

With ever increasing volume of data, it is impossible to tell stories without visualizations. Data visualization is an art of how to turn numbers into useful knowledge.

R Programming lets you learn this art by offering a set of inbuilt functions and libraries to build visualizations and present data. Before the technical implementations of the visualization, let’s see first how to select the right chart type.

Selecting the Right Chart Type

There are four basic presentation types:

Comparison

Composition

Distribution

Relationship

To determine which amongst these is best suited for your data, I suggest you should answer a few questions like,

How many variables do you want to show in a single chart?

How many data points will you display for each variable?

Will you display values over a period of time, or among items or groups?

* Scatter Plot
* Histogram
* Bar & Stack Bar Chart
* Box Plot
* Area Chart
* Heat Map
* Correlogram
* R - Pie Charts
* R - Line Graphs
* 3D Graphs
* Mosaic Map
* Map Visualization

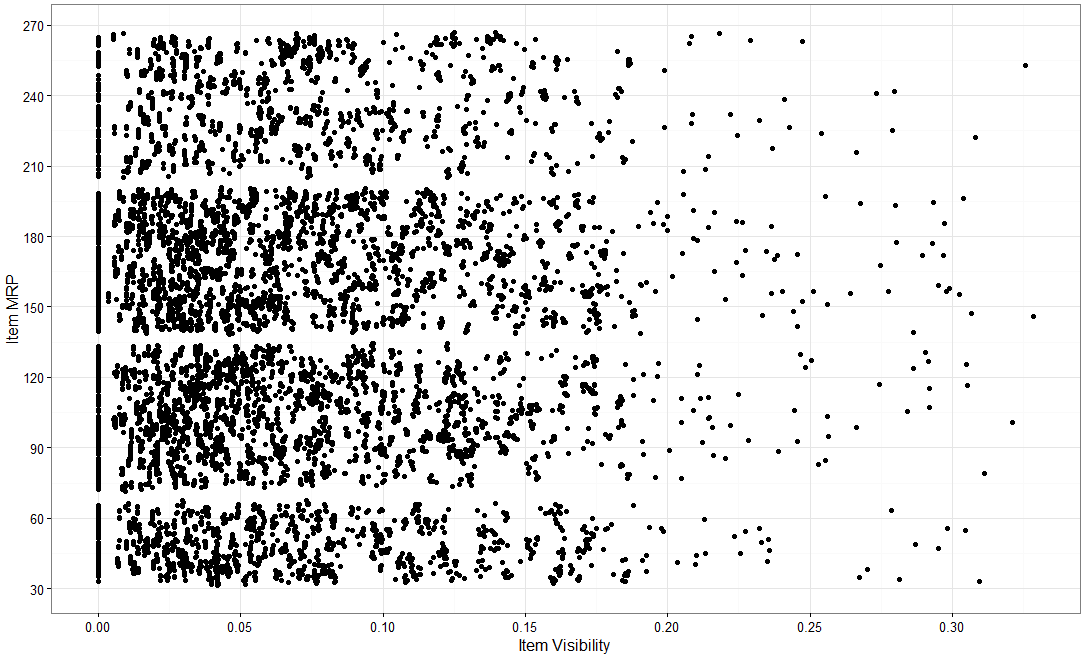
We’ll use ‘Big Mart data’ example as shown below to understand how to create visualizations in R. You can download the full dataset from here.

Now let’s see how to use these visualizations in R

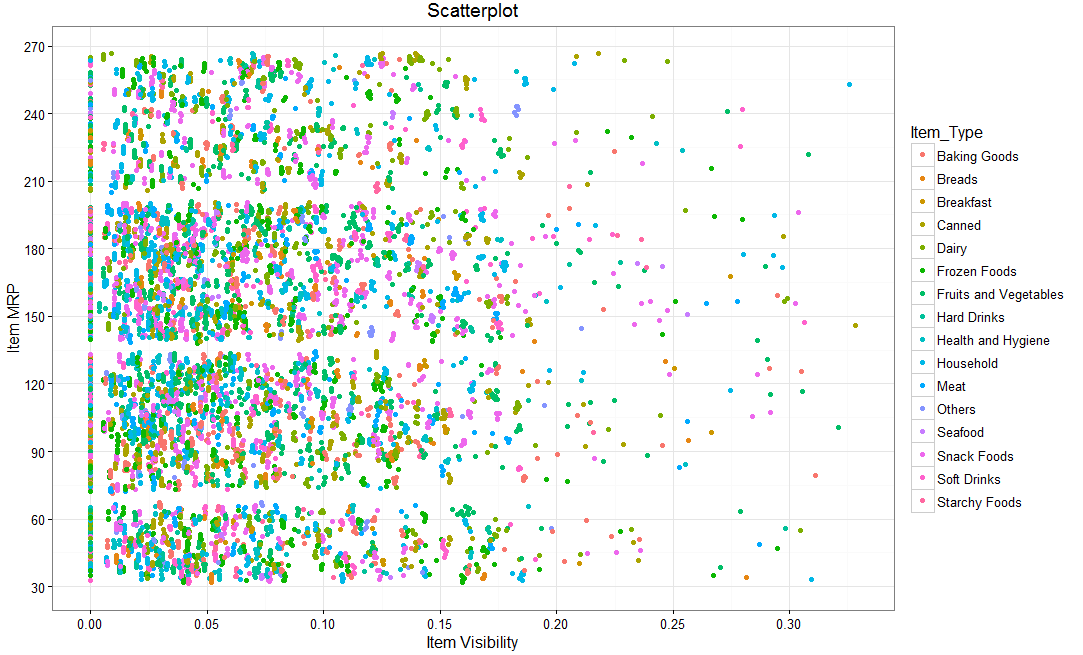
1. Scatter Plot

When to use: Scatter Plot is used to see the relationship between two continuous variables.

In our above mart dataset, if we want to visualize the items as per their cost data, then we can use scatter plot chart using two continuous variables, namely Item\_Visibility & Item\_MRP as shown below.



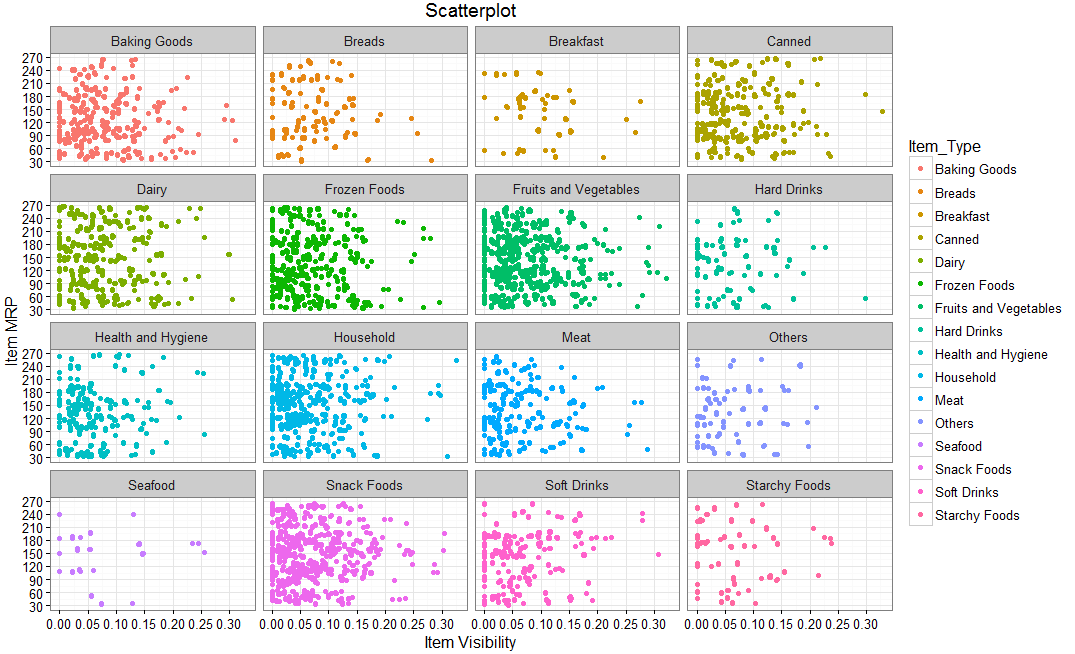
Here is the R code for simple scatter plot using function ggplot() with geom\_point().



library(ggplot2) // ggplot2 is an R library for visualizations train.

ggplot(train, aes(Item\_Visibility, Item\_MRP)) + geom\_point() + scale\_x\_continuous("Item Visibility", breaks = seq(0,0.35,0.05))+ scale\_y\_continuous("Item MRP", breaks = seq(0,270,by = 30))+ theme\_bw()

Now, we can view a third variable also in same chart, say a categorical variable (Item\_Type) which will give the characteristic (item\_type) of each data set. Different categories are depicted by way of different color for item\_type in below chart.



R code with an addition of category:

ggplot(train, aes(Item\_Visibility, Item\_MRP)) + geom\_point(aes(color = Item\_Type)) +

scale\_x\_continuous("Item Visibility", breaks = seq(0,0.35,0.05))+

scale\_y\_continuous("Item MRP", breaks = seq(0,270,by = 30))+

theme\_bw() + labs(title="Scatterplot")

We can even make it more visually clear by creating separate scatter plots for each separate Item\_Type as shown below.

R code for separate category wise chart:

ggplot(train, aes(Item\_Visibility, Item\_MRP)) + geom\_point(aes(color = Item\_Type)) +

scale\_x\_continuous("Item Visibility", breaks = seq(0,0.35,0.05))+

scale\_y\_continuous("Item MRP", breaks = seq(0,270,by = 30))+

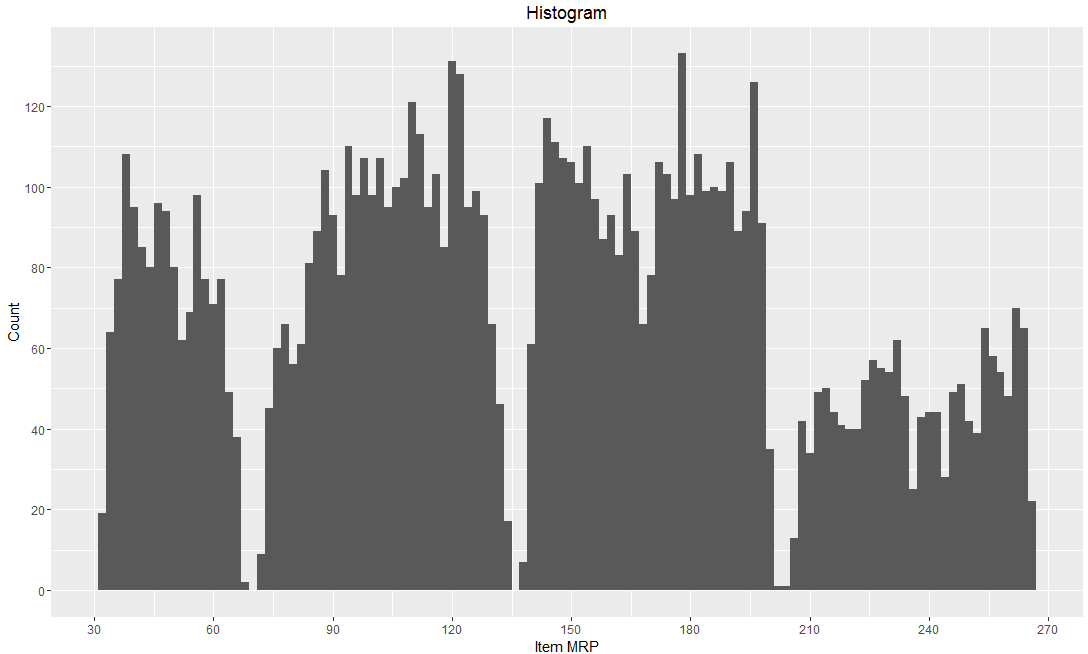
theme\_bw() + labs(title="Scatterplot") + facet\_wrap( ~ Item\_Type)

Here, facet\_wrap works superb & wraps Item\_Type in rectangular layout.

2. Histogram

When to use: Histogram is used to plot continuous variable. It breaks the data into bins and shows frequency distribution of these bins. We can always change the bin size and see the effect it has on visualization.

From our mart dataset, if we want to know the count of items on basis of their cost, then we can plot histogram using continuous variable Item\_MRP as shown below.



Here is the R code for simple histogram plot using function ggplot() with geom\_histogram().

ggplot(train, aes(Item\_MRP)) + geom\_histogram(binwidth = 2)+

scale\_x\_continuous("Item MRP", breaks = seq(0,270,by = 30))+

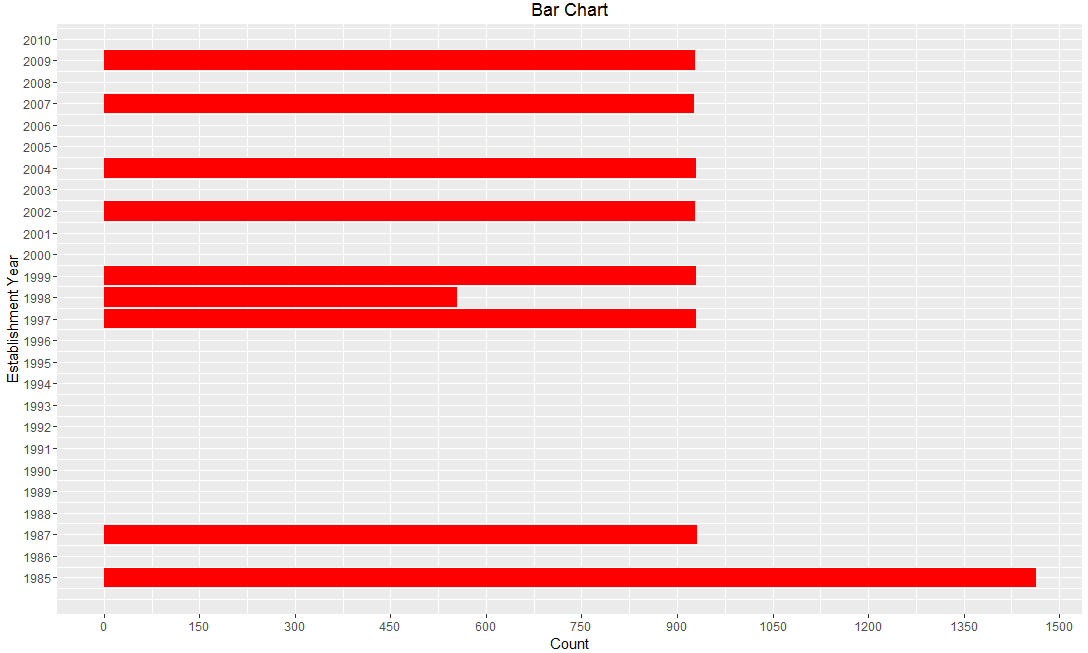
scale\_y\_continuous("Count", breaks = seq(0,200,by = 20))+

labs(title = "Histogram")

3. Bar & Stack Bar Chart

When to use: Bar charts are recommended when you want to plot a categorical variable or a combination of continuous and categorical variable.

From our dataset, if we want to know number of marts established in particular year, then bar chart would be most suitable option, use variable Establishment Year as shown below.



Here is the R code for simple bar plot using function ggplot() for a single continuous variable.

ggplot(train, aes(Outlet\_Establishment\_Year)) + geom\_bar(fill = "red")+theme\_bw()+

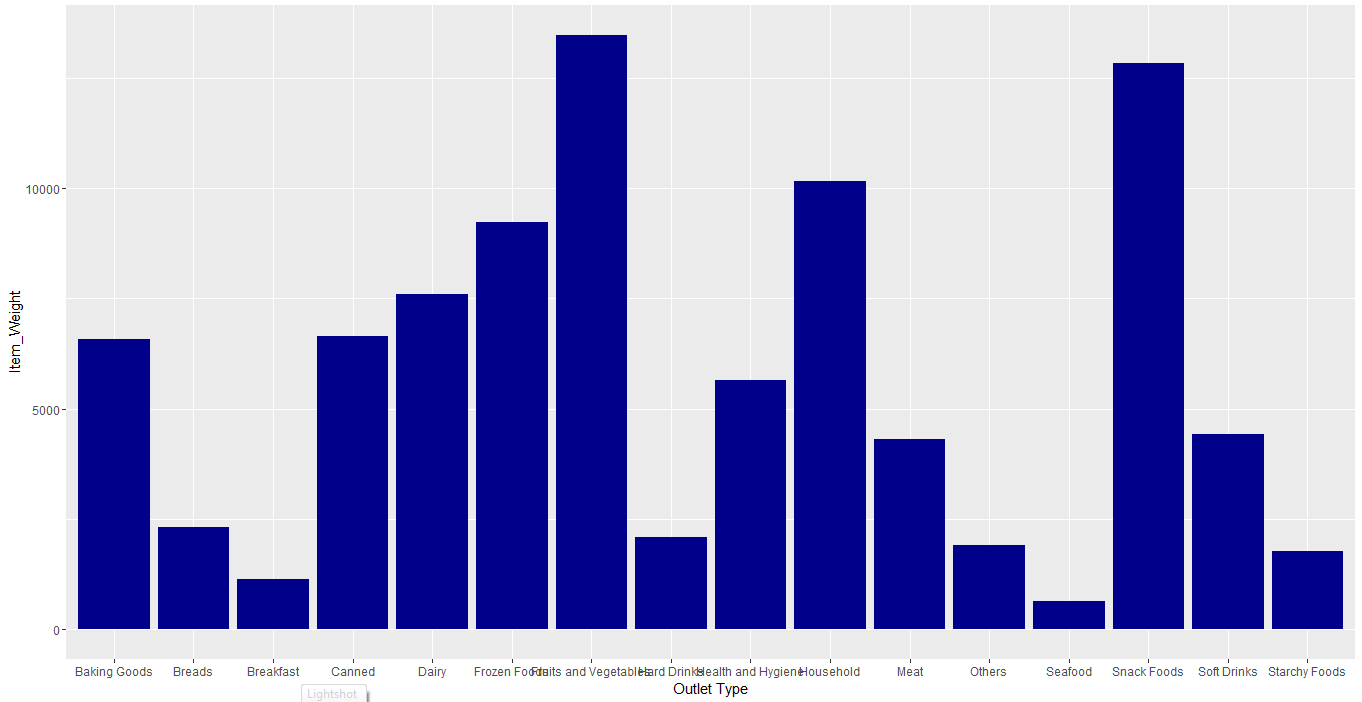
scale\_x\_continuous("Establishment Year", breaks = seq(1985,2010)) +

scale\_y\_continuous("Count", breaks = seq(0,1500,150)) +

coord\_flip()+ labs(title = "Bar Chart") + theme\_gray()

Vertical Bar Chart:

As a variation, you can remove coord\_flip() parameter to get the above bar chart vertically.

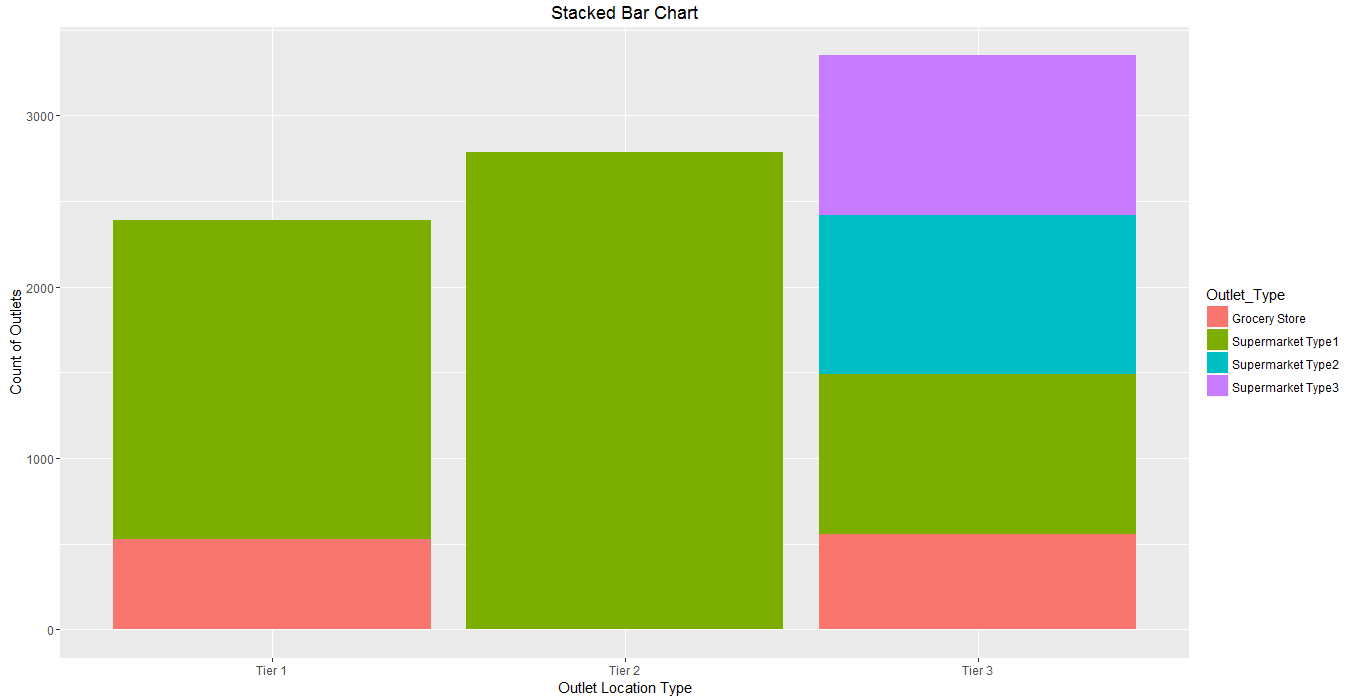


To know item weights (continuous variable) on basis of Outlet Type (categorical variable) on single bar chart, use following code:

ggplot(train, aes(Item\_Type, Item\_Weight)) + geom\_bar(stat = "identity", fill = "darkblue") + scale\_x\_discrete("Outlet Type")+ scale\_y\_continuous("Item Weight", breaks = seq(0,15000, by = 500))+ theme(axis.text.x = element\_text(angle = 90, vjust = 0.5)) + labs(title = "Bar Chart")

Stacked Bar chart:

Stacked bar chart is an advanced version of bar chart, used for visualizing a combination of categorical variables.



From our dataset, if we want to know the count of outlets on basis of categorical variables like its type (Outlet Type) and location (Outlet Location Type) both, stack chart will visualize the scenario in most useful manner.

Here is the R code for simple stacked bar chart using function ggplot().

ggplot(train, aes(Outlet\_Location\_Type, fill = Outlet\_Type)) + geom\_bar()+

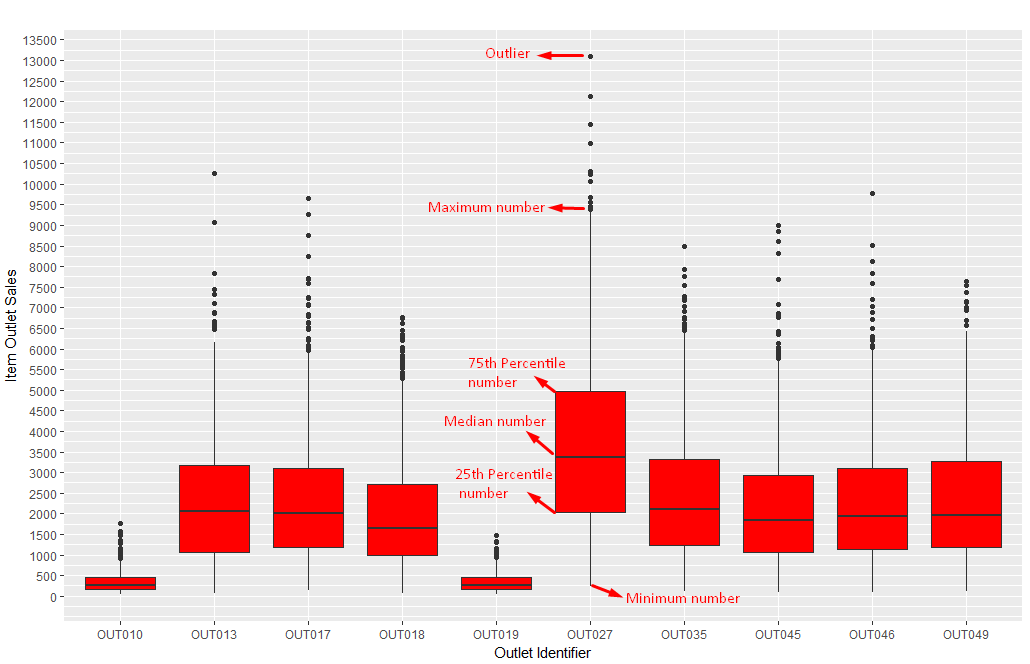
labs(title = "Stacked Bar Chart", x = "Outlet Location Type", y = "Count of Outlets")

4. Box Plot

When to use: Box Plots are used to plot a combination of categorical and continuous variables. This plot is useful for visualizing the spread of the data and detect outliers. It shows five statistically significant numbers- the minimum, the 25th percentile, the median, the 75th percentile and the maximum.

From our dataset, if we want to identify each outlet’s detailed item sales including minimum, maximum & median numbers, box plot can be helpful. In addition, it also gives values of outliers of item sales for each outlet as shown in below chart.

The black points are outliers. Outlier detection and removal is an essential step of successful data exploration.



Here is the R code for simple box plot using function ggplot() with geom\_boxplot.

ggplot(train, aes(Outlet\_Identifier, Item\_Outlet\_Sales)) + geom\_boxplot(fill = "red")+

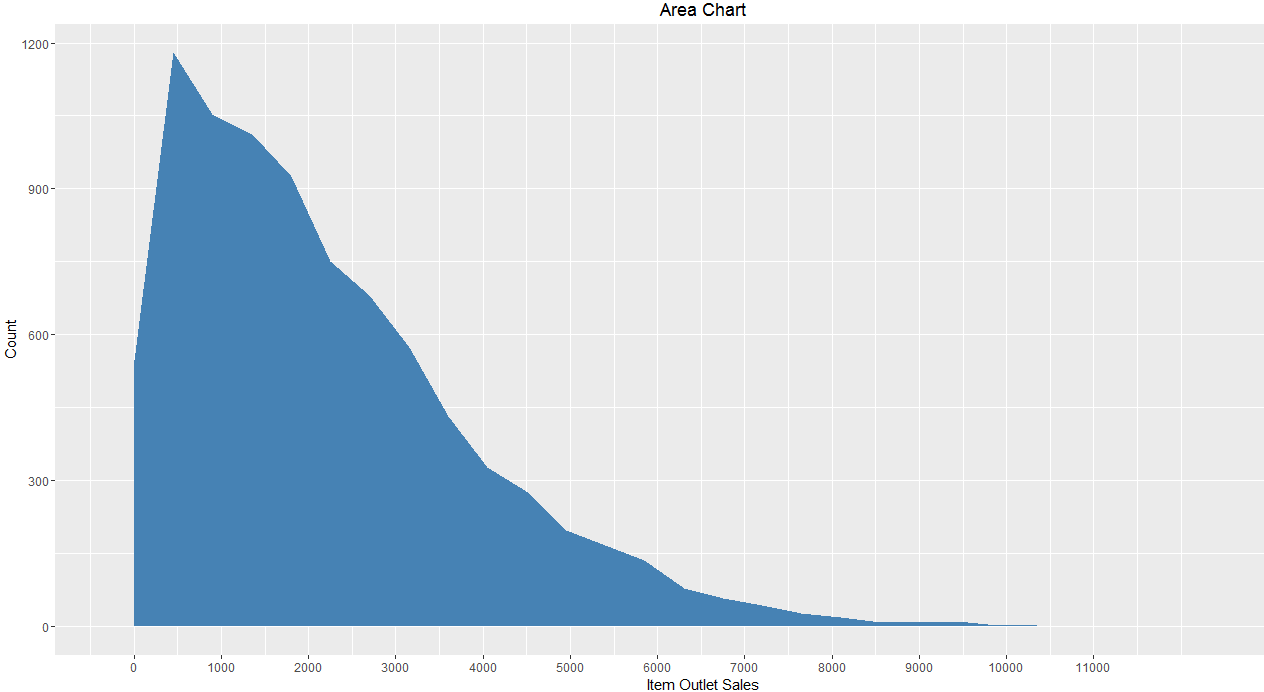
scale\_y\_continuous("Item Outlet Sales", breaks= seq(0,15000, by=500))+

labs(title = "Box Plot", x = "Outlet Identifier")

5. Area Chart

When to use: Area chart is used to show continuity across a variable or data set. It is very much same as line chart and is commonly used for time series plots. Alternatively, it is also used to plot continuous variables and analyze the underlying trends.

From our dataset, when we want to analyze the trend of item outlet sales, area chart can be plotted as shown below. It shows count of outlets on basis of sales.



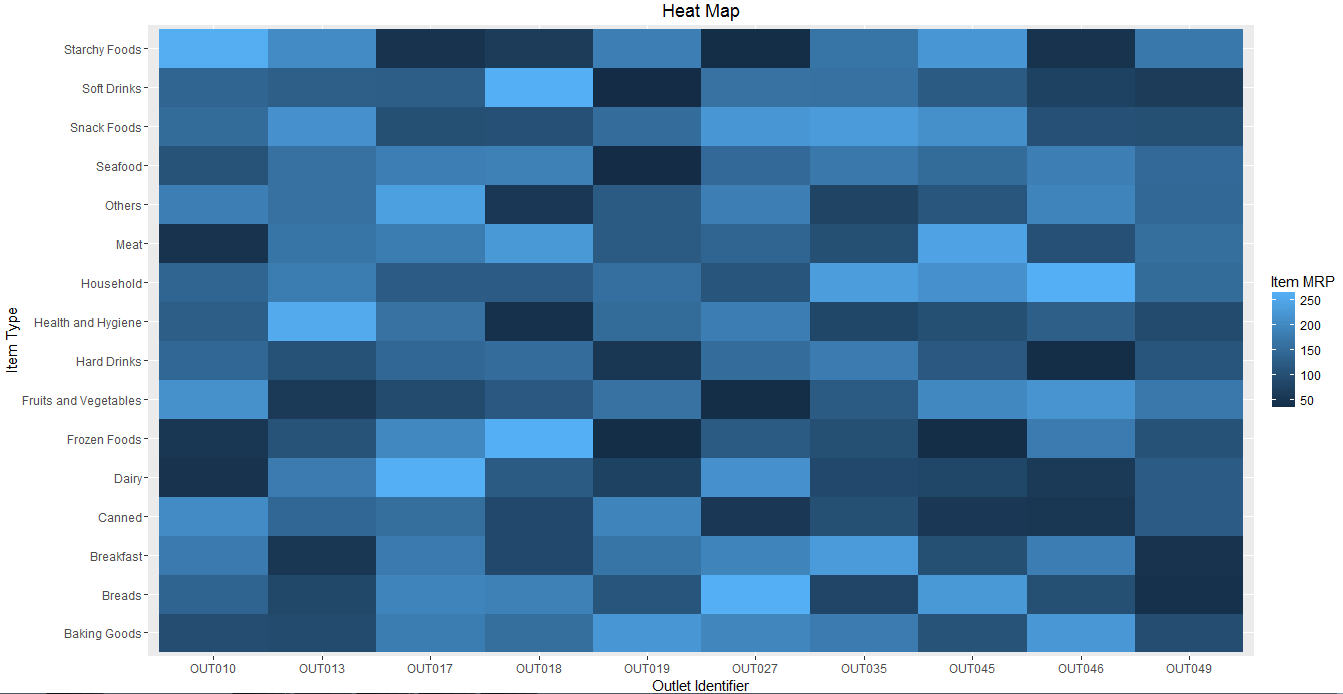
Here is the R code for simple area chart showing continuity of Item Outlet Sales using function ggplot() with geom\_area.

ggplot(train, aes(Item\_Outlet\_Sales)) + geom\_area(stat = "bin", bins = 30, fill = "steelblue") + scale\_x\_continuous(breaks = seq(0,11000,1000))+ labs(title = "Area Chart", x = "Item Outlet Sales", y = "Count")

6. Heat Map

When to use: Heat Map uses intensity (density) of colors to display relationship between two or three or many variables in a two dimensional image. Heat Map Analysis for website allows you to explore two dimensions as the axis and the third dimension by intensity of color.

From our dataset, if we want to know cost of each item on every outlet, we can plot heatmap as shown below using three variables Item MRP, Outlet Identifier & Item Type from our mart dataset.



The dark portion indicates Item MRP is close 50. The brighter portion indicates Item MRP is close to 250.

Here is the R code for simple heat map using function ggplot().

ggplot(train, aes(Outlet\_Identifier, Item\_Type))+

geom\_raster(aes(fill = Item\_MRP))+

labs(title ="Heat Map", x = "Outlet Identifier", y = "Item Type")+

scale\_fill\_continuous(name = "Item MRP")

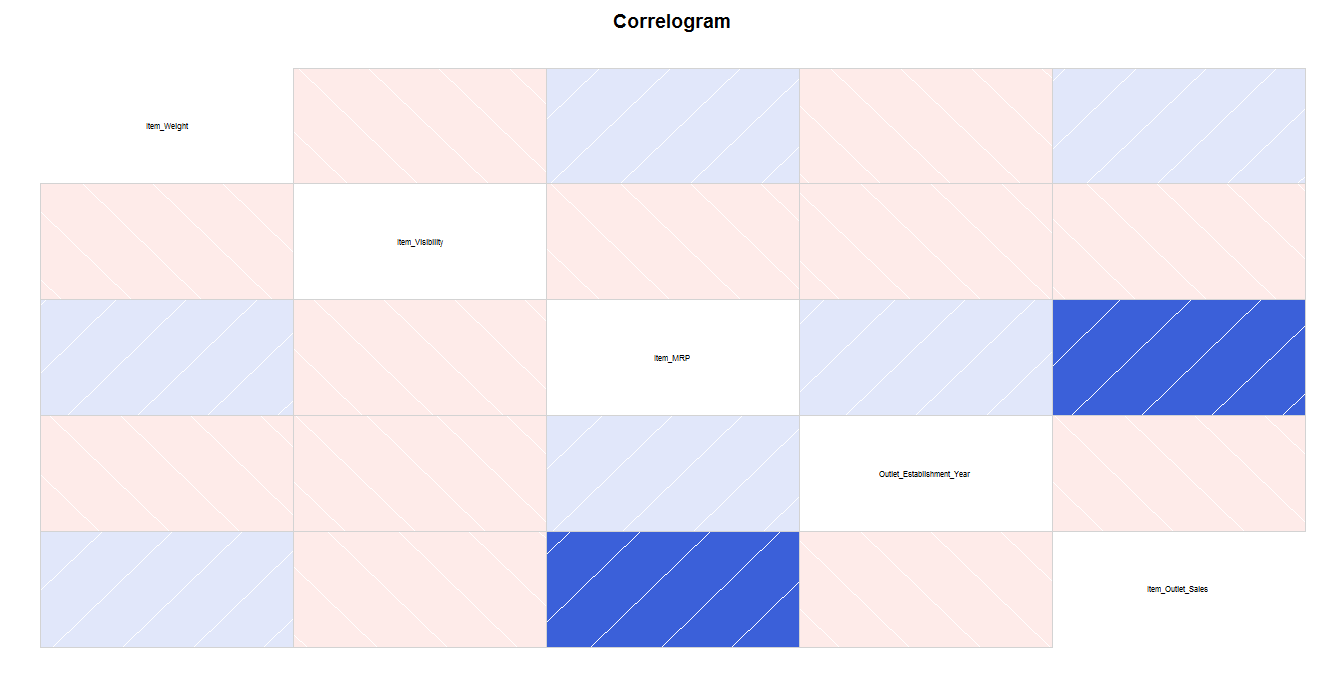
7. Correlogram

When to use: Correlogram is used to test the level of co-relation among the variable available in the data set. The cells of the matrix can be shaded or colored to show the co-relation value.

Darker the color, higher the co-relation between variables. Positive co-relations are displayed in blue and negative correlations in red color. Color intensity is proportional to the co-relation value.

From our dataset, let’s check co-relation between Item cost, weight, visibility along with Outlet establishment year and Outlet sales from below plot.

In our example, we can see that Item cost & Outlet sales are positively correlated while Item weight & its visibility are negatively correlated.



Here is the R code for simple correlogram using function corrgram().

install.packages("corrgram")

library(corrgram)

corrgram(train, order=NULL, panel=panel.shade, text.panel=panel.txt,

main="Correlogram")

Now I guess it should be easy for you to visualize the data using ggplot2 library in R Programming.

# Pie Charts

Pie charts are not recommended in the R documentation, and their features are somewhat limited. The authors recommend bar or dot plots over pie charts because people are able to judge length more accurately than volume. Pie charts are created with the function **pie(**x**, labels=)** where x is a non-negative numeric vector indicating the area of each slice and labels= notes a character vector of names for the slices.

## Simple Pie Chart

# Simple Pie Chart  
slices <- c(10, 12,4, 16, 8)  
lbls <- c("US", "UK", "Australia", "Germany", "France")  
pie(slices, labels = lbls, main="Pie Chart of Countries")

[](https://www.statmethods.net/graphs/images/pie1.jpg) click to view

## Pie Chart with Annotated Percentages

# Pie Chart with Percentages  
slices <- c(10, 12, 4, 16, 8)   
lbls <- c("US", "UK", "Australia", "Germany", "France")  
pct <- round(slices/sum(slices)\*100)  
lbls <- paste(lbls, pct) # add percents to labels   
lbls <- paste(lbls,"%",sep="") # ad % to labels   
pie(slices,labels = lbls, col=rainbow(length(lbls)),  
   main="Pie Chart of Countries")

[](https://www.statmethods.net/graphs/images/pie2.jpg) click to view

## 3D Pie Chart

The **pie3D( )** function in the plotrix package provides 3D exploded pie charts.

# 3D Exploded Pie Chart  
library(plotrix)  
slices <- c(10, 12, 4, 16, 8)   
lbls <- c("US", "UK", "Australia", "Germany", "France")  
pie3D(slices,labels=lbls,explode=0.1,  
   main="Pie Chart of Countries ")

[](https://www.statmethods.net/graphs/images/pie3.jpg) click to view

## Creating Annotated Pies from a data frame

# Pie Chart from data frame with Appended Sample Sizes  
mytable <- table(iris$Species)  
lbls <- paste(names(mytable), "\n", mytable, sep="")  
pie(mytable, labels = lbls,   
   main="Pie Chart of Species\n (with sample sizes)")

# Line Charts

## Overview

Line charts are created with the function **lines(**x**,** y**, type=)** where x and y are numeric vectors of (x,y) points to connect. **type=** can take the following values:

|  |  |
| --- | --- |
| **type** | **description** |
| **p** | points |
| **l** | lines |
| **o** | overplotted points and lines |
| **b, c** | points (empty if "c") joined by lines |
| **s, S** | stair steps |
| **h** | histogram-like vertical lines |
| **n** | does not produce any points or lines |

The **lines( )** function adds information to a graph. It can not produce a graph on its own. Usually it follows a **plot(**x**,**y**)** command that produces a graph.

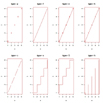
By default, **plot( )**plots the (x,y) points. Use the **type="n"** option in the **plot( )** command, to create the graph with axes, titles, etc., but without plotting the points.

(To practice creating line charts with this **lines( )** function, try this exercise.)

## Example

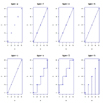
In the following code each of the **type=** options is applied to the same dataset. The **plot( )** command sets up the graph, but **does not** plot the points.

x <- c(1:5); y <- x # create some data   
par(pch=22, col="red") # plotting symbol and color   
par(mfrow=c(2,4)) # all plots on one page   
opts = c("p","l","o","b","c","s","S","h")   
for(i in 1:length(opts)){   
  heading = paste("type=",opts[i])   
  plot(x, y, type="n", main=heading)   
  lines(x, y, type=opts[i])   
}

[](https://www.statmethods.net/graphs/images/lines0.png) click to view

Next, we demonstrate each of the **type=** options when **plot( )** sets up the graph and ***does*** plot the points.

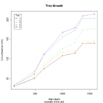
x <- c(1:5); y <- x # create some data  
par(pch=22, col="blue") # plotting symbol and color   
par(mfrow=c(2,4)) # all plots on one page   
opts = c("p","l","o","b","c","s","S","h")   
for(i in 1:length(opts){   
  heading = paste("type=",opts[i])   
  plot(x, y, main=heading)   
  lines(x, y, type=opts[i])   
}

[](https://www.statmethods.net/graphs/images/lines1.png) click to view

As you can see, the **type="c"** option only looks different from the **type="b"** option if the plotting of points is suppressed in the **plot( )** command.

To demonstrate the creation of a more complex line chart, let's plot the growth of 5 orange trees over time. Each tree will have its own distinctive line. The data come from the dataset **Orange**.

# Create Line Chart  
  
# convert factor to numeric for convenience   
Orange$Tree <- as.numeric(Orange$Tree)   
ntrees <- max(Orange$Tree)  
  
# get the range for the x and y axis   
xrange <- range(Orange$age)   
yrange <- range(Orange$circumference)   
  
# set up the plot   
plot(xrange, yrange, type="n", xlab="Age (days)",  
   ylab="Circumference (mm)" )   
colors <- rainbow(ntrees)   
linetype <- c(1:ntrees)   
plotchar <- seq(18,18+ntrees,1)  
  
# add lines   
for (i in 1:ntrees) {   
  tree <- subset(Orange, Tree==i)   
  lines(tree$age, tree$circumference, type="b", lwd=1.5,  
    lty=linetype[i], col=colors[i], pch=plotchar[i])   
}   
  
# add a title and subtitle   
title("Tree Growth", "example of line plot")  
  
# add a legend   
legend(xrange[1], yrange[2], 1:ntrees, cex=0.8, col=colors,  
   pch=plotchar, lty=linetype, title="Tree")

[](https://www.statmethods.net/graphs/images/linechart1.png) click to view

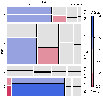
# Visualizing Categorical Data

The vcd package provides a variety of methods for visualizing multivariate categorical data, inspired by Michael Friendly's wonderful "Visualizing Categorical Data". Extended mosaic and association plots are described here. Each provides a method of visualizng complex data and evaluating deviations from a specified independence model. For more details, see The Strucplot Framework.

## Mosaic Plots

For extended mosaic plots, use **mosaic(x, condvar=, data=)** where **x** is a table or formula, **condvar=** is an optional conditioning variable, and **data=** specifies a data frame or a table. Include **shade=TRUE** to color the figure, and **legend=TRUE** to display a legend for the Pearson residuals.

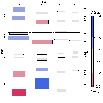
# Mosaic Plot Example  
library(vcd)  
mosaic(HairEyeColor, shade=TRUE, legend=TRUE)

[](https://www.statmethods.net/advgraphs/images/mosaic1.png) click to view

## Association Plots

To produce an extended association plot use **assoc(x, row\_vars, col\_vars)**where **x**is a contingency table, **row\_vars** is a vector of integers giving the indices of the variables to be used for the rows, and **col\_vars**is a vector of integers giving the indices of the variables to be used for the columns of the association plot.

# Association Plot Example  
library(vcd)  
assoc(HairEyeColor, shade=TRUE)

[](https://www.statmethods.net/advgraphs/images/assoc1.png) click to view

## Going Further

Both functions are complex and offer multiple input and output options. See **help(mosaic)**and **help(assoc)** for more details.

**Maps in R: Plotting data points on a map**

We will use a couple of datasets from the OpenFlight website for our examples.  
After loading the airports.dat file let's visualize the first few lines.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | > airports <- read.csv("http://openflights.svn.sourceforge.net/viewvc/openflights/openflights/data/airports.dat", header = FALSE)  > colnames(airports) <- c("ID", "name", "city", "country", "IATA\_FAA", "ICAO", "lat", "lon", "altitude", "timezone", "DST")  > head(airports)    ID                       name         city          country IATA\_FAA ICAO       lat      lon altitude timezone DST  1  1                     Goroka       Goroka Papua New Guinea      GKA AYGA -6.081689 145.3919     5282       10   U  2  2                     Madang       Madang Papua New Guinea      MAG AYMD -5.207083 145.7887       20       10   U  3  3                Mount Hagen  Mount Hagen Papua New Guinea      HGU AYMH -5.826789 144.2959     5388       10   U  4  4                     Nadzab       Nadzab Papua New Guinea      LAE AYNZ -6.569828 146.7262      239       10   U  5  5 Port Moresby Jacksons Intl Port Moresby Papua New Guinea      POM AYPY -9.443383 147.2200      146       10   U  6  6                 Wewak Intl        Wewak Papua New Guinea      WWK AYWK -3.583828 143.6692       19       10   U |

Latitude and longitude are reported for every airport in the dataset.  
Let's draw the map of Europe with the help of rworldmap package, as was shown in the previous post on maps:

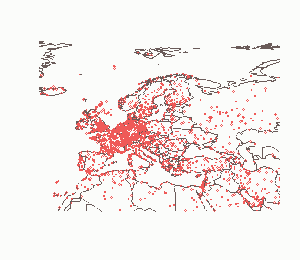


|  |  |
| --- | --- |
| 1  2  3  4 | > library(rworldmap)  > newmap <- getMap(resolution = "low")  > plot(newmap, xlim = c(-20, 59), ylim = c(35, 71), asp = 1) |

Then we can easily lay the airports over the map:



|  |  |
| --- | --- |
| 1  2 | > points(airports$lon, airports$lat, col = "red", cex = .6) |

*[](http://www.milanor.net/blog/wp-content/uploads/2012/12/airportsRWM.png)*

### Adding dimensions

In the introductory post I mentioned that ggmap actually builds on the ggplot graphics engine, thus all the strengths of ggplot are available when mapping data with ggmap.  
Here I will show a couple of examples on how to take advantage of this.

Let's load another dataset from OpenFlights in R.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | > routes <- read.csv("http://openflights.svn.sourceforge.net/viewvc/openflights/openflights/data/routes.dat", header=F)  > colnames(routes) <- c("airline", "airlineID", "sourceAirport", "sourceAirportID", "destinationAirport", "destinationAirportID", "codeshare", "stops", "equipment")  > head(routes)    airline airlineID sourceAirport sourceAirportID destinationAirport destinationAirportID codeshare stops equipment  1      2B       410           AER            2965                DME                 4029               0       CR2  2      2B       410           ASF            2966                LED                 2948               0       CR2  3      2B       410           CEK            2968                DME                 4029               0       CR2  4      2B       410           CEK            2968                KZN                 2990               0       CR2  5      2B       410           CEK            2968                OVB                 4078               0       CR2  6      2B       410           DME            4029                AER                 2965               0       CR2 |

Starting from the routes dataset, let's count the both number of routes departing from and arriving to a particular airport. I'm using another very useful package by Hadley Wickham for this task.



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | > library(plyr)  > departures <- ddply(routes, .(sourceAirportID), "nrow")  > names(departures)[2] <- "flights"  > arrivals <- ddply(routes, .(destinationAirportID), "nrow")  > names(arrivals)[2] <- "flights" |

Then, let's add the info on departing and arriving flights to the airports dataset (which contains the coordinates data.)



|  |  |
| --- | --- |
| 1  2  3 | > airportD <- merge(airports, departures, by.x = "ID", by.y = "sourceAirportID")  > airportA <- merge(airports, arrivals, by.x = "ID", by.y = "destinationAirportID") |

The goal is now to plot the airports on the map of Europe as circles whose area is proportional to the number of departing flights.

The first step is to get the map from GoogleMaps (or one of the other available services), like was shown last time.



|  |  |
| --- | --- |
| 1  2  3 | > library(ggmap)  > map <- get\_map(location = 'Europe', zoom = 4) |

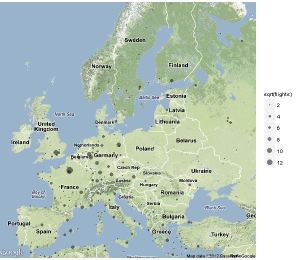
The following lines already get us quite close to producing the desired chart.



|  |  |
| --- | --- |
| 1  2  3 | > mapPoints <- ggmap(map) +  +   geom\_point(aes(x = lon, y = lat, size = sqrt(flights)), data = airportD, alpha = .5) |

The ggmap command prepares the drawing of the map. The geom\_point function adds the layer of data points, as would be normally done in a ggplot. A thorough explanation of ggplot is well beyond the scope of this post, but here are quick details on what is passed to geom\_point:  
- aes indicates how aesthetics (points in this case) are to be generated; the lon variable is associated to the x axis, lat to y, and the size of the points is proportional to the value of the variable flights (actually to its square root;)  
- data indicates the dataset where the variable passed to aes are to be found;  
- the alpha parameter controls the transparency of the plotted points (some degree of transparency will make the overlapping circles distinguishable.)

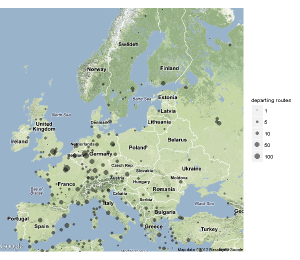
And here's what appears on the R plotting window when one types mapPoints in the console.

*[](http://www.milanor.net/blog/wp-content/uploads/2012/12/airportRoutes1.png)*

A few tweaks to the legend (so that it does report the actual number of departures rather than the square root,) and the chart is ready for publication.



|  |  |
| --- | --- |
| 1  2  3  4 | > mapPointsLegend <- mapPoints +  +   scale\_area(breaks = sqrt(c(1, 5, 10, 50, 100, 500)), labels = c(1, 5, 10, 50, 100, 500), name = "departing routes")  > mapPointsLegend |

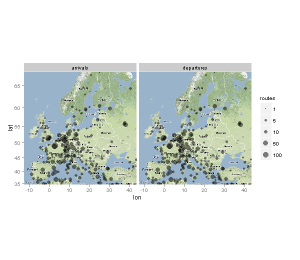
*[](http://www.milanor.net/blog/wp-content/uploads/2012/12/airportRoutes2.png)*

Once more, the map is a ggplot (type class(mapPoints) in your console to check) thus a nearly unlimited set of operations can be performed to improve it. For example, the number of departing flights could be portrayed by the color of the circles rather than their dimension.

As a final example for this post, I'll show the code to perform **faceting**. In other words we will have a couple of panels, one reporting the departing flights, the other the incoming ones.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | # create the data set containing both departures and arrivals  > airportD$type <- "departures"  > airportA$type <- "arrivals"  > airportDA <- rbind(airportD, airportA)    # map the data  # map + data points  > mapPointsDA <- ggmap(map) +  +   geom\_point(aes(x = lon, y = lat, size = sqrt(flights)), data = airportDA, alpha = .5)  # adjust the legend  > mapPointsLegendDA <- mapPointsDA +  +   scale\_area(breaks = sqrt(c(1, 5, 10, 50, 100, 500)), labels = c(1, 5, 10, 50, 100, 500), name = "routes")  # panels according to type (departure/arrival)  > mapPointsFacetsDA <- mapPointsLegendDA +  +   facet\_grid(. ~ type)  # plot the map  > mapPointsFacetsDA |

*[](http://www.milanor.net/blog/wp-content/uploads/2012/12/airportRoutes3.png)*