

# Using the Google Visualisation API with R

by Markus Gesmann and Diego de Castillo

**Abstract** The `googleVis` package provides an interface between R and the Google Visualisation API to create interactive charts which can be embedded into web pages. The best known of these charts is probably the Motion Chart, popularised by Hans Rosling in his TED talks. With the `googleVis` package users can easily create web pages with interactive charts based on R data frames and display them either via the local R HTTP help server or within their own sites.

## Motivation

In 2006 Hans Rosling gave an inspiring talk at TED<sup>1</sup> about social and economic developments in the world over the past 50 years, which challenged the views and perceptions of many listeners. Rosling had used extensive data analysis to reach his conclusions. To visualise his talk, he and his colleagues at `Gapminder` had developed animated bubble charts, see Figure 1.

Rosling's presentation popularised the idea and use of interactive charts, and as a result the software behind `Gapminder` was bought by Google and integrated as motion charts into their `Visualisation API`<sup>2</sup> one year later.

In 2010 Sebastián Pérez Saaibi (Saaibi, 2010) presented at the R/Rmetrics Workshop on Computational Finance and Financial Engineering the idea to use Google motion charts to visualise R output with the `rsp` package (Bengtsson, 2011).

Inspired by those talks and the desire to use interactive data visualisation tools to foster the dialogue between data analysts and others, the authors of this article started the development of the `googleVis` package (Gesmann and de Castillo, 2011).

## Google Visualisation API

The Google Visualisation API allows users to create interactive charts as part of Google documents, spreadsheets and web pages. This text will focus on the usage of the API as part of web sites.

The `Google Public Data Explorer` (<http://www.google.com/publicdata/home>) provides a good example, demonstrating the use of interactive charts and how they can help to analyse data. The charting data can either be embedded into the html file or read dynamically. The key to the Google Visualisation API

is that the data is structured in a “`DataTable`”, and this is where the `googleVis` package helps. It uses the functionality of the `rjsonio` package (Temple Lang, 2011) to transform R data frames into `JSON`<sup>3</sup> objects as the basis for a `DataTable`.

As an example, we will look at the html code of a motion chart from Google's visualisation gallery, which generates output similar to Figure 1:

```
1 <html>
2 <head>
3   <script type="text/javascript"
4     src="http://www.google.com/jsapi">
5   </script>
6   <script type="text/javascript">
7     google.load('visualization', '1',
8       {'packages':['motionchart']});
9     google.setOnLoadCallback(drawChart);
10    function drawChart() {
11      var data=new google.visualization.DataTable();
12      data.addColumn('string', 'Fruit');
13      data.addColumn('date', 'Date');
14      data.addColumn('number', 'Sales');
15      data.addColumn('number', 'Expenses');
16      data.addColumn('string', 'Location');
17      data.addRows([
18        ['Apples',new Date(1988,0,1),1000,300,'East'],
19        ['Oranges',new Date(1988,0,1),1150,200,'West'],
20        ['Bananas',new Date(1988,0,1),300,250,'West'],
21        ['Apples',new Date(1989,6,1),1200,400,'East'],
22        ['Oranges',new Date(1989,6,1),750,150,'West'],
23        ['Bananas',new Date(1989,6,1),788,617,'West']
24      ]);
25      var chart=new google.visualization.MotionChart(
26        document.getElementById('chart_div'));
27      chart.draw(data, {width: 600, height:300});
28    }
29  </script>
30 </head>
31 <body>
32   <div id="chart_div"
33     style="width:600px; height:300px;">
34   </div>
35 </body>
36 </html>
```

The code and data are processed and rendered in the browser and is not submitted to any server<sup>4</sup>.

You will notice that the above html code has five generic parts<sup>5</sup>:

- references to Google's AJAX (l. 4) and Visualisation API (ll. 7–8),
- data to visualise as a `DataTable` (ll. 11–24),
- an instance call to create the chart (ll. 25–26),

<sup>1</sup>[http://www.ted.com/talks/hans\\_rosling\\_shows\\_the\\_best\\_stats\\_you\\_ve\\_ever\\_seen.html](http://www.ted.com/talks/hans_rosling_shows_the_best_stats_you_ve_ever_seen.html)

<sup>2</sup><http://code.google.com/apis/visualization/documentation/index.html>

<sup>3</sup><http://www.json.org/>

<sup>4</sup>[http://code.google.com/apis/visualization/documentation/gallery/motionchart.html#Data\\_Policy](http://code.google.com/apis/visualization/documentation/gallery/motionchart.html#Data_Policy)

<sup>5</sup>For more details see [http://code.google.com/apis/chart/interactive/docs/adding\\_charts.html](http://code.google.com/apis/chart/interactive/docs/adding_charts.html)

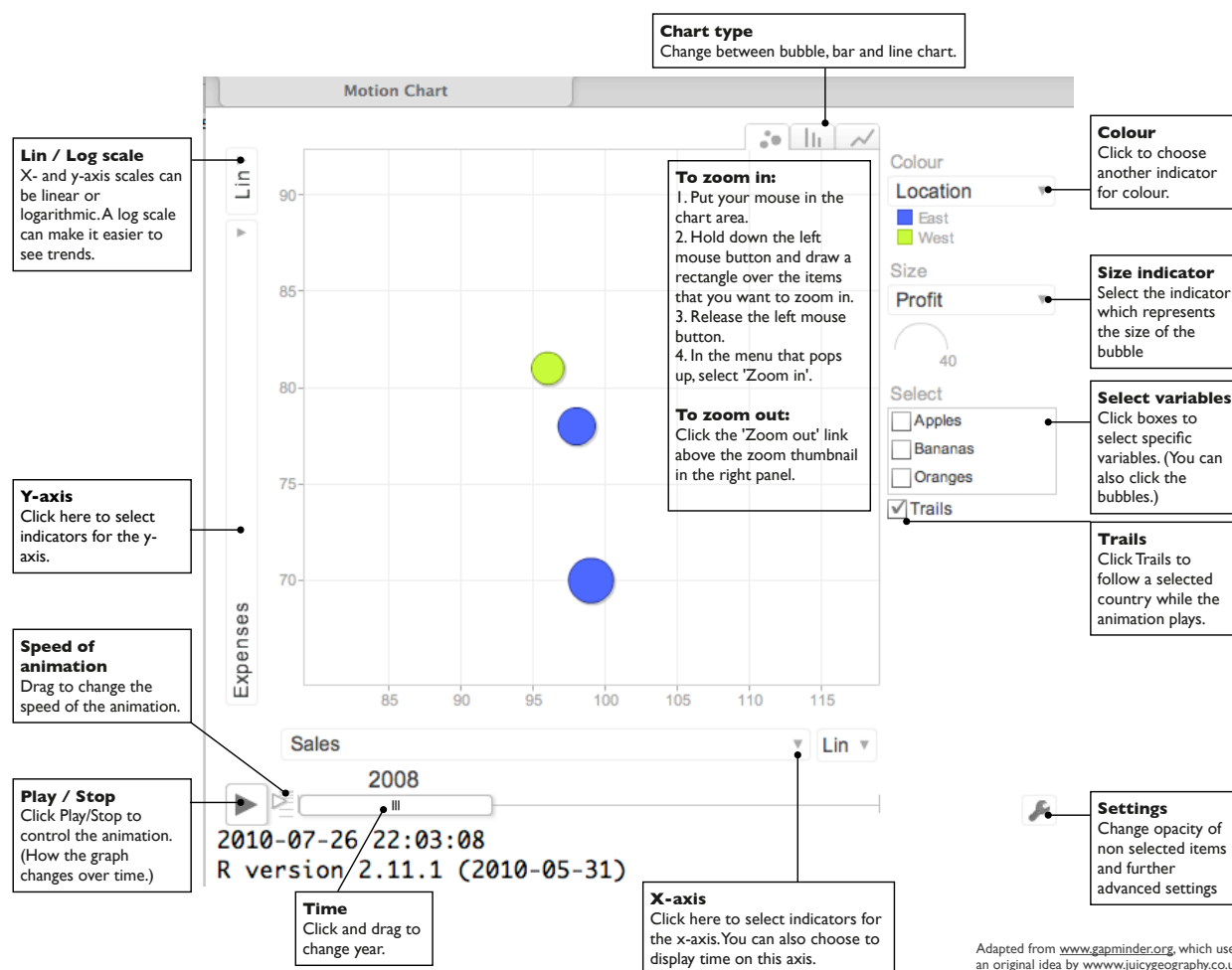


Figure 1: Overview of a Google Motion Chart. Screenshot of the output of `plot(gvisMotionChart(Fruits, idvar = 'Fruit', timevar = 'Year'))`

- a method call to draw the chart including options, shown here as width and height (l. 27),
- an HTML `<div>` element to add the chart to the page (ll. 32–34).

These principles hold true for most of the interactive charts of the Google Visualisation API.

However, before you use the API you should read the [Google Visualisation API Terms of Service](#)<sup>6</sup> and the [Google Maps/Google Earth APIs Terms of Service](#)<sup>7</sup>.

## The googleVis package

The **googleVis** package provides an interface between R and the Google Visualisation API. The functions of the package allow the user to visualise data stored in R data frames with the Google Visualisation API.

Version (0.2.12) of the package provides interfaces to Motion Charts, Annotated Time Lines, Geo Maps,

<sup>6</sup><http://code.google.com/apis/visualization/terms.html>

<sup>7</sup><http://code.google.com/apis/maps/terms.html>

Maps, Geo Charts, Intensity Maps, Tables, Gauges, and Tree Maps, as well as Line-, Bar-, Column-, Area-, Combo-, Scatter-, Candlestick-, Pie- and Org Charts; see Figure 2 for some examples.

The output of a **googleVis** function is html code that contains the data and references to JavaScript functions hosted by Google. A browser with an Internet connection is required to view the output, and for Motion Charts, Geo Maps and Annotated Time Lines also Flash. The actual chart is rendered in the browser.

Please note that Flash charts may not work when loaded as a local file due to security settings, and therefore may require to be displayed via a web server. Fortunately, R comes with an internal HTTP server which allows the **googleVis** package to display pages locally. Other options are to use the **R.rsp** package or **Rapache** (Horner, 2011) with **brew** (Horner, 2011). Both **R.rsp** and **brew** have the capability to extract and execute R code from html code, similar to the approach taken by Sweave (Leisch, 2002) for L<sup>A</sup>T<sub>E</sub>X.

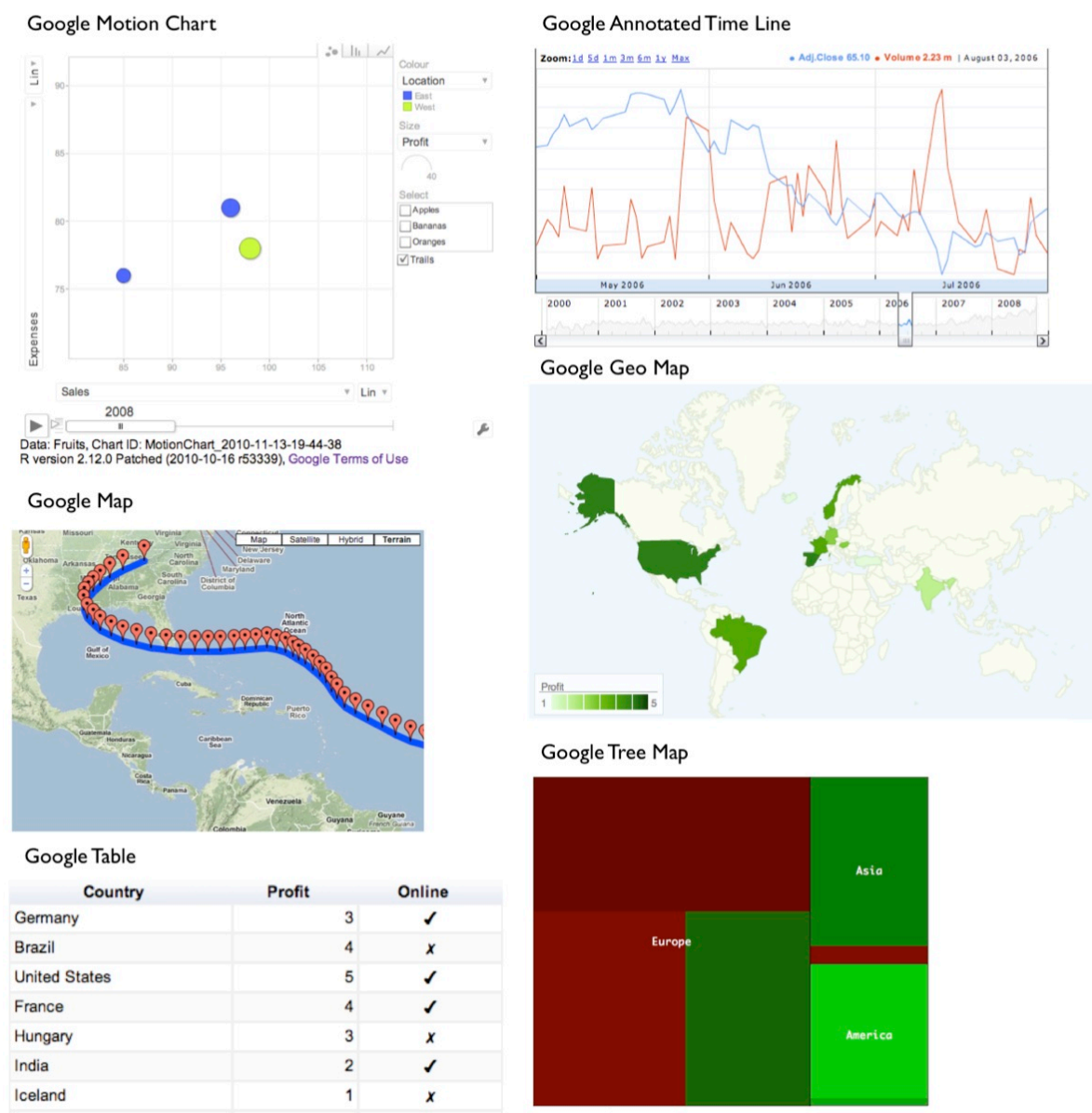


Figure 2: Screenshot of some of the outputs of `demo(googleVis)`. Clockwise from top left: `gvisMotionChart`, `gvisAnnotatedTimeLine`, `gvisGeoMap`, `gvisTreeMap`, `gvisTable`, and `gvisMap`.

The individual functions of the **googleVis** package are documented in detail in the help pages and package *vignette*. Here we will cover only the principles of the package.

As an example we will show how to generate a Motion Chart as displayed in Figure 1. It works similarly for the other APIs. Further examples are covered in the demos of the **googleVis** package and on the project Google Code site.

The design of the visualisation functions is fairly generic. The name of the visualisation function is 'gvis' followed by the chart type. Thus for the Motion Chart we have:

```
gvisMotionChart(data, idvar = 'id',
  timevar = 'date', options = list())
```

Here `data` is the input `data.frame` and the arguments `idvar` and `timevar` specify the column names of the id variable and time variable for the plot, while display options are set in an optional list. The options and data requirements follow those of the Google Visualisation API and are documented in the help pages, see `help('gvisMotionChart')`.

The output of a **googleVis** function is a list of lists (a nested list) containing information about the chart type, chart id, and the html code in a sub-list split into header, chart, caption and footer.

The idea behind this concept is that users get a complete web page while at the same time they can extract only specific parts, such as the chart. This is particularly helpful if the package functions are used in solutions where the user wants to feed the visuali-

sation output into other sites, or would like to embed them into rsp-pages, or use RApache or Google Gadgets.

The output of a **googleVis** function will be of class "gvis" and "list". Generic print (print.gvis) and plot (plot.gvis) methods exist to ease the handling of such objects.

To illustrate the concept we shall create a motion chart using the `Fruits` data set.

## Motion chart example

Following the documentation of the Google Motion Chart API we need a data set which has at least four columns: one identifying the variable we would like to plot, one time variable, and at least two numerical variables; further numerical and character columns are allowed.

As an example we use the `Fruits` data set:

```
R> data(Fruits)
R> Fruits[, -7] # ignore column 7
```

	Fruit	Year	Location	Sales	Expenses	Profit
1	Apples	2008	West	98	78	20
2	Apples	2009	West	111	79	32
3	Apples	2010	West	89	76	13
4	Oranges	2008	East	96	81	15
5	Bananas	2008	East	85	76	9
6	Oranges	2009	East	93	80	13
7	Bananas	2009	East	94	78	16
8	Oranges	2010	East	98	91	7
9	Bananas	2010	East	81	71	10

Here we will use the columns 'Fruit' and 'Year' as id and time variable respectively.

```
R> M <- gvisMotionChart(Fruits, idvar = "Fruit",
                        timevar = "Year")
```

The structural output of `gvisMotionChart` is a list of lists as described above:

```
R> str(M)
```

```
List of 3
 $ type   : chr "MotionChart"
 $ chartid: chr "MotionChartID12ae2fff"
 $ html   :List of 4
 ..$ header : chr "<!DOCTYPE html PUBLIC ..."
 ..$ chart  : Named chr [1:7] "<!-- Moti ..."
 ..$.- attr(*, "names")= chr [1:7] "jsH ..."
 ..$ caption: chr "<div><span>Data: Fruit..."
 ..$ footer : chr "\n<!-- htmlFooter -->\n..."
 - attr(*, "class")= chr [1:2] "gvis" "list"
```

The first two items of the list contain information about the chart type used and the individual chart id. The html output is a list with header, chart, caption and footer. This allows the user to extract only certain parts of the page, or to create a complete html page.

The header part of the html page has only basic html and formatting tags and provides a simple layout, see

```
R> print(M, 'header') # output not shown here
```

The actual Google visualisation code is stored with the data in the named character vector `chart` of the html list, see

```
R> print(M, 'chart') # output not shown here
```

The sub-items of the chart object give the user more granular access to JavaScript functions and html tags of the visualisation. A basic chart caption and html footer complete the html list:

```
R> print(M, 'caption') # output not shown here
R> print(M, 'footer') # output not shown here
```

## Displaying "gvis" objects

To display the page locally, type:

```
R> plot(M) # returns invisibly the file name
```

The plot method for "gvis" objects creates html files in a temporary folder using the type and chart id information of the object and it will display the output using the R HTTP help web server locally. The R command `tempdir()` shows the path of the per-session temporary directory.

Further examples are part of the **googleVis** demos, including one showing how a Geo Map can be animated with additional JavaScript, see `demo` (package = "googleVis").

## Combing charts with gvisMerge

The function `gvisMerge` takes two "gvis" objects and merges the underlying components into one page. The charts are aligned either horizontally or vertically next to each other in an HTML table.

The output of `gvisMerge` is a "gvis" object again. This allows us to apply the same function iteratively to create more complex chart layouts. The following example, see Figure 3, aligns a Geo Chart and Table below each other, and combines the output with a Motion Chart to the right.

```
G <- gvisGeoChart(Exports, "Country", "Profit",
                 options = list(width = 200, height = 100))
T <- gvisTable(Exports,
              options = list(width = 200, height = 270))
M <- gvisMotionChart(Fruits, "Fruit", "Year",
                   options = list(width = 400, height = 370))
GT <- gvisMerge(G, T, horizontal = FALSE)
GTM <- gvisMerge(GT, M, horizontal = TRUE,
               tableOptions =
                 "bgcolor = \"#CCCCC\" cellspacing = 10")
plot(GTM)
```



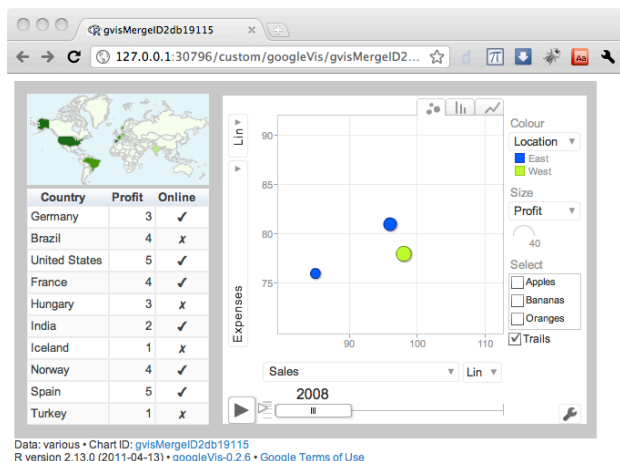


Figure 3: Three charts combined with gvisMerge.

## Embedding googleVis in web sites dynamically

With the R packages **R.rsp** and **brew** we have two options to integrate R snippets into html code. While the **R.rsp** package comes with its own internal web server, **brew** requires the Apache HTTP server with the RApache module installed.

Both packages allow the user to embed R code into html code. The R code is filtered by the **R.rsp** or RApache web server and executed at run time. As an example, we can insert the above motion chart into a page:

```
<html>
<body>
<% library(googleVis) %>
<% M <- gvisMotionChart(Fruits, idvar="Fruit",
  timevar="Year") %>
<%= M$html$chart %>
</body>
</html>
```

The syntax of **R.rsp** and **brew** are very similar. The R code included in `<%...%>` is executed when read by the HTTP server, but no R output will be displayed. To insert the R output into the html code we have to add an equal sign, `<%=...%>`, which acts as a cat statement. In the example above the chart code is embedded into the html code.

Examples for both approaches are part of the **googleVis** package. For more information see the vignettes and help files of the packages.

## Summary

Combining R with the Google Visualisation API enables users to quickly create powerful analysis tools,

which can be shared online. The user interface is easily accessible both to data and non-data analysts.

The “Statistics Relating to Lloyd’s” site (<http://www.lloyds.com/stats>) is an example where the functions of the **googleVis** package were used to create a data visualisation page. Further case studies and links to alternative packages are available on the project Google Code site<sup>8</sup>.

## Bibliography

- H. Bengtsson. R.rsp: R server pages. <http://CRAN.R-project.org/package=R.rsp>, 2011. R package version 0.7.0.
- M. Gesmann and D. de Castillo. googleVis: Using the Google Visualisation API with R. <http://code.google.com/p/google-motion-charts-with-r/>, 2011. R package version 0.2.12.
- Jeffrey Horner. brew: Templating framework for report generation. <http://CRAN.R-project.org/package=brew>, 2011. R package version 1.0-6.
- Jeffrey Horner. RApache: Web application development with R and Apache. <http://www.rapache.net/>, 2011.
- F. Leisch. Sweave: Dynamic generation of statistical reports using literate data analysis. In W. Härdle and B. Rönz, editors, *Compstat 2002 — Proceedings in Computational Statistics*, pages 575–580. Physica Verlag, Heidelberg, 2002. ISBN 3-7908-1517-9.
- S. P. Saaibi. R/RMETRICS Generator Tool for Google Motion Charts. 4th R/Rmetrics User and Developer Workshop, Meielisalp, Lake Thune, Switzerland, June 27 - July 1, 2010. <https://www.rmetrics.org/>.
- D. Temple Lang. RJSONIO: Serialize R objects to JSON, JavaScript Object Notation. <http://CRAN.R-project.org/package=RJSONIO>, 2011. R package version 0.96-0.

Markus Gesmann  
googleVis project  
London  
UK  
[markus.gesmann@gmail.com](mailto:markus.gesmann@gmail.com)

Diego de Castillo  
googleVis project  
Cologne  
Germany  
[decastillo@gmail.com](mailto:decastillo@gmail.com)

<sup>8</sup><http://code.google.com/p/google-motion-charts-with-r/>