O pacote `dplyr`

Vítor Wilher

Cientista de Dados | Mestre em Economia

Plano de Voo

Introdução

Carregando o pacote e *overwriting*

Explorando dados

Básico do `dplyr` e de lógica

Combinando operações múltiplas com o Pipe

Sumários

Desagrupar dados

Introdução

Parte significante do trabalho de análise de dados envolve trabalhar bases brutas e transformá-las em formatos mais interessantes. O R já vem com ferramentas para isso, mas elas não são tão eficientes quanto as disponibilizadas no pacote `dplyr`. A ideia do `dplyr` é oferecer uma *gramática dos dados*, uma maneira concisa e clara de manipulá-los. Ele é parte do mais amplo `tidyverse`, do qual também faz parte o `ggplot2`.

Como temos visto, o `tidyverse` é uma coleção de pacotes com ferramentas úteis para tratar, transformar, analisar e visualizar dados, muitos com uma filosofia comum. O `dplyr` é um dos pacotes mais populares do `tidyverse` e podemos carrega-lo sozinho ou com o resto dos pacotes.[1]

[1] Essa parte do curso é baseada em Grolemund and Wickham [2017].

Carregando o pacote e *overwriting*

Vamos carregar uma base de dados com vôos de Nova Iorque e o pacote `dplyr` (carregando junto o `tidyverse`). Caso você não tenha algum dos pacotes ainda, é só rodar o código `install.packages("nome do pacote")` que o R faz isso por você. Para carregarmos os pacotes, basta usar a função `library()`

```
library(nycflights13)
library(tidyverse)
```

Observe que o `dplyr` tem conflitos com R base, no sentido de que duas funções que já existem no pacote `stats`, que vem pré-carregado no R, são trocadas para as que o `dplyr` provê com o mesmo nome, são `lag()` e `filter()`. Caso você queira usar uma delas na versão base, vai precisar especificar que é do pacote `stats` escrevendo: `stats::lag()` e `stats::filter()`.

Explorando dados

`flights` vai carregar a base com os vôos:

```
flights
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

Ao rodarmos uma linha de código com um objeto de classe `data.frame`, normalmente temos de volta do R um resumo do objeto. Para ve-lo inteiro, use a função `View()` - atente para a letra maiúscula.

Básico do dplyr e de lógica

`filter()` te permite selecionar subconjuntos dos seus dados baseado em seus valores. O primeiro argumento é *sempre* um objeto `data.frame`, os subsequentes são argumentos lógicos que selecionem o que você quer:

```
filter(flights, month == 1, day == 1)
```

```
## # A tibble: 842 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # ... with 832 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

Básico do `dplyr` e de lógica

Observe que a saída dessa função é um objeto `data.frame` e o R entendeu como se tivéssemos escrito o nome desse objeto. Para armazena-lo precisamos fazer como sempre:

```
dados.filtrados <- filter(flights, month == 1, day == 1)
```

É especialmente importante tomar cuidado. Ao testar *igualdade*, sempre usamos o operador ==. Em programação, é importante ler o sinal de = como "é" e o sinal == como "igual". Podemos querer vôos que partiram em um mês e em outro escolhido. Não há problema, usamos o sinal |, a barra vertical, que deve ser lido como "ou".

Básico do dplyr e de lógica

```
filter(flights, month == 11 | month == 12)
```

```
## # A tibble: 55,403 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013    11     1        5           2359         6      352
##  2  2013    11     1       35           2250       105      123
##  3  2013    11     1      455            500        -5      641
##  4  2013    11     1      539            545        -6      856
##  5  2013    11     1      542            545        -3      831
##  6  2013    11     1      549            600       -11      912
##  7  2013    11     1      550            600       -10      705
##  8  2013    11     1      554            600        -6      659
##  9  2013    11     1      554            600        -6      826
## 10  2013    11     1      554            600        -6      749
## # ... with 55,393 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

Básico do `dplyr` e de lógica

Observe que o código *precisa* estar assim, pois `filter(flights, month == 11 | 12)` é interpretado como um teste lógico. A firmativa `11 | 12` é lida como verdadeira, então recebe o valor lógico de `TRUE`, que o R prontamente lê como 1 e entende que você está se referindo ao mês 1, janeiro. Para evitar isso, podemos usar o operador `%in%`.

```
nov_dec <- filter(flights, month %in% c(11, 12))
```

Básico do dplyr e de lógica

Podemos também usar o operador !, que nota negação:

```
filter(flights, !(arr_delay > 120 | dep_delay > 120))
```

```
## # A tibble: 316,050 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # ... with 316,040 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

Básico do dplyr e de lógica

```
filter(flights, arr_delay <= 120, dep_delay <= 120)
```

```
## # A tibble: 316,050 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # ... with 316,040 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

Arrumando dados com `arrange()`

`arrange()` funciona de maneira similar, mas ao invés de escolher pedaços dos dados, altera sua ordem. Alimentamos sempre um objeto `data.frame` e depois dizemos - em ordem - quais variáveis devem ser usadas para ordenação:

```
arrange(flights, year, month, day)
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

Arrumando dados com `arrange()`

Podemos usar `desc` para ordem descendente:

```
arrange(flights, desc(arr_delay))
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     9      641            900      1301     1242
##  2  2013     6    15     1432           1935      1137     1607
##  3  2013     1    10     1121           1635      1126     1239
##  4  2013     9    20     1139           1845      1014     1457
##  5  2013     7    22      845           1600      1005     1044
##  6  2013     4    10     1100           1900       960     1342
##  7  2013     3    17     2321            810       911      135
##  8  2013     7    22     2257            759       898      121
##  9  2013    12     5      756           1700       896     1058
## 10  2013     5     3     1133           2055       878     1250
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

Selecionando colunas com `select()`

É comum trabalhar com bases de dados que tenham centenas ou mesmo milhares de variáveis. Para isso podemos usar `select()` e simplificar a tarefa em mãos.

```
# selecionando colunas por nome
select(flights, year, month, day)
```

```
## # A tibble: 336,776 x 3
##     year month   day
##    <int> <int> <int>
##  1  2013     1     1
##  2  2013     1     1
##  3  2013     1     1
##  4  2013     1     1
##  5  2013     1     1
##  6  2013     1     1
##  7  2013     1     1
##  8  2013     1     1
##  9  2013     1     1
## 10  2013     1     1
## # ... with 336,766 more rows
```

Selecionando colunas com `select()`

```
# selecionando por grupo
select(flights, year:day)
```

```
## # A tibble: 336,776 x 3
##     year month   day
##    <int> <int> <int>
##  1  2013     1     1
##  2  2013     1     1
##  3  2013     1     1
##  4  2013     1     1
##  5  2013     1     1
##  6  2013     1     1
##  7  2013     1     1
##  8  2013     1     1
##  9  2013     1     1
## 10  2013     1     1
## # ... with 336,766 more rows
```

Selecionando colunas com `select()`

Existem várias funções úteis que combinam com `select()`:

- `starts_with("abc")` pega somente colunas com nomes que comecem com "abc"
- `ends_with("xyz")` faz o mesmo, mas para colunas que terminam da maneira entre parênteses
- `contains("ijk")` faz isso com colunas que tenham em qualquer parte de seus nomes o termo entre parênteses
- `num_range("x", 1:3)` seleciona as variáveis `x1,x2,x3`". Poderíamos alterar a amplitude dos números e do termo entre parênteses para nossas necessidades
- Para renomear variáveis usamos `rename()`

Selecionando colunas com `select()`

```
rename(flights, tail_num = tailnum)
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tail_num <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

Adicionando variáveis com `mutate()`

É comum precisar *criar* variáveis e podemos fazer isso comodamente com `mutate()`, que sempre irá adicionar a variável que especificarmos ao final do `data.frame`. Vamos gerar um objeto desssa classe, menor, e depois introduzir duas variáveis, `gain` que será a diferença dos atrasos de partida e chegada e `speed`, a velocidade média do vôo.

Adicionando variáveis com `mutate()`

```
base <- select(flights,
 year:day,
 ends_with("delay"),
 distance,
 air_time)
mutate(base,
 gain = arr_delay - dep_delay,
 speed = distance / air_time * 60)
```

```
## # A tibble: 336,776 x 9
##     year month   day dep_delay arr_delay distance air_time  gain speed
##    <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl>
##  1  2013     1     1         2        11     1400      227     9  370.
##  2  2013     1     1         4        20     1416      227    16  374.
##  3  2013     1     1         2        33     1089      160    31  408.
##  4  2013     1     1        -1       -18     1576      183   -17  517.
##  5  2013     1     1        -6       -25      762      116   -19  394.
##  6  2013     1     1        -4        12      719      150    16  288.
##  7  2013     1     1        -5        19     1065      158    24  404.
##  8  2013     1     1        -3       -14      229       53   -11  259.
##  9  2013     1     1        -3        -8      944      140    -5  405.
## 10  2013     1     1        -2         8      733      138    10  319.
## # ... with 336,766 more rows
```

Adicionando variáveis com `mutate()`

Se você quer *somente* as variáveis geradas, então use `transmute()`:

```
transmute(flights,
 gain = arr_delay - dep_delay,
 hours = air_time / 60)
```

```
## # A tibble: 336,776 x 2
##     gain hours
##    <dbl> <dbl>
##  1     9 3.78
##  2    16 3.78
##  3    31 2.67
##  4   -17 3.05
##  5   -19 1.93
##  6    16 2.5
##  7    24 2.63
##  8   -11 0.883
##  9    -5 2.33
## 10    10 2.3
## # ... with 336,766 more rows
```

Os mesmos truques que operadores que usamos para a calculadora do R também valem dentro das funções `mutate()` e `transmute()`.

Sumários com `summarise()`

O último verbo importante da gramática de dados do `dplyr` é `summarise()`. Ele essencialmente cria resumos do dataframe que estamos usando:

```
summarise(flights, delay = mean(dep_delay, na.rm = TRUE))
```

```
## # A tibble: 1 x 1
##   delay
##   <dbl>
## 1  12.6
```

Sumários com `summarise()`

Note que o argumetno `na.rm` da função `mean()` - que calcula médias - serve para fazer o R ignorar observações vazias ou faltantes, os `NA`. Isso é importante para a função `mean()`, que retorna erro quando lida com `NA`.

Sumários com `summarise()`

A função `summarise()` fica excepcionalmente poderosa quando combinada com `group_by()`, que agrupa os dados.

```
summarise(group_by(flights, year, month, day), delay = mean(dep_delay, na.rm = TRUE))
```

```
## # A tibble: 365 x 4
## # Groups:   year, month [?]
##     year month   day delay
##    <int> <int> <int> <dbl>
##  1  2013     1     1 11.5
##  2  2013     1     2 13.9
##  3  2013     1     3 11.0
##  4  2013     1     4  8.95
##  5  2013     1     5  5.73
##  6  2013     1     6  7.15
##  7  2013     1     7  5.42
##  8  2013     1     8  2.55
##  9  2013     1     9  2.28
## 10  2013     1    10  2.84
## # ... with 355 more rows
```

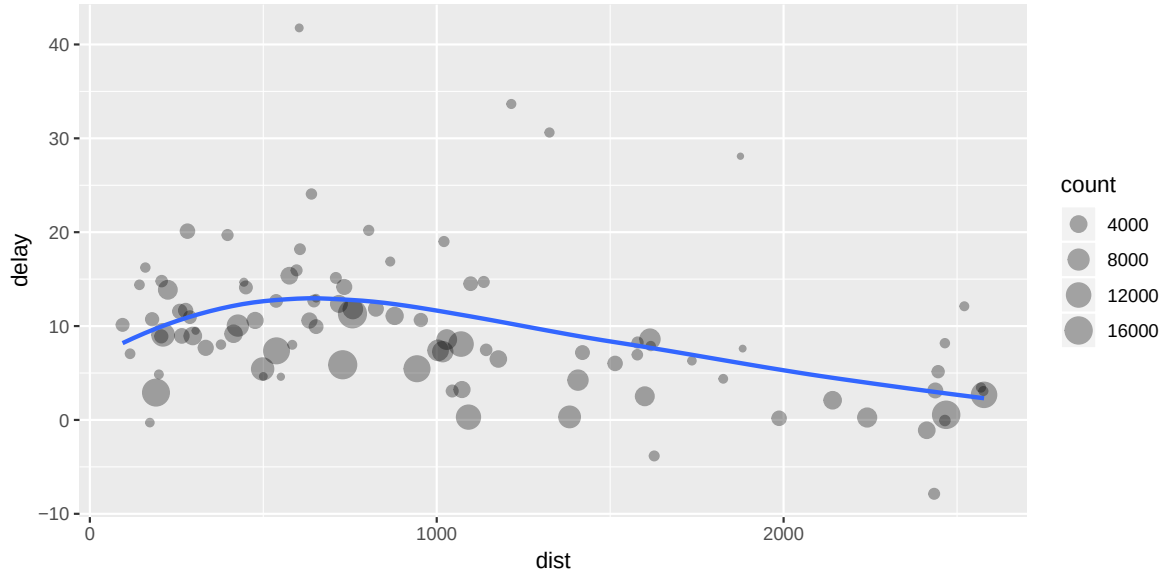
Combinando operações múltiplas com o Pipe

Imagine que estamos procurando uma relação entre algumas variáveis dos dados:

```
por_distancia <- group_by(flights, dest)
atraso <- summarize(por_distancia,
 count = n(), dist = mean(distance, na.rm = TRUE),
 delay = mean(arr_delay, na.rm = TRUE))
delay <- filter(atraso, count > 20, dest != "HNL")
```

Combinando operações múltiplas com o Pipe

```
ggplot(data = delay, mapping = aes(x = dist, y = delay)) +
 geom_point(aes(size = count), alpha = 1/3) +
 geom_smooth(se = FALSE)
```



Combinando operações múltiplas com o Pipe

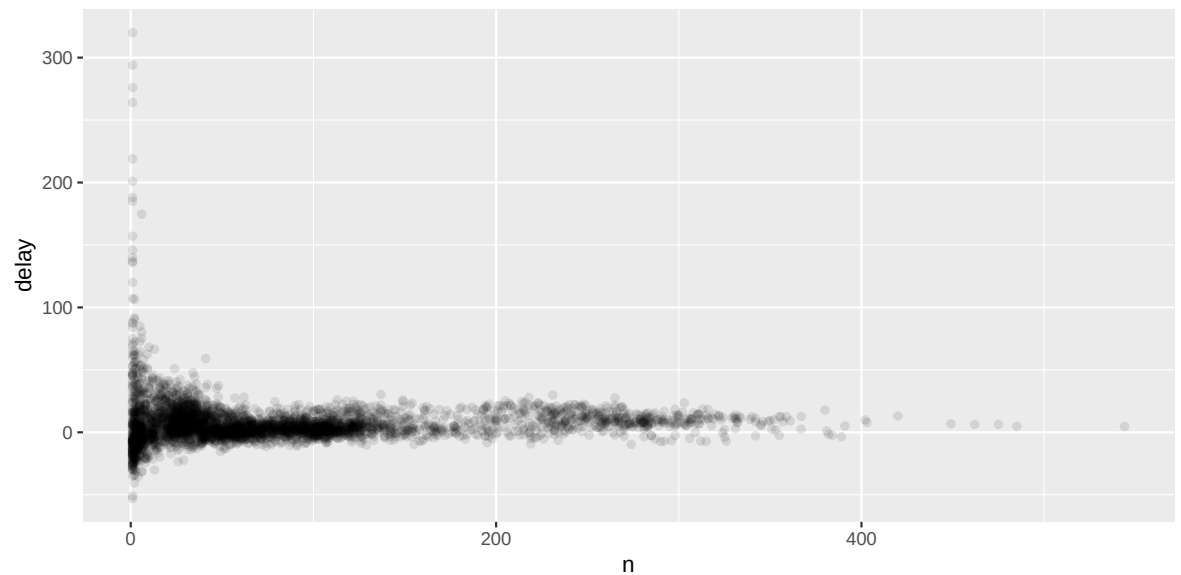
Fizemos uma sequência de passos grande e facilmente poderíamos ter errado algo no caminho. Além de que, qualquer alteração em uma linha de código provavelmente vai exigir que se altere em outras. Resolvemos isso com o operador %>%, o Pipe. Entenda ele como um cano, que "engata" funções.

```
atrasos <- flights %>%
 group_by(dest) %>%
 summarize(count = n(), dist = mean(distance, na.rm = TRUE), delay = mean(arr_delay, na.rm = TRUE)) %>%
 filter(count > 20, dest != "HNL")
```

Combinando operações múltiplas com o Pipe

Temos um código muito mais legível e rápido. Note que o pipe deve ser posto sempre como o sinal positivo em gráficos do `ggplot2`. Voltando aos dados, podemos querer cruzar atrasos com números de vôos no dia.

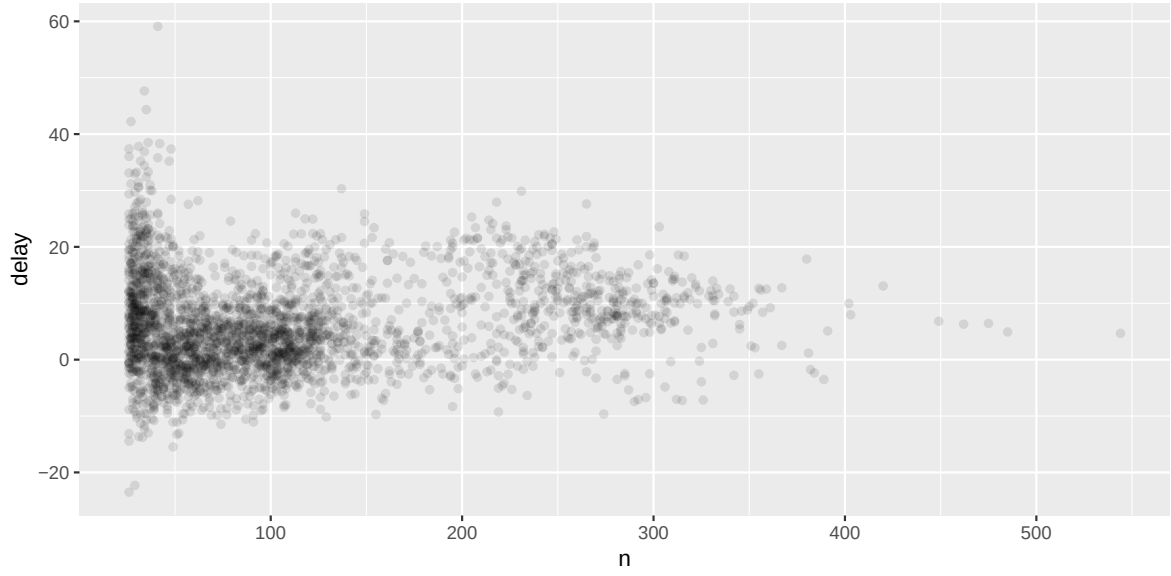
```
nao_cancelados <- flights %>%
 filter(!is.na(dep_delay), !is.na(arr_delay))
atrasos <- nao_cancelados %>%
 group_by(tailnum) %>%
 summarize(delay = mean(arr_delay, na.rm = TRUE),
 n = n())
ggplot(data = atrasos, mapping = aes(x = n, y = delay)) +
 geom_point(alpha = 1/10)
```



Combinando operações múltiplas com o Pipe

Observe que talvez não seja muito interessante manter na nossa análise exploratória dados de dias com pouquíssimos vôos - já que tendem a ser anômalos. Com um pipe, isso vira uma breve alteração no código

```
atrasos %>%
 filter(n > 25) %>%
 ggplot(mapping = aes(x = n, y = delay)) +
 geom_point(alpha = 1/10)
```



Sumários

Algumas funções são muito úteis para sumários:

- `IQR()` dá o intervalor interquartil
- `min()` e `max()` dão os valores mínimo e máximo da variável
- `sd()` dá o desvio-padrão
- `meadian()` dá a mediana
- `mad()` dá o desvio absoluto médio
- `var()` dá a variância

Desagrupar dados

Por fim, podemos querer *desagrupar* dados que vieram agrupados por `group_by()`. Basta usar `ungroup()`:

```
diario <- group_by(flights, year, month, day)
diario %>%
 ungroup() %>%
  summarise(flights = n())
```

```
## # A tibble: 1 x 1
##   flights
##     <int>
## 1  336776
```

G. Grolemund and H. Wickham. *R for Data Science*. O'Reilly Media, 2017.