

Lista Prática 2

Exercício 1 (*swirl*). Usando o pacote *swirl*, acesse o curso “R Programming” e faça as seguintes lições:

- a) 10: *lapply and sapply*
- b) 12: *Looking at Data*
- c) 13: *Simulation*

Adicionalmente, instale o curso “Getting and Cleaning Data” usando a seguinte função `swirl::install_course("Getting and Cleaning Data")` e faça as seguintes lições:

- d) 1: *Manipulating Data with dplyr*
- e) 2: *Grouping and Chaining with dplyr*

Ao instalar o curso e entrar na lição “1: Manipulating Data with dplyr” aparecerá o erro:

```
Error in yaml.load(readLines(con, warn = readLines.warn), error.label = erro  
r.label, :  
(C:/Users/fhnis_r40seqz/AppData/Local/Programs/R/R-4.3.0/library/swirl/Cour  
ses/Getting_and_Cleaning_Data/Manipulating_Data_with_dplyr/lesson.yaml) Scann  
er error: while scanning a tag at line 205, column 9 did not find expected wh  
itespace or line break at line 205, column 19
```

Copie o endereço até a pasta “Manipulating_Data_with_dplyr” (destacado na imagem acima) e substitua o arquivo “lesson.yaml” pelo arquivo disponível em:

<https://fhnishida.netlify.app/project/rec5004/lesson.yaml>

Por fim, use `swirl::install_course("Exploratory Data Analysis")` para instalar o curso “Exploratory Data Analysis” e faça as seguintes lições:

- f) 5: *Base Plotting System*
- g) 9: *GGPlot2 Part2*¹

Exercício 2. A base de dados deste exercício foi obtida do site *Hospital Compare* (<http://hospitalcompare.hhs.gov>) administrado pelo Departamento de Saúde e Serviços Humanos dos EUA. O propósito deste site é prover dados e informação sobre a qualidade dos tratamentos de mais de 4 mil hospitais certificados pelo Medicare. O arquivo que utilizaremos é o *medicare.csv*², que contém informação sobre taxa de mortalidade de 30 dias para alguns problemas de saúde.

Comece carregando a base de dados com o seguinte código e observe a sua estrutura:

¹O 1º exercício irá pedir para fazer um gráfico usando `qplot()` que é uma função pouco utilizada. Você pode dar `skip()` para avançar para a parte com gráficos usando a função `ggplot()`

²Versão resumida da base de dados fornecida no curso da John Hopkins no Coursera

```
1 outcome = read.csv2("https://fhnishida.netlify.app/project/rec5004/medicare.csv")
```

Como exercício, escreva a função `best()` que terá dois argumentos: (I) a sigla de um Estado americano, e (II) o nome do problema de saúde. Essa função, a partir base de dados `outcome`, retorna um vetor de texto com o nome do hospital que possui a melhor (menor) taxa de mortalidade (Death Rate), dadas sigla de Estado e problema de saúde. O nome do hospital está na coluna “Hospital.Name” e os problemas de saúde podem ser “heart attack”, “heart failure” e “pneumonia”. Se houver empate de melhor hospital, o nome do hospital a ser retornado pela função é dado pela ordem alfabética – se “Hospital A” e “Hospital B” estiverem empatados, deve-se retornar “Hospital A”. A função deve ter a seguinte estrutura:

```
1 best = function(estado, problema) {
2   # Filtre a base de dados pela sigla do estado
3
4   # Reordene a coluna taxa de mortalidade da doença selecionada de forma
   # crescente e, em caso de empate, coloque os nomes dos hospitais de forma alfabé
   # tica.
5
6   # Retorne o nome do hospital com a menor taxa de mortalidade
7 }
```

Como referência, seguem alguns exemplos de output da função:

```
1 > best("TX", "heart failure")
2 [1] "FORT DUNCAN MEDICAL CENTER"
3
4 > best("MD", "heart attack")
5 [1] "JOHNS HOPKINS HOSPITAL, THE"
```

Usando a função `best()` escrita, quais são os resultados retornados para os casos:

- a) `estado = "SC"` e `problema = "heart attack"`
- b) `estado = "NY"` e `problema = "pneumonia"`
- c) `estado = "AK"` e `problema = "pneumonia"`

Resposta: A ideia aqui é reordenar a base de dados, de forma que o hospital desejado fique na primeira linha (e seu nome será retornado pela função `best()`). Um das dificuldades é que precisamos reordenar uma das três colunas da taxa de mortalidade, além do nome do hospital. Uma forma de fazer isso é criando uma estrutura condicional para cada uma das doenças (*heart attack*, *heart failure* e *pneumonia*) e reordenando via função `arrange()`. Usando esta função, precisamos informar a coluna de taxa de mortalidade da doença incluída no argumento `problema` e, depois, o nome do hospital (para caso haja empate de nas taxas de mortalidade).

```
1 best = function(estado, problema) {
2   # Filtre a base de dados pela sigla do estado
3   bd = outcome %>% filter(State == estado)
4
5   # Reordene a coluna de taxa de mortalidade da doença selecionada
6   # de forma crescente e, em caso de empate, coloque os nomes dos
7   # hospitais de forma alfabética.
8   if (problema == "heart attack") {
9     bd = bd %>% arrange(DeathRatesHeartAttack, Hospital.Name)
10  } else if (problema == "heart failure") {
```

```

11     bd = bd %>% arrange(DeathRatesHeartFailure, Hospital.Name)
12   } else if (problema == "pneumonia") {
13     bd = bd %>% arrange(DeathRatesPneumonia, Hospital.Name)
14   }
15
16   # Retorne o nome do hospital com a menor taxa de mortalidade
17   bd$Hospital.Name[1]
18 }

```

- a) "MUSC MEDICAL CENTER"
- b) "MAIMONIDES MEDICAL CENTER"
- c) "YUKON KUSKOKWIM DELTA REG HOSPITAL"

□

Exercício 3. Carregue bases de dados de GDP para 190 países ranqueados e de informações educacionais a partir do seguinte código:

```

1 gdp = read.csv2("https://fhnishida.netlify.app/project/rec5004/GDP.csv")
2 educ = read.csv2("https://fhnishida.netlify.app/project/rec5004/EDSTATS.csv")

```

- a) Mescle as duas bases de dados a partir do código de país de 3 dígitos. Quantos códigos tiveram correspondência em ambas bases? Organize a base de dados mesclada de forma decrescente no GDP (de modo que USA fica na última linha). Qual é o país na 13^a linha?
- b) Quais são os rankings médios para os grupos de renda “High income: OECD” e “High income: nonOECD”?
- c) Divida os países em 5 grupos de (quase) mesmo tamanho de acordo com seus GDPs, criando a variável “GDP.Group”. Depois, faça uma tabela cruzada entre ela e “Income.Group”. Quantos países estão entre as nações com maior renda (High: OECD/nonOECD) e estão entre o 1^o e 2^o quintis do GDP?

Resposta:

- a) 189 correspondências e 13^o país é “St. Kitts and Nevis”. A maneira mais fácil de saber a quantidade de correspondências ao mesclar duas bases é fazendo um *inner join* (junção que mantém apenas valores presentes em ambas bases) e contando a quantidade linhas restantes.

```

1 library(dplyr)
2 ## JUNTANDO BASES E MANTENDO APENAS CASOS QUE APARECEM EM AMBAS
3 joined = merge(gdp, educ, by.x="Country", by.y="CountryCode", all=FALSE)
4 nrow(joined) # núm. de correspondências entre as duas bases (já que ALL=FALSE)

```

```

1 [1] 189

```

```

1 # Ordenando GDP de forma crescente
2 joined = joined %>% arrange(GDP)
3 joined[13, 1:4] # 13o país na ordem decrescente (e 4 primeiras colunas)

```

```

1 CountryCode Rank Long.Name.x GDP
2 KNA 178 St. Kitts and Nevis 767

```

- b) 32.9 e 91.9. Podemos filtrar a base de dados para cada um dos grupos e fazer as médias, mas há uma forma mais “elegante” usando `group_by()` e `summarise()`. Podemos agrupar a base de dados pela variável categórica `Income.Group` e fazer as médias de rank:

```

1 ## AVERAGE RANKING
2 joined %>% group_by(Income.Group) %>% summarise(
3   mean_rank = mean(Ranking)
4 )

```

```

1 Income.Group mean_rank
2 1 High income: OECD 33.0
3 2 High income: nonOECD 91.9
4 3 Low income 134.
5 4 Lower middle income 108.
6 5 Upper middle income 92.1

```

- c) 5 (4 nonOECD + 1 OECD). Primeiro, precisamos criar a variável `Group.GDP` que é uma transformação de variável contínua em categorias, usando a função `cut()`. Na variável categoria em questão haverá 5 grupos de GDP de tamanhos (quase) iguais. Portanto, além de saber quantis de mínimo e máximo (0% e 100%), precisamos encontrar os 4 quintis (20%, 40%, 60% e 80%). Para isto, usaremos a função `seq()` para criar o vetor de quantis {0%, 20%, 40%, 60%, 80%, 100%}, que usaremos para informar as quebras dos intervalos de GDP:

```

1 joined = joined %>% mutate(
2   GDP.Group = cut(GDP, breaks=quantile(GDP, breaks=seq(0, 1, 0.20)))
3 )
4 class(joined$GDP.Group) # Factor
5 levels(joined$GDP.Group) # Categorias de forma crescente

```

```

1 [1] "factor"
2
3 [1] "(40,4.25e+03]" "(4.25e+03,1.6e+04]" "(1.6e+04,5.08e+04]"
4 [4] "(5.08e+04,2.63e+05]" "(2.63e+05,1.62e+07]"

```

Agora, podemos fazer uma tabela cruzada de grupos de GDP e de Renda usando `table()` com ambas variáveis:

```

1 table(joined$GDP.Group, joined$Income.Group)

```

```

1 High: nonOECD High: OECD Low Lower middle Upper middle
2 (40,4.25e+03] 2 0 11 15 9
3 (4.25e+03,1.6e+04] 4 1 16 9 8
4 (1.6e+04,5.08e+04] 8 1 9 11 8
5 (5.08e+04,2.63e+05] 5 10 1 13 9
6 (2.63e+05,1.62e+07] 4 18 0 5 11

```

Note que cada linha (grupo de GDP) possui 37 ou 38 países, já que dividimos usando quintis. Para a resposta do item, precisamos verificar as colunas com renda alta (*High*) e na segunda linha (2º grupo com menor GDP – entre 1º e 2º quintis).

□

Exercício 4. Considere a base de dados *mtcars* (nativa no R) que contém o consumo de combustível e mais 10 características automobilísticas para 32 carros (modelos 1973-74). Em particular, queremos analisar a relação entre o consumo de combustível (*mpg*, em milhas por galão) e o peso do automóvel (*wt*, em mil libras) pelo seguinte modelo:

$$mpg_i = \alpha + \beta wt_i + \varepsilon_i$$

- Assuma $\hat{\alpha} = 36$ e $\hat{\beta} = -4$ e calcule a soma dos quadrados dos resíduos $\sum_{i=1}^N \hat{\varepsilon}_i^2$, tal que um resíduo é dado por: $\hat{\varepsilon}_i = mpg_i - \widehat{mpg}_i$.
- Crie uma função `resid_quad(alpha, beta)` que recebe como inputs possíveis valores de $\hat{\alpha}$ e $\hat{\beta}$, e retorna a soma dos quadrados dos resíduos do modelo acima a partir da base de dados *mtcars*.
- Considere os seguintes vetores de possíveis valores de $\hat{\alpha}$ e de $\hat{\beta}$:
`alpha_grid = seq(34, 38, length=11)` e `beta_grid = seq(-6, -2, length=11)`
 Crie uma matriz de dimensão 11×11 em que cada linha corresponde a um valor de $\hat{\alpha}$ e cada coluna corresponde a um valor de $\hat{\beta}$. Preencha essa matriz usando a função `resid_quad()` e verifique qual par $(\hat{\alpha}, \hat{\beta})$ minimiza a soma dos quadrados dos resíduos do modelo acima.

Resposta:

- Encontramos os valores ajustados usando $\widehat{mpg}_i = \hat{\alpha} - \hat{\beta}wt_i = 36 - 4wt_i$:

```
1 mpg_ajustado = (36 + -4 * mtcars$wt)
```

Os resíduos quadráticos são calculados pela diferença entre os valores ajustados com os valores observados de *mpg*, elevando ao quadrado.

```
1 resid = mtcars$mpg - mpg_ajustado
2 sum(resid^2) # soma dos quadrados dos resíduos
```

```
1 [1] 627.7735
```

- A função segue do código feito em (a), mas com input de um `alpha` e um `beta`:

```
1 resid_quad = function(alpha, beta) {
2   mpg_ajustado = alpha + beta * mtcars$wt
3   resid = mtcars$mpg - mpg_ajustado
4   sum(resid^2)
5 }
6
7 resid_quad(36, -4) # Testando para os valores do item (a)
```

```
1 [1] 627.7735
```

- Aqui, vamos criar uma matrix 11×11 em que cada linha corresponde a um valor de `alpha_grid` e cada coluna a um valor de `beta_grid`. Aplicaremos a função criada `resid_quad()` para preencher a matriz, considerando cada combinação de valores dos grids:

```

1 # Definindo os possíveis valores de alpha e beta
2 alpha_grid = seq(34, 38, length=11)
3 beta_grid = seq(-6, -2, length=11)
4
5 # Criando matriz de zeros com dimensão 11 x 11
6 matriz = matrix(0, nrow=length(alpha_grid), ncol=length(beta_grid))
7
8 # Preenchendo a matriz com soma dos quadrados dos resíduos , dadas todas possí
   veis combinações de valores de alpha e de beta
9 for (i in 1:length(alpha_grid)) {
10   for (j in 1:length(beta_grid)) {
11     matriz[i, j] = resid_quad(alpha_grid[i], beta_grid[j])
12   }
13 }
14
15 round(matriz, 1) # Visualizando matriz (com 1 casa decimal)
16 min(matriz) # Menor soma de resíduos quadráticos

```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]	[,10]	[,11]
[1,]	1222.2	820.1	533.5	362.4	306.7	366.6	542.0	832.8	1239.1	1760.9	2398.3
[2,]	1089.2	720.1	466.4	328.2	305.5	398.4	606.7	930.4	1369.7	1924.5	2594.7
[3,]	966.5	630.3	409.6	304.3	314.6	440.4	681.6	1038.3	1510.5	2098.3	2801.4
[4,]	854.0	550.7	363.0	290.7	333.9	492.6	766.8	1156.4	1661.6	2282.3	3018.4
[5,]	751.7	481.4	326.6	287.3	363.4	555.1	862.2	1284.8	1822.9	2476.5	3245.6
[6,]	659.7	422.4	300.5	294.1	403.2	627.8	967.8	1423.4	1994.5	2681.0	3483.0
[7,]	577.9	373.5	284.6	311.2	453.2	710.7	1083.7	1572.3	2176.3	2895.7	3730.7
[8,]	506.4	334.9	279.0	338.5	513.4	803.9	1209.9	1731.3	2368.3	3120.7	3988.6
[9,]	445.1	306.6	283.6	376.0	583.9	907.4	1346.3	1900.7	2570.5	3355.9	4256.8
[10,]	394.1	288.5	298.4	423.8	664.7	1021.0	1492.9	2080.2	2783.1	3601.4	4535.2
[11,]	353.3	280.6	323.5	481.8	755.6	1144.9	1649.7	2270.0	3005.8	3857.1	4823.8

```

13
14 [1] 278.9546

```

Porém, não queremos saber o menor valor de soma dos quadrados dos resíduos, mas sim os $\hat{\alpha}$ e $\hat{\beta}$ que geraram esse valor. Para isto, vamos usar a função `which(..., arr.ind=TRUE)` (argumento `arr.ind=TRUE` faz retornar os índices de cada dimensão da matriz/array):

```

1 # Índices de alpha e de beta que minimizam desvio quadrático
2 indices = which(matriz == min(matriz), arr.ind = TRUE)
3 indices

```

```

1      row col
2 [1,]    8   3

```

Agora, vamos extrair os índices de alpha e de beta do objeto `indices`, que serão utilizados para retornar os seus valores:

```

1 # Atribuindo os índices a objetos
2 i_alpha = indices[1, "row"]
3 i_beta = indices[1, "col"]
4
5 # Buscando valores de alpha e de beta a partir dos índices acima
6 alpha_grid[i_alpha]
7 beta_grid[i_beta]

```

```

1 [1] 36.8
2 [1] -5.2

```

Logo, considerando os valores disponíveis nos grids, a combinação $\hat{\alpha} = 36.8$ e $\hat{\beta} = -5.2$ geraram a menor soma dos quadrados dos resíduos.

□

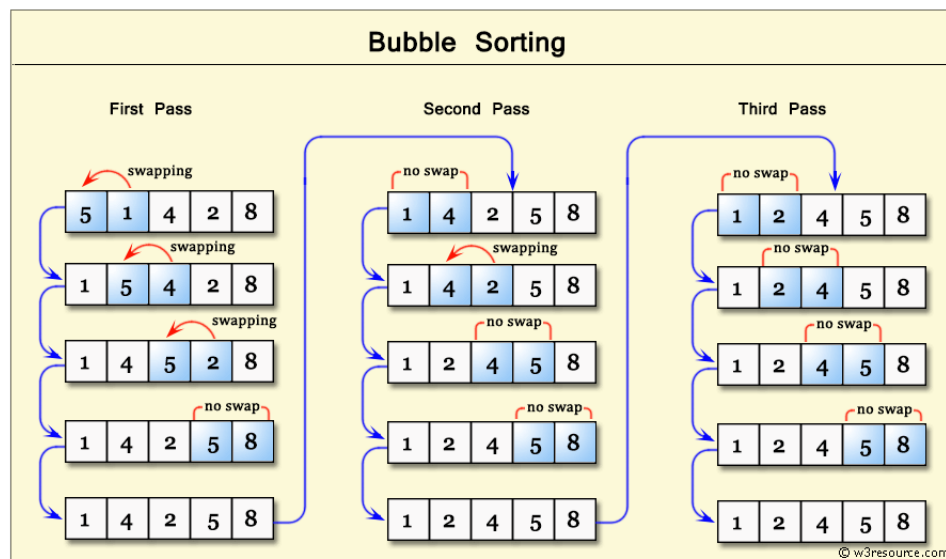
Exercício 5. Faça uma amostragem de 100 números sem reposição do vetor de números inteiros 1:100. Usando estruturas condicionais e de repetição/loop, reordene o vetor de forma decrescente.

a) Crie um algoritmo usando os seguintes passos:

1. Compare o número da 1^a posição com o da 2^a e, caso este seja maior, troque a ordem.
2. Compare o número da 2^a posição com o da 3^a e, caso este seja maior, troque a ordem.
- ⋮
99. Compare o número da 99^a posição com o da 100^a e, caso este seja maior, troque a ordem.

b) Os passos executados no item anterior foram suficientes? Adapte seu algoritmo para que o vetor esteja totalmente ordenado decrescentemente.

Resposta: O procedimento descrito no enunciado é o algoritmo de ordenação de bolha:



Primeiro, crie o vetor descrito no enunciado:

```
1 vetor = sample(1:100, 100)
2 print(vetor)
```

```
1  [1]  80   9  20  95  24  43  65  72  29  88  47  71  56  69  37  54  79
2  [18] 16   3  84  67  12  28  83   6  50  25  86  70  30  22 100  36  61
3  [35] 51  45  10  98  14  26  89  55  48  21  44  66  64  94  15  49  40
4  [52] 96  97   8   5  53  68  35  18  34  52  32  62   7  60  58  93  87
5  [69] 38  77  82  46  23   4  27  31  11  19  99  13  91   2   1  81  57
6  [86] 85  17  63  41  76  59  74  42  90  39  78  92  73  33  75
```

Iniciando no 1^o elemento do vetor, faremos uma comparação entre este elemento e o seguinte. Se este for maior do que aquele, inverteremos a ordem e, caso o elemento seja seguido por um número menor, a ordem continuará. Logo, faremos a mesma análise para a próxima dupla do vetor e repetiremos até chegar ao penúltimo elemento do vetor, que será comparado ao último.

```

1 for (i in 1:(length(vetor) - 1)) {
2     if (vetor[i] < vetor[i + 1]) {
3         aux = vetor[i + 1]
4         vetor[i + 1] = vetor[i]
5         vetor[i] = aux
6     }
7 }

```

1	[1]	80	20	95	24	43	65	72	29	88	47	71	56	69	37	54	79	16
2	[18]	9	84	67	12	28	83	6	50	25	86	70	30	22	100	36	61	51
3	[35]	45	10	98	14	26	89	55	48	21	44	66	64	94	15	49	40	96
4	[52]	97	8	5	53	68	35	18	34	52	32	62	7	60	58	93	87	38
5	[69]	77	82	46	23	4	27	31	11	19	99	13	91	3	2	81	57	85
6	[86]	17	63	41	76	59	74	42	90	39	78	92	73	33	75	1		

Note que este loop fez apenas o menor número ir à última posição do vetor. Precisamos rodar outros loops para levar o 2^o menor número para a penúltima posição, e assim por diante.

```

1 for (num_loops in 0:99) {
2     for (i in 1:99) {
3         if (vetor[i] < vetor[i + 1]) {
4             aux = vetor[i + 1]
5             vetor[i + 1] = vetor[i]
6             vetor[i] = aux
7         }
8     }
9 }

```

1	[1]	100	99	98	97	96	95	94	93	92	91	90	89	88	87	86	85	84
2	[18]	83	82	81	80	79	78	77	76	75	74	73	72	71	70	69	68	67
3	[35]	66	65	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50
4	[52]	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
5	[69]	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
6	[86]	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1		

[Opcional] Outra possibilidade (mais eficiente), é fazer o 2^o loop a ir até o antepenúltimo elemento (comparando com o penúltimo), e assim por diante. Isso evita fazer comparações com elementos que você já tem certeza que estão na ordem certa:

```

1 for (num_loops in 0:98) {
2     for (i in 1:(length(vetor) - 1 - num_loops)) {
3         if (vetor[i] < vetor[i + 1]) {
4             aux = vetor[i + 1]
5             vetor[i + 1] = vetor[i]
6             vetor[i] = aux
7         }
8     }
9 }

```


1	[1]	100	99	98	97	96	95	94	93	92	91	90	89	88	87	86	85	84
2	[18]	83	82	81	80	79	78	77	76	75	74	73	72	71	70	69	68	67
3	[35]	66	65	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50
4	[52]	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
5	[69]	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
6	[86]	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1		

□