Data visualization

*The commonality between science and art is in trying to see profoundly - to develop strategies of seeing and showing.*

Edward Tuft, *The Visual Display of Quantitative Information*

**Learning objectives**

- Students completing this chapter will:
  - Understand the principles of the grammar of graphics;
  - Be able to use ggvis to create common business graphics;
  - Be able to select data from a database for graphic creation;
  - Be able to depict locations on a Google map.

**Visual processing**

Humans are particularly skilled at processing visual information because it is an innate capability, compared to reading which is a learned skill. When we evolved on the Savannah of Africa, we had to be adept at processing visual information (e.g., recognizing danger) and deciding on the right course of action (fight or flight). Our ancestors were those who were efficient visual processors and quickly detected threats and used this information to make effective decisions. They selected actions that led to survival. Those who were inefficient visual information processors did not survive to reproduce. Even those with good visual skills failed to propagate if they made poor decisions relative to detected dangers. Consequently, we should seek opportunities to present information visually and play to our evolved strength. As people vary in their preference for visual and textual information, it often makes sense to support both types of reporting.

In order to learn how to visualize data, you need to become familiar with the grammar of graphics and ggvis (an R extension for graphics). In line with the learning of data modeling and SQL, we will take an intertwined spiral approach. First we will tackle the grammar of graphics (the abstract) and then move to ggvis (the concrete). You will also learn how to take the output of an SQL query and feed it directly into ggvis. The end result will be a comprehensive set of practical skills for data visualization.

**The grammar of graphics**

A grammar is a system of rules for generating valid statements in a language. A grammar makes it possible for people to communicate accurately and concisely. English has a rather complex grammar, as do most languages. In contrast, computer-related languages, such as SQL, have a relatively simple grammar because they are carefully designed for expressive power, consistency, and ease of use. Each of the dialects of data modeling also has a grammar, and each of these grammars is quite similar in that they all have the same foundational elements: entities, attributes, identifiers, and relationships.
A grammar has been designed by Wilkinson\textsuperscript{1} for creating graphics to enhance their expressiveness and comprehensiveness. From a mathematical perspective, a graph is a set of points. A graphic is a physical representation of a graph. Thus, a graph can have many physical representations, and one of the skills you need to gain is to be able to judge what is a meaningful graphic for your clients. A grammar for graphics provides you with many ways of creating a graphic, just as the grammar of English gives you many ways of writing a sentence. Of course, we differ in our ability to convey information in English. Similarly, we also differ in our skills in representing a graph in a visual format. The aesthetic attributes of a graph determine its ability to convey information. For a graphic, aesthetics are specified by elements such as size and color. One of the most applauded graphics is Minard’s drawing in 1861 of Napoleon’s advance on and retreat from Russia during 1812. The dominating aspect of the graphic is the dwindling size of the French army as it battled winter, disease, and the Russian army. The graph shows the size of the army, its location, and the direction of its movement. The temperature during the retreat is drawn at the bottom of graphic.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{minard.png}
\caption{Charles Joseph Minard’s graphic of Napoleon’s Russian expedition in 1812}
\end{figure}

Wilkinson’s grammar is based on six elements:

1. \textit{Data}: a set of data operations that creates variables from datasets;
2. \textit{Trans}: variable transformations;
3. \textit{Scale}: scale transformations;
4. \textit{Coord}: a coordinate system;
5. \textit{Element}: a graph and its aesthetic attributes;
6. \textit{Guide}: one or more guides.

Interest in the grammar of data visualization has increased in recent years because of the growth in data. There is ongoing search to find ways to reveal clearly the information in large data sets. Vega\(^2\) is a recent formulation building on Wilkinson's work. In Vega, a visualization consists of basic properties of a graph (such as the width and height) and definitions of the data to visualize, scales for mapping to data to a visual form, axes for representing scales, and graphic marks (such as rectangles, circles, and arcs) for displaying the data.

**ggvis**

ggvis is an implementation of Vega in R. Because it is based on a grammar, ggvis is a very powerful graphics tool for creating both simple and complex graphs. It is well-suited for generating multi-layered graphs because of its grammatical foundation. As a result, using ggvis you specify a series of steps (think of them as sentences) to create a graphic (think of it as a paragraph) to visually convey your message. ggvis is a descendant of ggplot2 and adds new features to support interactive visualization using shiny, another R package, which we will cover later.

**Data**

Most structured data, which is what we require for graphing, are typically stored in spreadsheets or databases. In the prior chapter introducing R, you learned how to read a spreadsheet exported as a CSV file and access a database. These are also the skills you need for reading a file containing data to be visualized.

**Transformation**

A transformation converts data into a format suitable for the intended visualization. In this case, we want to depict the relative change in carbon levels since pre-industrial periods, when the value was 280 ppm. Here are sample R commands.

```r
# compute a new column in carbon containing the relative change in CO2
carbon$relCO2 = (carbon$CO2-280)/280
```

There are many ways that you might want to transform data. The preceding example just illustrates the general nature of a transformation. You can also think of SQL as a transformation process as it can compute new values from existing columns.

**Coord**

A coordinate system describes where things are located. A geopositioning system (GPS) reading of latitude and longitude describes where you are on the surface of the globe. It is typically layered onto a map so you can see where you are relative to your current location. Most graphs are plotted on a two-dimensional (2D) grid with x (horizontal) and y (vertical) coordinates. ggvis currently supports six 2D coordinate systems, as shown in the following table. The default coordinate system for most packages is Cartesian.

---

\(^2\) [http://trifacta.github.io/vega/](http://trifacta.github.io/vega/)
**Coordinate systems**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cartesian</td>
<td>Cartesian coordinates</td>
</tr>
<tr>
<td>equal</td>
<td>Equal scale Cartesian coordinates</td>
</tr>
<tr>
<td>flip</td>
<td>Flipped Cartesian coordinates</td>
</tr>
<tr>
<td>trans</td>
<td>Transformed Cartesian coordinates</td>
</tr>
<tr>
<td>map</td>
<td>Map projections</td>
</tr>
<tr>
<td>polar</td>
<td>Polar coordinates</td>
</tr>
</tbody>
</table>

**Element**

An element is a graph and its attributes. Let’s start with a simple scatterplot of year against CO₂ emissions. We do this in two steps applying the ggvis approach of building a graphic by adding layers. ggvis uses the pipe function, %>%%, to specify a linear sequence of data processing. The output of each process is the input to the next.

- The foundation is the ggvis function, which identifies the source of the data and what is to be plotted. In the following example, the file carbon is fed into the ggvis function, which selects year and CO2 as the x and y, respectively, dimensions of the graph.
- A graph consists of a number of layers, and in the following example the points layer receives the x and y values from ggvis and plots each pair of points with a red dot.

```r
library(ggvis)
url <- 'http://people.terry.uga.edu/rwatson/data/carbon.txt'
carbon <- read.table(url, header=T, sep=',,')
# Select year(x) and CO2(y) to create a x-y point plot
# Specify red points, as you find that aesthetically pleasing
carbon %>% ggvis(~year,~CO2) %>% layer_points(fill:='red')
# Notice how `%>%` is used for creating a pipeline of commands
```
Graphics appear under the Plots tab of the bottom right window. You can select the Export tab to save a graphic to a file or the clipboard. Also, notice that you can use left and right arrows of the Plots tab to cycle through graphics created in a session.

**Scale**

Scales control the visualization of data. It is usually a good idea to have a zero point for the y axis so you don't distort the slope of the line. In the following example, the `scale_numeric` function is specified a zero point for the y axis with the command `zero=T`.

```r
carbon %>% ggvis(~year,~CO2) %>% layer_points(fill:='red') %>%
  scale_numeric('y',zero=T)
```
**Axes and legends**

Axes and legends are both forms of guides, which help the reader to understand a graphic. Readability enhancers such as axes, title, and tick marks are generated automatically based on the parameters specified in the ggvis command. You can override the defaults.

Let’s redo the graphic with the relative change since the beginning of industrialization in the mid 18th century, when the level of CO2 was around 280 ppm. This time, we will create a line plot. We also add some titles for the axes and specify a format of four consecutive digits for displaying a year on the x-axis. We also move or offset the title for the y-axis a bit to the left to improve readability.

```r
# compute a new column containing the relative change in CO2
carbon$relCO2 = (carbon$CO2-280)/280
carbon %>% ggvis(~year,~relCO2) %>% layer_lines(stroke:='blue') %>%
  scale_numeric('y',zero=T) %>%
  add_axis('y', title = "CO2 ppm of the atmosphere", title_offset=50) %>%
  add_axis('x', title = 'Year', format = '####')
```
As the prior graphic shows, the present level of CO2 in the atmosphere is now above 40 percent higher than the pre-industrial period and is continuing to grow.

**Assignment function**

You will have seen three types of assignment symbols in the preceding examples. The notion of scaling helps in understanding the difference between a value and a property. A property is fixed. So not matter how you resize the preceding graph, the stroke will always be blue. Whereas, a value, such as the title, is scalable. The difference between a value and a property can seem confusing, so don’t agonize over it.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>~</td>
<td>Data assignment</td>
<td>y ~ CO2</td>
</tr>
<tr>
<td>=</td>
<td>Set a value</td>
<td>title = ‘year’</td>
</tr>
<tr>
<td>:=</td>
<td>Set a property</td>
<td>stroke := ‘blue’</td>
</tr>
</tbody>
</table>

**Guides**

Axes and legends are both forms of guides, which help the reader to understand a graphic. In ggvis, legends and axes are generated automatically based on the parameters specified in the ggvis command. You have the capability to override the defaults for axes, but the legend is quite fixed in its format.

In the following graphic, for each axis there is a label and there are tick marks with values so that you eyeball a point on the graphic to its x and y values. The legend enables you to determine
which color, in this case, matches each year. A legend could also use shape (e.g., a diamond) and shape size to aid matching.

![Graph showing population growth](image)

**Skill builder**

Create a line plot using the data in the following table.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Population (billions)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

**Some recipes**

Learning the full power of ggvis is quite an undertaking, so here are a few recipes for visualizing data.

**Histogram**

Histograms are useful for showing the distribution of values in a single column. A histogram has a vertical orientation. The number of occurrences of a particular value in a column are automatically counted by the ggvis function. In the following sample code, we show the distribution of temperatures in Celsius for the Central Park temperature data. Notice that the Celsius conversion is rounded to an integer for plotting. Width specifies the size of the bin, which is two in this case. This means that the bin above the tick mark 10 contains all values in the range 9 to 11. There is online a list of names of colors you can use in R.³

```r
library(ggvis)
url <- 'http://people.terry.uga.edu/rwatson/data/centralparktemps.txt'
t <- read.table(url, header=T, sep=',')
t$C <- round((t$temperature - 32)*5/9,1)
t %>% ggvis(~C) %>% layer_histograms(width = 2, fill='cornflowerblue') %>%
    add_axis('x', title='Celsius') %>%
    add_axis('y', title='Frequency')
```
**Bar graph**

In the following example, we query a database to get data for plotting. Because the column selected for graphing, productLine is categorical, we need to use a bar graph.

```r
classic <- library(ggvis)
classic <- library(RMySQL)
classic <- conn <- dbConnect(RMySQL::MySQL(), "richardtwatson.com", 
dbname="ClassicModels", user="db1", password="student")
classic # Query the database and create file for use with R
classic d <- dbGetQuery(conn,"SELECT productLine from Products;")
classic # Plot the number of product lines by specifying the appropriate column name
classic d %>% ggvis(~productLine) %>% layer_bars(fill='chocolate')
classic add_axis('x',title='Product line') %>%
classic add_axis('y',title='Count')
```

![Bar graph image]

**Skill builder**

Create a bar graph using the data in the following table. Set population as the weight for each observation.

<table>
<thead>
<tr>
<th>Product line</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classic Cars</td>
<td>35</td>
</tr>
<tr>
<td>Motorcycles</td>
<td>20</td>
</tr>
<tr>
<td>Planes</td>
<td>15</td>
</tr>
<tr>
<td>Ships</td>
<td>10</td>
</tr>
<tr>
<td>Trains</td>
<td>5</td>
</tr>
<tr>
<td>Trucks and Buses</td>
<td>10</td>
</tr>
<tr>
<td>Vintage Cars</td>
<td>25</td>
</tr>
</tbody>
</table>
A scatterplot shows points on an x-y grid, which means you need to have an x and y with numeric values.

```r
library(ggvis)
library(RMySQL)
conn <- dbConnect(RMySQL::MySQL(), "richardtwatson.com", 
dbname="ClassicModels", user="db1", password="student")
# Get the monthly value of orders
d <- dbGetQuery(conn,"SELECT MONTH(orderDate) AS orderMonth, 
sum(quantityOrdered*priceEach) AS orderValue FROM Orders, OrderDetails WHERE 
Orders.orderNumber = OrderDetails.orderNumber GROUP BY orderMonth;")
# Plot data orders by month
# Show the points and the line
d %>% ggvis(~orderMonth, ~orderValue/1000000) %>%
  layer_lines(stroke:='blue') %>%
  layer_points(fill:='red') %>%
  add_axis('x', title = 'Month') %>%
  add_axis('y', title='Order value (millions)', title_offset=30)
```
It is sometimes helpful to show multiple scatterplots on the one grid. In ggvis, you can create groups of points for plotting. Sometimes, you might find it convenient to recode data so you can use different colors or lines to distinguish values in set categories. Let's examine grouping by year.

```r
library(ggvis)
library(RMySQL)
conn <- dbConnect(RMySQL::MySQL(), "richardtwatson.com", 
dbname="ClassicModels", user="db1", password="student")
d <- dbGetQuery(conn,"SELECT YEAR(orderDate) AS orderYear, MONTH(orderDate) AS Month, sum((quantityOrdered*priceEach)) AS Value FROM Orders, OrderDetails 
WHERE Orders.orderNumber = OrderDetails.orderNumber GROUP BY orderYear, Month;")
# Plot data orders by month and display by year
# ggvis expects grouping variables to be a factor, so convert
d$Year <- as.factor(d$orderYear)
d %>% group_by(Year) %>% ggvis(~Month,~Value/1000, stroke = ~Year) %>%
  layer_lines() %>%
  add_axis('x', title = 'Month') %>%
  add_axis('y',title='Order value (thousands)', title_offset=50)
```
Because ggvis is based on a grammar of graphics, with a few changes, you can create a bar graph of the same data.

\[
\text{d} \gg \text{group_by(Year)} \gg \text{ggvis( } \sim \text{Month, } \sim \text{Value/100000, fill = } \sim \text{Year) } \gg \text{layer_bars() } \gg
\]

\[
\text{add_axis('x', title = 'Month') } \gg \text{add_axis('y', title = 'Order value (thousands)', title_offset = 50)}
\]
Sometimes the data that you want to graph will be in multiple files. Here is a case where we want to show ClassicCars orders and payments by month on the same plot. Notice that the packages sqlfd and RMySQL can conflict when both are loaded, so we add an option statement to avoid this problem.

```r
library(ggvis)
library(sqldf)
library(RMySQL)
options(sqldf.driver = "SQLite") # to avoid conflict with RMySQL

# Load the driver
conn <- dbConnect(RMySQL::MySQL(), "richardtwatson.com",
                   dbname="ClassicModels", user="db1", password="student")

orders <- dbGetQuery(conn,"SELECT 'Orders' as Category, MONTH(orderDate) AS month,
                       sum((quantityOrdered*priceEach)) AS value FROM Orders, OrderDetails
                       WHERE Orders.orderNumber = OrderDetails.orderNumber and YEAR(orderDate) = 2004
                       GROUP BY Month;"

payments <- dbGetQuery(conn,"SELECT 'Payments' as Category,
                       MONTH(paymentDate) AS month, SUM(amount) AS value FROM Payments WHERE
                       YEAR(paymentDate) = 2004 GROUP BY MONTH;"

# concatenate the two files
```
Smoothing helps to detect a trend in a line plot. The following example shows the average mean temperatures for August for Central Park.

```r
library(sqldf)
options(sqldf.driver = "SQLite") # to avoid conflict with RMySQL
url <- "http://people.terry.uga.edu/rwatson/data/centralparktemps.txt"
```
**Skill builder**

Using the Central Park data, plot the temperatures for February and fit a straight line to the points. Use `layer_model_predictions()`. What do you observe?
**Box Plot**

A box plot is an effective means of displaying information about one or more variables. It shows minimum and maximum, range, lower quartile, median, and upper quartile for each variable. The following code creates a box plot for a single variable. Notice that we use `factor(0)` to indicate there is no grouping variable.

```r
library(ggvis)
library(RMySQL)
options(sqldf.driver = "SQLite") # to avoid conflict with RMySQL
conn <- dbConnect(RMySQL::MySQL(), "richardtwatson.com", dbname="ClassicModels", user="db1", password="student")
d <- dbGetQuery(conn,"SELECT amount from Payments;")
# Boxplot of amounts paid
d %>% ggvis(~factor(0),~amount) %>% layer_boxplots() %>%
  add_axis('x',title='Checks') %>%
  add_axis('y',title='')
```

![Box Plot Diagram](image_url)
The following example shows a box plot for each month, which is created by specifying month as a factor.

```r
library(ggvis)
library(RMySQL)
options(sql_dflf.driver = "SQLite") # to avoid conflict with RMySQL
conn <- dbConnect(RMySQL::MySQL(), "richardtwatson.com",
dbname="ClassicModels", user="db1", password="student")
d <- dbGetQuery(conn,"SELECT month(paymentDate) as month, amount from Payments;"

# Boxplot of amounts paid

d %>% ggvis(~factor(month),~amount) %>% layer_boxplots()
```
**Heat map**

A heat map visualizes tabular data. It is useful when you have two categorical variables cross tabulated. Consider the case for the ClassicModels database where we want to get an idea of the different number of model scales in each product line. We can get a quick picture with the following code.

```r
library(ggvis)
library(RMySQL)
options(sqldf.driver = "SQLite") # to avoid conflict with RMySQL
# Load the driver
conn <- dbConnect(RMySQL::MySQL(), "richardtwatson.com",
    dbname="ClassicModels", user="db1", password="student")
d <- dbGetQuery(conn,'SELECT count(*) as Frequency, productLine as Line,
    productScale as Scale from Products group by productLine, productScale')
d %>% ggvis(~Scale, ~Line, fill=~Frequency) %>%
  layer_rects(width = band(), height = band()) %>%
  layer_text(text=~Frequency, stroke='white', align='left',
    baseline='top') # add frequency to each cell
```
Interactive graphics

The ggvis package incorporates features of shiny, an R package, that enable you to create interactive graphics. For example, you might want to let the viewer select the color and width of a line on a graphic. Many of the interactive controls, shown in the following table, should be familiar as the various options are often used for web forms.
Interactive controls for ggvis

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>input_checkbox()</td>
<td>Check one or more boxes</td>
</tr>
<tr>
<td>input_checkboxgroup()</td>
<td>A group of checkboxes</td>
</tr>
<tr>
<td>input_numeric()</td>
<td>A spin box</td>
</tr>
<tr>
<td>input_radiobuttons()</td>
<td>Pick one from a set of options</td>
</tr>
<tr>
<td>input_select()</td>
<td>Select from a drop-down text box</td>
</tr>
<tr>
<td>input_slider()</td>
<td>Select using a slider</td>
</tr>
<tr>
<td>input_text()</td>
<td>Input text</td>
</tr>
</tbody>
</table>

**Selecting a property using a drop-down list**

The following example uses the `input_select` function to present the viewer with an option of one of three colors (red, green, or blue) for the stroke of the graph. When you execute the code, you will see a selection list on the left bottom. Because the graph is interactive, you will need to click on the stop button (top right of the Viewer window) before running anymore R commands.

```r
carbon$relCO2 = (carbon$CO2-280)/280
carbon %>% ggvis(~year,~relCO2) %>%
  layer_lines(stroke:=input_select(c("red", "green", "blue"))) %>%
  scale_numeric('y',zero=T) %>%
  add_axis('y', title = "CO2 ppm of the atmosphere", title_offset=50) %>%
  add_axis('x', title = 'Year', format='####')
```

**Skill builder**

Create a point plot using the data in the following table. Give the viewer a choice of three colors and three shapes (square, cross, or diamond) for the points.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Population (billions)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

**Selecting a numeric value with a slider**

A slider enables the viewer to select a numeric value in a range. In the following code, a slider is set up for values between 1 to 5, inclusively. Note in this case, the stroke width slider and color selector are defined outside the ggvis code. This is a useful technique for writing code that makes it possible for some chunks to be easily reused.

```r
carbon$relCO2 = (carbon$CO2-280)/280
slider <- input_slider(1, 5, label = "Width")
```
Using dplyr with ggvis

The dplyr package\(^4\) provides functions for efficiently manipulating data frames. It is designed to work with ggvis. In particular, you can use it with the pipe function ( %>% ) to transform input data or control which data are plotted.

Some dplyr functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter()</td>
<td>Select rows</td>
</tr>
<tr>
<td>select()</td>
<td>Select columns</td>
</tr>
<tr>
<td>arrange()</td>
<td>Sort rows</td>
</tr>
<tr>
<td>mutate()</td>
<td>Add new columns</td>
</tr>
<tr>
<td>summarize()</td>
<td>Compute summary statistics</td>
</tr>
</tbody>
</table>

Here are some examples using dplyr with the Central Park data. In each case, the SQL equivalent is shown before the dplyr function,

```
library(sqldf)
library(dplyr)
options(sqldf.driver = "SQLite") # to avoid conflict with RMySQL
url <- 'http://people.terry.uga.edu/rwatson/data/centralparktemps.txt'
t <- read.table(url, header=T, sep=',',)

# filter
sqldf("select * from t where year = 1999")
filter(t,year==1999)

# select
sqldf("select temperature from t")
select(t,temperature)

# a combination of filter and select
sqldf("select * from t where year > 1989 and year < 2000")
select(t,year, month, temperature) %>% filter(year > 1989 & year < 2000)

# arrange
sqldf("select * from t order by year desc, month")
arrange(t, desc(year),month)

# mutate -- create a new column
```

\(^4\) [http://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html](http://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html)
t_SQL <- sqldf("select year, month, temperature, (temperature-32)*5/9 as CTemp from t")
t_dplyr <- mutate(t,CTemp = (temperature-32)*5/9)
# summarize
sqldf("select avg(temperature) from t")
summarize(t,mean(temperature))

We can see the value of dplyr and the pipe function in the following example. In this case, we want to give the viewer the choice of which month’s temperatures to plot. We use a slider to select a month in the range 1 to 12, and then use dplyr to filter on that month. You can’t use sqldf to select the specified month because it is not designed to work with the pipe function.

library(ggvis)
library(dplyr)
url <- 'http://people.terry.uga.edu/rwatson/data/centralparktemps.txt'
t <- read.table(url, header=T, sep=',')
slider <- input_slider(1, 12,label="Month")
t %>%
  ggvis(~year,~temperature) %>%
  filter(month == eval(slider)) %>%
  layer_points() %>%
  add_axis('y', title = "Temperature", title_offset=50) %>%
  add_axis('x', title = 'Year', format='####')

Skill builder
Using the Central Park data, plot the temperatures for a selected year using a slider.

Geographic data
The ggmap package supports a variety of mapping systems, including Google maps. As you might expect, it offers many features, and we just touch on the basics in this example.

The Offices table in the Classic Models database includes the latitude and longitude of each office in officeLocation, which has a datatype of POINT. The following R code can be used to mark each office on a Google map. After loading the required packages, a database query is executed to return the longitude and latitude for each office. Then, a Google map of the United States is displayed. The marker parameter specifies the name of the table containing the values for longitude and latitude. Offices that are not in the U.S. (e.g., Sydney) are ignored. Adjust zoom, an integer, to get a suitably sized map. Zoom can range from 3 (a continent) to 20 (a building).

library(ggplot)
library(ggmap)
library(mapproj)
library(RMySQL)
options(sqldf.driver = "SQLite") # to avoid conflict with RMySQL
# connect to the database
conn <- dbConnect(RMySQL::MySQL(), "richardtwatson.com", 
dbname="ClassicModels", user="db1", password="student")

# Google maps requires lon and lat, in that order, to create markers

d <- dbGetQuery(conn,"SELECT y(officeLocation) AS lon, x(officeLocation) AS lat FROM Offices;")

# show offices in the United States
# vary zoom to change the size of the map

map <- get_googlemap('united states',marker=d,zoom=4)
ggmap(map) + labs(x = 'Longitude', y = 'Latitude') + ggtitle('US offices')

Skill builder

Create the following maps.

1. Offices in Europe
2. Office location in Paris
3. Customers in the US
4. Customers in Sydney
R resources

The developer of ggvis, Hadley Wickham, maintains a [web site](#). Another useful web site for data visualization is [FlowingData](#), which has an associated book, titled *Visualize This*. If you become heavily involved in visualization, we suggest you read some of the works of Edward Tufte. One of his books is listed in the references.

Summary

Because of evolutionary pressure, humans became strong visual processors. Consequently, graphics can be very effective for presenting information. The grammar of graphics provides a logical foundation for the construction of a graphic by adding successive layers of information. ggvis is a package implementing the grammar of graphics in R, the open source statistics platform. Data can be extracted from an MySQL database for processing and graphical presentation in R. Spatial data can be selected from a database and displayed on a Google map.

Key terms and concepts

- Bar chart
- Box plot
- dplyr
- ggvis
- Grammar of graphics
- Graph
- Graphic
- Heat map
- Histogram
- Scatterplot
- Smooth

References


Exercises

1. Given the following data on the world’s cargo carrying capacity in millions of dead weight tons (dwt)\(^3\) for the given years, graph the relationship as a point plot. Add a linear prediction model to the graph. What is the approximate increase in the world capacity each year?

\[
\text{dwt} \leftarrow \text{c}(2566, 3704, 4008, 5984, 7700, 8034, 8229, 7858, 8408, 8939) \\
\]

2. Create a bar graph showing the relative ocean shipping cost as a percentage of total cost for some common household items using the following data.

\[
\text{unitCost} \leftarrow \text{c}(700, 200, 150, 50, 15, 3, 1) \\
\text{shipCost} \leftarrow \text{c}(10.00, 1.50, 1.00, 0.15, 0.15, 0.05, 0.01) \\
\text{item} \leftarrow \text{c}(\text{'TV set'}, \text{'DVD player'}, \text{'Vacuum cleaner'}, \text{'Scotch whisky'}, \text{'Coffee'}, \text{'Biscuits'}, \text{'Beer'})
\]

SQL input

3. Visualize in blue the number of items for each product scale.

4. Prepare a line plot with appropriate labels for total payments for each month in 2004.

5. Create a histogram with appropriate labels for the value of orders received from the Nordic countries (Denmark, Finland, Norway, Sweden).

6. Create a heatmap for product lines and Norwegian cities.

7. Show on a Google map the customers in Japan.

8. Show on a Google map the European customers who have never placed an order.

File input


   a. Create bar chart for sales by each form of sale (online or auction).

10. Access [http://people.terry.uga.edu/rwatson/data/electricityprices2010-12.csv](http://people.terry.uga.edu/rwatson/data/electricityprices2010-12.csv), which contains details of electricity prices for a major city.\(^6\) Do a Box plot of cost. What do you conclude about cost?

11. Read the table [http://people.terry.uga.edu/rwatson/data/wealth.csv](http://people.terry.uga.edu/rwatson/data/wealth.csv) containing details of the wealth of various countries. Create histograms for each of the wealth measures. Consult the R color chart\(^7\) of a choice of colors.

---

5 Dead weight tonnage is a measure of how much weight, in tonnes, a ship can safely carry.

6 Note these prices have been transformed from the original values, but are still representative of the changes over time.


   a. Compute the correlation between temperature and electricity price. What do you conclude?

   b. Graph the relationship between temperature and electricity price.

   c. Graph the relationship between temperature and electricity price when the temperature is 95ºF and above.

   d. Create a single graph showing the relationship between temperature and electricity price differentiating by color when the price is above or below 90ºF. (Hint: Trying recoding temperature).

13. Prepare a bar chart and radar plot showing the various forms of fuel used in the US for the most recent month for which data are available. Download www.eia.gov/totalenergy/data/monthly/query/mer_data_csv.asp?table=T01.02