Using the Google Chart Tools with R: googleVis-0.4.4 Package Vignette

Markus Gesmann*, Diego de Castillo†
Contact: rvisualisation@gmail.com

August 23, 2013

Abstract

The `googleVis` package provides an interface between R and the Google Chart Tools. The Google Chart Tools offer 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.

The functions of the `googleVis` package allow the user to visualise data stored in R data frames with the Google Chart Tools without uploading the data to Google. The output of a `googleVis` function is html code that contains the data and references to JavaScript functions hosted by Google.

googleVis makes use of the internal R HTTP server to display the output locally. A modern browser with an Internet connection is required and for some charts Flash.

*markus.gesmann@gmail.com
†decastillo@gmail.com

Contents

1 Introduction **4**

1.1 Motivation 4

1.2 Google Chart Tools 4

2 The `googleVis` package **7**

2.1 Installation 7

2.2 Using the `googleVis` package 8

2.3 Motion Chart Example 10

2.4 Displaying `gvis` objects locally 16

2.5 Setting default behaviour of `print.gvis` and `plot.gvis` 17

2.6 Combining charts with `gvisMerge` 18

2.7 Setting options 19

2.7.1 Chart Editor 20

2.7.2 Dealing with apostrophes in column names 21

3 Embedding `googleVis` in web sites **22**

3.1 Integrating `gvis` objects in existing sites 22

3.2 Using `googleVis` output with WordPress 22

3.3 Using `googleVis` output with Google Sites, Blogger, etc. 23

3.4 Embedding `googleVis` in web sites dynamically 25

3.4.1 Using `googleVis` with `R.rsp` 25

3.4.2 Using `googleVis` with *RApache* and `brew` 26

3.4.3 Using `googleVis` with *Rook* 27

3.4.4 Using `googleVis` with `shiny` 29

4 Using `googleVis` with `knitr` **30**

5 Using `googleVis` in presentations **34**

6 Beyond R **34**

6.1 Registering to catch events 35

7 Frequent ask questions – FAQ **36**

7.1 Can I use googleVis output in PDF files? 36

7.2 Can I change the colour of the bubbles in motion charts? 36

7.3 Can I change the icons in Maps? 37

7.4 Why can't I see motion charts when I open them from a local directory? 37

7.5 Why can't I see motion charts on my iPad/iPhone? 37

7.6 How can I set axes limits with googleVis? 37

7.7 How can I change the look and feel of the charts? 38

7.8 Is it possible to use `googleVis` in corporate work? 38

7.9 Bugs and issues 38

8 Contact **38**

8.1 Collaboration 38

8.2 Citation 38

8.3 Training and consultancy 39

References **40**

1 Introduction

1.1 Motivation

More and more data is becoming available, and yet stories and insights are still often missed: we are lost in the data jungle and struggle to see the wood for the trees.

Hence, new tools are required to bring data to life, to engage with users, to enable them to slice and dice the data, to view it from various angles and to find stories worth telling: outliers, trends or even the obvious.

In 2006 Hans Rosling gave an inspiring talk at TED [Ros06] 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 team at Gapminder [Fou10b] had developed animated bubble charts, aka motion charts, see Figure 1.

Rosling's presentation popularised the idea and use of interactive charts. One year later the software behind Gapminder was bought by Google and integrated as motion charts into their Google Chart Tools [Inc12b], formerly known as Google Visualisation API.

In 2010 Sebastián Pérez Saaibi [Saa10] 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 `R.rsp` package [Ben12].

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 vignette started the development of the `googleVis` package [GdC13], [GdC11] in August 2010.

Of course there are many other alternative visualisation toolkits out there, e.g. d3js [Bos12], Many Eyes [RtICsg10], Open Flash Chart (Flash) [GHN10], OpenLayers (JavaScript) [Fou10c], Processing (Java) [FR10], and FLARE (ActionScript) [Lab10].

1.2 Google Chart Tools

The Google Chart Tools [Inc12b] allow users to create interactive charts as part of Google documents, spreadsheets and web pages. In this text we will focus on the usage of the API as part of web pages.

The Google Public Data Explorer [Inc12d] provides a good example, demonstrating the use of interactive charts and how they can help to analyse data. Please note, that all of those charts are rendered by the browser.

The charting data can either be embedded into the html file or read dynamically. The key to the Google Chart Tools is that the data is structured in a DataTable [Inc12e],

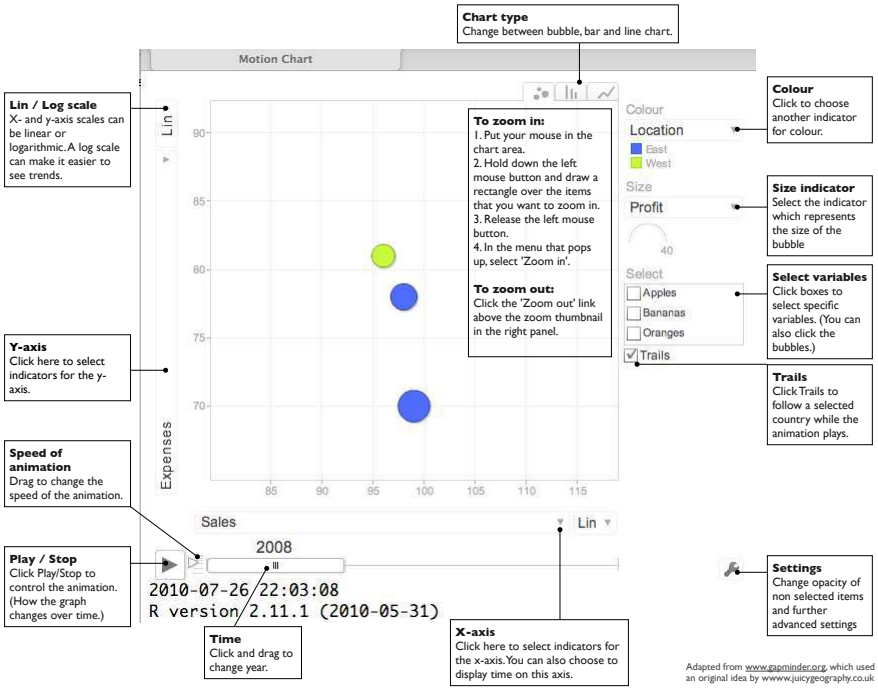

Figure 1: Overview of a Google Motion Chart. Screenshot of the output of `plot(gvisMotionChart(Fruits, idvar='Fruit', timevar='Year'))`

and this is where the `googleVis` package helps, as it transforms R data frames into JSON [JSO06] objects, using the `RJSONIO` package [Lan12], as the basis for a DataTable.

As an example we shall look at the html-code of a motion chart from Google's visualisation gallery [Inc12c], which generates output similar to Figure 1:

```
<html>
 <head>
  <script type="text/javascript"
    src="http://www.google.com/jsapi">
  </script>
  <script type="text/javascript">
    google.load('visualization', '1',
     {'packages':['motionchart']});
   google.setOnLoadCallback(drawChart);
   function drawChart() {
   var data=new google.visualization.DataTable();
   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 by the browser and is not submitted to any server[1].

You will notice that the above html code has five generic parts[2]:

- 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),
- 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 Chart Tools, see the examples in Figure 2.

However, before you use the API you should read the Google Terms of Service [Inc12a].

[1] `https://google-developers.appspot.com/chart/interactive/docs/gallery/motionchart.html#Data_Policy`

[2] For more details see `https://google-developers.appspot.com/chart/interactive/docs/index`

2 The googleVis package

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

Version (0.4.4) of the package provides interfaces to Motion Charts, Annotated Time Lines, Maps, Geo Maps, Geo Charts, Intensity Maps, Tables, Gauges, Tree Maps, further Line, Bar, Bubble, Column, Area, Stepped Area, Combo, Scatter, Candlestick, Pie and Org Charts; see Figure 2 for some examples. All available charts types are listed on the project site[3].

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.

2.1 Installation

You can install googleVis in the usual way from CRAN, e.g.:

```
R> install.packages('googleVis')
```

The installation was successful if the command library(googleVis) gives you the following message:

```
R> library(googleVis)

Welcome to googleVis version 0.4.4

Please read the Google API Terms of Use
before you use the package:
https://developers.google.com/terms/

Type ?googleVis to access the overall documentation and
vignette('googleVis') for the package vignette.
You can execute a demo of the package via: demo(googleVis)

More information is available on the googleVis project web-site:
http://code.google.com/p/google-motion-charts-with-r/

Contact: <rvisualisation@gmail.com>

To suppress the this message use:
suppressPackageStartupMessages(library(googleVis))
```

[3] http://code.google.com/p/google-motion-charts-with-r/wiki/GadgetExamples

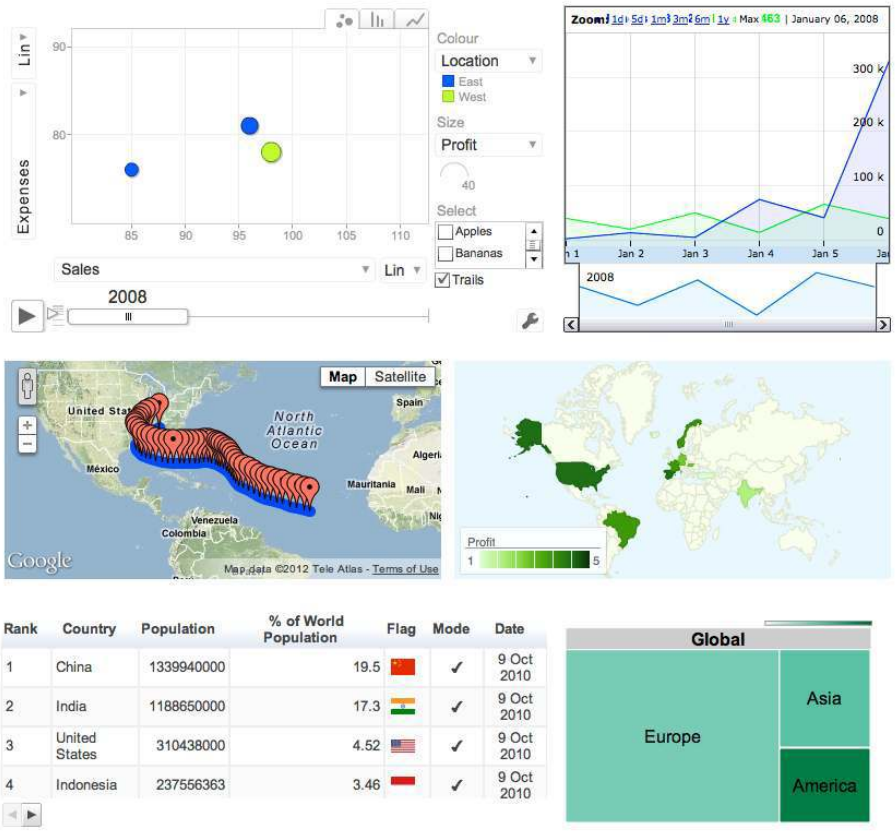

Figure 2: Screenshot of some of the outputs of `demo(googleVis)`. Clockwise from top left: `gvisMotionChart`, `gvisAnnotatedTimeLine`, `gvisGeoMap`, `gvisTreeMap`, `gvisTable`, and `gvisMap`.

2.2 Using the `googleVis` package

The individual functions of the `googleVis` package are documented in detail in the help pages. 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[4] of the `googleVis` package.

The design of the visualisation functions is fairly generic. The name of the visualisation function is `'gvis' + ChartType`. So for the Motion Chart we have:

[4]See `demo(package="googleVis")` for a list of the available demos.

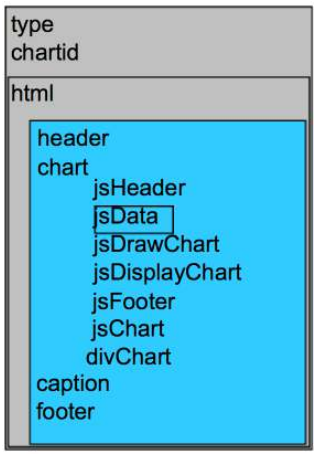

Figure 3: Schematic structure of a gvis list object.

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

Here `data` is the input `data.frame` and `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, which we discuss in more detail on page 19. The options and data requirements follow those of the Google Chart Tools and are documented in the help pages, see

```
R> help('gvisMotionChart')
```

The argument `chartid` allows the user to set a chart id of the output chart manually. If the argument is missing a random id using `tempfile(pattern='')` will be generated. Unique chart ids are required to place more than one chart on a web page.

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 with header, chart, caption and footer, see Figure 3.

The idea behind this concept is that users can get a complete web page, while at the same time offer a facility to extract specific parts, such as the chart itself. This is particularly helpful if the package functions are used in solutions where the user wants to feed the visualisation output into other sites, or would like to embed them into rsp-pages (see page 25), use *RApache* (see page 26) or Google Gadgets.

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

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

2.3 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
```

```
    Fruit Year Location Sales Expenses Profit       Date
1  Apples 2008     West    98       78     20 2008-12-31
2  Apples 2009     West   111       79     32 2009-12-31
3  Apples 2010     West    89       76     13 2010-12-31
4 Oranges 2008     East    96       81     15 2008-12-31
5 Bananas 2008     East    85       76      9 2008-12-31
6 Oranges 2009     East    93       80     13 2009-12-31
7 Bananas 2009     East    94       78     16 2009-12-31
8 Oranges 2010     East    98       91      7 2010-12-31
9 Bananas 2010     East    81       71     10 2010-12-31
```

Here we will use the columns `'Fruit'` and `'Year'` as id and time variable respectively. However we could have used `'Date'` instead of `'Year'` as well.

```
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 "MotionChartID1437b9f778f"
 $ html   :List of 4
  ..$ header : chr "<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0
  ..$ chart  : Named chr [1:7] "<!-- MotionChart generated in R 3.
  .. ..- attr(*, "names")= chr [1:7] "jsHeader" "jsData" "jsDrawCh
  ..$ caption: chr "<div><span>Data: Fruits &#8226; Chart ID: <a h
  ..$ footer : chr "\n  <!-- htmlFooter -->\n  <span> \n  R versio
 - 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:

```
R> M$type
```

```
[1] "MotionChart"
```

```
R> M$chartid
```

```
[1] "MotionChartID1437b9f778f"
```

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:

```
R> print(M, tag='header')
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
  <html xmlns="http://www.w3.org/1999/xhtml">
  <head>
  <title>MotionChartID1437b9f778f</title>
  <meta http-equiv="content-type" content="text/html;charset=utf-8" />
  <style type="text/css">
  body {
  color: #444444;
  font-family: Arial,Helvetica,sans-serif;
  font-size: 75%;
  }
  a {
  color: #4D87C7;
  text-decoration: none;
  }
  </style>
  </head>
  <body>
```

Here we used the `print` statement with the tag `'header'` instead of `M$html$header` to achieve a formatted screen output. This is the same output as `cat(M$html$chart)`.

The actual Google visualisation code is stored with the data as a named character vector in the `chart` item of the html list. The chart is made up of several JavaScript and HTML statements. Please notice that the JavaScript functions are uniquely named with the information of the chart id. This concept allows the user get all the chart code directly or only specific parts; see the examples in the help page of `print.gvis` for more details.

```
R> names(M$html$chart)

[1] "jsHeader"       "jsData"         "jsDrawChart"    "jsDisplayChart"
[5] "jsFooter"       "jsChart"        "divChart"
```

The complete chart can be displayed via:

```
R> print(M, tag='chart')  ## or cat(M$html$chart)

<!-- MotionChart generated in R 3.0.1 by googleVis 0.4.4 package -->
<!-- Fri Aug 23 20:01:56 2013 -->


  <!-- jsHeader -->
  <script type="text/javascript">

  // jsData
  function gvisDataMotionChartID1437b9f778f () {
  var data = new google.visualization.DataTable();
  var datajson =
  [
 [
 "Apples",
2008,
"West",
98,
78,
20,
"2008-12-31"
],
[
 "Apples",
2009,
"West",
111,
79,
32,
"2009-12-31"
],
[
 "Apples",
2010,
"West",
89,
76,
```

13,
"2010-12-31"
],
[
 "Oranges",
2008,
"East",
96,
81,
15,
"2008-12-31"
],
[
 "Bananas",
2008,
"East",
85,
76,
9,
"2008-12-31"
],
[
 "Oranges",
2009,
"East",
93,
80,
13,
"2009-12-31"
],
[
 "Bananas",
2009,
"East",
94,
78,
16,
"2009-12-31"
],
[
 "Oranges",
2010,
"East",
98,
91,
7,

```
"2010-12-31"
],
[
 "Bananas",
2010,
"East",
81,
71,
10,
"2010-12-31"
]
];
  data.addColumn('string','Fruit');
data.addColumn('number','Year');
data.addColumn('string','Location');
data.addColumn('number','Sales');
data.addColumn('number','Expenses');
data.addColumn('number','Profit');
data.addColumn('string','Date');
  data.addRows(datajson);
  return(data);
  }

  // jsDrawChart
  function drawChartMotionChartID1437b9f778f() {
  var data = gvisDataMotionChartID1437b9f778f();
  var options = {};
  options["width"] =    600;
options["height"] =    500;


    var chart = new google.visualization.MotionChart(
    document.getElementById('MotionChartID1437b9f778f')
    );
    chart.draw(data,options);


  }


  // jsDisplayChart
  (function() {
  var pkgs = window.__gvisPackages = window.__gvisPackages || [];
  var callbacks = window.__gvisCallbacks = window.__gvisCallbacks || [];
  var chartid = "motionchart";
```

```
// Manually see if chartid is in pkgs (not all browsers support Array.indexOf)
var i, newPackage = true;
for (i = 0; newPackage && i < pkgs.length; i++) {
if (pkgs[i] === chartid)
newPackage = false;
}
if (newPackage)
pkgs.push(chartid);

// Add the drawChart function to the global list of callbacks
callbacks.push(drawChartMotionChartID1437b9f778f);
})();
function displayChartMotionChartID1437b9f778f() {
var pkgs = window.__gvisPackages = window.__gvisPackages || [];
var callbacks = window.__gvisCallbacks = window.__gvisCallbacks || [];
window.clearTimeout(window.__gvisLoad);
// The timeout is set to 100 because otherwise the container div we are
// targeting might not be part of the document yet
window.__gvisLoad = setTimeout(function() {
var pkgCount = pkgs.length;
google.load("visualization", "1", { packages:pkgs, callback: function() {
if (pkgCount != pkgs.length) {
// Race condition where another setTimeout call snuck in after us; if
// that call added a package, we must not shift its callback
return;
}
while (callbacks.length > 0)
callbacks.shift()();
} });
}, 100);
}

// jsFooter
</script>

<!-- jsChart -->
<script type="text/javascript" src="https://www.google.com/jsapi?callback=displayChartMoti

<!-- divChart -->

<div id="MotionChartID1437b9f778f"
style="width: 600px; height: 500px;">
</div>
```

Similarly you can also access specific components of the chart, e.g. (output truncated)

```
R> cat(M$html$chart['jsChart']) # or print(M, 'jsChart')

  <!-- jsChart -->
  <script type="text/javascript" src="https://www.google.com/jsapi
```

A basic chart caption and html footer are the final items of the html list (output truncated):

```
R> print(M, tag='caption')

<div><span>Data: Fruits &#8226; Chart ID: <a href="Chart_MotionCha

R> print(M, tag='footer')

  <!-- htmlFooter -->
  <span>
  R version 3.0.1 (2013-05-16) &#8226; <a href="http://code.google
  &#8226; <a href="https://developers.google.com/terms/">Google Te
  </span></div>
  </body>
  </html>
```

2.4 Displaying `gvis` objects locally

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, usually under http://127.0.0.1.

Note that the chart caption provides a link to the chart code via the chart id for copy and paste.

The R command `tempdir()` will show you the path of the per-session temporary directory, in which the files were written. You can write the chart into a local html file via the `print` command with the file argument, e.g.

```
R> print(M, file="myGoogleVisChart.html")
```

Please note that Flash charts[5] may not work when loaded as a local file due to security settings, and therefore require to be displayed via a web server. However, you can overcame this issue by changing your Flash security settings. Tony Breyal posted the following solution on stackoverflow.com:

[5] Currently the following charts require Flash: motion chart, geo map and annotated time line

1. Go to `http://www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager04.html`
2. Click on the dropbox which says 'Edit location' and choose 'add location'
3. Click 'browse for folder'
4. Choose the folder in which you saved your html file
5. Click OK

Now open your `googleVis` html file and it should display successfully.

Alternatively use the function `plot.gvis` explicitly, e.g. suppose your html file is stored in `/Users/JoeBloggs/myGoogleVisChart.html`. Using the `plot.gvis` the file will be copied into a temporary directory and displayed via the R HTTP help server with, in the same way as a gvis-object:

```
R> plot.gvis("/Users/JoeBloggs/myGoogleVisChart.html")
```

Please note that this feature of `plot.gis` was introduced to `googleVis` with version 0.3.2.

2.5 Setting default behaviour of `print.gvis` and `plot.gvis`

In `googleVis` version 0.3.2 the function `plot.gvis` gained the same argument as `print.gvis`: `tag`. By default the `tag` argument is set to `NULL` in `plot.gvis` and the plot function will display its output in a browser window. However, if `tag` is not `NULL` the function `plot.gvis` will behave exactly like `print.gvis`.

The default `tag` can be set for both functions globally via the `options()` function. On package load `googleVis` sets `options(gvis.print.tag='html')` and `options(gvis.plot.tag=NULL)`.

Suppose you would set `options(gvis.plot.tag='chart')` then all following plot statements would print the chart part of the gvis-object only, without opening a browser window. This might seem a bit odd at first, yet it becomes helpful when you write R Markdown files for `knitr` or files for other packages such as `R.rsp`.

While you draft your file you may want to see the output of `googleVis` in an interactive way, so you set `options(gvis.plot.tag=NULL)` at the top of the file and you change the setting to `'chart'` before you parse the file, say with `knitr`. This will ensure that all plot statements return the HTML code of the chart, rather than opening browser windows. Section 4 on page 30 provides more details and a little `knitr` example.

2.6 Combining 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.

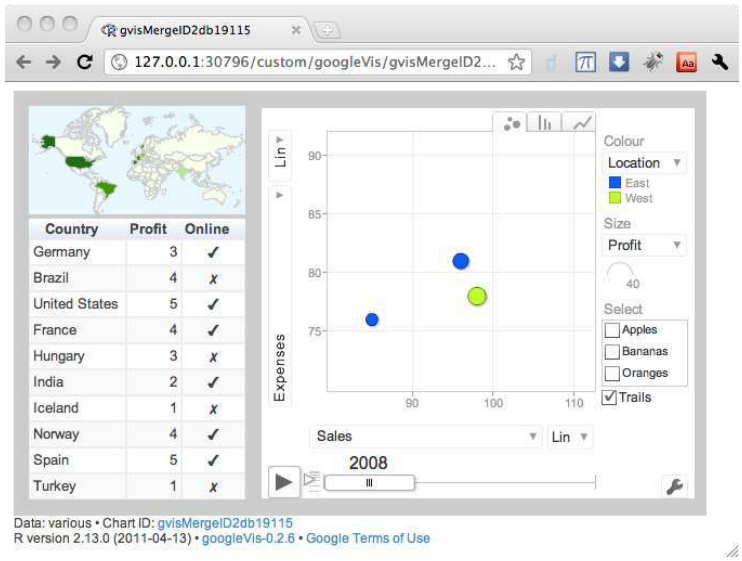

Figure 4: Three charts combined with `gvisMerge`.

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 4, aligns a geo chart and table below each other, and combines the output with a motion chart to the right:

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

R> plot(GTM)
```

2.7 Setting options

Setting the various options of a `googleVis` objects can be a bit cumbersome at first. The options follow those of the Google Chart Tools and can be set via a named list using the argument `options`. In the following example we create a line chart and set various options[6], see the output in Figure 5.

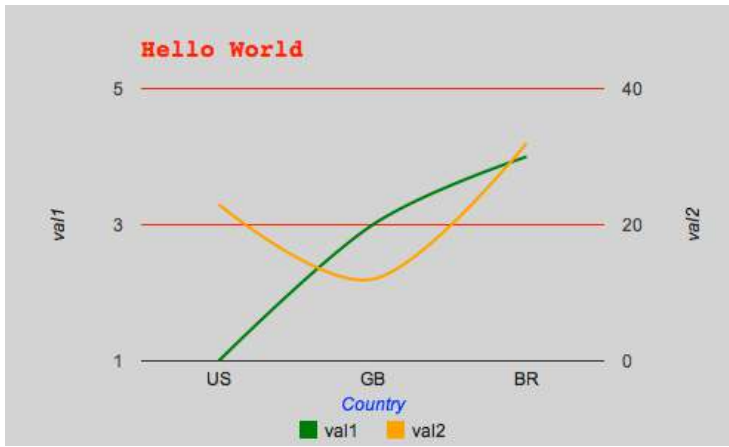

Figure 5: A line chart with various options set.

```
R> df <- data.frame(country=c("US", "GB", "BR"),
+                         val1=c(1,3,4), val2=c(23,12,32))
R> Line <-  gvisLineChart(df, xvar="country", yvar=c("val1","val2"),
+                         options=list(
+                           title="Hello World",
+                           titleTextStyle="{color:'red',
+                                            fontName:'Courier',
+                                            fontSize:16}",
+                           backgroundColor="#D3D3D3",
+                           vAxis="{gridlines:{color:'red', count:3}}",
+                           hAxis="{title:'Country', titleTextStyle:{color:'blue'}}",
+                           series="[{color:'green', targetAxisIndex: 0},
+                                    {color: 'orange',targetAxisIndex:1}]",
+                           vAxes="[{title:'val1'}, {title:'val2'}]",
+                           legend="bottom",
+                           curveType="function",
+                           width=500,
+                           height=300
```

[6]Please refer to the help file of the individual `googleVis` functions, as the options can vary from chart to chart.

```
+                                ))
R> plot(Line)
```

As you can see from the example above, the simpler options can be set by name=value, e.g. width=500, while the more complex options with sub-components are listed in curly brackets {}, and arrays, e.g. to define the two axes, use square brackets [ ].

Here is another example setting the colour axis of a geo chart, see Figure 6:

```
R> Geo <- gvisGeoChart(CityPopularity, locationvar='City',
+                      colorvar='Popularity',
+                       options=list(region='US', height=350,
+                        displayMode='markers',
+                            colorAxis="{values:[200,400,600,800],
+                         colors:[\'red', \'pink\', \'orange',\'green']}")
+                        )
R> plot(Geo)
```

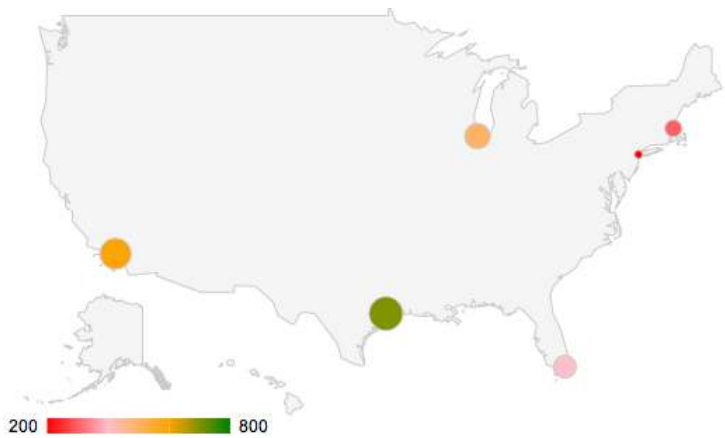

Figure 6: A geo chart with options set for a colour axis.

2.7.1 Chart Editor

A special option for all charts is `gvis.editor`, which adds an edit button to the page, allowing the user to edit, change and customise the chart on the fly, see the following example and Figure 7. The content of the list item `gvis.editor` describes the label of the browser button[7].

[7] See also https://google-developers.appspot.com/chart/interactive/docs/drawing_charts#charteditor

```
R> Editor <- gvisLineChart(df, options=list(gvis.editor='Edit me!'))
R> plot(Editor)
```

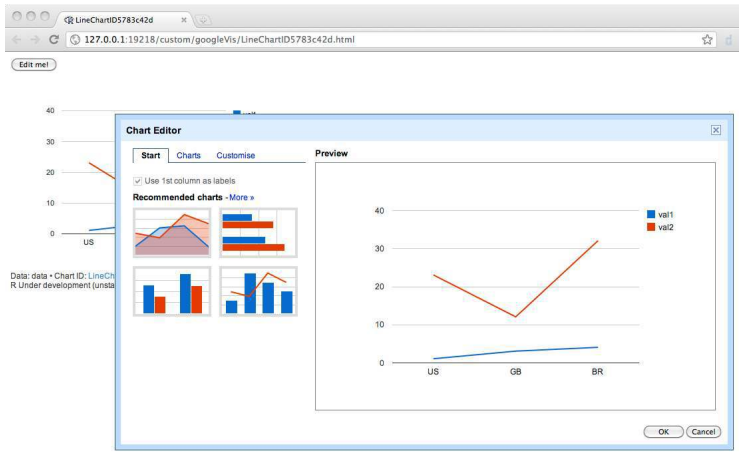

Figure 7: Example of `googleVis` chart with `options = list (gvis.editor = 'Edit me!')`.

2.7.2 Dealing with apostrophes in column names

The `googleVis` package converts data frames into JSON objects. The column names of the resulting JSON tables are encapsulated with single speech marks, see lines 12 – 16 of the code example on page 5.

Hence apostrophes in column names of your input data frame have to be encapsulated by a double backslash.

Here is a little example, see the output in Figure 8.

```
R> df <- data.frame("Year"=c(2009,2010), "Lloyd\\'s"=c(86.1, 93.3),
+                   "Munich Re\\'s R/I"=c(95.3, 100.5),
+                   check.names=FALSE)
R> df

  Year Lloyd\\'s Munich Re\\'s R/I
1 2009      86.1              95.3
2 2010      93.3             100.5

R> CR <- gvisColumnChart(df, options=list(vAxis='{baseline:0}',
+                                 title="Combined Ratio %",
+                                 legend="{position:'bottom'}"))
```

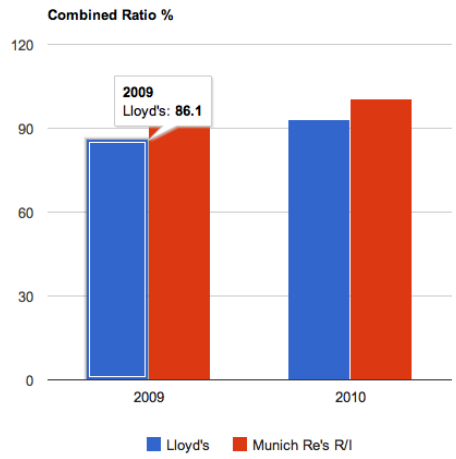

Figure 8: Visualising data with apostrophes in column names.

```
R> plot(CR)
```

3 Embedding `googleVis` in web sites

3.1 Integrating `gvis` objects in existing sites

Suppose you have an existing web page and would like to integrate the output of a `googleVis` function, such as `gvisMotionChart`. In this case you only need the chart output from `gvisMotionChart`. So you can either copy and paste the output from the R console

```
R> print(M, 'chart')  ## or cat(M$html$chart)
```

into your existing html page, or write the content directly into a file

```
R> print(M, 'chart', file='myfilename')
```

and process it from there.

3.2 Using `googleVis` output with WordPress

WordPress is a popular web software for creating websites and blogs. Here we give some tips about the usage with `googleVis` output.

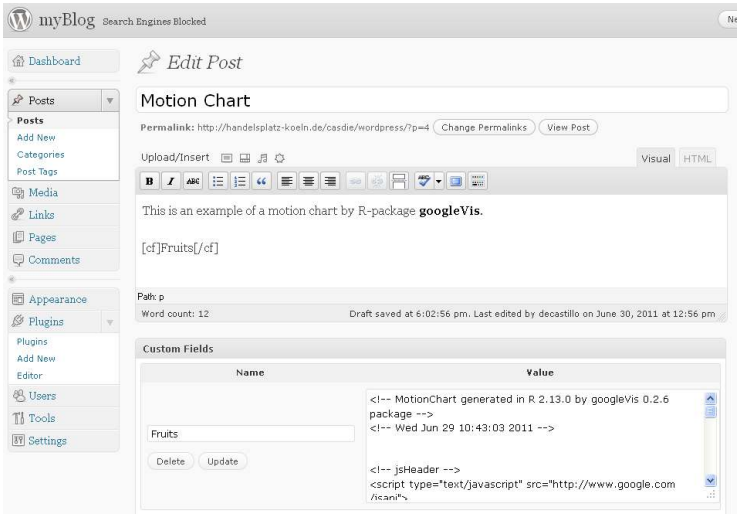

Figure 9: Including `googleVis` output in a WordPress blog entry.

By default WordPress does not allow JavaScript code (and hence `googleVis` output) to be inserted into a page. However, additional plugins allow you to extend the functions of WordPress.

One option of embedding JavaScript code inside a WordPress post is to use the "custom fields shortcode" plugin[8]. The plugin allows you to create a custom field for the `googleVis` code, which can be referred to in your article.

Suppose you created a motion chart in R:

```
R> M <- gvisMotionChart(Fruits, "Fruit", "Year",
+                       options=list(width=400, height=370))
```

Copy the chart code, e.g. from

```
R> print(M, 'chart')
```

and paste it into the value text area of a custom field in WordPress, e.g. with instance name `Fruits`. To include the motion chart into your article add `[cf]Fruits[/cf]` into the post, see Figure 9 for an illustration.

3.3 Using `googleVis` output with Google Sites, Blogger, etc.

Google Charts can be used with other Google products such as Google Sites, Blogger or Google Code wiki pages. However, in most cases the chart has to be embedded

[8] `http://wordpress.org/extend/plugins/custom-fields-shortcode`

as a Google Gadget. Google Gadgets are written in XML and can have HTML and Javascript components. Here is an example of a 'Hello World' program written using Google Gadget technology from Wikipedia.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Module>
<ModulePrefs title="simple hello world example" />
<Content type="html">
     <![CDATA[
       Hello, world!
     ]]>
</Content>
</Module>
```

The `googleVis` package comes with the function `createGoogleGadget`, which takes a gvis-object and wraps it into an XML gadget file. Here is an example with a motion chart:

```
R> M <- gvisMotionChart(Fruits, "Fruit", "Year")
R> G <- createGoogleGadget(M)

R> cat(G, file="myGadget.xml")
```

In order to use the gadget, the file `myGadget.xml` has to be hosted online, e.g. using Google Docs. Suppose the URL to the gadget is `http://example.com/myGadget.xml`, than you can embed the gadget

- in a Google Site via the menu:
  "Insert" -> "More gadgets ..." -> "Add gadget URL",
- in a Google Code wiki via the `wiki:gadget` tag, e.g.:
  `<wiki:gadget url="http://example.com/gadget.xml" />`,
- in Blogger via the design tab, see site gadgets.

However, the situation is different again, if you would like to include the `googleVis` output into a blogger post. In this case a gadget does not work. You can either copy and paste the chart directly into your post, but then it does not seem to work with MS Internet Explorer, or you use an indirect approach. This means that the `googleVis` chart has to be hosted on a separate page (e.g. in a public Dropbox folder) and is embedded into the blog post by using the `<iframe>` tag, e.g.:

```
<iframe width="100%" height="400px" frameborder="0"
src="http://example.com/myGoogleVisChart.html">
</iframe>
```

For an example see the following blog entry: http://lamages.blogspot.com/2011/09/including-googlevis-output-into-blogger.html.

3.4 Embedding `googleVis` in web sites dynamically

In this section we provide examples how the `googleVis` functions can be embedded into web sites dynamically.

The R packages `R.rsp` [Ben12] and `brew` [Hor11a] are 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 [Fou10a] with the *RApache* [Hor11b] module installed. Please note that currently the *RApache* module only runs on UNIX/Linux and Mac OS X.

Since version 2.11.0 R ships with its own internal web server and with *Rook* [Hor13] and `shiny` [RI13] there are also two alternatives that use this facility. Both packages allow users to deploy web application locally with very little overhead. The `shiny` framework is the newest and certainly easiest option to get started with web apps using R and `googleVis` .

3.4.1 Using `googleVis` with `R.rsp`

The `R.rsp` package allows the user to integrate R code into html code. The R code is parsed by the `R.rsp` web server and executed at run time.

As an example, we embed a motion chart into a rsp-page:

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

The R code included in <%...%> is executed when read by the R.rsp HTTP server, but no R output will be displayed. To embed the R output into the html code we have to add an equal sign, <%=...%>, which acts as a `cat` statement.

You find an example as part of the `googleVis` package. This example can be displayed via the following R command:

```
R> library(R.rsp)
R> browseRsp()
R> # Follow the link for googleVis in the opening browser window
```

The actual rsp-file is located within the `googleVis` package directory and again R allows you to find the file with the following command:

```
R> file.path(system.file("rsp", package = "googleVis"), "index.rsp")
```

For more information read the documentation of the `R.rsp` package.

3.4.2 Using `googleVis` with RApache and `brew`

RApache supports web application development using R and the Apache HTTP server. The *RApache* module embeds the R interpreter into the Apache web server. However, as we would like to mix R and html code we also need a parser and this is where the R package `brew` comes into place.

Files sitting in a dedicated `brew` folder of the HTTP repository are parsed by `brew` when opened in the browser. The R code is executed with *RApache* and the output is embedded into the site. Hence the approach is similar to `R.rsp` with the difference that the two tasks are split. This has the advantage that R does not have to run in a separate window.

Detailed installation instructions for *RApache* are available on the project site: http://rapache.net/manual.html, for specific comments on Mac OS X see: http://-worldofrcraft.blogspot.com/2010/08/installing-rapache-on-mac-os-x-snow.html

Following the installation of *RApache* you will have to configure Apache. Most likely you have to add something along the following lines to your `apache2.conf` or `httpd.conf` file (often found in `/etc/httpd` or `/private/etc/apache2/httpd.conf` on Mac OS X):

```
LoadModule R_module /usr/lib/apache2/modules/mod_R.so
## On Mac OS X more likely to be:
## LoadModule R_module libexec/apache2/mod_R.so
ROutputErrors
RSourceOnStartup "/var/www/rapache/R/startup.R"
## On Mac OS X the www folder is often equivalent to:
## /Library/WebServer/Documents/
```

The first line loads the R module when the Apache web server is started, the second line deals with error handling, while the `startup.R` file is suitable for initial set ups, e.g. libraries and global variables:

```
## Ensure the packages are installed so that mod_R
## has access to them, e.g. not in your home folder
library{googleVis}
library{lattice}
library{Cairo}
MyGlobalVar <- 42
```

To test that *RApache* is working open http://localhost/RApacheInfo and you should find details about your system, an example can be found on the *RApache* site: http://biostat.mc.vanderbilt.edu/rapache/files/RApacheInfo.html

The next step is to install the `brew` R package in the usual way:

```
R> install.packages('brew')
```

Following this we have to tell Apache that files in a specific folder should be parsed by `brew`. Again we edit the `apache2.conf` or `httpd.conf` and add the connection of the RHandler with the function `brew`:

```
<Directory /var/www/rapache/brew>
## On Mac OS more likely to be something like:
## <Directory /Library/WebServer/Documents/rapache/brew>
      SetHandler r-script
      RHandler brew::brew
</Directory>
```

That's all. Restart the HTTP daemon and you can start placing files in the `brew` directory and access them via `http://localhost/rapache/brew/filename`, e.g. a file containing:

```
<html>
<body>
<h1>Fruits</h1>
<% library(googleVis)
   M <- gvisMotionChart(Fruits, idvar="Fruit", timevar="Year") %>
<%= M$html$chart %>
</body>
</html>
```

You will notice that the `brew` syntax is very similar to `rsp`. For more information read the documentation of the *RApache* module and `brew` package. You find two simple examples of brew files in the `googleVis` package. Again the following R command shows you the folder path:

```
R> system.file("brew", package = "googleVis")
```

3.4.3 Using `googleVis` with Rook

Rook [Hor13] is a web server interface for R, written by Jeffrey Horner, the author of rApache and brew. Compared to other web frameworks *Rook* appears incredible lightweight. *Rook* doesn't need any configuration. It is an R package, which works out of the box with the R HTTP server. That means no configuration files

are needed. No files have to be placed in particular folders. Instead, *Rook* web applications can be run on a local desktop. However, *Rook* requires some knowledge of the HTTP protocol.

Here is a *Rook* app example with `googleVis`. It displays a little R data frame in a `googleVis` table by default. The user can change the visualisation by clicking on the *Edit me!* button and upload her/his own CSV-file, see Figure 10.

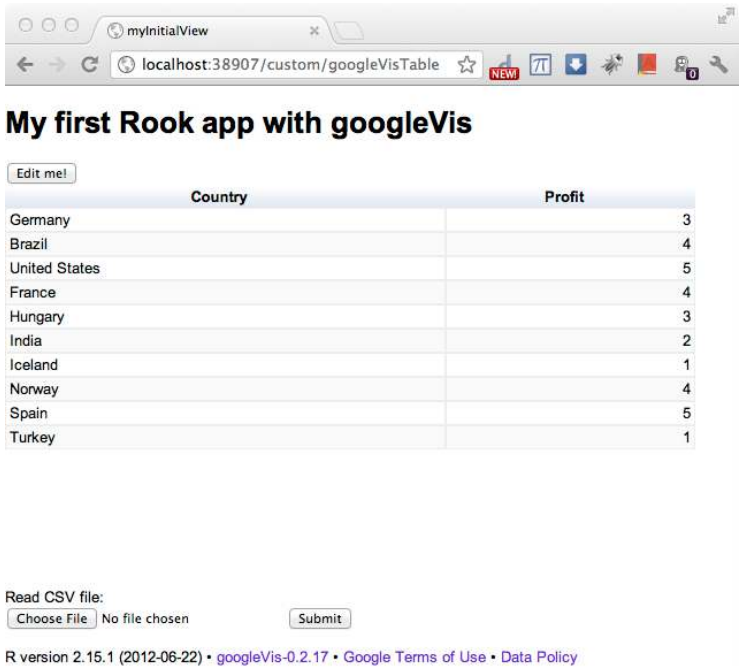

Figure 10: Screen shot of a Rook app with `googleVis` output.

```
R> require(Rook)
R> require(googleVis)
R> s <- Rhttpd$new()
R> s$start(listen='127.0.0.1')
R> my.app <- function(env){
+  ## Start with a table and allow the user to upload a CSV-file
+   req <- Request$new(env)
+   res <- Response$new()
+
+   ## Provide some data to start with
+   ## Exports is a sample data set of googleVis
+   data <- Exports[,1:2]
+   ## Add functionality to upload CSV-file
+   if (!is.null(req$POST())) {
```

```
+     ## Read data from uploaded CSV-file
+    data <- req$POST()[["data"]]
+    data <- read.csv(data$tempfile)
+   }
+   ## Create table with googleVis
+   tbl <- gvisTable(data,
+                    options=list(gvis.editor="Edit me!",
+                                 height=350),
+                                 chartid="myInitialView")
+   ## Write the HTML output and
+   ## make use of the googleVis HTML output.
+   ## See vignette('googleVis') for more details
+   res$write(tbl$html$header)
+   res$write("<h1>My first Rook app with googleVis</h1>")
+   res$write(tbl$html$chart)
+   res$write('
+ Read CSV file:<form method="POST" enctype="multipart/form-data">
+ <input type="file" name="data">
+ <input type="submit" name="Go">\n</form>')
+   res$write(tbl$html$footer)
+   res$finish()
+ }
R> s$add(app=my.app, name='googleVisTable')
R> ## Open a browser window and display the web app
R> s$browse('googleVisTable')
```

3.4.4 Using `googleVis` with `shiny`

Shiny[9] is a package by RStudio. Shiny makes it incredibly easy to build interactive web applications with R.

With version 0.4.0 of `googleVis` the support for `shiny` [RI13] apps has been added. Joe Cheng contributed the `renderGvis` function which allows users to use `googleVis` output in `shiny` in a similar way to other plotting functions. Note that `shiny` version $\geq$ 0.4.0 is required.

The following example has been taken from the help file of `renderGvis`. It displays a scatter chart where the user can select the data set to be displayed, see also the screen shot in Figure 11.

```
R> # server.R
R> library(googleVis)
R> shinyServer(function(input, output) {
+   datasetInput <- reactive({
```

[9] http://www.rstudio.com/shiny/

```
+     switch(input$dataset,
+            "rock" = rock,
+            "pressure" = pressure,
+            "cars" = cars)
+   })
+
+   output$view <- renderGvis({
+     gvisScatterChart(datasetInput())
+   })
+ })
R> # ui.R
R> shinyUI(pageWithSidebar(
+   headerPanel("googleVis on Shiny"),
+   sidebarPanel(
+     selectInput("dataset", "Choose a dataset:",
+                 choices = c("rock", "pressure", "cars"))
+   ),
+   mainPanel(
+     htmlOutput("view")
+   )
+ ))
```

You can run the example locally with the following statement.

```
R> library(shiny) ## Version >= 0.4.0 required
R> runApp(system.file("shiny/", package="googleVis"))
```

Further examples have been published on Markus' blog: `http://lamages.blogspot.co.uk/search/label/shiny`

4 Using `googleVis` with `knitr`

Using `googleVis` with `knitr` [10] [Xie13] is a convient way of creating interactive reproducible reports. The approach taken by `knitr` is similar to Sweave, you can combine R code with text and formatting tags. However, `knitr` can also export to HTML, which is required to embed `googleVis` charts.

To include `googleVis` output into a `knitr` document you have to set the chunk option `results` to `'asis'` and print the chart element only, as demonstrated in the following example:

```
```{r results='asis'}
M <- gvisMotionChart(Fruits, "Fruit", "Year",
```

[10] `http://yihui.name/knitr/`
```

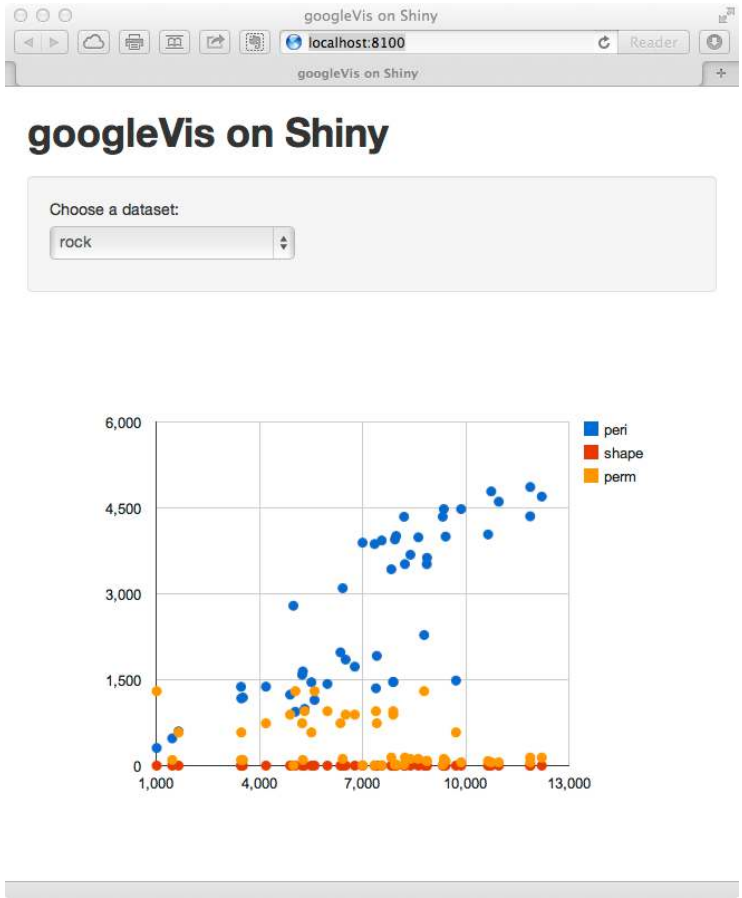

Figure 11: Screen shot a `shiny` app with `googleVis` output.

```
                    options=list(width=550, height=450))
print(M, 'chart')
```

A little example was published in a blog post: `http://lamages.blogspot.co.uk/2012/05/interactive-reports-in-r-with-knitr-and.html`

Version 0.3.2 of `googleVis` introduced `'tag'` as a new argument to the plot function `plot.gvis`, see also page 17. This argument is by default set to `NULL` and can be set globally outside the plot function via `options()`.

The argument `tag` influences the behaviour of the plot function. The `tag` parameter can be the same as for the print function `print.gvis`. Indeed, setting `options(gvis.plot.tag = 'chart')` will change the behaviour of plot to print, so `plot(x)` will no longer open a browser window, but produce the same output as

`print(x, tag='chart')`, if `x` is a gvis-object.

Hence, setting the option `gvis.plot.tag` in a `knitr` markdown Rmd-file to `'chart'` will automatically turn all following plot statements into html output, see the example below and Figure 12.

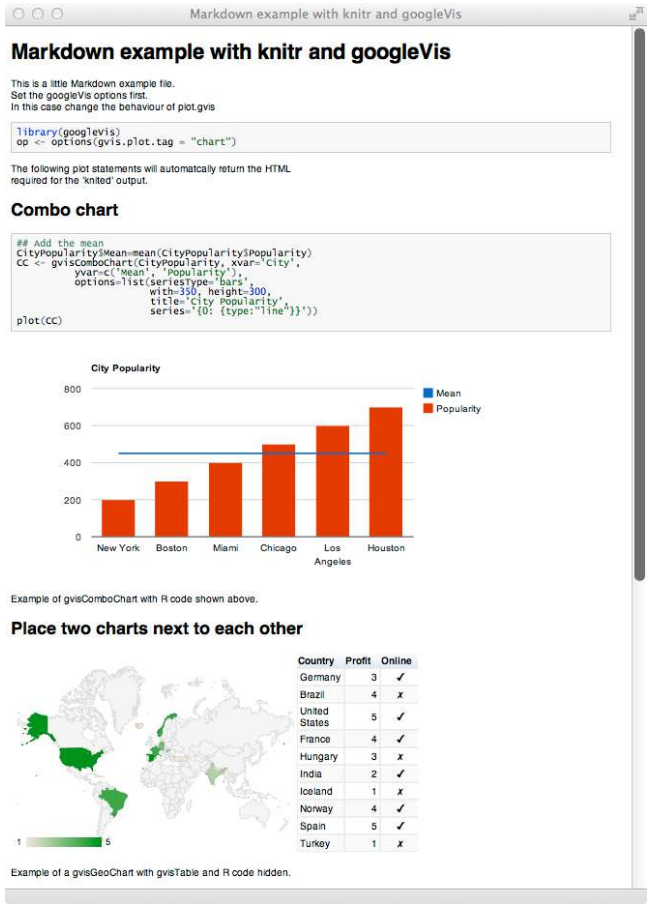

Figure 12: Screen shot of the markdown example output

```
# Markdown example with knitr and googleVis
==========================================
This is a little Markdown example file.
Set the googleVis options first.
In this case change the behaviour of plot.gvis
```{r setOptions, message=FALSE}
library(googleVis)
op <- options(gvis.plot.tag='chart')
```
```

```
The following plot statements will automatically return  the HTML
required for the 'knitted' output.

## Combo chart
```{r ComboExample, results='asis', tidy=FALSE}
## Add the mean
CityPopularity$Mean=mean(CityPopularity$Popularity)
CC <-  (CityPopularity, xvar='City',
          yvar=c('Mean', 'Popularity'),
          options=list(seriesType='bars',
                       width=450, height=300,
                       title='City Popularity',
                       series='{0: {type:\"line\"}}'))
plot(CC)
```
Example of gvisComboChart with R code shown above.

Place two charts next to each other
```{r gvisMergeExample, results='asis', echo=FALSE}
Geo <- gvisGeoChart(Exports, locationvar='Country', colorvar='Profit',
                    options=list(height=300, width=350))
Tbl <- gvisTable(Exports, options=list(height=300, width=200))
plot(gvisMerge(Geo, Tbl, horizontal=TRUE))
``````
Example of a gvisGeoChart with gvisTable and R code hidden.

Motion Chart
```{r MotionChartExample, results='asis', tidy=FALSE}
M <- gvisMotionChart(Fruits, 'Fruit', 'Year',
         options=list(width=400, height=350))
plot(M)
```
Please note that the Motion Chart is only displayed when hosted on a
web server, or is placed in a directory which has been added to the
trusted sources in the [security settings of Macromedia]
(http://www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager04.html
See the googleVis package vignette for more details.

```{r resetOptions}
## Set options back to original options
options(op)
```
```

Note that you can use the `options()` command in your `knitr` file to switch between an interactive mode, where you are likely to experiment, via copying and pasting R code into the console and running `knit` on the whole file.

A more comprehensive example is given in the help file to `?plot.gvis`.

5 Using `googleVis` in presentations

The Google Chart Tools are designed for web pages, so it should be no surprise that it can be difficult or impossible to embed `googleVis` output in traditional presentation software like MS PowerPoint[11], Google Docs, OpenOffice Impress or Apple Keynote.

The easiest way is to include screen shots into the slide with links to the live web pages. But this approach requires the presenter to switch between applications during her talk. This can be fun, but quite often it is not.

An alternative would be to build the presentation as a web page itself.

The JavaScript library deck.js [Tro11], a jQuery framework for creating HTML presentations by Caleb Troughton, provides an elegant way of combining the slide paradigm with web content such as YouTube videos or `googleVis` output. You find an example with `googleVis` charts on mages' blog: `http://lamages.blogspot.com/2011/11/interactive-presentations.html` .

Figure 13: `googleVis` presentation with deck.js: `http://dl.dropbox.com/u/7586336/blogger/deck.js/googleVis/index.html`

An alternative to deck.js is slidy[12] in combination with `knitr` and pandoc[13], see the following blog post for more details: `http://lamages.blogspot.co.uk/2012/05/interactive-html-presentation-with-r.html`. The basic idea is that you create a Markdown file with `knitr` and you post-process the output with pandoc into a slidy HTML presentation.

6 Beyond R

In this section we present ideas which go beyond the usual coding in R and are somewhat experimental.

[11] For MS PowerPoint on MS Windows the plug-in liveweb provides the functionality to insert web pages into a PowerPoint slide and refresh the pages real-time during slide show.

[12] `http://www.w3.org/Talks/Tools/Slidy2/`

[13] `http://johnmacfarlane.net/pandoc/`

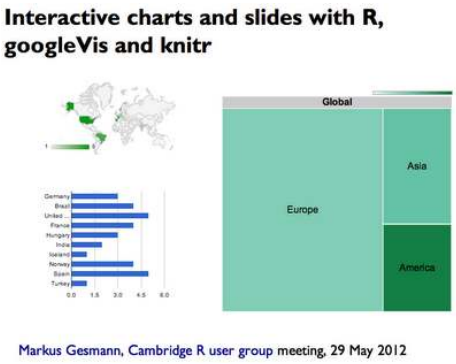

Figure 14: `googleVis` presentation with slidy: `http://lamages.blogspot.co.uk/2012/05/interactive-html-presentation-with-r.html`

6.1 Registering to catch events

Google visualisations can fire and receive events [14]. It exposes the following two JavaScript methods:

- `google.visualization.events.trigger()` fires an event,
- `google.visualization.events.addListener()` listens for events.

Here is an example of registering to receive the selection event from the Google documentation:

```
var table = new google.visualization.Table(document.getElementById('table_div'));
table.draw(data, options);
google.visualization.events.addListener(table, 'select', selectHandler);

function selectHandler() {
  alert('A table row was selected');
}
```

We will only deal with this special case of a 'select' event of the 'addListner' method. This event is available for most visualisations and acts on user interactions, e.g. user selection clicks.

The 'addListener' method expects JavaScript code, which can be embedded into a gvis-object via `options` as (undocumented) parameter `gvis.listener.jscode`.

Here are some examples:

Look up the selected item in Wikipedia:

[14] `http://code.google.com/apis/chart/interactive/docs/reference.html#addlistener`

```
R> jscode <- "window.open('http://en.wikipedia.org/wiki/'
+                   + data.getValue(chart.getSelection()[0].row,0)); "
R> J1 <- gvisGeoMap(Exports, locationvar='Country', numvar='Profit',
+                  options=list(dataMode="regions", gvis.listener.jscode=jscode))
R> plot(J1)
```

In the same way we can use the code in other charts, e.g. org- or line chart:

```
R> plot(gvisOrgChart(Regions,  options=list(gvis.listener.jscode=jscode)))
R> plot(gvisLineChart(Regions[,c(1,3)], options=list(gvis.listener.jscode=jscode)))
```

In the following more advanced example the selected value of a table is displayed in a message box:

```
R> jscode <- "
+        var sel = chart.getSelection();
+        var row = sel[0].row;
+        var text = data.getValue(row,1);
+        alert(text);
+ "
R> J2 <- gvisTable(Population, options=list(gvis.listener.jscode=jscode))
R> plot(J2)
```

For more details see the `demo(EventListener)` and Google Chart Tools Reference.

7 Frequent ask questions – FAQ

7.1 Can I use googleVis output in PDF files?

No, not directly.

The Google Chart Tools API is designed for dynamic web output on your screen and not on paper. See also the section 5, page 34, which covers how to use `googleVis` in presentations.

Note that this vignette uses screen shots to include examples of `googleVis` charts.

7.2 Can I change the colour of the bubbles in motion charts?

No, unfortunately not.

The colours are set by the Google Chart Tools API and cannot be changed by the user.

7.3 Can I change the icons in Maps?

No, not via `gvisMap`.

However, you can upload your data to a Google Fusion Table and link it to a second table which defines the icons. See the Google Fusion Table support page for more details: `http://support.google.com/fusiontables/answer/2679986?hl=en`

7.4 Why can't I see motion charts when I open them from a local directory?

See section 2.4 on page 16

7.5 Why can't I see motion charts on my iPad/iPhone?

Motion charts (also geo maps and annotated time lines) are rendered in your browser using Flash, unlike most other charts which use HTML5. Unfortunately Flash is not directly supported on iOS devices such as iPads and iPhones.

Apparently there is little appetite at Google to migrate motion charts to HTML5 or other technologies, see the following Google Visualisation API news group post: `http://goo.gl/5Zyuw`.

7.6 How can I set axes limits with googleVis?

Unfortunately, there are no arguments such as `ylim` and `xlim`. Instead, the Google Charts axes options are set via `hAxes` and `vAxes`, with h and v indicating the horizontal and vertical axis. More precisely, we have to set `viewWindowMode : 'explicit'` and set the `viewWindow` to the desired min and max values. Additionally, we have to wrap all of this in [{}] brackets as those settings are sub-options of `h/vAxes`. There are also options `minValue` and `maxValue`, but they only allow you to extend the axes ranges.

Here is a minimal example, setting the y-axis limits from 0 to 10:

```
R> library(googleVis)
R> dat <- data.frame(x=LETTERS[1:10],
+                    y=c(0, 4, -2, 2, 4, 3, 8, 15, 10, 4))
R> area1 <- gvisAreaChart(xvar="x", yvar="y", data=dat,
+             options=list(vAxes="[{viewWindowMode:'explicit',
+                          viewWindow:{min:0, max:10}}]",
+                          width=500, height=400,
+                          title="y-limits set from 0 to 10"),
+                          chartid="area1ylim")
R> plot(area1)
```

7.7 How can I change the look and feel of the charts?

The charts have a lot of options which allow you to change the look and feel of the output, see the help files for more details. However, `googleVis` provides only an interface to the Google Chart Tools. If you have specific questions to the charts then please join the Google Visualisation API newsgroup: `http://groups.google.com/group/google-visualization-api`.

For frequent ask questions regarding the API check: `https://developers.google.com/chart/interactive/faq`.

7.8 Is it possible to use `googleVis` in corporate work?

Review the Google Terms of Service (`https://developers.google.com/terms/`) and get in touch with your colleagues in IT / Legal. If in doubt contact Google directly.

7.9 Bugs and issues

Should you find any issues or bugs with googleVis, then please drop us a line or add them to our issues list: `http://code.google.com/p/google-motion-charts-with-r/issues/list`

8 Contact

8.1 Collaboration

Obviously, the package is work in progress and there are many other functions of the Google Chart Tools which are still untouched.

Please feel free to send us an email if you would like to be kept informed of new versions, or if you have any feedback, ideas, suggestions or would like to collaborate, our address is rvisualisation@gmail.com.

8.2 Citation

Please cite R and `googleVis` if you use it in your work or publications. Use

```
R> citation("googleVis")
```

and

```
R> citation()
```

for bib-entries and information on how to cite the software.

8.3 Training and consultancy

Please contact us if you would like to discuss training or consultancy: rvisualisation@gmail.com

References

[Ben12] Henrik Bengtsson. R.rsp: R server pages. http://CRAN.R-project.org/package=R.rsp, 2012. R package version 0.8.2.

[Bos12] Mike Bostock. Data-Driven Documents (d3.js), a visualization framework for internet browsers running JavaScript. http://d3js.org, 2012.

[Fou10a] Apache Foundation. Apache HTTP Server 2.2. http://httpd.apache.org, 2010.

[Fou10b] Gapminder Foundation. Gapminder. http://www.gapminder.org, 2010.

[Fou10c] Open Source Geospatial Foundation. Openlayers: Free maps for the web. http://www.openlayers.org/, 2010.

[FR10] Ben Fry and Casey Reas. Processing an open source programming language and environment to create images, animations, and interactions. http://processing.org/, 2010.

[GdC11] Markus Gesmann and Diego de Castillo. Using the Google Visualisation API with R. *The R Journal*, 3(2):40–44, December 2011.

[GdC13] Markus Gesmann and Diego de Castillo. googleVis: Using the Google Chart Tools with R. http://code.google.com/p/google-motion-charts-with-r/, 2013. R package version 0.4.4.

[GHN10] John Glazebrook, Guenther Harrasser, and George Neusse. Open flash chart. http://teethgrinder.co.uk/open-flash-chart/, 2010.

[Hor11a] Jeffrey Horner. brew: Templating framework for report generation. http://CRAN.R-project.org/package=brew, 2011. R package version 1.0-6.

[Hor11b] Jeffrey Horner. RApache: Web application development with R and Apache. http://www.rapache.net/, 2011.

[Hor13] Jeffrey Horner. *Rook: Rook - a web server interface for R*, 2013. R package version 1.0-9.

[Inc12a] Google Inc. Google API Terms of Service. https://developers.google.com/terms/, 2012.

[Inc12b] Google Inc. Google Chart Tools. https://google-developers.appspot.com/chart/interactive/docs/gallery, 2012.

[Inc12c] Google Inc. Google Motion Chart API. https://google-developers.appspot.com/chart/interactive/docs/gallery/motionchart.html, 2012.

[Inc12d] Google Inc. Google Public Data Explorer. http://www.google.com/publicdata/home, 2012.

[Inc12e] Google Inc. Google Visualisation Reference. https://developers.google.com/chart/interactive/docs/reference, 2012.

[JSO06] JSON.org. JSON. http://www.json.org/, 2006. RFC 4627 application/json.

[Lab10] UC Berkeley Visualization Lab. flare: Data visualisation for the web. http://flare.prefuse.org, 2010.

[Lan12] Duncan Temple Lang. RJSONIO: Serialize R objects to JSON, JavaScript Object Notation. http://www.omegahat.org/RJSONIO/, 2012. R package version 1.0-1.

[RI13] RStudio and Inc. *shiny: Web Application Framework for R*, 2013. R package version 0.4.0.

[Ros06] Hans Rosling. TED Talk: Hans Rosling shows the best stats you've ever seen. http://www.ted.com/talks/hans_rosling_shows_the_best_stats_you_ve_ever_seen.html, 2006.

[RtICsg10] IBM Research and the IBM Cognos software group. Many eyes. http://www-958.ibm.com/software/data/cognos/manyeyes/, 2010.

[Saa10] Sebastián Pérez Saaibi. *R/RMETRICS Generator Tool for Google Motion Charts*. https://www.rmetrics.org/, 2010. Meielisalp, Lake Thune Switzerland, June 27 - July 1, 2010.

[Tro11] Caleb Troughton. *deck.js: Modern HTML Presentations*, 2011. jQuery framework for creating HTML presentations.

[Xie13] Yihui Xie. *knitr: A general-purpose package for dynamic report generation in R*, 2013. R package version 1.1.