop]                            : 0   3rd Qu.: NA
...
```

```
summary(songAerosmith$song.hotttnesss)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   0.415   0.434   0.505   0.530   0.588   0.731       6
```

Working with Missing Values

Note that song.hotttness variable in songAerosmith data frame has 6 missing values.

To replace missing values with the average value across all non-missing observations

```
mean(songAerosmith$song.hotttnesss, na.rm = TRUE)  # calculate replacem
```

```
## [1] 0.5304
```

```
rowIndex <- is.na(songAerosmith$song.hotttnesss)  # identify
```

Working with Missing Values

```
songAerosmith[rowIndex, ]   # subset
```

```
##    artist.name artist.latitude artist.longitude artist.location
## 389   Aerosmith              NA               NA      Boston, MA
## 393   Aerosmith              NA               NA      Boston, MA
## 394   Aerosmith              NA               NA      Boston, MA
## 395   Aerosmith              NA               NA      Boston, MA
## 396   Aerosmith              NA               NA      Boston, MA
...
```

Working with Missing Values

```
songAerosmith$song.hotttnesss[rowIndex] <- mean(songAerosmith$song.hott
    na.rm = TRUE)  # replace

songAerosmith$song.hotttnesss  # verification 1

##  [1] 0.5304 0.4150 0.4433 0.4314 0.5304 0.5304 0.5304 0.5304 0.5955 0.73
## [11] 0.5662 0.5304

mean(songAerosmith$song.hotttnesss)  # verification

## [1] 0.5304
```

Deduping Data

Removing Duplicates: How can we ensure that we only have one entry for each artist? What will will choose to do with the values in the other columns

```
songAerosmith
```

```
##    artist.name artist.latitude artist.longitude artist.location
## 389   Aerosmith              NA               NA      Boston, MA
## 390   Aerosmith              NA               NA      Boston, MA
## 391   Aerosmith              NA               NA      Boston, MA
## 392   Aerosmith              NA               NA      Boston, MA
## 393   Aerosmith              NA               NA      Boston, MA
...
```

Deduping Data

We can remove duplicates, ignoring other columns… keep just 1 observation (first occurring) for values of other columns

```
songDedup <- songSubset3[!duplicated(songSubset3[1]), ]
summary(songDedup)
```

```
##                                      artist.name   artist.latitude
##  !!!                                        :   1   Min.   :-41.3
##  (hed) p.e.                                 :   1   1st Qu.: 34.1
##  :Blacks On :Blondes                        :   1   Median : 39.1
##  [kaleidoskop]                              :   1   Mean   : 37.5
##  089 Clique feat. Minnesota Snipe & Skinny Cueball:   1   3rd Qu.: 43.6
...
```

Deduping Data

We lose some information here… what?

```
songDedup[songDedup$artist.name == "Aerosmith", ]
```

```
##    artist.name artist.latitude artist.longitude artist.location
## 389   Aerosmith             NA              NA      Boston, MA
##    artist.hotttnesss artist.familiarity tempo duration song.hotttnesss
## 389            0.6107           0.8725 108.7      425             NA
##      term hfRatio
## 389 rock  0.6999
```

We have song hotttnesss information about half of the observations corresponding to Aerosmith. However, our deduping exercise left us with just the observation missing a value for song hottnesss.

Deduping Data

A better way may be to average song hotttnesss across artists. Instead of obtaining 1 observation for each artist, we can return the average of all observations for that artist

We can write a for loop, we'll replace the value of song hotness obtained in the previous de-duping with the average song hotness

```
for (i in unique(songSubset3$artist.name)) {
    sSub <- subset(songSubset3, artist.name == i)
    meanHot <- mean(sSub$song.hotttnesss, na.rm = TRUE)
    songDedup[songDedup$artist.name == i, "song.hotttnesss"] <- meanHot
}
```

Deduping Data

```
head(songDedup)
summary(songDedup)
```

```
songDedup[songDedup$artist.name == "Aerosmith", ]
```

```
##    artist.name artist.latitude artist.longitude artist.location
## 389   Aerosmith             NA              NA      Boston, MA
##    artist.hotttnesss artist.familiarity tempo duration song.hotttnesss
## 389            0.6107           0.8725 108.7      425         0.5304
##      term hfRatio
## 389 rock  0.6999
```

That took some time! There's a better way – the aggregate() function. See 'Aggregating Data'.

Summarizing Data

R provides a multitude of tools for exploratory data analysis, summarization and visualization.

Useful functions for birds eye view summaries: structure using str() and basic statistical summary using summary()

```
str(songSubset3)
summary(songSubset3)
```

Summarizing Data

Cross tabulations using xtabs() returns to number of observations at each value of a given variable.

```
xtabs(~term, songSubset3)
```

```
## term
##       ballad        blues      chanson     chill-out      country
##           47          385          208           90           75
##    dancehall        disco          dub easy listening         folk
##           97           58           19           65          141
##         funk       grunge      hip hop        house         jazz
...
```

Summarizing Data

Cross tabulations using xtabs can be used to return the number of observations at each combination of two or more variables.

```
songSubset3$song.hotttnesssCut <- cut(songSubset3$song.hotttnesss, breal
    na.rm = TRUE)[2:5], labels = c("low", "med", "high"))

xtabs(~term + song.hotttnesssCut, songSubset3)
```

```
##                 song.hotttnesssCut
## term             low med high
##   ballad          10   9    2
##   blues           45  35   24
##   chanson         30  25    6
##   chill-out       18  12   14
```

...

Summarizing Data

prop.table() can be used to return percent instead of count. The margin argument specifies percent of what – total across rows, columns or both.

Which term has the highest proportion of songs rated *highly* in hotttnesss?

```
prop.table(xtabs(~term + song.hotttnesssCut, songSubset3), margin = 1)
```

```
##                 song.hotttnesssCut
## term                 low     med    high
##   ballad         0.47619 0.42857 0.09524
##   blues          0.43269 0.33654 0.23077
##   chanson        0.49180 0.40984 0.09836
##   chill-out      0.40909 0.27273 0.31818
...
```

Sorting Data

To better identify which terms have the highest proportion of songs rates .538 and higher, we reorder the proportion table just generated by the third column.

```
songTable <- prop.table(xtabs(~term + song.hotttnesssCut, songSubset3),

colIndex <- order(songTable[, 3], decreasing = TRUE)

songTable <- songTable[colIndex, ]

head(songTable, 3)
```

```
##          song.hotttnesssCut
## term          low    med   high
##   punk    0.12057 0.2624 0.6170
##   techno 0.06667 0.3333 0.6000
##   metal   0.11935 0.2968 0.5839
```

Exercise

Use xtabs() and order() to identify the top 10 terms by observation count in songSubset3.

Poll: Which term is most frequently occuring in our dataset?

- pop
- rock
- punk
- blues
- techno
- disco
- hip hop

Solution

```
songTable2 <- xtabs(~term, songSubset3)

songTable2 <- songTable2[order(songTable2, decreasing = TRUE)]

songTable2[2:11]  # rock is the most frequently occuring term
```

```
## term
##   rock    pop  metal   jazz  blues hip hop  house    rap   punk
##   1619    777    485    461    385    364    304    283    213
## chanson
##     208
```

Visualizing Data

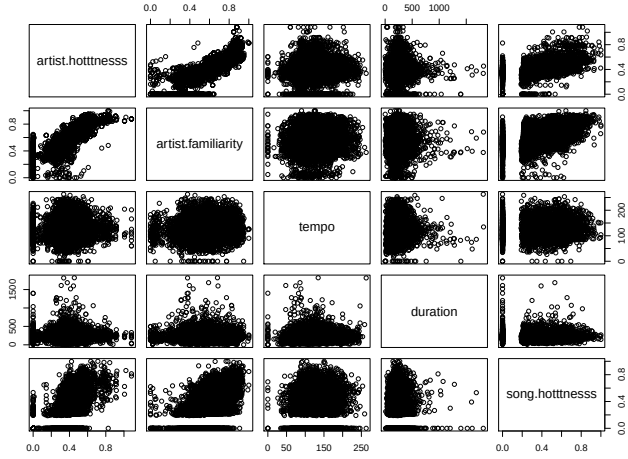
R ships with a variety of tools for visualizing data. Using plot() on a data frame with multiple (numeric!) columns returns a scatter plot matrix.

```
cols <- names(songSubset3)[sapply(songSubset3, is.numeric)]
cols
```

```
## [1] "artist.latitude"   "artist.longitude"  "artist.hotttnesss"
## [4] "artist.familiarity" "tempo"               "duration"
## [7] "song.hotttnesss"    "hfRatio"
```

```
plot(songData[, cols[3:7]])
```

Visualizing Data

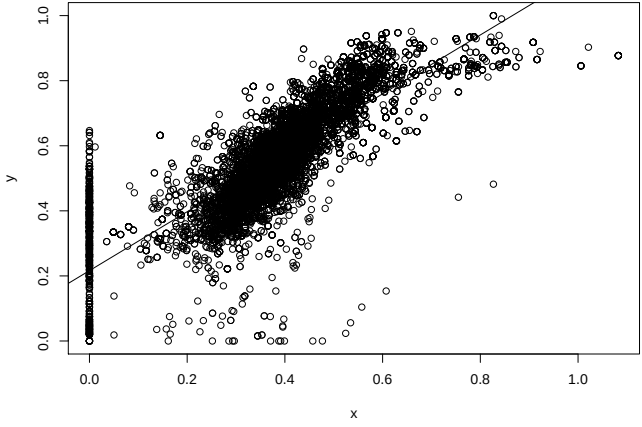


Visualizing Data

Using plot on two vectors (two arguments) creates a single scatter plot. We add an abline showing the results of linear regression on top.

```
x <- songSubset3$artist.hotttnesss
y <- songSubset3$artist.familiarity

plot(x, y)
abline(lm(y ~ x))
```

Visualizing Data



Visualizing Data

Summarizing the model returned by the linear regression

```
summary(lm(y ~ x))
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
...
```

Exercise

Create a new data frame that excludes observations of songSubset3 where artist.hotttnesss or artist familiarity is 0. Regenerate the previous plot & model based on this new data frame.

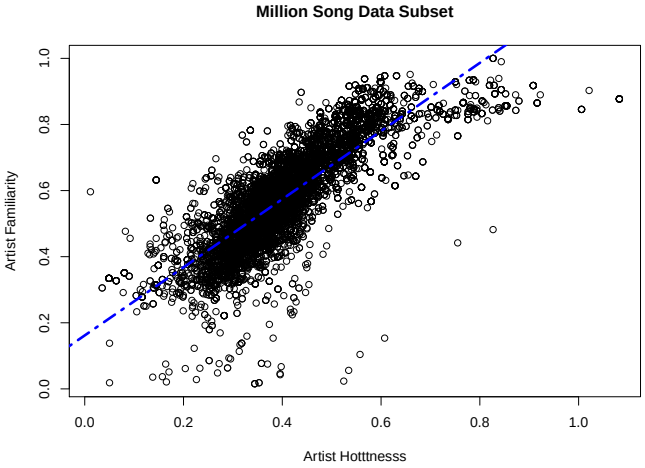
Take a look at the documentation for plot. Add axis titles (xlab, ylab) and a main title (main).

Take a look at the documentation for abline. Change the color of the line to “blue” (col), line width to “3” (lwd) and the line type to “twodash” (lty)

Solution

```
tempSubset <- songSubset3[songSubset3$artist.hotttnesss > 0 & songSubset
    0, ]
x <- tempSubset$artist.hotttnesss
y <- tempSubset$artist.familiarity

plot(x, y, xlab = "Artist Hotttnesss", ylab = "Artist Familiarity", mai
abline(lm(y ~ x), col = "blue", lwd = 3, lty = 6)
summary(lm(y ~ x))
```

Solution




```
##
## Call:
## lm(formula = Y ~ X)
```

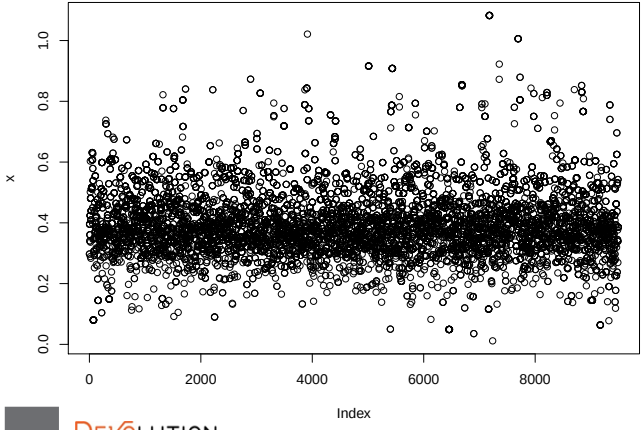
Visualizing Data

Note that, by default…

* `plot([2+ column dataframe])` returns a scatter plot matrix.
* `plot([two vectors])` returns one scatter plot.
* `plot([one vector])` returns one scatter plot.
* `plot([lm object])` returns 4 plots that can be used to evaluate the
* `plot([density object])` returns one scatter plot.
* `hist([one vector])` returns a histogram.

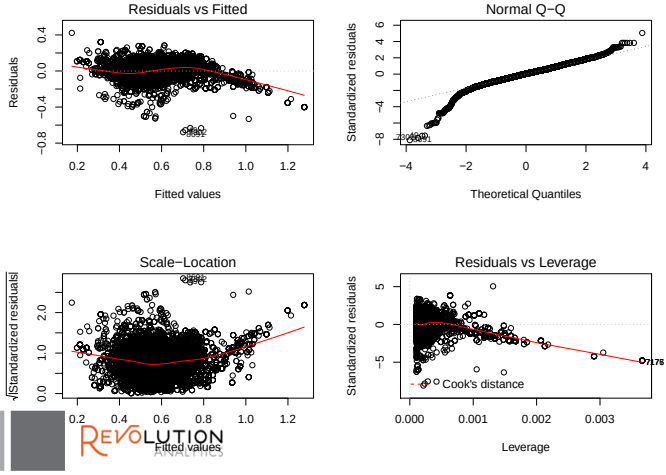
Visualizing Data

```
plot(x)
```



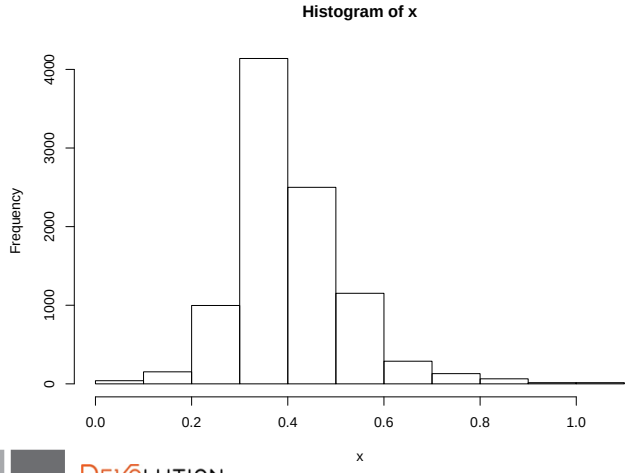
Visualizing Data

```
par(mfrow = c(2, 2))
plot(lm(y ~ x))
```



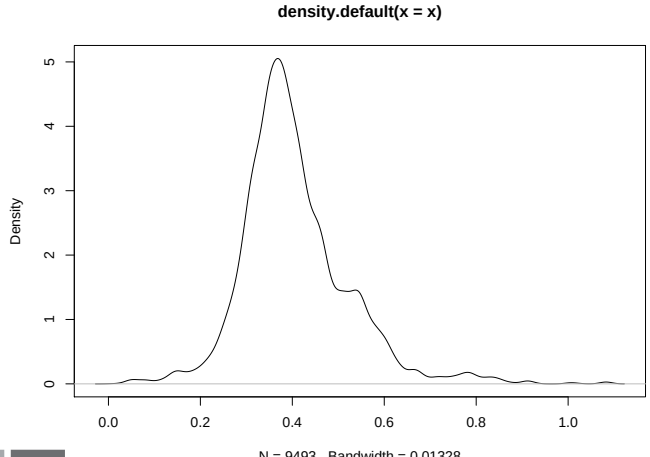
Visualizing Data

```
hist(x)
```



Visualizing Data

```
plot(density(x))
```



Visualizing Data

Plot is flexible function: the output depends on the input, the input can be a variety of different R objects.

For each valid input object class, a method is defined. When `plot(data.frame)` is run, `plot.data.frame` gets called which, in turn, calls `pairs()`. You can see `plot.data.frame` specific arguments by looking at the help file.

```
methods(plot)
help(plot.data.frame)
```

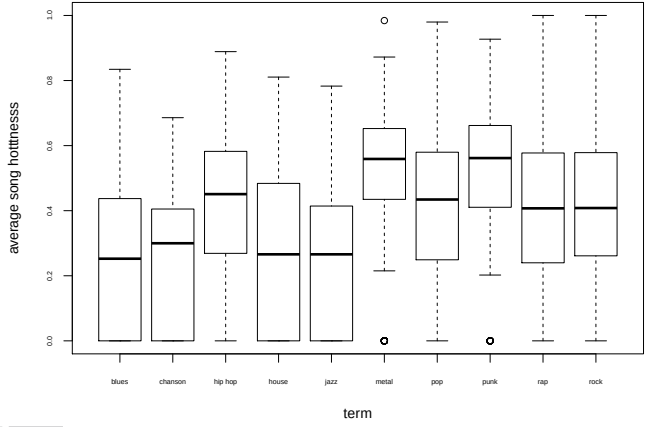
Visualizing Data

Additional Basic plots.

```
load("data/songTermHot.RData")

boxplot(song.hotttnesss ~ term, data = songTermHot, cex.axis = 0.5, xlab
    ylab = "average song hotttnesss")
```

Visualizing Data

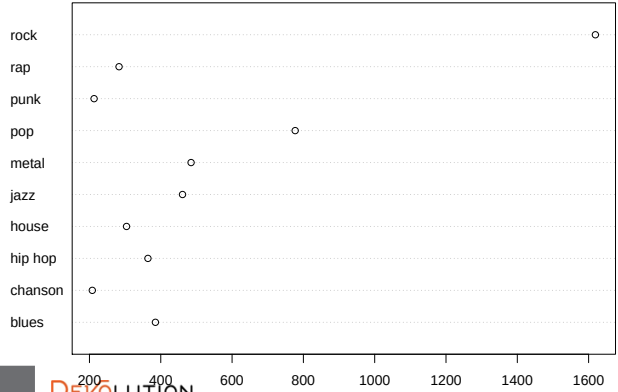
Additional Basic plots.



Visualizing Data

Additional Basic plots.

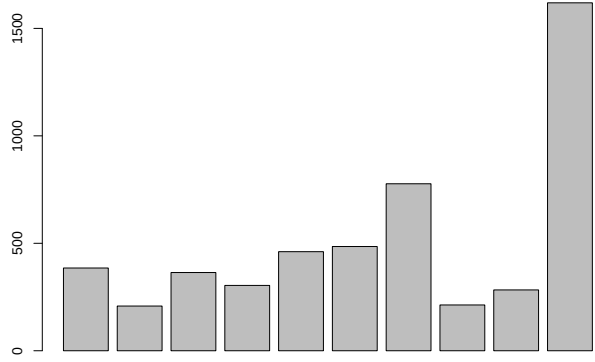
```
dotchart(xtabs(~songTermHot$term))
```



Visualizing Data

Additional Basic plots.

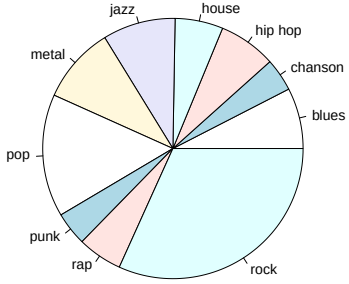
```
barplot(xtabs(~songTermHot$term))
```



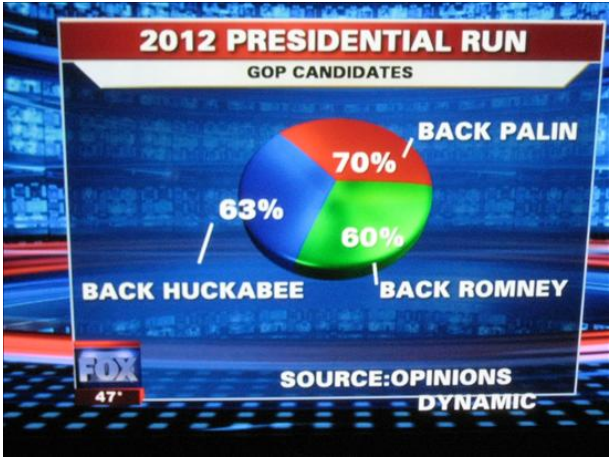
Visualizing Data

Additional Basic plots.

```
pie(xtabs(~songTermHot$term))
```



Pie Charts are Bad



http://www.graphgraph.com/2011/12/pie-charts-are-terrible/

Extras

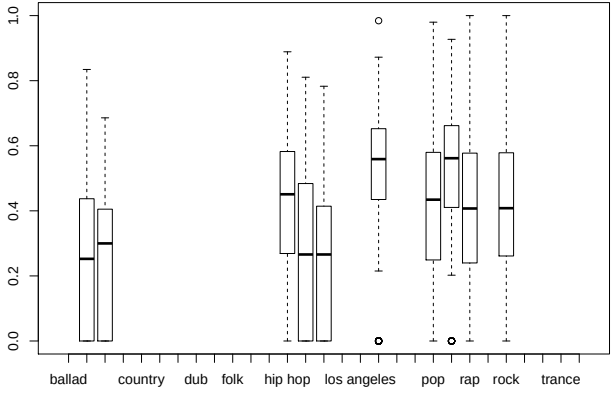
Factor Variables

Why did we load in a separate data set to generate plots for the top 10 most popular terms in our data set?

Running boxplot() on just a subset of our data pertaining to the top 10 most popular terms:

```
topTerms <- sort(xtabs(~term, songSubset3), decreasing = TRUE)
topTerms <- names(topTerms[2:11])

boxplot(song.hotttnesss ~ term, data = songSubset3[songSubset3$term %in%
    ])
```

Extras


0.0
0.2
0.4
0.6
0.8
1.0
ballad
country
dub
folk
hip hop
los angeles
pop
rap
rock
trance

Extras

We have correctly subset our data to include only observations corresponding to the top 10 terms. However, term is a factor variables whose levels were defined upon import (read.csv()) to include all unique values of term, not just the top 10. Even though out data subset does not include any other observations but those corresponding to the top 10 terms, place holders for all terms exist.

```
str(songSubset3$term[songSubset3$term %in% topTerms])
```

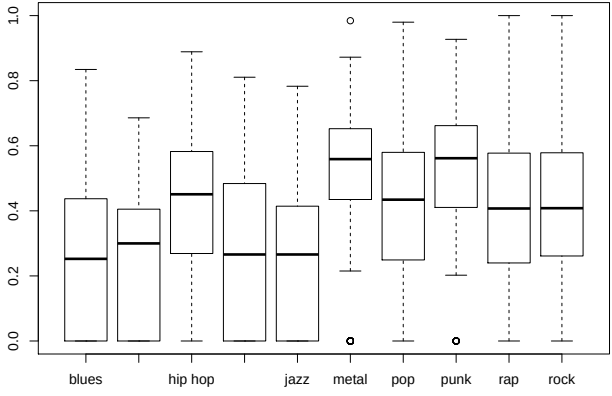
```
##  Factor w/ 29 levels "ballad","blues",..: 25 25 22 22 22 22 2 2 2 25 ...
```

Extras

We can redefine term as a factor variable with levels corresponding to only those unique values in our subset.

```
songTermHot <- songSubset3[songSubset3$term %in% topTerms, ]
songTermHot$term <- factor(songTermHot$term)  # this is key

boxplot(song.hotttnesss ~ term, data = songTermHot)

# save(songTermHot, file='songTermHot.RData')
```

Extras



Extras

Facts about Factor Variables: * for categorical data (numeric or string) * can be more memory efficient than strings * explicit levels/labels mean better graphs & tables * modeling of categorical variables in, say, lm()

… more information: http://www.ats.ucla.edu/stat/r/modules/factor_variables.htm

Extras

Custom Functions

One way to extend the functionality of R is to install and load additional packages and their containing functions. Another is to write your own functions.

Given a vector of values, we can normalize the values (ensure they have a mean of 0 and sd of 1) by, for each value, subtracting the mean and dividing the result by the sd.

```
x <- songData$song.hotttnesss
x <- (x - mean(x, na.rm = TRUE))/sd(x, na.rm = TRUE)
head(x)
```

```
## [1]      NA      NA      NA -1.387      NA      NA
```

Extras

Instead of recalculating x using these steps for each variable we'd like to normalize, we can define a custom function and apply it to a given variable.

```
normfunct <- function(x = songData$song.hotttnesss) {
    (x - mean(x, na.rm = TRUE))/sd(x, na.rm = TRUE)
}

x <- normfunct()
y <- normfunct(songData$duration)
```

Exercise

Use songData to create the perfect running playlist for efficient stride of 90 steps/minute/leg (180 or 90 bpm).

(http://gizmodo.com/5906815/the-most-mathematically-perfect-playlist-for-running/all)[http://gizmodo.com/5906815/the-most-mathematically-perfect-playlist-for-running/all]

tempo: tempo in BPM according to The Echo Nest

```
head(songData$tempo, 20)
```

```
##  [1] 127.51 171.99 149.03 130.16 193.36 149.79  45.43 135.93 136.82 112.6
## [11] 107.45 189.82 158.02 115.09 208.17 114.93 145.88 135.83 109.10 179.
```

Exercise

- Create a new variable tempDec which rounds tempo to the nearest 10 bpm. Hint: use round()
- Create a Subset of songData to identify songs where this rounded bpm/“tempo” is 180 or 90.
- Identify the top 10 terms by observation count. Pick three terms for your playlist.
- Subset the dataset again to only include the terms you chose and songs . We will randomly choose 10 songs from this term from our playlist.

Solution

```
songData$tempoDec <- round(songData$tempo, -1)
songTemp <- songData[songData$tempoDec == 180 | songData$tempoDec == 90

songTable <- xtabs(~songData$term)
head(songTable[order(songTable, decreasing = TRUE)], 11)
```

```
## songData$term
##  other   rock    pop  metal   jazz  blues hip hop  house    rap
##   3423   1619    777    485    461    385    364    304    283
##   punk chanson
##    213     208
```

Solution

```
# hip hop, rock and house

songTemp <- songTemp[songTemp$term == "rock" | songTemp$term == "house"
    "hip hop", ]

songTemp[sample(nrow(songTemp), 10), c("artist.name", "title", "release
```

```
##           artist.name                 title
## 9719   Les Sexareenos                Ruby D.
## 2331 Jedi Mind Tricks        Brute Force II
## 7703   Rachel Portman                Bridget
## 4344       Sly Dunbar Casava Piece Riddim
## 1815          Hatiras           Final Flash
...
```

Aggregating Data

R comes with multiple functions built for aggregating data by applying a function (such as mean) over observations subset by one or multiple variables. Unlike other programming languages, this functionality in R doesn't require you to specify the dimensions or element names of your resulting object.

We use the aggregate function here to return all columns averaged across artist names.

Aggregating Data

```
subset(songDedup, subset = artist.name == "Aerosmith")
```

```
##    artist.name artist.latitude artist.longitude artist.location
## 389   Aerosmith              NA               NA      Boston, MA
##    artist.hotttnesss artist.familiarity tempo duration song.hotttnesss
## 389            0.6107           0.8725 108.7      425         0.5304
##      term hfRatio
## 389 rock  0.6999
```

```
# Our goal, benchmark
```

Aggregating Data

```
artistAgg <- aggregate(. ~ artist.name, data = songSubset3, mean)

subset(artistAgg, subset = artist.name == "Aerosmith")
```

```
##  [1] artist.name        artist.latitude    artist.longitude
##  [4] artist.location    artist.hotttnesss  artist.familiarity
##  [7] tempo              duration           song.hotttnesss
## [10] term               hfRatio            song.hotttnesssCut
## <0 rows> (or 0-length row.names)
```

```
# What happened to Aerosmith?
```

Aggregating Data

Problem: mean() defaults to NA if *any* observations meaned are NA and aggregate(), by default, removes any rows with NA/NaN. Aerosmith disappears.

Solution: na.rm=TRUE is an argument for mean(), it allows us to calculate the mean of each var, excluding any NA values

Note: mean of variables where all values are missing will still default to NaN. aggregate() argument na.action="na.pass" leaves in NA containing rows

Aggregating Data

```
artistAgg2 <- aggregate(. ~ artist.name, data = songSubset3, mean, na.a
    na.rm = TRUE)

subset(artistAgg2, subset = artist.name == "Aerosmith")
```

```
##   artist.name artist.latitude artist.longitude artist.location
## 76   Aerosmith             NaN              NaN             109
##   artist.hotttnesss artist.familiarity tempo duration song.hotttnesss
## 76          0.6108           0.8725 121.1    312.3        0.5304
##    term hfRatio song.hotttnesssCut
## 76   25     0.7                2.5
```

Aggregating Data

Like xtabs(): we can use aggregate() to obtain contingency tables

The number of observations at each value of term. songSubset3 contains 47 songs labeled ballad.

```
aggregate(artist.name ~ term, data = songSubset3, length)
```

We can use any variable in our dataset in place of artist.name here. The key is that aggregate returns the length of artist.name (the count of the number of observations of artist.name) at each value of term.

```
##         term artist.name
## 1    ballad          47
## 2     blues         385
## 3   chanson         208
## 4 chill-out          90
## 5   country          75
## 6 dancehall          97
```

Extras

Apply functions

There exist many other functions like aggregate. Aggregate works on a variety of R objects and returns a data frame. Other functions are object specific in both their input and output.

- apply()
- lapply()
- sapply()
- tapply()
- mapply()
- by()

Extras

The plyr package provides similar functionality.

Resource for
apply functions & plyr: http://stackoverflow.com/questions/3505701/r-grouping-functions-sapply-vs-lapply-vs-apply-vs-tapply-vs-by-vs-7141669#7141669

Extras

Using the function apply() to apply the custom normalizing function we wrote earlier to each column of a songData subset:

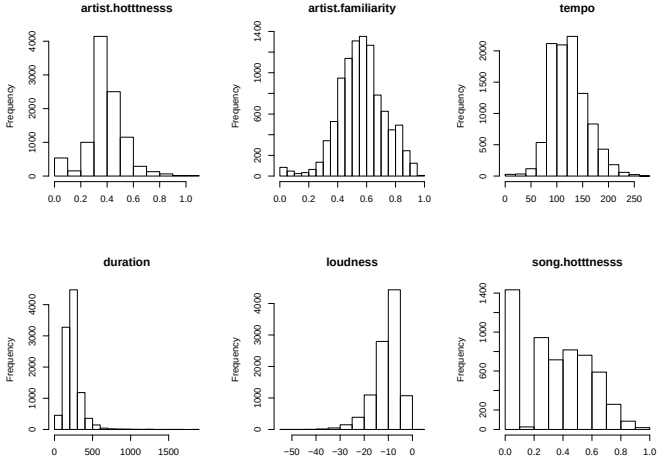
```
normfunct <- function(x = songData$song.hotttnesss) {
    (x - mean(x, na.rm = TRUE))/sd(x, na.rm = TRUE)
}

colNameSub <- c("artist.hotttnesss", "artist.familiarity", "tempo", "du
    "loudness", "song.hotttnesss")

x <- apply(songData[, colNameSub], 2, normfunct)
```

Extras

We can create a histogram plot for each normalized variable by using the function sapply() to apply the hist() function to each column of x.

```
par(mfrow = c(2, 3))
sapply(names(songData[, colNameSub]), function(x) hist(songData[, x], ma
    xlab = ""))
```

Extras




```
##                artist.hotttnesss artist.familiarity tempo
## breaks    Numeric,12                Numeric,21          Numeric,15
## counts    Integer,11                Integer,20          Integer,14
```

Bracketing Review

Poll: Which of the following can we use for subsetting or extracting elements of a data frame? Select all that apply.

- []
- $
- ()
- [[]]
- {}
- .
- subset()

Bracketing Review

Poll: Which of the following can we use to aid in specifying function arguments? Select all that apply.

- []
- $
- ()
- [[]]
- {}
- .
- subset()

Merging Data

The artist location data is neither standardized nor complete. We do have latitude and longitude coordinates! The dataset revGeocodeDF.RData used ggmap2 package's revgeocode() function to obtain state information for these coordinates.

Recall we loaded this RData file which created an object called 'd' in our workspace.

```
load("data/revGeocodeDF.RData")
```

Exercise

Explore this data set. What information is common between this data set and songData? What additional information does this data set provide? How can we augment g with this information?

Solution

```
summary(d)
summary(songData)

head(d)
head(songData)

View(d)
View(songData)
```

Exercise

1. Use subset() on *songSubset3* to identify the observation where (artist.latitude, artist.longitude) is (34.23294, -102.41020). What is the artist.location?
2. Use subset() on *d* to identify the observation where (artist.latitude, artist.longitude) is (34.23294, -102.41020). What is the geoCountry and geoSate?

Solution

```
subset(songSubset3, artist.latitude == 34.23294 & artist.longitude == -1

##    artist.name artist.latitude artist.longitude artist.location
## 35          BT          34.23          -102.4          Earth
## 36          BT          34.23          -102.4          Earth
## 37          BT          34.23          -102.4          Earth
## 38          BT          34.23          -102.4          Earth
## 39 BT feat. JES          34.23          -102.4          Earth
...

subset(d, artist.latitude == 34.23294 & artist.longitude == -102.4102)

##      artist.longitude artist.latitude    geoCountry geoState
## 1343          -102.4           34.23 United States    Texas
```

Merging Data

We will merge the two based on their shared variables: artist.latitude and artist.longitude to obtain more complete and standardized location information than artist.location provides.

```
songSubset3 <- merge(d, songSubset3, by = c("artist.latitude", "artist.l

subset(songSubset3, artist.latitude == 34.23294 & artist.longitude == -1

##     artist.latitude artist.longitude   geoCountry geoState  artist.name
## 1116        34.23        -102.4 United States   Texas        BT
## 1117        34.23        -102.4 United States   Texas        BT
## 1118        34.23        -102.4 United States   Texas        BT
## 1119        34.23        -102.4 United States   Texas BT feat. JES
## 1120        34.23        -102.4 United States   Texas        BT
```

Save Your Work

```
save(songSubset3, file = "data/Module4_songSubset3.RData")
```

REVOLUTION ANALYTICS

Thank you

Revolution Analytics is the leading commercial provider of software and support for the popular open source R statistics language.

www.revolutionanalytics.com, 1.855.GET.REVO, Twitter: @RevolutionR