Data Visualization with GGPLOT2 (Part 3)

Joseph Walker

October 3, 2017

Part 3

In **part 1** of this series, we explored the fundamentals of `ggplot2`. We learned about the **grammar of graphics** beginning with **data, aesthetics, and geometries.**

In **part 2**, we extended our understanding of data visaluzation by learning about additional graphical elements including: **statistics, coordinates, facets, and themes.** We even learned some best practices along the way.

In this final chapter, we will explore plots intended for a specialty audience. We will also learn about plots for specific data types such as **ternary plots, networks and maps.**

Statistical Plots for an Academic Audience

There are two common types of plots presented to an academic audience: Box plots and Density plots.

Box Plots

Box plots were first described by John Tukey in 1977 in his classic text, 'Exploratory Data Analysis'. The Box Plot gives us what Tukey describes as the 5 number summary:

- minimum
- 1st quartile
- 2nd quartile (the median)
- 3rd quartile
- maximum

This is advantageous over using the mean and standard deviation for data sets that may not be normally distributed and prone to extreme outliers.

The inner quartile range is the difference between the 3rd and 1st quartiles, or what we commonly see as the box in a box plot.

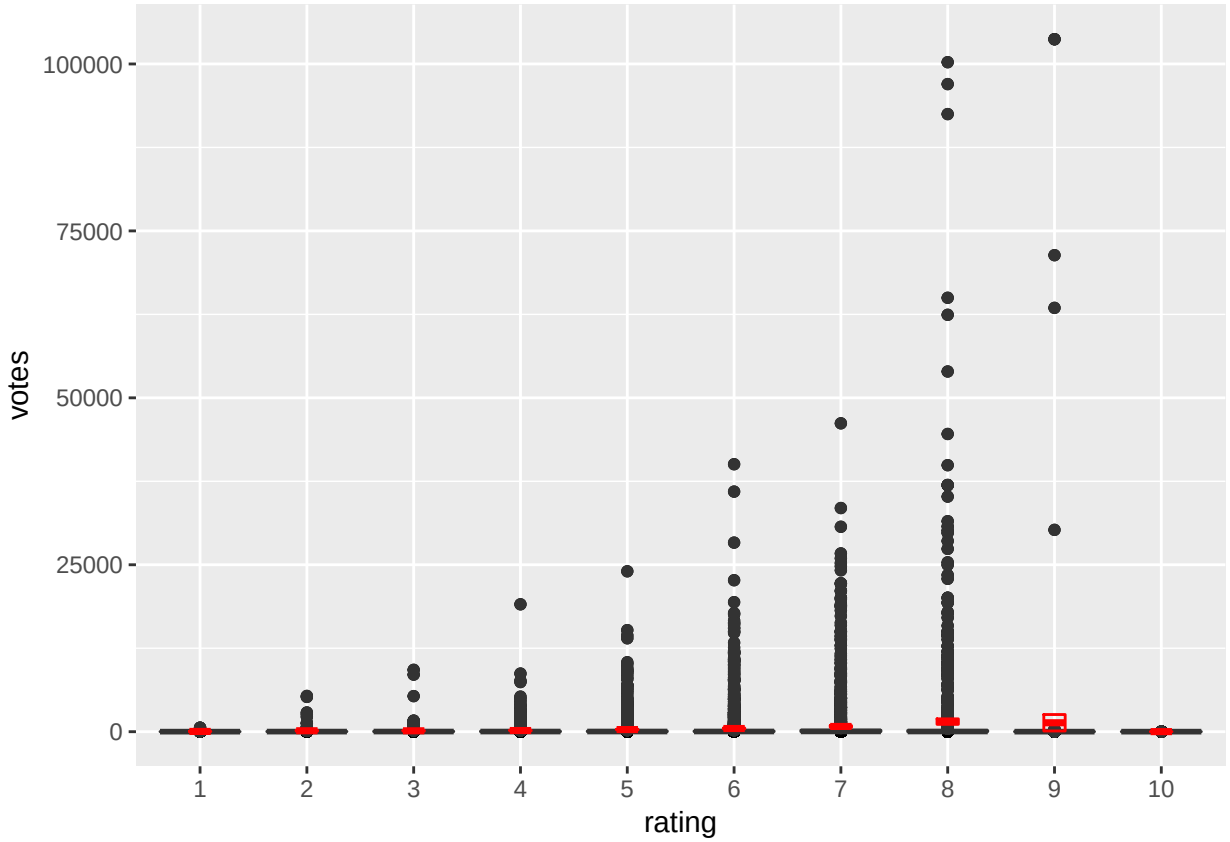
The following examples use the `movies` dataset from the `ggplot2movies` package.

```
str(movies)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    58788 obs. of  24 variables:
##  $ title      : chr  "$" "$1000 a Touchdown" "$21 a Day Once a Month" "$40,000" ...
##  $ year       : int  1971 1939 1941 1996 1975 2000 2002 2002 1987 1917 ...
##  $ length     : int  121 71 7 70 71 91 93 25 97 61 ...
##  $ budget     : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ rating     : num  6.4 6 8.2 8.2 3.4 4.3 5.3 6.7 6.6 6 ...
##  $ votes      : int  348 20 5 6 17 45 200 24 18 51 ...
##  $ r1         : num  4.5 0 0 14.5 24.5 4.5 4.5 4.5 4.5 4.5 ...
##  $ r2         : num  4.5 14.5 0 0 4.5 4.5 0 4.5 4.5 0 ...
##  $ r3         : num  4.5 4.5 0 0 0 4.5 4.5 4.5 4.5 4.5 ...
##  $ r4         : num  4.5 24.5 0 0 14.5 14.5 4.5 4.5 0 4.5 ...
```

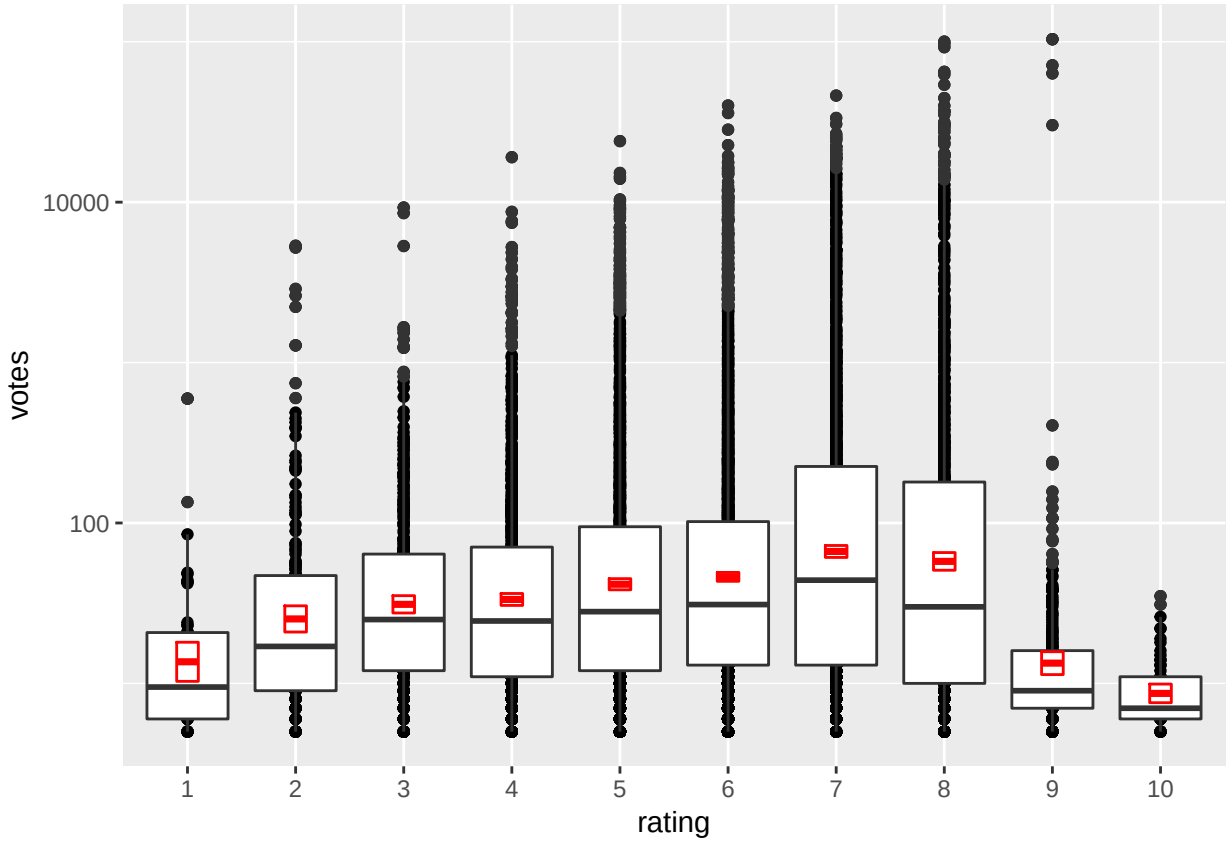
```
##  $ r5          : num  14.5 14.5 0 0 14.5 14.5 24.5 4.5 0 4.5 ...
##  $ r6          : num  24.5 14.5 24.5 0 4.5 14.5 24.5 14.5 0 44.5 ...
##  $ r7          : num  24.5 14.5 0 0 0 4.5 14.5 14.5 34.5 14.5 ...
##  $ r8          : num  14.5 4.5 44.5 0 0 4.5 4.5 14.5 14.5 4.5 ...
##  $ r9          : num  4.5 4.5 24.5 34.5 0 14.5 4.5 4.5 4.5 4.5 ...
##  $ r10         : num  4.5 14.5 24.5 45.5 24.5 14.5 14.5 14.5 24.5 4.5 ...
##  $ mpaa        : chr  "" "" "" "" ...
##  $ Action      : int  0 0 0 0 0 0 1 0 0 0 ...
##  $ Animation   : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ Comedy      : int  1 1 0 1 0 0 0 0 0 0 ...
##  $ Drama       : int  1 0 0 0 0 1 1 0 1 0 ...
##  $ Documentary: int  0 0 0 0 0 0 0 1 0 0 ...
##  $ Romance     : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Short       : int  0 0 1 0 0 0 0 1 0 0 ...
```

```r
#gather the movie genre into one column
movies <- movies %>% gather(key = 'genre', value = 'value', -c(1:17))

#convert mpaa to factor
movies$mpaa <- as_factor(movies$mpaa)

#relabel movies not rated with 'N/A'
movies$mpaa <- fct_recode(movies$mpaa, 'N/A' = "")

#set seed for reprducibility
set.seed(123)

#sample the movies dataset
movie_sample <- movies[sample(nrow(movies), 10001), ]

#factor the ratings variable and round to the nearest whole number
movie_sample$rating <- factor(round(movie_sample$rating))

#create a boxplot object
p <- ggplot(movie_sample, aes(x = rating, y = votes, group = rating)) +
  geom_point() +
  geom_boxplot() +
  stat_summary(fun.data = "mean_cl_normal",
               geom = "crossbar",
               width = 0.2,
               col = "red")

#boxplot
p
```

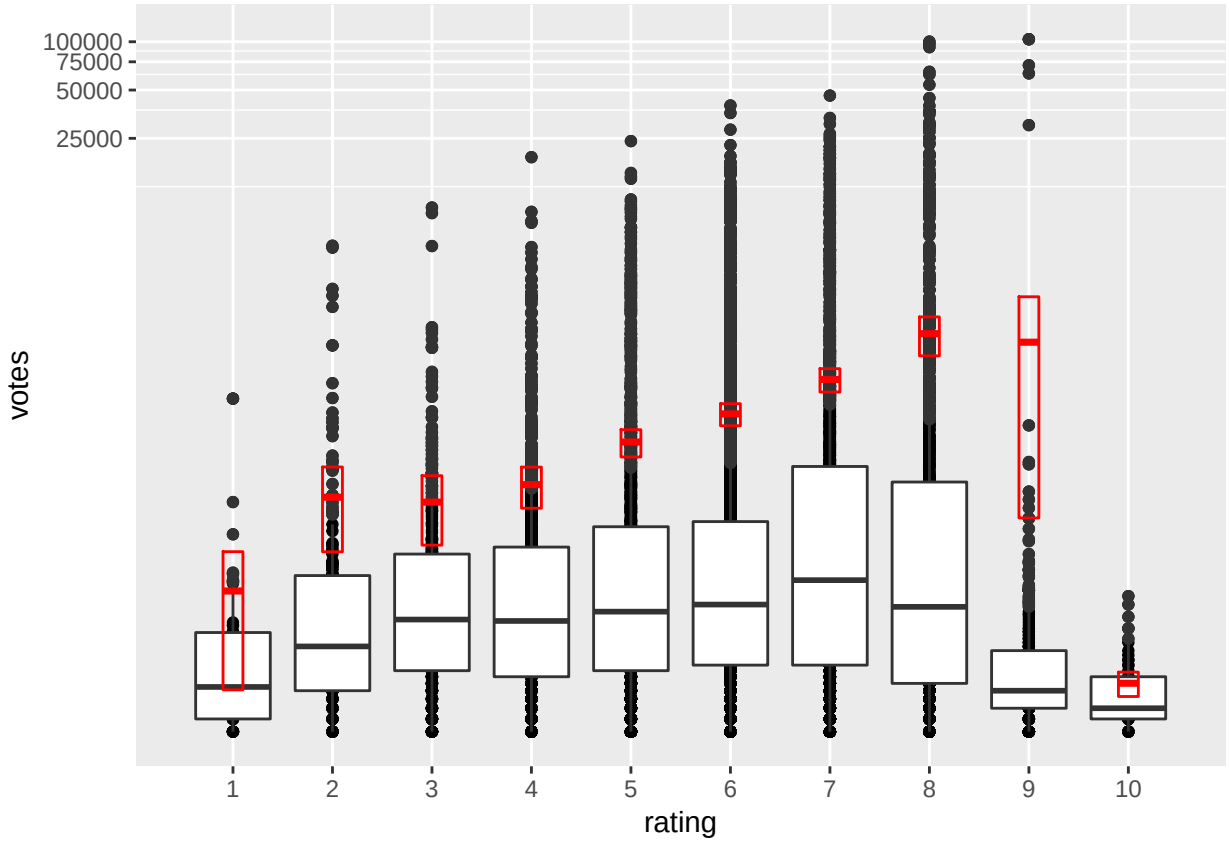


There is a large number of votes for rating. We will need to make some transformations on the data. Be careful as the tranformation will occur differently depending on how you call your stat functions and arguments.

```
#transformation happens before statistics are calculated
p + scale_y_log10()
```



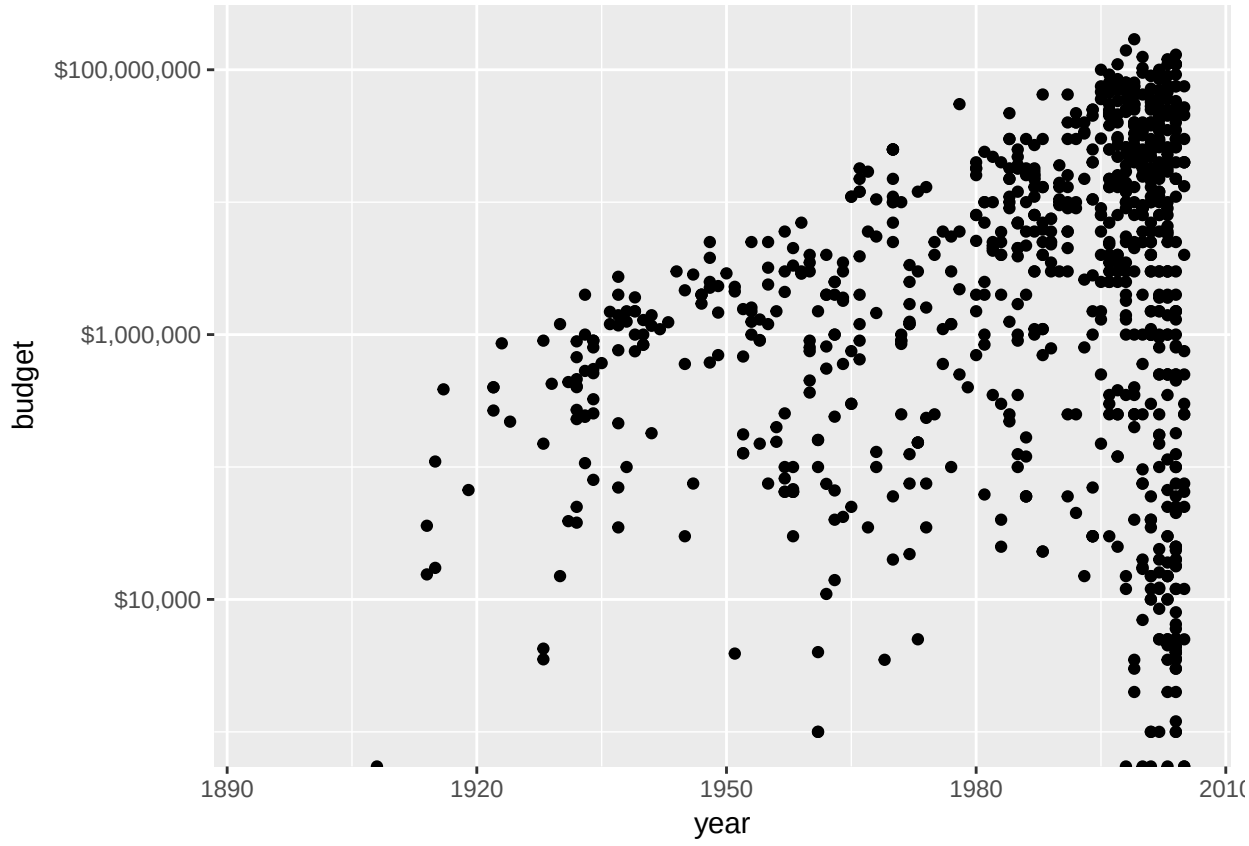

```
#transformation happens after the statistics are calculated
p + coord_trans(y = "log10")
```



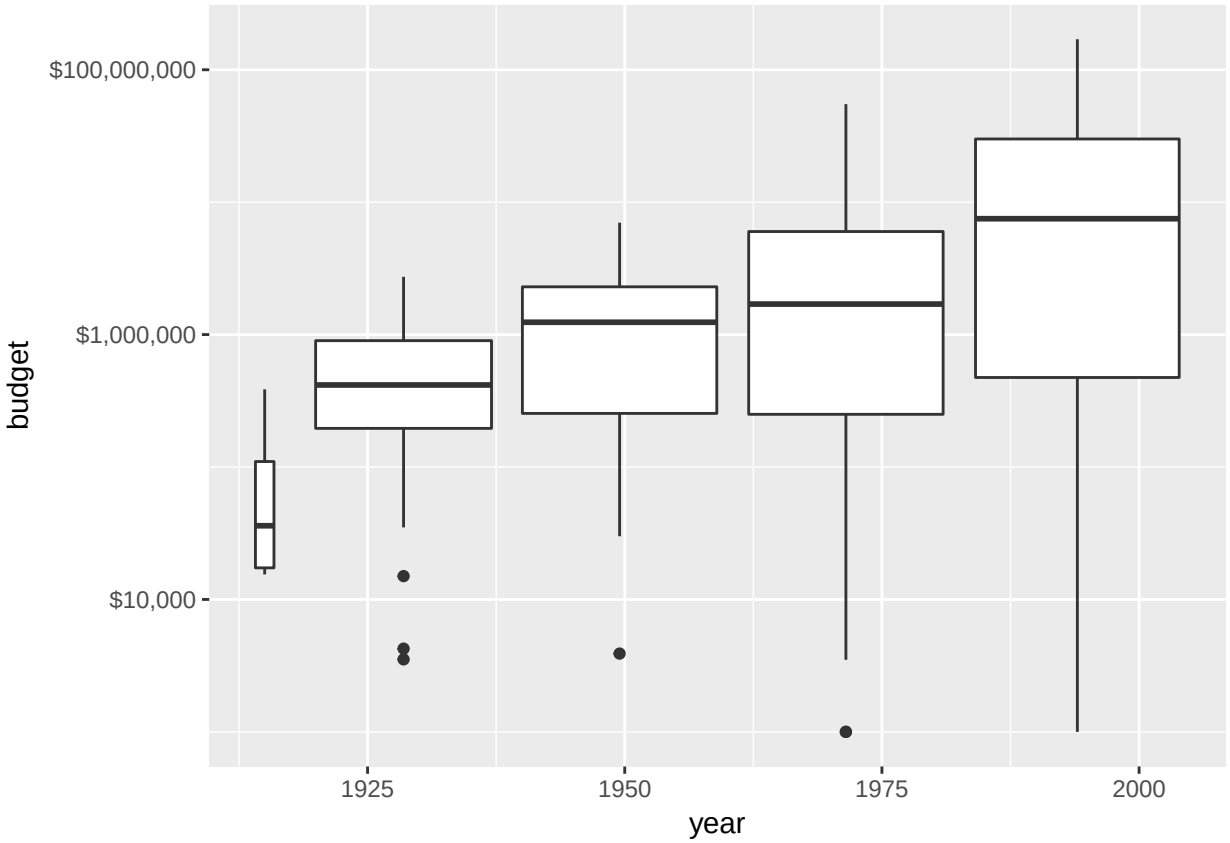
It is possible to cut up continuous variables into ordinal variables using the following functions which `cut` the data.

`cut_interval(x, n)` makes n groups from vector x with equal range.
`cut_number(x, n)` makes n groups from vector x with (approximately) equal numbers of observations.
`cut_width(x, width)` makes groups of width width from vector x.

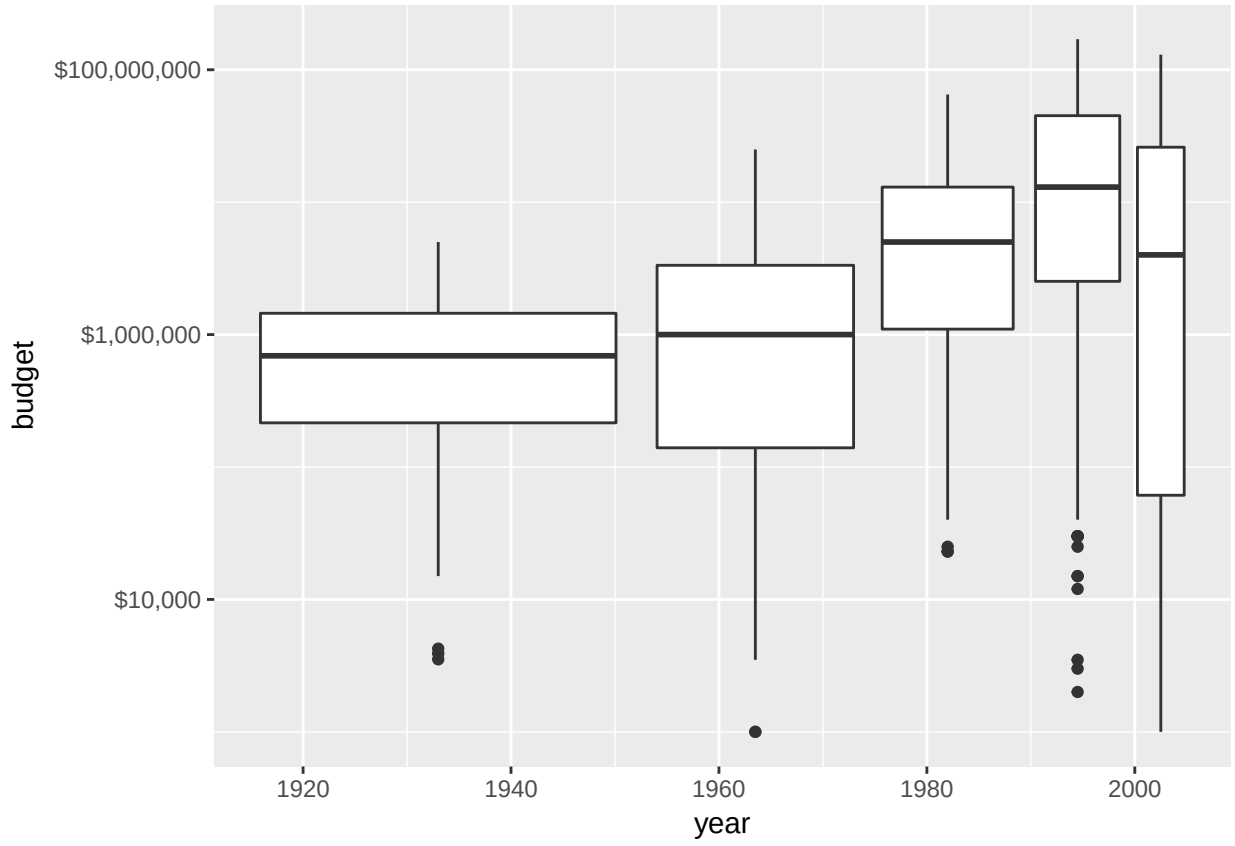
```
#create plot object from movies sample dataset
p <- ggplot(movie_sample, aes(x = year, y = budget)) + scale_y_log10(labels = scales::dollar)

#examine the plot
p + geom_point()
```



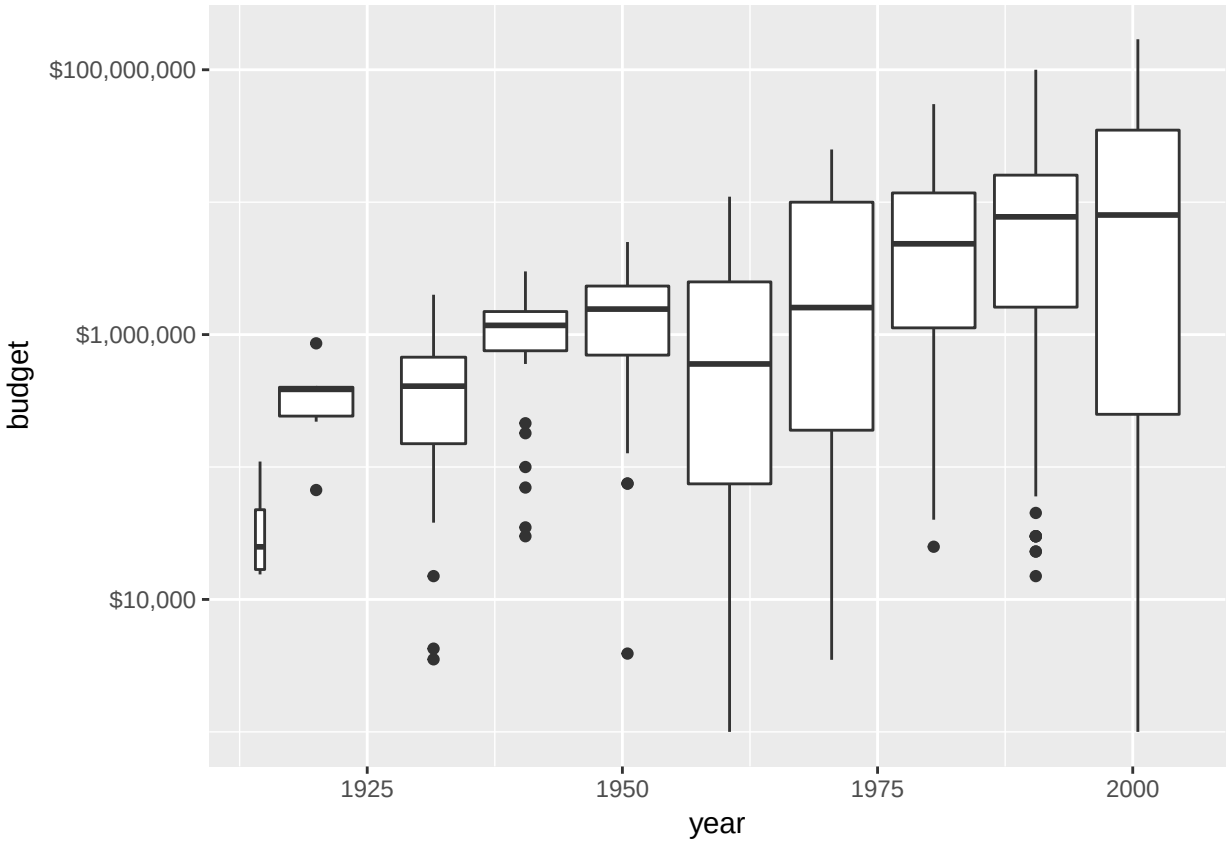

```
#use cut_interval to divide year into  sections with equal range
#every 25 years or so
p + geom_boxplot(aes(group = cut_interval(year, n = 5)))
```




```
#use cut_number function to divide year into groups with approximately equal number of observations
#reveals that we do not have budget records for the early part of the 20th century
p + geom_boxplot(aes(group = cut_number(year, n = 5)))
```

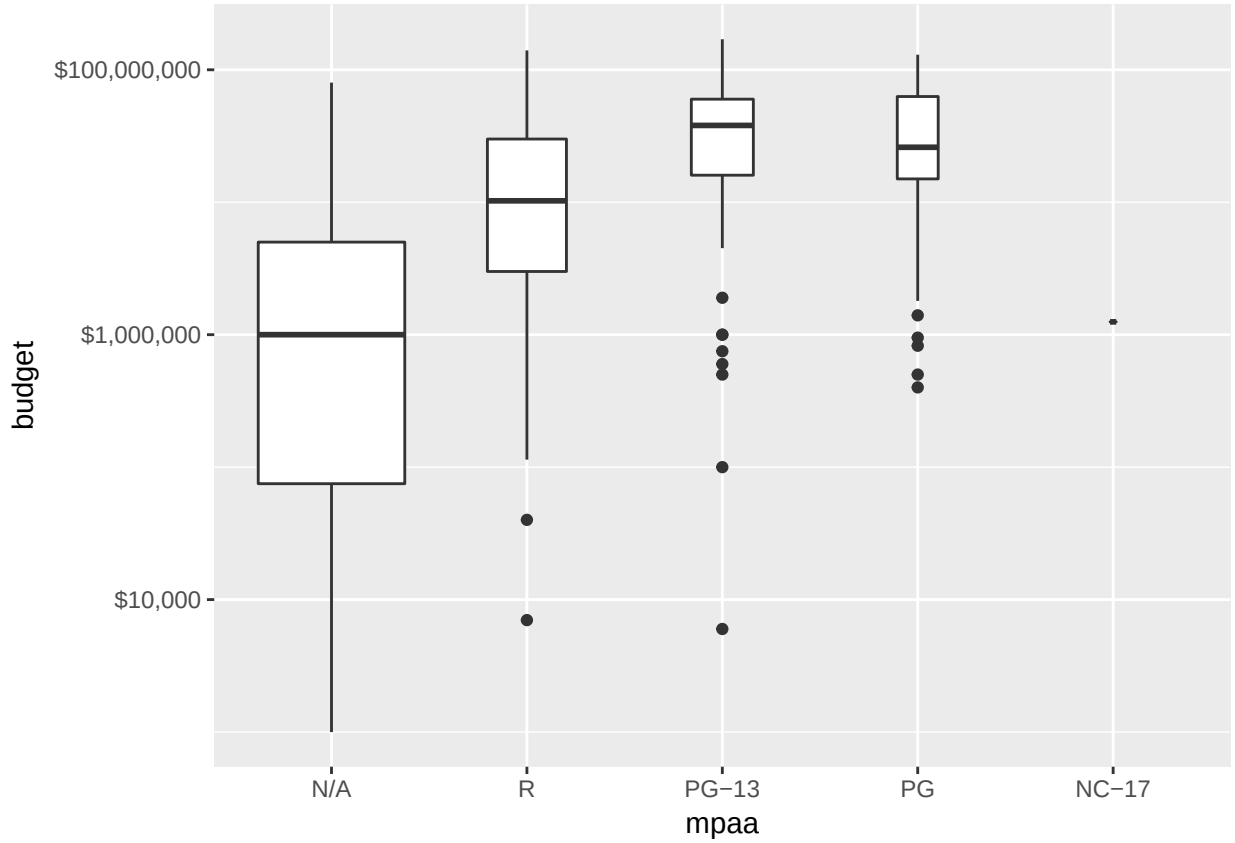



```
#use cut_width to make group of specified width
#i.e. decades
p + geom_boxplot(aes(group = cut_width(year, width = 10)))
```



One drawback of the box plot visualization is that you don't have any indication of the sample size for each group. One way of showing this variation is to use the `varwidth` argument.

```
#create plot object using movie_sample dataset
p <- ggplot(movie_sample, aes(x = mpaa, y= budget)) + scale_y_log10(labels = scales::dollar)

#view plot object with boxplot and adjust the widths of the boxes based on the sample size
p + geom_boxplot(varwidth = T)
```



And just so we can confirm this argument is doing what we expect it to, we can check the math manually.

```
movie_sample %>%
  group_by(mpaa) %>%
  summarize(count = n()) %>%
  knitr::kable(align = 'c')
```

mpaa	count
N/A	9107
R	594
PG-13	194
PG	103
NC-17	3

Density Plots

Theoretical density plots use the probability density function (PDF) to plot the distribution of univariate data. You have certainly seen these types of plots before. They include: normal, t, chi-squared, and F distributions.

Emprical density plots use real data using the Kernal Density Estimate.

The KDE is defined as:

> A sum of 'bumps' placed at the observations. The kernel function determines the shape of the bumps while the window width, h, determines their width.

The KDE calculates a normal distribution for each value in the data. These are known as the bumps.

To obtain the true density curve, we simply add up all the y-values for each bump along our x-axis.

There are three parameters that you may be tempted to adjust in a density plot:

bw - the smoothing bandwidth to be used, see ?density for details adjust - adjustment of the bandwidth, see density for details kernel - kernel used for density estimation, defined as “g” = gaussian “r” = rectangular “t” = triangular “e” = epanechnikov “b” = biweight “c” = cosine “o” = optcosine

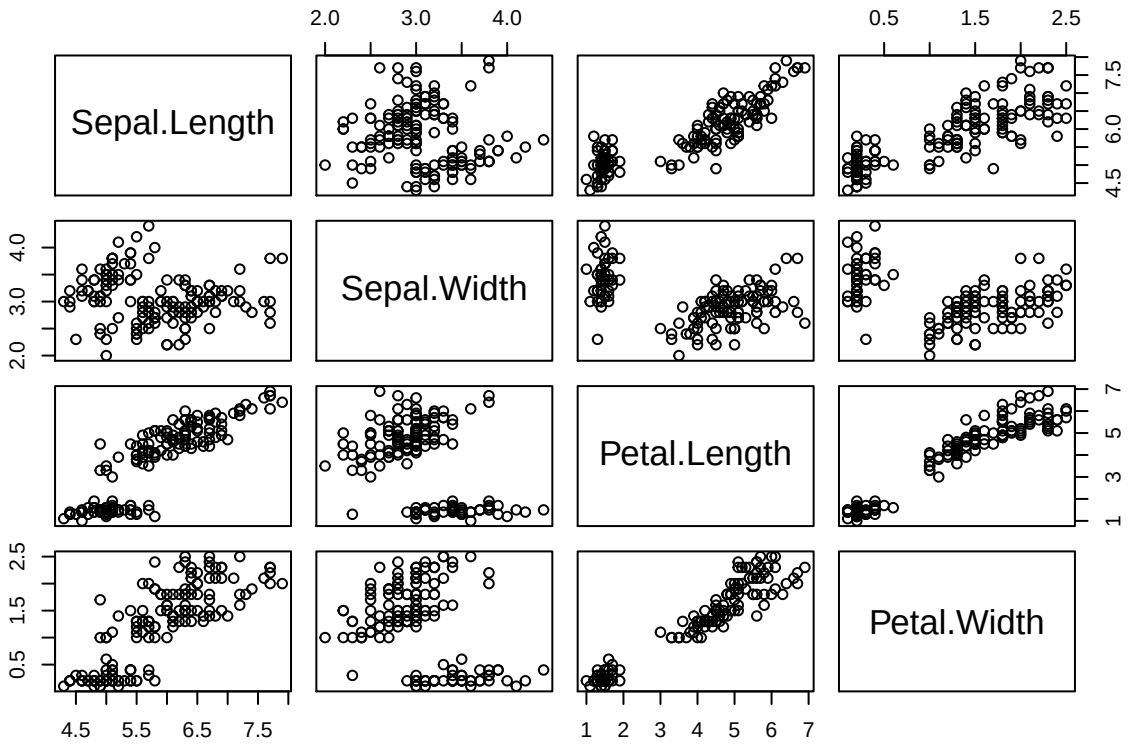
Plots for Specific Data Types

Scatter Plot Matrices

How do you define largenesss of a data set? Many observations? Many variables? A combination of both?

Base R provides a quick and dirty function, `pairs()` that will output a scatterplot matrix, or SPLOM. This function will only work for continuous variables.

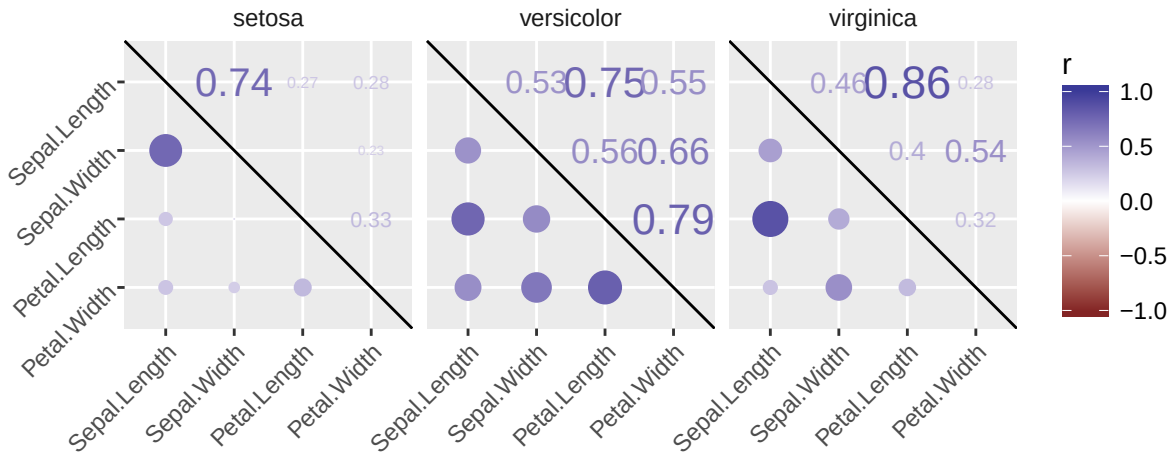
```
pairs(iris[1:4])
```



This can also be done in ggplot2

```
cor_list <- function(x) {
  L <- M <- cor(x)

  M[lower.tri(M, diag = TRUE)] <- NA
  M <- melt(M)
```

```
  names(M)[3] <- "points"

  L[upper.tri(L, diag = TRUE)] <- NA
  L <- melt(L)
  names(L)[3] <- "labels"

  merge(M, L)
}

# Calculate xx with cor_list
xx <- iris %>%
  group_by(Species) %>%
  do(cor_list(.[1:4]))

# Finish the plot
ggplot(xx, aes(x = Var1, y = Var2)) +
  geom_point(aes(col = points, size = abs(points)), shape = 16) +
  geom_text(aes(col = labels,  size = abs(labels), label = round(labels, 2))) +
  scale_size(range = c(0, 6)) +
  scale_color_gradient2("r", limits = c(-1, 1)) +
  scale_y_discrete("", limits = rev(levels(xx$Var1))) +
  scale_x_discrete("") +
  guides(size = FALSE) +
  geom_abline(slope = -1, intercept = nlevels(xx$Var1) + 1) +
  coord_fixed() +
  facet_grid(. ~ Species) +
  theme(axis.text.y = element_text(angle = 45, hjust = 1),
        axis.text.x = element_text(angle = 45, hjust = 1),
        strip.background = element_blank())
```



Ternary Plot

A **Ternary plot**, also known as a triangle plot, can be used for compositional trivariate data. This means that the variables add up to 100%.

You could visualize this type of data with a stacked bar plot, but the ternary plot is better suited for comparing the different compositions.

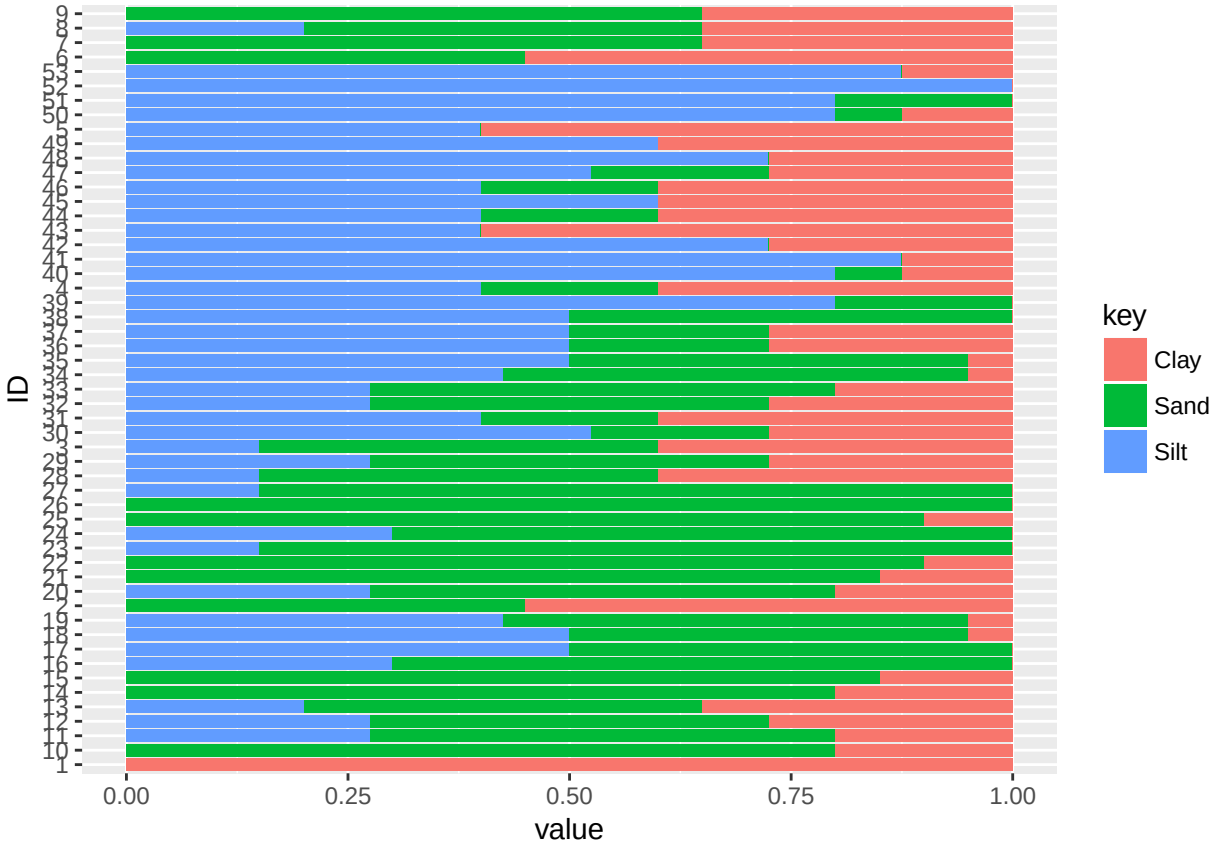
The following example uses the `USDA` data set, providing textural classification data of land samples.

```
#load dataset from the ggtern package
data("USDA")

#examine the USDA data set
str(USDA)
```

```
## 'data.frame':    53 obs. of  4 variables:
##  $ Clay : num  1 0.55 0.4 0.4 0.6 0.55 0.35 0.35 0.35 0.2 ...
##  $ Sand : num  0 0.45 0.45 0.2 0 0.45 0.65 0.45 0.65 0.8 ...
##  $ Silt : num  0 0 0.15 0.4 0.4 0 0 0.2 0 0 ...
##  $ Label: Factor w/ 12 levels "Clay","Sandy Clay",..: 1 1 1 1 1 2 2 2 3 3 ...
```

```
# create ID column
USDA$ID <- row.names(USDA)

#tidy USDA to long format
USDA_long <- gather(USDA, key, value, -c(Label, ID))
```

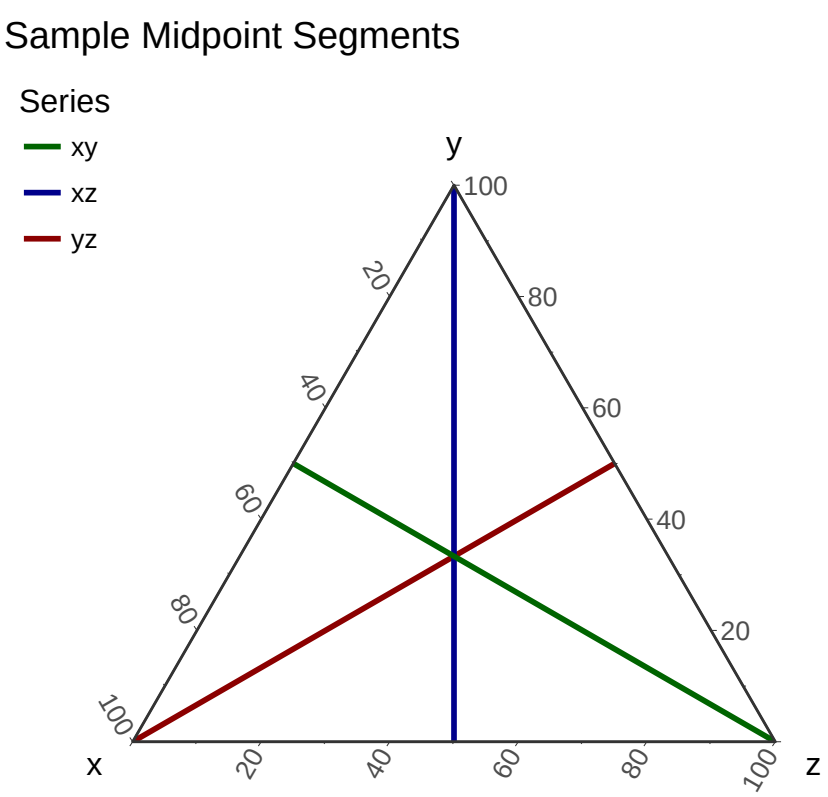
```r
#create stacked bar plot to depict data
ggplot(USDA_long, aes(x = ID, y = value, fill = key))+
  geom_col() +
  coord_flip()
```



This is a an acceptable plot to represent the composition of soil for each sample. But suppose we wanted to use a ternary plot to visaulize this.

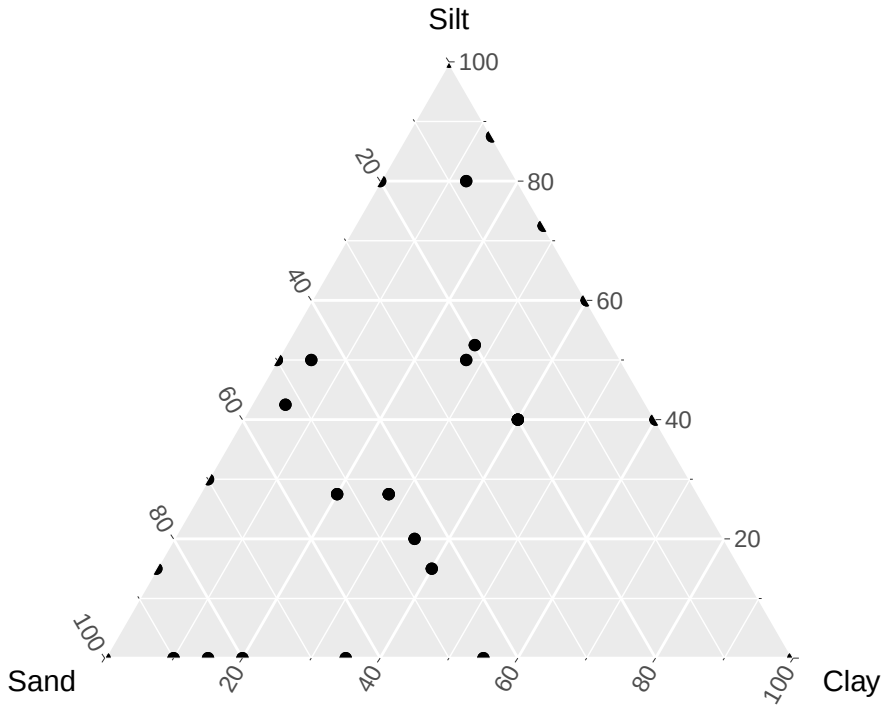
The ternary plot works like this:

```r
DATA <- data.frame(x = c(1,0,0),
                   y = c(0,1,0),
                   z = c(0,0,1),
                   xend = c(0,.5,.5),
                   yend = c(.5,0,.5),
                   zend = c(.5,.5,0),
                   Series = c("yz","xz","xy"))
ggtern(data=DATA,aes(x,y,z,xend=xend,yend=yend,zend=zend)) +
  geom_segment(aes(color=Series),size=1) +
  scale_color_manual(values=c("darkgreen","darkblue","darkred")) +
  theme_bw() + theme_nogrid() +
  theme(legend.position=c(0,1),legend.justification=c(0,1)) +
  labs(title = "Sample Midpoint Segments")
```



And with the USDA soil dataset,

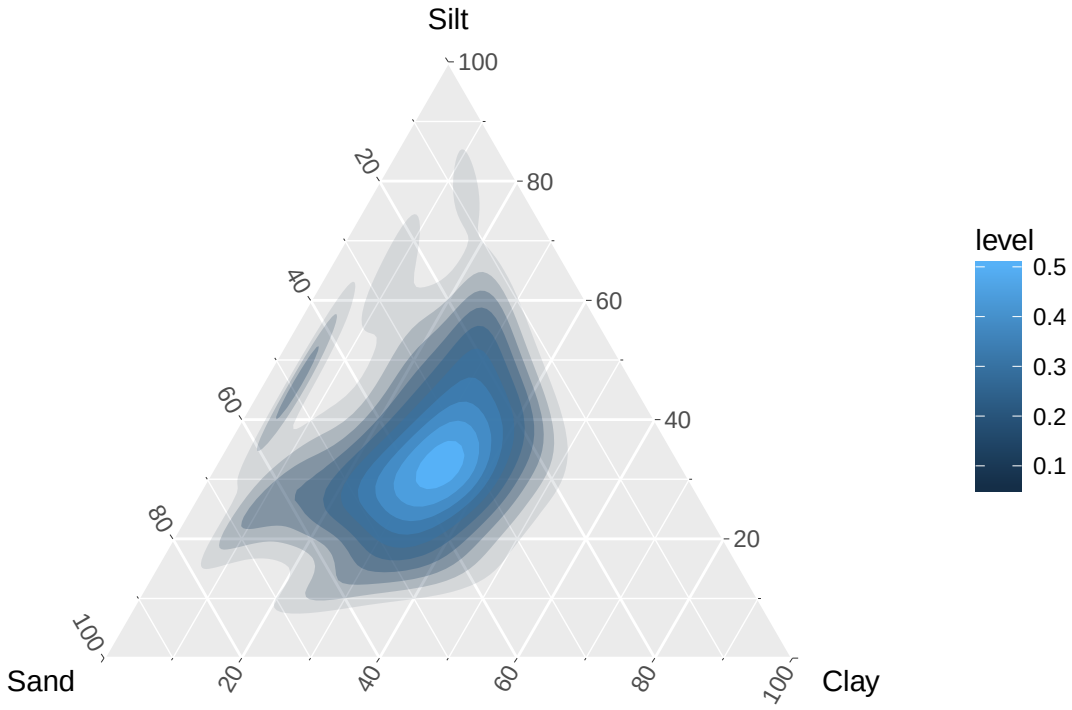
```
#create ternary diagram plot object
p <- ggtern(USDA, aes(x = Sand, y = Silt, z = Clay))

#plot points on ternary plot
p + geom_point()
```



The `ggtern` package is compatible with `ggplot2` functionality to create ternary plots.

As we saw in previous examples, we can combine 2d density plots and other types of visualizations with the ternary plot.

```
p +
  stat_density_tern(geom = 'polygon', aes(fill = ..level.., alpha = ..level..))+
  guides(alpha = F)
```



Network Plot

Visualizing relationships between factors in a variable.

For this, we will need the `geomnet` package.

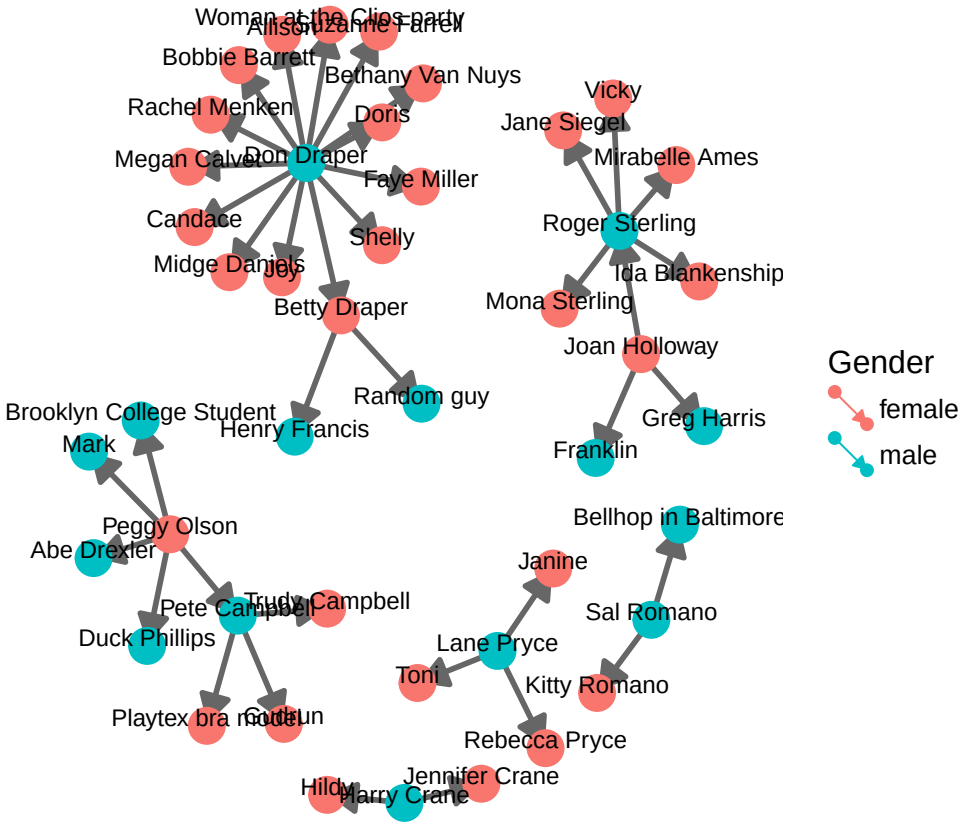
The following examples use the `madmen` dataset from the geomnet library.

```
str(madmen)
```

```
## List of 2
##  $ edges   :'data.frame':    39 obs. of  2 variables:
##   ..$ Name1: Factor w/ 9 levels "Betty Draper",..: 1 1 2 2 2 2 2 2 2 2 ...
##   ..$ Name2: Factor w/ 39 levels "Abe Drexler",..: 15 31 2 4 5 6 8 9 11 21 ...
##  $ vertices:'data.frame':    45 obs. of  2 variables:
##   ..$ label : Factor w/ 45 levels "Abe Drexler",..: 5 9 16 23 26 32 33 38 39 17 ...
##   ..$ Gender: Factor w/ 2 levels "female","male": 1 2 2 1 2 1 2 2 2 2 ...
```

```
#merge the edges and vertices datasets
mmnet <- full_join(madmen$edges, madmen$vertices, by = c('Name1' = 'label'))


#plot the relationship
ggplot(mmnet, aes(from_id = Name1, to_id = Name2, col = Gender )) +
  geom_net(labelon = T,
           size = 6,
           fontsize = 3,
```

```
          labelcolour = 'black',
          linewidth = 1,
          directed = T) +
  theme_void() +
  xlim(c(-.5, 1.05))
```



Diagnostic Plots

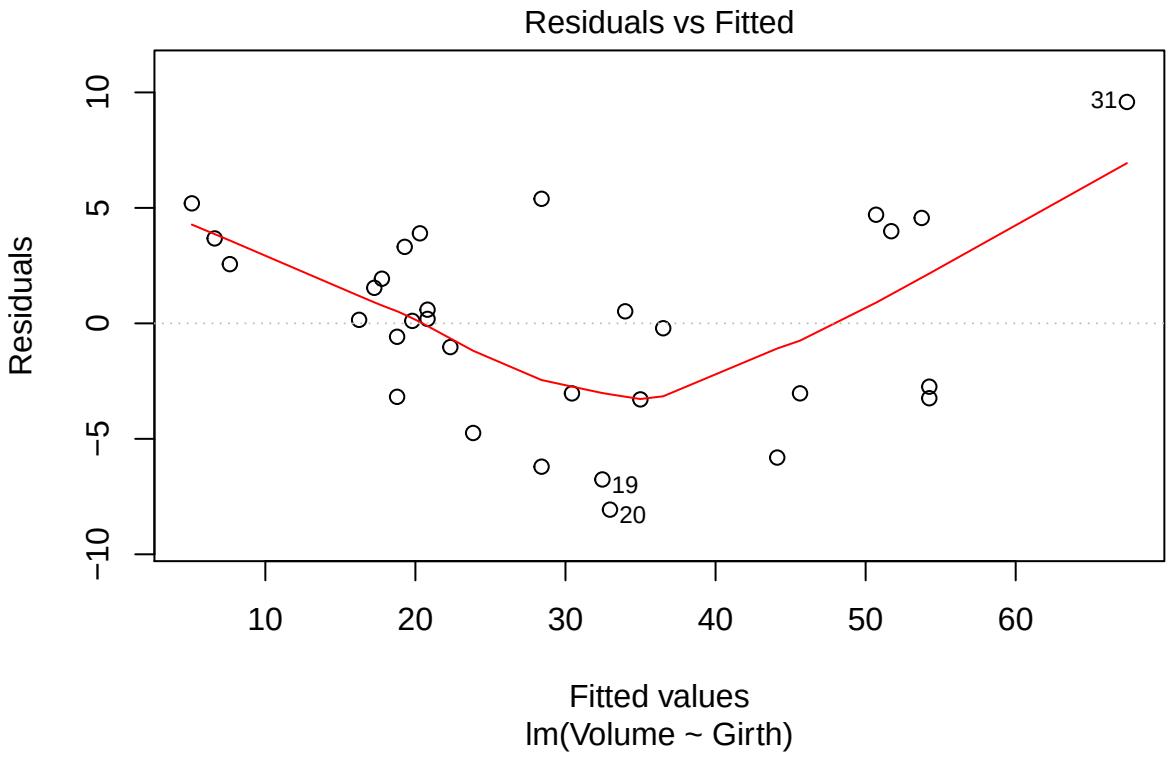
Can be used to assess how well the fit of a model is. The base R function `plot` will display 4 plots based on the model you use in your call to the function.

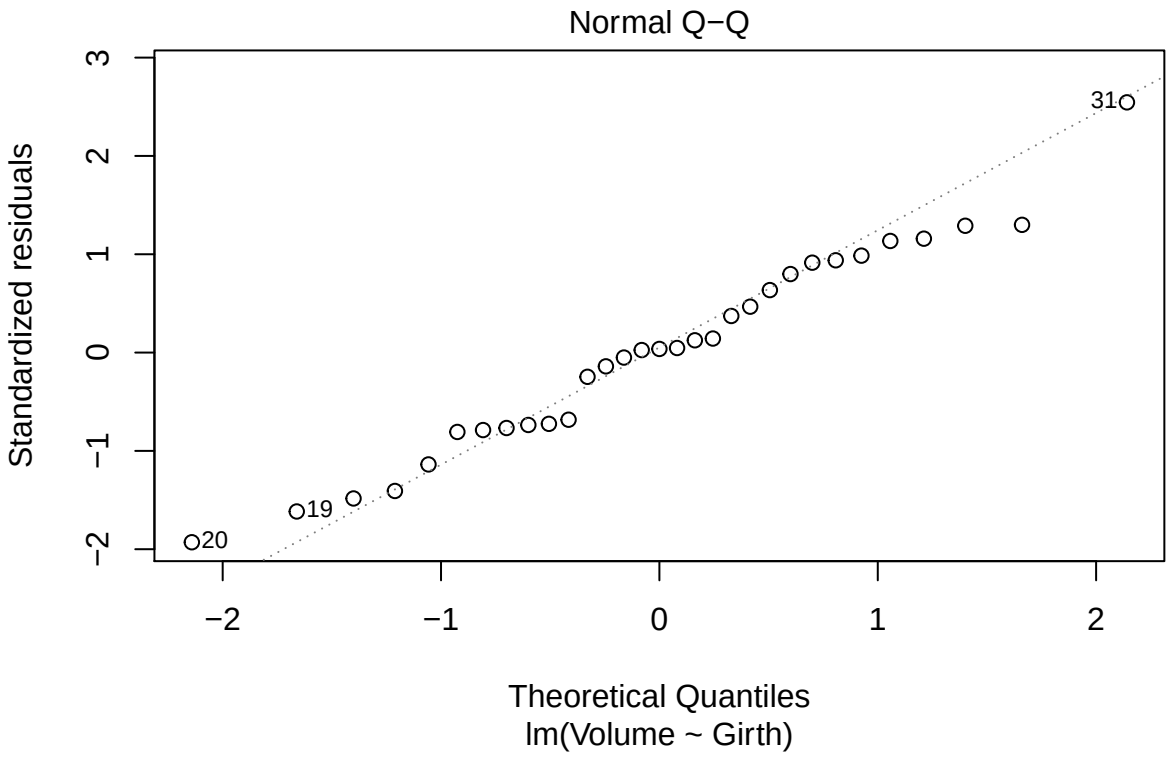
Here's an examples using the `trees` dataset from the `stats` package (default package loaded on startup).

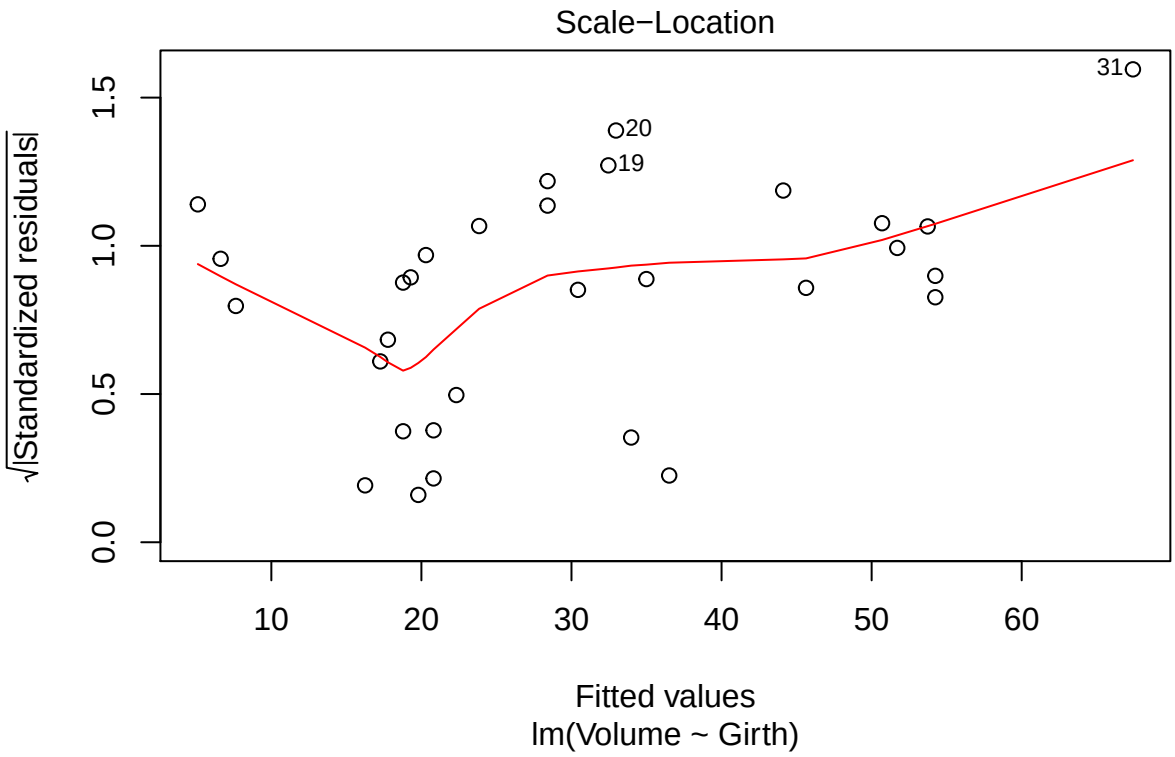
```
#examine the trees dataset
str(trees)
```

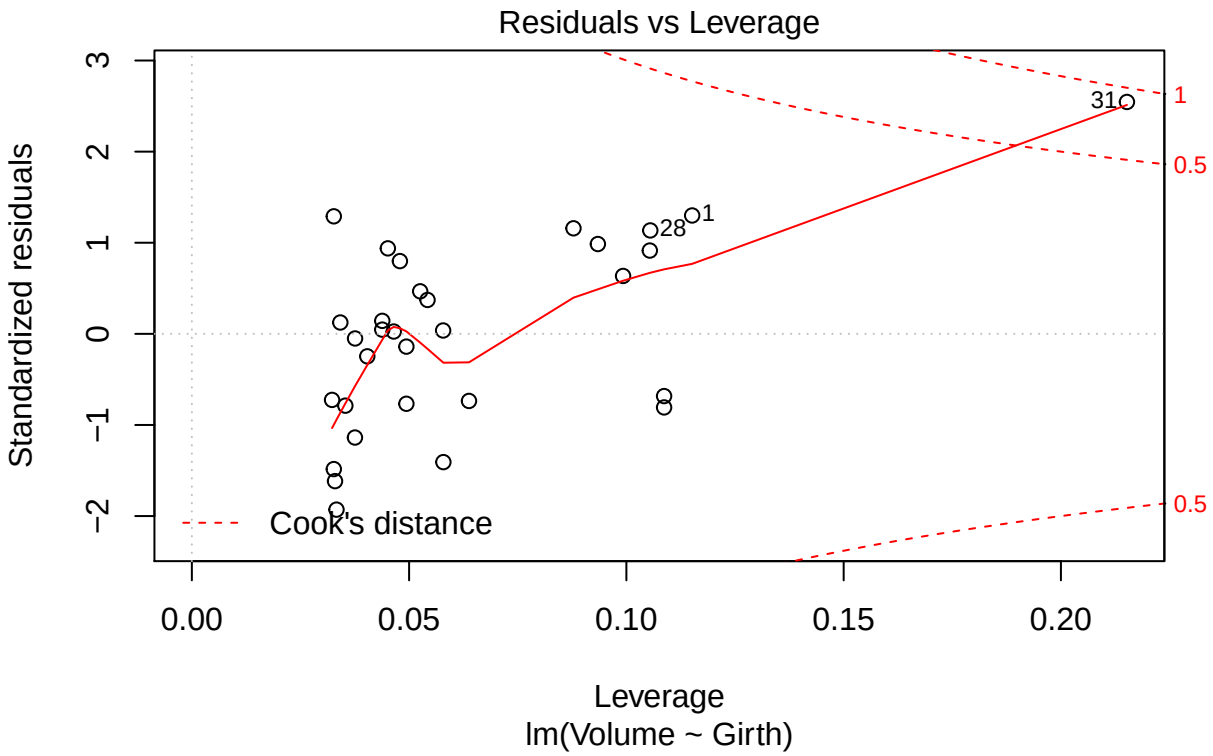
```
## 'data.frame':    31 obs. of  3 variables:
##  $ Girth : num  8.3 8.6 8.8 10.5 10.7 10.8 11 11 11.1 11.2 ...
##  $ Height: num  70 65 63 72 81 83 66 75 80 75 ...
##  $ Volume: num  10.3 10.3 10.2 16.4 18.8 19.7 15.6 18.2 22.6 19.9 ...
```

```
#create a linear model object
res <- lm(Volume ~ Girth, data = trees)

#display diagnostic plots of the model
plot(res)
```


Residuals vs Fitted
Residuals
10
5
0
−5
−10
10
20
30
40
50
60
31
19
20
Fitted values
lm(Volume ~ Girth)

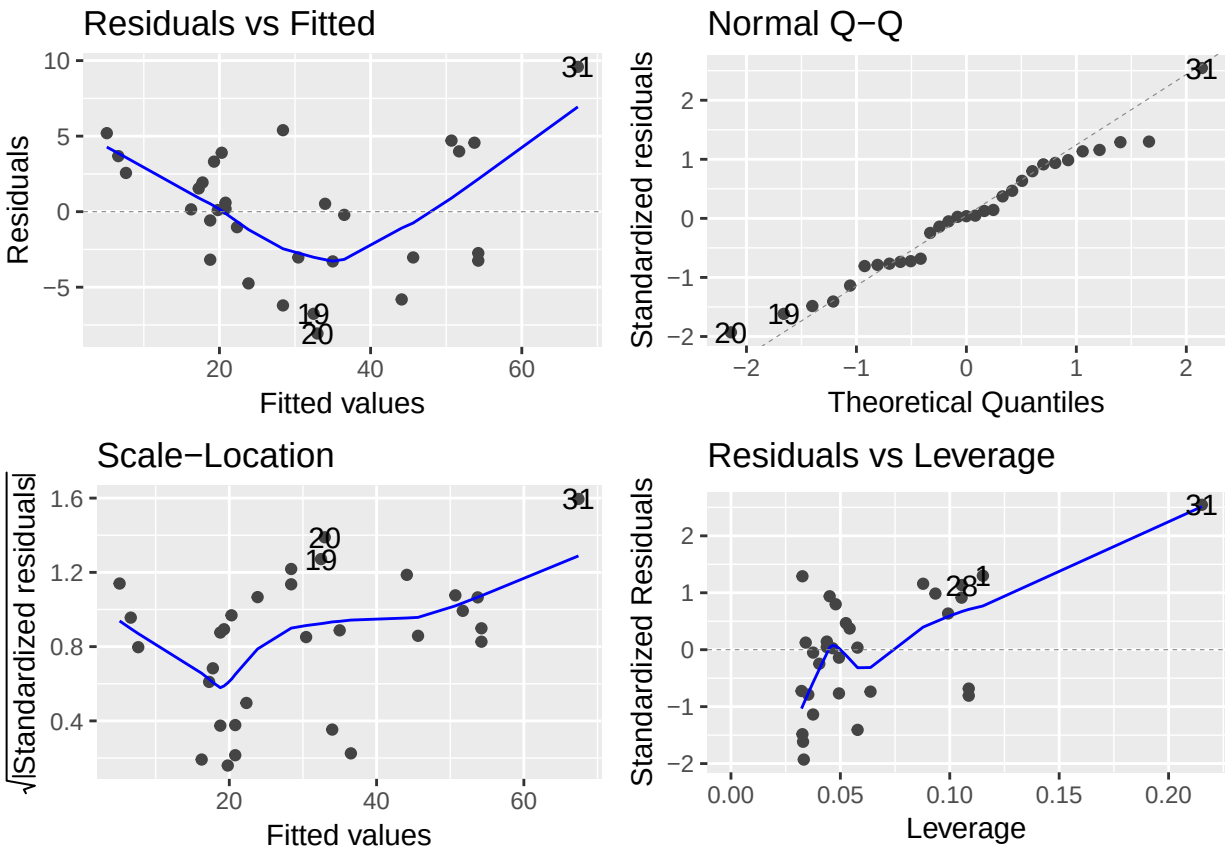

Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(Volume ~ Girth)
31
19
20
−2
−1
0
1
2
3


Scale−Location
√|Standardized residuals|
0.0
0.5
1.0
1.5
10
20
30
40
50
60
31
20
19
Fitted values
lm(Volume ~ Girth)



We can achieve similar results in ggplot2 using the `ggfortify` package. This package converts funcitons between the base R plot `graphics` and `ggplot2` using the `grid` graphics.

```
autoplot(res, ncol = 2)
```

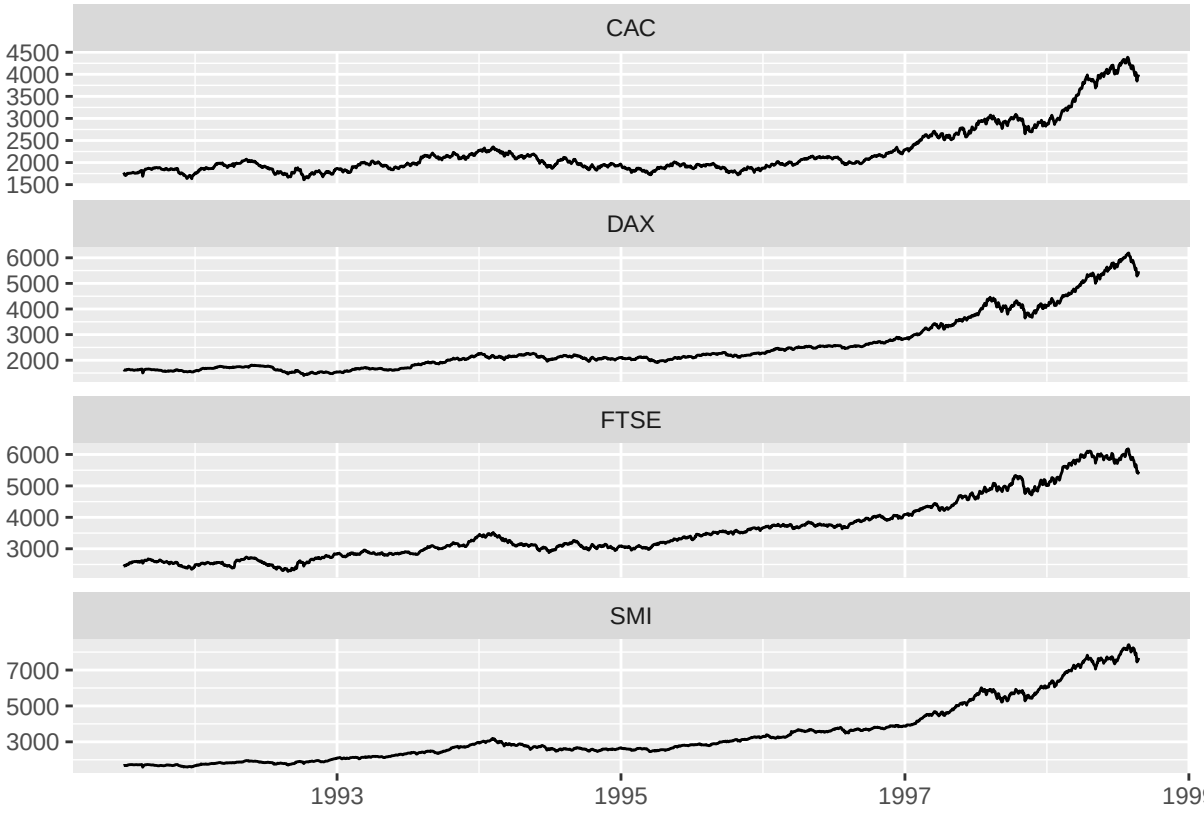


Autoplot works with time series as well!

```
#examine the EUStockMarkets timeseires dataset
str(EuStockMarkets)
```

```
##  Time-Series [1:1860, 1:4] from 1991 to 1999: 1629 1614 1607 1621 1618 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:4] "DAX" "SMI" "CAC" "FTSE"
```

```
autoplot(datasets::EuStockMarkets)
```



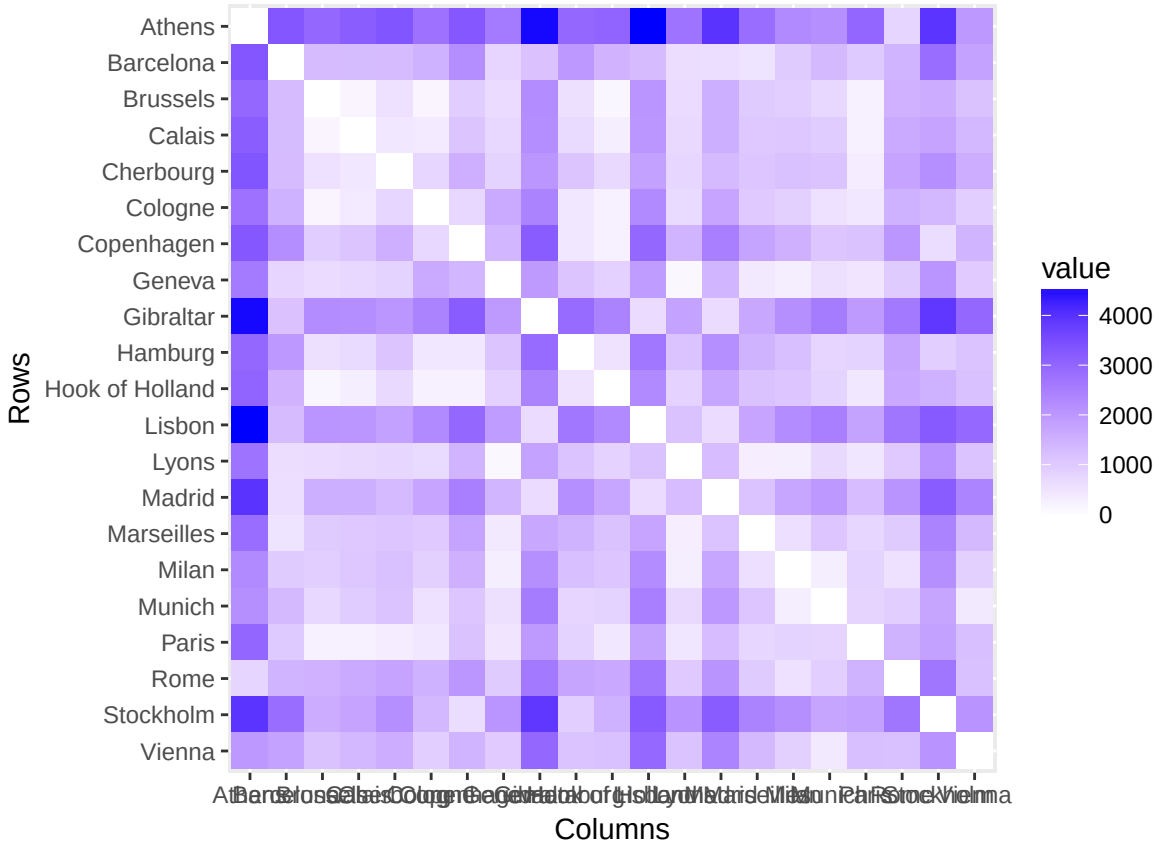
Distance Matrices and Multi Dimensional Scaling (MDS)

The cmdscale() function from the stats package performs Classical Multi-Dimensional Scaling and returns point coodinates as a matrix. Although autoplot() will work on this object, it will produce a heatmap, and not a scatter plot. However, if either eig = TRUE, add = TRUE or x.ret = TRUE is specified, cmdscale() will return a list instead of matrix. In these cases, the list method for autoplot() in the ggfortify package can deal with the output.

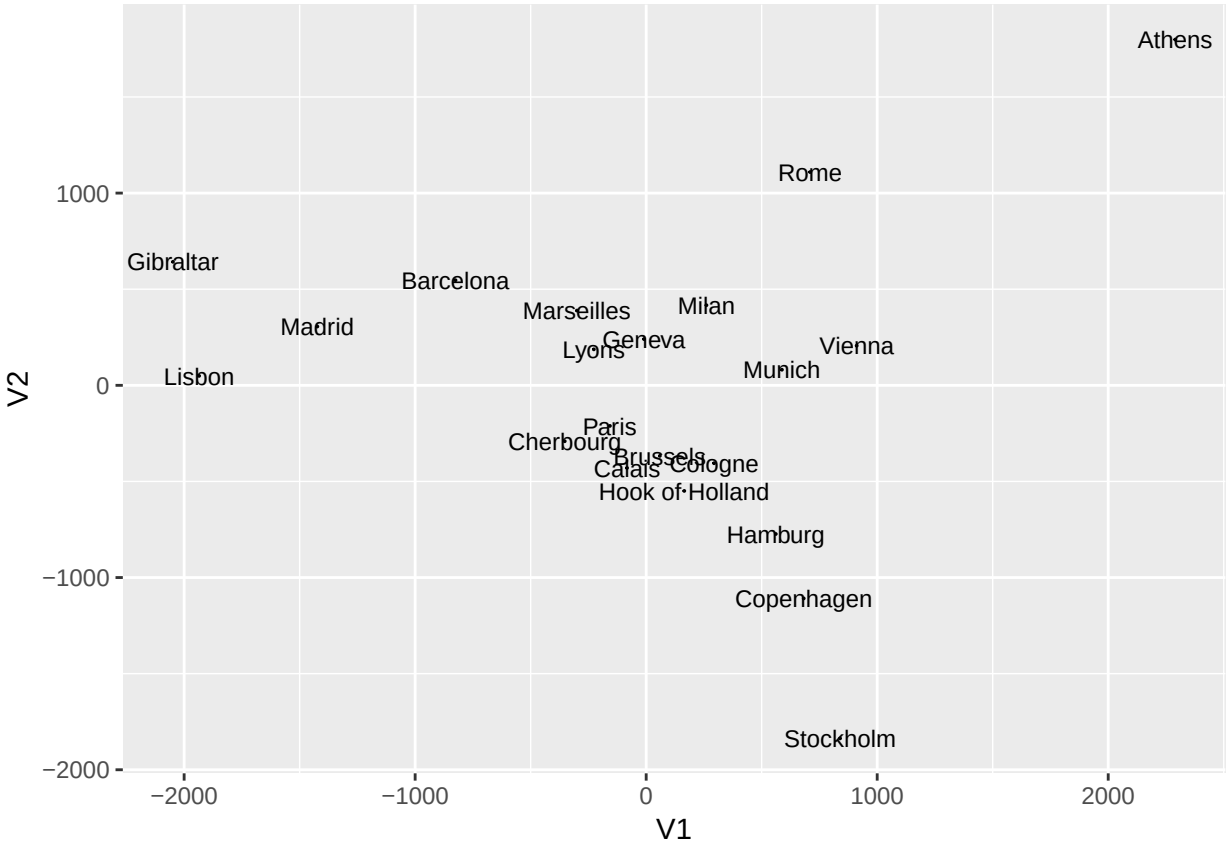
```
#examine the eurodist dataset
str(eurodist)
```

```
## Class 'dist'  atomic [1:210] 3313 2963 3175 3339 2762 ...
##   ..- attr(*, "Size")= num 21
##   ..- attr(*, "Labels")= chr [1:21] "Athens" "Barcelona" "Brussels" "Calais" ...
```

```
# Autoplot + ggplot2 tweaking
autoplot(eurodist) +
  coord_fixed()
```



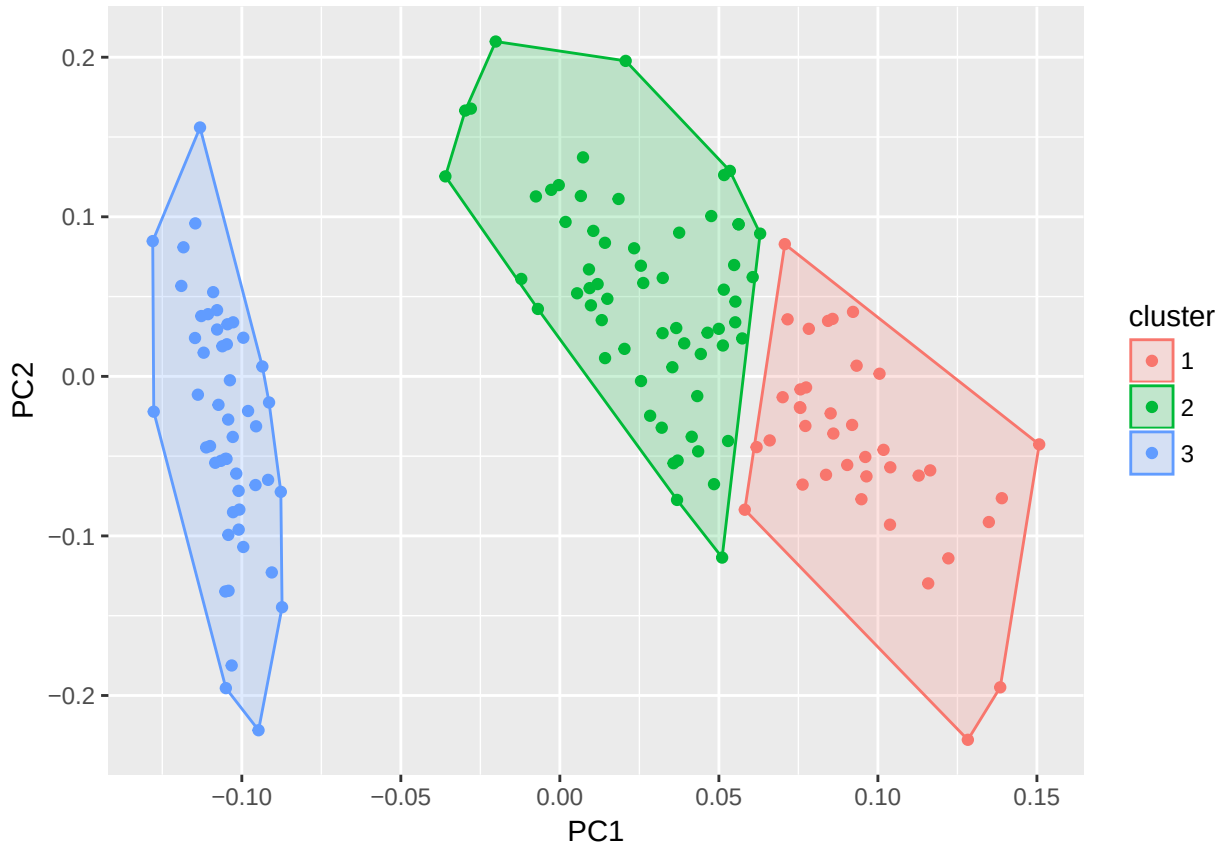

```
# Autoplot of MDS
autoplot(cmdscale(eurodist, eig = TRUE),
         label = TRUE,
         label.size = 3,
         size = 0)
```



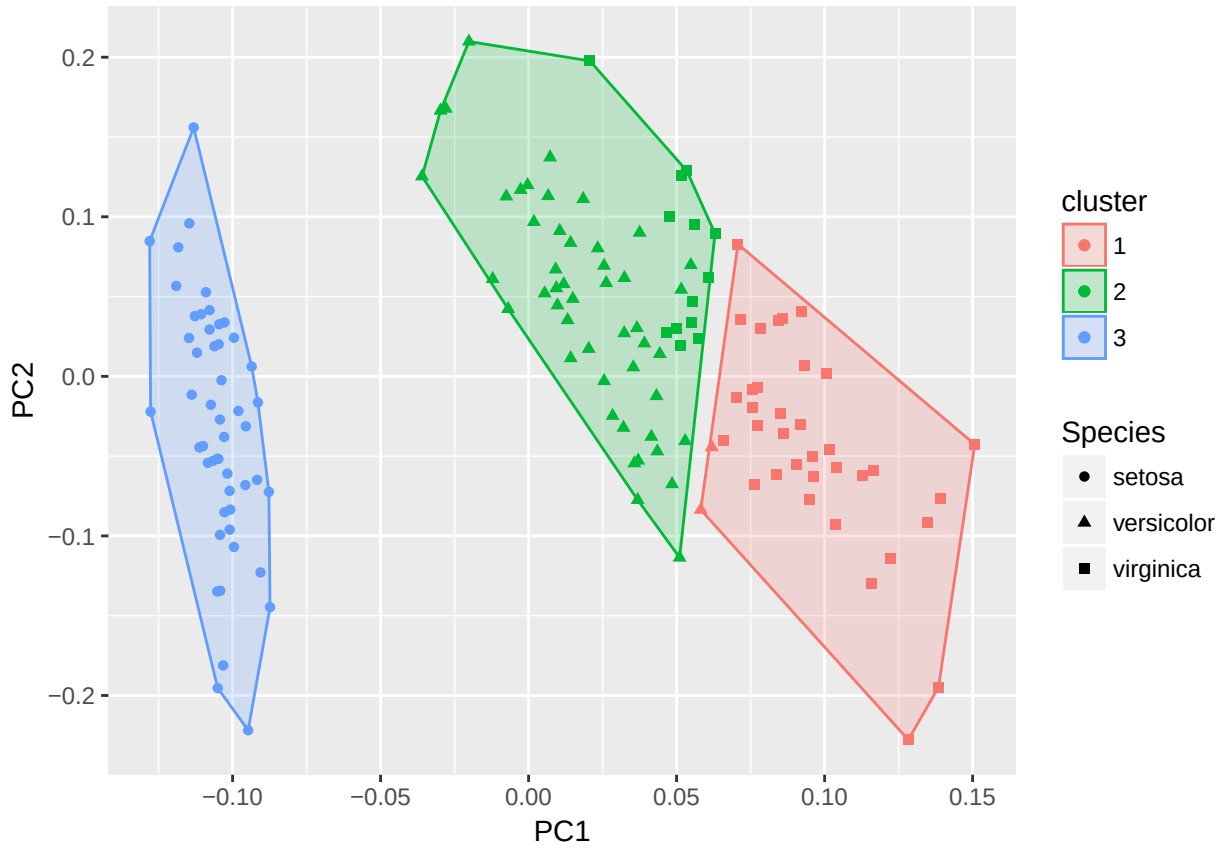
K-Means Clustering

ggfortify also supports stats::kmeans class objects. You must explicitly pass the original data to the autoplot function via the data argument, since kmeans objects don't contain the original data. The result will be automatically colored according to cluster.

```
# Perform clustering
iris_k <- kmeans(iris[-5], 3)

# Autoplot: color according to cluster
autoplot(iris_k, data = iris, frame = TRUE)
```




```
# Autoplot: above, plus shape according to species
autoplot(iris_k, data = iris, frame = TRUE, shape = 'Species')
```



Maps

Many people are turning to R as a mapping tools for Geogrpahic Information Systems (GIS). There are different types of maps, including:

- Choropleths - drawing a bunch of polygons
- Cartographic Maps

Choropleths

The `maps` package is the easiest way to obtain map polygons, although, there are only a few locations available.

The available maps of political boundaries are:

- Global: world, world2
- Country: france, italy, nz, usa
- USA: county, state

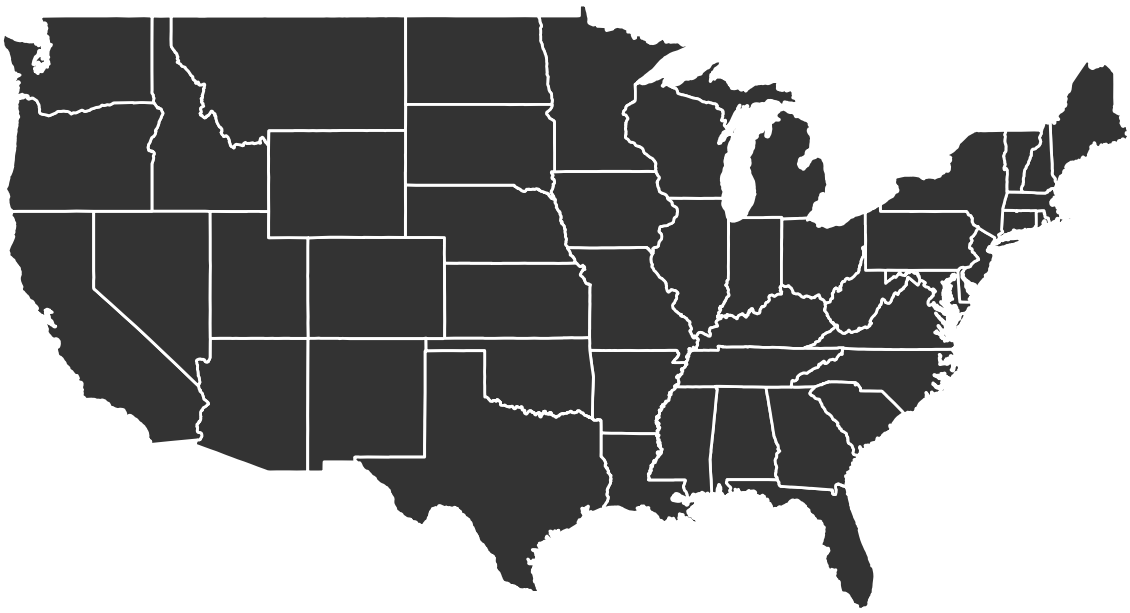
The maps can be accessed via `map_data()` from the `ggplot2` package, which converts the map into a data frame containing the variables long and lat. To draw the map, you need to use `geom_polygon()` which will connect the points of latitude and longitude for you.

```
#create map object using the map_data function
state <- map_data(map = "state")
```

```r
#examine usa data
str(state)
```

```
## 'data.frame':    15537 obs. of  6 variables:
##  $ long     : num  -87.5 -87.5 -87.5 -87.5 -87.6 ...
##  $ lat      : num  30.4 30.4 30.4 30.3 30.3 ...
##  $ group    : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ order    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ region   : chr  "alabama" "alabama" "alabama" "alabama" ...
##  $ subregion: chr  NA NA NA NA ...
```

```r
# Build the map
ggplot(state, aes(x = long, y = lat, group = group)) +
  geom_polygon(color = "white") +
  coord_map() +
  theme_void()
```



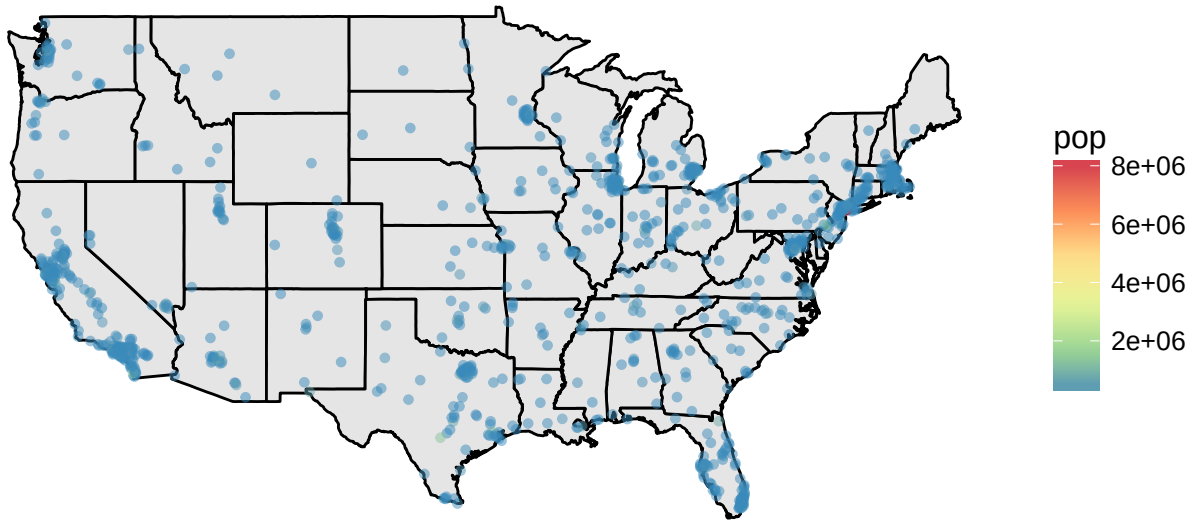
Let's add some information to our map. We will use the `us.cities` dataset from the `maps` package

```r
#load require package
library(maps)

#exmaine the dataset
str(us.cities)
```

```
## 'data.frame':    1005 obs. of  6 variables:
##  $ name       : chr  "Abilene TX" "Akron OH" "Alameda CA" "Albany GA" ...
##  $ country.etc: chr  "TX" "OH" "CA" "GA" ...
```

```
##  $ pop        : int  113888 206634 70069 75510 93576 45535 494962 44933 127159 88857 ...
##  $ lat        : num  32.5 41.1 37.8 31.6 42.7 ...
##  $ long       : num  -99.7 -81.5 -122.3 -84.2 -73.8 ...
##  $ capital    : int  0 0 0 0 2 0 0 0 0 0 ...
```

```
#Filter out Alaska and Hawaii
us.cities <- us.cities %>% filter(!country.etc %in% c('HI', 'AK'))

#Build the map with data from the us.cities dataframe
p <- ggplot(state, aes(x = long, y = lat, group = group)) +
  geom_polygon(color = 'black', fill = 'grey90') +
  coord_map() +
  theme_void()

#Diaplay map with city pop info as points
p + geom_point(data = us.cities,
               aes(group = country.etc, col = pop),
               size = 1.2, alpha = 0.5) +
  scale_color_distiller(palette = 'Spectral')
```



Cartographic Maps

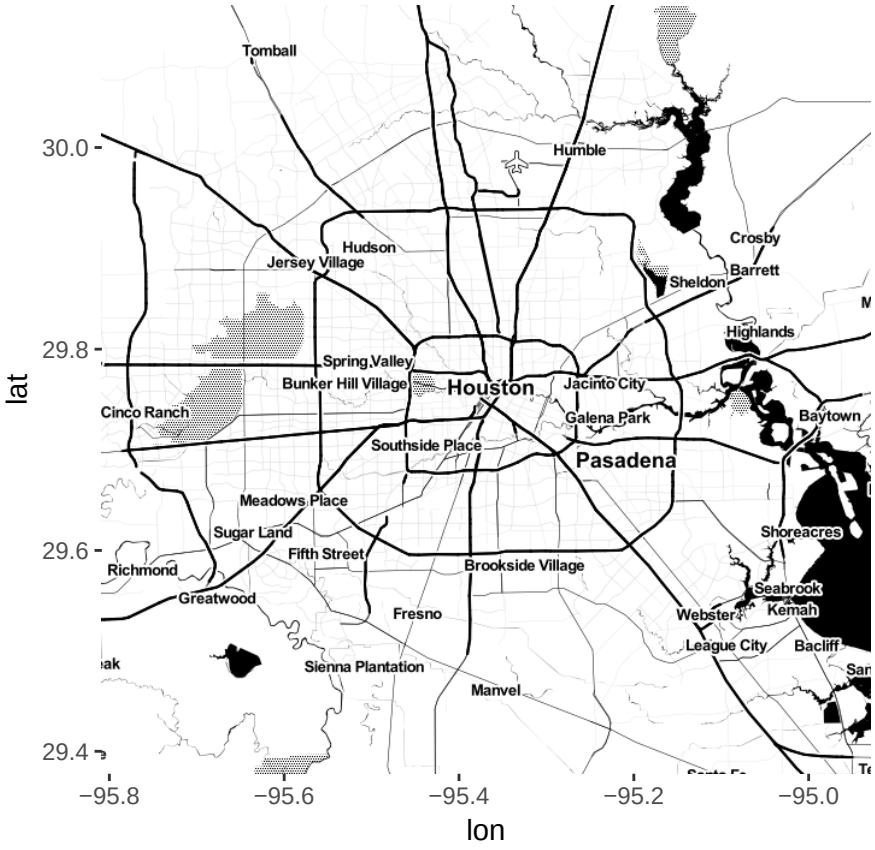
Topographical and/or Photographic Maps

Using the `ggmap`[1] package, we can use the `get_map()` function to access various maps, such as those offered

[1]D. Kahle and H. Wickham. ggmap: Spatial Visualization with ggplot2. The R Journal, 5(1), 144-161. URL http:

by google. The only essential argument is `location`, however, there are many other options which you can specify.

```
#load required package
library(ggmap)

houston <- get_map(location = "Houston, Texas", source = "stamen", maptype = "toner")

#plot location using ggmap
ggmap(houston)
```



Similar to choropleths, we can map points onto a cartographic map. Using the houston map from above, we can map the `crime` dataset from the `ggmap` package which include Houston related crimes between Jan 2010 to August 2010.

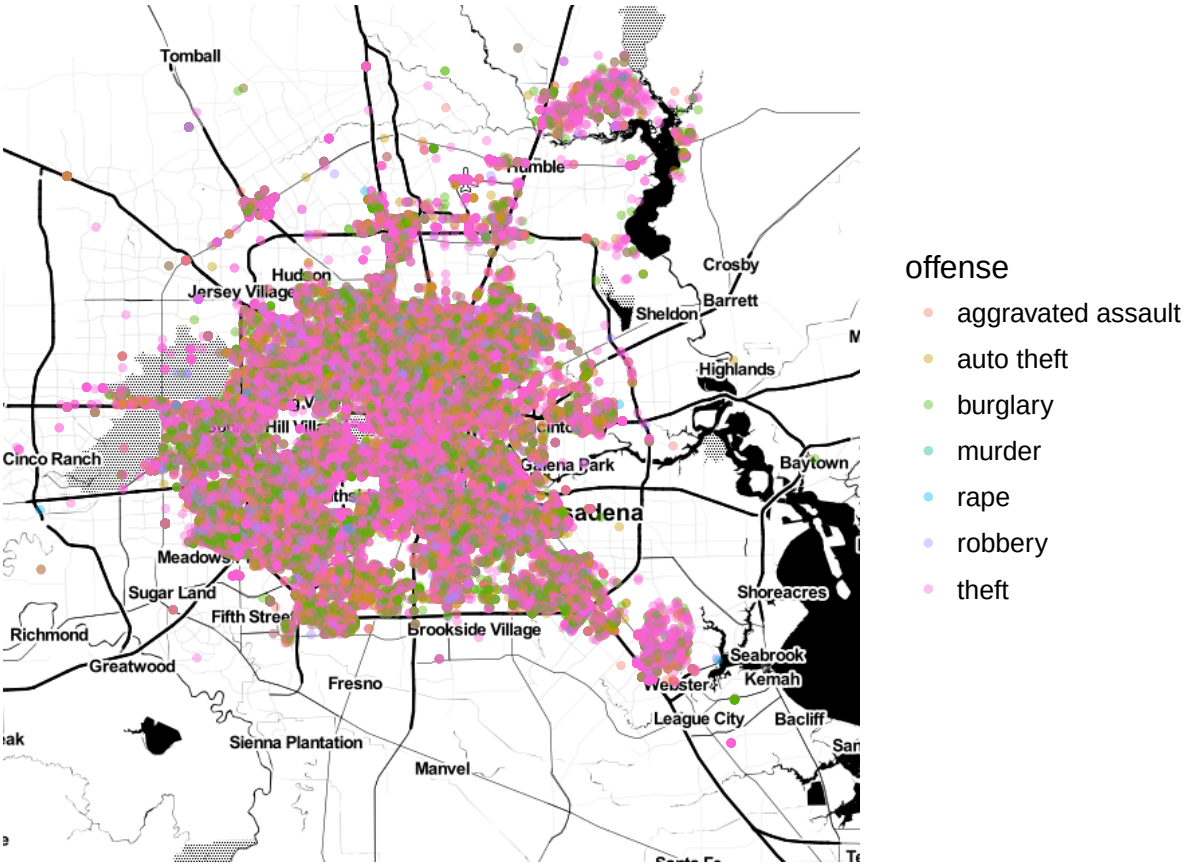
```
#load required package
library(ggthemes) #for map theme

#examine the crime dataset
str(crime)
```

```
## 'data.frame':    86314 obs. of  17 variables:
##  $ time    : POSIXt, format: "2009-12-31 22:00:00" "2009-12-31 22:00:00" ...
##  $ date    : chr  "1/1/2010" "1/1/2010" "1/1/2010" "1/1/2010" ...
##  $ hour    : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ premise : chr  "18A" "13R" "20R" "20R" ...
##  $ offense : Factor w/ 7 levels "aggravated assault",..: 4 6 1 1 1 3 3 3 3 3 ...
```

//journal.r-project.org/archive/2013-1/kahle-wickham.pdf

```
##  $ beat    : chr  "15E30" "13D10" "16E20" "2A30" ...
##  $ block   : chr  "9600-9699" "4700-4799" "5000-5099" "1000-1099" ...
##  $ street  : chr  "marlive" "telephone" "wickview" "ashland" ...
##  $ type    : chr  "ln" "rd" "ln" "st" ...
##  $ suffix  : chr  "-" "-" "-" "-" ...
##  $ number  : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ month   : Ord.factor w/ 8 levels "january"<"february"<..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ day     : Ord.factor w/ 7 levels "monday"<"tuesday"<..: 5 5 5 5 5 5 5 5 5 5 ...
##  $ location: chr  "apartment parking lot" "road / street / sidewalk" "residence / house" "residence /
##  $ address : chr  "9650 marlive ln" "4750 telephone rd" "5050 wickview ln" "1050 ashland st" ...
##  $ lon     : num  -95.4 -95.3 -95.5 -95.4 -95.4 ...
##  $ lat     : num  29.7 29.7 29.6 29.8 29.7 ...
```

```
#map the crime statistics colored by offense
ggmap(houston) +
  geom_point(data = crime, aes(col = offense), size = 1, alpha = 0.4)+
  theme_void()
```



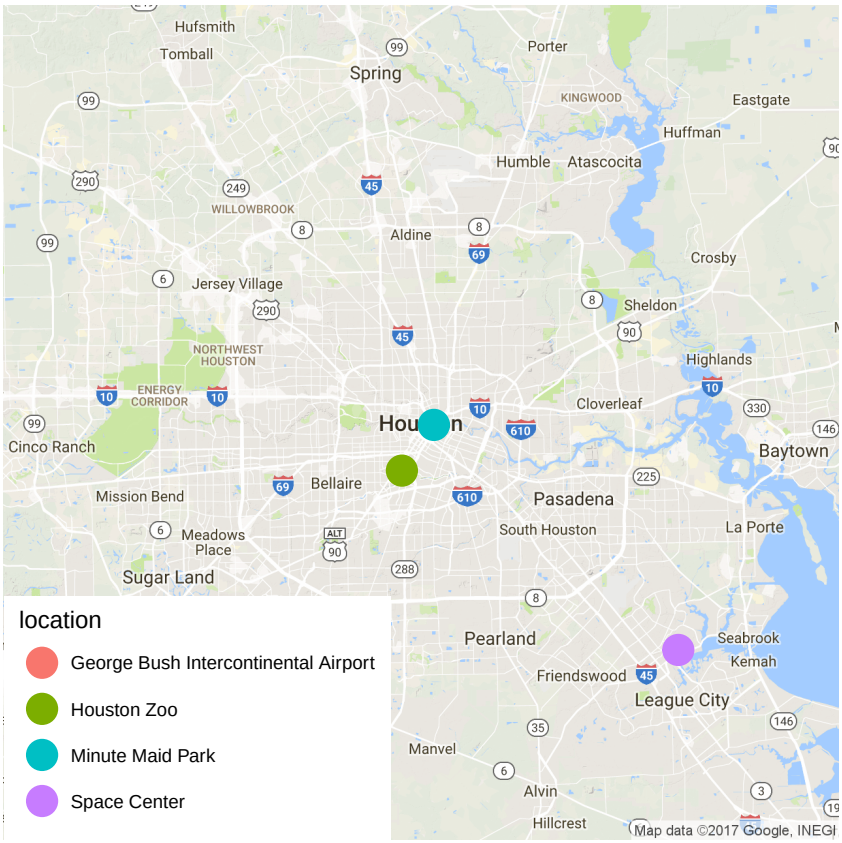
Not bad! ggmap even has functions which allow you to *geocode* locations by pulling the information from the internet (google).

```
#examine the sites around houston we are interested in
houston_sites <- c("Houston Zoo, Houston", "Space Center, Houston", "Minute Maid Park, Houston", "George

#geocode houston_sites
codes <- geocode(houston_sites)

codes$location <- str_replace(string = houston_sites, pattern = ", Houston", replace = "")
```

```
#examine codes
codes
```

```
##         lon      lat                                location
## 1 -95.38922 29.71544                        Houston Zoo
## 2 -95.09799 29.55119                      Space Center
## 3 -95.35552 29.75727                  Minute Maid Park
## 4        NA       NA George Bush Intercontinental Airport
```

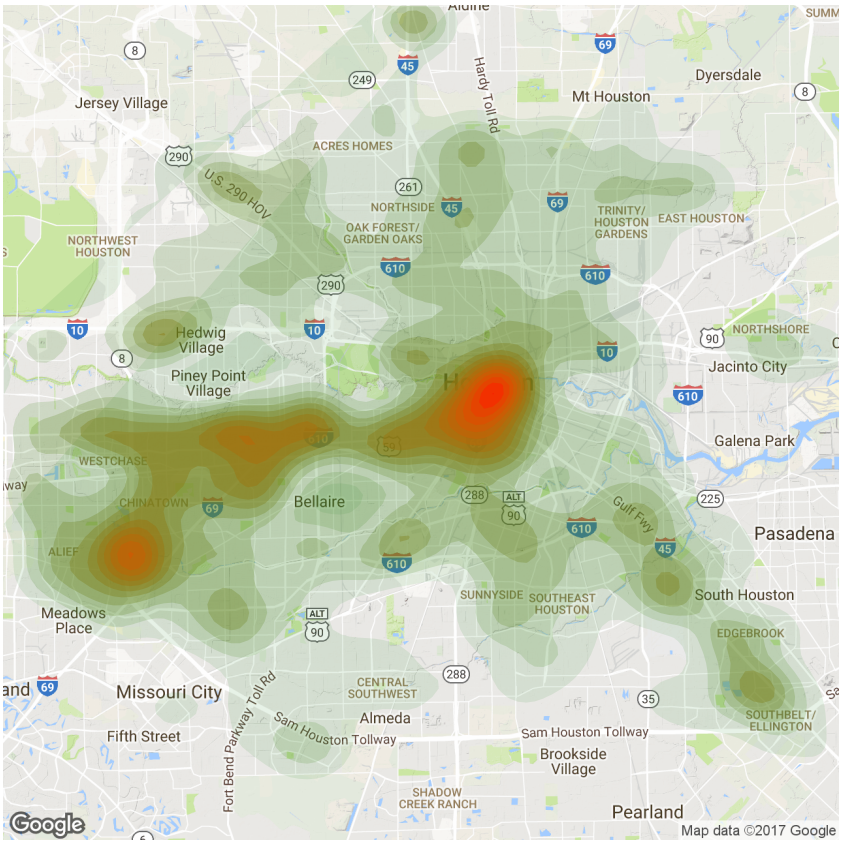
```
#create map object
houston <- get_map(location = "Houston, Texas", source = "google", maptype = 'terrain')

#map our newly coded locations
ggmap(houston) +
  geom_point(data = codes, aes(color = location), size = 5) +
  theme_map()
```



There's one more way we can define the coordinates of our map. Instead of using a particular location, The `bbox` function allows you to define the boundary box around your coordinates. We'll use the `wind` dataset which has wind data from Hurricane Ike.

```
#filter crime to include only points in Houston area
crime_sub <- crime %>% filter(lon <= -94.5 & lon > -97 & lat > 26 & lat < 30)

#create a bbox using the coordinates of the long and lat of the codes dataframe.
#f is the fraction to which the boundaries are to be extended.
bbox <- make_bbox(data = crime_sub, lon = lon, lat = lat, f = 1)
```

```
#create map object using bbox coordinates
houston <- get_map(location = bbox, zoom = 11,  maptype = 'terrain')

ggmap(houston) +
  stat_density2d(data = crime_sub,
                 aes(x = lon, y = lat, fill = ..level.., alpha = ..level..),
                 geom = 'polygon', bins = 15, show.legend = F) +
  scale_fill_gradient(low = "forestgreen", high = "red") +
  theme_map()
```



Animations

Animations are useful for dense temporal data, or geospatial data. They can be used as a great exploratory tool as well.

The `animation` and `gganimate` packages are useful for creating plots with motion.

You will need a graphic device converter such as ImageMagick or GraphicsMagick to get your animated plots to work.

For MAC users, you may also need to install FFMPEG.

```
library(animation)

#Use saveGIF function with for loop to create animated plot object
saveGIF(expr = {
```

```
  #sets environment options to use the graphics magick convert option
  ani.options(convert = "gm convert")

  #use for loop to create a series of plots which will be pasted together
  for (i in 1:10) plot(runif(25), ylim = 0:1)

  },movie.name = "runif.gif", img.name = 'runif', interval = 0.2, convert = 'gm convert')
```

```
## [1] TRUE
```

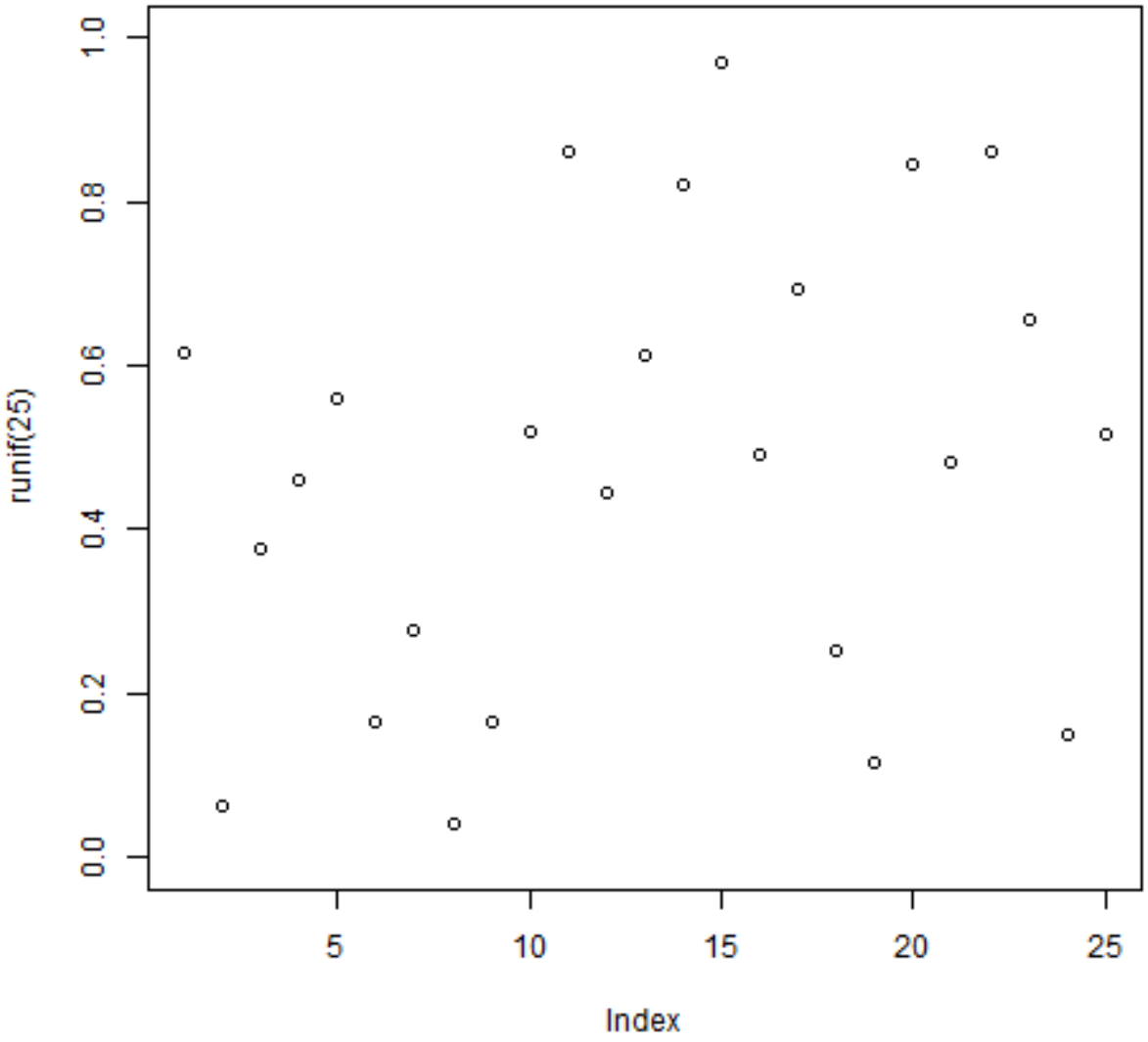


Figure 1: histogram

```
#load package
library(gganimate)
library(lubridate)
library(gapminder)

#turn sunspots data into dataframe
sunspots <- fortify(sunspots)

names(sunspots) <- c("date", "sunspots")

#create year and month variables using (lubridate functions) year & month
sunspots <- sunspots %>%
  mutate(year = year(date),
         month = month(date, label = T))

#create plot object with sunspots data. Apply year to the frame aesthetic
p <- ggplot(sunspots, aes(x = month, y = sunspots, group = year, frame = year))+
  geom_line()

#Make animated plot, save to file in working directory
gganimate(p = p, interval = 0.05, filename = 'sunspots.gif', convert = "gm convert")
```

Let's put our skills to the test by applying some animation to some cartographic/map data. For this we will use the `gapminder` data from the identically named package which provides demographic information on various countries from 1952 to 2007.