Modelo Futuros Dogecoin - Dados Históricos

O Mercado Futuro é o ambiente onde você pode ganhar com a alta ou baixa de um determinado ativo, seja ele uma commodity (Milho, Café, Boi Gordo), uma moeda (como o dólar), um Índice (Bovespa, Índice S&P 500) ou mesmo uma taxa de juros. Nele, são negociados contratos futuros.

O Dogecoin é um contrato futuro derivado do Índice Dogecoin, ou seja, é um ativo que tem como base o sobe e desce desse índice. Como esse tipo de operação envolve **risco considerável** e **oscilações frequentes no mercado**, ela é indicada apenas para aqueles que se encaixam no perfil de investidor arrojado.

Neste trabalho iremos implementar uma RNNs para realizar a predição diária do Dogecoin da Ibovespa.

O dataset "**FuturosDogecoin.csv**" possui informações dispostas em colunas :

- **Date**: Data das operações na bolsa (diária)
- **Close**: Valor de Fechamento do Índice da Ibovespa (no dia)
- **Open**: Valor da Abertura do Índice da Ibovespa (no dia)
- **High**: Valor máximo do Índice da Ibovespa (no dia)
- **Low**: Valor mínimo do Índice da Ibovespa (no dia)
- **Vol**: Volume de contratos negociados (no dia)

Bibliotecas

```
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense, Dropout, LSTM
import plotly.graph_objects as go
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

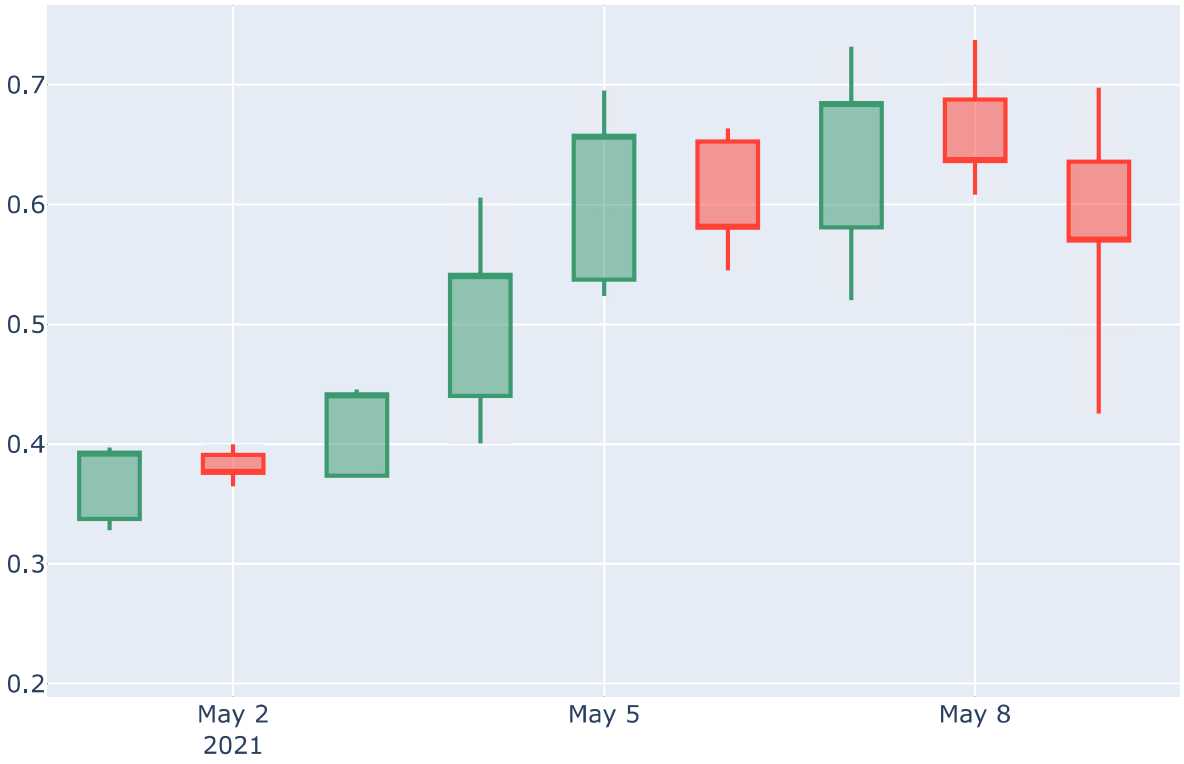
Carregando os dados

Vamos começar lendo o arquivo FuturosDogecoin.csv em um dataframe do pandas, mas antes vamos dar uma olhadinha no gráfico de variação do último mês do índice Ibovespa.

```
DataSet=pd.read_csv('FuturosDogecoin-teste.csv')
```

```
fig = go.Figure(data=[go.Candlestick(x=DataSet['Date'],
```

```
                    open=DataSet['Open'], high=DataSet['High'],
                    low=DataSet['Low'], close=DataSet['Close'])
                         ])

fig.update_layout(xaxis_rangeslider_visible=False)
fig.show()
```



Rede Neural Recorrente (RNN)

Antes de avançar para LSTM, primeiro vamos introduzir o conceito de Redes Recorrentes. Elas são redes utilizadas para reconhecer padrões quando os resultados do passado influenciam no resultado atual. Um exemplo disso são as séries temporais, em que a ordem dos dados é muito importante.

Nesta arquitetura, um neurônio tem como entrada seu estado anterior, além das entradas da camada anterior. A imagem abaixo ilustra esta nova modelagem.

Observe que H representa o estado. Assim, no estado H_1, o neurônio recebe como parâmetro de entrada X_1 e, além disso, seu estado anterior H_0. O principal problema desta arquitetura é

que os estados mais antigos são esquecidos muito rapidamente. Ou seja, para sequências em que precisamos lembrar além de um passado imediato, as redes RNNs são limitadas.

Rede LSTM

Uma rede LSTM tem origem em uma RNN (Rede Neural Recorrente). Mas ela resolve o problema de memória mudando sua arquitetura.

Nesta nova arquitetura, cada neurônio possui 3 gates, cada um com uma função diferente. São eles:

- Input Gate
- Output Gate
- Forget Gate

Agora, um neurônio LSTM recebe entradas de seu estado anterior, assim como ocorria na Rede Recorrente:

Agora vamos ler o arquivo do período desejável

```
DataSet=pd.read_csv('FuturosDogecoin-treino.csv')
DataSet=DataSet.dropna()
DataSet.head()
```

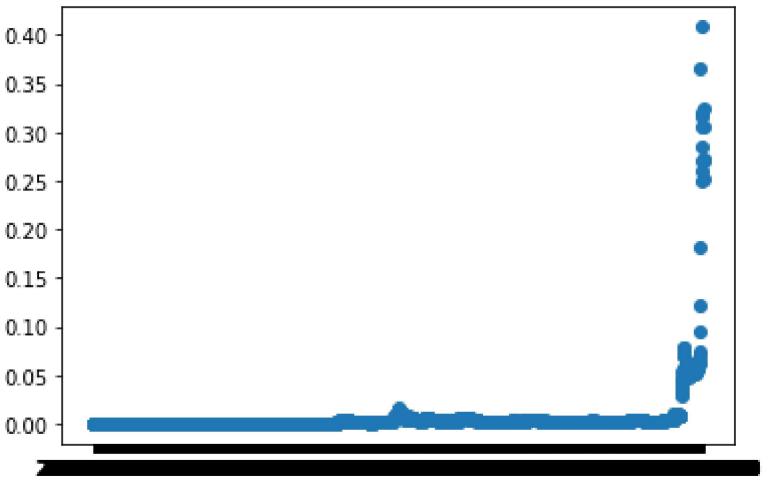
	Date	Open	High	Low	Close	Adj Close	Volume
0	2014-09-17	0.000293	0.000299	0.000260	0.000268	0.000268	1463600.0
1	2014-09-18	0.000268	0.000325	0.000267	0.000298	0.000298	2215910.0
2	2014-09-19	0.000298	0.000307	0.000275	0.000277	0.000277	883563.0
3	2014-09-20	0.000276	0.000310	0.000267	0.000292	0.000292	993004.0
4	2014-09-21	0.000293	0.000299	0.000284	0.000288	0.000288	539140.0

```
DataSet.describe()
```

	Open	High	Low	Close	Adj Close	Volume
count	2414.000000	2414.000000	2414.000000	2414.000000	2414.000000	2.414000e+03
mean	0.005506	0.006073	0.005063	0.005634	0.005634	2.734802e+08
std	0.025189	0.029055	0.022009	0.026074	0.026074	2.204348e+09
min	0.000087	0.000089	0.000085	0.000087	0.000087	1.669500e+04

Inicialmente iremos criar uma RNN baseada apenas no Valor de Abertura

```
plt.scatter(DataSet['Date'],DataSet['Open'],)
plt.show()

base_treinamento = DataSet.iloc[:, 1:2].values

#DataSet.drop(['Date','Close','High','Low', 'Volume'],axis=1,inplace=True)
```




```
base_treinamento
```

```
array([[2.93000e-04],
       [2.68000e-04],
       [2.98000e-04],
       ...,
       [2.72273e-01],
       [3.23232e-01],
       [3.04702e-01]])
```

Normalizar os dados do Mini Índice

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
DataScaled=scaler.fit_transform(base_treinamento)


print(DataScaled)
```

```
[[5.04471676e-04]
 [4.43249385e-04]
 [5.16716134e-04]
 ...
 [6.66554018e-01]
 [7.91347086e-01]
 [7.45969124e-01]]
```

Definição dos previsores

```
previsores = []
preco_real = []
NRecursao = 90
DataSetLen = len(DataScaled)
print(DataSetLen)
```

```
2414
```

```
for i in range(NRecursao, DataSetLen):
    previsores.append(DataScaled[i-NRecursao:i,0])
    preco_real.append(DataScaled[i,0])

previsores, preco_real = np.array(previsores), np.array(preco_real)
```

```
previsores.shape
```

```
(2324, 90)
```

Tranformar para o formato do Tensor do Keras

```
previsores = np.reshape(previsores, (previsores.shape[0], previsores.shape[1], 1))
```

```
previsores.shape
```

```
(1125, 90, 1)
```

Estrutura da Rede Neural

```
# Camada de entrada
regressor = Sequential()
regressor.add(LSTM(units = 100, return_sequences = True, input_shape = (previsores.shape[1
```

```
regressor.add(Dropout(0.3))

# Cada Oculta 1
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.3))

# Cada Oculta 2
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.3))

# Cada Oculta 3
regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.3))

# Camada de Saída
regressor.add(Dense(units = 1, activation = 'linear'))
```

Construindo a Rede

```
regressor.compile(optimizer = 'rmsprop', loss = 'mean_squared_error',
                  metrics = ['mean_absolute_error'])
regressor.fit(previsores, preco_real, epochs = 20, batch_size = 32)
```

```
Epoch 1/20
73/73 [==============================] - 20s 190ms/step - loss: 0.0047 - mean_absolu
Epoch 2/20
73/73 [==============================] - 14s 192ms/step - loss: 0.0020 - mean_absolu
Epoch 3/20
73/73 [==============================] - 14s 191ms/step - loss: 0.0022 - mean_absolu
Epoch 4/20
73/73 [==============================] - 14s 195ms/step - loss: 0.0015 - mean_absolu
Epoch 5/20
73/73 [==============================] - 14s 194ms/step - loss: 0.0016 - mean_absolu
Epoch 6/20
73/73 [==============================] - 14s 197ms/step - loss: 6.6706e-04 - mean_ab
Epoch 7/20
73/73 [==============================] - 14s 193ms/step - loss: 0.0014 - mean_absolu
Epoch 8/20
73/73 [==============================] - 14s 193ms/step - loss: 0.0012 - mean_absolu
Epoch 9/20
73/73 [==============================] - 14s 194ms/step - loss: 0.0012 - mean_absolu
Epoch 10/20
73/73 [==============================] - 14s 192ms/step - loss: 8.8327e-04 - mean_ab
Epoch 11/20
73/73 [==============================] - 14s 193ms/step - loss: 8.7988e-04 - mean_ab
Epoch 12/20
73/73 [==============================] - 14s 193ms/step - loss: 6.8386e-04 - mean_ab
Epoch 13/20
73/73 [==============================] - 14s 192ms/step - loss: 8.8988e-04 - mean_ab
Epoch 14/20
73/73 [==============================] - 14s 194ms/step - loss: 5.5377e-04 - mean_ab
Epoch 15/20
73/73 [==============================] - 14s 193ms/step - loss: 7.9201e-04 - mean_ab
Epoch 16/20
73/73 [==============================] - 14s 192ms/step - loss: 0.0011 - mean_absolu
```

```
Epoch 17/20
73/73 [==============================] - 14s 192ms/step - loss: 7.4402e-04 - mean_ab
Epoch 18/20
73/73 [==============================] - 14s 192ms/step - loss: 7.5662e-04 - mean_ab
Epoch 19/20
73/73 [==============================] - 14s 191ms/step - loss: 5.8007e-04 - mean_ab
Epoch 20/20
73/73 [==============================] - 14s 191ms/step - loss: 4.6829e-04 - mean_ab
<tensorflow.python.keras.callbacks.History at 0x7f40e8c56c90>
```

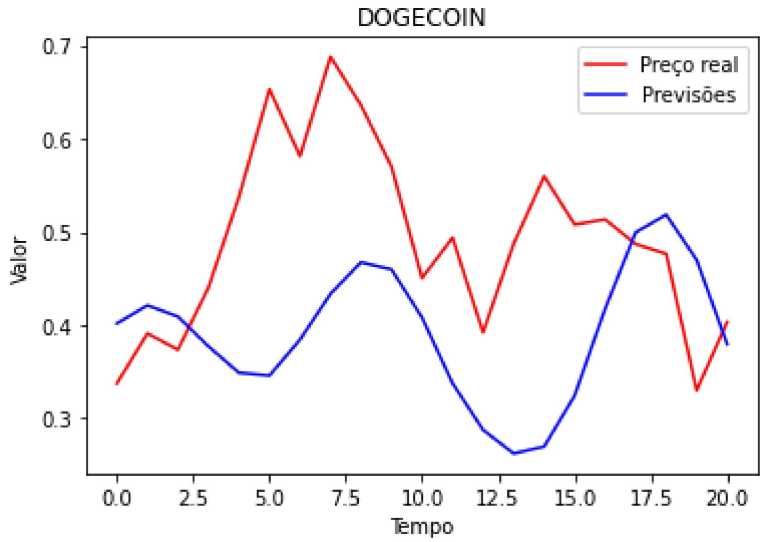
Conjunto de dados para o Teste

```
DataSet_teste=pd.read_csv('FuturosDogecoin-teste.csv')

preco_real_teste = DataSet_teste.iloc[:, 1:2].values

base_completa = pd.concat((DataSet['Open'], DataSet_teste['Open']), axis = 0)
entradas = base_completa[len(base_completa) - len(DataSet_teste) - NRecursao:].values

entradas = entradas.reshape(-1, 1)
entradas = scaler.transform(entradas)


DataSetTestLen = len(DataSet_teste)
NPredictions = 90

X_teste = []
for i in range(NRecursao, DataSetTestLen+NRecursao):
    X_teste.append(entradas[i-NRecursao:i, 0])

X_teste = np.array(X_teste)
X_teste = np.reshape(X_teste, (X_teste.shape[0], X_teste.shape[1], 1))

previsoes = regressor.predict(X_teste)
previsoes = scaler.inverse_transform(previsoes)



RNN=[]
predictions_teste=X_teste[0].T
predictions_teste=np.reshape(predictions_teste, (predictions_teste.shape[0], predictions_t

predictions_teste[0][NRecursao-1][0]=regressor.predict(predictions_teste)[0][0]
RNN.append(regressor.predict(predictions_teste)[0])

for i in range(NPredictions-1):
    predictions_teste=np.roll(predictions_teste,-1)
    predictions_teste[0][NRecursao-1][0]=regressor.predict(predictions_teste)[0][0]
    RNN.append(regressor.predict(predictions_teste)[0])
RNN = scaler.inverse_transform(RNN)

print(RNN.mean())
print(previsoes.mean())
print(preco_real_teste.mean())
```

```
print(preco_real_teste.mean())
```

```
0.29071910685553554
0.39167258
0.490889380952381
```

```
plt.plot(preco_real_teste, color = 'red', label = 'Preço real')
plt.plot(previsoes, color = 'blue', label = 'Previsões')
#plt.plot(RNN, color = 'green', label = 'RNN')

plt.title('DOGECOIN')
plt.xlabel('Tempo')
plt.ylabel('Valor')
plt.legend()
plt.show()
```




```
np.shape(previsoes)
```

```
(21, 1)
```

✓ 0s conclusão: 18:44