Swil R: Getting and Cleaning Data - Curso 3

1: Manipulating Data with dplyr

Pacote ‘dplyr’ carregado corretamente!

Nesta lição, você aprenderá como manipular dados usando dplyr. dplyr é um pacote R rápido e poderoso escrito por Hadley Wickham e Romain Francois que fornece uma gramática consistente e concisa para a manipulação de dados tabulares.

Um aspecto único do dplyr é que o mesmo conjunto de ferramentas permite que você trabalhe com dados tabulares de uma variedade de fontes, incluindo dataframes, datatables, bancos de dados e arrays multidimensionais. Nesta lição, vamos nos concentrar em dataframes, mas tudo o que você aprender se aplicará igualmente a outros formatos.

Como você deve saber, "CRAN é uma rede de servidores ftp e web em todo o mundo que armazena versões idênticas e atualizadas de código e documentação para R" (http://cran.rstudio.com/). O RStudio mantém um desses chamados 'espelhos CRAN' e generosamente disponibiliza seus logs de download ao público (http://cran-logs.rstudio.com/). Estaremos trabalhando com o log de 8 de julho de 2014, que contém informações sobre cerca de 225.000 downloads de pacotes.

Eu criei uma variável chamada path2csv, que contém o caminho completo do arquivo para o conjunto de dados. Chame read.csv () com dois argumentos, path2csv e stringsAsFactors = FALSE, e salve o resultado em uma nova variável chamada mydf. Verifique? Read.csv se precisar de ajuda.

mydf <- read.csv(path2csv, stringsAsFactors = FALSE)

dim(mydf) # Visualizar as dimensões

head(mydf) # Visualizar o cabeçalho com as primeiras 6 linhas

library(dplyr) # Carregar o pacote

packageVersion("dplyr") # Visualizar versão do pacote

A primeira etapa para trabalhar com dados no dplyr é carregar os dados em que os autores do pacote chamam um 'data frame tbl' ou 'tbl\_df'. Use o seguinte código para criar um novo tbl\_df chamado cran:

cran <- tbl\_df (mydf).

Para evitar confusão e manter as coisas funcionando sem problemas, vamos remover o dataframe original de sua área de trabalho com rm ("mydf").

De? Tbl\_df, "A principal vantagem de usar um tbl\_df sobre um dataframe regular é a impressão." Vamos ver o que isso significa. Digite cran para imprimir nosso tbl\_df no console.

Essa saída é muito mais informativa e compacta do que obteríamos se imprimíssemos o dataframe original (mydf) no console.

Primeiro, vemos a classe e as dimensões do conjunto de dados. Logo abaixo disso, temos uma prévia dos dados. Em vez de tentar imprimir o conjunto de dados inteiro, dplyr apenas nos mostra as primeiras 10 linhas de dados e apenas as colunas que cabem perfeitamente em nosso console. Na parte inferior, vemos os nomes e classes de qualquer variáveis ​​que não cabiam em nossa tela.

De acordo com a vinheta "Introdução ao dplyr" escrita pelos autores do pacote, "A filosofia dplyr é ter pequenas funções em que cada uma faz bem uma coisa."

Especificamente, dplyr fornece cinco 'verbos' que cobrem a maioria das tarefas fundamentais de manipulação de dados: selecionar (), filtrar (), organizar (), mutar () e resumir (). /select(), filter(), arrange(), mutate(), and summarize()/

Use ?Select para obter a documentação da primeira dessas funções básicas. Os arquivos de ajuda para as outras funções podem ser acessados ​​da mesma forma.

Como pode ser frequentemente o caso, especialmente com conjuntos de dados maiores, estamos apenas interessado em algumas das variáveis. Use select (cran, ip\_id, package, country) para selecione apenas as variáveis ​​ip\_id, package e country do conjunto de dados cran.

A primeira coisa a notar é que não precisamos digitar cran $ ip\_id, cran $ package, e cran $ country, como normalmente faríamos ao nos referir a colunas de um dataframe. A função select () sabe que estamos nos referindo às colunas do conjunto de dados cran.

select(cran, ip\_id, package, country) # Selecionar no cran apenas as variaveis de interesse

Além disso, observe que as colunas são retornadas para nós na ordem que especificamos, mesmo embora ip\_id seja a coluna mais à direita no conjunto de dados original.

Lembre-se de que em R, o operador `:` fornece uma notação compacta para a criação de um seqüência de números. Por exemplo, tente **5:20**.

Normalmente, essa notação é reservada para números, mas select () permite que você especifique uma sequência de colunas dessa forma, o que pode economizar muita digitação. Use **select (cran, r\_arch: country)** para selecionar todas as colunas começando de r\_arch eterminando com país.

Também podemos selecionar as mesmas colunas na ordem inversa. De uma chance.

**select(cran, country:r\_arch)** # Selecionar na ordem inversa

Em vez de especificar as colunas que queremos manter, também podemos especificar as colunas que queremos descartar. Para ver como isso funciona, **select (cran, -time)** para omitir a coluna de tempo.

O sinal negativo na frente do tempo diz a select () que NÃO queremos a coluna de tempo. Agora, vamos combinar estratégias para omitir todas as colunas de X até o tamanho (X: tamanho). Para ver como isso pode funcionar, vamos dar uma olhada em um exemplo numérico com **-5: 20**.

Ops! Isso nos deu um vetor de números de -5 a 20, que não é o que queremos. Em vez disso, queremos negar toda a sequência de números de 5 a 20, para obter -5, -6, -7, ..., -18, -19, -20. Tente a mesma coisa, exceto colocar 5:20 entre parênteses para que R saiba que queremos primeiro a sequência de números e, em seguida, aplique o sinal negativo a tudo.

-5:20 # Apenas do 1 ao 5 fica negativos

-(5:20) # Coloca todos os numeros em negativo

Use este conhecimento para omitir todas as colunas X: size usando select ().

select(cran, -(X:size)) # Omitir todas as colunas de X ate size

Agora que você sabe como selecionar um subconjunto de colunas usando select (), a próxima pergunta natural é "Como seleciono um subconjunto de linhas?" É aí que o filter() a função entra.

filter(cran, package == "swirl") # selecionar todas as linhas para as quais a variável de pacote é igual a "swirl"

# Repare que filter () reconhece 'pacote' como uma coluna de cran, sem eu especificar explicitamente cran$package.

O operador == pergunta se o que está à esquerda é igual ao que está à direita. Se sim, ele retorna TRUE. Se não, então FALSE. Neste caso, o pacote é um vetor inteiro (coluna) de valores, então package == "swirl" retorna um vetor de TRUEs e FALSEs. filter () então retorna apenas as linhas de cran correspondentes aos TRUEs.

Você pode especificar quantas condições desejar, separadas por vírgulas. Por exemplo **filter (cran, r\_version == "3.1.1", country == "US")** retornará todas as linhas de cran correspondendo a downloads de usuários nos EUA executando a versão R 3.1.1. Experimente.

As condições passadas para filter () podem fazer uso de qualquer uma das comparações padrão operadores. Puxe a documentação relevante com ?Comparison (que é um C maiúsculo).

Nossas duas últimas chamadas para filter() solicitaram todas as linhas para as quais alguma condição e outra condição eram TRUE. Também podemos solicitar linhas para as quais qualquer um condição ou outra condição são [TRUE. Por](https://ssl.microsofttranslator.com/bv.aspx?ref=TAns&from=&to=pt&a=VERDADEIRAs.Por) exemplo, **filter (cran, country == "US" | país == "IN")** nos dará todas as linhas para as quais a variável do país é igualou "EUA" ou "IN".Vamos tentar.

Agora, use filter () para buscar todas as linhas cujo tamanho seja estritamente maior que (>) 100500 (sem aspas, pois o tamanho é numérico) E r\_os é igual a "linux-gnu". Dica: você está passando três argumentos para filter (): o nome do conjunto de dados, o primeiro condição e a segunda condição.

Este é complicado**. filter (cran, size> 100500, r\_os == "linux-gnu")** nos dará todas as linhas para as quais o tamanho é estritamente maior que 100500 e r\_os é "linux-gnu".

Finalmente, queremos obter apenas as linhas para as quais a r\_version não está faltando. R representa valores ausentes com NA e esses valores ausentes podem ser detectados usando a função is.na ().

Para ver como isso funciona, tente **is.na (c (3, 5, NA, 10)).**

Agora, coloque um ponto de exclamação (!) Antes de is.na () para alterar todos os TRUEs para FALSEs e todos os FALSEs para TRUEs, nos dizendo o que NÃO é NA: **!Is.na (c (3, 5, NA, 10))**

Ok, pronto para colocar tudo isso junto? Use filter () para retornar todas as linhas de cran para o qual r\_version NÃO é NA. Dica: você precisará usar! Is.na () como parte de seu segundo argumento para filter ().

f**ilter(cran, !is.na(r\_version))**

Vimos como selecionar um subconjunto de colunas e linhas de nosso conjunto de dados usando selecione () e filtro (), respectivamente. Inerente em select () também estava a capacidade de Organizar nossas colunas selecionadas em qualquer ordem que desejarmos.

Às vezes, queremos ordenar as linhas de um conjunto de dados de acordo com os valores de uma determinada variável. Este é o trabalho de organize ().

Para ver como organizar () funciona, vamos primeiro pegar um subconjunto de cran. select () todas as colunas de size até ip\_id e armazene o resultado em cran2.

**cran2 <- select(cran, size:ip\_id)**

Agora, para ordenar as ROWS de cran2 de forma que ip\_id esteja em ordem crescente (de pequeno para grande), digite organizar (cran2, ip\_id). Você pode querer deixar seu console largo o suficiente para que você possa ver ip\_id, que é a última coluna.

**arrange(cran2, ip\_id)** # Organizar em ordem crescente

Para fazer o mesmo, mas em ordem decrescente, altere o segundo argumento para desc (ip\_id), onde desc () significa 'decrescente'. Vá em frente.

**arrange(cran2, desc(ip\_id))** # Organizar em ordem decrescente

Também podemos organizar os dados de acordo com os valores de várias variáveis. Para

Por exemplo, arrange (cran2, pacote, ip\_id) primeiro organizará por nomes de pacotes (em ordem crescente em ordem alfabética) e, em seguida, por ip\_id. Isso significa que se houver várias linhas com o mesmo valor para o pacote, elas serão classificadas por ip\_id (crescente numericamente). Tente **arrange (cran2, pacote, ip\_id)** agora.

Organize cran2 pelas três variáveis ​​a seguir, nesta ordem: país (crescente), r\_version (decrescente) e ip\_id (crescente).

Para ilustrar a próxima função principal em dplyr, vamos pegar outro subconjunto de nosso dados originais. Use select () para pegar 3 colunas de cran - ip\_id, pacote e tamanho (nessa ordem) - e armazene o resultado em uma nova variável chamada cran3.

É comum criar uma nova variável com base no valor de uma ou mais variáveis ​​já em um conjunto de dados. A função mutate () faz exatamente isso.

A variável size representa o tamanho do download em bytes, que são unidades de memória do computador. Hoje em dia, megabytes (MB) são uma unidade de medida mais comum.

Um megabyte é igual a 2 ^ 20 bytes. Isso é 2 elevado a 20, o que é aproximadamente um milhão de bytes!

Queremos adicionar uma coluna chamada size\_mb que contém o tamanho do download em

megabytes. Aqui está o código para fazer isso:

mutate (cran3, size\_mb = size / 2 ^ 20)

Uma unidade de memória ainda maior é um gigabyte (GB), o que equivale a 2 ^ 10 megabytes. Nós

pode muito bem adicionar outra coluna para o tamanho do download em gigabytes!

Um recurso muito bom de mutate () é que você pode usar o valor calculado para sua segunda coluna (size\_mb) para criar uma terceira coluna, tudo na mesma linha de código. Para ver isso em ação, repita exatamente o mesmo comando acima, mas adicione um terceiro

argumento que cria uma coluna chamada size\_gb e igual a size\_mb / 2 ^ 10.

Vamos tentar mais um para praticar. Finja que descobrimos uma falha no sistema que forneceu os valores originais para a variável de tamanho. Todos os valores em cran3 são

1000 bytes a menos do que deveriam ser. Usando cran3, crie apenas uma nova coluna

chamado correct\_size que contém o tamanho correto.

**mutate(cran3, correct\_size = size + 1000)**

O último dos cinco verbos dplyr principais, summarize (), reduz o conjunto de dados em uma única linha. Digamos que estamos interessados ​​em saber o tamanho médio do download.

summarize(cran, avg\_bytes = média (tamanho)) resultará no valor médio do tamanho variável. Aqui, escolhemos rotular o resultado como 'avg\_bytes', mas poderíamos ter nomeado qualquer coisa. De uma chance.

Isso não é particularmente interessante. sumarizar () é mais útil ao trabalhar com dados que foram agrupados pelos valores de uma variável específica.

Veremos os dados agrupados na próxima lição, mas a ideia é mutate () pode fornecer o valor solicitado PARA CADA grupo em seu conjunto de dados.

Nesta lição, você aprendeu como manipular dados usando os cinco principais funções. Na próxima lição, veremos como tirar proveito de alguns outros recursos úteis do dplyr para tornar sua vida como analista de dados muito mais fácil.

**swirl Lesson 2: Grouping and Chaining with dplyr**

Aviso: esta lição usa a função View (). View () pode não funcionar corretamente em cada ambiente de programação. Recomendamos fortemente o uso do RStudio para esta lição.

Na última lição, você aprendeu sobre os cinco principais 'verbos' de manipulação de dados em dplyr: select(), filter(), arrange(), mutate(), and summarize(), é mais poderoso quando aplicado a dados agrupados.

Continuaremos de onde paramos com o registro de download CRAN do RStudio de 8 de julho, 2014, que contém informações sobre cerca de 225.000 downloads de pacotes R (http://cran-logs.rstudio.com/).

Como na última lição, o pacote dplyr foi instalado automaticamente (se necessário) e carregado no início desta lição. Normalmente, isso é algo que você teria que fazer sozinho. Apenas para construir o hábito, digite **library(dplyr)** agora para carregar o pacote novamente.

Disponibilizei o conjunto de dados para você em um quadro de dados chamado mydf. Coloque o em um 'data frame tbl' usando a função tbl\_df () e armazene o resultado em um objeto chamado cran. Se você não tem certeza do que estou falando, deve começar com a lição anterior. Caso contrário, a prática leva à perfeição!

**cran <- tbl\_df(mydf)** # Pegar \*mydf\* tranaformando em \*tbl\_df\* e amazenando-a em nova variavel \*cran\*

**rm('mydf')** # Remover dado original

**cran** # Visualizar no console

Nosso primeiro objetivo é agrupar os dados por nome de pacote. Abra o arquivo de ajuda para group\_by ().

**?group\_by** # Abrir arquivo de ajuda

Agrupe cran pela variável de pacote e armazene o resultado em um novo objeto chamado by\_package.

**by\_package <- group\_by(cran, package)**

No topo da saída acima, você verá 'Grupos: pacote', que nos diz que este tbl foi agrupado pela variável de pacote. Tudo o resto parece o mesmo, mas agora qualquer operação que aplicamos aos dados agrupados ocorrerá por pacote.

Lembre-se de que quando aplicamos a média (tamanho) ao tbl\_df original por meio de sumarizar (), retornou um único número - a média de todos os valores na coluna de tamanho. Podemos nos preocupar com qual é esse número, mas não seria muito mais interessante observar o tamanho médio do download para cada pacote exclusivo?

Isso é exatamente o que você obterá se usar summarize() para aplicar a mean(size) aos dados agrupados em by\_package. Dê uma chance.

**summarize(by\_package, mean(size))** # Observar o tamanho médio do download para cada pacote exclusivo.

Em vez de retornar um único valor, resume () agora retorna o tamanho médio para CADA pacote em nosso conjunto de dados.

Vamos dar um passo adiante. Acabei de abrir um script R para você que contém uma chamada parcialmente construída para summarize (). Siga as instruções nos comentários do script. Quando estiver pronto para prosseguir, salve o script e digite submit () ou digite reset () para redefinir o script ao seu estado original.

# Compute four values, in the following order, from

# the grouped data:

#

# 1. count = n()

# 2. unique = n\_distinct(ip\_id)

# 3. countries = n\_distinct(country)

# 4. avg\_bytes = mean(size)

#

# A few thing to be careful of:

#

# 1. Separate arguments by commas

# 2. Make sure you have a closing parenthesis

# 3. Check your spelling!

# 4. Store the result in pack\_sum (for 'package summary')

# You should also take a look at ?n and ?n\_distinct, so

# that you really understand what is going on.

pack\_sum <- summarize(by\_package,

count = n(),

unique = n\_distinct(ip\_id),

countries = n\_distinct(country),

avg\_bytes = mean(size))

submit()

# A coluna \*count\*, criada com n (), contém o número total de linhas (ou seja, downloads) para cada pacote.

# A coluna \*unique\*, criada com n\_distinct (ip\_id), fornece o número total de downloads exclusivos para cada pacote, medido pelo número de ip\_ids distintos.

# A coluna \*countries\*, criada com n\_distinct (country), fornece o número de países em que cada pacote foi baixado.

# A coluna 'avg\_bytes', criada com mean(size), contém o tamanho médio do download (em bytes) para cada pacote.

A coluna 'contagem', criada com n (), contém o número total de linhas (ou seja, downloads) para cada pacote. A coluna 'exclusivo', criada com n\_distinct (ip\_id), fornece o número total de downloads exclusivos para cada pacote, medido pelo número de ip\_ids distintos. A coluna 'países', criada com n\_distinct (país), fornece o número de países em que cada pacote foi baixado. E, finalmente, a coluna 'avg\_bytes', criada com média (tamanho), contém o tamanho médio do download (em bytes) para cada pacote.

É importante que você entenda como cada coluna de pack\_sum foi criada e o que isso significa. Agora que resumimos os dados por pacotes individuais, vamos brincar um pouco mais para ver o que podemos aprender.

Naturalmente, gostaríamos de saber quais pacotes eram mais populares no dia em que os dados foram coletados (8 de julho de 2014). Vamos começar isolando o 1% principal de pacotes, com base no número total de downloads medido pela 'contagem' coluna

Precisamos saber o valor de ' count ' que divide os dados em 1% superior e 99% inferior dos pacotes com base no total de downloads. Em estatística, isso é chamado de quantil de amostra de 0,99 ou 99%. Use quantile (pack\_sum $ count, probs = 0,99) para determinar este número.

**quantile(pack\_sum$count, probs = 0.99)** # Saber o valor de \*count\* que divide os dados em 1% superior e 99% inferior dos pacotes com base no total de downloads.

Agora podemos isolar apenas os pacotes que tiveram mais de 679 downloads no total. Use filter () para selecionar todas as linhas de pack\_sum para as quais 'count' é estritamente maior (>) que 679. Armazene o resultado em um novo objeto chamado top\_counts.

quantile(pack\_sum$count, probs = 0.99) # Saber o valor de \*count\* que divide os dados em 1% superior e 99% inferior dos pacotes com base no total de downloads.

top\_counts <- filter(pack\_sum, count > 679.)

top\_counts

View(top\_counts)

arrange () as linhas de top\_counts com base na coluna 'count' e atribua o resultado para um novo objeto chamado top\_counts\_sorted. Queremos os pacotes com o maior número de downloads no topo, o que significa que queremos 'contagem' para estar em ordem decrescente. Se precisar de ajuda, consulte ?arrange e / ou ?desc.

**top\_counts\_sorted <- arrange(top\_counts, desc(count))** # Organizar em ordem decrescente a coluna \*count\* e atribuir a uma nova variavel

View(top\_counts\_sorted)

Se usarmos o número total de downloads como nossa métrica de popularidade, então o

saída mostra-nos os pacotes mais populares baixados do espelho RStudio CRAN

em 8 de julho de 2014. Sem surpresa, ggplot2 lidera o pacote com 4602 downloads, seguido por Rcpp, plyr, rJava,

... E se continuar, vai ver o turbilhão no número 43, com 820 no total

Talvez estejamos mais interessados ​​no número de downloads \* exclusivos \* neste dia específico. Em outras palavras, se um pacote for baixado dez vezes em um dia do mesmo computador, podemos contar como apenas um download. Isso é

o que a coluna \* única \* nos dirá.

Como fizemos com 'contagem', vamos encontrar o quantil de 0,99, ou 99%, para o 'único'

variável com quantil (pack\_sum $ unique, probs = 0,99).

**quantile(pack\_sum$unique, probs = 0.99)** # Para obter o quantil de 0,99, ou 99% para \*unique\*

Aplique o filtro () ao pack\_sum para selecionar todas as linhas correspondentes aos valores de 'único'

que são estritamente maiores que 465. Atribua o resultado a um objeto chamado

top\_unique.

top\_unique <- filter(pack\_sum, unique > 465) # selecionar todas as linhas correspondentes aos valores de \*unique\* que são estritamente maiores que 465.

View(top\_unique)

Agora organize () top\_unique pela coluna 'exclusivo', em ordem decrescente, para ver quais pacotes foram baixados do maior número de endereços IP exclusivos.

Atribua o resultado a top\_unique\_sorted.

top\_unique\_sorted <- arrange(top\_unique, desc(unique)) # Organizar em ordem decrescente, para ver quais pacotes foram mais baixados do maior número de endereços IP unique.

View(top\_unique\_sorted)

Agora, Rcpp está na liderança, seguido por stringr, digest, plyr e ggplot2. redemoinho

subiu alguns espaços para o número 40, com 698 downloads exclusivos. Bom!

Nossa métrica final de popularidade é o número de países distintos de onde cada pacote foi baixado. Faremos uma abordagem um pouco diferente para apresentar a você um método chamado 'encadeamento' (ou 'piping').

O encadeamento permite que você agrupe várias chamadas de função de uma forma compacta e legível, ao mesmo tempo em que obtém o resultado desejado. Para fazer isso

mais concreto, vamos calcular nossa última métrica de popularidade do zero, começando com nossos dados originais.

Abri um script que contém um código semelhante ao que você viu até agora. Não mude nada. Apenas estude por um minuto, certifique-se de entender tudo o que está lá e, em seguida, envie () quando estiver pronto para prosseguir.

# Don't change any of the code below. Just type submit()

# when you think you understand it.

# We've already done this part, but we're repeating it

# here for clarity.

by\_package <- group\_by(cran, package)

pack\_sum <- summarize(by\_package,

count = n(),

unique = n\_distinct(ip\_id),

countries = n\_distinct(country),

avg\_bytes = mean(size))

# Here's the new bit, but using the same approach we've

# been using this whole time.

top\_countries <- filter(pack\_sum, countries > 60)

result1 <- arrange(top\_countries, desc(countries), avg\_bytes)

# Print the results to the console.

print(result1)

É importante notar que classificamos principalmente por país, mas usamos avg\_bytes (em ordem crescente) como desempate. Isso significa que se dois pacotes foram baixados do mesmo número de países, o pacote com uma média menor

o tamanho do download recebeu uma classificação mais elevada.

Gostaríamos de obter o mesmo resultado do último script, mas evite salvar nossos resultados intermediários. Isso requer a incorporação de chamadas de função umas dentro das outras.

Isso é exatamente o que fizemos neste script. O resultado é equivalente, mas o código é muito menos legível e alguns dos argumentos estão longe da função a que pertencem. Novamente, apenas tente entender o que está acontecendo

aqui, em seguida, submit () quando estiver pronto para ver uma solução melhor.

# Don't change any of the code below. Just type submit()

# when you think you understand it. If you find it

# confusing, you're absolutely right!

result2 <-

arrange(

filter(

summarize(

group\_by(cran,

package

),

count = n(),

unique = n\_distinct(ip\_id),

countries = n\_distinct(country),

avg\_bytes = mean(size)

),

countries > 60

),

desc(countries),

avg\_bytes

)

print(result2)

submit()

Neste script, usamos um operador de encadeamento especial,%>%, que foi originalmente introduzido no pacote magrittr R e agora se tornou um componente chave do dplyr.

Você pode obter a documentação relacionada com ?chain. A vantagem de%>% é que nos permite encadear as chamadas de função de maneira linear. O código à direita de%>% opera no resultado do código à esquerda de%>%.

Mais uma vez, tente entender o código e digite submit () para continuar.

# Read the code below, but don't change anything. As

# you read it, you can pronounce the %>% operator as

# the word 'then'.

#

# Type submit() when you think you understand

# everything here.

result3 <-

cran %>%

group\_by(package) %>%

summarize(count = n(),

unique = n\_distinct(ip\_id),

countries = n\_distinct(country),

avg\_bytes = mean(size)

) %>%

filter(countries > 60) %>%

arrange(desc(countries), avg\_bytes)

# Print result to console

print(result3)

submit()

Portanto, os resultados dos três últimos scripts são todos idênticos. Mas, o terceiro script fornece uma alternativa conveniente e concisa para o mais tradicional

método que usamos anteriormente, que envolve salvar os resultados à medida que avançamos.

Parece que o Rcpp está no topo com downloads de 84 países diferentes, seguido por digest, stringr, plyr e ggplot2. O swirl subiu na classificação novamente, desta vez para o 27º lugar.

Para ajudar a esclarecer, vamos trabalhar com mais alguns exemplos de encadeamento

Vamos construir uma cadeia de comandos dplyr um passo de cada vez, começando com o script que acabei de abrir para você.