INF 1514

Introdução à Análise de Dados

Material 7

PUC
RIO

Pacote dplyr

- O pacote **dplyr** é hoje um dos pacotes mais utilizados para **manipulação** de **data frames**.
- Ele disponibiliza diversas funções, muitas “equivalentes” às funções básicas do R, mas com melhorias que nos **poupam tempo** e deixam o código **mais fácil de interpretar**.
- Em aulas anteriores já exploramos algumas funções do pacote como:
  - **select**: permite **selecionar colunas** dos **data frames**;
  - **filter:** permite **selecionar linhas** específicas dos **data frames**;
  - **arrange:** permite **ordenar** os **data frames**;
  - **mutate**: permite **criar** novas **colunas** nos **data frames**.
- Iremos explorar agora outras funções do pacote **dplyr**.

Pacote dplyr

- Considere o banco de dados Financeiro.db que apresenta as tabelas Cliente e NotaFiscal.
- Tabela Cliente.

Codigo	Nome	Estado
123	Transportes Tico	RJ
201	Hotel Aloft	SP
426	Gerson construções	MG
451	Distribuidora Antunes	RJ
678	Mercado Ganha Pouco	SP
890	Colégio 11 de Julho	ES

Pacote dplyr

- Tabela NotaFiscal.

NumeroNota	Serie	CodigoCliente	DataEmissao	Valor	Imposto	DataCancelamento
138092	A1	123	05/01/2019	3450	345	NULL
236781	B1	201	10/01/2019	3000	200	20/01/2019
387209	C2	426	03/05/2019	3800	190	NULL
674240	B1	201	06/07/2019	6480	432	NULL
764901	C1	NULL	10/07/2019	8540	427	NULL
834092	A2	123	11/08/2019	31560	789	NULL
834519	A1	426	25/08/2019	200	20	NULL
842854	C1	NULL	30/09/2019	3050	305	NULL
934891	B1	201	01/10/2019	4800	320	NULL

Pacote dplyr

- Carregando os data frames cliente e nota.

```
library (DBI)
library (RSQLite)
library (dplyr)

con <- dbConnect (RSQLite::SQLite(), dbname = "d:\\Temp\\Financeiro.db")

cliente <- dbGetQuery (con, "SELECT * from Cliente;")
nota <- dbGetQuery (con, "SELECT * from NotaFiscal;")

dbDisconnect(con)
```

Pacote dplyr

- A função **distinct** retorna somente os valores únicos removendo conteúdo duplicado.

```
estado.distinto <- distinct (cliente, Estado)
```

estado.distinto contém todos os estados presentes na coluna Estado mas sem repetição.

```
> estado.distinto
  Estado
1     RJ
2     SP
3     MG
4     ES
```

```
estado.all.distinto <- distinct (cliente, Estado, .keep_all = T)
```

.keep_all = T modifica o comando trazendo todas as colunas do datra frame cliente de um estado selecionado.

```
> estado.all.distinto
  Codigo                 Nome Estado
1    123    Transportes Tico     RJ
2    201          Hotel Aloft     SP
3    426  Gerson construções     MG
4    890 Colégio 11 de Julho     ES
```

Pacote dplyr

- A função **slice** extrai linhas por posição.

```
linhas.selecionadas.cliente <- slice(cliente, 2:4)
```

```
> cliente
  Codigo                 Nome Estado
1    123      Transportes Tico     RJ
2    201           Hotel Aloft     SP
3    426    Gerson construções     MG
4    451 Distribuidora Antunes     RJ
5    678   Mercado Ganha Pouco     SP
6    890   Colégio 11 de Julho     ES
```

```
> linhas.selecionadas.cliente
  Codigo                 Nome Estado
1    201           Hotel Aloft     SP
2    426    Gerson construções     MG
3    451 Distribuidora Antunes     RJ
```

linhas.selecionadas.cliente contém as linhas 2, 3 e 4 do data frame cliente.

Pacote dplyr

- A função **top_n** seleciona as **n** primeiras ou as **n** últimas linhas por valor de uma ou mais colunas.

```
top.maior.valor.nota <- top_n(nota, -3, Imposto)
```

```
> top.menor.valor.nota
  NumeroNota Serie CodigoCliente DataEmissao Valor Imposto DataCancelamento
1     236781    B1            201  2019-10-01  3000     200       2019-20-01
2     387209    C2            426  2019-03-05  3800     190             <NA>
3     834519    A1            426  2019-25-08   200      20             <NA>
```

top.menor.valor.nota contém as três últimas linhas (número -3 no comando) do data frame nota que possuem maior valor para a coluna Imposto.

Pacote dplyr

- A função **top_n** seleciona as primeiras ou as últimas linhas por valor de uma ou mais colunas.

```
top.maior.valor.nota <- top_n(nota, 2, Imposto)
```

```
> top.maior.valor.nota
  NumeroNota Serie CodigoCliente DataEmissao Valor Imposto DataCancelamento
1     674240    B1            201  2019-06-07  6480     432              <NA>
2     834092    A2            123  2019-11-08 31560     789              <NA>
```

top.maior.valor.nota contém as duas primeiras linhas (número 2 no comando) do data frame nota que possuem maior valor para a coluna Imposto.

Pacote dplyr

- A função **top_frac** seleciona, percentualmente, as primeiras ou as últimas linhas por valor de uma ou mais colunas.

```
top.maior.valor.nota.frac <- top_frac(nota, 0.35, Valor - Imposto)
```

```
> top.maior.valor.nota.frac
  NumeroNota Serie CodigoCliente DataEmissao Valor Imposto DataCancelamento
1     674240    B1            201  2019-06-07  6480     432              <NA>
2     764901    C1             NA  2019-10-07  8540     427              <NA>
3     834092    A2            123  2019-11-08 31560     789              <NA>
```

top.maior.valor.nota.frac contém as três primeiras linhas (35% do total de linhas) do data frame nota que possuem maior valor para a diferença entre as colunas Valor e Imposto.

Pacote dplyr

- A função **top_frac** seleciona, percentualmente, as primeiras ou as últimas linhas por valor de uma ou mais colunas.

```
top.menor.valor.nota.frac <- top_frac(nota, -0.60, Valor - Imposto)
```

```
> top.menor.valor.nota.frac
 NumeroNota Serie CodigoCliente DataEmissao Valor Imposto DataCancelamento
1     138092    A1           123  2019-05-01  3450     345            <NA>
2     236781    B1           201  2019-10-01  3000     200      2019-20-01
3     387209    C2           426  2019-03-05  3800     190            <NA>
4     834519    A1           426  2019-25-08   200      20            <NA>
5     842854    C1            NA  2019-30-09  3050     305            <NA>
```

top.menor.valor.nota.frac contém as cinco últimas linhas (60% do total de linhas e negativo) do data frame nota que possuem maior valor para a diferença entre as colunas Valor e Imposto.

Pacote dplyr

- A função **count** retorna o número de linhas com a combinação única do conteúdo das variáveis indicadas.

```
conta.cliente.estado <- count(cliente, Estado)
```

conta.cliente.estado contém os quatro estados presentes na coluna Estado do data frame cliente e as respectivas quantidades de clientes.

```
> conta.cliente.estado
  Estado n
1     ES 1
2     MG 1
3     RJ 2
4     SP 2
```

```
conta.nota.cliente.serie <- count(nota, CodigoCliente, Serie)
```

conta.nota.cliente.serie contém as seis combinações existentes das colunas CodigoCliente e Serie no data frame nota e as respectivas quantidades de linhas.

```
> conta.nota.cliente.serie
  CodigoCliente Serie n
1           123    A1 1
2           123    A2 1
3           201    B1 3
4           426    A1 1
5           426    C2 1
6            NA    C1 2
```

Pacote dplyr

- A função **summarise** permite criar um resumo do data frame, através do cálculo de estatísticas de suas colunas.

```
summarise1.nota <- summarise(nota, quantidade = n(), maximo = max(Valor))
```

summarise1.nota contém duas colunas, uma com a quantidade de linhas (função n()) do data frame nota e outra com o valor máximo (função max()) da coluna Valor.

```
> summarise1.nota
  quantidade maximo
1          9  31560
```

```
summarise2.nota <- summarise(nota, qtd = n(), serie = n_distinct(Serie), minimo = min(Valor))
```

summarise2.nota contém três colunas, uma com a quantidade de linhas (função n()) do data frame nota, outra com a quantidade distinta (função n_distinct()) de séries de nota fiscal e uma terceira com o valor mínimo (função min()) da coluna Valor.

```
> summarise2.nota
 quantidade serie minimo
1          9     5    200
```

Pacote dplyr

- A função **group_by** permite separar linhas em grupos.

```
grupo.cliente <- group_by(cliente, Estado)
sumario.cliente <- summarise(grupo.cliente, n())
```

```
> grupo.cliente
# A tibble: 6 x 3
# Groups:   Estado [4]
  Codigo Nome                  Estado
   <int> <chr>                 <chr>
1    123 Transportes Tico      RJ
2    201 Hotel Aloft           SP
3    426 Gerson construções    MG
4    451 Distribuidora Antunes RJ
5    678 Mercado Ganha Pouco   SP
6    890 Colégio 11 de Julho   ES
```

```
> sumario.cliente
# A tibble: 4 x 2
  Estado `n()`
  <chr>  <int>
1 ES         1
2 MG         1
3 RJ         2
4 SP         2
```

grupo.cliente contém quatro grupos definidos pela coluna Estado e sumario.cliente contém os quatro grupos, definidos pelos estados, e a quantidade de linhas em cada grupo.

Pacote dplyr

- A função **inner_join (a, b, by)** retorna um data frame com as linhas do data frame **a** que possuem correspondente no **b**. o data frame retornado possui todas as colunas de **a** e **b**. No parâmetro **by** é especificada a relação de correspondência.

```
inner.join.cliente.nota <- inner_join(cliente, nota, by = c("Codigo" = "CodigoCliente"))
```

```
> inner.join.cliente.nota
  Codigo                Nome Estado NumeroNota Serie DataEmissao Valor Imposto DataCancelamento
1    123   Transportes Tico     RJ     138092    A1  2019-05-01  3450     345             <NA>
2    123   Transportes Tico     RJ     834092    A2  2019-11-08 31560     789             <NA>
3    201        Hotel Aloft     SP     236781    B1  2019-10-01  3000     200       2019-20-01
4    201        Hotel Aloft     SP     674240    B1  2019-06-07  6480     432             <NA>
5    201        Hotel Aloft     SP     934891    B1  2019-01-10  4800     320             <NA>
6  426 Gerson construções     MG     387209    C2  2019-03-05  3800     190
7         <NA>
7    426 Gerson construções     MG     834519    A1  2019-25-08   200      20             <NA>
```

inner.join.cliente.nota contém as linhas dos data frames cliente e nota que possuem correspondência. O data frame possui todas as colunas de cliente e nota. A função inner_join é equivalente ao Inner Join do SQL

Pacote dplyr

- A função **inner_join (a, b, by)** retorna um data frame com as linhas do data frame **a** que possuem correspondente no **b**. o data frame retornado possui todas as colunas de **a** e **b**. No parâmetro **by** é especificada a relação de correspondência.

Também é possível e, muitas vezes necessário, utilizar mais de uma variável para vincular dois data frames. Nesse caso, a função ficaria da seguinte forma: inner_join (dataframe1, dataframe2, by = c("variavel_comum_1", "variavel_comum_2")).

Pacote dplyr

- A função **left_join (a, b, by)** retorna todas as linhas que estão no data frame **a** (mesmo sem correspondência em **b**) e as linhas de **b** que são comuns ao **a**. o data frame retornado possui todas as colunas de **a** e **b**. No parâmetro **by** é especificada a relação de correspondência.

```
left.join.nota.cliente <- left_join(nota, cliente, by = c("CodigoCliente" = "Codigo"))
```

```
> left.join.nota.cliente
NumeroNota Serie CodigoCliente DataEmissao Valor Imposto DataCancelamento              Nome Estado
1     138092    A1           123  2019-05-01  3450     345              <NA>   Transportes Tico     RJ
2     236781    B1           201  2019-10-01  3000     200       2019-20-01        Hotel Aloft     SP
3     387209    C2           426  2019-03-05  3800     190              <NA> Gerson construções     MG
4     674240    B1           201  2019-06-07  6480     432              <NA>        Hotel Aloft     SP
5     764901    C1            NA  2019-10-07  8540     427              <NA>               <NA>   <NA>
6     834092    A2           123  2019-11-08 31560     789              <NA>   Transportes Tico     RJ
7     834519    A1           426  2019-25-08   200      20              <NA> Gerson construções     MG
8     842854    C1            NA  2019-30-09  3050     305              <NA>               <NA>   <NA>
9     934891    B1           201  2019-01-10  4800     320              <NA>        Hotel Aloft     SP
```

left.join.nota.cliente contém todas as linhas do data frame nota que possuem ou não cliente e as linhas de cliente que possuem nota. É equivalente ao Left Join do SQL.

Pacote dplyr

- A função **right_join (a, b, by)** retorna um data frame com todas as linhas do data frame **b** (mesmo sem correspondência em **a**) e as linhas de **a** que são comuns a **b**. o data frame retornado possui todas as colunas de **a** e **b**. No parâmetro **by** é especificada a relação de correspondência.

```
right.join.nota.cliente <- right_join(nota, cliente, by = c("CodigoCliente" = "Codigo"))
```

```
> right.join.nota.cliente
```

NumeroNota	Serie	CodigoCliente	DataEmissao	Valor	Imposto	DataCancelamento	Nome	Estado
1	138092	A1	123	2019-05-01	3450	345	<NA>	Transportes Tico	RJ
2	236781	B1	201	2019-10-01	3000	200	2019-20-01	Hotel Aloft	SP
3	387209	C2	426	2019-03-05	3800	190	<NA>	Gerson construções	MG
4	674240	B1	201	2019-06-07	6480	432	<NA>	Hotel Aloft	SP
5	834092	A2	123	2019-11-08	31560	789	<NA>	Transportes Tico	RJ
6	834519	A1	426	2019-25-08	200	20	<NA>	Gerson construções	MG
7	934891	B1	201	2019-01-10	4800	320	<NA>	Hotel Aloft	SP
8	<NA>	<NA>	451	<NA>	NA	NA	<NA>	Distribuidora Antunes	RJ
9	<NA>	<NA>	678	<NA>	NA	NA	<NA>	Mercado Ganha Pouco	SP
10	<NA>	<NA>	890	<NA>	NA	NA	<NA>	Colégio 11 de Julho	ES

right.join.nota.cliente contém todas as linhas do data frame cliente que possuem ou não nota e as linhas de nota que possuem cliente. É equivalente ao Right Join do SQL

Pacote dplyr

- A função **full_join (a, b, by)** retorna um data frame com todas as linhas do data frame **a** (mesmo sem correspondência em **b**) e todas as linhas de **b** (mesmo sem correspondência em **a**).

```
full.join.cliente.nota <- full_join(cliente, nota, by = c("Codigo" = "CodigoCliente"))
```

```
> full.join.cliente.nota
```

Codigo	Nome	Estado	NumeroNota	Serie	DataEmissao	Valor	Imposto	DataCancelamento
1	123	Transportes Tico	RJ	138092	A1	2019-05-01	3450	345	<NA>
2	123	Transportes Tico	RJ	834092	A2	2019-11-08	31560	789	<NA>
3	201	Hotel Aloft	SP	236781	B1	2019-10-01	3000	200	2019-20-01
4	201	Hotel Aloft	SP	674240	B1	2019-06-07	6480	432	<NA>
5	201	Hotel Aloft	SP	934891	B1	2019-01-10	4800	320	<NA>
6	426	Gerson construções	MG	387209	C2	2019-03-05	3800	190	<NA>
7	426	Gerson construções	MG	834519	A1	2019-25-08	200	20	<NA>
8	451	Distribuidora Antunes	RJ	<NA>	<NA>	<NA>	NA	NA	<NA>
9	678	Mercado Ganha Pouco	SP	<NA>	<NA>	<NA>	NA	NA	<NA>
10	890	Colégio 11 de Julho	ES	<NA>	<NA>	<NA>	NA	NA	<NA>
11	NA	<NA>	<NA>	764901	C1	2019-10-07	8540	427	<NA>
12	NA	<NA>	<NA>	842854	C1	2019-30-09	3050	305	<NA>

full.join.cliente.nota contém todas as linhas do data frame cliente que possuem ou não nota e todas as linhas de nota que possuem ou não cliente. É equivalente ao Full Outer Join do SQL.

Pacote dplyr

- A função **semi_join (a, b, by)** retorna um data frame com todas as linhas do data frame **a** que possuem correspondência em **b**. o data frame retornado possui todas as colunas de **a** e **b**. No parâmetro **by** é especificada a relação de correspondência.

```
semi.join.cliente.nota <- semi_join(cliente, nota, by = c("Codigo" = "CodigoCliente"))
```

```
> semi.join.cliente.nota
  Codigo                Nome Estado
1    123   Transportes Tico     RJ
2    201         Hotel Aloft     SP
3    426 Gerson construções     MG
```

semi.join.cliente.nota contém todas as linhas do data frame cliente que possuem nota. Em SQL pode ser implementado usando Inner Joins, Exists ou In.

Pacote dplyr

- A função **anti_join (a, b, by)** retorna um data frame com todas as linhas do data frame **a** que **não** possuem correspondência em **b**. o data frame retornado possui todas as colunas de **a** e **b**. No parâmetro **by** é especificada a relação de correspondência.

```
anti.join.cliente.nota <- anti_join(cliente, nota, by = c("Codigo" = "CodigoCliente"))
```

```
> anti.join.cliente.nota
  Codigo                  Nome Estado
1    451 Distribuidora Antunes     RJ
2    678   Mercado Ganha Pouco     SP
3    890   Colégio 11 de Julho     ES
```

anti.join.cliente.nota contém todas as linhas do data frame cliente que não possuem nota. Em SQL pode ser implementado usando Inner Joins, Exists ou In.

Pacote dplyr

- A função **bind_rows (a, b)** retorna um data frame com todas as linhas do data frame **b** adicionadas ao data frame **a**.

```
cliente.RJ <- filter(cliente, Estado == "RJ")
novo.cliente.RJ <- data.frame(Codigo = 111, Nome = "Bike for all", Estado = "RJ")
cliente.RJ <- bind_rows(cliente.RJ, novo.cliente.RJ)
```

```
> cliente.RJ
  Codigo                      Nome Estado
1    123       Transportes Tico      RJ
2    451 Distribuidora Antunes      RJ
```

```
> cliente.RJ
  Codigo                      Nome Estado
1    123       Transportes Tico      RJ
2    451 Distribuidora Antunes      RJ
3    111           Bike for all      RJ
```

cliente.RJ contém ao final da sequência de comandos os dois clientes do estado “RJ” filtrados do data frame cliente mais o novo cliente (novo.cliente.RJ) adicionado pelo comando bind_rows.

Pacote dplyr

- A função **intersect (a, b)** retorna um data frame com todas as linhas comuns aos data frames **a** e **b**. Compara todas as colunas.

```
intersecao.cliente <- intersect(cliente, cliente.RJ)
```

```
> cliente
  Codigo                    Nome Estado
1    123       Transportes Tico     RJ
2    201             Hotel Aloft     SP
3    426     Gerson construções     MG
4    451 Distribuidora Antunes     RJ
5    678   Mercado Ganha Pouco     SP
6    890   Colégio 11 de Julho     ES
```

```
> cliente.RJ
  Codigo                    Nome Estado
1    123       Transportes Tico     RJ
2    451 Distribuidora Antunes     RJ
3    111           Bike for all     RJ
```

```
> intersecao.cliente
  Codigo                    Nome Estado
1    123       Transportes Tico     RJ
2    451 Distribuidora Antunes     RJ
```

intersecao.cliente contém somente os dois clientes comuns aos data frames cliente e cliente.RJ.

Pacote dplyr

- A função **union (a, b)** retorna um data frame com todas as linhas dos data frames **a** e **b** sem duplicidade. Compara todas as colunas.

```
uniao.cliente <- union(cliente, cliente.RJ)
```

```
> cliente
  Codigo                  Nome Estado
1    123       Transportes Tico     RJ
2    201             Hotel Aloft     SP
3    426     Gerson construções     MG
4    451 Distribuidora Antunes     RJ
5    678   Mercado Ganha Pouco     SP
6    890   Colégio 11 de Julho     ES
```

```
> cliente.RJ
  Codigo                  Nome Estado
1    123       Transportes Tico     RJ
2    451 Distribuidora Antunes     RJ
3    111            Bike for all     RJ
```

```
> uniao.cliente
  Codigo                  Nome Estado
1    123       Transportes Tico     RJ
2    201             Hotel Aloft     SP
3    426     Gerson construções     MG
4    451 Distribuidora Antunes     RJ
5    678   Mercado Ganha Pouco     SP
6    890   Colégio 11 de Julho     ES
7    111            Bike for all     RJ
```

uniao.cliente contém todos os clientes dos data frames cliente e cliente.RJ sem duplicar os clientes comuns.

Pacote dplyr

- A função **union_all (a, b)** retorna um data frame com todas as linhas dos data frames **a** e **b** sem duplicidade. Compara todas as colunas.

```
uniao.all.cliente <- union_all(cliente, cliente.RJ)
```

```
> cliente
  Codigo                    Nome Estado
1    123       Transportes Tico     RJ
2    201             Hotel Aloft     SP
3    426     Gerson construções     MG
4    451 Distribuidora Antunes     RJ
5    678   Mercado Ganha Pouco     SP
6    890   Colégio 11 de Julho     ES
> cliente.RJ
  Codigo                    Nome Estado
1    123       Transportes Tico     RJ
2    451 Distribuidora Antunes     RJ
3    111            Bike for all     RJ
```

```
> uniao.all.cliente
  Codigo                    Nome Estado
1    123       Transportes Tico     RJ
2    201             Hotel Aloft     SP
3    426     Gerson construções     MG
4    451 Distribuidora Antunes     RJ
5    678   Mercado Ganha Pouco     SP
6    890   Colégio 11 de Julho     ES
7    123       Transportes Tico     RJ
8    451 Distribuidora Antunes     RJ
9    111            Bike for all     RJ
```

uniao.all.cliente contém todos os clientes dos data frames cliente e cliente.RJ duplicando os clientes comuns.

Pacote dplyr

- A função **setdiff (a, b)** retorna um data frame com todas as linhas do data frame **a** que não estão no **b**. Compara todas as colunas.

```
diferenca.cliente <- setdiff(cliente, cliente.RJ)
```

```
> cliente
  Codigo                    Nome Estado
1    123       Transportes Tico     RJ
2    201             Hotel Aloft     SP
3    426     Gerson construções     MG
4    451 Distribuidora Antunes     RJ
5    678    Mercado Ganha Pouco     SP
6    890    Colégio 11 de Julho     ES
```

```
> cliente.RJ
  Codigo                    Nome Estado
1    123       Transportes Tico     RJ
2    451 Distribuidora Antunes     RJ
3    111            Bike for all     RJ
```

```
> diferenca.cliente
  Codigo                    Nome Estado
1    201             Hotel Aloft     SP
2    426     Gerson construções     MG
3    678 Mercado Ganha Pouco     SP
4    890 Colégio 11 de Julho     ES
```

uniao.all.cliente contém todos os clientes do data frame cliente que não estão no data frame cliente.RJ.

Operador %>% (pipe)

- O operador funciona da seguinte forma:

```
dataset %>% função()
```

- Isso significa que o objeto do lado esquerdo do operador (dataset) será inserido na função ao lado direito do operador no primeiro argumento da função.
- O operador %>% pode ser usado conjuntamente com inúmeras funções.
- Exemplo 1, de sequências equivalentes:

```
ordenado.cliente <- arrange(cliente, Codigo)
ordenado.cliente
```

```
ordenado.cliente <- cliente %>% arrange(Codigo)
ordenado.cliente
```

Operador %>% (pipe)

- Exemplo 2, o data frame cliente é submentido como parâmetro à função **filter** e o data frame resultante é submetido como parâmetro à função **arrange** resultando um data frame filtrado e ordenado:

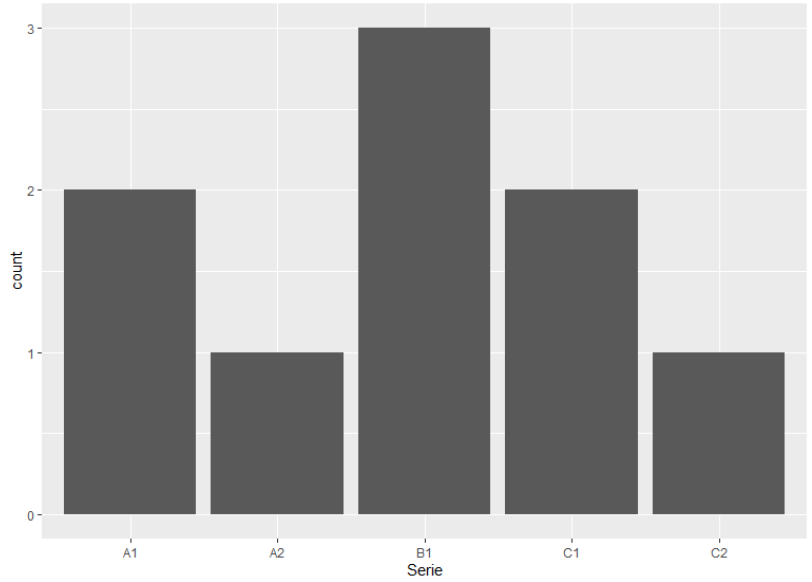
```
filtrado.ordenado.cliente <- cliente %>% filter(Estado == "RJ") %>% arrange(Codigo)
```

```
> filtrado.ordenado.cliente
  Codigo                   Nome Estado
1    123       Transportes Tico     RJ
2    451 Distribuidora Antunes     RJ
```

Operador %>% (pipe)

- Exemplo 3, o data frame **nota** é submentido como parâmetro à função **ggplot** que define o **dataset** como sendo **nota** e mapeia para **x** a coluna **Serie**. O gráfico apresenta no eixo **x** as séries e no eixo **y** a quantidade de notas para cada série:

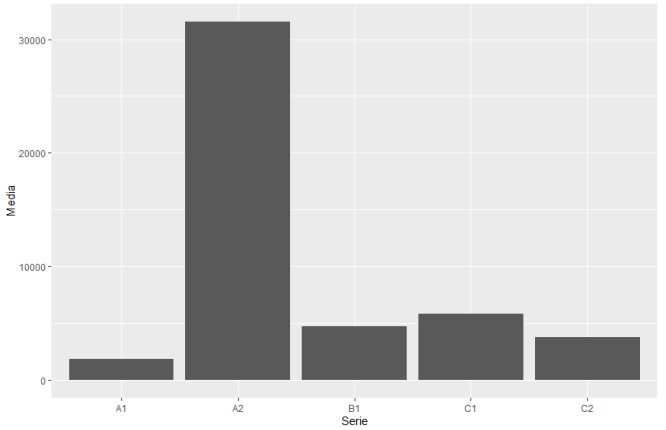
```
nota %>% ggplot(mapping = aes(x = Serie)) + geom_bar()
```



Operador %>% (pipe)

- Exemplo 4, o data frame **nota** é submentido como parâmetro à função **group_by** (agrupa **nota** por **Serie**), o resultado é submetido à função **summarise** (calcula a média do valor das notas por série) e o resultado da sumarização é submetido à função **ggplot** que define o **dataset** como o resultado da sumarização e mapeia para **x** a coluna **Serie** e para **y** a coluna **Media**. O gráfico apresenta no eixo **x** as séries e no eixo **y** a média do valor das notas para cada série:

```
nota %>% group_by(Serie) %>% summarise(Media = mean(Valor)) %>% ggplot(mapping =
aes(x = Serie, y = Media)) + geom_bar(stat = "identity")
```



Manipulação de hora e data

- Existem vários formatos para uma variável data e hora, como por exemplo:
  - 16/02/2020 14:11:43 - dia, mês, ano com 4 dígitos, horas, minutos e segundos;
  - 02/18/18 13:10 - mês, dia, ano com 2 dígitos, horas e minutos;
  - 12 Novembro 2020 - dia, mês por extenso e ano com 4 dígitos.
- Cada fonte de dados pode definir um padrão de hora e data diferente, porém, o R tem um formato padrão para manipulação dos dados.

Manipulação de hora e data

- O formato tradicional usado no R para data e hora é:
  - 2020-11-28 16:32:42 - ano com 4 dígitos, mês, dia, hora, minutos e segundos.
- Ou, somente a data:
  - 2020-11-28 - ano com 4 dígitos, mês, dia.
- Assim, quando os dados que for processar apresentarem uma variável de hora e data, é aconselhável transformá-la para o formato padrão do R.
- Os formatos de data e hora são definidos através de letras e para saber o que cada letra representa na definição do formato da variável de data, basta digitar no R o comando:
  - ?strptime

Manipulação de hora e data

- Para transformar dados originais de data e hora para o formato padrão do R, deve ser informado qual é o padrão original.

```
data1 <- c("14-02-2020")
data1 <- strptime(data1, format = "%d-%m-%Y")
data1
```

- Ao informar que o formato original dos dados é “%d-%m-%Y”, estamos avisando ao R que originalmente a variável de data está no formato dia (%d), mês (%m) e ano com 4 dígitos (%Y).

```
> data1
[1] "2020-02-14 -02"
```

- O -02 ao final da data1 convertida se refere à diferença de fuso horário em relação ao horário UTC (Tempo Universal Coordenado).

Manipulação de hora e data

- Para transformar dados originais de data e hora para o formato padrão do R, deve ser informado qual é o padrão original.

```
data2 <- c("15 Novembro 2018")
data2 <- strptime(data2, format = "%d %B %Y")
data2
```

- Ao informar que o formato original dos dados é “%d %B %Y”, estamos falando que primeiro é o dia (%d), depois o mês por extenso (%B) e depois o ano com 4 dígitos (%Y).

```
> data2
[1] "2018-11-15 -02"
```

- O -02 ao final da data2 convertida se refere à diferença de fuso horário em relação ao horário UTC (Tempo Universal Coordenado).

Manipulação de hora e data

- Para transformar dados originais de data e hora para o formato padrão do R, deve ser informado qual é o padrão original.

```
data.hora<-c("25 Fevereiro 18, 16:14:49")
data.hora <- strptime(data.hora, "%d %B %y, %H:%M:%S")
data.hora
```

- Ao informar que o formato original dos dados é “%d %B %y, %H:%M:%S”, estamos falando que primeiro é o dia (%d), depois o mês por extenso (%B), depois o ano com 2 dígitos (%y), depois as horas com 2 dígitos (%H), depois os minutos com 2 dígitos (%M) e depois os segundos com 2 dígitos (%S).

```
> data.hora
[1] "2018-02-25 16:14:49 -03"
```

- O -03 ao final da data.hora convertida se refere à diferença de fuso horário em relação ao horário UTC (Tempo Universal Coordenado).

Manipulação de hora e data

- Para obter o ano de uma data pode ser utilizado o comando:

```
data.hora<-c("25 Fevereiro 18, 16:14:49")
data.hora <- strptime(data.hora, "%d %B %y, %H:%M:%S")
data.hora

ano <- format(data.hora,"%Y")
ano
```

```
> ano
[1] "2018"
```

- Para obter o mês basta trocar %Y por %m no comando acima. Para obter o dia deve ser usado %d, as horas %H, os minutos %M e os segundos %S.
- Para obter as horas e minutos deve ser usado %H:%M.

Manipulação de hora e data

- A função **difftime** permite calcular a diferença entre duas datas:

```
data.hora1 <-c("25 Fevereiro 18, 16:14:49")
data.hora1 <- strptime(data.hora1, "%d %B %y, %H:%M:%S")

data.hora2 <-c("23 Fevereiro 18, 15:14:49")
data.hora2 <- strptime(data.hora2, "%d %B %y, %H:%M:%S")

diferenca.horas <- difftime(data.hora1, data.hora2, units='hours')
diferenca.horas


> diferenca.horas
Time difference of 49 hours
```

- A diferença também pode ser obtida em dias (days), minutos (mins), segundos (secs), etc.

Manipulação de hora e data

- Também é possível somar ou subtrair valores de uma data e hora, sendo o segundo a menor unidade:

```
data.hora1 <- c("25 Fevereiro 18, 16:14:49")
data.hora1 <- strptime(data.hora1, "%d %B %y, %H:%M:%S")
data.hora1 <- data.hora1 + 59
data.hora1
```

- À data.hora1 foi somado 59 segundos.

```
> data.hora1
[1] "2018-02-25 16:15:48 -03"
```

- Para reduzir 2 minutos pode ser usado:

```
data.hora1 <- c("25 Fevereiro 18, 16:14:49")
data.hora1 <- strptime(data.hora1, "%d %B %y, %H:%M:%S")
data.hora1 <- data.hora1 - 2*60
data.hora1
```

```
> data.hora1
[1] "2018-02-25 16:12:49 -03"
```

Pacote tidyverse

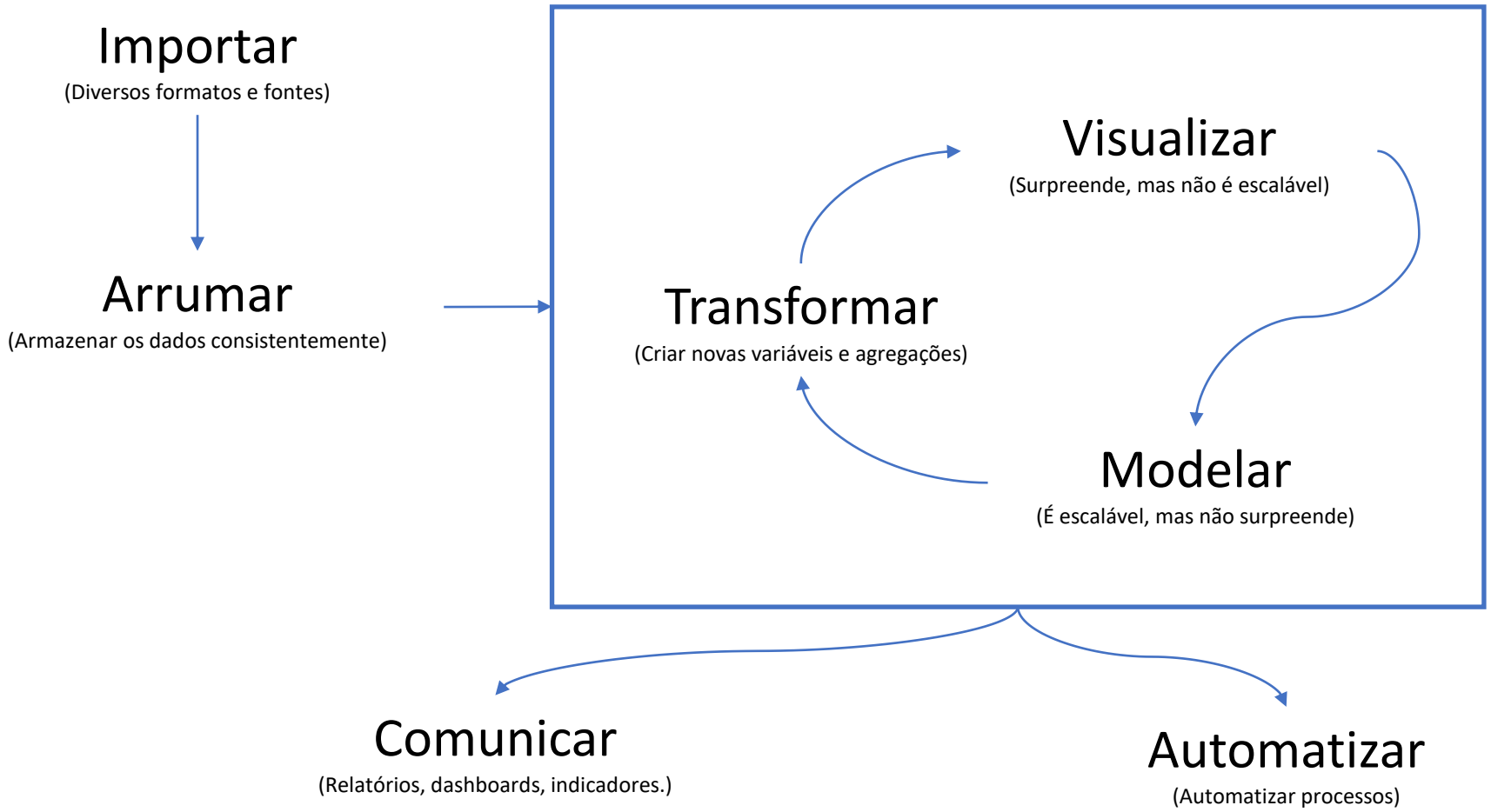
- O pacote **tidyverse** é um pacote **guarda-chuva** que consolida uma série de ferramentas que fazem parte o ciclo da ciência de dados.
- Oferece uma reimplementação e extensão das funcionalidades padrões do R para **manipulação** e **visualização** de dados.
- Os pacotes (abaixo) que fazem parte do **tidyverse** foram planejados e construídos para trabalhar em conjunto.

"broom”	"cli“	"crayon”	**"dplyr“**
"dbplyr"	"forcats"	**"ggplot2"**	"haven“
"hms"	"httr"	"jsonlite"	"lubridate“
"magrittr"	"modelr"	**"purrr"**	**"readr“**
"readxl"	"reprex"	"rlang"	"rstudioapi“
"rvest"	**"stringr"**	**"tibble"**	**"tidyr“**
"xml2"	"tidyverse"		

Pacote tidyverse

- Alguns dos princípios fundamentais do uso do **tidyverse** consiste em organizar funções simples usando o **pipe**.
- O pacote **stringr** apresenta uma série de recursos para manipulação de **strings**.
- O pacote **purrr** inclui funções semelhantes a **sapply** mas eles interagem melhor com outras funções **tidyverse**.
- O data frame é a estrutura nativa do R para representar tabelas de dados, o **tidyverse** agrega ao ambiente R o **tibble** que é uma reimplementação da estrutura do data frame com melhorias:
  - Método print (visualização) mais enxuto e informativo.
  - Mais consistente para seleção e modificação de conteúdo.
  - Colunas/cédulas podem representar objetos mais complexos.

Pacote tidyverse



Operações típicas com tidyverse

- Importar e/ou acessar dados.
- Ordenar registros.
- Selecionar e fatiar dados por linhas ou colunas.
- Filtrar registros.
- Renomear colunas.
- Reorganizar a disposição do conteúdo.
- Modificar e/ou transformar dados.
- Aplicar funções e calcular resumos.
- Agregar dados por categorias.
- Concatenar, juntar e/ou conciliar tabelas.

Seguindo, serão apresentados exemplos de algumas operações típicas com tidyverse em dados no formato tibble.

Pacote tidyverse (tibble)

```
> cliente
           Id                          Nome DDD  Telefone Classificacao
1  42450854525     Júlio Luiz Fausto Fernandes  11 360021983             L
2  23712872453              Alex Wulff Moraes  11 762087607             L
3  38614430159          Casemiro  Paulo Cormack  12 458393440             V
4  82559918404    Maria Cristina Garcia Vargas  12 336841205             L
5  33421388318            Thainá Diniz Frotté  12 699167650             N
6  14768110401            José Ximenes Vargas  11 334940412             L
                        ...
97  22841875954       Adilson de Padua Pedroso  11 962112712             L
98  40721885225             Telmo Knupp Filho  11 920013264             A
99  26071793432         Ivan Felizardo Sarmento  11 903649616             L
100 12150713373            Benício Arlia Duarte  11 947685566             A
> |
```

Data frame cliente à esquerda e o mesmo data frame transformado em tibble abaixo.

Print mais organizado.
Por padrão, apenas as dez primeiras linhas são apresentadas. Também são apresentadas a dimensão (100x5) e as classes de cada coluna.

```
> as_tibble(cliente)
# A tibble: 100 x 5
   Id          Nome                          DDD   Telefone  Classificacao
   <chr>       <chr>                         <chr> <chr>     <chr>
 1 42450854525 Júlio Luiz Fausto Fernandes   11    360021983 L
 2 23712872453 Alex Wulff Moraes             11    762087607 L
 3 38614430159 Casemiro  Paulo Cormack       12    458393440 V
 4 82559918404 Maria Cristina Garcia Vargas  12    336841205 L
 5 33421388318 Thainá Diniz Frotté           12    699167650 N
 6 14768110401 José Ximenes Vargas           11    334940412 L
 7 84823241282 Raul Vitorino Antunes         11    321840034 A
 8 28371521294 Rosali Baesso Marques         11    978342023 A
 9 93227565005 Margareth Sampaio Cruz        17    955388502 N
10 23205940270 Vânia Cordeiro Queiroz        17    913404039 L
# ... with 90 more rows
```

Pacote readr

- A função **read_csv2** permite carregar um arquivo **CSV** em uma **tibble**.

```
clientes <- read_csv2("NovosClientes.csv", col_types = cols(col_character(),
col_character(), col_character(), col_character(), col_integer()))
```

```
> clientes
# A tibble: 100 x 6
   CPF         Nome                       DDD   Telefone  Classificacao Divida
   <chr>       <chr>                      <chr> <chr>     <chr>          <int>
 1 10244521310 Margarida Palmas Stutz     12    959128774 L                 NA
 2 11125248785 Cláudio Peixoto Barellos   17    945271192 NA                NA
 3 12150713373 Benício Arlia Duarte       11    947685566 A                 NA
 4 14436831590 Tereza Pires Barbosa       22    994053970 A                 NA
 5 14768110401 José Ximenes Vargas        11    334940412 NA                NA
 6 15271527409 Alba Knupp Barrada         22    931442914 NA                NA
 7 15445645207 Marisa de Padua da Sousa   21    911021168 L                 NA
 8 15796045440 Marlene Silva Meyer        19    991341440 L                 NA
 9 18112077231 Júlio Rosa Viana           22    917547560 L                 NA
10 19167287409 Lina de Lima Uchôa         13    925186816 L                 NA
# ... with 90 more rows
```

Pacote dplyr

- A função **arrange** permite ordenar por uma ou mais colunas.

```
cli <- clientes %>% arrange(DDD, Nome)
```

```
> cli
# A tibble: 100 x 6
   CPF         Nome                          DDD   Telefone  Classificacao Divida
   <chr>       <chr>                         <chr> <chr>     <chr>           <int>
 1 22841875954 Adilson de Padua Pedroso      11    962112712 L                  NA
 2 23712872453 Alex Wulff Moraes             11    762087607 L                  NA
 3 12150713373 Benício Arlia Duarte          11    947685566 A                  NA
 4 36672030395 Clemilda Neves Figueiredo     11    906497335 V                  NA
 5 32615743481 Glauber Carvalho Muchão       11    929201474 L                  NA
 6 45484376300 Grecy Sales Alfradique        11    949096195 L                  NA
 7 68781251386 Iára Figueiro Chaves          11    960514681 V                  NA
 8 26071793432 Ivan Felizardo Sarmento       11    903649616 L                  NA
 9 14768110401 José Ximenes Vargas           11    334940412 NA                 NA
10 42450854525 Júlio Luiz Fausto Fernandes   11    360021983 L                  NA
# ... with 90 more rows
```

Pacote dplyr

- A função **slice** permite extrair linhas por posição.

```
cli <- clientes %>% arrange(DDD, Nome) %>% slice(2:5)
```

```
> cli
# A tibble: 4 x 6
  CPF         Nome                       DDD   Telefone  Classificacao Divida
  <chr>       <chr>                      <chr> <chr>     <chr>           <int>
1 23712872453 Alex Wulff Moraes          11    762087607 L                  NA
2 12150713373 Benício Arlia Duarte       11    947685566 A                  NA
3 36672030395 Clemilda Neves Figueiredo  11    906497335 V                  NA
4 32615743481 Glauber Carvalho Muchão    11    929201474 L                  NA
```

Pacote dplyr

- A função **filter** permite filtrar por uma ou mais colunas.

```
cli <- clientes %>% filter(DDD %in% c("14", "13"))
```

```
> cli
# A tibble: 3 x 6
  CPF         Nome                      DDD   Telefone  Classificacao Divida
  <chr>       <chr>                     <chr> <chr>     <chr>          <int>
1 19167287409 Lina de Lima Uchôa        13    925186816 L                 NA
2 19377803411 Bryan Louzano Schuenck    13    979111679 A                 NA
3 72869674287 Marilena Cristo Monnerat  14    983594857 A                 NA
```

Pacote dplyr

- A função **sample_n** permite extrair uma amostra de tamanho **size** com ou sem reposição.

```
cli <- clientes %>% sample_n(size = 5, replace = FALSE)
```

```
> cli
# A tibble: 5 x 6
  CPF         Nome                        DDD   Telefone  Classificacao Divida
  <chr>       <chr>                       <chr> <chr>     <chr>           <int>
1 84872296702 Dayvid Cretella Giacomini   19    927940075 N                  NA
2 65297312531 Casemiro  Palmas Billé      22    999427645 L                  NA
3 11125248785 Cláudio Peixoto Barellos    17    945271192 NA                 NA
4 87488541154 Simone Inácio Azevedo       15    926087052 A                  NA
5 28371521294 Rosali Baesso Marques       11    978342023 A                  NA
```

Pacote dplyr

- A função **sample_frac** permite extrair uma amostra percentual **size** com ou sem reposição.

```
cli <- clientes %>% sample_frac(size = 0.1, replace = FALSE)
```

```
> cli
# A tibble: 10 x 6
   CPF         Nome                                DDD   Telefone  Classificacao Divida
   <chr>       <chr>                               <chr> <chr>     <chr>          <int>
 1 84823241282 Raul Vitorino Antunes               11    321840034 A                 NA
 2 48224841901 Lucimberto Valansuela de Barros     27    915960939 A                 NA
 3 66552658001 Jayne Leite Temperini               21    936222787 L                 NA
 4 67164673276 Marilene Corrêa Linhares            21    940270580 L                 NA
 5 27387358435 Renata Bilé Saldanha                21    987407812 L                 NA
 6 63721743598 Valentim Nespoli Figueiras          28    943177423 L                 NA
 7 12150713373 Benício Arlia Duarte                11    947685566 A                 NA
 8 59004304991 Amanda Lopes Neto                   21    931167881 A                 NA
 9 11125248785 Cláudio Peixoto Barellos            17    945271192 NA                NA
10 28235543435 Carmen da Sousa Linhares            22    909637937 L                 NA
```

Pacote dplyr

- A função **mutate_if** permite alterar dados de colunas. No exemplo, os nomes são colocados em maiúsculo pela função **str_to_upper** do pacote **stringr**.

```
cli <- clientes %>% mutate_if(is.character, str_to_upper)
```

```
> cli
# A tibble: 100 x 6
   CPF         Nome                       DDD   Telefone  Classificacao Divida
   <chr>       <chr>                      <chr> <chr>     <chr>           <int>
 1 10244521310 MARGARIDA PALMAS STUTZ     12    959128774 L                  NA
 2 11125248785 CLÁUDIO PEIXOTO BARELLOS   17    945271192 NA                 NA
 3 12150713373 BENÍCIO ARLIA DUARTE       11    947685566 A                  NA
 4 14436831590 TEREZA PIRES BARBOSA       22    994053970 A                  NA
 5 14768110401 JOSÉ XIMENES VARGAS        11    334940412 NA                 NA
 6 15271527409 ALBA KNUPP BARRADA         22    931442914 NA                 NA
 7 15445645207 MARISA DE PADUA DA SOUSA   21    911021168 L                  NA
 8 15796045440 MARLENE SILVA MEYER        19    991341440 L                  NA
 9 18112077231 JÚLIO ROSA VIANA           22    917547560 L                  NA
10 19167287409 LINA DE LIMA UCHÔA         13    925186816 L                  NA
# ... with 90 more rows
```

Pacote dplyr

- A função **mutate_if** permite alterar dados de colunas. No exemplo, se o tipo da coluna for **integer** ela é convertida para **double**.

```
cli <- clientes %>% mutate_if(is.integer, as.double)
```

```
> cli
# A tibble: 100 x 6
   CPF         Nome                       DDD   Telefone  Classificacao Divida
   <chr>       <chr>                      <chr> <chr>     <chr>          <dbl>
 1 10244521310 Margarida Palmas Stutz     12    959128774 L                 NA
 2 11125248785 Cláudio Peixoto Barellos   17    945271192 NA                NA
 3 12150713373 Benício Arlia Duarte       11    947685566 A                 NA
 4 14436831590 Tereza Pires Barbosa       22    994053970 A                 NA
 5 14768110401 José Ximenes Vargas        11    334940412 NA                NA
 6 15271527409 Alba Knupp Barrada         22    931442914 NA                NA
 7 15445645207 Marisa de Padua da Sousa   21    911021168 L                 NA
 8 15796045440 Marlene Silva Meyer        19    991341440 L                 NA
 9 18112077231 Júlio Rosa Viana           22    917547560 L                 NA
10 19167287409 Lina de Lima Uchôa         13    925186816 L                 NA
# ... with 90 more rows
```

Pacote dplyr

- A função **replace_na** permite alterar dados de colunas. No exemplo, se o valor do campo Classificacao for **NA** ele assume o valor **'N'** e se o valor do campo Divida for **NA** ele assume o valor **0**.

```
cli <- clientes %>% replace_na(replace = list(Classificacao = 'N', Divida = 0))
```

```
> cli
# A tibble: 100 x 6
   CPF         Nome                      DDD   Telefone  Classificacao Divida
   <chr>       <chr>                     <chr> <chr>     <chr>           <dbl>
 1 10244521310 Margarida Palmas Stutz    12    959128774 L                   0
 2 11125248785 Cláudio Peixoto Barellos  17    945271192 N                   0
 3 12150713373 Benício Arlia Duarte      11    947685566 A                   0
 4 14436831590 Tereza Pires Barbosa      22    994053970 A                   0
 5 14768110401 José Ximenes Vargas       11    334940412 N                   0
 6 15271527409 Alba Knupp Barrada        22    931442914 N                   0
 7 15445645207 Marisa de Padua da Sousa  21    911021168 L                   0
 8 15796045440 Marlene Silva Meyer       19    991341440 L                   0
 9 18112077231 Júlio Rosa Viana          22    917547560 L                   0
10 19167287409 Lina de Lima Uchôa        13    925186816 L                   0
# ... with 90 more rows
```