

# Limpando e organizando dados

Renata Oliveira

# Retrospecto

# Modelo conceitual da análise de dados

# Retrospecto

# Modelo conceitual da análise de dados

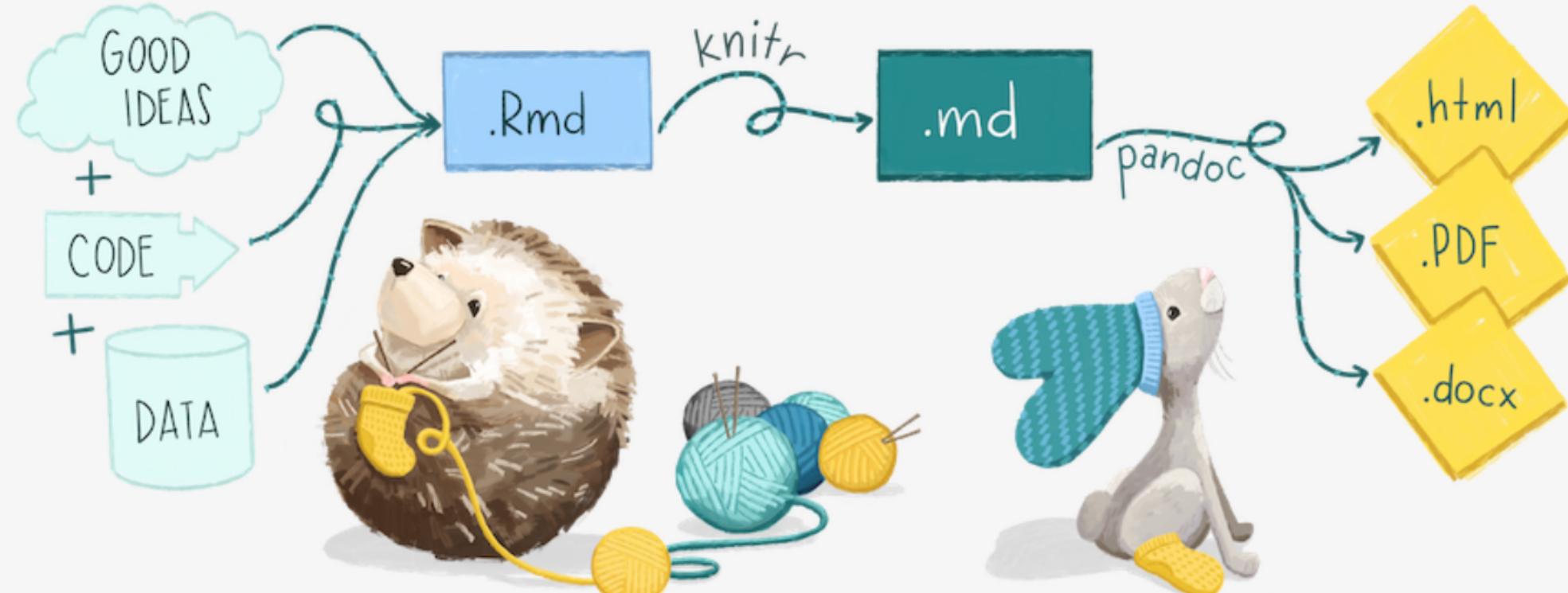
# Tipos de dados

- Os tipos de dados mais comuns são **character**, **numeric**, **factore** **logical**.

# Tipos de classes de objetos no R

- As classes de organização dos dados mais comuns são **vector**, **list**, **matrix** e **dataframe**.

# RMarkdown



Alison Hill **Teaching in Production**

# ggplot2

- ggplot() é a função principal no ggplot2
- As parcelas são construídas em camadas
- A estrutura do código para as parcelas pode ser resumida como

```
ggplot(data = [dataset],  
       mapping = aes(x = [x-variable], y = [y-variable])) +  
       geom_xxx() +  
       other options
```

# Número de variáveis envolvidas

- Análise univariada de dados - distribuição de uma única variável
- Análise de dados bivariados - relação entre duas variáveis
- Análise multivariada de dados - relação entre muitas variáveis ao mesmo tempo, geralmente focalizando a relação entre duas, enquanto condiciona para outras

# Tipos de variáveis

- As variáveis **numéricas** podem ser classificadas como **contínuas** ou **discretas** considerando se a variável pode ou não assumir um número infinito de valores ou apenas números inteiros não negativos, respectivamente.
- Se a variável for **categórica**, podemos determinar se ela é **ordinal** tendo em conta se os níveis têm ou não uma ordenação natural.

# Tidy data

As famílias infelizes são todas iguais; cada família infeliz é infeliz à sua própria maneira.

Leo Tolstoy

## Características dos dados tidy:

- Cada variável forma uma coluna.
- Cada observação forma uma linha.
- Livre de linhas/colunas duplicadas.
- Livre de erros de ortografia
- Relevante (por exemplo, livre de caracteres especiais)
- O tipo de dados apropriado para análise
- Livre de outliers (ou só contém outliers que tenham sido identificados/entendidos), e

## Características dos dados untidy:

!@#\$%^&\*()

# O que faz com que estes dados não estejam arrumados?

Airplanes on Hand in the AAF, By Major Type:  
Jul 1939 to Aug 1945

End of Month	Total	Very Heavy Bombers	Heavy Bombers	Medium Bombers	Light Bombers	Fighters	Reconnaissance	Transports	Trainers	Communications
<b>1939</b>										
Jul	2,402	-	16	400	276	494	356	118	735	7
Aug	2,440	-	18	414	276	492	359	129	745	7
[Germany invades Poland, 1 Sep 1939]										
Sep	2,473	-	22	428	278	489	359	136	754	7
Oct	2,507	-	27	446	277	490	365	137	758	7
Nov	2,536	-	32	458	275	498	375	136	755	7
Dec	2,546	-	39	464	274	492	378	131	761	7
<b>1940</b>										
Jan	2,588	-	45	466	271	464	409	128	798	7
Feb	2,658	-	49	470	271	458	415	128	860	7
Mar	2,709	-	54	468	267	453	415	125	920	7
Apr	2,806	-	54	468	263	451	416	125	1,022	7
May	2,906	-	54	470	259	459	410	124	1,123	7
Jun	2,966	-	54	478	166	477	414	127	1,243	7
[France surrenders to Germany, 25 Jun 1940] [Battle of Britain begins, 10 July 1940]										
Jul	3,102	-	56	483	161	500	410	128	1,357	7
Aug	3,295	-	65	485	158	539	407	128	1,506	7

Source: Army Air Forces Statistical Digest, WW II

# O que faz com que estes dados não estejam arrumados?

	A	AA	AB	AC	AD	AE	AF	AG	AH
1	Estimated HIV Prevalence% - (Ages 15-49)	2004	2005	2006	2007	2008	2009	2010	2011
2	Abkhazia						0.06	0.06	0.06
3	Afghanistan								
4	Akrotiri and Dhekelia								
5	Albania								
6	Algeria	0.1	0.1	0.1	0.1	0.1			
7	American Samoa								
8	Andorra								
9	Angola	1.9	1.9	1.9	1.9	2.1	2.1	2.1	
10	Anguilla								
11	Antigua and Barbuda								
12	Argentina	0.4	0.4	0.4	0.4	0.5	0.4	0.4	0.4
13	Armenia	0.1	0.1	0.1	0.1	0.1	0.2	0.2	0.2
14	Aruba								
15	Australia	0.1	0.1	0.1	0.1	0.1	0.2	0.2	0.2
16	Austria	0.2	0.2	0.2	0.3	0.3	0.3	0.4	0.4
17	Azerbaijan	0.06	0.06	0.06	0.1	0.1	0.1	0.1	0.1
18	Bahamas	3	3	3	3.1	3.1	2.9	2.8	2.8

Source: Gapminder, Estimated HIV prevalence among 15-49 year olds

# O que faz com que estes dados não estejam arrumados?

Subject	United States			
	Estimate	Margin of Error	Percent	Percent Margin of Error
EMPLOYMENT STATUS				
Population 16 years and over	255,797,692	+/-17,051	255,797,692	(X)
In labor force	162,184,325	+/-135,158	63.4%	+/-0.1
Civilian labor force	161,159,470	+/-127,501	63.0%	+/-0.1
Employed	150,599,165	+/-138,066	58.9%	+/-0.1
Unemployed	10,560,305	+/-27,385	4.1%	+/-0.1
Armed Forces	1,024,855	+/-10,363	0.4%	+/-0.1
Not in labor force	93,613,367	+/-126,007	36.6%	+/-0.1
Civilian labor force	161,159,470	+/-127,501	161,159,470	(X)
Unemployment Rate	(X)	(X)	6.6%	+/-0.1
Females 16 years and over	131,092,196	+/-11,187	131,092,196	(X)
In labor force	76,493,327	+/-75,824	58.4%	+/-0.1
Civilian labor force	76,350,498	+/-75,238	58.2%	+/-0.1
Employed	71,451,559	+/-79,007	54.5%	+/-0.1
Own children of the householder under 6 years	22,939,897	+/-14,240	22,939,897	(X)
All parents in family in labor force	14,957,537	+/-36,506	65.2%	+/-0.1
Own children of the householder 6 to 17 years	47,007,147	+/-19,644	47,007,147	(X)
All parents in family in labor force	33,238,793	+/-49,036	70.7%	+/-0.1

Source: US Census Fact Finder, General Economic Characteristics, ACS 2017

# Exibição vs. resumo de dados

Saída

```
## # A tibble: 87 × 3
##   name      height mass
##   <chr>     <int> <dbl>
## 1 Luke Skywalker 172   77
## 2 C-3PO        167   75
## 3 R2-D2        96    32
## 4 Darth Vader 202   136
## 5 Leia Organa 150    49
## 6 Owen Lars   178   120
## # ... with 81 more rows
```

Código

```
## # A tibble: 3 × 2
##   gender avg_ht
##   <chr>   <dbl>
## 1 feminine 165.
## 2 masculine 177.
## 3 <NA>     181.
```

# Exibição vs. resumo de dados

Saída

Código

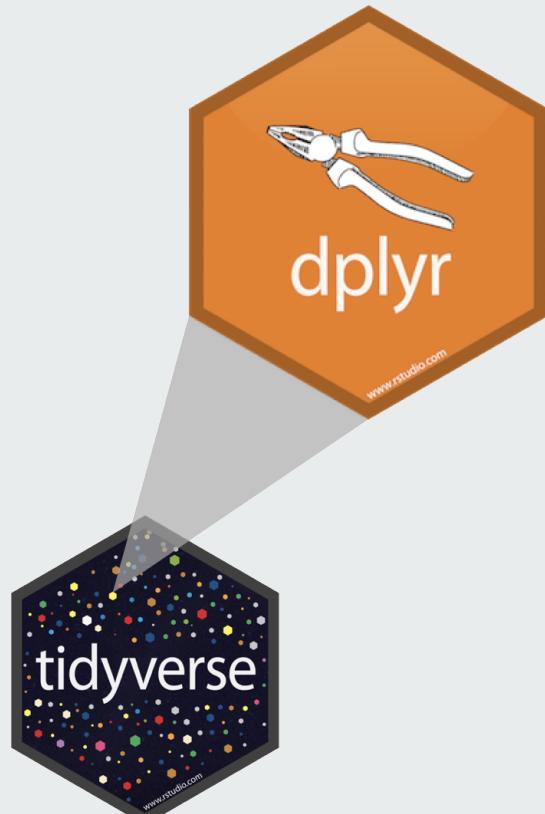
```
## # A tibble: 87 × 3
##   name      height mass
##   <chr>     <int> <dbl>
## 1 Luke Skywalker 172   77
## 2 C-3PO        167   75
## 3 R2-D2        96    32
## 4 Darth Vader 202   136
## 5 Leia Organa 150    49
## 6 Owen Lars   178   120
## # ... with 81 more rows
```

```
starwars %>%
  group_by(gender) %>%
  summarize(
    avg_ht = mean(height, na.rm = TRUE)
  )
```

# Gramática do data wrangling

# Gramática do data wrangling

Com base nos conceitos de funções como verbos que permitem a manipulação de dataframes



- **select**: seleção de colunas pelo nome
- **arrange**: reorganização das colunas
- **slice**: seleção de linhas pelo index(es)
- **filter**: seleção de linhas segundo algum critério
- **distinct**: aplica um filtro para que não haja colunas duplicadas
- **mutate**: adição de novas variáveis
- **summarise**: redução de variáveis a valores
- **group\_by**: operações em agrupamentos
- ... (many more)

# Regras das funções `dplyr`

- O primeiro argumento é **sempre** um dataframe
- Os argumentos subseqüentes dizem o que fazer com esse dataframe
- A saída é sempre um dataframe
- Não modifica os dados originais

# Dados: Reservas de hotéis

- Dados de dois hotéis: um resort e um hotel urbano
- Observações: Cada linha representa uma reserva de hotel
- Objetivo para a coleta de dados originais: Desenvolvimento de modelos de previsão para classificar a probabilidade de uma reserva de hotel ser cancelada (**Antonia et al., 2019**)

## Dados

```
library(readr)
hotels <- read_csv("./exercice/hotels.csv")
```

Source: **TidyTuesday**

# Primeiro olhar: Variáveis

```
names(hotels)
```

```
## [1] "hotel"
## [2] "is_canceled"
## [3] "lead_time"
## [4] "arrival_date_year"
## [5] "arrival_date_month"
## [6] "arrival_date_week_number"
## [7] "arrival_date_day_of_month"
## [8] "stays_in_weekend_nights"
## [9] "stays_in_week_nights"
## [10] "adults"
## [11] "children"
## [12] "babies"
## [13] "meal"
## [14] "country"
## [15] "market_segment"
## [16] "distribution_channel"
## [17] "is_repeated_guest"
## [18] "previous_cancellations"
...
...
```

# Segundo olhar: Visão geral

```
glimpse(hotels)
```

```
## Rows: 119,390
## Columns: 32
## $ hotel <chr> "Resort Hotel", "Resort ...
## $ is_canceled <dbl> 0, 0, 0, 0, 0, 0, 0, ...
## $ lead_time <dbl> 342, 737, 7, 13, 14, 14, ...
## $ arrival_date_year <dbl> 2015, 2015, 2015, 2015, ...
## $ arrival_date_month <chr> "July", "July", "July", ...
## $ arrival_date_week_number <dbl> 27, 27, 27, 27, 27, 27, ...
## $ arrival_date_day_of_month <dbl> 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ stays_in_weekend_nights <dbl> 0, 0, 0, 0, 0, 0, 0, ...
## $ stays_in_week_nights <dbl> 0, 0, 1, 1, 2, 2, 2, 2, ...
## $ adults <dbl> 2, 2, 1, 1, 2, 2, 2, 2, ...
## $ children <dbl> 0, 0, 0, 0, 0, 0, 0, ...
## $ babies <dbl> 0, 0, 0, 0, 0, 0, 0, ...
## $ meal <chr> "BB", "BB", "BB", "BB", ...
## $ country <chr> "PRT", "PRT", "GBR", "GB...
## $ market_segment <chr> "Direct", "Direct", "Dir...
## $ distribution_channel <chr> "Direct", "Direct", "Dir...
...
...
```

# Selecione uma única coluna

Ver apenas **lead\_time** (número de dias entre a reserva e a data de chegada):

```
select(hotels, lead_time)
```

```
## # A tibble: 119,390 × 1
##   lead_time
##   <dbl>
## 1 342
## 2 737
## 3 7
## 4 13
## 5 14
## 6 14
## # ... with 119,384 more rows
```

# Selecione uma única coluna

```
select(  
    hotels,  
    lead_time  
)
```

- Comece com a função (um verbo):  
**select()**

# Selecione uma única coluna

```
select(  
  hotels,  
  lead_time  
)
```

- Comece com a função (um verbo): **select()**
- Primeiro argumento: dataframe com o qual estamos trabalhando, **hotels**.

# Selecione uma única coluna

```
select(  
  hotels,  
  lead_time  
)
```

- Comece com a função (um verbo): **select()**
- Primeiro argumento: dataframe com o qual estamos trabalhando, **hotels**.
- Segundo argumento: variável que queremos acessar, **lead\_time**

# Selecione uma única coluna

```
select(  
  hotels,  
  lead_time  
)
```

```
## # A tibble: 119,390 × 1  
##   lead_time  
##     <dbl>  
## 1    342  
## 2    737  
## 3     7  
## 4    13  
## 5    14  
## 6    14  
## # ... with 119,384 more rows
```

- Comece com a função (um verbo): **select()**
- Primeiro argumento: dataframe com o qual estamos trabalhando, **hotels**.
- Segundo argumento: variável que queremos acessar, **lead\_time**
- Resultado: data frame com 119390 linhas e 1 coluna

dplyr funções precisam de um data frame como entrada e entregam um data frame.

```
select(hotels, lead_time)
```

```
## # A tibble: 119,390 × 1
##   lead_time
##   <dbl>
## 1 342
## 2 737
## 3 7
## 4 13
## 5 14
## 6 14
## # ... with 119,384 more rows
```

# Selecione várias colunas

Veja apenas o tipo de **hotel** e o **lead\_time**:

```
select(hotels, hotel, lead_time)
```

```
## # A tibble: 119,390 × 2
##   hotel      lead_time
##   <chr>     <dbl>
## 1 Resort Hotel    342
## 2 Resort Hotel    737
## 3 Resort Hotel     7
## 4 Resort Hotel    13
## 5 Resort Hotel    14
## 6 Resort Hotel    14
## # ... with 119,384 more rows
```

E se quiséssemos selecionar estas colunas, e depois organizar os dados em ordem decrescente conforme data?

# Data wrangling, passo-a-passo

Selecione:

```
hotels %>%  
  select(hotel, lead_time)
```

```
## # A tibble: 119,390 × 2  
##   hotel    lead_time  
##   <chr>     <dbl>  
## 1 Resort Hotel    342  
## 2 Resort Hotel    737  
## 3 Resort Hotel     7  
## 4 Resort Hotel    13  
## 5 Resort Hotel    14  
## 6 Resort Hotel    14  
## # ... with 119,384 more rows
```

Selecione, depois organize:

```
hotels %>%  
  select(hotel, lead_time) %>%  
  arrange(desc(lead_time))
```

```
## # A tibble: 119,390 × 2  
##   hotel    lead_time  
##   <chr>     <dbl>  
## 1 Resort Hotel    737  
## 2 Resort Hotel    709  
## 3 City Hotel     629  
## 4 City Hotel     629  
## 5 City Hotel     629  
## 6 City Hotel     629  
## # ... with 119,384 more rows
```

# Pipes

# O que é um pipe?

Na programação, **pipe** é uma técnica para passar informações de um processo para outro.

- Comece com o dataframe **hotels**, e utilize a função **select()**,

```
hotels %>%
  select(hotel, lead_time) %>%
  arrange(desc(lead_time))
```

```
## # A tibble: 119,390 × 2
##   hotel    lead_time
##   <chr>     <dbl>
## 1 Resort Hotel    737
## 2 Resort Hotel    709
## 3 City Hotel     629
## 4 City Hotel     629
## 5 City Hotel     629
## 6 City Hotel     629
## # ... with 119,384 more rows
```

# O que é um pipe?

Na programação, **pipe** é uma técnica para passar informações de um processo para outro.

- Comece com o dataframe **hotels**, e utilize a função **select()**,
- e então, selecione **hotel** and **lead\_time**,

```
hotels %>%  
  select(hotel, lead_time) %>%  
  arrange(desc(lead_time))
```

```
## # A tibble: 119,390 × 2  
##   hotel    lead_time  
##   <chr>     <dbl>  
## 1 Resort Hotel     737  
## 2 Resort Hotel     709  
## 3 City Hotel       629  
## 4 City Hotel       629  
## 5 City Hotel       629  
## 6 City Hotel       629  
## # ... with 119,384 more rows
```

# O que é um pipe?

Na programação, **pipe** é uma técnica para passar informações de um processo para outro.

- Comece com o dataframe **hotels**, e utilize a função **select()**,
- e então, selecione **hotel** and **lead\_time**,
- e então, organize de maneira decrescente conforme o **lead\_time**.

```
hotels %>%  
  select(hotel, lead_time) %>%  
  arrange(desc(lead_time))
```

```
## # A tibble: 119,390 × 2  
##   hotel      lead_time  
##   <chr>       <dbl>  
## 1 Resort Hotel     737  
## 2 Resort Hotel     709  
## 3 City Hotel       629  
## 4 City Hotel       629  
## 5 City Hotel       629  
## 6 City Hotel       629  
## # ... with 119,384 more rows
```

# Além

O pipe é implementado no pacote **magrittr**, embora não precisemos carregar este pacote explicitamente, pois **tidyverse** faz isto por nós.

Algun palpite sobre o porquê do pacote ser chamado de magrittr?



%>%  
magrittr

*Ceci n'est pas un pipe.*

# Como funciona o pipe?

- Você pode pensar na seguinte seqüência de ações - encontrar chaves, destravar o carro, ligar o carro, dirigir para o trabalho, estacionar.
- Expresso como um conjunto de funções aninhadas no pseudo-código R, isto pareceria:

```
park(drive(start_car(find("keys")), to = "work"))
```

- Escrevê-lo usando pipes dá-lhe uma estrutura mais natural (e mais fácil de ler):

```
find("keys") %>%
  start_car() %>%
  drive(to = "work") %>%
  park()
```

# Uma nota sobre pipes e estratificação

- utilizado principalmente em **dplyr** pipelines, *canalizamos a saída da linha de código anterior como a primeira entrada da próxima linha de código.*
- O **+** utilizado em **ggplot2** é utilizado para "estratificação", *criamos a parcela em camadas, separadas por +*

# dplyr



```
hotels +  
  select(hotel, lead_time)
```

```
## Error in select(hotel, lead_time): objeto 'hotel' não encontrado
```



```
hotels %>%  
  select(hotel, lead_time)
```

```
## # A tibble: 119,390 × 2  
##   hotel    lead_time  
##   <chr>     <dbl>  
## 1 Resort Hotel     342  
## 2 Resort Hotel     737  
## 3 Resort Hotel      7  
...  
...
```

# ggplot2



```
ggplot(hotels, aes(x = hotel, fill = deposit_type)) %>%  
  geom_bar()
```

```
## Error in `validate_mapping()`:  
## !`mapping` must be created by `aes()`  
## Did you use %>% instead of +?
```



```
ggplot(hotels, aes(x = hotel, fill = deposit_type)) +  
  geom_bar()
```



# Gramática do data wrangling

- **select**: seleção de colunas pelo nome
- **arrange**: reorganização das colunas
- **slice**: seleção de linhas pelo index(es)
- **filter**: seleção de linhas segundo algum critério
- **distinct**: aplica um filtro para que não haja colunas duplicadas
- **mutate**: adição de novas variáveis
- **summarise**: redução de variáveis a valores
- **group\_by**: operações em agrupamentos
- ... (many more)

# Data wrangling, passo-a-passo

Selecione:

```
hotels %>%  
  select(hotel, lead_time)
```

Selecione, depois organize:

```
hotels %>%  
  select(hotel, lead_time) %>%  
  arrange(desc(lead_time))
```

# O que é um pipe?

Na programação, **pipe** é uma técnica para passar informações de um processo para outro.

```
hotels %>%
  select(hotel, lead_time) %>%
  arrange(desc(lead_time))
```

# Operadores aritméticos R

Operador	Descrição
$x + y$	Adição de x com y
$x - y$	Subtração de y em x
$x * y$	Multiplicação de x e y
$x / y$	Divisão de x por y
$x^y$ ou $x^{**}y$	x elevado a y-ésima potência
$x\% \% y$	Resto da divisão de x por y (módulo)
$x\% /\% y$	Parte inteira da divisão de x por y

# Operadores de comparação no R

Operador	Significado
<code>==</code>	igual a
<code>!=</code>	diferente de
<code>&gt;</code>	maior que
<code>&lt;</code>	menor que
<code>&gt;=</code>	maior ou igual a
<code>&lt;=</code>	menor ou igual a

Os operadores de comparação sempre retornam um valor lógico TRUE ou FALSE.

# Operadores lógicos no R

Operador	Descrição	Explicação
&	AND lógico	Versão vetorizada. Compara dois elementos do tipo vetor e retorna um vetor de TRUEs e FALSEs
	OR lógico	Versão vetorizada. Compara dois elementos do tipo vetor e retorna um vetor de TRUEs e FALSEs
!	NOT lógico	Negação lógica. Retorna um valor lógico único ou um vetor de TRUE / FALSE.

Também conhecidos como operadores booleanos, permitem trabalhar com múltiplas condições relacionais na mesma expressão, e retornam valores lógicos verdadeiro ou falso.

# Algumas funções estatísticas para summarização de dados

Funções	Descrição
<code>min()</code>	mínimo
<code>max()</code>	máximo
<code>range()</code>	amplitude
<code>mean()</code>	média
<code>sum()</code>	soma
<code>median()</code>	mediana
<code>sd()</code>	desvio-padrão
<code>IQR()</code>	intervalo interquartil

Funções	Descrição
<code>quantile()</code>	quartis
<code>var()</code>	variância
<code>cor()</code>	correlação
<code>summary()</code>	métricas de summarização
<code>rowMeans()</code>	média das linhas
<code>colMeans()</code>	média das colunas
<code>rowSums()</code>	soma das linhas
<code>colSums()</code>	soma das colunas

# Tratamento de dados omissos

O R permite que sejam armazenados, em vetores e data.frames, o valor **NA** (Not Available), que representa dados que ainda não são conhecidos.

| **x == NA** trará sempre um resultado FALSE, mesmo que **x** não seja conhecido.

# Atividades em classe

1. Lab 3.1
2. Lab 3.2

60:00

# DÚVIDAS?