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

Jonathan dos Santos Lincher

Modelo Preditivo de Preços de Contratos Futuros de Commodities

Belo Horizonte
2021

Jonathan dos Santos Lincher

Modelo Preditivo de Preços de Contratos Futuros de Commodities

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
2021

SUMÁRIO

1. Introdução

1.1. Contextualização

O Brasil tem na sua história a agricultura com um dos pilares do seu desenvolvimento. Inicialmente tivemos o ciclo da cana-de-açúcar no Nordeste com a chegada dos europeus, depois o ciclo do café no Sudeste no período colonial e após os anos 70 um grande período de diversificação agrícola em que a soja se tornou a protagonista, principalmente no Centro Oeste do país.

Desde 2010, somos o terceiro maior produtor e exportador agrícola do mundo, atras dos Estados Unidos e da União Europeia. Diferentemente destes, a nossa capacidade de crescimento é grande no meio prazo por conta da possibilidade ter obtermos maior produtividade e aumento da área cultiva, pela mecanização do campo e da expansão das fronteiras agrícolas.

Dos principais produtos agropecuários, destacam-se a cana-de-açúcar, o café e a laranja, já que somos o maior produtor mundial; depois temos a soja, o fumo e a carne bovina, para quais temos a segunda colocação internacional; e por fim o milho, pois o Brasil é o terceiro país em volume de produção anual.

Em relação ao milho, metade da sua produção decorre de pequenos produtores e a outra metade de latifundiários que tem foco no milho transgênico. A sua produção tem a expectativa de crescer 3% nos próximos anos e as áreas agriculturáveis de aumentarem 1% nas próximas safras. Por conta da grande quantidade de produtores e da sua expectativa de crescimento, é um produto que tem grande importância em estudos e projeções.

O milho é uma commodity agrícola, tendo seu preço gerido pela sua cotação no mercado (e não pelo valor estipulado na produção), geralmente nas grandes bolsas de valores. Assim, é importante que os produtores evitem oscilações de preços desse produto. Uma forma de se proteger é negociar contratos futuros de milho no ambiente da Bolsa de Mercadorias e Futuros da B3. Esses contratos são acordos de compra e venda de sacas desse produto, que ocorrerão em uma data futura, por um preço preestabelecido no momento da negociação.

1.2. O problema proposto

Utilizaremos análise exploratória e modelagem preditiva para extrair informações das séries temporais do preço da saca de milho (60 Kg - à vista em reais e dólares); e dos preços e volume financeiro do contrato futuro de milho com liquidação financeira (código CMMFut, em que cada contrato corresponde a 450 sacas de 60 kg) para auxiliar produtores, compradores e investidores na tomada de decisão.

Temos como objetivos dessa análise:

- Realizar a análise descritiva dos dados desses dois ativos;
- Verificar a correlação entre eles;
- Criar modelos preditivos para os preços do contrato futuro de milho utilizando Ridge Regression e LSTM.

Os dados extraídos são dos anos de 2004 a 2020. Para análise exploratória, utilizamos os dados de todo o período. Quanto aos modelos preditivos, 70% dos dados foram utilizados como base de treinamento e o restante (30%) como dados de teste.

2. Coleta de Dados

Utilizamos três datasets para a realização deste trabalho, sendo um referente aos preços da saca de milho negociada a vista (milho_df), outro referente aos preços do contrato futuro de milho negociado em bolsa (ccmfut_df) e um terceiro referente aos dados históricos do dólar.

A série de preços (02/08/2004 a 03/03/2021) da saca de milho foi coletada em 03/03/2021 em formato .xls no site do Centro de Estudos Avançados em Economia Aplicada da ESALQ/USP por meio do link: https://www.cepea.esalq.usp.br/br/indicador/milho.aspx

O dataset tem o formato descrito na tabela abaixo:

Nome da coluna/campo	Descrição	Tipo
Data	Data de negociação do ativo	Pandas (datetime - index)
A vista R$	Valor em Reais por saca de 60 kg, à vista, descontado o Prazo de Pagamento pela taxa CDI/CETIP.	Pandas (float64)
A vista US$	Valor em Dólares por saca de 60 kg, à vista, descontado o Prazo de Pagamento pela taxa CDI/CETIP.	Pandas (float64)

Utilizamos os dois valores na análise exploratória e no modelo preditivo usando Ridge Regression.

Quanto aos dados do contrato futuro de milho, buscamos um dataset que contivesse os dados históricos dos contratos. A BM&F disponibiliza contratos de Milho Futuro com vencimento em Janeiro, Março, Maio, Julho, Agosto, Setembro e Novembro de cada ano. Cada um desses contratos possui o código de negociação composto pelo radical CCM , acrescido da letra correspondente ao mês de vencimento do contrato (F, H, K, N, Q, U ou X) e de dois números correspondentes ao ano de vencimento do contrato. Dessa forma, há diversos códigos de contratos com dados referentes ao seu breve período de negociação. Uma forma encontrada foi usar o código CCMFut (que representa todos os contratos) para extrair os dados históricos na plataforma ProfitChart da Nelogica (https://www.nelogica.com.br/). É possível usar a ferramenta gratuitamente durante um período de teste e extrair os dados em um arquivo .csv. Os dados foram extraídos em 05/03/2021, referentes ao período de 19/09/2008 a 05/03/2021, com o formato descrito abaixo:

Nome da coluna/campo	Descrição	Tipo
Data	Data da negociação	Pandas (datetime - index)
Abertura	Preço de abertura do ativo no dia	Pandas (float64)
Máxima	Maior preço do ativo no dia	Pandas (float64)

Mínima	Menor preço do ativo no dia	Pandas (float64)
Fechamento	Preço de fechamento do ativo no dia	Pandas (float64)
Volume Financeiro	Volume financeiro negociado no dia	Pandas (float64)

Utilizamos todos os dados, tanto na análise exploratória quanto no modelo preditivo Ridge Regression. Para o modelo usando LSTM, usamos os dados de preço de fechamento.

O último dataset (dolar_df) contém os dados históricos do dólar no período de 02/08/2004 a 03/03/2021. Foi obtido em formato .csv no site da Investing.com: https://br.investing.com/currencies/usd-brl-historical-data.

Nome da coluna/campo	Descrição	Tipo
Data	Data da cotação do câmbio	Pandas (datetime - index)
Último	Valor de fechamento do câmbio no dia	Pandas (float64)
Abertura	Preço de abertura do câmbio no dia	Pandas (float64)
Máxima	Maior preço do câmbio no dia	Pandas (float64)
Mínima	Menor preço do câmbio no dia	Pandas (float64)
Var%	Variação percentual do câmbio em relação ao dia anterior de negociação	Pandas (float64)

Utilizamos o preço de fechamento (fechamento) na análise exploratória dos preços do milho.

3. Processamento/Tratamento de Dados

O dataset milho_df apresenta 4130 linhas e 10 colunas:

```
In [8]: milho_df = pd.read_excel('Milho-CEPEA-ESALQ.xlsx')
        milho_df
```

Out[8]:

	INDICADOR DO MILHO ESALQ/BM&FBOVESPA	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	Fonte: Cepea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	Data	À vista R$	À vista US$	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	02/08/2004	18.24	5.98	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	03/08/2004	18.04	5.91	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...
4125	25/02/2021	85.59	15.55	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4126	26/02/2021	85.41	15.3	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4127	01/03/2021	85.59	15.29	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4128	02/03/2021	86.11	15.2	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4129	03/03/2021	87.06	15.14	NaN	NaN	NaN	NaN	NaN	NaN	NaN

4130 rows × 10 columns

Após a leitura e criação do dataframe, passamos para o tratamento dos dados.

Inicialmente, retiramos as linhas e colunas desnecessárias:

```
In [9]: milho_df = milho_df.drop(milho_df.index[0:3])
        milho_df = milho_df.drop(columns=milho_df.columns[3:])
        milho_df
```

Out[9]:

	INDICADOR DO MILHO ESALQ/BM&FBOVESPA	Unnamed: 1	Unnamed: 2
3	02/08/2004	18.24	5.98
4	03/08/2004	18.04	5.91
5	04/08/2004	18.02	5.9
6	05/08/2004	18.06	5.89
7	06/08/2004	18.13	5.98
...	...	...	...
4125	25/02/2021	85.59	15.55
4126	26/02/2021	85.41	15.3
4127	01/03/2021	85.59	15.29
4128	02/03/2021	86.11	15.2
4129	03/03/2021	87.06	15.14

4127 rows × 3 columns

O segundo passo foi renomear as colunas, incluindo o nome “milho”, para facilitar a visualização e manipulação dos dados, quando da junção dos datasets.

```
In [10]: milho_df.rename(columns= {'INDICADOR DO MILHO ESALQ/BM&FBOVESPA': 'Data'}, inplace=True)
         milho_df.rename(columns= {'Unnamed: 1': 'milho_reais'}, inplace=True)
         milho_df.rename(columns= {'Unnamed: 2': 'milho_dolares'}, inplace=True)
         milho_df
```

Out[10]:

	Data	milho_reais	milho_dolares
3	02/08/2004	18.24	5.98
4	03/08/2004	18.04	5.91
5	04/08/2004	18.02	5.9
6	05/08/2004	18.06	5.89
7	06/08/2004	18.13	5.98
...	...	...	...
4125	25/02/2021	85.59	15.55
4126	26/02/2021	85.41	15.3
4127	01/03/2021	85.59	15.29
4128	02/03/2021	86.11	15.2
4129	03/03/2021	87.06	15.14

4127 rows × 3 columns

Em seguida, verificamos as informações do DataFrame (índices, colunas, valores não-nulos, os tipos de dados e a utilização da memória):

```
In [11]: milho_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4127 entries, 3 to 4129
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Data           4127 non-null   object
 1   milho_reais    4127 non-null   object
 2   milho_dolares  4127 non-null   object
dtypes: object(3)
memory usage: 129.0+ KB
```

Mostrou-se necessário alterarmos os tipos dos dados referentes ao milho_reais e milho_dolar de object para float64:

```
In [12]: milho_df['milho_reais'] = milho_df['milho_reais'].astype(float)
         milho_df['milho_dolares'] = milho_df['milho_dolares'].astype(float)
         milho_df
```

Out[12]:

	Data	milho_reais	milho_dolares
3	02/08/2004	18.24	5.98
4	03/08/2004	18.04	5.91
5	04/08/2004	18.02	5.90
6	05/08/2004	18.06	5.89
7	06/08/2004	18.13	5.98
...	...	...	...
4125	25/02/2021	85.59	15.55
4126	26/02/2021	85.41	15.30
4127	01/03/2021	85.59	15.29
4128	02/03/2021	86.11	15.20
4129	03/03/2021	87.06	15.14

4127 rows × 3 columns

```
In [13]: milho_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4127 entries, 3 to 4129
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Data           4127 non-null   object 
 1   milho_reais    4127 non-null   float64
 2   milho_dolares  4127 non-null   float64
dtypes: float64(2), object(1)
memory usage: 129.0+ KB
```

Continuamos o tratamento, agora alterando o formato das datas para YYYY-MM-DD e colocando-as em ordem crescente:

```
In [18]: milho_df['Data']
```

```
Out[18]: 3       02/08/2004
         4       03/08/2004
         5       04/08/2004
         6       05/08/2004
         7       06/08/2004
                    ...
         4125    25/02/2021
         4126    26/02/2021
         4127    01/03/2021
         4128    02/03/2021
         4129    03/03/2021
         Name: Data, Length: 4127, dtype: object
```

```
In [19]: milho_df['Data'] = pd.to_datetime(milho_df['Data'],dayfirst=True)
         milho_df = milho_df.sort_values(by = ['Data'])
         milho_df['Data']
```

```
Out[19]: 3      2004-08-02
         4      2004-08-03
         5      2004-08-04
         6      2004-08-05
         7      2004-08-06
                   ...
         4125   2021-02-25
         4126   2021-02-26
         4127   2021-03-01
         4128   2021-03-02
         4129   2021-03-03
         Name: Data, Length: 4127, dtype: datetime64[ns]
```

O passo seguinte foi definir as datas como o index do dataframe:

```
In [20]: milho_df.index
```

```
Out[20]: Int64Index([   3,    4,    5,    6,    7,    8,    9,   10,   11,   12,
                     ...
                     4120, 4121, 4122, 4123, 4124, 4125, 4126, 4127, 4128, 4129],
                    dtype='int64', length=4127)
```

```
In [21]: milho_df.index = pd.to_datetime(milho_df.Data)
         milho_df.index.to_period('D')
         milho_df.index
```

```
Out[21]: DatetimeIndex(['2004-08-02', '2004-08-03', '2004-08-04', '2004-08-05',
                         '2004-08-06', '2004-08-09', '2004-08-10', '2004-08-11',
                         '2004-08-12', '2004-08-13',
                         ...
                         '2021-02-18', '2021-02-19', '2021-02-22', '2021-02-23',
                         '2021-02-24', '2021-02-25', '2021-02-26', '2021-03-01',
                         '2021-03-02', '2021-03-03'],
                        dtype='datetime64[ns]', name='Data', length=4127, freq=None)
```

Por fim, verificamos se há dados nulos:

```
In [24]: milho_df.isnull().sum()
```

```
Out[24]: Data            0
         milho_reais     0
         milho_dolares   0
         dtype: int64
```

Já o dataset do ccmfut (ccmfut_df) apresenta 2.919 linhas e 6 colunas:

```
In [14]: ccmfut_df = pd.read_excel('CCMFUT-ProfitChart.xlsx')
         ccmfut_df
```

Out[14]:

	Data	Abertura	Máxima	Mínima	Fechamento	Volume Financeiro
0	2021-03-05	94.10	96.46	94.10	95.85	380377381.5
1	2021-03-04	91.00	94.72	90.40	94.20	325967202.0
2	2021-03-03	89.00	91.10	88.86	91.05	211768164.0
3	2021-03-02	88.80	89.06	88.54	88.94	95795617.5
4	2021-03-01	88.92	89.18	88.00	88.70	133054398.0
...	...	...	...	...	...	...
2914	2008-10-02	22.64	22.64	22.64	22.64	11295.0
2915	2008-09-30	22.55	22.55	22.55	22.55	112500.0
2916	2008-09-26	22.68	22.68	22.68	22.68	565875.0
2917	2008-09-22	22.64	22.64	22.64	22.64	112950.0
2918	2008-09-19	22.64	22.64	22.64	22.64	1129500.0

2919 rows × 6 columns

Alteramos os nomes das colunas, incluindo o nome “ccmfut”, para facilitar a visualização e manipulação dos dados, quando da junção dos datasets.

```
In [15]: # Renomear colunas
         ccmfut_df.rename(columns= {'Abertura': 'ccmfut_abertura'}, inplace=True)
         ccmfut_df.rename(columns= {'Máxima': 'ccmfut_máxima'}, inplace=True)
         ccmfut_df.rename(columns= {'Mínima': 'ccmfut_mínima'}, inplace=True)
         ccmfut_df.rename(columns= {'Fechamento': 'ccmfut_fechamento'}, inplace=True)
         ccmfut_df.rename(columns= {'Volume Financeiro': 'ccmfut_volume_fin'}, inplace=True)
         ccmfut_df
```

Out[15]:

	Data	ccmfut_abertura	ccmfut_máxima	ccmfut_mínima	ccmfut_fechamento	ccmfut_volume_fin
0	2021-03-05	94.10	96.46	94.10	95.85	380377381.5
1	2021-03-04	91.00	94.72	90.40	94.20	325967202.0
2	2021-03-03	89.00	91.10	88.86	91.05	211768164.0
3	2021-03-02	88.80	89.06	88.54	88.94	95795617.5
4	2021-03-01	88.92	89.18	88.00	88.70	133054398.0
...	...	...	...	...	...	...
2914	2008-10-02	22.64	22.64	22.64	22.64	11295.0
2915	2008-09-30	22.55	22.55	22.55	22.55	112500.0
2916	2008-09-26	22.68	22.68	22.68	22.68	565875.0
2917	2008-09-22	22.64	22.64	22.64	22.64	112950.0
2918	2008-09-19	22.64	22.64	22.64	22.64	1129500.0

2919 rows × 6 columns

Não foi necessário alterar os tipos dos dados:

```
In [16]: ccmfut_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2919 entries, 0 to 2918
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype         
---  ------             --------------  -----         
 0   Data               2919 non-null   datetime64[ns]
 1   ccmfut_abertura    2919 non-null   float64       
 2   ccmfut_máxima      2919 non-null   float64       
 3   ccmfut_mínima      2919 non-null   float64       
 4   ccmfut_fechamento  2919 non-null   float64       
 5   ccmfut_volume_fin  2919 non-null   float64       
dtypes: datetime64[ns](1), float64(5)
memory usage: 137.0 KB
```

Excluímos as duas primeiras linhas, para que a data final coincida com a data final do dataset do milho (03/03/2021):

```
In [30]: ccmfut_df = ccmfut_df.drop(ccmfut_df.index[0:2])
         ccmfut_df
```

Out[30]:

	Data	ccmfut_abertura	ccmfut_máxima	ccmfut_mínima	ccmfut_fechamento	ccmfut_volume_fin
2	2021-03-03	89.00	91.10	88.86	91.05	211768164.0
3	2021-03-02	88.80	89.06	88.54	88.94	95795617.5
4	2021-03-01	88.92	89.18	88.00	88.70	133054398.0
5	2021-02-26	89.61	89.64	88.86	88.86	69200707.5
6	2021-02-25	89.45	89.72	88.81	89.63	106361550.0
...	...	...	...	...	...	...
2914	2008-10-02	22.64	22.64	22.64	22.64	11295.0
2915	2008-09-30	22.55	22.55	22.55	22.55	112500.0
2916	2008-09-26	22.68	22.68	22.68	22.68	565875.0
2917	2008-09-22	22.64	22.64	22.64	22.64	112950.0
2918	2008-09-19	22.64	22.64	22.64	22.64	1129500.0

2917 rows × 6 columns

Ordenamos os dados pela data em ordem crescente:

```
In [52]: ccmfut_df = ccmfut_df.sort_values(by = ['Data'])
         ccmfut_df
```

Out[52]:

	Data	ccmfut_abertura	ccmfut_máxima	ccmfut_mínima	ccmfut_fechamento	ccmfut_volume_fin
2918	2008-09-19	22.64	22.64	22.64	22.64	1129500.0
2917	2008-09-22	22.64	22.64	22.64	22.64	112950.0
2916	2008-09-26	22.68	22.68	22.68	22.68	565875.0
2915	2008-09-30	22.55	22.55	22.55	22.55	112500.0
2914	2008-10-02	22.64	22.64	22.64	22.64	11295.0
...	...	...	...	...	...	...
6	2021-02-25	89.45	89.72	88.81	89.63	106361550.0
5	2021-02-26	89.61	89.64	88.86	88.86	69200707.5
4	2021-03-01	88.92	89.18	88.00	88.70	133054398.0
3	2021-03-02	88.80	89.06	88.54	88.94	95795617.5
2	2021-03-03	89.00	91.10	88.86	91.05	211768164.0

2917 rows × 6 columns

Em seguida, definimos as datas como index:

```
In [27]: ccmfut_df.index
```

```
Out[27]: Int64Index([2918, 2917, 2916, 2915, 2914, 2913, 2912, 2911, 2910, 2909,
                     ...
                        9,    8,    7,    6,    5,    4,    3,    2,    1,    0],
                    dtype='int64', length=2919)
```

```
In [28]: ccmfut_df.index = pd.to_datetime(ccmfut_df.Data)
         ccmfut_df.index.to_period('D')
         ccmfut_df.index
```

```
Out[28]: DatetimeIndex(['2008-09-19', '2008-09-22', '2008-09-26', '2008-09-30',
                        '2008-10-02', '2008-10-03', '2008-10-06', '2008-10-07',
                        '2008-10-08', '2008-10-09',
                        ...
                        '2021-02-22', '2021-02-23', '2021-02-24', '2021-02-25',
                        '2021-02-26', '2021-03-01', '2021-03-02', '2021-03-03',
                        '2021-03-04', '2021-03-05'],
                       dtype='datetime64[ns]', name='Data', length=2919, freq=None)
```

Continuando, verificamos se há dados nulos no dataframe:

```
In [26]: ccmfut_df.isnull().sum()
```

```
Out[26]: Data                0
         ccmfut_abertura     0
         ccmfut_máxima       0
         ccmfut_mínima       0
         ccmfut_fechamento   0
         ccmfut_volume_fin   0
         dtype: int64
```

Por fim, para evitar conflitos, alterarmos o nome da coluna referente as datas nos dois datasets de “Data” para “data”:

```
milho_df.rename(columns= {'Data': 'data'}, inplace=True)
ccmfut_df.rename(columns= {'Data': 'data'}, inplace=True)
```

Os dois datasets estão formatados/tratados:

In [28]:
```
milho_df
```

Out[28]:

Data	data	milho_reais	milho_dolares
2004-08-02	2004-08-02	18.24	5.98
2004-08-03	2004-08-03	18.04	5.91
2004-08-04	2004-08-04	18.02	5.90
2004-08-05	2004-08-05	18.06	5.89
2004-08-06	2004-08-06	18.13	5.98
...	...	...	...
2021-02-25	2021-02-25	85.59	15.55
2021-02-26	2021-02-26	85.41	15.30
2021-03-01	2021-03-01	85.59	15.29
2021-03-02	2021-03-02	86.11	15.20
2021-03-03	2021-03-03	87.06	15.14

4127 rows × 3 columns

In [29]:
```
ccmfut_df
```

Out[29]:

Data	data	ccmfut_abertura	ccmfut_máxima	ccmfut_mínima	ccmfut_fechamento	ccmfut_volume_fin
2008-09-19	2008-09-19	22.64	22.64	22.64	22.64	1129500.0
2008-09-22	2008-09-22	22.64	22.64	22.64	22.64	112950.0
2008-09-26	2008-09-26	22.68	22.68	22.68	22.68	565875.0
2008-09-30	2008-09-30	22.55	22.55	22.55	22.55	112500.0
2008-10-02	2008-10-02	22.64	22.64	22.64	22.64	11295.0
...	...	...	...	...	...	...
2021-02-25	2021-02-25	89.45	89.72	88.81	89.63	106361550.0
2021-02-26	2021-02-26	89.61	89.64	88.86	88.86	69200707.5
2021-03-01	2021-03-01	88.92	89.18	88.00	88.70	133054398.0
2021-03-02	2021-03-02	88.80	89.06	88.54	88.94	95795617.5
2021-03-03	2021-03-03	89.00	91.10	88.86	91.05	211768164.0

2917 rows × 6 columns

Na sequência, criamos o dataset mc_df contendo os dados dos dois datasets anteriores. Ele tem 2917 linhas e 8 colunas. A referência é o ccmfut_df já que o preço de fechamento do contrato futuro é o nosso principal alvo.

```
In [30]: mc_df = ccmfut_df.merge(
             milho_df.set_index('data'), how='left', on='data'
         )
```

```
In [31]: mc_df
```

Out[31]:

	data	ccmfut_abertura	ccmfut_máxima	ccmfut_mínima	ccmfut_fechamento	ccmfut_volume_fin	milho_reais	milho_dolares
0	2008-09-19	22.64	22.64	22.64	22.64	1129500.0	23.47	12.82
1	2008-09-22	22.64	22.64	22.64	22.64	112950.0	23.31	13.00
2	2008-09-26	22.68	22.68	22.68	22.68	565875.0	23.24	12.54
3	2008-09-30	22.55	22.55	22.55	22.55	112500.0	22.99	12.08
4	2008-10-02	22.64	22.64	22.64	22.64	11295.0	22.95	11.36
...	...	...	...	...	...	...	...	...
2912	2021-02-25	89.45	89.72	88.81	89.63	106361550.0	85.59	15.55
2913	2021-02-26	89.61	89.64	88.86	88.86	69200707.5	85.41	15.30
2914	2021-03-01	88.92	89.18	88.00	88.70	133054398.0	85.59	15.29
2915	2021-03-02	88.80	89.06	88.54	88.94	95795617.5	86.11	15.20
2916	2021-03-03	89.00	91.10	88.86	91.05	211768164.0	87.06	15.14

2917 rows × 8 columns

Na sequência, verificamos a quantidade de dados nulos do mc_df:

```
In [32]: mc_df.isnull().sum()

Out[32]: data                0
         ccmfut_abertura     0
         ccmfut_máxima       0
         ccmfut_mínima       0
         ccmfut_fechamento   0
         ccmfut_volume_fin   0
         milho_reais         2
         milho_dolares       2
         dtype: int64
```

Tentamos preencher os dados nulos com a média móvel dos últimos 5 períodos, na tentativa de obter um dado mais próximo e menos distorcido:

```
In [33]: mc_df_mvmilho_reais = mc_df["milho_reais"].rolling(5).mean().shift(-5).round(0)
         mc_df_mvmilho_dolares = mc_df["milho_dolares"].rolling(5).mean().shift(-5).round(0)
         mc_df["milho_reais"].fillna(mc_df_mvmilho_reais, inplace=True)
         mc_df["milho_dolares"].fillna(mc_df_mvmilho_dolares, inplace=True)
```

Essa operação preencheu 1 linha:

```
In [34]: mc_df.isnull().sum()

Out[34]: data                0
         ccmfut_abertura     0
         ccmfut_máxima       0
         ccmfut_mínima       0
         ccmfut_fechamento   0
         ccmfut_volume_fin   0
         milho_reais         1
         milho_dolares       1
         dtype: int64
```

Fizemos o procedimento mais uma vez e não há mais dados nulos:

```
In [35]: mc_df_mvmilho_reais = mc_df["milho_reais"].rolling(5).mean().shift(-5).round(0)
         mc_df_mvmilho_dolares = mc_df["milho_dolares"].rolling(5).mean().shift(-5).round(0)
         mc_df["milho_reais"].fillna(mc_df_mvmilho_reais, inplace=True)
         mc_df["milho_dolares"].fillna(mc_df_mvmilho_dolares, inplace=True)

In [36]: mc_df.isnull().sum()

Out[36]: data                0
         ccmfut_abertura     0
         ccmfut_máxima       0
         ccmfut_mínima       0
         ccmfut_fechamento   0
         ccmfut_volume_fin   0
         milho_reais         0
         milho_dolares       0
         dtype: int64
```

Ordenamos os dados pela data:

```
In [41]: mc_df = mc_df.sort_values(by = ['data'])
         mc_df
```

E definimos a data como index do dataframe:

```
In [42]: mc_df.index = pd.to_datetime(mc_df.data)
         mc_df.index.to_period('D')
         mc_df.index
```

Os dados estão formatados:

```
In [43]: mc_df
```

Out[43]:

	data	ccmfut_abertura	ccmfut_máxima	ccmfut_mínima	ccmfut_fechamento	ccmfut_volume_fin	milho_reais	milho_dolares
data								
2008-09-19	2008-09-19	22.64	22.64	22.64	22.64	1129500.0	23.47	12.82
2008-09-22	2008-09-22	22.64	22.64	22.64	22.64	112950.0	23.31	13.00
2008-09-26	2008-09-26	22.68	22.68	22.68	22.68	565875.0	23.24	12.54
2008-09-30	2008-09-30	22.55	22.55	22.55	22.55	112500.0	22.99	12.08
2008-10-02	2008-10-02	22.64	22.64	22.64	22.64	11295.0	22.95	11.36
...	...	...	...	...	...	...	...	...
2021-02-25	2021-02-25	89.45	89.72	88.81	89.63	106361550.0	85.59	15.55
2021-02-26	2021-02-26	89.61	89.64	88.86	88.86	69200707.5	85.41	15.30
2021-03-01	2021-03-01	88.92	89.18	88.00	88.70	133054398.0	85.59	15.29
2021-03-02	2021-03-02	88.80	89.06	88.54	88.94	95795617.5	86.11	15.20
2021-03-03	2021-03-03	89.00	91.10	88.86	91.05	211768164.0	87.06	15.14

2917 rows × 8 columns

4. Análise e Exploração dos Dados

Iniciamos a análise e exploração dos dados com o resumo estatístico do milho_df:

```
In [347]: milho_df.describe()
```

Out[347]:

	milho_reais	milho_dolares
count	4127.000000	4127.000000
mean	30.409537	11.535544
std	12.350048	3.276651
min	13.320000	5.890000
25%	21.325000	9.250000
50%	27.770000	10.720000
75%	35.375000	13.750000
max	87.060000	19.960000

Na sequência, verificamos as datas em que ocorreram as máximas e mínimas dos preços do milho em reais e em dólares:

```
In [586]: milho_df[milho_df['milho_reais']==milho_df['milho_reais'].max()]
```

Out[586]:

Data	data	milho_reais	milho_dolares
2021-03-03	2021-03-03	87.06	15.14

```
In [587]: milho_df[milho_df['milho_dolares']==milho_df['milho_dolares'].max()]
```

Out[587]:

Data	data	milho_reais	milho_dolares
2011-07-01	2011-07-01	31.08	19.96

```
In [588]: milho_df[milho_df['milho_reais']==milho_df['milho_reais'].min()]
```

Out[588]:

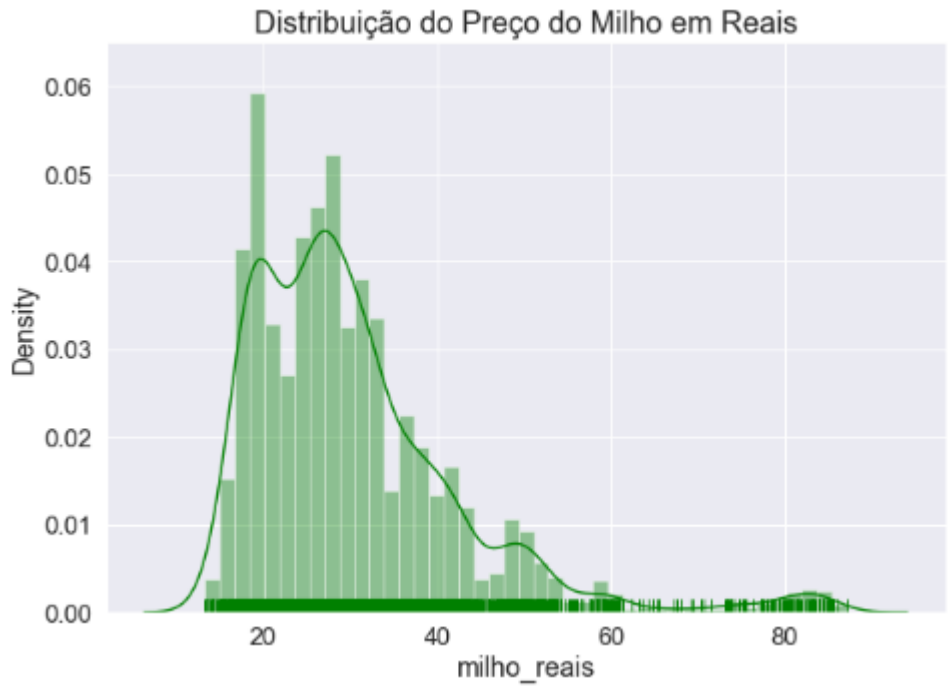
Data	data	milho_reais	milho_dolares
2006-03-30	2006-03-30	13.32	6.08

```
In [589]: milho_df[milho_df['milho_dolares']==milho_df['milho_dolares'].min()]
```

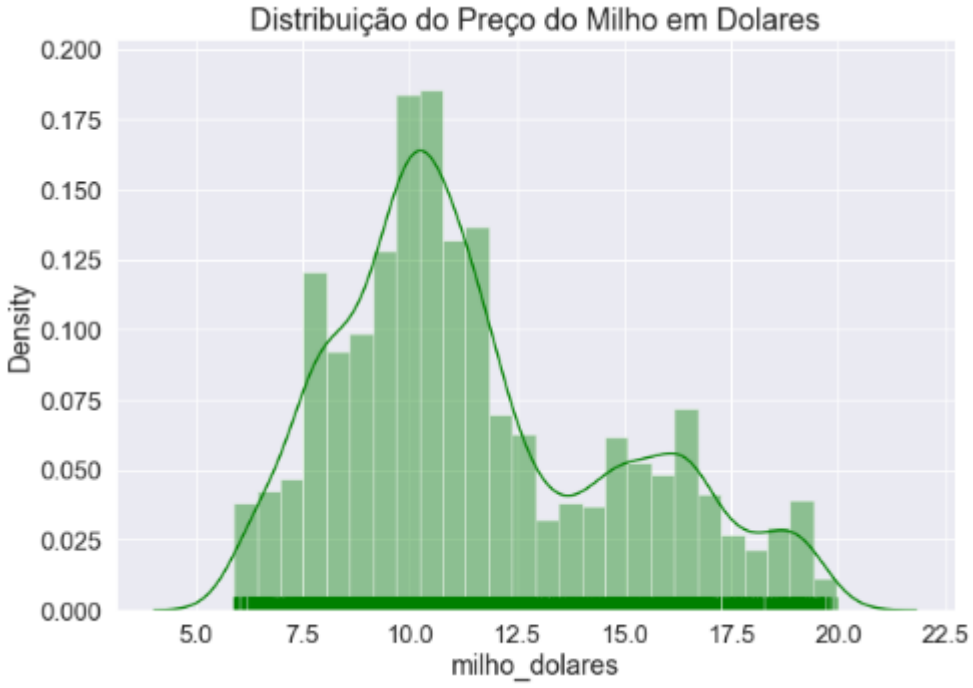
Out[589]:

Data	data	milho_reais	milho_dolares
2004-08-05	2004-08-05	18.06	5.89

O histograma da distribuição de frequência do preço milho em reais mostra que temos dois picos, um na região dos 20 reais e outro em 32 reais:



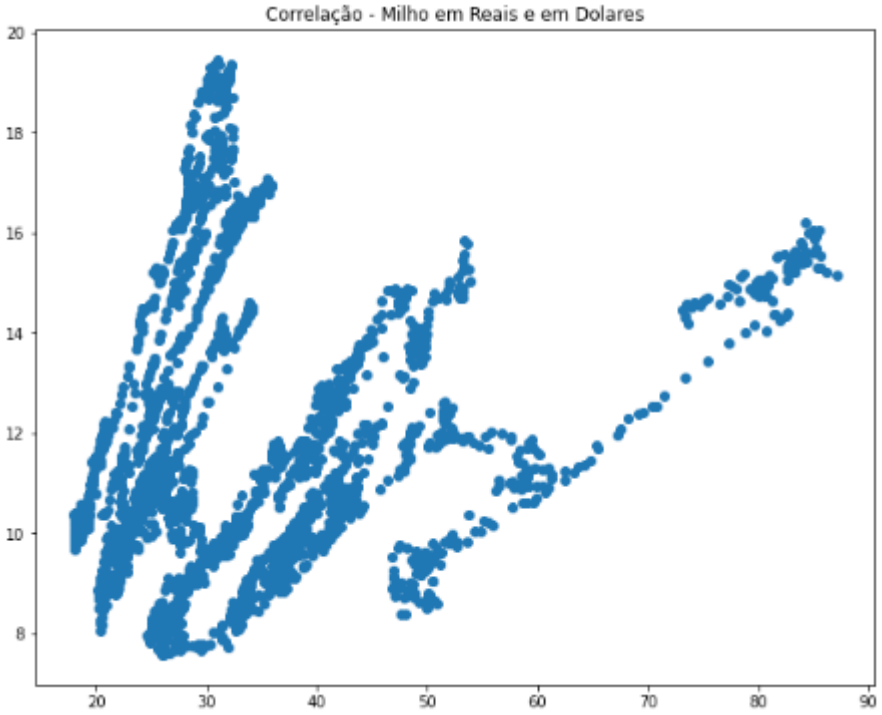
Já a histograma da distribuição de frequência do preço do milho em dólares mostra um pico na região dos 10 dólares:



A correlação entre os preços do milho em reais e em dólares é baixa:

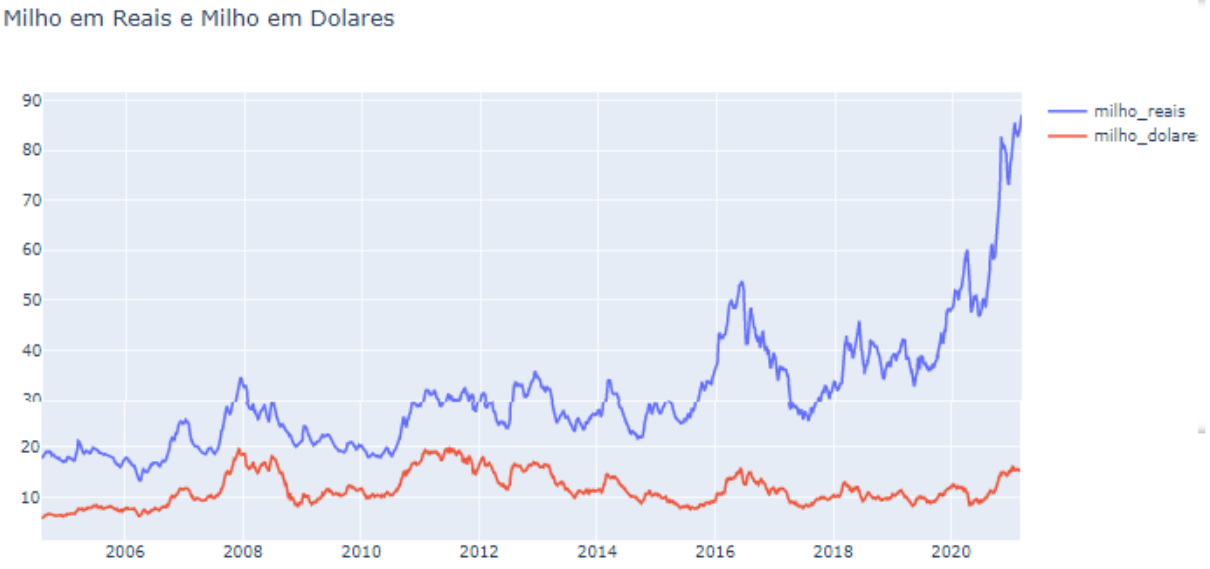
```
In [107]: mc_df['milho_reais'].corr(mc_df['milho_dolares'])
Out[107]: 0.1816166279649749
```

Pelo gráfico, verificamos que a correlação entre os dois preços não é uniforme, o que explica o baixo valor de correlação. É possível encontrarmos 3 retas com inclinação positiva e verificar que a angulação de cada reta se altera no tempo:



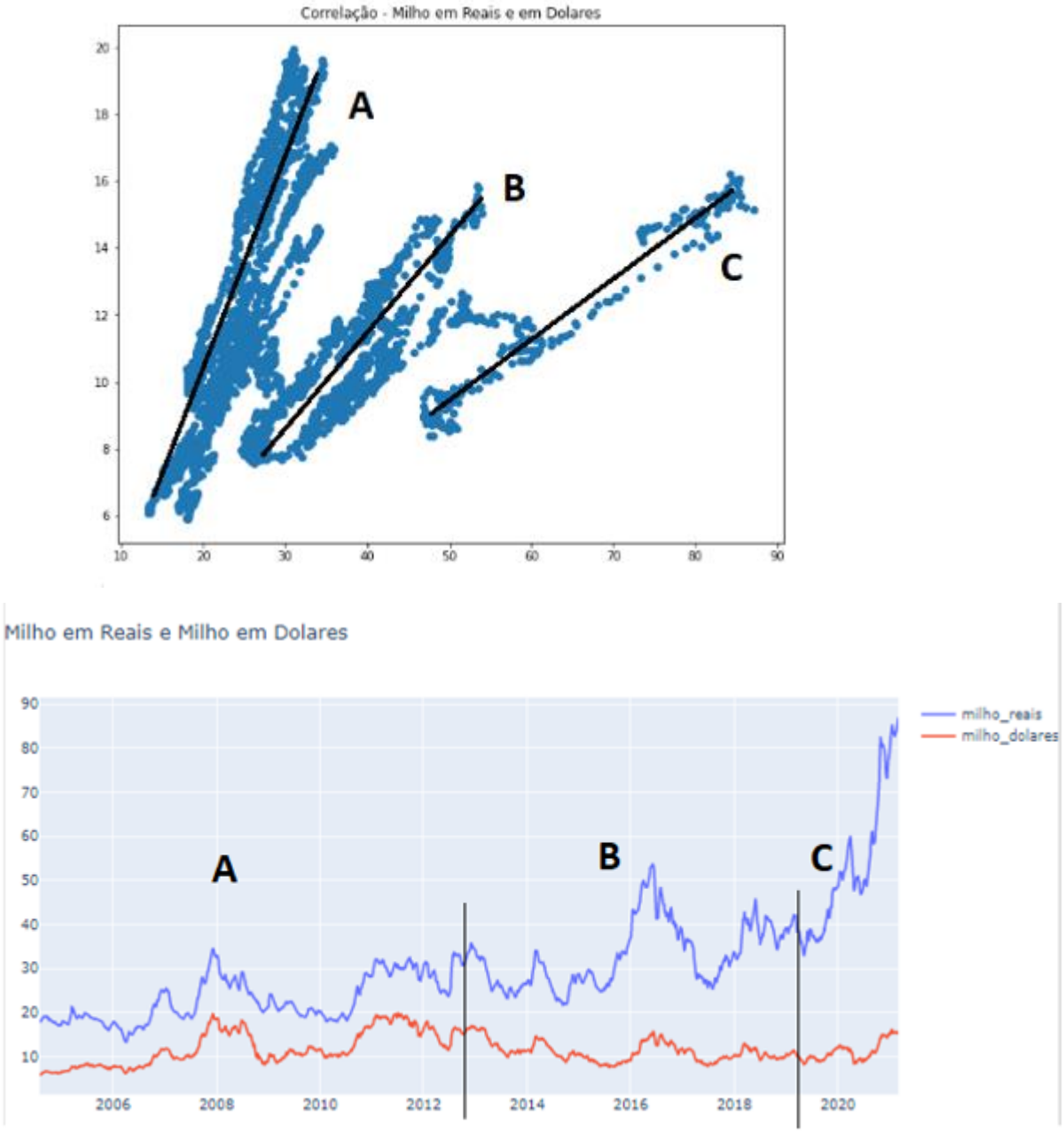
Criamos uma função para visualizar os dados da série histórica de preços do milho usando o Plotly:

```
In [ ]: def interactive_plot(df, title):
          fig = px.line(title = title)
          for i in df.columns[1:]:
            fig.add_scatter(x = df['data'], y = df[i], name = i)
          fig.show()
```



Percebe-se que os preços em dólar e milho se comportam da mesma forma até meados de 2013, após esse período há um movimento descendente em dólar e um ascendente em reais até 2016. Após 2016 temos uma grande alta no preço em reais e uma pequena alta em dólares.

Uma tentativa de relacionar as diferentes correlações no gráfico de dispersão e na série de preços foi feita na ferramenta Paint e está exibida na figura abaixo:



Seria a variação cambial (Dólar/Real) a causa da variação da correlação? Iremos verificar essa possibilidade analisando os dados históricos do dólar no período

de 02/08/2004 a 03/03/2021. Para isso, baixamos o histórico em formato .csv (USD_BRL Dados Históricos.csv) no endereço https://br.investing.com/currencies/usd-brl-historical-data .

```
In [235]: dolar_df = pd.read_csv('USD_BRL Dados Históricos.csv', decimal=",")
          dolar_df
```

Out[235]:

	Data	Último	Abertura	Máxima	Mínima	Var%
0	03.03.2021	5.6193	5.6872	5.7729	5.5806	-1,01%
1	02.03.2021	5.6764	5.6386	5.7327	5.6386	0,61%
2	01.03.2021	5.6418	5.5870	5.6427	5.5553	0,77%
3	26.02.2021	5.5986	5.5340	5.6093	5.4905	1,23%
4	25.02.2021	5.5308	5.4450	5.5390	5.4173	2,30%
...	...	...	...	...	...	...
4319	06.08.2004	3.0330	3.0722	3.0780	3.0300	-1,25%
4320	05.08.2004	3.0713	3.0540	3.0713	3.0500	0,58%
4321	04.08.2004	3.0537	3.0500	3.0660	3.0460	0,12%
4322	03.08.2004	3.0500	3.0450	3.0620	3.0440	0,11%
4323	02.08.2004	3.0465	3.0365	3.0585	3.0365	0,30%

4324 rows × 6 columns

Analisamos os tipos de dados e excluímos as colunas referentes aos preços do dólar na abertura, máxima, mínima e variação (Var%).

```
In [236]: dolar_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4324 entries, 0 to 4323
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Data      4324 non-null   object
 1   Último    4324 non-null   float64
 2   Abertura  4324 non-null   float64
 3   Máxima    4324 non-null   float64
 4   Mínima    4324 non-null   float64
 5   Var%      4324 non-null   object
dtypes: float64(4), object(2)
memory usage: 202.8+ KB
```

```
In [238]: dolar_df = dolar_df.drop(columns=dolar_df.columns[2:])
          dolar_df
```

Out[238]:

	Data	Último
0	03.03.2021	5.6193
1	02.03.2021	5.6764
2	01.03.2021	5.6418
3	26.02.2021	5.5986
4	25.02.2021	5.5308
...	...	...
4319	06.08.2004	3.0330
4320	05.08.2004	3.0713
4321	04.08.2004	3.0537
4322	03.08.2004	3.0500
4323	02.08.2004	3.0465

4324 rows × 2 columns

Em seguida alteramos o formato das datas para YYYY-MM-DD e em ordem crescente.

```
In [239]: dolar_df ['Data'] = pd.to_datetime(dolar_df ['Data'],dayfirst=True)
          dolar_df = dolar_df.sort_values(by = ['Data'])
          dolar_df['Data']

Out[239]: 4323   2004-08-02
          4322   2004-08-03
          4321   2004-08-04
          4320   2004-08-05
          4319   2004-08-06
                    ...
          4      2021-02-25
          3      2021-02-26
          2      2021-03-01
          1      2021-03-02
          0      2021-03-03
          Name: Data, Length: 4324, dtype: datetime64[ns]
```

Depois, definimos as datas como index:

```
In [149]: dolar_df.index = pd.to_datetime(dolar_df.Data)
          dolar_df.index.to_period('D')
          dolar_df.index

Out[149]: DatetimeIndex(['2004-08-02', '2004-08-03', '2004-08-04', '2004-08-05',
                         '2004-08-06', '2004-08-09', '2004-08-10', '2004-08-11',
                         '2004-08-12', '2004-08-13',
                         ...
                         '2021-02-18', '2021-02-19', '2021-02-22', '2021-02-23',
                         '2021-02-24', '2021-02-25', '2021-02-26', '2021-03-01',
                         '2021-03-02', '2021-03-03'],
                        dtype='datetime64[ns]', name='Data', length=4324, freq=None)
```

Em seguida, alteramos os nomes das colunas:

```
In [152]: dolar_df.rename(columns= {'Data': 'data'}, inplace=True)
          dolar_df.rename(columns= {'Último': 'Dolar_Último'}, inplace=True)
```

O dólar_df tem 4.324 linhas e duas colunas enquanto o milho_df tem 4.127 linhas e 3 colunas.

```
In [253]: dolar_df
```

Out[253]:

Data	data	Dolar_Último
2004-08-02	2004-08-02	3.0465
2004-08-03	2004-08-03	3.0500
2004-08-04	2004-08-04	3.0537
2004-08-05	2004-08-05	3.0713
2004-08-06	2004-08-06	3.0330
...	...	...
2021-02-25	2021-02-25	5.5308
2021-02-26	2021-02-26	5.5986
2021-03-01	2021-03-01	5.6418
2021-03-02	2021-03-02	5.6764
2021-03-03	2021-03-03	5.6193

4324 rows × 2 columns

```
In [254]: milho_df
```

Out[254]:

Data	data	milho_reais	milho_dolares
2004-08-02	2004-08-02	18.24	5.98
2004-08-03	2004-08-03	18.04	5.91
2004-08-04	2004-08-04	18.02	5.90
2004-08-05	2004-08-05	18.06	5.89
2004-08-06	2004-08-06	18.13	5.98
...	...	...	...
2021-02-25	2021-02-25	85.59	15.55
2021-02-26	2021-02-26	85.41	15.30
2021-03-01	2021-03-01	85.59	15.29
2021-03-02	2021-03-02	86.11	15.20
2021-03-03	2021-03-03	87.06	15.14

4127 rows × 3 columns

Considerando que o nosso foco é no valor do câmbio no dia da negociação da saca do milho iremos usar o milho_df como referência para fundir os dois dataframes. A colunas coincidentes receberão os sufixos _M (milho_df) e _D (dólar_df) para distinção.

```
In [255]: dolar_milho_df = pd.merge(milho_df,dolar_df, how='inner', on=['Data'],suffixes=('_M', '_D'))
          dolar_milho_df
```

Out[255]:

Data	data_M	milho_reais	milho_dolares	data_D	Dolar_Último
2004-08-02	2004-08-02	18.24	5.98	2004-08-02	3.0465
2004-08-03	2004-08-03	18.04	5.91	2004-08-03	3.0500
2004-08-04	2004-08-04	18.02	5.90	2004-08-04	3.0537
2004-08-05	2004-08-05	18.06	5.89	2004-08-05	3.0713
2004-08-06	2004-08-06	18.13	5.98	2004-08-06	3.0330
...	...	...	...	...	...
2021-02-25	2021-02-25	85.59	15.55	2021-02-25	5.5308
2021-02-26	2021-02-26	85.41	15.30	2021-02-26	5.5986
2021-03-01	2021-03-01	85.59	15.29	2021-03-01	5.6418
2021-03-02	2021-03-02	86.11	15.20	2021-03-02	5.6764
2021-03-03	2021-03-03	87.06	15.14	2021-03-03	5.6193

4127 rows × 5 columns

Na sequência, excluiremos a coluna data_D por ser redundante:

```
In [256]: dolar_milho_df = dolar_milho_df.drop(columns='data_D')
          dolar_milho_df
```

Out[256]:

Data	data_M	milho_reais	milho_dolares	Dolar_Último
2004-08-02	2004-08-02	18.24	5.98	3.0465
2004-08-03	2004-08-03	18.04	5.91	3.0500
2004-08-04	2004-08-04	18.02	5.90	3.0537
2004-08-05	2004-08-05	18.06	5.89	3.0713
2004-08-06	2004-08-06	18.13	5.98	3.0330
...	...	...	...	...
2021-02-25	2021-02-25	85.59	15.55	5.5308
2021-02-26	2021-02-26	85.41	15.30	5.5986
2021-03-01	2021-03-01	85.59	15.29	5.6418
2021-03-02	2021-03-02	86.11	15.20	5.6764
2021-03-03	2021-03-03	87.06	15.14	5.6193

Calculamos a correlação entre o preço da saca de milho em reais e o dólar do dia:

```
In [257]: dolar_milho_df['milho_reais'].corr(dolar_milho_df['Dolar_Último'])
Out[257]: 0.7824021362013128
```

A correlação é forte, próxima a 80%. Podemos considerar que a divergência entre o preço do milho em dólares para reais é explicada principalmente pela variação cambial no período.

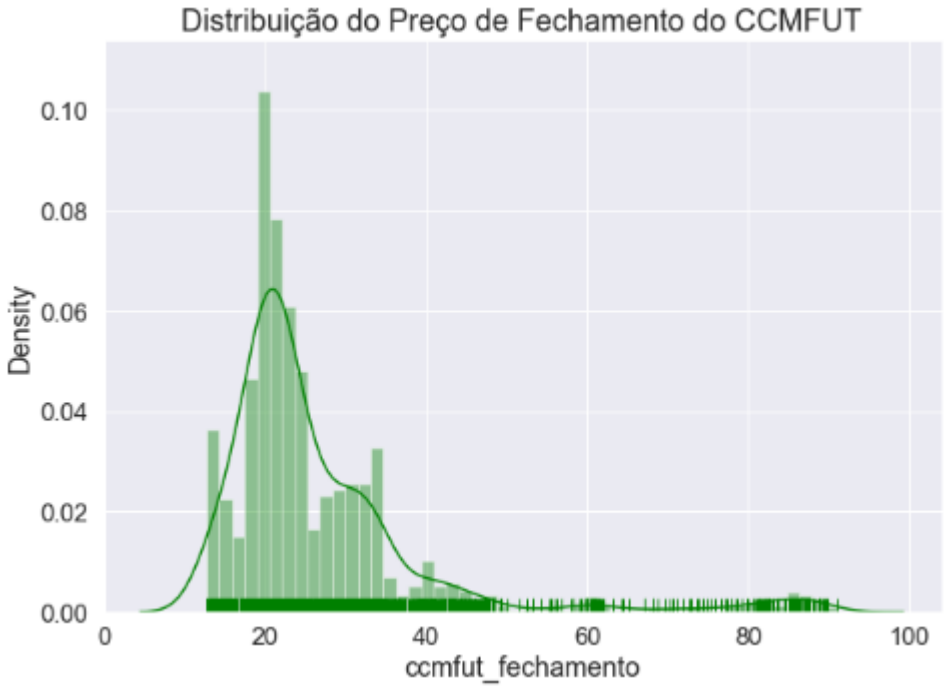
Continuaremos a análise e exploração dos dados com o resumo estatístico do ccmfut_df:

```
In [102]: ccmfut_df.describe()
```

Out[102]:

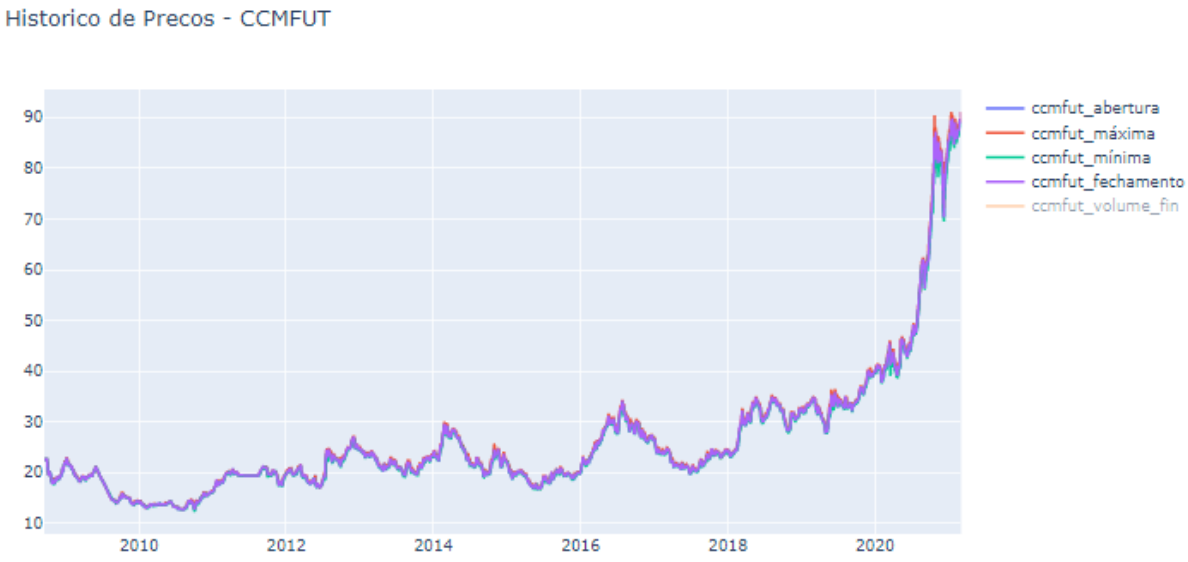
	ccmfut_abertura	ccmfut_máxima	ccmfut_mínima	ccmfut_fechamento	ccmfut_volume_fin
count	2917.000000	2917.000000	2917.000000	2917.000000	2.917000e+03
mean	26.723809	26.988920	26.474443	26.742749	3.300606e+07
std	13.506372	13.711821	13.312827	13.540517	4.274579e+07
min	12.680000	12.800000	12.310000	12.710000	0.000000e+00
25%	19.620000	19.780000	19.500000	19.640000	1.002070e+07
50%	22.560000	22.760000	22.350000	22.560000	2.244742e+07
75%	30.140000	30.510000	29.820000	30.140000	3.881796e+07
max	89.730000	91.120000	89.280000	91.050000	6.528013e+08

Plotamos o histograma do preço de fechamento, conforme figura abaixo:



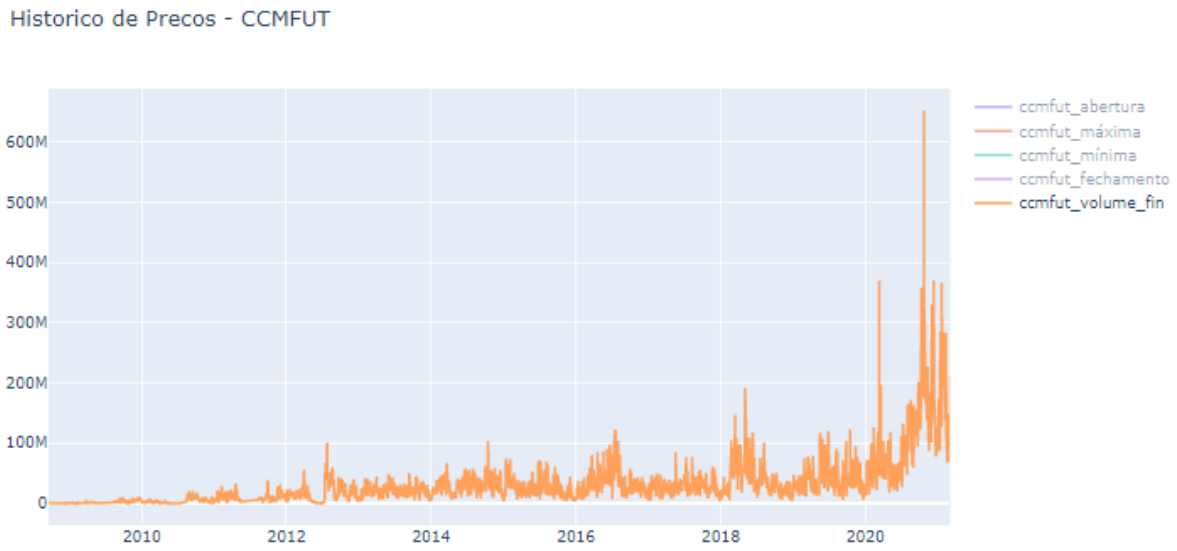
Verifica-se que há um pico na região dos 20 reais.

Plotamos também o histórico de preços, contendo os valores de preço de abertura, máxima, mínima e fechamento do CCMFUT na figura abaixo (desmarcamos o volume financeiro por possuir valores em escala diferente e prejudicar a visualização quando plotado conjuntamente):



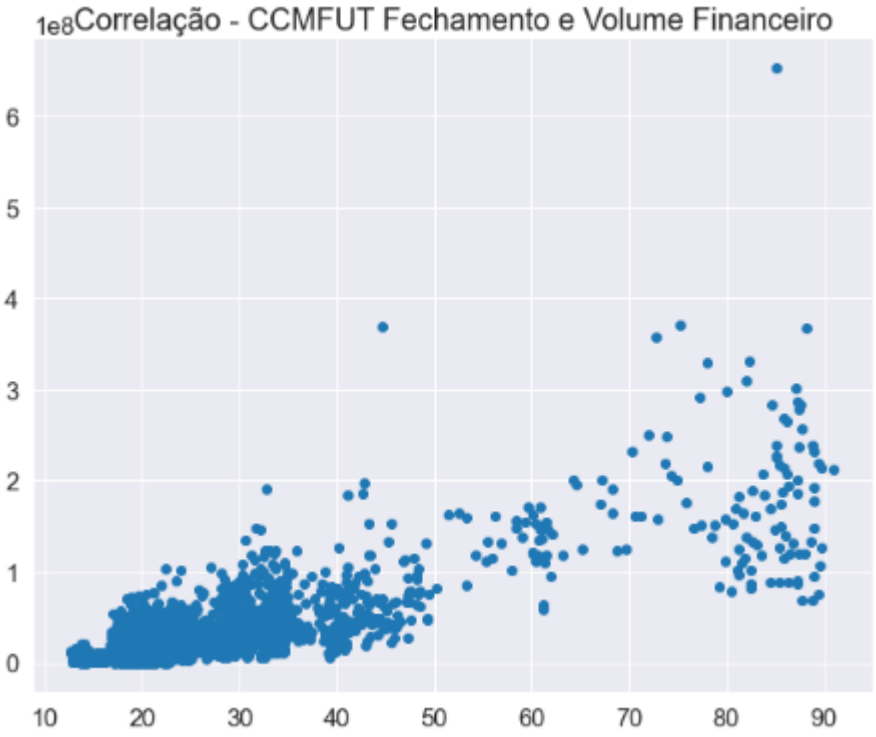
Verifica-se um aumento lento nos preços de 2011 a 2017. Após esse período, houve uma lateralização e um forte movimento de alta que se iniciou no final de 2019.

Em seguida, desmarcamos os preços e deixamos apenas o histórico do volume plotado. Verifica-se na imagem abaixo que tivemos um pico no volume em Março de 2020 e outro em Outubro de 2020:

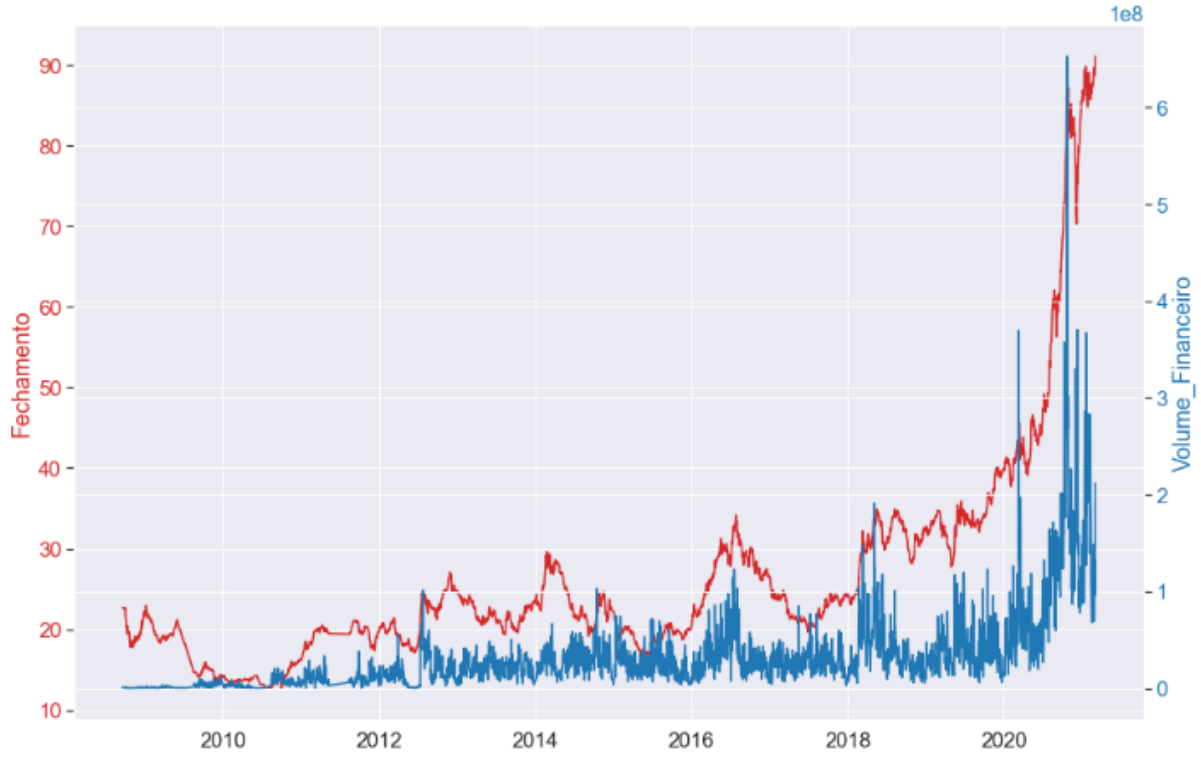


Haveria alguma correlação entre os preços de fechamento e o volume? O cálculo mostra haver uma correlação forte entre esses dois valores:

```
In [131]: ccmfut_df['ccmfut_fechamento'].corr(ccmfut_df['ccmfut_volume_fin'])
Out[131]: 0.7999203525936038
```



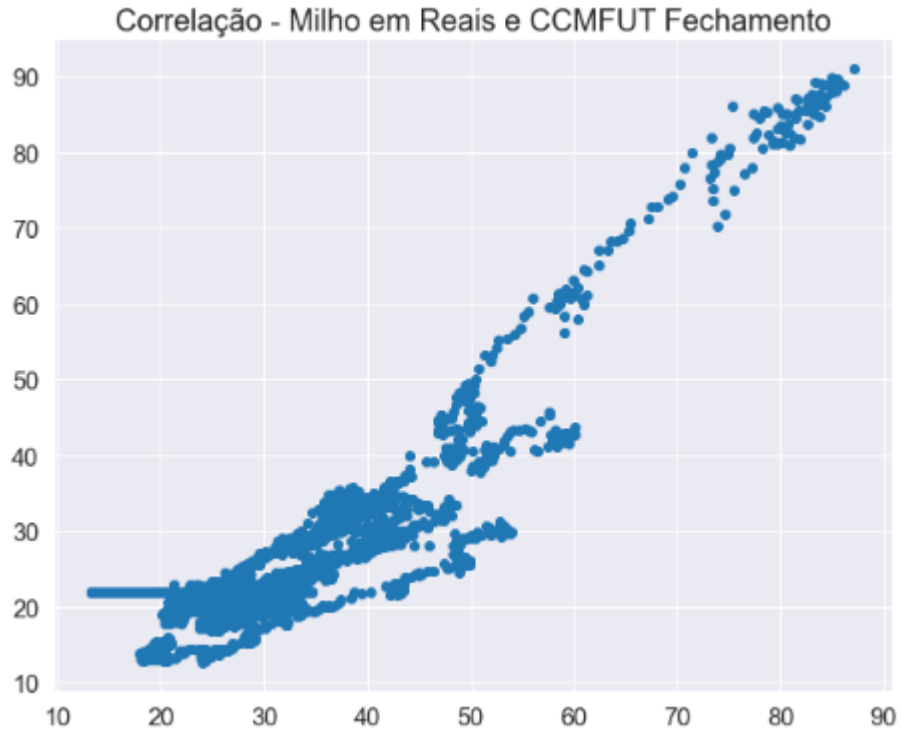
Para melhor visualização entre o preço de fechamento e volume, plotamos um gráfico com esses dois valores como eixo y:



Verifica-se que os alguns movimentos de alta no volume coincidem com os movimentos de alta nos preços. Além disso, alguns topos são formados conjuntamente com topos nos preços.

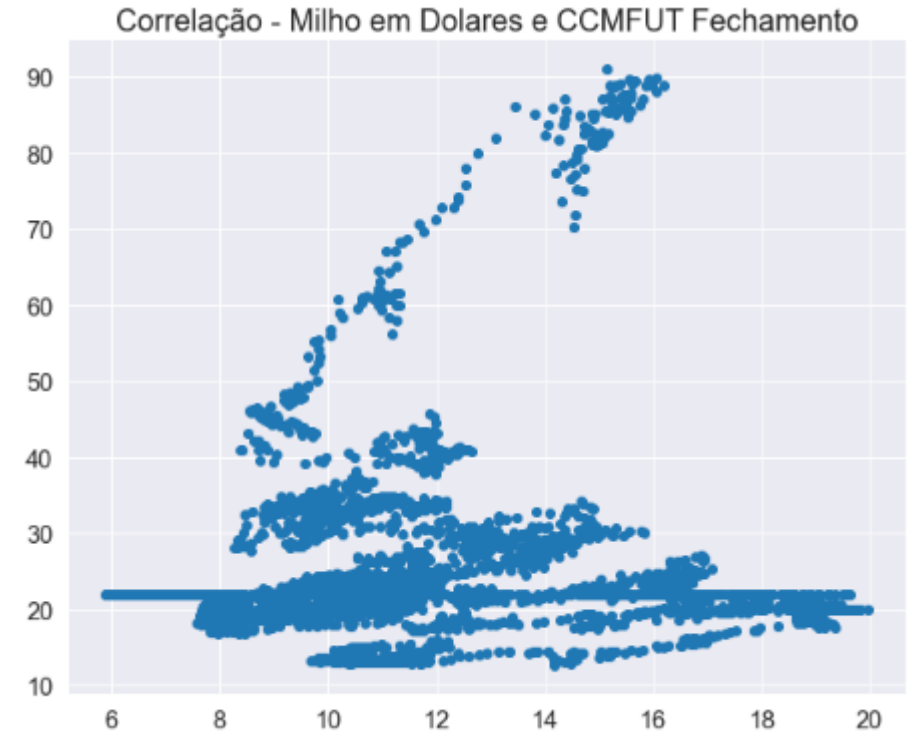
Por fim, em relação ao mc_df verificamos a correlação entre o valor do milho em reais e o preço de fechamento do contrato futuro de milho e plotamos o diagrama de dispersão:

```
In [287]: mc_df['milho_reais'].corr(mc_df['ccmfut_fechamento'])
Out[287]: 0.928074900682255
```



Já em relação ao preço do milho em dólares e o preço de fechamento do ccmfut, a correlação é baixa:

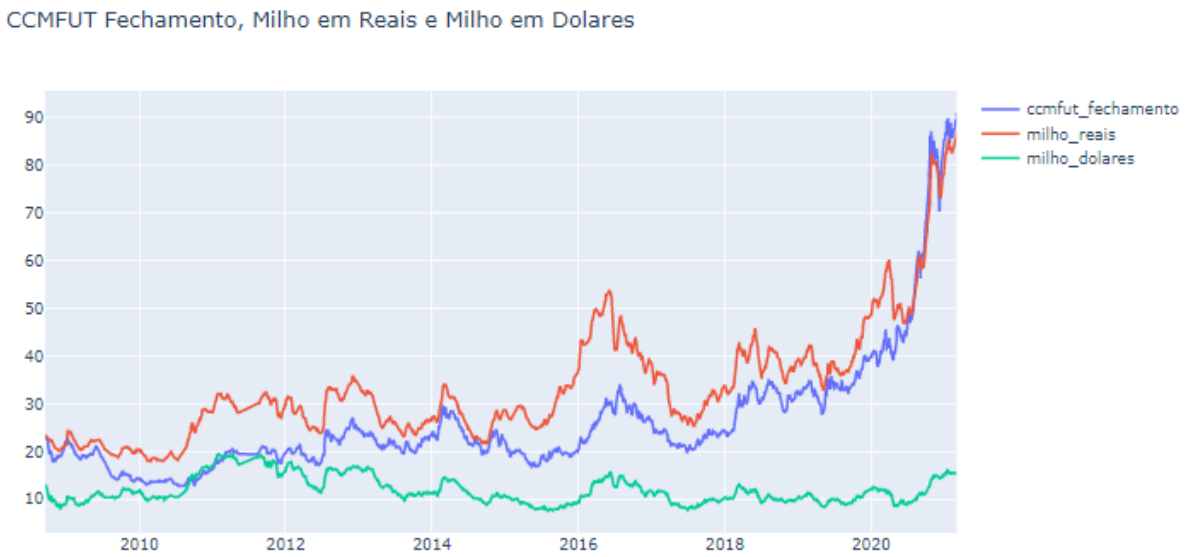
```
In [289]: mc_df['milho_dolares'].corr(mc_df['ccmfut_fechamento'])
Out[289]: 0.065512943089332
```



Para finalizar, alteramos a ordem das colunas do mc_df e plotamos a série de preços de fechamento do CCMFut, do preço da saca de milho em reais e em dólares para ver as relações:

```
In [365]: mc_df = mc_df[['data', 'ccmfut_abertura', 'ccmfut_máxima', 'ccmfut_mínima' , 'ccmfut_volume_fin', 'ccmfut_fechamento' , 'milho_r
```

mc_df = mc_df[['data', 'ccmfut_abertura', 'ccmfut_máxima', 'ccmfut_mínima' , 'ccmfut_volume_fin', 'ccmfut_fechamento' , 'milho_reais' , 'milho_dolares']]



Percebe-se que tanto o preço da saca de milho em reais quanto o fechamento do CCMFut seguem a mesma tendencia e os movimentos são bem parecidos. Por conta do nosso foco nos contratos futuros, criaremos modelos de machine learning para realizarem a predição dos preços desses contratos.

5. Criação de Modelos de Machine Learning

Neste tópico iremos descrever os modelos preditivos para os contratos futuros de milho criados em linguagem Python utilizando a biblioteca ScikitLearn e RNN (Recurrent Neural Network) utilizando a arquitetura LSTM (Long Short Term Memory) no Keras (TensorFlow).

5.1 – SikitLearn – Ridge Regression

Inicialmente, teremos que definir um alvo para o nosso modelo preditivo. Esperamos que o modelo consiga prever, na data atual, o preço do contrato futuro de milho em reais que do dia seguinte (fechamento). Retiramos a última linha, já que não haveria um alvo definido (não temos o preço do dia seguinte para defini-lo como alvo):

```
In [516]: mc_df['ccmfut_fechamento_alvo'] = mc_df[['ccmfut_fechamento']].shift(-1)
          mc_df = mc_df[:-1]
          mc_df
```

Out[516]:

data	data	ccmfut_abertura	ccmfut_máxima	ccmfut_mínima	ccmfut_volume_fin	ccmfut_fechamento	milho_reais	milho_dolares	ccmfut_fechamento_alvo
2008-09-19	2008-09-19	22.64	22.64	22.64	1129500.0	22.64	23.47	12.82	22.64
2008-09-22	2008-09-22	22.64	22.64	22.64	112950.0	22.64	23.31	13.00	22.68
2008-09-26	2008-09-26	22.68	22.68	22.68	565875.0	22.68	23.24	12.54	22.55
2008-09-30	2008-09-30	22.55	22.55	22.55	112500.0	22.55	22.99	12.08	22.64
2008-10-02	2008-10-02	22.64	22.64	22.64	11295.0	22.64	22.95	11.36	21.65
...	...	...	...	...	...	...	...	...	...
2021-02-24	2021-02-24	89.08	89.47	88.42	75830647.5	89.47	85.19	15.68	89.63
2021-02-25	2021-02-25	89.45	89.72	88.81	106361550.0	89.63	85.59	15.55	88.86
2021-02-26	2021-02-26	89.61	89.64	88.86	69200707.5	88.86	85.41	15.30	88.70
2021-03-01	2021-03-01	88.92	89.18	88.00	133054398.0	88.70	85.59	15.29	88.94
2021-03-02	2021-03-02	88.80	89.06	88.54	95796617.5	88.94	86.11	15.20	91.05

2916 rows × 9 columns

Em seguida, importamos a classe MinMaxScaler do pacote sklearn.preprocessing para padronizar o conjunto de dados. Como o volume financeiro tem uma escala diferente e alguns valores discrepantes, o algoritmo de aprendizagem precisa que os recursos variem em escalas comparáveis. Assim, criamos o objeto “sc” especificando o alcance dos recursos com o mínimo em 0 e o máximo em 1. O MinMaxScaler usa esses valores como padrão e não vemos necessidade em alterá-los ou utilizar outro valor. Por fim, passamos o método fit_transform para dimensionar os dados. Excluímos as datas desse processo, pois não há sentido em padronizar datas. O resultado é o array mc_df_scaled.

```
In [521]: from sklearn.preprocessing import MinMaxScaler
          sc = MinMaxScaler(feature_range = (0, 1))
          mc_df_scaled = sc.fit_transform(mc_df.drop(columns = ['data']))
```

O próximo passo é definir quais serão as entradas e saídas do nosso modelo. A entradas (recursos) serão os dados: ccmfut_abertura, ccmfut_máxima, ccmfut_mínima, ccmfut_volume_fin, ccmfut_fechamento, milho_reais e milho_dolares. Já a saída (alvo) sera o ccmfut_fechamento_alvo:

```
In [554]: X = mc_df_scaled[:,:7]
          y = mc_df_scaled[:,7:]
```

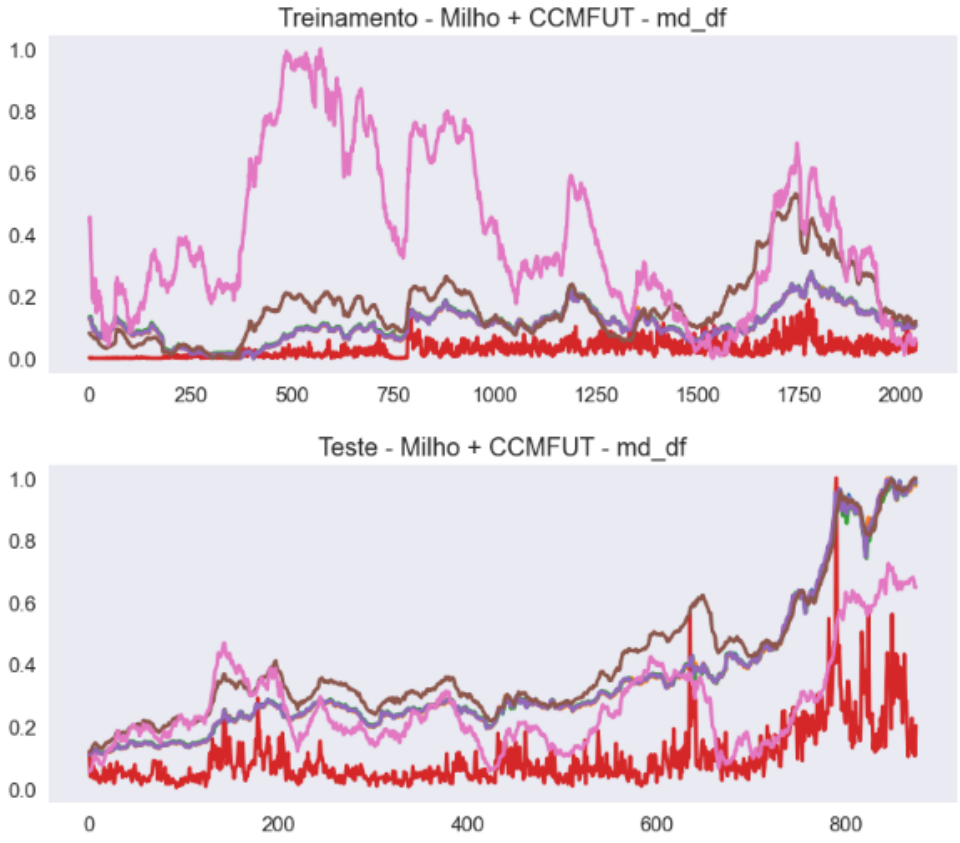
Convertemos os arrays para o formato correto:

```
In [304]: X = np.asarray(X)
          y = np.asarray(y)
          X.shape, y.shape

Out[304]: ((2916, 7), (2916, 1))
```

Agora dividiremos os dados na proporção de 70% para treino e 30% para testes do modelo, que é a proporção geralmente utilizada. Não usamos a função “train_test_split” por conta de alguns relatos de embaralhamento dos dados, o que não é desejado.

```
In [483]: split = int(0.70 * len(X))
          X_treino = X[:split]
          y_treino = y[:split]
          X_teste = X[split:]
          y_teste = y[split:]
```

Por fim, plotamos os dados de treinamento e teste:



Agora criaremos o nosso modelo e iremos treiná-lo.

Inicialmente, importaremos a classe Ridge do pacote sklearn.linear_model. Em seguida criamos o objeto regression_model especificando o Ridge com os seus parâmetros padrões (alpha = 1). Por fim, aplicamos o método fit ao objeto passando os dados de treino como parâmetros:

```
In [574]: from sklearn.linear_model import Ridge
          regression_model = Ridge ()
          regression_model.fit(X_treino, y_treino)

Out[574]: Ridge()
```

Na sequência, testamos o modelo e calculamos a sua acurácia, usando os dados de teste:

```
In [575]: lr_accuracy = regression_model.score(X_teste, y_teste)
          print("Linear Regression Score: ", lr_accuracy)

          Linear Regression Score:  0.9875266301913911
```

O parâmetro alpha determina a força de regularização. A regularização melhora o condicionamento do problema e reduz a variância das estimativas. Usando o valor padrão (1) temos uma acurácia de 0.9875 e um MSE igual a 0.0001576. Aumentando o alpha para 2 temos uma acurácia de 0.9695 e MSE de 0.0004395. Alterando o alpha para 5, temos a uma acurácia menor (0.8990) e um MSE = 0.001557. Por outro lado, diminuindo o alpha para 0.5 temos 0.9939 de acurácia (a melhor dentre os valores testados). Porém, o MSE é igual a 6.01238. Utilizaremos o valor 1 para alpha, pois graficamente percebeu-se melhores resultados do que alpha 2, apesar deste apresentar um MSE menor.

O próximo passo é aplicar o modelo em todos os dados. O resultado será um array.

```
In [576]: predicted_prices_mc = regression_model.predict(X)
          predicted_prices_mc

Out[576]: array([[0.12251923],
                 [0.12236751],
                 [0.12286197],
                 ...,
                 [0.93493601],
                 [0.93128384],
                 [0.92792567]])
```

Treinaremos o modelo unicamente com os dados de fechamento para verificar como se comporta.

Utilizaremos o mc_df e iremos refazer os mesmos passos anteriores. A única diferença é que teremos que excluir as colunas desnecessárias do dataframe:

In [321]:
```
mc_df = mc_df.drop(columns='ccmfut_abertura')
mc_df = mc_df.drop(columns='ccmfut_máxima')
mc_df = mc_df.drop(columns='ccmfut_mínima')
mc_df = mc_df.drop(columns='ccmfut_volume_fin')
mc_df = mc_df.drop(columns='milho_reais')
mc_df = mc_df.drop(columns='milho_dolares')
mc_df
```

Out[321]:

	data	ccmfut_fechamento	ccmfut_fechamento_alvo
data			
2008-09-19	2008-09-19	22.64	22.64
2008-09-22	2008-09-22	22.64	22.68
2008-09-26	2008-09-26	22.68	22.55
2008-09-30	2008-09-30	22.55	22.64
2008-10-02	2008-10-02	22.64	21.65
...	...	...	...
2021-02-24	2021-02-24	89.47	89.63
2021-02-25	2021-02-25	89.63	88.86
2021-02-26	2021-02-26	88.86	88.70
2021-03-01	2021-03-01	88.70	88.94
2021-03-02	2021-03-02	88.94	91.05

2916 rows × 3 columns

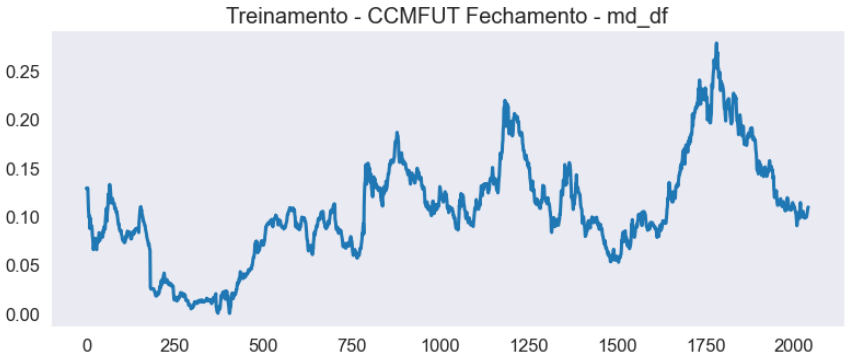
In [322]:
```
mc_df_scaled_fech = sc.fit_transform(mc_df.drop(columns = ['data']))
```

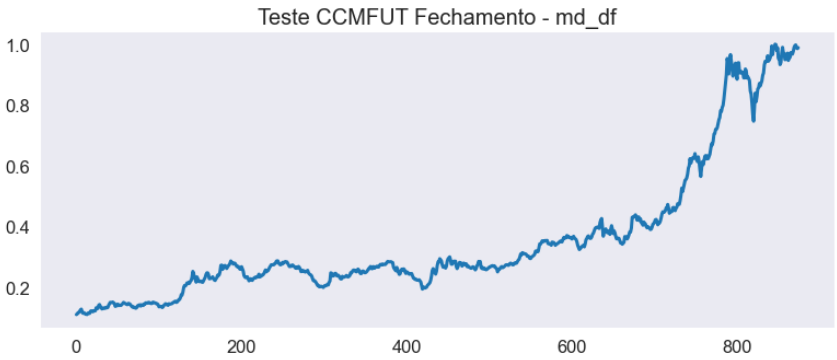
In [323]:
```
X = mc_df_scaled_fech[:,:1]
y = mc_df_scaled_fech[:,1:]
```

In [324]:
```
X = np.asarray(X)
y = np.asarray(y)
```

In [325]:
```
split = int(0.70 * len(X))
X_treino = X[:split]
y_treino = y[:split]
X_teste = X[split:]
y_teste = y[split:]
```

Plotamos o gráfico dos dados de treinamento e teste para verificar que estamos usando unicamente os dados do fechamento:






```
In [327]: regression_model.fit(X_treino, y_treino)

Out[327]: Ridge(alpha=1)

In [328]: lr_accuracy = regression_model.score(X_teste, y_teste)
          print("Linear Regression Score: ", lr_accuracy)

          Linear Regression Score:  0.9484580850390733

In [329]: predicted_prices_mc = regression_model.predict(X)
          predicted_prices_mc

Out[329]: array([[0.12355197],
                 [0.12355197],
                 [0.12398756],
                 ...,
                 [0.84466647],
                 [0.84292412],
                 [0.84553764]])
```

5.2 – Keras – LSTM

Iremos utilizar apenas os dados de preço de fechamento do ccmfut do mc_df então descartaremos as demais colunas:

```
In [321]: mc_df = mc_df.drop(columns='ccmfut_abertura')
          mc_df = mc_df.drop(columns='ccmfut_máxima')
          mc_df = mc_df.drop(columns='ccmfut_mínima')
          mc_df = mc_df.drop(columns='ccmfut_volume_fin')
          mc_df = mc_df.drop(columns='milho_reais')
          mc_df = mc_df.drop(columns='milho_dolares')
          mc_df = mc_df.drop(columns='ccmfut_fechamento_alvo')
          mc_df
```

Out[321]:

	data	ccmfut_fechamento
data		
2008-09-19	2008-09-19	22.64
2008-09-22	2008-09-22	22.64
2008-09-26	2008-09-26	22.68
2008-09-30	2008-09-30	22.55
2008-10-02	2008-10-02	22.64
...	...	...
2021-02-23	2021-02-23	89.01
2021-02-24	2021-02-24	89.47
2021-02-25	2021-02-25	89.63
2021-02-26	2021-02-26	88.86
2021-03-01	2021-03-01	88.70

2915 rows × 2 columns

Em seguida, criaremos um array com os dados de fechamento:

```
In [322]: training_data = mc_df.iloc[:, 1:].values
          training_data

Out[322]: array([[22.64],
                 [22.64],
                 [22.68],
                 ...,
                 [89.63],
                 [88.86],
                 [88.7 ]])
```

Da mesma forma que no modelo anterior, utilizaremos a classe MinMaxScaler do pacote sklearn.preprocessing para padronizar o conjunto de dados, para melhor desempenho. Assim, criamos o objeto “sc” especificando o alcance dos recursos com o mínimo em 0 e o máximo em 1. O MinMaxScaler usa esses valores como padrão e não vemos necessidade em alterá-los ou utilizar outro valor. Por fim, passamos o método fit_transform para dimensionar os dados. O resultado é o array training_set_scaled.

```
In [324]: from sklearn.preprocessing import MinMaxScaler
          sc = MinMaxScaler(feature_range = (0, 1))
          training_set_scaled = sc.fit_transform(training_data)
```

Na sequência, criaremos dois arrays sendo um referente aos recursos (X) e o outro sendo o alvo (y), que será o preço de fechamento do dia seguinte.

```
In [326]: X = []
          y = []
          for i in range(1, len(mc_df)):
              X.append(training_set_scaled [i-1:i, 0])
              y.append(training_set_scaled [i, 0])
```

Convertemos os arrays para o formato correto:

```
In [330]: X = np.asarray(X)
          y = np.asarray(y)
```

Agora dividiremos os dados na proporção de 70% para treino e 30% para testes do modelo:

```
In [332]: split = int(0.7 * len(X))
          X_train = X[:split]
          y_train = y[:split]
          X_test = X[split:]
          y_test = y[split:]
```

O modelo necessita ser alimentado por arrays de 3ª dimensão no formato batch. Usaremos a função reshape do numpy para adicionar uma dimensão nos dados de treinamento e de teste:

```
In [334]: X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
          X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
          X_train.shape, X_test.shape
Out[334]: ((2039, 1, 1), (875, 1, 1))
```

Agora criaremos o modelo. Não há consenso sobre qual a melhor quantidade de camadas, devendo ser definida caso a caso. Encontrarmos diversos artigos utilizando 1 camada de entrada, 3 camadas LSTM intercaldas por 2 camadas Dropout e 1 camada Dense de saida e procederemos dessa forma.

Inicialmente, definimos a camada de entrada, especificando o formato da dimensão com os índices 1 e 2 do array dos dados de treinamento.

Em seguida, definimos a 2ª camada como uma rede LSTM com 150 unidades (neurons) e passamos os inputs. Ajustamos o parâmetro return sequences igual a True para evitar problemas de dimensionalidade. É um boleano que define se deve retornar a última saída ou não. Realizamos testes com 10, 50, 100, 150 e 200 unidades e a que teve o menor MSE foi o modelo com 150 unidades.

A camada Dropout define aleatoriamente as unidades de entrada para 0 com uma frequência de taxa em cada etapa durante o tempo de treinamento, o que ajuda a prevenir overfitting. Srivastava, em seu trabalho “Dropout: A Simple Way to Prevent Neural Networks from Overfitting” descobriu empiricamente que o melhor valor para o Dropout seria 0.5 (o que significa que 50% das camadas serão descartadas) e por

isso utilizaremos esse valor no nosso modelo. Apesar disso, testamos os valores 0.1, 0.2 , 0.3, 0.4 e 0.5 e o que melhor performou foi o com o Dropout igual a 0,3.

Depois incluímos mais uma camada LSTM, outra camada Dropout e mais uma camada LSTM com os mesmos parâmetros anteriores.

Por fim, adicionamos a camada Densa que especifica a saída de 1 unidade (neuron) e função de ativação linear. É importante ser linear porque a saída será um valor contínuo (não é recomendada uma ativação sigmoide porque ela satura).

Definidas das camadas, criamos o modelo informando quais são as entradas e saídas. Depois compilamos o modelo, especificando o tipo de otimizador como “adam” que é o mais comum e a perda (mse), que é a função que queremos minimizar.

Finalmente, temos um sumario do modelo:

```
In [614]: inputs = keras.layers.Input(shape=(X_train.shape[1], X_train.shape[2]))
          x = keras.layers.LSTM(150, return_sequences= True)(inputs)
          x = keras.layers.Dropout(0.3)(x)
          x = keras.layers.LSTM(150, return_sequences=True)(x)
          x = keras.layers.Dropout(0.3)(x)
          x = keras.layers.LSTM(150)(x)
          outputs = keras.layers.Dense(1, activation='linear')(x)

          model = keras.Model(inputs=inputs, outputs=outputs)
          model.compile(optimizer='adam', loss="mse")
          model.summary()
```

```
Model: "model_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_13 (InputLayer)        [(None, 1, 1)]            0
_________________________________________________________________
lstm_36 (LSTM)               (None, 1, 150)            91200
_________________________________________________________________
dropout_24 (Dropout)         (None, 1, 150)            0
_________________________________________________________________
lstm_37 (LSTM)               (None, 1, 150)            180600
_________________________________________________________________
dropout_25 (Dropout)         (None, 1, 150)            0
_________________________________________________________________
lstm_38 (LSTM)               (None, 150)               180600
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 151
=================================================================
Total params: 452,551
Trainable params: 452,551
Non-trainable params: 0
_________________________________________________________________
```

Para treinar o modelo, passamos ao método fit os dados de X_train e y_train. Depois, definimos a quantidade de epochs (20), o tamanho batch (32) e o validation split (subconjunto dos dados de treinamento em que é feita uma validação cruzada para garantir que não haja overfitting no treinamento).

Aumentamos as epochs de 2 para 5, depois para 10 e por fim até 20 e verificamos a diminuição do loss. Não aumentamos mais para evitar overfitting. Definimos como 32 o número de amostras a serem trabalhadas antes de atualizar os parâmetros do modelo interno (batch size). Testamos o número de amostras igual a 64 e o loss

foi maior. Por fim, definimos como 0.2 a fração dos dados de treinamento a serem usados como dados de validação (validation split).

```
In [616]: history = model.fit(
              X_train, y_train,
              epochs = 20,
              batch_size = 32,
              validation_split = 0.2
          )
```

```
Epoch 1/20
51/51 [==============================] - 1s 10ms/step - loss: 3.2821e-05 - val_loss: 2.5579e-05
Epoch 2/20
51/51 [==============================] - 0s 9ms/step - loss: 3.6369e-05 - val_loss: 2.5414e-05
Epoch 3/20
51/51 [==============================] - 0s 10ms/step - loss: 2.9674e-05 - val_loss: 2.6047e-05
Epoch 4/20
51/51 [==============================] - 0s 9ms/step - loss: 3.4842e-05 - val_loss: 4.7576e-05
Epoch 5/20
51/51 [==============================] - 0s 10ms/step - loss: 3.3675e-05 - val_loss: 3.8707e-05
Epoch 6/20
51/51 [==============================] - 0s 9ms/step - loss: 3.2118e-05 - val_loss: 2.3506e-05
Epoch 7/20
51/51 [==============================] - 0s 10ms/step - loss: 3.3943e-05 - val_loss: 2.7484e-05
Epoch 8/20
51/51 [==============================] - 0s 10ms/step - loss: 3.1106e-05 - val_loss: 2.3927e-05
Epoch 9/20
51/51 [==============================] - 0s 10ms/step - loss: 3.2564e-05 - val_loss: 2.8278e-05
Epoch 10/20
51/51 [==============================] - 0s 10ms/step - loss: 3.0375e-05 - val_loss: 2.7701e-05
Epoch 11/20
51/51 [==============================] - 0s 10ms/step - loss: 3.2696e-05 - val_loss: 4.7155e-05
Epoch 12/20
51/51 [==============================] - 0s 10ms/step - loss: 3.0182e-05 - val_loss: 4.5038e-05
Epoch 13/20
51/51 [==============================] - 0s 10ms/step - loss: 3.0007e-05 - val_loss: 2.5538e-05
Epoch 14/20
51/51 [==============================] - 0s 10ms/step - loss: 3.0091e-05 - val_loss: 2.5467e-05
Epoch 15/20
51/51 [==============================] - 0s 10ms/step - loss: 3.3065e-05 - val_loss: 3.4122e-05
Epoch 16/20
51/51 [==============================] - 0s 10ms/step - loss: 3.2252e-05 - val_loss: 2.6870e-05
Epoch 17/20
51/51 [==============================] - 0s 10ms/step - loss: 3.1925e-05 - val_loss: 3.2508e-05
Epoch 18/20
51/51 [==============================] - 1s 10ms/step - loss: 2.8902e-05 - val_loss: 3.6465e-05
Epoch 19/20
51/51 [==============================] - 1s 10ms/step - loss: 3.0221e-05 - val_loss: 2.4646e-05
Epoch 20/20
51/51 [==============================] - 0s 10ms/step - loss: 3.5253e-05 - val_loss: 3.0769e-05
```

Finalmente, faremos as predições:

```
In [617]: predicted = model.predict(X)
```

6. Apresentação dos Resultados

A seguir serão apresentados os resultados dos modelos.

6.1 – Modelo Preditivo – Ridge Regression

Para apresentar o gráfico dos resultados do modelo, seguimos os passos abaixo.

Inicialmente, criamos uma lista e incluímos os dados preditos:

```
In [579]: predicted_mc = []
          for i in predicted_prices_mc:
            predicted_mc.append(i[0])
```

Em seguida criamos outra lista, com o os valores de fechamento (ccmfut):

```
In [580]: mc_fechamento = []
          for i in mc_df_scaled:
            mc_fechamento.append(i[4])
```

Por fim, criamos um dataframe com a data, os dados de fechamento e os valores preditos para o fechamento.

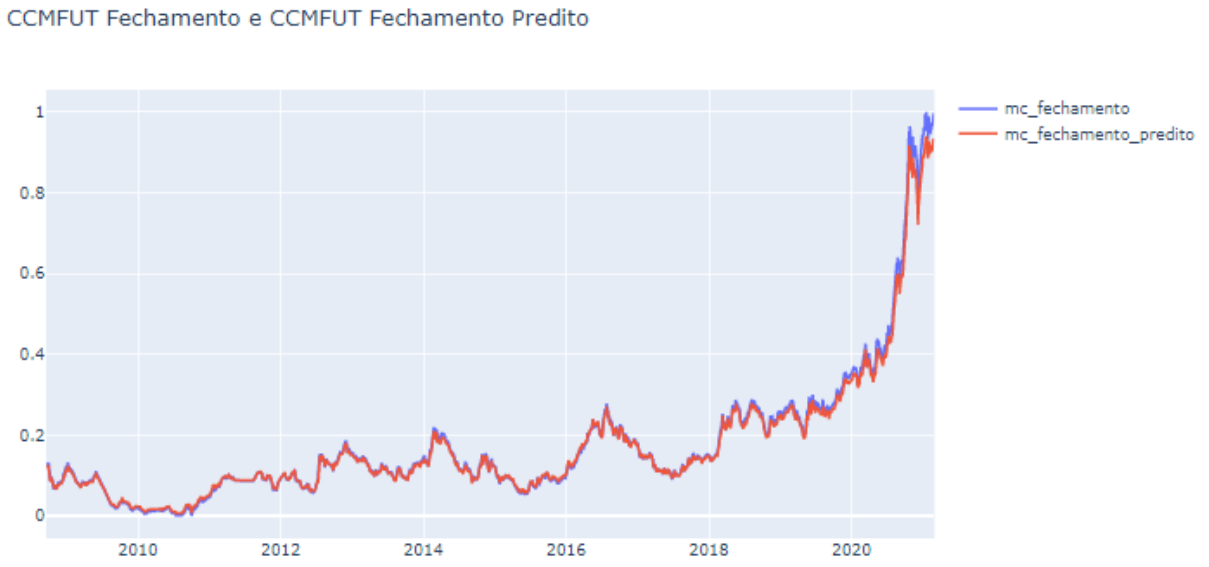
```
In [581]: mc_predicao = pd.DataFrame(columns = ['data' , 'mc_fechamento', 'mc_fechamento_predito'])
          mc_predicao['data'] = mc_df['data']
          mc_predicao['mc_fechamento'] = mc_fechamento
          mc_predicao['mc_fechamento_predito'] = predicted_mc
          mc_predicao
```

Out[581]:

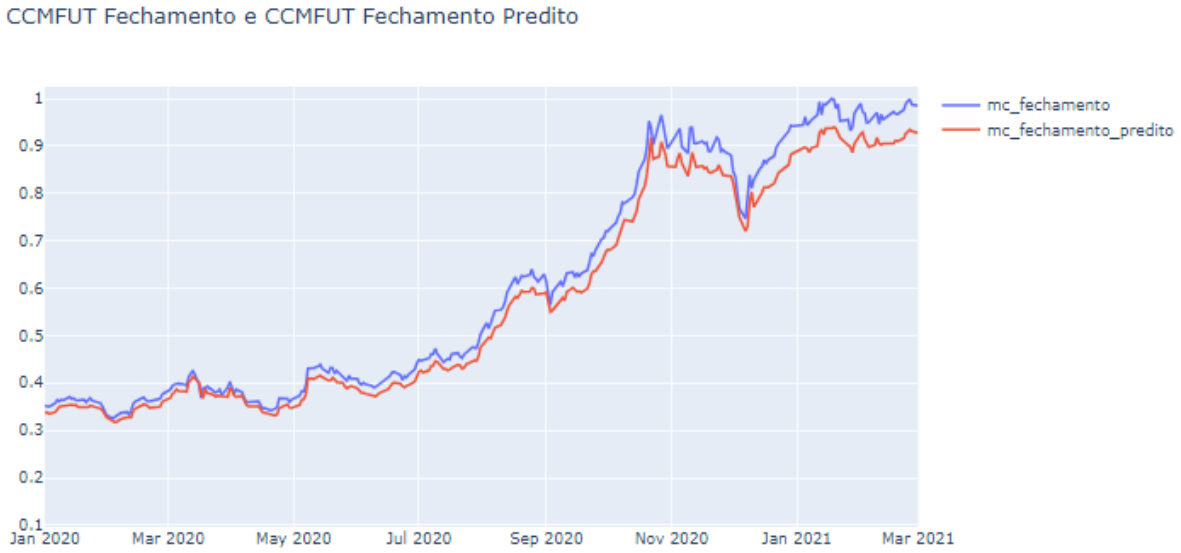
data	data	mc_fechamento	mc_fechamento_predito
2008-09-19	2008-09-19	0.128777	0.122519
2008-09-22	2008-09-22	0.128777	0.122368
2008-09-26	2008-09-26	0.129296	0.122862
2008-09-30	2008-09-30	0.127610	0.121247
2008-10-02	2008-10-02	0.128777	0.122342
...	...	...	...
2021-02-23	2021-02-23	0.989496	0.926798
2021-02-24	2021-02-24	0.995461	0.930015
2021-02-25	2021-02-25	0.997536	0.934936
2021-02-26	2021-02-26	0.987550	0.931284
2021-03-01	2021-03-01	0.985475	0.927926

2915 rows × 3 columns

O gráfico abaixo mostra os valores reais (mc_fechamento) de fechamento do CCMFut e os dados preditos.



Percebe-se que os dados previstos são bem próximos dos dados reais. Segue imagem aproximada do ano de 2020, que se destacava na figura anterior pelo espaçamento dos dados.



Finalmente, temos o erro encontrado:

In [247]:
```
#Cálculo do erro
mse = mean_squared_error(mc_predicao['mc_fechamento'], mc_predicao['mc_fechamento_predito'])
print('MSE: '+str(mse))
mae = mean_absolute_error(mc_predicao['mc_fechamento'], mc_predicao['mc_fechamento_predito'])
print('MAE: '+str(mae))
rmse = math.sqrt(mean_squared_error(mc_predicao['mc_fechamento'], mc_predicao['mc_fechamento_predito']))
print('RMSE: '+str(rmse))
```

```
MSE: 0.00015769156814244282
MAE: 0.006908134365400925
RMSE: 0.012557530336114774
```

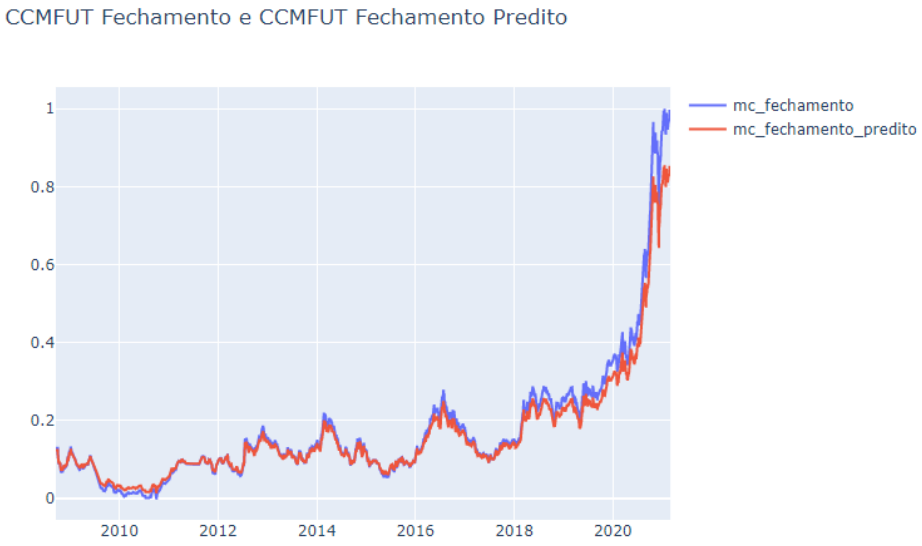
Repetindo os passos acima, analisaremos os resultados usando apenas os dados de fechamento para treinamento e teste:

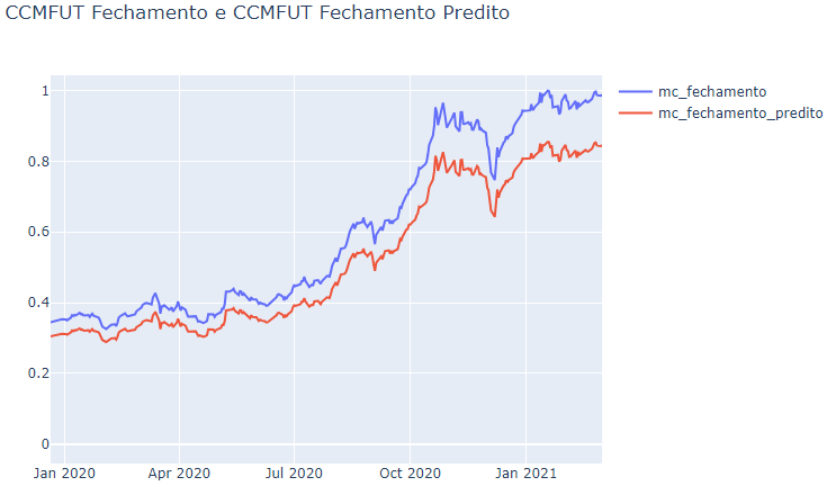
```
In [330]: predicted_mc = []
          for i in predicted_prices_mc:
            predicted_mc.append(i[0])
```

```
In [332]: mc_predicao = mc_predicao.drop(columns='mc_fechamento_predito')
          mc_predicao
```

```
In [333]: mc_predicao['mc_fechamento_predito'] = predicted_mc
```

```
In [335]: interactive_plot(mc_predicao, "CCMFUT Fechamento e CCMFUT Fechamento Predito")
```





```
In [336]: #Cálculo do erro
mse = mean_squared_error(mc_predicao['mc_fechamento'], mc_predicao['mc_fechamento_predito'])
print('MSE: '+str(mse))
mae = mean_absolute_error(mc_predicao['mc_fechamento'], mc_predicao['mc_fechamento_predito'])
print('MAE: '+str(mae))
rmse = math.sqrt(mean_squared_error(mc_predicao['mc_fechamento'], mc_predicao['mc_fechamento_predito']))
print('RMSE: '+str(rmse))

MSE: 0.0009740986387099513
MAE: 0.017127976172408
RMSE: 0.03121055332271364
```

Podemos perceber que ao utilizarmos todos os dados, o modelo possui um erro menor no modelo Ridge Regression do que quando usamos apenas os dados de fechamento.

6.2 – Modelo Preditivo – LSTM

Para apresentar os resultados do modelo, procedemos da seguinte forma:

Inicialmente, criamos uma lista e incluímos os dados preditos (2915 dados):

```
In [359]: test_predicted_LSTM = []

          for i in predicted_LSTM:
            test_predicted_LSTM.append(i[0])

In [360]: len(test_predicted_LSTM)

Out[360]: 2915
```

Em seguida, criamos um dataframe com a data, os dados de preço de fechamento e os valores preditos para o fechamento.

```
In [361]: df_predicao_LSTM = pd.DataFrame(columns = ['data' , 'Fechamento', 'Fechamento_Predito'])

In [362]: df_predicao_LSTM

Out[362]:
```

data	Fechamento	Fechamento_Predito

Preenchemos a coluna data com as datas do dataframe mc_df, excluindo a primeira linha (pois não há alvo para esta data):

```
In [363]: df_predicao_LSTM['data'] = mc_df[1:]['data']
```

```
In [364]: df_predicao_LSTM
```

Out[364]:

	data	Fechamento	Fechamento_Predito
data			
2008-09-22	2008-09-22	NaN	NaN
2008-09-26	2008-09-26	NaN	NaN
2008-09-30	2008-09-30	NaN	NaN
2008-10-02	2008-10-02	NaN	NaN
2008-10-03	2008-10-03	NaN	NaN
...	...	...	...
2021-02-24	2021-02-24	NaN	NaN
2021-02-25	2021-02-25	NaN	NaN
2021-02-26	2021-02-26	NaN	NaN
2021-03-01	2021-03-01	NaN	NaN
2021-03-02	2021-03-02	NaN	NaN

2915 rows × 3 columns

Depois criamos uma outra lista, com os dados de fechamento normalizados:

```
In [365]: Fechamento_Scaled = []
          for i in training_set_scaled:
            Fechamento_Scaled.append(i[0])
```

```
In [366]: len(Fechamento_Scaled)
```

```
Out[366]: 2916
```

Há 2916 dados de preços de fechamento. Da mesma forma que fizemos com as datas, preencheremos o dataframe com os dados de fechamento com a exceção do 1º:

```
In [368]: df_predicao_LSTM['Fechamento'] = Fechamento_Scaled[1:]
```

Finalmente preencheremos o dataframe com os dados de fechamentos preditos:

```
In [370]: df_predicao_LSTM['Fechamento_Predito'] = test_predicted_LSTM
```

```
In [371]: df_predicao_LSTM
```

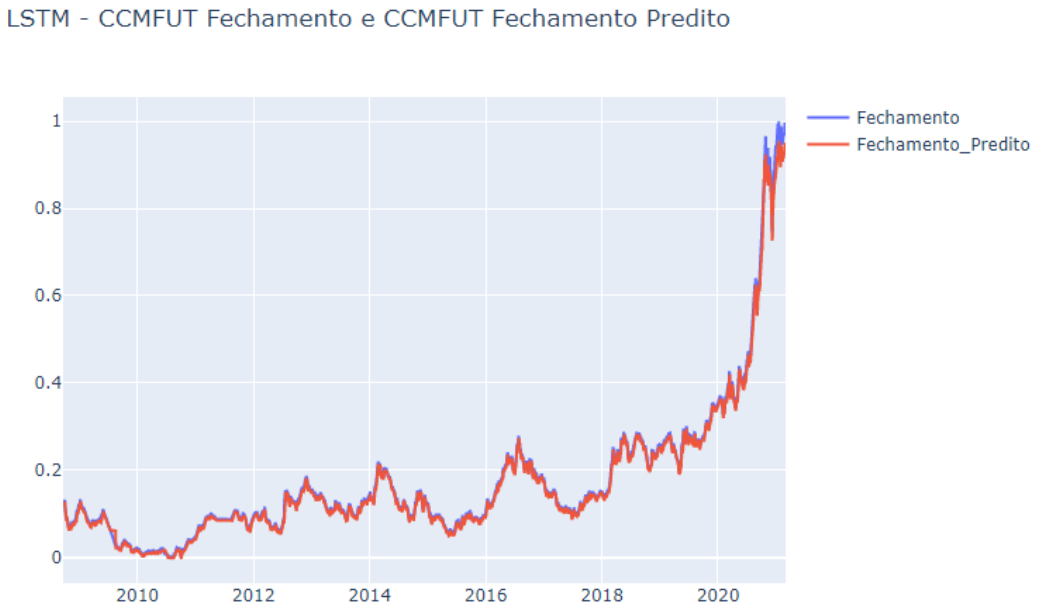
Out[371]:

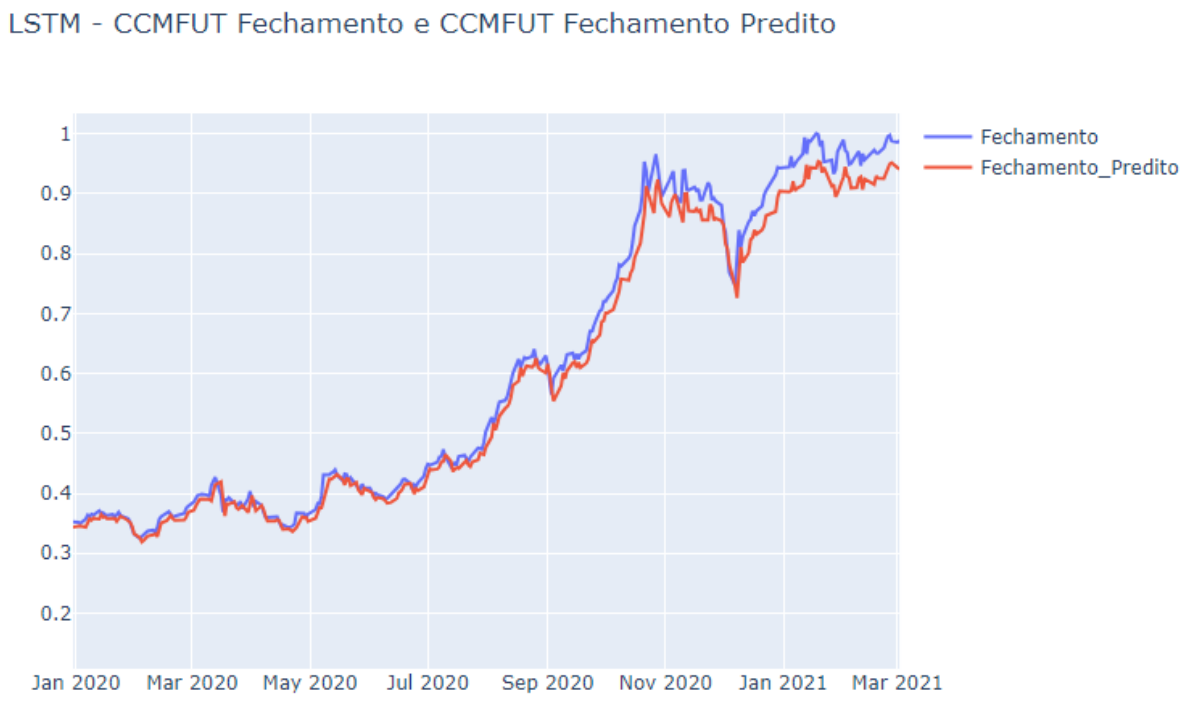
	data	Fechamento	Fechamento_Predito
data			
2008-09-22	2008-09-22	0.128777	0.124064
2008-09-26	2008-09-26	0.129296	0.124064
2008-09-30	2008-09-30	0.127610	0.124580
2008-10-02	2008-10-02	0.128777	0.122903
2008-10-03	2008-10-03	0.115938	0.124064
...	...	...	...
2021-02-24	2021-02-24	0.995461	0.944320
2021-02-25	2021-02-25	0.997536	0.949502
2021-02-26	2021-02-26	0.987550	0.951302
2021-03-01	2021-03-01	0.985475	0.942629
2021-03-02	2021-03-02	0.988588	0.940823

2915 rows × 3 columns

Plotamos o gráfico:

```
In [372]: interactive_plot(df_predicao_LSTM, "LSTM - CCMFUT Fechamento e CCMFUT Fechamento Predito")
```





Percebe-se que os dados preditos são mais próximos dos valores de fechamento, em comparação com o modelo Ridge Regression.

Finalmente, encontrarmos o erro:

```
In [373]: #Cálculo do erro
mse = mean_squared_error(df_predicao_LSTM['Fechamento'], df_predicao_LSTM['Fechamento_Predito'])
print('MSE: '+str(mse))
mae = mean_absolute_error(df_predicao_LSTM['Fechamento'], df_predicao_LSTM['Fechamento_Predito'])
print('MAE: '+str(mae))
rmse = math.sqrt(mean_squared_error(df_predicao_LSTM['Fechamento'], df_predicao_LSTM['Fechamento_Predit
print('RMSE: '+str(rmse))

MSE: 0.00011615309602060521
MAE: 0.0069011389543802695
RMSE: 0.010777434575102055
```

O MSE é inferior ao modelo Ridge Regression tanto em relação ao modelo que recebeu apenas os dados de fechamento quanto ao que utilizamos todos os dados disponíveis.

7. Links

Link para o vídeo: https://www.youtube.com/watch?v=BzGJf712jyI

Link para o repositório: https://github.com/jonslincher/TCC-PucMinas-2021

REFERÊNCIAS

PALAVRO, Cristiano. **Agricultura e a história brasileira**. 25/08/2014 http://revistasafra.com.br/agricultura-e-a-historia-brasileira/ (acessado em 05/03/2021)

F. ALVES PENA, Rodolfo. **Agropecuária no Brasil: principais produtos**. https://mundoeducacao.uol.com.br/geografia/agropecuaria-no-brasil-principais-produtos.htm (acessado em 06/03/2021)

ModalMais. **Milho: entenda como operar milho na Bolsa de Valores.** https://www.modalmais.com.br/mercado-futuro/milho-como-investir#:~:text=O%20milho%20futuro%20%C3%A9%20um,preestabelecido%20no%20momento%20da%20negocia%C3%A7%C3%A3o. (acessado em 07/03/2021)

Qshick. **Ridge Regression for Better Usage**. https://towardsdatascience.com/ridge-regression-for-better-usage-2f19b3a202db (acessado em 09/03/2021)

Udacity, 2021, Curso: **Introduction to Machine Learning**. Disponível em https://www.udacity.com/course/intro-to-machine-learning--ud120 (acessado em 02/2021)

Coursera, 2021, Curso: **Sequences, Time Series and Prediction**. Disponível em https://www.coursera.org/learn/tensorflow-sequences-time-series-and-prediction (acessado em 02/2021)

Udemy , 2021, Curso: **Python & Machine Learning for Financial Analysis**, Disponível em: https://www.udemy.com/course/ml-and-python-in-finance-real-cases-and-practical-solutions/ (acessado entre 02/2021 e 04/2021)

Pandas – Python Data Analysis Library. Disponível em: https://pandas.pydata.org/ (acessado em fevereiro e março de 2021)

Scikit-Learn – Machine Learning in Python. Disponível em: https://scikit-learn.org/stable/index.html (acessado em fevereiro e março de 2021)

Keras: the Python deep learning API. Disponível em: https://keras.io/ (acessado em março e abril de 2021)

https://stackoverflow.com/questions/18689823/pandas-dataframe-replace-nan-values-with-average-of-columns (acessado em 10/03/2021)

https://stackoverflow.com/questions/13148429/how-to-change-the-order-of-dataframe-columns (acessado em 15/03/2021)

https://stackoverflow.com/questions/13187778/convert-pandas-dataframe-to-numpy-array (acessado em 17/03/2021)

https://stackoverflow.com/questions/15741759/find-maximum-value-of-a-column-and-return-the-corresponding-row-values-using-pan (acessado em 19/03/2021)

VERSLOOT, Christian. **How to use Dropout with Keras?** https://www.machinecurve.com/index.php/2019/12/18/how-to-use-dropout-with-keras/ (acessado em 03/04/2021)

APÊNDICE

In [218]:
```
# 1 - BIBLIOTECAS e PACOTES

import numpy as np
import seaborn as sns
import pandas as pd
from pandas import DataFrame
from pandas.util.testing import assert_frame_equal
from pandas_datareader import data
import matplotlib.pyplot as plt
import datetime
import math
```

In [219]:
```
from scipy import stats
```

In [220]:
```
import plotly.express as px
import plotly.figure_factory as ff
from copy import copy
```

In [221]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

In [222]:
```
import sklearn.metrics
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

In [223]:
```
from tensorflow import keras
```

In [224]:
```
# 2 - PROCESSAMENTO/TRATAMENTO DOS DADOS
```

In [225]:
```
milho_df = pd.read_excel('Milho-CEPEA-ESALQ.xlsx')
milho_df
```

Out[225]:

	INDICADOR DO MILHO ESALQ/BM&FBOVESPA	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unna
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	Fonte: Cepea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2	Data	À vista R$	À vista US$	NaN	NaN	NaN	NaN	NaN	NaN	
3	02/08/2004	18.24	5.98	NaN	NaN	NaN	NaN	NaN	NaN	
4	03/08/2004	18.04	5.91	NaN	NaN	NaN	NaN	NaN	NaN	
...	...	...	...	...	...	...	...	...	...	
4125	25/02/2021	85.59	15.55	NaN	NaN	NaN	NaN	NaN	NaN	
4126	26/02/2021	85.41	15.3	NaN	NaN	NaN	NaN	NaN	NaN	
4127	01/03/2021	85.59	15.29	NaN	NaN	NaN	NaN	NaN	NaN	
4128	02/03/2021	86.11	15.2	NaN	NaN	NaN	NaN	NaN	NaN	
4129	03/03/2021	87.06	15.14	NaN	NaN	NaN	NaN	NaN	NaN	

4130 rows × 10 columns

In [226]:
```
milho_df = milho_df.drop(milho_df.index[0:3])
milho_df = milho_df.drop(columns=milho_df.columns[3:])
milho_df
```

Out[226]:

	INDICADOR DO MILHO ESALQ/BM&FBOVESPA	Unnamed: 1	Unnamed: 2
3	02/08/2004	18.24	5.98
4	03/08/2004	18.04	5.91
5	04/08/2004	18.02	5.9
6	05/08/2004	18.06	5.89
7	06/08/2004	18.13	5.98
...	...	...	...
4125	25/02/2021	85.59	15.55
4126	26/02/2021	85.41	15.3
4127	01/03/2021	85.59	15.29
4128	02/03/2021	86.11	15.2
4129	03/03/2021	87.06	15.14

4127 rows × 3 columns

In [227]:
```
milho_df.rename(columns= {'INDICADOR DO MILHO ESALQ/BM&FBOVESPA': 'Data'}, inplace=True)
milho_df.rename(columns= {'Unnamed: 1': 'milho_reais'}, inplace=True)
milho_df.rename(columns= {'Unnamed: 2': 'milho_dolares'}, inplace=True)
milho_df
```

Out[227]:

	Data	milho_reais	milho_dolares
3	02/08/2004	18.24	5.98
4	03/08/2004	18.04	5.91
5	04/08/2004	18.02	5.9
6	05/08/2004	18.06	5.89
7	06/08/2004	18.13	5.98
...	...	...	...
4125	25/02/2021	85.59	15.55
4126	26/02/2021	85.41	15.3
4127	01/03/2021	85.59	15.29
4128	02/03/2021	86.11	15.2
4129	03/03/2021	87.06	15.14

4127 rows × 3 columns

In [228]:
```
milho_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4127 entries, 3 to 4129
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Data           4127 non-null   object
 1   milho_reais    4127 non-null   object
 2   milho_dolares  4127 non-null   object
dtypes: object(3)
memory usage: 129.0+ KB
```

In [229]:
```
milho_df['milho_reais'] = milho_df['milho_reais'].astype(float)
milho_df['milho_dolares'] = milho_df['milho_dolares'].astype(float)
milho_df
```

Out[229]:

	Data	milho_reais	milho_dolares
3	02/08/2004	18.24	5.98
4	03/08/2004	18.04	5.91
5	04/08/2004	18.02	5.90
6	05/08/2004	18.06	5.89
7	06/08/2004	18.13	5.98
...	...	...	...
4125	25/02/2021	85.59	15.55
4126	26/02/2021	85.41	15.30
4127	01/03/2021	85.59	15.29
4128	02/03/2021	86.11	15.20
4129	03/03/2021	87.06	15.14

4127 rows × 3 columns

In [230]:
```
milho_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4127 entries, 3 to 4129
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Data           4127 non-null   object
 1   milho_reais    4127 non-null   float64
 2   milho_dolares  4127 non-null   float64
dtypes: float64(2), object(1)
memory usage: 129.0+ KB
```

```
In [231]: milho_df['Data']
```

```
Out[231]: 3       02/08/2004
          4       03/08/2004
          5       04/08/2004
          6       05/08/2004
          7       06/08/2004
                     ...
          4125    25/02/2021
          4126    26/02/2021
          4127    01/03/2021
          4128    02/03/2021
          4129    03/03/2021
          Name: Data, Length: 4127, dtype: object
```

```
In [232]: milho_df['Data'] = pd.to_datetime(milho_df['Data'],dayfirst=True)
          milho_df = milho_df.sort_values(by = ['Data'])
          milho_df['Data']
```

```
Out[232]: 3      2004-08-02
          4      2004-08-03
          5      2004-08-04
          6      2004-08-05
          7      2004-08-06
                    ...
          4125   2021-02-25
          4126   2021-02-26
          4127   2021-03-01
          4128   2021-03-02
          4129   2021-03-03
          Name: Data, Length: 4127, dtype: datetime64[ns]
```

```
In [233]: milho_df.index
```

```
Out[233]: Int64Index([   3,    4,    5,    6,    7,    8,    9,   10,   11,   12,
                       ...
                       4120, 4121, 4122, 4123, 4124, 4125, 4126, 4127, 4128, 4129],
                      dtype='int64', length=4127)
```

```
In [234]: milho_df.index = pd.to_datetime(milho_df.Data)
          milho_df.index.to_period('D')
          milho_df.index
```

```
Out[234]: DatetimeIndex(['2004-08-02', '2004-08-03', '2004-08-04', '2004-08-05',
                         '2004-08-06', '2004-08-09', '2004-08-10', '2004-08-11',
                         '2004-08-12', '2004-08-13',
                         ...
                         '2021-02-18', '2021-02-19', '2021-02-22', '2021-02-23',
                         '2021-02-24', '2021-02-25', '2021-02-26', '2021-03-01',
                         '2021-03-02', '2021-03-03'],
                        dtype='datetime64[ns]', name='Data', length=4127, freq=None)
```

```
In [235]: milho_df.isnull().sum()
```

```
Out[235]: Data            0
          milho_reais     0
          milho_dolares   0
          dtype: int64
```

In [236]:
```
ccmfut_df = pd.read_excel('CCMFUT-ProfitChart.xlsx')
ccmfut_df
```

Out[236]:

	Data	Abertura	Máxima	Mínima	Fechamento	Volume Financeiro
0	2021-03-05	94.10	96.46	94.10	95.85	380377381.5
1	2021-03-04	91.00	94.72	90.40	94.20	325967202.0
2	2021-03-03	89.00	91.10	88.86	91.05	211768164.0
3	2021-03-02	88.80	89.06	88.54	88.94	95795617.5
4	2021-03-01	88.92	89.18	88.00	88.70	133054398.0
...	...	...	...	...	...	...
2914	2008-10-02	22.64	22.64	22.64	22.64	11295.0
2915	2008-09-30	22.55	22.55	22.55	22.55	112500.0
2916	2008-09-26	22.68	22.68	22.68	22.68	565875.0
2917	2008-09-22	22.64	22.64	22.64	22.64	112950.0
2918	2008-09-19	22.64	22.64	22.64	22.64	1129500.0

2919 rows × 6 columns

In [237]:
```
ccmfut_df.rename(columns= {'Abertura': 'ccmfut_abertura'}, inplace=True)
ccmfut_df.rename(columns= {'Máxima': 'ccmfut_máxima'}, inplace=True)
ccmfut_df.rename(columns= {'Mínima': 'ccmfut_mínima'}, inplace=True)
ccmfut_df.rename(columns= {'Fechamento': 'ccmfut_fechamento'}, inplace=True)
ccmfut_df.rename(columns= {'Volume Financeiro': 'ccmfut_volume_fin'}, inplace=True)
ccmfut_df
```

Out[237]:

	Data	ccmfut_abertura	ccmfut_máxima	ccmfut_mínima	ccmfut_fechamento	ccmfut_volume_fin
0	2021-03-05	94.10	96.46	94.10	95.85	380377381.5
1	2021-03-04	91.00	94.72	90.40	94.20	325967202.0
2	2021-03-03	89.00	91.10	88.86	91.05	211768164.0
3	2021-03-02	88.80	89.06	88.54	88.94	95795617.5
4	2021-03-01	88.92	89.18	88.00	88.70	133054398.0
...	...	...	...	...	...	...
2914	2008-10-02	22.64	22.64	22.64	22.64	11295.0
2915	2008-09-30	22.55	22.55	22.55	22.55	112500.0
2916	2008-09-26	22.68	22.68	22.68	22.68	565875.0
2917	2008-09-22	22.64	22.64	22.64	22.64	112950.0
2918	2008-09-19	22.64	22.64	22.64	22.64	1129500.0

2919 rows × 6 columns

In [238]:
```
ccmfut_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2919 entries, 0 to 2918
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype         
---  ------             --------------  -----         
 0   Data               2919 non-null   datetime64[ns]
 1   ccmfut_abertura    2919 non-null   float64       
 2   ccmfut_máxima      2919 non-null   float64       
 3   ccmfut_mínima      2919 non-null   float64       
 4   ccmfut_fechamento  2919 non-null   float64       
 5   ccmfut_volume_fin  2919 non-null   float64       
dtypes: datetime64[ns](1), float64(5)
memory usage: 137.0 KB
```

In [239]:
```
ccmfut_df = ccmfut_df.drop(ccmfut_df.index[0:2])
ccmfut_df
```

Out[239]:

	Data	ccmfut_abertura	ccmfut_máxima	ccmfut_mínima	ccmfut_fechamento	ccmfut_volume_fin
2	2021-03-03	89.00	91.10	88.86	91.05	211768164.0
3	2021-03-02	88.80	89.06	88.54	88.94	95795617.5
4	2021-03-01	88.92	89.18	88.00	88.70	133054398.0
5	2021-02-26	89.61	89.64	88.86	88.86	69200707.5
6	2021-02-25	89.45	89.72	88.81	89.63	106361550.0
...	...	...	...	...	...	...
2914	2008-10-02	22.64	22.64	22.64	22.64	11295.0
2915	2008-09-30	22.55	22.55	22.55	22.55	112500.0
2916	2008-09-26	22.68	22.68	22.68	22.68	565875.0
2917	2008-09-22	22.64	22.64	22.64	22.64	112950.0
2918	2008-09-19	22.64	22.64	22.64	22.64	1129500.0

2917 rows × 6 columns

In [240]:
```
ccmfut_df = ccmfut_df.sort_values(by = ['Data'])
ccmfut_df
```

Out[240]:

	Data	ccmfut_abertura	ccmfut_máxima	ccmfut_mínima	ccmfut_fechamento	ccmfut_volume_fin
2918	2008-09-19	22.64	22.64	22.64	22.64	1129500.0
2917	2008-09-22	22.64	22.64	22.64	22.64	112950.0
2916	2008-09-26	22.68	22.68	22.68	22.68	565875.0
2915	2008-09-30	22.55	22.55	22.55	22.55	112500.0
2914	2008-10-02	22.64	22.64	22.64	22.64	11295.0
...	...	...	...	...	...	...
6	2021-02-25	89.45	89.72	88.81	89.63	106361550.0
5	2021-02-26	89.61	89.64	88.86	88.86	69200707.5
4	2021-03-01	88.92	89.18	88.00	88.70	133054398.0
3	2021-03-02	88.80	89.06	88.54	88.94	95795617.5
2	2021-03-03	89.00	91.10	88.86	91.05	211768164.0

2917 rows × 6 columns

In [241]:
```
ccmfut_df.index
```

Out[241]:
```
Int64Index([2918, 2917, 2916, 2915, 2914, 2913, 2912, 2911, 2910, 2909,
            ...
              11,   10,    9,    8,    7,    6,    5,    4,    3,    2],
           dtype='int64', length=2917)
```

In [242]:
```
ccmfut_df.index = pd.to_datetime(ccmfut_df.Data)
ccmfut_df.index.to_period('D')
ccmfut_df.index
```

Out[242]:
```
DatetimeIndex(['2008-09-19', '2008-09-22', '2008-09-26', '2008-09-30',
               '2008-10-02', '2008-10-03', '2008-10-06', '2008-10-07',
               '2008-10-08', '2008-10-09',
               ...
               '2021-02-18', '2021-02-19', '2021-02-22', '2021-02-23',
               '2021-02-24', '2021-02-25', '2021-02-26', '2021-03-01',
               '2021-03-02', '2021-03-03'],
              dtype='datetime64[ns]', name='Data', length=2917, freq=None)
```

In [243]:
```
ccmfut_df.isnull().sum()
```

Out[243]:
```
Data                0
ccmfut_abertura     0
ccmfut_máxima       0
ccmfut_mínima       0
ccmfut_fechamento   0
ccmfut_volume_fin   0
dtype: int64
```

In [244]:
```
milho_df.rename(columns= {'Data': 'data'}, inplace=True)
ccmfut_df.rename(columns= {'Data': 'data'}, inplace=True)
```

In [245]:
```
milho_df
```

Out[245]:

Data	data	milho_reais	milho_dolares
2004-08-02	2004-08-02	18.24	5.98
2004-08-03	2004-08-03	18.04	5.91
2004-08-04	2004-08-04	18.02	5.90
2004-08-05	2004-08-05	18.06	5.89
2004-08-06	2004-08-06	18.13	5.98
...	...	...	...
2021-02-25	2021-02-25	85.59	15.55
2021-02-26	2021-02-26	85.41	15.30
2021-03-01	2021-03-01	85.59	15.29
2021-03-02	2021-03-02	86.11	15.20
2021-03-03	2021-03-03	87.06	15.14

4127 rows × 3 columns

In [246]:
```
ccmfut_df
```

Out[246]:

Data	data	ccmfut_abertura	ccmfut_máxima	ccmfut_mínima	ccmfut_fechamento	ccmfut_volume_fin
2008-09-19	2008-09-19	22.64	22.64	22.64	22.64	1129500.0
2008-09-22	2008-09-22	22.64	22.64	22.64	22.64	112950.0
2008-09-26	2008-09-26	22.68	22.68	22.68	22.68	565875.0
2008-09-30	2008-09-30	22.55	22.55	22.55	22.55	112500.0
2008-10-02	2008-10-02	22.64	22.64	22.64	22.64	11295.0
...	...	...	...	...	...	...
2021-02-25	2021-02-25	89.45	89.72	88.81	89.63	106361550.0
2021-02-26	2021-02-26	89.61	89.64	88.86	88.86	69200707.5
2021-03-01	2021-03-01	88.92	89.18	88.00	88.70	133054398.0
2021-03-02	2021-03-02	88.80	89.06	88.54	88.94	95795617.5
2021-03-03	2021-03-03	89.00	91.10	88.86	91.05	211768164.0

2917 rows × 6 columns

In [247]:
```
mc_df = ccmfut_df.merge(
    milho_df.set_index('data'), how='left', on='data'
)
```

In [248]:
```
mc_df
```

Out[248]:

	data	ccmfut_abertura	ccmfut_máxima	ccmfut_mínima	ccmfut_fechamento	ccmfut_volume_fin	milho_reais	milho_d
0	2008-09-19	22.64	22.64	22.64	22.64	1129500.0	23.47	
1	2008-09-22	22.64	22.64	22.64	22.64	112950.0	23.31	
2	2008-09-26	22.68	22.68	22.68	22.68	565875.0	23.24	
3	2008-09-30	22.55	22.55	22.55	22.55	112500.0	22.99	
4	2008-10-02	22.64	22.64	22.64	22.64	11295.0	22.95	
...	...	...	...	...	...	...	...	
2912	2021-02-25	89.45	89.72	88.81	89.63	106361550.0	85.59	

In [249]:
```
mc_df.isnull().sum()
```

Out[249]:
```
data                0
ccmfut_abertura     0
ccmfut_máxima       0
ccmfut_mínima       0
ccmfut_fechamento   0
ccmfut_volume_fin   0
milho_reais         2
milho_dolares       2
dtype: int64
```

In [250]:
```
mc_df_mvmilho_reais = mc_df["milho_reais"].rolling(5).mean().shift(-5).round(0)
mc_df_mvmilho_dolares = mc_df["milho_dolares"].rolling(5).mean().shift(-5).round(0)
mc_df["milho_reais"].fillna(mc_df_mvmilho_reais, inplace=True)
mc_df["milho_dolares"].fillna(mc_df_mvmilho_dolares, inplace=True)
```

In [251]:
```
mc_df.isnull().sum()
```

Out[251]:
```
data                0
ccmfut_abertura     0
ccmfut_máxima       0
ccmfut_mínima       0
ccmfut_fechamento   0
ccmfut_volume_fin   0
milho_reais         1
milho_dolares       1
dtype: int64
```

In [252]:
```
mc_df_mvmilho_reais = mc_df["milho_reais"].rolling(5).mean().shift(-5).round(0)
mc_df_mvmilho_dolares = mc_df["milho_dolares"].rolling(5).mean().shift(-5).round(0)
mc_df["milho_reais"].fillna(mc_df_mvmilho_reais, inplace=True)
mc_df["milho_dolares"].fillna(mc_df_mvmilho_dolares, inplace=True)
```

In [253]:
```
mc_df.isnull().sum()
```

Out[253]:
```
data                0
ccmfut_abertura     0
ccmfut_máxima       0
ccmfut_mínima       0
ccmfut_fechamento   0
ccmfut_volume_fin   0
milho_reais         0
milho_dolares       0
dtype: int64
```

In [254]:
```
mc_df = mc_df.sort_values(by = ['data'])
mc_df
```

Out[254]:

	data	ccmfut_abertura	ccmfut_máxima	ccmfut_mínima	ccmfut_fechamento	ccmfut_volume_fin	milho_reais	milho_dola
0	2008-09-19	22.64	22.64	22.64	22.64	1129500.0	23.47	12
1	2008-09-22	22.64	22.64	22.64	22.64	112950.0	23.31	13
2	2008-09-26	22.68	22.68	22.68	22.68	565875.0	23.24	12
3	2008-09-30	22.55	22.55	22.55	22.55	112500.0	22.99	12
4	2008-10-02	22.64	22.64	22.64	22.64	11295.0	22.95	11
...	...	...	...	...	...	...	...	
2912	2021-02-25	89.45	89.72	88.81	89.63	106361550.0	85.59	15
2913	2021-02-26	89.61	89.64	88.86	88.86	69200707.5	85.41	15
2914	2021-03-01	88.92	89.18	88.00	88.70	133054398.0	85.59	15
2915	2021-03-02	88.80	89.06	88.54	88.94	95795617.5	86.11	15
2916	2021-03-03	89.00	91.10	88.86	91.05	211768164.0	87.06	15

2917 rows × 8 columns

In [255]:
```
mc_df.index = pd.to_datetime(mc_df.data)
mc_df.index.to_period('D')
mc_df.index
```

Out[255]:
```
DatetimeIndex(['2008-09-19', '2008-09-22', '2008-09-26', '2008-09-30',
               '2008-10-02', '2008-10-03', '2008-10-06', '2008-10-07',
               '2008-10-08', '2008-10-09',
               ...
               '2021-02-18', '2021-02-19', '2021-02-22', '2021-02-23',
               '2021-02-24', '2021-02-25', '2021-02-26', '2021-03-01',
               '2021-03-02', '2021-03-03'],
              dtype='datetime64[ns]', name='data', length=2917, freq=None)
```

In [256]:
```
mc_df
```

Out[256]:

	data	ccmfut_abertura	ccmfut_máxima	ccmfut_mínima	ccmfut_fechamento	ccmfut_volume_fin	milho_reais	milho_c
data								
2008-09-19	2008-09-19	22.64	22.64	22.64	22.64	1129500.0	23.47	
2008-09-22	2008-09-22	22.64	22.64	22.64	22.64	112950.0	23.31	
2008-09-26	2008-09-26	22.68	22.68	22.68	22.68	565875.0	23.24	
2008-09-30	2008-09-30	22.55	22.55	22.55	22.55	112500.0	22.99	
2008-10-02	2008-10-02	22.64	22.64	22.64	22.64	11295.0	22.95	
...	...	...	...	...	...	...	...	
2021-	2021-	89.45	89.72	88.81	89.63	106361550.0	85.59	

In [257]:
```
# 3 - Analise e Exploracao dos Dados
```

In [258]:
```
milho_df.describe()
```

Out[258]:

	milho_reais	milho_dolares
count	4127.000000	4127.000000
mean	30.409537	11.535544
std	12.350048	3.276651
min	13.320000	5.890000
25%	21.325000	9.250000
50%	27.770000	10.720000
75%	35.375000	13.750000
max	87.060000	19.960000

In [259]:
```
milho_df[milho_df['milho_reais']==milho_df['milho_reais'].max()]
```

Out[259]:

Data	data	milho_reais	milho_dolares
2021-03-03	2021-03-03	87.06	15.14

In [260]:
```
milho_df[milho_df['milho_dolares']==milho_df['milho_dolares'].max()]
```

Out[260]:

Data	data	milho_reais	milho_dolares
2011-07-01	2011-07-01	31.08	19.96

In [261]:
```
milho_df[milho_df['milho_reais']==milho_df['milho_reais'].min()]
```

Out[261]:

Data	data	milho_reais	milho_dolares
2006-03-30	2006-03-30	13.32	6.08

In [262]:
```
milho_df[milho_df['milho_dolares']==milho_df['milho_dolares'].min()]
```

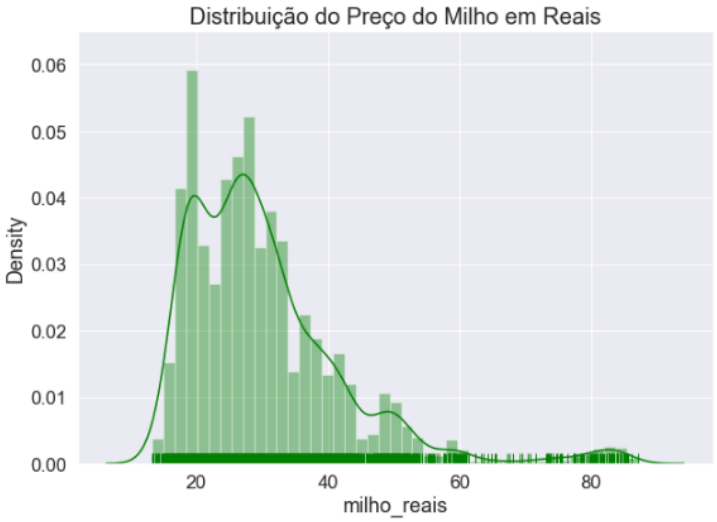
Out[262]:

Data	data	milho_reais	milho_dolares
2004-08-05	2004-08-05	18.06	5.89

In [263]:
```
plt.figure(figsize=(10,7))
sns.set_context('notebook', font_scale=1.5, rc={'font.size':20, 'axes.titlesize':20, 'axes.labelsize':18})
sns.distplot(milho_df['milho_reais'], rug=True, color= 'green')
sns.set_style('darkgrid')
plt.title('Distribuição do Preço do Milho em Reais')
```

```
C:\Users\Jonathan Lincher\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

C:\Users\Jonathan Lincher\anaconda3\lib\site-packages\seaborn\distributions.py:2055: FutureWarning:

The `axis` variable is no longer used and will be removed. Instead, assign variables directly to `x` or `y`.
```

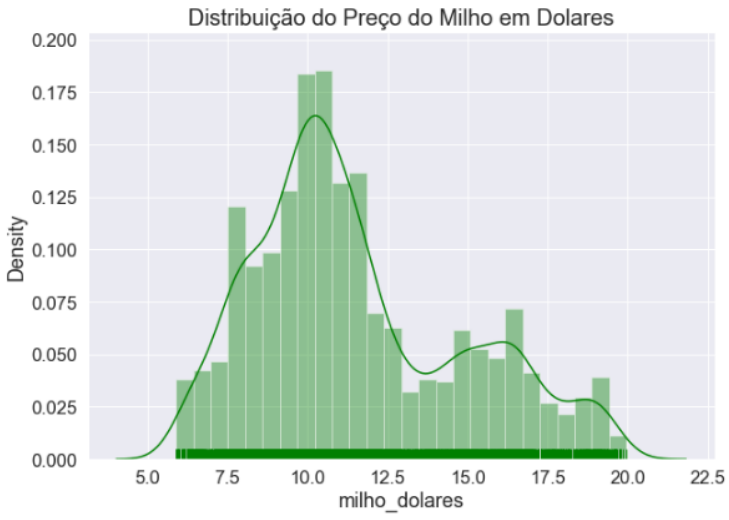
Out[263]: Text(0.5, 1.0, 'Distribuição do Preço do Milho em Reais')



In [264]:
```
plt.figure(figsize=(10,7))
sns.set_context('notebook', font_scale=1.5, rc={'font.size':20, 'axes.titlesize':20, 'axes.labelsize':18})
sns.distplot(milho_df['milho_dolares'], rug=True, color= 'green')
sns.set_style('darkgrid')
plt.title('Distribuição do Preço do Milho em Dolares')
```

```
C:\Users\Jonathan Lincher\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

C:\Users\Jonathan Lincher\anaconda3\lib\site-packages\seaborn\distributions.py:2055: FutureWarning:

The `axis` variable is no longer used and will be removed. Instead, assign variables directly to `x` or `y`.
```

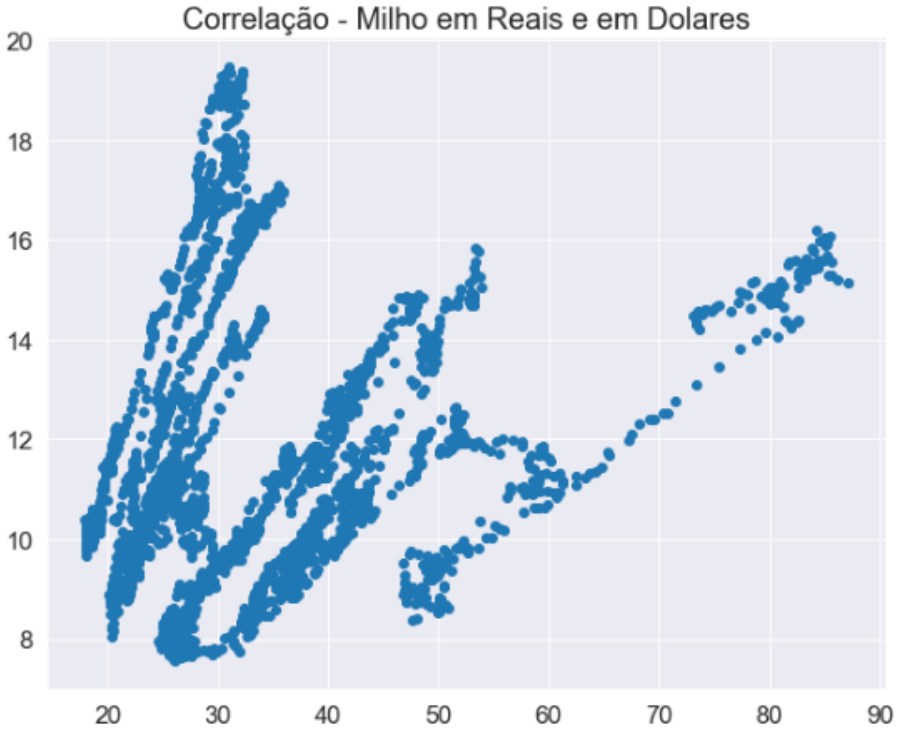
Out[264]: Text(0.5, 1.0, 'Distribuição do Preço do Milho em Dolares')



```
In [265]: mc_df['milho_reais'].corr(mc_df['milho_dolares'])
```

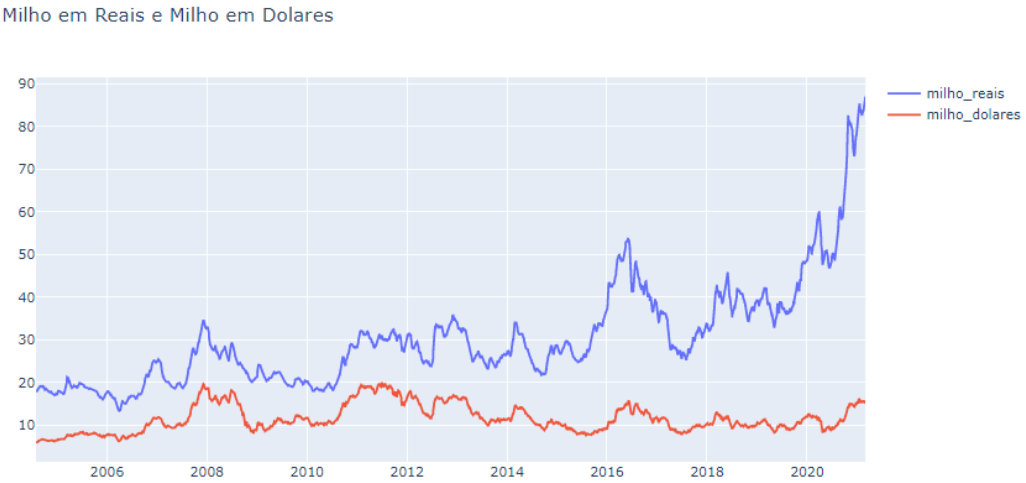
```
Out[265]: 0.1816166279649749
```

```
In [266]: data1=mc_df['milho_reais']
          data2=mc_df['milho_dolares']
          plt.scatter(data1, data2)
          plt.title('Correlação - Milho em Reais e em Dolares')
          plt.gcf().set_size_inches(10, 8)
          plt.show()
```




```
In [267]: def interactive_plot(df, title):
            fig = px.line(title = title)
            for i in df.columns[1:]:
              fig.add_scatter(x = df['data'], y = df[i], name = i)
            fig.show()
```

```
In [268]: interactive_plot(milho_df, "Milho em Reais e Milho em Dolares")
```



In [269]:
```
dolar_df = pd.read_csv('USD_BRL Dados Históricos.csv', decimal=",")
dolar_df
```

Out[269]:

	Data	Último	Abertura	Máxima	Mínima	Var%
0	03.03.2021	5.6193	5.6872	5.7729	5.5806	-1,01%
1	02.03.2021	5.6764	5.6386	5.7327	5.6386	0,61%
2	01.03.2021	5.6418	5.5870	5.6427	5.5553	0,77%
3	26.02.2021	5.5986	5.5340	5.6093	5.4905	1,23%
4	25.02.2021	5.5308	5.4450	5.5390	5.4173	2,30%
...	...	...	...	...	...	...
4319	06.08.2004	3.0330	3.0722	3.0780	3.0300	-1,25%
4320	05.08.2004	3.0713	3.0540	3.0713	3.0500	0,58%
4321	04.08.2004	3.0537	3.0500	3.0660	3.0460	0,12%
4322	03.08.2004	3.0500	3.0450	3.0620	3.0440	0,11%
4323	02.08.2004	3.0465	3.0365	3.0585	3.0365	0,30%

4324 rows × 6 columns

In [270]:
```
dolar_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4324 entries, 0 to 4323
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Data      4324 non-null   object
 1   Último    4324 non-null   float64
 2   Abertura  4324 non-null   float64
 3   Máxima    4324 non-null   float64
 4   Mínima    4324 non-null   float64
 5   Var%      4324 non-null   object
dtypes: float64(4), object(2)
memory usage: 202.8+ KB
```

In [271]:
```
dolar_df = dolar_df.drop(columns=dolar_df.columns[2:])
dolar_df
```

Out[271]:

	Data	Último
0	03.03.2021	5.6193
1	02.03.2021	5.6764
2	01.03.2021	5.6418
3	26.02.2021	5.5986
4	25.02.2021	5.5308
...	...	...
4319	06.08.2004	3.0330
4320	05.08.2004	3.0713
4321	04.08.2004	3.0537
4322	03.08.2004	3.0500
4323	02.08.2004	3.0465

4324 rows × 2 columns

In [272]:
```
dolar_df ['Data'] = pd.to_datetime(dolar_df ['Data'],dayfirst=True)
dolar_df = dolar_df.sort_values(by = ['Data'])
dolar_df['Data']
```

Out[272]:
```
4323   2004-08-02
4322   2004-08-03
4321   2004-08-04
4320   2004-08-05
4319   2004-08-06
          ...
4      2021-02-25
3      2021-02-26
2      2021-03-01
1      2021-03-02
0      2021-03-03
Name: Data, Length: 4324, dtype: datetime64[ns]
```

In [273]:
```
dolar_df.index = pd.to_datetime(dolar_df.Data)
dolar_df.index.to_period('D')
dolar_df.index
```

Out[273]:
```
DatetimeIndex(['2004-08-02', '2004-08-03', '2004-08-04', '2004-08-05',
               '2004-08-06', '2004-08-09', '2004-08-10', '2004-08-11',
               '2004-08-12', '2004-08-13',
               ...
               '2021-02-18', '2021-02-19', '2021-02-22', '2021-02-23',
               '2021-02-24', '2021-02-25', '2021-02-26', '2021-03-01',
               '2021-03-02', '2021-03-03'],
              dtype='datetime64[ns]', name='Data', length=4324, freq=None)
```

In [274]:
```
dolar_df.rename(columns= {'Data': 'data'}, inplace=True)
dolar_df.rename(columns= {'Último': 'Dolar_Último'}, inplace=True)
```

In [275]:
```
dolar_df
```

Out[275]:

Data	data	Dolar_Último
2004-08-02	2004-08-02	3.0465
2004-08-03	2004-08-03	3.0500
2004-08-04	2004-08-04	3.0537
2004-08-05	2004-08-05	3.0713
2004-08-06	2004-08-06	3.0330
...	...	...
2021-02-25	2021-02-25	5.5308
2021-02-26	2021-02-26	5.5986
2021-03-01	2021-03-01	5.6418
2021-03-02	2021-03-02	5.6764
2021-03-03	2021-03-03	5.6193

4324 rows × 2 columns

In [276]:
```
milho_df
```

Out[276]:

Data	data	milho_reais	milho_dolares
2004-08-02	2004-08-02	18.24	5.98
2004-08-03	2004-08-03	18.04	5.91
2004-08-04	2004-08-04	18.02	5.90
2004-08-05	2004-08-05	18.06	5.89
2004-08-06	2004-08-06	18.13	5.98
...	...	...	...
2021-02-25	2021-02-25	85.59	15.55
2021-02-26	2021-02-26	85.41	15.30
2021-03-01	2021-03-01	85.59	15.29
2021-03-02	2021-03-02	86.11	15.20
2021-03-03	2021-03-03	87.06	15.14

4127 rows × 3 columns

```
In [277]: dolar_milho_df = pd.merge(milho_df,dolar_df, how='inner', on=['Data'],suffixes=('_M', '_D'))
          dolar_milho_df
```

Out[277]:

Data	data_M	milho_reais	milho_dolares	data_D	Dolar_Último
2004-08-02	2004-08-02	18.24	5.98	2004-08-02	3.0465
2004-08-03	2004-08-03	18.04	5.91	2004-08-03	3.0500
2004-08-04	2004-08-04	18.02	5.90	2004-08-04	3.0537
2004-08-05	2004-08-05	18.06	5.89	2004-08-05	3.0713
2004-08-06	2004-08-06	18.13	5.98	2004-08-06	3.0330
...	...	...	...	...	...
2021-02-25	2021-02-25	85.59	15.55	2021-02-25	5.5308
2021-02-26	2021-02-26	85.41	15.30	2021-02-26	5.5986
2021-03-01	2021-03-01	85.59	15.29	2021-03-01	5.6418
2021-03-02	2021-03-02	86.11	15.20	2021-03-02	5.6764
2021-03-03	2021-03-03	87.06	15.14	2021-03-03	5.6193

4127 rows × 5 columns

```
In [278]: dolar_milho_df = dolar_milho_df.drop(columns='data_D')
          dolar_milho_df
```

Out[278]:

Data	data_M	milho_reais	milho_dolares	Dolar_Último
2004-08-02	2004-08-02	18.24	5.98	3.0465
2004-08-03	2004-08-03	18.04	5.91	3.0500
2004-08-04	2004-08-04	18.02	5.90	3.0537
2004-08-05	2004-08-05	18.06	5.89	3.0713
2004-08-06	2004-08-06	18.13	5.98	3.0330
...	...	...	...	...
2021-02-25	2021-02-25	85.59	15.55	5.5308
2021-02-26	2021-02-26	85.41	15.30	5.5986
2021-03-01	2021-03-01	85.59	15.29	5.6418
2021-03-02	2021-03-02	86.11	15.20	5.6764
2021-03-03	2021-03-03	87.06	15.14	5.6193

4127 rows × 4 columns

```
In [279]: dolar_milho_df['milho_reais'].corr(dolar_milho_df['Dolar_Último'])
```

Out[279]: 0.7824021362013128

```
In [280]: ccmfut_df.describe()
```

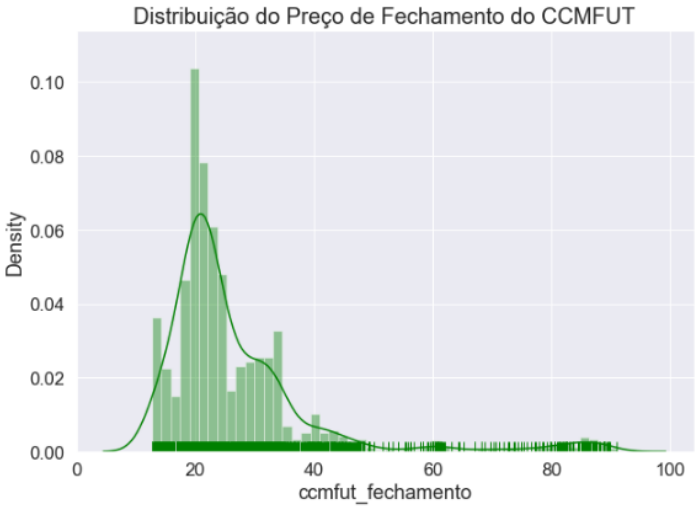
Out[280]:

	ccmfut_abertura	ccmfut_máxima	ccmfut_mínima	ccmfut_fechamento	ccmfut_volume_fin
count	2917.000000	2917.000000	2917.000000	2917.000000	2.917000e+03
mean	26.723809	26.988920	26.474443	26.742749	3.300606e+07
std	13.506372	13.711821	13.312827	13.540517	4.274579e+07
min	12.680000	12.800000	12.310000	12.710000	0.000000e+00
25%	19.620000	19.780000	19.500000	19.640000	1.002070e+07
50%	22.560000	22.760000	22.350000	22.560000	2.244742e+07
75%	30.140000	30.510000	29.820000	30.140000	3.881796e+07
max	89.730000	91.120000	89.280000	91.050000	6.528013e+08

In [281]:
```
plt.figure(figsize=(10,7))
sns.set_context('notebook', font_scale=1.5, rc={'font.size':20, 'axes.titlesize':20, 'axes.labelsize':18})
sns.distplot(ccmfut_df['ccmfut_fechamento'], rug=True, color= 'green')
sns.set_style('darkgrid')
plt.title('Distribuição do Preço de Fechamento do CCMFUT')
```

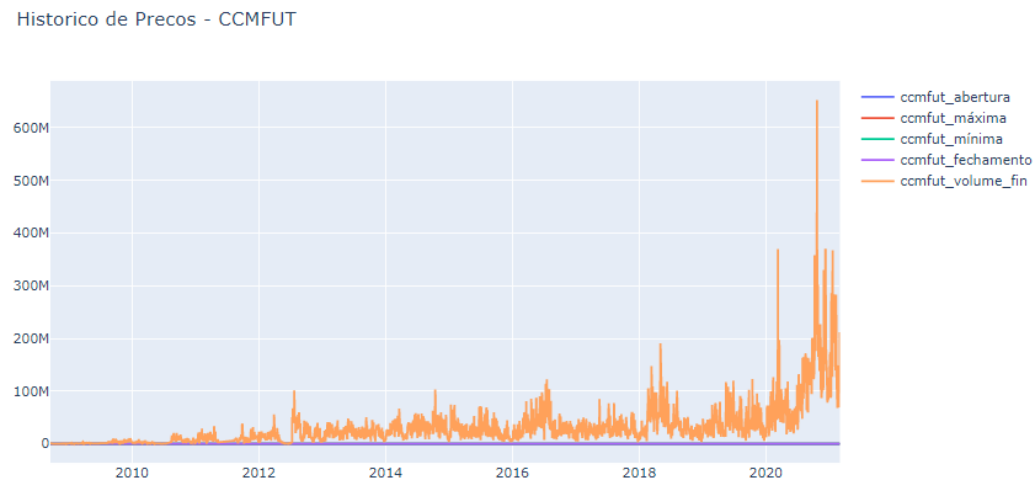
```
C:\Users\Jonathan Lincher\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

C:\Users\Jonathan Lincher\anaconda3\lib\site-packages\seaborn\distributions.py:2055: FutureWarning:

The `axis` variable is no longer used and will be removed. Instead, assign variables directly to `x` or `y`.
```

Out[281]:
```
Text(0.5, 1.0, 'Distribuição do Preço de Fechamento do CCMFUT')
```



In [282]:
```
def interactive_plot(df, title):
  fig = px.line(title = title)
  for i in df.columns[1:]:
    fig.add_scatter(x = df['data'], y = df[i], name = i)
  fig.show()
```

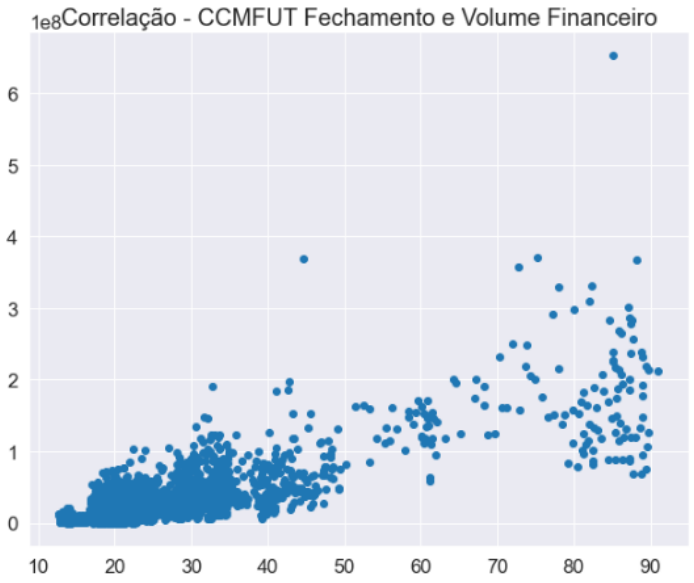
In [283]:
```
interactive_plot(ccmfut_df, "Historico de Precos - CCMFUT")
```



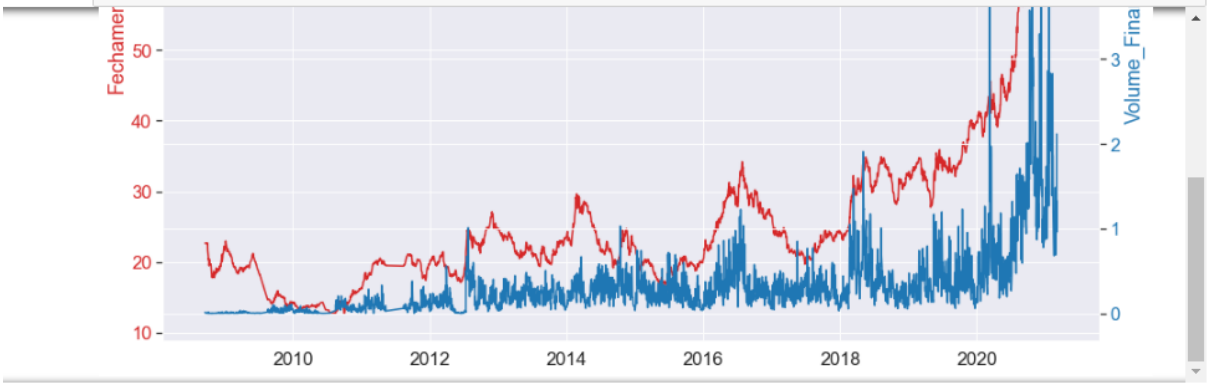
In [284]:
```
ccmfut_df['ccmfut_fechamento'].corr(ccmfut_df['ccmfut_volume_fin'])
```

Out[284]: 0.7999203525936038

In [285]:
```
data1=ccmfut_df['ccmfut_fechamento']
data2=ccmfut_df['ccmfut_volume_fin']
plt.scatter(data1, data2)
plt.title('Correlação - CCMFUT Fechamento e Volume Financeiro')
plt.gcf().set_size_inches(10, 8)
plt.show()
```



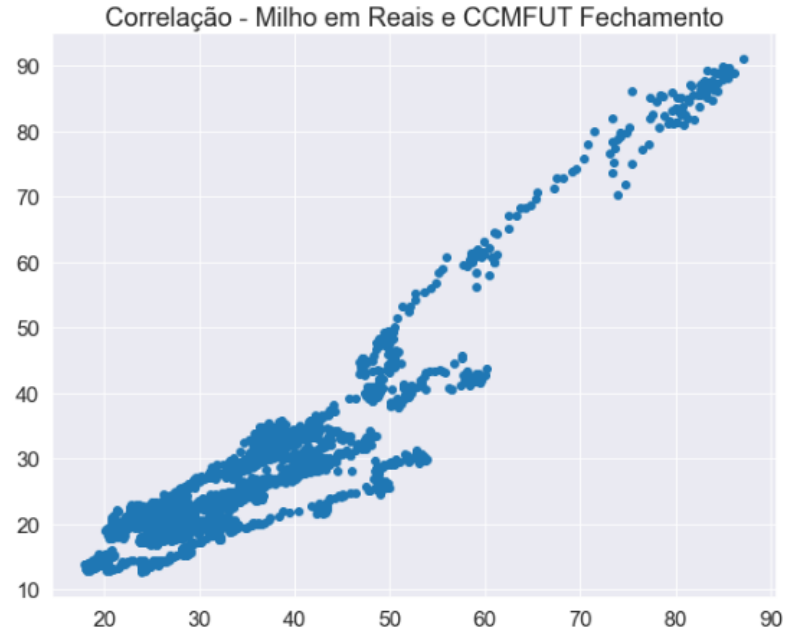
In [286]:
```
b = ccmfut_df["data"]
data1 = ccmfut_df["ccmfut_fechamento"]
data2 = ccmfut_df["ccmfut_volume_fin"]

fig, ax1 = plt.subplots()

color = 'tab:red'
ax1.set_xlabel('')
ax1.set_ylabel('Fechamento', color=color)
ax1.plot(b, data1, color=color)
ax1.tick_params(axis='y', labelcolor=color)

ax2 = ax1.twinx()

color = 'tab:blue'
ax2.set_ylabel('Volume_Financeiro', color=color)
ax2.plot(b, data2, color=color)
ax2.tick_params(axis='y', labelcolor=color)
plt.gcf().set_size_inches(15, 10)
plt.show()
```



In [287]:
```
mc_df['milho_reais'].corr(mc_df['ccmfut_fechamento'])
```

Out[287]: 0.928074900682255

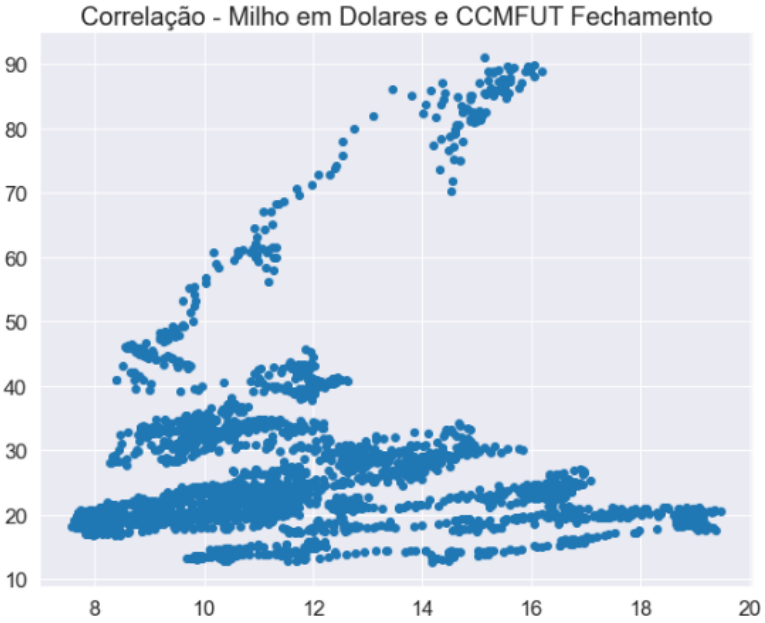
In [288]:
```
data1=mc_df['milho_reais']
data2=mc_df['ccmfut_fechamento']
plt.scatter(data1, data2)
plt.title('Correlação - Milho em Reais e CCMFUT Fechamento')
plt.gcf().set_size_inches(10, 8)
plt.show()
```



In [289]:
```
mc_df['milho_dolares'].corr(mc_df['ccmfut_fechamento'])
```

Out[289]: 0.065512943089332

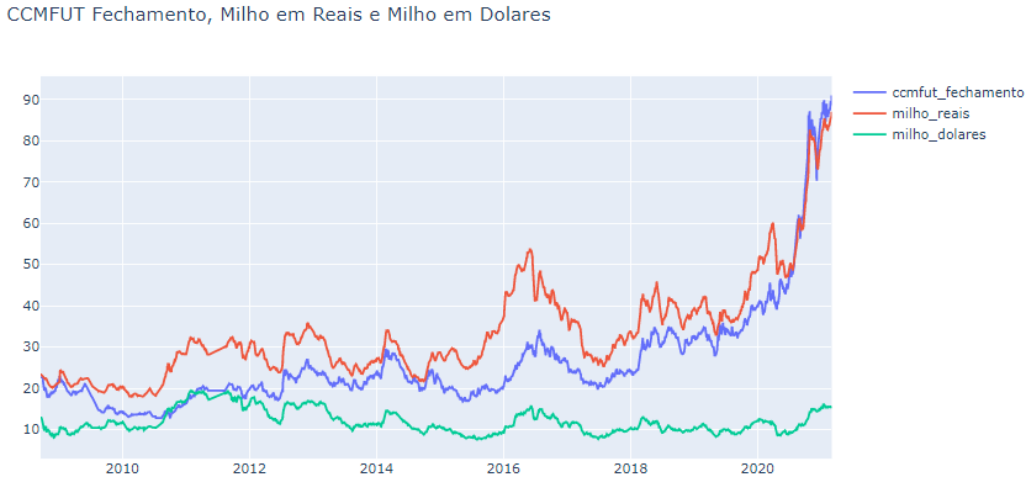
In [290]:
```
data1=mc_df['milho_dolares']
data2=mc_df['ccmfut_fechamento']
plt.scatter(data1, data2)
plt.title('Correlação - Milho em Dolares e CCMFUT Fechamento')
plt.gcf().set_size_inches(10, 8)
plt.show()
```



In [291]:
```
mc_df = mc_df[['data', 'ccmfut_abertura', 'ccmfut_máxima', 'ccmfut_mínima' , 'ccmfut_volume_fin', 'ccmfut_fechamento' , 'milho_re
```

In [292]:
```
def interactive_plot(df, title):
  fig = px.line(title = title)
  for i in df.columns[5:]:
    fig.add_scatter(x = df['data'], y = df[i], name = i)
  fig.show()
```

In [293]:
```
interactive_plot(mc_df, "CCMFUT Fechamento, Milho em Reais e Milho em Dolares")
```



In [294]:
```
# 4 - Ridge Regression
```

In [295]:
```
mc_df['ccmfut_fechamento_alvo'] = mc_df[['ccmfut_fechamento']].shift(-1)
mc_df = mc_df[:-1]
mc_df
```

2008-09-30	2008-09-30	22.55	22.55	22.55	112500.0	22.55	22.99	12.08	22.64
2008-10-02	2008-10-02	22.64	22.64	22.64	11295.0	22.64	22.95	11.36	21.65
...	...	...	...	...	...	...	...	...	...
2021-02-24	2021-02-24	89.08	89.47	88.42	75830647.5	89.47	85.19	15.68	89.63
2021-02-25	2021-02-25	89.45	89.72	88.81	106361550.0	89.63	85.59	15.55	88.86
2021-02-26	2021-02-26	89.61	89.64	88.86	69200707.5	88.86	85.41	15.30	88.70
2021-03-01	2021-03-01	88.92	89.18	88.00	133054398.0	88.70	85.59	15.29	88.94
2021-03-02	2021-03-02	88.80	89.06	88.54	95795617.5	88.94	86.11	15.20	91.05

2916 rows × 9 columns

In [296]:
```
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
mc_df_scaled = sc.fit_transform(mc_df.drop(columns = ['data']))
```

In [297]:
```
mc_df_scaled
```

Out[297]:
```
array([[0.12926671, 0.12563841, 0.13420813, ..., 0.08071617, 0.44107744,
        0.12675517],
       [0.12926671, 0.12563841, 0.13420813, ..., 0.07836807, 0.45622896,
        0.12726576],
       [0.12978585, 0.12614913, 0.13472782, ..., 0.07734077, 0.41750842,
        0.12560633],
       ...,
       [0.99844257, 0.98110317, 0.99454333, ..., 0.98972703, 0.64983165,
        0.97000255],
       [0.98948735, 0.97522983, 0.98337014, ..., 0.99236865, 0.6489899 ,
        0.97306612],
       [0.98792992, 0.97369765, 0.99038586, ..., 1.        , 0.64141414,
        1.        ]])
```

In [298]:
```
mc_df_scaled.shape
```

Out[298]:
```
(2916, 8)
```

In [299]:
```
X = mc_df_scaled[:,:7]
y = mc_df_scaled[:,7:]
```

In [300]:
```
X
```

Out[300]:
```
array([[0.12926671, 0.12563841, 0.13420813, ..., 0.12877707, 0.08071617,
        0.44107744],
       [0.12926671, 0.12563841, 0.13420813, ..., 0.12877707, 0.07836807,
        0.45622896],
       [0.12978585, 0.12614913, 0.13472782, ..., 0.12929581, 0.07734077,
        0.41750842],
       ...,
       [0.99844257, 0.98110317, 0.99454333, ..., 0.98755025, 0.98972703,
        0.64983165],
       [0.98948735, 0.97522983, 0.98337014, ..., 0.9854753 , 0.99236865,
        0.6489899 ],
       [0.98792992, 0.97369765, 0.99038586, ..., 0.98858773, 1.        ,
        0.64141414]])
```

In [301]:
```
X.shape
```

Out[301]:
```
(2916, 7)
```

In [302]:
```
X[0,6]
```

Out[302]:
```
0.44107744107744096
```

In [303]:
```
y
```

Out[303]:
```
array([[0.12675517],
       [0.12726576],
       [0.12560633],
       ...,
       [0.97000255],
       [0.97306612],
       [1.        ]])
```

In [304]:
```
X = np.asarray(X)
y = np.asarray(y)
X.shape, y.shape
```

Out[304]:
```
((2916, 7), (2916, 1))
```

In [305]:
```
X
```

Out[305]:
```
array([[0.12926671, 0.12563841, 0.13420813, ..., 0.12877707, 0.08071617,
        0.44107744],
       [0.12926671, 0.12563841, 0.13420813, ..., 0.12877707, 0.07836807,
        0.45622896],
       [0.12978585, 0.12614913, 0.13472782, ..., 0.12929581, 0.07734077,
        0.41750842],
       ...,
       [0.99844257, 0.98110317, 0.99454333, ..., 0.98755025, 0.98972703,
        0.64983165],
       [0.98948735, 0.97522983, 0.98337014, ..., 0.9854753 , 0.99236865,
        0.6489899 ],
       [0.98792992, 0.97369765, 0.99038586, ..., 0.98858773, 1.        ,
        0.64141414]])
```

In [306]:
```
y
```

Out[306]:
```
array([[0.12675517],
       [0.12726576],
       [0.12560633],
       ...,
       [0.97000255],
       [0.97306612],
       [1.        ]])
```

In [307]:
```
split = int(0.70 * len(X))
X_treino = X[:split]
y_treino = y[:split]
X_teste = X[split:]
y_teste = y[split:]
```

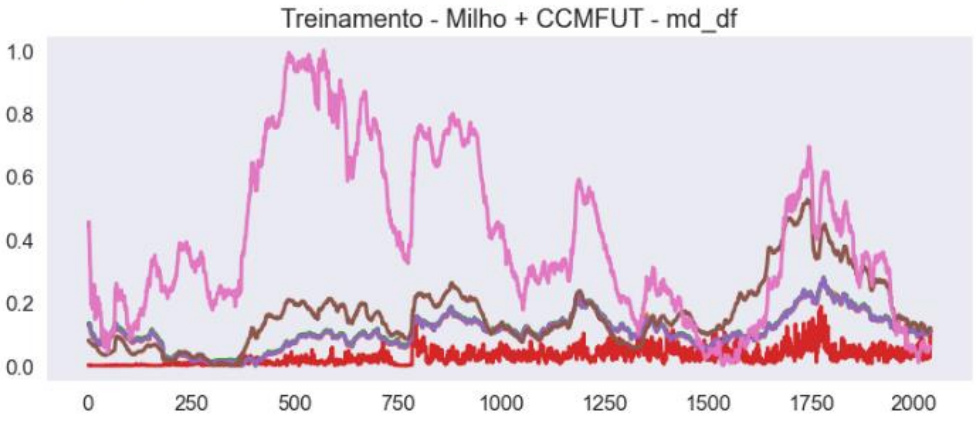
In [308]:
```
X_treino.shape, y_treino.shape
```

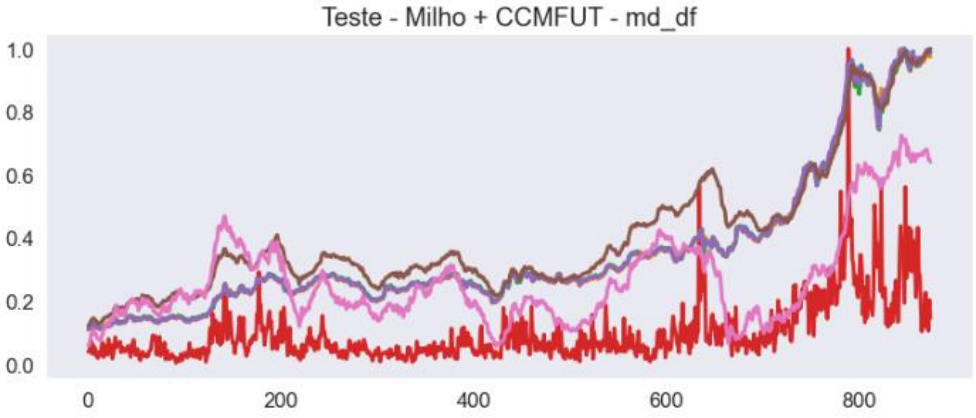
Out[308]:
```
((2041, 7), (2041, 1))
```

In [309]:
```
X_teste.shape, y_teste.shape
```

Out[309]:
```
((875, 7), (875, 1))
```

In [310]:
```
def show_plot_mc(data, title):
  plt.figure(figsize = (13, 5))
  plt.plot(data, linewidth = 3)
  plt.title(title)
  plt.grid()

show_plot_mc(X_treino, 'Treinamento - Milho + CCMFUT - md_df')
show_plot_mc(X_teste, 'Teste - Milho + CCMFUT - md_df')
```





In [311]:
```
from sklearn.linear_model import Ridge
regression_model = Ridge (alpha=1)
regression_model.fit(X_treino, y_treino)
```

Out[311]:
```
Ridge(alpha=1)
```

In [312]:
```
lr_accuracy = regression_model.score(X_teste, y_teste)
print("Linear Regression Score: ", lr_accuracy)
```

```
Linear Regression Score:  0.9874692042284041
```

In [313]:
```
predicted_prices_mc = regression_model.predict(X)
predicted_prices_mc
```

Out[313]:
```
array([[0.12060122],
       [0.12045201],
       [0.12093868],
       ...,
       [0.91668274],
       [0.91338118],
       [0.91366199]])
```

In [314]:
```
predicted_mc = []
for i in predicted_prices_mc:
  predicted_mc.append(i[0])
```

In [315]:
```
mc_fechamento = []
for i in mc_df_scaled:
  mc_fechamento.append(i[4])
```

In [316]:
```
mc_predicao = pd.DataFrame(columns = ['data' , 'mc_fechamento', 'mc_fechamento_predito'])
mc_predicao['data'] = mc_df['data']
mc_predicao['mc_fechamento'] = mc_fechamento
mc_predicao['mc_fechamento_predito'] = predicted_mc
mc_predicao
```

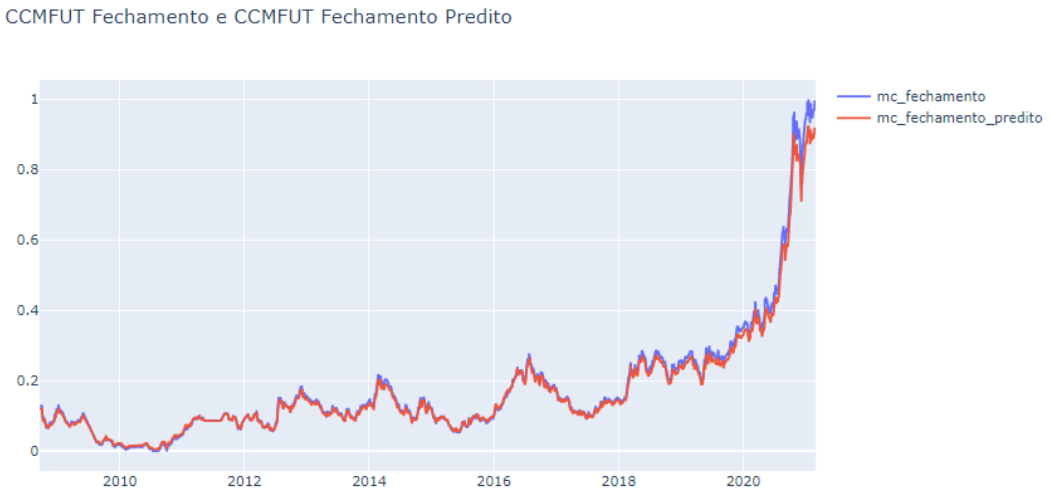
Out[316]:

	data	mc_fechamento	mc_fechamento_predito
data			
2008-09-19	2008-09-19	0.128777	0.120601
2008-09-22	2008-09-22	0.128777	0.120452
2008-09-26	2008-09-26	0.129296	0.120939
2008-09-30	2008-09-30	0.127610	0.119349
2008-10-02	2008-10-02	0.128777	0.120427
...	...	...	...
2021-02-24	2021-02-24	0.995461	0.915434
2021-02-25	2021-02-25	0.997536	0.920280
2021-02-26	2021-02-26	0.987550	0.916683
2021-03-01	2021-03-01	0.985475	0.913381
2021-03-02	2021-03-02	0.988588	0.913662

2916 rows × 3 columns

In [317]:
```
def interactive_plot(data, title):
  fig = px.line(title = title)
  for i in data.columns[1:]:
    fig.add_scatter(x = data['data'], y = data[i], name = i)
  fig.show()
```

In [318]:
```
interactive_plot(mc_predicao, "CCMFUT Fechamento e CCMFUT Fechamento Predito")
```



In [319]:
```
#Cálculo do erro
mse = mean_squared_error(mc_predicao['mc_fechamento'], mc_predicao['mc_fechamento_predito'])
print('MSE: '+str(mse))
mae = mean_absolute_error(mc_predicao['mc_fechamento'], mc_predicao['mc_fechamento_predito'])
print('MAE: '+str(mae))
rmse = math.sqrt(mean_squared_error(mc_predicao['mc_fechamento'], mc_predicao['mc_fechamento_predito']))
print('RMSE: '+str(rmse))
```

```
MSE: 0.0002551720660963173
MAE: 0.008955258899856846
RMSE: 0.015974106112591004
```

In [320]:
```
mc_df
```

Out[320]:

	data	ccmfut_abertura	ccmfut_máxima	ccmfut_mínima	ccmfut_volume_fin	ccmfut_fechamento	milho_reais	milho_dolares	ccmfut_fechamento_alvo
data									
2008-09-19	2008-09-19	22.64	22.64	22.64	1129500.0	22.64	23.47	12.82	22.64
2008-09-22	2008-09-22	22.64	22.64	22.64	112950.0	22.64	23.31	13.00	22.68
2008-09-26	2008-09-26	22.68	22.68	22.68	565875.0	22.68	23.24	12.54	22.55
2008-09-30	2008-09-30	22.55	22.55	22.55	112500.0	22.55	22.99	12.08	22.64
2008-10-02	2008-10-02	22.64	22.64	22.64	11295.0	22.64	22.95	11.36	21.65
...	...	...	...	...	...	...	...	...	...
2021-02-24	2021-02-24	89.08	89.47	88.42	75830647.5	89.47	85.19	15.68	89.63
2021-02-25	2021-02-25	89.45	89.72	88.81	106361550.0	89.63	85.59	15.55	88.86
2021-02-26	2021-02-26	89.61	89.64	88.86	69200707.5	88.86	85.41	15.30	88.70
2021-03-01	2021-03-01	88.92	89.18	88.00	133054398.0	88.70	85.59	15.29	88.94
2021-03-02	2021-03-02	88.80	89.06	88.54	95795617.5	88.94	86.11	15.20	91.05

2916 rows × 9 columns

In [321]:
```
mc_df = mc_df.drop(columns='ccmfut_abertura')
mc_df = mc_df.drop(columns='ccmfut_máxima')
mc_df = mc_df.drop(columns='ccmfut_mínima')
mc_df = mc_df.drop(columns='ccmfut_volume_fin')
mc_df = mc_df.drop(columns='milho_reais')
mc_df = mc_df.drop(columns='milho_dolares')
mc_df
```

Out[321]:

	data	ccmfut_fechamento	ccmfut_fechamento_alvo
data			
2008-09-19	2008-09-19	22.64	22.64
2008-09-22	2008-09-22	22.64	22.68
2008-09-26	2008-09-26	22.68	22.55
2008-09-30	2008-09-30	22.55	22.64
2008-10-02	2008-10-02	22.64	21.65
...	...	...	...
2021-02-24	2021-02-24	89.47	89.63
2021-02-25	2021-02-25	89.63	88.86
2021-02-26	2021-02-26	88.86	88.70
2021-03-01	2021-03-01	88.70	88.94
2021-03-02	2021-03-02	88.94	91.05

2916 rows × 3 columns

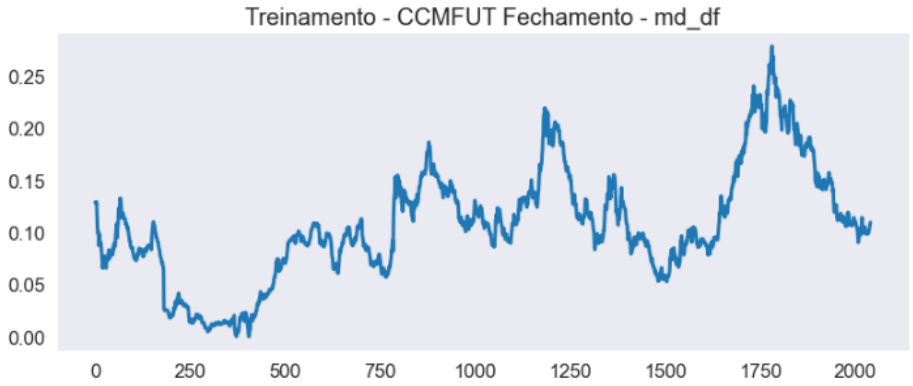
In [322]:
```
mc_df_scaled_fech = sc.fit_transform(mc_df.drop(columns = ['data']))
```

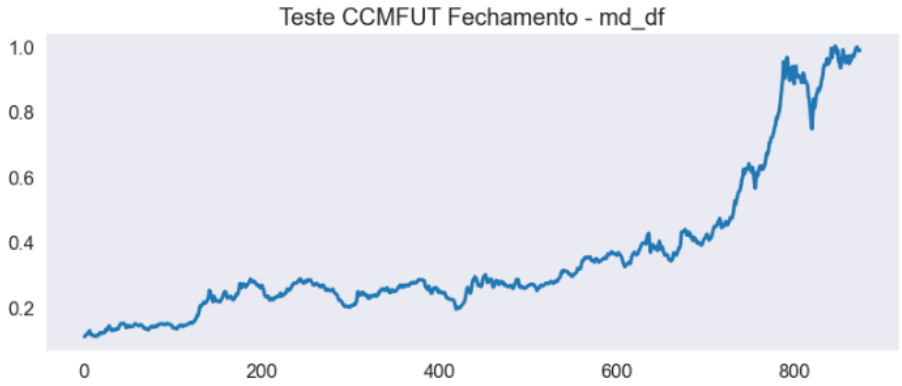
In [323]:
```
X = mc_df_scaled_fech[:,:1]
y = mc_df_scaled_fech[:,1:]
```

In [324]:
```
X = np.asarray(X)
y = np.asarray(y)
```

In [325]:
```
split = int(0.70 * len(X))
X_treino = X[:split]
y_treino = y[:split]
X_teste = X[split:]
y_teste = y[split:]
```

In [326]:
```
def show_plot_mc(data, title):
  plt.figure(figsize = (13, 5))
  plt.plot(data, linewidth = 3)
  plt.title(title)
  plt.grid()

show_plot_mc(X_treino, 'Treinamento - CCMFUT Fechamento - md_df')
show_plot_mc(X_teste, 'Teste CCMFUT Fechamento - md_df')
```





In [327]:
```
regression_model.fit(X_treino, y_treino)
```

Out[327]:
```
Ridge(alpha=1)
```

In [328]:
```
lr_accuracy = regression_model.score(X_teste, y_teste)
print("Linear Regression Score: ", lr_accuracy)
```

```
Linear Regression Score:  0.9484580850390733
```

In [329]:
```
predicted_prices_mc = regression_model.predict(X)
predicted_prices_mc
```

Out[329]:
```
array([[0.12355197],
       [0.12355197],
       [0.12398756],
       ...,
       [0.84466647],
       [0.84292412],
       [0.84553764]])
```

In [330]:
```
predicted_mc = []
for i in predicted_prices_mc:
  predicted_mc.append(i[0])
```

In [331]:
```
mc_predicao
```

Out[331]:

	data	mc_fechamento	mc_fechamento_predito
data			
2008-09-19	2008-09-19	0.128777	0.120601
2008-09-22	2008-09-22	0.128777	0.120452
2008-09-26	2008-09-26	0.129296	0.120939
2008-09-30	2008-09-30	0.127610	0.119349
2008-10-02	2008-10-02	0.128777	0.120427
...	...	...	...
2021-02-24	2021-02-24	0.995461	0.915434
2021-02-25	2021-02-25	0.997536	0.920280
2021-02-26	2021-02-26	0.987550	0.916683
2021-03-01	2021-03-01	0.985475	0.913381
2021-03-02	2021-03-02	0.988588	0.913662

2916 rows × 3 columns

In [332]:
```
mc_predicao = mc_predicao.drop(columns='mc_fechamento_predito')
mc_predicao
```

Out[332]:

	data	mc_fechamento
data		
2008-09-19	2008-09-19	0.128777
2008-09-22	2008-09-22	0.128777
2008-09-26	2008-09-26	0.129296
2008-09-30	2008-09-30	0.127610
2008-10-02	2008-10-02	0.128777
...	...	...
2021-02-24	2021-02-24	0.995461
2021-02-25	2021-02-25	0.997536
2021-02-26	2021-02-26	0.987550
2021-03-01	2021-03-01	0.985475
2021-03-02	2021-03-02	0.988588

2916 rows × 2 columns

In [333]:
```
mc_predicao['mc_fechamento_predito'] = predicted_mc
```

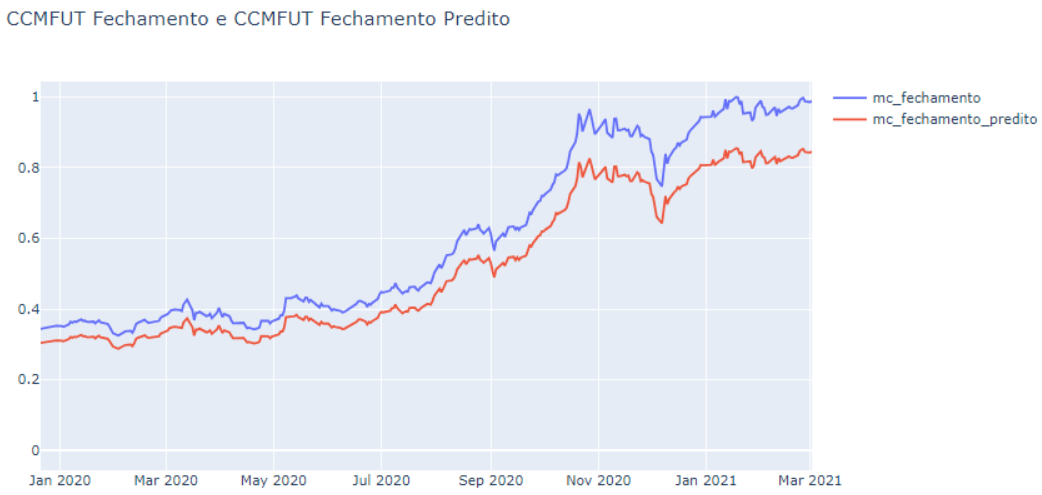
In [334]:
```
mc_predicao
```

Out[334]:

	data	mc_fechamento	mc_fechamento_predito
data			
2008-09-19	2008-09-19	0.128777	0.123552
2008-09-22	2008-09-22	0.128777	0.123552
2008-09-26	2008-09-26	0.129296	0.123988
2008-09-30	2008-09-30	0.127610	0.122572
2008-10-02	2008-10-02	0.128777	0.123552
...	...	...	...
2021-02-24	2021-02-24	0.995461	0.851309
2021-02-25	2021-02-25	0.997536	0.853052
2021-02-26	2021-02-26	0.987550	0.844666
2021-03-01	2021-03-01	0.985475	0.842924
2021-03-02	2021-03-02	0.988588	0.845538

2916 rows × 3 columns

In [335]:
```
interactive_plot(mc_predicao, "CCMFUT Fechamento e CCMFUT Fechamento Predito")
```



In [336]:
```
#Cálculo do erro
mse = mean_squared_error(mc_predicao['mc_fechamento'], mc_predicao['mc_fechamento_predito'])
print('MSE: '+str(mse))
mae = mean_absolute_error(mc_predicao['mc_fechamento'], mc_predicao['mc_fechamento_predito'])
print('MAE: '+str(mae))
rmse = math.sqrt(mean_squared_error(mc_predicao['mc_fechamento'], mc_predicao['mc_fechamento_predito']))
print('RMSE: '+str(rmse))
```

```
MSE: 0.0009740986387099513
MAE: 0.017127976172408
RMSE: 0.03121055332271364
```

In [337]:
```
# 5 - RNN - LSTM
```

In [338]:
```
mc_df
```

Out[338]:

data	data	ccmfut_fechamento	ccmfut_fechamento_alvo
2008-09-19	2008-09-19	22.64	22.64
2008-09-22	2008-09-22	22.64	22.68
2008-09-26	2008-09-26	22.68	22.55
2008-09-30	2008-09-30	22.55	22.64
2008-10-02	2008-10-02	22.64	21.65
...	...	...	...
2021-02-24	2021-02-24	89.47	89.63
2021-02-25	2021-02-25	89.63	88.86
2021-02-26	2021-02-26	88.86	88.70
2021-03-01	2021-03-01	88.70	88.94
2021-03-02	2021-03-02	88.94	91.05

2916 rows × 3 columns

In [339]:
```
mc_df = mc_df.drop(columns='ccmfut_fechamento_alvo')
mc_df
```

Out[339]:

data	data	ccmfut_fechamento
2008-09-19	2008-09-19	22.64
2008-09-22	2008-09-22	22.64
2008-09-26	2008-09-26	22.68
2008-09-30	2008-09-30	22.55
2008-10-02	2008-10-02	22.64
...	...	...
2021-02-24	2021-02-24	89.47
2021-02-25	2021-02-25	89.63
2021-02-26	2021-02-26	88.86
2021-03-01	2021-03-01	88.70
2021-03-02	2021-03-02	88.94

2916 rows × 2 columns

In [340]:
```
training_data = mc_df.iloc[:, 1:].values
training_data
```

Out[340]:
```
array([[22.64],
       [22.64],
       [22.68],
       ...,
       [88.86],
       [88.7 ],
       [88.94]])
```

In [341]:
```
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
training_set_scaled = sc.fit_transform(training_data)
```

In [342]:
```
training_set_scaled
```

Out[342]:
```
array([[0.12877707],
       [0.12877707],
       [0.12929581],
       ...,
       [0.98755025],
       [0.9854753 ],
       [0.98858773]])
```

In [343]:
```
X = []
y = []
for i in range(1, len(mc_df)):
    X.append(training_set_scaled [i-1:i, 0])
    y.append(training_set_scaled [i, 0])
```

In [344]:
```
X
```

```
array([0.08390611]),
array([0.08196084]),
array([0.08325768]),
array([0.08325768]),
array([0.08325768]),
array([0.07832966]),
array([0.07664376]),
array([0.07729218]),
array([0.0771625]),
array([0.08014525]),
array([0.08183115]),
array([0.08390611]),
array([0.08325768]),
array([0.08157178]),
array([0.08234989]),
array([0.0822202]),
array([0.08546233]),
array([0.08442485]),
array([0.08611075]),
```

In [345]:
```
y
```

Out[345]:
```
[0.1287770717157308,
 0.12929581117883546,
 0.12760990792374535,
 0.1287770717157308,
 0.11593827000389054,
 0.10647127480223059,
 0.10076514070807938,
 0.09830112825833223,
 0.10063545584230318,
 0.08740759953313451,
 0.09843081312410842,
 0.09246530929840488,
 0.09013098171443393,
 0.08896381792244848,
 0.09013098171443393,
 0.08727791466735832,
 0.08079367137855015,
 0.07729218000259372,
 0.071975100505771,
```

In [346]:
```
X = np.asarray(X)
y = np.asarray(y)
```

In [347]:
```
X.shape , y.shape
```

Out[347]:
```
((2915, 1), (2915,))
```

In [348]:
```
split = int(0.7 * len(X))
X_train = X[:split]
y_train = y[:split]
X_test = X[split:]
y_test = y[split:]
```

In [349]:
```
X_train.shape , y_train.shape, X_test.shape, y_test.shape
```

Out[349]:
```
((2040, 1), (2040,), (875, 1), (875,))
```

In [350]:
```
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
X_train.shape, X_test.shape
```

Out[350]:
```
((2040, 1, 1), (875, 1, 1))
```

In [351]:
```
X_train
```

Out[351]:
```
array([[[0.12877707]],

       [[0.12877707]],

       [[0.12929581]],

       ...,

       [[0.10361821]],

       [[0.1054338 ]],

       [[0.10815718]]])
```

In [352]:
```
X_train.shape
```

Out[352]:
```
(2040, 1, 1)
```

In [353]:
```
inputs = keras.layers.Input(shape=(X_train.shape[1], X_train.shape[2]))
x = keras.layers.LSTM(150, return_sequences= True)(inputs)
x = keras.layers.Dropout(0.3)(x)
x = keras.layers.LSTM(150, return_sequences=True)(x)
x = keras.layers.Dropout(0.3)(x)
x = keras.layers.LSTM(150)(x)
outputs = keras.layers.Dense(1, activation='linear')(x)

model = keras.Model(inputs=inputs, outputs=outputs)
model.compile(optimizer='adam', loss="mse")
model.summary()
```

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 1, 1)]            0
_________________________________________________________________
lstm_6 (LSTM)                (None, 1, 150)            91200
_________________________________________________________________
dropout_4 (Dropout)          (None, 1, 150)            0
_________________________________________________________________
lstm_7 (LSTM)                (None, 1, 150)            180600
_________________________________________________________________
dropout_5 (Dropout)          (None, 1, 150)            0
_________________________________________________________________
lstm_8 (LSTM)                (None, 150)               180600
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 151
=================================================================
Total params: 452,551
Trainable params: 452,551
Non-trainable params: 0
_________________________________________________________________
```

In [354]:
```
X_train.shape, y_train.shape
```

Out[354]:
```
((2040, 1, 1), (2040,))
```

In [355]:
```
history = model.fit(
    X_train, y_train,
    epochs = 20,
    batch_size = 32,
    validation_split = 0.2
)
```

```
Epoch 2/20
51/51 [==============================] - 1s 11ms/step - loss: 0.0010 - val_loss: 4.1449e-05
Epoch 3/20
51/51 [==============================] - 1s 11ms/step - loss: 6.6296e-05 - val_loss: 2.5949e-05
Epoch 4/20
51/51 [==============================] - 1s 11ms/step - loss: 4.0457e-05 - val_loss: 2.4025e-05
Epoch 5/20
51/51 [==============================] - 0s 10ms/step - loss: 3.9753e-05 - val_loss: 3.7790e-05
Epoch 6/20
51/51 [==============================] - 1s 10ms/step - loss: 4.1150e-05 - val_loss: 2.5174e-05
Epoch 7/20
51/51 [==============================] - 0s 8ms/step - loss: 3.2069e-05 - val_loss: 4.0390e-05
Epoch 8/20
51/51 [==============================] - 0s 9ms/step - loss: 3.2752e-05 - val_loss: 2.4666e-05
Epoch 9/20
51/51 [==============================] - 0s 8ms/step - loss: 3.4260e-05 - val_loss: 2.8544e-05
Epoch 10/20
51/51 [==============================] - 0s 7ms/step - loss: 3.5984e-05 - val_loss: 2.3459e-05
Epoch 11/20
51/51 [==============================] - 0s 7ms/step - loss: 3.1575e-05 - val_loss: 2.9729e-05
```

In [356]:
```
predicted_LSTM = model.predict(X)
```

In [357]:
```
predicted_LSTM
```

Out[357]:
```
array([[0.12406403],
       [0.12406403],
       [0.12458012],
       ...,
       [0.9513022 ],
       [0.9426287 ],
       [0.9408232 ]], dtype=float32)
```

In [358]:
```
predicted.shape
```

Out[358]:
```
(2915, 1)
```

In [359]:
```
test_predicted_LSTM = []

for i in predicted_LSTM:
  test_predicted_LSTM.append(i[0])
```

In [360]:
```
len(test_predicted_LSTM)
```

Out[360]:
```
2915
```

In [361]:
```
df_predicao_LSTM = pd.DataFrame(columns = ['data' , 'Fechamento', 'Fechamento_Predito'])
```

In [362]:
```
df_predicao_LSTM
```

Out[362]:

data	Fechamento	Fechamento_Predito

In [363]:
```
df_predicao_LSTM['data'] = mc_df[1:]['data']
```

In [364]:
```
df_predicao_LSTM
```

Out[364]:

	data	Fechamento	Fechamento_Predito
data			
2008-09-22	2008-09-22	NaN	NaN
2008-09-26	2008-09-26	NaN	NaN
2008-09-30	2008-09-30	NaN	NaN
2008-10-02	2008-10-02	NaN	NaN
2008-10-03	2008-10-03	NaN	NaN
...	...	...	...
2021-02-24	2021-02-24	NaN	NaN
2021-02-25	2021-02-25	NaN	NaN
2021-02-26	2021-02-26	NaN	NaN
2021-03-01	2021-03-01	NaN	NaN
2021-03-02	2021-03-02	NaN	NaN

In [365]:
```
Fechamento_Scaled = []
for i in training_set_scaled:
  Fechamento_Scaled.append(i[0])
```

In [366]:
```
len(Fechamento_Scaled)
```

Out[366]: 2916

In [367]:
```
df_predicao_LSTM
```

Out[367]:

data	data	Fechamento	Fechamento_Predito
2008-09-22	2008-09-22	NaN	NaN
2008-09-26	2008-09-26	NaN	NaN
2008-09-30	2008-09-30	NaN	NaN
2008-10-02	2008-10-02	NaN	NaN
2008-10-03	2008-10-03	NaN	NaN
...	...	...	...
2021-02-24	2021-02-24	NaN	NaN
2021-02-25	2021-02-25	NaN	NaN
2021-02-26	2021-02-26	NaN	NaN
2021-03-01	2021-03-01	NaN	NaN
2021-03-02	2021-03-02	NaN	NaN

2915 rows × 3 columns

In [368]:
```
df_predicao_LSTM['Fechamento'] = Fechamento_Scaled[1:]
```

In [369]:
```
df_predicao_LSTM
```

Out[369]:

data	data	Fechamento	Fechamento_Predito
2008-09-22	2008-09-22	0.128777	NaN
2008-09-26	2008-09-26	0.129296	NaN
2008-09-30	2008-09-30	0.127610	NaN
2008-10-02	2008-10-02	0.128777	NaN
2008-10-03	2008-10-03	0.115938	NaN
...	...	...	...
2021-02-24	2021-02-24	0.995461	NaN
2021-02-25	2021-02-25	0.997536	NaN
2021-02-26	2021-02-26	0.987550	NaN
2021-03-01	2021-03-01	0.985475	NaN
2021-03-02	2021-03-02	0.988588	NaN

2915 rows × 3 columns

In [370]:
```
df_predicao_LSTM['Fechamento_Predito'] = test_predicted_LSTM
```

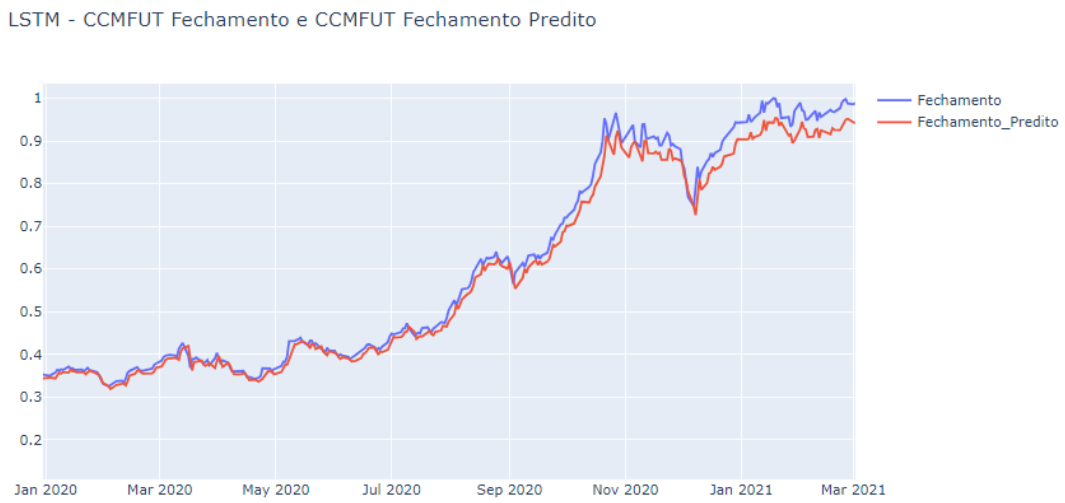
In [371]:
```
df_predicao_LSTM
```

Out[371]:

data	data	Fechamento	Fechamento_Predito
2008-09-22	2008-09-22	0.128777	0.124064
2008-09-26	2008-09-26	0.129296	0.124064
2008-09-30	2008-09-30	0.127610	0.124580
2008-10-02	2008-10-02	0.128777	0.122903
2008-10-03	2008-10-03	0.115938	0.124064
...	...	...	...
2021-02-24	2021-02-24	0.995461	0.944320
2021-02-25	2021-02-25	0.997536	0.949502
2021-02-26	2021-02-26	0.987550	0.951302
2021-03-01	2021-03-01	0.985475	0.942629
2021-03-02	2021-03-02	0.988588	0.940823

2915 rows × 3 columns

In [372]:
```
interactive_plot(df_predicao_LSTM, "LSTM - CCMFUT Fechamento e CCMFUT Fechamento Predito")
```



In [373]:
```
#Cálculo do erro
mse = mean_squared_error(df_predicao_LSTM['Fechamento'], df_predicao_LSTM['Fechamento_Predito'])
print('MSE: '+str(mse))
mae = mean_absolute_error(df_predicao_LSTM['Fechamento'], df_predicao_LSTM['Fechamento_Predito'])
print('MAE: '+str(mae))
rmse = math.sqrt(mean_squared_error(df_predicao_LSTM['Fechamento'], df_predicao_LSTM['Fechamento_Predito']))
print('RMSE: '+str(rmse))
```

```
MSE: 0.00011615309602060521
MAE: 0.0069011389543802695
RMSE: 0.010777434575102055
```