TrabalhoLSTM

May 21, 2021

0.1 Modelo Futuros Mini Ethereum - 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 mini índice é um contrato futuro derivado do Índice Ethereum, 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 Mini Índice da Ibovespa.

O dataset **"FuturosMiniEthereum.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)

0.1.1 Bibliotecas

[1]:
```
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
```

0.1.2 Carregando os dados

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

[2]:
```
DataSet=pd.read_csv('FuturosEthereum-teste.csv')
```

[3]:
```
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()
```

0.2 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.

0.2.1 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:

0.2.2 Agora vamos ler o arquivo do período desejável

[4]:
```
DataSet=pd.read_csv('FuturosEthereum-treino.csv')
DataSet=DataSet.dropna()
DataSet.head()
```

[4]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2015-08-07	2.831620	3.536610	2.521120	2.772120	2.772120	164329.0
1	2015-08-08	2.793760	2.798810	0.714725	0.753325	0.753325	674188.0
2	2015-08-09	0.706136	0.879810	0.629191	0.701897	0.701897	532170.0
3	2015-08-10	0.713989	0.729854	0.636546	0.708448	0.708448	405283.0
4	2015-08-11	0.708087	1.131410	0.663235	1.067860	1.067860	1463100.0

[5]:
```
DataSet.describe()
```

[5]:

	Open	High	…	Adj Close	Volume
count	2090.000000	2090.000000	…	2090.000000	2.090000e+03
mean	308.398014	319.833590	…	309.636474	6.075951e+09
std	425.797837	443.141785	…	429.030976	8.970136e+09
min	0.431589	0.482988	…	0.434829	1.021280e+05
25%	13.185075	13.531575	…	13.176775	3.173252e+07
50%	189.438515	196.909904	…	189.644058	1.972050e+09
75%	359.281754	371.925491	…	359.041008	8.570984e+09
max	2757.734131	2797.972412	…	2773.207031	6.073363e+10

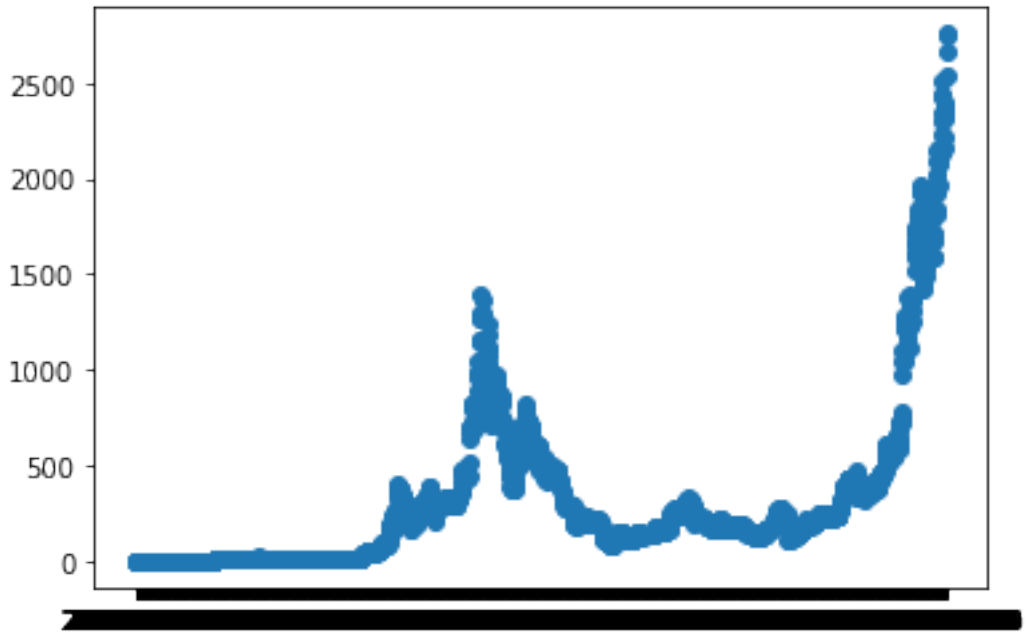
[8 rows x 6 columns]

0.2.3 Inicialmente iremos criar uma RNN baseada apenas no Valor de Abertura

[6]:
```
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)
```



[8]:
```
base_treinamento
```

[8]:
```
array([[2.83162000e+00],
       [2.79376000e+00],
```

```
        [7.06136000e-01],
        …,
        [2.66468555e+03],
        [2.74864966e+03],
        [2.75773413e+03]])
```

0.2.4 Normalizar os dados do Mini Índice

[9]:
```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
DataScaled=scaler.fit_transform(base_treinamento)
```

[10]:
```
print(DataScaled)
```

```
[[8.70427152e-04]
 [8.56696341e-04]
 [9.95708653e-05]
 …
 [9.66253763e-01]
 [9.96705304e-01]
 [1.00000000e+00]]
```

0.2.5 Definição dos previsores

[18]:
```
previsores = []
preco_real = []
NRecursao = 90
DataSetLen = len(DataScaled)
print(DataSetLen)
```

```
2090
```

[19]:
```
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)
```

[20]:
```
previsores.shape
```

[20]: (2000, 90)

0.2.6 Tranformar para o formato do Tensor do Keras

[21]:
```
previsores = np.reshape(previsores, (previsores.shape[0], previsores.shape[1],␣
↪1))
```

[22]:
```
previsores.shape
```

[22]:
```
(2000, 90, 1)
```

0.2.7 Estrutura da Rede Neural

[ ]:
```

```

[23]:
```
# Camada de entrada
regressor = Sequential()
regressor.add(LSTM(units = 100, return_sequences = True, input_shape =␣
↪(previsores.shape[1], 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'))
```

0.2.8 Construindo a Rede

[24]:
```
regressor.compile(optimizer = 'rmsprop', loss = 'mean_squared_error',
                  metrics = ['mean_absolute_error'])
regressor.fit(previsores, preco_real, epochs = 100, batch_size = 32)
```

```
Epoch 1/100
63/63 [==============================] - 18s 182ms/step - loss: 0.0106 -
mean_absolute_error: 0.0587
Epoch 2/100
63/63 [==============================] - 11s 181ms/step - loss: 0.0032 -
mean_absolute_error: 0.0332
Epoch 3/100
```

```
63/63 [==============================] - 11s 176ms/step - loss: 0.0027 -
mean_absolute_error: 0.0317
Epoch 4/100
63/63 [==============================] - 11s 173ms/step - loss: 0.0026 -
mean_absolute_error: 0.0285
Epoch 5/100
63/63 [==============================] - 11s 176ms/step - loss: 0.0022 -
mean_absolute_error: 0.0284
Epoch 6/100
63/63 [==============================] - 11s 176ms/step - loss: 0.0020 -
mean_absolute_error: 0.0280
Epoch 7/100
63/63 [==============================] - 12s 184ms/step - loss: 0.0020 -
mean_absolute_error: 0.0256
Epoch 8/100
63/63 [==============================] - 12s 184ms/step - loss: 0.0021 -
mean_absolute_error: 0.0266
Epoch 9/100
63/63 [==============================] - 12s 183ms/step - loss: 0.0016 -
mean_absolute_error: 0.0252
Epoch 10/100
63/63 [==============================] - 12s 184ms/step - loss: 0.0019 -
mean_absolute_error: 0.0246
Epoch 11/100
63/63 [==============================] - 11s 181ms/step - loss: 0.0012 -
mean_absolute_error: 0.0222
Epoch 12/100
63/63 [==============================] - 11s 175ms/step - loss: 0.0016 -
mean_absolute_error: 0.0228
Epoch 13/100
63/63 [==============================] - 11s 178ms/step - loss: 0.0016 -
mean_absolute_error: 0.0235
Epoch 14/100
63/63 [==============================] - 11s 179ms/step - loss: 0.0014 -
mean_absolute_error: 0.0227
Epoch 15/100
63/63 [==============================] - 11s 176ms/step - loss: 0.0012 -
mean_absolute_error: 0.0206
Epoch 16/100
63/63 [==============================] - 11s 176ms/step - loss: 0.0013 -
mean_absolute_error: 0.0212
Epoch 17/100
63/63 [==============================] - 11s 176ms/step - loss: 0.0010 -
mean_absolute_error: 0.0206
Epoch 18/100
63/63 [==============================] - 11s 178ms/step - loss: 0.0010 -
mean_absolute_error: 0.0197
Epoch 19/100
```

```
63/63 [==============================] - 11s 177ms/step - loss: 0.0012 -
mean_absolute_error: 0.0204
Epoch 20/100
63/63 [==============================] - 11s 175ms/step - loss: 0.0011 -
mean_absolute_error: 0.0206
Epoch 21/100
63/63 [==============================] - 11s 176ms/step - loss: 0.0013 -
mean_absolute_error: 0.0207
Epoch 22/100
63/63 [==============================] - 11s 175ms/step - loss: 9.3160e-04 -
mean_absolute_error: 0.0189
Epoch 23/100
63/63 [==============================] - 11s 174ms/step - loss: 9.6013e-04 -
mean_absolute_error: 0.0185
Epoch 24/100
63/63 [==============================] - 11s 178ms/step - loss: 0.0012 -
mean_absolute_error: 0.0201
Epoch 25/100
63/63 [==============================] - 11s 176ms/step - loss: 9.7575e-04 -
mean_absolute_error: 0.0194
Epoch 26/100
63/63 [==============================] - 11s 175ms/step - loss: 0.0010 -
mean_absolute_error: 0.0187
Epoch 27/100
63/63 [==============================] - 11s 177ms/step - loss: 0.0011 -
mean_absolute_error: 0.0195
Epoch 28/100
63/63 [==============================] - 11s 177ms/step - loss: 0.0012 -
mean_absolute_error: 0.0211
Epoch 29/100
63/63 [==============================] - 11s 178ms/step - loss: 0.0011 -
mean_absolute_error: 0.0186
Epoch 30/100
63/63 [==============================] - 11s 181ms/step - loss: 7.8500e-04 -
mean_absolute_error: 0.0185
Epoch 31/100
63/63 [==============================] - 11s 176ms/step - loss: 9.8730e-04 -
mean_absolute_error: 0.0187
Epoch 32/100
63/63 [==============================] - 11s 176ms/step - loss: 0.0011 -
mean_absolute_error: 0.0199
Epoch 33/100
63/63 [==============================] - 11s 173ms/step - loss: 0.0010 -
mean_absolute_error: 0.0189
Epoch 34/100
63/63 [==============================] - 11s 173ms/step - loss: 0.0013 -
mean_absolute_error: 0.0195
Epoch 35/100
```

```
63/63 [==============================] - 11s 174ms/step - loss: 9.8068e-04 -
mean_absolute_error: 0.0176
Epoch 36/100
63/63 [==============================] - 11s 175ms/step - loss: 9.2939e-04 -
mean_absolute_error: 0.0185
Epoch 37/100
63/63 [==============================] - 11s 175ms/step - loss: 0.0012 -
mean_absolute_error: 0.0194
Epoch 38/100
63/63 [==============================] - 11s 174ms/step - loss: 9.6969e-04 -
mean_absolute_error: 0.0182
Epoch 39/100
63/63 [==============================] - 11s 173ms/step - loss: 8.2974e-04 -
mean_absolute_error: 0.0172
Epoch 40/100
63/63 [==============================] - 11s 175ms/step - loss: 8.5349e-04 -
mean_absolute_error: 0.0174
Epoch 41/100
63/63 [==============================] - 12s 184ms/step - loss: 9.1215e-04 -
mean_absolute_error: 0.0182
Epoch 42/100
63/63 [==============================] - 12s 187ms/step - loss: 8.4554e-04 -
mean_absolute_error: 0.0177
Epoch 43/100
63/63 [==============================] - 12s 184ms/step - loss: 0.0011 -
mean_absolute_error: 0.0192
Epoch 44/100
63/63 [==============================] - 12s 184ms/step - loss: 8.3003e-04 -
mean_absolute_error: 0.0178
Epoch 45/100
63/63 [==============================] - 11s 180ms/step - loss: 8.7891e-04 -
mean_absolute_error: 0.0176
Epoch 46/100
63/63 [==============================] - 11s 181ms/step - loss: 8.5457e-04 -
mean_absolute_error: 0.0181
Epoch 47/100
63/63 [==============================] - 11s 179ms/step - loss: 8.8883e-04 -
mean_absolute_error: 0.0176
Epoch 48/100
63/63 [==============================] - 11s 172ms/step - loss: 7.1617e-04 -
mean_absolute_error: 0.0170
Epoch 49/100
63/63 [==============================] - 11s 177ms/step - loss: 7.4395e-04 -
mean_absolute_error: 0.0163
Epoch 50/100
63/63 [==============================] - 11s 173ms/step - loss: 8.0962e-04 -
mean_absolute_error: 0.0160
Epoch 51/100
```

```
63/63 [==============================] - 11s 173ms/step - loss: 8.9467e-04 -
mean_absolute_error: 0.0171
Epoch 52/100
63/63 [==============================] - 11s 173ms/step - loss: 9.1947e-04 -
mean_absolute_error: 0.0180
Epoch 53/100
63/63 [==============================] - 11s 171ms/step - loss: 7.5348e-04 -
mean_absolute_error: 0.0174
Epoch 54/100
63/63 [==============================] - 11s 172ms/step - loss: 0.0010 -
mean_absolute_error: 0.0180
Epoch 55/100
63/63 [==============================] - 11s 171ms/step - loss: 8.4986e-04 -
mean_absolute_error: 0.0172
Epoch 56/100
63/63 [==============================] - 11s 177ms/step - loss: 7.0141e-04 -
mean_absolute_error: 0.0167
Epoch 57/100
63/63 [==============================] - 11s 174ms/step - loss: 7.7178e-04 -
mean_absolute_error: 0.0165
Epoch 58/100
63/63 [==============================] - 11s 178ms/step - loss: 8.0533e-04 -
mean_absolute_error: 0.0172
Epoch 59/100
63/63 [==============================] - 11s 177ms/step - loss: 7.3314e-04 -
mean_absolute_error: 0.0165
Epoch 60/100
63/63 [==============================] - 11s 177ms/step - loss: 7.0981e-04 -
mean_absolute_error: 0.0161
Epoch 61/100
63/63 [==============================] - 11s 177ms/step - loss: 6.7033e-04 -
mean_absolute_error: 0.0157
Epoch 62/100
63/63 [==============================] - 11s 177ms/step - loss: 7.7409e-04 -
mean_absolute_error: 0.0160
Epoch 63/100
63/63 [==============================] - 11s 177ms/step - loss: 7.0291e-04 -
mean_absolute_error: 0.0157
Epoch 64/100
63/63 [==============================] - 11s 179ms/step - loss: 8.6078e-04 -
mean_absolute_error: 0.0171
Epoch 65/100
63/63 [==============================] - 12s 183ms/step - loss: 9.4839e-04 -
mean_absolute_error: 0.0178
Epoch 66/100
63/63 [==============================] - 11s 181ms/step - loss: 5.9985e-04 -
mean_absolute_error: 0.0158
Epoch 67/100
```

```
63/63 [==============================] - 12s 186ms/step - loss: 6.8990e-04 -
mean_absolute_error: 0.0164
Epoch 68/100
63/63 [==============================] - 11s 181ms/step - loss: 8.1053e-04 -
mean_absolute_error: 0.0164
Epoch 69/100
63/63 [==============================] - 11s 176ms/step - loss: 7.3680e-04 -
mean_absolute_error: 0.0160
Epoch 70/100
63/63 [==============================] - 11s 175ms/step - loss: 8.0002e-04 -
mean_absolute_error: 0.0161
Epoch 71/100
63/63 [==============================] - 11s 175ms/step - loss: 7.6728e-04 -
mean_absolute_error: 0.0158
Epoch 72/100
63/63 [==============================] - 11s 175ms/step - loss: 7.9832e-04 -
mean_absolute_error: 0.0162
Epoch 73/100
63/63 [==============================] - 11s 176ms/step - loss: 7.4304e-04 -
mean_absolute_error: 0.0164
Epoch 74/100
63/63 [==============================] - 11s 181ms/step - loss: 6.9806e-04 -
mean_absolute_error: 0.0159
Epoch 75/100
63/63 [==============================] - 11s 178ms/step - loss: 7.9505e-04 -
mean_absolute_error: 0.0165
Epoch 76/100
63/63 [==============================] - 11s 179ms/step - loss: 6.8144e-04 -
mean_absolute_error: 0.0159
Epoch 77/100
63/63 [==============================] - 11s 175ms/step - loss: 7.8674e-04 -
mean_absolute_error: 0.0162
Epoch 78/100
63/63 [==============================] - 11s 175ms/step - loss: 7.3630e-04 -
mean_absolute_error: 0.0158
Epoch 79/100
63/63 [==============================] - 11s 172ms/step - loss: 5.9120e-04 -
mean_absolute_error: 0.0156
Epoch 80/100
63/63 [==============================] - 10s 166ms/step - loss: 7.6319e-04 -
mean_absolute_error: 0.0161
Epoch 81/100
63/63 [==============================] - 11s 167ms/step - loss: 9.2105e-04 -
mean_absolute_error: 0.0170
Epoch 82/100
63/63 [==============================] - 11s 171ms/step - loss: 6.3826e-04 -
mean_absolute_error: 0.0158
Epoch 83/100
```

```
63/63 [==============================] - 11s 175ms/step - loss: 7.0190e-04 -
mean_absolute_error: 0.0156
Epoch 84/100
63/63 [==============================] - 11s 173ms/step - loss: 7.9875e-04 -
mean_absolute_error: 0.0166
Epoch 85/100
63/63 [==============================] - 11s 174ms/step - loss: 7.7809e-04 -
mean_absolute_error: 0.0162
Epoch 86/100
63/63 [==============================] - 11s 175ms/step - loss: 6.4176e-04 -
mean_absolute_error: 0.0147
Epoch 87/100
63/63 [==============================] - 11s 178ms/step - loss: 7.2543e-04 -
mean_absolute_error: 0.0153
Epoch 88/100
63/63 [==============================] - 11s 179ms/step - loss: 8.3206e-04 -
mean_absolute_error: 0.0163
Epoch 89/100
63/63 [==============================] - 11s 177ms/step - loss: 7.9672e-04 -
mean_absolute_error: 0.0157
Epoch 90/100
63/63 [==============================] - 11s 173ms/step - loss: 7.2527e-04 -
mean_absolute_error: 0.0155
Epoch 91/100
63/63 [==============================] - 11s 174ms/step - loss: 7.6337e-04 -
mean_absolute_error: 0.0156
Epoch 92/100
63/63 [==============================] - 11s 174ms/step - loss: 5.4174e-04 -
mean_absolute_error: 0.0143
Epoch 93/100
63/63 [==============================] - 11s 174ms/step - loss: 8.3118e-04 -
mean_absolute_error: 0.0164
Epoch 94/100
63/63 [==============================] - 11s 178ms/step - loss: 5.6554e-04 -
mean_absolute_error: 0.0144
Epoch 95/100
63/63 [==============================] - 11s 175ms/step - loss: 6.7491e-04 -
mean_absolute_error: 0.0144
Epoch 96/100
63/63 [==============================] - 11s 180ms/step - loss: 6.5369e-04 -
mean_absolute_error: 0.0161
Epoch 97/100
63/63 [==============================] - 11s 175ms/step - loss: 5.4592e-04 -
mean_absolute_error: 0.0152
Epoch 98/100
63/63 [==============================] - 11s 175ms/step - loss: 6.7181e-04 -
mean_absolute_error: 0.0155
Epoch 99/100
```

```
63/63 [==============================] - 11s 173ms/step - loss: 7.1616e-04 -
mean_absolute_error: 0.0154
Epoch 100/100
63/63 [==============================] - 11s 177ms/step - loss: 7.2076e-04 -
mean_absolute_error: 0.0162
```

[24]:
```
<tensorflow.python.keras.callbacks.History at 0x7fd1f5b2b950>
```

0.2.9 Conjunto de dados para o Teste

[25]:
```
DataSet_teste=pd.read_csv('FuturosEthereum-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)
```

[26]:
```
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)
```

[27]:
```
RNN=[]
predictions_teste=X_teste[0].T
predictions_teste=np.reshape(predictions_teste, (predictions_teste.shape[0],␣
 ↪predictions_teste.shape[1], 1))

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])
```

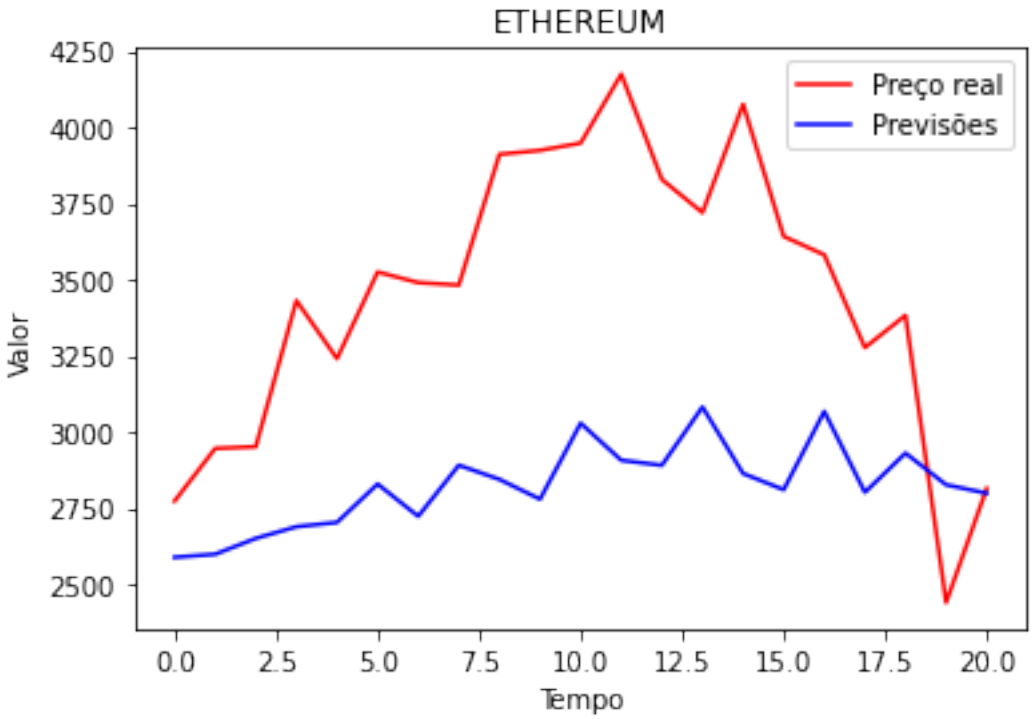
```
RNN = scaler.inverse_transform(RNN)

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

```
2096.7557163208708
2823.6003
3455.175944095238
```

[28]:
```
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('ETHEREUM')
plt.xlabel('Tempo')
plt.ylabel('Valor')
plt.legend()
plt.show()
```



[29]:
```
np.shape(previsoes)
```

[29]:
```
(21, 1)
```

[ ]: