R／RStudioでやさしく学ぶ プログラミングとデータ分析

【ダウンロード特典】

※マイナビ出版発行の『R／RStudioでやさしく学ぶプログラミングとデータ分析』(掌田津耶乃 著)の紙版または電子版をご購入いただいた方のみの特典です。デバイスの台数は問わずご利用いただけます。
購入者以外の第三者に渡す行為は禁止されていますのでご注意ください。

leafletによるマップ表示

難易度：★☆☆☆☆

leafletでマップを表示する

最後に、グラフとは少し違いますが、データを視覚化する際に覚えておきたい「マップ」の利用についても触れておきましょう。

Googleマップなどのマップ機能を利用してさまざまなデータを視覚化していることはよくあります。例えばCOVID-19の都道府県ごとの感染者数などは、一覧表で見るよりマップ上に円などの大きさで表示されたほうが直感的にわかるでしょう。こうした用途のためにマップの基本的な使い方ぐらいは覚えておきたいところですね。

マップの利用は、「leaflet」というパッケージを使います。このパッケージは、グラフを表示するのに使われる「Plots」ではなく、HTMLの表示としてマップを表示します。これはRStudioでは「Viewer」ペインになり、RGuiではWebブラウザのウィンドウを使って表示されます。このため、これまでのグラフ関係の関数で利用されていたような機能とは基本的な機能が違います。例えばグラフでadd =TRUEで別のグラフを重ねたり、といったことはleafletのマップでは行えません。この点、注意してください。

leafletを利用するためには、まず以下の文を実行してパッケージをインストールします。これは最初に一度だけ実行すれば、以後は実行する必要はありません。

リストA-1-1

```
01 install.packages("leaflet")
```

実際にRコンソールやスクリプトでleafletを利用する際は、以下の文を実行してパッケージをロードしておきます。

リストA-1-2

```
01 library(leaflet)
```

これでleafletの機能が使えるようになります。libraryを実行すれば、そのスクリプトでは、以後この文を実行する必要はありません。

leafletの基本

では、マップ表示の手順を説明しましょう。これは大きく3つの作業が必要になります。最初に行うのは、leafletオブジェクトの作成です。

書式 leafletオブジェクトを作成する

```
leaflet()
```

これでオブジェクトが作成されます。このオブジェクトは、"leaflet"と"htmlwidget"というクラスとして作成されます。leafletクラスに、マップ関連の機能がまとめられています。
オブジェクトを作成したら、次に「タイルレイヤー」を追加します。

書式 タイルレイヤーを追加する

```
addTiles(マップ)
```

タイルレイヤーというのは、実際に表示されるマップのレイヤーのことです。leafletでは、オブジェクト内にマップを表示するタイルレイヤーを追加して表示をします。addTilesは、指定のマップ(leafletオブジェクト)にデフォルトのマップレイヤーを追加します。これは、OSM (Open Street Map)によるマップレイヤーです。
これでマップ自体は表示されるようになります。しかし、この状態では地球全体が表示された状態なので、やはり表示したい場所を指定し、適切にズームした状態で表示させたいですね。これを行うのが以下の関数です。

書式 表示したい場所を指定する

```
setView(マップ, lat = 緯度, lng = 経度, zoom = ズーム)
```

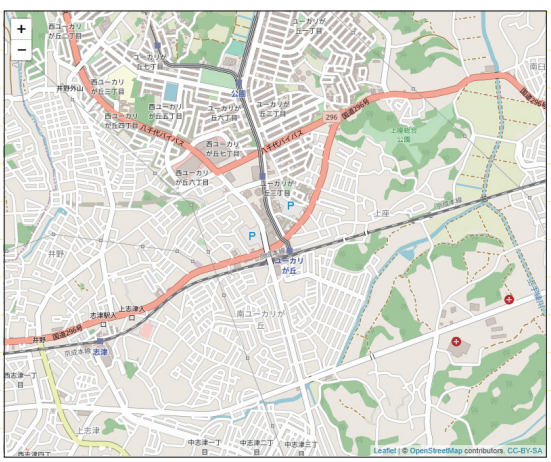
latとlngには、緯度と経度の値を指定します。この位置がマップの中心となるように表示されます。またzoomは、1～18の整数を指定します。1が地球全体となり、大きくなるほどズームアップしていきます。

マップを表示する

では、実際にマップを表示してみましょう。以下のスクリプトを実行してみてください。佐倉市周辺のマップが表示されます。

リストA-1-3

```
01 m <- leaflet()
02 m <- addTiles(m)
03 m <- setView(m, lat = 35.723, lng =140.156, zoom = 15)
04 m
```



実行すると佐倉市ユーカリが丘のマップが表示される

ここでは、leafletで作成したオブジェクトを変数mに代入しています。そして、addTilesでタイルレイヤーを追加したものをmに代入し直し、さらにsetViewしたものをmに代入し直します。addTilesやsetViewは、引数のマップの内容を直接書き換えるわけではありません。ですから返された値を変数に入れ直して利用することになります。最後に完成したマップのmを出力すれば、マップが表示されます。

パイプ演算子による書き方

このやり方は、Rのスタンダードな書き方によるマップ表示の基本です。ただ、このやり方は、なんだかまどろっこしい感じがしますね。毎回、mを引数にして結果をまたmに代入して、というのはどうもスマートではありません。Rでは、このように「あるオブジェクトを引数にして関数を呼び出し、その結果をまたオブジェクトの変数に入れ直す」といった作業を行う場合、非常に便利な演算子が用意されています。それは「%>%」というものです。

HINT

%>%は、dplyrパッケージに用意されている機能です。利用の際はdplyrパッケージを用意する必要があります。dplyrについては、「4-10 dplyrパッケージによるフィルター処理」を参照ください。

この%>%という記号は「パイプ演算子」と呼ばれるものです。この%>%演算子は以下のように使います。

書式 パイプ演算子

```
オブジェクト %>% 関数
```

この演算子は、左辺の値を右辺の関数に渡し、結果を再び左辺に返します。この%>% 演算子を使用することで、面倒な記述をシンプルに記述できるようになります。例えば、こんなスクリプトがあったとしましょう。

```
変数1 <- function1(値1)
変数2 <- function2(変数1)
変数3 <- function3(変数2)
```

function1、function2、function3といった関数があって、関数の戻り値を次の関数の引数にして呼び出す、といったことを繰り返していますね。
上記のような記述を、%>%演算子を使用すれば以下のように書き直すことができます。

```
変数 <- function1(値1) %>% function2() %>% function3()
```

%>%演算子を使用することで、関数を連続して適用する場合に、より簡潔な記述が可能になります。ただ連続して関数を呼び出せるだけでなく、渡すべき引数が省略され、さらにシンプルになっているのがわかりますね。%>%演算子を使用することで、関数の結果を変数に格納する必要がなくなり、コードが簡潔になるのです。

では、このパイプ演算子というものを使って、先ほどのマップ表示のスクリプトを書き換えてみましょう。すると次のようになります。

リストA-1-4

```
01 leaflet() %>%
02   addTiles() %>%
03   setView(m, lat = 35.723, lng =140.156, zoom = 15)
```

これで、先ほどと全く同じマップが表示されます。%>%を使うことで、ぐっとスクリプトがわかりやすくなりますね！

マップに追加できる要素

マップには、さまざまな要素を追加することができます。それらにより、さまざまな情報をマップに表示できるようになります。

まずは「マーカー」から説明しましょう。マーカーは、マップの特定の場所にマークを付けるためのものですね。これは「addMarkers」という関数で作成できます。

書式 マーカーを設定する

```
addMarkders(マップ, lat = 緯度, lng = 経度, label = ラベル)
```

引数には、leafletで作成したマップのオブジェクトを指定します。これは、すでに説明したように%>%演算子で連続呼び出しする場合は省略できます。lat、lngでマーカーを表示する位置を指定し、labelではマーカーにマウスポインタがあるときにツールチップで表示されるテキストを指定します。これは不要なら省略しても構いません。

これらは、それぞれ1つの値だけを指定することもできますが、ベクトルで指定することで複数のマーカーをまとめて設定することもできます。この場合、それぞれの引数に用意するベクトルの成分の数をきちんと揃えておくようにしてください。

では、実際に簡単な利用例を挙げておきましょう。

リストA-1-5

```
01 leaflet() %>%
02   addTiles() %>%
03   setView(m, lat = 35.723, lng =140.156,
04           zoom = 15) %>%
05   addMarkers(lat = 35.723, lng =140.156,
06              label = "これはマーカーです！")
```

実行すると、マップの中央付近にマーカーが表示されます。マーカーの上にマウスポイントを移動すると、「これはマーカーです！」とツールチップが表示されます。

ここではsetViewの後に%>%をつけ、続けてaddMarkersを呼び出しています。最初に引数として用意するマップは省略され、lat、lng、labelといった値だけ用意すればマーカーを表示できるのです。非常に簡単ですね！

マップにマーカーが1つ追加される

マーカーの表示オプション

マーカーでは、マウスポインタによりツールチップが表示できます。これは「ラベル」とよばれるもので、オプションを指定することで表示場所や表示する内容などを細かく設定できます。

ラベルのオプションは、「labelOptions」という引数で用意します。これはlabelOptionsという関数を使って値を作成します。この関数は、オプションに関する引数が多数用意されています。

書式 labelOptionsオプション

```
labelOptions(
    direction = 位置,
    noHide = 論理値,
    offset = c(横, 縦),
    textsize = サイズ,
    style = スタイル(labelOptions)
    ……略……
)
```

主な引数をまとめると、このようになるでしょう。こうした引数を用意したlabelOptionsをlabelOptionという引数に設定することで、ラベルの表示を設定できます。

では、これも実行例を挙げておきましょう。

リストA-1-6

```
01 leaflet() %>%
02   addTiles() %>%
03   setView(m, lat = 35.723, lng =140.156,
04           zoom = 15) %>%
05   addMarkers(lat = 35.723, lng =140.156,
06              label = "これはマーカーです！",
07              labelOptions = labelOptions(
08                direction = "bottom",
09                noHide = TRUE,
10                offset = c(0, 25),
11                textsize = "18pt",
12                style = labelOptions(color = "red")
13              ))
```

マーカーの下に赤いテキストでラベルが表示される

実行すると、マーカーの下に「これはマーカーです！」という赤いテキストのラベルが表示されます。
ここでは、labelOptions関数に多数の引数を用意していますね。用意されている値について簡単に説明しておきましょう。

labelOptions関数のオプション

値	内容
direction	ラベルの位置。これはテキストで指定します。"bottom"でマーカー下に表示させています
noHide	常時表示させるためのものです。TRUEにすることで常に表示されます
offset	表示位置からのオフセット(位置をずらす幅)を指定します。c(0, 25)で、下に25だけずらしてあります
textsize	テキストのサイズ指定です。これは数値ではなく、Webで使われるスタイルシートで使われる値で指定します。ここでは"18pt"にしてあります
style	その他の細かな表示は、スタイルシートの値をstyleにまとめて指定します。これは、labelOptions関数を使って指定します。今回はcolor = "red"というようにcolorスタイルを"red"にして赤いテキストにしています

わかりにくいのは、styleの設定でしょう。ここでは、設定したいスタイルとその値をlabelOptions関数でまとめます。leafletのマップはHTMLのWebページとして作成されるので、表示などはスタイルシートで調整できるのです。

吹き出し(ポップアップ)の表示

マップ内にテキストの説明などを表示したい場合、マーカーのラベルなどよりも「ポップアップ」を作成したほうがいいでしょう。ポップアップは、マップに表示される「吹き出し」のような部品です。ここにテキストを表示することで、長いテキストなども表示できます。
このポップアップは、「addPopups」関数で作成できます。

書式 ポップアップを作成する

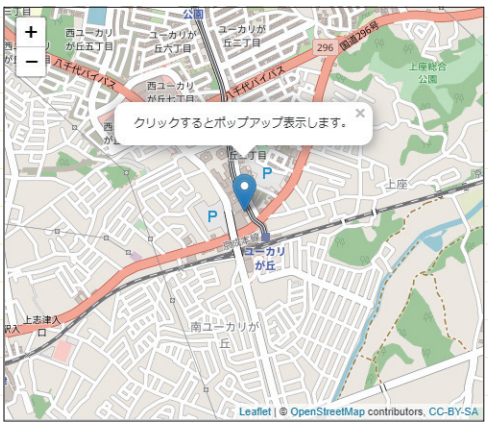
```
addPopups(マップ, lat = 緯度, lng = 経度, popup = 表示内容)
```

基本的な引数は、先ほどのaddMarkersに用意したものとほぼ同じですね。こちらでは、popupというオプションに、表示するテキストを用意すれば、それがポップアップとして表示されます。このaddPopupsも、引数はすべてベクトルを指定することができます。これにより、同時に複数のポップアップを表示することができます。

では、これも利用例を挙げておきましょう。

リストA-1-7

```
leaflet() %>%
  addTiles() %>%
  setView(m, lat = 35.723, lng =140.156,
          zoom = 15) %>%
  addMarkers(lat = 35.723, lng =140.156,
             label = "これはマーカーです！") %>%
  addPopups(lat = 35.725, lng =140.156,
            popup = "クリックするとポップアップ表示します。")
```



マーカーの上にポップアップが表示される

実行すると、マーカーの少し上にポップアップが表示されます。ポップアップにはクローズボックスがあり、クローズボックスや他の場所をクリックすると閉じて消えます。

マーカーをクリックするとポップアップする

ポップアップは、クリックすると消えてしまい、もう表示されません。一時的に表示するものは、最初から表示しておくより、どこかをクリックしたら必要に応じて現れるほうが便利でしょう。

ポップアップは、他の部品をクリックして呼び出すこともできます。部品を作成する関数に「popup」引数を用意することで、その部品をクリックしたらポップアップが現れるようにできるのです。

例として、先ほどのaddMarkersで作成したマーカーをクリックしたらポップアップが現れるようにしてみましょう。

リストA-1-8

```
leaflet() %>%
  addTiles() %>%
  setView(m, lat = 35.723, lng =140.156,
          zoom = 15) %>%
  addMarkers(lat = 35.723, lng =140.156,
             label = "これはマーカーです！",
             popup = paste('<p>イメージも表示できます。</p>',
               '<div><img src="https://picsum.photos/100"/></div>')
             )
```

マーカーをクリックするとポップアップが現れる

マップに追加されたマーカーをクリックすると、その上にポップアップが現れます。このポップアップでは、テキストだけでなくイメージも表示されます。

ここでは、addMarkers関数にpopupという引数を用意し、そこに表示するテキストを指定していますね。このテキストは、ここではHTMLのソースコードとして作成しています。popupでは、HTMLのコードが使えるのです。ここでは<img>を使ってイメージを表示させています。HTMLが使えると、Webページで表示されるさまざまなものをポップアップに追加できるようになります。

図形を追加する

その他、簡単な図形をマップに追加することもできます。ここでは「円」「四角形」「直線」「多角形」についてまとめておきましょう。

書式 円の描画

```
addCircles(マップ, lat = 緯度, lng = 経度, radius = 半径)
```

書式 四角形の描画

```
addRectangles(マップ, lat1 = 緯度, lng1 = 経度,  lat2 = 緯度, lng2 = 経度)
```

書式 直線の描画

```
addPolylines(マップ, lat = 緯度, lng = 経度)
```

書式 多角形の描画

```
addPolygons(マップ, lat = 緯度, lng = 経度)
```

これらは表示場所や大きさに関する引数を用意すると、その場所に図形を作成します。addCirclesは、位置と大きさ(半径)を指定するだけですから、一番簡単でしょう。addRectanglesは、描画する四角形の2つの対角の位置を指定します。addPolylinesとaddPolygonsでは、図形の角の位置をベクトルにまとめたものをlat、lngに用意します。

これらの図形を描く関数で共通して使えるオプションとして、以下のようなものも用意されています。

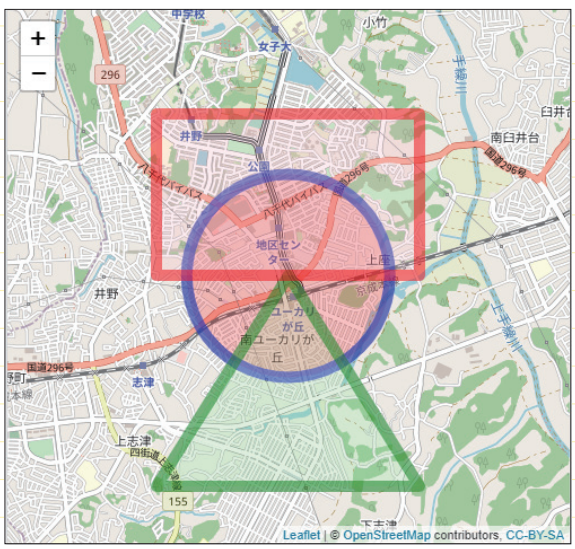
addRectangles関数とaddPolygon関数sのオプション

color	図形の色
fillColor	内部の塗りつぶし色
stroke	輪郭線のON/OFF（TRUEなら表示）
weight	輪郭線の太さ
opacity	透過度（0.0～1.0）

これらを指定することで、図形の色や輪郭線の表示を調整することができます。では、これらの利用例を挙げておきましょう。

リストA-1-9

```
leaflet() %>%
  addTiles() %>%
  setView(m, lat = 35.723, lng =140.156,
          zoom = 14) %>%
  addCircles(lat = 35.723, lng =140.156, radius = 700, color = "blue",
             stroke = T, fillColor = "red", weight = 10) %>%
  addPolygons(lat = c(35.723, 35.71, 35.71),
              lng = c(140.156, 140.166, 140.146),
              weight = 10, color = "green", fillColor = "lightgreen") %>%
  addRectangles(lat1 = 35.733, lng1 =140.146,
                lat2 = 35.723, lng2 =140.166,
                color = "red", fillColor = "pink", weight = 10)
```



マップ上に四角形、円、三角形を表示する

これを実行すると、マップに赤の四角形、青い円、緑の三角形が表示されます。ここでは、addCircles、addPolygons、addRectanglesといった関数を呼び出しています。これらを使って図形を描いているのがわかるでしょう。いずれも、使っている引数はほとんど同じものですから、どれか1つ使えるようになれば、残りの関数もすぐに使えるようになります。

マップでデータを表示する

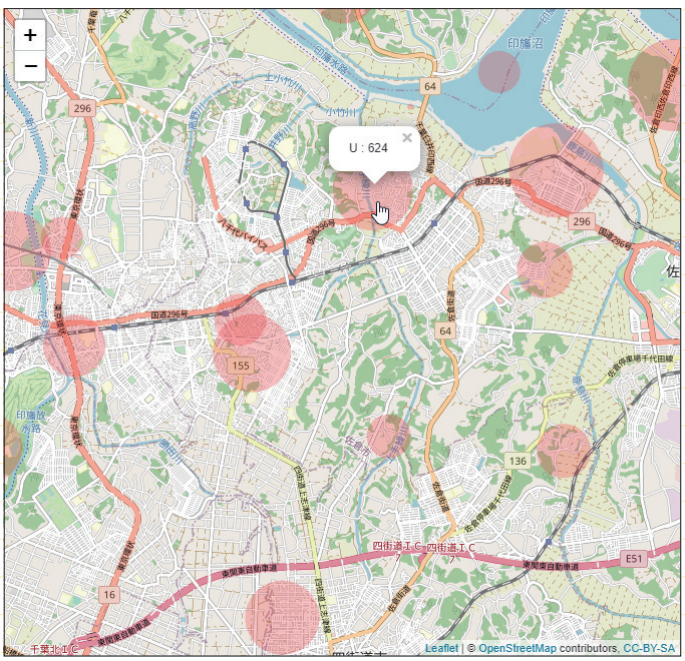
これで、leafletによるマップの基本的な利用方法はわかりました。では、ここまでの知識を活用し、マップを使ってデータを視覚化する方法を考えてみましょう。

各地のデータを視覚的に表示したい、という場合、もっとも簡単なのはaddCirclesによる円でデータの大きさを表現するものでしょう。値の大きさに応じて円の半径を変えたり、あるいはcolorによる色やopacityによる透過度などを変更すれば、それだけでデータを視覚化できます。

では、簡単なサンプルを作成してみましょう。

リストA-1-10

```
01 data.rd <- as.integer(rnorm(25,mean=500,sd=250))
02 data.df <- data.frame(
03   lat = rnorm(25,mean=35.72,sd=0.05),
04   lng = rnorm(25,mean=140.15,sd=0.05),
05   rd = data.rd,
06   pop = paste(LETTERS[1:25], " : ", data.rd)
07 )
08
09 leaflet() %>%
10   addTiles() %>%
11   setView(lng =140.156, lat = 35.723, zoom = 17) %>%
12   addCircles(data.df$lng, data.df$lat, color="red",
13              radius = data.df$rd, stroke=FALSE,
14              popup = data.df$pop)
```



ランダムに作成したデータを円として表示する

実行すると、ランダムに作成したデータをもとに25個の円をマップに描きます。円をクリックすると、その円の名前と値が表示されます。いわゆるバブルチャートをマップに重ね合わせたようなものが作成できることがわかります。

ここでは、あらかじめデータをデータフレームの形にまとめてあります。多数の項目を持つ多量のデータをわかりやすく管理するには、データフレームを利用するのが最適です。ここで作成したdata.dfには、lat、lng、rd、popといった列が用意され、ここに緯度、経度、半径、ポップアップの内容がそれぞれまとめられています。leafletでは、addCircles関数でこれらの値を引数に指定して円を描いているのです。

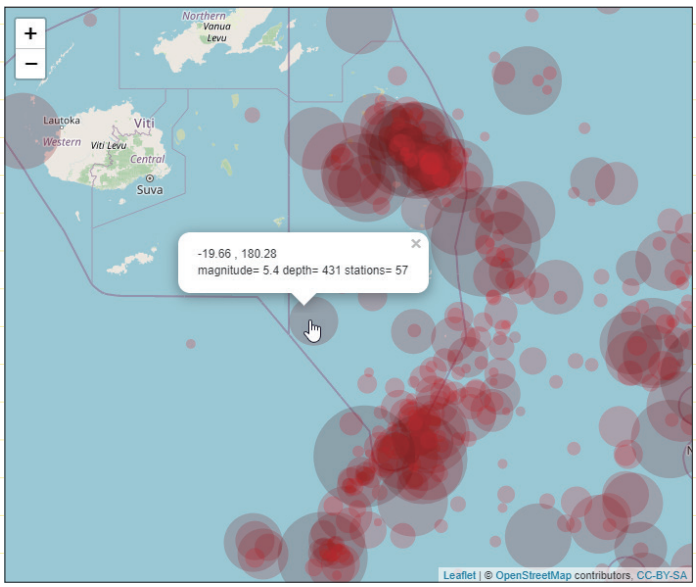
popupを用意しているので、円をクリックすればその円のデータが表示されます。マップでは直感的にわかるような図形を表示し、具体的なデータはクリックすれば表示される、というのがマップでデータを扱う場合の基本となるでしょう。

quakesの地震データを可視化する

では、実際にデータをマップで可視化してみましょう。例として、地震のデータセットである「quakes」の情報を可視化してみることにします。

リストA-1-11

```
01 data.popup <- paste(quakes$lat,",",quakes$long,
02                     "<br>magnitude=", quakes$mag,
03                     " depth=", quakes$depth,
04                     " stations=", quakes$stations)
05
06 leaflet() %>%
07   addTiles() %>%
08   setView(lng =quakes$long[1], lat = quakes$lat[1], zoom = 7) %>%
09   addCircles(lat = quakes$lat, lng = quakes$long,
10              color= rgb(1 - 1 / 2.5 * (quakes$mag - 4), 0, 0),
11              radius = quakes$stations * 500, stroke=FALSE,
12              popup = data.popup)
```



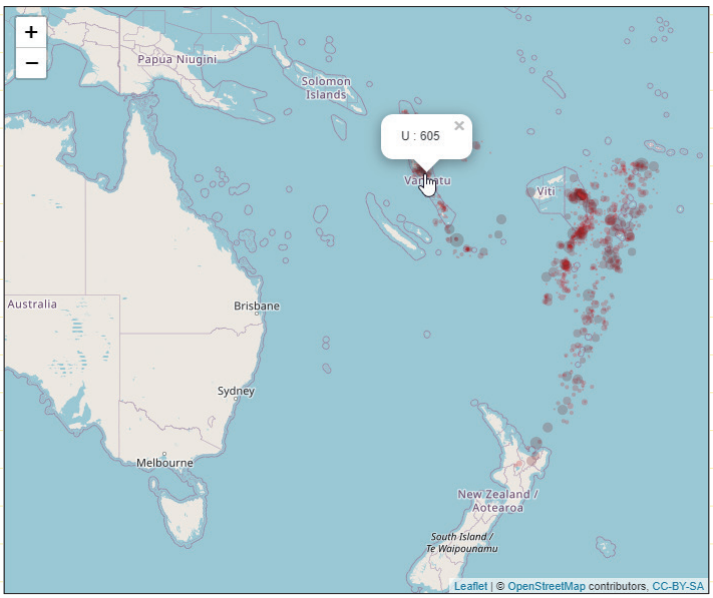
地震の大きさが円で表示される

実行すると、マップに多数の円が描かれます。quakesでは全部で1000個のデータがありますから、それらすべてがaddCirclesで円として描かれているのです。

ここでは、lat、lngにはquakesの$latと$lngの値をそのまま指定しています。colorには、rgb(1 - 1 / 2.5 * (quakes$mag - 4), 0, 0)という値が設定されていますね。これは$magの値に応じてrgbの赤の輝度が変わるようにしているのです。また円の半径であるradiusは、quakes$stations * 500として$stationsの数をもとに指定しています。

popupで表示するポップアップの内容は、あらかじめdata.popupという変数にデータを用意してあります。ここでは、paste関数を使い、quakes内の各データの値を1つのテキストにまとめています。これにより、クリックするとその地震のデータがすべて表示されるようになっていたのです。

表示されたマップをどんどん縮小していくと、ニュージーランドの上にあるバヌアツやトンガ、フィジーなどの地震データであることがわかります。quakesのデータを眺めただけでは、どこでどんなに大きな地震が起きていたかわからなかったことでしょう。マップを利用することで、データをもっと具体的な形で理解できるようになります。



地図を縮小すると、フィジー諸島周辺の地震データだったことがわかる