capitulo (3) estrutura de dados

# Estruturas de Dados

Neste capítulo vamos apresentar ao leitor os tipos de dados e estruturas de dados fundamentais do R. Nota-se que não são apresentados todos os tipos e estruturas de dados, entretanto garantimos que os aqui apresentados serão o suficiente para resolver quase todos (se não todos) os problemas e aplicações que o leitor se deparar com. Também ressalvamos que talvez esse capítulo seja o mais importante desta parte do livro, pois um bom entendimento sobre os tipos de informações e como elas são armazenadas (dentro do R) é essencial para o desenvolvimento de habilidades para lidar com dados. Ressaltamos que o objetivo deste capítulo é fazer com o que leitor saiba "quais são e como são"" as estruturas de dados, pois as ferramentas de "como trabalhar" com as estruturas de dados serão apresentadas ao longo do livro.

Sem mais delongas os tipo de dados são como uma determinada informação pode ser registrada e já as estruturas de dados são como as informações são armazenadas de modo que possam ser utilizadas, um pouco abstrata essa explicação até agora não acham? Bem, uma analogia que pode ajudar a compreender a **diferença entre tipos de dados e estruturas de dados** é a seguinte: Acredito que seja familiar a todos o final do filme Toy Story (o primeiro da trilogia), quando o Woody e o Buzz perdem o caminhão da mudança contratado pela Mãe do Andy (digo mãe e não família pois o Pai do Andy é um mistério). Bem se recordarmos bem alguns brinquedos o Andy trouxe com ele dentro do carro (em especial teria trazido o Woody e o Buzz) e ao mesmo tempo outro brinquedos estavam sendo transportados no caminhão. Onde eu quero chegar com isso? Os brinquedos seriam um tipo de informação (no caso de uma mudança poderíamos pensar que as roupas seriam outro tipo de informação) e o caminhão da transportadora e o carro da mão do Andy seriam as diferentes estruturas nas quais é possível carregar os dados.

Com a diferença entre tipos e estruturas de dados explicada vamos agora apresentar ao leitor explicitamente o funcionamento de tais coisas no R. Começamos apresentando os tipos de dados e em seguida as estrutura.

## Tipos de dados

Para apresentar os diferentes tipos de como as informações podem ser armazenadas vamos utilizar uma base fictícia de empregados apresentada na tabela @ref(tab:empregados). Nela temos 6 colunas representando diferentes variáveis.

Empregados

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nome | Tempo\_emprego | Setor | Peso | Nascimento | Casado |
| Mari | 1 | Estrutura | 35.45 | 1999-11-17 | FALSE |
| Shaakir | 2 | Financeiro | 39.77 | 1999-10-20 | TRUE |
| Brehauna | 3 | Financeiro | 35.77 | 1999-07-30 | TRUE |
| John | 4 | Estrutura | 37.31 | 1999-03-25 | TRUE |
| Hector | 5 | Financeiro | 37.24 | 1999-05-24 | FALSE |
| Zachery | 6 | Marketing | 41.80 | 1999-07-13 | TRUE |
| Itzel | 7 | Financeiro | 40.33 | 1999-01-01 | FALSE |
| Jazmine | 8 | Financeiro | 44.51 | 1999-07-05 | TRUE |
| Lynn | 9 | Estrutura | 46.07 | 1999-03-05 | TRUE |
| Dominque | 10 | Financeiro | 39.78 | 1999-06-01 | FALSE |

Cada coluna possui um tipo diferente de informação e cada informação é armazenada de um jeito diferente. A lista abaixo sintetiza os tipos de dados e a coluna da tabela com a qual ela está relacionada[[1]](#footnote-23).

* Character (ou string): São caracteres textuais que podem assumir qualquer valor. Na tabela é a coluna relaciona ao Nome. Os elementos dos vetores de caracteres devem estar entre aspas "elemento";
* Integer: São números inteiros, neste caso relacionado com a coluna idade;
* Double: São números decimais, neste caso relacionado à coluna de peso;
* Factor: São caracteres textuais mais podem assumir somente um determinado tipo de valor. Neste caso seria a coluna que traz o setor no qual o empregado trabalha. Outro exemplo de factor seria o tipo sanguíneo em um banco de dados de pacientes;
* Dates: São valores textuais que são interpretados como datas, neste caso relacionado à coluna nascimento;
* Logical: Assumem o valor TRUE ou FALSE, também chamados de valores booleanos.

A função str() permite verificar a estrutura dos nossos objetos, o chunk abaixo mostra a utilização da função na nossa base de dados de empregados

> str(empregados)  
# 'data.frame': 10 obs. of 6 variables:  
# $ Nome : chr "Mari" "Shaakir" "Brehauna" "John" ...  
# $ Tempo\_emprego: int 1 2 3 4 5 6 7 8 9 10  
# $ Setor : Factor w/ 3 levels "Estrutura","Financeiro",..: 1 2 2 1 2 3 2 2 1 2  
# $ Peso : num 35.5 39.8 35.8 37.3 37.2 ...  
# $ Nascimento : Date, format: "1999-11-17" "1999-10-20" ...  
# $ Casado : logi FALSE TRUE TRUE TRUE FALSE TRUE ...

Seguindo na nossa analogia da mudança as estruturas de dados seriam os meios através dos quais podemos transportar os objetos de uma casa para outro, como carros, caminhões etc. Deste modo, nesta seção vamos apresenta ao leitor o que são e exemplos de como construir: (i) Vetores, (ii) Matrizes, (iii) Data frames e (iv) Listas

## Vetores

É a estrutura mais básica (e fundamental) do R. De maneira objetiva um **vetor é uma sequência de elementos do mesmo tipo**. Agora vamos introduzir aqui 4 aspectos importantes sobre vetores.

### Criando Vetores

Há diversas maneiras de se criar vetores e a sua imaginação é de fato o limite, porém há jeitos mais triviais que são muito utilizados. Dois destes triviais são através da função concatenar, c(), e a função sequência, seq(). O chunk abaixo ilustra maneiras diferentes de se criar a sequência 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

> c(1,2,3,4,5,6,7,8,9,10)  
# [1] 1 2 3 4 5 6 7 8 9 10  
> c(1:10)  
# [1] 1 2 3 4 5 6 7 8 9 10  
> seq(from = 1, to = 10, by=1)  
# [1] 1 2 3 4 5 6 7 8 9 10  
> seq(from = 1, length.out = 10,by = 1)  
# [1] 1 2 3 4 5 6 7 8 9 10

Quando quisermos salvar um determinado vetor em um objeto x, basta utilizarmos o operador de atribuição <- da seguinte maneira x <- c(1:10).

A lógica é a mesma para criar vetores com outros tipos de dados, como por exemplo criar um vetor contendo as primeiras letras do alfabeto e outro com os primeiros dias de 2017.

> letras <- c("a","b","c","d")  
> letras  
# [1] "a" "b" "c" "d"  
> dias <- seq(from = as.Date("01/01/2017",format="%d/%m/%Y"),length.out = 5,by = "day")  
> dias  
# [1] "2017-01-01" "2017-01-02" "2017-01-03" "2017-01-04" "2017-01-05"

Não se preocupe caso não tenha ficado claro a sintaxe do vetor dias, mais para frente vamos discutir em maior detalhe datas[[2]](#footnote-26)

### Acessando elementos

Acessar elementos específicos de vetores é algo muito útil (acreditem no que eu digo) e fácil, basta utilizar x[i], onde x é um vetor e i o i-ésimo elemento[[3]](#footnote-28). No chunk abaixo segue mostro alguns exemplos de como acessar elementos dos vetores

> x <- c(1:10) # criando um vetor x  
> x[7:9] # pegando elementos de 7 a 9  
# [1] 7 8 9  
> x[length(x)] <- 0 # substituindo o último elemento por 0  
> y <- x[1:4]

A última linha do chunk acima crian um novo vetor y com os 4 primeiros valores do vetor x, ou seja, mais um jeito de se criar vetores.  
Mais adiante no livro[[4]](#footnote-29) mostraremos que os filtros utilizados nos dados (como por exemplo selecionar de um banco de dados apenas os pacientes com pressão arterial acima de 140 X 90 mmHg) nada mais é do que acessar elementos do vetores que satisfaçam de terminadas condições.

### Operações Vetorizadas

Dizer que as operações são vetorizadas é apenas dizer que ao aplicar uma funções em um vetor essa função será aplicada em cada um dos elementos do vetor. Daí inclusive a importância de internamente o R não aceitar vetores com tipos de informação diferentes, pois não faria sentido, por exemplo, efetuar a multiplicação em um vetor que contem letras!! A ideia é bastante simples e intuitiva, porém não custa nada visualizar alguns exemplos.

> a <- c(1,4,9,16,25,36,49,64,81)  
> sqrt(a) # sqrt é a raiz quadrada  
# [1] 1 2 3 4 5 6 7 8 9  
> a+1  
# [1] 2 5 10 17 26 37 50 65 82

Perceba que cada vez que uma função é aplicada no vetor ela na verdade é aplicada em cada elemento que compõe o vetor. O último aspecto importante para se introduzir sobre vetores é a reciclagem.

### Reciglagem

O que você imagina que acontece quando a operação c(1,2)+c(1,1,1,1) é efetuada? O nome da subseção já da uma boa dica. O que acontece é que o vetor de menor comprimento é reciclado, isto é, reutilizado, para que a operação seja efetuada e resultando no seguinte vetor c(2,3,2,3). Importante salientar que isso ocorre quando comprimento do maior vetor é múltiplo do comprimento do vetor menor, caso não seja este caso será retornado um erro. O chunk abaixo explicita essas duas situações

> c(1,1)+c(1,2,1,2)  
# [1] 2 3 2 3  
> c(1,1)+c(1,2,1,2,1)  
# Warning in c(1, 1) + c(1, 2, 1, 2, 1): longer object length is not a  
# multiple of shorter object length  
# [1] 2 3 2 3 2

## Matrizes

A segunda estutura apresentada é a matriz, que nada mais é do que uma extensão do vetor para uma dimensão a mais. Colocando de outro jeito uma **matriz é uma estrutura com colunas com o mesmo número de linhas e que armazenam o mesmo tipo de informação**. O leitor provavelmente já se deparou em algum momento da vida com uma matriz do seguinte formato

> matrix(data = letters[1:6],ncol=3)  
# [,1] [,2] [,3]  
# [1,] "a" "c" "e"   
# [2,] "b" "d" "f"

Assim como foi feito com vetores, vamos mostrar como criar matrizes do zero, como acessar seu elementos e por fim como discutiremos a título de informação, não se preocupem, sobre operações matemática com matrizes.

### Criando Matrizes

Para criar matriz utilize a função de nome matrix(), que surpresa, na qual são necessários no mínimo dois argumentos. O argumento data para o qual deve ser passado um vetor que irá popular a matriz e o segundo argumento necessário é ou ncolou nrow, especificando claramente o número de colunas ou o número de linhas. Perceba que é importante que o vetor passado para data tenha um comprimento compátivel com o número de linhas ou número de colunas. Abaixo tem o exemplo de como criar uma matriz

> matriz\_simples <- matrix(data = c(1:25),nrow = 5,byrow = TRUE)  
> matriz\_simples  
# [,1] [,2] [,3] [,4] [,5]  
# [1,] 1 2 3 4 5  
# [2,] 6 7 8 9 10  
# [3,] 11 12 13 14 15  
# [4,] 16 17 18 19 20  
# [5,] 21 22 23 24 25

O terceiro argumento byrow recebe um valor lógico, e tem como default FALSE e como o nome diz ele indica se a matriz deve ser preenchida por linhas, byrow=TRUE ou por colunas. Perceba que na primeira matriz apresentada nesta seção não tem o esse argumento e por consequência a matriz foi preenchida por colunas.

### Acessando Elementos

Como a matriz é uma extensão dos vetores a ideia é a mesma, com a única diferença que é necessário diferenciar o indexador de linha e de coluna. E isso é feito de maneira muito simples e direta com o operador [], da seguinte maneira: matriz[i,j] onde i denota as linhas e j denota as colunas. Caso quiséssemos pegar somente os elementos das duas primeiras linhas e das duas últimas colunas da matriz\_simples uma maneira seria

> matriz\_simples[-c(3:5),c(4,5)]  
# [,1] [,2]  
# [1,] 4 5  
# [2,] 9 10

Note que usamos -c(3:5) para indicar que queremos todas as linhas menos as linhas 3, 4, 5. O mesmo poderia ter sido feito para as colunas. Vale destacar uma sutileza importante para compreender a utilização de matrizes no R. Pode haver casos em que desejamos acessar apenas uma coluna (ou linha) específica de uma determinada matriz e acredito que a tentativa mais natural para todos seria algo como matriz\_simples[,3], mas ao executar esse comando no console teríamos como resultado

> matriz\_simples[,3]  
# [1] 3 8 13 18 23

Um vetor! E neste caso teríamos que nos perguntar: Queremos que o resultado da nossa operação (selecionar uma ou coluna especifica) seja um vetor ou uma matriz coluna (ou linha)? Caso a resposta seja um vetor deve ser executado o comando como acima, porém caso não seja esse o caso adicionamos o argumento drop=FALSE fazendo com que a estrutura de matriz seja preservada.

> matriz\_simples[,3, drop = F]  
# [,1]  
# [1,] 3  
# [2,] 8  
# [3,] 13  
# [4,] 18  
# [5,] 23

Por fim, note que o resultado de não selecionar nenhuma coluna é selecionar todas as colunas, e o mesmo se aplica para colunas. E neste caso as operações que efetuarmos nas matrizes coluna ou linha serão automaticamente vetorizadas.

### Operações com matrizes

Vamos aqui mostrar a título de informação algumas operações matemáticas com matrizes. Digo a título de informação, pois constatei na minha experiência que a utilização dessas operações é rara, porém acredito que seja importante saber da existência.

Antes de mostarmos os procedimentos vamos criar duas matrizes para trabalharmos com elas, vamos criativamente chama-las matrix\_1 e matrix\_2.

> matrix\_1 <- matrix(data = 1:6,ncol = 3)  
> matrix\_2 <- matrix(data = 1:6, ncol = 2)

#### Multiplicação e Soma

De maneira geral podemos nos deparar com dois casos: (i) queremos somar/multiplicar um número com/por uma matriz ou (ii) queremos multiplicar duas matrizes.

Quando nos encontramos no primeiro (somar/multiplicar um número por um matriz) caso basta usar os operadores apresentados na seção **colocar aqui a referencia para a seção R como calculadora** e a operação é aplicada em cada elemento da matrix, como feito no chunk que segue

> 3+matrix\_1  
# [,1] [,2] [,3]  
# [1,] 4 6 8  
# [2,] 5 7 9  
> 3\*matrix\_2  
# [,1] [,2]  
# [1,] 3 12  
# [2,] 6 15  
# [3,] 9 18  
> sqrt(matrix\_1)  
# [,1] [,2] [,3]  
# [1,] 1.000000 1.732051 2.236068  
# [2,] 1.414214 2.000000 2.449490

Já no segundo caso (multiplicar matrizes) temos que colocar o operador \* entre %s para que o R entenda que queremos efetuar operações matriciais

> matrix\_1 %\*% matrix\_2  
# [,1] [,2]  
# [1,] 22 49  
# [2,] 28 64

É claro que as matrizes devem ter dimensões vpalidas para que a multiplicação seja efetuada.

#### Determinante, Transposta e Inversa

Brevemente vamos apresentar as funções para obtermos o determinante, a transposta e a inversa de uma matriz. Começamos com o determinante que é calculado através da função det():

> matriz <- matrix(c(1:4),ncol=2,byrow = T)  
> det(matriz)  
# [1] -2

A transposta por sua vez é calculada através da função t():

> t(matriz)  
# [,1] [,2]  
# [1,] 1 3  
# [2,] 2 4

E por fim a inversa é obtida através da função solve():

> solve(matriz)  
# [,1] [,2]  
# [1,] -2.0 1.0  
# [2,] 1.5 -0.5

Passamos agora para a estrutura chamda de *data frame*, que funciona com os principios de vetores e matrizes, porém com algumas modificações que a tornam muito útil na prática.

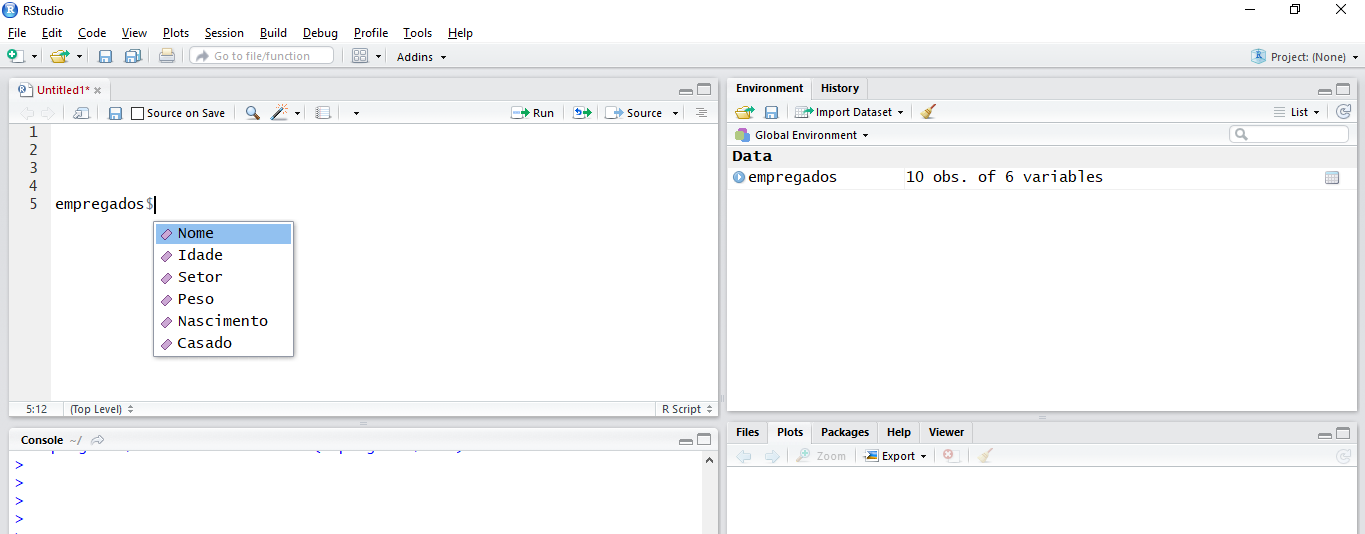
## Data Frames

Os data frames possuem a mesma estrutura que uma matriz, isto é, colunas com o mesmo número de linhas. Porém, há duas vantagens de se trabalhar com data frames[[5]](#footnote-39): (i) Cada coluna pode contar tipos de informações diferentes e (ii) nomeando as colunas podemos acessar colunas específicas sem precisar saber exatamente a indexação da coluna. Essas duas diferenças tornam os data frames o jeito mais comum de se trabalhar com bases de dados no R. Pois podemos ter diversas variáveis, relacionadas a mesma observação, com tipos de informação diferente em uma mesma estrutura! O leitor atento percebeu que a tabela @ref(tab:empregados) é na verdade um exemplo de data frame, onde cada coluna representa uma variável com tipo diferente de informação.

Para acessar colunas pelo nome utiliza-se o $ logo em seguida do nome do data frame. Caso quissésemos acessar só os nomes do nosso data frame de empregados poderíamos fazer da seguinte maneira:

> empregados$Nome  
# [1] "Mari" "Shaakir" "Brehauna" "John" "Hector" "Zachery"   
# [7] "Itzel" "Jazmine" "Lynn" "Dominque"

O RStudio possui uma ferramenta conveniente para nos ajudar, ao digitar empregados$ e em seguida pressionar TAB o RStudio mostrará um menu *dropdown* com os nomes das colunas do data frame, como mostra a figura @ref(fig:rstudio-dropdown-menu), de modo que não é necessário sabermos qual o número da coluna em que se encontra a variável de interesse e nem ter que saber de cabeça o nome de todas as variáveis disponíveis no data frame.



Menu Dropdown do RStudio

Outra utilidade do $ quando se tratando de data frames é adicionar novas variáveis ao data frame. Ainda no nosso exemplo do data frame com informações sobre os empregados, poderíamos pensar que foi oferecido aos empregados um treinamento no qual a participação é optativa e depois que o treinamento foi realizado gostaríamos de adicionar uma coluna indicando quais empregados optaram por participar do treinamento. Para facilitar o entendimento vamos supor que os 5 primeiros empregados () optaram por participar do treinamento e os últimos 5 por não participar. Desse modo queremos adicionar uma coluna com o nome de Treinamento que receberá o valor sim caso o empregado tenha participado do treinamento e não caso ele não tenha participado. Isso é feito da seguinte maneira:

> empregados$Treinamento <- c(rep("sim",5),rep("não",5))

A sintaxe c(rep("sim",5),rep("não",5)) cria um vetor repetindo o termo sim cinco vezes e depois repetindo o termo não também cinco vezes. Agora se verificarmos nosso data frame veremos que a coluna Treinamento adicionada com sucesso.

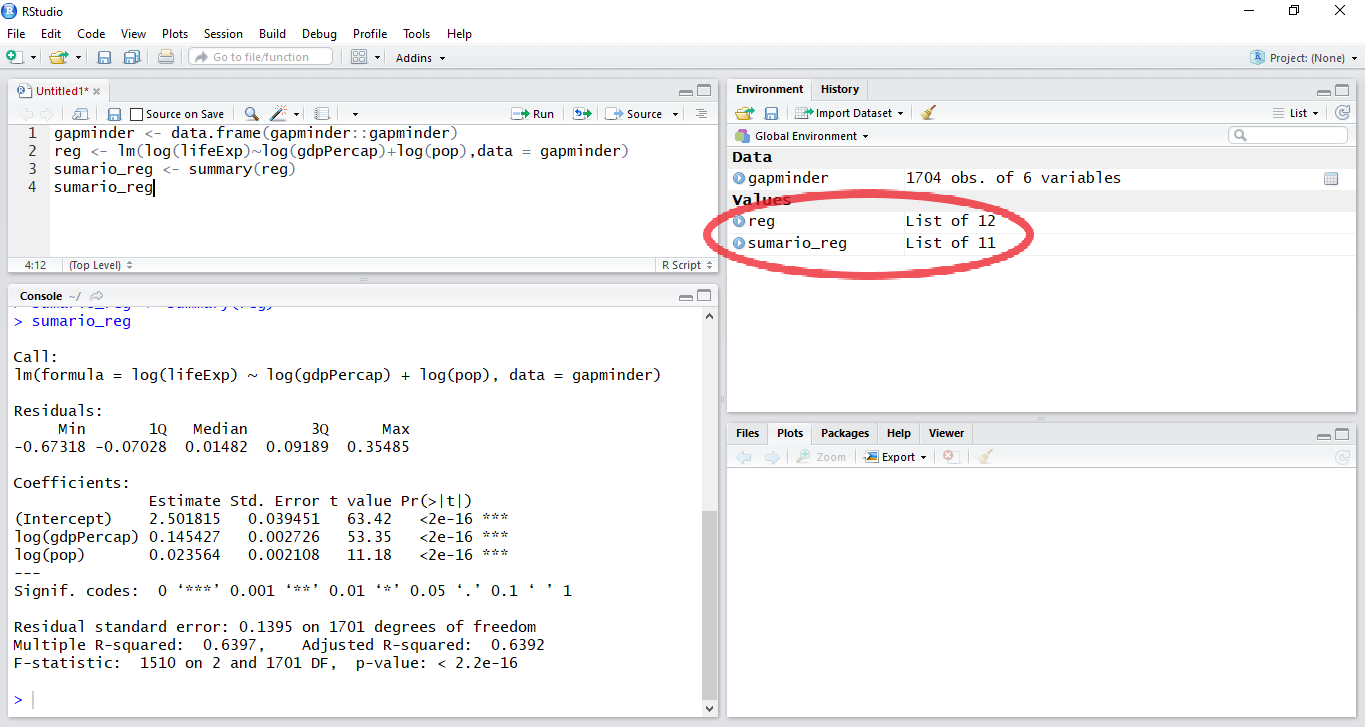
> str(empregados)  
# 'data.frame': 10 obs. of 7 variables:  
# $ Nome : chr "Mari" "Shaakir" "Brehauna" "John" ...  
# $ Tempo\_emprego: int 1 2 3 4 5 6 7 8 9 10  
# $ Setor : Factor w/ 3 levels "Estrutura","Financeiro",..: 1 2 2 1 2 3 2 2 1 2  
# $ Peso : num 35.5 39.8 35.8 37.3 37.2 ...  
# $ Nascimento : Date, format: "1999-11-17" "1999-10-20" ...  
# $ Casado : logi FALSE TRUE TRUE TRUE FALSE TRUE ...  
# $ Treinamento : chr "sim" "sim" "sim" "sim" ...

Destaco aqui que não mostro como criar data frame do zero, pois como mencionado no início desta seção, os data frames são muito utilizados para armazenar dados que serão importados para dentro do R **(colocar aqui o capitulo que falará sobre importar dados para o R)** e não digitados no console ou script.

## Listas

A última estrutura de dados é chamada de lista (ou vetor recursivo). De maneira objetiva a lista é um vetor no qual seus elementos podem ser qualquer outro tipo de estrutura, isto quer dizer que podemos ter uma lista que contém vetor, matrizes, data frames, listas e etc. Perceba que ao passo que o data frame tem colunas com o mesmo número de linhas as listas tem elementos que não só podem ter número de linhas diferentes como estruturas de dados diferentes! Caso a explicação até agora esteja um pouco abstrata, não se preocupe pois antes de entrarmos na discussão mesmo sobre listas vamos apresentar um exemplo de como as listas são realmente úteis e versáteis quando utilizamos o R.

O objetivo desse exemplo é mostrar uma aplicação de como as listas são usadas dentro R, faremos isso mostrando que ao efetuar uma regressão no R os resultados são armazenados em uma lista. Para tal vamos utilizar o dataset *gapminder* (disponível no pacote gapminder) que é um excerto sobre dados de expectativa de vida, PIB per capita e população por país. Vamos construir um modelo do tipo log-log, para avaliar os efeitos em termos percentuais, onde a expectativa de vida depende do PIB percapita e do tamanho da população.

A figura @ref(fig:exemplo-versatilidade-lista) traz o resultado da estimação da regressão. Perceba que a regressão, salva com o nome de reg, e o sumário da regressão, salvo como sumario\_reg, são listas com 12 e 11 elementos respectivamente. E ao executar o sumário da regressão (linha 4) temos o resultado completo do sumário armazenado na lista. (Obs: não se preocupe em entender o código da figura, mais para frente **colocar aqui a seção de sregressão** iremos discutir em detalhes como fazer regressão no R). ``` 

Espero que esse exemplo tenha ilustrado como as listas são versáteis e tenha facilitado o entendimento das mesmas, pois tenho certeza que o leitor irá se deparar em diversas situações com listas.

### Criando listas

Para criar uma lista utiliza-se função list() onde os argumentos são os nomes dos elementos e os elementos em si. Uma lista simples pode ser criada da seguinte maneira:

> lista <- list(nomes = empregados$Nome,  
+ data\_frame = empregados[1:4,-1],  
+ lista = list(numeros=1:10,aleatorio=rnorm(4)))

Veja que é possível ter listas como elementos de uma lista (por isso listas são também chamadas de vetores recursivos). Para visualizar a lista no console podemos executar o nome da listano conseole, neste caso lista, nos dando o seguinte resultado:

> lista  
# $nomes  
# [1] "Mari" "Shaakir" "Brehauna" "John" "Hector" "Zachery"   
# [7] "Itzel" "Jazmine" "Lynn" "Dominque"  
#   
# $data\_frame  
# Tempo\_emprego Setor Peso Nascimento Casado Treinamento  
# 1 1 Estrutura 35.45460 1999-11-17 FALSE sim  
# 2 2 Financeiro 39.77155 1999-10-20 TRUE sim  
# 3 3 Financeiro 35.76569 1999-07-30 TRUE sim  
# 4 4 Estrutura 37.31283 1999-03-25 TRUE sim  
#   
# $lista  
# $lista$numeros  
# [1] 1 2 3 4 5 6 7 8 9 10  
#   
# $lista$aleatorio  
# [1] 0.01189748 -1.09948991 0.75981426 0.55666501

De modo que conseguimos visualizar e entender o que está dentro da nossa lista, porém em casos reais as listas são maiores e com objetos mais complexos e a melhor opção para entender a lista em questão é utilizar a função str(), que nos da o seguinte resultado

> str(lista)  
# List of 3  
# $ nomes : chr [1:10] "Mari" "Shaakir" "Brehauna" "John" ...  
# $ data\_frame:'data.frame': 4 obs. of 6 variables:  
# ..$ Tempo\_emprego: int [1:4] 1 2 3 4  
# ..$ Setor : Factor w/ 3 levels "Estrutura","Financeiro",..: 1 2 2 1  
# ..$ Peso : num [1:4] 35.5 39.8 35.8 37.3  
# ..$ Nascimento : Date[1:4], format: "1999-11-17" ...  
# ..$ Casado : logi [1:4] FALSE TRUE TRUE TRUE  
# ..$ Treinamento : chr [1:4] "sim" "sim" "sim" "sim"  
# $ lista :List of 2  
# ..$ numeros : int [1:10] 1 2 3 4 5 6 7 8 9 10  
# ..$ aleatorio: num [1:4] 0.0119 -1.0995 0.7598 0.5567

O output retornado no console da função str() nos mostra que o nosso objeto lista é uma lista de 3 elementos. Onde o primeiro elemento é o vetor nomes, o segundo elemento é um data frame com nome de data\_framee o terceiro elemento é uma lista com dois elementos chamada de lista.

### Acessando elementos das listas

De maneira geral há duas maneiras de acessar os elemtos de uma lista: (i) pelo nome dos elementos; e (ii) através da indexação. Para acessar pelo nome basta utilizar o $ como feito com data frames, note que o menu dropdown funciona aqui da mesma forma.

A outra maneira de acessar elementos de uma lista á através da indexação, que neste caso devemos ter atenção a um sútil detalhe. De maneira similar as outras estruturas de dados usamos o nome\_da\_lista[i] para acessar o i-ésimo elemento da lista, mas também podemos acessar o mesmo elemento utilizando [[]], isso mesmo chaves duplas.

> x <- lista[1]  
> y <- lista[[1]]

Essa diferença pode parecer inofensiva de início, porém ao observar qual é a estrutura de x e de y nos revela a diferença.

> str(x)  
# List of 1  
# $ nomes: chr [1:10] "Mari" "Shaakir" "Brehauna" "John" ...  
> str(y)  
# chr [1:10] "Mari" "Shaakir" "Brehauna" "John" "Hector" ...

Vemos que x é uma lista com o vetor nomes sendo o único elemnto e que y é o vetor nomes. Ou seja, utilizar [] (chaves simples) retorna uma lista ao passo que utilizar [[]] (chaves duplas) retorna a estrutura de dados no elemento que está sendo acessado!

## resumo (melhorar nome)

Para sintetizar o que foi apresentado neste capítulo a lista abaixo mostra uma ideia geral de para que são utilizadas na maior parte das vezes cada uma das estruturas de dados no R:

* vetor = usados salvar uma variável com elementos do mesmo tipo de dado;
* matriz = usadadas para armazenar mais de uma variável do mesmo tipo de dado;
* data frame = usados para armazenar mais de uma variável, podendo serem de tipos de dados diferentes (banco de dados);
* listas = usadas para salvar resultador de estimações valores retornados de função etc.

1. Na estatística existem basicamente dois tipos de variáveis cada uma delas com dois subtipos. A primeira é chamada de variáveis qualitativas, que podem assumir valores nominais, por exemplo Nomes, ou assumir valores ordinais, por exemplo dividir peso pena, peso médio etc. O segundo tipo de variável é chamado de variáveis quantitativas, que podem ser intervalos, por exemplo a previsão do tempo quando nos traz temperatura miníma e máxima, ou assumir valore pontuais, como por exemplo a medição do pesos das pessoas [↑](#footnote-ref-23)
2. **colocar aqui depois a seção na qual isso é discutido** [↑](#footnote-ref-26)
3. Ressaltamos para aqueles que são familiares com outro linguagem de programação, como C, a indexação dos vetores no R começa em 1 e não em 0. [↑](#footnote-ref-28)
4. **colocar aqui no final as seções corretas** [↑](#footnote-ref-29)
5. Para aqueles que conhecem um pouco de computação a única desvantagem do data frame em relação a matriz é que as operação feitas com data frames são significativamente mais lentas do que quando feitas com matrizes, isso ocorre devido ao jeti de como as informações são salvas na mémoria do computador [↑](#footnote-ref-39)