IF702 - REDES NEURAIS

Grupo:

- Adriano Felipe Cabral Filho (afcf)
- Felipe Rodrigues de Souza (frs3)
- Lucas Vinícius José da Silva (lvjs)
- Pedro José Carneiro de Souza (pjcs)
- Vitor Sousa Silva (vss2)

Obs: por alguma razão os gráficos do optuna não apareceram no arquivo, mas são tecidos comentários sobre os mesmos na última seção do documento.

In [ ]:
```
!git clone https://github.com/brynmwangy/predicting-bitcoin-prices-using-LSTM/
!pip3 install tensorflow --quiet
!pip3 install numpy --quiet
!pip3 install scikit-learn --quiet
!pip3 install pandas --quiet
!pip3 install matplotlib --quiet
!pip3 install optuna --quiet
!pip3 install plotly --quiet
!pip3 install nbformat --quiet

import tensorflow as tf
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
scaler = StandardScaler()

dados = pd.read_csv('./predicting-bitcoin-prices-using-LSTM/btc.csv')
dados = dados[::-1]
dados.reset_index()

dados = scaler.fit_transform(dados['Close'].to_numpy().reshape(-1,1))

def window_data(data, window_size):
    X = []
    y = []

    i = 0
    while (i + window_size) <= len(data) - 1:
        X.append(data[i:i+window_size])
        y.append(data[i+window_size])

        i += 1
    assert len(X) ==  len(y)
    return X, y
X, y = window_data(dados, 7)

X_treino = np.array(X[:1018])
y_treino = np.array(y[:1018])
X_teste  = np.array(X[1018:])
y_teste  = np.array(y[1018:])

print("Tamanho do X treino {}".format(X_treino.shape))
print("Tamanho do y treino {}".format(y_treino.shape))
print("Tamanho do X teste  {}".format(X_teste.shape))
print("Tamanho do y teste  {}".format(y_teste.shape))

gpus = tf.config.list_physical_devices('GPU')
if gpus:
  try:
    for gpu in gpus:
      tf.config.experimental.set_memory_growth(gpu, True)
    logical_gpus = tf.config.list_logical_devices('GPU')
    print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPUs")
  except RuntimeError as e:
    print(e)
print(gpus)
```

```
fatal: destination path 'predicting-bitcoin-prices-using-LSTM' already exists and is not an empty directory.
WARNING:tensorflow:From /home/vitor/.local/lib/python3.8/site-packages/tensorflow/python/compat/v2_compat.py:101: disable_resource_variables (from tensorflow.python.
ops.variable_scope) is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
Tamanho do X treino (1018, 7, 1)
Tamanho do y treino (1018, 1)
Tamanho do X teste  (248, 7, 1)
Tamanho do y teste  (248, 1)
[]
```

In [ ]:
```
configs = {
  "batch_size"   : 7,
  "window_size"  : 7,
  "hidden_layer" : 32,
  "clip_margin"  : 20,
  "learning_rate": 0.001,
  "epochs"       : 60,
  "stddev"       : 0.05,
  "epochs_print" : 10
}
```

Parâmetros

Tamanho do batch/janela: 7 Hidden layer: 32 Margem de clip: 20 Taxa de aprendizado: 0.001

In [ ]:
```
# Pesos da via de entrada, inputs, objetivos, camada oculta e viés

inputs = tf.placeholder( tf.float32, [ configs["batch_size"], configs["window_size"], 1 ] )
targets = tf.placeholder( tf.float32, [ configs["batch_size"], 1 ] )

weights_input_gate = tf.Variable( tf.truncated_normal([ 1, configs["hidden_layer"] ], stddev=0.05) )
weights_input_hidden = tf.Variable( tf.truncated_normal( [ configs["hidden_layer"], configs["hidden_layer"] ], stddev=0.05) )
bias_input = tf.Variable( tf.zeros( [ configs["hidden_layer"] ] ) )
```

In [ ]:
```
# Pesos da via do esquecimento, camada oculta e viés

weights_forget_gate = tf.Variable( tf.truncated_normal( [1, configs["hidden_layer"]], stddev=0.05) )
weights_forget_hidden = tf.Variable( tf.truncated_normal( [ configs["hidden_layer"], configs["hidden_layer"] ], stddev = configs["stddev"] ) )
bias_forget = tf.Variable( tf.zeros( configs["hidden_layer"] ) )
```

In [ ]:
```
# Pesos da via de saída, camada oculta e viés

weights_output_gate = tf.Variable( tf.truncated_normal( [1, configs["hidden_layer"]], configs["hidden_layer"] ) )
```

```
weights_output_hidden = tf.Variable( tf.truncated_normal( [ configs["hidden_layer"], configs["hidden_layer"] ], stddev = configs["stddev"] ) )
bias_output = tf.Variable( tf.zeros( configs["hidden_layer"] ) )
```

In [ ]:
```
# Pesos da célula de memória, camada oculta e viés

weights_memory_cell = tf.Variable( tf.truncated_normal( [1, configs["hidden_layer"]], stddev=0.05 ) )
weights_memory_cell_hidden = tf.Variable( tf.truncated_normal( [configs["hidden_layer"], configs["hidden_layer"]], stddev = configs["stddev"] ) )
bias_memory_cell = tf.Variable( tf.zeros( [configs["hidden_layer"] ] ) )
```

In [ ]:
```
# Pesos da camada de saída (e viés)

weights_output = tf.Variable( tf.truncated_normal( [configs["hidden_layer"], 1], stddev = configs["stddev"] ) )
bias_output_layer = tf.Variable( tf.zeros([1]) )
```

In [ ]:
```
# Montando nossa célula LSTM

def LSTM_cell(input, output, state):
    # Via de entrada
    input_gate = tf.sigmoid(tf.matmul(input, weights_input_gate) + tf.matmul(output, weights_input_hidden) + bias_input)

    # Via do esquecimento
    forget_gate = tf.sigmoid(tf.matmul(input, weights_forget_gate) + tf.matmul(output, weights_forget_hidden) + bias_forget)

    # Célula de memória
    memory_cell = tf.tanh(tf.matmul(input, weights_memory_cell) + tf.matmul(output, weights_memory_cell_hidden) + bias_memory_cell)

    # Estado (combinação dos anteriores)
    state = state * forget_gate + input_gate * memory_cell

    # Via de saída
    output_gate = tf.sigmoid(tf.matmul(input, weights_output_gate) + tf.matmul(output, weights_output_hidden) + bias_output)

    # Saída (aplicação de função tangente à via de saída)
    output = output_gate * tf.tanh(state)
    return state, output
```

In [ ]:
```
# Percorrendo janela de dados e salvando resultados

outputs = []
for bs in range(configs["batch_size"]):
    batch_state  = np.zeros([1, configs["hidden_layer"]], dtype=np.float32)
    batch_output = np.zeros([1, configs["hidden_layer"]], dtype=np.float32)

    for ws in range(configs["window_size"]):
        batch_state, batch_output = LSTM_cell( tf.reshape( inputs[bs][ws], (-1, 1) ), batch_state, batch_output )

    outputs.append( tf.matmul( batch_output, weights_output ) + bias_output_layer )
```

In [ ]:
```
# Definindo perda

losses = []

for o in range(len(outputs)):
    losses.append( tf.losses.mean_squared_error( tf.reshape( targets[o], (-1, 1) ), outputs[o] ) )

loss = tf.reduce_mean(losses)

# Definindo o otimizador com o gradiente de clipagem (?)
gradients   = tf.gradients( loss, tf.trainable_variables() )
clipped, _  = tf.clip_by_global_norm( gradients, configs["clip_margin"] )
optimizer   = tf.train.AdamOptimizer( configs["learning_rate"] )
trained_opt = optimizer.apply_gradients( zip( gradients, tf.trainable_variables() ) )

session = tf.Session()
session.run( tf.global_variables_initializer() )

for e in range(configs["epochs"]):
    trained_scores = []
    ee = 0
    epoch_loss = []
    while(ee + configs["batch_size"]) <= len(X_treino):
        X_batch = X_treino[ee:ee+configs["batch_size"]]
        y_batch = y_treino[ee:ee+configs["batch_size"]]

        o, c, _ = session.run( [outputs, loss, trained_opt], feed_dict={inputs:X_batch, targets: y_batch} )

        epoch_loss.append(c)
        trained_scores.append(o)
        ee += configs["batch_size"]

    if(e % configs["epochs_print"]) == 0:
        print( 'Epoch {}/{}'.format(e, configs["epochs"]), ' LOSS: {}'.format( np.mean( epoch_loss ) ) )
```

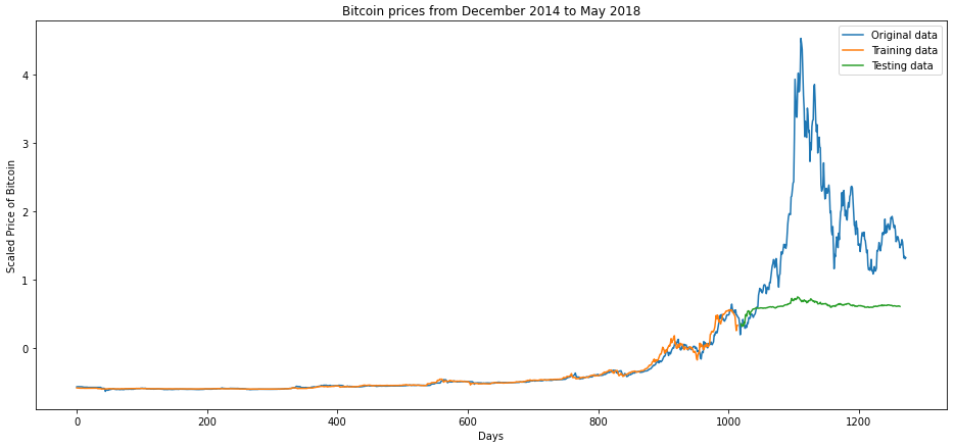
```
Epoch 0/60  LOSS: 0.2114369124174118
Epoch 10/60  LOSS: 0.012839341536164284
Epoch 20/60  LOSS: 0.012306488119065762
Epoch 30/60  LOSS: 0.009708047844469547
Epoch 40/60  LOSS: 0.0033579072915017605
Epoch 50/60  LOSS: 0.0004409407847560942
```

In [ ]:
```
sup = []
for ts in range(len(trained_scores)):
    for tsr in range(len( trained_scores[ts] )):
        sup.append( trained_scores[ts][tsr][0] )

tests = []
t = 0
while( t + configs["batch_size"] <= len(X_teste) ):
    o = session.run( [outputs], feed_dict = { inputs: X_teste[t:t+configs["batch_size"]] } )
    t += configs["batch_size"]
    tests.append(o)

novos_tests = []
for ts in range(len(tests)):
    for tsr in range(len( tests[ts][0] )):
        novos_tests.append( tests[ts][0][tsr] )

resultados_tests = []
for r in range(1264):
    if r >= 1019:
        resultados_tests.append(novos_tests[r-1019])
    else:
        resultados_tests.append(None)
```

In [ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.figure(figsize=(16, 7))
plt.title('Bitcoin prices from December 2014 to May 2018')
plt.xlabel('Days')
```

```
plt.ylabel('Scaled Price of Bitcoin')
plt.plot(dados, label='Original data')
plt.plot(sup, label='Training data')
plt.plot(resultados_tests, label='Testing data')
plt.legend()
plt.show()
```

```
/home/vitor/.local/lib/python3.8/site-packages/numpy/core/_asarray.py:136: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray
  return array(a, dtype, copy=False, order=order, subok=True)
```



Parâmetros do Estudo 1

Trials: 16

Tamanho do batch/janela: 7

Hidden layer: 32 à 128

Margem de clip: 4 à 64

Taxa de aprendizado: 0.05 à 0.15

In [ ]:

```
import optuna

configs = {
  "batch_size"   : 7,
  "window_size"  : 7,
  "hidden_layer" : 32,
  "clip_margin"  : 32,
  "learning_rate": 0.01,
  "epochs"       : 60,
  "stddev"       : 0.05,
  "epochs_print" : 10
}
bestCase = {
   "batch_size": 0,
   "outputs": [],
   "loss": 1000000
}
bestTrial = 0

def perda1(trial):

    configs["batch_size"] = configs["window_size"] = 7 #trial.suggest_int("batch_size", 3, 31)
    configs["hidden_layer"] = trial.suggest_int("hidden_layer", 32, 128)
    configs["clip_margin"] = trial.suggest_int("clip_margin", 4, 64)
    configs["learning_rate"] = trial.suggest_float("learning_rate", 0.01, 0.1)
    configs["stddev"] = trial.suggest_float("stddev", 0.05, 0.15)

    print("\nConfigs", configs, "\n")

    # Pesos da via de entrada, inputs, objetivos, camada oculta e viés
    inputs = tf.placeholder( tf.float32, [ configs["batch_size"], configs["window_size"], 1 ] )
    targets = tf.placeholder( tf.float32, [ configs["batch_size"], 1 ] )

    weights_input_gate = tf.Variable( tf.truncated_normal([ 1, configs["hidden_layer"] ], stddev=0.05) )
    weights_input_hidden = tf.Variable( tf.truncated_normal( [ configs["hidden_layer"], configs["hidden_layer"] ], stddev=0.05) )
    bias_input = tf.Variable( tf.zeros( [ configs["hidden_layer"] ] ) )

    # Pesos da via do esquecimento, camada oculta e viés
    weights_forget_gate = tf.Variable( tf.truncated_normal( [1, configs["hidden_layer"]], stddev=0.05) )
    weights_forget_hidden = tf.Variable( tf.truncated_normal( [ configs["hidden_layer"], configs["hidden_layer"] ], stddev = configs["stddev"] ) )
    bias_forget = tf.Variable( tf.zeros( configs["hidden_layer"] ) )

    # Pesos da via de saída, camada oculta e viés
    weights_output_gate = tf.Variable( tf.truncated_normal( [1, configs["hidden_layer"]], configs["hidden_layer"] ) )
    weights_output_hidden = tf.Variable( tf.truncated_normal( [ configs["hidden_layer"], configs["hidden_layer"] ], stddev = configs["stddev"] ) )
    bias_output = tf.Variable( tf.zeros( configs["hidden_layer"] ) )

    # Pesos da célula de memória, camada oculta e viés
    weights_memory_cell = tf.Variable( tf.truncated_normal( [1, configs["hidden_layer"]], stddev=0.05 ) )
    weights_memory_cell_hidden = tf.Variable( tf.truncated_normal( [configs["hidden_layer"], configs["hidden_layer"]], stddev = configs["stddev"] ) )
    bias_memory_cell = tf.Variable( tf.zeros( [configs["hidden_layer"] ] ) )

    # Pesos da camada de saída (e viés)
    weights_output = tf.Variable( tf.truncated_normal( [configs["hidden_layer"], 1], stddev = configs["stddev"] ) )
    bias_output_layer = tf.Variable( tf.zeros([1]) )


    # Montando nossa célula LSTM
    def LSTM_cell(input, output, state):
        # Via de entrada
        input_gate = tf.sigmoid(tf.matmul(input, weights_input_gate) + tf.matmul(output, weights_input_hidden) + bias_input)

        # Via do esquecimento
        forget_gate = tf.sigmoid(tf.matmul(input, weights_forget_gate) + tf.matmul(output, weights_forget_hidden) + bias_forget)

        # Célula de memória
        memory_cell = tf.tanh(tf.matmul(input, weights_memory_cell) + tf.matmul(output, weights_memory_cell_hidden) + bias_memory_cell)

        # Estado (combinação dos anteriores)
        state = state * forget_gate + input_gate * memory_cell

        # Via de saída
        output_gate = tf.sigmoid(tf.matmul(input, weights_output_gate) + tf.matmul(output, weights_output_hidden) + bias_output)

        # Saída (aplicação de função tangente à via de saída)
        output = output_gate * tf.tanh(state)
        return state, output
```

```
    outputs = []
    for bs in range(configs["batch_size"]):
        batch_state  = np.zeros([1, configs["hidden_layer"]], dtype=np.float32)
        batch_output = np.zeros([1, configs["hidden_layer"]], dtype=np.float32)

        for ws in range(configs["window_size"]):
            batch_state, batch_output = LSTM_cell( tf.reshape( inputs[bs][ws], (-1, 1) ), batch_state, batch_output )

        outputs.append( tf.matmul( batch_output, weights_output ) + bias_output_layer )

    # Definindo perda

    losses = []

    for o in range(len(outputs)):
        losses.append( tf.losses.mean_squared_error( tf.reshape( targets[o], (-1, 1) ), outputs[o] ) )

    loss = tf.reduce_mean(losses)

    # Definindo o otimizador com o gradiente de clipagem (?)
    gradients   = tf.gradients( loss, tf.trainable_variables() )
    clipped, _  = tf.clip_by_global_norm( gradients, configs["clip_margin"] )
    optimizer   = tf.train.AdamOptimizer( configs["learning_rate"] )
    trained_opt = optimizer.apply_gradients( zip( gradients, tf.trainable_variables() ) )

    session = tf.Session()
    session.run( tf.global_variables_initializer() )

    best_epochs = [10.0]

    for e in range(configs["epochs"]):
        trained_scores = []
        ee = 0
        epoch_loss = []
        while(ee + configs["batch_size"]) <= len(X_treino):
            X_batch = X_treino[ee:ee+configs["batch_size"]]
            y_batch = y_treino[ee:ee+configs["batch_size"]]

            o, c, _ = session.run( [outputs, loss, trained_opt], feed_dict={inputs:X_batch, targets: y_batch} )

            epoch_loss.append(c)
            trained_scores.append(o)
            ee += configs["batch_size"]

        if(e % configs["epochs_print"]) == 0:
            if e == 0:
                print( 'Epoch  {}/{}'.format(e, configs["epochs"]), ' LOSS: {}'.format( np.mean( epoch_loss ) ) )
            else:
                print( 'Epoch {}/{}'.format(e, configs["epochs"]), ' LOSS: {}'.format( np.mean( epoch_loss ) ) )

    loss = sum(epoch_loss)/len(epoch_loss)
    if loss < bestCase["loss"]:
        bestCase["loss"] = loss
        global sup
        sup = []
        for ts in range(len(trained_scores)):
            for tsr in range(len( trained_scores[ts] )):
                sup.append( trained_scores[ts][tsr][0] )

        tests = []
        t = 0
        while( t + configs["batch_size"] <= len(X_teste) ):
            o = session.run( [outputs], feed_dict = { inputs: X_teste[t:t+configs["batch_size"]] } )
            t += configs["batch_size"]
            tests.append(o)

        novos_tests = []
        for ts in range(len(tests)):
            for tsr in range(len( tests[ts][0] )):
                novos_tests.append( tests[ts][0][tsr] )

        global resultados_tests
        resultados_tests = []
        for r in range(1264):
            if r >= 1019:
                resultados_tests.append(novos_tests[r-1019])
            else:
                resultados_tests.append(None)

    # Retornando a média de LOSS
    return loss

study_1 = optuna.create_study(direction='minimize')
study_1.optimize(perda1, n_trials=16)
```

```
[I 2021-11-16 02:34:43,781] A new study created in memory with name: no-name-59a039c6-5c2c-4259-b3a1-4db85caf2e61
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 71, 'clip_margin': 14, 'learning_rate': 0.0633319318479532, 'epochs': 60, 'stddev': 0.12933382482918607,
'epochs_print': 10}

Epoch  0/60  LOSS: 0.24634768068790436
Epoch 10/60  LOSS: 0.42830589413642883
Epoch 20/60  LOSS: 0.006934761069715023
Epoch 30/60  LOSS: 0.014057870022952557
Epoch 40/60  LOSS: 0.006995856296271086
Epoch 50/60  LOSS: 0.007985973730683327
[I 2021-11-16 02:35:31,800] Trial 0 finished with value: 0.015463595697359217 and parameters: {'hidden_layer': 71, 'clip_margin': 14, 'learning_rate': 0.063331931847
9532, 'stddev': 0.12933382482918607}. Best is trial 0 with value: 0.015463595697359217.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 35, 'clip_margin': 48, 'learning_rate': 0.040600832440222806, 'epochs': 60, 'stddev': 0.0618636746025036
4, 'epochs_print': 10}

Epoch  0/60  LOSS: 0.07793401926755905
Epoch 10/60  LOSS: 0.06869269162416458
Epoch 20/60  LOSS: 0.004831113386899233
Epoch 30/60  LOSS: 0.025925491005182266
Epoch 40/60  LOSS: 0.011901136487722397
Epoch 50/60  LOSS: 0.011899720877408981
[I 2021-11-16 02:36:11,226] Trial 1 finished with value: 0.10919421006841856 and parameters: {'hidden_layer': 35, 'clip_margin': 48, 'learning_rate': 0.0406008324402
22806, 'stddev': 0.06186367460250364}. Best is trial 0 with value: 0.015463595697359217.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 68, 'clip_margin': 33, 'learning_rate': 0.027684182198906518, 'epochs': 60, 'stddev': 0.1016758124772670
6, 'epochs_print': 10}

Epoch  0/60  LOSS: 0.04525112360715866
Epoch 10/60  LOSS: 0.009183439426124096
Epoch 20/60  LOSS: 0.08996368199586868
Epoch 30/60  LOSS: 0.006026847753673792
Epoch 40/60  LOSS: 0.0035885737743228674
Epoch 50/60  LOSS: 0.01802368275821209
[I 2021-11-16 02:37:05,005] Trial 2 finished with value: 0.01911936511345755 and parameters: {'hidden_layer': 68, 'clip_margin': 33, 'learning_rate': 0.0276841821989
06518, 'stddev': 0.10167581247726706}. Best is trial 0 with value: 0.015463595697359217.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 81, 'clip_margin': 6, 'learning_rate': 0.034666614115877646, 'epochs': 60, 'stddev': 0.10058708121365512,
'epochs_print': 10}

Epoch  0/60  LOSS: 0.09068140387535095
Epoch 10/60  LOSS: 0.007022745907306671
```

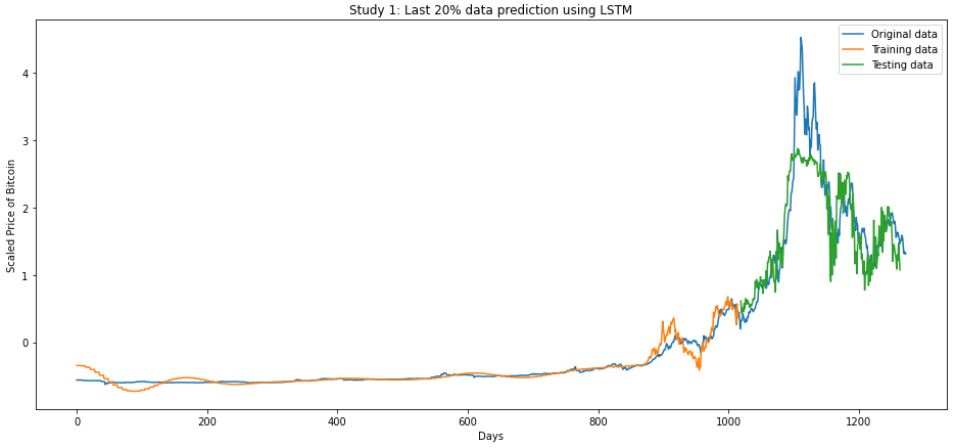
```
Epoch 20/60  LOSS: 0.005236579105257988
Epoch 30/60  LOSS: 0.01195207517594099
Epoch 40/60  LOSS: 0.01938570663332939
Epoch 50/60  LOSS: 0.1386166512966156
[I 2021-11-16 02:37:59,172] Trial 3 finished with value: 0.023053284938131236 and parameters: {'hidden_layer': 81, 'clip_margin': 6, 'learning_rate': 0.0346666141158
77646, 'stddev': 0.10058708121365512}. Best is trial 0 with value: 0.015463595697359217.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 124, 'clip_margin': 6, 'learning_rate': 0.036836180955451665, 'epochs': 60, 'stddev': 0.07123679781276535, 'epochs_print': 10}

Epoch  0/60  LOSS: 1.0196027755737305
Epoch 10/60  LOSS: 0.09389112144708633
Epoch 20/60  LOSS: 0.01529747061431408
Epoch 30/60  LOSS: 0.015114999376237392
Epoch 40/60  LOSS: 0.013435245491564274
Epoch 50/60  LOSS: 0.011418633162975311
[I 2021-11-16 02:38:55,237] Trial 4 finished with value: 0.0057614614846768 52 and parameters: {'hidden_layer': 124, 'clip_margin': 6, 'learning_rate': 0.036836180955
451665, 'stddev': 0.07123679781276535}. Best is trial 4 with value: 0.005761461484676852.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 50, 'clip_margin': 32, 'learning_rate': 0.05657534665892468, 'epochs': 60, 'stddev': 0.10152453377377922, 'epochs_print': 10}

Epoch  0/60  LOSS: 0.027741944417357445
Epoch 10/60  LOSS: 0.012591270729899406
Epoch 20/60  LOSS: 0.00860643107444048
Epoch 30/60  LOSS: 0.004774277564138174
Epoch 40/60  LOSS: 0.02148338221013546
Epoch 50/60  LOSS: 0.0023393852170556784
[I 2021-11-16 02:39:46,447] Trial 5 finished with value: 0.005365704418303768 and parameters: {'hidden_layer': 50, 'clip_margin': 32, 'learning_rate': 0.056575346658
92468, 'stddev': 0.10152453377377922}. Best is trial 5 with value: 0.005365704418303768.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 125, 'clip_margin': 50, 'learning_rate': 0.09895332669439717, 'epochs': 60, 'stddev': 0.09590280597657397, 'epochs_print': 10}

Epoch  0/60  LOSS: 1.0036219358444214
Epoch 10/60  LOSS: 0.01969262771308422
Epoch 20/60  LOSS: 0.00839470699429512
Epoch 30/60  LOSS: 0.018914133310317993
Epoch 40/60  LOSS: 0.04784975200891495
Epoch 50/60  LOSS: 0.035986728966236115
[I 2021-11-16 02:40:41,718] Trial 6 finished with value: 0.03267629125666488 and parameters: {'hidden_layer': 125, 'clip_margin': 50, 'learning_rate': 0.098953326694
39717, 'stddev': 0.09590280597657397}. Best is trial 5 with value: 0.005365704418303768.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 39, 'clip_margin': 45, 'learning_rate': 0.07764038042649608, 'epochs': 60, 'stddev': 0.07979700376619127, 'epochs_print': 10}

Epoch  0/60  LOSS: 0.2186170518398285
Epoch 10/60  LOSS: 0.02035936340689659
Epoch 20/60  LOSS: 0.020047931000590324
Epoch 30/60  LOSS: 0.04344770312309265
Epoch 40/60  LOSS: 0.019691869616508484
Epoch 50/60  LOSS: 0.02671739086508751
[I 2021-11-16 02:41:30,677] Trial 7 finished with value: 0.023230003413726958 and parameters: {'hidden_layer': 39, 'clip_margin': 45, 'learning_rate': 0.077640380426
49608, 'stddev': 0.07979700376619127}. Best is trial 5 with value: 0.005365704418303768.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 71, 'clip_margin': 56, 'learning_rate': 0.04094016257400482, 'epochs': 60, 'stddev': 0.05998509481420051
5, 'epochs_print': 10}

Epoch  0/60  LOSS: 0.060604508966207504
Epoch 10/60  LOSS: 0.013889957219362259
Epoch 20/60  LOSS: 0.010826966725289822
Epoch 30/60  LOSS: 0.009556318633258343
Epoch 40/60  LOSS: 0.00945307593792677
Epoch 50/60  LOSS: 0.0277418065816164
[I 2021-11-16 02:42:20,352] Trial 8 finished with value: 0.017076246655018435 and parameters: {'hidden_layer': 71, 'clip_margin': 56, 'learning_rate': 0.040940162574
00482, 'stddev': 0.059985094814200515}. Best is trial 5 with value: 0.005365704418303768.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 53, 'clip_margin': 63, 'learning_rate': 0.08822946005987356, 'epochs': 60, 'stddev': 0.11519463172441914, 'epochs_print': 10}

Epoch  0/60  LOSS: 0.028217285871505737
Epoch 10/60  LOSS: 0.19234664738178253
Epoch 20/60  LOSS: 0.04247713461518288
Epoch 30/60  LOSS: 0.013777139596641064
Epoch 40/60  LOSS: 0.0024612327106297016
Epoch 50/60  LOSS: 0.06594859808683395
[I 2021-11-16 02:43:01,555] Trial 9 finished with value: 0.008818625053248949 and parameters: {'hidden_layer': 53, 'clip_margin': 63, 'learning_rate': 0.088229460059
87356, 'stddev': 0.11519463172441914}. Best is trial 5 with value: 0.005365704418303768.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 97, 'clip_margin': 27, 'learning_rate': 0.011666642412384555, 'epochs': 60, 'stddev': 0.1386461365656535, 'epochs_print': 10}

Epoch  0/60  LOSS: 0.22199332714080081
Epoch 10/60  LOSS: 0.007945024408400059
Epoch 20/60  LOSS: 0.003504543099552393
Epoch 30/60  LOSS: 0.006697486154735088
Epoch 40/60  LOSS: 0.003918299917131662
Epoch 50/60  LOSS: 0.008770763874053955
[I 2021-11-16 02:43:57,264] Trial 10 finished with value: 0.0142870552388495 56 and parameters: {'hidden_layer': 97, 'clip_margin': 27, 'learning_rate': 0.01166664241
2384555, 'stddev': 0.1386461365656535}. Best is trial 5 with value: 0.005365704418303768.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 125, 'clip_margin': 20, 'learning_rate': 0.05977132500452518, 'epochs': 60, 'stddev': 0.07806664466493624, 'epochs_print': 10}

Epoch  0/60  LOSS: 0.37427809834480286
Epoch 10/60  LOSS: 0.06734833121299744
Epoch 20/60  LOSS: 0.005465580150485039
Epoch 30/60  LOSS: 0.006255015265196562
Epoch 40/60  LOSS: 0.005888625048100948
Epoch 50/60  LOSS: 0.03961179405450821
[I 2021-11-16 02:44:50,831] Trial 11 finished with value: 0.022002875140047 95 and parameters: {'hidden_layer': 125, 'clip_margin': 20, 'learning_rate': 0.059771325004
52518, 'stddev': 0.07806664466493624}. Best is trial 5 with value: 0.005365704418303768.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 105, 'clip_margin': 4, 'learning_rate': 0.014604565345756368, 'epochs': 60, 'stddev': 0.07968879257407203, 'epochs_print': 10}

Epoch  0/60  LOSS: 0.13661111934185028
Epoch 10/60  LOSS: 0.0026706361677497625
Epoch 20/60  LOSS: 0.00672927126288414
Epoch 30/60  LOSS: 0.0063286814834864784
Epoch 40/60  LOSS: 0.06800814718008041
Epoch 50/60  LOSS: 0.14811848104000092
[I 2021-11-16 02:45:38,821] Trial 12 finished with value: 0.006055218872586857 and parameters: {'hidden_layer': 105, 'clip_margin': 4, 'learning_rate': 0.014604565345
5756368, 'stddev': 0.07968879257407203}. Best is trial 5 with value: 0.005365704418303768.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 48, 'clip_margin': 38, 'learning_rate': 0.051191575402560564, 'epochs': 60, 'stddev': 0.11740063446692542, 'epochs_print': 10}

Epoch  0/60  LOSS: 0.01984240487217903
Epoch 10/60  LOSS: 0.018642157316207886
Epoch 20/60  LOSS: 0.010722176171839237
Epoch 30/60  LOSS: 0.017138483002781868
Epoch 40/60  LOSS: 0.029165877029299736
Epoch 50/60  LOSS: 0.014209786430001259
[I 2021-11-16 02:46:07,739] Trial 13 finished with value: 0.011112098819351025 and parameters: {'hidden_layer': 48, 'clip_margin': 38, 'learning_rate': 0.051195175402
2560564, 'stddev': 0.11740063446692542}. Best is trial 5 with value: 0.005365704418303768.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 105, 'clip_margin': 15, 'learning_rate': 0.07235512717227 48, 'epochs': 60, 'stddev': 0.05180155951874785, 'epochs_print': 10}

Epoch  0/60  LOSS: 0.14075283706188202
Epoch 10/60  LOSS: 0.01780426874756813
Epoch 20/60  LOSS: 0.008228941820561886
Epoch 30/60  LOSS: 0.0036683904472738504
```

```
Epoch 40/60  LOSS: 0.0601534731686115 26
Epoch 50/60  LOSS: 0.0341523513197898 86
[I 2021-11-16 02:47:03,776] Trial 14 finished with value: 0.059314812659588 29 and parameters: {'hidden_layer': 105, 'clip_margin': 15, 'learning_rate': 0.0723551271722748, 'stddev': 0.05180155951874785}. Best is trial 5 with value: 0.005365704418303768.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 92, 'clip_margin': 26, 'learning_rate': 0.050000640106874, 'epochs': 60, 'stddev': 0.08939693540768119, 'epochs_print': 10}

Epoch  0/60  LOSS: 0.4528822898864746
Epoch 10/60  LOSS: 0.03881917521357536
Epoch 20/60  LOSS: 0.01317871268838644
Epoch 30/60  LOSS: 0.038725025951862335
Epoch 40/60  LOSS: 0.035278867930173874
Epoch 50/60  LOSS: 0.0555773563683033
[I 2021-11-16 02:47:51,336] Trial 15 finished with value: 0.057802968130585 97 and parameters: {'hidden_layer': 92, 'clip_margin': 26, 'learning_rate': 0.050000640106874, 'stddev': 0.08939693540768119}. Best is trial 5 with value: 0.005365704418303768.
```

In [ ]:

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.figure(figsize=(16, 7))
plt.title('Study 1: Last 20% data prediction using LSTM')
plt.xlabel('Days')
plt.ylabel('Scaled Price of Bitcoin')
plt.plot(dados, label='Original data')
plt.plot(sup, label="Training data")
plt.plot(resultados_tests, label='Testing data')
plt.legend()
plt.show()
```

```
/home/vitor/.local/lib/python3.8/site-packages/numpy/core/_asarray.py:136: VisibleDeprecationWarning:

Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray
```



In [ ]:

```
optuna.visualization.plot_optimization_history(study_1)
```

In [ ]:

```
optuna.visualization.plot_slice(study_1)
```

In [ ]:

```
optuna.visualization.plot_param_importances(study_1)
```

Parâmetros do Estudo 2

Trials: 16

Tamanho do batch/janela: 7

Hidden layer: 64 à 256

Margem de clip: 64 à 128

Taxa de aprendizado: 0.1 à 0.5

In [ ]:

```
configs = {
  "batch_size"   : 7,
  "window_size"  : 7,
  "hidden_layer" : 32,
  "clip_margin"  : 32,
  "learning_rate": 0.01,
  "epochs"       : 60,
  "stddev"       : 0.05,
  "epochs_print" : 10
}
bestCase = {
   "batch_size": 0,
   "outputs": [],
   "loss": 1000000
}
bestTrial = 0

def perda2(trial):

    configs["batch_size"] = configs["window_size"] = 7 #trial.suggest_int("batch_size", 3, 31)
    configs["hidden_layer"] = trial.suggest_int("hidden_layer", 128, 256)
    configs["clip_margin"] = trial.suggest_int("clip_margin", 64, 128)
    configs["learning_rate"] = trial.suggest_float("learning_rate", 0.1, 0.5)
    configs["stddev"] = trial.suggest_float("stddev", 0.05, 0.15)

    print("\nConfigs", configs, "\n")

    # Pesos da via de entrada, inputs, objetivos, camada oculta e viés
    inputs = tf.placeholder( tf.float32, [ configs["batch_size"], configs["window_size"], 1 ] )
    targets = tf.placeholder( tf.float32, [ configs["batch_size"], 1 ] )

    weights_input_gate = tf.Variable( tf.truncated_normal([ 1, configs["hidden_layer"] ], stddev=0.05) )
    weights_input_hidden = tf.Variable( tf.truncated_normal( [ configs["hidden_layer"], configs["hidden_layer"] ], stddev=0.05) )
    bias_input = tf.Variable( tf.zeros( [ configs["hidden_layer"] ] ) )

    # Pesos da via do esquecimento, camada oculta e viés
    weights_forget_gate = tf.Variable( tf.truncated_normal( [1, configs["hidden_layer"]], stddev=0.05) )
    weights_forget_hidden = tf.Variable( tf.truncated_normal( [ configs["hidden_layer"], configs["hidden_layer"] ], stddev = configs["stddev"] ) )
    bias_forget = tf.Variable( tf.zeros( configs["hidden_layer"] ) )
```

```
    # Pesos da via de saída, camada oculta e viés
    weights_output_gate = tf.Variable( tf.truncated_normal( [1, configs["hidden_layer"]], configs["hidden_layer"] ) )
    weights_output_hidden = tf.Variable( tf.truncated_normal( [ configs["hidden_layer"], configs["hidden_layer"] ], stddev = configs["stddev"] ) )
    bias_output = tf.Variable( tf.zeros( configs["hidden_layer"] ) )

    # Pesos da célula de memória, camada oculta e viés
    weights_memory_cell = tf.Variable( tf.truncated_normal( [1, configs["hidden_layer"]], stddev=0.05 ) )
    weights_memory_cell_hidden = tf.Variable( tf.truncated_normal( [configs["hidden_layer"], configs["hidden_layer"]], stddev = configs["stddev"] ) )
    bias_memory_cell = tf.Variable( tf.zeros( [configs["hidden_layer"] ] ) )

    # Pesos da camada de saída (e viés)
    weights_output = tf.Variable( tf.truncated_normal( [configs["hidden_layer"], 1], stddev = configs["stddev"] ) )
    bias_output_layer = tf.Variable( tf.zeros([1]) )


    # Montando nossa célula LSTM
    def LSTM_cell(input, output, state):
        # Via de entrada
        input_gate = tf.sigmoid(tf.matmul(input, weights_input_gate) + tf.matmul(output, weights_input_hidden) + bias_input)

        # Via do esquecimento
        forget_gate = tf.sigmoid(tf.matmul(input, weights_forget_gate) + tf.matmul(output, weights_forget_hidden) + bias_forget)

        # Célula de memória
        memory_cell = tf.tanh(tf.matmul(input, weights_memory_cell) + tf.matmul(output, weights_memory_cell_hidden) + bias_memory_cell)

        # Estado (combinação dos anteriores)
        state = state * forget_gate + input_gate * memory_cell

        # Via de saída
        output_gate = tf.sigmoid(tf.matmul(input, weights_output_gate) + tf.matmul(output, weights_output_hidden) + bias_output)

        # Saída (aplicação de função tangente à via de saída)
        output = output_gate * tf.tanh(state)
        return state, output


    outputs = []
    for bs in range(configs["batch_size"]):
        batch_state  = np.zeros([1, configs["hidden_layer"]], dtype=np.float32)
        batch_output = np.zeros([1, configs["hidden_layer"]], dtype=np.float32)

        for ws in range(configs["window_size"]):
            batch_state, batch_output = LSTM_cell( tf.reshape( inputs[bs][ws], (-1, 1) ), batch_state, batch_output )

        outputs.append( tf.matmul( batch_output, weights_output ) + bias_output_layer )

    # Definindo perda

    losses = []

    for o in range(len(outputs)):
        losses.append( tf.losses.mean_squared_error( tf.reshape( targets[o], (-1, 1) ), outputs[o] ) )

    loss = tf.reduce_mean(losses)

    # Definindo o otimizador com o gradiente de clipagem (?)
    gradients   = tf.gradients( loss, tf.trainable_variables() )
    clipped, _  = tf.clip_by_global_norm( gradients, configs["clip_margin"] )
    optimizer   = tf.train.AdamOptimizer( configs["learning_rate"] )
    trained_opt = optimizer.apply_gradients( zip( gradients, tf.trainable_variables() ) )

    session = tf.Session()
    session.run( tf.global_variables_initializer() )

    best_epochs = [10.0]

    for e in range(configs["epochs"]):
        trained_scores = []
        ee = 0
        epoch_loss = []
        while(ee + configs["batch_size"]) <= len(X_treino):
            X_batch = X_treino[ee:ee+configs["batch_size"]]
            y_batch = y_treino[ee:ee+configs["batch_size"]]

            o, c, _ = session.run( [outputs, loss, trained_opt], feed_dict={inputs:X_batch, targets: y_batch} )

            epoch_loss.append(c)
            trained_scores.append(o)
            ee += configs["batch_size"]

        if(e % configs["epochs_print"]) == 0:
            if e == 0:
                print( 'Epoch  {}/{}'.format(e, configs["epochs"]), ' LOSS: {}'.format( np.mean( epoch_loss ) ) )
            else:
                print( 'Epoch {}/{}'.format(e, configs["epochs"]), ' LOSS: {}'.format( np.mean( epoch_loss ) ) )

    loss = sum(epoch_loss)/len(epoch_loss)
    if loss < bestCase["loss"]:
        bestCase["loss"] = loss
        global sup
        sup = []
        for ts in range(len(trained_scores)):
            for tsr in range(len( trained_scores[ts] )):
                sup.append( trained_scores[ts][tsr][0] )

        tests = []
        t = 0
        while( t + configs["batch_size"] <= len(X_teste) ):
            o = session.run( [outputs], feed_dict = { inputs: X_teste[t:t+configs["batch_size"]] } )
            t += configs["batch_size"]
            tests.append(o)

        novos_tests = []
        for ts in range(len(tests)):
            for tsr in range(len( tests[ts][0] )):
                novos_tests.append( tests[ts][0][tsr] )

        global resultados_tests
        resultados_tests = []
        for r in range(1264):
            if r >= 1019:
                resultados_tests.append(novos_tests[r-1019])
            else:
                resultados_tests.append(None)

    # Retornando a média de LOSS
    return loss

study_2 = optuna.create_study(direction='minimize')
study_2.optimize(perda2, n_trials=16)
```

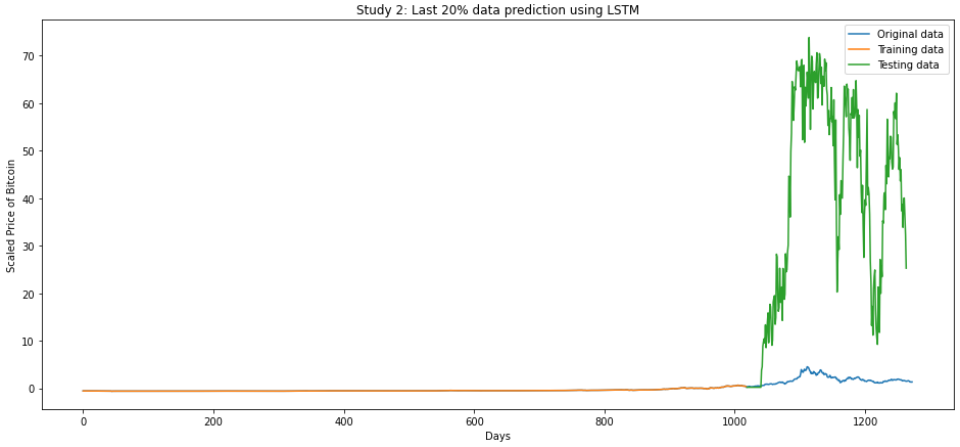
```
[I 2021-11-16 01:39:29,770] A new study created in memory with name: no-name-9c8c7dca-7a8c-495c-a2a7-29db1d73ff24
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 197, 'clip_margin': 87, 'learning_rate': 0.28731147414473723, 'epochs': 60, 'stddev': 0.1089252104777970
7, 'epochs_print': 10}

Epoch  0/60  LOSS: 23.786651611328125
```

```
Epoch 10/60  LOSS: 0.24432078003883362
Epoch 20/60  LOSS: 0.41070613265037537
Epoch 30/60  LOSS: 0.3337078094482422
Epoch 40/60  LOSS: 0.3942481279373169
Epoch 50/60  LOSS: 0.39073801040649414
[I 2021-11-16 01:40:37,928] Trial 0 finished with value: 0.4070438464828103 and parameters: {'hidden_layer': 197, 'clip_margin': 87, 'learning_rate': 0.28731147414473723, 'stddev': 0.10892521047779707}. Best is trial 0 with value: 0.4070438464828103.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 170, 'clip_margin': 114, 'learning_rate': 0.4523912711864134, 'epochs': 60, 'stddev': 0.08878662748097808, 'epochs_print': 10}

Epoch  0/60  LOSS: 49.38376998901367
Epoch 10/60  LOSS: 1.1461553573608398
Epoch 20/60  LOSS: 0.7815670371055603
Epoch 30/60  LOSS: 1.511423945426941
Epoch 40/60  LOSS: 1.233652949333191
Epoch 50/60  LOSS: 0.8182525038719177
[I 2021-11-16 01:41:30,868] Trial 1 finished with value: 0.8176512446790278 and parameters: {'hidden_layer': 170, 'clip_margin': 114, 'learning_rate': 0.4523912711864134, 'stddev': 0.08878662748097808}. Best is trial 0 with value: 0.4070438464828103.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 256, 'clip_margin': 64, 'learning_rate': 0.3996040075301285, 'epochs': 60, 'stddev': 0.14158656955436272, 'epochs_print': 10}

Epoch  0/60  LOSS: 20.84975242614746
Epoch 10/60  LOSS: 0.11732295900583267
Epoch 20/60  LOSS: 0.721122145652771
Epoch 30/60  LOSS: 0.13634297251701355
Epoch 40/60  LOSS: 0.5880559086799622
Epoch 50/60  LOSS: 0.5027512907981873
[I 2021-11-16 01:43:23,077] Trial 2 finished with value: 3.2326829405936475 and parameters: {'hidden_layer': 256, 'clip_margin': 64, 'learning_rate': 0.3996040075301285, 'stddev': 0.14158656955436272}. Best is trial 0 with value: 0.4070438464828103.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 188, 'clip_margin': 101, 'learning_rate': 0.3315468413796342, 'epochs': 60, 'stddev': 0.0858313948787198, 'epochs_print': 10}

Epoch  0/60  LOSS: 27.547929763793945
Epoch 10/60  LOSS: 0.059783220291137695
Epoch 20/60  LOSS: 0.2501884996891022
Epoch 30/60  LOSS: 0.3942742645740509
Epoch 40/60  LOSS: 0.39415571093559265
Epoch 50/60  LOSS: 0.394089937210083
[I 2021-11-16 01:44:27,054] Trial 3 finished with value: 0.394061185141298 and parameters: {'hidden_layer': 188, 'clip_margin': 101, 'learning_rate': 0.3315468413796342, 'stddev': 0.0858313948787198}. Best is trial 3 with value: 0.394061185141298.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 222, 'clip_margin': 128, 'learning_rate': 0.34958082828225423, 'epochs': 60, 'stddev': 0.08335425296033779, 'epochs_print': 10}

Epoch  0/60  LOSS: 36.842529296875
Epoch 10/60  LOSS: 0.3151966631412506
Epoch 20/60  LOSS: 0.5250840187072754
Epoch 30/60  LOSS: 0.4146926999092102
Epoch 40/60  LOSS: 0.611014187335968
Epoch 50/60  LOSS: 0.6533364057540894
[I 2021-11-16 01:45:59,566] Trial 4 finished with value: 0.7215765356500422 and parameters: {'hidden_layer': 222, 'clip_margin': 128, 'learning_rate': 0.34958082828225423, 'stddev': 0.08335425296033779}. Best is trial 3 with value: 0.394061185141298.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 150, 'clip_margin': 89, 'learning_rate': 0.4352354648278759, 'epochs': 60, 'stddev': 0.05471615975105827, 'epochs_print': 10}

Epoch  0/60  LOSS: 34.289833068847656
Epoch 10/60  LOSS: 0.05729949474334717
Epoch 20/60  LOSS: 0.4459030032157898
Epoch 30/60  LOSS: 0.5124076008796692
Epoch 40/60  LOSS: 0.5124531984329224
Epoch 50/60  LOSS: 0.5288580060005188
[I 2021-11-16 01:46:47,212] Trial 5 finished with value: 0.5112410216228329 and parameters: {'hidden_layer': 150, 'clip_margin': 89, 'learning_rate': 0.4352354648278759, 'stddev': 0.05471615975105827}. Best is trial 3 with value: 0.394061185141298.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 256, 'clip_margin': 116, 'learning_rate': 0.1403621396559157, 'epochs': 60, 'stddev': 0.13264440905635375}. Best is trial 6 with value: 0.037891683366636494.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 145, 'clip_margin': 70, 'learning_rate': 0.17547804684358953, 'epochs': 60, 'stddev': 0.09628385490397316, 'epochs_print': 10}

Epoch  0/60  LOSS: 8.117172241210938
Epoch 10/60  LOSS: 1.3416537046432495
Epoch 20/60  LOSS: 0.021451493725180626
Epoch 30/60  LOSS: 0.11527025699615479
Epoch 40/60  LOSS: 0.5566946864128113
Epoch 50/60  LOSS: 0.046272534877061844
[I 2021-11-16 01:48:27,664] Trial 6 finished with value: 0.037891683366636494 and parameters: {'hidden_layer': 256, 'clip_margin': 116, 'learning_rate': 0.1403621396559157, 'stddev': 0.13264440905635375}. Best is trial 6 with value: 0.037891683366636494.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 145, 'clip_margin': 70, 'learning_rate': 0.17547804684358953, 'epochs': 60, 'stddev': 0.09628385490397316, 'epochs_print': 10}

Epoch  0/60  LOSS: 5.875904083251953
Epoch 10/60  LOSS: 0.0733150988817215
Epoch 20/60  LOSS: 0.09903957694768906
Epoch 30/60  LOSS: 0.12582091987133026
Epoch 40/60  LOSS: 0.12872925400733948
Epoch 50/60  LOSS: 0.1298445761203766
[I 2021-11-16 01:49:20,112] Trial 7 finished with value: 0.13058684349500063 and parameters: {'hidden_layer': 145, 'clip_margin': 70, 'learning_rate': 0.17547804684358953, 'stddev': 0.09628385490397316}. Best is trial 6 with value: 0.037891683366636494.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 148, 'clip_margin': 79, 'learning_rate': 0.4852534232031165, 'epochs': 60, 'stddev': 0.05696840617526562, 'epochs_print': 10}

Epoch  0/60  LOSS: 71.16302490234375
Epoch 10/60  LOSS: 0.05358676239848137
Epoch 20/60  LOSS: 0.1727895587682724
Epoch 30/60  LOSS: 0.8098796010017395
Epoch 40/60  LOSS: 0.7553268074989319
Epoch 50/60  LOSS: 0.5882998704910278
[I 2021-11-16 01:50:03,662] Trial 8 finished with value: 0.6990398244007745 and parameters: {'hidden_layer': 148, 'clip_margin': 79, 'learning_rate': 0.4852534232031165, 'stddev': 0.05696840617526562}. Best is trial 6 with value: 0.037891683366636494.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 137, 'clip_margin': 94, 'learning_rate': 0.17033627126963374, 'epochs': 60, 'stddev': 0.14136041009600375, 'epochs_print': 10}

Epoch  0/60  LOSS: 1.8708647489547773
Epoch 10/60  LOSS: 0.7280149459838867
Epoch 20/60  LOSS: 0.04417095333337784
Epoch 30/60  LOSS: 0.040284786373376846
Epoch 40/60  LOSS: 0.04415031149983406
Epoch 50/60  LOSS: 0.04367367923259735
[I 2021-11-16 01:50:48,375] Trial 9 finished with value: 0.044185904561928856 and parameters: {'hidden_layer': 137, 'clip_margin': 94, 'learning_rate': 0.17033627126963374, 'stddev': 0.14136041009600375}. Best is trial 6 with value: 0.037891683366636494.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 255, 'clip_margin': 128, 'learning_rate': 0.1037578631718944, 'epochs': 60, 'stddev': 0.12284902845629478, 'epochs_print': 10}

Epoch  0/60  LOSS: 1.0462384223937988
Epoch 10/60  LOSS: 0.11001550406217575
Epoch 20/60  LOSS: 0.030465254560112953
Epoch 30/60  LOSS: 0.42259588837623596
Epoch 40/60  LOSS: 0.069508858025074
Epoch 50/60  LOSS: 0.059178173542022705
[I 2021-11-16 01:52:44,870] Trial 10 finished with value: 0.056302929960008505 and parameters: {'hidden_layer': 255, 'clip_margin': 128, 'learning_rate': 0.1037578631718944, 'stddev': 0.12284902845629478}. Best is trial 6 with value: 0.037891683366636494.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 228, 'clip_margin': 107, 'learning_rate': 0.18723507680238366, 'epochs': 60, 'stddev': 0.14822330642526704, 'epochs_print': 10}

Epoch  0/60  LOSS: 2.756204843521118
Epoch 10/60  LOSS: 0.11442955583333969
Epoch 20/60  LOSS: 0.23447415232658386
Epoch 30/60  LOSS: 0.2346375286579132
```

```
Epoch 40/60  LOSS: 0.2365354597568512
Epoch 50/60  LOSS: 0.2856990396976471
[I 2021-11-16 01:54:19,352] Trial 11 finished with value: 0.057764799662267 06 and parameters: {'hidden_layer': 228, 'clip_margin': 107, 'learning_rate': 0.1872350768
0238366, 'stddev': 0.14822330642526704}. Best is trial 6 with value: 0.037891683366636494.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 129, 'clip_margin': 113, 'learning_rate': 0.10347124612018005, 'epochs': 60, 'stddev': 0.1328100108071492
1, 'epochs_print': 10}

Epoch  0/60  LOSS: 1.0992286205291748
Epoch 10/60  LOSS: 0.06571561843156815
Epoch 20/60  LOSS: 0.005645844619721174
Epoch 30/60  LOSS: 0.008067562244832516
Epoch 40/60  LOSS: 0.08349921554327011
Epoch 50/60  LOSS: 0.049387332051992416
[I 2021-11-16 01:55:08,521] Trial 12 finished with value: 0.060676627478727006 and parameters: {'hidden_layer': 129, 'clip_margin': 113, 'learning_rate': 0.103471246
12018005, 'stddev': 0.13281001080714921}. Best is trial 6 with value: 0.037891683366636494.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 231, 'clip_margin': 96, 'learning_rate': 0.20224390082039828, 'epochs': 60, 'stddev': 0.1195452790528310
5, 'epochs_print': 10}

Epoch  0/60  LOSS: 15.236525535583496
Epoch 10/60  LOSS: 0.05388883128762245
Epoch 20/60  LOSS: 0.20344148576259613
Epoch 30/60  LOSS: 0.24556398391723633
Epoch 40/60  LOSS: 0.24551323056221008
Epoch 50/60  LOSS: 0.24550965428352356
[I 2021-11-16 01:56:43,754] Trial 13 finished with value: 0.245515810332578 66 and parameters: {'hidden_layer': 231, 'clip_margin': 96, 'learning_rate': 0.20224390082
039828, 'stddev': 0.11954527905283105}. Best is trial 6 with value: 0.037891683366636494.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 200, 'clip_margin': 118, 'learning_rate': 0.25292799781418357, 'epochs': 60, 'stddev': 0.1481371505366925
5, 'epochs_print': 10}

Epoch  0/60  LOSS: 21.37560272216797
Epoch 10/60  LOSS: 0.05967293307185173
Epoch 20/60  LOSS: 0.1526317596435547
Epoch 30/60  LOSS: 0.29312822222709656
Epoch 40/60  LOSS: 0.29295840859413147
Epoch 50/60  LOSS: 0.2928783595561981
[I 2021-11-16 01:57:59,679] Trial 14 finished with value: 0.292846120495013 24 and parameters: {'hidden_layer': 200, 'clip_margin': 118, 'learning_rate': 0.2529279978
1418357, 'stddev': 0.14813715053669255}. Best is trial 6 with value: 0.037891683366636494.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 172, 'clip_margin': 103, 'learning_rate': 0.1451839422241405, 'epochs': 60, 'stddev': 0.1334613661500942
7, 'epochs_print': 10}

Epoch  0/60  LOSS: 1.589428424835205
Epoch 10/60  LOSS: 0.17467264831066132
Epoch 20/60  LOSS: 0.1492558717727661
Epoch 30/60  LOSS: 0.22260628640651703
Epoch 40/60  LOSS: 0.20385459065437317
Epoch 50/60  LOSS: 0.2078532874584198
[I 2021-11-16 01:59:00,455] Trial 15 finished with value: 0.329917701648948 2 and parameters: {'hidden_layer': 172, 'clip_margin': 103, 'learning_rate': 0.14518394222
41405, 'stddev': 0.13346136615009427}. Best is trial 6 with value: 0.037891683366636494.
```

In [ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.figure(figsize=(16, 7))
plt.title('Study 2: Last 20% data prediction using LSTM')
plt.xlabel('Days')
plt.ylabel('Scaled Price of Bitcoin')
plt.plot(dados, label='Original data')
plt.plot(dados[:int((len(dados)/5)*4)], label="Training data")
plt.plot(resultados_tests, label='Testing data')
plt.legend()
plt.show()
```

```
/home/vitor/.local/lib/python3.8/site-packages/numpy/core/_asarray.py:136: VisibleDeprecationWarning:

Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you mea
nt to do this, you must specify 'dtype=object' when creating the ndarray
```



In [ ]:
```
optuna.visualization.plot_optimization_history(study_2)
```

In [ ]:
```
optuna.visualization.plot_slice(study_2)
```

In [ ]:
```
optuna.visualization.plot_param_importances(study_2)
```

Parâmetros do Estudo 3

Trials: 16
Tamanho do batch/janela: 7
Hidden layer: 256 à 512
Margem de clip: 128 à 256
Taxa de aprendizado: 0.3 à 0.7

In [ ]:
```
configs = {
  "batch_size"   : 7,
  "window_size"  : 7,
  "hidden_layer" : 32,
  "clip_margin"  : 32,
  "learning_rate": 0.01,
```

```
  "epochs"       : 60,
  "stddev"       : 0.05,
  "epochs_print" : 10
}
bestCase = {
   "batch_size": 0,
   "outputs": [],
   "loss": 1000000
}
bestTrial = 0

def perda3(trial):

    configs["batch_size"] = configs["window_size"] = 7 #trial.suggest_int("batch_size", 3, 31)
    configs["hidden_layer"] = trial.suggest_int("hidden_layer", 8, 32)
    configs["clip_margin"] = trial.suggest_int("clip_margin", 16, 32)
    configs["learning_rate"] = trial.suggest_float("learning_rate", 0.001, 0.005)
    configs["stddev"] = trial.suggest_float("stddev", 0.05, 0.15)

    print("\nConfigs", configs, "\n")

    # Pesos da via de entrada, inputs, objetivos, camada oculta e viés
    inputs = tf.placeholder( tf.float32, [ configs["batch_size"], configs["window_size"], 1 ] )
    targets = tf.placeholder( tf.float32, [ configs["batch_size"], 1 ] )

    weights_input_gate = tf.Variable( tf.truncated_normal([ 1, configs["hidden_layer"] ], stddev=0.05) )
    weights_input_hidden = tf.Variable( tf.truncated_normal( [ configs["hidden_layer"], configs["hidden_layer"] ], stddev=0.05) )
    bias_input = tf.Variable( tf.zeros( [ configs["hidden_layer"] ] ) )

    # Pesos da via do esquecimento, camada oculta e viés
    weights_forget_gate = tf.Variable( tf.truncated_normal( [1, configs["hidden_layer"]], stddev=0.05) )
    weights_forget_hidden = tf.Variable( tf.truncated_normal( [ configs["hidden_layer"], configs["hidden_layer"] ], stddev = configs["stddev"] ) )
    bias_forget = tf.Variable( tf.zeros( configs["hidden_layer"] ) )

    # Pesos da via de saída, camada oculta e viés
    weights_output_gate = tf.Variable( tf.truncated_normal( [1, configs["hidden_layer"]], configs["hidden_layer"] ) )
    weights_output_hidden = tf.Variable( tf.truncated_normal( [ configs["hidden_layer"], configs["hidden_layer"] ], stddev = configs["stddev"] ) )
    bias_output = tf.Variable( tf.zeros( configs["hidden_layer"] ) )

    # Pesos da célula de memória, camada oculta e viés
    weights_memory_cell = tf.Variable( tf.truncated_normal( [1, configs["hidden_layer"]], stddev=0.05 ) )
    weights_memory_cell_hidden = tf.Variable( tf.truncated_normal( [configs["hidden_layer"], configs["hidden_layer"]], stddev = configs["stddev"] ) )
    bias_memory_cell = tf.Variable( tf.zeros( [configs["hidden_layer"] ] ) )

    # Pesos da camada de saída (e viés)
    weights_output = tf.Variable( tf.truncated_normal( [configs["hidden_layer"], 1], stddev = configs["stddev"] ) )
    bias_output_layer = tf.Variable( tf.zeros([1]) )


    # Montando nossa célula LSTM
    def LSTM_cell(input, output, state):
        # Via de entrada
        input_gate = tf.sigmoid(tf.matmul(input, weights_input_gate) + tf.matmul(output, weights_input_hidden) + bias_input)

        # Via do esquecimento
        forget_gate = tf.sigmoid(tf.matmul(input, weights_forget_gate) + tf.matmul(output, weights_forget_hidden) + bias_forget)

        # Célula de memória
        memory_cell = tf.tanh(tf.matmul(input, weights_memory_cell) + tf.matmul(output, weights_memory_cell_hidden) + bias_memory_cell)

        # Estado (combinação dos anteriores)
        state = state * forget_gate + input_gate * memory_cell

        # Via de saída
        output_gate = tf.sigmoid(tf.matmul(input, weights_output_gate) + tf.matmul(output, weights_output_hidden) + bias_output)

        # Saída (aplicação de função tangente à via de saída)
        output = output_gate * tf.tanh(state)
        return state, output


    outputs = []
    for bs in range(configs["batch_size"]):
        batch_state  = np.zeros([1, configs["hidden_layer"]], dtype=np.float32)
        batch_output = np.zeros([1, configs["hidden_layer"]], dtype=np.float32)

        for ws in range(configs["window_size"]):
            batch_state, batch_output = LSTM_cell( tf.reshape( inputs[bs][ws], (-1, 1) ), batch_state, batch_output )

        outputs.append( tf.matmul( batch_output, weights_output ) + bias_output_layer )

    # Definindo perda

    losses = []

    for o in range(len(outputs)):
        losses.append( tf.losses.mean_squared_error( tf.reshape( targets[o], (-1, 1) ), outputs[o] ) )

    loss = tf.reduce_mean(losses)

    # Definindo o otimizador com o gradiente de clipagem (?)
    gradients   = tf.gradients( loss, tf.trainable_variables() )
    clipped, _  = tf.clip_by_global_norm( gradients, configs["clip_margin"] )
    optimizer   = tf.train.AdamOptimizer( configs["learning_rate"] )
    trained_opt = optimizer.apply_gradients( zip( gradients, tf.trainable_variables() ) )

    session = tf.Session()
    session.run( tf.global_variables_initializer() )

    best_epochs = [10.0]

    for e in range(configs["epochs"]):
        trained_scores = []
        ee = 0
        epoch_loss = []
        while(ee + configs["batch_size"]) <= len(X_treino):
            X_batch = X_treino[ee:ee+configs["batch_size"]]
            y_batch = y_treino[ee:ee+configs["batch_size"]]

            o, c, _ = session.run( [outputs, loss, trained_opt], feed_dict={inputs:X_batch, targets: y_batch} )

            epoch_loss.append(c)
            trained_scores.append(o)
            ee += configs["batch_size"]

        if(e % configs["epochs_print"]) == 0:
            if e == 0:
                print( 'Epoch  {}/{}'.format(e, configs["epochs"]), ' LOSS: {}'.format( np.mean( epoch_loss ) ) )
            else:
                print( 'Epoch {}/{}'.format(e, configs["epochs"]), ' LOSS: {}'.format( np.mean( epoch_loss ) ) )

    loss = sum(epoch_loss)/len(epoch_loss)
    if loss < bestCase["loss"]:
        bestCase["loss"] = loss
        global sup
        sup = []
        for ts in range(len(trained_scores)):
            for tsr in range(len( trained_scores[ts] )):
```

```
                sup.append( trained_scores[ts][tsr][0] )

        tests = []
        t = 0
        while( t + configs["batch_size"] <= len(X_teste) ):
            o = session.run( [outputs], feed_dict = { inputs: X_teste[t:t+configs["batch_size"]] } )
            t += configs["batch_size"]
            tests.append(o)

        novos_tests = []
        for ts in range(len(tests)):
            for tsr in range(len( tests[ts][0] )):
                novos_tests.append( tests[ts][0][tsr] )

        global resultados_tests
        resultados_tests = []
        for r in range(1264):
            if r >= 1019:
                resultados_tests.append(novos_tests[r-1019])
            else:
                resultados_tests.append(None)

    # Retornando a média de LOSS
    return loss

study_3 = optuna.create_study(direction='minimize')
study_3.optimize(perda3, n_trials=16)
```

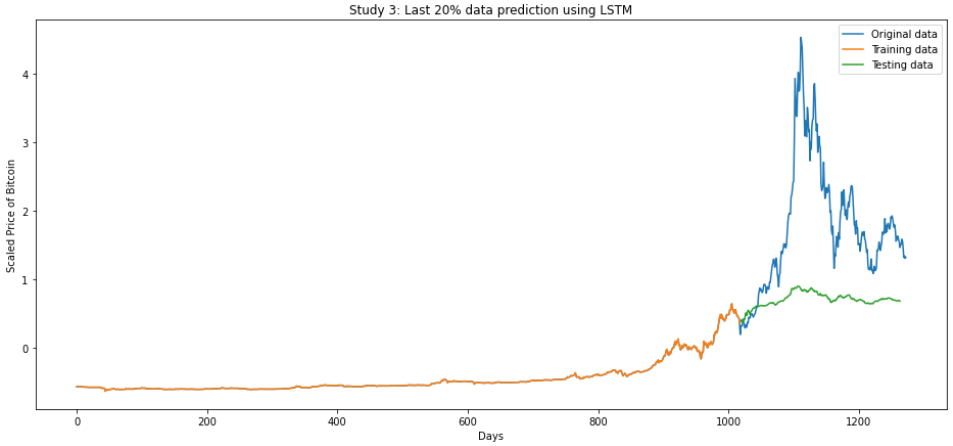
```
[I 2021-11-16 02:20:28,338] A new study created in memory with name: no-name-beff6c9a-658c-42ac-870a-1735035dc870
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 30, 'clip_margin': 22, 'learning_rate': 0.0037347286149582125, 'epochs': 60, 'stddev': 0.10553049163883402, 'epochs_print': 10}

Epoch  0/60  LOSS: 0.12742051482200623
Epoch 10/60  LOSS: 0.012536275200545788
Epoch 20/60  LOSS: 0.006031524855643511
Epoch 30/60  LOSS: 0.0007955959299579263
Epoch 40/60  LOSS: 0.000582517241127789
Epoch 50/60  LOSS: 0.0005526210879907012
[I 2021-11-16 02:21:12,211] Trial 0 finished with value: 0.0004801760641965239 3 and parameters: {'hidden_layer': 30, 'clip_margin': 22, 'learning_rate': 0.0037347286149582125, 'stddev': 0.10553049163883402}. Best is trial 0 with value: 0.0004801760641965239 3.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 20, 'clip_margin': 16, 'learning_rate': 0.002206639599309111, 'epochs': 60, 'stddev': 0.13337702038120913, 'epochs_print': 10}

Epoch  0/60  LOSS: 3.8276615142822266
Epoch 10/60  LOSS: 0.014626799151301384
Epoch 20/60  LOSS: 0.011472244746983051
Epoch 30/60  LOSS: 0.011270339600741863
Epoch 40/60  LOSS: 0.010271896608173847
Epoch 50/60  LOSS: 0.00484803831204772
[I 2021-11-16 02:21:55,501] Trial 1 finished with value: 0.0013894868760589503 and parameters: {'hidden_layer': 20, 'clip_margin': 16, 'learning_rate': 0.002206639599309111, 'stddev': 0.13337702038120913}. Best is trial 0 with value: 0.0004801760641965239 3.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 11, 'clip_margin': 26, 'learning_rate': 0.004227672658599115, 'epochs': 60, 'stddev': 0.05890245612756484, 'epochs_print': 10}

Epoch  0/60  LOSS: 0.1708863228559494
Epoch 10/60  LOSS: 0.008308297954499722
Epoch 20/60  LOSS: 0.004092466551810503
Epoch 30/60  LOSS: 0.002523035043850541
Epoch 40/60  LOSS: 0.0011005369015038013
Epoch 50/60  LOSS: 0.0011642819736152887
[I 2021-11-16 02:22:40,207] Trial 2 finished with value: 0.0011698623783621739 and parameters: {'hidden_layer': 11, 'clip_margin': 26, 'learning_rate': 0.004227672658599115, 'stddev': 0.05890245612756484}. Best is trial 0 with value: 0.0004801760641965239 3.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 14, 'clip_margin': 19, 'learning_rate': 0.003377067548621225, 'epochs': 60, 'stddev': 0.07679359438819774, 'epochs_print': 10}

Epoch  0/60  LOSS: 0.2637307643890381
Epoch 10/60  LOSS: 0.018171796575188637
Epoch 20/60  LOSS: 0.009936193004250526
Epoch 30/60  LOSS: 0.007227221038192511
Epoch 40/60  LOSS: 0.0010843881173059344
Epoch 50/60  LOSS: 0.0006544325733557343
[I 2021-11-16 02:23:20,744] Trial 3 finished with value: 0.0006450670262454189 and parameters: {'hidden_layer': 14, 'clip_margin': 19, 'learning_rate': 0.003377067548621225, 'stddev': 0.07679359438819774}. Best is trial 0 with value: 0.0004801760641965239 3.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 31, 'clip_margin': 22, 'learning_rate': 0.0011244264980527455, 'epochs': 60, 'stddev': 0.07698484217075284, 'epochs_print': 10}

Epoch  0/60  LOSS: 0.2072707861661911
Epoch 10/60  LOSS: 0.013411223888397217
Epoch 20/60  LOSS: 0.011885814368724823
Epoch 30/60  LOSS: 0.006627291440963745
Epoch 40/60  LOSS: 0.0006833273801021278
Epoch 50/60  LOSS: 0.0004563819093164056 5
[I 2021-11-16 02:24:04,538] Trial 4 finished with value: 0.0004387548195379936 and parameters: {'hidden_layer': 31, 'clip_margin': 22, 'learning_rate': 0.0011244264980527455, 'stddev': 0.07698484217075284}. Best is trial 4 with value: 0.0004387548195379936.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 18, 'clip_margin': 31, 'learning_rate': 0.003329377747437302, 'epochs': 60, 'stddev': 0.10821253658754612, 'epochs_print': 10}

Epoch  0/60  LOSS: 0.808253288269043
Epoch 10/60  LOSS: 0.01880737580358982
Epoch 20/60  LOSS: 0.012180887162685394
Epoch 30/60  LOSS: 0.008943503722548485
Epoch 40/60  LOSS: 0.006223150063306093
Epoch 50/60  LOSS: 0.0013978707138448954
[I 2021-11-16 02:24:47,960] Trial 5 finished with value: 0.0011916806593286185 and parameters: {'hidden_layer': 18, 'clip_margin': 31, 'learning_rate': 0.003329377747437302, 'stddev': 0.10821253658754612}. Best is trial 4 with value: 0.0004387548195379936.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 23, 'clip_margin': 27, 'learning_rate': 0.0030071545557830153, 'epochs': 60, 'stddev': 0.05474258273152165 4, 'epochs_print': 10}

Epoch  0/60  LOSS: 3.029115915298462
Epoch 10/60  LOSS: 0.015111998654901981
Epoch 20/60  LOSS: 0.0119004612788558
Epoch 30/60  LOSS: 0.009546173736453056
Epoch 40/60  LOSS: 0.008808310143649578
Epoch 50/60  LOSS: 0.006581937894225120 5
[I 2021-11-16 02:25:24,399] Trial 6 finished with value: 0.0008531552140891955 and parameters: {'hidden_layer': 23, 'clip_margin': 27, 'learning_rate': 0.0030071545557830153, 'stddev': 0.05474258273152165 4}. Best is trial 4 with value: 0.0004387548195379936.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 8, 'clip_margin': 27, 'learning_rate': 0.001231174400482109, 'epochs': 60, 'stddev': 0.14094330501950722, 'epochs_print': 10}

Epoch  0/60  LOSS: 0.3934575319290161
Epoch 10/60  LOSS: 0.011271101422607899
Epoch 20/60  LOSS: 0.0017917088698595762
Epoch 30/60  LOSS: 0.0005708968383260071
Epoch 40/60  LOSS: 0.0004703162994701415 3
Epoch 50/60  LOSS: 0.0004333894758019596 3
[I 2021-11-16 02:26:06,656] Trial 7 finished with value: 0.0004360726011609899 3 and parameters: {'hidden_layer': 8, 'clip_margin': 27, 'learning_rate': 0.001231174400482109, 'stddev': 0.14094330501950722}. Best is trial 7 with value: 0.0004360726011609899 3.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 24, 'clip_margin': 27, 'learning_rate': 0.0026469178069660902, 'epochs': 60, 'stddev': 0.06063117870609974, 'epochs_print': 10}

Epoch  0/60  LOSS: 2.3672053813934326
Epoch 10/60  LOSS: 0.018493549898266792
Epoch 20/60  LOSS: 0.011574538424611092
Epoch 30/60  LOSS: 0.010370494797825813
```

```
Epoch 40/60  LOSS: 0.008938018232584
Epoch 50/60  LOSS: 0.00445427093654871
[I 2021-11-16 02:26:47,408] Trial 8 finished with value: 0.002078600580459571 and parameters: {'hidden_layer': 24, 'clip_margin': 27, 'learning_rate': 0.0026469178069660902, 'stddev': 0.06063117870609974}. Best is trial 7 with value: 0.00043607260116098993.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 23, 'clip_margin': 21, 'learning_rate': 0.003122987425426211, 'epochs': 60, 'stddev': 0.06153782807391946, 'epochs_print': 10}

Epoch  0/60  LOSS: 3.0468831062316895
Epoch 10/60  LOSS: 0.018257133662700653
Epoch 20/60  LOSS: 0.012274855747818947
Epoch 30/60  LOSS: 0.010673454031348228
Epoch 40/60  LOSS: 0.009268088266253471
Epoch 50/60  LOSS: 0.001639710390008986
[I 2021-11-16 02:27:18,532] Trial 9 finished with value: 0.0016898407357718403 and parameters: {'hidden_layer': 23, 'clip_margin': 21, 'learning_rate': 0.003122987425426211, 'stddev': 0.06153782807391946}. Best is trial 7 with value: 0.00043607260116098993.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 8, 'clip_margin': 31, 'learning_rate': 0.0012914884095612668, 'epochs': 60, 'stddev': 0.1484905195053996, 'epochs_print': 10}

Epoch  0/60  LOSS: 0.4017440378665924
Epoch 10/60  LOSS: 0.015868326649069786
Epoch 20/60  LOSS: 0.00542991841211915
Epoch 30/60  LOSS: 0.0015131945256143808
Epoch 40/60  LOSS: 0.0007766035269014537
Epoch 50/60  LOSS: 0.0009121187031269073
[I 2021-11-16 02:27:55,680] Trial 10 finished with value: 0.001072110742205919 and parameters: {'hidden_layer': 8, 'clip_margin': 31, 'learning_rate': 0.0012914884095612668, 'stddev': 0.1484905195053996}. Best is trial 7 with value: 0.00043607260116098993.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 32, 'clip_margin': 24, 'learning_rate': 0.0010899252712851255, 'epochs': 60, 'stddev': 0.08537265229045344, 'epochs_print': 10}

Epoch  0/60  LOSS: 0.20904411375522614
Epoch 10/60  LOSS: 0.012366831302642822
Epoch 20/60  LOSS: 0.01079853530973196
Epoch 30/60  LOSS: 0.00799987930804491
Epoch 40/60  LOSS: 0.0005992865771986544
Epoch 50/60  LOSS: 0.00042068568291142583
[I 2021-11-16 02:28:39,780] Trial 11 finished with value: 0.000456215770326441 and parameters: {'hidden_layer': 32, 'clip_margin': 24, 'learning_rate': 0.0010899252712851255, 'stddev': 0.0853726522904534}. Best is trial 7 with value: 0.00043607260116098993.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 28, 'clip_margin': 29, 'learning_rate': 0.0017920075740621118, 'epochs': 60, 'stddev': 0.12802238957680404, 'epochs_print': 10}

Epoch  0/60  LOSS: 0.2187896966934204
Epoch 10/60  LOSS: 0.012097591534256935
Epoch 20/60  LOSS: 0.005702266003936529
Epoch 30/60  LOSS: 0.0004958926583640277
Epoch 40/60  LOSS: 0.000359265977749601
Epoch 50/60  LOSS: 0.0002908901951741427
[I 2021-11-16 02:29:25,913] Trial 12 finished with value: 0.0003548882499517531 and parameters: {'hidden_layer': 28, 'clip_margin': 29, 'learning_rate': 0.0017920075740621118, 'stddev': 0.12802238957680404}. Best is trial 12 with value: 0.0003548882499517531.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 26, 'clip_margin': 29, 'learning_rate': 0.0018417951021639603, 'epochs': 60, 'stddev': 0.12806204320697395, 'epochs_print': 10}

Epoch  0/60  LOSS: 0.3499254882335663
Epoch 10/60  LOSS: 0.010308372788131237
Epoch 20/60  LOSS: 0.0035767140798270702
Epoch 30/60  LOSS: 0.006851503625512123
Epoch 40/60  LOSS: 0.0030439298134297132
Epoch 50/60  LOSS: 0.002161358715966344
[I 2021-11-16 02:30:05,012] Trial 13 finished with value: 0.00239677051871820 33 and parameters: {'hidden_layer': 26, 'clip_margin': 29, 'learning_rate': 0.0018417951021639603, 'stddev': 0.12806204320697395}. Best is trial 12 with value: 0.0003548882499517531.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 27, 'clip_margin': 29, 'learning_rate': 0.0018183981182289845, 'epochs': 60, 'stddev': 0.14930431145435258, 'epochs_print': 10}

Epoch  0/60  LOSS: 0.29849743843078613
Epoch 10/60  LOSS: 0.011748317629098892
Epoch 20/60  LOSS: 0.004437736701220274
Epoch 30/60  LOSS: 0.0026547126471996307
Epoch 40/60  LOSS: 0.00284851947799325
Epoch 50/60  LOSS: 0.001782021252438426
[I 2021-11-16 02:30:51,679] Trial 14 finished with value: 0.0023361934949426887 and parameters: {'hidden_layer': 27, 'clip_margin': 29, 'learning_rate': 0.0018183981182289845, 'stddev': 0.14930431145435258}. Best is trial 12 with value: 0.0003548882499517531.
Configs {'batch_size': 7, 'window_size': 7, 'hidden_layer': 14, 'clip_margin': 32, 'learning_rate': 0.0016600287562503746, 'epochs': 60, 'stddev': 0.12618904711676174, 'epochs_print': 10}

Epoch  0/60  LOSS: 1.624199390411377
Epoch 10/60  LOSS: 0.0257050059735775
Epoch 20/60  LOSS: 0.013584212400019169
Epoch 30/60  LOSS: 0.009902846999466642
Epoch 40/60  LOSS: 0.00888871680945158
Epoch 50/60  LOSS: 0.007261344697326422
[I 2021-11-16 02:31:36,613] Trial 15 finished with value: 0.0008607498633644919 and parameters: {'hidden_layer': 14, 'clip_margin': 32, 'learning_rate': 0.0016600287562503746, 'stddev': 0.1261890471167614}. Best is trial 12 with value: 0.0003548882499517531.
```

In [ ]:

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.figure(figsize=(16, 7))
plt.title('Study 3: Last 20% data prediction using LSTM')
plt.xlabel('Days')
plt.ylabel('Scaled Price of Bitcoin')
plt.plot(dados, label='Original data')
plt.plot(dados[:int((len(dados)/5)*4)], label="Training data")
plt.plot(resultados_tests, label='Testing data')
plt.legend()
plt.show()
```

```
/home/vitor/.local/lib/python3.8/site-packages/numpy/core/_asarray.py:136: VisibleDeprecationWarning:

Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you mea
nt to do this, you must specify 'dtype=object' when creating the ndarray
```



In [ ]:
```
optuna.visualization.plot_optimization_history(study_3)
```

In [ ]:
```
optuna.visualization.plot_slice(study_3)
```

In [ ]:
```
optuna.visualization.plot_param_importances(study_3)
```

Resultados

Utilizamos para todos os testes o mesmo tamanho de batch size e window size por acreditarmos ser uma boa configuração padrão. Isto posto, alteramos apenas as quantidades de hidden layer, taxa de aprendizagem, desvio padrão e margem de clip. Podemos ver a partir dos gráficos que o primeiro estudo foi o que obteve melhor resultado. No segundo estudo, ao tentarmos otimizar os parâmetros acabamos por exagerar e a predição saiu muito errada para cima, bastante distante do objetivo. No terceiro estudo, tentamos reduzir o problema do segundo estudo, e acabamos por encontrar a situação inversa, nossa predição errou os valores para baixo, ainda assim, foi melhor que a do segundo estudo. Quanto a importância dos parâmetros, observados o do primeiro, vemos que o tamanho da camada oculta é a que mais impactou no desempenho de nossa rede LSTM; em segundo lugar, temos o desvio padrão como fator importante ao primeiro estudo e a taxa de aprendizagem em terceiro. De acordo com o primeiro estudo, a melhor configuração seria algo em torno de 100 neurônios, uma taxa de clipagem em torno de 20, com taxa de aprendizado de 5% e desvio padrão indiferente.