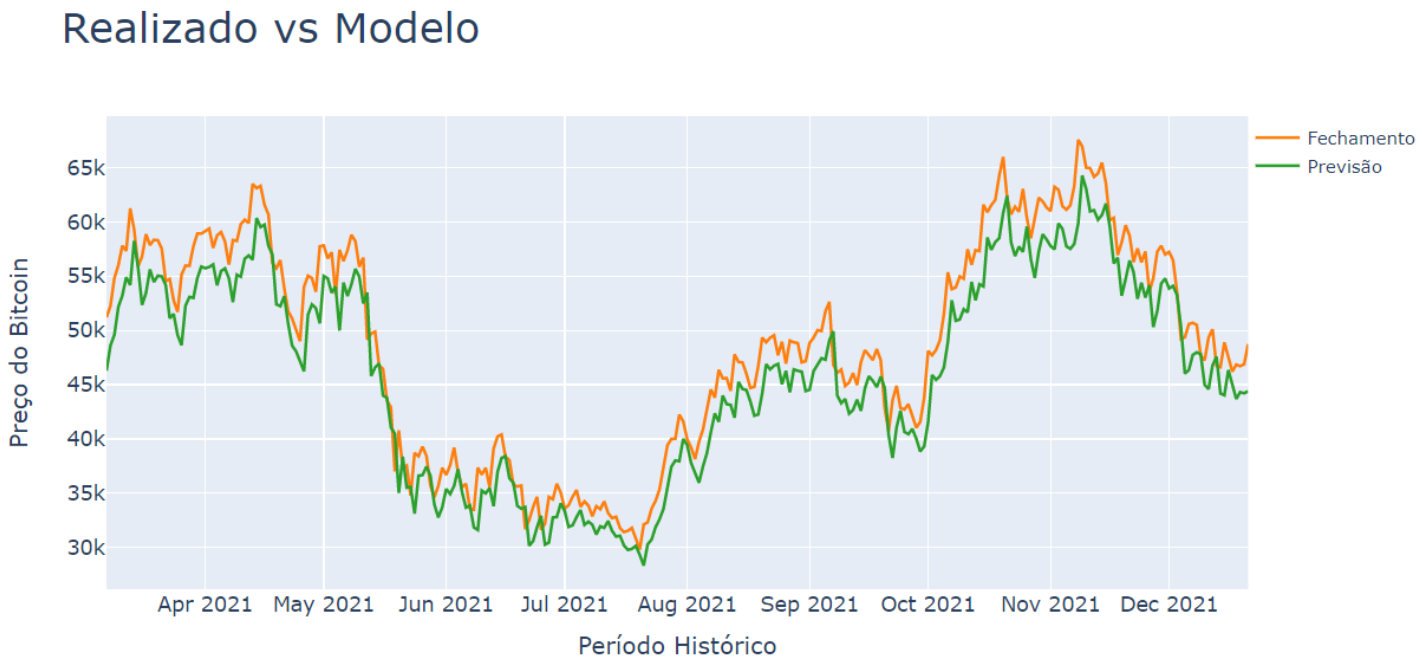
@Odemir Depieri Jr

Prevendo preço do Bitcoin



com Deep Learrning

Keras

yahoo! finance

scikit learn

plotly

NumPy

pandas

matplotlib

@Odemir Depieri Jr

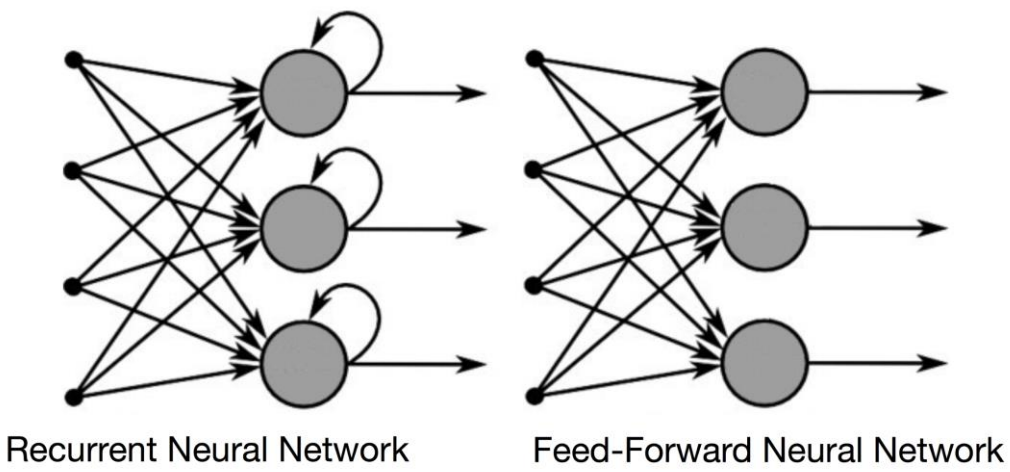
Redes Neurais Recorrentes

RRN

As **redes recorrentes** são um tipo de **rede neural artificial** projetada para reconhecer padrões em sequências de dados, como texto, genomas, caligrafia, palavra falada ou dados de séries numéricas que emanam de sensores, bolsas de valores e agências governamentais. Esses algoritmos consideram tempo e sequência, eles têm uma dimensão temporal.

As redes recorrentes, por outro lado, tomam como entrada **não apenas o exemplo de entrada atual** que veem, mas também o que **perceberam anteriormente** no tempo.

Diferença entre uma rede neural vs rede recorrente



Artigos

https://www.deeplearningbook.com.br/redes-neurais-recorrentes/

https://www.monolitonimbus.com.br/modelo-sequencial-do-keras/

https://keras.io/api/optimizers/

https://qastack.com.br/programming/38714959/understanding-keras-lstms

https://medium.com/luisfredgs/an%C3%A1lise-de-sentimentos-com-redes-neurais-recorrentes-lstm-a5352b21e6aa

Vídeos

https://www.youtube.com/watch?v=bDDP0m4jjH0

https://www.youtube.com/watch?v=bIcadBu--u8&t=4597s

@Odemir Depieri Jr

Vamos importar nossas Libs

In [77]:
```
# Libs Necessárias
import numpy as np
import pandas as pd
import pandas_datareader as web

# Libs para gráficos
import matplotlib.pyplot as plt

# Libs para utilizar o Plotly
import plotly.express as px
import plotly.graph_objects as Dash

# pip install yfinance
import yfinance as yf

# Libs para uso de Machine Learning do Keras
from keras.models import Sequential
from keras.layers import Dense, LSTM

# Ignorando avisos
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

Carregamento dos Dados

In [27]:
```
# Pegar os dados do Bitcoin com o Yfinance
Base_Dados = web.get_data_yahoo( 'BTC-USD', start='2018-01-01' )

# Verificando
Base_Dados.head()
```

Out[27]:

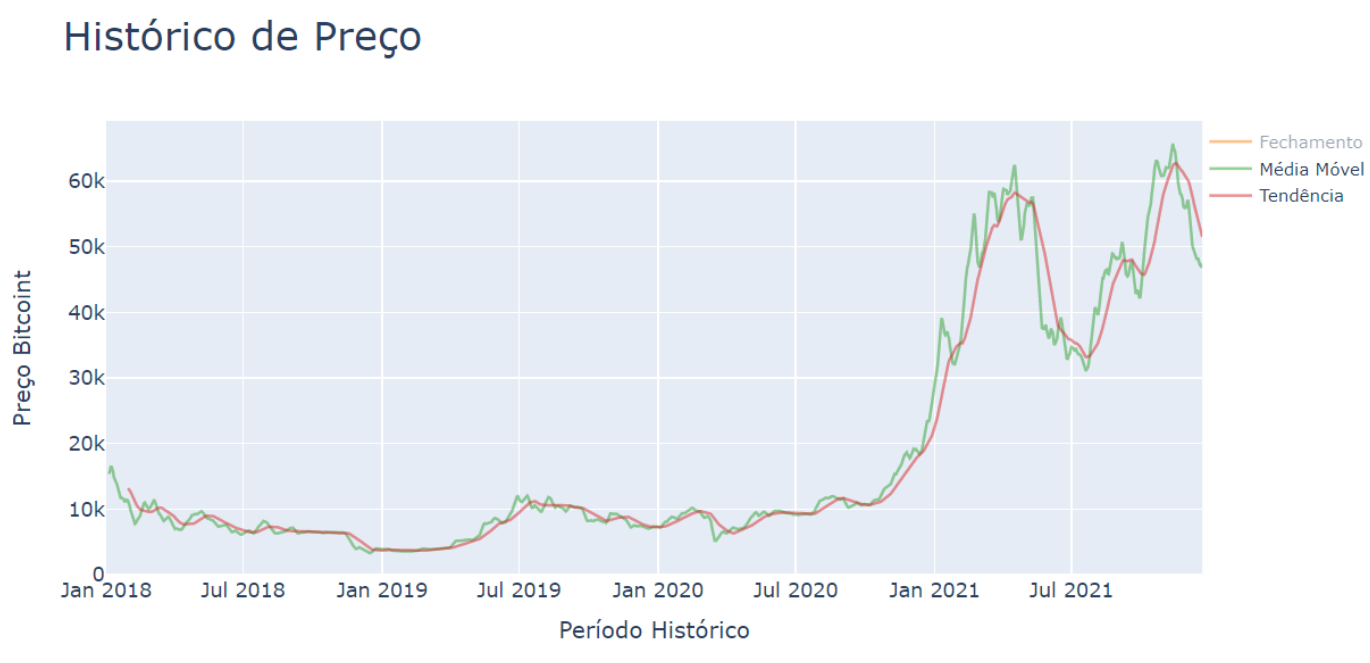
Date	High	Low	Open	Close	Volume	Adj Close
2018-01-01	14112.200195	13154.700195	14112.200195	13657.200195	10291200000	13657.200195
2018-01-02	15444.599609	13163.599609	13625.000000	14982.099609	16846600192	14982.099609
2018-01-03	15572.799805	14844.500000	14978.200195	15201.000000	16871900160	15201.000000
2018-01-04	15739.700195	14522.200195	15270.700195	15599.200195	21783199744	15599.200195
2018-01-05	17705.199219	15202.799805	15477.200195	17429.500000	23840899072	17429.500000

Plotando o Fechamento

In [28]:
```
# Plot para verificar preço de fechamento
# Chamando o gráfico
fig = px.line( Base_Dados, y='Close' )
fig.show()
```

Vamos gerar a média móvel de 5 dias e 30 dias e plotar com o preço de fechamento

In [29]:
```
 # --- Gráficos com as médias
# Gerar a média movel do fechamento das ações
Media_Movel = Base_Dados['Close'].rolling(5).mean()
Media_Movel_Tendencia = Base_Dados['Close'].rolling(30).mean()

# Definindo uma figura
Figura = Dash.Figure()

# Incluindo o Eixo no Gráfico - Abertura
Figura.add_trace(Dash.Scatter(x = Base_Dados.index, y = Base_Dados.Close,
                    mode='lines',
                    name='Fechamento',
                    marker_color = '#FF7F0E'))

# Incluindo o Eixo no Gráfico - Maior
Figura.add_trace(Dash.Scatter(x = Base_Dados.index, y = Media_Movel,
                    mode='lines',
                    name='Média Móvel', opacity=0.5,
                    marker_color = '#2CA02C',
                    ))

# Incluindo o Eixo no Gráfico - Menor
Figura.add_trace(Dash.Scatter(x = Base_Dados.index, y = Media_Movel_Tendencia,
                    mode='lines',
                    name='Tendência', opacity=0.5,
                    marker_color = '#D62728'))

# Modificando o Layout do Gráfico
Figura.update_layout(
    title='Histórico de Preço', # Titulo
    titlefont_size = 28, # Tamanho da Fonte

    # Parametros para mexer no eixo X
    xaxis = dict(
        title='Período Histórico', # Titulo do Eixo x
        titlefont_size=16, # Tamanho fonte do Titulo
        tickfont_size=14), # Tamanho da fonte do eixo

    # Tamanho do Grafico
    height = 500,

    # Parametros para mexer no eixo y
    yaxis=dict(
        title='Preço Bitcoint', # Titulo do Eixo y
        titlefont_size=16, # Tamanho fonte do Titulo
        tickfont_size=14), # Tamanho da fonte do eixo

    # Parametros para mexer na legenda
    legend=dict(
        y=1, x=1, # Posição da Legenda
        bgcolor='rgba(255, 255, 255, 0)', # Cor de fundo
        bordercolor='rgba(255, 255, 255, 0)')) # Cor da Bornda

# Mostrando o Gráfico
Figura.show()
```



Na legenda é possível filtrar quais linhas quer visualizar ... Retirei o preço de fechamento para verificar a média móvel e a tendência

Analisar algumas estatísticas

In [30]:
```
# Estatisticas
Base_Dados.describe()
```

Out[30]:

	High	Low	Open	Close	Volume	Adj Close
count	1451.000000	1451.000000	1451.000000	1451.000000	1.451000e+03	1451.000000
mean	18629.674165	17602.708805	18143.495957	18164.297852	2.574279e+10	18164.297852
std	18120.960422	17054.668580	17625.322246	17637.313523	2.123456e+10	17637.313523
min	3275.377930	3191.303467	3236.274658	3236.761719	2.923670e+09	3236.761719
25%	7268.298096	6875.060791	7092.000732	7089.992188	7.920519e+09	7089.992188
50%	9594.419922	9255.035156	9426.110352	9427.687500	2.200451e+10	9427.687500
75%	24147.757812	22987.091797	23757.772461	23826.430664	3.633740e+10	23826.430664
max	68789.625000	66382.062500	67549.734375	67566.828125	3.509679e+11	67566.828125

Vamos escalonar nossas dados para deixar em escalas mais comparações

In [85]:
```
# --- Pre processamento

# Função para escolanemnto
from sklearn.preprocessing import MinMaxScaler

# Chamando a Função
Funcao_MinMAx = MinMaxScaler( feature_range=(0,1) )

# Criando uma copia
Dados_Treino = Base_Dados.filter(['Close'])

# Aplicando a função
Dados_Treino_Escalados = Funcao_MinMAx.fit_transform( Dados_Treino )

# VErificando
Dados_Treino_Escalados[0:5]
```

Out[85]:
```
array([[0.16198395],
       [0.18257929],
       [0.18598206],
       [0.19217201],
       [0.22062372]])
```

Separando os dados de treinamento

In [86]:
```
# --- Separação dos dados de treino e teste

# Listas para receber os dados
x_treinamento = []
y_treinamento = []

# Loop para separar os dados de treino e teste
# Nesse Loop vamos seprar os dados em blocos de 60 valores
for Loop in range( 60, len(Dados_Treino_Escalados) ):

    # Separando os dados de treinamento x
    Filtrando_Amostra_Treinamento_x = Dados_Treino_Escalados[ Loop-60 : Loop, 0 ]
    x_treinamento.append( Filtrando_Amostra_Treinamento_x )

    # Separando os dados de treinamento y
    Filtrando_Amostra_Treinamento_y = Dados_Treino_Escalados[Loop, 0]
    y_treinamento.append( Filtrando_Amostra_Treinamento_y )

# Transformando as listas em Array
x_treinamento, y_treinamento = np.array(x_treinamento), np.array(y_treinamento)

# Convertendo o array para Matriz
x_treinamento = np.reshape(x_treinamento, (x_treinamento.shape[0], x_treinamento.shape[1], 1))

# Verificando a demisão da nossa matriz
x_treinamento.shape
```

Out[86]:
```
(1391, 60, 1)
```

Treinamento do modelo

In [87]:
```
# --- Treinamento do Modelo

# Definindo a função do Keras
# Essa função é uma pilha linear de camadas do Keras
Modelo = Sequential()

# Adicinando as camadas e parametros para nossa rede neural
# Treinamento da Rede Neural Recorrente

# LSTM - Long Short-Term Memory
Modelo.add(LSTM(50, return_sequences = True,
                input_shape = (x_treinamento.shape[1], 1)))
Modelo.add(LSTM(50, return_sequences = False))

# Adicionando as camadas na rede neural
Modelo.add(Dense(25))
Modelo.add(Dense(1))
Modelo.compile(optimizer = 'adam', loss = 'mean_squared_error')

# Treinando o modelo
Modelo.fit(x_treinamento, y_treinamento, batch_size = 1, epochs = 10)
```

```
Epoch 1/10
1391/1391 [==============================] - 55s 37ms/step - loss: 0.0025
Epoch 2/10
1391/1391 [==============================] - 48s 34ms/step - loss: 0.0011
Epoch 3/10
1391/1391 [==============================] - 50s 36ms/step - loss: 9.9706e-04
Epoch 4/10
1391/1391 [==============================] - 46s 33ms/step - loss: 7.3004e-04
Epoch 5/10
1391/1391 [==============================] - 47s 34ms/step - loss: 7.1187e-04
Epoch 6/10
1391/1391 [==============================] - 44s 31ms/step - loss: 6.2487e-04
Epoch 7/10
1391/1391 [==============================] - 44s 32ms/step - loss: 4.8589e-04
Epoch 8/10
1391/1391 [==============================] - 46s 33ms/step - loss: 6.6631e-04
Epoch 9/10
1391/1391 [==============================] - 45s 32ms/step - loss: 5.1454e-04
Epoch 10/10
1391/1391 [==============================] - 47s 34ms/step - loss: 5.2354e-04
```

Inclui apenas 10 épocas com 1 batch.
Sugiro aumentar um pouco ... Fiz dessa forma porque
não estava com paciência para esperar hahaha

Separar os dados de Teste e fazer as previsões

In [89]:
```
import math

# Arredondando o numero para cima usando o 'math.ceil'
Dados_Fechamento_Valores_Tamanho = math.ceil( len(Dados_Treino) * .8)

Dados_Fechamento_Valores_Tamanho
```

Out[89]:
```
1161
```

In [110]:
```
# Definindo amostra para ser testada
Dados_Teste = Dados_Treino_Escalados[Dados_Fechamento_Valores_Tamanho - 60: , :]

# Lista para recever os dados de teste
x_teste = []

# Lista com os dados de teste
y_Teste = Dados_Treino_Escalados[Dados_Fechamento_Valores_Tamanho:, :]

# Loop para fixar amostra para teste
for Loop in range (60, len(Dados_Teste)):
    x_teste.append(Dados_Teste[Loop - 60:Loop, 0])

# Transformando os dados em um array
x_teste = np.array(x_teste)

# Convertendo o array para Matriz
x_teste = np.reshape(x_teste, (x_teste.shape[0], x_teste.shape[1], 1))

# Aplicando as Previsões
Previsoes = Modelo.predict(x_teste)

# Calculando o erro quadrático médio
rsme = np.sqrt(np.mean(Previsoes - y_Teste) ** 2)
print('Erro Quadrático Médio:', rsme )

# Invertendo para escalas reais
Previsoes = Funcao_MinMAx.inverse_transform(Previsoes)
```

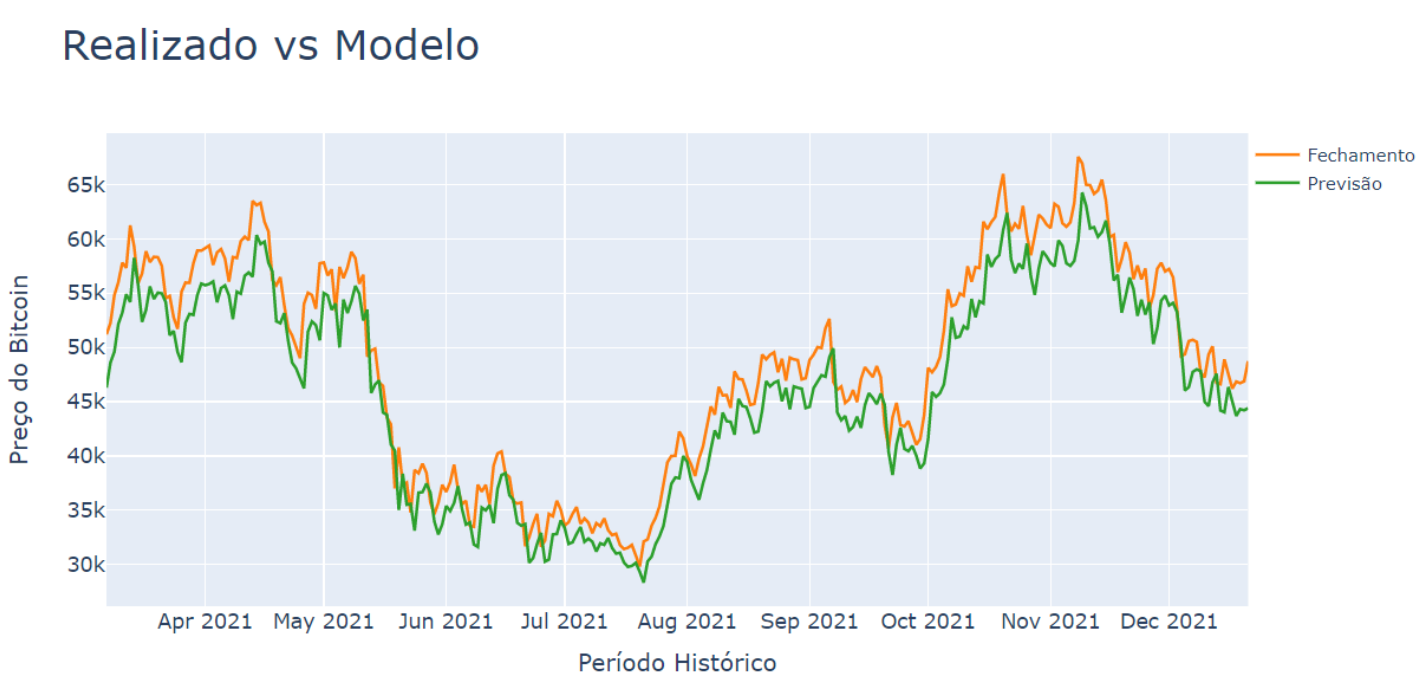
```
Erro Quadrático Médio: 0.04171338904735702
```

O erro quadrado médio ficou baixo porque os dados
estão escalonados, cuidado para não se enganar ;D

Hora de plotar as previsões

```
# Criando a base para verificar o real x modelo
Validação = Dados_Treino[Dados_Fechamento_Valores_Tamanho:]

# Atribuindo as previsões no DataSet
Validação['Previsões'] = Previsoes
```

```
# Criando um Gráfico Dinâmico
# No gráfico é possivel filtrar pela legenda a informação
# Utlizar zoons

# Definindo uma figura
Figura = Dash.Figure()

# Incluindo o Eixo no Gráfico - Fechamento
Figura.add_trace(Dash.Scatter(x = Validação.index, y = Validação.Close,
                    mode='lines',
                    name='Fechamento',
                    marker_color = '#FF7F0E',
                    ))

# Incluindo o Eixo no Gráfico - Previsão
Figura.add_trace(Dash.Scatter(x = Validação.index, y = Validação.Previsões,
                    mode='lines',
                    name='Previsão',
                    marker_color = '#2CA02C',
                    ))

# Modificando o Layout do Gráfico
Figura.update_layout(
    title='Realizado vs Modelo', # Titulo
    titlefont_size = 28, # Tamanho da Fonte

    # Parametros para mexer no eixo X
    xaxis = dict(
        title='Período Histórico', # Titulo do Eixo x
        titlefont_size=16, # Tamanho fonte do Titulo
        tickfont_size=14), # Tamanho da fonte do eixo

    # Tamanho do Grafico
    height = 500,

    # Parametros para mexer no eixo y
    yaxis=dict(
        title='Preço do Bitcoin', # Titulo do Eixo y
        titlefont_size=16, # Tamanho fonte do Titulo
        tickfont_size=14), # Tamanho da fonte do eixo

    # Parametros para mexer na legenda
    legend=dict(
        y=1, x=1, # Posição da Legenda
        bgcolor='rgba(255, 255, 255, 0)', # Cor de fundo
        bordercolor='rgba(255, 255, 255, 0)')) # Cor da Bornda

# Mostrando o Gráfico
Figura.show()
```



Uruuu !

Final

Esse guia foi elaborada para demostrar como prever preço do Bitcoin usando um modelo de Deep Learning

Link do código
https://drive.google.com/file/d/1MHmQtBARIILjTYDP_rObbLVA9tH1g2iy/view?usp=sharing

Odemir Depieri Jr

Data Intelligence Analyst Sr
Tech Lead
Specialization AI