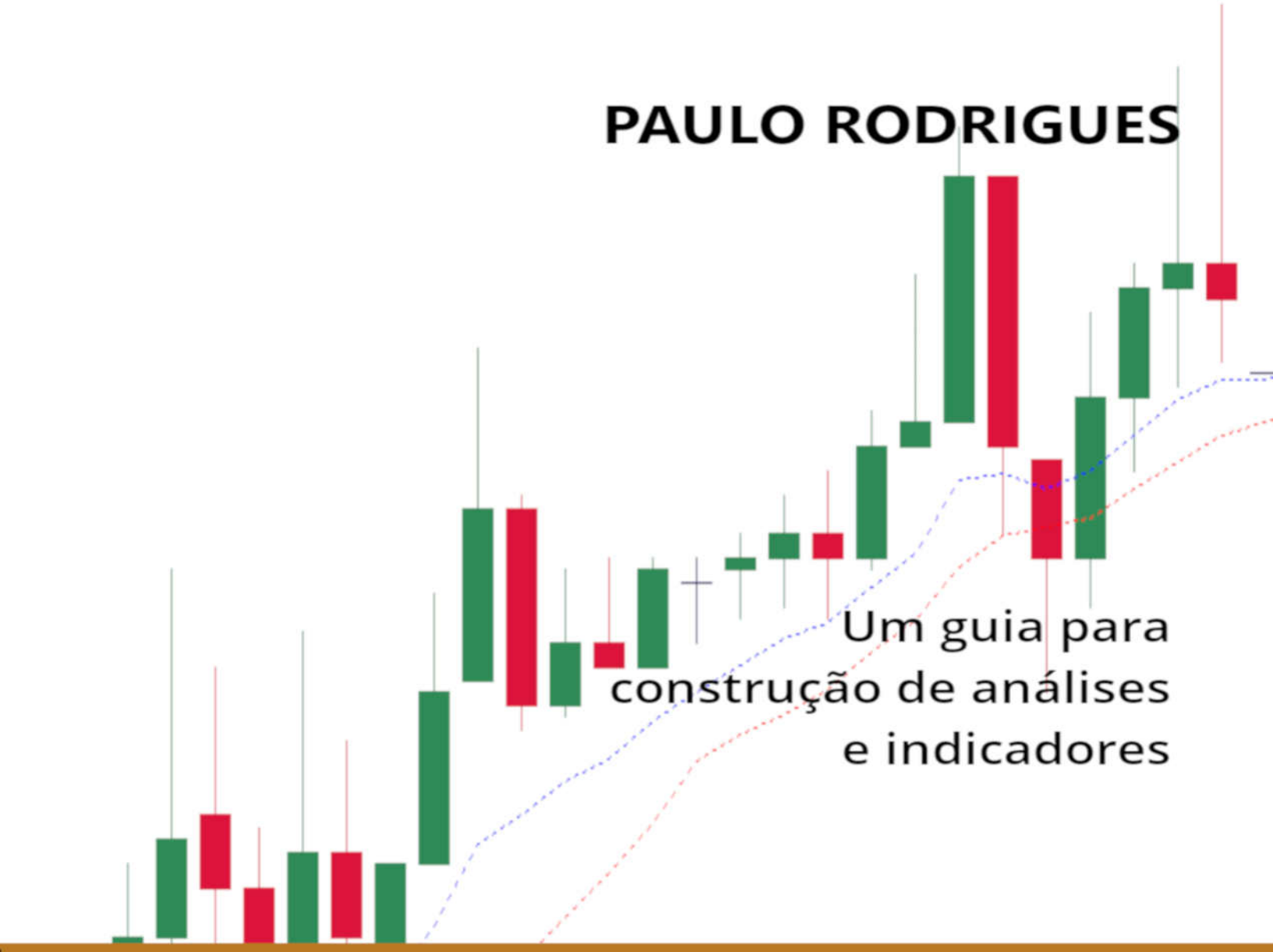
PAULO RODRIGUES



Um guia para
construção de análises
e indicadores

Python Aplicado

Bolsa de Valores

1º Edição

Sumário

Sobre o autor

Paulo Rodrigues atua como desenvolvedor e arquiteto de *software* há mais de 15 anos. Apaixonado por pesquisa e inovação, tendo Python como sua linguagem de programação favorita. Atualmente reside no Brasil e durante seu tempo livre gosta de estudar estratégias de *trading* e suas automatizações.

Como entrar em contato

Envie seu comentários e suas dúvidas sobre este livro através de uma das minhas redes sociais, disponíveis na página web www.codekraft.com.br ou www.codekraft.co. Lá você também encontrará erratas, exemplos e quaisquer outras informações adicionais.

Prefácio

Você é um programador que gosta de finanças, ou um investidor que gosta de programação?

Independentemente da resposta, o que temos de concreto é que a cada dia passamos a ter mais e mais dados disponíveis em todas as áreas de conhecimento que nos cerca e o universo das finanças não é exceção.

Devido a esse grande volume de informações, não somos mais capazes de analisá-los de forma manual, ou ao menos, levaríamos muito tempo para fazê-lo. Nesse sentido, a tecnologia passa a ter um papel valioso no mundo das finanças, pois permite trabalharmos na análise de uma grande quantidade de dados gastando pouquíssimo tempo.

Podemos realizar a comparação de ativos e suas operações diárias com rapidez, automatizar operações que antes eram manuais e demandavam muito tempo ou ainda delegar processos de decisão para algoritmos computacionais. Com este viés, o objetivo deste guia é demonstrar ao leitor como é possível implementar alguns indicadores e estratégias utilizadas em bolsa de valores de forma fácil e rápida, utilizando o poder computacional.

Python, devido a sua versatilidade e a sua consolidada comunidade, vem sendo usado já há alguns anos no campo de ciência de dados e aprendizado de máquina. E é relativamente fácil encontrar bibliotecas para análises de dados ou até mesmo análises financeiras para *python*.

Aqui passaremos por algumas análises estatísticas, criação de indicadores, como cruzamento de médias móveis, HiLo, MACD, Bandas de Bollinger, IFR, ADX, etc. Ainda criaremos um sistema de *backtesting* do zero, para testarmos nossos indicadores e estratégias.

Ao contrário de muitos livros sobre *python* aplicado ao mundo financeiro*,* onde muitas vezes metade do livro é dedicado a introdução a linguagem de programação, este guia pretende ser o mais objetivo possível. Não trataremos de assuntos básicos da linguagem. Logo, é esperado que o leitor possua alguns conhecimentos básicos em *python.*

Outro ponto fundamental que difere este livro dos demais, é a aplicabilidade no mundo real. Entre os livros que abordam o mundo das finanças em *python*, a grande maioria é acadêmico e não apresenta conteúdo que funcione no mercado do dia a dia, ou então apresentam análises puramente estatísticas que funcionam muito bem em dados históricos, mas falham ao analisar o mundo real. Aqui vale um ponto de alerta: devemos levar em conta que ganhos passados não representam ganhos futuros.

Este guia é dividido em três partes. A primeira parte trata de uma análise de portfólio e comparativos entre ações. A segunda parte, e mais densa, trata da implementação dos indicadores mais populares conhecidos na análise técnica, como médias móveis, HiLo, IFR, OBV, ADX, etc. Na última parte, desenvolvemos uma estratégia de média móvel e um sistema de *backtesting.*

Ainda é válido lembrar que os exemplos encontrados neste guia não representam indicação ou recomendação de compra ou venda, nem garantem ganhos futuros, mas trazem ao leitor uma visão de importantes indicadores utilizados no dia a dia de muitos investidores e como implementá-los e automatizá-los utilizando *python*.

I

Começando...

Funcionamento do mercado de ações

O produto conhecido como ação é o que podemos chamar de menor parte ou fração de uma dada empresa. Digamos que é a menor parte que você poderia comprar de uma empresa. Por consequência, quando você adquire uma ação, você está de certa forma tornando-se “sócio” de alguma companhia. A partir daí, poderá ganhar dinheiro com a valorização de seus ativos (ações) e/ou com o lucro da empresa, distribuídos como dividendos ou juros sobre capital próprio.

Geralmente as ações são negociadas na chamada bolsa de valores, e cada país tem a sua própria, podendo inclusive, existir mais de uma bolsa de valores no mesmo país. No Brasil a bolsa de valores está sediada em São Paulo e é conhecida como B3, onde são negociados ativos do mercado à vista e do mercado futuro. Para negociar um ativo sendo pessoa física, costuma-se utilizar um intermediador, que pode ser um banco ou uma corretora de valores, e é através deles que uma ordem de compra ou de venda é enviada para a bolsa de valores.

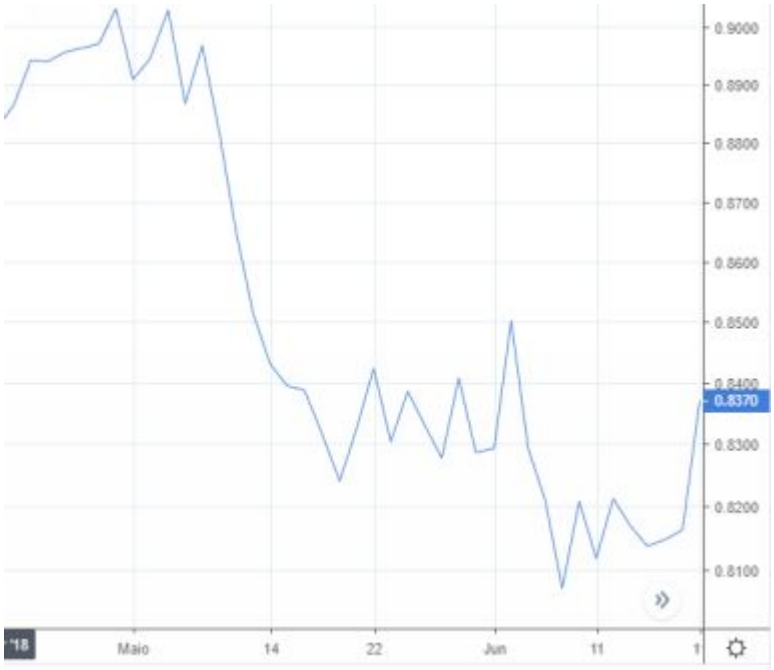
Conforme as ordens de compras e vendas são executadas ao longo dos dias, semanas e meses, um histórico de negociação vai sendo formado, e analisando este histórico é possível realizar uma avaliação ou medição do desempenho de um ativo. A obtenção desse histórico pode ser feita de várias formas, incluindo a obtenção de dados em tempo real. Mas para fins de estudos, trabalharemos com uma base de dados históricos provida pelo Yahoo.

Quando queremos nos referenciar a uma ação, geralmente utilizamos o seu código. Por exemplo, a empresa VALE tem o código definido como VALE3 na B3. Já a Apple tem seu código definido como AAPL na Nasdaq e a Microsoft utiliza o código MSFT. E é através destes códigos que recuperaremos os dados históricos na API fornecida pela Yahoo.

Séries temporais

Uma série temporal é uma sequência numérica cuja distribuição é dada por um período de tempo. Em investimentos, mais especificamente nos nossos exemplos, usaremos o período diário. Isso quer dizer que todos nossos dados estarão divididos em intervalos de 1 dia. Assim teremos, por exemplo, uma ação X, no dia Y com o preço Z.

Porém, cada ação exibe quatro preços distintos em cada dia. Sendo o preço de abertura do dia, preço máximo atingido no dia, preço mínimo atingido no dia e preço de fechamento do dia, além do volume de negócios ou volume financeiro no dia. Mais tarde veremos como interpretar e desenhar estes preços.



O ambiente de desenvolvimento

Nosso ambiente de desenvolvimento não será nada complicado. Nós só precisaremos do *python* e algumas poucas bibliotecas auxiliares.

Logo, o primeiro passo é você se certificar de que possua o P*ython 3* instalado no seu computador. Uma vez que *python* esteja instalado, devemos realizar a criação de um ambiente virtual.

Um ambiente virtual permite que possamos criar ambientes *pythons* e instalar bibliotecas de forma isolada do restante do sistema, não comprometendo o ambiente principal do *python* na máquina. É possível até mesmo ter ambientes virtuais rodando diferentes versões do *python.*

Para criar um ambiente virtual utilizaremos o utilitário nativo do *python,* chamado venv:

```
python -m venv python-aplicado
cd python-aplicado
Scripts\activate
mkdir lessons
cd lessons
```

Para linux:

```
python -m venv python-aplicado
cd python-aplicado
source bin/activate
mkdir lessons
cd lessons
```

O que fizemos foi criar um ambiente virtual chamado "python-aplicado" e em seguida ativar o ambiente com o comando *"activate"*.

O terceiro passo é a instalação do *Jupyter Notebook*. O *Jupyter Notebook* é uma aplicação *web* que permite que possamos criar, executar e compartilhar códigos, documentos, visualizar gráficos, entre várias outras coisas. Um excelente tutorial de sua utilização pode ser visto em https://www.datacamp.com/community/tutorials/tutorial-jupyter-notebook.

Junto do *Jupyter Notebook,* iremos instalar outras bibliotecas. Isto pode ser feito como visto a seguir:

```
python -m pip install --upgrade pip
pip install pandas-datareader
pip install jupyterlab
pip install plotly
```

Por fim, recomendo que você clone o repositório git contendo o notebook com todos os códigos utilizados neste livro:

```
git clone https://github.com/paulorodriguesxv/applied-python-stock-exchange.git
```

Para iniciar o notebook:

```
cd applied-python-stock-exchange
jupyter lab lesson.ipynb
```

Agora que estamos com nosso ambiente em execução, podemos falar um pouco a respeito das bibliotecas principais que utilizaremos ao longo dos nossos projetos. São basicamente 3 bibliotecas:

Pandas

Conforme veremos em nossos exemplos, precisaremos fazer algumas manipulações de dados, hora submetendo um conjunto de dados à equação, hora extraindo informações para construção de indicadores ou ainda comparando diferentes conjuntos de dados. E por mais que o *python* seja uma linguagem de programação muito prática, todas essas manipulações se tornariam custosas para serem implementadas. Aqui é onde surge a biblioteca chamada *Pandas.*

*S*endo umas das bibliotecas mais populares no mundo *python* voltado à análise de dados, esta biblioteca foi criada por volta de 2008, com o objetivo de auxiliar na manipulação de estruturas de dados, como *arrays*, matrizes multidimensionais, série de dados etc. Com ela, seus dados podem ser armazenados em diversos formatos, entre os mais populares estão as *Series* e os *DataFrames.*

Pandas Datareader

Antes mesmo que possamos pensar em criar qualquer sistema ou automatizar qualquer análise, precisamos de uma fonte de dados, um lugar que possamos buscar os dados financeiros que queremos analisar.

Podemos obter dados de ações de diferentes provedores de dados, como o Yahoo, Google e Alpha Vantage, e utilizaremos a biblioteca chamada *pandas-datareader* para trabalhar com estes dados. Com ela é possível converter os dados em *DataFrames* para serem manipulados através de funções da biblioteca *Pandas*.

Plotly

Um aspecto importante em qualquer análise de dados é como estes serão visualizados. Para *python* existem muitas bibliotecas para este fim, como o *matplotlib, seaborn e Altair*. Porém utilizaremos a biblioteca chamada *Plotly*.

Plotly é uma empresa que fornece uma vasta biblioteca para criação de gráficos, *dashboards* e análises estatísticas. Eles fornecem um SDK para *python, javascript e R*. Aqui iremos usá-la como alternativa ao *matplot* para plotagem de gráficos, principalmente devido à maior simplicidade para construção de determinados gráficos, como gráficos de *candlestick* que iremos ver adiante. Para habilitar o uso dos gráficos do Plotly é necessário realizar um cadastro e gerar uma chave no site https://plot.ly/python/

Trabalhando com os dados

Antes de mergulharmos em nossos indicadores e estratégias, vamos exercitar de maneira rápida, alguns comandos e conceitos. Começando pela forma como buscaremos nossos dados.

Nossos dados serão obtidos através da biblioteca *pandas-datareader.* Esta biblioteca nos permite buscar dados de diferentes fontes.

Vamos experimentar:

```
import pandas_datareader as pdr
import datetime
import numpy as np
vale3 = pdr.get_data_yahoo('VALE3.SA',
                           start=datetime.datetime(2016, 10, 1),
                           end=datetime.datetime(2019, 1, 1))
```

Para termos uma noção geral com que tipo de dados estamos trabalhando, muitas vezes, olhamos uma pequena amostragem do conjunto de dados, seja do início ou do final do conjunto.

Para lermos as primeiras 5 linhas deste conjunto podemos utilizar o comando *head:*

```
# get first 5 rows
display(vale3.head())
```

Date	High	Low	Open	Close	Volume	Adj Close
2016-10-03	18.270000	17.750000	17.850000	18.100000	3689100.0	16.558498
2016-10-04	18.299999	17.559999	18.200001	17.650000	5321900.0	16.146822
2016-10-05	18.010000	17.709999	17.879999	17.860001	3221600.0	16.338940
2016-10-06	17.950001	17.700001	17.840000	17.799999	4178300.0	16.284048
2016-10-07	18.280001	17.719999	17.969999	18.000000	10040900.0	16.467014

Para lermos as 5 últimas linhas do conjunto de dados, podemos utilizar o comando *tail:*

```
# get last 5 rows
display(vale3.tail())
```

Date	High	Low	Open	Close	Volume	Adj Close
2018-12-21	51.250000	50.220001	50.279999	50.860001	28272600.0	50.860001
2018-12-26	50.700001	49.400002	50.110001	50.439999	13356400.0	50.439999
2018-12-27	50.389999	49.279999	50.099998	49.500000	14332000.0	49.500000
2018-12-28	51.200001	50.250000	50.310001	51.000000	11914300.0	51.000000
2019-01-02	51.369999	49.790001	50.009998	51.090000	17319600.0	51.090000

Podemos também analisar rapidamente como estão distribuídos os dados em nosso conjunto:

```
    # get statistcs of dataset
display(vale3.describe())
```

	High	Low	Open	Close	Volume	Adj Close
count	562.000000	562.000000	562.000000	562.000000	5.620000e+02	562.000000
mean	39.292349	38.187829	38.742651	38.741566	1.340595e+07	37.324981
std	11.174830	10.944557	11.084350	11.056554	8.204266e+06	11.597862
min	17.950001	17.559999	17.840000	17.650000	0.000000e+00	16.146822
25%	30.180000	29.224999	29.685000	29.650000	6.294125e+06	27.622377
50%	35.754999	34.809999	35.265001	35.230000	1.305660e+07	33.391148
75%	50.434999	49.057501	49.697500	49.675000	1.881795e+07	48.307610
max	62.419998	60.480000	62.200001	62.200001	5.642970e+07	62.200001

O *DataFrame* retornado pelo pandas é composto por índices e colunas. Em nossos exemplos, uma vez que trabalharemos com séries temporais, os índices serão quase sempre datas, representando os dias.

```
    # Inspect the index
    print(vale3.index)
    # Inspect the columns
    print(vale3.columns)
```

```
DatetimeIndex(['2016-10-03', '2016-10-04', '2016-10-05', '2016-10-06',
               '2016-10-07', '2016-10-10', '2016-10-11', '2016-10-13',
               '2016-10-14', '2016-10-17',
               ...
               '2018-12-14', '2018-12-17', '2018-12-18', '2018-12-19',
               '2018-12-20', '2018-12-21', '2018-12-26', '2018-12-27',
               '2018-12-28', '2019-01-02'],
              dtype='datetime64[ns]', name='Date', length=562, freq=None)
Index(['High', 'Low', 'Open', 'Close', 'Volume', 'Adj Close'], dtype='object')
```

Como nosso conjunto é indexado por dada, podemos facilmente filtrar os períodos que gostaríamos de analisar. Por exemplo, para retornar os 10 primeiros registros, a partir de 01/01/2018, podemos executar:

```
vale3['2018-01-01':].head(10)
```

Date	High	Low	Open	Close	Volume	Adj Close
2018-01-02	41.740002	40.439999	40.439999	41.720001	14156500.0	40.095676
2018-01-03	41.880001	41.299999	41.830002	41.470001	12744200.0	39.855412
2018-01-04	42.369999	41.520000	41.810001	41.639999	18433000.0	40.018787
2018-01-05	42.290001	41.310001	41.570000	42.290001	15251300.0	40.643486
2018-01-08	43.230000	42.400002	42.400002	43.230000	14542800.0	41.546883
2018-01-09	43.750000	42.930000	43.580002	43.070000	15986200.0	41.393112
2018-01-10	42.950001	42.419998	42.740002	42.470001	12149500.0	40.816479
2018-01-11	43.299999	42.599998	42.599998	43.299999	10236300.0	41.614159
2018-01-12	43.660000	42.750000	42.950001	43.549999	15014300.0	41.854427
2018-01-15	43.720001	43.270000	43.500000	43.470001	7253400.0	41.777542

Agora um exemplo mais prático. Digamos que gostaríamos de descobrir a variação percentual intradiária de um ativo, isto é, a variação entre o preço de abertura e o preço de fechamento para um ativo dentro em um determinado dia. Para isso precisamos, para cada linha do conjunto de dados, dividir o preço de fechamento pelo preço de abertura. Graças ao pandas, essa atividade é extremamente fácil. Basta dividir as colunas do *DataFrame*, de forma direta.

```
vale3.Close / vale3.Open
```

Porém, como queremos os valores em percentuais, precisamos aplicar uma pequena transformação, com auxílio da função *apply*. Uma variação positiva indica que o ativo subiu de preço ao longo do dia. Enquanto que uma variação negativa indica que o ativo sofreu uma queda de preço naquele dia.

```
pct_c = (vale3.Close / vale3.Open).apply(lambda x: x - 1) * 100
display(pct_c.head(10))
```

```
Date
2016-10-03    1.400560
2016-10-04   -3.021984
2016-10-05   -0.111849
2016-10-06   -0.224220
2016-10-07    0.166949
2016-10-10    4.440787
2016-10-11   -1.483044
2016-10-13   -2.739726
2016-10-14   -0.441989
2016-10-17    0.829185
dtype: float64
```

Há uma maneira mais elegante de analisar a variação do preço do ativo, que é através de um gráfico de distribuição. Isso é útil para determinarmos se a variação do ativo é normalmente distribuída. E dependendo da resposta, nossas estratégias deverão ser adaptadas.

O cálculo da variação é simples, e utilizaremos escala logarítmica para melhor representar os dados.

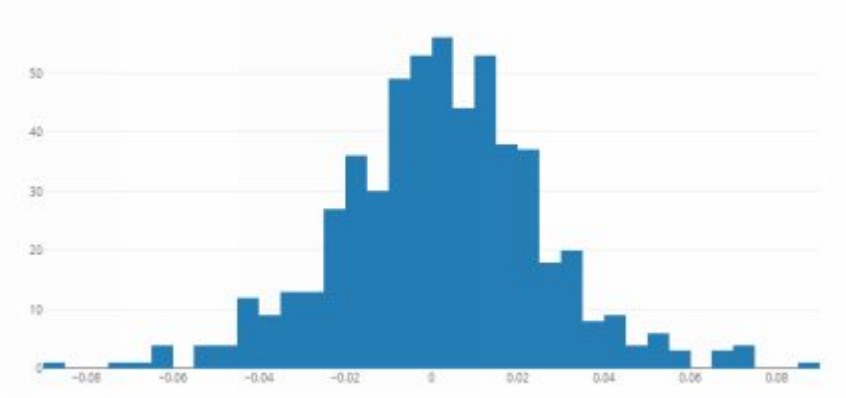
```
# Daily log returns
daily_log_returns = np.log(vale3.Close.pct_change()+1)
```

Aqui aplicamos duas novas funções: *np.Log* e *pct_change.* A primeira calcula o logaritmo natural de um número. Enquanto a segunda, realiza o cálculo da variação percentual entre o elemento atual e o elemento anterior de um dado conjunto de dados.

Com isso em mãos, conseguimos construir um gráfico, através do uso da biblioteca *plotly.*

```
import plotly.plotly as py
plotly_username = "username"
plotly_key = "api_ley"
# auth to plotly
py.sign_in(plotly_username, plotly_key)
data = [go.Histogram(x=daily_log_returns)]
py.iplot(data, filename='basic histogram')
```

Como pode ser observado no gráfico de distribuição, temos o “formato de sino” característico de uma distribuição normal.



Buscando dados históricos de ações

Agora que já nos familiarizamos um pouco com nossas bibliotecas, podemos começar a trabalhar em nossas implementações. Começaremos por buscar nossos dados, não apenas de uma única ação, mas de um conjunto delas.

```
import pandas as pd
import numpy as np
import plotly.plotly as py
import datetime
import pandas_datareader as pdr
import plotly.graph_objs as go
```

Vamos criar uma função que faça o download dos dados a partir da base de dados do Yahoo e os agrupe por ação, ordenando-os por data. Esta função além de invocar o download, realizará um mapeamento entre os dados buscados e o ativo.

```
  def get(tickers, startdate, enddate):
def data(ticker):
```

```
    return (pdr.get_data_yahoo(ticker, start=startdate, end=enddate))
    datas = map (data, tickers)
    return(pd.concat(datas, keys=tickers, names=['Ticker', 'Date']))
```

Executaremos o download dos dados da Apple, Microsoft e do "S&P500", que é o índice americano das 500 empresas listadas na NYSE ou NASDAQ, qualificados devido ao seu tamanho de mercado, sua liquidez e sua representação de grupo industrial.

```
    tickers = ['AAPL', 'MSFT', '^GSPC']
    start_date = datetime.datetime(2016, 1, 1)
    end_date = datetime.datetime(2018, 12, 31)
    all_data = get(tickers, start_date, end_date)
```

A função chave aqui é a **get_data_yahoo**, que é responsável por buscar os dados históricos dos ativos. Dentre os dados retornados, temos o preço de abertura, o preço de fechamento, o preço máximo e o preço mínimo que o ativo atingiu no dia e o volume de ações negociados. Mais pra frente falaremos a respeito destes dados, quando trabalharmos com *candlestick*.

Primeiro vamos dar uma olhada nos nossos dados, através da função *head*, que realiza uma pré-visualização de parte do nosso *dataframe*

```
    # first 10 records
    all_data.head(10)
```

		High	Low	Open	Close	Volume	Adj Close
Ticker	Date						
AAPL	2016-01-04	105.370003	102.000000	102.610001	105.349998	67649400.0	99.499107
	2016-01-05	105.849998	102.410004	105.750000	102.709999	55791000.0	97.005730
	2016-01-06	102.370003	99.870003	100.559998	100.699997	68457400.0	95.107361
	2016-01-07	100.129997	96.430000	98.680000	96.449997	81094400.0	91.093399
	2016-01-08	99.110001	96.760002	98.550003	96.959999	70798000.0	91.575073
	2016-01-11	99.059998	97.339996	98.970001	98.529999	49739400.0	93.057869
	2016-01-12	100.690002	98.839996	100.550003	99.959999	49154200.0	94.408447
	2016-01-13	101.190002	97.300003	100.320000	97.389999	62439600.0	91.981194
	2016-01-14	100.480003	95.739998	97.959999	99.519997	63170100.0	93.992889
	2016-01-15	97.709999	95.360001	96.199997	97.129997	79010000.0	91.735634

```
# last 10 records
all_data.tail(10)
```

		High	Low	Open	Close	Volume
Ticker	Date					
^GSPC	2018-12-17	2601.129883	2530.540039	2590.750000	2545.939941	4.616350e+09
	2018-12-18	2573.989990	2528.709961	2559.899902	2546.159912	4.470880e+09
	2018-12-19	2585.290039	2488.959961	2547.050049	2506.959961	5.127940e+09
	2018-12-20	2509.629883	2441.179932	2496.770020	2467.419922	5.585780e+09
	2018-12-21	2504.409912	2408.550049	2465.379883	2416.620117	7.609010e+09
	2018-12-24	2410.340088	2351.100098	2400.560059	2351.100098	2.613930e+09
	2018-12-26	2467.760010	2346.580078	2363.120117	2467.699951	4.233990e+09
	2018-12-27	2489.100098	2397.939941	2442.500000	2488.830078	4.096610e+09
	2018-12-28	2520.270020	2472.889893	2498.770020	2485.739990	3.702620e+09
	2018-12-31	2509.239990	2482.820068	2498.939941	2506.850098	3.442870e+09

Ganho da carteira

Como primeiro exercício, vamos analisar a rentabilidade da carteira para o período que selecionamos. Supondo que tivéssemos comprado algumas ações no início do período e vendido ao final, só precisaríamos calcular a relação entre o preço do início do período com o preço do final do período para obter o rendimento, porém, a fim de exemplificarmos algumas funções e cálculos que possam ser úteis, faremos algumas operações intermediárias.

Por exemplo, para descobrirmos a variação percentual diária de um ativo, podemos usar a função *shift* do *pandas*. A função *shift* desloca o índice da lista para o valor passado como parâmetro e retorna uma nova lista com base nesse deslocamento. Neste exemplo, estamos dividindo a lista de valores atuais por ela mesma, deslocada em -1, ou seja, pelo valor anterior.

```
# transform rows into columns
daily_close_px = all_data[['Adj Close']].reset_index().pivot('Date', 'Ticker', 'Adj Close')
# get the actual price and divide by previous price to find the price variation
cart_pct_change = daily_close_px / daily_close_px.shift(1) - 1
```

Também podemos verificar o valor acumulado diariamente, através da função cumprod, que retorna o valor acumulado da lista, neste exemplo, o percentual acumulado.

```
# sum all prices variation percentages
cart_return = (1 + cart_pct_change).cumprod()
cart_return.head(10)
```

Por fim, para verificarmos o rendimento final da carteira, basta visualizar o último valor da lista de produtos acumulados.

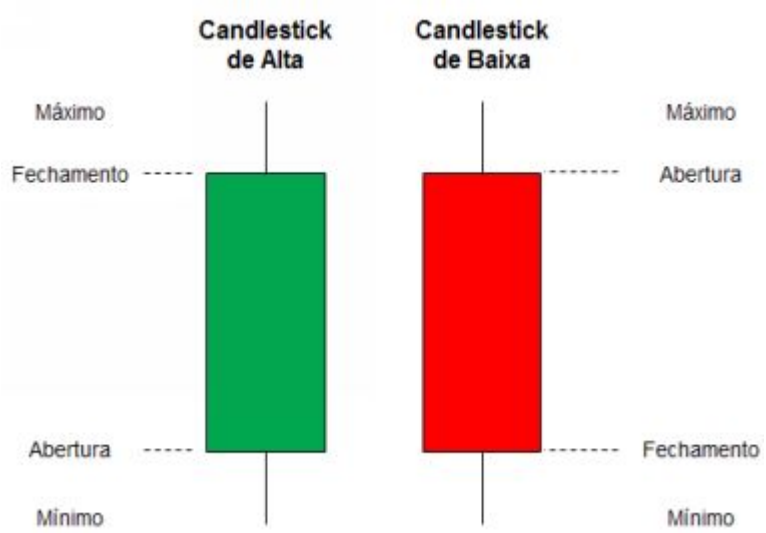
```
# percentual changes
final_return = cart_return[-1:].apply(lambda x: (x - 1) *100)
print(final_return)
```

Ticker	AAPL	MSFT	^GSPC
Date			
2018-12-31	57.85707	98.055033	24.554075

Como veremos mais adiante, está forma simplista não deveria ser utilizada, ainda mais tratando-se de um período tão longo e sem considerar o percentual da carteira como um todo.

Plotando o candlestick

Uma técnica muito utilizada por investidores é a análise técnica ou análise gráfica, sendo esta análise fortemente baseada no que chamamos de *candlestick* ou candelabros japoneses. Ao contrário do gráfico de linhas, onde conseguimos representar apenas um dos quatro preços disponíveis do ativo (abertura, fechamento, máxima ou mínima), o *candlestick* dá a possibilidade de representar todos os preços no mesmo gráfico. Esse formato de representação de preços tem origem no Japão do século XVIII, nas negociações de arroz da época, sendo popularizado por Steve Nison nos Estados Unidos.



O *candlestick* apresenta quatro preços e, se estivermos trabalhando com uma periodicidade diária, representará o preço de abertura da ação no dia, o preço no fechamento, o preço máximo e o preço mínimo que a ação atingiu no dia. Um *candlestick* de alta significa que o preço de fechamento foi mais alto que o preço de abertura, enquanto que o *candlestick* de baixa significa que o preço de abertura foi menor que o preço de fechamento.

Tomando as ações da Microsoft como exemplo para plotagem, primeiro precisamos obter estes quatro preços, que para nossa sorte, estão disponíveis entre os dados retornados pelo *pandas-datareader,* faltando somente plotá-los através da função *Candlestick* do *plotly*.

```
trace = go.Candlestick(x=all_data.loc['MSFT'].index,
            open=all_data.loc['MSFT'].Open,
            high=all_data.loc['MSFT'].High,
```

```
            low=all_data.loc['MSFT'].Low,
            close=all_data.loc['MSFT'].Close)
data = [trace]
py.iplot(data)
```

Preparando os dados

Para efeitos iniciais de estudo quantitativo, por hora trabalharemos apenas com o preço de fechamento do mercado (*close*), que em geral, é o valor utilizado na grande maioria das estratégias de cálculo, ou, pelo menos, nas mais simples. Assim, precisamos efetuar alguma limpeza nos dados, por exemplo, nós queremos ter certeza de que todos os dias da semana estejam no *dataset*, mas nem sempre isso acontece. Imagine por exemplo os dias que foram feriados e a bolsa não teve expediente. Nesta situação o *dataset* retornado pelo Yahoo (*get_data_yahoo*) não terá registro, necessitando que criemos registros para os dias faltantes.

Primeiro reorganizaremos a estrutura do *dataset* para facilitar a manipulação dos dados e para pegarmos o preço de fechamento.

```
# rearrange data
data = all_data.reset_index()
data = data.set_index(['Date','Ticker']).sort_index()
# Getting just the adjusted closing prices. This will return a Pandas DataFrame
# The index in this DataFrame is the major index of the panel_data.
close = data['Close']
```

Descobrimos quais foram os dias úteis no período dentro do range de data que selecionamos anteriormente. Então reindexamos nosso *dataset* com esse novo *range* de data. Isto fará com que datas inexistentes sejam criadas com os valores nulos (*NaN*)

```
# Getting all weekdays between 01/01/2000 and 31/12/2018
all_weekdays = pd.date_range(start=start_date, end=end_date, freq='B')
# How do we align the existing prices in adj_close with our new set of dates?
# All we need to do is reindex close using all_weekdays as the new index
close = close.reindex( pd.MultiIndex.from_product([all_weekdays, tickers], names=['Date',
'Ticker']),fill_value=np.NaN)
#close = close.fillna(0)
close.head(10)
```

Por fim, transformaremos a coluna ticker em 3 colunas distintas, uma para cada ativo.

```
close = close.reset_index().pivot(index="Date", columns="Ticker", values="Close")
```

Agora nosso dataset está pronto e podemos dar uma olhada nele.

close.describe()

Ticker	AAPL	MSFT	^GSPC
count	251.000000	251.000000	251.000000
mean	189.053427	101.033984	2746.214183
std	20.593861	7.917807	100.409310
min	146.830002	85.010002	2351.100098
25%	173.139999	93.989998	2690.295044
50%	186.050003	101.160004	2743.149902
75%	207.760002	108.005001	2814.755005
max	232.070007	115.610001	2930.750000

II

Buy and Hold…

Podemos dizer que existem três estratégias básicas de posicionamento em ações, que são conhecidas pelo nome de *day trade, swing trade e position trade.* Em *day trade,* compra-se e vende-se o ativo no mesmo dia, em *swing trade* mantemos a posição no ativo por alguns dias ou até semanas, e em *position trade,* compramos um ativo pensando no longo prazo, mantendo a posição por meses ou anos. Esta última modalidade também é conhecida por *Buy and Hold,* que em tradução livre é "comprar e manter". E é sobre essa estratégia que dedicaremos este capítulo.

Análise de preço

Como queremos um sistema de análise voltado para o longo prazo, este sistema deve nos permitir apurar qual o ganho que uma determinada carteira de ações possa produzir, a partir de um capital inicial. Pensando em análise técnica, poderíamos usar a série temporal de preços como temos feito até então e trabalhar com os números reais dos preços, como mostrado anteriormente. Porém numa análise puramente estatística, uma desvantagem em trabalhar com séries temporais de preço é que essa abordagem costuma conter séries temporais não estacionárias, o que estatisticamente é menos estável ao longo do tempo. Para amenizar esse problema, podemos usar

séries temporais baseadas não no preço real do ativo, mas na sua variação percentual, podendo ser obtido tanto pelo cálculo da variação relativa simples do preço, quanto pela variação relativa logarítmica do preço. Está fora do escopo deste guia entrar em explicações estatísticas a respeito de cada cálculo, mas vamos assumir como verdade que o cálculo em escala logarítmica produz melhores resultados. Com o pandas, o valor percentual é facilmente calculado pela função *pct_change.*

```
data = all_data.pct_change(1)
data.loc['MSFT'].Close.head()
```

```
Date
2016-01-04   -0.652593
2016-01-05    0.004562
2016-01-06   -0.018165
2016-01-07   -0.034783
2016-01-08    0.003067
Name: Close, dtype: float64
```

Note que pct_change é equivalente ao que fizemos no cálculo do ganho da carteira anteriormente.

```
(daily_close_px / daily_close_px.shift(1) - 1)['MSFT'].head()
```

```
Date
2016-01-04        NaN
2016-01-05   0.004562
2016-01-06  -0.018165
2016-01-07  -0.034783
2016-01-08   0.003067
Name: MSFT, dtype: float64
```

Uma vez que trabalharemos utilizando uma escala logarítmica, usaremos uma outra estratégia para cálculo do percentual. Primeiro aplicamos o logaritmo no valor do número e depois calculamos a diferença através função *diff*, que calcula a diferença dos valores ao longo de uma lista *(out[n] = a[n+1] – a[n])*.

```
log_returns = np.log(close.dropna()).diff()
log_returns.head()
```

Ticker	AAPL	MSFT	^GSPC
Date			
2016-01-04	NaN	NaN	NaN
2016-01-05	-0.025379	0.004552	0.002010
2016-01-06	-0.019764	-0.018332	-0.013202
2016-01-07	-0.043121	-0.035402	-0.023986
2016-01-08	0.005274	0.003062	-0.010898

Lembrando da propriedade do quociente de logaritmo:

$$\log_a \frac{b}{c} = \log_a b - \log_a c$$

Podemos dizer que o código *python* acima é o equivalente a esta equação:

$$r(t) = \log\left(\frac{p(t)}{p(t-1)}\right)$$

E a lista resultante é o somatório dado por:

$$c(t) - \sum_{k=1}^{t} r(t)$$

Além do preço relativo da ação, muitas vezes também trabalhamos com o total relativo acumulado, que é a medida que indica a rentabilidade de um dado período. Os gráficos abaixo representam o valor percentual da ação seguido pelo gráfico representando o total acumulado. Por exemplo, no gráfico de total acumulado, notamos que se uma pessoa tivesse investido $1000 na Apple no início dos anos 2000, teria rentabilizado seu investimento em quase 6 vezes.

```
  layout = go.Layout(
title='Result',
yaxis=dict(
      title='Cumulative log returns',
```

```
        titlefont=dict(
            size=18,
            color='#7f7f7f'
        )
)
  )
  axis = []
  for d in log_returns:
axi = go.Scatter(
                x=log_returns.index,
                y=log_returns[d].cumsum(),
                name = d,
                opacity = 1)
axis.append(axi)
  fig = dict(data=axis, layout=layout)
  py.iplot(fig)
```


```
  layout = go.Layout(
title='Result',
yaxis=dict(
        title='Total relative returns (%)',
        titlefont=dict(
            size=18,
            color='#7f7f7f'
        )
)
  )
  axis = []
  for d in log_returns:
axi = go.Scatter(
                x=log_returns.index,
                y= 100 * (np.exp(log_returns[d].cumsum()) -1),
                name = d,
                opacity = 1)
axis.append(axi)
  fig = dict(data=axis, layout=layout)
```

```
py.iplot(fig)
```

Análise do portfólio

Uma das estratégias conhecidas para redução de riscos em investimentos é a diversificação, ainda que haja controvérsias quanto isto. Um dos pontos contrários a esta estratégia é apresentado por Max Gunther no livro *Os Axiomas de Zurique.* De maneira geral o autor diz que mais vale uma única cesta com todos os ovos dentro do que dividí-los em cestas diferentes, contanto que esta cesta seja muito bem cuidada. Este seria um *trade off* entre o risco e o retorno. Por hora, assumiremos que nosso portfólio deve ser diversificado, por exemplo, contendo ações de empresas de diferentes setores da economia.

Dado este cenário, como podemos calcular o rendimento da carteira como um todo? Antes de respondermos a essa pergunta, precisamos tratar de dois conceitos importantes: operar comprado e operar vendido.

Operar comprado é a forma mais simples de lidar com ações e é por onde a maioria dos iniciantes começam a operar. Isso significa que você executou a compra de uma ação que não possuía, e uma vez que passa a ter a ação, você está posicionado **comprado** nela.

Por outro lado, operar **vendido** significa que você vendeu uma ação, sem tê-la. Isso mesmo, você pode vender algo que você não tenha, desde que consiga comprar a tempo para honrar esta venda no futuro. Essa

operação é um tanto mais complexa, pois envolve, entre outras coisas, a tomada de aluguel de ações, por isso ficará de fora do escopo deste capítulo. Trataremos de um portfólio onde operaremos apenas no modo comprado.

Vamos criar uma carteira fictícia, onde teremos inicialmente o montante de $10.000 para distribuir em ações de nossa escolha. Uma das formas de distribuir o montante inicial entre as ações que comporão a carteira é dando "pesos" para cada ação em termos de volume monetário. Por exemplo, em um portfólio com 3 ações, podemos dizer que a ação A terá o peso de 0.2, a ação B o peso de 0.5 e ação C o peso de 0.3. Observe que o somatório totaliza 1, representando 100% do valor disponível. Assim nosso capital estaria dividido em $2000 para A, $5000 para B e $3000 para C.

Vamos fazer um exemplo prático com nossos 3 ativos, AAPL, MSFT e ^GSPC. Queremos descobrir quanto a carteira rendeu em um determinado dia. Para isso primeiro devemos atribuir um peso às ações. Digamos que o peso de nossa carteira está distribuído de forma igualitária e que calcularemos o rendimento do último dia disponível na nossa carteira. A forma mais simples de realizar este cálculo é multiplicando a matriz de pesos pela matriz de preços da ação neste dia. Primeiro devemos transpor a matriz dos preços:

```
r_t = log_returns.tail(1).transpose()
```

Date	2018-12-31 00:00:00
Ticker	
AAPL	0.009619
MSFT	0.011686
^GSPC	0.008457

Em seguida criaremos a matriz dos pesos das ações dentro da carteira. Aqui utilizamos 1/3, pois cada uma das três ações possuem o mesmo peso dentro da carteira.

```
weights_vector = pd.DataFrame(1 / 3, index=r_t.index, columns=r_t.columns)
display(weights_vector)
```

Date	2018-12-31 00:00:00
Ticker	
AAPL	0.333333
MSFT	0.333333
^GSPC	0.333333

Realizamos a multiplicação e temos o rendimento da carteira como um todo:

```
portfolio_log_return = weights_vector.transpose().dot(r_t)
display(portfolio_log_return)
```

Date	2018-12-31 00:00:00
Date	
2018-12-31	0.00992

A fim de calcular para todo o período de tempo, devemos repetir o processo anterior, mas considerando todos os dias do período. O valor que procuramos estará na diagonal da matriz resultante, pois veja que é quando as datas das matrizes utilizadas na multiplicação se interseccionam. Para resgatar este valor, utilizamos a função diag, do numpy.

```
weights_matrix = pd.DataFrame(1 / 3, index=log_returns.index, columns=log_returns.columns)
weights_matrix.head()
```

Ticker	AAPL	MSFT	^GSPC
Date			
2000-01-03	0.333333	0.333333	0.333333
2000-01-04	0.333333	0.333333	0.333333
2000-01-05	0.333333	0.333333	0.333333
2000-01-06	0.333333	0.333333	0.333333
2000-01-07	0.333333	0.333333	0.333333

```
temp_var = weights_matrix.dot(log_returns.transpose())
```

```
temp_var.tail()
```

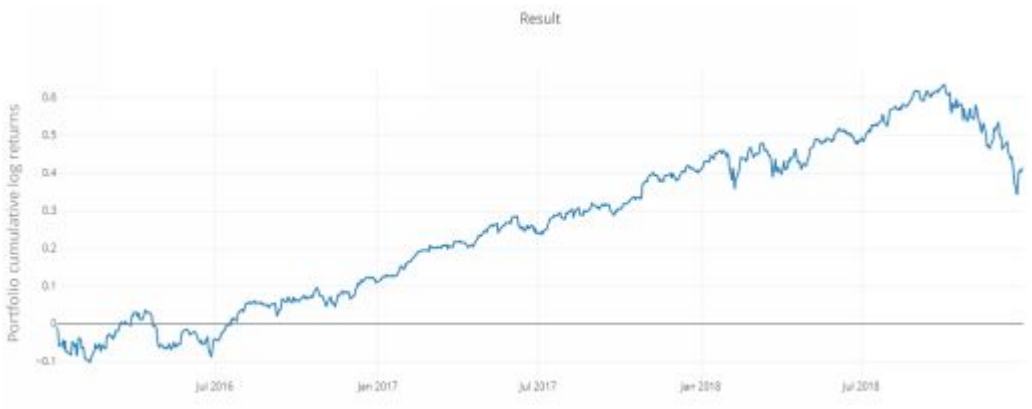
Date	2000-01-03 00:00:00	2000-01-04 00:00:00	2000-01-05 00:00:00	2000-01-06 00:00:00	2000-01-07 00:00:00	2000-01-10 00:00:00	2000-01-11 00:00:00	2000-01-12 00:00:00	2000-01-13 00:00:00	2000-01-14 00:00:00	...
Date											
2018-12-24	NaN	-0.053847	0.008979	-0.04121	0.028665	0.000216	-0.030533	-0.033119	0.044963	0.029452	...
2018-12-26	NaN	-0.053847	0.008979	-0.04121	0.028665	0.000216	-0.030533	-0.033119	0.044963	0.029452	...
2018-12-27	NaN	-0.053847	0.008979	-0.04121	0.028665	0.000216	-0.030533	-0.033119	0.044963	0.029452	...
2018-12-28	NaN	-0.053847	0.008979	-0.04121	0.028665	0.000216	-0.030533	-0.033119	0.044963	0.029452	...
2018-12-31	NaN	-0.053847	0.008979	-0.04121	0.028665	0.000216	-0.030533	-0.033119	0.044963	0.029452	...

5 rows × 4779 columns

```
portfolio_log_returns = pd.Series(np.diag(temp_var), index=log_returns.index)
portfolio_log_returns = portfolio_log_returns[1:]
portfolio_log_returns.tail()
```

A plotagem dos gráficos são similares ao que já fizemos anteriormente

```
  layout = go.Layout(
title='Result',
yaxis=dict(
      title='Portfolio cumulative log returns',
      titlefont=dict(
          size=18,
          color='#7f7f7f'
      )
)
  )
  axis = go.Scatter(
          x=portfolio_log_returns.index,
          y=portfolio_log_returns.cumsum(),
          opacity = 1)
  fig = dict(data=[axis], layout=layout)
  py.iplot(fig)
```



E para o gráfico de retorno relativo total:

```
  total_relative_returns = (np.exp(portfolio_log_returns.cumsum()) - 1)
  layout = go.Layout(
title='Result',
yaxis=dict(
      title='Portfolio total relative returns (%s)',
      titlefont=dict(
          size=18,
          color='#7f7f7f'
      )
)
  )
  axis = go.Scatter(
          x = total_relative_returns.index,
          y = total_relative_returns * 100,
          opacity = 1)
  fig = dict(data=[axis], layout=layout)
  py.iplot(fig)
```

Como podemos observar desde os anos 2000, nossa carteira chegou a ter um rendimento de mais de 500%. Observe que nesses últimos dois gráficos plotados, não temos as ações plotadas de forma separada, mas uma única linha indicando o retorno total da carteira, ou seja, não importaria se nossa carteira tivesse 1 ou 100 ativos. Aqui não estamos considerando reinvestimento de dividendos e outras ganhos, apenas a valorização das ações.

III

Médias móveis...

Uma vez que já obtivemos os dados e estamos mais familiarizados com eles, podemos partir para nossa primeira análise ou primeiro sistema de indicação de *trading,* isto é, nosso sistema de indicação de compra e venda de ações. Para isso veremos nosso primeiro indicador técnico, conhecido como média móvel.

Média móvel em estatística é um valor calculado a partir de uma série de valores de diferentes amostras de uma população e é utilizado para suavizar os dados desta amostragem, sendo muito utilizado como um indicador de tendência na análise de ativos, ou seja, indicando se o ativo tende a continuar uma tendência de alta ou uma tendência de baixa. Alexander Elder, em seu livro "*Como se transformar em um operador e investidor de sucesso"*, nos fornece a história das médias móveis, como elas são calculadas e a psicologia de mercado por trás delas, por isso recomendo a sua leitura.

Outra característica das médias móveis, que está mais para um efeito colateral, é que elas podem gerar uma análise em “atraso” para algumas situações, pois é baseada em preços passados. No entanto, muitos indicadores utilizados por analistas e investidores, nos mercados de ações, são baseados em médias móveis, tais como o cruzamento de médias móveis, HiLo Activator, Bandas de Bollinger, MACD, etc. Ainda podemos classificar as médias móveis quanto a sua maneira de cálculo: média móvel simples e média móvel exponencial.

Média móvel simples

Em um exemplo descomplicado, vamos imaginar que gostaríamos de calcular a média móvel simples de um período de 5 dias de uma ação que teve preço de fechamento nos últimos 5 dias de 20, 21, 22, 21, 26. Então a MM simples seria dada por:

$$\text{MM simples} = (20 + 21 + 22 + 21 + 26) / 5$$

Neste exemplo nossa MM teria um valor de 22.

Agora imaginemos que no sexto dia, o valor da ação seja 30. Desta forma, o cálculo da MM de 5 períodos seria dado por:

$$\text{MM simples} = (21 + 22 + 21 + 26 + 30) / 5$$

E nossa MM simples passaria a ter o valor de 24.

Observe que a janela de cálculo é sempre deslocada para os valores dos últimos 5 dias. Fazendo este cálculo sucessivamente para os dias que viriam pela frente, teríamos um novo conjunto de valores formados pelas várias MM simples

$$\text{Conjunto de MMS} = (22, 24, \ldots)$$

Realizar a plotagem dos dados utilizando python é relativamente simples, vamos precisar unir a classe Candlestick com a classe Scatter:

```
msft = all_data.loc['MSFT'].dropna()
msft = msft['2018-06-01':]
window = 21
```

```
# 21 days moving averages of the closing prices
MA = msft.Close.rolling(window=window).mean()
trace_avg = go.Scatter(
            x=MA.index,
            y=MA,
            name = "MSFT MA(21)",
            line = dict(color = '#BEBECF'),
            opacity = 1)
trace_candles = go.Candlestick(x=msft.index,
                open=msft.Open,
                high=msft.High,
                low=msft.Low,
                close=msft.Close,
                name = "Price")
data = [trace_avg, trace_candles]
fig = dict(data=data)
py.iplot(fig)
```

Observe como a média móvel, em cinza, cruza os *candles*. Mais a frente veremos como isto pode ser útil para analisar pontos de compra e de venda.

Média móvel exponencial

Assim como a média móvel simples, a média móvel exponencial é calculada com base nos preços anteriores definidos pela janela de períodos que está sendo utilizado. No entanto, a média móvel exponencial tem como característica diminuir a defasagem, isto é, a velocidade com que ela muda em relação ao preço, sendo mais rápida, pois ela aplica um peso maior aos preços mais recentes.

O cálculo exponencial da média móvel é um pouco mais complexo. Primeiro inicia-se calculando o coeficiente de multiplicação para o período

N que queremos trabalhar, que chamaremos de K. A seguir calcula-se uma média móvel simples para os primeiros N dias, que servirá de ponto inicial para os cálculos. A partir do período N+1 multiplica-se o preço de fechamento por K, multiplicando a MM do dia anterior por (1-K) e adicionando 2 aos números. Isto resulta nas seguintes equações.

$$MME = Preço\ Hoje \times K + MME\ ontem \times (1-K)$$

onde que:

$$K = 2 / (N + 1)$$

Observe as diferenças na plotagem de uma MMS, em laranja, e uma MME, em azul. Ambas com 21 períodos:

O desenho deste gráfico é um pouco mais complexo do que o desenho da média móvel, uma vez que exige mais etapas de cálculo. Vamos ver como podemos implementar isto em *python*, a seguir.

Primeiro precisamos calcular o fator K e a média móvel simples:

```
window = 21
K = ( 2 / (window + 1))
msft = all_data.loc['MSFT'].dropna()
msft = msft['2018-06-01':]
MA = msft.Close.rolling(window=window).mean().dropna()
```

Em seguida criaremos nosso *dataframe* de apoio, que será útil para o cálculo da média móvel exponencial, chamada simplesmente de EMA:

```
ema_data = pd.DataFrame(index=MA.index)
```

```
ema_data['Price'] = msft.Close
ema_data['MA'] = MA
ema_data['EMA'] = np.NaN
```

Agora vem a implementação da equação, onde utilizamos um loop de apoio. É válido lembrar que precisamos inicializar o primeiro valor da EMA como sendo o valor da média simples anterior:

```
ema_data.EMA[0] = ema_data.MA[1]
```

E assim temos :

```
for i in range(1, len(ema_data)):
   ema_data.EMA[i] = (ema_data.Price[i] * K) + ((1 - K) * ema_data.EMA[i-1])
```

Para realizar a plotagem do gráfico, seguimos a mesma rotina do gráfico anterior. Aqui, para questão de comparação, exibiremos tanto a média móvel simples quanto a média móvel exponencial.

```
trace_ma = go.Scatter(
            x=ema_data.index,
            y=ema_data.MA,
            name = "MSFT MA(21)",
            line = dict(color = '#BEBECF'),
            opacity = 1)
trace_ema = go.Scatter(
            x=ema_data.index,
            y=ema_data.EMA,
            name = "MSFT EMA(21)",
            line = dict(color = '#17BECF'),
            opacity = 1)
trace_candles = go.Candlestick(x=msft.index,
                open=msft.Open,
                high=msft.High,
                low=msft.Low,
                close=msft.Close,
                name = "Price")
data = [trace_ma, trace_ema, trace_candles]
fig = dict(data=data)
py.iplot(fig)
```

Cruzamento de médias móveis

Usaremos uma técnica conhecida desde a década de 1960, chamada cruzamento de médias móveis, que consiste em comprar ou vender um ativo sempre que as médias se cruzarem. Você pode estar se perguntando "Como assim médias se cruzando?".

Vamos criar um indicador que é composto por dois conjuntos de médias móveis simples. Uma conhecida como média curta ou short, e outra conhecida como média longa ou long. Para a média curta, usaremos um valor de 9 dias como período, e para a média longa um período de 21 dias.

Em *python* estes valores são facilmente calculados com a ajuda do *pandas*, utilizando a função ***rolling.***

Nosso objetivo é checar quando os cruzamentos de média acontecem e então efetuarmos uma compra ou venda. Por exemplo, sempre que a média de 9 dias cruzar para baixo da média de 21 dias, vendemos. Se a média de 9 dias cruzar a média de 21 dias para cima, compramos. No nosso exercício, começaremos calculando as médias de 9 e 21 períodos para as ações da Microsoft.

```
# Get the MSFT timeseries. This now returns a Pandas Series object indexed by date and drop NA rows.
msft = close.loc[:, 'MSFT'].dropna()
# Calculate the 9 and 21 days moving averages of the closing prices
```

```
short_rolling_msft = msft.rolling(window=9).mean()
long_rolling_msft = msft.rolling(window=21).mean()
```

A função *rolling* foi utilizada em conjunto com a função *mean()*. Neste cenário estamos calculando as médias dos valores selecionadas na nossa "janela" de tempo. Poderíamos utilizar outras funções, como *sum()* para somá-los ou *std()* para calcular o desvio padrão da nossa "janela de tempo".

Para a plotagem do gráfico, começamos por criar os 3 eixos necessários, que são o eixo da média curta, eixo da média longa, eixo do preço.

```
trace_short = go.Scatter(
            x=short_rolling_msft.index,
            y=short_rolling_msft,
            name = "MSFT Short",
            line = dict(color = '#17BECF'),
            opacity = 1)
trace_long = go.Scatter(
            x=long_rolling_msft.index,
            y=long_rolling_msft,
            name = "MSFT Long",
            line = dict(color = '#7F7F7F'),
            opacity = 1)
trace_price = go.Scatter(
            x=msft.index,
            y=msft,
            name = "MSFT Price",
            line = dict(color = '#B22222'),
            opacity = 0.8)
data = [trace_short, trace_long, trace_price]
```

Então invocamos a função de plotagem:

```
fig = dict(data=data)
py.iplot(fig)
```

Visualizando o gráfico, observamos as duas médias móveis e o preço do ativo. No nosso exemplo, poderíamos ter comprado o ativo no dia 13/04/2018 ao preço de 93,08, que é exatamente quando a média curta cruzou a média longa para cima. Ainda de acordo com o indicador, poderíamos ter vendido no dia 26/06/2018, ao preço de 99,08, totalizando um ganho aproximado de 6.5%.

Assim chegamos ao final da primeira lição, onde vimos de como obter dados históricos das ações, o que são médias móveis e como elas podem nos ajudar a construir um indicador ou sistema de *trading*.

IV

MACD…

Como vimos no capítulo anterior, as médias móveis filtram as oscilações dos preços, ajudando a identificar tendências. Também vimos o cruzamento de médias móveis, que é uma técnica um pouco mais elaborada para detecção de início de tendências. Neste capítulo veremos um indicador ainda mais avançado, o MACD.

O MACD, cujo nome completo é Convergência-Divergência da Média Móvel, usa não duas, mas três médias móveis, sendo que essas médias são todas exponenciais. Porém a plotagem é de apenas duas linhas, e o cruzamento destas linhas emitirão os sinais de negociação.

Observe no gráfico acima, onde temos duas linhas: uma azul e uma vermelha. A linha azul é o que chamamos de linha MACD, e é composta por duas médias móveis exponenciais subtraídas uma da outra. A linha vermelha é chamada linha de sinal, que nada mais é que a própria linha MACD ajustada por uma outra média exponencial.

Os sinais de negociação emitidos pelo indicador são semelhante ao de cruzamento de médias móveis. Quando a linha da média rápida MACD cruza a linha de sinal lenta, emite-se um sinal de compra, como pode ser visto no ponto 1 e ponto 3 do gráfico. Já quando a linha MACD cruza a linha de sinal para baixo, emite-se um sinal de venda, conforme pode ser observado no ponto 2 do gráfico.

A implementação em *python* é relativamente simples. Tudo que precisamos é resgatar o código que utilizamos no capítulo anterior, onde calculamos a média móvel exponencial, e isolarmos esse código em uma função para que possamos a reaproveitá-lo.

Esta função deverá receber um parâmetro indicando qual a janela de tempo que usaremos e um *dataframe* com os preços de fechamento.

```
def get_ema(window, prices):
    K = ( 2 / (window + 1))
  ma = prices.rolling(window=window).mean().dropna()
  data = pd.DataFrame(index=ma.index)
  data['Price'] = prices
  data['EMA'] = np.NaN
  data.EMA[0] = ma[1]
  for i in range(1, len(data)):
      data.EMA[i] = (data.Price[i] * K) + ((1 - K) * data.EMA[i-1])
  return data
```

Uma vez criada nossa função para cálculo da EMA, podemos iniciar coletando os dados do preço de fechamento e calculando cada uma das três médias. Aqui iremos usar as médias de 12, 26 e 9 períodos. Esta configuração é a configuração padrão consolidada no mercado e proposta, por exemplo, pelo Alexandre Elder.

```
    msft = all_data.loc['MSFT'].dropna()
    msft = msft['2018-06-01':]
    mm_12 = get_ema(12, msft.Close)
    mm_26 = get_ema(26, msft.Close)
    mm_macd = mm_12.EMA - mm_26.EMA
    mm_signal = get_ema(9, mm_macd.dropna()).EMA
```

Por fim, podemos tratar da plotagem dos dados.

```
from plotly import tools

msft = msft[mm_signal.index[0]:]
trace_macd = go.Scatter(
                x=mm_macd.index,
                y=mm_macd,
                name = "MACD",
                line = dict(color = '#17BECF'),
                opacity = 1)
trace_signal = go.Scatter(
                x=mm_signal.index,
                y=mm_signal,
                name = "Signal",
                line = dict(color = '#B22222'),
                opacity = 1)
trace_candles = go.Candlestick(x=msft.index,
                open=msft.Open,
                high=msft.High,
                low=msft.Low,
                close=msft.Close)
fig = tools.make_subplots(rows=2, cols=1)
fig.append_trace(trace_candles, 1, 1)
fig.append_trace(trace_macd, 2, 1)
fig.append_trace(trace_signal, 2, 1)
fig['layout']['yaxis2'].update(range=[-3, 5])
py.iplot(fig, filename='macd-line')
```

Observe que o gráfico abaixo, temos pontos de entrada, simbolizados pelo cruzamento da linha MACD, em azul, e da linha de sinal, em vermelho.

Até aqui vimos como calcular e desenhar o MACD, tendo como resultado final duas linhas que se cruzam para emitir sinais de compra ou

sinais de venda. Porém, há uma forma alternativa de plotar esse indicador e obter estes sinais, conhecida como histograma MACD.

O histograma MACD é o gráfico na forma de histograma criado pela diferença entre a linha MACD e a linha de sinal. A plotagem do MACD dessa forma promove uma visão mais aguçada do a produzida pelo MACD original, sobre o equilíbrio de poder entre compradores e vendedores.

Quando a linha MACD estiver acima da linha de sinal, o histograma será desenhado acima da linha zero, caso contrário, o histograma será desenhado abaixo da linha zero.

A metodologia de avaliação dos sinais é simples: se o indicador estiver acima da linha zero e com inclinação negativa, ou seja, diminuindo seu valor, isso pode indicar que a tendência de alta está perdendo força. No sentido oposto, quando o histograma está abaixo da linha zero e o indicador começar a subir, pode indicar que a tendência de queda pode estar chegando ao fim.

Para simplificar a plotagem, em *python*, iremos construir o histograma no formato de linha. Para isto usaremos a classe *Scatter*, com o parâmetro *fill* setado para *"tozeroy"*.

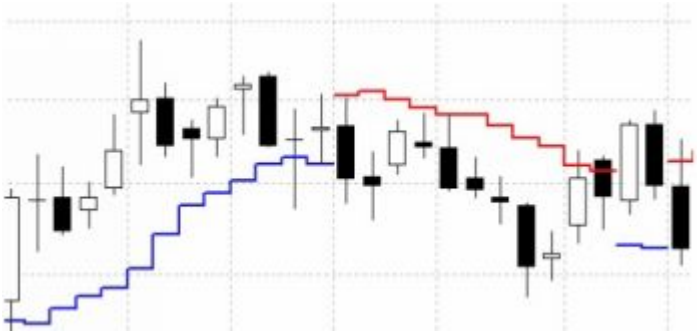
```
hist_macd = mm_macd – mm_signal
trace_hist_macd = go.Scatter(
                  x=hist_macd.index,
                  y=hist_macd,
                  fill='tozeroy')
trace_candles = go.Candlestick(x=msft.index,
                  open=msft.Open,
                  high=msft.High,
                  low=msft.Low,
                  close=msft.Close)
fig = tools.make_subplots(rows=2, cols=1)
fig.append_trace(trace_candles, 1, 1)
fig.append_trace(trace_hist_macd, 2, 1)
fig['layout']['yaxis2'].update(range=[-2, 2])
py.iplot(fig, filename='hist-macd-line')
```

Ainda existem outros sinais que podemos interpretar a partir do histograma MACD, como as divergências de alta e as divergências de baixa, mas por tornar a análise um pouco mais avançada, por hora, não entraremos em detalhes.

V

HiLo Activator…

HiLo Activator é um indicador que também utiliza médias móveis. Seu nome vem do inglês *High Low* (Alto Baixo) e seu objetivo é tentar identificar se um ativo está em tendência de alta ou de baixa, sendo muito utilizado como estratégia para identificar pontos de entrada ou saída de operações, e é conhecido como “rastreador de tendências”.



Em grande parte dos *softwares* de negociação, este indicador é desenhado na forma de “escada”, pois formam linhas no formato de escada abaixo ou acima dos *candles*. Se a tendência é de alta, será formada uma linha contínua abaixo do *candle.* Se a tendência é de baixa, será formada uma linha contínua acima do *candle.*

Onde a linha azul indica que a tendência é de alta, e a linha vermelha indica que a tendência é de baixa.

Ele emite sinais de compra ou sinais de venda sempre que há uma mudança no seu desenho. Tomando a figura anterior como exemplo, ao alternar entre o desenho azul, na banda inferior para o vermelho, na banda superior, ele emitiu um sinal de venda.

Seguindo esses sinais emitidos pelo indicador, conseguimos seguir as tendências quando elas acontecem. E devido a esta qualidade, este indicador é utilizado em muitas estratégias de *trend following.*

Cálculo

O HiLo é calculado utilizando duas médias móveis, a primeira média é calculada com base no preço de máximo do *candle*, a segunda média é calculada com base no preço mínimo do *candle*.

A regra para o cálculo é simples, se o preço fechou acima da média móvel de máxima (azul), traçamos a linha azul, significando tendência compradora. Se o preço fechou abaixo da média móvel de mínima, traçamos a linha vermelha, significando tendência vendedora. Caso o preço tenha fechado entre as duas médias, devemos aguardar, indica que estamos sem tendência definida ou ainda que a tendência anterior se mantenha, mas algo pode estar mudando.

Segue o código *python* para plotar o gráfico anterior:

```
aapl = all_data.loc['AAPL']['2018-01-01':'2018-12-31'].dropna()
aapl_high_avg = aapl.High.rolling(window=8).mean()
aapl_low_avg = aapl.Low.rolling(window=8).mean()
trace_high = go.Scatter(
            x=aapl_high_avg.index,
            y=aapl_high_avg,
            name = "AAPL High Avg",
            line = dict(color = '#17BECF'),
            opacity = 1)
trace_low = go.Scatter(
            x=aapl_low_avg.index,
            y=aapl_low_avg,
            name = "AAPL Low Avg",
            line = dict(color = '#B22222'),
            opacity = 1)
```

```
trace = go.Candlestick(x=aapl.index,
            open=aapl.Open,
            high=aapl.High,
            low=aapl.Low,
            close=aapl.Close)
data = [trace_high, trace_low, trace]
py.iplot(data)
```

Apesar de o HiLo utilizar duas médias, como plotado anteriormente, elas não devem ser vistas ao mesmo tempo, ou seja, o indicador possui uma "inteligência" para descobrir quando deverá exibir a média das máximas ou quando deverá exibir a médias das mínimas.

Uma vez que tenhamos as médias calculadas, podemos organizar o gráfico para que represente o HiLo somente em um sentido, como é correto. Isso quer dizer que o HiLo muda o desenho conforme um fechamento rompe o limite superior ou inferior das médias. Dessa forma, quando a linha estiver sendo traçada em azul, por baixo dos *candles,* estamos em um momento de alta dos preços e a linha vermelha não deverá ser plotada. Quando a linha estiver sendo traçada por cima dos *candles,* em vermelho, significa que estamos em um momento de queda dos preços e a linha azul não deverá ser plotada.

No gráfico acima podemos observar que nos momentos em que o preço subiu, a linha azul foi desenhada enquanto que a linha vermelha não. Também podemos observar que em momentos que houve quedas nos preços, a linha vermelha foi desenhada, enquanto a azul não. Este é o

segredo por trás do HiLo: desenhar as linhas de orientação de acordo com o cenário do mercado.

Para o gráfico do HiLo acima plotado acima, utilizamos médias móveis de 8 períodos. A estratégia a ser aplicada utilizando este indicador é comprar quando a linha passar do vermelho para o azul e vender quando a linha passar do azul para o vermelho. O gráfico foi plotado da seguinte forma:

```
aapl_high = pd.DataFrame(index=aapl.index)
aapl_low = pd.DataFrame(index=aapl.index)
aapl_high['high'] = np.where(aapl.Close > aapl_high_avg, aapl_low_avg, np.NaN)
aapl_low['low'] = np.where(aapl.Close < aapl_low_avg, aapl_high_avg, np.NaN)
    trace_high = go.Scatter(
                x=aapl_high.index,
                y=aapl_high,
                line = dict(color = '#17BECF'),
                opacity = 1)
    trace_low = go.Scatter(
                x=aapl_low.index,
                y=aapl_low,
                line = dict(color = '#B22222'),
                opacity = 1)
    trace = go.Candlestick(x=aapl.index,
                open=aapl.Open,
                high=aapl.High,
                low=aapl.Low,
                close=aapl.Close)
    data = [trace_high, trace_low, trace]
    py.iplot(data)
```

Esse indicador pode ser muito útil para indicar pontos de entrada em mercado com tendências, porém pode apresentar sinais ruins caso o mercado esteja lateral. Também é importante considerar o período escolhido para o valor da média móvel, função do ativo e do período de tempo gráfico.

VI

Bandas de Bollinger...

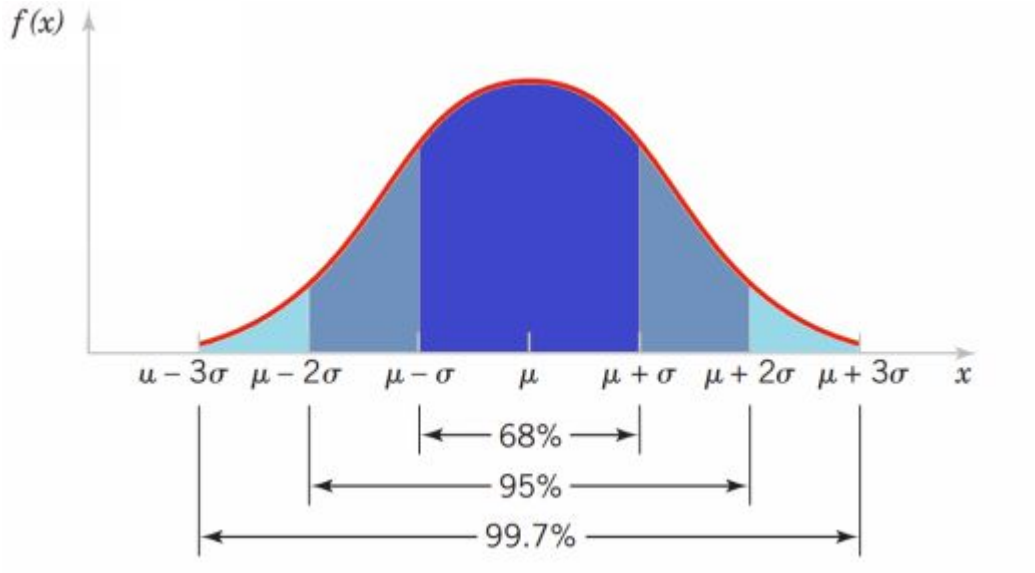
Criado por John Bollinger, o indicador homônimo ajuda a identificar a variação dos preços com base em níveis de desvios padrões, em outras palavras, significa que é um indicador capaz de medir a volatilidade do preço. Por exemplo, supondo que o preço atue em uma zona de equilíbrio, ora variando para cima e ora variando para baixo, a utilização deste indicador pode ser útil para ajudar na visualização de pontos de mínimos e pontos de máximos no preço, ajudando a identificar quando os preços estão muito valorizados ou muitos desvalorizados. Ao identificar que um preço está subvalorizado, tende-se a efetuar uma compra esperando que o preço volte a subir, ou efetuar uma venda, quando supervalorizado, esperando que o preço volte a cair.

Este indicador caracteriza-se pelo desenho de duas linhas abrangendo um “canal” de preço, criando uma banda superior e uma banda inferior.

O cálculo desse indicador é realizado através da aplicação de uma média móvel de 20 períodos ao preço de fechamento somado a dois desvios padrões de 20 períodos.

Banda Superior = Média Móvel Simples (20 dias) + (2 x Desvio Padrão de 20 dias)

Banda Inferior = Média Móvel Simples (20 dias) – (2 x Desvio Padrão de 20 dias)



Um ponto interessante sobre a escolha do valor de dois desvios padrões, é que ele vem de encontro com o de distribuição normal da estatística. Em uma distribuição normal, 95% dos valores cairão há uma distância de até 2 desvios padrões. Porém, como veremos mais a frente, isto não é totalmente verdade para as Bandas de Bollinger e não deve ser considerado como fator estatístico.

É importante interpretar e saber como aproveitar os sinais que este indicador tem a nos oferecer. A seguir veremos alguns dos sinais que este indicador pode apontar.

Suporte e resistências

Suporte é uma região onde os preços encontram dificuldade para continuar caindo. Geralmente, é nesse ponto que os compradores entram na operação, e aqui é representado pelas setas em vermelho. Já as setas azuis, indicam pontos de resistências, que é onde os vendedores assumem o controle do mercado. Por exemplo, quem comprou nos pontos indicados pelas setas vermelhas tentarão realizar o lucro, vendendo seus ativos nos pontos em azul.

No gráfico abaixo, podemos observar como as bandas proveram de forma bem visual, os níveis de suporte e resistência do preço.

Quebra do canal

Quando os limites do canal formado se estreitam, ou seja, a variação entre o preço máximo e preço mínimo começa a estabelecer um canal bem definido, pode indicar que a demanda e a oferta estão em equilíbrio. A quebra desse padrão, pode sinalizar que haverá uma forte tendência de alta ou de baixa. Por exemplo, na figura a seguir, podemos verificar (1) que houve uma quebra do canal, e a seguir uma forte queda nos preços.

Quando o preço ultrapassa os limites do canal, seja pela banda superior ou pela banda inferior, costuma-se dizer que o equilíbrio entre a demanda e oferta foi rompido, assim, inicia-se uma tendência altista ou baixadas. Porém devo deixar claro que o rompimento do canal por si só não representa sinal de compra ou de venda de determinado ativo. É necessário a avaliar outros fatores para realizar tal consideração, como a confirmação combinada com outros indicadores.

Logo aprenderemos a calcular e a plotar as Bandas de Bollinger utilizando *python,* mas antes verificaremos algumas das regras que John Bollingeer estabeleceu:

1. Bandas de Bollinger fornecem uma definição relativa de alto e baixo. Por definição, o preço é alto na banda superior e baixo na banda inferior.

2. Essa definição relativa pode ser usada para comparar o movimento do preço e com o movimento do indicador para chegar a decisões de compra e venda mais precisas.

3. Indicadores apropriados podem ser derivados do momento, volume, sentimento, interesse aberto, dados entre mercados, etc.

4. Se mais de um indicador for usado, os indicadores não devem estar diretamente relacionados entre si. Por exemplo, um indicador de momentum pode complementar um indicador de volume com sucesso, mas dois indicadores de momentum não são melhores que um.

5. As Bandas de Bollinguer pode ser usado no reconhecimento de padrões para definir / esclarecer padrões de preços, como padrões "M" no topo e "W" no fundo

6. Cruzamento ou encontro dos preços com as bandas são apenas isso. Não sinalizam sinais de venda ou sinais de compra por si só.

7. Nos mercados de tendências, seja de alta ou de baixa, o preço irá cruzar tanto a banda superior, quanto a banda inferior.

8. Fechamentos fora das bandas são inicialmente sinais de continuação, não sinais de reversão.

9. Os parâmetros padrão de 20 períodos para os cálculos de média móvel e desvio padrão, e dois desvios padrão para a largura das bandas são apenas isso, padrões. Os parâmetros reais necessários para qualquer mercado / ativo podem ser diferentes e precisam ser adaptados.

10. O cruzamento dos preços com a linha central das Bandas de Bollinger não deve ser considerado como um ótimo confirmador de uma nova tendência. Pelo contrário, deve ser considerado como uma confirmação média.

11. Se a média for alongada, o número de desvios padrão precisa ser aumentado; desvio de 2 para 20 períodos, desvio de 2,1 para 50 períodos. Da mesma forma, se a média for encurtada, o número de desvios padrão deve ser reduzido; 1,9 para 10 períodos.

12. As bandas de Bollinger tradicionais são baseadas em uma média móvel simples. Isso ocorre porque uma média simples é usada no cálculo do desvio padrão e desejamos ser logicamente consistentes.

13. Bandas de Bollinger exponencial eliminam mudanças repentinas na largura das bandas causadas por grandes mudanças de preço que saem da parte de trás da janela de cálculo. As médias exponenciais devem ser usadas para AMBAS a faixa do meio e no cálculo do desvio padrão.

14. Não faça suposições estatísticas baseadas no uso do cálculo do desvio padrão na construção das bandas. A distribuição dos preços de títulos não é normal e o tamanho típico da amostra na maioria das implementações do Bandas de Bollinger é muito pequeno para significância estatística. (Na prática, normalmente encontramos 90%, e não 95%, dos dados dentro do Bollinger Bands com os parâmetros padrão)

15. Bandas de Bollinger não fornecem uma sinalização contínua; em vez disso, elas ajudam a identificar configurações onde as probabilidades podem estar a seu favor

Aplicação

A implementação em python é bastante simples, primeiro precisamos preparar dois *dataframes*, um com a média móvel e um com os desvios padrões, ambos com 20 períodos.

```
Window = 20
aapl = all_data.loc['AAPL']['2018-01-01':'2018-12-31'].dropna()
aapl_avg = aapl.Close.rolling(window=window).mean().dropna()
aapl_std = aapl.Close.rolling(window=window).std().dropna()
```

Com os dados básicos calculados, podemos construir o *dataframe* final para as Bandas de Bollinger. Este *dataframe* consistirá de valores para banda superior, a banda inferior e a banda do meio, que é a própria média móvel.

```
aapl_bollinger = pd.DataFrame(index=aapl_avg.index)
aapl_bollinger['mband'] = aapl_avg
aapl_bollinger['uband'] = aapl_avg + aapl_std.apply(lambda x: (x * 2))
aapl_bollinger['lband'] = aapl_avg - aapl_std.apply(lambda x: (x * 2))
```

Observe que as colunas para a banda superior é a soma do *dataframe* da média móvel somado a duas vezes o *dataframe* de desvios padrões. Enquanto que o *dataframe* da banda inferior é constituído pela subtração do *dataframe* da média móvel por duas vezes o *dataframe* de desvios padrões.

Por fim, podemos plotar o gráfico. É válido notar que neste exemplo combinamos dois tipos diferentes de plotagem, utilizamos a classe

Candlestick em conjunto com a classe *Scatter*, para produzir o resultado final. Abaixo está o código *python*:

```
aapl_prices = aapl[aapl_bollinger.index[0]:]
prices = go.Candlestick(x=aapl_prices.index,
            open=aapl_prices.Open,
            high=aapl_prices.High,
            low=aapl_prices.Low,
            name = "Prices",
            close=aapl_prices.Close)
uband = go.Scatter(
            x=aapl_bollinger.index,
            y=aapl_bollinger.uband,
            name = "Upper Band",
            line = dict(color = '#17BECF'),
            opacity = 1)
mband = go.Scatter(
            x=aapl_bollinger.index,
            y=aapl_bollinger.mband,
            name = "Moving average",
            line = dict(color = '#B22222'),
            opacity = 1)
lband = go.Scatter(
            x=aapl_bollinger.index,
            y=aapl_bollinger.lband,
            name = "Lower Band",
            line = dict(color = '#17BECF'),
            opacity = 1)
data = [prices, uband, lband, mband]
py.iplot(data)
```

Resultando no gráfico abaixo:


Prices
Upper Band
Lower Band
Moving average
Nov 2018

VII

Índice de Força Relativa (IFR)…

O índice de força relativa (*Relative Strength Index – RSI*), assim como médias móveis e as Bandas de Bollinger, é um dos indicadores mais populares entre os investidores. Este indicador foi idealizado por Welles Wilder, Jr., e mede a força de um ativo em negociação, através da observação das mudanças nos preços de fechamento. Diferente dos indicadores que vimos, os quais todos eram do tipo rastreadores de tendências, o IFR é um indicador classificado como oscilador.

Indicadores baseados em médias móveis costumam emitir seus sinais de compra ou de venda com algum atraso. Já os indicadores do tipo oscilador, emitem seus sinais de operações antes que as reversões ocorram.

O IFR em particular, é um indicador antecedente ou coincidente, mas nunca consequente e pode ser calculado através da equação é dada pela fórmula:

IFR = 100 – 100 / (1 + FR), onde
FR = Média de Ganho / Média de perdas

O valor comumente utilizado no cálculo das médias iniciais do fator FR é o período 14 dias. Assim, os primeiros cálculos de ganho médio e perda média são médias simples de 14 períodos dado por:

1º ganho médio = soma dos ganhos nos últimos 14 períodos ÷ 14
1ª perda média = soma das perdas nos últimos 14 períodos ÷ 14

A partir de então, e para todos os valores subsequentes, os cálculos devem ser baseados na média anterior:

Ganho Médio = [(ganho médio anterior) x 13 + Ganho atual] ÷ 14
Perda Média = [(perda média anterior) x 13 + Perda atual] ÷ 14

Observando atentamente, podemos notar que a fórmula utilizada para cálculo do IFR os valores resultantes sempre estarão dentro da faixa que vai de 0 a 100, fazendo com que sua interpretação seja muito simples.

A interpretação básica para o IFR é que ao atingir o pico máximo e reverter para baixo, indicará um topo nos preços, e ao cair até o valor mínimo e virar para cima, indica um fundo nos preços. Aliado a isso ainda temos as regiões de sobre compra e sobre venda.

Regiões de sobre venda e sobre compra são regiões onde os preços do ativo estão considerados muito caros ou muito baratos. Se estiver caro, o senso comum diz que há chance de que logo sofra uma reversão para queda, e se estiver muito barato, tenderá a subir.

A região de sobre compra costuma ser definida pelos analistas como um valor variando entre 70 e 100, dependendo do ativo que se está analisando. Já a região de sobre venda, é definida entre 0 e 30. No gráfico a seguir podemos ver essa duas regiões:

Observe no gráfico que o ponto em (1) está numa região de sobre compra. O IFR emite esse sinal e após alguns pregões o preço da ação começa a cair vertiginosamente. Algo semelhante acontece no ponto (2), onde o IFR emite um sinal de sobre venda, indicando que os preços estão baratos e alguns pregões após o sinal, os preços começam a subir lentamente.

Retomando a equação para o cálculo do IFR, vamos analisar como implementá-la em *python*.

Primeiramente, como de costume, devemos separar um *dataframe* com os dados que iremos trabalhar:

```
window = 14
aapl = all_data.loc['AAPL'].dropna()
aapl = aapl['2018-06-01':'2019-02-22']
ifr = pd.DataFrame(index=aapl.index)
```

Agora precisamos calcular os ganhos ou perdas que tivemos entre um dia e outro. Isto pode ser calculado pela função diff.

Uma vez que tenhamos variação, identificar e separar as que obtivemos ganhos e as que obtivemos perdas:

```
# get days with positive profit (gain)
ifr['gain'] = ifr_changes[ifr_changes > 0]
# get days with negative profit (loss)
ifr['loss'] = ifr_changes[ifr_changes < 0] * (-1)
```

O próximo passo é calcular a média inicial de ganho e perda, pois é a partir dela que todo o restante é calculado:

```
# gain and loss mean
ifr['gainAvg'] = np.NaN
ifr['lossAvg'] = np.NaN
# get the mean for "window" period
ifr.gainAvg[window] = ifr.iloc[0:window].gain.mean()
ifr.lossAvg[window] = ifr.iloc[0:window].loss.mean()
```

E agora podemos aplicar a fórmula do IFR:

```
for i in range(1, len(ifr)):
    ifr.gainAvg[i] = (ifr.gainAvg[i-1] * (window -1) + ifr.gain[i]) / window
    ifr.lossAvg[i] = (ifr.lossAvg[i-1] * (window -1) + ifr.loss[i]) / window
ifr['value'] = 100 - (100 / (1 + (ifr.gainAvg / ifr.lossAvg )))
```

Com todos os cálculos efetuados, já podemos plotar o gráfico. Aqui os parâmetros de plotagem utilizados são um pouco mais complexo, pois precisamos desenhar duas linhas horizontais no gráfico do IFR:

```
trace_ifr = go.Scatter(
                x=ifr.index,
                y=ifr.value,
                xaxis='x2',
                yaxis='y2')
trace_candles = go.Candlestick(x=aapl.index,
                open=aapl.Open,
                high=aapl.High,
                low=aapl.Low,
                close=aapl.Close)
data = [trace_ifr, trace_candles]
layout = go.Layout(
  xaxis=dict(
     domain=[0, 1],
     rangeslider={"visible": False},
  ),
  yaxis=dict(
     domain=[0.75, 1],
  ),
  xaxis2=dict(
     domain=[0, 1]
  ),
  yaxis2=dict(
     domain=[0, 0.45],
     anchor='x2',
     range=[0, 100]
  ),
     shapes=[
        {
          'xref': 'x2',
          'yref': 'y2',
          'type': 'line',
          'x0': aapl.index[0],
          'y0': '20',
          'x1': aapl.index[-1],
          'y1': '20',
          'line': {
             'color': 'rgb(50, 171, 96)',
             'width': 2,
             'dash': 'dashdot',
          },
        },
        {
          'xref': 'x2',
          'yref': 'y2',
          'type': 'line',
          'x0': aapl.index[0],
          'y0': '70',
          'x1': aapl.index[-1],
          'y1': '70',
```

```
            'line': {
                'color': 'rgb(50, 171, 96)',
                'width': 2,
                'dash': 'dashdot',
            },
              }
        ]
)
fig = go.Figure(data=data, layout=layout)
py.iplot(fig, filename='ifr-plot')
```

O código acima resultará na plotagem do gráfico de *candles* e no IFR, conforme podemos ver abaixo.

Esse gráfico representa as ações da Apple. Note como entre agosto e setembro o IFR, que rompeu linha verde tracejada na banda superior (Mais de 70), sinalizou que o papel estava caro.

Logo que o IFR perdeu o patamar de sobrecompra e apontou para baixo, abaixo dos 70, o papel em pouco tempo começou a cair de preço.

VIII

On Balance Volume (OBV)...

Até este momento vimos indicadores que são calculados apenas com base nos preços de um ativo. Porém, existem indicadores que levam em consideração, além do preço, o volume negociado.

O indicador conhecido como OBV é um destes indicadores. É muito útil em situações onde queremos medir a força da tendência, identificar ou confirmar possíveis reversões ou ainda sinalizar o surgimento de novas tendências. Em outras palavras, podemos medir a pressão compradora e a pressão vendedora.

Dessa forma, um aumento no OBV indica um maior volume positivo, sinalizando uma maior quantidade de compradores. Em virtude da alta concentração de compradores, o ativo tende subir de preço. Já uma queda no OBV, indica um maior volume negativo, demonstrando uma maior concentração de vendedores.

Para chegarmos nos valores finais deste indicador, devemos somar o volume atual com o volume anterior se o preço de fechamento é maior que o preço de abertura. Caso o preço de fechamento seja menor que o preço de abertura, então devemos subtrair o volume anterior do volume atual. Caso os preços de abertura e fechamento permaneçam iguais, o volume se mantêm inalterado.

Se preço fechamento > preço abertura:
OBV atual = OBV anterior + Volume do Último Período
Se preço fechamento < preço abertura:
OBV atual = OBV anterior – Volume do Último Período
Se não houve alterações: preço fechamento = preço abertura

A análise feita com o auxílio do OBV deve levar em consideração dois tipos de sinais: Sinais de divergência e Confirmação de tendências

Sinais de divergência aparecem quando a tendência do OBV não está de acordo com a tendência dos preços, por exemplo: OBV caindo e os preços subindo. Esse tipo de sinal é considerado o mais forte do OBV, indicando sinais de compra ou venda, dependendo se a divergência é de alta ou a divergência é de baixa.

O OBV ainda pode ser utilizado para confirmar novas tendências. Por exemplo, podemos traçar suportes e resistências no gráfico do OBV. Quando ocorrer o rompimento dessa resistência ou suporte no OBV, poderemos ter uma reversão de preço.

Neste gráfico podemos observar, no ponto 1, que tínhamos uma resistência, e logo que o OBV rompeu para cima, os preços subiram. No ponto 2, tinha umas LTA (linha de tendência de alta) no OBV, logo que ela foi rompida, emitindo um sinal de venda, o preço do ativo começou a cair.

Conforme pode ser visto na sequência, a aplicação em *python* é relativamente simples:

Primeiramente separamos os dados que trabalharemos.

```
aapl = aapl['2018-06-01':'2019-02-22']
aapl = all_data.loc['AAPL'].dropna()
obv = pd.DataFrame(index=aapl.index)
```

Em seguida, calculamos as diferenças entre os preços de abertura e de fechamento.

```
obv_changes = aapl.Close - aapl.Open
```

Quando a diferença entre os preços de abertura e fechamento for negativa, multiplicaremos por -1 para facilitar a soma posterior.

```
obv['open'] = aapl.Open
obv['close'] = aapl.Close
obv['volume'] = np.where(obv_changes > 0, aapl.Volume , aapl.Volume * (-1))
```

E então utilizamos o método *cumsum()*. Este método realiza a soma cumulativa dos valores de uma coluna, ele soma o valor N com N-1, N-1 com N-2 e assim por diante.

```
obv['volume_sum'] = obv.volume.cumsum()
```

Por fim, plotamos o gráfico:

```
trace_obv = go.Scatter(
```

```
                x=obv.index,
                y=obv.volume_sum,
                xaxis='x2',
                yaxis='y2')
trace_candles = go.Candlestick(x=aapl.index,
                open=aapl.Open,
                high=aapl.High,
                low=aapl.Low,
                close=aapl.Close)
data = [trace_obv, trace_candles]
layout = go.Layout(
  xaxis=dict(
     domain=[0, 1],
     rangeslider={"visible": False},
  ),
  yaxis=dict(
     domain=[0.75, 1],
  ),
  xaxis2=dict(
     domain=[0, 1]
  ),
  yaxis2=dict(
     domain=[0, 0.45],
     anchor='x2'
  )
)
fig = go.Figure(data=data, layout=layout)
py.iplot(fig, filename='obv-plot')
```

É importante salientar que o importante no Saldo de Volume é sempre o valor relativo e a direção na qual ele está se movendo, o valor absoluto não é relevante. Por isso o OBV pode apresentar valores absolutos diferentes em plataformas diferentes ou ainda dependendo de qual a data de referência do início do cálculo. Porém, o desenho da linha do OBV será sempre igual.

IX

Average Directional Index (ADX)…

Criado por J. Welles Wilder, Jr. durante a década de 1970, o sistema direcional atua como um indicador de tendência, sendo capaz de apontar a existência de tendência e indicar sua força. Sendo muito útil quando utilizado em conjunto com algum outro indicador, a fim de confirmar um sinal de entrada ou saída de uma operação.

O ADX é um indicador que varia de 0 a 100, sendo usual considerar que o mercado está sem tendência quando o indicador está com valor abaixo de

20. Estando com valor entre 20 a 40, há tendência. E acima de 40, a tendência é considerada forte. Ainda vale ressaltar que o indicador não aponta se a tendência é de alta ou de baixa. Ele apenas identifica a sua força.

Esse indicador pode ser considerado um caso especial entre os indicadores, pois é derivado de dois outros indicadores, que são conhecidos como Indicador Direcional Positivo (+DI) e Indicador Direcional Negativo (-DI). Estes dois indicadores são da categoria de osciladores, e variam de 0 a 100, e na maioria das vezes, são calculados para 14 períodos. Logo, o gráfico completo do ADX apresenta 3 linhas: o próprio ADX, o +DI e o -DI.

Antes de aprendermos a interpretar e a calcular o ADX, vamos analisar um exemplo prático do seu uso.

Por exemplo, se estivermos utilizando um indicador baseado somente em médias móveis, como o HiLo, e o mercado tornar-se lateral, isto é, sem tendência, o HiLo emitiria vários sinais falsos de compra ou venda. Neste cenário possivelmente entraríamos em uma operação e logo em seguida seríamos tirados dela, devido a um *stop loss* por reversão na tendência do preço.

Veja na figura abaixo um exemplo de mercado lateral onde o HiLo sinaliza uma venda as 14:00, e na sequência, sinaliza reversão de tendência por volta das 14:45 praticamente no mesmo patamar de preço no qual iniciamos a operação. Ele continua repetindo esse ciclo conforme o tempo passa. Note também que em conjunto com esse gráfico, temos também um outro subgráfico com 3 linhas plotadas. Estas linhas são o ADX, +DI e -DI

respectivamente. A linha na cor preta está sinalizando ausência de tendência, pois continua abaixo do valor 20 ao longo de todo o tempo.

Assim, ao usar o ADX em conjunto com o HiLo, poderíamos evitar seguir os sinais emitidos pelo HiLo enquanto não houvesse sinal de tendência (ADX acima de 20) nos preços, evitando entrar na operação e sair sem ganhos, ou pior, sair com perdas.

Aplicação

Dentre os indicadores que vimos até aqui, este é, sem dúvida, o mais complexo para ser calculado. Assim, resumiremos em alguns passos o que precisaremos implementar em *python*:

1. Identificar a faixa verdadeira, conhecida como True Range – TR
2. Calcular o movimento direcional – DM (+DM e -DM)
3. Realizar uma suavização da TR, +DM e -DM
4. Calcular os indicadores direcionais +DI e -DI
5. Calcular o indicador ADX

Começaremos obtendo o conjunto de dados para os cálculos. Neste exemplo trabalharemos com os dados da VALE3 ao longo do ano de 2018.

```
tickers = ['VALE3.SA']
start_date = datetime.datetime(2018, 1, 1)
end_date = datetime.datetime(2018, 12, 31)
data = get(tickers, start_date, end_date).loc['VALE3.SA'].dropna()
```

No primeiro passo calcularemos a TR. O valor da TR será o maior valor entre os seguintes valores:

A) A diferença entre a máxima de hoje e a mínima de hoje

B) A diferença entre a máxima de hoje e o fechamento de ontem

C) A diferença entre a mínima de hoje e o fechamento de ontem

Em python, a implementação do passo 1 pode ser feita da seguinte forma:

```
# previous values. Shift by 1
data_ant = data.shift(1)
# intermediate dataset, whose purpose is to find out the TR value
tr = pd.DataFrame([
      data.High - data.Low,
      np.absolute(data.High - data_ant.Close),
      np.absolute(data.Low - data_ant.Close)])
     # to facilitate our calculation, we have to transpose the matrix,
tr = tr.transpose()
# get the max value of each row
tr= tr.max(axis=1)
# assign TR to our dataset
data['tr'] = tr
```

O que resultará em:

Date	High	Low	Open	Close	Volume	Adj Close	tr
2018-01-03	41.880001	41.299999	41.830002	41.470001	12744200.0	39.855412	0.580002
2018-01-04	42.369999	41.520000	41.810001	41.639999	18433000.0	40.018787	0.899998
2018-01-05	42.290001	41.310001	41.570000	42.290001	15251300.0	40.643486	0.980000
2018-01-08	43.230000	42.400002	42.400002	43.230000	14542800.0	41.546883	0.939999
2018-01-09	43.750000	42.930000	43.580002	43.070000	15986200.0	41.393112	0.820000

O passo dois, que é o cálculo do movimento direcional (DM), é realizado comparando a faixa máxima-mínima de hoje com a faixa máxima-mínima do período anterior, sendo que sempre é um número positivo. A implementação em *python* pode ser demonstrada da seguinte forma:

```
# get +DM
dm_plus = np.where((data.High - data_ant.High) > (data_ant.Low - data.Low),
                   (data.High - data_ant.High).apply(
                         lambda x: np.max([x, 0])),
                   0)
# get -DM
dm_minus = np.where((data_ant.Low - data.Low) > (data.High - data_ant.High),
                   (data_ant.Low - data.Low).apply(
                         lambda x: np.max([x, 0])),
                   0)
# assign DM's to our dataset
data['dmPlus'] = dm_plus
data['dmMinus'] = dm_minus
# remove first line, becouse we won't use it
data = data[1:]
```

Resultando em:

Date	High	Low	Open	Close	Volume	Adj Close	tr	dmPlus	dmMinus
2018-01-03	41.880001	41.299999	41.830002	41.470001	12744200.0	39.855412	0.580002	0.139999	0.000000
2018-01-04	42.369999	41.520000	41.810001	41.639999	18433000.0	40.018787	0.899998	0.489998	0.000000
2018-01-05	42.290001	41.310001	41.570000	42.290001	15251300.0	40.643486	0.980000	0.000000	0.209999
2018-01-08	43.230000	42.400002	42.400002	43.230000	14542800.0	41.546883	0.939999	0.939999	0.000000
2018-01-09	43.750000	42.930000	43.580002	43.070000	15986200.0	41.393112	0.820000	0.520000	0.000000

Em posse da TR, +DM e -DM, podemos suavizá-los. Esta etapa é similar a aplicar uma média móvel aos dados, a fim de suavizar as flutuações. Em nosso calculo usaremos a média móvel de 14 períodos:

```
window = 14
data['trAvg'] = np.NaN
data['dmPlusAvg'] = np.NaN
data['dmMinusAvg'] = np.NaN
# sum of 14 periodos for calcaulation of the tr, dmPlus and dmMinus
data.trAvg[window-1] = np.sum(data.iloc[0:window].tr)
data.dmPlusAvg[window-1] = np.sum(data.iloc[0:window].dmPlus)
data.dmMinusAvg[window-1] = np.sum(data.iloc[0:window].dmMinus)
# get data from 14th row until end of the dataset
```

```
for i in range(window, len(data)):
   data.trAvg[i] = data.trAvg[i-1] - (data.trAvg[i-1] / window) + data.tr[i]
   data.dmPlusAvg[i] = data.dmPlusAvg[i-1] - (data.dmPlusAvg[i-1] / window) + data.dmPlus[i]
        data.dmMinusAvg[i] = data.dmMinusAvg[i-1] - (data.dmMinusAvg[i-1] / window) + data.dmMinus[i]
data = data.dropna()
data.head()
```

	tr	dmPlus	dmMinus	trAvg	dmPlusAvg	dmMinusAvg
	0.939999	0.00	0.379997	11.820004	3.329998	2.150002
	1.840000	0.00	1.410000	12.815718	3.092141	3.406430
	1.200001	0.02	0.000000	13.100310	2.891274	3.163113
	0.000000	0.00	0.000000	12.164574	2.684755	2.937177
	1.170002	0.00	1.070000	12.465677	2.492986	3.797378

Com os valores suavizados, o quinto passo é implementado dividindo-se os valores suavizados do DM pelos valores suavizados da TR:

```
# get DI's values
data['diPlus'] = (data.dmPlusAvg / data.trAvg) * 100
data['diMinus'] = (data.dmMinusAvg / data.trAvg) * 100
```

Assim, obtendo os valores para +DI e do -DI:

	diPlus	diMinus
	28.172564	18.189517
	24.127724	26.580094
	22.070273	24.145332
	22.070273	24.145332
	19.998805	30.462669

O sexto e último passo é o cálculo do próprio ADX. Aqui é onde entram os dois indicadores anteriores, os DI's. As etapas de cálculo são simples:

A) Calcular o valor DX, que é dado pelo valor absoluto da subtração entre +DI por -DI, divido pela soma do +DI com -DI.

B) Tendo em mãos o valor de DX, aplica-se uma suavização por um valor de 14 períodos, obtendo assim o ADX.

A implementação em *python* pode ser feita da seguinte forma:

```
# get DX value
data['DX'] = (np.absolute(data.diPlus - data.diMinus) / (data.diPlus + data.diMinus)) * 100
data['ADX'] = np.NaN
    # average of 14 periods
    data.ADX[window-1] = data.DX[0:window].mean()
    # get data from 14th row until end of the dataset
    for i in range(window, len(data)):
  data.ADX[i] = (data.ADX[i-1] * (window - 1) + data.DX[i]) / window
    data = data.dropna()
```

E assim obtivemos as três componentes do nosso indicador, +DI, -DI e ADX. Resultando em:

diPlus	diMinus	DX	ADX
16.590358	22.996570	16.182643	10.812390
16.864095	20.977427	10.869890	10.816497
26.958481	17.837981	20.359867	11.498166
32.858876	16.340445	33.574511	13.075048
32.819590	15.638390	35.455875	14.673678

Finalizada a implementação do nosso roteiro, podemos plotar o gráfico. Aqui plotaremos o gráfico do ADX em conjunto com os *candles*:

```
    trace_candles = go.Candlestick(x=data.index,
                        name="VALE3.SA",
                        open=data.Open,
                        high=data.High,
                        low=data.Low,
                        close=data.Close)
    trace_di_minus = go.Scatter(
                        x=data.index,
                        y=data.diMinus,
                        name = "-DI",
                        line = dict(color = '#B22222'),
                        opacity = 1)
    trace_di_plus = go.Scatter(
                        x=data.index,
                        y=data.diPlus,
```

```
                    name = "+DI",
                    line = dict(color = '#17BECF'),
                    opacity = 1)
trace_adx = go.Scatter(
                    x=data.index,
                    y=data.ADX,
                    name = "ADX",
                    line = dict(color = '#9932CC'),
                    opacity = 1)
fig = dict(data=[trace_di_minus, trace_di_plus, trace_adx])
py.iplot(fig, filename='adx-line')
```

Analisando a plotagem de nosso gráfico, podemos observar que de março a final de abril, a linha roxa (ADX) ficou abaixo do patamar de 20. Durante este período o valor da ação permaneceu em uma faixa lateral, nem subindo, nem caindo. O ADX só voltou a mostrar tendência perto de maio, e a partir de então observamos uma alteração de preços na ação, entrando em uma tendência de alta.

X

Estratégia da média 9...

Nos capítulos anteriores passamos por alguns dos indicadores mais populares na análise técnica. Agora chegou o momento de aplicarmos estratégias de compra e venda que utilizam estes indicadores. Começaremos por uma estratégia que usa média móvel, conhecida como estratégia da média móvel 9. Este *setup* foi descrito por um *trader* americano chamado Larry Willams. Sua metodologia é bem simples: basicamente utiliza-se uma média móvel exponencial de 9 períodos para emitir sinais de compra ou venda.

Supondo que a MM estava com inclinação negativa, espera-se até que a média 9 cruze o preço e incline para cima, marca-se o *candle* que produziu essa virada e realiza-se uma compra quando o próximo *candle* superar a máxima do *candle* que produziu a virada da média, colocando-se um *stop loss* no valor mínimo do *candle* que produziu a virada.

Já quando a média móvel 9 cruza o preço apontando para baixo, deve-se realizar uma venda invertendo a lógica anterior. Assim quando a média apontar para baixo, marca-se o *candle* que produziu essa virada, quando o próximo *candle* atingir um valor menor que a mínima do *candle* que produziu a virada para baixo, realiza-se uma compra, colocando um *stop loss* na máxima do *candle* que produziu a virada.

O termo *stop loss* refere-se ao limite de perda que estamos dispostos a aceitar, caso a operação não vá ao nosso favor. Em uma operação de compra, o *stop loss* é o valor em que efetuaremos uma venda caso o preço comece a cair além do nosso valor de compra. Por exemplo, imagine que compramos uma ação X ao valor de $10 e estabelecemos um *stop loss* à $9. Isso quer dizer que caso nossa ação, ao invés de valorizar, sofra uma desvalorização, abandonaremos a operação ao chegar no valor de $9 para evitarmos mais perdas, caso ela continue perdendo valor.

Assim sendo, nosso algoritmo deverá implementar as seguintes regras:

Condição de compra:

- MME 9 caindo, esperar que vire para cima (com *candle* fechado)
- Marcar a máxima do *candle* que fez a virada ocorrer
- No *candle* seguinte, ao romper sua máxima, efetua-se a compra. Caso a sua máxima não seja rompida, a entrada continua válida enquanto a MME 9 continuar subindo, mas deve-se aguardar pelo *candle* que rompa a máxima do *candle* que fez a virada ocorrer.

Condições de venda:

- MME 9 subindo, esperar que vire para baixo (com *candle* fechado)
- Marcar a mínima do *candle* que fez a virada ocorrer
- No *candle* seguinte, ao romper sua mínima, efetua-se a venda. Caso a sua mínima não seja rompida, a entrada continua válida enquanto a MME 9 continuar descendo, mas deve-se aguardar pelo *candle* que rompa a máxima do *candle* que fez a virada ocorrer

A fim de facilitar nossa exemplificação, nosso algoritmo em python terá algumas simplificações para um melhor entendimento.

O primeiro passo que precisamos executar é obter os dados do ativo que iremos trabalhar. Neste exemplo, trabalharemos com as ações da VALE3.

```
tickers = ['VALE3.SA']
start_date = datetime.datetime(2018, 1, 1)
end_date = datetime.datetime(2018, 6, 28)
all_data = get(tickers, start_date, end_date)
```

Então calculamos a média exponencial de 9 períodos para este conjunto de dados:

```
vale3 = all_data.loc['VALE3.SA'].dropna()
```

```
vale3['EMA'] = get_ema(9, vale3.Close).EMA
```

O primeiro ponto crítico surge agora, onde precisaremos calcular a inclinação da curva num determinado ponto, para saber se a inclinação é ascendente, indicando que o preço está subindo, ou descendente, indicando que o preço está caindo. Isto implica em calcular os pontos de inflexão em um dado intervalo. Porém, é uma atividade matemática que exige um conhecimento mais avançado.

Para facilitar nossa vida, vamos aplicar o cálculo de inclinação da reta, ou coeficiente angular. Para isso usaremos sempre valor atual e valor anterior a este. Se o resultado for positivo, nosso preço pode estar subindo, se for negativo, nosso preço pode estar caindo. A equação é dada pela fórmula abaixo:

$$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$$

Logo, em *python*, a implementação para descobrir o valor de M é feita subtraindo-se somente o eixo Y, já que o eixo X terá sempre o "Delta X" igual 1, pois a variação é diária.

```
# get the "slope of EMA9 9"
dif = vale3.EMA – vale3.shift(1).EMA
# ema 9 crossed the candle and ema 9 has an ascendent slope  ?
vale3['mark_max'] = np.where((vale3.EMA < vale3.High) & (vale3.EMA > vale3.Low) & (dif > 0),
vale3.High, 0)
# ema 9 crossed the candle  and ema 9 has an descendent slope  ?
vale3['mark_min'] = np.where((vale3.EMA > vale3.Low) & (vale3.EMA < vale3.High) & (dif < 0),
vale3.Low, 0)
```

Com os dados pré-calculados, podemos obter o ponto de entrada e ponto de *stop loss*. Para a operação de compra, o ponto de entrada é dado pelo candle seguinte, caso esse ultrapasse o valor do candle que teve a

média cruzada, conforme a regra que estabelecemos anteriormente. O contrário vale para a operação de venda:

```
# get start point
vale3['buy_start'] = np.where((vale3.Low < vale3.shift(1).mark_max) & (vale3.High > vale3.shift(1).mark_max), vale3.shift(1).mark_max, np.NaN)
vale3['sell_start'] = np.where((vale3.Low < vale3.shift(1).mark_min) & (vale3.High > vale3.shift(1).mark_min), vale3.shift(1).mark_min, np.NaN)
# set stop loss
vale3['buy_stop'] = np.where((vale3.Low < vale3.shift(1).mark_max) & (vale3.High > vale3.shift(1).mark_max), vale3.shift(1).Low, np.NaN)
vale3['sell_stop'] = np.where((vale3.Low < vale3.shift(1).mark_min) & (vale3.High > vale3.shift(1).mark_min), vale3.shift(1).High, np.NaN)
```

Logo, para operações de compras teríamos os seguintes pontos de *start* e *stop loss:*

```
vale3[["buy_start", "buy_stop"]][(vale3.buy_start > 0)]
```

Date	buy_start	buy_stop
2018-01-18	43.220001	42.360001
2018-02-07	42.470001	39.930000
2018-04-02	42.750000	41.740002
2018-04-09	43.099998	42.250000
2018-06-04	53.080002	50.650002
2018-06-05	53.450001	51.759998
2018-06-29	49.439999	47.500000

Como resultado, foram sugeridos 7 pontos de entrada para compra. Vamos analisar no gráfico o que teria acontecido se tivéssemos entrado nessas operações. Ainda, como estratégia de saída da operação para realização do lucro (vender o que compramos), consideraremos que realizaremos o lucro sempre que o preço cair abaixo da MME 9.

Já no primeiro ponto destacado podemos ver que as coisas não foram muito bem para nós. Logo após 3 períodos do ponto de entrada, teríamos atingido nosso *stop loss*. Assim, essa operação geraria um pequeno prejuízo, de 2% do nosso valor total da compra.

No ponto 2, supondo que tivéssemos comprado no dia 07/02 e vendido somente por volta do dia 28/02, que foi quando o preço voltou a cair abaixo da MME 9, teríamos obtido lucro.

Nos pontos 3 e 4 podemos verificar duas sinalizações de compra. Para efeitos de simulações, consideraremos que a compra foi feita no ponto 3, dia 04/06. Então segundo nossa regra, teríamos mantido a compra até dia 21/05. Isso representaria um lucro bem expressivo, de aproximadamente 25%.

O ponto 5 sinalizou uma compra e pelas nossas regras de negociação, não apresentou lucro, pois o vendemos logo que o preço cruzou para abaixo da MME 9, alguns dias após a compra. Nessa situação provavelmente teríamos saído da operação no mesmo preço ao qual entramos.

Para operações de vendas também podemos aplicar os mesmos princípios teríamos os seguintes pontos de *start* e *stop loss:*

```
vale3[["buy_start", "buy_stop"]][(vale3.buy_start > 0)]
```

Date	sell_start	sell_stop
2018-02-05	40.750000	41.650002
2018-02-09	41.070000	42.130001
2018-03-01	44.650002	47.090000
2018-03-28	40.720001	42.389999
2018-05-22	53.000000	55.099998
2018-05-28	51.049999	52.790001
2018-05-30	49.990002	52.060001
2018-06-11	50.660000	53.990002

Como resultado, também foram sugeridos vários pontos de entrada para venda. Não iremos analisá-los no gráfico, mas caso o leitor o faça, poderá observar um comportamento semelhante ao que ocorreu aos pontos de

compra. Alguns apresentarão lucros, outros acabarão por atingir o *stop loss*, gerando alguma perda, e outros ainda não apresentarão perdas nem ganhos.

Note que nosso algoritmo não está otimizado, não estamos trabalhando com gerenciamento de risco ou gerenciamento de patrimônio, nem com *trailing-stop.* Estes são itens que poderiam aumentar nosso percentual de lucro e tornar nossas operações mais seguras. Ainda sobre o *trailing-stop*, trata de um *stop loss* móvel, onde moveríamos o valor de saída da operação conforme o crescimento do percentual de lucro na operação em que estamos posicionados.

Ainda sobre a abordagem de cálculo que utilizamos, ela pode gerar muitos ruídos devido a forma como estamos calculando a inclinação da média móvel. Uma forma de amenizar este problema é utilizar o cruzamento de médias. Nesse caso poderíamos utilizar o cruzamento da média móvel exponencial de 9 períodos com a média móvel exponencial de 1 período. Quando a MME 1 cruzar a MM9 para cima, temos um possível sinal de compra. Quando a MME1 cruzar a MME9 para baixo, temos um possível sinal de venda. As demais condições se mantêm, como o ponto de entrada e o *stop loss.*

```
vale3 = all_data.loc['VALE3.SA'].dropna()
vale3['EMA9'] = get_ema(9, vale3.Close).EMA
vale3['EMA1'] = get_ema(1, vale3.Close).EMA
buy_cross = (vale3.shift(1).EMA1 < vale3.shift(1).EMA9) & (vale3.EMA1 > vale3.shift(1).EMA9)
sell_cross = (vale3.shift(1).EMA1 > vale3.shift(1).EMA9) & (vale3.EMA1 < vale3.shift(1).EMA9)
# ema 9 crossed the candle (open/close price) and ema 9 has an ascendent slope  ?
vale3['mark_max'] = np.where((vale3.EMA9 < vale3.High) & (vale3.EMA9 > vale3.Low) & (buy_cross == True), vale3.High, 0)
# ema 9 crossed the candle (open/close price) and ema 9 has an descendent slope  ?
vale3['mark_min'] = np.where((vale3.EMA9 > vale3.Low) & (vale3.EMA9 < vale3.High) & (sell_cross == True), vale3.Low, 0)
# get start point
vale3['buy_start'] = np.where((vale3.Low < vale3.shift(1).mark_max) & (vale3.High > vale3.shift(1).mark_max), vale3.shift(1).mark_max, np.NaN)
vale3['sell_start'] = np.where((vale3.Low < vale3.shift(1).mark_min) & (vale3.High > vale3.shift(1).mark_min), vale3.shift(1).mark_min, np.NaN)
# set stop loss
vale3['buy_stop'] = np.where((vale3.Low < vale3.shift(1).mark_max) & (vale3.High > vale3.shift(1).mark_max), vale3.shift(1).Low, np.NaN)
vale3['sell_stop'] = np.where((vale3.Low < vale3.shift(1).mark_min) & (vale3.High > vale3.shift(1).mark_min), vale3.shift(1).High, np.NaN)
```

Como pode ser observado nas imagens a seguir, o conjunto dos pontos de entrada para compra se mantiveram bem parecidos com o que havíamos calculado anteriormente. Porém, o conjunto de pontos de entrada para venda foi bem reduzido, ou seja, os pontos não ficaram tão suscetíveis a pequenas variações de inclinação da MME 9.

Date	buy_start	buy_stop
2018-01-18	43.220001	42.360001
2018-02-07	42.470001	39.930000
2018-04-02	42.750000	41.740002
2018-06-04	53.080002	50.650002
2018-06-29	49.439999	47.500000

Date	sell_start	sell_stop
2018-02-09	41.070000	42.130001
2018-03-01	44.650002	47.090000
2018-03-23	41.619999	42.250000
2018-05-22	53.000000	55.099998
2018-06-11	50.660000	53.990002

XI

Backtesting…

Ok, até aqui analisamos algumas formas de realizar operações com ações, compra e vender para o longo prazo, comprar e vender utilizando o cruzamento de médias móveis e comprar e vender utilizando o indicador HiLo, estratégia com média móvel de 9 períodos, etc. Mas qual destes métodos poderia produzir os melhores resultados?

Bem, esta é uma pergunta muito difícil de responder devido à quantidade de variáveis que estão envolvidas. Existe um ditado que diz que os ganhos passados não são garantias de ganhos futuros. Assim, um indicador ou estratégia que apresentou boa performance num período X,

pode não apresentar a mesma performance no período Y. Por exemplo, o HiLo. Quando um mercado não “anda”, ou seja, os preços não possuem tendências de alta nem de baixa, o indicador apresenta pontos de entrada ou pontos de saída falsos. O mesmo vale para o cruzamento de médias móveis. Até mesmo o *buy and hold* pode sofrer problemas: seus dividendos podem sofrer variação de um ano para outro, ou quando resolver vender seus ativos após alguns meses ou anos, eles podem ter desvalorizado.

Para tentar minimizar esses problemas, é muito comum executamos algo chamado *Backtesting,* ou seja, executar nosso modelo de negociação utilizando dados históricos do mercado. Dessa forma conseguimos estimar o comportamento do modelo. Para isso, os detalhes utilizados no modelo deverão ser os mais detalhados possíveis, por exemplo: valores históricos corretos, dados de volume financeiro, a granularidade correta da informação e assim por diante.

A partir dos dados levantados pelo *backtest* podemos ajustar nosso modelo para ter uma boa performance. Porém, é importante estar atento para evitar o viés. Viés ou *overfitting*, é o ajuste demasiado do modelo, fazendo com que o mesmo tenha ótima performance em dados históricos, mas quando se depara com dados novos, não consegue ter uma taxa boa de sucesso, ou seja, não generaliza suficientemente.

Ao avaliar o resultado de testes de um modelo, com base em dados históricos, é importante levar em conta não somente o retorno do investimento obtido, pois não há garantias que a performance vai ser repetida no futuro. Métricas como *drawdown* máximo, percentual de acertos e fator de lucro são importantes.

Drawdown

Em termos gerais, *drawdown* é medida da diminuição do capital num dado período de tempo. Olhe por exemplo o gráfico a seguir do rendimento do nosso portfólio fictício:

Em determinado período houve uma desvalorização de 90% do capital, e mais a frente, outra de 30%. Para o período total, o *drawdown* máximo será o maior entre os *drawdowns* do período analisado, aqui 90%.

Observe que mesmo obtendo lucro ao final de 2016 se comparado ao início de 2008, em determinados momentos houve quedas significativas. Isso abala o emocional do investidor, fazendo com que se desfaça do ativo em momentos que não deveria. Logo, se não temos "estômago" para lidar com grandes quedas no capital, ou seja, operar com mais volatilidade, o uso do *drawdown* ajuda e percebermos isso no nosso modelo.

Fator de lucro

Fator de lucro é o dado que indica a relação entre lucro bruto e prejuízo bruto. Ele pode ajudar a avaliar a eficiência do modelo de negociação escolhido. Por exemplo, imagine que tenhamos dois modelos de negociação, modelo A e modelo B respectivamente.

O modelo A teve um ganho bruto de 100.000 e uma perda bruta de 50.000. O modelo B, por sua vez, teve um ganho bruto de 80.000 e uma perda bruta de 20.000, assim temos:

Modelo A = 100.000 / 50.000 = 2

Modelo B = 80.000 / 20.000 = 4

Embora o modelo A tenha gerado o lucro bruto maior que o modelo B, sua perda também foi maior. Assim dizemos que modelo B foi mais eficiente que no modelo A. Muitos analistas sugerem que o fator de lucro seja de 3 pra cima, porém um fator de 2 já é razoável.

Operar Vendido

Ainda como preparatório para nosso sistema de *backtesting,* preciso falar brevemente sobre o que é operar vendido.

Até agora nossas discussões foram sobre comprar uma ação, vendê-la em algum momento no futuro e então contabilizar o lucro ou prejuízo. Quando abrimos uma posição em algum ativo, primeiro executando a compra, e fechamos a posição executando a venda, dá-se o nome "estar comprado". Porém podemos trabalhar de forma invertida, forma conhecida pelo nome "estar vendido".

Estar vendido em um ativo significa que abrimos uma posição vendendo o ativo, mesmo sem tê-lo, e fecharemos a posição comprando. Pode parecer complicado à primeira vista, por isto vamos para um exemplo:

Imaginemos que temos um amigo que é dono de uma mercearia e sabemos que lá ele vende o litro de um refrigerante ao custo de $2. E também temos uma tia que está precisando comprar um litro de refrigerante. Adicionado a isto, você também leu no noticiário que o mercado de refrigerantes está com excesso de refrigerante no varejo e que todos dias na última semana o preço tem caído. Sabendo disto, no início da manhã você vai até esse seu amigo e pede pra ele um litro de refrigerante, dizendo que vai devolver ao final do dia. Nesse momento o refrigerante ainda está custando $2. Seu amigo que não é bobo, te empresta, mas somente mediante ao pagamento de uma taxa. Ele vai te cobrar $0,10 pra te emprestar. Então você pega o refrigerante, leva até a sua tia e vende para ela por $2, ou seja, você vendeu para ela algo que não é seu, pois lembre-se que o refrigerante você tomou emprestado do seu amigo. Porém, você é esperto e sabe que a probabilidade de o refrigerante ter o preço reduzido até o final do dia é grande. Você espera até o meio da tarde e então encontra uma outra

mercearia, onde o preço do mesmo refrigerante já está sendo vendido a $1.50. Você compra o refrigerante a este preço e vai até seu amigo, dono da outra mercearia, para devolver o refrigerante que você tinha pego emprestado. Assim, você teve um lucro de $0.40, sem sequer ter precisado desembolsar do seu próprio dinheiro ($2 do refrigerante vendido pra tia, -$0.10 do aluguel do refrigerante, -$1.50 da compra do refrigerante para devolver ao seu amigo = 2 +(-0.10) +(-1,5) = 0.4)

Claro que essa venda do “refrigerante” tem risco, imagina se o preço do refrigerante em vez de cair, subir. Você chegaria ao final do dia e teria que comprar por um preço maior do que $2 para devolver o refrigerante ao seu amigo, e isto poderia fazer você ter um bom prejuízo.

Este exemplo é muito similar com “operar vendido” com ações, onde nesta situação buscamos vender algo que não temos, pedindo emprestado (e pagando aluguel por isso) na esperança de que mais a frente consigamos recomprar mais barato, a fim de ter lucro.

Nosso backtesting

Voltando ao nosso *backtest*, construiremos uma rotina que permita simular compras e vendas de ações. Para efeitos de simplificação, não computaremos o *drawdown* e o fator de lucro. Porém, este último estará "embutido" no cálculo final, pois forneceremos o ganho líquido do modelo.

Nosso sistema operará tanto posicionado comprado no ativo, quanto vendido, pois queremos ganhar com as quedas do ativo também. Para isso, a regra será essa.

- Se estou líquido, ou seja, não posicionado, entro no primeiro sinal que aparecer, seja de compra ou de venda.
- Se estou comprado e o próximo sinal é outro sinal de compra, não faço nada.
- Se estou compradoe o próximo sinal é de venda, fecho minha posição vendendo o ativo e abro uma nova posição de venda, pois espera-se que o ativo caia.
- Se estou vendido e próximo sinal é de venda, não faço nada.
- Se estou vendido e o próximo sinal é de compra, fecho minha posição comprando o ativo e abro uma nova posição de compra, pois espera-se que o ativo suba.
- Deverá trabalhar apenas com o lote padrão de 100 ações.
- Ao operar vendido, não computará o valor de aluguel das ações

Nosso sistema deverá ter duas funções principais, uma para fazer o *trade* e outra principal, que decidirá quando realizar o trade. Para que seja um sistema flexível, ele deverá receber os sinais de compra ou venda de fora, ou seja, ter os sinais injetados nele. Assim poderemos passar uma lista de sinais de compra e venda pra ele, e ele executará a compra ou venda com base nesses sinais. Vamos ao código:

Definir os tipos de trades, onde "*long*" é uma compra, e "*short*" é uma venda.

```
    from enum import Enum
    class TradeType(Enum):
  NONE = 0
  LONG = 1
  SHORT  = -1
```

A estrutura básica da classe de *backtest* será composta por uma função de trade(_trade), uma função principal (start) e funções auxiliares para calcular o número de trades com lucro e número de trades com prejuízo.

```
    class BackTest:
  def __init__(self, stock, signals, **kwargs):
  def _get_win_trades(self, x):
  def _get_loss_trades(self, x):
  def _trade(self, position, index):
  def start(self):
```

Segue o código completo:

```
from enum import Enum
class TradeType(Enum):
  NONE = 0
  LONG = 1
  SHORT  = -1
class BackTest:
  def __init__(self, stock, signals, **kwargs):
    self._position = 0
    self._position_opened = False
    self._amount = 0
    self._stock = stock
    self.trades = []
    self._signals = signals
    self._status = None
    self._std_qty_shares = 100
    self._reverse = True
    self._show_logs = ("log" in kwargs) and (kwargs["log"])
    self.win_trades = 0
    self.loss_trades = 0
  def _get_win_trades(self, x):
    return x >= 0
  def _get_loss_trades(self, x):
    return x < 0
  def _trade(self, position, index):
```

```
        """
        :position : LONG or SHORT
        :index : index of stock list
        """
        # if has a opened trading, then it will close the position
        if self._position_opened:
            # set the operating profit
             profit = ((self._std_qty_shares * self._stock.Close[index]) - self._amount)  * (self._position *
-1)
            # add proft to trade list
            self.trades.append(profit)
            # close position
            self._position_opened = False
            # log
            if self._show_logs:
                                        print(f'Opened[{self._amount}]  ->  Closed[{self._std_qty_shares  *
self._stock.Close[index]}] {position.name}')
        else:
            # If it hasn't an opened position, open it.
            self._position_opened = True
            self._amount = self._std_qty_shares * self._stock.Close[index]
    def start(self):
        """
            execute the backtest
        """
        for index, item in self._stock.iterrows():
            # for long signal (buy)
            if TradeType(self._signals.signal[index]) == TradeType.LONG:
                # Last signal was long signal? go to next iteration
                if TradeType(self._position) == TradeType.LONG: continue
                self._position = self._signals.signal[index]
                #if it has a opened short position, then sell
                if self._position_opened:
                    self._trade(TradeType.SHORT, index)
                    # because I'm want to revert my position
                    if self._reverse:
                        self._trade(TradeType.LONG, index)
                else:
                    # ok, I need to open a position
                    self._trade(TradeType.LONG, index)
            # for short signal (sell)
            if TradeType(self._signals.signal[index]) == TradeType.SHORT:
                if TradeType(self._position) == TradeType.SHORT: continue
                self._position = self._signals.signal[index]
                if self._position_opened:
                    self._trade(TradeType.LONG, index)
                    if self._reverse:
                        self._trade(TradeType.SHORT, index)
                else:
                    self._trade(TradeType.SHORT, index)
```

```
    self.win_trades = len(list(filter(self._get_win_trades, self.trades)))
    self.loss_trades = len(list(filter(self._get_loss_trades, self.trades)))
```

Essa nossa classe deverá ser inicializada com dois parâmetros, o primeiro é o *DataFrame* com o histórico de preços do ativo, e o segundo, é o *DataFrame* com os sinais de compra e venda. Tomando o HiLo como modelo de negociação, vamos ver como fica a geração dos sinais de compra e venda:

```
signal_start_date = '2018-01-01'
signal_end_date = '2018-12-31'
aapl = all_data.loc['AAPL'][signal_start_date:signal_end_date].dropna()
def build_hilo_signal_list(window=5):
  aapl_high_avg = aapl.High.rolling(window=window).mean()
  aapl_low_avg = aapl.Low.rolling(window=window).mean()
  aapl_hilo = pd.DataFrame(index=aapl.index)
   aapl_hilo['high'] = np.where(aapl.Close > aapl_high_avg, 1, 0)
   aapl_hilo['low'] = np.where(aapl.Close < aapl_low_avg, 1, 0)
  aapl_hilo['signal'] = (aapl_hilo.high - aapl_hilo.low)
  return aapl_hilo
```

Aplicando os sinais ao *backtest*, temos:

```
    signals = build_hilo_signal_list(window=8)
    bt = BackTest(aapl, signals, log=True)
    bt.start()
    print(f'Profit: {sum(bt.trades)}')
```

```
Opened[17527.999877929688] -> Closed[17422.000122070312] LONG
Opened[17422.000122070312] -> Closed[16433.999633789062] SHORT
Opened[16433.999633789062] -> Closed[17502.999877929688] LONG
Opened[17502.999877929688] -> Closed[17997.999572753906] SHORT
Opened[17997.999572753906] -> Closed[17530.00030517578] LONG
Opened[17530.00030517578] -> Closed[17161.000061035156] SHORT
Opened[17161.000061035156] -> Closed[17280.00030517578] LONG
Opened[17280.00030517578] -> Closed[16910.000610351562] SHORT
Opened[16910.000610351562] -> Closed[18630.999755859375] LONG
Opened[18630.999755859375] -> Closed[19024.000549316406] SHORT
Opened[19024.000549316406] -> Closed[19069.99969482422] LONG
Opened[19069.99969482422] -> Closed[18717.999267578125] SHORT
Opened[18717.999267578125] -> Closed[18991.000366210938] LONG
Opened[18991.000366210938] -> Closed[20150.0] SHORT
Opened[20150.0] -> Closed[22130.00030517578] LONG
Opened[22130.00030517578] -> Closed[22641.000366210938] SHORT
Opened[22641.000366210938] -> Closed[21788.00048828125] LONG
```

```
Opened[21788.00048828125] -> Closed[22494.99969482422] SHORT
Opened[22494.99969482422] -> Closed[22428.99932861328] LONG
Opened[22428.99932861328] -> Closed[22272.999572753906] SHORT
Opened[22272.999572753906] -> Closed[21508.999633789062] LONG
Opened[21508.999633789062] -> Closed[22222.000122070312] SHORT
Opened[22222.000122070312] -> Closed[20747.999572753906] LONG
Opened[20747.999572753906] -> Closed[18482.000732421875] SHORT
Opened[18482.000732421875] -> Closed[17472.000122070312] LONG
Profit: 989.9993896484375
```

Como pode ser observado, nosso primeiro *backtest* produziu um lucro de $989,99. O mecanismo é simples: quando é compra, o backtest faz a compra de 100 ações, quando é venda, faz a venda de 100 ações. Ao encerrar a posição armazena o lucro/prejuízo em uma lista de resultados.

Esta abordagem é simples, mas ainda tem um ajuste a ser feito. Como visto no exemplo anterior, usamos uma média de 8 períodos para calcular o HiLo. Mas como sabemos se esse é o melhor valor? Bem, não sabemos. Temos que ir experimentando valores.

Para isso, podemos criar um laço para automatizar o teste de valores para nós.

```
best_profit = None
print(f'BACKTEST HILO - Start Date: {signal_start_date} / End Date: {signal_end_date}')
print('-' * 100)
for window in range(2, 22):
  signals = build_hilo_signal_list(window=window)
  bb = BackTest(aapl, signals, log=False)
  bb.start()
  profit = sum(bb.trades)
   if (best_profit is None) or (best_profit[1] < profit):
     best_profit = (window, profit)
  print(f'Profit for window={window}: {profit}')
print('-' * 100)
print(f'Best Profit: window={best_profit[0]}: {best_profit[1]}')
```

```
BACKTEST HILO - Start Date: 2018-01-01 / End Date: 2018-12-31
----------------------------------------------------------------
Profit for window=2: 1243.0038452148438
Profit for window=3: 405.00030517578125
Profit for window=4: 1092.999267578125
Profit for window=5: 6072.996520996094
Profit for window=6: 5559.996032714844
Profit for window=7: 4229.997253417969
```

```
Profit for window=8: 989.9993896484375
Profit for window=9: 2574.0036010742188
Profit for window=10: 1591.0003662109375
Profit for window=11: 727.9998779296875
Profit for window=12: -2371.0006713867188
Profit for window=13: -2291.0018920898438
Profit for window=14: -1723.0010986328125
Profit for window=15: -3179.998779296875
Profit for window=16: -2897.9995727539062
Profit for window=17: -3229.0008544921875
Profit for window=18: -1571.002197265625
Profit for window=19: -1926.0009765625
Profit for window=20: 15.00091552734375
Profit for window=21: -561.0000610351562
------------------------------------------------------------
Best Profit: window=5: 6072.996520996094
```

Analisando médias e indo do valor 2 ao valor 21, o melhor valor para média seria o valor 5, pois é onde temos o maior lucro líquido. Como exercício para o leitor, é recomendável testar outros períodos históricos também, para garantir que nosso modelo não está enviesado.

Iremos ainda realizar o mesmo *backtest* para o cruzamento de médias móveis, assim podemos comparar o desempenho do HiLo com o Cruzamento de médias móveis:

```
signal_start_date = '2018-01-01'
signal_end_date = '2018-12-31'
aapl = all_data.loc['AAPL'][signal_start_date:signal_end_date].dropna()
def build_cross_avg_signal_list(short_window=5, long_window=9):
  aapl_short_avg = aapl.Close.rolling(window=short_window).mean()
  aapl_long_avg = aapl.Close.rolling(window=long_window).mean()
  aapl_cross_avg = pd.DataFrame(index=aapl.index)
   aapl_cross_avg['signal'] = np.where(aapl_short_avg > aapl_long_avg, 1, -1)
  return aapl_cross_avg
```

```
best_profit = None
print(f'BACKTEST CROSS AVG - Start Date: {signal_start_date} / End Date: {signal_end_date}')
print('-' * 100)
for short_window in range(2, 22):
  for long_window in range(2, 22):
                                signals     =     build_cross_avg_signal_list(short_window=short_window,
long_window=long_window)
      bb = BackTest(aapl, signals, log=False)
      bb.start()
```

```
        profit = sum(bb.trades)
        if (best_profit is None) or (best_profit[1] < profit):
            best_profit = ((short_window,long_window), profit)
        print(f'Profit for window={short_window}/{long_window}: {profit}')
print('-' * 100)
print(f'Best Profit: window={best_profit[0]}: {best_profit[1]}')
```

```
        BACKTEST CROSS AVG - Start Date: 2018-01-01 / End Date: 2018-12-31
        Profit for window=2/2: 0
Profit for window=2/3: -3558.013916015625
Profit for window=2/4: -1431.0104370117188
Profit for window=2/5: 1570.9915161132812
        …
Profit for window=21/18: 3016.9967651367188
Profit for window=21/19: 4614.996337890625
Profit for window=21/20: -215.00396728515625
Profit for window=21/21: 0
----------------------------------------------------------------
Best Profit: window=(5, 8): 9961.001586914062
```

Para o modelo de cruzamento de médias, precisamos que os períodos das médias sejam diferentes entre si, pois um é o período rápido, e outro o período lento, por isso foi utilizado dois laços, resultando na combinação de períodos (5,8), respectivamente para média rápida e média lenta.

Comparativo

Como visto, o *backtesting* é fundamental para a análise de uma estratégia de *trading*, e deve ser bem pensado a fim de evitar conclusões incorretas sobre o modelo de negociação. Ainda que nosso modelo tenha apresentado o melhor resultado do mundo, nada garante que conseguiremos repetir a mesma performance no futuro. Porém o *backtesting* pode apontar um norte, pelo menos para nos dizer se nosso modelo funcionaria bem ou não no que já conhecemos.

Uma vez que testamos dois modelos, HiLo e Cruzamento de médias, podemos fazer um comparativo em relação ao lucro líquido retornado.

Para o HiLo tivemos o melhor cenário utilizando o período de 5 dias para a média móvel, totalizando o resultado de $6072.99. Já para o cruzamento de médias, obtivemos um resultado de $9961.00, ou seja, quase 65% a mais de lucro que o HiLo. Mas bem, isto quer dizer que o cruzamento de médias é melhor que o HiLo? Não necessariamente.

Nosso *backtesting* é baseado num algoritmo simples de compra e venda. Nele não consideramos drawdown, não implementamos técnicas de *stops-loss* que, poderiam ter evitado perdas grandes, nem testamos o modelo em outros tempos gráficos. Também não descontamos o valor do aluguel de ações ao operar posições vendidas. Todos estes dados impactariam no sistema de negociação e o objetivo deste guia não foi abordar esse nível de detalhamento, mas passar uma ideia geral de como podemos implementar alguns indicadores e estratégias utilizando *python*.

XII

Próximos passos…

O objetivo deste livro foi demonstrar ao leitor como é possível implementar alguns indicadores e estratégias utilizadas em bolsa de valores de forma fácil e rápida, utilizando *python*. Isso significa que você viu um bocado de código, aprendeu um pouco sobre a construção dos indicadores mais conhecidos da análise técnica, e ficou ciente de estratégias utilizadas no mercado financeiro na análise de ativos na bolsa de valores. Porém, essa é apenas a ponta do iceberg.

Atualmente há uma variedade de programas computacionais que automatizam a análise de dados no mercado financeiro, robôs que automatizam as operações de compra e venda, e mais recentemente, empregam o uso de inteligência artificial para execução dessas operações.

No primeiro capítulo nós aprendemos a obter dados históricos do valor das ações. No mundo real, isto não é o suficiente. Dependendo da sua estratégia de negociação, provavelmente você precisará dos dados em tempo real para alimentar seus sistemas, e isto envolve consumo de API's ou serviços de terceiros.

Nos capítulos que trabalhamos com indicadores, criamos *scripts* simples, em notebooks de exemplo. No entanto, no mundo real, eles precisam ser componentizados e otimizados para minimizar o uso de processamento e memória, além de serem tolerantes a falhas. Outra preocupação é onde estes serviços serão executados. Deve ser um ambiente que tenha alta tolerância a falha, como uma nuvem.

Houve alguns pontos, que por não ser objetivo deste guia, não foram explorados. Porém, é importante citá-los aqui. O primeiro é o gerenciamento de risco, isto é, o quanto você está disposto a perder quando as coisas não forem bem. E isto vai acontecer.

Quando planejamos um sistema de operação de compra e venda, devemos nos preocupar com *stop loss* e *trailing-stop. O* primeiro vai fazer

com que suas perdas sejam controladas, já o segundo, vai evitar que você perca o que já ganhou. Em nosso sistema de *backtesting* não o implementamos, mas fica como tarefa para o leitor o fazê-lo e verificar se os ganhos são maximizados.

Outro ponto que deve ser levado em consideração, é o uso de mais de um indicador para confirmar a entrada ou saída da operação. Em nossos exemplos construímos o *backtesting* considerando sempre o sinal de apenas um indicador. Poderíamos, com pequeno esforço, gerar sinais combinados de dois ou mais indicadores ao aplicarmos executarmos nosso *backtesting*. Por exemplo, poderíamos aplicar a estratégia da média nove exponencial aliada ao ADX para termos a confirmação de que existe uma tendência antes de efetivamente entrar na operação.

Por fim, se você é um programador que está adentrando ao mundo das finanças, eu recomendo a leitura destes livros:

- Como Se Transformar Em Um Operador e Investidor de Sucesso – Alexander Elder
- Os Axiomas de Zurique – Max Gunther
- Aprenda a operar no mercado de ações – Alexander Elder

Por outro lado, se você é investidor e está adentrando ao mundo da programação e desenvolvimento de sistemas para automatizar suas análise ou operações, eu recomendo a leitura desse livro:

- Python Fluente: Programação Clara, Concisa e Eficaz – Luciano Ramalho

Glossário

Ativo – Bens, valores, créditos, títulos e outros instrumentos financeiros que formam o patrimônio de uma empresa ou de uma pessoa, e que pode ser convertido em dinheiro.

Andar de lado – Expressão utilizada para indicar que o mercado está sem tendência de alta ou tendência de baixa definida, isto é, em tendência lateral.

Corretora – Entidade que atua no sistema financeiro habilitada no comércio de títulos e valores mobiliários. É a intermediária dos investidores nas transações de ações.

Day Trade - Compra e venda de um ativo, efetuadas no mesmo dia.

Swing Trade – É uma estratégia de negociação especulativa nos mercados financeiros onde um ativo comerciável é mantido entre um e vários dias em um esforço para lucrar com mudanças de preço ou 'swings'

Operar a descoberto – Operação sem proteção, ou seja, sem o ativo em custódia ou sem aluguel do ativo.

Stop loss – É uma ordem de compra ou venda, que é enviada para a correta, que indica sua intenção de interromper a perda (loss) em uma posição aberta.

Take Profit – É uma de compra ou venda que permite a fixação de um nível de lucro quando o preço atinge determinado valor.

Trailing stop – é um algoritmo executado conjuntamente com a ordem de “stop loss”. Quando é executado o *trailing stop*, o nível da *stop loss* será alterado de acordo com o movimento do preço. Se for aberta uma posição de compra, e o preço do instrumento aumenta em um determinado valor, se o *trailing stop* estiver ativado, a ordem de *stop loss* acompanhará a subida de preço.