

Desenvolvimento de *Data Products*
Brian Caffo e Sean Kross

Traduzido por Rodrigo Hermont Ozon*

Junho, 2020

*Economista e Mestre em Desenvolvimento Econômico pela UFPR.



Sobre o autor desta tradução:

Rodrigo Hermont Ozon, economista e apaixonado por econometria, pelas aplicações de modelos econômicos a problemas reais e cotidianos vivenciados na sociedade e na realidade do mundo empresarial e corporativo.

Seus contatos podem ser acessados em:

- [Github](#)
- [Linkedin](#)

Resumo

Esta tradução tem por finalidade contribuir com aqueles profissionais que trabalham com *data science* e que precisam de alternativas mais robustas e sólidas para prestação de seus serviços, entregas e aplicações. Este curso ensina como criar ferramentas para melhorar o processo de análise de dados, tomar decisões orientadas por dados ou para outra infraestrutura que suporte outros produtos de dados.

Este e-book foi escrito no [overleaf](#) com o pacote knitr para o e-book Developing Data Products disponível em [seankross.com](#).

A minha amada família e ao Pedro e a Zoe, – ”*Os filhos são herança do Senhor, uma recompensa que ele dá..*”

[Salmo 127:3](#)

Sumário

0.1 Sobre esse livro	7
0.2 O que é data product ?	7
0.3 O objetivo deste livro	7
1 R Markdown	8
1.1 Introdução ao R Markdown	8
1.2 Markdown: Sintaxe básica	8
1.2.1 Cabeçalhos	8
1.2.2 Texto	8
1.2.3 Negrito e itálico	8
1.2.4 Listas	8
1.2.5 Links	9
1.2.6 Imagens	9
1.2.7 Citações	9
1.2.8 Códigos	9
1.2.9 Código no corpo do texto	9
1.2.10 Códigos do  no Markdown	9
1.3 Escrevendo e renderizando documentos	9
1.4 Criando apresentações no R Markdown	11
1.5 Compartilhando documentos em R Markdown	11
1.6 Considerações finais	11
2 Shiny	12
2.1 Instalando o Shiny	12
2.2 O seu primeiro ShinyApp	12
2.3 Mais elementos UI	14
2.4 Inputs e Outputs	14
2.5 UI dinâmica	17
2.6 Reatividade	19
2.7 Interatividade	22
2.8 Compartilhando Apps	23
2.9 Conclusão	23
2.10 Estilo e marcação	24
2.11 Diferentes tipos de inputs	24
2.12 Fazendo seu site ficar interativo	24
2.13 Colocando tudo junto	25
2.14 Outro exemplo	26
2.15 Compartilhando seu app	26
2.16 Fale sobre o servidor Shiny posteriormente	27
3 Leaflet	27
3.1 Iniciando	27
3.2 Adicionando marcadores	28
3.3 Desenhando	30
3.4 Conclusão	31
4 Gráficos interativos	32
4.1 googleVis	32
5 Pacotes	34
6 Objetos	34
7 Swirl	34

Listings

1	ui.R App 1	12
2	server.R App 1	13
3	Shiny codificado em HTML	13
4	App 2: ui.R App 2	14
5	server.R App 2	14
6	ui.R App 3	15
7	server.R App 3	15
8	Chamando Shiny Apps do Sean Kross localmente	15
9	ui.R App 4	16
10	server.R App 4	16
11	ui.R App 5	18
12	server.R App 5	18
13	ui.R App 6	19
14	server.R App 6	19
15	ui.R app 7	21
16	ui.R App 8	22
17	server.R App 8	22

Nota do tradutor

Esse e-book traduzido é oriundo de [Developing Data Products](#) disponível como bookdown escrito pelos autores Brian Caffo e Sean Kross em 29/03/2017.

Para que os scripts funcionem corretamente recomendo que você faça a integração do seu  Studio com o Overleaf observando [esse tutorial aqui](#).

**As traduções aqui são somente as transcrições. Não me preocupei em aperfeiçoá-las para a língua portuguesa com maior nível de clareza nos textos. As figuras e imagens não foram traduzidas.*

Prefácio

0.1 Sobre esse livro

Este livro foi escrito como um livro complementar para o curso Developing Data Products Coursera como parte da Especialização em Ciência de Dados. No entanto, se você não fizer a aula, o livro permanecerá por si próprio. Um componente útil do livro é uma série de vídeos do YouTube que compõem a classe Coursera.

O livro pretende ser uma introdução de baixo custo ao importante campo de produtos de dados. O público-alvo são estudantes alfabetizados numérica e computacionalmente, que gostariam de usar essas habilidades em Ciência de Dados. O livro é oferecido gratuitamente como uma série de documentos R Markdown no Github e em formas mais convenientes (epub, mobi) no LeanPub.

Este livro está licenciado sob uma Licença Internacional Creative Commons Atribuição-Uso Não-Comercial-Compartilhamento pela mesma licença 4.0, que requer atribuição de autor para trabalhos derivados, uso não comercial de trabalhos derivados e que as alterações são compartilhadas da mesma maneira que o trabalho original.

Introdução

0.2 O que é data product ?

Começaremos este livro definindo o tópico deste curso, de data products. Um produto de dados é a saída de produção de uma análise de dados. Por exemplo, uma análise de dados pode criar um algoritmo inteligente de aprendizado de máquina. Um produto de dados incorpora esse algoritmo em um site para que os usuários possam inserir valores e obter previsões. Sites de análise interativa, gráficos, aplicativos, pacotes  , apresentações e relatórios são todos produtos de dados. Neste livro, focamos apenas em alguns desses componentes. Principalmente por motivos de espaço, mas também porque nossa especialização no Coursera cobre outras (como redação de relatórios).

Antes de iniciar este livro, você poderá usar  . Essa linguagem servirá como ponto de partida para todos os nossos produtos de dados. Felizmente, se você não conhece  , Roger Peng tem uma ótima [aula de Coursera](#) e um [livro no LeanPub](#) sobre o assunto; pegue e leia primeiro. A aula acontece todos os meses e ambos podem ser obtidos gratuitamente.

Por que  ? Bem, para começar, é o que eu sei. Mas também é uma linguagem de análise de dados muito prevalente. Portanto, é conveniente criar o produto de dados na mesma linguagem em que a análise é feita. Além disso, a lista de ferramentas que você precisa aprender além do  para desenvolver produtos de dados é enorme e inclui: HTML, Javascript, D3, Python , Amazon Web Services e assim por diante. Em certo sentido, as ferramentas que apresentamos são melhor consideradas como ferramentas de prototipagem antes de criar um esforço maior de produção. No entanto, para muitos aplicativos, eles podem ficar sozinhos. O Shiny, em particular, está passando por rápida adoção, desenvolvimento e crescimento.

0.3 O objetivo deste livro

Este livro (e a classe correspondente) tem um objetivo simples: começar a criar produtos de dados, apresentando-lhe algumas ferramentas muito legais em  . Nós apenas arranhamos a superfície na maioria dessas plataformas fantásticas e, infelizmente, omitimos algumas importantes. É melhor seguir este livro com um projeto de dados simples em mente. Portanto, antes de começar, pense em um aplicativo Web orientado a dados que você deseja criar. Tente usar as ferramentas em andamento para criar versões simplificadas do seu aplicativo. Esperamos que, no final, você tenha um kit de ferramentas grande o suficiente para poder aprender o que precisa para criar seu aplicativo ou produto.

1 R Markdown

[Assista esse vídeo antes de começar](#)

1.1 Introdução ao R Markdown

O Markdown é uma linguagem de marcação leve, que enfatiza a facilidade de escrever e ler o conteúdo e pode ser facilmente convertida em HTML. Markdown foi inventado por [John Gruber](#) com contribuições significativas de [Aaron Swartz](#). O R Markdown estende o markdown para incluir os resultados do código , incluindo tabelas e visualizações. O pacote rmarkdown pode criar páginas da web, PDFs e apresentações de slides a partir de documentos R Markdown. O verdadeiro poder do R Markdown é a capacidade de interligar código e linguagem natural. Essa abordagem é conhecida como [programação alfabetizada](#).

1.2 Markdown: Sintaxe básica

Abaixo estão alguns exemplos da sintaxe do Markdown. Você pode encontrar um guia de sintaxe abrangente no [site do RStudio](#).

1.2.1 Cabeçalhos

```
1 # Eu sou enorme!
2 ## Ainda sou grande
3 ### Tenho um tamanho razoável
```

1.2.2 Texto

```
1 It was the best of times, it was the worst of times, it was the age of wisdom,
2 it was the age of foolishness, it was the epoch of belief, it was the epoch of
3 incredulity, it was the season of Light, it was the season of Darkness, it was
4 the spring of hope, it was the winter of despair, we had everything before us,
5 we had nothing before us, we were all going direct to Heaven, we were all going
6 direct the other way      in short, the period was so far like the present period,
7 that some of its noisiest authorities insisted on its being received, for good
8 or for evil, in the superlative degree of comparison only.
```

1.2.3 Negrito e itálico

```
1 *italico*
2 **negrito**
3 ***negrito e italicico***
```

1.2.4 Listas

Desordenada:

```
1 - um item
2 - outro item
3 - terceiro item
```

Ordenada:

```
1 1. um item
2 2. outro item
3 3. terceiro item
```

1.2.5 Links

```
1 [JHU's Homepage](http://www.jhu.edu)
```

1.2.6 Imagens

```
1 ! [A seagull](https://farm9.staticflickr.com/8221/8259009216_4b4e6f994c_m.jpg)
```

1.2.7 Citações

```
1 > Good night, Mrs. Calabash, wherever you are.
```

1.2.8 Códigos

```
1 # Code from Hadley Wickham's purr package
2
3 reduce <- function(.x, .f, ..., .init) {
4   .f <- as_function(.f, ...)
5
6   f <- function(x, y) {
7     .f(x, y, ...)
8   }
9
10 Reduce(f, .x, init = .init)
11 }
```

1.2.9 Código no corpo do texto

```
1 Você pode usar a função `c()` para criar um vetor.
```

1.2.10 Códigos do R no Markdown

Se caso você precise utilizar a linguagem do R, utilize a estrutura do chunk:

```
1 """
2 # Write some R code here!
3 x <- rnorm(10)
4 x^2
5 """
6
7 """
8 ## [1] 0.584149265 0.495505654 0.418426473 0.289899226 1.372044525 0.392424083
9 ## [7] 0.002753627 0.033064010 0.796271729 2.001400414
10 """
11
12 """
13 plot(x)
14 """
15
16 ! [plot of chunk unnamed-chunk-1](figure/unnamed-chunk-1-1.png)
```

Os resultados da computação x^2 e o gráfico produzido pelo gráfico(x) serão exibidos nesse chunk de código R assim que você renderizar esse arquivo R Markdown em uma página da web, PDF ou apresentação. Existem várias opções úteis para personalizar esses blocos de código R que discutiremos em uma seção posterior.

1.3 Escrevendo e renderizando documentos

Abra um novo arquivo em um editor de texto sem formatação e tenha seu console R pronto. Eu recomendo usar o RStudio, pois combina bem um editor de texto sem formatação, um console R e um

navegador de arquivos. Escreva alguma remarcação no seu editor de texto sem formatação ou copie e cole a remarcação  abaixo:

```

1 ---
2 title: "Simulacao no R"
3 author: Brian Caffo
4 date: July 28, 2016
5 ---
6
7 ## Simulacao simples no R
8
9 Bem-vindo ao meu tutorial sobre como fazer simulacoes muito simples em R. Na primeira
10 simulacao que vamos tentar eh lancar uma moeda, o que pode resultar no
11 o lance da moeda sobe ou desce. Podemos simular lancando uma moeda
12 executando a funcao 'lance_uma_moeda()' definida abaixo:
13
14 ````r
15 lance_uma_moeda <- function(){
16   sample(c("H", "T"), 1)
17 }
18 ````

19
20 ## Simulacao 1
21
22 Vamos testar o resultado simples da execucao de um lancamento de moeda:
23
24
25 ````r
26 lance_uma_moeda()
27 ````

28
29 ````

30 ## [1] "H"
31 ````

32
33 ## Lance muitas moedas
34
35 Em vez de jogar uma moeda de cada vez, podemos definir uma funcao que sera lancar
36 um numero especifico de moedas:
37
38
39 ````r
40 lanca_moedas <- function(n){
41   sample(c("H", "T"), n, replace = TRUE)
42 }
43 ````

44
45 ## Resultados de preenchimento de moedas
46
47 Vamos dar uma olhada nos resultados do lancamento de 100 moedas com um grafico de barras:
48
49
50 ````r
51 barplot(table(lanca_moedas(100)))
52 ````

53
54 ![[plot of chunk unnamed-chunk-5]](figure/unnamed-chunk-5-1.png)

```

Um recurso do R Markdown que é diferente do Markdown comum é a presença do yaml no início do

arquivo. O Yaml é um meio simples para fornecer metadados sobre o documento que você está escrevendo. A front matter do yaml é escrita entre um par de três hífens (—) no início do documento. No arquivo Rmd acima, especifiquei um título, autor e data para este documento, que será colocado no início do documento.

Salve seu documento R Markdown em seu diretório de trabalho atual como fboxsimple_sim.Rmd. Para transformar este documento em uma página da Web, instale primeiro o pacote rmarkdown, se você ainda não possui o `install.packages("rmarkdown")`. Após a instalação do pacote, você pode carregar o pacote rmarkdown com `library(rmarkdown)`. Você pode renderizar seu R Markdown em uma página da Web digitando `render("simple_sim.Rmd")` no console do R. Após a produção do documento HTML, é possível visualizar seu novo documento digitando `browseURL("simple_sim.html")` no console do R. Parabéns por criar seu primeiro produto de dados!

Se você deseja criar um PDF em vez de um documento HTML, precisará fornecer argumentos diferentes para a função `render()`. Digite `render ("simple_sim.Rmd", output_format = pdf_document())` no console do R para criar um PDF.

Existem várias vantagens e desvantagens entre a distribuição de documentos HTML ou PDF. O HTML permite que seu produto de dados seja distribuído como um site que permite incorporar visualizações interativas, algumas das quais falaremos sobre a criação em um capítulo posterior. Os documentos PDF são mais independentes em comparação com o HTML, embora sejam documentos estáticos - eles não respondem como uma página da Web. Em geral, se você quiser que outros desenvolvam seu produto de dados

1.4 Criando apresentações no R Markdown

Além de criar páginas da web e PDFs a partir do R Markdown, você também pode criar apresentações de slides. Use o documento R Markdown da seção anterior e insira

`render("simple_sim.Rmd", output_format = ioslides_presentation())` no console do R. Agora abra o deck de slides com `browseURL("simple_sim.html")`. Como você pode ver, ainda é uma página da web, exceto que o documento foi renderizado como um slide slide!

Cada slide é demarcado no seu arquivo de remarcão R com dois sinais de sustenido , seguidos pelo título do slide. Os campos especificados na matéria frontal do yaml criam o primeiro slide. Se você preferir produzir slides em PDF, insira `render ("simple_sim.Rmd", output_format = beamer_presentation())` no console do  para criar slides em PDF com a estrutura do *TEX* Beamer.

1.5 Compartilhando documentos em R Markdown

Recomendamos os seguintes sites para compartilhar os arquivos HTML que você produz a partir de documentos R Markdown. Se você usou o The Data Scientist's Toolbox, deve conhecer as páginas do GitHub. Se você conhece o Git e não o usou antes da documentação deles para criar um site, é muito simples. Se você não estiver familiarizado com o Git e estiver usando o RStudio, aproveite os RPubs, que talvez sejam a maneira mais fácil de compartilhar um documento HTML do RStudio. Você pode encontrar instruções simples para usar RPubs aqui. Se você estiver procurando um site que hospede qualquer arquivo HTML gratuitamente, incluindo arquivos HTML gerados pelo R Markdown, consulte o NeoCities.

1.6 Considerações finais

Com o R Markdown, você pode entrelaçar código de várias linguagens de programação, incluindo R, com texto, tabelas e gráficos. A partir da origem de um arquivo Rmd, você pode produzir vários tipos diferentes de produtos de dados, incluindo sites, PDFs e até e-books. Para mais informações sobre o R Markdown, recomendamos que você visite <http://rmarkdown.rstudio.com/>.

2 Shiny

[Assista a este vídeo antes de começar](#)

Shiny é um milagre da programação moderna e ajuda a tornar o R uma das linguagens mais competitivas para a comunicação de informações a partir de dados. O Shiny é desenvolvido pelo RStudio e é descrito como ”Uma estrutura de aplicativos da web para R”. Diferentemente das estruturas de aplicativos da Web em muitos outros idiomas, o tempo necessário para escrever um aplicativo Shiny bonito, funcional e pronto para dispositivos móveis é realmente mínimo, o que o torna ideal para prototipagem rápida e implantação fácil. O pessoal do RStudio tem o seguinte argumento: *“Transforme suas análises em aplicativos interativos da Web. Não é necessário conhecimento de HTML, CSS ou JavaScript”*. Essa afirmação é principalmente verdadeira, embora um pouco de conhecimento em HTML seja útil para entender alguns dos conceitos. Se você estiver interessado em aprender mais sobre HTML, CSS e Javascript, recomendamos qualquer um dos seguintes recursos:

- [Mozilla Developer Network Tutorials](#)
- [HTML CSS do Khan Academy](#)
- [Tutoriais do Free Code Camp](#)

Continuaremos como se o seu conhecimento em HTML fosse muito básico e não fosse mais avançado do que entender níveis ou fontes de cabeçalho.

2.1 Instalando o Shiny

Primeiro, verifique se você tem a versão mais recente do R instalada. Se estiver no Windows, verifique se você tem o Rtools instalado. Em seguida, você pode instalar o Shiny com

```
1 install.packages("shiny")
2 library(shiny)
```

2.2 O seu primeiro ShinyApp

Vamos criar nosso primeiro aplicativo. Como muitos projetos deste livro, começaremos com um aplicativo de exemplo simples. Primeiro, crie um novo diretório onde você pode armazenar todos os arquivos associados ao seu aplicativo. Gosto de organizar meu aplicativo Shiny em dois arquivos separados: ui.R, que contém todos os elementos da interface do usuário, e server.R, que contém a lógica do aplicativo, incluindo o código para carregar e manipular dados. Você pode encontrar o código para cada um desses arquivos abaixo:

```
1 library(shiny)
2
3 fluidPage(
4   titlePanel("Data science FTW!"),
5   sidebarLayout(
6     sidebarPanel(
7       h3("Texto da barra lateral")
8     ),
9     mainPanel(
10       h3("Texto do meio do painel")
11     )
12   )
13 )
```

Listing 1: ui.R App 1

```

1 library(shiny)
2
3 function(input, output){
4
5 }
```

Listing 2: server.R App 1

Este aplicativo é extremamente mínimo, apenas para lhe familiarizar com a sintaxe de aplicativos Shiny, incluindo a natureza aninhada dos elementos da interface do usuário. Crie esses dois arquivos dentro do seu novo diretório e altere seu diretório de trabalho atual para essa pasta usando `setwd()`. Após carregar shiny com o `library(shiny)`, insira `runApp()` no console do R para iniciar o aplicativo. Como alternativa, você pode fornecer o caminho para o diretório que contém os arquivos ui.R e server.R como um argumento para `runApp()`. Seu aplicativo deve ser algo como isto:

Seu primeiro Shiny app

Vamos compreender o código em ui.R.

A função `fluidPage()` especifica um tipo de interface do usuário para o Shiny exibir. As `fluidPages` tentam se reorganizar de maneira inteligente, dependendo do tamanho da tela que está exibindo o aplicativo, o que torna esse layout melhor em dispositivos móveis. Em vez de um `fluidPage()`, você pode usar um `fixedPage()` que não redimensionará seu aplicativo. A função `titlePanel()` define um título na parte superior da página e a função `sidebarLayout()` define o layout de tudo abaixo do painel de título. O `sidebarLayout()` divide a página em uma barra lateral e em uma parte principal da página, que são especificadas com a função `sidebarPanel()` e a função `mainPanel()`, respectivamente. Dentro do `sidebarPanel()` e do `mainPanel()`, coloquei um cabeçalho `h3()`, que apenas exibe algum texto e é o mesmo que especificar uma tag `<h3>` em HTML.

```

1 <div class="container-fluid">
2   <h2>Data science FTW!</h2>
3   <div class="row">
4     <div class="col-sm-4">
5       <form class="well">
6         <h3>Sidebar Text</h3>
7       </form>
8     </div>
9     <div class="col-sm-8">
10       <h3>Main Panel Text</h3>
11     </div>
12   </div>
13 </div>
```

Listing 3: Shiny codificado em HTML

2.3 Mais elementos UI

Vamos dar uma olhada em outro aplicativo simples com uma interface um pouco mais complexa:

```

1 library(shiny)
2
3 fluidPage(
4   titlePanel("Tags HTML"),
5   sidebarLayout(
6     sidebarPanel(
7       h1("H1 Texto"),
8       h3("H3 Texto"),
9       em("Texto enfatizado")
10    ),
11    mainPanel(
12      h3("Texto do painel central"),
13      code("Código!")
14    )
15  )
16)
```

Listing 4: App 2: ui.R App 2

```

1 library(shiny)
2
3 function(input, output){
4
5 }
```

Listing 5: server.R App 2

Este código produz o seguinte app:

<http://127.0.0.1:4690> | Open in Browser |

Tags HTML

H1 Texto

H3 Texto

Texto enfatizado

Texto do painel central

Código!

Segundo Shiny App

Neste exemplo, adicionamos alguns elementos da interface do usuário e você pode ver como eles são renderizados no aplicativo. As funções `h1()` e `h3()` especificam cabeçalhos, a função `em()` coloca o texto em itálico e a função `code()` renderiza o texto com uma fonte monoespaçada para parecer com código de computador. Você pode encontrar uma lista de todas as diferentes marcas de texto da interface do usuário disponíveis, digitando `?Builder` no console do R.

2.4 Inputs e Outputs

Agora que você conhece o básico da interface do usuário do Shiny, pode começar a aprender como fazer com que seus aplicativos Shiny façam algo útil! O recurso mais importante de um aplicativo Web é que um usuário pode fornecer entradas interagindo com seu aplicativo e, em seguida, você pode mostrar as saídas do usuário que respondem a essas entradas. Vamos começar com o seguinte aplicativo simples e Shiny:

```

1 library(shiny)
2
3 fluidPage(
4   titlePanel("Slider App"),
5   sidebarLayout(
6     sidebarPanel(
7       h1("Mova o Slider!"),
8       sliderInput("slider1", "Slide Me!", 0, 100, 0)
9     ),
10    mainPanel(
11      h3("Valor do slider:"), 
12      textOutput("texto")
13    )
14  )
15 )

```

Listing 6: ui.R App 3

```

1 library(shiny)
2
3 function(input, output){
4   output$text <- renderText(input$slider1)
5 }

```

Listing 7: server.R App 3

O código acima produzirá um aplicativo parecido com este:

<http://127.0.0.1:4690> | Open in Browser |



Terceiro Shiny App

Uma captura de tela deste aplicativo não faz justiça. Certifique-se de executá-lo no seu próprio computador! Você pode fazer isso facilmente inserindo o seguinte no console do R:

```
1 shiny::runGitHub("seankross/developing-data-products", subdir = "assets/shinyapps/app3/")
```

Listing 8: Chamando Shiny Apps do Sean Kross localmente

Tente deslizar o controle deslizante (slider) para frente e para trás. O número exibido na página deve mudar para refletir o valor do controle deslizante. No ui.R, a única nova função é `sliderInput()`, que define o elemento da interface do usuário do slider na página. Essa função utiliza vários argumentos, incluindo um identificador exclusivo para esse controle deslizante ("slider1" neste caso), o texto a ser exibido acima do controle deslizante e, finalmente, os valores mínimos, máximos e iniciais para o controle deslizante.

A segunda nova função na interface do usuário é `textOutput()`. Esta função renderiza o texto que é calculado em server.R. O único argumento passado para essa função é uma sequência com o ID do texto especificado em server.R. Designaremos um ID simples chamado "texto" que usaremos posteriormente no server.R.

Este aplicativo é o primeiro que mostrei que tem algo notável acontecendo no server.R. O arquivo server.R deve retornar uma função; portanto, para aplicativos simples, é apropriado definir uma função

anônima, como fizemos neste exemplo. Esta função deve ter argumentos denominados `input` e `output`. Você deve pensar nos valores desses argumentos dentro do servidor como uma `list()` que cada um nomeou elementos que você pode consultar (no caso de `input`) e atribuir valores (no caso de `output`). Neste exemplo, estamos recuperando o valor do controle deslizante da entrada de acordo com o identificador que atribuímos a ele em ui.R ("slider1"). A expressão `input$slider1` é avaliada como o valor atual do slider. Em seguida, agrupamos essa expressão com a função `renderText()` que renderiza um valor para que possa ser impressa como texto em um aplicativo Shiny. O valor renderizado é então armazenado na lista de `output` no elemento de `text`. A interface do usuário do aplicativo pode recuperar esse valor, uma vez que especificamos o ID do texto em ui.R, e o valor do controle deslizante é exibido na tela.

Se você entende como o aplicativo descrito acima está funcionando, entende as entradas e saídas básicas no Shiny! No lado do servidor de um aplicativo Shiny, você costuma escrever código que recebe entradas da interface do usuário, faz algumas computações e deve renderizar o resultado dessa computação para mostrá-lo ao usuário. Geralmente você pode fazer isso com a família de funções renderizadas incluída no Shiny. No próximo aplicativo, mostraremos uma plotagem para o usuário com base em algumas entradas, eventualmente usando a função `renderPlot()`.

```

1 library(shiny)
2
3 fluidPage(
4   titlePanel("Plota Numeros Aleatorios"),
5
6   sidebarLayout(
7     sidebarPanel(
8       numericInput("numPontos", "Quantos numeros aleatorios deseja plotar ?",
9                   value = 1000, min = 1, max = 1000, step = 1),
10      sliderInput("sliderX", "Seleciona os valores minimos e maximos para X",
11                  -100, 100, value = c(-50, 50)),
12      sliderInput("sliderY", "Seleciona o valor minimo e maximo para Y",
13                  -100, 100, value = c(-50, 50)),
14      checkboxInput("mostraEixox", "Mostra/Esconde X Nome do eixo", value = TRUE),
15      checkboxInput("mostraEixoy", "Mostra/Esconde Y Nome do eixo", value = TRUE),
16      checkboxInput("mostraTitulo", "Mostra/Esconde Titulo")
17    ),
18
19    mainPanel(
20      h3("Grafico de numeros aleatorios"),
21      plotOutput("plot1")
22    )
23  )
24)
```

Listing 9: ui.R App 4

```

1 library(shiny)
2
3 function(input, output) {
4   output$plot1 <- renderPlot({
5     set.seed(2016-12-13)
6
7     dataX <- runif(input$numPontos, input$sliderX[1], input$sliderX[2])
8     dataY <- runif(input$numPontos, input$sliderY[1], input$sliderY[2])
9
10    xlab <- ifelse(input$mostraEixox, "Eixo X", "")
11    ylab <- ifelse(input$mostraEixoy, "Eixo Y", "")
12    main <- ifelse(input$mostraTitulo, "Titulo", "")
13
14    plot(dataX, dataY, xlab = xlab, ylab = ylab, main = main,
15         axes = FALSE)
```

```

15     xlim = c(-100, 100), ylim = c(-100, 100))
16   })
17 }
```

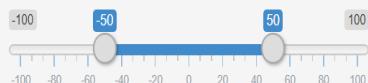
Listing 10: server.R App 4

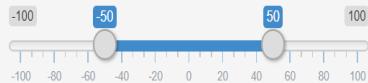
Este código gera um resultado como este:

<http://127.0.0.1:4690> | Open in Browser | Publish

Plota Numeros Aleatorios

Quantos numeros aleatorios deseja plotar ?
1000

Seleciona os valores minimos e maximos para X


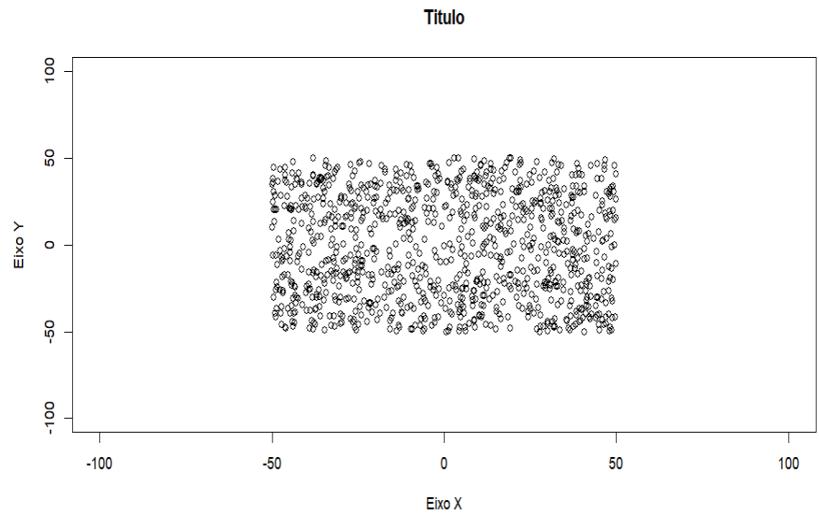
Seleciona o valor minimo e maximo para Y


Mostra/Esconde X Nome do eixo

Mostra/Esconde Y Nome do eixo

Mostra/Esconde Titulo

Grafico de numeros aleatorios



Quarto Shiny App

Você pode rodar esse app localmente compilando a seguinte linha:

```
1 shiny::runGitHub("seankross/developing-data-products", subdir = "assets/shinyapps/app4/")
```

Este aplicativo tem mais código do que qualquer um dos aplicativos que criamos antes, mas é apenas uma extensão das mesmas idéias de que falamos. No ui.R existem algumas novas funções. A função `numericInput()` permite que os usuários digitem um número em uma caixa de texto. Você já viu a função `sliderInput()`, mas neste caso existem dois valores iniciais, o que permite ao usuário selecionar um intervalo de valores em um controle deslizante entre um valor mínimo e máximo. Em seguida, há a função `checkboxInput()` que cria uma caixa de seleção na página que o usuário pode alternar. Finalmente, a função `plotOutput()` exibe um gráfico que é renderizado em server.R.

Toda a ação neste arquivo server.R ocorre em uma expressão passada para `renderPlot()`, pois uma plotagem é a única saída desse aplicativo. O número de números aleatórios a serem gerados é recuperado da entrada `$num_pontos` e cada controle deslizante retorna um vetor numérico de comprimento dois, com o primeiro elemento como o valor do controle deslizante mais à esquerda e o segundo elemento como o valor do controle deslizante mais à direita. O título do eixo é ativado e desativado pela entrada `$mostra_titulo`, que é TRUE se a caixa de seleção estiver marcada e FALSE caso contrário. O mesmo princípio se aplica aos rótulos dos eixos. Finalmente, o gráfico é construído com uma função regular `plot()` usando entradas da interface do usuário como parâmetros.

2.5 UI dinâmica

Convém alterar como a interface do usuário do seu aplicativo Shiny é exibida, dependendo de outras entradas da interface do usuário no seu aplicativo. Esse é um padrão comum visto em formulários on-line,

onde sua escolha na parte inicial de um formulário restringe as opções em potencial nas partes posteriores do formulário. Aqui está um pequeno aplicativo que ilustra como alterar dinamicamente a interface do usuário no Shiny:

```

1 library(shiny)
2
3 fluidPage(
4   titlePanel("UI Dinamica"),
5
6   sidebarLayout(
7     sidebarPanel(
8       selectInput("pais", "Em que pais voce vive?", 
9                  choices = c("EUA", "Canada")),
10      uiOutput("regiao")
11    ),
12
13    mainPanel(
14      textOutput("mensagem")
15    )
16  )
17 )
```

Listing 11: ui.R App 5

```

1 library(shiny)
2 library(minimap)
3
4 function(input, output) {
5   output$region <- renderUI({
6     if(input$pais == "EUA"){
7       selectInput("estado", "Em que estado voce vive?", choices = usa_abb)
8     } else {
9       selectInput("pt", "Em qual cidade ou territorio voce vive?", choices = canada_abb)
10    }
11  })
12
13  output$mensagem <- renderText({
14    if(input$country == "EUA"){
15      paste0("Voce mora nos EUA no estado de ", input$estado, ".")
16    } else {
17      paste0("Voce mora no Canada na provincia/estado de ", input$pt, ".")
18    }
19  })
20 }
21 }
```

Listing 12: server.R App 5

Esse código irá produzir o seguinte app:

The screenshot shows a Shiny application interface. At the top, there's a header with the URL 'http://127.0.0.1:5025' and an 'Open in Browser' button. The main content area has a title 'UI Dinamica'. Below it, there are two dropdown menus. The first menu is labeled 'Which Country do you live in?' and has 'USA' selected. The second menu is labeled 'Em que estado voce vive?' and has 'AK' selected. To the right of these menus, a message box displays the text 'Voce mora no estado de AK.'

Escolha a região

Lembrando que você pode rodar ele localmente chamando:

```
1 shiny::runGitHub("seankross/developing-data-products", subdir = "assets/shinyapps/app5/")
```

O ui.R neste aplicativo contém elementos que já vimos antes, exceto a função `uiOutput()`. Esta função exibe a interface do usuário que é calculada dentro do server.R! Dentro do server.R, dê uma olhada na função `renderUI()`, que contém o código que você normalmente veria dentro do ui.R. A instrução `if` determina qual `selectInput()` é exibido, nos estados dos EUA ou nas províncias e territórios do Canadá. A manipulação dinâmica da interface do usuário pode ser uma ferramenta poderosa para controlar o uso e o fluxo do seu aplicativo.

2.6 Reatividade

```
1 library(shiny)
2
3 fluidPage(
4   titlePanel("Prever os cavalos de potencia"),
5
6   sidebarLayout(
7     sidebarPanel(
8       sliderInput("sliderMPG", "Quantas km/litro o carro faz?", 10, 35, value = 20),
9       checkboxInput("showModel1", "Show/Hide Model 1", value = TRUE),
10      checkboxInput("showModel2", "Show/Hide Model 2", value = TRUE)
11    ),
12
13    mainPanel(
14      plotOutput("plot1"),
15      h3("Cavalos de potencia previstos do Modelo 1:"), 
16      textOutput("pred1"),
17      h3("Cavalos de potencias previstos do Model 2:"), 
18      textOutput("pred2")
19    )
20  )
21 )
```

Listing 13: ui.R App 6

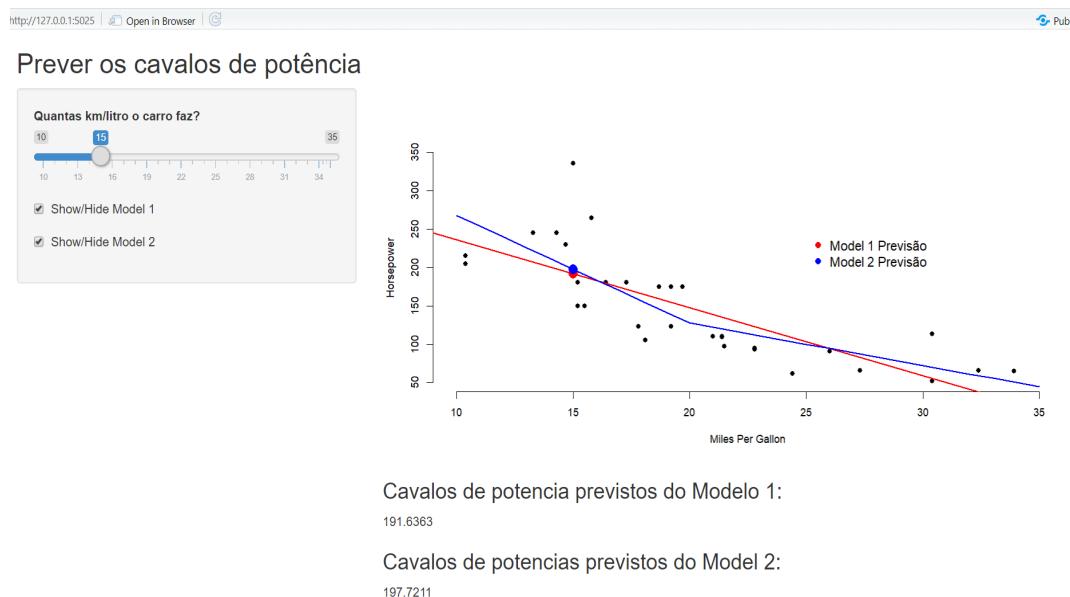
```
1 library(shiny)
```

```

2
3 function(input, output) {
4   # Create Spline
5   mtcars$mpgsp <- ifelse(mtcars$mpg - 20 > 0, mtcars$mpg - 20, 0)
6
7   # Ajuste de modelos
8   model1 <- lm(hp ~ mpg, data = mtcars)
9   model2 <- lm(hp ~ mpgsp + mpg, data = mtcars)
10
11  model1pred <- reactive{
12    mpgInput <- input$sliderMPG
13    predict(model1, newdata = data.frame(mpg = mpgInput))
14  }
15
16  model2pred <- reactive{
17    mpgInput <- input$sliderMPG
18    predict(model2, newdata =
19      data.frame(mpg = mpgInput,
20                  mpgsp = ifelse(mpgInput - 20 > 0,
21                                         mpgInput - 20, 0)))
22  }
23
24  output$plot1 <- renderPlot{
25    mpgInput <- input$sliderMPG
26
27    plot(mtcars$mpg, mtcars$hp, xlab = "Miles Per Gallon",
28          ylab = "Horsepower", bty = "n", pch = 16,
29          xlim = c(10, 35), ylim = c(50, 350))
30    if(input$showModel1){
31      abline(model1, col = "red", lwd = 2)
32    }
33    if(input$showModel2){
34      model2lines <- predict(model2, newdata = data.frame(
35        mpg = 10:35, mpgsp = ifelse(10:35 - 20 > 0, 10:35 - 20, 0)
36      ))
37      lines(10:35, model2lines, col = "blue", lwd = 2)
38    }
39    legend(25, 250, c("Model 1 Previs o", "Model 2 Previs o"), pch = 16,
40           col = c("red", "blue"), bty = "n", cex = 1.2)
41    points(mpgInput, model1pred(), col = "red", pch = 16, cex = 2)
42    points(mpgInput, model2pred(), col = "blue", pch = 16, cex = 2)
43  }
44
45  output$pred1 <- renderText{
46    model1pred()
47  }
48
49  output$pred2 <- renderText{
50    model2pred()
51  }
52 }
```

Listing 14: server.R App 6

O código gera o seguinte resultado:



Modelando a base de dados cars

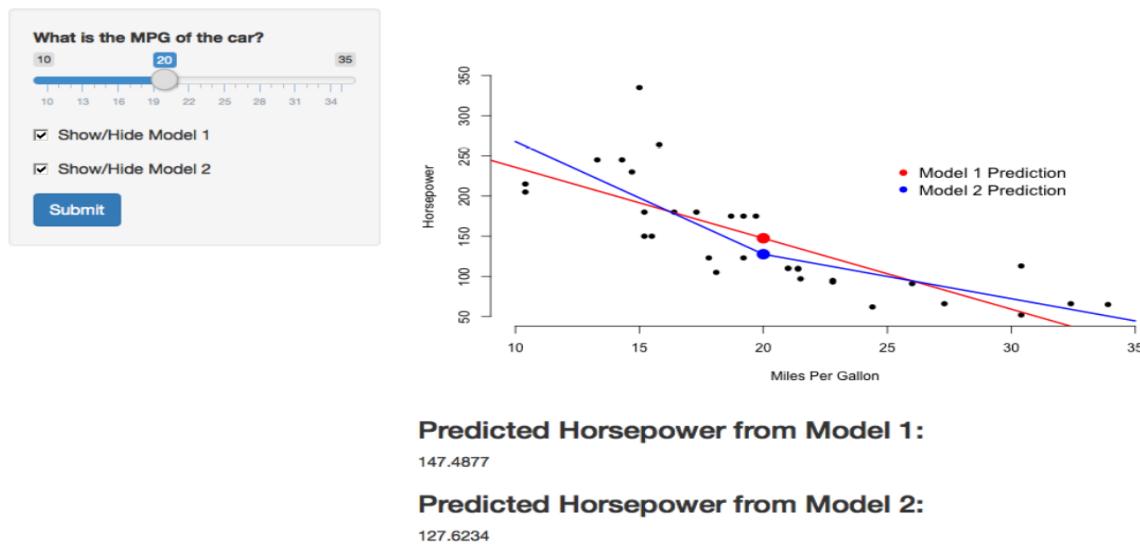
Rode localmente com os comandos:

```
1 shiny::runGitHub("seankross/developing-data-products", subdir = "assets/shinyapps/app6/")
```

```
1 library(shiny)
2
3 fluidPage(
4   titlePanel("Previsão dos cavalos de potencia"),
5
6   sidebarLayout(
7     sidebarPanel(
8       sliderInput("sliderMPG", "Quantos km/litro o carro faz?", 10, 35, value = 20),
9       checkboxInput("showModel1", "Show/Hide Model 1", value = TRUE),
10      checkboxInput("showModel2", "Show/Hide Model 2", value = TRUE),
11      submitButton("Submit")
12    ),
13
14   mainPanel(
15     plotOutput("plot1"),
16     h3("Cavalos de potencia previstos Modelo 1:"),  
textOutput("pred1"),
17     h3("Cavalos de potencia previstos Modelo 2:"),  
textOutput("pred2")
18   )
19 )
20 )
21 )
22 )
```

Listing 15: ui.R app 7

Predict Horsepower from MPG



Você pode rodar localmente:

```
1 shiny::runGitHub("seankross/developing-data-products", subdir = "assets/shinyapps/app7/")
```

2.7 Interatividade

```
1 library(shiny)
2
3 fluidPage(
4   titlePanel("Modelo de visualiza o nuvem de pontos"),
5   sidebarLayout(
6     sidebarPanel(
7       h3("Slope"),
8       textOutput("slopeOut"),
9       h3("Intercept"),
10      textOutput("intOut")
11    ),
12    mainPanel(
13      plotOutput("plot1", brush = brushOpts(
14        id = "brush1"
15      ))
16    )
17  )
18 )
```

Listing 16: ui.R App 8

```
1 library(shiny)
2
3 function(input, output) {
4   model <- reactive({
5     brushed_data <- brushedPoints(trees, input$brush1,
6                                    xvar = "Girth", yvar = "Volume")
7     if(nrow(brushed_data) < 2){
8       return(NULL)
9     }
10    lm(Volume ~ Girth, data = brushed_data)
11  })
12
13  output$slopeOut <- renderText({
14    if(is.null(model())){
```

```

15     "No Model Found"
16 } else {
17     model() [[1]][2]
18 }
19 })
20
21 output$intOut <- renderText({
22   if(is.null(model())){
23     "No Model Found"
24   } else {
25     model() [[1]][1]
26   }
27 })
28
29 output$plot1 <- renderPlot({
30   plot(trees$Girth, trees$Volume, xlab = "Girth",
31         ylab = "Volume", main = "Tree Measurements",
32         cex = 1.5, pch = 16, bty = "n")
33   if(!is.null(model())){
34     abline(model(), col = "blue", lwd = 2)
35   }
36 })
37 }

```

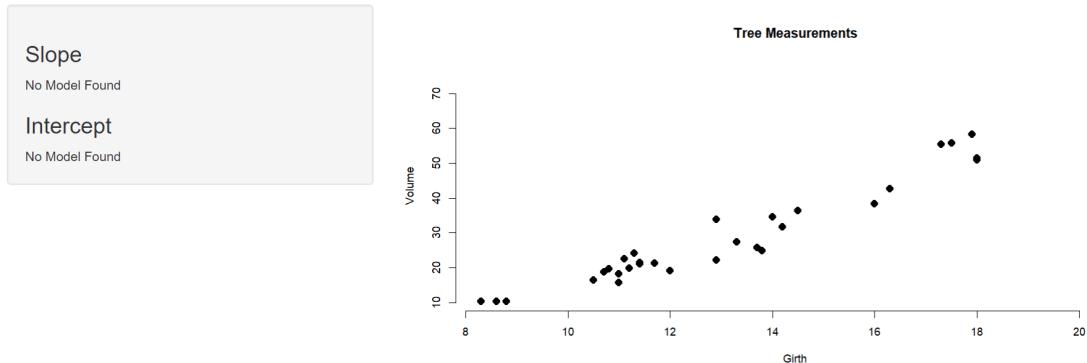
Listing 17: server.R App 8

Esse código produz um app como esse:

<http://127.0.0.1:5025> | |

Pul

Modelo de visualização nuvem de pontos



Nuvem de pontos

Você pode rodar isso localmente com:

```
1 shiny::runGitHub("seankross/developing-data-products", subdir = "assets/shinyapps/app8/")
```

2.8 Compartilhando Apps

2.9 Conclusão

Se é como a maioria dos usuários quando encontra o Shiny pela primeira vez, provavelmente está se perguntando "O que está acontecendo? Por que o sintaxe é tão estranha?" Para se acostumar com a programação de aplicativos Shiny, você precisa jogar um pouco do seu pensamento sobre o ; é um estilo diferente de programação.

Vamos analisar o que está acontecendo com nossos dois arquivos para tornar as coisas mais claras. A função ui.R está controlando a interface. A função shinyUI está alertando o para isso. A função interna, fluidPage, diz ao shinyUI que tipo de página criar. Nesse caso, é uma página que pode se reorganizar com base no que você inclui na interface do usuário e se reorganizar para que o aplicativo seja fácil de usar

nas telas de dispositivos móveis. Você especifica todos os elementos da página usando funções. Nesse caso, queremos um título `titlePanel("Data science FTW!")`. Então queremos que o `sidebarPanel` contenha certos elementos. Portanto, a função `sidebarPanel` recebe argumentos (novamente funções) de seus elementos. A instrução `h3 ('Sidebar text')` está dizendo que queremos o texto Sidebar na barra lateral (já que essa função ocorre dentro da função `sidebarPanel`) e queremos que ela esteja no tamanho da fonte h3. Se você souber um pouco de html, reconhecerá h3 como o tamanho da fonte do terceiro nível de cabeçalho. Semelhante à função `sidebarPanel`, a função `mainPanel` utiliza argumentos funcionais para o painel principal.

Provavelmente, o erro de sintaxe mais frequente para o Shiny não é colocar vírgulas nos lugares certos da ui.R. Lembre-se de que os elementos da página são inseridos como argumentos, portanto, eles precisam de vírgulas, como todos os argumentos para as funções .

O arquivo server.R é um pouco mais fácil. A função `shinyServer` diz ao  que está lidando com um servidor Shiny. A função do servidor sempre aceita um argumento de uma função anônima com entradas e saídas de argumentos. Nesse caso, nossa função não faz nada.

2.10 Estilo e marcação

Talvez a maneira mais fácil de ilustrar a marcação seja através de um exemplo. Enquanto mantém o arquivo server.R o mesmo, altere o arquivo ui.R para o seguinte:

```
1 {lang=r, line-numbers=off} ~ shinyUI(pageWithSidebar( headerPanel(  Illustrating      markup )
  , sidebarPanel( h1( Sidebar      panel ), h1( H1      text ), h2( H2      Text ), h3( H3
    Text ), h4( H4      Text )
2
3 ), mainPanel( h3( Main      Panel      text ), code( some      code ), p( some      ordinary      text )
  ) ) ~
```

2.11 Diferentes tipos de inputs

O Shiny permite muitos tipos diferentes de inputs. No exemplo a seguir, alteramos o ui.R para permitir alguns tipos de entrada diferentes. Depois de pegar o jeito, qualquer novidade será fácil. Deixamos server.R como está, para que nossas entradas não façam nada. Veja abaixo a entrada numérica, caixa de seleção e entrada de data.

```
1 {lang=r, line-numbers=off} ~ shinyUI(pageWithSidebar( headerPanel(  Illustrating      inputs )
  , sidebarPanel( numericInput( id1      , Numeric      input, labeled id1 , 0, min = 0,
    max = 10, step = 1), checkboxGroupInput( id2      , Checkbox      , c( Value 1 =
      1 , Value 2 = 2 , Value 3 = 3 )), dateInput( date      ,
    Date : )
2 ), mainPanel(
3
4 ) ) ~
```

Execute isso para ver os inputs. Eles também são mostrados na próxima imagem alguns parágrafos abaixo.

Consulte esta lição no tutorial do site do Shiny para obter uma lista completa de inputs disponíveis. Depois de pegar o jeito de um ou dois deles, o restante será fácil. Portanto, faça o exemplo acima primeiro e depois tente alguns dos outros.

No momento, nada é feito com nossas entradas. Vamos ver se conseguimos descobrir como, pelo menos, colocá-los no server.R.

2.12 Fazendo seu site ficar interativo

Agora que já sabemos um pouco sobre como criar uma interface de usuário Shiny, vamos tornar o server.R reativo às entradas da ui.R. Primeiro, vamos adaptar nosso arquivo ui.R simples anterior para ilustrar. Pegue

o último exemplo, onde coletamos uma entrada numérica, caixa de seleção e data e substituímos a função mainPanel por abaixo.

```
1 {lang=r, line-numbers=off} mainPanel( h3( Illustrating outputs ), h4( You entered ),
  verbatimTextOutput( oid1 ), h4( You entered ), verbatimTextOutput( oid2 ),
  h4( You entered ), verbatimTextOutput( odate ) )
```

Nosso objetivo é fazer com que nosso sidebarPanel colete as entradas e nosso painel principal as exiba. As variáveis oid1, oid2 e odate são todas variáveis de resultado que definimos em nossa função server.R. Aqui está:

```
1 {lang=r, line-numbers=off} ~ shinyServer( function(input, output) { output oid1 =
  renderPrint({input id1}) output oid2 = renderPrint({input id2}) output odate =
  renderPrint({input date}) } ) ~
```

Portanto, nossa função recebe o input\$id1 e imprime oid1. Observe que id1 foi o nome dado à nossa entrada numérica em nossa função ui.R. Isso está incorreto na input\$id1. O rótulo id2 foi fornecido aos dados da caixa de seleção em ui.R e seu valor está incorreto na input\$id2. Da mesma forma, date foi o rótulo fornecido para a data de entrada e é armazenado na input\$date.

É importante no server.R quebrar a forma como pensamos sobre os programas R sendo executados linearmente. Em vez disso, pense nas funções sendo executadas reativamente para alterar a entrada da função ui.R. A função renderPrint pega a entrada reativa e a atribui à variável de saída. Observe a sintaxe peculiar para renderPrint nas (R Statements).

Nesse caso, nossa função server.R está apenas recebendo nossos dados inseridos e retornando-os de volta. Nomeamos nossas variáveis de saída oid1, oid2 e odate, os mesmos nomes usados em ui.R. A instrução renderPrint diz que está sendo enviada de volta ao ui.R para exibição formatada.

2.13 Colocando tudo junto

Agora vamos criar nosso algoritmo de previsão. Para os arquivos ui.R, vamos tentar:

```
1 {lang=r, line-numbers=off} ~ shinyUI( pageWithSidebar( # Application title headerPanel(
  Diabetes prediction ),
```

```
1 sidebarPanel(
  numericInput('glucose', 'Glucose mg/dL', 90, min = 50, max = 200, step = 5),
  submitButton('Submit')
),
mainPanel(
  h3('Results of prediction'),
  h4('You entered'),
  verbatimTextOutput("inputValue"),
  h4('Which resulted in a prediction of '),
  verbatimTextOutput("prediction")
)
```

Portanto, temos um painel da barra lateral que absorve o valor da glicose. O submitButton coloca um botão que aguarda até que o botão seja pressionado para enviar valores para sever.R. Discutiremos mais sobre os botões de envio no próximo capítulo. O painel principal mostra a saída. Observe que verbatimTextOutput é a função usada para exibir a saída de nossas funções server.R.

Nossa função server.R é

```
1 {lang=r, line-numbers=off} ~ diabetesRisk <- function(glucose) glucose / 200
2
3 shinyServer( function(input, output) { output inputValue <- renderPrint({input glucose})
  output prediction <- renderPrint({diabetesRisk(input glucose)}) } ) ~
```

Observe que nossa função de previsão é definida fora da função shinyServer. A função shinyServer recebe a entrada e repete o nível de glicose de entrada e o risco de diabetes gerado.

2.14 Outro exemplo

Uma ótima maneira de usar o Shiny é criar gráficos interativos. Vamos seguir um exemplo simples, exatamente como o que fizemos no capítulo sobre manipulação. O benefício de usar o Shiny sobre manipular é a capacidade de compartilhar o aplicativo amplamente como uma página da web.

Aqui está a nossa função ui.R

```
1 {lang=r, line-numbers=off} ~ shinyUI(pageWithSidebar( headerPanel( Example plot ),
  sidebarPanel( sliderInput( mu , Guess at the mean , value = 70, min = 62, max =
  74, step = 0.05,) ), mainPanel( plotOutput( newHist ) ) ) ) ~
```

Observe que plotOutput é a função usada para plotar o histograma gerado. Vamos considerar a função server.R.

```
1 {lang=r, line-numbers=off} ~ library(UsingR) data(galton)
2
3 shinyServer( function(input, output) { output newHist <- renderPlot({ hist(galton child,
  xlab= child height , col= lightblue ,main= Histogram ) mu <- input mulines(
  c(mu,mu),c(0,200),col="red",lwd=5)mse< mean ((galton child - mu)^2) text(63, 150,
  paste( mu = , mu)) text(63, 140, paste( MSE = , round(mse, 2))) }) } ) ~
```

Isso mostra como criamos um conjunto de código um tanto elaborado na instrução renderPlot que gera o gráfico. Observe especificamente a linha `mu <- input$mu`. Esse é o valor do controle deslizante que usamos para gerar nossa linha horizontal.

2.15 Compartilhando seu app

Agora que temos um aplicativo funcional, gostaríamos de compartilhá-lo com o mundo. Você pode simplesmente postar o código e, quaisquer que sejam os arquivos de dados necessários, os usuários poderão usar o runApp para ver o aplicativo. No entanto, é muito melhor exibi-lo como um aplicativo da web independente. Isso requer a execução de um servidor Shiny para hospedar o aplicativo. Em vez de criar e implantar nosso próprio servidor Shiny, contaremos com o serviço shinyapps.io do RStudio. Você pode ir para <https://www.shinyapps.io/>, onde será mostrado como criar uma conta. Depois de criar a conta, você poderá hospedar seus aplicativos brilhantes lá. Observe que este é um serviço freemium, de modo que, se você deseja hospedar muitos aplicativos e possuir recursos sofisticados, precisará do serviço pago (ou hospedar seu próprio servidor brilhante).

Após fazer o login, você precisará fazer algumas instalações básicas. Primeiro, ferramentas. (Este é um pacote essencial por várias razões.) Portanto, `install.packages("devtools")` no prompt de comando do R. Isso permitirá que você instale o pacote shinyapps diretamente do github com o comando

```
devtools::install_github('rstudio/shinyapps')
```

Em seguida, você deve executar algum código que possa copiar do site shinyapps.io. Parece

```
1 {lang=r, line-numbers=off} shinyapps::setAccountInfo(name=      , token=      , secret=
)
```

Isso informa ao RStudio como enviar seu código para shinyapps.io e fornece as permissões para isso. Agora, mude para o diretório em que estão os arquivos server.R e ui.R e você pode enviar seu código com:

```
1 {lang=r, line-numbers=off} ~ deployApp(appName = myFirstApp ) ~
```

Certifique-se de que o shinyapps esteja carregado (com a biblioteca (shinyapps)). Se você precisar de um caminho para seus arquivos, coloque-o no argumento para deployApp. O argumento appName parece ser necessário e você deseja assim mesmo para saber qual aplicativo está em shinyapps.io. Se tudo der certo, o aplicativo será iniciado e abrirá uma janela do navegador com um link para ele. No meu caso, o link foi <https://bcocco.shinyapps.io/myFirstApp>.

Você pode gerenciar seu aplicativo no navegador em shinyapps.io. Agora, quando você acessa shinyapps.io e clica em Aplicativos / Todos ou Aplicativos / Em execução, deve ver seu aplicativo.

A partir daqui, você pode iniciar, parar e excluir seu aplicativo. Você também pode fazer isso no seu console , com as ferramentas shinapps. Eu recomendaria usar o navegador da Web, pois é um pouco mais fácil, mas se você chegar ao ponto em que está escrevendo muitos aplicativos, provavelmente deseja aprender os comandos do console.

2.16 Fale sobre o servidor Shiny posteriormente

É importante distinguir entre um aplicativo Shiny (aplicativo) e um servidor Shiny. É necessário um servidor Shiny para hospedar um aplicativo Shiny para o mundo. Caso contrário, somente aqueles que possuem o Shiny instalado e têm acesso ao seu código poderão executar seu aplicativo da web, o que, em alguns casos, anula o propósito de criar um aplicativo da web em primeiro lugar. Neste livro, não abordaremos a criação de um servidor Shiny, pois isso requer a compreensão de um pouco de administração de servidor Linux. Em vez disso, executaremos nossos aplicativos localmente e usaremos o serviço do RStudio para hospedar aplicativos Shiny (seus servidores) em uma plataforma chamada shinyapps.io.

O RStudio trabalha com o servidor para que você só precise se preocupar com a criação do aplicativo. O Shinyapps.io é gratuito até um ponto em que você só pode executar 5 aplicativos por um determinado período de tempo por mês. Isso será bom para nossos propósitos, mas se você realmente quiser criar aplicativos Shiny, terá que optar por um plano pago ou executar seu próprio servidor.

3 Leaflet

3.1 Iniciando

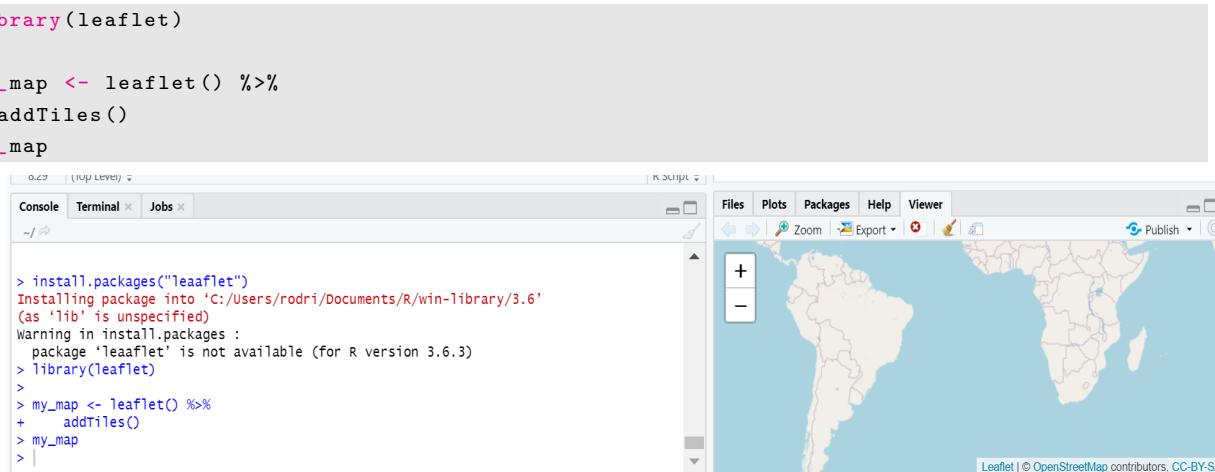
O Leaflet é uma biblioteca Javascript que permite aos usuários criar mapas interativos que até ficam bem em dispositivos móveis. O pessoal maravilhoso do RStudio criou seu próprio pacote Leaflet , que permite aos usuários do  criar seus próprios mapas do Leaflet sem precisar escrever nenhum Javascript. Vamos começar a fazer alguns mapas!

Primeiro você deve instalar o Leaflet:

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

Depois de instalar o leaflet, você pode começar a fazer mapas! Você pode armazenar seu mapa em uma variável como qualquer outro valor. Neste exemplo, vamos armazenar nosso mapa em uma variável chamada my_map. Você precisa usar a função `addTiles()` para adicionar uma camada base ao seu mapa e pode usar o operador de canal (`%>%`) para adicionar recursos sequencialmente ao seu mapa. Veja como você pode criar um mapa básico muito simples:

```
1 library(leaflet)
2
3 my_map <- leaflet() %>%
4   addTiles()
5 my_map
```



```
> install.packages("leaflet")
Installing package into 'C:/Users/rodri/Documents/R/win-library/3.6'
(as 'lib' is unspecified)
Warning in install.packages :
  package 'leaflet' is not available (for R version 3.6.3)
> library(leaflet)
>
> my_map <- leaflet() %>%
+   addTiles()
> my_map
> |
```

Uau!, é o mundo inteiro! Você poderá mover o mapa e aumentar e diminuir o zoom conforme desejar. Você criou seu primeiro mapa interativo! O leaflet usa dados gratuitos e de código aberto do projeto OpenStreetMap.

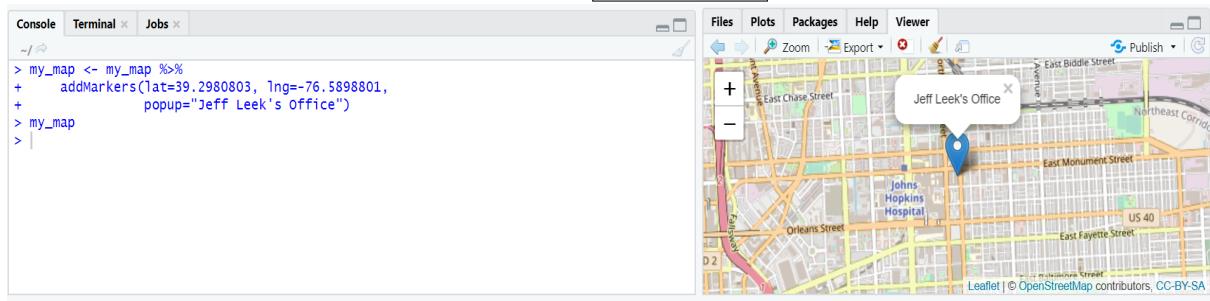
Por mais adorável que o mundo seja, você pode especificar um local específico e um nível de zoom se estiver tentando mostrar um local específico no seu mapa. Você pode fazer isso com a função `setView()` especificando a longitude, latitude e nível de zoom:

```
1 my_map <- my_map %>%
2   setView(lat = 39.2898036, lng = -76.6051842, zoom = 13)
3 my_map
```

Olha, é a linda Baltimore, a maior cidade da América!

3.2 Adicionando marcadores

Este é certamente um mapa mais interessante do que tínhamos antes, mas ainda falta recursos. Vamos adicionar um marcador a este mapa usando a função `addMarkers()`:



Como você pode ver, você pode adicionar um marcador apenas especificando longitude e latitude. Opcionalmente, você pode adicionar algum texto pop-up que aparece quando você clica no marcador. Você não precisa adicionar um marcador de cada vez. Para criar vários marcadores ao mesmo tempo, use um quadro de dados com colunas denominadas lat e lng para latitude e longitude:

```
1 suppressWarnings(set.seed(2016-04-25))
2 df <- data.frame(lat = runif(20, min = 39.2, max = 39.3),
3                   lng = runif(20, min = -76.6, max = -76.5))
4 df %>%
5   leaflet() %>%
6   addTiles() %>%
7   addMarkers()
```

Como você pode ver, você pode usar o operador de tubo para canalizar um quadro de dados diretamente no `leaflet()`.

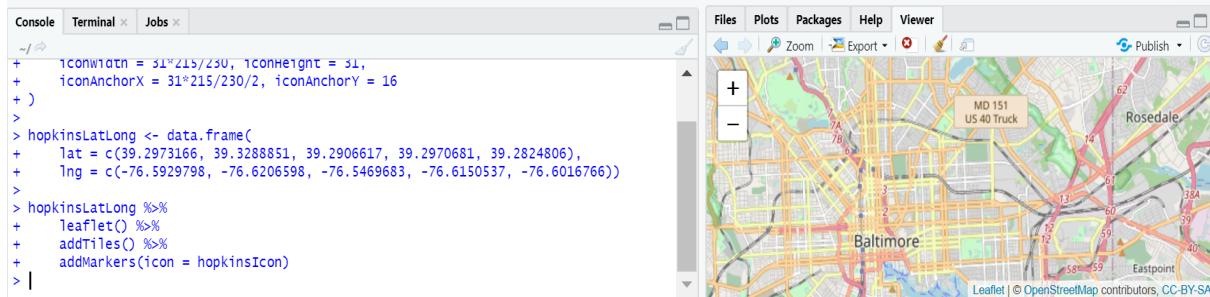
O ícone azul “pins/marcadores” padrão que o Leaflet fornece é bom, mas você também pode criar seu próprio ícone com a função `makeIcon()`. Você precisará fornecer alguns argumentos para essa função, incluindo um link para o URL do seu ícone, além de variáveis de largura e altura. Depois de criar um objeto ícone, você pode usá-lo especificando o argumento `icon` em `addMarkers()`:

```
1 hopkinsIcon <- makeIcon(
2   iconUrl = "http://brand.jhu.edu/content/uploads/2014/06/university.shield.small_blue_.
3   png",
4   iconWidth = 31*215/230, iconHeight = 31,
5   iconAnchorX = 31*215/230/2, iconAnchorY = 16
6 )
7 hopkinsLatLong <- data.frame(
8   lat = c(39.2973166, 39.3288851, 39.2906617, 39.2970681, 39.2824806),
9   lng = c(-76.5929798, -76.6206598, -76.5469683, -76.6150537, -76.6016766))
```

```

10
11 hopkinsLatLong %>%
12   leaflet() %>%
13   addTiles() %>%
14   addMarkers(icon = hopkinsIcon)

```

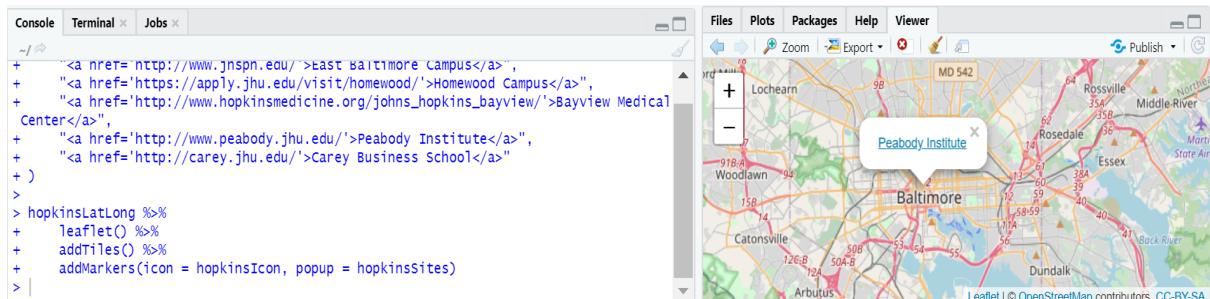


Agora, mapeamos bem todos os locais do campus da Johns Hopkins na cidade de Baltimore. Você pode personalizar pop-ups de texto incluindo qualquer código HTML válido em um vetor. Em seguida, você pode fornecer esse vetor como argumento pop-up em `addMarkers()` da seguinte maneira:

```

1 hopkinsSites <- c(
2   "<a href='http://www.jhsph.edu/'>East Baltimore Campus</a>",
3   "<a href='https://apply.jhu.edu/visit/homewood/'>Homewood Campus</a>",
4   "<a href='http://www.hopkinsmedicine.org/johns_hopkins_bayview/'>Bayview Medical Center</
5     a>",
6   "<a href='http://www.peabody.jhu.edu/'>Peabody Institute</a>",
7   "<a href='http://carey.jhu.edu/'>Carey Business School</a>"
8 )
9
9 hopkinsLatLong %>%
10  leaflet() %>%
11  addTiles() %>%
12  addMarkers(icon = hopkinsIcon, popup = hopkinsSites)

```

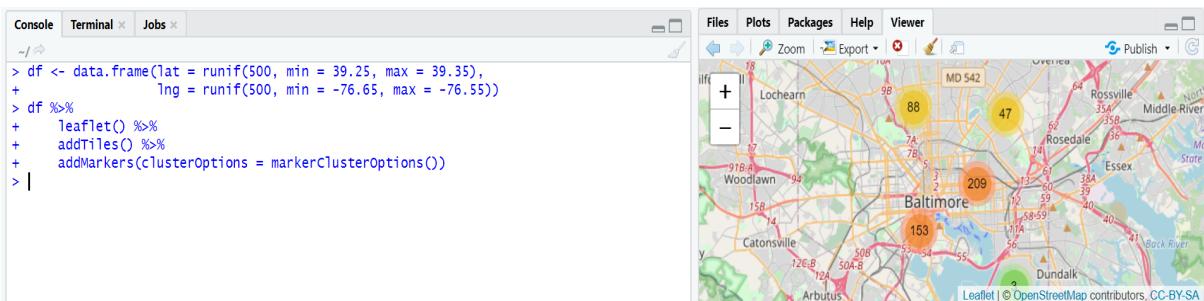


Cada local do campus da Johns Hopkins agora possui um link para o site do respectivo campus!

```

1 df <- data.frame(lat = runif(500, min = 39.25, max = 39.35),
2                   lng = runif(500, min = -76.65, max = -76.55))
3 df %>%
4   leaflet() %>%
5   addTiles() %>%
6   addMarkers(clusterOptions = markerClusterOptions())

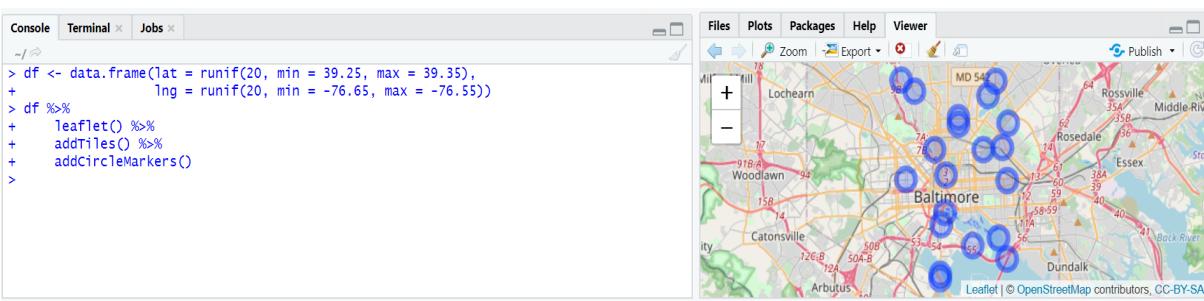
```



```

1 df <- data.frame(lat = runif(500, min = 39.25, max = 39.35),
2                   lng = runif(500, min = -76.65, max = -76.55))
3 df %>%
4   leaflet() %>%
5   addTiles() %>%
6   addMarkers(clusterOptions = markerClusterOptions())

```



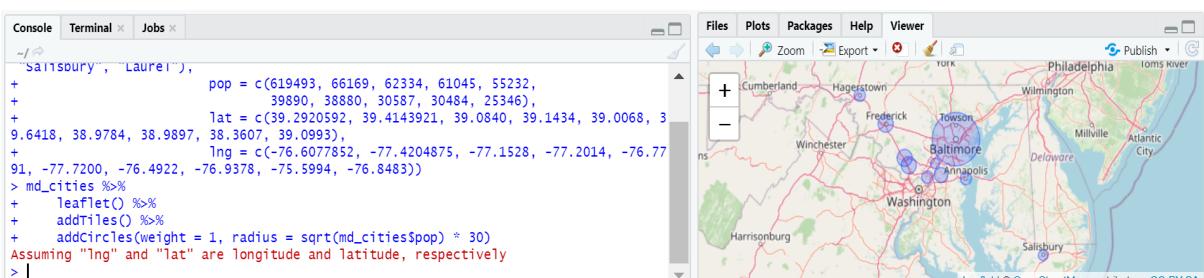
3.3 Desenhandando

```

1 md_cities <- data.frame(name = c("Baltimore", "Frederick", "Rockville", "Gaithersburg",
2                           "Bowie", "Hagerstown", "Annapolis", "College Park", "Salisbury", "Laurel"),
3                           pop = c(619493, 66169, 62334, 61045, 55232,
4                           39890, 38880, 30587, 30484, 25346),
5                           lat = c(39.2920592, 39.4143921, 39.0840, 39.1434, 39.0068, 39.6418,
6                           38.9784, 38.9897, 38.3607, 39.0993),
7                           lng = c(-76.6077852, -77.4204875, -77.1528, -77.2014, -76.7791,
8                           -77.7200, -76.4922, -76.9378, -75.5994, -76.8483))
9 md_cities %>%
10  leaflet() %>%
11    addTiles() %>%
12    addCircles(weight = 1, radius = sqrt(md_cities$pop) * 30)

```

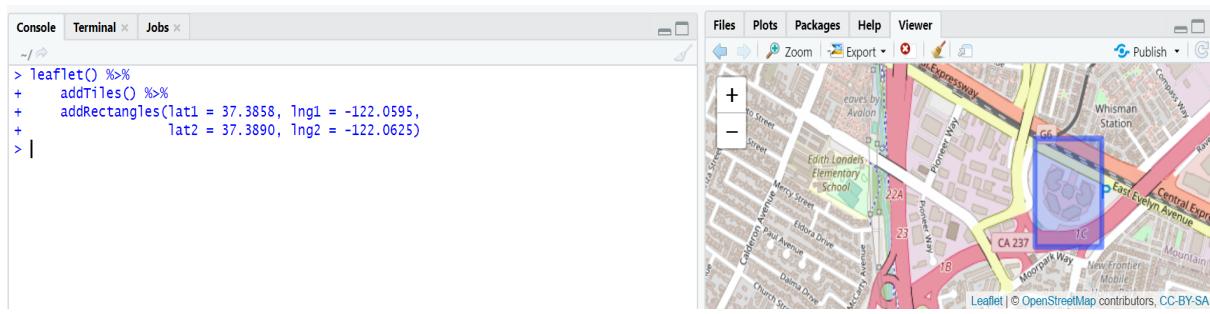
1 ## Assumindo que 'lng' e 'lat' sejam longitude e latitude, respectivamente.



```

1 leaflet() %>%
2   addTiles() %>%
3   addRectangles(lat1 = 37.3858, lng1 = -122.0595,
4                  lat2 = 37.3890, lng2 = -122.0625)

```



```

Console Terminal Jobs
> leaflet() %>%
+   addTiles() %>%
+   addRectangles(lat1 = 37.3858, lng1 = -122.0595,
+                 lat2 = 37.3890, lng2 = -122.0625)
> |
```

```

1 df <- data.frame(lat = runif(20, min = 39.25, max = 39.35),
2                   lng = runif(20, min = -76.65, max = -76.55),
3                   col = sample(c("red", "blue", "green"), 20, replace = TRUE),
4                   stringsAsFactors = FALSE)
5
6 df %>%
7   leaflet() %>%
8   addTiles() %>%
9   addCircleMarkers(color = df$col) %>%
10  addLegend(labels = LETTERS[1:3], colors = c("blue", "red", "green"))

```

3.4 Conclusão

Para mais detalhes sobre o pacote do leaflet para R, visite <http://rstudio.github.io/leaflet/>.

4 Gráficos interativos

O **R** possui um conjunto cada vez mais diversificado de ferramentas para a criação de gráficos interativos. Você já viu alguns com gadgets do Shiny. Essas ferramentas, no entanto, dependem da execução do **R** em segundo plano. Outras ferramentas convertem os gráficos em exibições javascript que podem ser incorporadas em páginas da web e apresentações em html. Veremos uma ótima versão disso mais tarde para mapas com folheto. Uma abordagem relacionada conecta **R** a outros sistemas de plotagem bem desenvolvidos que possuem interatividade e outros recursos interessantes, como tipos de plotagem padrão altamente inovadores.

Duas bibliotecas de plotagem bem desenvolvidas são plot.ly e Google Charts. Ambos possuem excelentes ferramentas para criar, adaptar e exibir gráficos interativos na web. Eles também têm recursos complementares para inserir e manipular os dados, como as folhas do Google. No entanto, geralmente é o caso de análises avançadas de dados em **R**, com o desejo de enviar visualizações finais para um gráfico do Google ou plot.ly. Este capítulo tem esses casos de uso em mente. Os pacotes **R** relevantes são googleVis e plot.ly. No caso de plot.ly, o pacote **R** é criado pela empresa que produz o programa gráfico. No entanto, ambos os pacotes são amplos e bem documentados. Abordaremos o googleVis primeiro.

4.1 googleVis

Assista a esses vídeos antes de começar: [parte 1](#) e [parte 2](#)

A ideia básica de um gráfico do googleVis é que se use **R** para configurar seus dados no formato exigido pelo googleVis, a função googleVis **R** cria uma página HTML relevante. A página HTML resultante usa bibliotecas javascript do Google Charts e o resultado é um gráfico HTML interativo.

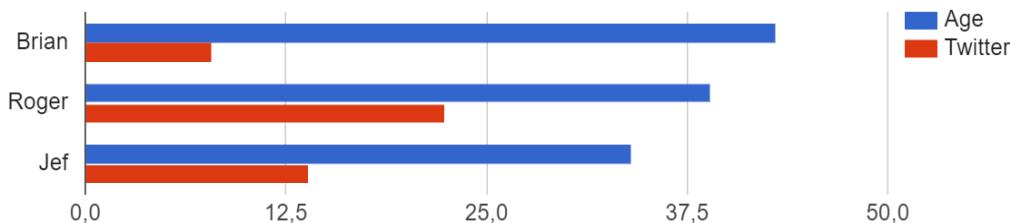
Obviamente, se você ainda não o fez, instale o googleVis (por exemplo, `install.packages("googleVis")`), que só precisa ser feito uma vez e depois carregue a `library(googleVis)`. O pacote googleVis está excepcionalmente bem documentado. Neste capítulo, basicamente passamos por um tour pelo manual para dar aos usuários um empurrão em alguns pontos de partida simples. Depois de terminar o capítulo, você poderá aproveitar esses exemplos para usar todos os aspectos do googleVis.

Um exemplo simples de criação de um gráfico do Google em **R** é dado abaixo. Primeiro, criamos um conjunto de dados simples e depois o plotamos com um gráfico de barras. Vamos criar um conjunto de dados com milhares de seguidores e seguidores do Twitter dos instrutores do Coursera Data Science Specialization. (Observe que as idades são falsas, já que eu não conheço as idades de Jeff ou Roger.)

```

1 ## Cria uma amostra de dados
2 df = data.frame(Instrutor=c("Brian", "Roger", "Jef"),
3                 Idade=c(43,39,34),
4                 Twitter=c(7.9,22.4,13.9))
5 bar = gvisBarChart(df, xvar="Instrutor", yvar=c("Idade","Twitter"))
6 bar

```



Dados: data • Chart ID: BarChartID75ce1b491fa9 • googleVis-0.6.2 R version 3.3.2 (2016-10-31) • [Google Terms of Use](#) • [Documentation and Data Policy](#)

Claramente, Brian precisa intensificar seus tweets.

Se você estiver vendo este livro como uma página da Web, observe a interatividade passando o mouse sobre as linhas. Se você estiver executando o código localmente, poderá ter notado que a barra de chamada não criou o gráfico, print (bar) se comportaria de maneira semelhante. Em vez disso, esses comandos imprimem o HTML relevante para o gráfico. É mostrado desta maneira, pois é isso que é necessário para incorporar a plotagem em um documento de remarcão R compilado com knitr. Para mostrar os resultados, use a opção chunk `results = 'asis'`.

Talvez a maneira mais fácil de visualizar os resultados seja usar o método de plotagem S3 associado aos objetos do googleVis.

```
1 plot(Line)
```

Isso criará um arquivo HTML temporário e o abrirá usando um servidor local. A exibição da trama para o mundo exigirá a hospedagem da trama em outro lugar. Uma maneira fácil de fazer isso, como fizemos neste livro, é incorporar os gráficos usando o knitr e hospedar seu documento do knitr, como você costuma fazer. Como alternativa, você pode pegar o código do gráfico e incorporá-lo a qualquer página da Web hospedada criada. Os desenvolvedores anteciparam esse caso de uso e, portanto, os vários utilitários para imprimir o HTML e o javascript. O mais útil é que o método de impressão associado aos objetos do googleVis imprima o código com várias quantidades de preâmbulo HTML. `?print.gvis` mostra as opções.

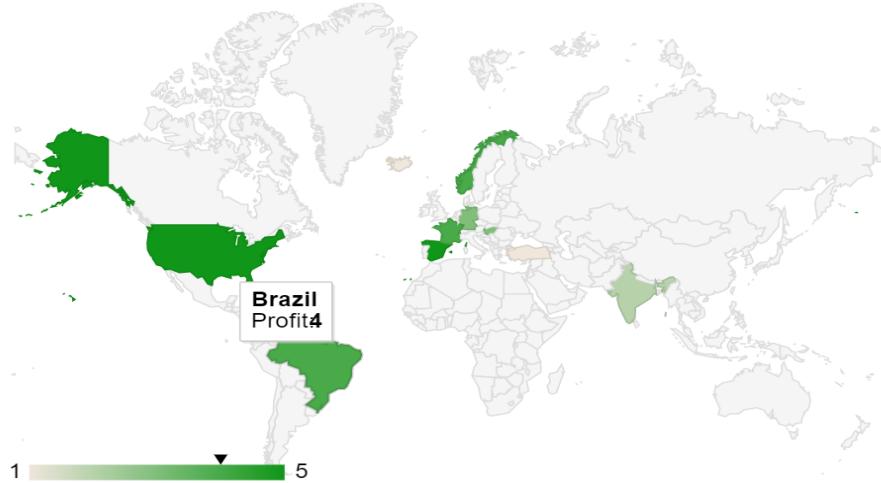
- Motion charts: `gvisMotionChart`
- Interactive maps: `gvisGeoChart`
- Interactive tables: `gvisTable`
- Line charts: `gvisLineChart`
- Bar charts: `gvisColumnChart`
- Tree maps: `gvisTreeMap`

A documentação completa pode ser encontrada [aqui](#). Abordaremos alguns exemplos e, desses, você poderá facilmente generalizar para os outros abordados no pacote.

Um gráfico geográfico é um mapa interativo. Assim, o `googleVis` pode corresponder a grande parte da funcionalidade do leaflet. Considere esta plotagem do conjunto de dados Exports. Primeiro, olhe para os dados.

```
1 head(Exports)
2
3 ##          Country Profit Online
4 ## 1      Germany     3   TRUE
5 ## 2      Brazil      4  FALSE
6 ## 3 United States    5   TRUE
7 ## 4      France      4   TRUE
8 ## 5      Hungary     3  FALSE
9 ## 6      India       2   TRUE
```

```
1 G = gvisGeoChart(Exports, locationvar="Country",
2                   colorvar="Profit", options=list(width=600, height=400))
3 print(G, "chart")
```



As opções em um gráfico do googleVis são fornecidas em um formato de lista. Estes estão bem documentados no manual de ajuda de cada função. Os parâmetros devem ser definidos com o valor nomeado na documentação do Google Charts para o gráfico associado. Por exemplo, considere a função gvisGeoChart. As opções podem ser encontradas [aqui](#).

Por exemplo, você pode ampliar o gráfico em uma região específica especificando region = na lista de opções. Considere focar o mapa na Europa. Os códigos de região são fornecidos [aqui](#).

```
1 G2 = gvisGeoChart(Exports, locationvar="Country",
2                     colorvar="Profit", options=list(width=600, height=400,region="150"))
3 print(G2, "chart")
```



Vamos considerar a configuração de mais opções.

5 Pacotes

[Assista a esse vídeo antes de continuar.](#)

6 Objetos

[Assista a esse vídeo antes de continuar.](#)

7 Swirl

[Assista a esse vídeo antes de continuar.](#)

Referências

- [1] Overleaf, online **L^AT_EX** editor. Disponível em Overleaf.com
- [2] Xie, Y. **Dynamic Documents with R and knitr** 2nd edition, 2015.
- [3] **Reproducible Research using R and Overleaf**. Disponível em [Reproducible Research using RMarkDown and Overleaf](https://ReproducibleResearchUsingRMarkDownAndOverleaf.com)