

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data

Rafael Rocha da Silva Proença

Ciência de dados aplicada à Bolsa de Valores:
Um estudo focado no curto prazo

Belo Horizonte
2023

Rafael Rocha da Silva Proença

**Ciência de dados aplicada à Bolsa de Valores:
Um estudo focado no curto prazo**

Trabalho de Conclusão de Curso apresentado
ao Curso de Especialização em Ciência de
Dados e Big Data como requisito parcial à
obtenção do título de especialista.

Belo Horizonte

2023

SUMÁRIO

1. Introdução.....	4
1.1. Contextualização.....	4
1.2. O problema proposto.....	5
1.3. Objetivos.....	6
2. Coleta de Dados.....	7
3. Processamento/Tratamento de Dados.....	11
4. Análise e Exploração dos Dados.....	16
4.1 Análises e Indicadores Preliminares.....	17
4.2 Adicionando indicadores básicos: clusterização de ações.....	19
4.3 Criando indicadores de análise técnica: Conceitos.....	25
4.4 Criando indicadores de análise técnica: Aplicação.....	29
4.5 Criando indicadores de análise técnica: Tratamento de dados.....	33
5. Criação de Modelos de Machine Learning.....	35
5.1 Introdução.....	35
5.2 Seleção de features.....	38
5.3 Construção de modelos.....	43
6. Interpretação dos Resultados.....	48
6.1 Introdução.....	48
6.2 Apresentação de resultados.....	49
7. Apresentação dos Resultados.....	53
8. Links.....	55
REFERÊNCIAS.....	56
APÊNDICE.....	58

1. Introdução

1.1. Contextualização

Há de se concordar que muitos brasileiros possuem uma visão deturpada a respeito da bolsa de valores. Crises econômicas, pandemias, guerras e mudanças radicais de governo contribuem para a flutuação dos preços de diversas ações em curto período de tempo. Para uma população majoritariamente pobre, que observa essas flutuações, e é bombardeada sobretudo em suas redes sociais pelas opiniões de analistas e “coachs” financeiros duvidosos, fica disseminada a ideia de que a bolsa de valores seria uma espécie de cassino, jogo de adivinhação, ou pirâmide legalizada.

Nada poderia estar mais errado do que isso. Analistas e “coachs financeiros” de redes sociais, via de regra, estão muito mais interessados em vender seus próprios cursos do que em promover, verdadeiramente, uma educação financeira ou estratégias de investimento de qualidade. Não são portanto uma boa base para estabelecer impressões sobre a bolsa.

Por outro lado, é bem verdade que muitos conceituados especialistas advogam, e com razão de causa, que a maioria das pessoas devem ser priorizados investimentos de longo prazo no sentido de catalisar ganhos e formar uma reserva sólida (estratégia *buy and hold*). Entretanto muitos investidores, de todos os extratos sociais, procuram a bolsa para obter ganhos em um curto prazo, afinal nem todos os planos da vida são planos de longo prazo. Vale ressaltar que, mesmo para investimentos no curto prazo, apesar dos riscos mais elevados, existem estratégias de investimento que podem propiciar ganhos interessantes.

Com base nesse segundo cenário, o presente trabalho visa então o desenvolvimento de um algoritmo que ajude a destrinchar o mundo dos investimentos de curto prazo. Por meio da linguagem *python*, pretende-se analisar uma série de ações indexadas no índice ibovespa, selecioná-las com relação ao seu potencial de ganho e riscos no sentido de formar uma potencial carteira e, por meio da construção de indicadores técnicos e elaboração de scripts de machine learning, construir modelos de previsão de preços de fechamento para o dia seguinte, gerando mais informação para auxiliar na tomada de decisão.

Vale ressaltar que o algoritmo não substitui a análise humana, e nem dá conta de todas as variáveis envolvidas na formação do preço de uma ação, mas pode ser usado como uma ferramenta para auxiliar nas decisões de curto prazo. Espera-se com esse trabalho possa contribuir para disseminar um pouco de educação financeira com foco no curto prazo, bem como promover uma melhor compreensão dos indicadores mais importantes na previsão de preços e tendências de ativos financeiros na bolsa de valores.

1.2. O problema proposto

Serão abordadas nesse trabalho diversas questões referentes a compra e venda de ações, promovendo a ciência de dados e machine learning como ferramentas importantes na tomada de decisão. Nesse sentido, utilizando a técnica dos 5w, podemos definir com clareza nosso problema:

(Why?) O problema é importante à medida que propaga um melhor entendimento sobre a bolsa de valores e investimentos de curto prazo.

(Who?) Os principais beneficiários do trabalho são justamente pequenos investidores e investidores novatos, bem como qualquer um que deseje investir no curto prazo.

(What?): Como principal objetivo da análise temos a construção de uma ferramenta de análise e auxílio à tomada de decisão no curto prazo.

(Where?): Os dados são públicos, de fácil acesso, e obtidos da própria B3.

(When?): Estamos analisando um período de 8 anos de dados da B3.

1.3. Objetivos

O trabalho busca, por meio da ciência de dados, selecionar ações que pertençam ao índice Ibovespa, determinar um potencial de lucratividade no curto prazo, mensurar seus riscos, selecionar uma possível carteira com base nesse potencial de ganhos, e tentar prever, por meio de técnicas de machine learning aplicadas a indicadores financeiros, o preço de fechamento para o dia seguinte das ações desta carteira por meio de 4 algoritmos diferentes de machine learning, que serão comparados ao fim do trabalho.

2. Coleta de Dados

Nesse trabalho foi utilizado como fonte de dados primordial as séries históricas de todas as ações da B3, a bolsa de valores oficial do Brasil. Os dados foram obtidos por meio de download do próprio site da B3¹. Os dados estão contidos em arquivos posicionais, e são baixados dentro de arquivos comprimidos no formato zip.

Cada arquivo *COTAHIST.AAAA.TXT* contém os seguintes classificadores: Tipo de registro, Data do pregão, Código de BDI, Nome da empresa e Código de Negociação. Esses classificadores são aplicados às cotações históricas relativas à negociação de todos os papéis-mercado no período de um ano. Para o presente trabalho foi escolhido o período histórico de 2015 a 2022, portanto 8 arquivos contendo 8 anos de dados.









 COTAHIST_A2015	02/12/2022 10:59	Pasta compactada	13.799 KB
 COTAHIST_A2016	02/12/2022 10:59	Pasta compactada	15.348 KB
 COTAHIST_A2017	02/12/2022 10:58	Pasta compactada	16.140 KB
 COTAHIST_A2018	02/12/2022 10:58	Pasta compactada	19.032 KB
 COTAHIST_A2019	02/12/2022 10:57	Pasta compactada	26.026 KB
 COTAHIST_A2020	18/11/2022 13:00	Pasta compactada	36.311 KB
 COTAHIST_A2021	18/11/2022 13:02	Pasta compactada	48.515 KB
 COTAHIST_A2022	14/01/2023 06:02	Pasta compactada	57.225 KB

Figura 1: Cotações históricas utilizadas no projeto

¹ Obtido em: https://www.b3.com.br/pt_br/market-data-e-indices/servicos-de-dados/market-data/historico/mercado-a-vista/cotacoes-historicas/

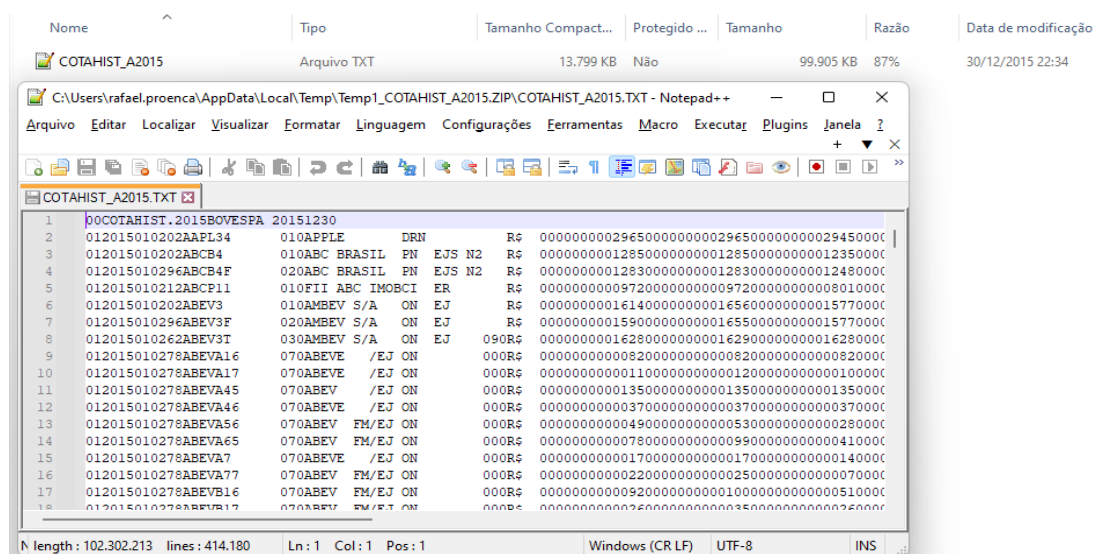


Figura 2: Formato dos dados obtidos

Para o dataset utilizado neste projeto, temos a seguinte tabela com a descrição do layout dos arquivos. A tabela também foi obtida no site da B3 e explicita de que maneira deve ocorrer a separação de campos. Na primeira linha de cada arquivo temos o *Header* e na última o *Trailer*. Estes campos não nos trazem informação útil para o projeto. Servem somente para marcar o início e fim dos arquivos, portanto são desconsiderados no script.

Entretanto, tudo que se encontra entre estes campos nos é dado útil. Como afirmado anteriormente, os dados estão em distribuição posicional. Cada linha representa os dados de uma determinada cotação histórica, por papel negociado no mercado.

Dessa maneira, a primeira atividade após a obtenção desses dados é a separação das features de maior interesse. Como dito, a B3 disponibiliza em sua plataforma o layout dos dados para download². Para o trabalho as linhas de dados não foram usadas em sua totalidade, ou seja, os dados foram utilizados apenas parcialmente de acordo com o interesse. Portanto é mais interessante trazermos aqui especificamente o layout posicional dos dados que foram extraídos e utilizados, que pode ser visto abaixo:

² O layout completo pode também ser obtido no link https://www.b3.com.br/pt_br/market-data-e-indices/servicos-de-dados/market-data/historico/mercado-a-vista/cotacoes-historicas/

Nome do campo	Descrição/Conteúdo	Tipo e tamanho	Posição inicial	Posição final
data_pregao	FORMATO “AAAAMMDD”	N(08)	03	10
cod_bdi	UTILIZADO PARA CLASSIFICAR OS PAPÉIS NA EMISSÃO DO BOLETIM DIÁRIO DE INFORMAÇÕES	X(02)	11	12
sigla_acao	CÓDIGO DE NEGOCIAÇÃO DO PAPEL	X(12)	13	24
nome_acao	NOME RESUMIDO DA EMPRESA EMISSORA DO PAPEL	X(12)	28	39
preco_abertura	PREÇO DE ABERTURA DO PAPELMERCADO NO PREGÃO	(11)V99	57	69
preco_max	PREÇO MÁXIMO DO PAPELMERCADO NO PREGÃO	(11)V99	70	82
preco_min	PREÇO MÍNIMO DO PAPELMERCADO NO PREGÃO	(11)V99	83	95
preco_fechamento	PREÇO DO ÚLTIMO NEGÓCIO DO PAPEL- MERCADO NO PREGÃO	(11)V99	109	121
qtd_negocios	QUANTIDADE TOTAL DE TÍTULOS NEGOCIADOS NESTE PAPEL- MERCADO	N(18)	153	170
vol_negocios	VOLUME TOTAL DE TÍTULOS NEGOCIADOS NESTE PAPEL- MERCADO	(16)V99	171	188

Tabela 1: Layout dos dados utilizados

Um segundo dataframe utilizado foi o da composição da carteira Ibovespa, foco deste trabalho, a fim de selecionar somente as ações que fazem parte do índice Ibovespa. A lista está disponível em formato csv, fornecido também pelo site da B3³. No momento de execução deste trabalho o índice era composto por 92 ações diferentes dos mais diversos setores econômicos.

³ Disponível em https://www.b3.com.br/pt_br/market-data-e-indices/indices/indices-amplos/indice-ibovespa-ibovespa-composicao-da-carteira.htm

ibov.csv				
	codigo;acao;tipo;qtde_teorica;part_percentual			
1	RRRP3;3R PETROLEUM;ON	NM;200.453.863;0,377		
2	ALPA4;ALPARGATAS;PN	N1;202.765.994;0,154		
3	ABEV3;AMBEV S/A;ON	4.386.652.506;3,272		
4	AMER3;AMERICANAS;ON	NM;596.086.291;0,315		
5	ARZZ3;AREZZO CO;ON	NM;60.358.029;0,254		
6	ASAI3;ASSAI;ON	NM;795.949.801;0,752		
7	AZUL4;AZUL;PN	N2;327.646.296;0,209		
8	B3SA3;B3;ON	NM;5.901.731.302;3,330		
9	BPAN4;BANCO PAN;PN	N1;341.549.703;0,110		
10	BBSE3;BBSEGURIDADE;ON	NM;671.682.536;0,974		
11	BRML3;BR MALLS PAR;ON	NM;828.273.884;0,329		
12	BBDC3;BRADESCO;ON	N1;1.508.883.586;0,964		
13	BBDC4;BRADESCO;PN	N1;5.156.077.326;3,881		
14	BRAP4;BRADESPAR;PN	EDJ N1;251.025.058;0,334		
15	BBAS3;BRASIL;ON	NM;1.420.731.069;2,374		
16				

Figura 3: Ações que compõem o índice Ibovespa

3. Processamento/Tratamento de Dados

Utilizando os arquivos mencionados no capítulo anterior, bem como o layout já apresentado, estamos prontos para iniciar o processamento dos dados históricos, com a finalidade de transformá-los em um dataframe adequado aos objetivos do projeto. Foi construído então um pequeno script chamado “*ETL_BOVESPA*” em linguagem de programação *python*, usando a ferramenta *Jupyter notebook* e as libs *pandas*, *os*, *numpy* e *glob*.

Em um primeiro momento, reunimos todos os arquivos *zip* dentro de uma pasta chamada “*cota_hist*”. Colocamos o script fora desta pasta e apontamos para que lesse todos os arquivos nessa pasta, contendo o nome “*COTAHIST*.zip*”. Após fazemos a leitura desses arquivos, geramos um dataframe e, por meio de substrings, extraímos somente os dados importantes com base em suas posições no arquivo, tendo excluído *header* e *trailer*, produzindo o resultado que podemos ver abaixo:

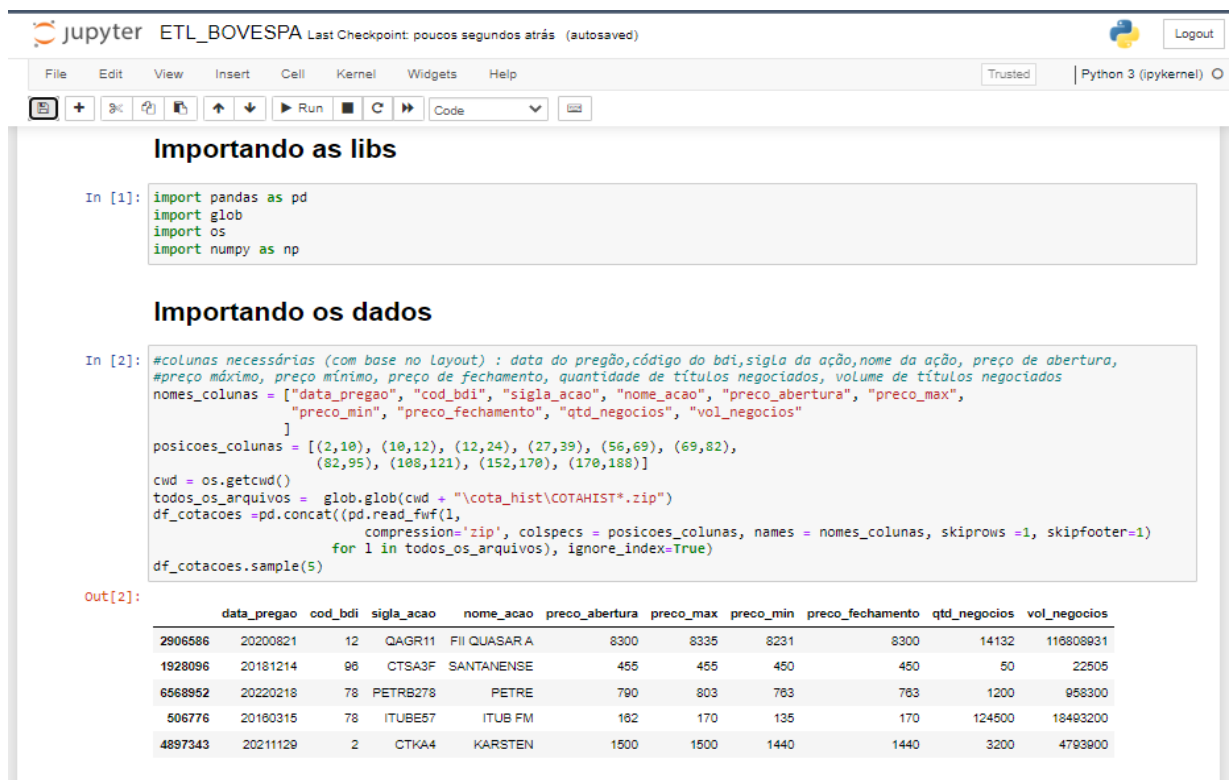


Figura 4: Extração inicial de dados

Como se pode ver na imagem abaixo, fizemos um primeiro tratamento de dados: por meio da coluna “cod_bdi” filtramos somente o valor 2. Isso é importante pois, de acordo com o layout disponibilizado pela B3, o código bdi 2 representa as ações no lote padrão, ou seja, disponibilizadas abertamente ao amplo público para negociação, e que são o foco do presente trabalho.

Pode-se ver também que, posteriormente, importamos um segundo dataframe (“ibov.csv”), que contem a lista de todas as ações que correspondem ao índice ibovespa, a fim de filtrarmos em nosso dataframe somente as ações que fazem parte do índice.

Selecionando os dados - lote padrão e índice Ibovespa

```
In [3]: #somente ações no Lote padrão (cod_bdi=2)

df_cotacoes = df_cotacoes[df_cotacoes["cod_bdi"]==2]
df_cotacoes = df_cotacoes.drop(["cod_bdi"],1)
df_cotacoes.sample(5)
```

C:\Users\RAFAEL~1.PRO\AppData\Local\Temp\ipykernel_23088\2115312662.py:4: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only.

```
df_cotacoes = df_cotacoes.drop(["cod_bdi"],1)
```

Out[3]:

	data_pregao	sigla_acao	nome_acao	preco_abertura	preco_max	preco_min	preco_fechamento	qtd_negocios	vol_negocios
766177	20181003	BIOM3	BIOMM	723	723	711	711	1200	868400
197348	20150830	SPRI3	SPRINGER	24	24	24	24	2000	48000
4020612	20210126	SHOW3	TIME FOR FUN	384	386	338	338	837100	298852200
1947008	20181227	OMGE3	OMEGA GER	1659	1774	1659	1733	191400	326134800
2054486	20190825	ALPA4	ALPARGATAS	2040	2078	2004	2029	449500	920514800


```
In [4]: #pegando somente as siglas das ações do índice Ibovespa
df_ibov_index=pd.read_csv("cota_hist\ibov.csv",sep=";")
#filtrando somente as cotações do índice Ibovespa
df_cotacoes = pd.merge(df_cotacoes, df_ibov_index, left_on="sigla_acao", right_on="codigo").drop(["codigo","acao","qtde_teorica"])
df_cotacoes.sample(5)
```

Out[4]:

	data_pregao	sigla_acao	nome_acao	preco_abertura	preco_max	preco_min	preco_fechamento	qtd_negocios	vol_negocios	tipo
126589	20220822	ENEV3	ENEVA	1475	1519	1487	1487	6599900	9898806200	ON NM
135902	20200406	SUZB3	SUZANO S.A.	4063	4137	3930	4002	9361800	37755159800	ON NM
50324	20180822	EMBR3	EMBRAER	2438	2438	2350	2430	2466700	5921320800	ON NM
51260	20220505	EMBR3	EMBRAER	1422	1461	1408	1449	9524900	13684600500	ON NM
102385	20201210	SLCE3	SLC AGRICOLA	2642	2656	2592	2637	558200	1468868000	ON NM

Figura 5: Filtrando o lote padrão e o índice ibovespa

Com base na imagem anterior podemos observar que existem problemas no formato de nossos dados. Primeiramente a data parece estar em um formato pouco interessante para se visualizar e trabalhar com fórmulas. Fizemos então uma conversão para o conhecido formato “AAAA-mm-DD” (“ano-mês-dia”), que vai permitir uma melhor visualização das datas dos pregões. Em um segundo momento buscamos tratar os valores numéricos. Os preços de abertura, fechamento, mínimo e máximo estão todos como inteiros, sem o divisor de real e centavos. Para cada um deles fizemos então a divisão por 100 e convertemos para o tipo *float*. Assim temos separados reais e centavos.

Ajustando a formatação

```
In [5]: #ajuste de data
df_cotacoes["data_pregao"] = pd.to_datetime(df_cotacoes["data_pregao"], format = "%Y%m%d")
df_cotacoes.sample(5)
```

```
Out[5]:
```

	data_pregao	sigla_acao	nome_acao	preco_abertura	preco_max	preco_min	preco_fechamento	qtd_negocios	vol_negocios	tipo
118111	2020-07-07	VIVT3	TELEF BRASIL	4872	4888	4790	4849	68300	329952300	ON
5310	2020-06-22	ARZZ3	AREZZO CO	4617	4750	4552	4629	1082000	4969013700	ON NM
73330	2015-07-21	KLB11	KLABIN S/A	1987	2029	1947	2007	1895200	3770870200	UNT N2
66305	2019-01-21	HYPE3	HYPERA	3301	3301	3232	3270	754100	2461772000	ON NM
120942	2018-09-06	ENGI11	ENERGISA	2080	2127	2080	2119	485200	1026306100	UNT N2

```
In [6]: #ajustes preços duas casas decimais
df_cotacoes["preco_abertura"] = (df_cotacoes["preco_abertura"]/100).astype(float)
df_cotacoes["preco_max"] = (df_cotacoes["preco_max"]/100).astype(float)
df_cotacoes["preco_min"] = (df_cotacoes["preco_min"]/100).astype(float)
df_cotacoes["preco_fechamento"] = (df_cotacoes["preco_fechamento"]/100).astype(float)
df_cotacoes.sample(5)
```

```
Out[6]:
```

	data_pregao	sigla_acao	nome_acao	preco_abertura	preco_max	preco_min	preco_fechamento	qtd_negocios	vol_negocios	tipo
39848	2016-02-26	CVCB3	CVC BRASIL	12.06	12.96	12.06	12.85	783600	987465900	ON NM
134508	2019-10-04	IRBR3	IRB BRASIL RE	35.75	36.47	35.50	36.35	2885300	10387092100	ON NM
93045	2015-03-17	RADL3	RAIADROGASIL	26.00	26.13	25.48	25.79	1659400	4282959800	ON NM
101487	2017-04-26	SLCE3	SLC AGRICOLA	18.54	19.00	18.50	19.00	277900	523365900	ON NM
48914	2020-10-09	ELET6	ELETROBRAS	31.10	31.85	30.79	30.79	3955000	12330075000	PNB N1

Figura 6: Conversão de datas, valores numéricos e armazenamento do dataframe

Um grande problema presente em análises de dados é a consistência de dados. Muitas vezes os dados são disponíveis de forma duplicada, com dados faltantes, inválidos, ou que podem tender para infinito durante um processo de divisão por um número muito pequeno. Dessa forma é importante verificarmos e tratarmos os dados, antes de salvarmos um dataframe para uso.

Neste trabalho optou-se pela exclusão de todas as possíveis linhas duplicadas. Em seguida verificamos se existiam nulos ou possíveis infinitos, que não foram encontrados. Fizemos uma verificação descritiva dos dados e salvamos o dataframe em um formato csv. Ele será nossa fonte de dados para iniciar a construção de nosso modelo.

Lidando com duplicidades e ausências

```
In [7]: #removendo possíveis duplicatas
df_cotacoes = df_cotacoes.drop_duplicates()

In [8]: #checando se existem dados vazios ou infinitos
data_to_drop = df_cotacoes.isin([np.inf, -np.inf, np.nan])

df_cotacoes[data_to_drop == True].count()

Out[8]: data_pregao      0
        sigla_acao      0
        nome_acao       0
        preco_abertura  0
        preco_max       0
        preco_min       0
        preco_fechamento 0
        qtd_negocios    0
        vol_negocios    0
        tipo            0
        dtype: int64
```

Figura 7: Duplicidades e ausências

Dataframe tratado

```
In [9]: df_cotacoes.describe()
```

```
Out[9]:
```

	preco_abertura	preco_max	preco_min	preco_fechamento	qtd_negocios	vol_negocios
count	148259.000000	148259.000000	148259.000000	148259.000000	1.482590e+05	1.482590e+05
mean	26.728911	27.179908	26.253917	26.721135	8.133815e+08	1.655556e+10
std	20.978023	21.356588	20.820934	21.024284	1.498295e+07	3.172388e+10
min	0.670000	0.690000	0.630000	0.660000	1.000000e+02	1.400000e+04
25%	13.420000	13.670000	13.160000	13.410000	1.391650e+08	3.284075e+09
50%	22.110000	22.510000	21.700000	22.100000	3.459600e+08	7.847814e+09
75%	34.570000	35.120000	33.990000	34.560000	8.862000e+08	1.675873e+10
max	579.000000	622.000000	561.420000	621.790000	4.991948e+08	1.099840e+12

```
In [10]: df_cotacoes.to_csv(cwd + "\cota_hist\cotacoes.csv.zip", index_label = "index", compression="zip")
```

```
In [11]: df_cotacoes.sample(5)
```

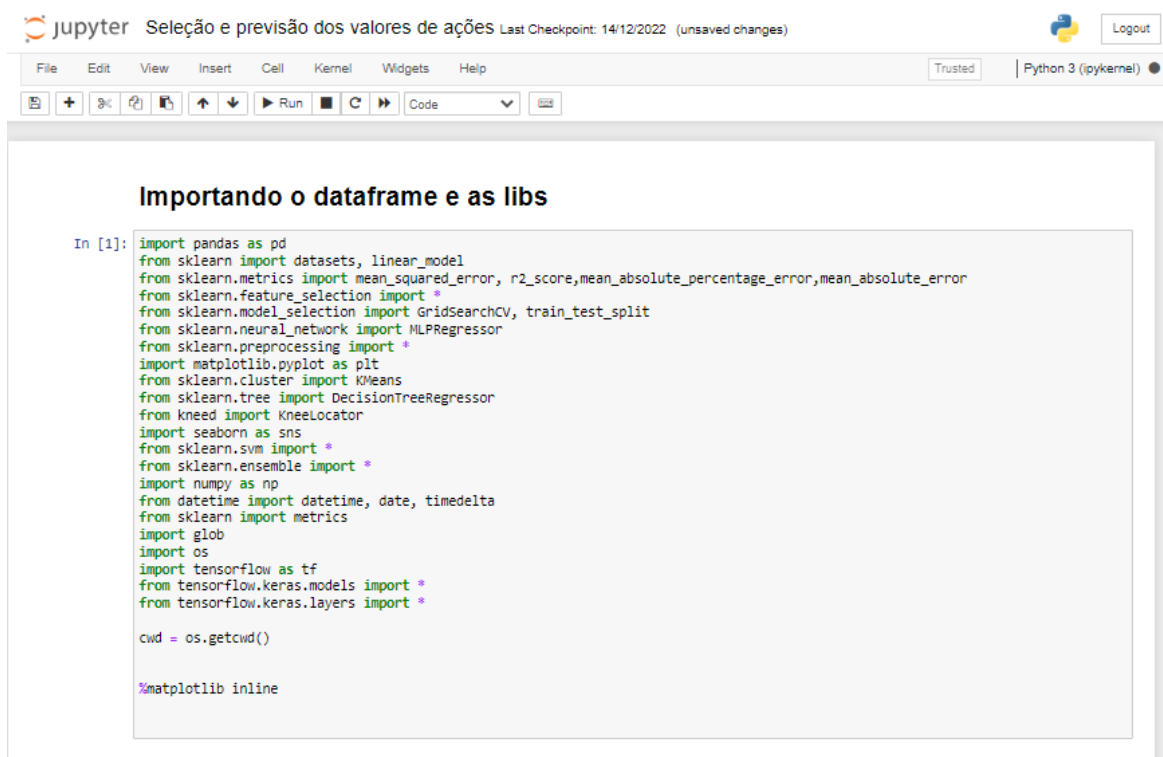
```
Out[11]:
```

	data_pregao	sigla_acao	nome_acao	preco_abertura	preco_max	preco_min	preco_fechamento	qtd_negocios	vol_negocios	tipo
49163	2021-10-14	ELET6	ELETROBRAS	39.75	40.42	39.19	40.16	2724600	10838344200	PNB N1
128738	2018-11-07	BPAC11	BTGP BANCO	21.37	22.30	20.70	22.07	2531100	5373616700	UNT N2
108505	2021-09-10	TAEE11	TAESA	37.46	37.79	37.03	37.03	2159700	8039346800	UNT EDJ N2
96622	2021-08-31	RENT3	LOCALIZA	56.79	56.89	54.84	55.46	7940300	44118067200	ON NM
105666	2018-03-16	SULA11	SUL AMERICA	22.19	22.57	22.06	22.50	1368200	3137230500	UNT N2

Figura 8: Dataframe disponível para análise

4. Análise e Exploração dos Dados

Criamos um segundo script chamado “*Seleção e previsão dos valores de ações*”, responsável pela análise e exploração dos dados gerados pelo script anterior, além da criação de indicadores, tratamento de dados e da verificação de preço futuro usando machine learning. Utilizamos nesse script a linguagem de programação *python*, a ferramenta *Jupyter notebook* e diversas libs como: *pandas*, *os*, *numpy*, *gloob* além de *sklearn* e *tensorflow*. Após a importação das *libs*, importamos também o dataframe gerado pelo script anterior e verificamos a consistência de dados, para garantir que está idêntico à forma como deixamos.



```
jupyter Seleção e previsão dos valores de ações Last Checkpoint: 14/12/2022 (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

In [1]: import pandas as pd
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_percentage_error, mean_absolute_error
from sklearn.feature_selection import *
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import *
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.tree import DecisionTreeRegressor
from kneed import KneeLocator
import seaborn as sns
from sklearn.svm import *
from sklearn.ensemble import *
import numpy as np
from datetime import datetime, date, timedelta
from sklearn import metrics
import glob
import os
import tensorflow as tf
from tensorflow.keras.models import *
from tensorflow.keras.layers import *

cwd = os.getcwd()

%matplotlib inline
```

Figura 9: Libs utilizadas no script

Verificando o dataframe

```
In [2]: #importando o dataframe
df = pd.read_csv("../cota_hist\\cotacoes.csv.zip", compression='zip', index_col="index")
df.sample(5)
```

```
Out[2]:
```

	data_pregao	sigla_acao	nome_acao	preco_abertura	preco_max	preco_min	preco_fechamento	qtd_negocios	vol_negocios	tipo
index										
128635	2018-08-11	BPAC11	BTGP BANCO	18.33	18.61	18.62	18.34	2507200	4577579200	UNT N2
25338	2021-07-13	BRML3	BR MALLS PAR	10.25	10.67	10.15	10.54	21697500	22793042900	ON NM
36791	2019-10-17	CSAN3	COSAN	57.83	57.80	56.70	56.70	758100	4331822400	ON NM
101443	2017-02-17	SLCE3	SLC AGRICOLA	17.80	17.80	17.58	17.60	204300	380719800	ON NM
92186	2019-09-27	QUAL3	QUALICORP	29.52	30.58	29.19	30.35	1680500	5059872200	ON NM

```
In [3]: df.describe()
```

```
Out[3]:
```

	preco_abertura	preco_max	preco_min	preco_fechamento	qtd_negocios	vol_negocios
count	148259.000000	148259.000000	148259.000000	148259.000000	1.482590e+05	1.482590e+05
mean	26.728911	27.179908	26.253917	26.721135	8.133615e+06	1.655556e+10
std	20.978023	21.356586	20.620934	21.024284	1.498295e+07	3.172388e+10
min	0.670000	0.690000	0.630000	0.660000	1.000000e+02	1.400000e+04
25%	13.420000	13.670000	13.160000	13.410000	1.391650e+06	3.284075e+09
50%	22.110000	22.510000	21.700000	22.100000	3.459600e+06	7.847814e+09
75%	34.570000	35.120000	33.990000	34.560000	8.892000e+06	1.675873e+10
max	579.000000	622.000000	561.420000	621.790000	4.991948e+08	1.099840e+12

Figura 10: Importando e verificando o dataframe

4.1 Análises e Indicadores Preliminares

Após a importação e validação do dataframe, iniciamos de fato a análise e exploração de dados. Para isso o primeiro passo é o enriquecimento do dataframe com alguns indicadores financeiros de interesse. O primeiro indicador utilizado e facilmente acessível é o retorno diário da ação. Usando como base o preço de fechamento das ações, podemos calcular o retorno diário dos ativos no período, comparando com o preço de fechamento do dia com o preço de fechamento do dia anterior.

Retorno diário

```
In [4]: %%capture
#dataframe de ações
array_siglas = df["sigla_acao"].unique()

df_enriquecido = pd.DataFrame()
for i in array_siglas:
    #ordenando por index
    df_sigla = df[df["sigla_acao"] == i].sort_values(["data_pregao"])
    #retorno
    df_sigla["variacao"] = df_sigla["preco_fechamento"].diff()
    #taxas de retorno
    df_sigla["taxa_de_retorno_simples_diaria"] = df_sigla["variacao"] / df_sigla["preco_fechamento"].shift(1)
    df_enriquecido = df_enriquecido.append(df_sigla)
```

```
In [5]: df_enriquecido[["data_pregao", "sigla_acao", "preco_fechamento", "variacao", "taxa_de_retorno_simples_diaria"]].sample(5)
```

```
Out[5]:
```

	data_pregao	sigla_acao	preco_fechamento	variacao	taxa_de_retorno_simples_diaria
index					
18763	2018-10-24	BRAP4	32.30	-1.30	-0.038890
95564	2017-05-22	RENT3	41.20	-1.79	-0.041638
74537	2020-06-09	KLBN11	19.68	0.08	0.004082
119439	2017-11-10	WEGE3	21.65	-0.03	-0.001384
119658	2018-10-02	WEGE3	19.74	0.25	0.012827

Figura 11: Retornos diários

O retorno diário de uma ação possibilita o cálculo do retorno médio diário, o retorno médio anualizado e a volatilidade da mesma. São medidas importantes que servem de base para determinar os riscos de um investimento. O retorno médio diário é calculado por meio da média dos retornos diários em determinado período de tempo, no caso o período total de nosso dataframe. O retorno médio anualizado é o retorno médio diário multiplicado pelo número de pregões em um ano (aqui estimados como 250 pregões anuais). São estimativas preliminares do retorno esperado.

Como medida de risco de um investimento, nas análises de mercado comumente se adota a volatilidade. A volatilidade de um ativo pode ser entendida como a intensidade das suas oscilações de preço, ou seja, pode ser representada pelo desvio padrão dos preços de um ativo. Em nosso caso, como usamos o retorno médio anualizado, a volatilidade será representada pelo desvio padrão anualizado.

Aqui também aplicamos um primeiro filtro de dados, a fim de delimitarmos melhor nosso escopo e diminuir previamente os riscos de investimento: selecionamos para análise somente as ações que obtiveram um percentual de retorno positivo no período e que apresentaram uma taxa de retorno média anualizada positiva. Essa estratégia possui a finalidade de selecionar ações com menor possibilidade de perda, baseando-se no histórico das cotações de preço de fechamento.

Retorno médio anual e volatilidade

```
In [6]: df_estatistico = df_enriquecido.groupby('sigla_acao').agg(
        preco_fechamento_min=pd.NamedAgg(column="preco_fechamento", aggfunc="min"),
        preco_fechamento_medio=pd.NamedAgg(column="preco_fechamento", aggfunc="mean"),
        preco_fechamento_inicial=pd.NamedAgg(column="preco_fechamento", aggfunc="first"),
        preco_fechamento_final=pd.NamedAgg(column="preco_fechamento", aggfunc="last"),
        preco_fechamento_variancia=pd.NamedAgg(column="preco_fechamento", aggfunc="var"),
        taxa_de_retorno_media_diaria=pd.NamedAgg(column="taxa_de_retorno_simples_diaria", aggfunc="mean"),
        desvpad_medio_diario = pd.NamedAgg(column="taxa_de_retorno_simples_diaria", aggfunc="std"))

        df_estatistico["taxa_de_retorno_media_anual"]=df_estatistico["taxa_de_retorno_media_diaria"]*250 #pregões anuais
        df_estatistico["taxa_de_retorno_media_anual"] = df_estatistico["taxa_de_retorno_media_anual"]

        df_estatistico["volatilidade"]=df_estatistico["desvpad_medio_diario"]*250*0.5 #a volatilidade é o desvio padrão anualizado

        df_estatistico["percentual_retorno"] = 100*(df_estatistico["preco_fechamento_final"] - df_estatistico["preco_fechamento_inicial"]
        df_estatistico= df_estatistico.dropna())

        df_estatistico = df_estatistico[df_estatistico["percentual_retorno"]>0]
        df_estatistico = df_estatistico[df_estatistico["taxa_de_retorno_media_anual"]>0]
        df_estatistico=df_estatistico[["taxa_de_retorno_media_diaria","desvpad_medio_diario","taxa_de_retorno_media_anual","volatilidade"]
        df_estatistico = df_estatistico.round(6)
        df_estatistico.sample(5)
```

Out[6]:

	taxa_de_retorno_media_diaria	desvpad_medio_diario	taxa_de_retorno_media_anual	volatilidade
sigla_acao				
USIM5	0.000946	0.038812	0.238588	0.613879
RENT3	0.000800	0.031125	0.199878	0.492137
SUZB3	0.001034	0.027029	0.258537	0.427386
BBSE3	0.000249	0.019906	0.062365	0.314737
VIVT3	0.000182	0.018744	0.045615	0.298383

Figura 12: Cálculo de Retorno e Volatilidade dos ativos

4.2 Adicionando indicadores básicos: clusterização de ações

Visualizando a distribuição da taxa de retorno média anual pela volatilidade, podemos perceber a existência de *outliers*, ou seja, valores atípicos que apresentam um grande afastamento dos demais valores da série, quando comparados aos demais. A presença desses valores pode ser danosa à análise estatística e a algoritmos de machine learning, produzindo falsos resultados e conclusões, pois há tendência de hipervalorização dessas entidades em detrimento das demais.

Vizualizando os dados por volatilidade e taxa de retorno media anual

```
In [7]: #cada ponto em azul significa uma ação diferente.
plt.subplots(figsize=(15, 5))
plt.subplot(1, 2, 1)
sns.scatterplot(data=df_estatistico, x="volatilidade", y="taxa_de_retorno_media_anual")

plt.subplot(1, 2, 2)
sns.boxplot(df_estatistico[['volatilidade', "taxa_de_retorno_media_anual"]])

plt.tight_layout()
plt.show()
```

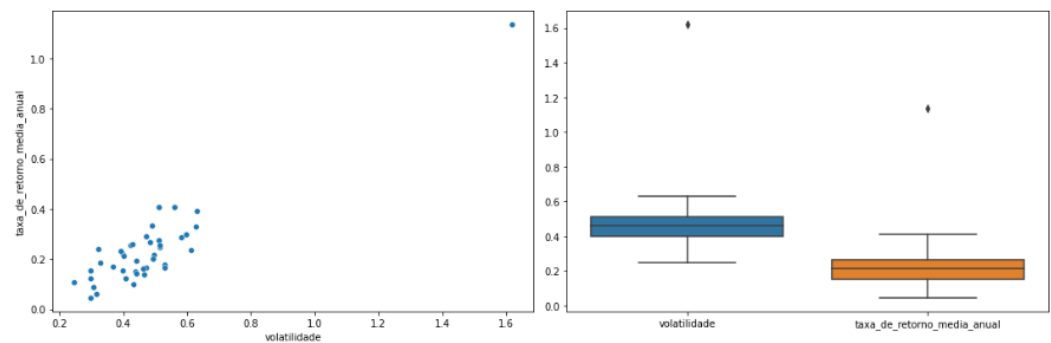


Figura 13: Visualização dos dados Por taxa de retorno média anual e volatilidade

Desta forma é necessário que esses *outliers* sejam removidos para o prosseguimento das análises. Para tanto utilizamos o método *inter quartile range (IQR)*. Nessa abordagem estatística calcula-se o *IQR* como a diferença de valores entre a linha limite do terceiro quartil (Q3) e do primeiro quartil (Q1). Deve-se entender que Q1 é o valor do conjunto que delimita os 25% menores valores de uma distribuição. Q3 delimita os 75% menores valores de uma distribuição (ou 25% maiores).

Após o cálculo do *IQR*, estabelece-se como limite inferior o valor de $Q1 - 1,5 \cdot IQR$, e como limite superior, o valor $Q3 + 1,5 \cdot IQR$. Tudo abaixo do limite inferior ou acima do limite superior é então descartado. Como detectamos *outliers* tanto na taxa de retorno média anual quanto na volatilidade, aplicamos o método para ambas as medidas, construindo uma função em *python*.

Removendo outliers - IQR

```
In [8]: #Removendo outliers

def remove_outlier(df_in, col_names):
    df_out=df_in
    for col_name in col_names :
        q1 = df_in[col_name].quantile(0.25)
        q3 = df_in[col_name].quantile(0.75)
        iqr = q3-q1 #Interquartile range
        fence_low = q1-1.5*iqr
        fence_high = q3+1.5*iqr
        df_out = df_out.loc[(df_out[col_name] > fence_low) & (df_out[col_name] < fence_high)]

    return df_out
df_estatistico = remove_outlier(df_estatistico,["taxa_de_retorno_medio_anual","volatilidade"])

sns.scatterplot(data=df_estatistico, x= "volatilidade", y="taxa_de_retorno_medio_anual")
plt.show()
```

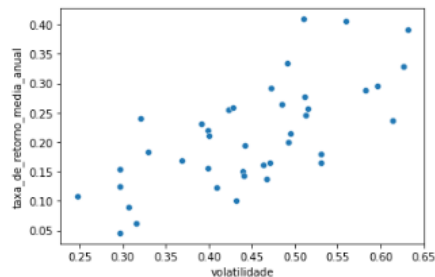


Figura 14: Remoção de outliers

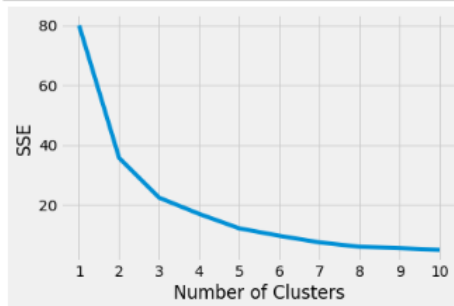
Com os outliers removidos pudemos agrupar ações com base em suas características de taxa de retorno médio anual e volatilidade, através de métodos de clusterização para, futuramente, agregar essa informação ao nosso dataframe, a fim de produzir mais um possível indicador. Para tanto, optou-se por utilizar o conhecido algoritmo K-Means disponível na biblioteca Scikit-Learn e que consiste em um método de segregar em torno de centros (centroídes) diversos dados de características semelhantes.

Podemos definir quantos clusters quisermos no K-Means. Entretanto, para garantir que haja equilíbrio entre homogeneidade de observações dentro de um cluster e, ao mesmo tempo, o máximo de distinção entre observações de clusters distintos, buscou-se otimizar o número de clusters através do método do cotovelo, onde o K-means é executado para diversas quantidades de clusters, estabelecendo um número ótimo.

Análise de clusters - definindo o número de clusters ideal

```
In [9]: def elbow_definition(features):
        kmeans_kwargs = {
            "init": "random",
            "n_init": 10,
            "max_iter": 600,
            "random_state": 42,
        }
        scaler = StandardScaler()
        scaled_features = scaler.fit_transform(features)
        sse = []
        for k in range(1, 11):
            kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
            kmeans.fit(scaled_features)
            sse.append(kmeans.inertia_)
        plt.style.use("fivethirtyeight")
        plt.plot(range(1, 11), sse)
        plt.xticks(range(1, 11))
        plt.xlabel("Number of Clusters")
        plt.ylabel("SSE")
        plt.show()
        kl = KneeLocator(
            range(1, 11), sse, curve="convex", direction="decreasing"
        )
        return kl.elbow

elbow1 = elbow_definition(df_estatistico[["taxa_de_retorno_medio_anual", "volatilidade"]])
print("elbows", elbow1)
```



elbows 3

Figura 15: Utilização do método do cotovelo para definição do número ideal de clusters

Clusterizando pelo número ótimo de clusters (3)

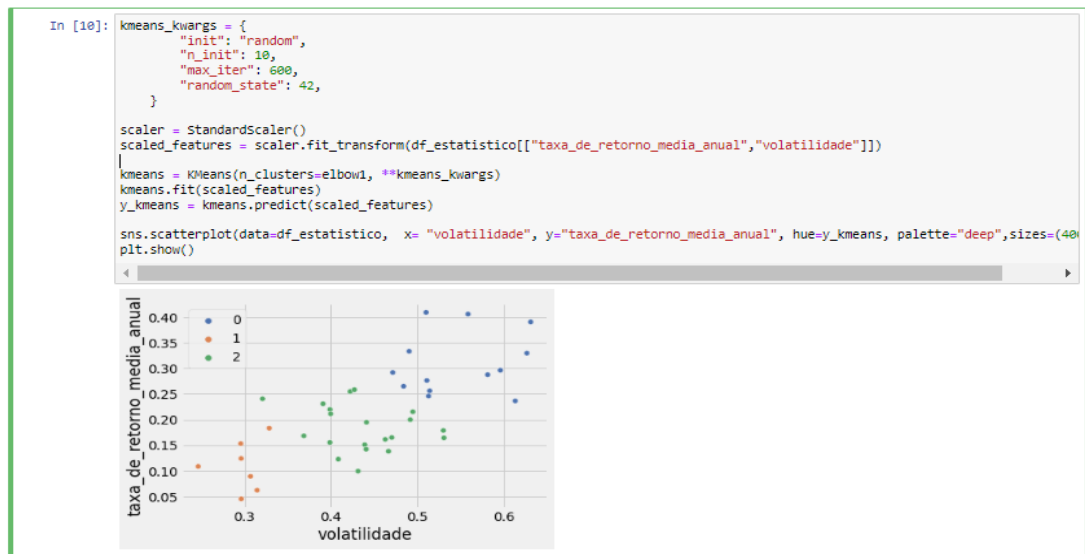


Figura 16: Aplicando o K-means pelo número ótimo de clusters

Com base no valor ótimo, pudemos clusterizar nossos registros, agrupando ações em termos de seu potencial de ganho e risco e, em seguida, agregar ao nosso dataframe a informação de qual cluster o registro pertence. Nosso dataframe enriquecido passa então a contar com as colunas extras de 'taxa_de_retorno_simples_diaria', 'taxa_de_retorno_medio_diaria', 'desvpad_medio_diario', 'taxa_de_retorno_medio_anual', 'volatilidade' e 'cluster'.

Agregando indicadores e cluster ao dataframe de pregões diários

```
In [11]: #criando uma coluna com o valor do cluster
df_estatistico["cluster"] = y_kmeans
df_estatistico.sample(5)
```

```
Out[11]:
```

	taxa_de_retorno_media_diaria	desvpad_medio_diario	taxa_de_retorno_media_anual	volatilidade	cluster
sigla_acao					
ENGI11	0.000982	0.020299	0.240614	0.320948	2
USIM5	0.000946	0.038812	0.238588	0.813679	0
VALE3	0.001188	0.029836	0.292024	0.471752	0
SBSP3	0.000924	0.024728	0.231039	0.390979	2
JBSS3	0.000882	0.031308	0.215379	0.495029	2

```
In [12]: #Enriquecendo o dataframe de cotações diárias com os : trazendo para o dataframe via inner join (merge)
df = pd.merge(df_enriquecido, df_estatistico, left_on="sigla_acao", right_index=True)
df.sample(5)
```

```
Out[12]:
```

	data_pregao	sigla_acao	nome_acao	preco_abertura	preco_max	preco_min	preco_fechamento	qtd_negocios	vol_negocios	tipo	variacao	taxa_r
index												
60109	2018-01-04	GGBR4	GERDAU	13.03	13.78	13.03	13.65	27558900	37371862100	FN N1	0.70	
12206	2016-05-02	BBSE3	BBSEGURIDADE	30.11	30.30	29.31	29.51	5224900	15489389100	ON NM	-0.49	
81427	2016-04-11	MRVE3	MRV	12.17	12.29	11.83	11.94	4172600	5039837000	ON NM	-0.14	
15456	2021-06-18	BEEF3	MINERVA	9.64	9.65	9.53	9.64	6542700	6273135900	ON NM	0.03	
43173	2021-08-03	CYRE3	CYRELA REALT	20.90	20.90	19.85	20.53	8933800	18113937400	ON NM	-0.37	

```
In [13]: df.columns
```

```
Out[13]: Index(['data_pregao', 'sigla_acao', 'nome_acao', 'preco_abertura', 'preco_max',  
               'preco_min', 'preco_fechamento', 'qtd_negocios', 'vol_negocios', 'tipo',  
               'variacao', 'taxa_de_retorno_simples_diaria',  
               'taxa_de_retorno_media_diaria', 'desvpad_medio_diario',  
               'taxa_de_retorno_media_anual', 'volatilidade', 'cluster'],  
              dtype='object')
```

Figura 17: Agregação de novos dados

4.3 Criando indicadores de análise técnica: Conceitos

Antes de prosseguir com a explicação da criação de indicadores via código, vale a pena esgotar algumas questões: Quais são os indicadores mais usados em negociações de ações no curto prazo? Qual o fundamento por trás destes indicadores? O que eles agregam na análise?

Primeiramente vale a pena conceitualizar alguns tipos de transação no curto prazo. Enquanto no *day trade* a compra e venda de ações ocorre no mesmo dia, no *swing trade* a compra e venda são separadas por períodos que podem variar de um dia a algumas semanas. O *swing trade* busca aproveitar da volatilidade do mercado no curto prazo para lucrar com a diferença dos preços de compra e venda, mas sempre em dias separados.

A grande vantagem do *swing trade*, comparado ao *day trade*, é a possibilidade de consolidar lucros no curto prazo sem a necessidade de acompanhamento do mercado a todo instante, fazendo mais sentido às pessoas que desejam operar no curto prazo mas tem uma outra profissão a qual se dedicam em tempo integral. No *swing trade* as decisões de compra e venda de uma ação são tomadas com base na análise técnica, utilizando para isso padrões gráficos e indicadores técnicos. Nos próximos parágrafos explicaremos alguns dos principais indicadores usados.

Um indicador muito usado em análises de curto prazo é a análise de médias móveis por meio das Bandas de Bollinger, metodologia criada por John A. Bollinger. As Bandas de Bollinger nada mais são do que as análises das médias móveis e suas respectivas volatilidades, em diferentes períodos de tempo. Via de regra, com base no valor histórico de um ativo, calculam-se as médias móveis e os desvios padrões para cada dia, observando as 13, 60 e 200 operações anteriores, mas outros períodos podem ser agregados. Em nosso caso usamos 5, 13, 60 e 200.

As Bandas de Bollinger são então um conjunto de três bandas (inferior, intermediária e superior) para todos os dias de operação de um ativo baseado em cada média móvel, em conjunto com desvio padrão e um fator multiplicativo k (em geral $k=2$). O valor da média móvel corresponde a banda intermediária. A banda superior pode ser definida como a soma da média móvel com k vezes o desvio padrão dos preços do ativo e a banda inferior a subtração da média móvel com k vezes o desvio padrão dos preços do ativo. Um exemplo pode ser visto abaixo⁴:

⁴Obtido de https://pt.wikipedia.org/wiki/An%C3%A1lise_de_Bollinger

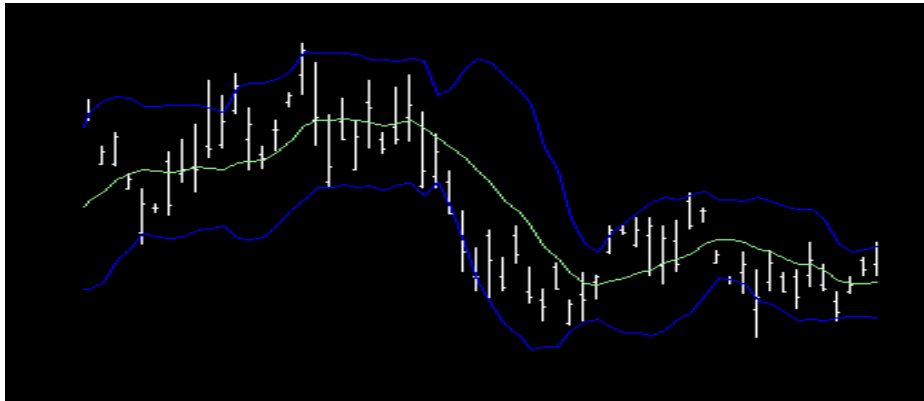


Figura 18: Bandas de Bollinger num período de dez dias e com a largura de duas vezes o desvio padrão.

Outro indicador muito utilizado em análise técnica é o chamado *IFR – Índice de Força Relativa*. Criado por J. Welles Wilder, é um indicador que visa avaliar se um ativo se encontra em estado de sobrecompra, ou seja, o preço do ativo está mais alto que o que realmente vale ou se o ativo está em estado de sobrevenda, com o preço menor do que realmente vale. Ativos em sobrecompra são aqueles que passaram por um processo de valorização intenso, tendendo a iniciar um movimento de baixa a qualquer momento. Na sobrevenda temos o oposto. O ativo passou por prolongada desvalorização e tenderá a reverter.

O cálculo do *IFR* se dá em duas etapas: primeiramente calcula-se a chamada *Força Relativa Simples*. Nela calcula-se a soma de todos os ganhos (variação positiva de preços) em determinado período de tempo. Em seguida esse valor é dividido pelo módulo da soma de todas as variações negativas de preços (perdas) num determinado período de tempo.

$$FR_{Simples} = \frac{Un,i}{||Dn,i||}$$

Em seguida, numa segunda etapa é calculada a *Força Relativa Clássica*, que é mais facilmente explicada pela fórmula:

$$FR_{clássica} = \frac{Un,i-1 \times |n-1| + Ui}{Dn,i-1 \times |n-1| + Di}$$

Onde n é o número de períodos, $Un,i-1$ o ganho médio em n períodos anteriores ao pregão avaliado, $Dn,i-1$ a perda média em n períodos anteriores ao pregão avaliado, Ui o ganho atual e Di a perda atual. Ou seja: a *Força Relativa Simples* está contida na *Força Relativa Clássica* ao calcular os ganhos e perdas médias anteriores. Por fim calculam-se os *Índices de Força Relativa*, tanto para a força relativa simples quanto para a força relativa clássica:

$$IFR_{simples} = 100 + \frac{100}{1 + FR_{simples}}$$

$$IFR_{clássico} = 100 + \frac{100}{1 + FR_{clássica}}$$

Para o presente trabalho optou-se pela utilização do *IFRclássico* durante a análise em detrimento do simples, uma vez que o mesmo é o padrão na indústria. Portanto, toda menção ao *IFR*, daqui em diante, estará se referindo ao *IFRclássico*. Cabe mencionar também que, em natureza de sua fórmula, o *IFR* variará sempre entre 0 e 100. Segundo o próprio J. Welles Wilder, o *IFR* superior a 70 significa um ativo sobrecomprado, ao passo que um *IFR* inferior a 30 sugere um ativo sobrevendido.

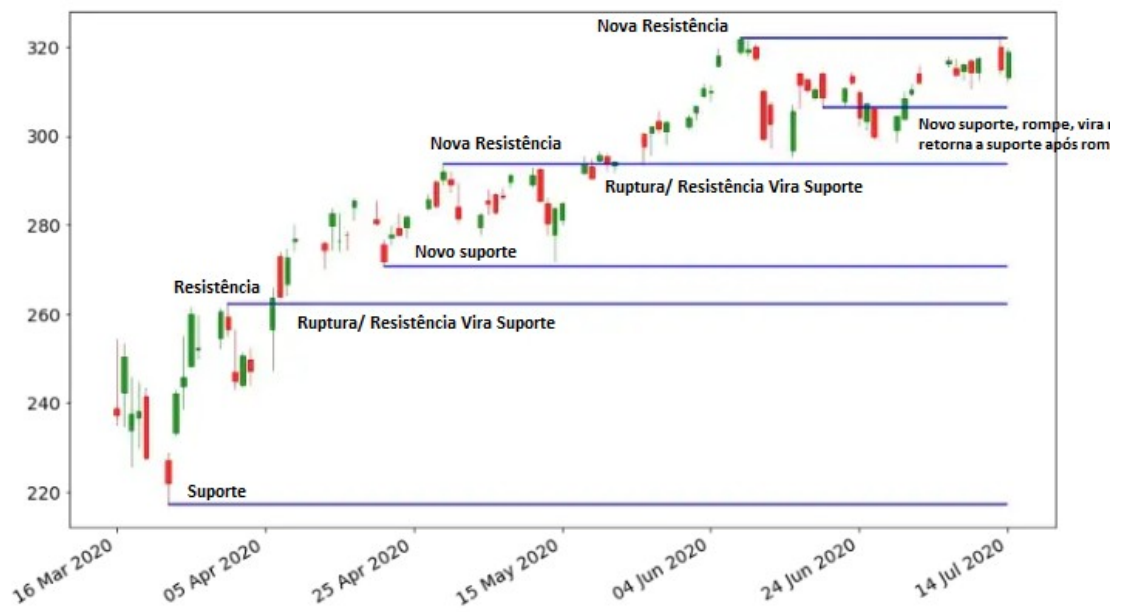
Por fim, um terceiro indicador muito utilizado em análise técnica é a análise de linhas de suporte e resistência. Linhas de suporte e resistência são patamares de oscilação. Uma linha de suporte (fundo) é uma linha que denota um limite inferior de preço da ação em determinado período de tempo, enquanto a linha de resistência (topos) correspondem a linhas de limite superior de preço da ação em determinado período de tempo. Ou seja: em uma alta a tendência é não ultrapassar uma linha de topo do passado. Em uma tendência de queda, a tendência é não ultrapassar a linha de fundo do passado.

As linhas de suporte e resistência são determinadas por meio de um padrão de fractal de 5 candles, em um gráfico de candles. Ou seja: observando um determinado candle como candle central, se seu valor de mínimo é menor que a mínima dos dois candles anteriores e dos dois candles seguintes, então teremos uma linha de suporte (fundo) representada pela mínima deste candle. Essa linha continuará sendo a linha de suporte válida até uma próxima aparecer.

Já uma linha de resistência é identificada observando um determinado candle como candle central, se seu valor de máximo é maior que o máximo dos dois candles anteriores e dos dois candles seguintes, então teremos uma linha de resistência (topo) representada pela máxima deste candle, e continuará sendo a linha de resistência válida até uma próxima aparecer.

Uma outra regra importante a ser levada em conta é que a partir do momento que um nível de resistência é rompido, ele vira um nível de suporte. A partir do momento que um nível de suporte é rompido ele vira um nível de resistência. Quando o nível de resistência se rompe a tendência é de aumento abrupto de preços. Quando o suporte se rompe, queda abrupta de preços. Esse tipo de comportamento pode ser visto abaixo⁵:

⁵Gráfico adaptado de <https://towardsdatascience.com/detection-of-price-support-and-resistance-levels-in-python-baedc44c34c9>



Fiaura 19: Linhas de suporte e resistência

4.4 Criando indicadores de análise técnica: Aplicação

Explicados os principais indicadores utilizados no *swing trade*, cabe agora explicar do como serão utilizados em nossa análise. Nossa análise, como já discutido, visa tentar, por meio de indicadores, prever o preço provável de fechamento de ações no dia seguinte, fornecendo mais insumos para a tomada de decisão. Veja bem, **ações** e não uma ação em específico.

Diferentemente de muitos trabalhos já realizados anteriormente, onde escolhe-se primeiro uma única ação ou uma pequena carteira de ações e se aplica um modelo de machine learning com base em poucos parâmetros, resolveu-se criar um modelo que pudesse servir, de forma mais genérica, a uma **significativa quantidade de ações** indexadas no índice Ibovespa.

Desta maneira, acabamos criando uma ferramenta mais robusta na fomentação da tomada de decisão. Por essas características de projeto, a especificação e seleção de indicadores é tão importante. Nesse sentido um primeiro passo, a nível de código, é criarmos uma estrutura de loop para aplicar os indicadores e cálculos a cada ação do dataframe. Criamos então um array com a sigla de todas as ações e agimos da seguinte forma:

- Até agora temos, para o nosso dataframe completo as seguintes colunas e seguintes ações a serem analisadas:

```
In [16]: #colunas
df.columns

Out[16]: Index(['data_pregao', 'sigla_acao', 'nome_acao', 'preco_abertura', 'preco_max',
               'preco_min', 'preco_fechamento', 'qtd_negocios', 'vol_negocios', 'tipo',
               'variacao', 'taxa_de_retorno_simples_diaria',
               'taxa_de_retorno_media_diaria', 'desvpad_medio_diario',
               'taxa_de_retorno_media_anual', 'volatilidade', 'cluster'],
              dtype='object')

In [17]: #array de ações
array_siglas = df["sigla_acao"].unique()
array_siglas

Out[17]: array(['ALPA4', 'ARZZ3', 'BBAS3', 'BBSE3', 'BEEF3', 'BPAN4', 'BRAP4',
               'BRKM5', 'CPFE3', 'CSNA3', 'CVRE3', 'ELET3', 'ELET6', 'ENBR3',
               'GGBR4', 'GOAU4', 'HYPE3', 'JBSS3', 'KLBNI1', 'MRFG3', 'MRVE3',
               'PETR3', 'PETR4', 'POSI3', 'RENT3', 'SANB11', 'SBSP3', 'SLCE3',
               'SULA11', 'TAEI11', 'USIM5', 'VALE3', 'VIVT3', 'WEGE3', 'ENGI11',
               'ENEV3', 'BPAC11', 'RAIL3', 'SUZB3', 'RRRP3'], dtype=object)
```

Figura 20: Ações selecionadas após todos os filtros

- Criamos primeiramente um dataframe vazio (“df_enriched”), que receberá mais à frente os resultados de nosso loop. Para cada sigla de ação presente em nosso array de siglas, pegamos o dataframe que temos até agora (“df” em nosso código), filtramos esse dataframe pela primeira sigla e organizamos de maneira crescente, gerando outro dataframe (“df_sigla”).

Adicionando indicadores de análise técnica

```
In [18]: %%capture
df_enriched = pd.DataFrame()
for i in array_siglas:
    #ordenando por data
    df_sigla = df[df["sigla_acao"] == i].sort_values(["data_pregao"])
```

Figura 21: Loop para inserção de indicadores

- Começamos o enriquecimento do dataframe “df_sigla” criando as colunas de médias móveis e calculando as *Bandas de Bollinger* para períodos de curtíssimo, curto, médio e longo prazo, no caso 5, 13, 60 e 200 dias, armazenando-os em colunas com os respectivos nomes. Calculamos também algumas propriedades internas das bandas, como sua volatilidade esperada (diferença entre bandas) e a tendência de movimento da média móvel (diferença entre preço de fechamento e média móvel).

```
#Bandas de Bollinger
for n in [5,13,60,200]:
    df_sigla[f"mm_{n}_dias"] = df_sigla["preco_fechamento"].rolling(n).mean() #meio
    df_sigla[f"desv_pad_{n}_dias"] = df_sigla["preco_fechamento"].rolling(n).std() #desvio padrao
    k = 2 #padrao
    df_sigla[f"banda_superior_{n}_dias"] = df_sigla[f"mm_{n}_dias"] + k*df_sigla[f"desv_pad_{n}_dias"] #banda superior
    df_sigla[f"banda_inferior_{n}_dias"] = df_sigla[f"mm_{n}_dias"] - k*df_sigla[f"desv_pad_{n}_dias"] #banda inferior
    df_sigla[f"volatilidade_entre_bandas_{n}_dias"] = 2*k*df_sigla[f"desv_pad_{n}_dias"] #diferença entre bandas
    df_sigla[f"diferenca_fechamento_e_mm_{n}_dias"] = df_sigla["preco_fechamento"] - df_sigla[f"mm_{n}_dias"]
```

Figura 22: Bandas de Bollinger aplicadas ao loop

- Em seguida calculamos o índice de força relativa. Para cada linha de nosso dataframe calculamos a perda ou ganho, comparando data atual e data anterior. Em seguida, calculamos as médias dos ganhos e perdas num período de 14 dias. Calculamos então as forças de resistência simples e clássica e, a partir delas, os índices de força de resistência simples e clássico.

```
#IFR – Índice de Força Relativa
df_sigla["ganho"] = np.where(df_sigla["variacao"] > 0, df_sigla["variacao"], 0)
df_sigla["perda"] = np.where(df_sigla["variacao"] < 0, df_sigla["variacao"].abs(), 0)
#De acordo com J. Welles Wilder, criador do indicador, os parâmetros recomendados para análise são um período de 14
#Valores acima de 70 sugerem um ativo sobrecomprado e abaixo de 30 indicam um ativo sobrevendido.
n=14
df_sigla[f"mm_{n}_dias_ganho"] = df_sigla["ganho"].rolling(n).mean()
df_sigla[f"mm_{n}_dias_perda"] = df_sigla["perda"].abs().rolling(n).mean() #em absolutos
df_sigla[f"fr_simples"] = df_sigla[f"mm_{n}_dias_ganho"] / df_sigla[f"mm_{n}_dias_perda"]
df_sigla[f"fr_classica"] = (df_sigla[f"mm_{n}_dias_ganho"].shift(1)*(n-1)+df_sigla["ganho"])/\
(df_sigla[f"mm_{n}_dias_perda"].shift(1)*(n-1)+df_sigla["perda"])
df_sigla["ifr_simples"] = 100 - (100/(1+df_sigla["fr_simples"]))
df_sigla["ifr_classica"] = 100 - (100/(1+df_sigla["fr_classica"]))
```

Figura 23: Índice de Força Relativa aplicado ao loop

- Calculamos as fractais de suporte e resistência. Em um padrão de 5 candles, se o candle central apresenta mínima com relação a seus dois vizinhos anteriores e dois vizinhos posteriores, é uma fractal de suporte. Se o candle central apresenta máxima, com relação a seus dois vizinhos anteriores e dois vizinhos posteriores, é uma fractal de resistência. Fractais de suporte tendem a ser linhas de suporte e fractais de resistência tendem a ser linhas de resistência, ao menos que haja o rompimento do suporte e resistências.

```
#Cálculo de suporte/resistência > num padrão de 5 candles, o candle central apresenta máxima ou mínima
#maior que os dois candles ao seu extremo
df_sigla['fractal_resistencia'] = np.select([(df_sigla['preco_max'] > df_sigla['preco_max'].shift(2)) & \
(df_sigla['preco_max'] > df_sigla['preco_max'].shift(1)) & (df_sigla['preco_max'] > df_sigla['preco_max'].shift(-1)) & \
(df_sigla['preco_max'] > df_sigla['preco_max'].shift(-2))], [df_sigla['preco_max']])
df_sigla['fractal_suporte'] = np.select([(df_sigla['preco_min'] < df_sigla['preco_min'].shift(2)) & \
(df_sigla['preco_min'] < df_sigla['preco_min'].shift(1)) & (df_sigla['preco_min'] < df_sigla['preco_min'].shift(-1)) & \
(df_sigla['preco_min'] < df_sigla['preco_min'].shift(-2))], [df_sigla['preco_min']])
#substitui 0 pelo valor anterior não nulo
df_sigla['fractal_resistencia'] = df_sigla['fractal_resistencia'].replace(to_replace=0, method='ffill')
df_sigla['fractal_suporte'] = df_sigla['fractal_suporte'].replace(to_replace=0, method='ffill')

#Estabelecendo linhas de suporte e resistência com base nos rompimentos

df_sigla['linha_resistencia'] = np.where((df_sigla["preco_min"] >= df_sigla['fractal_suporte']), \
df_sigla['fractal_resistencia'], df_sigla['fractal_suporte'])
df_sigla['linha_suporte'] = np.where((df_sigla["preco_min"] >= df_sigla['fractal_suporte']), df_sigla['fractal_suporte'], \
df_sigla["preco_min"])
df_sigla['linha_suporte'] = np.where((df_sigla["preco_max"] <= df_sigla['fractal_resistencia']), \
df_sigla['fractal_suporte'], df_sigla['fractal_resistencia'])
df_sigla['linha_resistencia'] = np.where((df_sigla["preco_max"] <= df_sigla['fractal_resistencia']), \
df_sigla['fractal_resistencia'], df_sigla["preco_max"])
```

Figura 24: Linhas de suporte e resistência

- Por fim, trazemos para nosso dataframe da sigla o preço de fechamento da ação para um dia futuro, que é o que queremos prever por meio de nossos modelos. Simplesmente para determinada data trazemos o preço de fechamento do dia seguinte. Inserimos todo resultado consolidado para nossa primeira sigla no dataframe, que antes estava vazio. Procedemos no loop pulando para a próxima sigla do nosso array, fazendo as mesmas análises, até que tenhamos todos os indicadores para todas as siglas.

```
df_sigla['fractal_resistencia'], df_sigla["preco_max"])

#Preço de fechamento dias futuros
f=1 #um dia
df_sigla["preco_{f}_dias_futuros".format(f)=df_sigla["preco_fechamento"].shift(-f)

df_enriched = df_enriched.append(df_sigla)
```

Figura 25: Preço de fechamento - 1 dia futuro

Esse loop completo foi demonstrado ao longo dos prints, mas pode também ser visto na próxima página, na íntegra. Devido ao tamanho da célula não foi possível realizar o *print screen* sem perda na qualidade.

```

%%capture
df_enriched = pd.DataFrame()
for i in array_siglas:
    #ordenando por data
    df_sigla = df[df["sigla_acao"] == i].sort_values(["data_pregao"])

    #Bandas de Bollinger
    for n in [5,13,60,200]:
        df_sigla[f"mm_{n}_dias"]=df_sigla["preco_fechamento"].rolling(n).mean()#meio
        df_sigla[f"desv_pad_{n}_dias"]=df_sigla["preco_fechamento"].rolling(n).std()#desvio padrao
        k = 2 #padrão
        df_sigla[f"banda_superior_{n}_dias"]=df_sigla[f"mm_{n}_dias"] + k*df_sigla[f"desv_pad_{n}_dias"] #banda superior
        df_sigla[f"banda_inferior_{n}_dias"]=df_sigla[f"mm_{n}_dias"] - k*df_sigla[f"desv_pad_{n}_dias"] #banda inferior
        df_sigla[f"volatilidade_entre_bandas_{n}_dias"] = 2*k*df_sigla[f"desv_pad_{n}_dias"] #diferença entre bandas
        df_sigla[f"diferenca_fechamento_e_mm_{n}_dias"] = df_sigla["preco_fechamento"] - df_sigla[f"mm_{n}_dias"]

    #IFR — Índice de Força Relativa
    df_sigla["ganho"] = np.where(df_sigla['variacao'] > 0, df_sigla['variacao'], 0)
    df_sigla["perda"] = np.where(df_sigla['variacao'] < 0, df_sigla['variacao'].abs(), 0)
    #De acordo com J. Welles Wilder, criador do indicador, os parâmetros recomendados para análise são um período de 14 dias.
    #Valores acima de 70 sugerem um ativo sobrecomprado e abaixo de 30 indicam um ativo sobrevendido.
    n=14
    df_sigla[f"mm_{n}_dias_ganho"]=df_sigla["ganho"].rolling(n).mean()
    df_sigla[f"mm_{n}_dias_perda"]=df_sigla["perda"].abs().rolling(n).mean() #em absolutos
    df_sigla["fr_simples"] = df_sigla[f"mm_{n}_dias_ganho"]/df_sigla[f"mm_{n}_dias_perda"]
    df_sigla["fr_classica"] = (df_sigla[f"mm_{n}_dias_ganho"].shift(1)*(n-1)+df_sigla["ganho"])/\
(df_sigla[f"mm_{n}_dias_perda"].shift(1)*(n-1)+df_sigla["perda"])
    df_sigla["ifr_simples"] = 100 - (100/(1+df_sigla["fr_simples"]))
    df_sigla["ifr_classica"] = 100 - (100/(1+df_sigla["fr_classica"]))

    #Cálculo de suporte/resistência > num padrão de 5 candles, o candle central apresenta máxima ou mínima
    #maior que os dois candles ao seu extremo
    df_sigla['fractal_resistencia'] = np.select([(df_sigla['preco_max'] > df_sigla['preco_max'].shift(2)) & \
(df_sigla['preco_max'] > df_sigla['preco_max'].shift(1)) & (df_sigla['preco_max'] > df_sigla['preco_max'].shift(-1)) & \
(df_sigla['preco_max'] > df_sigla['preco_max'].shift(-2))], [df_sigla['preco_max']])
    df_sigla['fractal_suporte'] = np.select([(df_sigla['preco_min'] < df_sigla['preco_min'].shift(2)) & \
(df_sigla['preco_min'] < df_sigla['preco_min'].shift(1)) & (df_sigla['preco_min'] < df_sigla['preco_min'].shift(-1)) & \
(df_sigla['preco_min'] < df_sigla['preco_min'].shift(-2))], [df_sigla['preco_min']])
    #substitui 0 pelo valor anterior não nulo
    df_sigla['fractal_resistencia'] = df_sigla['fractal_resistencia'].replace(to_replace=0, method='ffill')
    df_sigla['fractal_suporte'] = df_sigla['fractal_suporte'].replace(to_replace=0, method='ffill')

    #Estabelecendo linhas de suporte e resistência com base nos rompimentos

    df_sigla['linha_resistencia'] = np.where((df_sigla["preco_min"] >= df_sigla['fractal_suporte']), \
df_sigla['fractal_resistencia'], df_sigla['fractal_suporte'])
    df_sigla['linha_suporte'] = np.where((df_sigla["preco_min"] >= df_sigla['fractal_suporte']), df_sigla['fractal_suporte'], \
df_sigla["preco_min"])
    df_sigla['linha_suporte'] = np.where((df_sigla["preco_max"] <= df_sigla['fractal_resistencia']), \
df_sigla['fractal_suporte'], df_sigla["fractal_resistencia"])
    df_sigla['linha_resistencia'] = np.where((df_sigla["preco_max"] <= df_sigla['fractal_resistencia']), \
df_sigla['fractal_resistencia'], df_sigla["preco_max"])

    #Preço de fechamento dias futuros
    f=1 #um dia
    df_sigla["preco_{_}dias_futuros".format(f)]=df_sigla["preco_fechamento"].shift(-f)

    df_enriched = df_enriched.append(df_sigla)

```


4.5 Criando indicadores de análise técnica: Tratamento de dados

Após a geração de novos dados para enriquecer a análise, devemos tratar esses dados. Por exemplo, ao agregar médias móveis ao dataframe, obteremos muitos valores nulos oriundos dos dados iniciais, onde não há preços de fechamentos anteriores. Por conseguinte, todos os indicadores que dependem de uma divisão com relação as médias móveis, como os índices de força relativa, poderão resultar em infinitos, pelos mesmos aspectos já mencionados.

Dessa forma, para proceder com a análise vamos dropar esses valores para podermos utilizar em nossos modelos somente dados numéricos consistentes, sem zeros, nulos ou infinitos que possam destruir nossa análise.

Checando se há nulos e infinitos

```
In [20]: #checando
data_to_drop = df_enriched.isin([np.inf, -np.inf, np.nan])

data_to_drop[data_to_drop == True].count()
```

```
Out[20]: data_pregao          0
sigla_acao          0
nome_acao           0
preco_abertura      0
preco_max            0
preco_min            0
preco_fechamento    0
qtd_negocios         0
vol_negocios         0
tipo                0
variacao             0
taxa_de_retorno_simples_diaria  40
taxa_de_retorno_media_diaria    0
desvpad_medio_diario    0
taxa_de_retorno_media_anual     0
volatilidade           0
cluster               0
mm_5_dias             160
desv_pad_5_dias        160
banda_superior_5_dias   160
banda_inferior_5_dias   160
volatilidade_entre_bandas_5_dias  160
diferenca_fechamento_e_mm_5_dias  160
mm_13_dias            480
desv_pad_13_dias       480
banda_superior_13_dias  480
banda_inferior_13_dias  480
volatilidade_entre_bandas_13_dias  480
diferenca_fechamento_e_mm_13_dias  480
mm_60_dias            2360
desv_pad_60_dias       2360
banda_superior_60_dias  2360
banda_inferior_60_dias  2360
```

Figura 26: Verificação de nulos e infinitos

```

In [21]: # Transformando infinito em nan
df_enriched.replace([np.inf, -np.inf], np.nan, inplace=True)
# dropando esses valores
df_enriched=df_enriched.dropna()
# arredondando o que sobra
df_enriched = df_enriched.round(6)
df_enriched

Out[21]:

```

	nome_acao	preco_abertura	preco_max	preco_min	preco_fechamento	qtd_negocios	vol_negocios	tipo	...	mm_14_dias_perda	fr_simples	fr_classica	ifr_si
	ALPARGATAS	7.98	8.35	7.84	8.35	177200	144398200	PN N1	...	0.080000	2.940476	3.000916	74.6
	ALPARGATAS	8.40	8.50	8.15	8.38	433300	363205200	PN N1	...	0.052143	3.424658	2.978938	77.3
	ALPARGATAS	8.40	8.56	8.02	8.15	432900	355426300	PN N1	...	0.088571	2.447917	2.557042	70.9
	ALPARGATAS	8.15	8.38	8.15	8.35	295800	245500700	PN N1	...	0.088571	2.031250	2.672276	67.0
	ALPARGATAS	8.37	8.38	8.08	8.35	203400	168481300	PN N1	...	0.088571	1.208333	2.031250	54.7

	3R PETROLEUM	33.52	33.90	32.81	33.10	3782900	12546145300	ON NM	...	0.415714	0.400344	0.478487	28.5
	3R PETROLEUM	34.30	36.07	33.80	35.86	6847800	24048936400	ON NM	...	0.318429	1.148984	0.911049	53.4
	3R PETROLEUM	35.90	36.59	35.24	36.11	2430800	8762984000	ON NM	...	0.227143	1.679245	1.209759	62.6
	3R PETROLEUM	36.80	36.98	35.82	36.00	3210500	11581166100	ON NM	...	0.188429	2.045977	1.618937	67.1
	3R PETROLEUM	36.00	37.14	35.75	37.03	3034000	11145942000	ON NM	...	0.188429	2.429119	2.470970	70.8

Figura 27: Dataframe tratado

Como construiremos modelos de machine learning, sobretudo baseados em regressões, não podemos usar dados textuais. Dessa forma as colunas “sigla_da_acao” e “tipo” precisam passa por um tipo de encoder para serem codificadas e poderem ser usadas.

Codificando as features que são string

```

In [22]: le = LabelEncoder()
le.fit(df_enriched["sigla_acao"])
df_enriched["sigla_acao_label"]=le.transform(df_enriched["sigla_acao"])
le.fit(df_enriched["tipo"])
df_enriched["tipo_label"]=le.transform(df_enriched["tipo"])

In [23]: df_enriched

Out[23]:

```

	...	fr_classica	ifr_simples	ifr_classica	fractal_resistencia	fractal_suporte	linha_resistencia	linha_suporte	preco_1_dias_fut
PN N1	...	3.000916	74.622356	75.005722	8.43	7.65	8.43	7.65	
PN N1	...	2.978938	77.399381	74.867664	8.43	7.65	8.50	8.43	
PN N1	...	2.557042	70.998979	71.888751	8.56	7.65	8.56	7.65	
PN N1	...	2.672276	67.010309	72.768929	8.56	7.65	8.56	7.65	
PN N1	...	2.031250	54.716981	67.010309	8.56	7.65	8.56	7.65	
...	
3N IM	...	0.478487	28.588957	32.270738	34.55	31.45	34.55	31.45	3
3N IM	...	0.911049	53.486387	47.672730	34.55	31.45	36.07	34.55	3
3N IM	...	1.209759	62.678056	54.746189	34.55	31.45	36.59	34.55	3
3N IM	...	1.618937	67.169811	61.816563	34.55	31.45	36.98	34.55	3
3N IM	...	2.470970	70.837989	71.189807	34.55	31.45	37.14	34.55	3

Figura 28: Aplicação do LabelEncoder

5. Criação de Modelos de Machine Learning

5.1 Introdução

Antes de iniciarmos a nossa abordagem, cabe aqui elucidar um pouco melhor a respeito das tarefas que serão realizadas, dos modelos que serão adotados e das razões para isso. A ideia central neste trabalho, primordialmente, é construir um modelo que preveja o preço de fechamento de ações. Porém, diferentemente de muitos modelos que são vistos corriqueiramente, a ideia é criar um modelo generalista, que possa ser aplicado a uma grande gama de ações indexadas no índice Ibovespa.

De forma simples, podemos imaginar a seguinte situação: uma pessoa, caso tenha interesse tanto em uma ação específica do índice quanto em múltiplas ações que façam parte do escopo do trabalho, poderia simplesmente fazer os downloads dos dados do site, calcular os indicadores e aplicar o mesmo modelo para a previsão do preço de fechamento de todas as suas ações de interesse, com uma taxa aceitável de erro, sem a necessidade de construção de um modelo para cada ação, gerando uma economia de tempo e recursos computacionais, e ainda assim tendo insumos acertivos para a tomada de decisão.

Como base no problema proposto e nos dados obtidos até aqui, temos uma situação onde o preço de fechamento de uma ação para o dia seguinte (tópico de interesse, nossa *label*) está intrinsecamente ligado a uma série de múltiplas variáveis, como nossos indicadores, médias, históricos (nossas *features*), cada uma com maior ou menor impacto sobre o tópico de interesse. E essa relação entre features e label pode ser abordada por meio de diversos algoritmos de machine learning diferentes.

Nesse sentido, optou-se por utilizar diversos métodos de análise de regressão. A regressão é um método confiável de se trabalhar a relação entre *features* e *label* sobretudo em tarefas preditivas, além de serem algoritmos de fácil entendimento, com documentação amplamente disponível. Entre os principais algoritmos de regressão estão a regressão linear, a regressão polinomial, as árvores de decisão para regressão e a *random forrest regression*. Estes quatro algoritmos serão utilizados e comparados nesse trabalho.

Na regressão linear a relação *feature(s)* X *label* se dá por meio de uma equação linear. Quando se usa somente uma variável preditora, tem-se a *Regressão Linear Simples*. Quando se usa mais de uma variável preditora, tem-se a *Regressão Linear Múltipla ou Multivariada*. Essa última, é justamente a usada no trabalho. Na *Regressão Linear Simples* uma determinada *feature* se relaciona de maneira linear com o *label*, de modo que podemos traçar uma equação linear do tipo:

$$Y = \alpha + \beta \times X + \varepsilon$$

Aqui, X é nossa variável preditora (a feature). A influência da variável preditora é dada pelo β , que no gráfico diz o quão inclinado está a reta. Já α é o valor que descreve o intercepto: onde a linha está quando o valor de X é zero. O valor ϵ corresponde ao erro.

De forma semelhante, na *Regressão Linear Múltipla ou Multivariada* o conjunto das features tenta aproximar a label de maneira linear para cada uma das features:

$$Y = \alpha + \beta_1 \times X_1 + \beta_2 \times X_2 + \beta_3 \times X_3 + \dots + \beta_n \times X_n + \epsilon$$

Aqui, cada uma das variáveis X_1 a X_n representa uma das features e cada um dos β sua respectiva influência. Já α é o valor que descreve o intercepto: onde a linha está quando o valor de todos os X é zero. O valor ϵ corresponde ao erro.

A regressão polinomial também é um modelo de regressão que se difere da regressão linear a medida que pode estabelecer relações não lineares entre *features* e *labels*. A relação *feature(s) X label* se dá por meio de um polinômio de grau k^6 . Na regressão polinomial de grau k com n variáveis teremos:

$$Y = \alpha + (\beta_1 \times X_1^1 + \dots + \beta_k \times X_1^k) + (\gamma_1 \times X_2^1 + \dots + \gamma_k \times X_2^k) + \dots + (\mu_1 \times X_n^1 + \dots + \mu_k \times X_n^k) + \epsilon$$

De forma semelhante, aqui, cada uma das variáveis X_1 a X_n representa uma das features. A letra grega α segue representando o valor que descreve o intercepto, assim como o valor ϵ continua a corresponder ao erro. As demais letras gregas correspondem a influência de determinada variável elevada aquela determinada potência, dentro do polinômio.

Um outro algoritmo que usaremos em nossa análise consiste nas árvores de decisão. São algoritmos de aprendizado de máquina supervisionado, extremamente versáteis, utilizados tanto em tarefas de classificação quanto de regressão. Nas árvores de decisão, os *decision nodes* se relacionam hierarquicamente. Um nó raiz (*root node*) é dividido em diversos sub-nós, cada um representando uma análise feita, um critério de filtragem, um cálculo realizado, uma comparação.

Quando um sub-nó é dividido em mais sub-nós, é chamado nó de decisão (*decision node*). Simboliza que está acontecendo aprofundamento nas análises. Quando um sub-nó não é dividido em nenhum sub-nó adicional, é chamado de nó folha (*leaf node*) e significa que temos possíveis saídas. O processo de divisão de sub-nós é conhecido como *splitting*. Uma subseção da árvore de decisão é chamada ramo, ou *branch*.

⁶ De fato, pode-se dizer que a regressão linear é uma regressão polinomial de grau 1

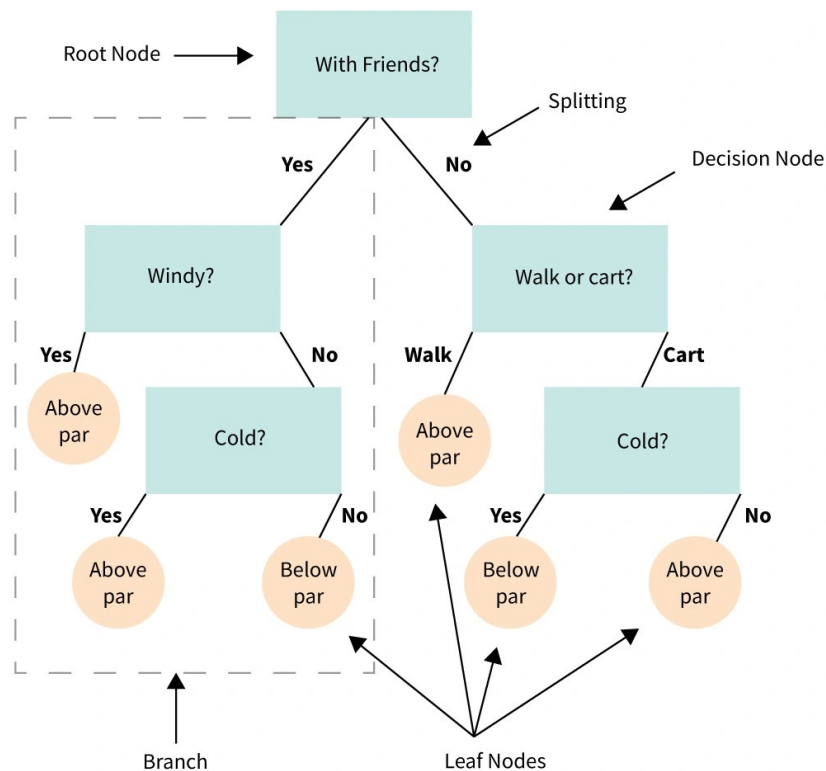


Figura 29: Imagem disponível em <https://www.mastersindatascience.org/learning/machine-learning-algorithms/decision-tree/>

Já o *random forrest* consiste basicamente em uma derivação das árvores de decisão. O algoritmo cria, de forma aleatória, diversas árvores de decisão e combina e otimiza o resultado de todas elas para chegar no resultado final. Um grande número de modelos (árvores) relativamente não correlacionados realizam suas previsões usando diferentes amostras e diferentes *features*. Essas árvores formam uma espécie de comitê (*ensemble*) onde cada árvore dá seu “voto” em prol do valor que acha correto, trabalhando com a ideia de sabedoria da multidão. A decisão mais votada é a que irá ser levada como resultado final.

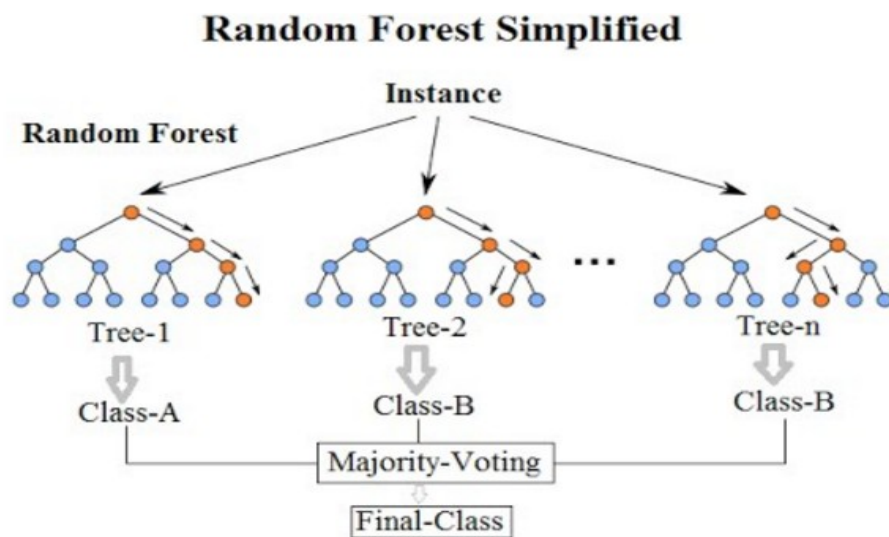


Figura 30: Disponível em https://en.wikipedia.org/wiki/Random_forest

Por serem algoritmos extremamente conhecidos, testados, facilmente replicáveis, com alto poder de extrapolação e que, via de regra, necessitam de poucos ajustes para serem aplicados a uma gama de operações diferentes, estes foram os algoritmos de machine learning escolhidos primordialmente para a realização deste trabalho, haja em vista que se pretende construir modelos generalistas, assertivos, capazes de prever os preços de uma ampla gama de ativos.

5.2 Seleção de features

A partir daqui podemos verificar quais as colunas estão presentes em nosso dataframe. Podemos verificar que temos muitas informações que não agregam ou são redundantes e podem trazer algum tipo de overfitting aos nossos modelos. Por exemplo os valores das colunas 'variacao', 'taxa_de_retorno_simples_diaria', 'taxa_de_retorno_media_diaria' e 'desvpad_medio_diario' já estão contidas de alguma maneira na análise de clusters executadas anteriormente. Os valores de ganhos, perdas e forças de resistência simples e clássica já fazem parte do cálculo do índice de força de resistência. Os desvios padrões para 5, 13, 60 e 200 períodos já estão contidos nos cálculos das Bandas de Bollinger.

Selecionando as melhores features - Verificando as colunas

```
In [24]: df_enriched.columns

Out[24]: Index(['data_pregao', 'sigla_acao', 'nome_acao', 'preco_abertura', 'preco_max',
               'preco_min', 'preco_fechamento', 'qtd_negocios', 'vol_negocios', 'tipo',
               'variacao', 'taxa_de_retorno_simples_diaria',
               'taxa_de_retorno_medio_diaria', 'desvpad_medio_diario',
               'taxa_de_retorno_medio_anual', 'volatilidade', 'cluster', 'mm_5_dias',
               'desv_pad_5_dias', 'banda_superior_5_dias', 'banda_inferior_5_dias',
               'volatilidade_entre_bandas_5_dias', 'diferenca_fechamento_e_mm_5_dias',
               'mm_13_dias', 'desv_pad_13_dias', 'banda_superior_13_dias',
               'banda_inferior_13_dias', 'volatilidade_entre_bandas_13_dias',
               'diferenca_fechamento_e_mm_13_dias', 'mm_60_dias', 'desv_pad_60_dias',
               'banda_superior_60_dias', 'banda_inferior_60_dias',
               'volatilidade_entre_bandas_60_dias',
               'diferenca_fechamento_e_mm_60_dias', 'mm_200_dias', 'desv_pad_200_dias',
               'banda_superior_200_dias', 'banda_inferior_200_dias',
               'volatilidade_entre_bandas_200_dias',
               'diferenca_fechamento_e_mm_200_dias', 'ganho', 'perda',
               'mm_14_dias_ganho', 'mm_14_dias_perda', 'ifr_simples', 'ifr_classica',
               'ifr_simples', 'ifr_classica', 'fractal_resistencia', 'fractal_suporte',
               'linha_resistencia', 'linha_suporte', 'preco_1_dias_futuros',
               'sigla_acao_label', 'tipo_label'],
              dtype='object')
```

Figura 31: Verificação das features

Dessa forma, ao selecionarmos as features, um primeiro passo seria a remoção de valores já contidos no cálculo de indicadores. Assim evitamos overfitting e ganharemos tempo de processamento em nosso modelo, pois não trabalharemos com redundâncias.

Selecionando as melhores features: removendo informações redundantes

```
In [25]: 1 df_enriched = df_enriched.drop(['variacao', 'taxa_de_retorno_simples_diaria', 'taxa_de_retorno_medio_diaria', \
2      'desvpad_medio_diario', 'ganho', 'perda', 'ifr_simples', 'ifr_classica', 'fractal_resistencia', 'fractal_suporte', \
3      'desv_pad_5_dias', 'desv_pad_13_dias', 'desv_pad_60_dias', 'desv_pad_200_dias', 'mm_14_dias_ganho', \
4      'mm_14_dias_perda'], axis=1)
5 df_enriched.columns

Out[25]: Index(['data_pregao', 'sigla_acao', 'nome_acao', 'preco_abertura', 'preco_max',
               'preco_min', 'preco_fechamento', 'qtd_negocios', 'vol_negocios', 'tipo',
               'taxa_de_retorno_medio_anual', 'volatilidade', 'cluster', 'mm_5_dias',
               'banda_superior_5_dias', 'banda_inferior_5_dias',
               'volatilidade_entre_bandas_5_dias', 'diferenca_fechamento_e_mm_5_dias',
               'mm_13_dias', 'banda_superior_13_dias', 'banda_inferior_13_dias',
               'volatilidade_entre_bandas_13_dias',
               'diferenca_fechamento_e_mm_13_dias', 'mm_60_dias',
               'banda_superior_60_dias', 'banda_inferior_60_dias',
               'volatilidade_entre_bandas_60_dias',
               'diferenca_fechamento_e_mm_60_dias', 'mm_200_dias',
               'banda_superior_200_dias', 'banda_inferior_200_dias',
               'volatilidade_entre_bandas_200_dias',
               'diferenca_fechamento_e_mm_200_dias', 'ifr_simples', 'ifr_classica',
               'linha_resistencia', 'linha_suporte', 'preco_1_dias_futuros',
               'sigla_acao_label', 'tipo_label'],
              dtype='object')
```

Figura 32: Remoção de features redundantes

Ao separarmos features e labels, cabe também remover as variáveis textuais e substituir pelas codificadas, devido aos modelos de regressão propostos. Um outro ponto importante na seleção de features e labels é que devemos tratar nossos valores numéricos antes de usá-los através do dimensionamento de recursos. Para cada ação há grande divergência entre seus preços, seja ao longo do tempo ou seja entre um ativo e outro.

Dessa forma é preciso dimensionar esses dados para que o modelo de machine learning não acabe por interpretar que valores muito altos ou muito baixos são mais importantes que valores medianos. Optou-se então pela utilização do *MinMaxScaler*. O *MinMaxScaler* do *Sklearn* é uma técnica de pré-processamento que coloca os dados na mesma escala, dimensionando todos os recursos de dados no intervalo [0, 1] (ou então no intervalo [-1, 1] se houver valores negativos no conjunto de dados).

Separando features, labels e pré processamento

```
In [26]: 1 features = df_enriched[['sigla_acao_label', 'tipo_label', 'preco_abertura', 'preco_max', 'preco_min', 'preco_fecham
2         'qtd_negocios', 'vol_negocios', 'taxa_de_retorno_media_anual', 'volatilidade', 'cluster', 'mm_5_dias',
3         'banda_superior_5_dias', 'banda_inferior_5_dias', 'volatilidade_entre_bandas_5_dias',
4         'diferenca_fechamento_e_mm_5_dias', 'mm_13_dias', 'banda_superior_13_dias', 'banda_inferior_13_dias',
5         'volatilidade_entre_bandas_13_dias', 'diferenca_fechamento_e_mm_13_dias', 'mm_60_dias', 'banda_superior_60_dias',
6         'banda_inferior_60_dias', 'volatilidade_entre_bandas_60_dias', 'diferenca_fechamento_e_mm_60_dias', 'mm_200_dias',
7         'banda_superior_200_dias', 'banda_inferior_200_dias', 'volatilidade_entre_bandas_200_dias',
8         'diferenca_fechamento_e_mm_200_dias', 'ifr_simples', 'ifr_classica', 'linha_resistencia', 'linha_suporte']]
9
10 label_1 = df_enriched["preco_1_dias_futuros"]
11
12 scaler = MinMaxScaler().fit(features)
13 features_normalized = pd.DataFrame(scaler.transform(features), columns=features.columns)
14
15 features_normalized.sample(5)
```

Out[26]:

	sigla_acao_label	tipo_label	preco_abertura	preco_max	preco_min	preco_fechamento	qtd_negocios	vol_negocios	taxa_de_retorno_media_an
23512	0.358974	0.250000	0.102201	0.103283	0.104141	0.104091	0.002939	0.001795	0.296
58310	0.974359	0.000000	0.328691	0.329063	0.331046	0.332847	0.003667	0.006831	0.000
66143	0.871795	0.250000	0.251060	0.253803	0.239647	0.239854	0.033661	0.046701	0.585
64856	0.692308	0.250000	0.110444	0.111049	0.110594	0.111381	0.006457	0.004198	0.455
9169	0.153846	0.416667	0.004322	0.004083	0.003839	0.003564	0.000139	0.000008	0.685

5 rows × 35 columns

Figura 33: Separação de features e labels e pré processamento

Em seguida, como vamos aplicar 4 modelos diferentes para tentarmos realizar as previsões, cabe fazermos a seleção das melhores features para esses modelos. Mesmo já tendo realizado uma triagem e dimensionamento de features, é importante que não utilizarmos features demais a ponto de causar um *overfitting*, tornando o modelo excelente para os dados observados no treino e teste, mas incapaz de extrapolá-los. Mas também é preciso estar atento para não remover variáveis demais a ponto de causar *underfitting*, que é quando o modelo não se ajusta nem aos dados de treinamento.

Através da biblioteca *sklearn* temos acesso à classe *sklearn.feature_selection.SelectKBest*. Por meio dessa classe, através de funções de score, podemos organizar e selecionar as melhores features para nossos modelos. Ela atua calculando qual a importância de cada feature na determinação do label (no caso o preço de fechamento para um dia futuro), lhe atribuindo um valor de peso. Podemos então classificar essas features em ordem de importância, da maior para a menor e aplicar critérios de seleção.

Para nosso modelo de regressão linear e regressão polinomial, seguimos a recomendação da documentação do *sklearn*. Utilizamos a função *score f_regression*, que analisa a

importância da relação feature/label em tarefas de regressão. Como critério de corte na seleção das features mais importantes, optou-se por armazenar em uma variável a soma de todos os pesos obtidos pelas features e selecionar somente as features cujo valor de peso fosse maior ou igual a 0,01% do somatório de pesos.

```
In [27]: 1 # melhores features regressão linear e regressão polinomial
2 f = features_normalized
3 l=label_1
4 feature_list = f.columns.values.tolist()
5
6 k_best_features = SelectKBest(f_regression , k="all")
7 k_best_features.fit_transform(f,l)
8 k_best_features_score = k_best_features.scores_
9 raw_pairs = zip(feature_list[:],k_best_features_score)
10 ordered_pairs= list(reversed(sorted(raw_pairs, key = lambda x: x[1])))
11
12 k_best_features_final = dict(ordered_pairs[:])
13 total = sum(k_best_features_final.values())
14 k_best_features_percent = {key: value / total for key, value in k_best_features_final.items()}
15 best_features = k_best_features_final.keys()
16 print("para ---> ",l.name)
17 k_best_features_percent_above_1_percent = dict((k, v) for k, v in k_best_features_percent.items() if v >= 0.01/100)
18 best_features = k_best_features_percent_above_1_percent.keys()
19 print("features mais importantes ---> ",best_features)

para ---> preco_1_dias_futuros
features mais importantes ---> dict_keys(['preco_fechamento', 'preco_max', 'preco_min', 'preco_abertura', 'mm_5_dias',
inferior_5_dias', 'linha_suporte', 'mm_13_dias', 'banda_superior_5_dias', 'linha_resistencia', 'banda_inferior_13_dias',
superior_13_dias', 'mm_60_dias', 'banda_inferior_60_dias', 'banda_superior_60_dias', 'mm_200_dias', 'banda_superior_200_dias',
banda_inferior_200_dias', 'volatilidade_entre_bandas_200_dias', 'volatilidade_entre_bandas_60_dias', 'volatilidade_entre_bandas_13_dias', 'volatilidade_entre_bandas_5_dias'])
```

Figura 34: Seleção das melhores features – Regressão Linear e Polinomial

Processo semelhante foi utilizado na determinação das features para o modelo de *random forrest regression*, porém nesse caso em específico utilizamos a função `score_mutual_info_regression` no lugar da função `score_f_regression`. Modelos baseados em *random forrest regression* usam a combinação de várias árvores de decisão aleatórias. Cada uma treinada em um subconjunto amostral de dados, também aleatórios, diferindo inclusive nas quantidade de features usadas.

A a resposta do modelo é baseada nas médias entre esses diversos *subsets* de árvores de decisão. Dessa forma, com subsets com variáveis completamente diferentes, a informação mútua entre as variáveis tem grande importância nesse tipo de modelo. Por isso a função `score_mutual_info_regression` é mais recomendada do que a função `score_f_regression`.

```

In [28]: 1 # melhores features random forrest regressor
          2 f = features_normalized
          3 l=label_1
          4 feature_list = f.columns.values.tolist()
          5
          6 k_best_features = SelectKBest(mutual_info_regression , k="all")
          7 k_best_features.fit_transform(f,l)
          8 k_best_features_score = k_best_features.scores_
          9 raw_pairs = zip(feature_list[:],k_best_features_score)
          10 ordered_pairs= list(reversed(sorted(raw_pairs, key = lambda x: x[1])))
          11
          12 k_best_features_final = dict(ordered_pairs[:])
          13 best_features = k_best_features_final.keys()
          14 print("para ---> ",l.name)
          15 k_best_features_percent_above_1_percent = dict((k, v) for k, v in k_best_features_percent.items() if v >= 0.01/100)
          16 best_features = k_best_features_percent_above_1_percent.keys()
          17 print("features mais importantes ---> ",best_features)

para ---> preco_1_dias_futuros
features mais importantes ---> dict_keys(['preco_fechamento', 'preco_max', 'preco_min', 'preco_abertura', 'mm_5_dias',
inferior_5_dias', 'linha_suporte', 'mm_13_dias', 'banda_superior_5_dias', 'linha_resistencia', 'banda_inferior_13_dias',
superior_13_dias', 'mm_60_dias', 'banda_inferior_60_dias', 'banda_superior_60_dias', 'mm_200_dias', 'banda_superior_200_dias',
banda_inferior_200_dias', 'volatilidade_entre_bandas_200_dias', 'volatilidade_entre_bandas_60_dias', 'volatilidade_entre_bandas_5_dias'])

```

Figura 35: Seleção das melhores features – Árvres de decisão e Random Forrest

Para os três tipos de modelo pudemos constatar que as features de maior importância na determinação do preço de fechamento para um dia a frente, segundo os critérios apresentados são, da mais importante para a menos importante:

- 'preco_fechamento';
- 'preco_max';
- 'preco_min';
- 'preco_abertura';
- 'mm_5_dias';
- 'banda_inferior_5_dias';
- 'linha_suporte';
- 'mm_13_dias';
- 'banda_superior_5_dias';
- 'linha_resistencia';
- 'banda_inferior_13_dias';
- 'banda_superior_13_dias';
- 'mm_60_dias';
- 'banda_inferior_60_dias';

- 'banda_superior_60_dias';
- 'mm_200_dias';
- 'banda_superior_200_dias';
- 'banda_inferior_200_dias';
- 'volatilidade_entre_bandas_200_dias';
- 'volatilidade_entre_bandas_60_dias';
- 'volatilidade_entre_bandas_13_dias';
- 'volatilidade_entre_bandas_5_dias'

Com base nessas informações podemos iniciar a construção e aplicação de nossos modelos preditivos.

5.3 Construção de modelos

Após a seleção das features mais importantes, o próximo passo na construção dos modelos pode ser iniciado: num primeiro momento vamos separar um dataframe com as melhores features selecionadas no passo anterior. Em seguida vamos reescalar essas features usando o MinMaxScaler do sklearn. Este passo é importante pois, dada a grande variância de valores em todas as features selecionadas podemos ter distorções que são corrigidas ao se colocar os dados numa mesma escala, para processamento.

Após a criação do nosso dataframe de features, prosseguimos na criação do nosso dataframe *label*, que contém a entidade que queremos prever. No caso, um dia futuro ("preco_1_dias_futuros"). Em seguida, vamos dividir nossos dataframes de features e labels em 3 partes: treino, teste e validação. Como prática, adotou-se o padrão de que o dataframe de treino consistirá em 70% dos dados, obtidos de maneira aleatória de nossa amostra. Da sobra de 30%, para o teste separamos aleatoriamente 70%, e para a validação 30%. Dessa forma, para nosso dataframe de treino teremos 47121 registros, para o teste, 14136 registros, para a validação 6059 registros. Podemos ver abaixo

Modelos preditivos

Preparando os dados

```
In [29]: 1 df_enriched_acao = df_enriched
2
3 best_features = df_enriched_acao[['preco_fechamento', 'preco_max', 'preco_min', 'preco_abertura', 'mm_5_dias',
4                                   'banda_inferior_5_dias', 'linha_suporte', 'mm_13_dias', 'banda_superior_5_dias',
5                                   'linha_resistencia', 'banda_inferior_13_dias', 'banda_superior_13_dias', 'mm_60_dias',
6                                   'banda_inferior_60_dias', 'banda_superior_60_dias', 'mm_200_dias',
7                                   'banda_superior_200_dias', 'banda_inferior_200_dias', 'volatilidade_entre_bandas_
8                                   'volatilidade_entre_bandas_60_dias', 'volatilidade_entre_bandas_13_dias',
9                                   'volatilidade_entre_bandas_5_dias']].sort_index()
10
11 #dimensionamento das features
12 scaler = MinMaxScaler().fit(best_features)
13
14 best_features_normalized = scaler.transform(best_features)
15
16 label = df_enriched_acao["preco_1_dias_futuros"].sort_index()
17 #Criando um dataframe para aglutinar as previsões
18 df_final_prediction = df_enriched[['data_pregao', "sigla_acao", "nome_acao", "preco_min", "preco_max",
19                                   "preco_abertura", "preco_fechamento"]].sort_index()
20
21 #separando dataset em treino, teste e validação
22 X_train, X_test, y_train, y_test = train_test_split(best_features_normalized, label, test_size=0.3, random_state=42)
23 X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size=0.3, random_state=42)
24
25 print("Tamanho dos datasets de treino, teste e validação:")
26 print("X_train:", len(X_train), "X_test:", len(X_test), "X_val:", len(X_val), "\ny_train:", len(y_train), "y_test:",
27       len(y_test), "y_val:", len(y_val))
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Tamanho dos datasets de treino, teste e validação:
X_train: 47121 X_test: 14136 X_val: 6059
y_train: 47121 y_test: 14136 y_val: 6059

Figura 36: Preparação de dados para os modelos

Após a seleção das features mais importantes e a separação dos dataframes de treino, teste e validação, cabe a aplicação dos diferentes modelos preditivos já mencionados (regressão linear múltipla, *decision trees*, *random forrest* e regressão polinomial). Para os modelos, aplicaremos a mesma métrica de medição de erros e performance. Para a medida dos erros utilizaremos a *MAPE – Mean Absolute Percentage Error*. Essa medida de erro consiste em uma média do percentual de desvio em relação à observação, ou seja, a média da diferença percentual entre observações e previsões.

Em nosso caso multiplicamos por 100 o valor de *MAPE* para termos o valor em termos percentuais. Da mesma forma, fazendo $100 - MAPE$ somos capazes de obter a *porcentagem de acerto*. Como medida de performance, escolhemos utilizar o coeficiente de determinação (R^2), que representa a porcentagem de variação na resposta que é explicada pelo modelo. Com seu valor entre 0 e 1, sendo 1 um ajuste perfeito dos dados e 0 a total incapacidade do modelo de explicar os dados, também multiplicamos por 100 para termos a resposta em percentuais.

Dessa forma, o melhor modelo é para a nossa análise de previsão de valores de ações será aquele que, tanto em relação ao dataframe de teste quanto em relação ao dataframe

de validação tiver o maior R^2 possível e o menor *MAPE* possível. Isso significa que o modelo explica a maior quantidade possível de dados, com máxima proximidade aos valores.

Construindo um primeiro modelo de regressão linear, podemos observar que nosso R^2 , no teste, é superior a 99,5% enquanto na validação, é superior a 99,7%, o que é um excelente resultado, e que nosso *MAPE* fica abaixo de 1,9% tanto no teste quanto na validação.

Regressão linear

```
In [30]: 1 #Regressão linear (treino e teste)
2 print("Construindo o modelo de regressão linear")
3 lr = linear_model.LinearRegression()
4 lr.fit(X_train,y_train)
5 prediction_test = lr.predict(X_test)
6 print(f"Coefficiente de determinação do modelo (Teste) R2 Score: {r2_score(y_test,prediction_test)*100:2f}%")
7 print(f"Mean Absolute Percentage Error - MAPE: {mean_absolute_percentage_error(y_test,prediction_test)*100:2f}%")
8 print(f"Porcentagem de acerto - (100 - MAPE): {100 - mean_absolute_percentage_error(y_test,prediction_test)*100:2f}%")
9
10 #Executando a previsão com regressão linear
11 final_prediction_array_lr = lr.predict(X_val)
12 print(f"Coefficiente de determinação do modelo (Validação) R2 Score: {r2_score(y_val,final_prediction_array_lr)*100:2f}%")
13 print(f"Mean Absolute Percentage Error - MAPE: {mean_absolute_percentage_error(y_val,final_prediction_array_lr)*100:2f}%")
14 print(f"Porcentagem de acerto - (100 - MAPE): {100 - mean_absolute_percentage_error(y_val,final_prediction_array_lr)*100:2f}%")
15
16
```

Construindo o modelo de regressão linear
 Coeficiente de determinação do modelo (Teste) R2 Score: 99.558724%
 Mean Absolute Percentage Error - MAPE: 1.945117%
 Porcentagem de acerto - (100 - MAPE): 98.054883%
 Coeficiente de determinação do modelo (Validação) R2 Score: 99.753737%
 Mean Absolute Percentage Error - MAPE: 1.966622%
 Porcentagem de acerto - (100 - MAPE): 98.033378%

Figura 37: Regressão linear

Com nosso modelo de árvore de decisão, antes de criar o modelo de fato arriscando parâmetros, optou-se por realizar um processo de *tuning*. Para isso utilizamos a ferramenta GridSearchCV. O GridSearchCV é um módulo do Scikit Learn cujo objetivo é a criação de combinações sistemáticas de parâmetros e avaliação dessas combinações. Dessa forma antes de aplicarmos ao modelo podemos selecionar a combinação de parâmetros que tiver o melhor *score*.

Decision Tree

```
In [31]: 1 %%capture
2 #Decision Tree Regression (parametros)
3 parameters_dtr = {
4     'splitter': ['best', 'random'],
5     'max_features': ['auto', 'sqrt', 'log2'],
6     'random_state' : [42]
7 }
8
9 print("Determinando os melhores parâmetros")
10 grid_dtr = GridSearchCV(DecisionTreeRegressor(),parameters_dtr,verbose=1)
11 grid_dtr.fit(X_train,y_train)
12
```

Figura 38: GridSearchCV aplicado à árvores de decisão

Após construirmos um modelo utilizando os melhores parâmetros encontrados obtemos um R^2 de 99,26% no teste e 99,53% na validação, e um *MAPE* da ordem de 2,8% tanto no teste quanto na validação. Aparentemente, bons resultados, mas piores do que o modelo de regressão linear.

```
In [32]: 1 print("Construindo o modelo Decision Tree Regression com os melhores parâmetros encontrados")
2 print('best_estimator_', str(grid_dtr.best_estimator_))
3 print('best_score_', str(grid_dtr.best_score_))
4 dtr=grid_dtr.best_estimator_
5 dtr.fit(X_train,y_train)
6 prediction_test_dtr = dtr.predict(X_test)
7
8
9 print(f"Coefficiente de determinação do modelo (Teste) R2 Score: {r2_score(y_test,prediction_test_dtr)*100:2f}%")
10 print(f"Mean Absolute Percentage Error - MAPE: {mean_absolute_percentage_error(y_test,prediction_test_dtr)*100:2f}%")
11 print(f"Porcentagem de acerto - (100 - MAPE): {100 - mean_absolute_percentage_error(y_test,prediction_test_dtr)*100:2f}%")
12
13 #Executando a previsão com Decision Tree Regression
14 final_prediction_array_dtr = dtr.predict(X_val)
15 print(f"Coefficiente de determinação do modelo (Validação) R2 Score: {r2_score(y_val,final_prediction_array_dtr)*100:2f}%")
16 print(f"Mean Absolute Percentage Error - MAPE: {mean_absolute_percentage_error(y_val,final_prediction_array_dtr)*100:2f}%")
17 print(f"Porcentagem de acerto - (100 - MAPE): {100 - mean_absolute_percentage_error(y_val,final_prediction_array_dtr)*100:2f}%")
18
```

Construindo o modelo Decision Tree Regression com os melhores parâmetros encontrados
best_estimator_ DecisionTreeRegressor(max_features='auto', random_state=42)
best_score_ 0.995196844786555

C:\Users\rafael.proenca\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\tree_classes.py:306: FutureWarning: 'max_features='auto'' has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly set max_features=1.0`.
warnings.warn(

Coefficiente de determinação do modelo (Teste) R2 Score: 99.256648%
Mean Absolute Percentage Error - MAPE: 2.773927%
Porcentagem de acerto - (100 - MAPE): 97.226073%
Coefficiente de determinação do modelo (Validação) R2 Score: 99.525679%
Mean Absolute Percentage Error - MAPE: 2.756869%
Porcentagem de acerto - (100 - MAPE): 97.243131%

Figura 39: Árvore de decisão otimizada

No algoritmo *Random Forrest*, adotamos abordagem semelhante ao modelo de árvores de decisão, no que tange a seleção de melhores parâmetros, realizando previamente o *tuning* com o *GridSearchCV*.

Random Forest Regression

```
In [33]: 1 %%capture
2 #Random Forrest Regression (parametros)
3 parameters_rfr = {
4     'n_estimators': [10,20],
5     'max_features': ['sqrt','auto'],
6     'max_depth' : [10,20],
7     'random_state' : [42]
8 }
9
10 print("Determinando os melhores parâmetros")
11 grid_rfr = GridSearchCV(RandomForestRegressor(),parameters_rfr,verbose=1,scoring='r2')
12 grid_rfr.fit(X_train,y_train)
13
```

Figura 40: GridSearchCV aplicado a Random Forrest

Após construirmos um modelo utilizando os melhores parâmetros encontrados obtivemos um R^2 de 99,54% no teste e 99,73% na validação, e um *MAPE* da ordem de 2% tanto no teste quanto na validação. Resultados melhores que do modelo de árvores de decisão, mas piores do que o modelo de regressão linear.

```
In [34]: 1 print("Construindo o modelo Random Forrest Regression com os melhores parâmetros encontrados")
2 print('best_estimator_', str(grid_rfr.best_estimator_))
3 print('best_score_', str(grid_rfr.best_score_))
4 rfr=grid_rfr.best_estimator_
5 rfr.fit(X_train,y_train)
6 prediction_test_rfr = rfr.predict(X_test)
7
8 print(f"Coefficiente de determinação do modelo (Teste) R2 Score: {r2_score(y_test,prediction_test_rfr)*100:2f}%")
9 print(f"Mean Absolute Percentage Error - MAPE: {mean_absolute_percentage_error(y_test,prediction_test_rfr)*100:2f}%")
10 print(f"Porcentagem de acerto - (100 - MAPE): {100 - mean_absolute_percentage_error(y_test,prediction_test_rfr)*100:2f}%")
11
12
13 #Executando a previsão com Random Forrest Regression
14 final_prediction_array_rfr = rfr.predict(X_val)
15 print(f"Coefficiente de determinação do modelo (Validação) R2 Score: {r2_score(y_val,final_prediction_array_rfr)*100:2f}%")
16 print(f"Mean Absolute Percentage Error - MAPE: {mean_absolute_percentage_error(y_val,final_prediction_array_rfr)*100:2f}%")
17 print(f"Porcentagem de acerto - (100 - MAPE): {100 - mean_absolute_percentage_error(y_val,final_prediction_array_rfr)*100:2f}%")
```

Construindo o modelo Random Forrest Regression com os melhores parâmetros encontrados
best_estimator_ RandomForestRegressor(max_depth=10, max_features='auto', n_estimators=20, random_state=42)
best_score_ 0.997322179556275

C:\Users\rafael.proenca\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\ensemble_forest.py:416: FutureWarning: 'max_features='auto'' has been deprecated in 1.1 and will be removed in 1.3. To keep the past behaviour, explicitly _features=1.0 or remove this parameter as it is also the default value for RandomForestRegressors and ExtraTreesRegressors
warn(

Coefficiente de determinação do modelo (Teste) R2 Score: 99.543515%
Mean Absolute Percentage Error - MAPE: 1.983462%
Porcentagem de acerto - (100 - MAPE): 98.016538%
Coefficiente de determinação do modelo (Validação) R2 Score: 99.726568%
Mean Absolute Percentage Error - MAPE: 2.005044%
Porcentagem de acerto - (100 - MAPE): 97.994956%

Figura 41: Random Forrest otimizado

Por fim temos nosso modelo de regressão polinomial. Ao construir um modelo de regressão polinomial o primeiro passo é a adequação das features a um polinômio de grau n . Neste trabalho, optou-se por utilizar o grau 3 pois, após diversos testes, pudemos verificar que não havia ganhos significativos em utilizar graus maiores e havia significativo aumento do trabalho computacional. Em nosso modelo de regressão polinomial obtivemos um R^2 de 95,2% no teste e 99,5% na validação, e um *MAPE* da ordem de 2,2% no teste e 2% na validação.

Regressão polinomial

```
In [35]: 1 #Regressão polinomial (treino e teste)
2 print("Construindo o modelo de regressão polinomial (3º grau)")
3 poly_reg = PolynomialFeatures(degree=3)
4 X_poly = poly_reg.fit_transform(X_train)
5 pol_reg = linear_model.LinearRegression()
6 pol_reg.fit(X_poly, y_train)
7 prediction_test_poly = pol_reg.predict(poly_reg.fit_transform(X_test))
8
9 print(f"Coefficiente de determinação do modelo (Teste) R2 Score: {r2_score(y_test,prediction_test_poly)*100:2f}%")
10 print(f"Mean Absolute Percentage Error - MAPE: {mean_absolute_percentage_error(y_test,prediction_test_poly)*100:2f}%")
11 print(f"Porcentagem de acerto - (100 - MAPE): {100 - mean_absolute_percentage_error(y_test,prediction_test_poly)*100:2f}%")
12
13 #Executando a previsão com regressão polinomial
14 final_prediction_array_poly = pol_reg.predict(poly_reg.fit_transform(X_val))
15 print(f"Coefficiente de determinação do modelo (Validação) R2 Score: {r2_score(y_val,final_prediction_array_poly)*100:2f}%")
16 print(f"Mean Absolute Percentage Error - MAPE: {mean_absolute_percentage_error(y_val,final_prediction_array_poly)*100:2f}%")
17 print(f"Porcentagem de acerto - (100 - MAPE): {100 - mean_absolute_percentage_error(y_val,final_prediction_array_poly)*100:2f}%")
18
```

Construindo o modelo de regressão polinomial (3º grau)
 Coeficiente de determinação do modelo (Teste) R2 Score: 95.183891%
 Mean Absolute Percentage Error - MAPE: 2.193955%
 Porcentagem de acerto - (100 - MAPE): 97.806045%
 Coeficiente de determinação do modelo (Validação) R2 Score: 99.495350%
 Mean Absolute Percentage Error - MAPE: 2.048602%
 Porcentagem de acerto - (100 - MAPE): 97.951398%

Figura 42: Regressão polinomial de terceiro grau

6. Interpretação dos Resultados

6.1 Introdução

Anteriormente apresentamos os diversos algoritmos de machine learning utilizados neste trabalho. Nesta sessão debateremos os resultados de cada um deles no nível prático, identificando aquele que se comportou melhor para o fim proposto.

Como pudemos ver anteriormente, durante a fase de preparação de dados para a construção de nossos modelos de machine learning, elaboramos dataframes de treino, teste e validação (tanto para as features quanto para as labels). Estes dataframes únicos são de suma importância e foram elaborados para serem usados repetidamente nos diversos algoritmos de machine learning propostos, a fim de que todos eles passassem pelos mesmos padrões e elaborem seus cálculos preditivos sobre as mesmas massas de dados. Dessa maneira evitamos a situação de que um modelo possa performar melhor que outro devido a um fator “sorte” na seleção dos dataframes de treino, teste e validação.

Um outro ponto de suma importância é o fato de que para os modelos do tipo árvores de decisão e modelo *random forrest* fizemos questão de selecionar os melhores parâmetros e de dar ao algoritmo algumas opções, minimizando qualquer problema que pudesse ser relacionado, seja para melhor ou para pior funcionamento do modelo, à seleção de parâmetros. Fizemos questão de nesses casos escolhermos justamente os melhores parâmetros em cada um dos modelos.

6.2 Apresentação de resultados

Durante a construção dos modelos de machine learning foram levantadas métrica de medição de erros e performance, idênticas para todos os modelos. Podemos elaborar então, com base nessas métricas, uma matriz de resultados dos modelos.

Modelo	Observação	R ²	MAPE
Regressão Linear	-	Teste = 99.558724% Validação = 99.753737%	Teste = 1.945117% Validação = 1.966622%
Árvore de decisão	max_features='auto' random_state=42	Teste = 99.256648% Validação = 99.525679%	Teste = 2.773927% Validação = 2.756869%
Random Forrest	max_depth=10 max_features='auto' n_estimators=20 random_state=42	Teste = 99.543515% Validação = 99.726568%	Teste = 1.983462% Validação = 2.005044%
Regressão Polinomial	3º grau	Teste = 95.183891% Validação = 95.183891%	Teste = 2.193955% Validação = 2.048602%

Com base em nossa matriz de resultados podemos verificar que a regressão linear obteve o melhor resultado com relação às métricas de erro (menor MAPE) e o melhor resultado em relação à performance (maior R²). Em seguida, *random forrest* apresenta resultados muito interessantes, ficando em segundo lugar. Por fim, temos a tanto a regressão polinomial quanto as árvores de decisão. Embora o R² das árvores de decisão seja maior que da Regressão Polinomial, na Regressão Polinomial temos um MAPE menor. Ou seja, enquanto as árvores de decisão explicam uma variedade maior de respostas, ela o faz com menor porcentagem de acerto, quando comparado a regressão polinomial.

Para fins meramente comparativos e didáticos, foi elaborado um dataframe de validações. Foi criado um dataframe chamado `df_prediction`, visando agregar os valores de `y_val` (nosso valor de preços de um dia futuro) e os valores de previsão de cada modelo. Em seguida, recuperamos o dataframe `df_final_prediction`, que contém dados de sigla de ações, data do pregão, preços de abertura, fechamento, máxima e mínima e fazemos um join deste dataframe com o `df_prediction`, gerando um novo `df_final_prediction` com os valores de previsão agregados.

Dataframe de validações

```
In [36]: 1 #Montando um dataframe com as previsões
2 df_prediction = pd.DataFrame(y_val)
3 df_prediction[f"{label.name}_previsao_lr"] = final_prediction_array_lr.tolist()
4 df_prediction[f"{label.name}_previsao_dtr"] = final_prediction_array_dtr.tolist()
5 df_prediction[f"{label.name}_previsao_rfr"] = final_prediction_array_rfr.tolist()
6 df_prediction[f"{label.name}_previsao_poly"] = final_prediction_array_poly.tolist()
7
8
9 #Fazendo inner join (merge o pandas) entre a previsão e as datas
10 df_final_prediction = pd.merge(df_final_prediction.sort_index(), df_prediction[[f"{label.name}",\
11 f"{label.name}_previsao_lr",f"{label.name}_previsao_dtr",\
12 f"{label.name}_previsao_rfr",f"{label.name}_previsao_poly"]]\
13 .sort_index(), left_index=True, right_index=True)
14
15 df_final_prediction = df_final_prediction.sort_index()
16
17
```

Figura 43: Dataframe de validações

Dessa forma podemos obter em um único dataframe as siglas de ações, datas dos pregões, preços de fechamento para o dia seguinte e preços de fechamento previstos para todos os modelos criados.

```
In [37]: 1 df_final_prediction[["preco_1_dias_futuros","preco_1_dias_futuros_previsao_lr",\
2 "preco_1_dias_futuros_previsao_dtr", "preco_1_dias_futuros_previsao_rfr",\
3 "preco_1_dias_futuros_previsao_poly"]].sample(10)
```

Out[37]:

preco_1_dias_futuros	preco_1_dias_futuros_previsao_lr	preco_1_dias_futuros_previsao_dtr	preco_1_dias_futuros_previsao_rfr	preco_1_dias_futuros_previsao_poly
27.68	27.454440	27.80	27.447398	27.693407
44.27	43.171953	43.78	43.303761	43.497980
35.23	35.135308	33.89	35.220635	34.813955
53.41	49.447878	43.47	49.738448	49.888060
20.94	20.823376	21.37	20.889117	20.773609
27.98	28.979080	28.72	28.949914	29.071108
10.97	10.955344	11.03	10.909286	10.991243
10.10	10.029001	10.14	10.001632	10.003183
37.23	36.740221	36.03	36.458551	36.589676
49.61	49.700954	50.69	49.874583	50.060080

Figura 44: Dataframe de validações

Podemos ainda visualizar esses resultados em termos gráficos. Seleccionamos 4 ações para podermos visualizar graficamente a proximidade dos resultados e das previsões. São elas Alpargatas, Petrobras, Vale e Vivo. Nosso preço de 1 dia futuro original corresponde às linhas pretas. A regressão linear corresponde à linha verde, as árvores de decisão às linhas amarelas, *random forrest* às linhas vermelhas e regressão polinomial às linhas azuis.

```

In [38]: 1 #plotando graficamente
2 sigla = "ALPA4"
3 df_final_prediction_acao = df_final_prediction[df_final_prediction["sigla_acao"]==sigla].tail(50)
4 f = "preco_1_dias_futuros"
5 plt.figure(figsize=(20,5))
6 plt.title(f"Peço previsto vs Preço fechamento \n {f} {sigla}")
7
8 plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_acao[f"{f}_previsao_dtr"],\
9          label="Preço Previsto R.F.R.", color="yellow",marker="d")
10 plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_acao[f"{f}_previsao_rfr"],\
11          label="Preço Previsto R.F.R.", color="red",marker="^")
12 plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_acao[f"{f}_previsao_poly"],\
13          label="Preço Previsto R.P.", color="blue",marker="s")
14 plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_acao[f"{f}_previsao_lr"],\
15          label="Preço Previsto R.L.", color="green",marker="o")
16 plt.plot(df_final_prediction_acao["data_pregao"], df_final_prediction_acao[f"{f}"],label="Preço Real", \
17          color="black",marker="o")
18 #get current axes
19 ax = plt.gca()
20 #hide x-axis
21 ax.get_xaxis().set_visible(False)
22 plt.xlabel("Data Pregão")
23 plt.ylabel("Preço de Fechamento")
24
Out[38]: Text(0, 0.5, 'Preço de Fechamento')

```

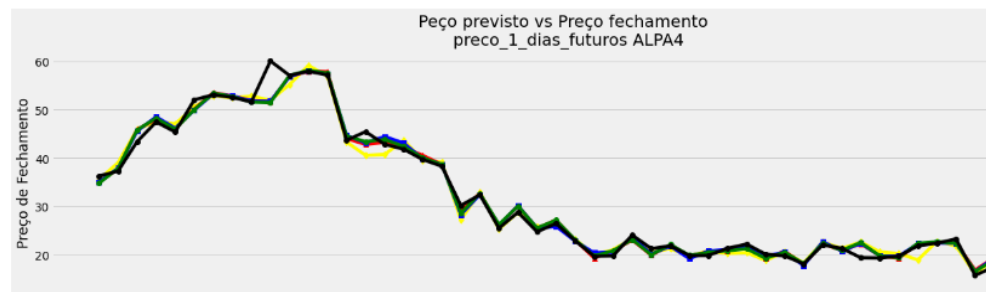


Figura 45: Comparativo ALP4

```

In [39]: 1 #plotando graficamente
2 sigla = "PETR3"
3 df_final_prediction_acao = df_final_prediction[df_final_prediction["sigla_acao"]==sigla].tail(50)
4 f = "preco_1_dias_futuros"
5 plt.figure(figsize=(20,5))
6 plt.title(f"Peço previsto vs Preço fechamento \n {f} {sigla}")
7 plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_acao[f"{f}_previsao_dtr"],\
8          label="Preço Previsto R.F.R.", color="yellow",marker="d")
9 plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_acao[f"{f}_previsao_rfr"],\
10          label="Preço Previsto R.F.R.", color="red",marker="^")
11 plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_acao[f"{f}_previsao_poly"],\
12          label="Preço Previsto R.P.", color="blue",marker="s")
13 plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_acao[f"{f}_previsao_lr"],\
14          label="Preço Previsto R.L.", color="green",marker="o")
15 plt.plot(df_final_prediction_acao["data_pregao"], df_final_prediction_acao[f"{f}"],label="Preço Real", \
16          color="black",marker="o")
17 #get current axes
18 ax = plt.gca()
19 #hide x-axis
20 ax.get_xaxis().set_visible(False)
21 plt.xlabel("Data Pregão")
22 plt.ylabel("Preço de Fechamento")
23
Out[39]: Text(0, 0.5, 'Preço de Fechamento')

```

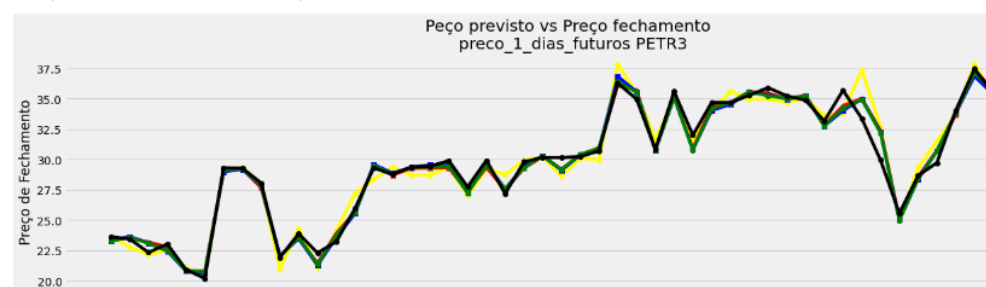


Figura 46: Comparativo PETR3

```

In [40]: 1 #plotando graficamente
2 sigla = "VALE3"
3 df_final_prediction_acao = df_final_prediction[df_final_prediction["sigla_acao"]==sigla].tail(50)
4 f = "preco_1_dias_futuros"
5 plt.figure(figsize=(20,5))
6 plt.title(f"Peço previsto vs Preço fechamento \n {f} {sigla}")
7 plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_acao[f"{f}_previsao_dtr"],\
8          label="Preço Previsto R.F.R.", color="yellow",marker="d")
9 plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_acao[f"{f}_previsao_rfr"],\
10          label="Preço Previsto R.F.R.", color="red",marker="^")
11 plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_acao[f"{f}_previsao_poly"],\
12          label="Preço Previsto R.P.", color="blue",marker="s")
13 plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_acao[f"{f}_previsao_lr"],\
14          label="Preço Previsto R.L.", color="green",marker="o")
15 plt.plot(df_final_prediction_acao["data_pregao"], df_final_prediction_acao[f"{f}"],label="Preço Real", \
16          color="black",marker="o")
17 #get current axes
18 ax = plt.gca()
19 #hide x-axis
20 ax.get_xaxis().set_visible(False)
21 plt.xlabel("Data Pregão")
22 plt.ylabel("Preço de Fechamento")

```

Out[40]: Text(0, 0.5, 'Preço de Fechamento')

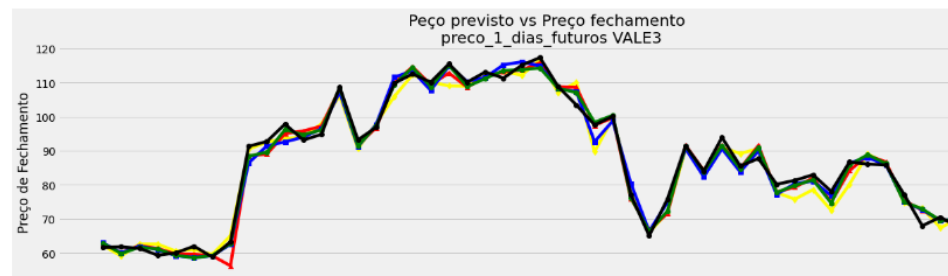


Figura 47: Comparativo VALE3

```

In [41]: 1 #plotando graficamente
2 sigla = "VIVT3"
3 df_final_prediction_acao = df_final_prediction[df_final_prediction["sigla_acao"]==sigla].tail(50)
4 f = "preco_1_dias_futuros"
5 plt.figure(figsize=(20,5))
6 plt.title(f"Peço previsto vs Preço fechamento \n {f} {sigla}")
7 plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_acao[f"{f}_previsao_dtr"],\
8          label="Preço Previsto R.F.R.", color="yellow",marker="d")
9 plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_acao[f"{f}_previsao_rfr"],\
10          label="Preço Previsto R.F.R.", color="red",marker="^")
11 plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_acao[f"{f}_previsao_poly"],\
12          label="Preço Previsto R.P.", color="blue",marker="s")
13 plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_acao[f"{f}_previsao_lr"],\
14          label="Preço Previsto R.L.", color="green",marker="o")
15 plt.plot(df_final_prediction_acao["data_pregao"], df_final_prediction_acao[f"{f}"],label="Preço Real", \
16          color="black",marker="o")
17 #get current axes
18 ax = plt.gca()
19 #hide x-axis
20 ax.get_xaxis().set_visible(False)
21 plt.xlabel("Data Pregão")
22 plt.ylabel("Preço de Fechamento")

```

Out[41]: Text(0, 0.5, 'Preço de Fechamento')

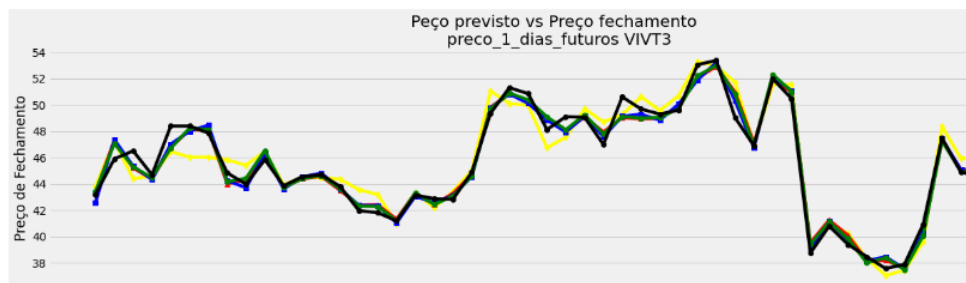


Figura 48: Comparativo VIVT3

Observamos que, nos 4 casos, a linha preta que representa o verdadeiro preço de um dia futuro é sempre acompanhada pelas demais. Nos casos de desvio há uma tendência de correção das demais linhas ao valor da linha preta. Isso significa que nos 4 casos há um acompanhamento do valor real. Que os 4 modelos se comportam de maneira acurada, com variações em sua precisão. Também é notável, visualmente, que as árvores de decisão se afastam mais do valor original para o dia seguinte, justamente o que é traduzido por seu MAPE maior.

Dessa forma, com base nas informações obtidas por meio das métricas geradas e da análise gráfica, podemos estabelecer que a regressão linear simples, quando aplicada em um dataframe enriquecido com os indicadores financeiros que geramos ao longo deste trabalho, proporcionou melhores resultados no que tange a previsão de preços de ação para um dia futuro. Podemos concluir também que é possível a construção de um modelo de previsão de preços mais genérico, direcionado a um conjunto de ações, e não só a uma ação específica, usando os indicadores financeiros adequados.

7. Apresentação dos Resultados

O presente trabalho consiste na construção de diferentes modelos de *machine learning* visando a previsão do preço de fechamento de uma gama de ações para um dia futuro. A captura de dados ocorreu por meio de download do próprio site da bovespa, sendo coletados dados históricos de todas as cotações ao longo de oito anos. Foram realizadas etapas de seleção das ações que estão somente no lote padrão, seleção das ações que estão hoje no índice ibovespa e formatação de dados. O dataframe obtido então foi salvo em um arquivo csv em "`cota_hist\cotacoes.csv.zip`".

Durante a análise e exploração, os dados foram importados em um outro notebook, enriquecidos com indicadores econômicos dos mais diversos e trabalhados no sentido de tratar possíveis discrepâncias e outliers. Com isso obtivemos um dataframe com 40 ações diferentes, enriquecido com os dados de indicadores de análise técnica. Por fim selecionamos as melhores features e construímos nossos modelos. Nos quatro modelos construídos (regressão linear, árvores de decisão, *random forest*, regressão polinomial) pudemos verificar que há uma aceitável taxa de erros e uma boa performance em termos de previsão de preço para um dia futuro.

Dos quatro modelos a regressão linear performou melhor que os demais em ambos os quesitos. Conseguimos obter um erro percentual médio absoluto da ordem de 1.97%, juntamente com um coeficiente de determinação da ordem de 99.75% na validação de dados. Tudo isso levando-se em consideração que foi construído um modelo genérico, capaz de expressar esse comportamento para 40 ações distintas.

Desta maneira conclui-se que é possível realmente a construção de um modelo genérico para previsão de preços de fechamento do dia seguinte, que abarque diversas ações de uma única vez. Isso, na análise do autor do trabalho, se deve ao fato de se buscar o enriquecimento dos dataframes com indicadores de análise técnica e médias móveis que, intrinsecamente, carregam informações do histórico das próprias ações.

Cabe ressaltar que o presente trabalho não visa substituir a análise crítica na escolha de movimentos de compra ou venda de ações, mas servir como mais uma ferramenta em possíveis decisões. Portanto toda decisão no que tange compra ou venda de ações deve partir de uma decisão essencialmente pessoal.

Para trabalhos futuros, uma abordagem interessante pode ser a comparação de modelos de regressão com o resultado obtido por alguns modelos de redes neurais. Pode-se também tentar verificar a possibilidade de construção de modelos semelhantes para ações fora do índice ibovespa, ou tentar previsões para prazos futuros maiores que 1 dia.

Por fim acreditamos que o trabalho ficaria bem resumido em um modelo de Canvas, como o proposto por Massadani.

<p>1-Problem Statement</p> <p>Estamos tentando propor um algoritmo que possibilite de alguma maneira a previsão de preços de ações do índice ibovespa.</p>	<p>2-Outcomes/Predictions</p> <p>Temos como foco a previsão de preços de fechamento de diversas ações para um dia futuro.</p>	<p>3-Data Acquisition</p> <p>A origem de nossos dados está no histórico de cotações disponível no site da BOVESPA.</p>
<p>4-Modeling</p> <p>Construção de modelos de regressão linear, árvores de decisão, <i>random forrest</i> e regressão polinomial.</p>	<p>5-Model Evaluation</p> <p>Os modelos são avaliados em um dataframe específico de validação, diferente do usado em treino e teste. As métricas usadas são o erro médio percentual absoluto para erro e coeficiente de determinação para performance.</p>	<p>6-Data Preparation</p> <p>Tratamento de nulos e infinitos, remoção de outliers e seleção de carteira baseada no índice ibovespa, além da seleção de features.</p>

O fluxo adequado para este projeto seria então:

1 => 3 => 6 => 4 => 2 => 5

8. Links

Link para o vídeo:

<https://youtu.be/b6aYyF3Fae8>

Link para o repositório:

[https://drive.google.com/drive/folders/1nEfKLd6VCy732taEjVoiTuRYcWCOTpHd?
usp=share_link](https://drive.google.com/drive/folders/1nEfKLd6VCy732taEjVoiTuRYcWCOTpHd?usp=share_link)

REFERÊNCIAS

BRUCE, Peter; BRUCE, Andrew. **Estatística Prática para Cientistas de Dados: 50 Conceitos Essenciais**. Rio de Janeiro: Alta Books, 2019.

GÉRON, Aurélien. **Mãos a Obra Aprendizado de Máquina com Scikit-Learn, Keras e Tensorflow - 2ª Edição atualizada com a Tensorflow2: Conceitos, Ferramentas e Técnicas para a Construção de Sistemas Inteligentes**. Rio de Janeiro: Alta Books, 2021.

VANDERPLAS, Jake. **Python Data Science Handbook: Essential Tools for Working with Data**. USA: O'Reilly Media Inc, 2017.

Cotações históricas IBOVESPA. Disponível em: <https://www.b3.com.br/pt_br/market-data-e-indices/servicos-de-dados/market-data/historico/mercado-a-vista/cotacoes-historicas/>. Acesso em: 05/01/2023.

Índice IBOVESPA. Disponível em: <https://www.b3.com.br/pt_br/market-data-e-indices/indices/indices-amplos/indice-ibovespa-ibovespa-composicao-da-carteira.htm>. Acesso em: 05/01/2023.

Linhas de suporte e resistência. Disponível em: <<https://towardsdatascience.com/detection-of-price-support-and-resistance-levels-in-python-baedc44c34c9>>. Acesso em: 15/01/2023.

Bandas de Bollinger. Disponível em: <https://pt.wikipedia.org/wiki/An%C3%A1lise_de_Bollinger>. Acesso em: 15/01/2023.

Random Forrest. Disponível em: <https://en.wikipedia.org/wiki/Random_forest>. Acesso em: 15/01/2023.

Random Forrest Regression - Implementação. Disponível em: <<https://medium.com/@theclickreader/random-forest-regression-explained-with-implementation-in-python-3dad88caf165>>. Acesso em: 15/01/2023.

Árvores de decisão. Disponível em: <<https://www.mastersindatascience.org/learning/machine-learning-algorithms/decision-tree/>>. Acesso em: 15/01/2023.

Cálculo de Indicadores em Python. Disponível em: <<https://quantbrasil.com.br/como-calculas-bandas-de-bollinger-em-python/>>, <<https://quantbrasil.com.br/aprenda-a>>

calcular-o-ifr-indice-de-forca-relativa/>, <<https://quantbrasil.com.br/como-identificar-linhas-de-suporte-e-resistencia-utilizando-python/>>. Acesso em: 15/01/2023.

APÊNDICE

Programação/Scripts

ETL_BOVESPA :

```
#!/usr/bin/env python

# coding: utf-8
# # Importando as libs
# In[1]:
import pandas as pd
import glob
import os
import numpy as np
# # Importando os dados
# In[ ]:
#colunas necessárias (com base no layout) : data do pregão,código do
bdi,sigla da ação,nome da ação, preço de abertura,
#preço máximo, preço mínimo, preço de fechamento, quantidade de títulos
negociados, volume de títulos negociados
nomes_colunas = ["data_pregao", "cod_bdi", "sigla_acao", "nome_acao",
"preco_abertura", "preco_max",
                "preco_min", "preco_fechamento", "qtd_negocios", "vol_negocios"
                ]
posicoes_colunas = [(2,10), (10,12), (12,24), (27,39), (56,69), (69,82),
(82,95), (108,121), (152,170), (170,188)]
cwd = os.getcwd()
todos_os_arquivos = glob.glob(cwd + "\\cota_hist\COTAHIST*.zip")
df_cotacoes =pd.concat((pd.read_fwf(1,
                compression='zip', colspecs = posicoes_colunas, names = no-
mes_colunas, skiprows =1, skipfooter=1)
                for l in todos_os_arquivos), ignore_index=True)
df_cotacoes.sample(5)
# In[ ]:
df_cotacoes["sigla_acao"].unique()
# # Selecionando os dados - lote padrão e índice Ibovespa
# In[ ]:
#somente ações no lote padrão (cod_bdi=2)
df_cotacoes = df_cotacoes[df_cotacoes["cod_bdi"]==2]
df_cotacoes = df_cotacoes.drop(["cod_bdi"],1)
df_cotacoes.sample(5)
# In[ ]:
#pegando somente as siglas das ações do índice Ibovespa
df_ibov_index=pd.read_csv(".\\cota_hist\\ibov.csv",sep=";")
#filtrando somente as cotações do índice Ibovespa
```

```

df_cotacoes = pd.merge(df_cotacoes, df_ibov_index,
left_on="sigla_acao", right_on="codigo").drop(["codigo", "acao", "qtde_teori-
ca", "part_percentual"], axis=1)
df_cotacoes.sample(5)
# # Ajustando a formatação
# In[ ]:
#ajuste de data
df_cotacoes["data_pregao"] = pd.to_datetime(df_cotacoes["data_pregao"],
format = "%Y%m%d")
df_cotacoes.sample(5)
# In[ ]:
#ajustes preços duas casas decimais
df_cotacoes["preco_abertura"] = (df_cotacoes["preco_abertura"] /
100).astype(float)
df_cotacoes["preco_max"] = (df_cotacoes["preco_max"] / 100).astype(float)
df_cotacoes["preco_min"] = (df_cotacoes["preco_min"] / 100).astype(float)
df_cotacoes["preco_fechamento"] = (df_cotacoes["preco_fechamento"] /
100).astype(float)
df_cotacoes.sample(5)
# # Lidando com duplicidades e ausências
# In[ ]:
#removendo possíveis duplicatas
df_cotacoes = df_cotacoes.drop_duplicates()
# In[ ]:
#checando se existem dados vazios ou infinitos
data_to_drop = df_cotacoes.isin([np.inf, -np.inf, np.nan])

df_cotacoes[data_to_drop == True].count()
# # Dataframe tratado
# In[ ]:
df_cotacoes.describe()
# In[ ]:
df_cotacoes.to_csv(cwd + "\cota_hist\cotacoes.csv.zip", index_label =
"index", compression="zip")
# In[ ]:
df_cotacoes.sample(5)
# In[ ]:
df_cotacoes["sigla_acao"].unique()
# In[ ]:

```

Seleção e previsão dos valores de ações:

```
#!/usr/bin/env python

# coding: utf-8
# # Importando o dataframe e as libs
# In[1]:
import pandas as pd
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error,
r2_score, mean_absolute_percentage_error, mean_absolute_error
from sklearn.feature_selection import *
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import *
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.tree import DecisionTreeRegressor
from kneed import KneeLocator
import seaborn as sns
from sklearn.svm import *
from sklearn.ensemble import *
import numpy as np
from datetime import datetime, date, timedelta
from sklearn import metrics
import glob
import os
import tensorflow as tf
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
cwd = os.getcwd()
get_ipython().run_line_magic('matplotlib', 'inline')
# # Verificando o dataframe
# In[2]:
# importando o dataframe
df = pd.read_csv("../cota_hist/cotacoes.csv.zip", compression='zip', index_col="index")
df.sample(5)
# In[3]:
df.describe()
# # Retorno diário
# In[4]:
get_ipython().run_cell_magic('capture', '', '#dataframe de ações\
narray_siglas = df["sigla_acao"].unique()\n\nndf_enriquecido = pd.DataFrame()\n
nfor i in array_siglas: \n    #ordenando por index\n    df_sigla = df[df["si-
```

```

gla_acao"] == i].sort_values(["data_pregao"])\n    #retorno\n    df_sigla["va-
riacao"]=df_sigla["preco_fechamento"].diff()\n    #taxas de retorno\n
df_sigla["taxa_de_retorno_simples_diaria"]=df_sigla["variacao"]/df_sigla["pre-
co_fechamento"].shift(1) \n    df_enriquecido = df_enriquecido.append(df_si-
gla)')

# In[5]:
df_enriquecido[["data_pregao","sigla_acao","preco_fechamento","variacao"
,"taxa_de_retorno_simples_diaria"]].sample(5)

# In[6]:
df_enriquecido["sigla_acao"].unique()
# # Retorno médio anual e volatilidade
# In[7]:
df_estatistico = df_enriquecido.groupby('sigla_acao').agg(
    preco_fechamento_min=pd.NamedAgg(column="preco_fechamento", agg-
func="min"),
    preco_fechamento_medio=pd.NamedAgg(column="preco_fechamento", agg-
func="mean"),
    preco_fechamento_inicial=pd.NamedAgg(column="preco_fechamento", agg-
func="first"),
    preco_fechamento_final=pd.NamedAgg(column="preco_fechamento", agg-
func="last"),
    preco_fechamento_variancia=pd.NamedAgg(column="preco_fechamento", agg-
func="var"),
    taxa_de_retorno_media_diaria=pd.NamedAgg(column="taxa_de_retorno_sim-
ples_diaria", aggfunc='mean'),
    desvpad_medio_diario = pd.NamedAgg(column="taxa_de_retorno_simples_dia-
ria", aggfunc='std'))
df_estatistico["taxa_de_retorno_media_anual"]=df_estatistico["taxa_de_re-
torno_media_diaria"]*250 #pregões anuais
df_estatistico["taxa_de_retorno_media_anual"] = df_estatisti-
co["taxa_de_retorno_media_anual"]
df_estatistico["volatilidade"]=df_estatistico["desvpad_medio_diario"]*25
0**0.5 #a volatilidade é o desvio padrão anualizado

df_estatistico["percentual_retorno"] = 100*(df_estatistico["preco_fecha-
mento_final"] - df_estatistico["preco_fechamento_inicial"])/df_estatisti-
co["preco_fechamento_inicial"]
df_estatistico= df_estatistico.dropna()
df_estatistico = df_estatistico[df_estatistico["percentual_retorno"]>0]
df_estatistico = df_estatistico[df_estatisti-
co["taxa_de_retorno_media_anual"]>0]
df_estatistico=df_estatistico[["taxa_de_retorno_media_diaria","desvpad_m
edio_diario","taxa_de_retorno_media_anual","volatilidade"]]
df_estatistico = df_estatistico.round(6)
df_estatistico.sample(5)

# In[8]:

```

```

df_estatistico[df_estatistico.columns[0]].count()
# # Vizualizando os dados por volatilidade e taxa de retorno media anual
# In[9]:
#cada ponto em azul significa uma ação diferente.
plt.subplots(figsize=(15, 5))
plt.subplot(1, 2, 1)
sns.scatterplot(data=df_estatistico, x= "volatilidade", y="taxa_de_re-
torno_media_anual")
plt.subplot(1, 2, 2)
sns.boxplot(df_estatistico[['volatilidade',"taxa_de_retorno_media_anual"
]])
plt.tight_layout()
plt.show()
# # Removendo outliers - IQR
# In[10]:
#Removendo outliers
def remove_outlier(df_in, col_names):
    df_out=df_in
    for col_name in col_names :
        q1 = df_in[col_name].quantile(0.25)
        q3 = df_in[col_name].quantile(0.75)
        iqr = q3-q1 #Interquartile range
        fence_low  = q1-1.5*iqr
        fence_high = q3+1.5*iqr
        df_out = df_out.loc[(df_out[col_name] > fence_low) &
(df_out[col_name] < fence_high)]

    return df_out
df_estatistico = remove_outlier(df_estatistico,
["taxa_de_retorno_media_anual","volatilidade"])
sns.scatterplot(data=df_estatistico, x= "volatilidade", y="taxa_de_re-
torno_media_anual")
plt.show()
# # Análise de clusters - definindo o número de clusters ideal
# In[11]:
def elbow_definition(features):
    kmeans_kwargs = {
        "init": "random",
        "n_init": 10,
        "max_iter": 600,
        "random_state": 42,
    }
    scaler = StandardScaler()
    scaled_features = scaler.fit_transform(features)
    sse = []
    for k in range(1, 11):

```

```

    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(scaled_features)
    sse.append(kmeans.inertia_)
plt.style.use("fivethirtyeight")
plt.plot(range(1, 11), sse)
plt.xticks(range(1, 11))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.show()
kl = KneeLocator(
    range(1, 11), sse, curve="convex", direction="decreasing"
)
return kl.elbow
elbow1 = elbow_definition(df_estatistico[["taxa_de_retorno_medio_anual", "volatilidade"]])
print("elbows", elbow1)
# # Clusterizando pelo número ótimo de clusters (3)
# In[12]:
kmeans_kwargs = {
    "init": "random",
    "n_init": 10,
    "max_iter": 600,
    "random_state": 42,
}
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df_estatistico[["taxa_de_retorno_medio_anual", "volatilidade"]])
kmeans = KMeans(n_clusters=elbow1, **kmeans_kwargs)
kmeans.fit(scaled_features)
y_kmeans = kmeans.predict(scaled_features)
sns.scatterplot(data=df_estatistico, x="volatilidade", y="taxa_de_retorno_medio_anual", hue=y_kmeans, palette="deep", sizes=(400, 400))
plt.show()
# # Agregando indicadores e cluster ao dataframe de pregões diários
# In[13]:
# criando uma coluna com o valor do cluster
df_estatistico["cluster"] = y_kmeans
df_estatistico.sample(5)
# In[14]:
df_estatistico[df_estatistico.columns[0]].count()
# In[15]:
# Enriquecendo o dataframe de cotações diárias com os : trazendo para o dataframe via inner join (merge)
df = pd.merge(df_enriquecido, df_estatistico, left_on="sigla_acao", right_index=True)
df.sample(5)

```

```

# In[16]:
#colunas
df.columns
# In[17]:
#array de ações
array_siglas = df["sigla_acao"].unique()
array_siglas
# # Adicionando indicadores de análise técnica
# In[18]:
get_ipython().run_cell_magic('capture', '', 'df_enriched = pd.Data-
Frame()\nfor i in array_siglas:\n    #ordenando por data\n    df_sigla =
df[df["sigla_acao"] == i].sort_values(["data_pregao"])\n    \n    #Bandas de
Bollinger\n    for n in [5,13,60,200]:\n
df_sigla[f"mm_{n}_dias"]=df_sigla["preco_fechamento"].rolling(n).mean()#meio\n
df_sigla[f"desv_pad_{n}_dias"]=df_sigla["preco_fechamento"].rolling(n).std()#d
esvio padrao\n        k = 2 #padrão\n
df_sigla[f"banda_superior_{n}_dias"]=df_sigla[f"mm_{n}_dias"] +
k*df_sigla[f"desv_pad_{n}_dias"] #banda superior\n        df_sigla[f"banda_in-
ferior_{n}_dias"]=df_sigla[f"mm_{n}_dias"] - k*df_sigla[f"desv_pad_{n}_dias"]
#banda inferior\n        df_sigla[f"volatilidade_entre_bandas_{n}_dias"] =
2*k*df_sigla[f"desv_pad_{n}_dias"] #diferença entre bandas\n
df_sigla[f"diferenca_fechamento_e_mm_{n}_dias"] = df_sigla["preco_fechamento"]
- df_sigla[f"mm_{n}_dias"] \n        \n        \n    #IFR – Índice de Força
Relativa \n    df_sigla["ganho"] = np.where(df_sigla['\variacao\'] > 0,
df_sigla['\variacao\'], 0) \n    df_sigla["perda"] = np.where(df_sigla['\vari-
acao\'] < 0, df_sigla['\variacao\'].abs(), 0) \n    #De acordo com J. Welles
Wilder, criador do indicador, os parâmetros recomendados para análise são um
período de 14 dias.\n    #Valores acima de 70 sugerem um ativo sobrecomprado e
abaixo de 30 indicam um ativo sobrevendido.\n    n=14\n
df_sigla[f"mm_{n}_dias_ganho"]=df_sigla["ganho"].rolling(n).mean()\n    df_si-
gla[f"mm_{n}_dias_perda"]=df_sigla["perda"].abs().rolling(n).mean() #em abso-
lutos\n    df_sigla["fr_simples"] =df_sigla[f"mm_{n}_dias_ganho"]/
df_sigla[f"mm_{n}_dias_perda"]\n    df_sigla["fr_classica"] =
(df_sigla[f"mm_{n}_dias_ganho"].shift(1)*(n-1)+df_sigla["ganho"])/\\\n
(df_sigla[f"mm_{n}_dias_perda"].shift(1)*(n-1)+df_sigla["perda"])\n    df_si-
gla["ifr_simples"] = 100 - (100/(1+df_sigla["fr_simples"]))\n
df_sigla["ifr_classica"] = 100 - (100/(1+df_sigla["fr_classica"])))\n    \n
#Cálculo de suporte/resistência > num padrão de 5 candles, o candle central
apresenta máxima ou mínima \n    #maior que os dois candles ao seu extremo\n
df_sigla['\fractal_resistencia\'] = np.select([(df_sigla['\preco_max\'] >
df_sigla['\preco_max\'].shift(2)) &\\\n        (df_sigla['\preco_max\'] > df_si-
gla['\preco_max\'].shift(1)) & (df_sigla['\preco_max\'] >
df_sigla['\preco_max\'].shift(-1)) & \\\n        (df_sigla['\preco_max\'] >
df_sigla['\preco_max\'].shift(-2))], [df_sigla['\preco_max\']])\n    df_si-
gla['\fractal_suporte\'] = np.select([(df_sigla['\preco_min\'] <
df_sigla['\preco_min\'].shift(2)) &\\\n        (df_sigla['\preco_min\'] < df_si-
```



```

glgla[['preco_min']].shift(1)) & (df_sigla[['preco_min']] <
df_sigla[['preco_min']].shift(-1)) & \\\n      (df_sigla[['preco_min']] <
df_sigla[['preco_min']].shift(-2))), [df_sigla[['preco_min']]])\n      #substi-
tuei 0 pelo valor anterior não nulo\n      df_sigla[['fractal_resistencia']] =
df_sigla[['fractal_resistencia']].replace(to_replace=0, method='ffill')\n
df_sigla[['fractal_suporte']] = df_sigla[['fractal_suporte']].replace(to_re-
place=0, method='ffill')\n      \n      \n      #Estabelecendo linhas de suporte e
resistência com base nos rompimentos\n      \n
df_sigla[['linha_resistencia']] = np.where((df_sigla["preco_min"] >= df_si-
glgla[['fractal_suporte']]), \\\n
df_sigla[['fractal_resistencia']], df_sigla[['fractal_suporte']])\n      df_si-
glgla[['linha_suporte']] = np.where((df_sigla["preco_min"] >= df_sigla[['frac-
tal_suporte']]), df_sigla[['fractal_suporte']], \\\n
df_sigla["preco_min"])\n      df_sigla[['linha_suporte']] =
np.where((df_sigla["preco_max"] <= df_sigla[['fractal_resistencia']]), \\\n
df_sigla[['fractal_suporte']], df_sigla["fractal_resistencia"])\n      df_si-
glgla[['linha_resistencia']] = np.where((df_sigla["preco_max"] <=
df_sigla[['fractal_resistencia']]), \\\n
df_sigla[['fractal_resistencia']], df_sigla["preco_max"])\n      \n      #Preço de
fechamento dias futuros\n      f=1 #um dia\n      df_sigla["preco_{}_dias_futu-
ros".format(f)]=df_sigla["preco_fechamento"].shift(-f)\n      \n      \n
df_enriched = df_enriched.append(df_sigla)\n')
# In[19]:
df_enriched.sample(10) #-
df_enriched.drop(['variacao', 'taxa_de_retorno_simples_diaria'], axis=1)
# ### Checando se há nulos e infinitos
# In[20]:
#checando
data_to_drop = df_enriched.isin([np.inf, -np.inf, np.nan])
data_to_drop[data_to_drop == True].count()
# In[21]:
# Transformando infinito em nan
df_enriched.replace([np.inf, -np.inf], np.nan, inplace=True)
#dropando esses valores
df_enriched=df_enriched.dropna()
#arredondando o que sobra
df_enriched = df_enriched.round(6)
df_enriched
# ### Codificando as features que são string
# In[22]:
le = LabelEncoder()
le.fit(df_enriched["sigla_acao"])
df_enriched["sigla_acao_label"]=le.transform(df_enriched["sigla_acao"])
le.fit(df_enriched["tipo"])
df_enriched["tipo_label"]=le.transform(df_enriched["tipo"])
# In[23]:

```

```

df_enriched
# # Iniciando nosso modelo de machine learning
# # Selecionando as melhores features - Verificando as colunas
# In[24]:
df_enriched.columns
# # Selecionando as melhores features: removendo informações redundantes
# In[25]:
df_enriched = df_enriched.drop(['variacao', 'taxa_de_retorno_simples_di-
aria', 'taxa_de_retorno_medio_diaria', 'desvpad_medio_diario', 'ganho', 'per-
da', 'fr_simples', 'fr_classica', 'fractal_resistencia', 'fractal_suporte',
'desv_pad_5_dias', 'desv_pad_13_dias',
'desv_pad_60_dias', 'desv_pad_200_dias', 'mm_14_dias_ganho',
'mm_14_dias_perda'], axis=1)
df_enriched.columns
# ## Separando features, labels e pré processamento
# In[26]:
features = df_enriched[['sigla_acao_label', 'tipo_label', 'preco_abertu-
ra', 'preco_max', 'preco_min', 'preco_fechamento',
'qtd_negocios', 'vol_negocios', 'taxa_de_retorno_medio_anual', 'vola-
tilidade', 'cluster', 'mm_5_dias',
'banda_superior_5_dias',
'banda_inferior_5_dias', 'volatilidade_entre_bandas_5_dias',
'diferenca_fechamento_e_mm_5_dias', 'mm_13_dias',
'banda_superior_13_dias', 'banda_inferior_13_dias',

'volatilidade_entre_bandas_13_dias', 'diferenca_fechamento_e_mm_13_dias',
'mm_60_dias', 'banda_superior_60_dias',
'banda_inferior_60_dias', 'volatilidade_entre_bandas_60_dias', 'dife-
renca_fechamento_e_mm_60_dias', 'mm_200_dias',
'banda_superior_200_dias',
'banda_inferior_200_dias', 'volatilidade_entre_bandas_200_dias',
'diferenca_fechamento_e_mm_200_dias', 'ifr_simples', 'ifr_classica',
'linha_resistencia', 'linha_suporte']]
label_1 = df_enriched["preco_1_dias_futuros"]
#removendo outliers
scaler = MinMaxScaler().fit(features)
features_normalized =
pd.DataFrame(scaler.transform(features), columns=features.columns)
features_normalized.sample(5)
# In[27]:
# melhores features regressão linear e regressão polinomial
f = features_normalized
l=label_1
feature_list = f.columns.values.tolist()
k_best_features = SelectKBest(f_regression , k="all")
k_best_features.fit_transform(f,l)

```

```

k_best_features_score = k_best_features.scores_
raw_pairs = zip(feature_list[:],k_best_features_score)
ordered_pairs= list(reversed(sorted(raw_pairs, key = lambda x: x[1])))
k_best_features_final = dict(ordered_pairs[:])
total = sum(k_best_features_final.values())
k_best_features_percent = {key: value / total for key, value in
k_best_features_final.items()}
best_features = k_best_features_final.keys()
print("para ---> ",l.name)
k_best_features_percent_above_1_percent = dict((k, v) for k, v in
k_best_features_percent.items() if v >= 0.01/100)
best_features = k_best_features_percent_above_1_percent.keys()
print("features mais importantes ---> ",best_features)
# In[28]:
# melhores features Decision Tree e random forrest regressor
f = features_normalized
l=label_1
feature_list = f.columns.values.tolist()
k_best_features = SelectKBest(mutual_info_regression , k="all")
k_best_features.fit_transform(f,l)
k_best_features_score = k_best_features.scores_
raw_pairs = zip(feature_list[:],k_best_features_score)
ordered_pairs= list(reversed(sorted(raw_pairs, key = lambda x: x[1])))
k_best_features_final = dict(ordered_pairs[:])
best_features = k_best_features_final.keys()
print("para ---> ",l.name)
k_best_features_percent_above_1_percent = dict((k, v) for k, v in
k_best_features_percent.items() if v >= 0.01/100)
best_features = k_best_features_percent_above_1_percent.keys()
print("features mais importantes ---> ",best_features)
# ### Conclusões:
# Em todos os casos, as melhores features, que contribuem com pelo menos
0,1% do peso total, são:
# [['preco_fechamento', 'preco_max', 'preco_min', 'preco_abertura',
'mm_5_dias', 'banda_inferior_5_dias', 'linha_suporte', 'mm_13_dias',
'banda_superior_5_dias', 'linha_resistencia', 'banda_inferior_13_dias', 'ban-
da_superior_13_dias', 'mm_60_dias', 'banda_inferior_60_dias',
'banda_superior_60_dias', 'mm_200_dias', 'banda_superior_200_dias', 'banda_in-
ferior_200_dias', 'volatilidade_entre_bandas_200_dias',
'volatilidade_entre_bandas_60_dias', 'volatilidade_entre_bandas_13_dias', 'vo-
latilidade_entre_bandas_5_dias']]
# # Modelos preditivos
# ## Preparando os dados
# In[29]:
df_enriched_acao = df_enriched

```

```

best_features = df_enriched_acao[['preco_fechamento', 'preco_max', 'preco_min', 'preco_abertura', 'mm_5_dias',
                                'banda_inferior_5_dias', 'linha_suporte', 'mm_13_dias', 'banda_superior_5_dias',
                                'linha_resistencia', 'banda_inferior_13_dias', 'banda_superior_13_dias', 'mm_60_dias',
                                'banda_inferior_60_dias', 'banda_superior_60_dias', 'mm_200_dias',
                                'banda_superior_200_dias', 'banda_inferior_200_dias', 'volatilidade_entre_bandas_200_dias',
                                'volatilidade_entre_bandas_60_dias', 'volatilidade_entre_bandas_13_dias',
                                'volatilidade_entre_bandas_5_dias']].sort_index()

#dimensionamento das features
scaler = MinMaxScaler().fit(best_features)
best_features_normalized = scaler.transform(best_features)
label = df_enriched_acao["preco_1_dias_futuros"].sort_index()
#Criando um dataframe para aglutinar as previsões
df_final_prediction = df_enriched[["data_pre-gao", "sigla_acao", "nome_acao", "preco_min", "preco_max",
                                   "preco_abertura", "preco_fechamento"]].sort_index()

#separando dataset em treino, teste e validação
X_train, X_test, y_train, y_test = train_test_split(best_features_normalized, label, test_size=0.3, random_state=42)
X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size=0.3, random_state=42)
print("Tamanho dos datasets de treino, teste e validação:")
print("X_train:", len(X_train), "X_test:", len(X_test), "X_val:", len(X_val), "\ny_train:", len(y_train), "y_test:", len(y_test), "y_val:", len(y_val))

# ### Regressão linear
# In[30]:
#Regressão linear (treino e teste)
print("Construindo o modelo de regressão linear")
lr = linear_model.LinearRegression()
lr.fit(X_train, y_train)
prediction_test = lr.predict(X_test)
print(f"Coeficiente de determinação do modelo (Teste) R2 Score: {r2_score(y_test, prediction_test)*100:2f}%")
print(f"Mean Absolute Percentage Error - MAPE: {mean_absolute_percentage_error(y_test, prediction_test)*100:2f}%")
print(f"Porcentagem de acerto - (100 - MAPE): {100 - mean_absolute_percentage_error(y_test, prediction_test)*100:2f}%")
#Executando a previsão com regressão linear
final_prediction_array_lr = lr.predict(X_val)

```

```

    print(f"Coeficiente de determinação do modelo (Validação) R2 Score:
{r2_score(y_val,final_prediction_array_lr)*100:2f}%")
    print(f"Mean Absolute Percentage Error - MAPE: {mean_absolute_percenta-
ge_error(y_val,final_prediction_array_lr)*100:2f}%")
    print(f"Porcentagem de acerto - (100 - MAPE): {100 - mean_absolute_per-
centage_error(y_val,final_prediction_array_lr)*100:2f}%")
    # ### Decision Tree
    # In[31]:
    get_ipython().run_cell_magic('capture', '', '#Decision Tree Regression
(parametros)\nparameters_dtr = { \n    \'splitter\': [\'best\', \'random\'],\n
\'max_features\': [\'auto\', \'sqrt\', \'log2\'],\n    \'random_state\' :
[42]\n}\n\nprint("Determinando os melhores parâmetros")\ngrid_dtr = GridSear-
chCV(DecisionTreeRegressor(),parameters_dtr,verbose=1)\n
ngrid_dtr.fit(X_train,y_train)')
    # In[32]:
    print("Construindo o modelo Decision Tree Regression com os melhores pa-
râmetros encontrados")
    print('best_estimator_', str(grid_dtr.best_estimator_))
    print('best_score_', str(grid_dtr.best_score_))
    dtr=grid_dtr.best_estimator_
    dtr.fit(X_train,y_train)
    prediction_test_dtr = dtr.predict(X_test)
    print(f"Coeficiente de determinação do modelo (Teste) R2 Score: {r2_sco-
re(y_test,prediction_test_dtr)*100:2f}%")
    print(f"Mean Absolute Percentage Error - MAPE: {mean_absolute_percenta-
ge_error(y_test,prediction_test_dtr)*100:2f}%")
    print(f"Porcentagem de acerto - (100 - MAPE): {100 - mean_absolute_per-
centage_error(y_test,prediction_test_dtr)*100:2f}%")
    #Executando a previsão com Decision Tree Regression
    final_prediction_array_dtr = dtr.predict(X_val)
    print(f"Coeficiente de determinação do modelo (Validação) R2 Score:
{r2_score(y_val,final_prediction_array_dtr)*100:2f}%")
    print(f"Mean Absolute Percentage Error - MAPE: {mean_absolute_percenta-
ge_error(y_val,final_prediction_array_dtr)*100:2f}%")
    print(f"Porcentagem de acerto - (100 - MAPE): {100 - mean_absolute_per-
centage_error(y_val,final_prediction_array_dtr)*100:2f}%")
    # ### Random Forest Regression
    # In[33]:
    get_ipython().run_cell_magic('capture', '', '#Random Forrest Regression
(parametros)\nparameters_rfr = { \n    \'n_estimators\': [10,20],\n
\'max_features\': [\'sqrt\',\'auto\'],\n    \'max_depth\' : [10,20],\n
\'random_state\' : [42]\n}\n\nprint("Determinando os melhores parâmetros")\n
ngrid_rfr = GridSearchCV(RandomForestRegressor(),parameters_rfr,verbose=1,sco-
ring=\'r2\')\ngrid_rfr.fit(X_train,y_train)')
    # In[34]:

```

```

    print("Construindo o modelo Random Forrest Regression com os melhores
parâmetros encontrados")
    print('best_estimator_', str(grid_rfr.best_estimator_))
    print('best_score_', str(grid_rfr.best_score_))
    rfr=grid_rfr.best_estimator_
    rfr.fit(X_train,y_train)
    prediction_test_rfr = rfr.predict(X_test)
    print(f"Coeficiente de determinação do modelo (Teste) R2 Score: {r2_score(y_test,prediction_test_rfr)*100:2f}%")
    print(f"Mean Absolute Percentage Error - MAPE: {mean_absolute_percentage_error(y_test,prediction_test_rfr)*100:2f}%")
    print(f"Porcentagem de acerto - (100 - MAPE): {100 - mean_absolute_percentage_error(y_test,prediction_test_rfr)*100:2f}%")
    #Executando a previsão com Random Forrest Regression
    final_prediction_array_rfr = rfr.predict(X_val)
    print(f"Coeficiente de determinação do modelo (Validação) R2 Score: {r2_score(y_val,final_prediction_array_rfr)*100:2f}%")
    print(f"Mean Absolute Percentage Error - MAPE: {mean_absolute_percentage_error(y_val,final_prediction_array_rfr)*100:2f}%")
    print(f"Porcentagem de acerto - (100 - MAPE): {100 - mean_absolute_percentage_error(y_val,final_prediction_array_rfr)*100:2f}%")
    # ### Regressão polinomial
    # In[35]:
    #Regressão polinomial (treino e teste)
    print("Construindo o modelo de regressão polinomial (3º grau)")
    poly_reg = PolynomialFeatures(degree=3)
    X_poly = poly_reg.fit_transform(X_train)
    pol_reg = linear_model.LinearRegression()
    pol_reg.fit(X_poly, y_train)
    prediction_test_poly = pol_reg.predict(poly_reg.fit_transform(X_test))
    print(f"Coeficiente de determinação do modelo (Teste) R2 Score: {r2_score(y_test,prediction_test_poly)*100:2f}%")
    print(f"Mean Absolute Percentage Error - MAPE: {mean_absolute_percentage_error(y_test,prediction_test_poly)*100:2f}%")
    print(f"Porcentagem de acerto - (100 - MAPE): {100 - mean_absolute_percentage_error(y_test,prediction_test_poly)*100:2f}%")
    #Executando a previsão com regressão polinomial
    final_prediction_array_poly =
pol_reg.predict(poly_reg.fit_transform(X_val))
    print(f"Coeficiente de determinação do modelo (Validação) R2 Score: {r2_score(y_val,final_prediction_array_poly)*100:2f}%")
    print(f"Mean Absolute Percentage Error - MAPE: {mean_absolute_percentage_error(y_val,final_prediction_array_poly)*100:2f}%")
    print(f"Porcentagem de acerto - (100 - MAPE): {100 - mean_absolute_percentage_error(y_val,final_prediction_array_poly)*100:2f}%")
    # ### Dataframe de validações

```

```

# In[36]:
#Montando um dataframe com as previsões
df_prediction = pd.DataFrame(y_val)
df_prediction[f"{label.name}_previsao_lr"] = final_prediction_array_lr.-
tolist()
df_prediction[f"{label.name}_previsao_dtr"] =
final_prediction_array_dtr.tolist()
df_prediction[f"{label.name}_previsao_rfr"] =
final_prediction_array_rfr.tolist()
df_prediction[f"{label.name}_previsao_poly"] =
final_prediction_array_poly.tolist()
#Fazendo inner join (merge o pandas) entre a previsão e as datas
df_final_prediction = pd.merge(df_final_prediction.sort_index(),
df_prediction[[f"{label.name}",
f"{label.name}_previsao_lr", f"{label.name}_previsao_dtr",
name}_previsao_rfr", f"{label.name}_previsao_poly"]]]
.sort_index(), left_index=True, right_index=True)
df_final_prediction = df_final_prediction.sort_index()
# In[37]:
df_final_prediction[["sigla_acao", "preco_1_dias_futuros", "preco_1_dias_f
uturos_previsao_lr",
"preco_1_dias_futuros_previsao_dtr",
"preco_1_dias_futuros_previsao_rfr",
ros_previsao_poly"]].sample(10)
# ##### Gráficos
# In[38]:
#plotando graficamente
sigla = "ALPA4"
df_final_prediction_acao = df_final_prediction[df_final_prediction["si-
gla_acao"]==sigla].tail(10)
f = "preco_1_dias_futuros"
plt.figure(figsize=(20, 5))
plt.title(f"Peço fechamento previsto vs Preço fechamento real \n {f}
{sigla}")
plt.plot(df_final_prediction_acao["data_pregao"], df_final_prediction_aca
o[f"{f}_previsao_dtr"], label="Preço Previsto D.T.R.", color="yel-
low", marker="d")
plt.plot(df_final_prediction_acao["data_pregao"], df_final_prediction_aca
o[f"{f}_previsao_rfr"], label="Preço Previsto R.F.R.", co-
lor="red", marker="^")
plt.plot(df_final_prediction_acao["data_pregao"], df_final_prediction_aca
o[f"{f}_previsao_poly"], label="Preço Previsto R.P.", co-
lor="blue", marker="s")
plt.plot(df_final_prediction_acao["data_pregao"], df_final_prediction_aca
o[f"{f}_previsao_lr"], label="Preço Previsto R.L.", co-
lor="green", marker="o")

```

```

plt.plot(df_final_prediction_acao["data_pregao"],
df_final_prediction_acao[f"{f}"],label="Preço Real",          co-
lor="black",marker="o")
    #get current axes
    ax = plt.gca()
    #hide x-axis
    ax.get_xaxis().set_visible(False)
    plt.legend()
    plt.xlabel("Data Pregão")
    plt.ylabel("Preço de Fechamento")
    # In[39]:
    #plotando graficamente
    sigla = "PETR3"
    df_final_prediction_acao = df_final_prediction[df_final_prediction["si-
gla_acao"]==sigla].tail(10)
    f = "preco_1_dias_futuros"
    plt.figure(figsize=(20 ,5))
    plt.title(f"Peço fechamento previsto vs Preço fechamento real \n {f}
{sigla}")
    plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_aca
o[f"{f}_previsao_dtr"],          label="Preço Previsto D.T.R.", color="yel-
low",marker="d")
    plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_aca
o[f"{f}_previsao_rfr"],          label="Preço Previsto R.F.R.", co-
lor="red",marker="^")
    plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_aca
o[f"{f}_previsao_poly"],          label="Preço Previsto R.P.", co-
lor="blue",marker="s")
    plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_aca
o[f"{f}_previsao_lr"],          label="Preço Previsto R.L.", co-
lor="green",marker="o")
    plt.plot(df_final_prediction_acao["data_pregao"],
df_final_prediction_acao[f"{f}"],label="Preço Real",          co-
lor="black",marker="o")
    #get current axes
    ax = plt.gca()
    #hide x-axis
    ax.get_xaxis().set_visible(False)
    plt.legend()
    plt.xlabel("Data Pregão")
    plt.ylabel("Preço de Fechamento")
    # In[40]:
    #plotando graficamente
    sigla = "VALE3"
    df_final_prediction_acao = df_final_prediction[df_final_prediction["si-
gla_acao"]==sigla].tail(10)

```



```

f = "preco_1_dias_futuros"
plt.figure(figsize=(20 ,5))
plt.title(f"Peço fechamento previsto vs Preço fechamento real \n {f}
{sigla}")
plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_aca
o[f"{f}_previsao_dtr"],          label="Preço Previsto D.T.R.", color="yel-
low",marker="d")
plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_aca
o[f"{f}_previsao_rfr"],          label="Preço Previsto R.F.R.", co-
lor="red",marker="^")
plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_aca
o[f"{f}_previsao_poly"],          label="Preço Previsto R.P.", co-
lor="blue",marker="s")
plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_aca
o[f"{f}_previsao_lr"],          label="Preço Previsto R.L.", co-
lor="green",marker="o")
plt.plot(df_final_prediction_acao["data_pregao"],
df_final_prediction_acao[f"{f}"],label="Preço Real",          co-
lor="black",marker="o")
#get current axes
ax = plt.gca()
#hide x-axis
ax.get_xaxis().set_visible(False)
plt.legend()
plt.xlabel("Data Pregão")
plt.ylabel("Preço de Fechamento")
# In[41]:
#plotando graficamente
sigla = "VIVT3"
df_final_prediction_acao = df_final_prediction[df_final_prediction["si-
gla_acao"]==sigla].tail(10)
f = "preco_1_dias_futuros"
plt.figure(figsize=(20 ,5))
plt.title(f"Peço fechamento previsto vs Preço fechamento real \n {f}
{sigla}")
plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_aca
o[f"{f}_previsao_dtr"],          label="Preço Previsto D.T.R.", color="yel-
low",marker="d")
plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_aca
o[f"{f}_previsao_rfr"],          label="Preço Previsto R.F.R.", co-
lor="red",marker="^")
plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_aca
o[f"{f}_previsao_poly"],          label="Preço Previsto R.P.", co-
lor="blue",marker="s")

```

```

plt.plot(df_final_prediction_acao["data_pregao"],df_final_prediction_aca
o[f"{f}_previsao_lr"],          label="Preço Previsto R.L.", co-
lor="green",marker="o")
plt.plot(df_final_prediction_acao["data_pregao"],
df_final_prediction_acao[f"{f}"],label="Preço Real",          co-
lor="black",marker="o")
#get current axes
ax = plt.gca()
#hide x-axis
ax.get_xaxis().set_visible(False)
plt.legend()
plt.xlabel("Data Pregão")
plt.ylabel("Preço de Fechamento")
# In[ ]:

```