Trading system com tidyquant 1

Luan Cézari Maria

29/04/2020

O TIDYQUANT

Tidyquant é uma biblioteca que integra diversos recursos para coletar, analisar e modelar dados financeiros em formato tibble, permitindo uma interação perfeita com o tidyverse. Os quatro principais pacotes utilizados na análise financeira com R e que estão integrados no tidyquant são:

xts - pacote usado para lidar com séries temporais. O pacote subjacente ao mesmo é o zoo, que também está integrado.

quantmod - pacote designado para importar, manipular e modelar dados quantitativos.

TTR - pacote que inclui várias funções para computar indicadores técnicos.

PerformanceAnalytics - pacote que inclui um conjunto de funções econométricas para análise de performance e risco. O pacote analisa retornos, não preços.

Primeiramente precisamos carregar os pacotes **tidyverse** e **tidyquant**. Fazemos isso usando a já conhecida função **library()**

```
library(tidyquant)
library(tidyverse)
```

Caso algum dos pacotes não esteja baixado será necessário baixá-lo usando a função **install.packages()**

IMPORTANDO DADOS

O pacote tidyquant nos permite importar cotações diárias e intradiária, além de dados econômicos, de diversas fontes diferentes. Nesse curso usaremos as cotações obtidas a partir da API da Alpha Vantage. Para acessar todas as fontes de dados disponíveis utilize o código:

```
tq_get_options()
```

```
##  [1] "stock.prices"       "stock.prices.japan" "economic.data"
##  [4] "quandl"             "quandl.datatable"   "tiingo"
##  [7] "tiingo.iex"         "tiingo.crypto"      "alphavantager"
## [10] "alphavantage"       "rblpapi"
```

A API da alpha vantage

Para obter uma API da alpha vantage basta acessar o link https://www.alphavantage.co/support/#api-key e preencher o formulário. A versão gratuita da API tem um limite de 5 requests por minuto e 500 requests por dia. Para importar um volume maior de dados é necessário contratar a API premium.

A API nos permite importar dados intradiários, diários, semanais e mensais de todos os mercados de ações, do forex e do mercado de criptomoedas. Nesse curso abordaremos apenas como trabalhar com ações negociadas na B3, para aprender a importar dados de outros mercados acesse a documentação da API: https://www.alphavantage.co/documentation/.

Importando dados com a API da apha vantage

Primeiramente precisamos configurar a API no RStudio usando o código:

```
av_api_key("9VUXH0JCQ4C5U56K")
```

A função **tq_get()** é utilizada para importar dados históricos no tidyquant. Dentro dela precisamos informar o ticker do ativo, a fonte dos dados, o argumento que identifica o tipo de dado a ser importado (no caso especifico da API alpha vantage), o tamanho do conjunto de dados a importar e o intervalo(no caso de dados intradiários). Um exemplo da função aplicada com todos os argumentos pode ser vista a seguir:

```
tq_get(c("PETR4.SA", "MGLU3.SA"), #ticker do ativo
       get = "alphavantage", #fonte dos dados
       av_fun = "TIME_SERIES_INTRADAY", #argumento com o tipo de dados
       outputsize = "full", #o tamanho do conjunto de dados a importar
       interval = "60min") #intervalo
```

```
## # A tibble: 822 x 7
##    symbol   timestamp            open  high   low close  volume
##    <chr>    <dttm>              <dbl> <dbl> <dbl> <dbl>   <dbl>
##  1 PETR4.SA 2020-02-07 08:00:00  29    29.3  28.8  29.2 9501810
##  2 PETR4.SA 2020-02-07 09:00:00  29.2  29.4  29.1  29.2 7174935
##  3 PETR4.SA 2020-02-07 10:00:00  29.2  29.2  28.8  29.1 8309600
##  4 PETR4.SA 2020-02-07 11:00:00  29.1  29.3  29.0  29.2 2820980
##  5 PETR4.SA 2020-02-07 12:00:00  29.2  29.3  29.1  29.3 4257055
##  6 PETR4.SA 2020-02-07 13:00:00  29.3  29.3  29.1  29.1 3340840
##  7 PETR4.SA 2020-02-10 08:00:00  28.9  29.1  28.8  29.1 4120205
##  8 PETR4.SA 2020-02-10 09:00:00  29.1  29.1  28.7  28.7 4980575
##  9 PETR4.SA 2020-02-10 10:00:00  28.7  29.0  28.7  28.9 3377730
## 10 PETR4.SA 2020-02-10 11:00:00  28.9  29    28.8  28.9 2171580
## # ... with 812 more rows
```

Note que pelo fato do pacote estar integrado ao tidyverse é possível colocar o vetor contendo o ticker do lado de fora da função seguido por um pipe, da seguinte forma:

```
PETR4 <- "PETR4.SA" %>%
  tq_get(get = "alphavantage",
         av_fun = "TIME_SERIES_DAILY",
         outputsize = "full")
```

Os outros possíveis valores aplicáveis aos argumentos dentro do contexto da alpha vantage são brevementes descritos a seguir:

- av_fun
  - **TIME_SERIES_INTRADAY** importa dados intradiários dos mercados de ações.
  - **TIME_SERIES_DAILY** importa dados diários dos mercados de ações.
  - **TIME_SERIES_DAILY_ADJUSTED** importa dados diários dos mercados de ações com fechamento ajustado, distribuição de dividendos e desdobramento de ações.
  - **TIME_SERIES_WEEKLY** importa dados semanais dos mercados de ações
  - **TIME_SERIES_WEEKLY_ADJUSTED** importa dados semanais dos mercados de ações com fechamento ajustado, distribuição de dividendos e desdobramento de ações.
  - **TIME_SERIES_MONTHLY** importa dados mensais dos mercados de ações
  - **TIME_SERIES_MONTHLY_ADJUSTED** importa dados mensais dos mercados de ações com fechamento ajustado, distribuição de dividendos e desdobramento de ações.
- outputsize
  - **compact** retorna apenas os 100 últimos dados. [PADRÃO]
  - **full** retorna todos os dados disponíveis.
- interval (apenas para dados intradiários)
  - **1min** candles de 1 minuto
  - **5min** candles de 5 minutos
  - **15min** candles de 15 minutos
  - **30min** candles de 30 minutos
  - **60min** candles de 60 minutos

  Podemos buscar o código dos títulos disponíveis através do link https://www.alphavantage.co/query?function=SYMBOL_SEARCH&keywords=COLOQUEOTERMOAQUI&apikey=$key. COM A API CARREGADA basta substituir o termo a pesquisar logo a frente de keywords= no lugar de *COLOQUEOTERMOAQUI*.

Associando informações aos dados importados

Por vezes desejamos associar junto aos dados importados outras informações relativas às empresas que serão úteis futuramente no nosso trabalho. Podemos facilmente associá-las no processo de importação criando um tibble contendo os tickers dos ativos desejados na primeira coluna e as informações adicionais nas demais. A seguir demonstramos o uso dessa prática associando o setor de atuação aos dados de cada empresa:

```
tickers <- tibble(acoes = c("PETR4.SA", "ITUB4.SA", "MGLU3.SA"),
                  setor = c("petróleo e gás", "serviços financeiros", "Serviços"))

tickers %>%
  tq_get(get = "alphavantage",
         av_fun = "TIME_SERIES_WEEKLY",
         outputsize = "full")
```

```
## # A tibble: 2,556 x 8
##    acoes    setor          timestamp   open  high   low close  volume
##    <chr>    <chr>          <date>     <dbl> <dbl> <dbl> <dbl>   <dbl>
##  1 PETR4.SA petróleo e gás 2000-05-19  6.39  6.47  6.36  6.42 2269121
##  2 PETR4.SA petróleo e gás 2000-05-26  6.42  6.44  6.33  6.39 2907902
##  3 PETR4.SA petróleo e gás 2000-06-02  6.39  6.43  6.32  6.38 2413537
##  4 PETR4.SA petróleo e gás 2000-06-09  6.38  6.41  6.27  6.33 2310050
##  5 PETR4.SA petróleo e gás 2000-06-16  6.33  6.37  6.26  6.33 2511500
##  6 PETR4.SA petróleo e gás 2000-06-23  6.33  6.43  6.30  6.42 3260295
##  7 PETR4.SA petróleo e gás 2000-06-30  6.42  6.44  6.32  6.39 2525502
```

```
##  8 PETR4.SA petróleo e gás 2000-07-07  6.39  6.45  6.30  6.36 2857212
##  9 PETR4.SA petróleo e gás 2000-07-14  6.36  6.39  6.19  6.27 1879813
## 10 PETR4.SA petróleo e gás 2000-07-21  6.27  6.38  6.26  6.38 2574185
## # ... with 2,546 more rows
```

Lidando com valores faltantes

Um problema relativamente comum de acontecer não apenas na análise de séries financeiras mas em ciência de dados em geral é a presença de valores faltantes. Após importar os dados é importante analisar a consistência do mesmo pois dados faltantes podem distorcer o resultado da sua análise e levar a perda financeira. No caso dos dados importados pela API da Alpha Vantage os dados faltantes são substituidos por 0 por padrão. Isso pode levar a distorção quando aplica-se fórmulas matemáticas por isso vamos substituir os 0's por NA. Para visualizar se há algum 0 no nosso conjunto de dados podemos usar a função:

```
PETR4 %>% summarize_all(~sum(.==0))
```

```
## # A tibble: 1 x 7
##   symbol timestamp  open  high   low close volume
##    <int>     <int> <int> <int> <int> <int>  <int>
## 1      0         0     0     0     0     0      0
```

Uma vez identificados os valores faltantes podemos substituí-los todos de uma vez utilizando a função:

```
PETR4[PETR4 == 0] <- NA
```

O resultado pode ser observado aplicando novamente a função usada para identificar os 0:

```
PETR4 %>% summarise_all(~sum(.==0, na.rm=TRUE))
```

```
## # A tibble: 1 x 7
##   symbol timestamp  open  high   low close volume
##    <int>     <int> <int> <int> <int> <int>  <int>
## 1      0         0     0     0     0     0      0
```

MANIPULANDO DADOS

Existe um amplo número de funções feitas para trabalhar com séries temporais que funcionam dentro do ecossistema do tidyquant. Essa seção pretende apresentar esse conjunto de funções e demonstrar como elas funcionam junto com as duas funções fundamentais **tq_transmute()** e **tq_mutate()**.

tq_transmute() e tq_mutate()

As funções **tq_transmute()** e **tq_mutate()** funcionam de maneira similar às funções **transmute()** e **mutate()** do pacote **dplyr** e são fundamentais no processo de modelagem e análise.

A função **tq_transmute()** aplica uma função em um coluna e retorna um novo dataframe com o resultado. Uma diferença fundamental da função **tq_transmute()** para a função **transmute()** do pacote **dplyr** é que a primeira permite retornar um dataframe com um número diferente de linhas do dataframe base. Essa característica é fundamental para funções que envolvem mudança de período. A seguir temos um exemplo da função **tq_transmute()** sendo usada para converter um conjunto de fechamentos diários em fechamentos mensais:

```
PETR4 %>%
  tq_transmute(select = "close", #seleciona em qual coluna será aplicada a transformação
               mutate_fun = to.monthly, #seleciona a função a ser aplicada na coluna
               indexAt = "lastof") #argumento específico da função aplicada
```

```
## # A tibble: 241 x 2
##    timestamp  close
##    <date>     <dbl>
##  1 2000-05-31  6.59
##  2 2000-06-30  6.40
##  3 2000-07-31  6.38
##  4 2000-08-31  6.33
##  5 2000-09-30  6.37
##  6 2000-10-31  6.45
##  7 2000-11-30  5.92
##  8 2000-12-31  5.72
##  9 2001-01-31  6.81
## 10 2001-02-28  6.86
## # ... with 231 more rows
```

Nesse exemplo o argumento **select** nos permite selecionar em qual coluna será aplicada a transformação. O argumento **mutate_fun** determina qual função será aplicada a coluna (as funções existentes serão apresentadas mais a frente). Argumentos adicionais especificos das funções aplicadas podem ser adicionadas. No exemplo o argumento **indexAt** determina qual data será usada como a primeira do período.

A função **tq_mutate()** adiciona ao dataframe base uma nova coluna ou um conjunto de colunas com o resultado da aplicação da função escolhida. A seguir temos um exemplo de aplicação da função **tq_mutate()** sendo usada para aplicar o indicador MACD nos fechamentos, note como mudamos o nome das colunas usando o argumento **col_rename**:

```
PETR4 <- PETR4 %>%
  tq_mutate(select = "close",
            mutate_fun = MACD,
            col_rename = c("MACD", "Sinal"))
tail(PETR4)
```

```
## # A tibble: 6 x 9
##   symbol   timestamp   open  high   low close   volume   MACD  Sinal
##   <chr>    <date>     <dbl> <dbl> <dbl> <dbl>    <dbl>  <dbl>  <dbl>
## 1 PETR4.SA 2020-04-29  17.8  18.5  17.5  18.2 94528400 -0.522 -3.01
## 2 PETR4.SA 2020-04-30  18.0  18.4  17.7  18.0 80263900  0.175 -2.38
## 3 PETR4.SA 2020-05-04  17.4  17.6  17.2  17.4 60268400  0.399 -1.82
## 4 PETR4.SA 2020-05-05  17.9  18.5  17.9  17.9 75043400  0.830 -1.29
## 5 PETR4.SA 2020-05-06  17.9  18.1  17.3  17.3 67937500  0.849 -0.862
## 6 PETR4.SA 2020-05-07  17.8  17.9  17.4  17.6 74488500  1.00  -0.489
```

tq_transmutate_xy() e tq_mutate_xy()

As funções **tq_transmute_xy()** e **tq_mutate_xy()** são variantes das funções **tq_transmute()** e **tq_mutate()** que nos permitem aplicar ao conjunto de dados funções que usem duas variáveis ao invés de uma. A única diferença no código é que usandos os argumentos **x** e **y** no lugar do argumento **select** para selecionar as variáveis. Um exemplo de aplicação é visto a seguir onde usamos a função **tq_transmute_xy()** para calcular a diferença percentual entre a abertura e o fechamento de cada período:

```
PETR4 %>%
  tq_transmute_xy(x = low,
                  y = high,
                  mutate_fun = Delt,
                  type = "arithmetic",
                  col_rename = "amplitude_diaria")
```

```
## # A tibble: 4,913 x 2
##    timestamp  amplitude_diaria
##    <date>                <dbl>
##  1 2000-05-08          0.00956
##  2 2000-05-09          0.00939
##  3 2000-05-10          0.0101
##  4 2000-05-11          0.00724
##  5 2000-05-12          0.0159
##  6 2000-05-15          0.0124
##  7 2000-05-16          0.0164
##  8 2000-05-17          0.00358
##  9 2000-05-18          0.00717
## 10 2000-05-19          0.00861
## # ... with 4,903 more rows
```

Nesse exemplo a função **Delt()** equivale ao delta na matemática que por convenção é usado para representar variação.

Funções compatíveis com tq_transmute() e tq_mutate()

Existe um conjunto muito grande de funções vindas dos pacotes **zoo**, **xts**, **quantmod**, **TTR** e **PerformanceAnalytics** que por padrão são compatíveis com as funções **tq_transmute()** e **tq_mutate()**. Para visualizar a estrutura de todas as funções compatíveis podemos aplicar a função **str()** na função **tq_transmute_fun_options()** da seguinte forma:

```
tq_transmute_fun_options() %>% str()
```

```
## List of 5
##  $ zoo                 : chr [1:14] "rollapply" "rollapplyr" "rollmax" "rollmax.default" ...
##  $ xts                 : chr [1:27] "apply.daily" "apply.monthly" "apply.quarterly" "apply.weekly" .
##  $ quantmod            : chr [1:25] "allReturns" "annualReturn" "ClCl" "dailyReturn" ...
##  $ TTR                 : chr [1:62] "adjRatios" "ADX" "ALMA" "aroon" ...
##  $ PerformanceAnalytics: chr [1:7] "Return.annualized" "Return.annualized.excess" "Return.clean" "Re
```

Funções do pacote zoo

Podemos acessar todas as funções do pacote zoo compatíveis através da função:

```
tq_transmute_fun_options()$zoo
```

```
##  [1] "rollapply"          "rollapplyr"         "rollmax"
##  [4] "rollmax.default"    "rollmaxr"           "rollmean"
##  [7] "rollmean.default"   "rollmeanr"          "rollmedian"
## [10] "rollmedian.default" "rollmedianr"        "rollsum"
## [13] "rollsum.default"    "rollsumr"
```

De uma forma geral as funções são:

- Funções roll apply:
  - Função genérica que aplica uma função numa janela móvel de tempo.
  - Forma: `rollapply(data, width, FUN, ..., by = 1, by.column = TRUE, fill = if (na.pad) NA, na.pad = FALSE, partial = FALSE, align = c("center", "left", "right"), coredata = TRUE)`.
  - Mais detalhes na documentação da função acessível através do comando **help()** ou **?**.

Função do pacote xts

Podemos acessar todas as funções disponíveis do pacote xts através da função:

```
tq_transmute_fun_options()$xts
```

```
##  [1] "apply.daily"     "apply.monthly"   "apply.quarterly" "apply.weekly"
##  [5] "apply.yearly"    "diff.xts"        "lag.xts"         "period.apply"
##  [9] "period.max"      "period.min"      "period.prod"     "period.sum"
## [13] "periodicity"     "to.daily"        "to.hourly"       "to.minutes"
## [17] "to.minutes10"    "to.minutes15"    "to.minutes3"     "to.minutes30"
## [21] "to.minutes5"     "to.monthly"      "to.period"       "to.quarterly"
## [25] "to.weekly"       "to.yearly"       "to_period"
```

De uma forma geral as funções são:

- Funções period e apply:
  - Aplicam uma função a um seguimento de tempo (por exemplo **max**, **min**, etc)
  - Forma: `apply.daily(x, FUN, ...)`
  - Opções inclusas: **apply.daily**, **apply.weekly**, **apply.monthly**, **apply.quarterly**, **apply.yearly**.
- Funções to.period:
  - Converte uma série temporal para outra com menor periodicidade (diario para mensal)
  - Forma: `to.period(x, period = 'months', k = 1, indexAt, name = NULL, OHLC = TRUE, ...)`.
  - Opções inclusas: **to.period**, **to.minutes**, **to.hourly**, **to.daily**, **to.weekly**, **to.monthly**, **to.quarterly**, **to.yearly**.
- **Nota:** o retorno das funções **to.period** e **to.monthly(hourly, daily, ...)** é diferente. A função **to.period** retorna o resultado no formato date enquanto to.months retorna no formato MON YYYY. É melhor utilizar a função **to.period** se você pretende trabalhar com as séries temporais usando o pacote **lubridate**.

Funções do pacote quantmod

Podemos acessar todas as funções do pacote quantmod compatíveis através da função:

```
tq_transmute_fun_options()$quantmod
```

```
##  [1] "allReturns"      "annualReturn"    "ClCl"            "dailyReturn"
##  [5] "Delt"            "HiCl"            "Lag"             "LoCl"
##  [9] "LoHi"            "monthlyReturn"   "Next"            "OpCl"
## [13] "OpHi"            "OpLo"            "OpOp"            "periodReturn"
## [17] "quarterlyReturn" "seriesAccel"     "seriesDecel"     "seriesDecr"
## [21] "seriesHi"        "seriesIncr"      "seriesLo"        "weeklyReturn"
## [25] "yearlyReturn"
```

De uma forma geral as funções são:

- Variação percentual (Delt) e função Lag:
  - Delt: `Delt(x1, x2 = NULL, k = 0, type = c("arithmetic", "log"))`
    - Variações do Delt: ClCl, HiCl, LoCl, LoHi, OpCl, OpHi, OpLo, OpOp
    - Recomendo o uso da função Delt ao invés das variações sempre
  - Lag: `Lag(x, k=1)` / Next: `Next(x, k=1)`
- Função periodReturn:
  - Nos da os retornos aritméticos ou logarítmicos de várias periodicidades incluindo diário, semanal, mensal, trimestral e anual.
  - Forma: `periodReturn(x, period = 'monthly', subset = NULL, type = 'arithmetic', leading = TRUE, ...)`

*Fuções series: + Retorna valores que descrevem uma série. + Opções inclusas: aumento/redução, aceleração/desaceleração e maxima/minima + Forma:`seriesHi(x), seriesIncr(x, thresh = 0, diff. = 1L), seriesAccel(x)`

Funções do pacote TTR

Podemos acessar todas as funções do pacote TTR disponíveis através da função:

```
tq_mutate_fun_options()$TTR
```

```
##  [1] "adjRatios"          "ADX"                "ALMA"
##  [4] "aroon"              "ATR"                "BBands"
##  [7] "CCI"                "chaikinAD"          "chaikinVolatility"
## [10] "CLV"                "CMF"                "CMO"
## [13] "DEMA"               "DonchianChannel"    "DPO"
## [16] "DVI"                "EMA"                "EMV"
## [19] "EVWMA"              "GMMA"               "growth"
## [22] "HMA"                "KST"                "lags"
## [25] "MACD"               "MFI"                "momentum"
## [28] "OBV"                "PBands"             "ROC"
## [31] "rollSFM"            "RSI"                "runCor"
## [34] "runCov"             "runMAD"             "runMax"
## [37] "runMean"            "runMedian"          "runMin"
## [40] "runPercentRank"     "runSD"              "runSum"
## [43] "runVar"             "SAR"                "SMA"
## [46] "SMI"                "SNR"                "stoch"
## [49] "TDI"                "TRIX"               "ultimateOscillator"
## [52] "VHF"                "VMA"                "volatility"
## [55] "VWAP"               "VWMA"               "wilderSum"
## [58] "williamsAD"         "WMA"                "WPR"
## [61] "ZigZag"             "ZLEMA"
```

De uma forma geral as funções são:

- Welles Wilder's Directional Movement Index:
  - `ADX(HLC, n = 14, maType, ...)`
- Bollinger Bands:
  - `BBands(HLC, n = 20, maType, sd = 2, ...)`
- Rate of Change / Momentum:
  - `ROC(x, n = 1, type = c("continuous", "discrete"), na.pad = TRUE)`: Rate of Change
  - `momentum(x, n = 1, na.pad = TRUE)`: Momentum
- Moving Averages (maType):
  - `SMA(x, n = 10, ...)`: Simple Moving Average
  - `EMA(x, n = 10, wilder = FALSE, ratio = NULL, ...)`: Exponential Moving Average
  - `DEMA(x, n = 10, v = 1, wilder = FALSE, ratio = NULL)`: Double Exponential Moving Average
  - `WMA(x, n = 10, wts = 1:n, ...)`: Weighted Moving Average
  - `EVWMA(price, volume, n = 10, ...)`: Elastic, Volume-Weighted Moving Average
  - `ZLEMA(x, n = 10, ratio = NULL, ...)`: Zero Lag Exponential Moving Average
  - `VWAP(price, volume, n = 10, ...)`: Volume-Weighted Moving Average Price
  - `VMA(x, w, ratio = 1, ...)`: Variable-Length Moving Average
  - `HMA(x, n = 20, ...)`: Hull Moving Average
  - `ALMA(x, n = 9, offset = 0.85, sigma = 6, ...)`: Arnaud Legoux Moving Average
- MACD Oscillator:
  - `MACD(x, nFast = 12, nSlow = 26, nSig = 9, maType, percent = TRUE, ...)`
- Relative Strength Index:
  - `RSI(price, n = 14, maType, ...)`
- runFun:
  - `runSum(x, n = 10, cumulative = FALSE)`: Retorna a soma em uma janela móvel de n períodos
  - `runMin(x, n = 10, cumulative = FALSE)`: Retorna a mínima em uma janela móvel de n períodos
  - `runMax(x, n = 10, cumulative = FALSE)`: Retorna a máxima em uma janela móvel de n períodos
  - `runMean(x, n = 10, cumulative = FALSE)`: Retorna a média em uma janela móvel de n períodos
  - `runMedian(x, n = 10, non.unique = "mean", cumulative = FALSE)`: Retorna a mediana em uma janela móvel de n períodos +`runCov(x, y, n = 10, use = "all.obs", sample = TRUE, cumulative = FALSE)`: Retorna covariâncias em uma janéla móvel de n períodos
  - `runCor(x, y, n = 10, use = "all.obs", sample = TRUE, cumulative = FALSE)`: Retorna correlações em uma janéla móvel de n períodos
  - `runVar(x, y = NULL, n = 10, sample = TRUE, cumulative = FALSE)`: Retorna variâncias em uma janela móvel de n períodos
  - `runSD(x, n = 10, sample = TRUE, cumulative = FALSE)`: Retorna desvios padrões em uma janela móvel de n períodos
  - `runMAD(x, n = 10, center = NULL, stat = "median", constant = 1.4826, non.unique = "mean", cumulative = FALSE)`: Retorna os desvios absolutos da média/mediana em uma janela móvel de n períodos
  - `wilderSum(x, n = 10)`: Retorna o Welles Wilder style weighted sum em uma janela móvel de n períodos

- Stochastic Oscillator / Stochastic Momentum Index:
  - `stoch(HLC, nFastK = 14, nFastD = 3, nSlowD = 3, maType, bounded = TRUE, smooth = 1, ...)`: Stochastic Momentum Index
  - `SMI(HLC, n = 13, nFast = 2, nSlow = 25, nSig = 9, maType, bounded = TRUE, ...)`:

Funções do pacote PerformanceAnalytics

Podemos acessar todas as funções compatívels do pacote PerformanceAnalytics através da função:

```
tq_mutate_fun_options()$PerformanceAnalytics
```

```
## [1] "Return.annualized"        "Return.annualized.excess"
## [3] "Return.clean"             "Return.cumulative"
## [5] "Return.excess"            "Return.Geltner"
## [7] "zerofill"
```

De uma forma geral as funções são:

- Return.annualized:
  - Seleciona os retornos em um período e transforma em retornos anualizados
- Return.annualized.excess:
  - Anualiza os retornos de um período e subtrai a taxa de juros livro de risco
- Return.clean:
  - Remove os outliers dos retornos
- Return.excess:
  - Subtrai a taxa de juros livre de risco dos retornos para obter o prêmio de risco
- zerofill:
  - Usado para substituir os NA por zeros

EXEMPLOS DE APLICAÇÃO

Exemplo 1: Usando periodReturn para converter preços em retornos

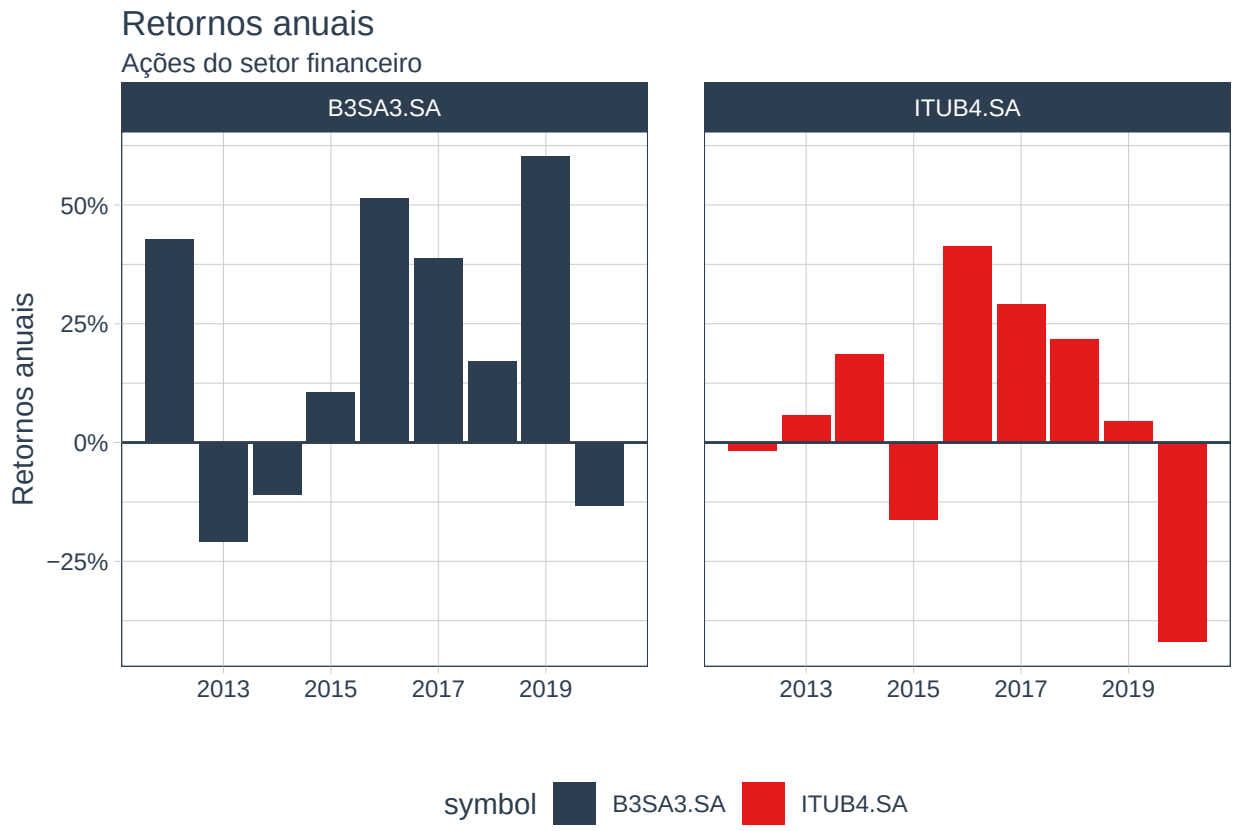
Importando os dados e calculando os retornos

```
# Gerando o vetor com as ações
acoes <- tq_get(c("B3SA3.SA", "ITUB4.SA", "BBDC4.SA", "ITSA4.SA"),
                get = "alphavantage",
                av_fun = "TIME_SERIES_DAILY",
                outputsize = "full")

# Calculando os retornos anuais dos fechamentos
retornos_anuais <- acoes %>%
  group_by(symbol) %>% #Agrupa as ações de acordo com o simbolo
```

```
  tq_transmute(select = close,
               mutate_fun = periodReturn,
               period = "yearly",
               type = "arithmetic")
retornos_anuais
```

```
## # A tibble: 31 x 3
## # Groups:   symbol [2]
##    symbol   timestamp  yearly.returns
##    <chr>    <date>              <dbl>
##  1 B3SA3.SA 2011-12-29         -0.188
##  2 B3SA3.SA 2012-12-28          0.429
##  3 B3SA3.SA 2013-12-30         -0.210
##  4 B3SA3.SA 2014-12-30         -0.110
##  5 B3SA3.SA 2015-12-30          0.107
##  6 B3SA3.SA 2016-12-29          0.515
##  7 B3SA3.SA 2017-12-29          0.387
##  8 B3SA3.SA 2018-12-28          0.171
##  9 B3SA3.SA 2019-12-30          0.603
## 10 B3SA3.SA 2020-05-07         -0.134
## # ... with 21 more rows
```

```
# Plotando o resultado em um gráfico usando ggplot2
retornos_anuais %>% filter(year(timestamp) >= 2012) %>% #seleciona os dados a partir de 2012
  ggplot(aes(x = year(timestamp), y = yearly.returns, fill = symbol)) +
  geom_col() + #seleciona o tipo de grafico
  geom_hline(yintercept = 0, color = palette_light()[[1]]) + #adiciona linha no 0%
  scale_y_continuous(labels = scales::percent) + #coloca o eixo y em porcentagem
  labs(title = "Retornos anuais",
       subtitle = "Ações do setor financeiro",
       y = "Retornos anuais", x = "") +
  facet_wrap(. ~ symbol, ncol = 2) + #gera a matriz de graficos por simbolo
  theme_tq() + #modifica o tema
  scale_fill_tq() #adiciona cores que harmonizam com o theme_tq
```

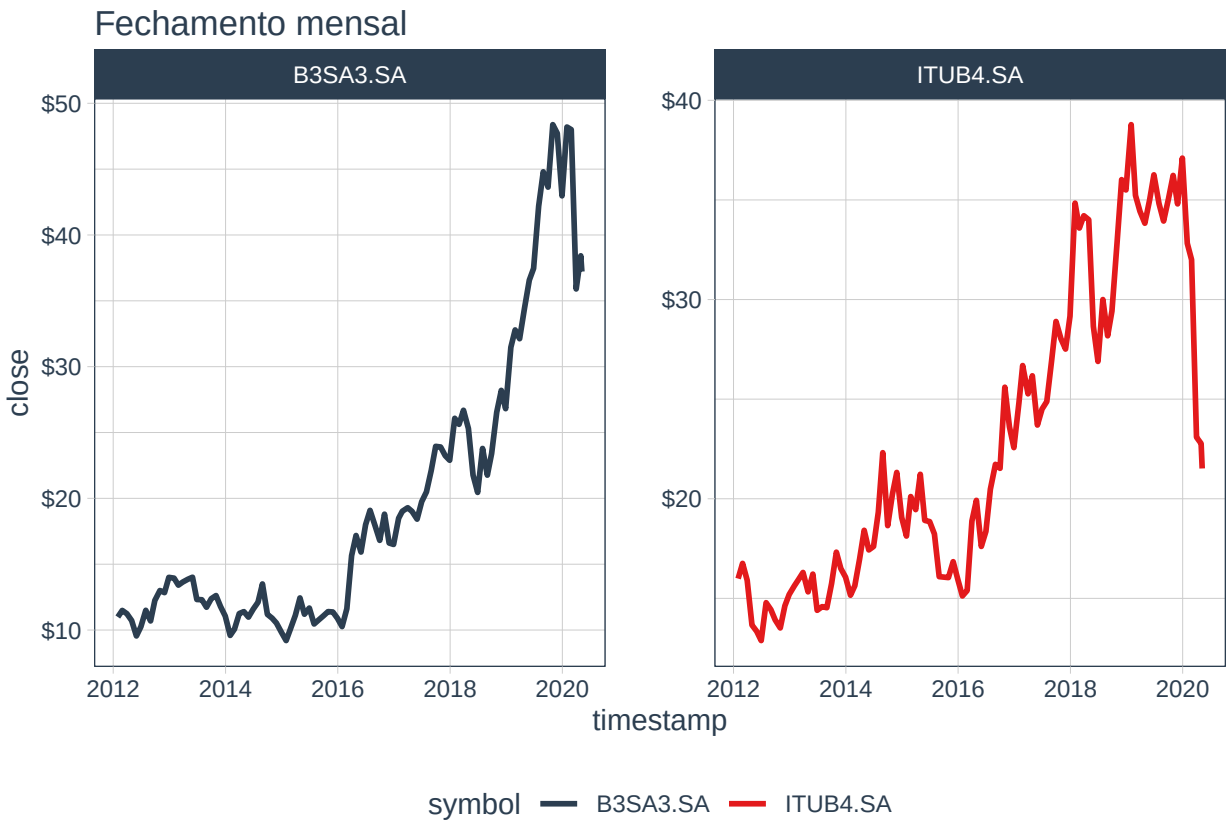


Exemplo 2: Usando to.period para mudar a periodicidade de diário para mensal

```
# Grafico de linha com fechamentos diarios
acoes %>% filter (year(timestamp) >= 2012) %>%
  group_by(symbol) %>%
  ggplot(aes(x = timestamp, y = close, color = symbol)) +
  geom_line(size = 1) +
  labs(title = "Fechamento diário") +
  facet_wrap(~ symbol, ncol = 2, scales = "free_y") +
  scale_y_continuous(labels = scales::dollar) +
  theme_tq() +
  scale_color_tq()
```


```
# Transformando dados diarios em mensais
acoes_mensal <- acoes %>%
  group_by(symbol) %>%
  tq_transmute(select = open:volume, #seleciona todas as colunas entre open e volume
               mutate_fun = to.period,
               period = "months")
```

```
# Grafico de linha com fechamentos mensais
acoes_mensal %>% filter(year(timestamp) >= 2012) %>%
  ggplot(aes(x = timestamp, y = close, color = symbol)) +
  geom_line(size = 1) +
  labs(title = "Fechamento mensal") +
  facet_wrap(~ symbol, ncol = 2, scales = "free_y") +
  scale_y_continuous(labels = scales::dollar) +
  theme_tq() +
  scale_color_tq()
```



Exemplo 3: Usando runCor para visualizar a correlação móvel dos retornos

Nesse exemplo calcularemos a correlação móvel entre o retorno mensal das ações das empresas selecionadas do mercado financeiro e o FIND11, ETF que imita o índice setorial do mercado financeiro.

```
# Calculando o retorno mensal das ações
retornos_mensais <- acoes %>%
  group_by(symbol) %>%
  tq_transmute(select = close,
               mutate_fun = periodReturn,
               period = "monthly",
               col_rename = "retornos_acoes") %>%
  filter(year(timestamp) >= 2012)
```

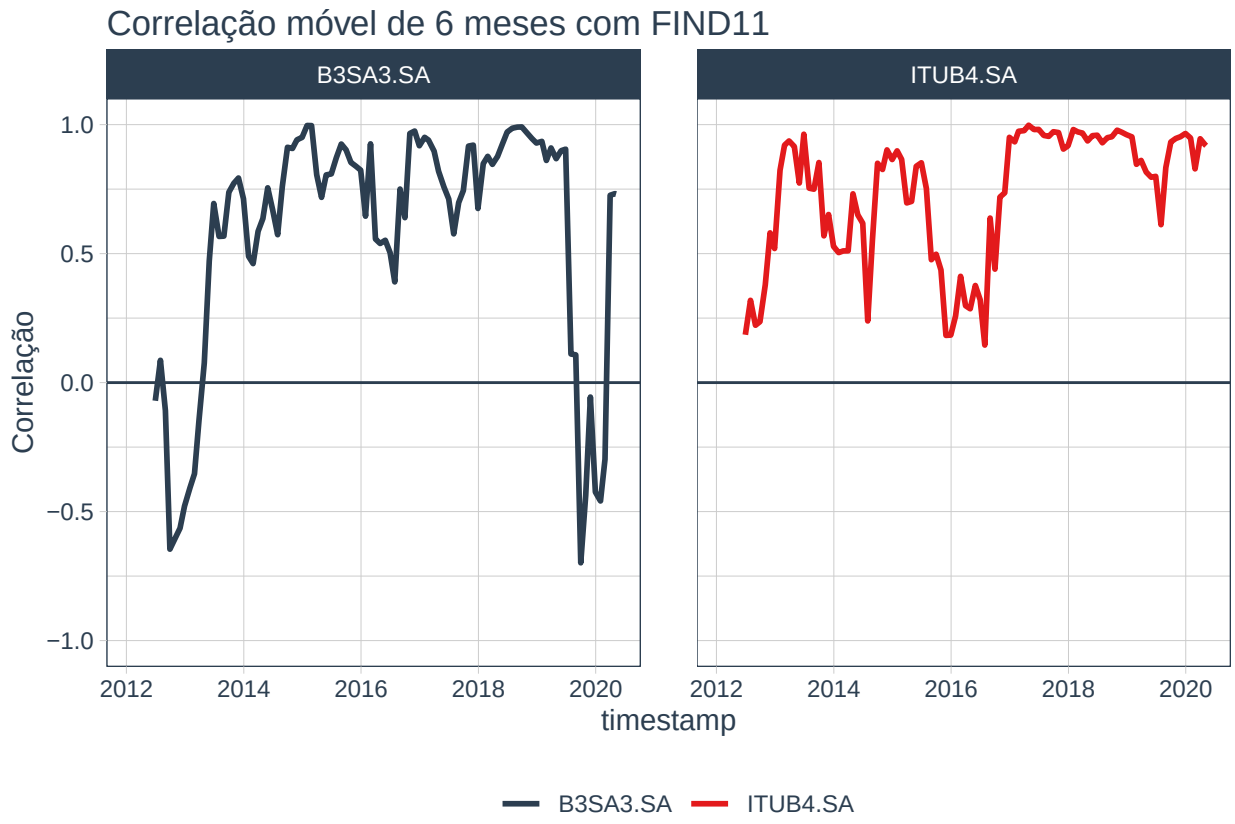
```
av_api_key("9VUXH0JCQ4C5U56K")
```

```
# Calculando o retorno mensal do índice
indice_retornos_mensais <- tq_get("FIND11.SA",
                                  get = "alphavantage",
                                  av_fun = "TIME_SERIES_DAILY",
                                  outputsize = "full")%>% tq_transmute(select=close,
                                                                       mutate_fun=periodReturn,
                                                                       period="monthly",
                                                                       col_rename = "retornos_indice") %
  filter(year(timestamp) >= 2012)
```

```r
# Une os dois retornos num mesmo dataframe
retornos_unidos <- left_join(retornos_mensais,
                             indice_retornos_mensais,
                             by = "timestamp")
retornos_unidos
```

```
## # A tibble: 202 x 4
## # Groups:   symbol [2]
##    symbol   timestamp  retornos_acoes retornos_indice
##    <chr>    <date>              <dbl>           <dbl>
##  1 B3SA3.SA 2012-01-31         0.121         -0.0143
##  2 B3SA3.SA 2012-02-29         0.0455         0.0289
##  3 B3SA3.SA 2012-03-30        -0.0218        -0.115
##  4 B3SA3.SA 2012-04-30        -0.0463         0.00773
##  5 B3SA3.SA 2012-05-31        -0.109          0.0232
##  6 B3SA3.SA 2012-06-29         0.0733        -0.0242
##  7 B3SA3.SA 2012-07-31         0.122          0.0180
##  8 B3SA3.SA 2012-08-31        -0.0704         0.0308
##  9 B3SA3.SA 2012-09-28         0.146         -0.0159
## 10 B3SA3.SA 2012-10-31         0.0612        -0.0334
## # ... with 192 more rows
```

```r
# Calculando a correlação móvel
correlacao_movel <- retornos_unidos %>%
  tq_transmute_xy(x = retornos_acoes,
                  y = retornos_indice,
                  mutate_fun = runCor,
                  n = 6, #determina quantos períodos serão usados no calculo da correlação
                  col_rename = "correlacao_6")

# Plotando as correlações em gráficos de linha
correlacao_movel %>%
  ggplot(aes(x = timestamp, y = correlacao_6, color = symbol)) +
  geom_hline(yintercept = 0, color = palette_light()[[1]]) +
  geom_line(size = 1) +
  labs(title = "Correlação móvel de 6 meses com FIND11",
       y = "Correlação",
       color = "") +
  scale_y_continuous(limits = c(-1,1)) +
  facet_wrap(~ symbol, ncol = 2) +
  theme_tq() +
  scale_color_tq()
```



MODELAGEM EM ESCALA COM TIDYQUANT

O maior benefício do pacote tidyquant é a possibilidade de utilizar o workflow do tidyquant para modelar uma análise sobre um ativo e então replicá-la para um conjunto de outros. Essa capacidade é extremamente útil no processo de análise financeira pois tipicamente pretendemos analisar vários ativos antes da tomada de decisão. Agora veremos com um pouco mais de detalhes algumas técnicas úteis para importar e manipular grupos de dados.

Modelando dados financeiros usando o purrr

Uma dos melhores recursos do **tidyverse** é a possibilidade de se utilizar a função **map** do pacote **purrr** em dados aninhados com a função **nest**. Fazemos isso em dois passos:

1. Modelamos um único ativo
2. Aplicamos o modelo em escala

Vamos demonstrar o processo através de um exemplo, aplicando uma modelo de regressão para detectar tendências positivas:

Exemplo: usando regressão para identificar tendências

Nesse exemplo usaremos um modelo linear simples para identificar a tendência em retornos anuais e determinar se os retornos estão aumentando ou diminuindo ao longo do tempo.

1. Modelando um único ativo Primeiro importamos os dados de uma única ação usando **tq_get**. Como já importamos anteriormente os dados da PETR4 vamos utilizá-lo.

Em seguida criamos uma função que nos permita calcular os retornos logarítmicos anuais. Usaremos a já conhecida função **tq_transmute()** com argumentos `type = log` e `period = "yearly` para garantir que teremos retornos logarítmicos anuais da seguinte forma:

```
retornos_log_anuais <- function(ativo){
  ativo %>% tq_transmute(select = close,
                         mutate_fun = periodReturn,
                         type = "log",
                         period = "yearly")
}
```

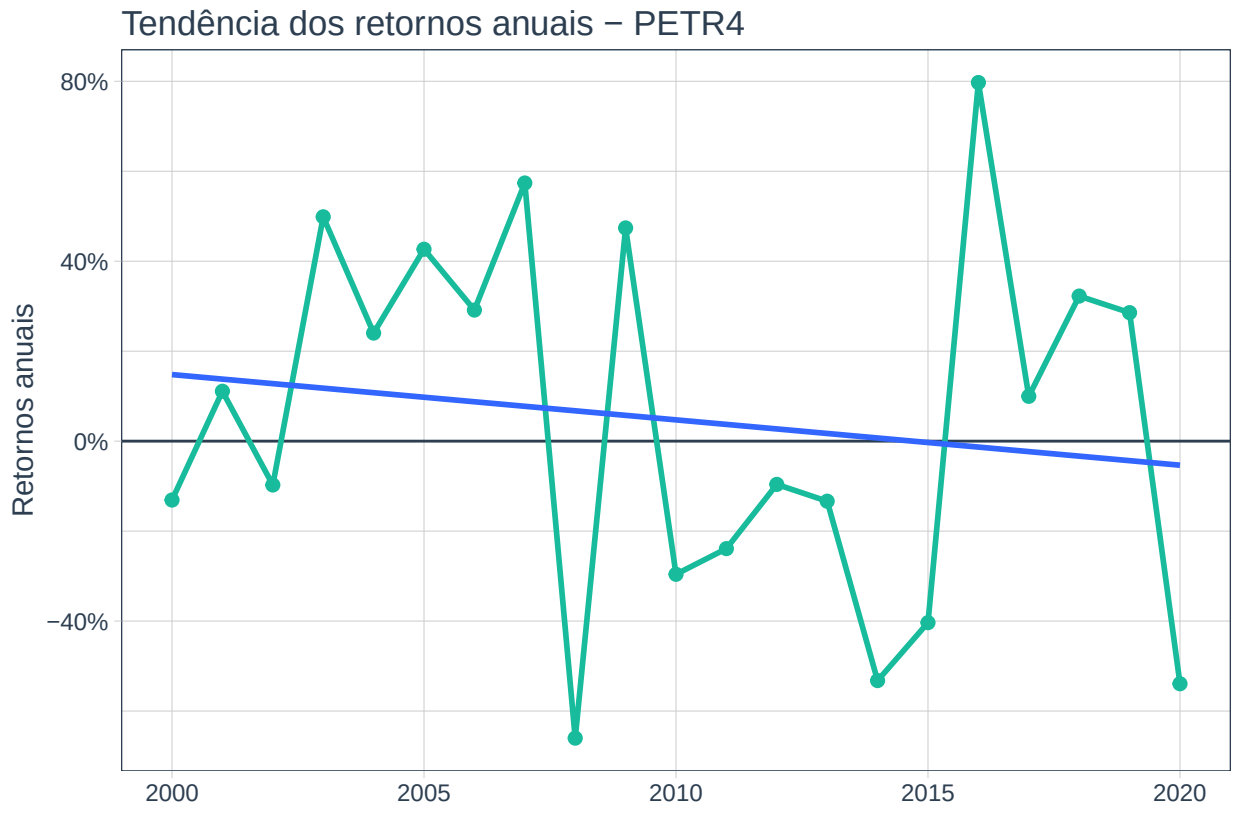
Vamos testar nossa nova função nos dados importados da petrobrás:

```
log_return_PETR4 <- retornos_log_anuais(PETR4)
log_return_PETR4
```

```
## # A tibble: 21 x 2
##    timestamp  yearly.returns
##    <date>              <dbl>
##  1 2000-12-28        -0.131
##  2 2001-12-28         0.111
##  3 2002-12-30        -0.0975
##  4 2003-12-30         0.499
##  5 2004-12-30         0.240
##  6 2005-12-29         0.427
##  7 2006-12-28         0.291
##  8 2007-12-28         0.574
##  9 2008-12-30        -0.660
## 10 2009-12-30         0.474
## # ... with 11 more rows
```

Vamos visualizar os dados com a tendência num gráfico:

```
log_return_PETR4 %>%
  ggplot(aes(x = year(timestamp), y = yearly.returns)) +
  geom_hline(yintercept = 0, color = palette_light()[[1]]) +
  geom_point(size = 2, color = palette_light()[[3]]) + #plota os dados num gráfico de pontos
  geom_line(size = 1, color = palette_light()[[3]]) + #plota os dados num gráfico de linha
  geom_smooth(method = "lm", se = FALSE) + #adiciona uma regressão linear sem margem de erro
  scale_y_continuous(labels = scales::percent) +
  labs(title = "Tendência dos retornos anuais - PETR4",
       x = "",
       y = "Retornos anuais",
       color = "") +
  theme_tq()
```



Agora podemos utilizar a função **lm()** para calcular a regressão linear porém temos um problema: a função **lm()** não está adaptada para trabalhar com dados tidy.

```
mod <- lm(yearly.returns ~ year(timestamp), data = log_return_PETR4)
mod
```

```
##
## Call:
## lm(formula = yearly.returns ~ year(timestamp), data = log_return_PETR4)
##
## Coefficients:
##     (Intercept)  year(timestamp)
##        20.29148         -0.01007
```

Para lidar com isso podemos utilizar o pacote **broom** para forçar o resultado da função ao formato desejado. Existem primariamente três funções no pacote:

1. **augment()**: amplia um dataframe em formato tibble com informações de outro objeto
2. **glance()**: recebe um modelo e retorna um resumo estatístico do mesmo em uma linha
3. **tidy()**: converte o resultado de um modelo para o formato tibble

Usamos, então, a função **tidy** para ajustar o resultado do nosso modelo linear:

```
library(broom)
tidy(mod)
```

```
## # A tibble: 2 x 5
##   term            estimate std.error statistic p.value
##   <chr>              <dbl>     <dbl>     <dbl>   <dbl>
## 1 (Intercept)      20.3      29.8        0.681   0.504
## 2 year(timestamp)  -0.0101    0.0148    -0.679   0.505
```

Podemos então adicionar essa função ao nosso workflow:

```
get_model <- function(ativo){
  annual_returns <- retornos_log_anuais(ativo)
  mod <- lm(yearly.returns ~ year(timestamp), data = annual_returns)
  tidy(mod)
}
```

Testando nossa nova função:

```
get_model(PETR4)
```

```
## # A tibble: 2 x 5
##   term            estimate std.error statistic p.value
##   <chr>              <dbl>     <dbl>     <dbl>   <dbl>
## 1 (Intercept)      20.3      29.8        0.681   0.504
## 2 year(timestamp)  -0.0101    0.0148    -0.679   0.505
```

Agora que confirmamos que nosso modelo funciona em um ativo estamos prontos para aplicá-lo em escala.

2. Aplicando o modelo em escala Uma vez que temos a certeza de que nosso modelo funciona em um ativo aplicá-lo em um conjunto de ativos é simples. Podemos agora aplicar nosso modelo ao conjunto de ativos usando **dplyr::mutate** e **purrr::map**. A função **mutate()** adiciona uma coluna a um dataframe em formato tibble (de maneira similar ao tq_mutate) e a função **map()** aplica uma função selecionada, no nosso caso a nossa função **get_model**, ao nosso tibble com os ativos usando a coluna **symbol** como referência. Além disso as funções **nest()** e **unnest()** respectivamente agrupam as linhas de um tibble com base em valores de uma coluna e desagrupam os dados agrupados mantendo as transformações aplicadas nos dados agrupados no dataframe expandido. As funções **filter()**, **arrange()** e **select()** servem para manipular o dataframe e permitir as operações desejadas. A seguir aplicamos todas as funções num passo a passo para demonstrar a aplicação:

```
#Nest
acoes %>%
  group_by(symbol) %>%
  nest()
```

```
## # A tibble: 2 x 2
## # Groups:   symbol [2]
##   symbol   data
##   <chr>    <list>
## 1 B3SA3.SA <tibble [2,222 x 6]>
## 2 ITUB4.SA <tibble [4,775 x 6]>
```

```r
#Amostra dos tibbles aninhados dentro de data
acoes %>%
  group_by(symbol) %>%
  nest() %>% filter(symbol == "B3SA3.SA") %>% .$data
```

```
## [[1]]
## # A tibble: 2,222 x 6
##    timestamp   open  high   low close   volume
##    <date>     <dbl> <dbl> <dbl> <dbl>    <dbl>
##  1 2011-05-09  11.9  12.1  11.8  12.1     5065
##  2 2011-05-11  12.1  12.1  11.8  11.8 14752500
##  3 2011-05-12  11.7  11.9  11.6  11.7  7371700
##  4 2011-05-13  11.6  11.6  11.3  11.5 12827000
##  5 2011-05-17  11.5  11.6  11.3  11.4 12759000
##  6 2011-05-18  11.4  11.4  11.1  11.1 12433000
##  7 2011-05-19  11.2  11.3  11.0  11.0  8425400
##  8 2011-05-20  11.0  11.2  11.0  11.0 12932900
##  9 2011-05-23  11    11.1  11.0  11    7299800
## 10 2011-05-24  11.0  11.2  11.0  11.0 10408000
## # ... with 2,212 more rows
```

```r
# Aplica a função get_model nos dados agrupados
acoes %>%
  group_by(symbol) %>%
  nest() %>%
  mutate(model = map(data, get_model))
```

```
## # A tibble: 2 x 3
## # Groups:   symbol [2]
##   symbol   data                model
##   <chr>    <list>              <list>
## 1 B3SA3.SA <tibble [2,222 x 6]> <tibble [2 x 5]>
## 2 ITUB4.SA <tibble [4,775 x 6]> <tibble [2 x 5]>
```

```r
# Desagrupa e coleta a inclinação
acoes %>%
  group_by(symbol) %>%
  nest() %>%
  mutate(model = map(data, get_model)) %>%
  unnest(model) %>%
  filter(term == "year(timestamp)") %>% #filtra apenas os dados do coeficiente
  arrange(desc(estimate)) %>% #rearranja os dados em ordem decrescente de coeficiente
  select(-term) #remove a coluna term
```

```
## # A tibble: 2 x 6
## # Groups:   symbol [2]
##   symbol   data                estimate std.error statistic p.value
##   <chr>    <list>                 <dbl>     <dbl>     <dbl>   <dbl>
## 1 B3SA3.SA <tibble [2,222 x 6]>   0.0303   0.0300      1.01   0.342
## 2 ITUB4.SA <tibble [4,775 x 6]>  -0.0109   0.00945    -1.15   0.263
```

PLOTANDO GRÁFICOS COM TIDYQUANT

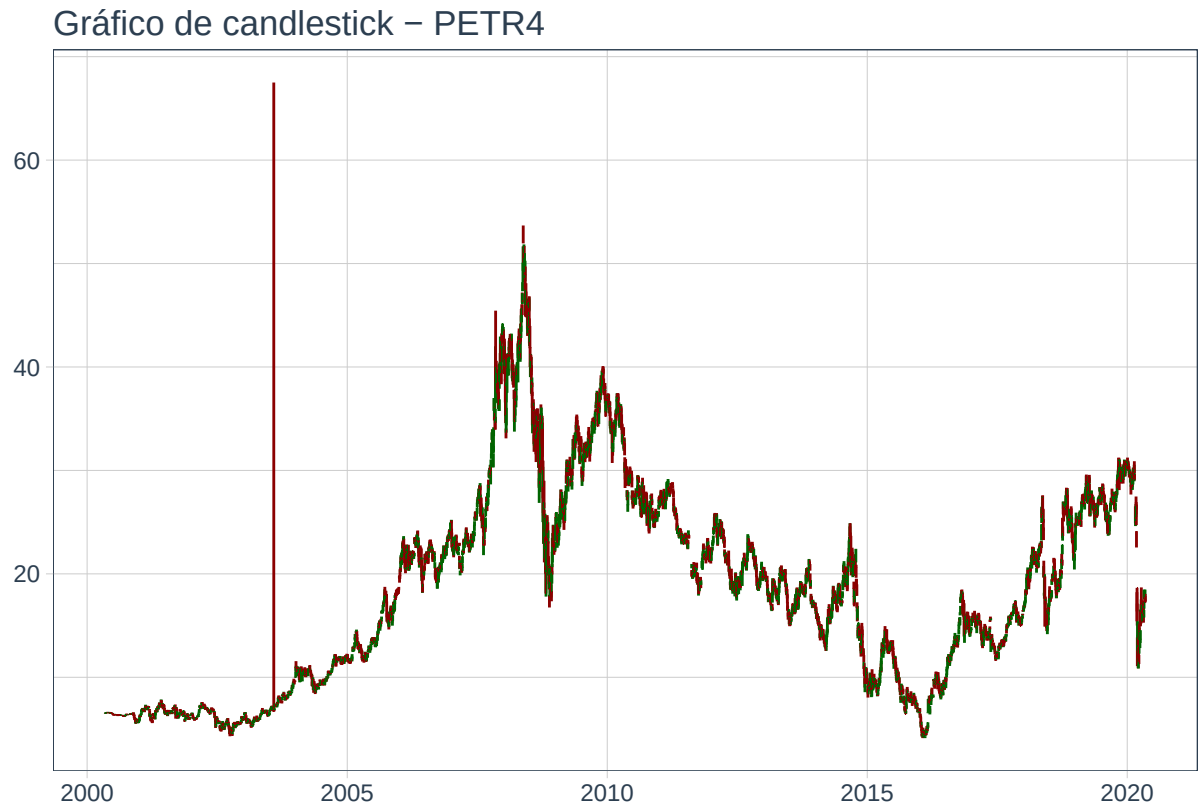
O pacote tiqyquant possui um conjunto de ferramentas gráficas para ajudar o usuário a plotar rapidamente gráficos dentro da gramática do **ggplot2**. Existem três categorias de geometrias e uma categoria de manipulação de coordenadas adicionadas junto ao tidyquant:

- **Tipos de gráfico**: estão disponíveis os gráficos de barra e de candlestick acessíveis respectivamente pelos códigos **geom_barchart()** e **geom_candlestick()**. Como o gráfico de candlestick é amplamente mais utilizado e possui as mesmas informações que um gráfico de barra apresentaremos apenas o de candlestick. Para utilizar o gráfico de barra basta utilizar a mesma estrutura apresentada a seguir (com excessão dos argumentos fill, já que o gráfico de barra não tem preenchimento) substituindo **geom_candlestick()** por **geom_barchart()**.
- **Médias móveis**: Sete tipos de médias móveis estão disponíveis utilizando **geom_ma()**.
- **Bandas de bollinger**: Bandas de bollinger podem ser visualizadas utilizando **geom_bb()**. A média utilizada nas bandas de bollinger pode ser qualquer uma das sete disponíveis em geom_ma.
- **Zoom em intervalos de tempo**: estão disponíveis duas funções de coordenada (**coord_x_date()** e **coord_x_datetime()**) que permitem dar zoom em uma parte específica do gráfico sem perda de informação. Isso é especialmente útil quando usamos médias móveis e bandas de bollinger.

Gráfico de candlestick

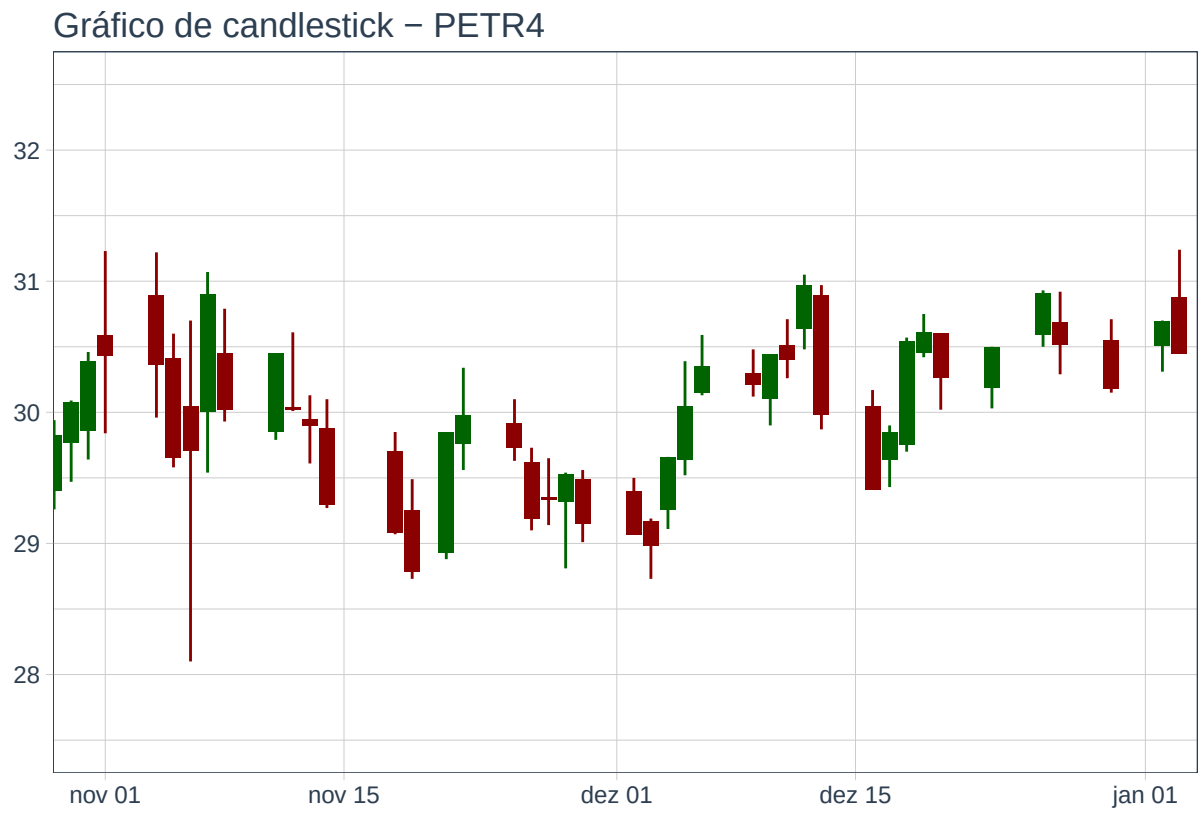
Como ja dito os plots do tidyquant seguem a gramática do ggplot. Primeiramente dentro do **aes()** do **ggplot()** precisamos determinar as duas variáveis que serão usadas montar os eixos do gráfico, no caso o fechamento e o tempo. Em seguida adicionamos o layer **geom_candlestick()**. Note que o candle precisa da máxima, mínima, abertura e fechamento para ser formado por isso precisamos adicionar essas variáveis no **aes()** dentro da geometria. Junto ao **aes()** temos ainda a opção de adicionar argumentos que modificam as cores dos candles. Os argumentos **colour_up** e **colour_down** modificam as bordas e os argumentos **fill_up** e **fill_down** o preenchimento dos candles. Um exemplo de aplicação dos códigos apresentados podem ser vistos a seguir:

```
PETR4 %>%
    ggplot(aes(x = timestamp, y = close)) +
    geom_candlestick(aes(open = open, close = close, #adiciona abertura e fechamento
                         high = high, low = low),    #adiciona maxima e minima
                     colour_up = "darkgreen", colour_down = "darkred", #muda cor das bordas
                     fill_up = "darkgreen", fill_down = "darkred") + #muda cor do preenchimento
    labs(title = "Gráfico de candlestick - PETR4",
         y = "",
         x = "") +
    theme_tq()
```



O resultado, como podemos ver, não é satisfatório pois o conjunto de dados é muito grande e com isso a legibilidade dos candles é prejudicada. Para tornar o gráfico mais legível precisamos aproximar o gráfico em uma região, podemos fazer isso de duas maneiras. A primeira é utilizar a já conhecida função **filter()** para filtrar apenas as datas que desejamos observar. O problema dessa abordagem é que no caso de utilizarmos por exemplo uma média móvel (explicado mais abaixo) diminuir a quantidade de dados vai fazer com que as primeiras entradas do intervalo não tenham média já que a quantidade de dados foi reduzida. Uma abordagem alternativa, presente no tidyquant, é utilizar as funções **coord_x_date()** ou **coord_x_datetime** com os argumentos **xlim** e **ylim** para dar zoom na parte desejada do gráfico. Um exemplo da utilização dessa função é dada a seguir:

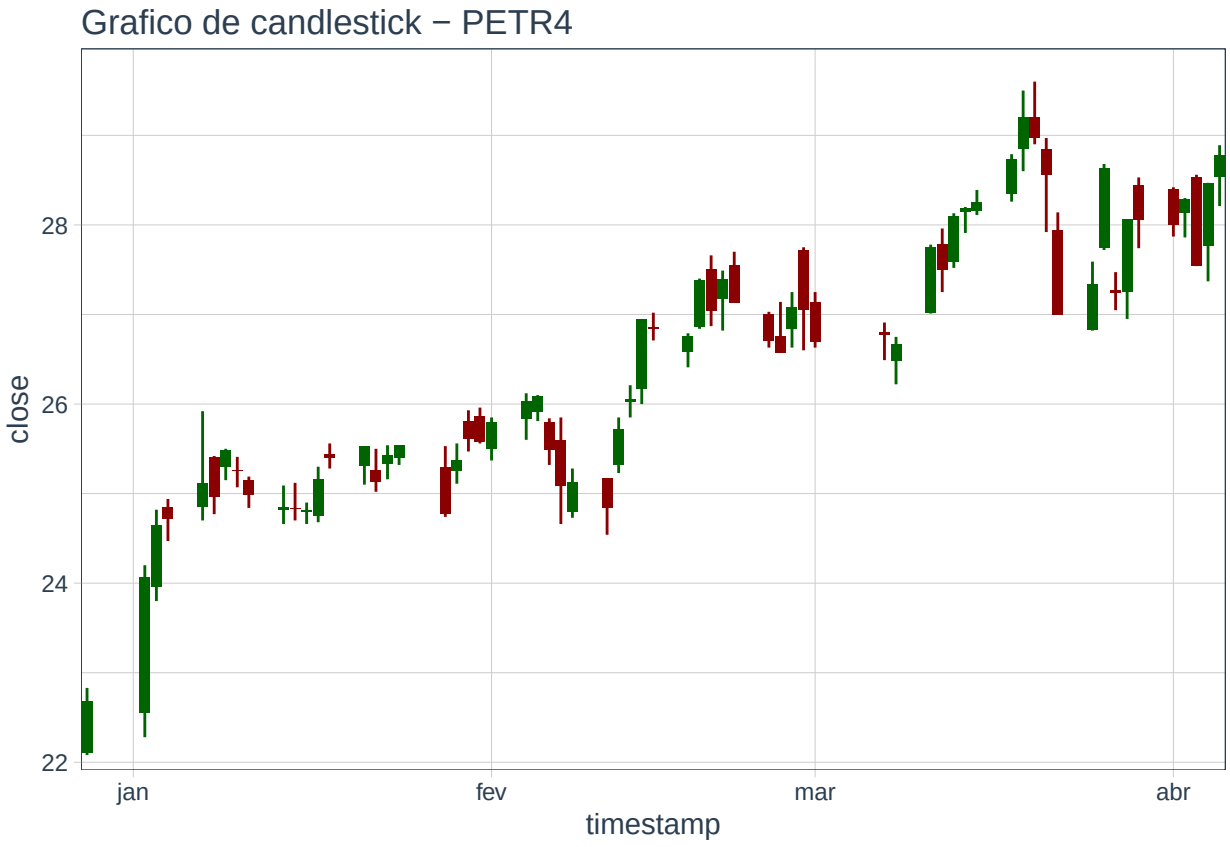
```
PETR4 %>%
  ggplot(aes(x = timestamp, y = close)) +
  geom_candlestick(aes(open = open, close = close,
                       high = high, low = low),
                   colour_up = "darkgreen", colour_down = "darkred",
                   fill_up = "darkgreen", fill_down = "darkred") +
  labs(title = "Gráfico de candlestick - PETR4",
       y = "",
       x = "") +
  theme_tq() +
  coord_x_date(xlim = c("2019-11-1", "2020-1-1"), #determina os limites do eixo x
               ylim = c(27.5, 32.5)) #determina os limites do eixo y
```



Ter que ajustar manualmente os limites do eixo y o tempo todo pode ser um problema quando se pretende realizar uma série de análises de ativos ou intervalos diferentes. Uma alternativa para lidar com esse problema é criar uma função que automaticamente seleciona a máxima e a mínima de um período determinado e utiliza esses valores como limites verticais como visto a seguir:

```
candlestick <- function(ativo, inicio, fim){

  #converte as datas em formato character para formato date
  inicio <- as_date(inicio) #define inicio como data
  fim <- as_date(fim) #define fim como data


  #encontra o minimo dentro das datas determinadas
  min <- ativo %>%
    filter(timestamp >= inicio, timestamp <= fim) %>%
    tq_transmute(select = "low",
                 mutate_fun = seriesLo) %>% #funcao que seleciona menor valor de uma serie
    pull(low) #extrai a coluna low do tibble resultante


  #encontra a maxima dentro das datas determinadas
  max <- ativo %>%
    filter(timestamp >= inicio, timestamp <= fim) %>%
    tq_transmute(select = "high",
                 mutate_fun = seriesHi) %>%  #funcao que seleciona maior valor de uma serie
    pull(high) #extrai a coluna high do tibble resultante
```

```
  #plota o grafico com as variaveis geradas
  ativo %>%
    ggplot(aes(x = timestamp, y = close)) +
    geom_candlestick(aes(open = open, close = close,
                         high = high, low = low),
                     colour_up = "darkgreen", colour_down = "darkred",
                     fill_up = "darkgreen", fill_down = "darkred") +
    labs(title = paste("Grafico de candlestick -", deparse(substitute(ativo))),
         xlim = "",
         ylim = "") +
    theme_tq() +
    coord_x_date(xlim = c(inicio, fim),
                 ylim = c(min, max))}

#aplicando a funcao criada
candlestick(PETR4, "2019-1-1", "2019-4-1")
```



Médias móveis

Podemos adicionar médias móveis facilmente nos gráficos de candlestick usando a geometria correta.

Os tipos de média móvel a seguir estão disponíveis: *Simple moving averages (SMA)* Exponential moving averages (EMA) *Weighted moving averages (WMA)* Double exponential moving averages (DEMA) *Zero-*

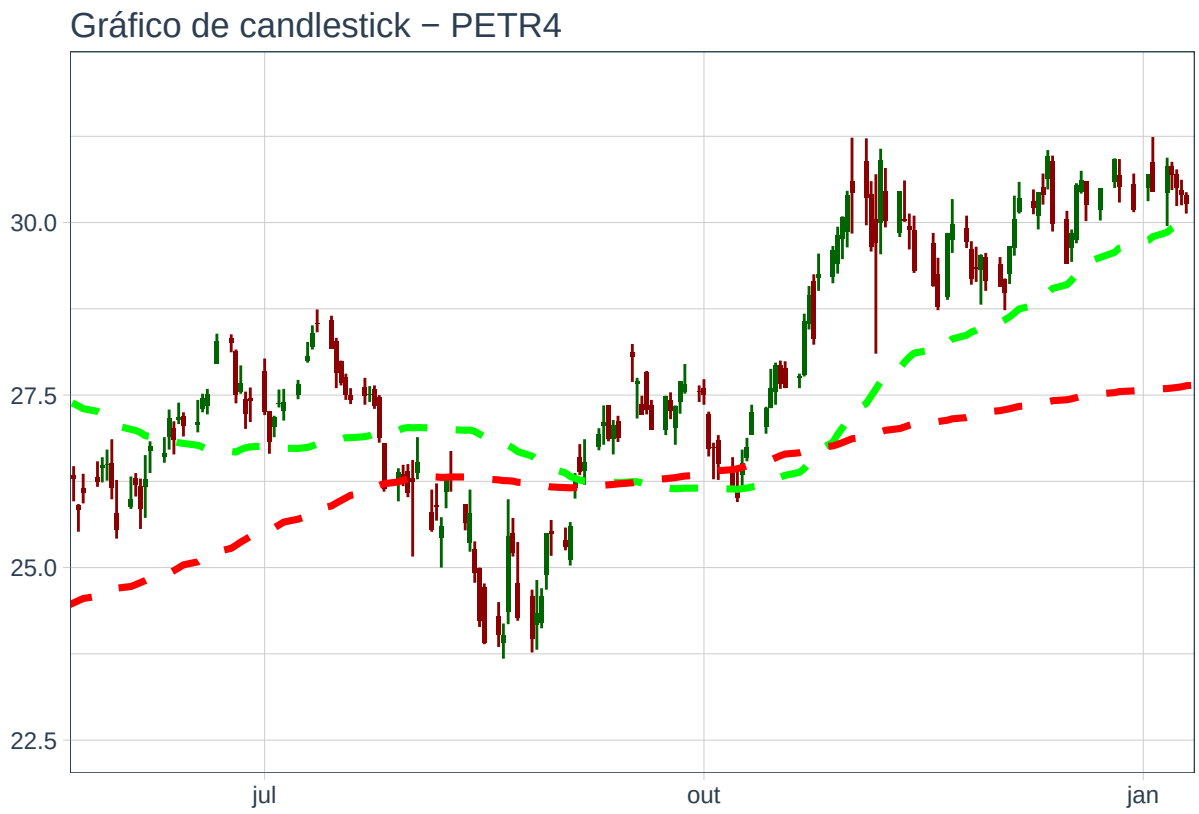
lag exponential moving averages (ZLEMA) Volume-weighted moving averages (VWMA) *Elastic, volume-weighted moving averages (EVWMA)

Para adicionar a média móvel utilizamos a função **geom_ma** como uma camada dentro da estrutura do ggplot junto aos argumentos específicos de cada média móvel. Como a função **geom_ma** incorpora as médias móveis do pacote **TTR** podemos seguir o seguinte passo-a-passo para plotar qualquer tipo de média móvel:

1. Selecionamos a função de média móvel, **ma_fun**, desejada.
2. Determinamos os argumentos característicos do tipo de média móvel. Para descobrir os argumentos necessários podemos acessar a documentação diretamente do pacote **TTR** usando ?TTR::[SIGLA DA MEDIA MOVEL DESEJADA]
3. Determina or argumentos da função **aes()**. Se os argumentos tiverem sido passadas para o **aes()** globalmente dentro da função **ggplot()** esse passo é desnecessário.
4. Aplica a média móvel no workflow do ggplot

Um exemplo adicionando duas médias móveis simples, de 50 e de 200 dias, pode ser visto a seguir:

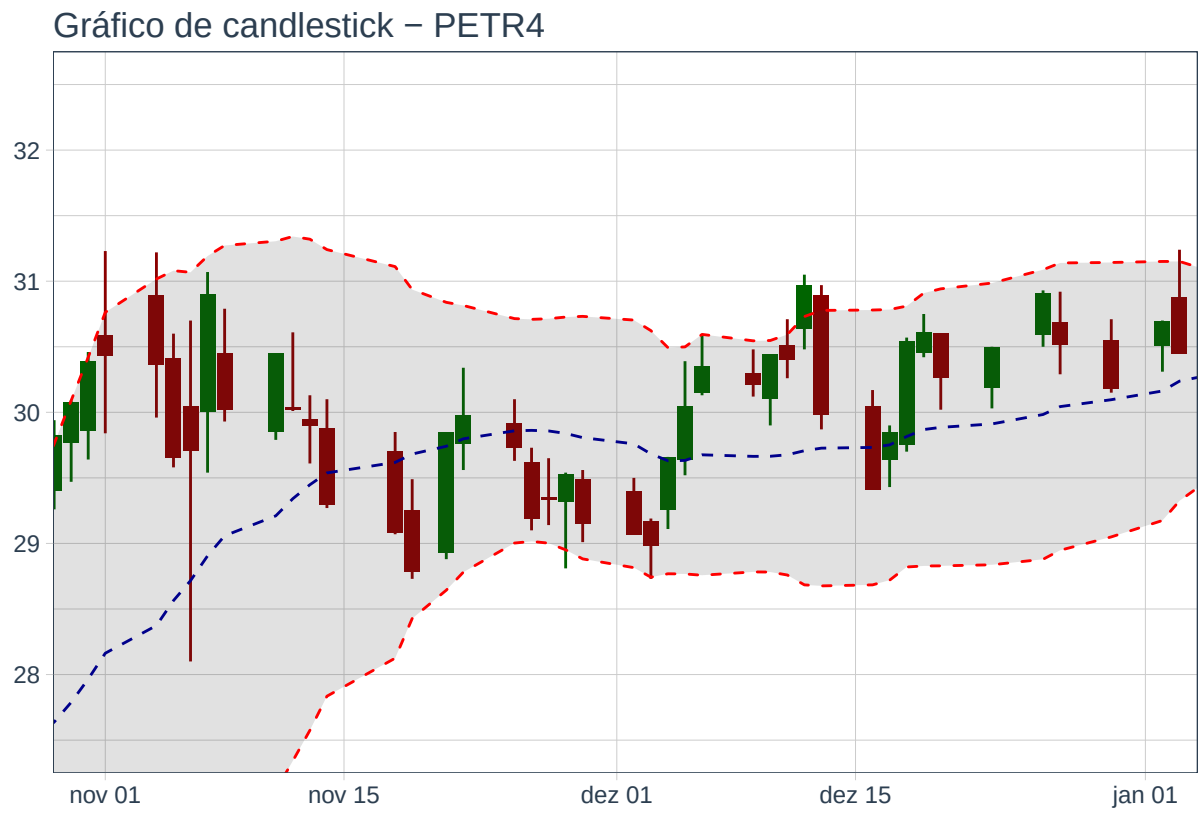
```
PETR4 %>%
  ggplot(aes(x = timestamp, y = close)) +
  geom_candlestick(aes(open = open, close = close,
                       high = high, low = low),
                   colour_up = "darkgreen", colour_down = "darkred",
                   fill_up = "darkgreen", fill_down = "darkred") +
  labs(title = "Gráfico de candlestick - PETR4",
       y = "",
       x = "") +
  theme_tq() +
  coord_x_date(xlim = c("2019-06-1", "2020-1-1"),
               ylim = c(22.5, 32)) +
  geom_ma(ma_fun = SMA, n = 50, color = "green", size = 1.25) + #adiciona MM de 50 periodos
  geom_ma(ma_fun = SMA, n = 200, color = "red", size = 1.25)   #adiciona MM de 200 periodos
```



Bandas de bollinger

Por utilizar médias móveis a função **geom_bbands()** funciona de maneira similar às médias móveis, as mesmas 7 médias móveis estão disponíveis no cálculo. A única diferença é que as bandas de bollinger exigem a adição do argumento **sd** com o desvio padrão desejado além da máxima, da mínima e do fechamento dentro do **aes()**. É importante ressaltar que a função **geom_bbands()** precisa ser adicionada DEPOIS da função **geom_candlestick()** para que esta possa sobrepor os candles como exemplificado a seguir:

```
PETR4 %>%
  ggplot(aes(x = timestamp, y = close, open = open, high = high, low = low, close = close)) +
  geom_candlestick(colour_up = "darkgreen", colour_down = "darkred",
                   fill_up = "darkgreen", fill_down = "darkred") +
  labs(title = "Gráfico de candlestick - PETR4",
       y = "",
       x = "") +
  theme_tq() +
  coord_x_date(xlim = c("2019-11-1", "2020-1-1"), #determina os limites do eixo x
               ylim = c(27.5, 32.5)) +
  geom_bbands(ma_fun = SMA, sd = 2, n = 20)
```



Funcionalidades do GGPLOT

O pacote base **ggplot** possui várias funcionalidades que podem ser úteis para melhorar a qualidade dos nossos plots. Algumas delas como a adição de título e a modificação da legenda dos eixos já foram apresentadas agora apresentaremos mais 3 transformações que também podem ser úteis, a transformação do eixo y para escala logarítimica, a adição de uma linha de regressão linear e o plot do volume.

Escala logarítmica no eixo Y

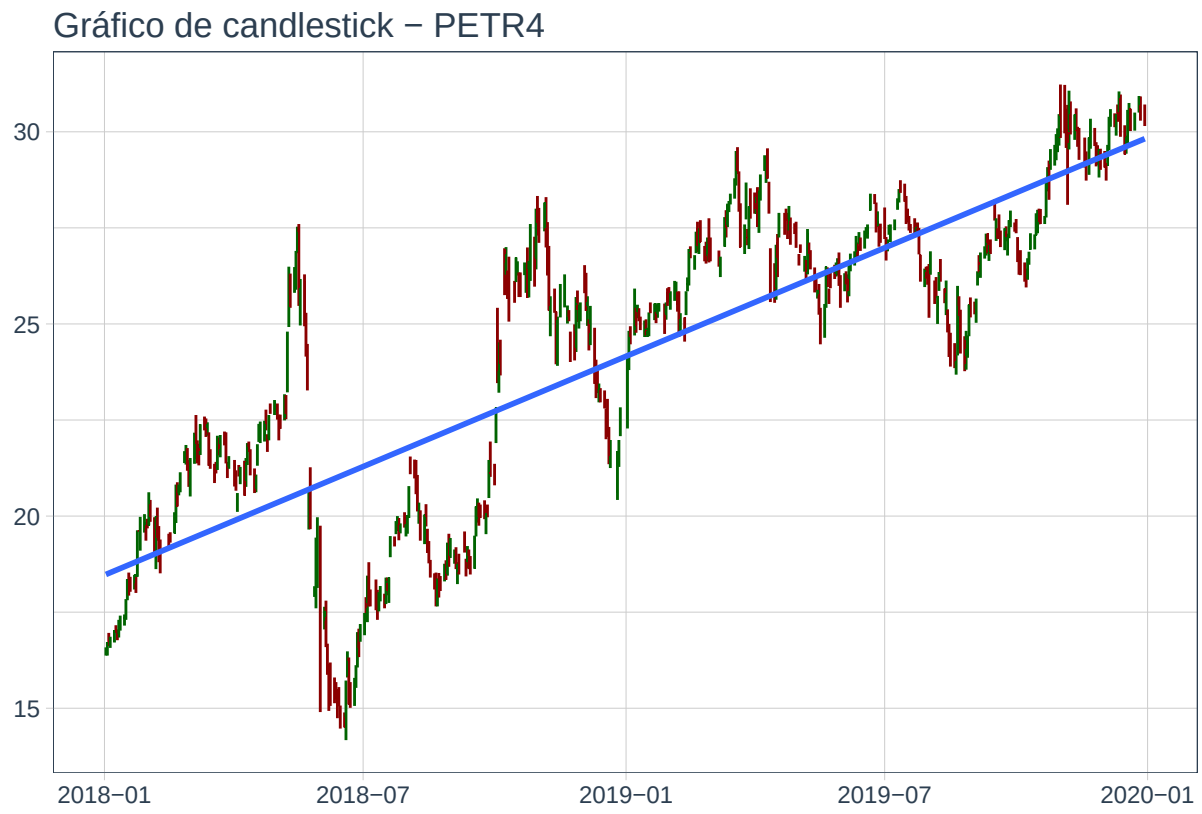
Para alterar a escala basta seguir o código padrão do pacote **ggplot2**:

```
PETR4 %>%
  ggplot(aes(x = timestamp, y = close)) +
  geom_candlestick(aes(open = open, close = close,
                       high = high, low = low),
                   colour_up = "darkgreen", colour_down = "darkred",
                   fill_up = "darkgreen", fill_down = "darkred") +
  labs(title = "Gráfico de candlestick - PETR4",
       y = "",
       x = "") +
  theme_tq() +
  scale_y_log10()
```

Linha de regressão linear

Para adicionar uma linha de regressão linear basta adicionar a função **geom_smooth** com o argumento **method = "lm"** ao workflow do ggplot:

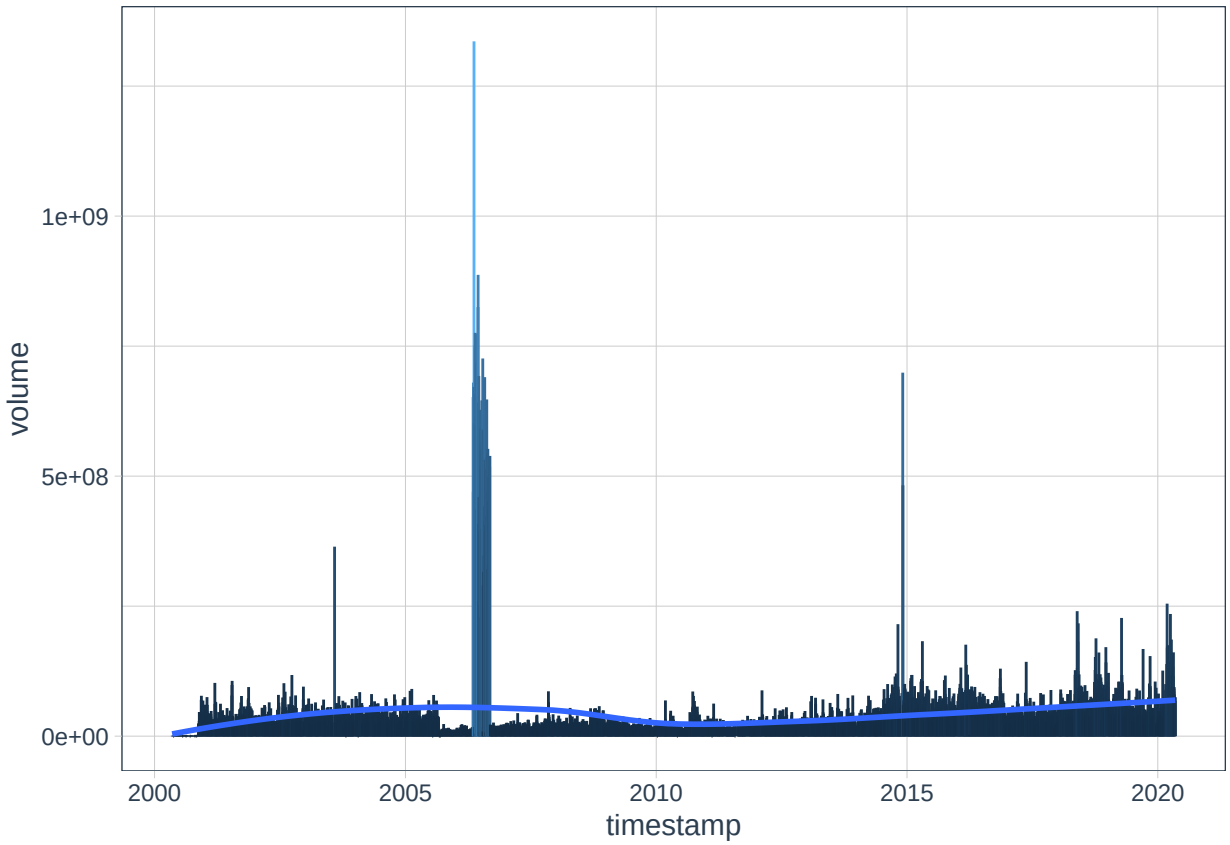
```
PETR4 %>% filter(year(timestamp) >= 2018, year(timestamp) < 2020) %>%
  ggplot(aes(x = timestamp, y = close)) +
  geom_candlestick(aes(open = open, close = close,
                       high = high, low = low),
                   colour_up = "darkgreen", colour_down = "darkred",
                   fill_up = "darkgreen", fill_down = "darkred") +
  labs(title = "Gráfico de candlestick - PETR4",
       y = "",
       x = "") +
  theme_tq() +
  geom_smooth(method = "lm", se = FALSE)
```



Plotando volume

Podemos facilmente plotar o volume utilizando a função **geom_segment**:

```
PETR4 %>%
  ggplot(aes(x = timestamp, y = volume)) +
  geom_segment(aes(xend = timestamp, yend = 0, color = volume)) +
  geom_smooth(method = "loess", se = FALSE) +
    theme_tq() +
    theme(legend.position = "none")
```



Plotando indicadores

As funções necessárias para plotar indicadores dependem de qual indicador queremos plotar por isso é impossível apresentar todos aqui. Ao invés disso vamos apenas exemplificar mostrando como plotar um MACD.

Plotando MACD

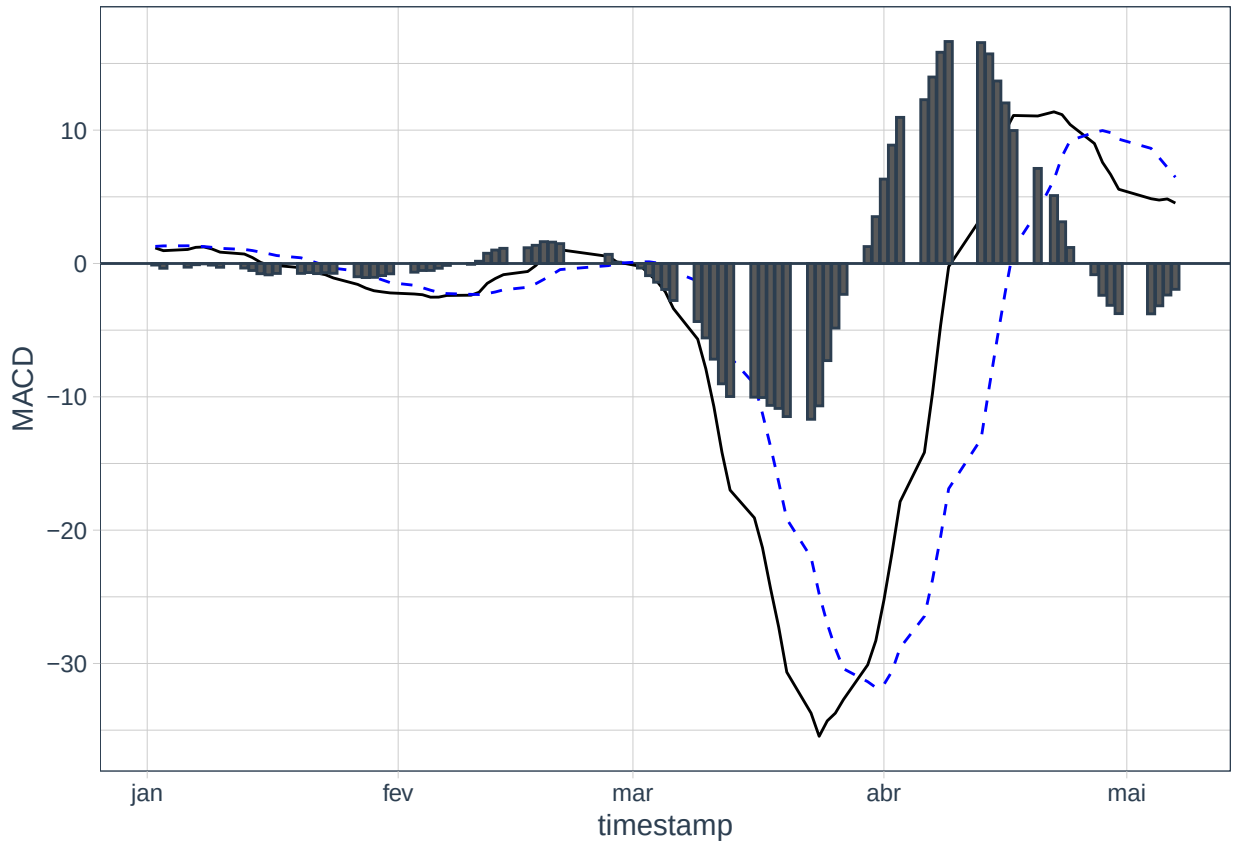
Realizaremos essa tarefa em dois passos. Primeiro criaremos um tibble com os dados necessários depois plotaremos o resultado.

```
#Criando tibble com MACD
PETR4_macd <- PETR4 %>%
  tq_transmute(select = close,
               mutate_fun = MACD,
               nFast      = 12,
               nSlow      = 26,
               nSig       = 9,
               maType     = SMA,
               col_rename = c("MACD", "sinal")) %>%
  mutate(diff = MACD - sinal)
PETR4_macd
```

```
## # A tibble: 4,913 x 4
```

```
##    timestamp            MACD sinal  diff
##    <dttm>              <dbl> <dbl> <dbl>
##  1 2000-05-08 00:00:00    NA    NA    NA
##  2 2000-05-09 00:00:00    NA    NA    NA
##  3 2000-05-10 00:00:00    NA    NA    NA
##  4 2000-05-11 00:00:00    NA    NA    NA
##  5 2000-05-12 00:00:00    NA    NA    NA
##  6 2000-05-15 00:00:00    NA    NA    NA
##  7 2000-05-16 00:00:00    NA    NA    NA
##  8 2000-05-17 00:00:00    NA    NA    NA
##  9 2000-05-18 00:00:00    NA    NA    NA
## 10 2000-05-19 00:00:00    NA    NA    NA
## # ... with 4,903 more rows
```

```
#Plot do MACD
PETR4_macd %>% filter(year(timestamp) >= 2020) %>%
  ggplot(aes(x = timestamp)) +
  geom_hline(yintercept = 0, color = palette_light()[[1]]) + #adiciona linha horizontal em y=0
  geom_line(aes(y = MACD)) + #adiciona linha do MACD
  geom_line(aes(y = sinal), color = "blue", linetype = 2) + #adiciona linha do sinal
  geom_bar(aes(y = diff), stat = "identity", color = palette_light()[[1]]) + #adiciona diferenca
  theme_tq() +
  scale_color_tq()
```



ANÁLISE DE PERFORMANCE NO TIDYQUANT

Visão geral

Existe um amplo conjunto de teorias e métodos usados para se avaliar o desempenho de um ativo ou portfólio, felizmente o pacote **PerformanceAnalytics** possui vários desses métodos e estes estão integrados ao **tidyquant**. Duas funções servem de base no processo de análise:

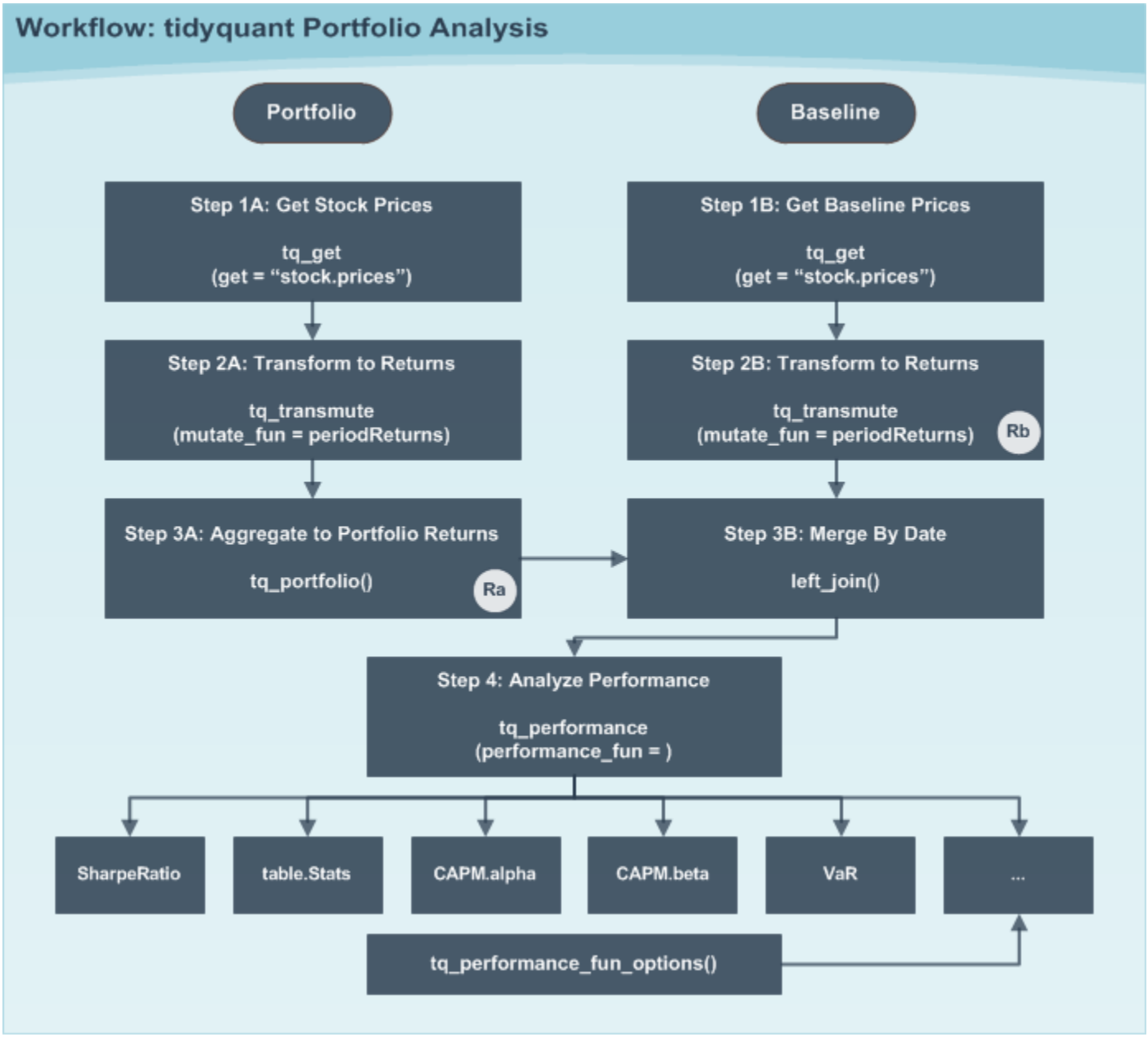
- **tq_performance()** é uma função base a partir da qual aplicamos funções de análise de performance, de forma semelhante a função **tq_transmute**, e que retorna métricas avaliativas em formato tidy.
- **tq_portfolio()** é uma ferramenta conveniente que nos permite facilmente agregar o retorno de vários ativos individuais em uma ou várias carteiras.

Conceitos-chave

Um importante conceito que precisamos ter em mente e que já foi apresentado anteriormente é que o pacote PerformanceAnalytics trabalha com **retornos** e não com séries de preços por isso é necessário converter os dados antes de aplicar as funções. Outro conceito importante é o conceito de benchmark (ou baseline). O benchmark é a referência que comparamos com a performance do nosso ativo ou portfólio para julgar se o seu desempenho foi ou não satisfatório. Os benchmarks mais comuns são o CDI (que não será abordado nesse material), índices teóricos (sendo o mais comum o IBOV) e o comportamento do mesmo ativo no mesmo período seguindo uma estratégia de buy & hold.

Workflow

O passo a passo para performar análises de desempenho usando o tidyquant é resumo no diagrama a seguir disponibilizado:



A seguir exemplificamos o processo para a análise de desempenho de um único ativo e de uma carteira.

Desempenho de um ativo

Análises de desempenho de ativos isolados são mais simples de executar pois não precisam ser agregagos em um portfólio. A seguir exemplificamos o processo de aplicação do índice de Sharpe:

Step 1A: Get Stock prices O processo de importar preços já é conhecido. Usamos a função **tq_get()** junto a todos os argumentos apropriados da seguinte forma:

```
PETR4 <- "PETR4.SA" %>%
  tq_get(get = "alphavantage",
```

```
        av_fun = "TIME_SERIES_DAILY",
        outputsize = "full")
```

Step 2A: Transforme to Returns Esse passo também já é conhecido. Usamos a função **tq_transmute()** junto com a função do quantmod **periodReturn** para converter os fechamentos em retornos. No nosso exemplo convertemos os fechamentos diarios em retornos anuais e também renomeamos a coluna resultante porém esses dois passos não são estritamente necessários:

```
PETR4_retorno_anual <- PETR4 %>% tq_transmute(select = close,
                                              mutate_fun = periodReturn,
                                              period = "yearly",
                                              col_rename = "Ra")
PETR4_retorno_anual
```

```
## # A tibble: 21 x 2
##    timestamp       Ra
##    <date>       <dbl>
##  1 2000-12-28 -0.123
##  2 2001-12-28  0.117
##  3 2002-12-30 -0.0929
##  4 2003-12-30  0.647
##  5 2004-12-30  0.272
##  6 2005-12-29  0.532
##  7 2006-12-28  0.338
##  8 2007-12-28  0.775
##  9 2008-12-30 -0.483
## 10 2009-12-30  0.606
## # ... with 11 more rows
```

Step 3A: Aggregate to Portfolio Returns Como nossa análise se resume a um único ativo isolado e não a um portfólio podemos pular esse passo.

Steps 1B, 2B and 3B Os passos 1B, 2B e 3B dizem respeito a como tratar o benchmark e associá-lo a série de fechamentos do ativo ou carteira. Como o índice de sharpe não utiliza nenhum benchmark esses passos também podem ser pulados.

Step 4: Analyze Performance O último passo é utilizar a função **tq_performance()** para aplicar análise de performance na nossa série de retornos. Usando a função **args()** podemos acessar os argumentos da função:

```
args(tq_performance)
```

```
## function (data, Ra, Rb = NULL, performance_fun, ...)
## NULL
```

Vemos que são necessários 4 argumentos: * **data** é o argumento em que adicionamos o dataframe contendo os dados

- **Ra** é o argumento em que adicionamos a série de retornos do ativo ou portfólio

- **Rb** é o argumento em que adicionamos a série de retornos do benchmark, quando necessário.
- **performance_fun** é o argumento em que adicionamos a função de performance escolhida

  Nossa função escolhida é a **SharpeRatio** que calcula o índice de sharpe. Para calcular o índice de sharpe precisamos, além da série de retornos, pelo menos da taxa de juros livre de risco, adicionada pelo argumento **Rf**. Além desse, outros argumentos específicos da função **SharpeRatio** estão disponíveis e podem ser acessadas a partir da documentação da mesma usando **help(SharpeRatio)** ou **?SharpeRatio**.

```
PETR4_retorno_anual %>%
  tq_performance(
    Ra = Ra, #chama o vetor dentro do dataframe que contem a serie de retornos
    performance_fun = SharpeRatio, #aplica o indice de sharpe
    Rf = 0.03, #especifica a taxa de juros livre de risco
    FUN = "StdDev") #especifica o uso do desvio padrao como denominador
```

```
## # A tibble: 1 x 1
##   `StdDevSharpe(Rf=3%,p=95%)`
##                         <dbl>
## 1                       0.226
```

Desempenho de um portfolio

A análise de portfólio é um pouco mais complicada do que a análise de uma carteira pois nela é necessário, antes de aplicar a função de análise, agregar os dados das ações individuais em uma carteira. Felizmente o pacote tidyquant possui a função **tq_porffolio()** que simplifica esse processo. Nesse exemplo calcularemos o alfa e o beta da carteira usando o CAPM.

Step 1A: Get Stock Prices Usaremos como base no nosso exemplo os dados das ações do setor financeiro:

```
acoes <- tq_get(c("B3SA3.SA", "ITUB4.SA", "BBDC4.SA", "ITSA4.SA"),
                get = "alphavantage",
                av_fun = "TIME_SERIES_DAILY",
                outputsize = "full")
```

Step 2A: Transforme to Returns O processo de transformação dos fechamentos em retornos é similar ao processo aplicado para um único ativo com excessão de que agora precisamos agrupar os dados por simbolo antes de aplicar a função tq_transmute:

```
SF_retornos_anuais <- acoes %>%
  filter(year(timestamp) >= 2012) %>% #filtra dados a partir de 2012
  group_by(symbol) %>% #agrupa os dados por simbolo
  tq_transmute(select = close,
               mutate_fun = periodReturn,
               period = "yearly",
               col_rename = "Ra")
SF_retornos_anuais
```

```
## # A tibble: 36 x 3
## # Groups:   symbol [4]
```

```
##    symbol   timestamp       Ra
##    <chr>    <date>       <dbl>
##  1 B3SA3.SA 2012-12-28  0.436
##  2 B3SA3.SA 2013-12-30 -0.210
##  3 B3SA3.SA 2014-12-30 -0.110
##  4 B3SA3.SA 2015-12-30  0.107
##  5 B3SA3.SA 2016-12-29  0.515
##  6 B3SA3.SA 2017-12-29  0.387
##  7 B3SA3.SA 2018-12-28  0.171
##  8 B3SA3.SA 2019-12-30  0.603
##  9 B3SA3.SA 2020-05-15 -0.0798
## 10 ITUB4.SA 2012-12-28 -0.0194
## # ... with 26 more rows
```

Steps 1B and 2B: Baseline Period Returns Para importar e transformar os dados do nosso benchmark seguimos os mesmos passos aplicados nos ativos individuais. Note como juntamos os dois códigos em um só obtendo diretamente os retornos sem a necessidade de definir uma variável para os preços.

```
IBOV_retornos_anuais <- "^BVSP" %>%
  tq_get(get = "stock.prices", #note que a estrutura de importacao dos dados e diferente pois
         from = "2012-01-01",  #estamos importando dados do yahoo finance e nao da alpha vantage
         to = "2020-05-08") %>%
  tq_transmute(select = close,
               mutate_fun = periodReturn,
               period = "yearly",
               col_rename = "Rb") %>%
    rename(timestamp = date) %>% #renomeia a variavel date para timestamp
  mutate(date = (year(timestamp))) #muda a coluna data para conter apenas o ano

IBOV_retornos_anuais
```

```
## # A tibble: 9 x 3
##   timestamp       Rb  date
##   <date>       <dbl> <dbl>
## 1 2012-12-28  0.0540  2012
## 2 2013-12-30 -0.155   2013
## 3 2014-12-30 -0.0291  2014
## 4 2015-12-30 -0.133   2015
## 5 2016-12-29  0.389   2016
## 6 2017-12-29  0.269   2017
## 7 2018-12-28  0.150   2018
## 8 2019-12-30  0.319   2019
## 9 2020-05-07 -0.326   2020
```

Step 3A: Aggregate to Portfolio Period Returns Para agregar os fechamentos individuais em um portfolio usamos a função **tq_portfolio()**. Para fazer isso precisamos primeiro definir os pesos de cada ativo dentro da carteira e existem duas formas de se fazer isso.

1. Fornecendo um vetor com os pesos.
2. Fornecendo um tibble de duas colunas sendo a primeira contendo os simbolos dos ativos e a segunda contendo os pesos.

A seguir vamos apresentar as duas formas.

Metodo 1: Fornecendo um vetor com os pesos Nesse método basta escrever um vetor numerico comum usando **c()**. Note que alguns pré-requisitos naturalmente precisam ser preenchidos, sendo eles:

1. A soma dos pesos deve ser necessariamente igual a 1. Se esse requisito não for preenchido a função irá automaticamente dimensionar o vetor para 1 e uma mensagem de erro aparecerá.
2. O numero de pesos deve ser igual ao número de ativos.

A seguir um exemplo prático.

```
pesos = c(0.3, 0.0, 0.3, 0.4)
carteira_retornos_anuais <- SF_retornos_anuais %>%
  tq_portfolio(assets_col = symbol,     #define em qual coluna esta os simbolos
               returns_col = Ra,        #define em qual coluna esta os retornos
               weights = pesos,         #define o objeto com os pesos
               col_rename = "Ra") %>%   #renomeia a coluna resultante
  mutate(date = year(timestamp))   #modifica timestamp para conter apenas o ano
carteira_retornos_anuais
```

```
## # A tibble: 9 x 3
##   timestamp       Ra  date
##   <date>       <dbl> <dbl>
## 1 2012-12-28  0.158   2012
## 2 2013-12-30 -0.0990  2013
## 3 2014-12-30  0.0873  2014
## 4 2015-12-30 -0.162   2015
## 5 2016-12-29  0.478   2016
## 6 2017-12-29  0.336   2017
## 7 2018-12-28  0.211   2018
## 8 2019-12-30  0.312   2019
## 9 2020-05-15 -0.285   2020
```

Observe que atribuímos peso 0 a um dos ativos. Nem todos os ativos precisam estar incluídos na carteira porém nessa abordagem precisamos definir um peso para cada ativo do conjunto de dados.

Método 2: Fornecendo um tibble com simbolos e pesos Essa segunda abordagem como o próprio título já deixa claro envolve criar um tibble com os símbolos na primeira linha e os pesos na segunda. Essa abordagem tem a vantagem de que não é necessário atribuir um peso para cada ativo do conjunto de dados, apenas para os que serão efetivamente utilizados. Essa propriedade é especialmente útil quando pretendemos utilizar apenas alguns ativos dentro de um conjunto com dados de dezenas ou mesmo centenas deles. A seguir exemplificamos:

```
mapa_pesos <- tibble(
  symbols = c("B3SA3.SA", "BBDC4.SA", "ITSA4.SA"),
  weigths = c(0.3, 0.3, 0.4)
)

SF_retornos_anuais %>%
  tq_portfolio(assets_col = symbol, #define em qual coluna esta os simbolos
               returns_col = Ra,    #define em qual coluna esta os retornos
               weights = mapa_pesos,#define o objeto com os pesos
               col_rename = "Ra")   #renomeia a coluna resultante
```

```
## # A tibble: 9 x 2
##   timestamp       Ra
##   <date>       <dbl>
## 1 2012-12-28  0.158
## 2 2013-12-30 -0.0990
## 3 2014-12-30  0.0873
## 4 2015-12-30 -0.162
## 5 2016-12-29  0.478
## 6 2017-12-29  0.336
## 7 2018-12-28  0.211
## 8 2019-12-30  0.312
## 9 2020-05-15 -0.285
```

```
carteira_retornos_anuais
```

```
## # A tibble: 9 x 3
##   timestamp       Ra  date
##   <date>       <dbl> <dbl>
## 1 2012-12-28  0.158   2012
## 2 2013-12-30 -0.0990  2013
## 3 2014-12-30  0.0873  2014
## 4 2015-12-30 -0.162   2015
## 5 2016-12-29  0.478   2016
## 6 2017-12-29  0.336   2017
## 7 2018-12-28  0.211   2018
## 8 2019-12-30  0.312   2019
## 9 2020-05-15 -0.285   2020
```

Step 3B: Unindo Ra e Rb Agora precisamos unir os retornos da carteira e do benchmark em um único ativo. Fazemos isso por meio da função **left_join()** da seguinte forma:

```
carteira_e_benchmark <-
  carteira_retornos_anuais %>%
  left_join(IBOV_retornos_anuais,
            by = "date")  %>% #define qual coluna sera usada de referencia para a juncao
  mutate(timestamp.y = NULL, date = NULL)
carteira_e_benchmark
```

```
## # A tibble: 9 x 3
##   timestamp.x      Ra      Rb
##   <date>        <dbl>   <dbl>
## 1 2012-12-28   0.158   0.0540
## 2 2013-12-30  -0.0990 -0.155
## 3 2014-12-30   0.0873 -0.0291
## 4 2015-12-30  -0.162  -0.133
## 5 2016-12-29   0.478   0.389
## 6 2017-12-29   0.336   0.269
## 7 2018-12-28   0.211   0.150
## 8 2019-12-30   0.312   0.319
## 9 2020-05-15  -0.285  -0.326
```

Step 4: Analyze Performance Nesse último estágio aplicamos a função **Table.CAPM** dentro da função **PerformanceAnalytics**.

```
carteira_e_benchmark %>%
  tq_performance(Ra = Ra,
                 Rb = Rb,
                 performance_fun = table.CAPM)
```

```
## # A tibble: 1 x 12
##   ActivePremium  Alpha AnnualizedAlpha  Beta `Beta-` `Beta+` Correlation
##           <dbl>  <dbl>           <dbl> <dbl>   <dbl>   <dbl>       <dbl>
## 1        0.0539 0.0537          0.0537  1.03    1.18   0.875       0.982
## # ... with 5 more variables: `Correlationp-value` <dbl>,
## #   InformationRatio <dbl>, `R-squared` <dbl>, TrackingError <dbl>,
## #   TreynorRatio <dbl>
```

Funções disponíveis

Assim como no caso da função **tq_transmute()** existe um conjunto muito grande de funções compatíveis com a função **tq_performance()**. Não é objetivo desse material explorar todos eles por isso, ao invés de descrever uma por uma, indicaremos apenas como acessar todas elas. Podemos fazer isso através da função:

```
tq_performance_fun_options()
```

```
## $table.funs
##  [1] "table.AnnualizedReturns" "table.Arbitrary"
##  [3] "table.Autocorrelation"   "table.CAPM"
##  [5] "table.CaptureRatios"     "table.Correlation"
##  [7] "table.Distributions"     "table.DownsideRisk"
##  [9] "table.DownsideRiskRatio" "table.DrawdownsRatio"
## [11] "table.HigherMoments"     "table.InformationRatio"
## [13] "table.RollingPeriods"    "table.SFM"
## [15] "table.SpecificRisk"      "table.Stats"
## [17] "table.TrailingPeriods"   "table.UpDownRatios"
## [19] "table.Variability"
##
## $CAPM.funs
##  [1] "CAPM.alpha"       "CAPM.beta"        "CAPM.beta.bear"   "CAPM.beta.bull"
##  [5] "CAPM.CML"         "CAPM.CML.slope"   "CAPM.dynamic"     "CAPM.epsilon"
##  [9] "CAPM.jensenAlpha" "CAPM.RiskPremium" "CAPM.SML.slope"   "TimingRatio"
## [13] "MarketTiming"
##
## $SFM.funs
## [1] "SFM.alpha"       "SFM.beta"        "SFM.CML"         "SFM.CML.slope"
## [5] "SFM.dynamic"     "SFM.epsilon"     "SFM.jensenAlpha"
##
## $descriptive.funs
## [1] "mean"           "sd"             "min"            "max"
## [5] "cor"            "mean.geometric" "mean.stderr"    "mean.LCL"
## [9] "mean.UCL"
##
## $annualized.funs
```

```
## [1] "Return.annualized"        "Return.annualized.excess"
## [3] "sd.annualized"            "SharpeRatio.annualized"
##
## $VaR.funs
## [1] "VaR"  "ES"   "ETL"  "CDD"  "CVaR"
##
## $moment.funs
##  [1] "var"              "cov"              "skewness"         "kurtosis"
##  [5] "CoVariance"       "CoSkewness"       "CoSkewnessMatrix" "CoKurtosis"
##  [9] "CoKurtosisMatrix" "M3.MM"            "M4.MM"            "BetaCoVariance"
## [13] "BetaCoSkewness"   "BetaCoKurtosis"
##
## $drawdown.funs
## [1] "AverageDrawdown"   "AverageLength"     "AverageRecovery"
## [4] "DrawdownDeviation" "DrawdownPeak"      "maxDrawdown"
##
## $Bacon.risk.funs
## [1] "MeanAbsoluteDeviation" "Frequency"             "SharpeRatio"
## [4] "MSquared"              "MSquaredExcess"        "HurstIndex"
##
## $Bacon.regression.funs
##  [1] "CAPM.alpha"       "CAPM.beta"        "CAPM.epsilon"     "CAPM.jensenAlpha"
##  [5] "SystematicRisk"   "SpecificRisk"     "TotalRisk"        "TreynorRatio"
##  [9] "AppraisalRatio"   "FamaBeta"         "Selectivity"      "NetSelectivity"
##
## $Bacon.relative.risk.funs
## [1] "ActivePremium"    "ActiveReturn"     "TrackingError"    "InformationRatio"
##
## $Bacon.drawdown.funs
## [1] "PainIndex"     "PainRatio"     "CalmarRatio"   "SterlingRatio"
## [5] "BurkeRatio"    "MartinRatio"   "UlcerIndex"
##
## $Bacon.downside.risk.funs
##  [1] "DownsideDeviation"     "DownsidePotential"     "DownsideFrequency"
##  [4] "SemiDeviation"         "SemiVariance"          "UpsideRisk"
##  [7] "UpsidePotentialRatio"  "UpsideFrequency"       "BernardoLedoitRatio"
## [10] "DRatio"                "Omega"                 "OmegaSharpeRatio"
## [13] "OmegaExcessReturn"     "SortinoRatio"          "M2Sortino"
## [16] "Kappa"                 "VolatilitySkewness"    "AdjustedSharpeRatio"
## [19] "SkewnessKurtosisRatio" "ProspectRatio"
##
## $misc.funs
## [1] "KellyRatio"   "Modigliani"   "UpDownRatios"
```

Mais detalhes sobre as funções podem ser encontradas na documentação delas.

Customizando usando o ...

A função **tq_performance()** é apenas um wrapper para a função original **Return.portfolio()** por isso podemos usar os argumentos da segunda na primeira. Para acessar os argumentos disponíveis podemos utilizar a documentação da função, disponíveis através da função **help()**, ou diretamente através da função **args()**:

```r
args(Return.portfolio)
```

```
## function (R, weights = NULL, wealth.index = FALSE, contribution = FALSE,
##     geometric = TRUE, rebalance_on = c(NA, "years", "quarters",
##         "months", "weeks", "days"), value = 1, verbose = FALSE,
##     ...)
## NULL
```