candlestick_chart_recognition

May 19, 2019

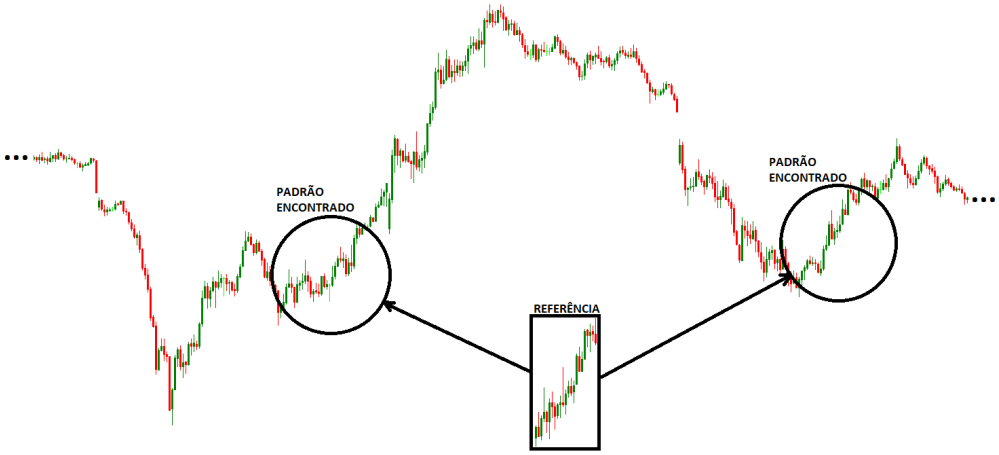
1 Projeto 'Candlestick Set Recognition'

1.0.1 Erivelton Marques de Carvalho

17-05-2019

1.1 Resumo

Este projeto executa um processo para reconhecer uma gráfico pequeno, sendo este a referência, dentro de outro bem maior. Podem ser encontradas em várias partes. O tipo de gráfico que iremos trabalhar é chamado de 'candlestick', aplicado no mercado financeiro, especificamente nas bolsas de valores. O sistema executará uma varredura, ponto a ponto, do início ao fim do gráfico maior e armazenará o conjunto de pontos que tem alta correlação com os pontos do gráfico de referência. Abaixo, segue a figura de um gráfico desse tipo com uma analogia do processo.



A partir dos resultados, o investidor pode tomar algumas decisões, como comprar um determinado lote de ações de uma empresa considerando que, segundo os padrões encontrados no gráfico de histórico (grafico maior), o preço tende a subir.

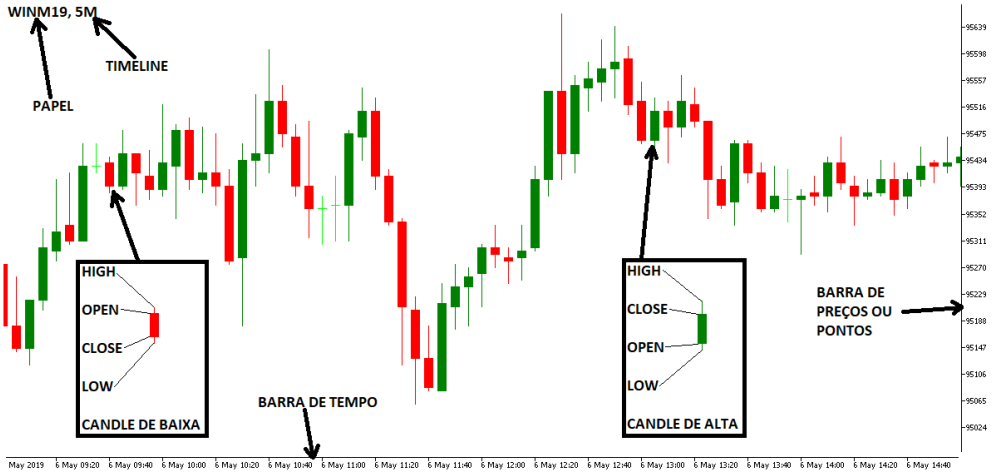
A métrica utilizada é o **Coeficiente de Correlação de 'Person'**, aplicado entre os conjuntos de registros/observações do gráfico de referência e de cada parte, de igual tamanho da referência, captada no gráfico de histórico no processo de varredura. A correlação de 'Person' é considerada o tipo normal dentre outras existentes e será utilizada por ser mais simples. E o valor da métrica, que servirá de fronteira entre as partes semelhantes ou não em relação ao gráfico de referência, será determinada pelo próprio usuário, lembrando que esse coeficiente é um número real entre '-1.0' e '1.0', significando que quanto mais próximo de '1.0', mais semelhante será.

Ao aplicar os cálculos de correlação, este projeto considera a comparação entre os pares de pontos na **forma bruta (raw)**, sem nenhuma transformação. Utilizaremos a linguagem de programação 'Python', com uso do ambiente de programação 'Jupyter Notebook'.

1.2 1. Conceitos

1.2.1 1.1 'Candlestick'

O gráfico de movimentações financeiras de uma ação de empresa ou índice de bolsa de valores pode ser do tipo 'candlestick' que tem como ponto básico o 'candle' por ser parecido com uma vela. A figura a seguir mostra uma parte de um gráfico desse tipo com a estrutura de 'candles'.



Onde: - '**HIGH**' é o valor máximo que o preço ou pontuação chegou dentro do tempo do 'candle'. - '**LOW**' é o valor mínimo que o preço ou pontuação chegou dentro do tempo do 'candle'. - '**OPEN**' é o valor de abertura de preço ou pontuação que iniciou o 'candle' atual ao terminar o tempo do 'candle' anterior. - '**CLOSE**' é o valor de fechamento de preço ou pontuação que terminou o 'candle' atual, dando início ao 'candle' seguinte. - '**CANDLE' DE ALTA** é o 'candle' onde o preço ou pontuação de fechamento ('**CLOSE**') é **MAIOR** que o de abertura ('**OPEN**'), informando que no tempo decorrido houve um aumento de valor. Neste caso, a alta no valor é representada pela cor verde do 'candle', mas não é padrão, podendo ser usada qualquer cor. - '**CANDLE' DE BAIXA** é o 'candle' onde o preço ou pontuação de fechamento ('**CLOSE**') é **MENOR** que o de abertura ('**OPEN**'), informando que no tempo decorrido houve uma diminuição de valor. Neste caso, a baixa no valor é representada pela cor vermelha do 'candle', mas não é padrão, podendo ser usada qualquer cor. - **PAPEL** é a identificação do produto financeiro, ou seja, o que está sendo negociado, podendo ser ações de empresas e índices da bolsa de valores. Neste caso, o papel '**WINM19**' significa que o gráfico é de negociação do Mini-Índice Futuro com vencimento em Junho de 2019, formado pela média das principais ações da **B3** (bolsa de valores brasileira, sediada em São Paulo - SP). Como outros exemplos, a B3 tem ações da Companhia Vale (VALE3), da Petrobrás (PETR3, PETR4), entre muitos outros. Maiores informações podem ser encontradas no portal da B3. - '**TIMELINE**' é o padrão de tempo para todos os 'candle'. Neste caso, **5M** significa que cada 'candle' representa a variação de preço ou pontuação, da abertura ao fechamento, em exatamente 5 minutos. algumas das 'timelines' usadas, são 1min, 5min, 15min, 1h, 4h, 1d, 1sem e 1mes. - **BARRA DE TEMPO** é a escala temporal no eixo horizontal. Neste caso, cada 'candle' dista em 5 minutos de um para o outro na sequência temporal. - **BARRA DE PREÇOS OU PONTOS** é a escala, no eixo vertical, de variação de preços de ações de empresas ou de pontos de índices de bolsas de valores. Neste caso, são mostrados pontuações de índices.

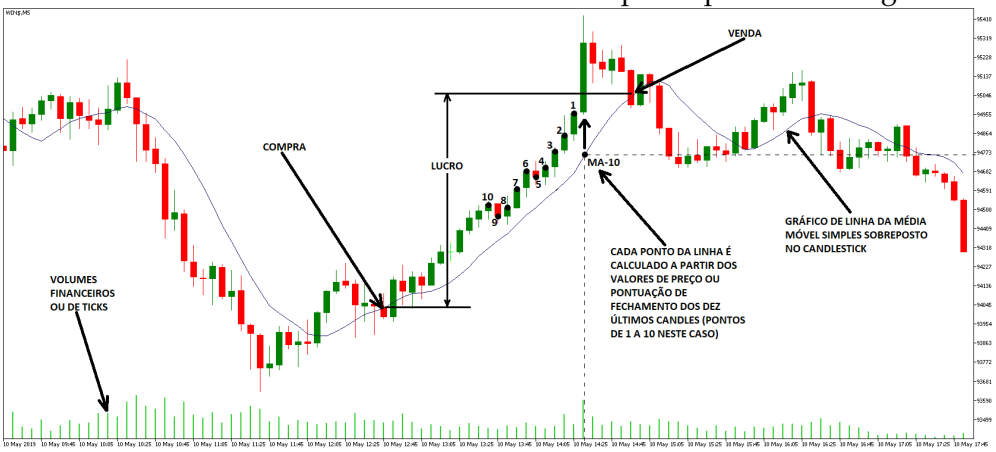
1.2.2 1.2 Plataforma de Negociação

É um software onde os operadores de bolsas de valores podem visualizar o gráfico e, mediante uma conta credenciada numa corretora que presta serviço a uma bolsa de valores, podem executar ordens de negociação de compra e venda de papéis. Há muitas plataformas e todas elas tem características comuns e próprias. Dentre as comuns, podemos alterar a 'timeline', as cores e o tipo de gráfico (há outros tipos além do candlestick, como de linha e OHLC).

Uma das plataformas de negociação mais difundidas no mundo é a **MetaTrader 5**, mantido pela empresa **MetaQuotes Software Corp.**. Seu instalador com instruções está disponível no site da MetaQuotes para o sistema operacional Windows. Usaremos o MetaTrader 5 para recuperarmos dados de um determinado papel usando uma conta do tipo 'demo' adquirido após cadastro em uma corretora brasileira que disponibiliza operações na B3 através dessa e de outras plataformas. Mas a própria MetaQuotes disponibiliza gratuitamente, sem a necessidade de uma corretora, uma conta 'demo' para qualquer usuário operar de forma simulada no mercado **FOREX** (tipo de mercado de abrangência mundial, semelhante a uma bolsa de valores, que disponibiliza operações de câmbio entre as principais moedas do mundo).

1.2.3 1.3 Indicadores e Robôs Investidores

Os **Indicadores** são ferramentas disponibilizadas pelas plataformas de negociação para auxiliar o operador na sua tomada de decisão. Por exemplo, temos a clássica **Média Móvel - MA (Movie Average)** que é o resultado da média aritmética dos 'N' últimos 'candles' considerando um dos preços padrões dos 'candles' (OPEN, HIGH, LOW, CLOSE), mas podendo ser usado outros critérios de cálculos para a média. Num exemplo hipotético, se considerarmos os dez últimos 'candles' e o preço 'CLOSE', será desenhado um gráfico de linha sobreposto ao 'candlestick' onde cada ponto da linha foi calculado pela média aritmética de todos os preços ou pontuações 'CLOSE' de cada um dos dez 'candles' conferido do referido ponto para trás. A figura abaixo mostra isso.



A média móvel, com suas variações, é só um dos inúmeros indicadores existentes. E a partir deles, podemos desenvolver os tão chamados **Robôs Investidores**. Na plataforma 'MetaTrader 5', eles são chamados de 'Expert Advisor - EA' e o usuário pode criar um EA baseado totalmente em um ou mais indicadores. A diferença entre indicador e robô é que a primeira é utilizada pelo operador para que ele mesmo dê entrada e saída de uma negociação. Por exemplo, numa tendência de subida do gráfico de 'candlestick', utilizando os devidos botões de comando da plataforma, o operador realiza uma ordem de compra em algum momento quando o preço ou pontuação atual toca na linha de média móvel e, posteriormente, realiza a venda no toque seguinte, concretizando algum lucro. A figura anterior também mostra essa negociação. Já o segundo, o robô, faz esse

processo de forma automática, sem intervenção do operador. É claro que desenvolver um robô não se resume a utilizar apenas indicadores, mas fica a critério do programador utilizar também código próprietário para otimizar as operações.

Devemos ter em mente no exemplo prático anterior, para melhor entendimento, que no momento do primeiro toque do preço ou pontuação atual com a média móvel, os 'candles' seguintes ainda serão formados e o operador deve prever para qual direção o valor irá, se cair ou subir, obtendo lucro ou prejuízo. E para otimizar isso, ele pode inclusive utilizar ferramentas estatísticas e até mesmo sofisticados algoritmos de inteligência artificial. Neste trabalho, utilizaremos apenas ferramentas estatísticas.

1.2.4 1.4 Volumes de Negócios

Dentre os tipos de indicadores quanto à sua origem, há os calculados e os originais. Os próprios 'candles' podem ser vistos como indicadores originais, enquanto a média móvel é calculada. Outros indicadores originais, coletados diretamente dos servidores de dados, são os volumes de negócios, sendo dois tipos, e o 'spreed'. Falaremos aqui dos volumes de negócios.

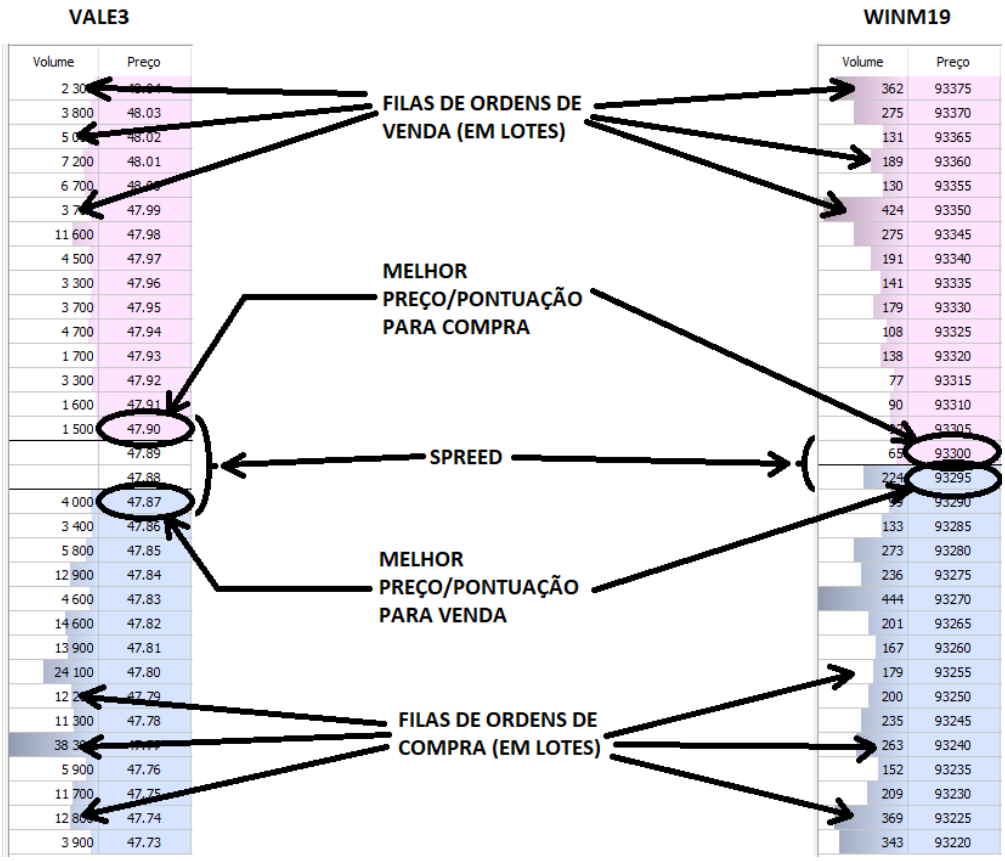
O primeiro tipo é o **Volume Financeiro**, no qual é registrado para cada 'candle' a soma de todos os **lotes** comprados e vendidos. O lote é a unidade básica de volume financeiro, sendo que cada papel tem seu valor. Por exemplo, o papel 'PETR4' da B3 tem o valor do lote a R\$26,62 e o valor para o papel 'WINM19' é de R\$25,00. Porém, no 'PETR4' só pode ser negociado multiplos de 100 lotes, enquanto no 'WINM19' pode ser negociado qualquer quantidade. Mas essas regras de negociação não terão efeito neste projeto. O operador pode compra e/ou vender a quantidade de lotes que quiser, dentro de seus limites financeiros, seguindo as regras de cada papel.

O segundo tipo é o **Volume de 'Ticks'**, no qual é registrado para cada 'candle' a soma de todos os '**ticks**' comprados e vendidos. O 'tick' é a unidade básica de movimentação de preço ou pontuação. Para entender melhor, saiba que o preço ou pontuação se movimenta de forma discreta, ou seja, sobe ou desce 'tick' a 'tick'. Por exemplo, o preço no papel 'PETR4' varia a cada R\$0,01. Se o preço desse papel estiver num determinado momento a R\$21,43 e um operador executa uma ordem de compra imediata, o sistema da B3, obviamente, irá executar essa compra com quem está vendendo. Assim, o sistema irá procurar uma ordem de venda de outro operador que esteja na fila de um 'tick' mais próximo, ou seja, a R\$21,44 , ou a R\$21,45 , ou a R\$21,46 , etc, sempre a cada R\$0,01. Para o papel 'WINM19', que utiliza pontuação em vez de preço, o 'tick' é de 5 pontos, ou seja, o valor do índice sobe ou desce a cada 5 pontos. Portanto, somando-se todos os 'ticks' movimentados, para cima e para baixo, dentro do tempo de um 'candle', temos o seu respectivo volume de 'ticks'. A figura anterior, do item '1.3', também mostra o indicador de volume, podendo ser financeiro ou de 'ticks'.

Todas essas informações podem ser encontradas com maiores detalhes na documentação da plataforma 'MetaTrader 5' e em muitos 'sites' especializados no assunto.

1.2.5 1.5 Spread e Profundidade de Mercado

Todo papel negociado tem dois preços ou pontuações correntes: (melhor) compra e (melhor) venda. O primeiro será sempre maior que o segundo e a diferença entre eles nos dá o '**spread**', como mostra a figura abaixo. Nessa figura, temos uma tabela dinâmica extraída da plataforma 'MetaTrader 5' que é chamada de profundidade de mercado, referentes aos papéis 'VALE3' e 'WINM19'. As figuras representam o dinamismo das negociações paralizadas em um dado momento. Para entender melhor, saiba que o sistema tem, basicamente, dois tipos de ordens para compra e venda: direta (a mercado) e posicionada (na fila).



Quando um operador envia uma ordem de **compra direta**, também conhecida como ordem a mercado, com uma determinada quantidade de lotes, o sistema compra essa quantidade diretamente no melhor preço ou pontuação de compra, que na verdade é uma fila de lotes de venda, pois sempre compramos de quem está vendendo. Caso a quantidade de lotes na fila do melhor preço/pontuação de compra seja maior que a determinada pelo operador, apenas há a subtração e a quantidade na fila é reduzida. Mas, quando tem quantidade menor, a subtração é feita e a fila nesse preço/pontuação é zerada. Para completar a compra dos lotes restantes da ordem, o sistema continua realizando a compra no preço/pontuação seguinte de compra (acima), desde que tenha lotes de venda suficientes na fila, passando a ser o próximo melhor preço/pontuação para compra. Se ainda não tiver lotes suficientes, o sistema passa para o próximo preço/pontuação, repetindo esse processo até acabar com os lotes de compra da ordem do operador. O que descrevemos aqui foi o processo de movimento do preço/pontuação para cima (alta). O mesmo acontece de forma análoga e inversa para uma ordem direta de venda, ocorrendo o processo de movimento do preço/pontuação para baixo (baixa).

Quando o operador envia uma ordem de **compra posicionada**, não há de fato uma execução da ordem, mas um posicionamento dos lotes em fila num preço ou pontuação determinado pelo próprio operador. Por exemplo, utilizando as informações da figura anterior, se ele enviar uma ordem de compra com 100 lotes para o papel 'VALE3' no preço R\$47,76 , esses lotes ficarão na fila do preço mensionado e se somarão aos 5900 lotes já existentes lá, resultando em 6000 lotes. O posicionamento significa que o operador espera que o melhor preço de compra do papel caia do atual R\$47,90 para R\$47,76 , quando o sistema executará, de fato, a compra dos 100 lotes que ele quer. Diferentemente da ordem direta, na posicionada não há movimento de preço caso a quantidade de lotes de compra na fila seja maior que a quantidade consumida (execução de

compra de quem está vendendo). Continuando no exemplo, se o melhor preço caiu até chegar na posição desejada do operador e, momentaneamente, houver uma execução que consumiu 70 lotes da ordem com o melhor preço subindo logo em seguida, a ordem ainda terá 30 lotes de compra na fila do preço desejado. Nesse caso, o operador deverá esperar o preço baixar novamente até que todos os lotes dele sejam consumidos. Todo esse processo acontece de forma análoga e inversa para uma ordem posicionada de venda.

Sobre o 'spread', é uma característica muito ligada ao dinamismo do preço/pontuação e liquidez (volume financeiro) do papel e é medida em 'ticks'. Há papéis que raramente o 'spread' é alto, como no 'WINM19' da figura anterior (93300 - 93295 = 5 = 1 'tick'). Já no 'VALE3', tem momentos de menor dinamismo, como também mostrado na figura anterior (47,90 - 47,87 = 0,03 = 3 'ticks'). Há papéis com 'spreads' muito altos, com mais de 5 'ticks', geralmente com liquidez muito baixa, significando pouco volume de negociação.

Todas essas informações também podem ser encontradas com maiores detalhes na documentação da plataforma 'MetaTrader 5' e em muitos 'sites' especializados no assunto.

1.3 2. Base de Dados

A base de dados ('dataset') utilizada neste projeto foi extraída por uma ferramenta de recuperação de dados históricos da plataforma 'MetaTrader 5' que faz um download a partir da base de dados da B3 por intermédio dos servidores de uma corretora. Esses dados são específicos do papel '**WINM19**', com 'timeline' de **5min**, com dezenas de milhares de 'candles' com seus respectivos preços de abertura, máximo, mínimo e de fechamento coletados entre as datas-horas '**02-04-2019 09:00:00**' e '**30-04-2019 11:35:00**', sempre dentro do horário de expediênte da B3 para esse papel que ocorre nos dias úteis entre **09:00:00** e **18:00:00**. Vejamos um 'overview' dos dados:

[1]:
```
# Carregando os dados do arquivo para um 'dataframe'...

import pandas as pd

df_raw = pd.read_csv('B3_CANDLES_WIN_M5_201804020900_201904301135.csv',␣
 ↪sep='\t')

# Mostrando os cinco primeiros rergistros do 'dataframe'...
print('Dimensões:',df_raw.shape)
df_raw.head()
```

```
Dimensões: (28580, 9)
```

[1]:

	<DATE>	<TIME>	<OPEN>	<HIGH>	<LOW>	<CLOSE>	<TICKVOL>	<VOL>	\
0	2018.04.02	09:00:00	91081	91225	90942	90995	5494	20340	
1	2018.04.02	09:05:00	90995	91017	90937	90979	4640	15613	
2	2018.04.02	09:10:00	90979	91097	90969	91070	4679	13901	
3	2018.04.02	09:15:00	91065	91102	91038	91043	3605	11205	
4	2018.04.02	09:20:00	91043	91065	91001	91043	3565	10508	

	<SPREAD>
0	1
1	1

```
2         1
3         1
4         1
```

[2]:
```
# Mostrando os cinco últimos registros do 'dataframe'...

df_raw.tail()
```

[2]:
```
           <DATE>    <TIME>  <OPEN>  <HIGH>  <LOW>  <CLOSE>  <TICKVOL>  <VOL>  \
28575  2019.04.30  11:15:00   96900   96935  96715    96725      20861  70833
28576  2019.04.30  11:20:00   96730   96850  96640    96670      27967  97054
28577  2019.04.30  11:25:00   96665   96750  96500    96560      27718  95275
28578  2019.04.30  11:30:00   96565   96660  96525    96645      12092  41762
28579  2019.04.30  11:35:00   96645   96680  96425    96525      25128  89698

       <SPREAD>
28575         1
28576         1
28577         1
28578         1
28579         1
```

O 'dataframe' extraido diretamente do arquivo apresenta as seguintes 'features': - **< DATE >** : Data de criação do 'candle'. - **< TIME >** : Hora de criação do 'candle'. - **< OPEN >** : Valor (pontuação) de abertura do 'candle'. - **< HIGH >** : Valor (pontuação) máximo do 'candle'. - **< LOW >** : Valor (pontuação) mínimo do 'candle'. - **< CLOSE >** : Valor (pontuação) de fechamento do 'candle'. - **< TICKVOL >** : Volume de ticks negociado no período do 'candle. - **< VOL >** : Volume financeiro negociado no período do 'candle'. - **< SPREAD >** : Diferença entre os melhores valores (pontuação) de compra e venda no fechamento do 'candle'.

Cada observação (linha de registro) corresponde a um 'candle' de 5min. Vamos agora reorganizar os nomes das 'features' removendo os sinais 'maior que' e 'menor que', bem como deixar mais adequado o nome 'VOL' alterando para 'FINVOL' (volume financeiro).

[3]:
```
# Alterando os nomes das 'features'...

df_raw.columns = ['DATE', 'TIME', 'OPEN', 'HIGH', 'LOW', 'CLOSE', 'TICKVOL',␣
 ↪'FINVOL', 'SPREAD']
df_raw.head()
```

[3]:
```
         DATE      TIME   OPEN   HIGH    LOW  CLOSE  TICKVOL  FINVOL  SPREAD
0  2018.04.02  09:00:00  91081  91225  90942  90995     5494   20340       1
1  2018.04.02  09:05:00  90995  91017  90937  90979     4640   15613       1
2  2018.04.02  09:10:00  90979  91097  90969  91070     4679   13901       1
3  2018.04.02  09:15:00  91065  91102  91038  91043     3605   11205       1
4  2018.04.02  09:20:00  91043  91065  91001  91043     3565   10508       1
```

Vamos observar se os dados estão devidamente tipificados.

[4]:
```
# Mostra os tipos de dados de todas as 'features'...

print(df_raw.dtypes)
```

```
DATE       object
TIME       object
OPEN        int64
HIGH        int64
LOW         int64
CLOSE       int64
TICKVOL     int64
FINVOL      int64
SPREAD      int64
dtype: object
```

Percebemos que 'DATE' e 'TIME' não estão de fato como de tipos data e hora. Portanto, vamos alterá-los e reorganizar o 'dataframe' para que as duas 'features' sejam uma só e do tipo 'data-hora'.

[5]:
```
# Alterando tipos de dados e 'features' novamente...

df_raw.DATE = list(str(a) + ' ' + str(b) for a, b in zip(df_raw.DATE.values,␣
 ↪df_raw.TIME.values))
df_raw = df_raw.drop(['TIME'], axis=1)
df_raw.rename(columns={'DATE': 'DATETIME'}, inplace=True)
df_raw.DATETIME = pd.to_datetime(df_raw.DATETIME)
print(df_raw.dtypes)
df_raw.head()
```

```
DATETIME    datetime64[ns]
OPEN                 int64
HIGH                 int64
LOW                  int64
CLOSE                int64
TICKVOL              int64
FINVOL               int64
SPREAD               int64
dtype: object
```

[5]:

	DATETIME	OPEN	HIGH	LOW	CLOSE	TICKVOL	FINVOL	SPREAD
0	2018-04-02 09:00:00	91081	91225	90942	90995	5494	20340	1
1	2018-04-02 09:05:00	90995	91017	90937	90979	4640	15613	1
2	2018-04-02 09:10:00	90979	91097	90969	91070	4679	13901	1
3	2018-04-02 09:15:00	91065	91102	91038	91043	3605	11205	1
4	2018-04-02 09:20:00	91043	91065	91001	91043	3565	10508	1

Agora vamos observar os dados estatísticos básicos.

[6]:
```
# Mostrando números estatísticos gerais do 'dataframe'...

df_raw.describe()
```

[6]:
```
                OPEN           HIGH           LOW          CLOSE  \
count   28580.000000   28580.000000   28580.00000   28580.000000
mean    88595.329986   88677.080651   88512.41627   88594.803079
std      7300.254217    7297.881916    7301.53694    7300.122394
min     72443.000000   72752.000000   72417.00000   72741.000000
25%     82000.000000   82076.000000   81921.00000   82000.000000
50%     89445.000000   89534.500000   89358.00000   89442.000000
75%     95320.000000   95391.000000   95250.00000   95320.000000
max    101802.000000  101827.000000  101732.00000  101802.000000

             TICKVOL         FINVOL   SPREAD
count   28580.000000   28580.000000  28580.0
mean    10562.033345   36113.442022      1.0
std      6749.222027   22402.810567      0.0
min       375.000000    1631.000000      1.0
25%      5825.000000   20326.750000      1.0
50%      9096.000000   31358.000000      1.0
75%     13649.250000   46610.000000      1.0
max     74144.000000  248721.000000      1.0
```

[7]:
```
# Verificando dados faltantes...

df_raw.isnull().sum()
```

[7]:
```
DATETIME    0
OPEN        0
HIGH        0
LOW         0
CLOSE       0
TICKVOL     0
FINVOL      0
SPREAD      0
dtype: int64
```

Percebemos que não há dados faltantes e que a 'feature' 'SPREAD' tem todas as amostras iguais a '1.0'. Portanto, podemos eliminá-la do 'dataframe'.

[8]:
```
# Removendo a 'feature' 'SPREAD'...

df_raw = df_raw.drop(['SPREAD'], axis=1)
df_raw.head()
```

[8]:

	DATETIME	OPEN	HIGH	LOW	CLOSE	TICKVOL	FINVOL
0	2018-04-02 09:00:00	91081	91225	90942	90995	5494	20340
1	2018-04-02 09:05:00	90995	91017	90937	90979	4640	15613
2	2018-04-02 09:10:00	90979	91097	90969	91070	4679	13901
3	2018-04-02 09:15:00	91065	91102	91038	91043	3605	11205
4	2018-04-02 09:20:00	91043	91065	91001	91043	3565	10508

1.4 3. Aplicando a Correlação de Person nos Dados Brutos

O que queremos ao comparar dois conjuntos de pontos, formando gráficos, é verificar se ambos tem variação semelhante, independente da distância entre os pares de valores de cada ponto dos conjuntos. Vamos usar a função 'pearsonr()' do módulo 'stats' da biblioteca 'scipy' que retorna uma tupla com o coeficiente de correlação e a probabilidade de ser ao acaso, nessa ordem, no qual iremos utilizar somente o primeiro.

A estrutura do processo de varredura no 'dataframe' para cálculo da correlação, requer o seguinte: - Um '**dataset' de dados brutos**, representando todo o **histórico do 'candlestick'**, por onde nosso sistema fará a varredura 'candle' a 'candle' na sequência temporal. - Um '**dataset' de referência**, bem menor, representando a(s) parte(s) que queremos encontrar no dataset de dados brutos. Chamaremos de '**subset' padrão**. - '**Datasets' de comparação**, sendo todos do mesmo tamanho do 'subset' padrão, extraídos 'candle' a 'candle' na varredura do 'dataset' de dados brutos. Chamaremos de '**subsets' extraídos**. - Uma **referência de pontuação** de cada 'candle' para comporem os elementos dos 'subsets'. Podemos utilizar qualquer uma das 'features', mas neste projeto vamos nos concentrar na **pontuação de fechamento - 'CLOSE'**, pois é a mais utilizada pelos operadores de bolsas de valores.

Para começar, vamos escolher, hipoteticamente, um 'subset' padrão. Poderíamos extrair de outro arquivo baixado da B3 pela plataforma 'MetaTrader 5', fora do intervalo de datas do arquivo que gerou nosso 'dataframe', mas vamos extrair do próprio 'dataset' de dados brutos. Isso nos dá a convicção de que um dos 'subsets' extraídos na varredura deve ser exatamente o mesmo do padrão, obtendo coeficiente de correlação igual a '1.0'. Os códigos a seguir mostram a parte do gráfico de 'candlestick' onde extraímos nosso 'subset' padrão, usando a biblioteca 'plotly' que incorpora ferramentas para gráficos de 'candlestick'. Vamos analisá-lo.

[9]:
```
# Bibliotecas necessárias para o tratamento de gráficos do tipo 'candlestick'...

import plotly.offline as py
import plotly.graph_objs as go
import matplotlib.pyplot as plt
from datetime import datetime
import numpy as np
py.init_notebook_mode()
```

[10]:
```
# Extração de uma parte do grande 'dataset' de dados brutos...

df_raw_part = df_raw[np.logical_and(df_raw['DATETIME'] >= '2018-10-23 09:45:00',
                                    df_raw['DATETIME'] <= '2018-10-25 15:55:
 ↪00')]
print('Dimensões:', df_raw_part.shape)
df_raw_part.head()
```

```
Dimensões: (289, 7)
```

[10]:

	DATETIME	OPEN	HIGH	LOW	CLOSE	TICKVOL	FINVOL
15096	2018-10-23 09:45:00	87608	87680	87536	87572	10401	28198
15097	2018-10-23 09:50:00	87572	87587	87443	87459	9001	23003
15098	2018-10-23 09:55:00	87459	87505	87289	87351	14013	36648
15099	2018-10-23 10:00:00	87351	87361	87243	87269	12587	36902
15100	2018-10-23 10:05:00	87269	87366	87259	87351	11391	33163

[11]:
```
# Gráfico de 'candlestick' tomando como base temporal a 'feature' 'DATETIME'...

trace = go.Candlestick(x=df_raw_part['DATETIME'],
                       open=df_raw_part['OPEN'],
                       high=df_raw_part['HIGH'],
                       low=df_raw_part['LOW'],
                       close=df_raw_part['CLOSE'])

layout = go.Layout(xaxis = dict(rangeslider = dict(visible = False)))

data = [trace]

fig = go.Figure(data=data,layout=layout)
py.offline.iplot(fig, filename='simple_candlestick_without_range_slider')
```

Tomando como base temporal, no eixo horizontal, a 'feature' 'DATETIME', observamos claramente a falta de registros de 'candles' fora do horário de expediente de trabalho da bolsa de valores. Isso não é bom para nosso trabalho. Portanto, iremos considerar, a partir de agora, o índice do 'dataset' de dados brutos como nossa base temporal, eliminando os espaços vazios fora do

expediente, mas mantendo o mesmo intervalo de data e hora. Então, podemos escolher nosso 'subset' padrão.

[12]:
```
# Gráfico de 'candlestick' tomando como base temporal o índice do 'dataset' de␣
 ↪dados brutos
# e mostrando o nosso 'subset' padrão...

trace = go.Candlestick(x=df_raw_part.index,
                       open=df_raw_part['OPEN'],
                       high=df_raw_part['HIGH'],
                       low=df_raw_part['LOW'],
                       close=df_raw_part['CLOSE'])

layout = {'title': "Parte do 'dataset' e escolha do 'subset' padrão",
          'yaxis': {'title': 'Pontos'},
          'xaxis': {'rangeslider': {'visible': False}},
          'shapes': [{'x0': 15297,
                      'x1': 15304,
                      'y0': 0, 'y1': 1, 'xref': 'x', 'yref': 'paper',
                      'line': {'color': 'rgb(30,30,30)', 'width': 1}},
                     {'x0': 15203,
                      'x1': 15261,
                      'y0': 0, 'y1': 1, 'xref': 'x', 'yref': 'paper',
                      'line': {'color': 'rgb(30,30,30)', 'width': 1}}
                    ],
          'annotations': [{'x': 15304,
                           'y': 0.05,
                           'xref': 'x', 'yref': 'paper',
                           'showarrow': False, 'xanchor': 'left',
                           'text': 'Salto na pontuação'},
                          {'x': 15203,
                           'y': 0.05,
                           'xref': 'x', 'yref': 'paper',
                           'showarrow': False, 'xanchor': 'left',
                           'text': "'Subset' padrão escolhido"}
                          ]
         }

data = [trace]

fig = go.Figure(data=data,layout=layout)
py.offline.iplot(fig, filename='simple_candlestick_without_range_slider')
```

Nessa parte do 'dataset', eliminamos os espaços vazios em relação ao eixo horizontal, mas no eixo vertical percebemos um salto de pontuação do índice '15300' para o '15301'. Isso acontece porque nas primeiras negociações no início do expediente de trabalho o preço ou pontuação negociado está distante do último valor do dia anterior. Nesse papel, esses saltos são comuns e vamos aceitar normalmente em nosso projeto. Abaixo, temos nosso 'subset' padrão que utilizaremos em todo nosso trabalho.

[13]:
```
# Nosso 'subset' padrão...

subset_default = df_raw_part.loc[15203:15261]
print('Dimensões:', subset_default.shape)
subset_default.head()
```

```
Dimensões: (59, 7)
```

[13]:

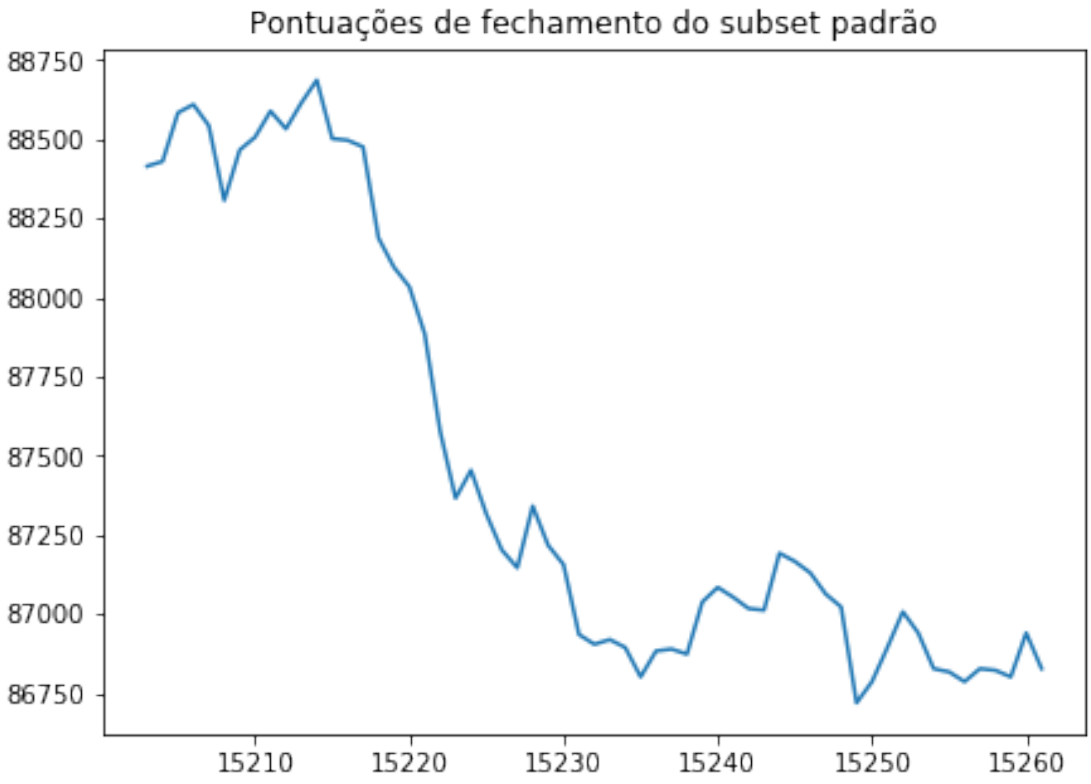
	DATETIME	OPEN	HIGH	LOW	CLOSE	TICKVOL	FINVOL
15203	2018-10-24 09:45:00	88291	88435	88271	88415	10671	30563
15204	2018-10-24 09:50:00	88415	88517	88368	88430	14593	39926
15205	2018-10-24 09:55:00	88430	88589	88394	88584	10034	29821
15206	2018-10-24 10:00:00	88579	88625	88502	88610	11614	34996
15207	2018-10-24 10:05:00	88599	88646	88486	88543	11067	34382

[14]:
```
# Gráfico de 'candlestick' do nosso 'subset' padrão...

trace = go.Candlestick(x=subset_default.index,
                       open=subset_default['OPEN'],
                       high=subset_default['HIGH'],
                       low=subset_default['LOW'],
```

```
                close=subset_default['CLOSE'])

layout = go.Layout(title = "Amostra ampliada do 'Subset' padrão escolhido",
                   xaxis = dict(rangeslider = dict(visible = False)))

data = [trace]

fig = go.Figure(data=data,layout=layout)
py.offline.iplot(fig, filename='simple_candlestick_without_range_slider')
```

Agora, vamos trabalhar na aplicação da correlação. Como já foi dito, escolhemos trabalhar com a pontuação de fechamento ('CLOSE'). Assim, podemos plotar um gráfico de linha somente com essas pontuações do 'subset' padrão.

[15]:
```
# Gráfico de linha das pontuações de fechamento do 'subset' padrão...

fig = plt.figure(figsize=(7,5))
plt.plot(subset_default.index, subset_default.CLOSE)
plt.title("Pontuações de fechamento do subset padrão")
plt.show()
```



Determinamos, então, uma função que faz a varredura em todo o 'dataset', comparando cada 'subset' extraído com o padrão, retornando aqueles com coeficiente de correlação acima de um determinado valor. Para esse processo, seguiremos as seguintes premissas: - Criação de um parâmetro de fronteira para o coeficiente de correlação com o objetivo de separar os 'subsets' extraídos que sejam semelhantes dos não semelhantes ao padrão, caso seus coeficientes estejam acima ou abaixo do valor desse parâmetro. - Para o cálculo da correlação, evitando dados em branco, os 'subsets' extraídos devem ter exatamente o mesmo tamanho do padrão. - Vamos considerar que a função irá receber a 'feature' que se quer referenciar nos cálculos, não obrigatoriamente sendo a 'CLOSE', dando mais flexibilidade. - A varredura será feita do primeiro ao último 'candle' possível do 'dataset'. Uma ilustração de parte desse passo é mostrada no gráfico abaixo. Devemos considerar que o último 'candle' da varredura é aquele que coincide com a posição do último menos o tamanho do 'subset' padrão para que não tenhamos 'subsets' extraídos com tamanho menor. - A função retornará uma tupla com dois elementos. O primeiro será uma lista contendo as posições iniciais no 'dataset' onde foram encontradas 'subsets' extraídos semelhantes ao padrão. E o segundo elemento será uma lista dos coeficientes de correlação de 'Person' correspondente, na mesma ordem, a cada posição da lista do primeiro elemento.

[16]:
```
# Demonstração gráfica da varredura de 'subsets' extraídos...

df_raw_part2 = df_raw_part.loc[15096:15170]
trace = go.Candlestick(x=df_raw_part2.index,
```

```
                open=df_raw_part2['OPEN'],
                high=df_raw_part2['HIGH'],
                low=df_raw_part2['LOW'],
                close=df_raw_part2['CLOSE'])

layout = {'title': "Demonstração gráfica da varredura de 'subsets' extraídos",
          'yaxis': {'title': 'Pontos'},
          'xaxis': {'rangeslider': {'visible': False}},
          'shapes': [{'x0': 15119, 'x1': 15146, 'y0': 0.05, 'y1': 0.9, 'xref':␣
 ↪'x', 'yref': 'paper',
                      'line': {'color': 'rgb(30,30,30)', 'width': 1}},
                     {'x0': 15120, 'x1': 15147, 'y0': 0.1, 'y1': 0.95, 'xref':␣
 ↪'x', 'yref': 'paper',
                      'line': {'color': 'rgb(30,30,30)', 'width': 1}},
                     {'x0': 15121, 'x1': 15148, 'y0': 0.15, 'y1': 1.0, 'xref':␣
 ↪'x', 'yref': 'paper',
                      'line': {'color': 'rgb(30,30,30)', 'width': 1}}
                    ],
          'annotations': [{'x': 15117, 'y': 0.5, 'xref': 'x', 'yref': 'paper',␣
 ↪'showarrow': False, 'xanchor': 'left',
                           'text': '...'},
                          {'x': 15119, 'y': 0.05, 'xref': 'x', 'yref': 'paper',␣
 ↪'showarrow': False, 'xanchor': 'left',
                           'text': 'subset n'},
                          {'x': 15120, 'y': 0.10, 'xref': 'x', 'yref': 'paper',␣
 ↪'showarrow': False, 'xanchor': 'left',
                           'text': 'subset n+1'},
                          {'x': 15121, 'y': 0.15, 'xref': 'x', 'yref': 'paper',␣
 ↪'showarrow': False, 'xanchor': 'left',
                           'text': 'subset n+2'},
                          {'x': 15148, 'y': 0.5, 'xref': 'x', 'yref': 'paper',␣
 ↪'showarrow': False, 'xanchor': 'left',
                           'text': '...'},
                         ]
        }

data = [trace]

fig = go.Figure(data=data,layout=layout)
py.offline.iplot(fig, filename='simple_candlestick_without_range_slider')
```

[17]:
```
# Função de extração de 'subsets' semelhantes a um padrão.

def get_similar_subsets(dataset, subset, col_dataset='CLOSE',␣
↪col_subset='CLOSE', min_rate_corr=0.9, printout=False):

    """
    Esta função faz uma busca por conjuntos menores de 'candles' em sequência␣
↪numa base maior
    conforme um conjunto padrão. A métrica de correlação de 'Pearson' é␣
↪utilizada como fronteira
    para dividir os conjuntos de 'candles' em semelhante ou não ao padrão. O␣
↪retorno de valor
    desta função é uma tupla com duas listas, sendo a primeira representando as␣
↪posições no
    'dataset' dos conjuntos semelhantes ao padrão e a segunda representando os␣
↪coeficientes de
    correlação deles.

    VARIÁVEIS DE ENTRADA:

    - dataset: tipo 'pandas.DataFrame'
        É um 'dataframe', representando um conjunto de 'candles', onde será␣
↪extraída as posições
        que contenham subconjuntos semelhantes ao padrão.

    - subset: tipo 'pandas.DataFrame'
        É um 'dataframe', representando um subconjunto de 'candles', que␣
↪servirá de padrão de
        semelhança para os subconjuntos extraídos do 'dataset'.
```

```
    - col_dataset: tipos 'str' ou 'int'
        É o nome ou número da coluna do 'dataset' para ser usada no cálculo da␣
↪correlação.

    - col_subset: tipos 'str' ou 'int'
        É o nome ou número da coluna do 'subset' para ser usada no cálculo da␣
↪correlação.

    - min_rate_corr: tipos 'float' ou 'int' no espaço [-1.0,1.0]
        É o parâmetro que representa o mínimo valor do coeficiente de␣
↪correlação de 'Pearson' que
        servirá de fronteira para os 'subsets' extraídos, sendo que cada um␣
↪terá seu coeficiente
        calculado, onde esse parâmetro irá classificá-los internamente em␣
↪semelhante ou não ao
        padrão, separando os semelhantes como resultado da função.

    - printout: tipo 'bool'
        É um ativador que serve para imprimir ou não na saída indicadores␣
↪internos.

    RESULTADO:

    - return: tipo 'tuple(list,list)'
        Esta função retorna uma tupla com dois elementos. O primeiro é uma␣
↪lista contendo as
        posições (índices do 'dataset') que representam o primeiro 'candle' de␣
↪cada 'subset'
        semelhante extraído. O segundo é uma lista, na mesma ordem da primeira,␣
↪que representa os
        coeficientes de correlação de 'Pearson' de todos os 'subsets'␣
↪semelhantes.
    """

    ## Tratamento de erros das variáveis de entrada.

    import pandas

    # Verificação de tipos.
    if type(dataset) != pandas.DataFrame:
        raise ValueError("'dataset' deve ser do tipo 'pandas.DataFrame'.")
    if type(subset) != pandas.DataFrame:
        raise ValueError("'subset' deve ser do tipo 'pandas.DataFrame'.")
    if not(type(col_dataset) == str or type(col_dataset) == int):
        raise ValueError("'col_dataset' deve ser do tipo 'str' ou 'int'.")
```

```
    if not(type(col_subset) == str or type(col_subset) == int):
        raise ValueError("'col_subset' deve ser do tipo 'str' ou 'int'.")
    if not(type(min_rate_corr) == int or type(min_rate_corr) == float):
        raise ValueError("'rate_corr' deve ser do tipo 'int' ou 'float'.")

    # Verificação de conteúdo.
    if subset.shape[0] > dataset.shape[0]:
        raise ValueError("Tamanho do 'subset' (quantidade de registros) deve␣
↪ser menor ou igual do 'dataset'.")
    if type(col_dataset) == str:
        if not(col_dataset in dataset.columns.values):
            raise ValueError("Não há nenhuma coluna no 'dataset' com o nome '"␣
↪+ col_dataset + "'.")
        else:
            try:
                float(dataset[col_dataset].values[0])
            except:
                raise ValueError("Os dados da coluna " + col_dataset + " do␣
↪'dataset' devem ser numéricos.")
    else:
        try:
            float(dataset[dataset.columns[col_dataset]].values[0])
        except:
            raise ValueError("Os dados da coluna " + col_dataset + " do␣
↪'dataset' devem ser numéricos.")
        if col_dataset < 0:
            raise ValueError("'col_dataset' deve ser maior ou igual a zero.")
        if col_dataset >= len(dataset.columns.values):
            raise ValueError("Não há nenhuma coluna no 'dataset' com o índice␣
↪'" + str(col_dataset) + "'.")
    if type(col_subset) == str:
        try:
            float(subset[col_subset].values[0])
        except:
            raise ValueError("Os dados da coluna " + col_subset + " do 'subset'␣
↪devem ser numéricos.")
        if not(col_subset in subset.columns.values):
            raise ValueError("Não há nenhuma coluna no 'subset' com o nome '" +␣
↪col_subset + "'.")
    else:
        try:
            float(subset[subset.columns[col_subset]].values[0])
        except:
            raise ValueError("Os dados da coluna " + col_subset + " do 'subset'␣
↪devem ser numéricos.")
        if col_subset < 0:
```

```
            raise ValueError("'col_subset' deve ser maior ou igual a zero.")
        if col_subset >= len(subset.columns.values):
            raise ValueError("Não há nenhuma coluna no 'subset' com o índice '"␣
↪+ str(col_subset) + "'.")
    if float(min_rate_corr) < -1.0 or float(min_rate_corr) > 1.0:
        raise ValueError("'rate_corr' deve estar no intervalo [-1.0,1.0].")

    ## Corpo principal da função.

    # Pegando os índices das colunas caso as variáveis de entrada 'col_dataset'␣
↪e 'col_subset' sejam 'strings'.
    if type(col_dataset) == str:
        col_dataset = list(dataset.columns.values).index(col_dataset)
    if type(col_subset) == str:
        col_subset = list(dataset.columns.values).index(col_subset)

    # Carregamento das bibliotecas necessárias para napoio e cálculo de␣
↪correlações.
    from scipy import stats
    from time import time
    from tqdm import tqdm

    # Varredura no 'dataset' para encontrar 'subsets' semelhantes ao padrão␣
↪através do cálculo da correlação
    # de 'Pearson' entre as colunas dadas em 'col_dataset' e 'col_subset' de␣
↪cada dataframe de entrada.
    size_subset = subset.shape[0]
    list_subset_similar = []
    list_corr_subset = []
    last_position = dataset.shape[0] - 1 - size_subset
    tm = time()
    for i in tqdm(range(last_position + 1)):
        if i > last_position:
            break
        subset_extracted = dataset.iloc[i:i+size_subset,:]
        corr,_ = stats.pearsonr(subset_extracted[subset_extracted.
↪columns[col_dataset]].values,
                                subset[subset.columns[col_subset]].values)
        if corr >= min_rate_corr:
            list_subset_similar.append(dataset.index.values[i])
            list_corr_subset.append(corr)
    if printout:
        print('- Varredura executada em', round(time()-tm,2), '\b(s).')
        print('- Foram encontrados', len(list_subset_similar), '\'subsets\'␣
↪extraídos semelhantes ao padrão',
```

```
            'utilizando as colunas ', "'"+dataset.columns[col_dataset]+"'",␣
↪'do \'dataset\' e',
            "'"+subset.columns[col_subset]+"'", 'do \'subset\' com␣
↪coeficientes de correlação de',
            '\'Pearson\' igual ou maior que \'' + str(min_rate_corr) + '\'.')

  return list_subset_similar, list_corr_subset
```

A função no código acima é genérica, flexível. Não tem obrigatoriedade de que os dataframes sejam representações de 'candlesticks', podendo ser de qualquer natureza, desde que as colunas escolhidas obedeçam as regras de tipos das variáveis de entrada. Neste trabalho, vamos utilizar essa função para usar a 'feature' 'CLOSE' do nosso 'dataset' de dados brutos e do 'subset' padrão que escolhemos. Como primeiro exemplo, vamos aplicar a função diretamente com os parâmetros padrões (colunas com 'CLOSE' e coeficiente de correlação mínima de '0.9'), mas mostrando o resultado na saída.

[19]:
```
subsets1, corrs1 = get_similar_subsets(df_raw, subset_default, printout=True)
```

```
100%||
28521/28521 [00:06<00:00, 4479.11it/s]

- Varredura executada em 6.39(s).
- Foram encontrados 627 'subsets' extraídos semelhantes ao padrão utilizando as
colunas  'CLOSE' do 'dataset' e 'CLOSE' do 'subset' com coeficientes de
correlação de 'Pearson' igual ou maior que '0.9'.
```

Nessa primeira tentativa, obtivemos uma quantidade muito grande de 'subsets' reconhecidos, dificultando a análise. Vamos repetir o processo aumentando o valor mínimo de coeficiente de correlação para reduzir essa quantidade.

[21]:
```
subsets2, corrs2 = get_similar_subsets(df_raw, subset_default, min_rate_corr=0.
↪95, printout=True)
```

```
100%||
28521/28521 [00:06<00:00, 4446.97it/s]

- Varredura executada em 6.42(s).
- Foram encontrados 30 'subsets' extraídos semelhantes ao padrão utilizando as
colunas  'CLOSE' do 'dataset' e 'CLOSE' do 'subset' com coeficientes de
correlação de 'Pearson' igual ou maior que '0.95'.
```

Agora, com bem menos 'subsets' reconhecidos, vamos mostrar se de fato são semelhantes ao padrão. Para simplificar, vamos utilizar o gráfico de linhas com apenas os dados da 'feature' 'CLOSE', lembrando que a lista do resultado da função anterior não são conjuntos inteiros de 'candles', mas apenas o índice no 'dataset' de dados brutos para o primeiro dos 'candles de cada 'subset' reconhecido dentro desse 'dataset'.

[22]:
```
# Confirmando os dados do resultado...

print("Índices em 'df_raw' de localização inicial dos 'subsets' reconhecidos:")
print(subsets2)
print("-----------------------------------------")
print("Coeficientes de correlação, na mesma ordem, de cada 'subset' reconhecido:
 ↪")
print(corrs2)
```

```
Índices em 'df_raw' de localização inicial dos 'subsets' reconhecidos:
[365, 2153, 2154, 2155, 9737, 15067, 15200, 15201, 15202, 15203, 15204, 15205,
15206, 19551, 19552, 19553, 20034, 20035, 20036, 20037, 22018, 22019, 22020,
27399, 28115, 28116, 28117, 28118, 28119, 28120]
-----------------------------------------
Coeficientes de correlação, na mesma ordem, de cada 'subset' reconhecido:
[0.9530217193164404, 0.9525798667728674, 0.9574178211979937, 0.9583498564537077,
0.9543742135751833, 0.9526560886216351, 0.9530390541096287, 0.9689221839351155,
0.985407153699156, 1.0, 0.9851166455505797, 0.9679102687040013,
0.951059030176661, 0.9537548707084549, 0.9588287028528437, 0.9560382996174929,
0.9614444666919035, 0.9623493751564967, 0.9626586331918893, 0.9539083314006024,
0.9520593861503935, 0.9542572959099037, 0.9511307978803133, 0.950084255505282,
0.9514522133755656, 0.9594018929324194, 0.9664292094609749, 0.9732573257375167,
0.9712667857760724, 0.9584364247311837]
```

[23]:
```
# Mostrando graficamente os 'subsets' reconhecidos...

fig = plt.figure(figsize=(16,70))
quad_y = 2
quad_x = int(len(subsets2) / quad_y)
if len(subsets2) % quad_y:
    quad_x += 1
for i in range(len(subsets2)):
    # Extrai do 'dataset' de dados brutos o 'subset' reconhecido.
    subset = df_raw.loc[subsets2[i]:subsets2[i]+subset_default.shape[0]-1]
    ax = fig.add_subplot(quad_x, quad_y, i + 1)
    ax.plot(subset.index.values, subset.CLOSE.values)
    ax.set_title("Subset reconhecido na posição do índice " + str(subsets2[i]) +
                 ". Corr=" + str(round(corrs2[i],2)) + ".")
plt.show()
```


Subset reconhecido na posição do índice 365. Corr=0.95
Subset reconhecido na posição do índice 2153. Corr=0.95
Subset reconhecido na posição do índice 2154. Corr=0.96
Subset reconhecido na posição do índice 2155. Corr=0.96
Subset reconhecido na posição do índice 9737. Corr=0.95
Subset reconhecido na posição do índice 15067. Corr=0.95
Subset reconhecido na posição do índice 15200. Corr=0.95
Subset reconhecido na posição do índice 15201. Corr=0.97
Subset reconhecido na posição do índice 15202. Corr=0.99
Subset reconhecido na posição do índice 15203. Corr=1.0
Subset reconhecido na posição do índice 15204. Corr=0.99
Subset reconhecido na posição do índice 15205. Corr=0.97
Subset reconhecido na posição do índice 15206. Corr=0.95
Subset reconhecido na posição do índice 19551. Corr=0.95
Subset reconhecido na posição do índice 19552. Corr=0.96
Subset reconhecido na posição do índice 19553. Corr=0.96
Subset reconhecido na posição do índice 20034. Corr=0.96
Subset reconhecido na posição do índice 20035. Corr=0.96
Subset reconhecido na posição do índice 20036. Corr=0.96
Subset reconhecido na posição do índice 20037. Corr=0.95
Subset reconhecido na posição do índice 22018. Corr=0.95
Subset reconhecido na posição do índice 22019. Corr=0.95
Subset reconhecido na posição do índice 22020. Corr=0.95
Subset reconhecido na posição do índice 27399. Corr=0.95
Subset reconhecido na posição do índice 28115. Corr=0.95
Subset reconhecido na posição do índice 28116. Corr=0.96
Subset reconhecido na posição do índice 28117. Corr=0.97
Subset reconhecido na posição do índice 28118. Corr=0.97
Subset reconhecido na posição do índice 28119. Corr=0.97
Subset reconhecido na posição do índice 28120. Corr=0.96

Observando os gráficos dos 'subsets' reconhecidos, percebemos com facilidade a semelhança deles com o 'subset' padrão. É exatamente isso que queremos. Porém, temos dois problemas a resolver, nos obrigando a limpar a lista de 'subsets' reconhecidos. O primeiro problema é a presença de 'subsets' muito próximos, sendo vizinhos no 'dataset' de dados brutos, sendo considerados redundâncias. Isso pode ser visto entre os gráficos acima, por exemplo, os 'subsets' em sequência de posições 20034, 20035, 20036 e 20037. O segundo problema é a presença do 'subset' exatamente igual ao padrão, visto que nosso 'subset' padrão é uma cópia de parte desse 'dataset'. Isso significa que nosso 'subset' padrão encontrou ele mesmo no 'dataset' de dados brutos, obviamente com coeficiente de correlação igual a '1.0'. Além disso, esse segundo problema vem acompanhando do primeiro.

Para limpár-mos a lista de 'subsets' reconhecidos, vamos executar dois passos. Primeiro, vamos remover os subsets vizinhos, deixando aquele que tiver o maior coeficiente de correlação.

[24]:
```
# Função de remoção de 'subsets' vizinhos...

def subsets_cleaner(subsets, corrs):
    """Remove 'subsets' vizinhos, deixando aquele com maior coeficiente de␣
 ↪correlação."""
    list_subsets_cleaned = []
    list_corrs_cleaned = []
    for i in range(len(subsets)):
        if i > 0:
            subset_current = subsets[i]
            corr_current = corrs[i]
            if subsets[i-1] == subset_current - 1:
                if corr_current > list_corrs_cleaned[-1]:
                    list_subsets_cleaned[-1] = subset_current
                    list_corrs_cleaned[-1] = corr_current
            else:
                list_subsets_cleaned.append(subset_current)
                list_corrs_cleaned.append(corr_current)
        else:
            list_subsets_cleaned.append(subsets[0])
            list_corrs_cleaned.append(corrs[0])
    return list_subsets_cleaned, list_corrs_cleaned
```

[25]:
```
# Aplicando a função em nosso resultado...

subsets2_cleaned, corrs2_cleaned = subsets_cleaner(subsets2, corrs2)
print("'Subsets' originais:")
print(subsets2)
print("'Subsets' limpos:")
print(subsets2_cleaned)
print("-----------------------------------")
```

```
print("Coeficientes originais:")
print(corrs2)
print("Coeficientes limpos:")
print(corrs2_cleaned)
```

```
'Subsets' originais:
[365, 2153, 2154, 2155, 9737, 15067, 15200, 15201, 15202, 15203, 15204, 15205,
15206, 19551, 19552, 19553, 20034, 20035, 20036, 20037, 22018, 22019, 22020,
27399, 28115, 28116, 28117, 28118, 28119, 28120]
'Subsets' limpos:
[365, 2155, 9737, 15067, 15203, 19552, 20036, 22019, 27399, 28118]
-----------------------------------
Coeficientes originais:
[0.9530217193164404, 0.9525798667728674, 0.9574178211979937, 0.9583498564537077,
0.9543742135751833, 0.9526560886216351, 0.9530390541096287, 0.9689221839351155,
0.985407153699156, 1.0, 0.9851166455505797, 0.9679102687040013,
0.951059030176661, 0.9537548707084549, 0.9588287028528437, 0.9560382996174929,
0.9614444666919035, 0.9623493751564967, 0.9626586331918893, 0.9539083314006024,
0.9520593861503935, 0.9542572959099037, 0.9511307978803133, 0.950084255505282,
0.9514522133755656, 0.9594018929324194, 0.9664292094609749, 0.9732573257375167,
0.9712667857760724, 0.9584364247311837]
Coeficientes limpos:
[0.9530217193164404, 0.9583498564537077, 0.9543742135751833, 0.9526560886216351,
1.0, 0.9588287028528437, 0.9626586331918893, 0.9542572959099037,
0.950084255505282, 0.9732573257375167]
```

Após a limpeza, ficamos com uma quantidade menor de 'subsets' reconhecidos, no qual um deles (de coeficiente '1.0') é o nosso 'subset' padrão dentro do 'dataset' de dados brutos. Podemos simplesmente ignorá-lo, mas numa situação real, esperamos que o 'subset' padrão seja escolhido de outra fonte que não seja o 'dataset' de histórico. Por exemplo, podemos extrair em tempo real no mercado financeiro. Portanto, vamos remover da lista de 'subsets' reconhecidos, bem como de seus respectivos coeficientes, o elemento que é nosso 'subset' padrão.

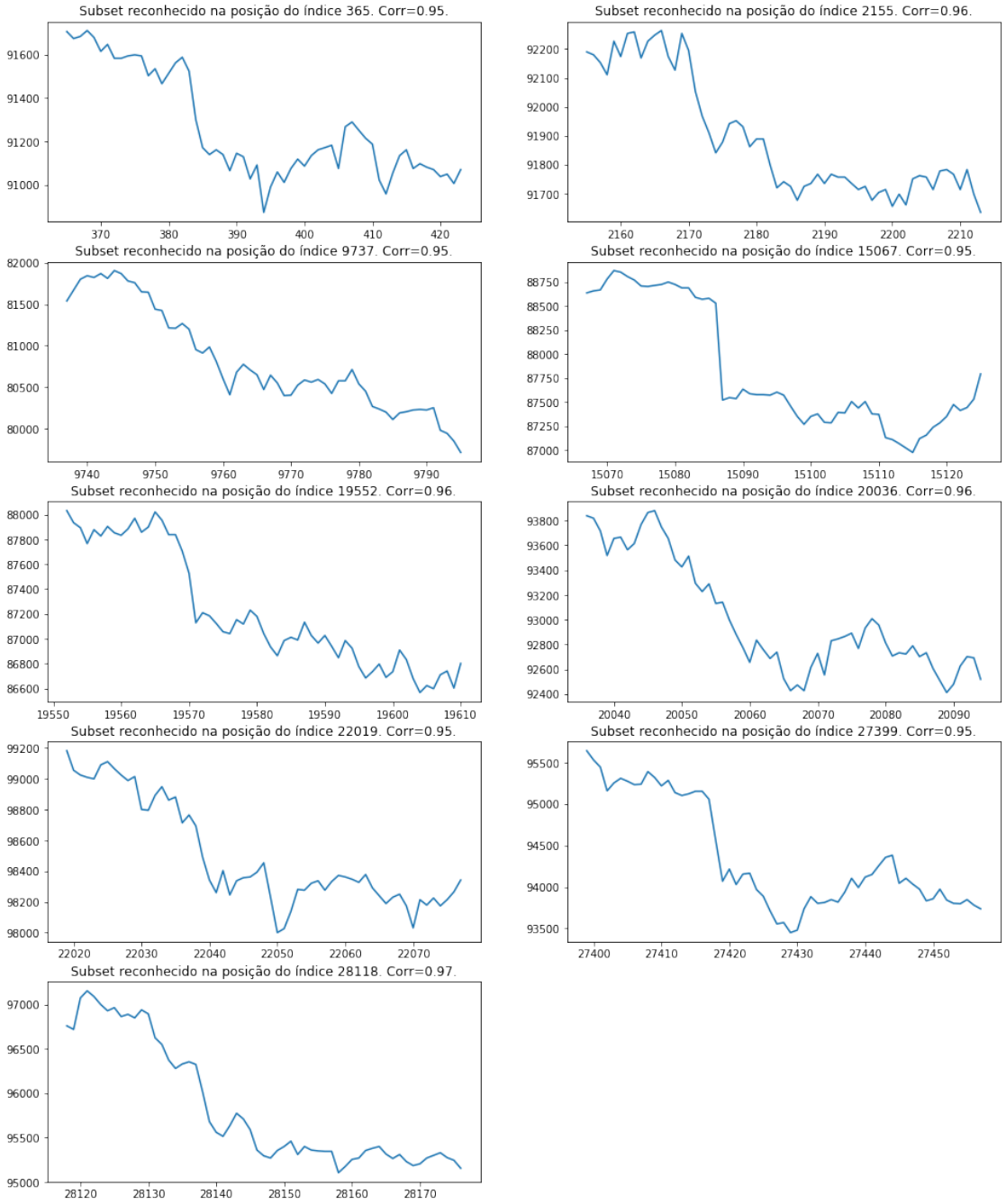
[26]:
```
# Removendo das listas nosso 'subset' padrão...

if subset_default.index.values[0] in subsets2_cleaned:
    first_index_subset_default = subsets2_cleaned.index(subset_default.index.
 ↪values[0])
    del(subsets2_cleaned[first_index_subset_default])
    del(corrs2_cleaned[first_index_subset_default])
print("Índice da posição do nosso 'subset' padrão no 'dataset' de dados brutos:
 ↪")
print(subset_default.index.values[0])
print("-----------------------------------")
print("'Subsets' limpos sem o padrão:")
print(subsets2_cleaned)
```

```
print("-------------------------------------")
print("Coeficientes limpos sem o do padrão:")
print(corrs2_cleaned)
```

```
Índice da posição do nosso 'subset' padrão no 'dataset' de dados brutos:
15203
-------------------------------------
'Subsets' limpos sem o padrão:
[365, 2155, 9737, 15067, 19552, 20036, 22019, 27399, 28118]
-------------------------------------
Coeficientes limpos sem o do padrão:
[0.9530217193164404, 0.9583498564537077, 0.9543742135751833, 0.9526560886216351,
0.9588287028528437, 0.9626586331918893, 0.9542572959099037, 0.950084255505282,
0.9732573257375167]
```

Vamos agora rever os gráficos de linha com os 'subsets' limpos.

[27]:
```
# Mostrando graficamente os 'subsets' limpos com gráficos de linha...

fig = plt.figure(figsize=(16,20))
quad_y = 2
quad_x = int(len(subsets2_cleaned) / quad_y)
if len(subsets2_cleaned) % quad_y:
    quad_x += 1
for i in range(len(subsets2_cleaned)):
    # Extrai do 'dataset' de dados brutos o 'subset' reconhecido.
    subset = df_raw.loc[subsets2_cleaned[i]:subsets2_cleaned[i]+subset_default.
 ↪shape[0]-1]
    ax = fig.add_subplot(quad_x, quad_y, i + 1)
    ax.plot(subset.index.values, subset.CLOSE.values)
    ax.set_title("Subset reconhecido na posição do índice " +␣
 ↪str(subsets2_cleaned[i]) +
                 ". Corr=" + str(round(corrs2_cleaned[i],2)) + ".")
plt.show()
```



Ficou agora mais claro e conciso o nosso primeiro reconhecimento de 'subsets'. Podemos ver de forma mais realista por gráficos de 'candlestick', observando a parte reconhecida numa parte maior pertencente ao 'dataset' de dados brutos.

[28]:
```
# Mostrando graficamente os 'subsets' limpos com gráficos de 'candlestick'...
```

```
for i in range(len(subsets2_cleaned)):
    # Extrai do 'dataset' de dados brutos uma parte maior que o 'subset' reconhecido.
    subset = df_raw.loc[subsets2_cleaned[i]-subset_default.shape[0]:subsets2_cleaned[i]+2*subset_default.shape[0]-1]
    trace = go.Candlestick(x=subset.index,
                           open=subset['OPEN'],
                           high=subset['HIGH'],
                           low=subset['LOW'],
                           close=subset['CLOSE'])

    # Destacando a parte reconhecida.
    layout = {'title': "Subset reconhecido na posição do índice " + str(subsets2_cleaned[i]) +
                       ". Corr=" + str(round(corrs2_cleaned[i],2)) + ".",
                       'yaxis': {'title': 'Pontos'},
                       'xaxis': {'rangeslider': {'visible': False}},
                       'shapes': [{'x0': subsets2_cleaned[i], 'x1':subsets2_cleaned[i]+subset_default.shape[0]-1,
                                   'y0': 0, 'y1': 1, 'xref': 'x', 'yref': 'paper',
                                   'line': {'color': 'rgb(30,30,30)', 'width': 1}}
                                  ],
                       'annotations': [{'x': subsets2_cleaned[i], 'y': 0, 'xref': 'x',
                                        'yref': 'paper', 'showarrow': False, 'xanchor':'left',
                                        'text': "'subset' reconhecido"}
                                      ]
                      }

    data = [trace]

    fig = go.Figure(data=data,layout=layout)
    py.offline.iplot(fig, filename='simple_candlestick_without_range_slider')
```

Subset reconhecido na posição do índice 365. Corr=0.95.

Subset reconhecido na posição do índice 2155. Corr=0.96.

Subset reconhecido na posição do índice 9737. Corr=0.95.

Subset reconhecido na posição do índice 15067. Corr=0.95.

Subset reconhecido na posição do índice 15203. Corr=1.0.

Subset reconhecido na posição do índice 19552. Corr=0.96.

Subset reconhecido na posição do índice 20036. Corr=0.96.

Subset reconhecido na posição do índice 22019. Corr=0.95.

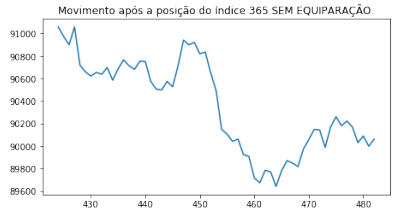
Observando os gráficos de 'candlestick', encontramos uma forma de ajudar o operador de bolsa de valores na sua decisão de comprar lotes desse papel. Supondo que nosso 'subset' padrão foi coletado em tempo real, sendo os últimos 'candles' gerados pela B3 para esse papel, podemos calcular a probabilidade de movimentação da pontuação. Para isso, observamos os gráficos de 'candlesticks' anteriores, onde em dois deles, após o padrão reconhecido, a pontuação baixou, em um deles se manteve estável e nos seis restante subiu. Se a intenção do operador é fazer uma compra de lotes desse papel, então ele terá neste momento uma probabilidade de aproximadamente 68% de sucesso esperando que a pontuação suba: baixa=2, estável=1, alta=6, 6/(2+1+6)=0.6667 .

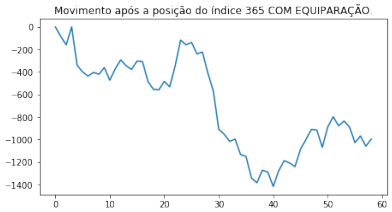
1.5 4. Além do Reconhecimento de Padrão: Prenvendo o Movimento

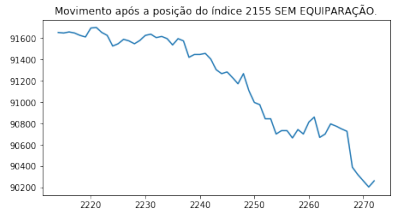
Queremos agora uma visualização de sobreposição dos gráficos de linha após os 'subsets' reconhecidos como forma de visualizar a concentração de movimentação da pontuação. Contudo, será melhor visualizado se equiparados pelos seus pontos iniciais. Portanto, vamos primeiro transformar esses pontos, posicionando o primeiro deles na origem removendo a constante incorporada à 'feature' 'CLOSE' subtraindo todos os pontos pelo valor do primeiro ponto. Isso será feito pela função abaixo, depois essas transformações serão visualizadas nos gráficos individuais. Sobre a quantidade de 'candles' após os 'subsets' reconhecidos, vamos considerar a mesma quantidade do padrão.

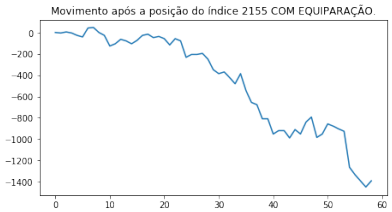
[29]:
```
# Função para equiparar todos os valores de uma lista a partir do primeiro␣
↪elemento.

def equalize_by_first_point(List):
    """Função para equiparar todos os valores de uma lista a partir do primeiro␣
↪elemento."""
    return [x - List[0] for x in List]
```

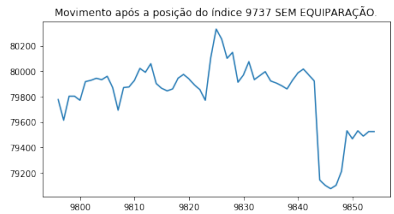
[31]:
```
# Mostrando graficamente a movimentação da pontuação posterior aos 'subsets'␣
↪limpos com gráficos de linha...

fig = plt.figure(figsize=(16,40))
quad_y = 2
quad_x = int(len(2*subsets2_cleaned) / quad_y)
size_quad = 0.
for i in range(len(subsets2_cleaned)):
    # Extrai do 'dataset' de dados brutos os 'candles' após os 'subset'␣
↪reconhecido.
    subset = df_raw.loc[subsets2_cleaned[i]+subset_default.shape[0]:
↪subsets2_cleaned[i]+2*subset_default.shape[0]-1]
    # Plota os pontos antes da equiparação.
    ax = fig.add_subplot(quad_x, quad_y, size_quad + 1)
    ax.plot(subset.index.values, subset.CLOSE.values)
    ax.set_title("Movimento após a posição do índice " +
                 str(subsets2_cleaned[i]) + " SEM EQUIPARAÇÃO.")
    size_quad += 1
    # Plota os pontos depois da equiparação.
    ax = fig.add_subplot(quad_x, quad_y, size_quad + 1)
    ax.plot(range(subset.shape[0]), equalize_by_first_point(subset.CLOSE.
↪values))
    ax.set_title("Movimento após a posição do índice " +
                 str(subsets2_cleaned[i]) + " COM EQUIPARAÇÃO.")
    size_quad += 1
plt.show()
```

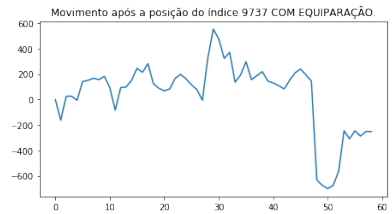

Movimento após a posição do índice 365 SEM EQUIPARAÇÃO.

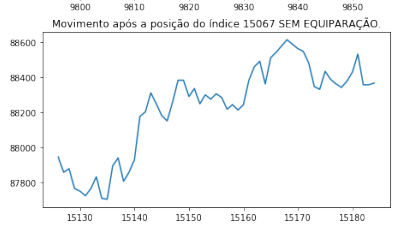

Movimento após a posição do índice 365 COM EQUIPARAÇÃO.

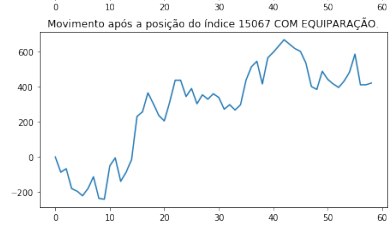

Movimento após a posição do índice 2155 SEM EQUIPARAÇÃO.

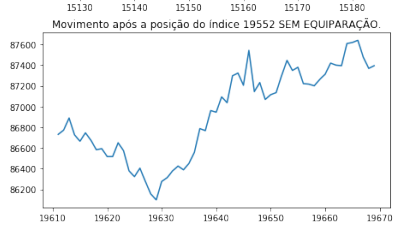

Movimento após a posição do índice 2155 COM EQUIPARAÇÃO.

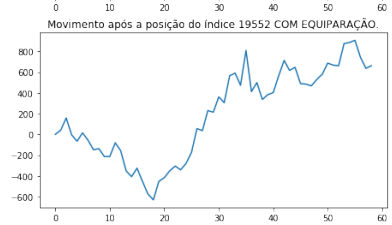

Movimento após a posição do índice 9737 SEM EQUIPARAÇÃO.


Movimento após a posição do índice 9737 COM EQUIPARAÇÃO.


Movimento após a posição do índice 15067 SEM EQUIPARAÇÃO.


Movimento após a posição do índice 15067 COM EQUIPARAÇÃO.


Movimento após a posição do índice 19552 SEM EQUIPARAÇÃO.


Movimento após a posição do índice 19552 COM EQUIPARAÇÃO.


Movimento após a posição do índice 20036 SEM EQUIPARAÇÃO.


Movimento após a posição do índice 20036 COM EQUIPARAÇÃO.


Movimento após a posição do índice 22019 SEM EQUIPARAÇÃO.


Movimento após a posição do índice 22019 COM EQUIPARAÇÃO.


Movimento após a posição do índice 27399 SEM EQUIPARAÇÃO.

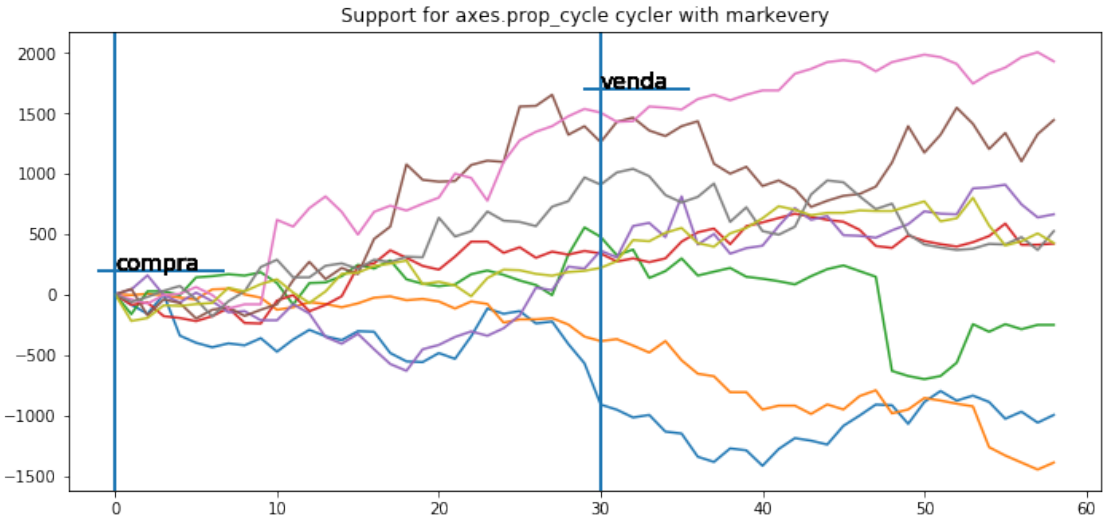

Movimento após a posição do índice 27399 COM EQUIPARAÇÃO.


Movimento após a posição do índice 28118 SEM EQUIPARAÇÃO.


Movimento após a posição do índice 28118 COM EQUIPARAÇÃO.

Agora vamos sobrepor os gráficos com equiparação.

[32]:
```
# Sobreposição dos gráficos de linha equiparados dos movimentos posteriores dos
# 'subsets' reconhecidos com todos os pontos iniciais começando da origem.

fig = plt.figure()
ax = fig.add_axes([0.0, 0.0, 1.5, 1.0])
for i in range(len(subsets2_cleaned)):
    # Extrai do 'dataset' de dados brutos os 'candles' após os 'subset'␣
 ↪reconhecido.
    subset = df_raw.loc[subsets2_cleaned[i]+subset_default.shape[0]:
 ↪subsets2_cleaned[i]+2*subset_default.shape[0]-1]
    # Plota os pontos considerando que todos os gráficos comecem da origem.
    ax.plot(range(subset_default.shape[0]), equalize_by_first_point(subset.
 ↪CLOSE.values))
    ax.axvline(x=0, ymin=0, ymax=1)
    ax.axhline(xmin=0.03, y=200, xmax=0.15)
    ax.axvline(x=30, ymin=0, ymax=1)
    ax.axhline(xmin=0.5, y=1700, xmax=0.6)
    ax.text(x=0, y=200, s="compra", fontsize=14)
    ax.text(x=30, y=1700, s="venda", fontsize=14)
plt.title('Support for axes.prop_cycle cycler with markevery')
plt.show()
```



O gráfico acima foi construído com a mesma quantidade de 'candles' dos 'subsets' padrão e reconhecidos, porém, contado a partir do último 'candle' de cada 'subset', mostrando o que viria depois. Percebemos nesse gráfico que sete dos nove 'subsets' reconhecidos teve, em seu histórico, continuidade de movimentação da pontuação de estável para cima até na metade da quantidade de 'candles' considerados. A decisão de compra do operador no início faria ele decidir também por vender após metade da quantidade de 'candles' do padrão.

Podemos, então, gerar um único gráfico com seus pontos sendo uma combinação probabilística dos gráficos dos movimentos posteriores equiparados para melhorar a visualização para tomada de decisão do operador. A probabilidade será baseada em dois resultados: tendência de alta e de baixa. Faremos o cálculo probabilístico seguindo os seguintes passos: 1. Criaremos um parâmetro de referência que determina uma quantidade de pontos (variação de preço ou pontuação neste caso) a partir do qual consideraremos se um ponto é de movimento de baixa ou de alta. 2. Criaremos um parâmetro, retornado de uma função externa, para decaimento das probabilidades calculadas que os reduz com o avanço no eixo horizonta, representando na prática um tipo de decaimento com o passar do tempo. 3. Formaremos uma matriz 2D com todos os pontos dos gráficos, onde cada linha é uma lista de todos os pontos de mesmo índice das listas de pontos de cada gráfico. 4. Em cada linha dessa matriz, determinaremos quais pontos são de baixa (menor que o parâmetro de referência) ou de alta (maior ou igual ao parâmetro de referência), gerando nova lista com cada elemento sendo um par de quantidade de pontos de baixa e de alta, nesta ordem. 5. Na lista gerada de quantidade de baixas e de altas, calculamos a probabilidade para alta e para baixa, considerando o parâmetro de decaimento, gerando uma nova lista com esses valores na mesma ordem. 6. E para concluir, a partir da lista de probabilidades de baixa e de alta, criaremos a lista final, deixando em cada elemento apenas a probabilidade de maior valor, sendo que aqueles que forem do tipo 'alta' serão positivos e do tipo 'baixa' serão negativos.

Não precisamos, necessariamente, criar nossa lista final separando os passos acima, podendo alguns deles serem feitos no mesmo momento, pois, o que realmente importa é o resultado final. Abaixo, segue o código para uma função que retorna essa lista e, posteriormente, a visualização do gráfico de probabilidades de predição do movimento do preço ou pontuação.

[148]:
```
# Função que retorna uma lista de probabilidades de predição do movimento de␣
↪preço ou pontuação.

def prob_predict_price_punctuation_movement(dataset, ind_subsets, size_subsets,␣
↪size_predict, col_dataset='CLOSE',
                                            ref_equalize=0.0, func_decay=None,␣
↪printout=False):
    """
    Retorna uma lista de probabilidades de predição do movimento de preço ou␣
↪pontuação,
    sendo positivos os de 'alta' ou iguais à referência e negativos os de␣
↪'baixa'.

    VARIÁVEIS DE ENTRADA:

    - dataset: tipo 'pandas.DataFrame'
        É um 'dataframe', representando um conjunto de 'candles', onde será␣
↪referenciada
```

```
        as posições que contenham subconjuntos semelhantes ao padrão com seus␣
↪primeiros
        'candles' representados na lista 'subsets'.

    - ind_subsets: tipos 'list' ou 'numpy.ndarray'
        É uma lista contendo todos os valores de índices no 'dataset' que␣
↪representam o
        primeiro registro de cada 'subset'.

    - size_subsets: tipo 'int'
        Parâmetro que representa o tamanho dos 'subsets', ou seja, a quantidade␣
↪de
        'candles' que cada 'subset' tem.

    - size_predict: tipo 'int'
        Parâmetro que representa o tamanho da lista do resultado desta função,␣
↪ou seja, a
        quantidade de 'candles' considerados para predição a partir do último␣
↪de cada
        'subset'.

    - col_dataset: tipos 'str' ou 'int'
        É o nome ou número da coluna do 'dataset' para ser usada no cálculo do␣
↪movimento
        de preço ou pontuação.

    - ref_equalize: tipos 'float' ou 'int'
        Parâmetro de referência que determina uma quantidade de pontos, já␣
↪equiparados a
        partir do qual consideraremos se um ponto é de movimento de baixa ou de␣
↪alta. A
        equiparação consiste em posicionar o primeiro ponto de cada 'subset' na␣
↪origem
        removendo a constante incorporada à coluna em 'col_dataset' subtraindo␣
↪todos os
        pontos pelo valor do primeiro ponto.

    - func_decay: tipos 'None' ou 'function'
        Função para aplicação de um parâmetro de decaimento nas probabilidades␣
↪calculadas
        que os reduz na medida que se avança no eixo horizontal, representando␣
↪na prática
        um tipo de decaimento com o passar do tempo.

        Essa função deverá ter apenas uma variável de entrada obrigatória e␣
↪será executada,
```

```
        internamente, quantas vezes forem necessárias, sendo a mesma quantidade␣
↪de
        elementos em 'size_predict', pois as variáveis de entrada para essa␣
↪função, uma
        para cada execução, serão exatamente a sequência de números obtido pela␣
↪linha de
        código 'range(0,size_predict)'.

        Para cada execução interna dessa função, o decaimento é calculado com␣
↪as
        seguintes linhas de códigos:

        - 'prob_high-func_decay(i)': probabilidade de 'alta' com valor positivo.
        - 'prob_high+func_decay(i)-1': probabilidade de 'baixa' com valor␣
↪negativo.

        Onde, 'prob_high' é a probabilidade do ponto atual de movimento de␣
↪'alta' e
        'i' é o índice do ponto atual gerado pela iteração na linha de código
        'range(0,size_predict)'.

        Caso 'func_decay' seja igual a 'None', o decaimento não é aplicado.␣
↪Isso é feito
        zerando o resultado da função com a linha de código '(i)*0.0'.

    - printout: tipo 'bool'
        É um ativador que serve para imprimir ou não na saída indicadores␣
↪internos.

    RESULTADO:

    - return: tipo 'list'
        É uma lista contendo as probabilidade de maiores valores, entre␣
↪movimentos de
        'baixa' e de 'alta', para cada ponto de mesmo índice a partir da origem␣
↪para os
        pontos posteriores dos 'subsets', sendo que aqueles que forem do tipo␣
↪'alta'
        serão positivos e do tipo 'baixa' serão negativos. Essa lista contém as␣
↪predições
        probabilística para o 'subset' padrão que gerou os 'subsets'␣
↪reconhecidos.
    """

    ## Tratamento de erros das variáveis de entrada.
```

```
    import pandas
    import numpy

    # Verificação de tipos.
    if type(dataset) != pandas.DataFrame:
        raise ValueError("'dataset' deve ser do tipo 'pandas.DataFrame'.")
    if not(type(ind_subsets) == list or type(ind_subsets) == numpy.ndarray):
        raise ValueError("'ind_subsets' deve ser do tipo 'list' ou 'numpy.
↪ndarray'.")
    if type(size_subsets) != int:
        raise ValueError("'size_subsets' deve ser do tipo 'int'.")
    if type(size_predict) != int:
        raise ValueError("'size_predict' deve ser do tipo 'int'.")
    if not(type(col_dataset) == str or type(col_dataset) == int):
        raise ValueError("'col_dataset' deve ser do tipo 'str' ou 'int'.")
    if not(type(ref_equalize) == int or type(ref_equalize) == float):
        raise ValueError("'ref_equalize' deve ser do tipo 'int' ou 'float'.")
    if not(func_decay == None or str(type(func_decay)) == "<class 'function'>"):
        raise ValueError("'func_decay' deve ser do tipo 'None' ou 'function'.")

    # Verificação de conteúdo.
    if type(col_dataset) == str:
        if not(col_dataset in dataset.columns.values):
            raise ValueError("Não há nenhuma coluna no 'dataset' com o nome '"␣
↪+ col_dataset + "'.")
        else:
            try:
                float(dataset[col_dataset].values[0])
            except:
                raise ValueError("Os dados da coluna " + col_dataset + " do␣
↪'dataset' devem ser numéricos.")
    else:
        try:
            float(dataset[dataset.columns[col_dataset]].values[0])
        except:
            raise ValueError("Os dados da coluna " + col_dataset + " do␣
↪'dataset' devem ser numéricos.")
        if col_dataset < 0:
            raise ValueError("'col_dataset' deve ser maior ou igual a zero.")
        if col_dataset >= len(dataset.columns.values):
            raise ValueError("Não há nenhuma coluna no 'dataset' com o índice␣
↪'" + str(col_dataset) + "'.")

    ## Corpo principal da função.

    # Pegando o índice da coluna caso a variável de entrada 'col_dataset' seja␣
↪'strings'.
```

```
    if type(col_dataset) == str:
        col_dataset = list(dataset.columns.values).index(col_dataset)

    # Carregamento das bibliotecas necessárias para apoio.
    from time import time
    from tqdm import tqdm

    # Predição do movimento de preço ou pontuação.
    if func_decay == None:
        func_decay = (lambda x: x * 0.0)
    lists_indexes = []
    tm = time()
    if printout:
        print("Extraindo os pontos posteriores de cada 'subset':")
    for ind_sub in tqdm(ind_subsets):
        # Extrai do 'dataset' de dados brutos os 'candles' após os 'subset'␣
↪reconhecido.
        subset = dataset.loc[ind_sub+size_subsets:
↪ind_sub+size_subsets+size_predict-1]
        # Aplica a equiparação no conjunto de valores da coluna escolhida␣
↪'col_dataset'.
        lists_indexes.append(equalize_by_first_point(subset[subset.
↪columns[col_dataset]].values))
    lists_indexes = [*zip(*lists_indexes)] # transposta da matriz 2D.
    lists_probs = []
    if printout:
        print("Calculando a probabilidade para cada ponto extraído de cada␣
↪'subset':",
              numpy.array(lists_indexes).shape)
    for i, list_element in  tqdm(enumerate(lists_indexes)):
        count_low = 0
        count_high = 0
        for element in list_element:
            if element >= ref_equalize:
                count_high += 1
            else:
                count_low += 1
        prob_high = count_high/len(ind_subsets)
        # Probabilidades de 'alta' e 'baixa' com decaimento.
        prob_current = 0.0
        try:
            if count_high >= count_low:
                prob_current = prob_high-func_decay(i) # probabilidade de alta␣
↪com valor positivo.
                if prob_current < 0.0:
                    prob_current = 0.0
```

```
            else:
                prob_current = prob_high+func_decay(i)-1 # probabilidade de␣
 ↪baixa com valor negativo.
                if prob_current > 0.0:
                    prob_current = 0.0
        except Exception as e:
            raise ValueError("Verifique se 'func_decay' é uma função de apenas␣
 ↪uma variável " +
                             "de entrada obrigatória e que seja a primeira. Em␣
 ↪caso afirmativo, " +
                             "verifique o erro inerente ao código interno da␣
 ↪função. Erro: " + str(e))
        lists_probs.append(prob_current)
    lists_probs[0] = 0.0 # a lista deve começar sempre com o primeiro elemento␣
 ↪zerado.
    if printout:
        print('- Processo executado em', round(time()-tm,2), '\b(s).')
        print('- A lista de probabilidades contém', len(lists_probs),
              'elementos referentes aos valores da coluna ', "'"+dataset.
 ↪columns[col_dataset]+"'",
              'do \'dataset\' com valor de referência equiparado em \'' +␣
 ↪str(ref_equalize) + '\' .')

    return lists_probs
```

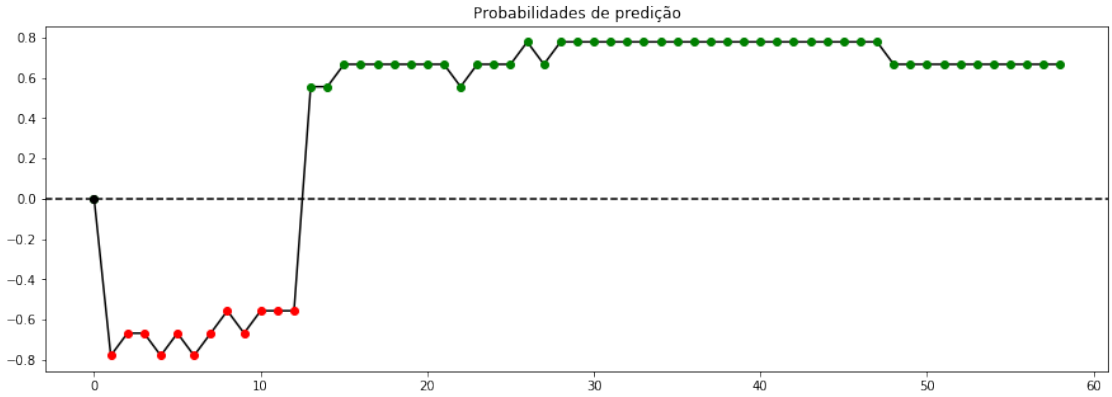
[230]:
```
# Aplicação da função anterior.

probs_predicts1 = prob_predict_price_punctuation_movement(df_raw,␣
 ↪subsets2_cleaned,
                                                          len(subset_default),
                                                          len(subset_default),
                                                          printout=True)
print(np.around(np.array(probs_predicts),2))
```

```
Extraindo os pontos posteriores de cada 'subset':

100%|
| 9/9 [00:00<00:00, 3007.63it/s]

Calculando a probabilidade para cada ponto extraído de cada 'subset': (59, 9)

59it [00:00, 29597.41it/s]

- Processo executado em 0.01(s).
- A lista de probabilidades contém 59 elementos referentes aos valores da coluna
'CLOSE' do 'dataset' com valor de referência equiparado em '0.0' .
[ 0.   -0.78 -0.67 -0.67 -0.78 -0.67 -0.78 -0.67 -0.56 -0.67 -0.56 -0.56
 -0.56  0.56  0.56  0.67  0.67  0.67  0.67  0.67  0.67  0.67  0.56  0.67
```

```
0.67  0.67  0.78  0.67  0.78  0.78  0.78  0.78  0.78  0.78  0.78  0.78
0.78  0.78  0.78  0.78  0.78  0.78  0.78  0.78  0.78  0.78  0.78  0.78
0.67  0.67  0.67  0.67  0.67  0.67  0.67  0.67  0.67  0.67  0.67]
```

Agora vamos plotar um gráfico personalizado das probabilidades de predição.

[235]:
```
# Gráfico personalizado das probabilidades de predição...

probs_predicts1_np = np.array(probs_predicts1)
slower = np.ma.masked_where(probs_predicts1_np >= 0.0, probs_predicts1_np)
supper = np.ma.masked_where(probs_predicts1_np < 0.0, probs_predicts1_np)
zeros = np.ma.masked_where(probs_predicts1_np != 0.0, probs_predicts1_np)
fig = plt.figure(figsize=(15,5))
plt.axhline(y=0, xmin=0, xmax=1, color='k', linestyle='--')
plt.plot(range(subset_default.shape[0]), probs_predicts1, 'k',
         range(subset_default.shape[0]), slower, 'ro',
         range(subset_default.shape[0]), supper, 'go',
         range(subset_default.shape[0]), zeros, 'ko')
plt.title("Probabilidades de predição")
plt.show()
```



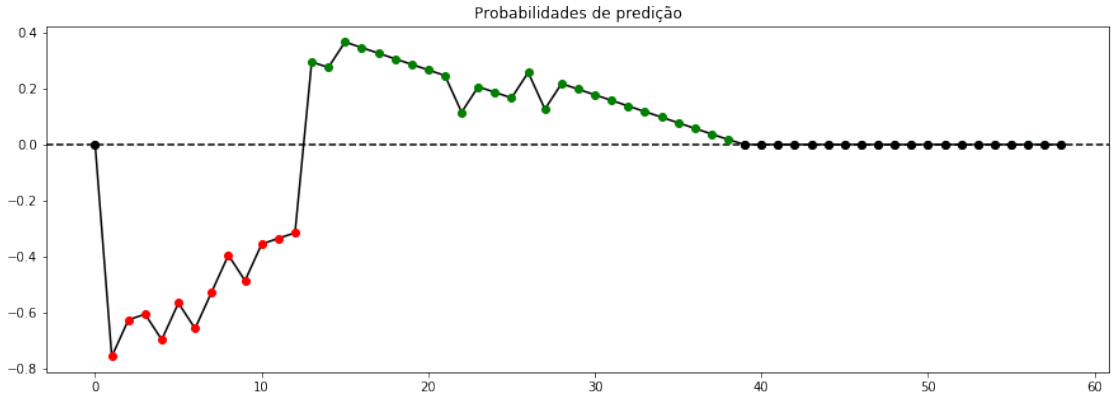
Vamos agora aplicar a função para incluir uma função externa de decaimento linear. Aqui é importante ressaltar que **podemos incluir na função externa de decaimento** um novo parâmetro que chamaremos de **parâmetro de decaimento** para definir a curva de probabilidades ao longo do eixo horizontal.

[250]:
```
# Aplicação da função anterior com função externa de decaimento linear.

par_decay_linear2 = 0.02
probs_predicts2 = prob_predict_price_punctuation_movement(df_raw,␣
 ↪subsets2_cleaned,
```

```
                                                        len(subset_default),
                                                        len(subset_default),
                                                        func_decay=(lambda x:␣
 ↪x * par_decay_linear2),
                                                        printout=True)
print(np.around(np.array(probs_predicts2),2))
```

```
Extraindo os pontos posteriores de cada 'subset':

100%|
| 9/9 [00:00<00:00, 3008.11it/s]

Calculando a probabilidade para cada ponto extraído de cada 'subset': (59, 9)

59it [00:00, 29572.65it/s]

- Processo executado em 0.02(s).
- A lista de probabilidades contém 59 elementos referentes aos valores da coluna
'CLOSE' do 'dataset' com valor de referência equiparado em '0.0' .
[ 0.   -0.76 -0.63 -0.61 -0.7  -0.57 -0.66 -0.53 -0.4  -0.49 -0.36 -0.34
 -0.32  0.3   0.28  0.37  0.35  0.33  0.31  0.29  0.27  0.25  0.12  0.21
  0.19  0.17  0.26  0.13  0.22  0.2   0.18  0.16  0.14  0.12  0.1   0.08
  0.06  0.04  0.02  0.    0.    0.    0.    0.    0.    0.    0.    0.
  0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.  ]
```

[251]:
```
# Gráfico personalizado das probabilidades de predição com função externa de␣
 ↪decaimento linear.

probs_predicts2_np = np.array(probs_predicts2)
slower = np.ma.masked_where(probs_predicts2_np >= 0.0, probs_predicts2_np)
supper = np.ma.masked_where(probs_predicts2_np < 0.0, probs_predicts2_np)
zeros = np.ma.masked_where(probs_predicts2_np != 0.0, probs_predicts2_np)
fig = plt.figure(figsize=(15,5))
plt.axhline(y=0, xmin=0, xmax=1, color='k', linestyle='--')
plt.plot(range(subset_default.shape[0]), probs_predicts2, 'k',
         range(subset_default.shape[0]), slower, 'ro',
         range(subset_default.shape[0]), supper, 'go',
         range(subset_default.shape[0]), zeros, 'ko')
plt.title("Probabilidades de predição")
plt.show()
```

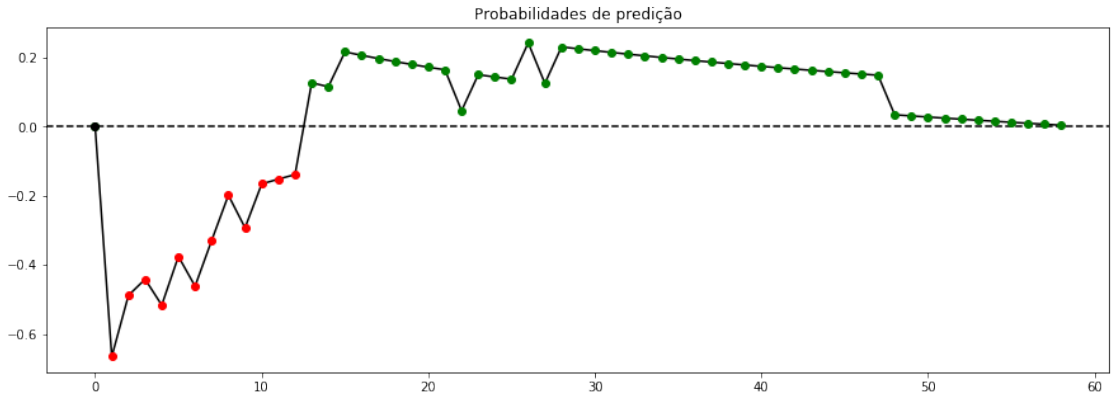


E, neste terceiro exemplo, aplicaremos uma função externa de decaimento por logarítmo natural.

[252]:
```
# Aplicação da função anterior com função externa de decaimento logaritmo.

par_decay_log3 = 0.1625
probs_predicts3 = prob_predict_price_punctuation_movement(df_raw,␣
 ↪subsets2_cleaned,
                                                          len(subset_default),
                                                          len(subset_default),
                                                          func_decay=(lambda x:␣
 ↪np.log1p(x) * par_decay_log3),
                                                          printout=True)
print(np.around(np.array(probs_predicts3),2))
```

```
Extraindo os pontos posteriores de cada 'subset':

100%|
| 9/9 [00:00<00:00, 3008.83it/s]

Calculando a probabilidade para cada ponto extraído de cada 'subset': (59, 9)

59it [00:00, 29593.87it/s]

- Processo executado em 0.01(s).
- A lista de probabilidades contém 59 elementos referentes aos valores da coluna
'CLOSE' do 'dataset' com valor de referência equiparado em '0.0' .
[ 0.   -0.67 -0.49 -0.44 -0.52 -0.38 -0.46 -0.33 -0.2  -0.29 -0.17 -0.15
 -0.14  0.13  0.12  0.22  0.21  0.2   0.19  0.18  0.17  0.16  0.05  0.15
  0.14  0.14  0.24  0.13  0.23  0.23  0.22  0.21  0.21  0.2   0.2   0.2
  0.19  0.19  0.18  0.18  0.17  0.17  0.17  0.16  0.16  0.16  0.15  0.15
  0.03  0.03  0.03  0.02  0.02  0.02  0.02  0.01  0.01  0.01  0.  ]
```

[253]:
```
# Gráfico personalizado das probabilidades de predição com função externa de␣
 ↪decaimento logaritmo.

probs_predicts3_np = np.array(probs_predicts3)
slower = np.ma.masked_where(probs_predicts3_np >= 0.0, probs_predicts3_np)
supper = np.ma.masked_where(probs_predicts3_np < 0.0, probs_predicts3_np)
zeros = np.ma.masked_where(probs_predicts3_np != 0.0, probs_predicts3_np)
fig = plt.figure(figsize=(15,5))
plt.axhline(y=0, xmin=0, xmax=1, color='k', linestyle='--')
plt.plot(range(subset_default.shape[0]), probs_predicts3, 'k',
         range(subset_default.shape[0]), slower, 'ro',
         range(subset_default.shape[0]), supper, 'go',
         range(subset_default.shape[0]), zeros, 'ko')
plt.title("Probabilidades de predição")
plt.show()
```



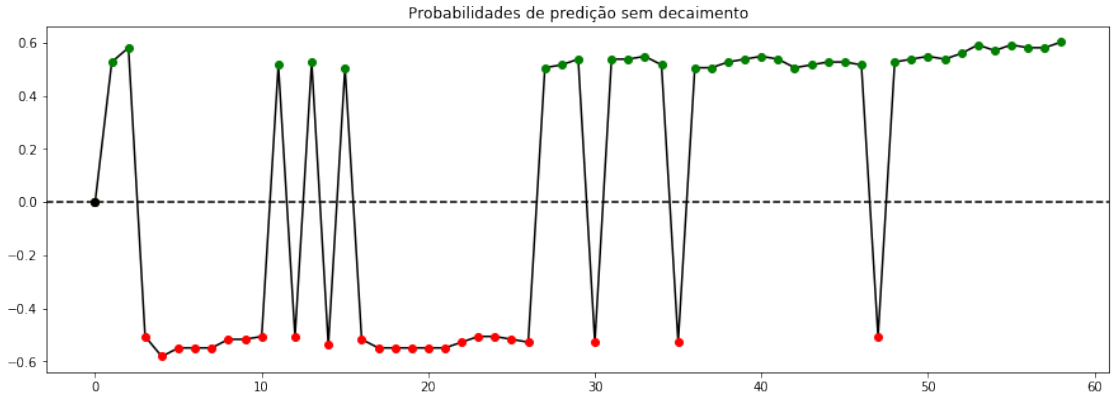
Com todo o trabalho de predição concluído, podemos aplicar para uma quantidade bem maior de 'subsets' reconhecidos, como é o caso das listas já atribuídas a 'subsets1' e 'corrs1' calculadas para coeficiente de correlação mínimo de '0.9'. Usaremos elas para repetir todos os passos seguintes a atribuição delas e visualizaremos o resultado.

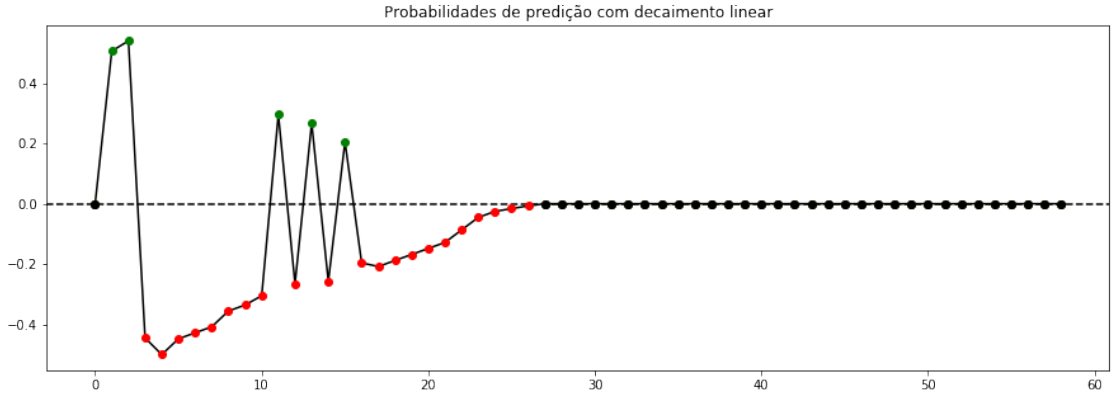
[254]:
```
## Xxx...

# Limpeza
subsets1_cleaned, _ = subsets_cleaner(subsets1, corrs1)

# Remoção das listas nosso 'subset' padrão.
if subset_default.index.values[0] in subsets1_cleaned:
    first_index_subset_default = subsets1_cleaned.index(subset_default.index.
 ↪values[0])
    del(subsets1_cleaned[first_index_subset_default])
```

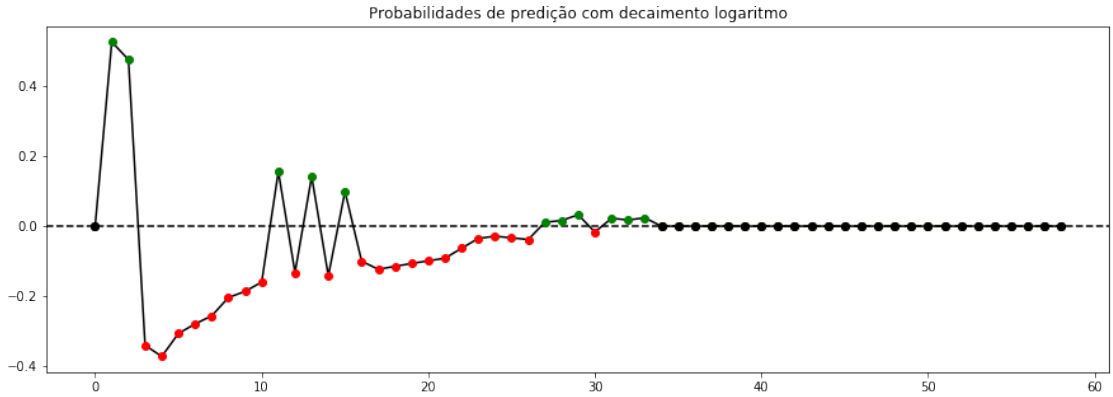
```
# Aplicação da função de predição com função externa de decaimento logaritmo.
probs_predicts4_raw = prob_predict_price_punctuation_movement(df_raw,␣
 ↪subsets1_cleaned,
                                                              len(subset_default),
                                                              len(subset_default))
par_decay_linear4 = 0.02
probs_predicts4_linear = prob_predict_price_punctuation_movement(df_raw,␣
 ↪subsets1_cleaned,
                                                              len(subset_default),
                                                              len(subset_default),
                                                              func_decay=(lambda x:␣
 ↪x * par_decay_linear4))
par_decay_log4 = 0.15
probs_predicts4_log = prob_predict_price_punctuation_movement(df_raw,␣
 ↪subsets1_cleaned,
                                                              len(subset_default),
                                                              len(subset_default),
                                                              func_decay=(lambda x:
                                                                          np.log(x)␣
 ↪* par_decay_log4
                                                                          if x > 0.0␣
 ↪else 0.0))

## Plotagem dos gráfico personalizado linear e logaritmo das probabilidades de␣
 ↪predição...
# Sem decaimento.
probs_predicts4_raw_np = np.array(probs_predicts4_raw)
slower = np.ma.masked_where(probs_predicts4_raw_np > 0.0,␣
 ↪probs_predicts4_raw_np)
supper = np.ma.masked_where(probs_predicts4_raw_np < 0.0,␣
 ↪probs_predicts4_raw_np)
zeros = np.ma.masked_where(probs_predicts4_raw_np != 0.0,␣
 ↪probs_predicts4_raw_np)
fig = plt.figure(figsize=(15,5))
plt.axhline(y=0, xmin=0, xmax=1, color='k', linestyle='--')
plt.plot(range(subset_default.shape[0]), probs_predicts4_raw, 'k',
         range(subset_default.shape[0]), slower, 'ro',
         range(subset_default.shape[0]), supper, 'go',
         range(subset_default.shape[0]), zeros, 'ko')
plt.title("Probabilidades de predição sem decaimento")
plt.show()
# Com decaimento linear.
probs_predicts4_linear_np = np.array(probs_predicts4_linear)
slower = np.ma.masked_where(probs_predicts4_linear_np > 0.0,␣
 ↪probs_predicts4_linear_np)
```

```
supper = np.ma.masked_where(probs_predicts4_linear_np < 0.0,␣
 ↪probs_predicts4_linear_np)
zeros = np.ma.masked_where(probs_predicts4_linear_np != 0.0,␣
 ↪probs_predicts4_linear_np)
fig = plt.figure(figsize=(15,5))
plt.axhline(y=0, xmin=0, xmax=1, color='k', linestyle='--')
plt.plot(range(subset_default.shape[0]), probs_predicts4_linear, 'k',
         range(subset_default.shape[0]), slower, 'ro',
         range(subset_default.shape[0]), supper, 'go',
         range(subset_default.shape[0]), zeros, 'ko')
plt.title("Probabilidades de predição com decaimento linear")
plt.show()
# Com decaimento logaritmo.
probs_predicts4_log_np = np.array(probs_predicts4_log)
slower = np.ma.masked_where(probs_predicts4_log_np > 0.0,␣
 ↪probs_predicts4_log_np)
supper = np.ma.masked_where(probs_predicts4_log_np < 0.0,␣
 ↪probs_predicts4_log_np)
zeros = np.ma.masked_where(probs_predicts4_log_np != 0.0,␣
 ↪probs_predicts4_log_np)
fig = plt.figure(figsize=(15,5))
plt.axhline(y=0, xmin=0, xmax=1, color='k', linestyle='--')
plt.plot(range(subset_default.shape[0]), probs_predicts4_log, 'k',
         range(subset_default.shape[0]), slower, 'ro',
         range(subset_default.shape[0]), supper, 'go',
         range(subset_default.shape[0]), zeros, 'ko')
plt.title("Probabilidades de predição com decaimento logaritmo")
plt.show()
```

```
100%|
| 93/93 [00:00<00:00, 5828.82it/s]
59it [00:00, 6573.79it/s]
100%|
| 93/93 [00:00<00:00, 5827.86it/s]
59it [00:00, 6573.10it/s]
100%|
| 93/93 [00:00<00:00, 5828.29it/s]
59it [00:00, 6573.62it/s]
```







A flexibilidade no uso de uma função externa para decaimento das probabilidades pode facilitar para o operador ou o robô financeiro alterar essa função conforme o comportamento do mercado na bolsa de valores. Assim, entendemos que nos tempos mais próximos o operador tem maior oportunidade para entrar com uma negociação de acordo com as probabilidades presentes,

tendo uma ideia de quando, num futuro próximo, ele deve finalizar a negociação devido a baixa probabilidade de continuação do movimento esperado que foi predita.

1.6 CONCLUSÃO

Há muitas formas de otimizar a predição da movimentação do preço ou pontuação. Podemos usar, por exemplo, a regressão linear e a própria correlação aplicada à 'features' diferentes além da 'CLOSE' com 'CLOSE' ('HIGH' com 'LOW, 'OPEN' com 'CLOSE', 'FINVOL' com 'FINVOL', 'TICKVOL' com 'TICKVOL', etc). Por isso, este trabalho significa um ponto de partida para aplicação de novas ferramentas para predição de movimentação de preços ou pontuações de papéis das bolsas e valores.

Este trabalho demandou muitas ferramentas de plotagem e conhecimentos sobre o mercado financeiro de bolsas de valores. Para próximos trabalhos, as funções definidas aqui podem ser utilizadas para um sistema de tempo real, coletando num 'loop' um 'subset' padrão de outro sistema que disponibiliza as últimas cotações de qualquer papel (últimos 'candles'). Outros trabalhos podem ser realizados a partir deste com uso de ferramentas de 'Machine Learning' para otimização dos resultados.