Bitcoin Price Prediction Project with ARIMA, XGBoost and LSTM

LSTM

This notebook is part of Bitcoin Price Prediction Project with LSTM model. Complete project consists of three notebooks with sections as below.

1. Abstract
2. Data description # (01_BTC_Price_Prediction - ARIMA.ipynb notebook)
3. Gathering data # (01_BTC_Price_Prediction - ARIMA.ipynb notebook)
4. Data preprocessing
5. ARIMA Model # (01_BTC_Price_Prediction - ARIMA.ipynb notebook)
   - 5.1. ARIMA => Introduction
   - 5.2. ARIMA Model => function definition
   - 5.3. ARIMA Model => Forecasting
     - complete data
     - data start from 2018-01-01
     - data start from 2021-01-01
   - 5.4. ARIMA Model => summary
6. XGBoost # (02_BTC_Price_Prediction - XGBoost.ipynb notebook)
   - 6.1. XGBRegressor => Introduction
   - 6.2. XGBRegressor => function definition
   - 6.3. XGBRegressor => Forecasting
     - complete data
     - data start from 2018-01-01
     - data start from 2021-01-01
   - 6.4. XGBRegressor => summary
7. LSTM Model # (this notebook)
   - 7.1. LSTM => Introduction
   - 7.2. LSTM => function definition
   - 7.3. LSTM => Forecasting
     - complete data
     - data start from 2018-01-01
     - data start from 2021-01-01
   - 7.4. LSTM => summary
8. Project summary # (this notebook)

In [1]:

```
import pandas as pd
import numpy as np
from numpy import array

from datetime import datetime, date

import matplotlib.pyplot as plt

from tqdm import tqdm
```

4. Data preprocessing

In [2]:

```
# restore data backup
btc_df_gathered = pd.read_csv("BTC_Preprocess_Backup.csv", sep = ",", skipinitialspace=True)
btc_df_gathered.head()
```

Out[2]:

	time_open	time_close	open	high	low	close	market_cap	volume
0	2010-07-17T00:00:00Z	2010-07-17T23:59:59Z	0.04951	0.04951	0.04951	0.04951	NaN	NaN
1	2010-07-18T00:00:00Z	2010-07-18T23:59:59Z	0.04951	0.04951	0.04951	0.04951	NaN	NaN
2	2010-07-19T00:00:00Z	2010-07-19T23:59:59Z	0.08584	0.08584	0.08584	0.08584	NaN	NaN
3	2010-07-20T00:00:00Z	2010-07-20T23:59:59Z	0.08080	0.08080	0.08080	0.08080	NaN	NaN
4	2010-07-21T00:00:00Z	2010-07-21T23:59:59Z	0.07474	0.07474	0.07474	0.07474	NaN	NaN

In [3]:

```
# copy data to new df
btc_df_1 = btc_df_gathered
```

In [4]:

```
# check datatypes
btc_df_1.dtypes
```

Out[4]:

```
time_open      object
time_close     object
open          float64
high          float64
low           float64
close         float64
market_cap    float64
volume        float64
dtype: object
```

In [5]:

```
# change df index to date
btc_df_1.index = pd.to_datetime(btc_df_1["time_close"]).dt.date

# drop "time_open", "time_close" columns
btc_df_1.drop(["time_open", "time_close"], axis=1, inplace=True)
btc_df_1.head()
```

Out[5]:

time_close	open	high	low	close	market_cap	volume
2010-07-17	0.04951	0.04951	0.04951	0.04951	NaN	NaN
2010-07-18	0.04951	0.04951	0.04951	0.04951	NaN	NaN
2010-07-19	0.08584	0.08584	0.08584	0.08584	NaN	NaN
2010-07-20	0.08080	0.08080	0.08080	0.08080	NaN	NaN
2010-07-21	0.07474	0.07474	0.07474	0.07474	NaN	NaN

In [6]:

```
# show df shape
btc_df_1.shape
```

Out[6]: (4276, 6)

In [7]:

```
btc_df_1.describe()
```

Out[7]:

	open	high	low	close	market_cap	volume
count	4276.000000	4276.000000	4276.000000	4276.000000	3.260000e+03	3.017000e+03
mean	7620.722003	7818.648308	7411.105750	7632.136514	1.835636e+11	1.205864e+10
std	14287.143745	14651.680612	13896.018691	14300.996427	2.947438e+11	1.710047e+10
min	0.049510	0.049510	0.049510	0.049510	7.784112e+08	2.857830e+06
25%	111.217500	114.591000	107.450000	111.500000	6.510601e+09	5.567090e+07
50%	638.137000	656.522000	627.096000	638.393000	6.466303e+10	3.682395e+09
75%	8191.225324	8321.616465	7952.276240	8196.622730	1.752686e+11	1.885380e+10
max	67589.872823	68692.137037	66457.970074	67589.768671	1.275266e+12	1.363702e+11

7. LSTM Model

7.1. LSTM => Introduction

This notebook was created to predict the Bitcoin price (in USD) using the LSTM model. The model has been programmed in a function to perform a forecast with different data ranges. LSTM model contains parameters that have been empirically optimized.

An additional function has been produced to prepare a sequence of a data - number of past days to look for one future day forecasting. In this notebook, LSTM model will look just one day in to the history to predict next day.

The mean square error (RMSE) was used to evaluate the performance of the model - it should be as low as possible.

Training data is the first 80% of the dataset and the rest is in the test datasets.

At the end of the notebook, a summary of the task is presented along with a graph of the RMSE results of the model.

Long Short-Term Memory - in short LSTM models are type of recurrent neural network (RNN) capable of learning sequences of observations. This makes LSTM well suited for time series forecasting.

LSTM works better with normalized data. For that reason, MinMaxScaler has been used in range from 0 to 1. Reshape data for purpose of LSTM has been done as well.

After fitting, MinMaxScaler provides "inverse_transform" method to receive more "readable" results.

EarlyStopping was used and penalty is set to l2(0.00001) in both data variants.

In [8]:

```
from sklearn.metrics import mean_squared_error

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.regularizers import l2
```

7.2. LSTM => function definition

In [9]:

```
# split data into sequence
def make_sequence(data, n_steps):

    """
    Function for determinate the sequence pattern of the data.
    n_steps is the amount of days to look back for prediction of the next day;
    n_steps = 2 ==> look back in to 2 day to predict day 3.
    n_steps = 3 ==> look back in to 3 day to predict day 4.
    """

    X = []
    y = []

    for i in range(len(data)):

        # find the end of the sequence
        end_seq = i + n_steps

        # check if we are bout of the sequence
        if end_seq > len(data)-1:
            break

        # get input and output parts of the sequence
        seq_x, seq_y = data[i:end_seq], data[end_seq]
        X.append(seq_x)
        y.append(seq_y)

    return array(X), array(y)
```

In [10]:

```
def lstm_variants(df_variants: list, train_size: float, steps: int):

    """
    This function includes LSTM modeling for Bitcoin price prediction.
    Parameters have been empirically optimized.
    The function is prepared for prediction, based on a sets of data - eg. different ranges of historical data.
    """

    variant = 0  # data variant number
    rmse = []    # LSTM rmse

    # general loop fo data viaraints (data range)
    for df in tqdm(df_variants):

        # ----------------------------------------------------------------
        # increment for data variant number
        variant = variant + 1
        print(f"\nData variant no:", variant, "\n")
        # ----------------------------------------------------------------

        # ----------------------------------------------------------------
        # train, test, split the data
        # train = data * train_size
        # test  = data * (1 - train_size)
        size = int(len(df) * train_size)
        train_data, test_data = df[:size], df[size:]

        # split data into sequence
        X_train, y_train = make_sequence(train_data, steps)
        X_test, y_test = make_sequence(test_data, steps)

        # reshape from [samples, timesteps] into [samples, timesteps, features]
        n_features = 1
        X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], n_features))
        X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], n_features))
        # ----------------------------------------------------------------

        # ----------------------------------------------------------------
        # define early_stopping and penalty
        reg = l2(0.00001)
        early_stopping = EarlyStopping(monitor="val_loss",        # Quantity to be monitored.
                                       patience=20,               # Number of epochs with no improvement after
                                       restore_best_weights=True) # Whether to restore model weights from
                                                                  # the epoch with the best value of the monit
        # ----------------------------------------------------------------

        # ----------------------------------------------------------------
        # create model
        model=Sequential()

        # first LSTM layer
        model.add(LSTM(units = 50,                         # Positive integer, dimensionality of the output spac
                       activation="relu",                  # Activation function to use. Default: `tanh`
                       input_shape = (steps, n_features), # input data shape - "None" will accept "any" given
                       kernel_regularizer=reg              # Regularizer function applied to the `kernel` weight
                      ))

        # output layer
        model.add(Dense(1))  # units: Positive integer, dimensionality of the output space.
                             # Here equals 1 as 1 prediction.
        model.summary()

        # compile the model
        model.compile(optimizer="adam",            # String (name of optimizer) or optimizer instance.
                      loss="mean_squared_error")   # Loss function. Maybe be a string (name of loss function)

        # fit model to the training data
        history = model.fit(
            X_train, y_train,
            epochs=200,                         # Number of epochs to train the model.
            batch_size=64,                      # Number of samples per gradient update. default: 32
            validation_data=(X_test, y_test),   # Data on which to evaluate the loss and any model metrics at the
            callbacks=[early_stopping],         # List of callbacks to apply during training. Here early stop
            verbose=1                           # Verbosity mode. 0 = silent, 1 = progress bar
        )

        # print fiting params
        print("\n", history.params)
        # ----------------------------------------------------------------

        # ----------------------------------------------------------------
        # plot training losses
        loss = history.history["loss"]
        val_loss = history.history["val_loss"]
        epochs = range(len(loss))

        plt.figure(figsize=(8,4), dpi=100)
        plt.plot(epochs, loss, "green", label="Training loss")
        plt.plot(epochs, val_loss, "blue", label="Test loss")
        plt.title("Training and test loss")
        plt.xlabel("Epochs")
        plt.ylabel("Loss")
        plt.legend(loc="best", fontsize=8)
        plt.show()
        # ----------------------------------------------------------------

        # ----------------------------------------------------------------
        # X_test reshape
        X_input = [x.reshape((1, steps, n_features)) for x in X_test]

        # prediction
        y_pred = [(model.predict(x, verbose=0)) for x in X_input]
        y_pred = [y_pred[i][0][0] for i in range(len(y_pred))]
        y_pred = pd.Series(y_pred, index=test_data[steps:].index)
        # ----------------------------------------------------------------

        # ----------------------------------------------------------------
        # plot results with train data
        plt.figure(figsize=(10,5), dpi=100)
        plt.plot(train_data, "green", label="Train data")
        plt.plot(test_data, "blue", label="Actual Price/ Test data")
        plt.plot(y_pred, "orange", label="Predicted Price")

        plt.title("BTC Price Prediction - with forecast")
        plt.xlabel("Time")
        plt.ylabel("BTC Price")
        plt.legend(loc="best", fontsize=8)
        plt.show()

        # plot results predictions and test data - zoom-in results
        plt.figure(figsize=(10,5), dpi=100)
        plt.plot(test_data, "blue", label="Actual Price/ Test data")
        plt.plot(y_pred, "orange", label="Predicted Price")

        plt.title("BTC Price Prediction - with forecast")
        plt.xlabel("Time")
        plt.ylabel("BTC Price")
        plt.legend(loc="best", fontsize=8)
        plt.show()
        # ----------------------------------------------------------------

        # ----------------------------------------------------------------
        # print performance
        rmse_l = mean_squared_error(test_data[steps:].values, y_pred.values, squared=False)
        print(model.__class__.__name__, "variant", variant,":", "RMSE = %.3f" % rmse_l, "\n")
        rmse.append(float(format(rmse_l, ".3f")))
        # ----------------------------------------------------------------

    # end of general loop for data viaraints (data range)

    return rmse
```

7.3. LSTM => Forecasting

In [11]:

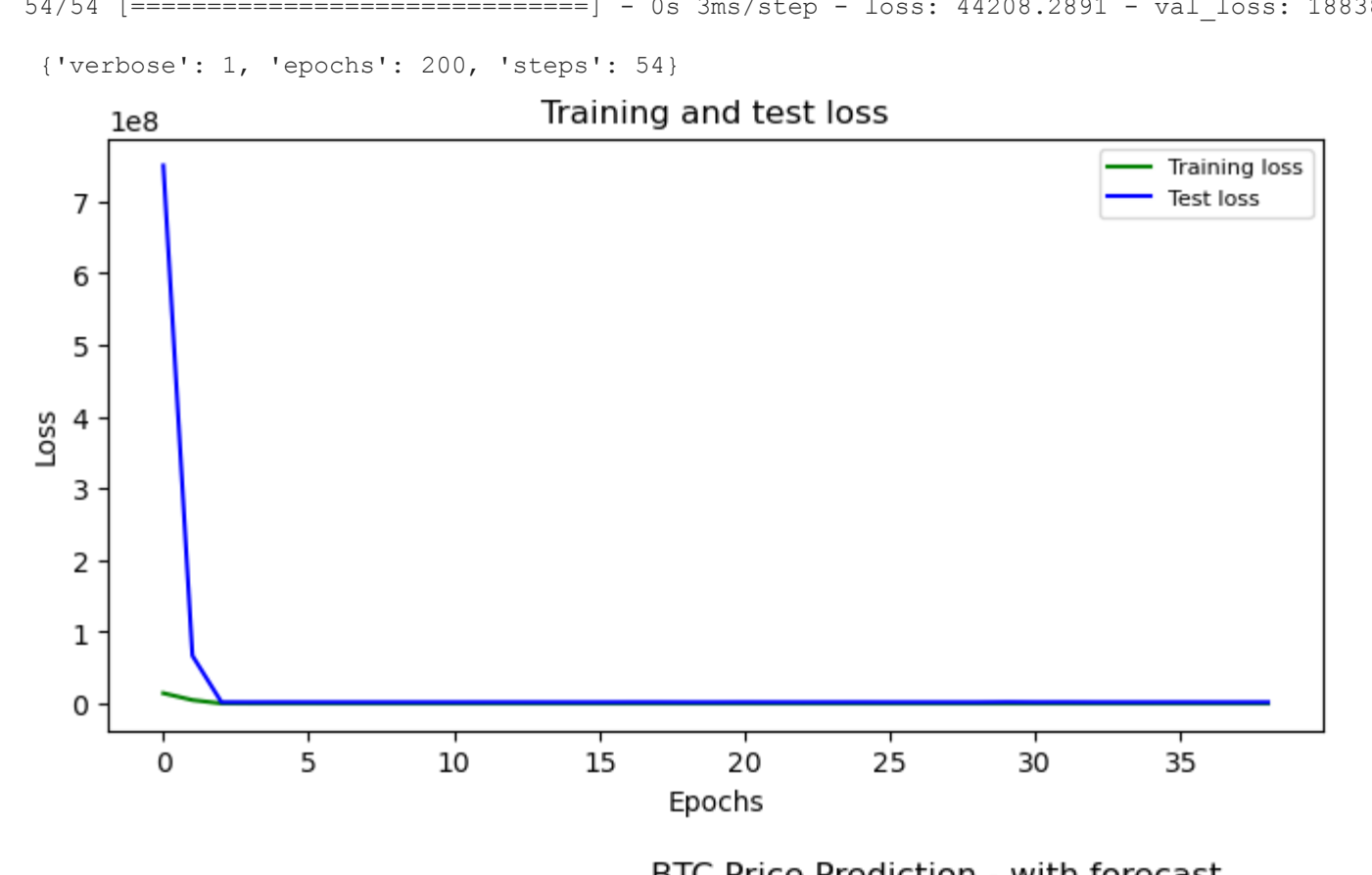
```
# copy full dataset
df_var_1 = btc_df_1["close"]

# copy data starting from 2018-01-01 and replace NaN values to 0
df_var_2 = btc_df_1["close"].iloc[2725:]

# copy data starting from 2021-01-01 and replace NaN values to 0
df_var_3 = btc_df_1["close"].iloc[3821:]

data_variants = [
    df_var_1,
    df_var_2,
    df_var_3
]
```

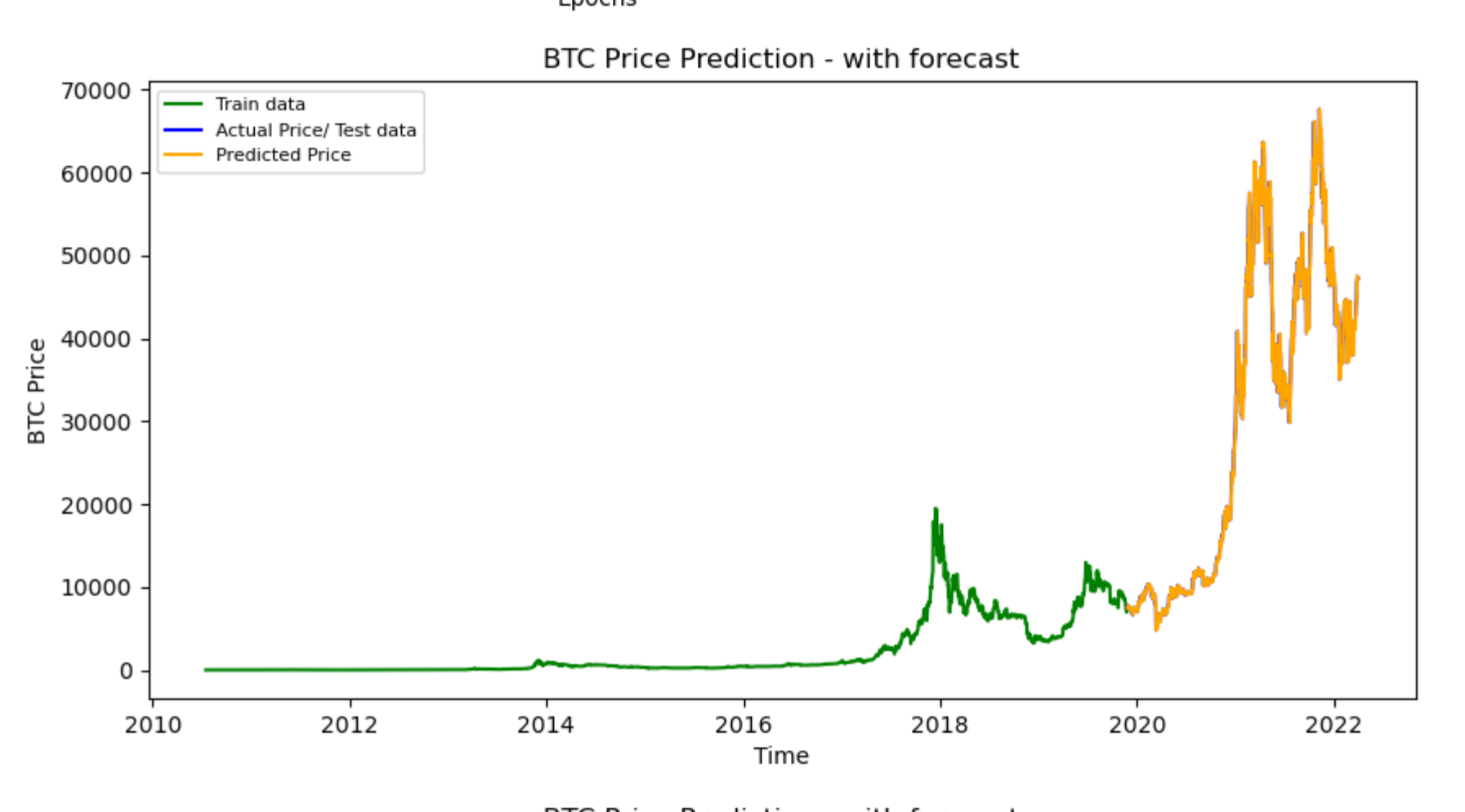
In [12]:

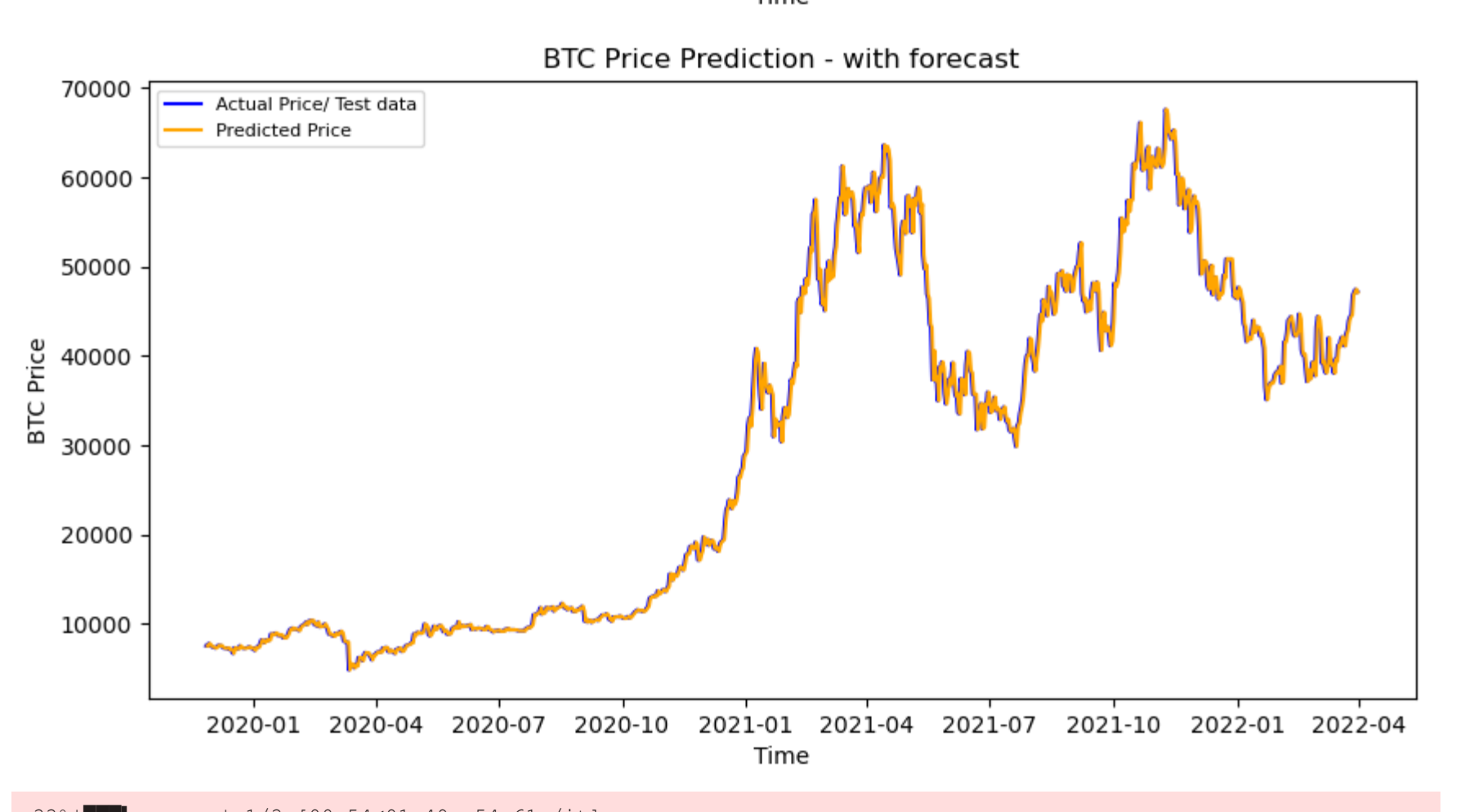
```
%%time

# Forecasting
lstm_variants_rmse = lstm_variants(df_variants=data_variants, train_size=0.8, steps=1)
```

```
  0%|          | 0/3 [00:00<?, ?it/s]
Data variant no: 1

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 50)                10400

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 10,451
Trainable params: 10,451
Non-trainable params: 0
_________________________________________________________________
Epoch 1/200
54/54 [==============================] - 2s 8ms/step - loss: 14449962.0000 - val_loss: 750267840.0000
Epoch 2/200
54/54 [==============================] - 0s 4ms/step - loss: 4889513.5000 - val_loss: 66805156.0000
Epoch 3/200
54/54 [==============================] - 0s 3ms/step - loss: 139405.1406 - val_loss: 1863850.6250
Epoch 4/200
54/54 [==============================] - 0s 3ms/step - loss: 44337.7969 - val_loss: 1829327.3750
Epoch 5/200
54/54 [==============================] - 0s 3ms/step - loss: 44147.0195 - val_loss: 1888469.6250
Epoch 6/200
54/54 [==============================] - 0s 3ms/step - loss: 44296.5078 - val_loss: 1859150.2500
Epoch 7/200
54/54 [==============================] - 0s 7ms/step - loss: 44130.6172 - val_loss: 1828081.3750
Epoch 8/200
54/54 [==============================] - 0s 3ms/step - loss: 44232.6016 - val_loss: 1834264.8750
Epoch 9/200
54/54 [==============================] - 0s 3ms/step - loss: 44537.0586 - val_loss: 1833163.7500
Epoch 10/200
54/54 [==============================] - 0s 3ms/step - loss: 44216.3438 - val_loss: 1838281.7500
Epoch 11/200
54/54 [==============================] - 0s 3ms/step - loss: 44077.2383 - val_loss: 1828905.0000
Epoch 12/200
54/54 [==============================] - 0s 3ms/step - loss: 44174.2305 - val_loss: 1847508.3750
Epoch 13/200
54/54 [==============================] - 0s 3ms/step - loss: 44360.3672 - val_loss: 1861474.7500
Epoch 14/200
54/54 [==============================] - 0s 3ms/step - loss: 44463.0820 - val_loss: 1840830.1250
Epoch 15/200
54/54 [==============================] - 0s 3ms/step - loss: 44347.4414 - val_loss: 1827875.5000
Epoch 16/200
54/54 [==============================] - 0s 3ms/step - loss: 44444.6250 - val_loss: 1827778.6250
Epoch 17/200
54/54 [==============================] - 0s 3ms/step - loss: 44109.7852 - val_loss: 1902953.0000
Epoch 18/200
54/54 [==============================] - 0s 3ms/step - loss: 46300.9727 - val_loss: 1917391.7500
Epoch 19/200
54/54 [==============================] - 0s 3ms/step - loss: 45036.0273 - val_loss: 1827631.8750
Epoch 20/200
54/54 [==============================] - 0s 3ms/step - loss: 44983.6484 - val_loss: 1830603.5000
Epoch 21/200
54/54 [==============================] - 0s 3ms/step - loss: 44838.0547 - val_loss: 1914473.7500
Epoch 22/200
54/54 [==============================] - 0s 3ms/step - loss: 45255.1641 - val_loss: 1867231.2500
Epoch 23/200
54/54 [==============================] - 0s 3ms/step - loss: 44412.6250 - val_loss: 1847387.3750
Epoch 24/200
54/54 [==============================] - 0s 3ms/step - loss: 44365.0703 - val_loss: 1882596.0000
Epoch 25/200
54/54 [==============================] - 0s 3ms/step - loss: 44282.9805 - val_loss: 1904663.7500
Epoch 26/200
54/54 [==============================] - 0s 3ms/step - loss: 44120.9414 - val_loss: 1830839.1250
Epoch 27/200
54/54 [==============================] - 0s 3ms/step - loss: 44863.4375 - val_loss: 1837792.0000
Epoch 28/200
54/54 [==============================] - 0s 3ms/step - loss: 44502.9297 - val_loss: 1828243.0000
Epoch 29/200
54/54 [==============================] - 0s 3ms/step - loss: 44607.2930 - val_loss: 1828883.1250
Epoch 30/200
54/54 [==============================] - 0s 3ms/step - loss: 44290.0352 - val_loss: 2124370.2500
Epoch 31/200
54/54 [==============================] - 0s 3ms/step - loss: 45847.9023 - val_loss: 1918996.0000
Epoch 32/200
54/54 [==============================] - 0s 3ms/step - loss: 44760.1523 - val_loss: 1981174.6250
Epoch 33/200
54/54 [==============================] - 0s 3ms/step - loss: 45051.8789 - val_loss: 1865748.2500
Epoch 34/200
54/54 [==============================] - 0s 3ms/step - loss: 45004.6992 - val_loss: 1848342.6250
Epoch 35/200
54/54 [==============================] - 0s 3ms/step - loss: 44696.7773 - val_loss: 1847893.5000
Epoch 36/200
54/54 [==============================] - 0s 3ms/step - loss: 44523.6602 - val_loss: 1882308.7500
Epoch 37/200
54/54 [==============================] - 0s 3ms/step - loss: 44621.5352 - val_loss: 1838324.7500
Epoch 38/200
54/54 [==============================] - 0s 3ms/step - loss: 44392.3633 - val_loss: 1839152.5000
Epoch 39/200
54/54 [==============================] - 0s 3ms/step - loss: 44208.2891 - val_loss: 1883848.5000
```

{'verbose': 1, 'epochs': 200, 'steps': 54}

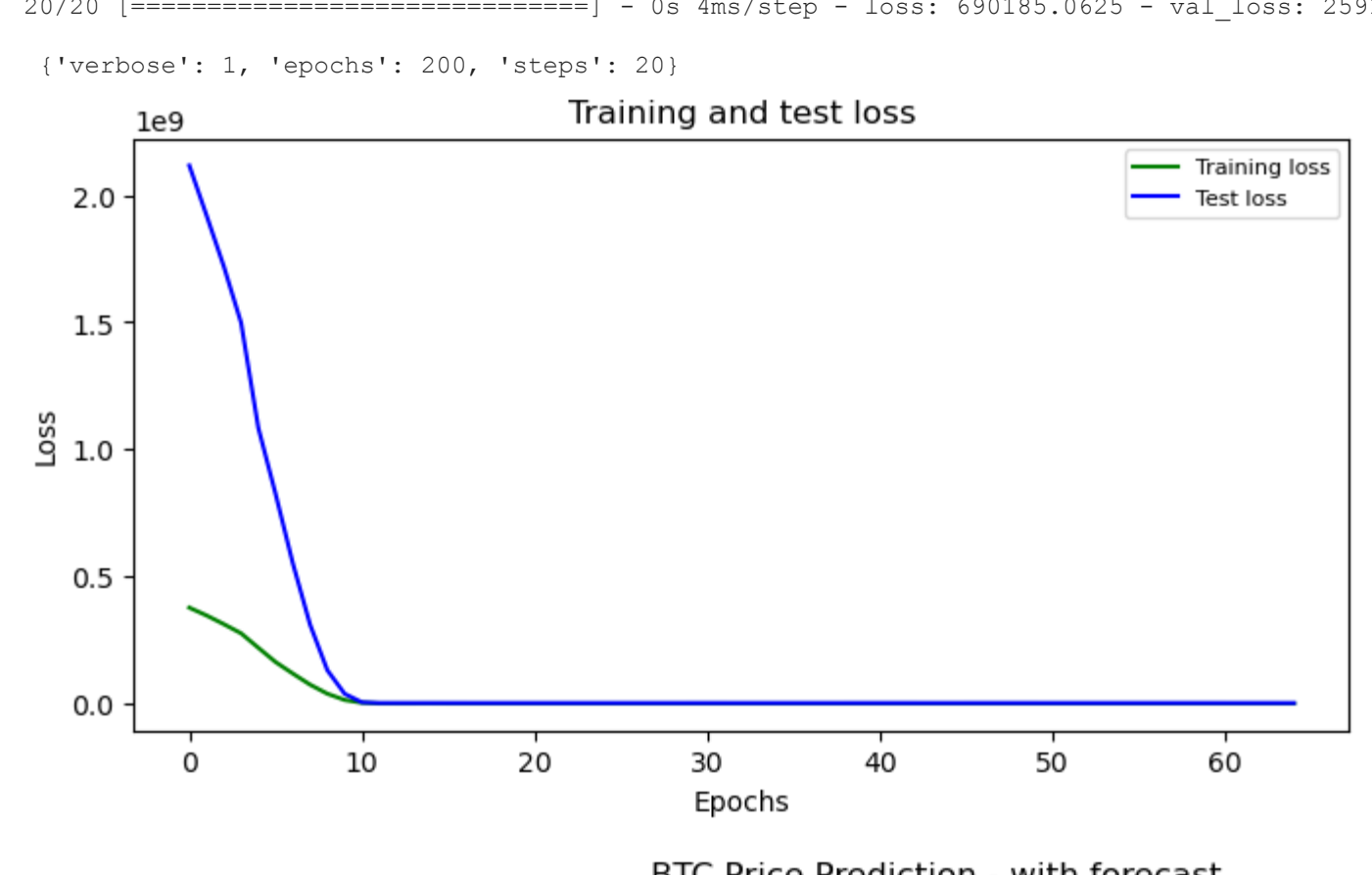


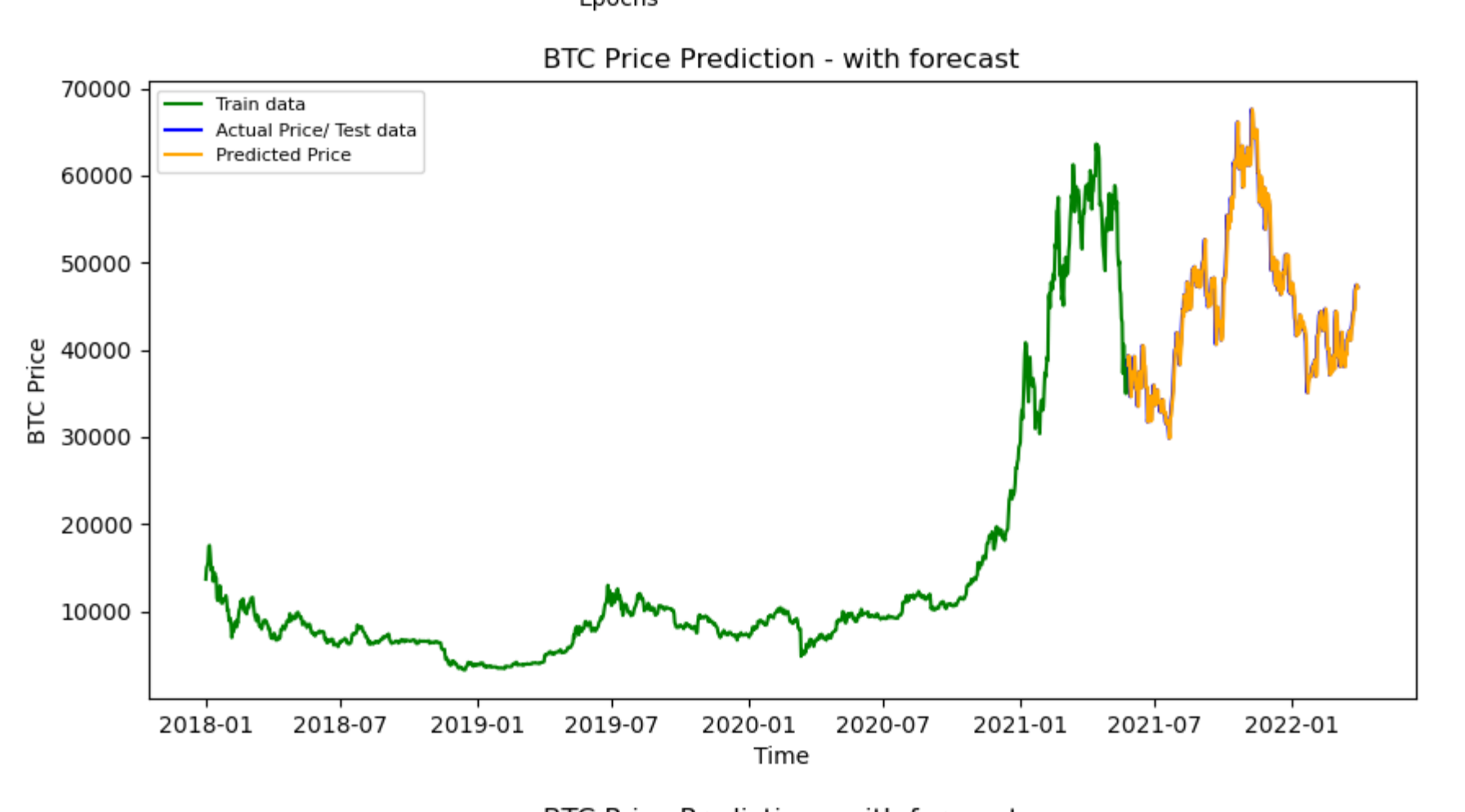


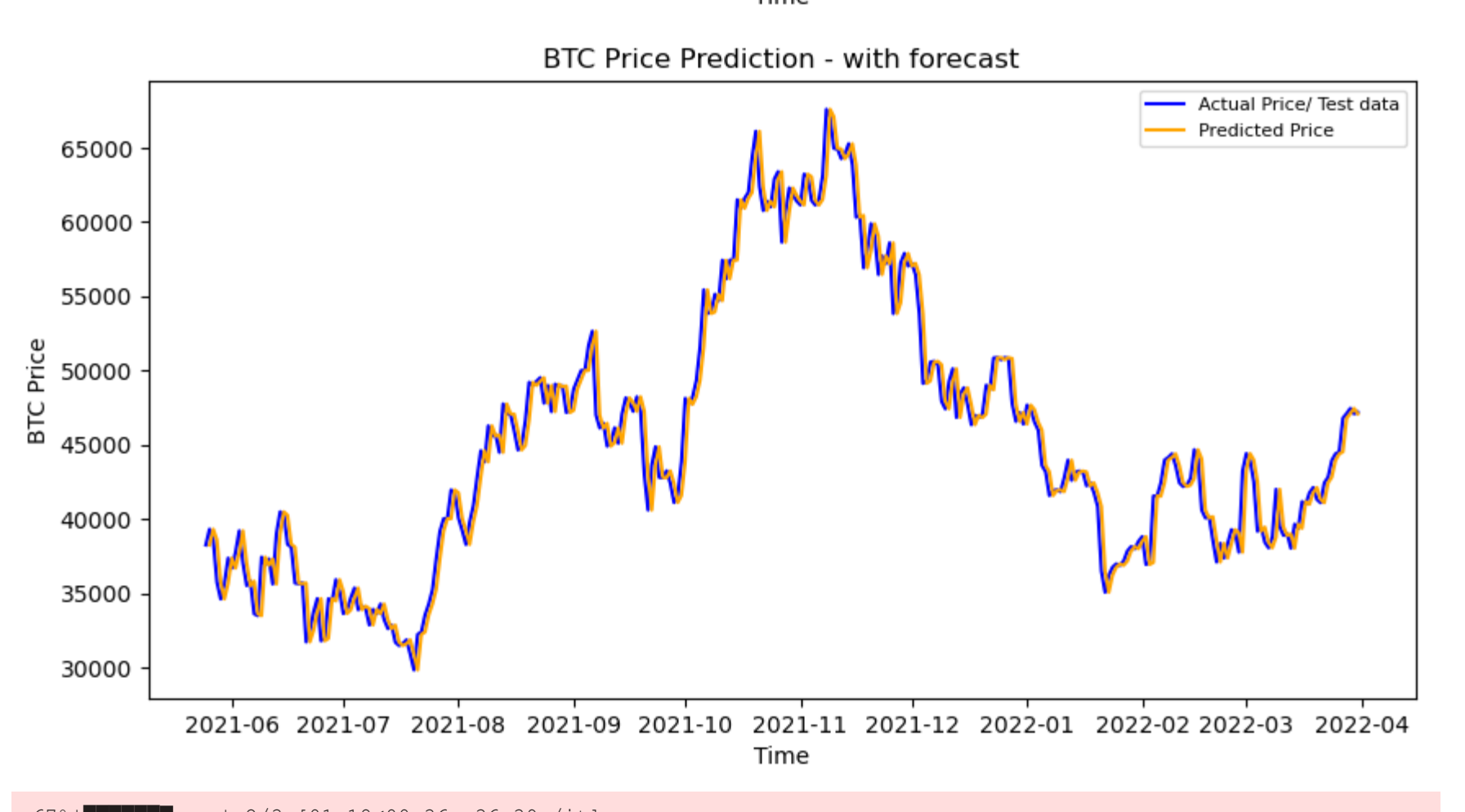



```
 33%|███▎      | 1/3 [00:54<01:49, 54.61s/it]
Sequential variant 1 : RMSE = 1351.900


Data variant no: 2

Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_1 (LSTM)               (None, 50)                10400

 dense_1 (Dense)             (None, 1)                 51

=================================================================
Total params: 10,451
Trainable params: 10,451
Non-trainable params: 0
_________________________________________________________________
Epoch 1/200
20/20 [==============================] - 2s 22ms/step - loss: 377719872.0000 - val_loss: 2116540544.0000
Epoch 2/200
20/20 [==============================] - 0s 3ms/step - loss: 346911296.0000 - val_loss: 1920161024.0000
Epoch 3/200
20/20 [==============================] - 0s 3ms/step - loss: 312759680.0000 - val_loss: 1719822976.0000
Epoch 4/200
20/20 [==============================] - 0s 3ms/step - loss: 276748128.0000 - val_loss: 1498330368.0000
Epoch 5/200
20/20 [==============================] - 0s 4ms/step - loss: 219880144.0000 - val_loss: 1084488064.0000
Epoch 6/200
20/20 [==============================] - 0s 4ms/step - loss: 163704272.0000 - val_loss: 824687488.0000
Epoch 7/200
20/20 [==============================] - 0s 3ms/step - loss: 118350032.0000 - val_loss: 551454784.0000
Epoch 8/200
20/20 [==============================] - 0s 3ms/step - loss: 74787808.0000 - val_loss: 310355520.0000
Epoch 9/200
20/20 [==============================] - 0s 3ms/step - loss: 39155800.0000 - val_loss: 131393816.0000
Epoch 10/200
20/20 [==============================] - 0s 3ms/step - loss: 14619879.0000 - val_loss: 38814588.0000
Epoch 11/200
20/20 [==============================] - 0s 3ms/step - loss: 3605456.5000 - val_loss: 5503261.0000
Epoch 12/200
20/20 [==============================] - 0s 3ms/step - loss: 784417.1250 - val_loss: 2647904.2500
Epoch 13/200
20/20 [==============================] - 0s 3ms/step - loss: 701358.0625 - val_loss: 2675101.5000
Epoch 14/200
20/20 [==============================] - 0s 3ms/step - loss: 684982.7500 - val_loss: 2597172.7500
Epoch 15/200
20/20 [==============================] - 0s 3ms/step - loss: 832763.3750 - val_loss: 2610095.2500
Epoch 16/200
20/20 [==============================] - 0s 3ms/step - loss: 691701.2500 - val_loss: 2605411.5000
Epoch 17/200
20/20 [==============================] - 0s 4ms/step - loss: 688509.5000 - val_loss: 2590515.5000
Epoch 18/200
20/20 [==============================] - 0s 3ms/step - loss: 693133.7500 - val_loss: 2594324.7500
Epoch 19/200
20/20 [==============================] - 0s 3ms/step - loss: 693658.0000 - val_loss: 2597652.2500
Epoch 20/200
20/20 [==============================] - 0s 3ms/step - loss: 688190.7500 - val_loss: 2591413.0000
Epoch 21/200
20/20 [==============================] - 0s 3ms/step - loss: 689770.6250 - val_loss: 2590815.2500
Epoch 22/200
20/20 [==============================] - 0s 3ms/step - loss: 691171.0000 - val_loss: 2590746.5000
Epoch 23/200
20/20 [==============================] - 0s 3ms/step - loss: 688824.5625 - val_loss: 2590153.0000
Epoch 24/200
20/20 [==============================] - 0s 3ms/step - loss: 688475.0625 - val_loss: 2593616.5000
Epoch 25/200
20/20 [==============================] - 0s 3ms/step - loss: 691580.1250 - val_loss: 2592601.7500
Epoch 26/200
20/20 [==============================] - 0s 3ms/step - loss: 689944.6250 - val_loss: 2593883.0000
Epoch 27/200
20/20 [==============================] - 0s 4ms/step - loss: 689120.1250 - val_loss: 2591028.7500
Epoch 28/200
20/20 [==============================] - 0s 3ms/step - loss: 687009.2500 - val_loss: 2591554.0000
Epoch 29/200
20/20 [==============================] - 0s 3ms/step - loss: 689371.6250 - val_loss: 2591674.0000
Epoch 30/200
20/20 [==============================] - 0s 4ms/step - loss: 691423.5625 - val_loss: 2591276.5000
Epoch 31/200
20/20 [==============================] - 0s 4ms/step - loss: 699352.7500 - val_loss: 2614834.5000
Epoch 32/200
20/20 [==============================] - 0s 3ms/step - loss: 690817.0000 - val_loss: 2591073.0000
Epoch 33/200
20/20 [==============================] - 0s 3ms/step - loss: 689458.1875 - val_loss: 2590046.7500
Epoch 34/200
20/20 [==============================] - 0s 4ms/step - loss: 687060.6250 - val_loss: 2590953.0000
Epoch 35/200
20/20 [==============================] - 0s 4ms/step - loss: 689042.1875 - val_loss: 2592048.7500
Epoch 36/200
20/20 [==============================] - 0s 3ms/step - loss: 687048.1875 - val_loss: 2589534.5000
Epoch 37/200
20/20 [==============================] - 0s 3ms/step - loss: 687783.5625 - val_loss: 2590356.7500
Epoch 38/200
20/20 [==============================] - 0s 3ms/step - loss: 686826.0000 - val_loss: 2594816.0000
Epoch 39/200
20/20 [==============================] - 0s 3ms/step - loss: 692661.6250 - val_loss: 2591635.7500
Epoch 40/200
20/20 [==============================] - 0s 3ms/step - loss: 692586.9375 - val_loss: 2597881.5000
Epoch 41/200
20/20 [==============================] - 0s 3ms/step - loss: 691029.4375 - val_loss: 2589555.2500
Epoch 42/200
20/20 [==============================] - 0s 3ms/step - loss: 687953.3750 - val_loss: 2598372.2500
Epoch 43/200
20/20 [==============================] - 0s 3ms/step - loss: 686991.7500 - val_loss: 2590873.0000
Epoch 44/200
20/20 [==============================] - 0s 4ms/step - loss: 687180.2500 - val_loss: 2594788.7500
Epoch 45/200
20/20 [==============================] - 0s 4ms/step - loss: 687902.5625 - val_loss: 2589512.0000
Epoch 46/200
20/20 [==============================] - 0s 3ms/step - loss: 700826.1875 - val_loss: 2589521.7500
Epoch 47/200
20/20 [==============================] - 0s 3ms/step - loss: 693690.2500 - val_loss: 2621053.2500
Epoch 48/200
20/20 [==============================] - 0s 3ms/step - loss: 692660.6875 - val_loss: 2591710.2500
Epoch 49/200
20/20 [==============================] - 0s 3ms/step - loss: 691423.5625 - val_loss: 2598290.2500
Epoch 50/200
20/20 [==============================] - 0s 3ms/step - loss: 687500.0625 - val_loss: 2590712.5000
Epoch 51/200
20/20 [==============================] - 0s 3ms/step - loss: 696783.7500 - val_loss: 2595452.7500
Epoch 52/200
20/20 [==============================] - 0s 3ms/step - loss: 705046.1875 - val_loss: 2589513.2500
Epoch 53/200
20/20 [==============================] - 0s 3ms/step - loss: 690779.0000 - val_loss: 2594258.5000
Epoch 54/200
20/20 [==============================] - 0s 3ms/step - loss: 688042.0000 - val_loss: 2591076.0000
Epoch 55/200
20/20 [==============================] - 0s 3ms/step - loss: 686111.8750 - val_loss: 2591099.7500
Epoch 56/200
20/20 [==============================] - 0s 3ms/step - loss: 694474.8125 - val_loss: 2627198.2500
Epoch 57/200
20/20 [==============================] - 0s 6ms/step - loss: 699835.0625 - val_loss: 2590142.2500
Epoch 58/200
20/20 [==============================] - 0s 3ms/step - loss: 689788.2500 - val_loss: 2599953.5000
Epoch 59/200
20/20 [==============================] - 0s 3ms/step - loss: 692140.2500 - val_loss: 2593183.5000
Epoch 60/200
20/20 [==============================] - 0s 3ms/step - loss: 690963.1875 - val_loss: 2599678.2500
Epoch 61/200
20/20 [==============================] - 0s 3ms/step - loss: 690102.1250 - val_loss: 2595844.2500
Epoch 62/200
20/20 [==============================] - 0s 3ms/step - loss: 696507.7500 - val_loss: 2592126.0000
Epoch 63/200
20/20 [==============================] - 0s 3ms/step - loss: 695925.9375 - val_loss: 2606123.2500
Epoch 64/200
20/20 [==============================] - 0s 3ms/step - loss: 689551.5000 - val_loss: 2590813.0000
Epoch 65/200
20/20 [==============================] - 0s 4ms/step - loss: 690185.0625 - val_loss: 2591217.7500
```

{'verbose': 1, 'epochs': 200, 'steps': 20}

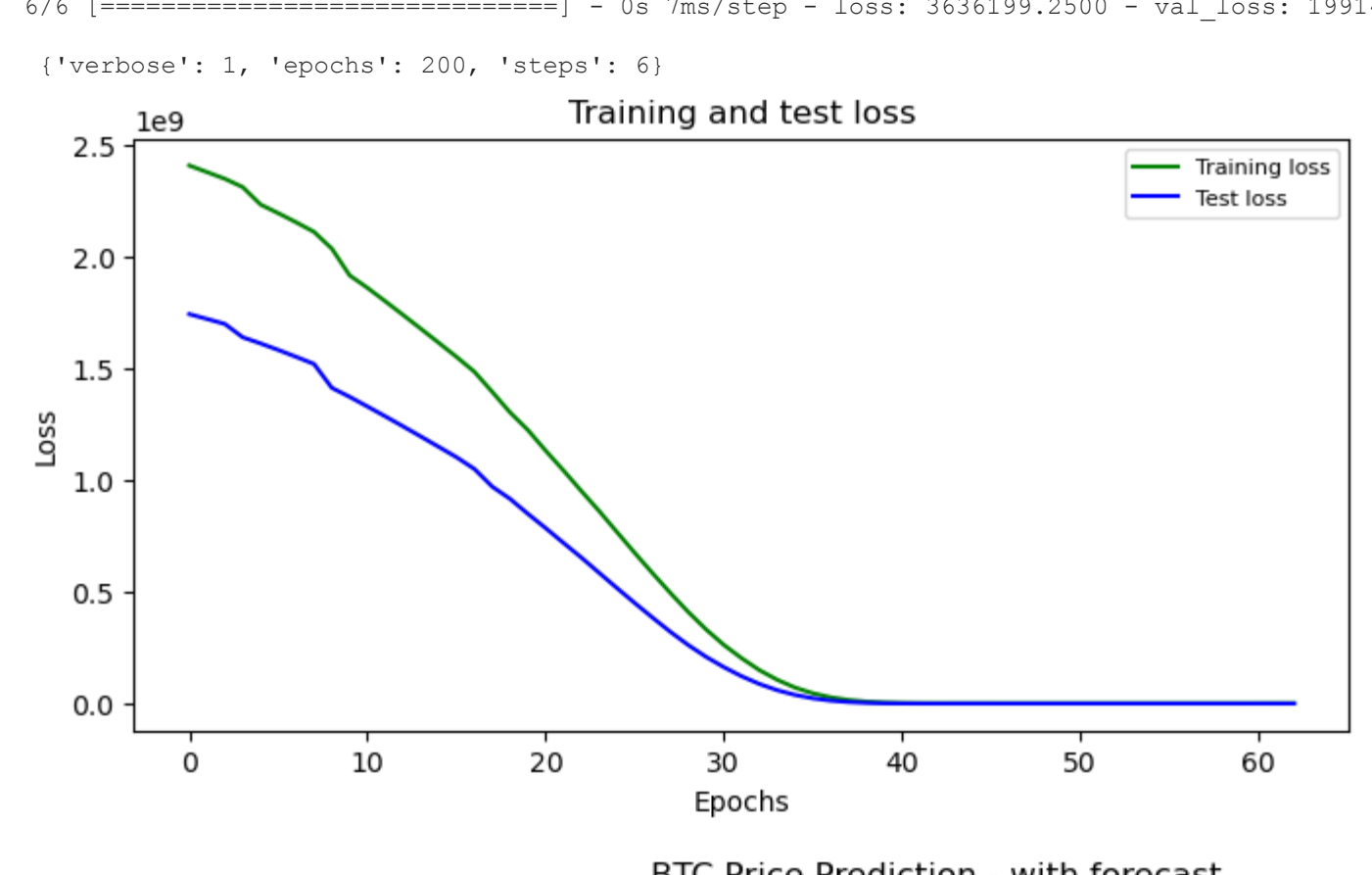


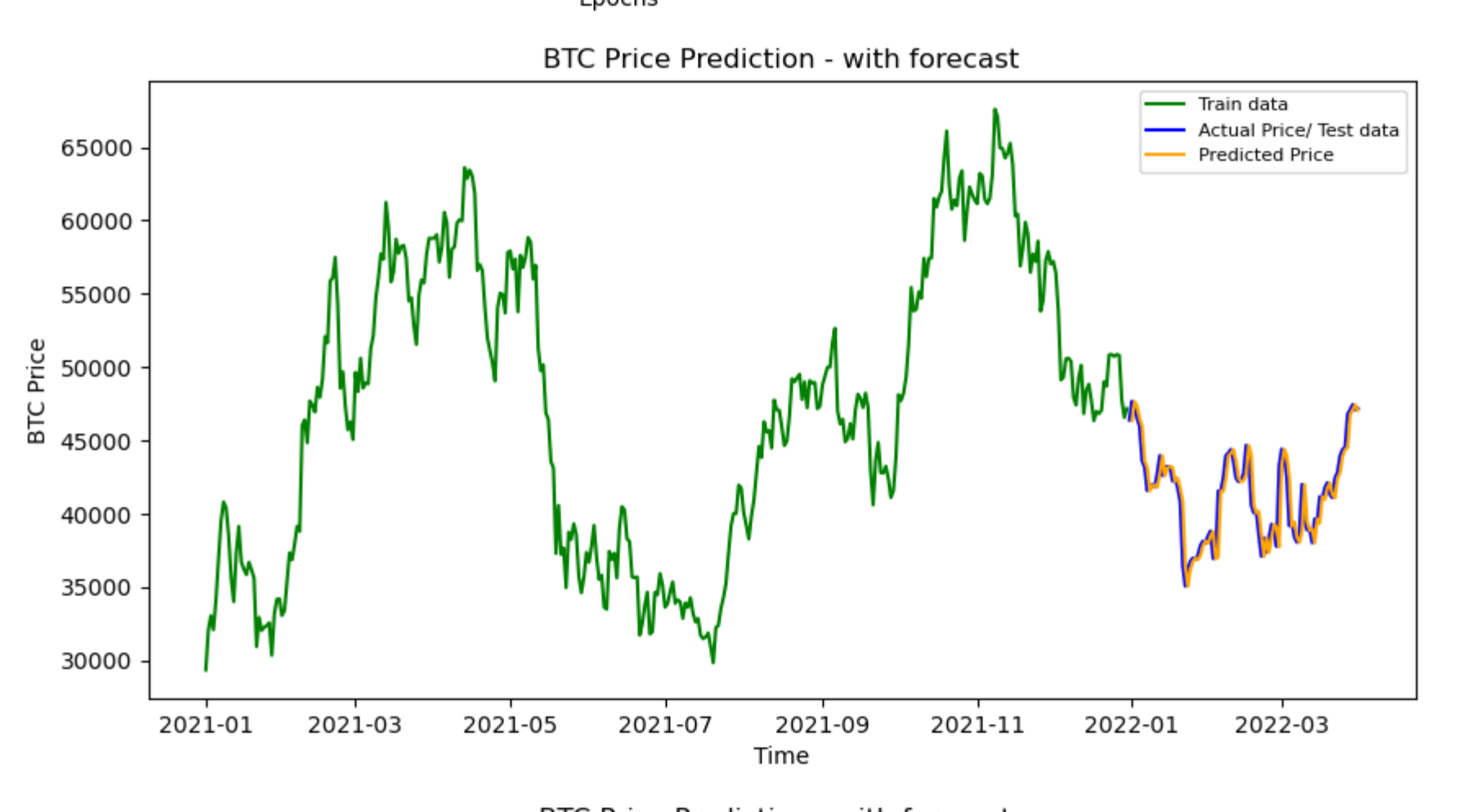


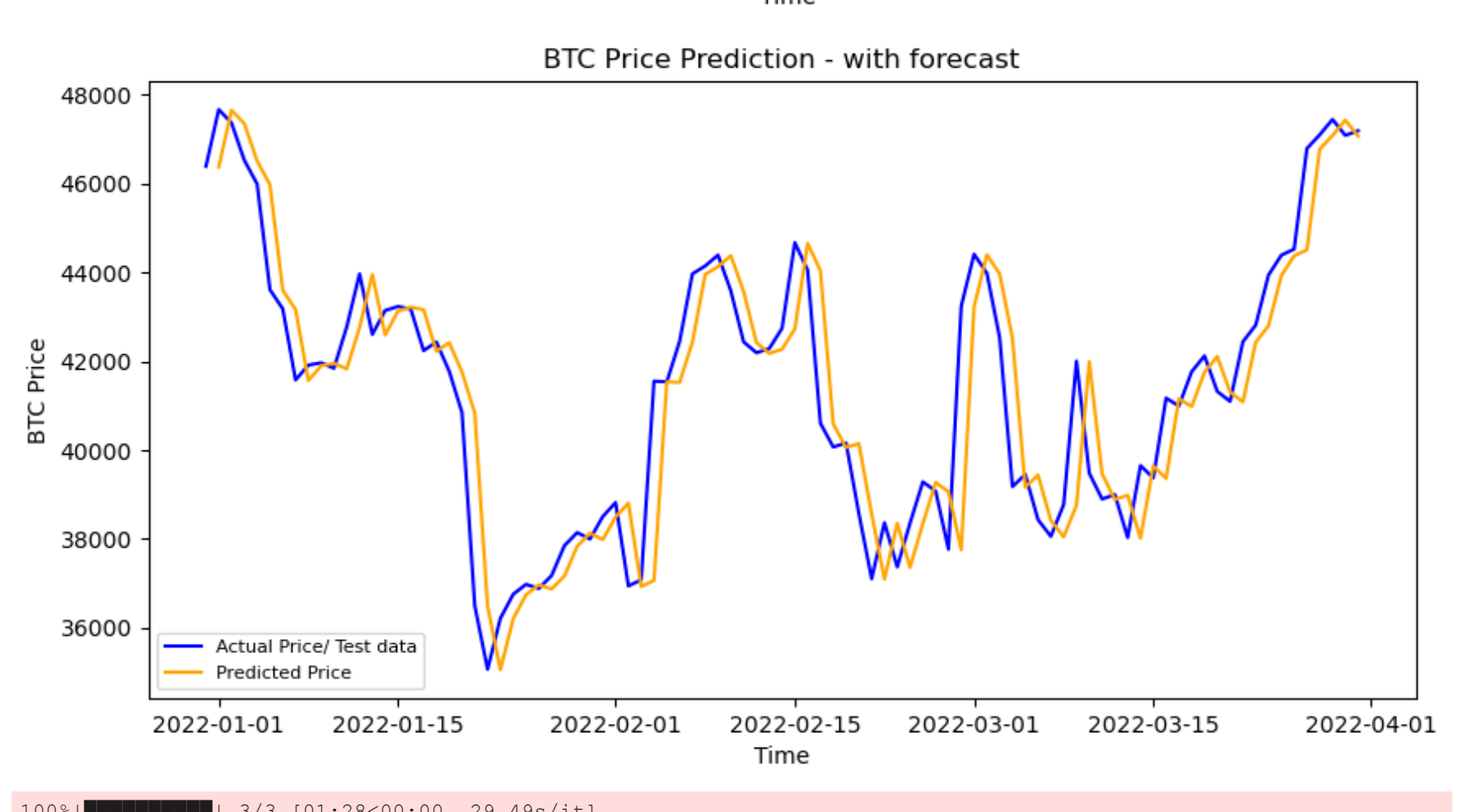



```
 67%|██████▋   | 2/3 [01:18<00:36, 36.39s/it]
Sequential variant 2 : RMSE = 1609.196


Data variant no: 3

Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_2 (LSTM)               (None, 50)                10400

 dense_2 (Dense)             (None, 1)                 51

=================================================================
Total params: 10,451
Trainable params: 10,451
Non-trainable params: 0
_________________________________________________________________
Epoch 1/200
6/6 [==============================] - 2s 58ms/step - loss: 2406541056.0000 - val_loss: 1742250880.0000
Epoch 2/200
6/6 [==============================] - 0s 9ms/step - loss: 2376730368.0000 - val_loss: 1720464128.0000
Epoch 3/200
6/6 [==============================] - 0s 9ms/step - loss: 2346829824.0000 - val_loss: 1698168064.0000
Epoch 4/200
6/6 [==============================] - 0s 7ms/step - loss: 2309859584.0000 - val_loss: 1638621184.0000
Epoch 5/200
6/6 [==============================] - 0s 24ms/step - loss: 2231984896.0000 - val_loss: 1611249536.0000
Epoch 6/200
6/6 [==============================] - 0s 23ms/step - loss: 2193039616.0000 - val_loss: 1581871360.0000
Epoch 7/200
6/6 [==============================] - 0s 7ms/step - loss: 2152512512.0000 - val_loss: 1551130752.0000
Epoch 8/200
6/6 [==============================] - 0s 7ms/step - loss: 2109886208.0000 - val_loss: 1519478912.0000
Epoch 9/200
6/6 [==============================] - 0s 8ms/step - loss: 2034972032.0000 - val_loss: 1412771712.0000
Epoch 10/200
6/6 [==============================] - 0s 7ms/step - loss: 1915374592.0000 - val_loss: 1372435456.0000
Epoch 11/200
6/6 [==============================] - 0s 7ms/step - loss: 1859481984.0000 - val_loss: 1329360896.0000
Epoch 12/200
6/6 [==============================] - 0s 7ms/step - loss: 1799616768.0000 - val_loss: 1285274240.0000
Epoch 13/200
6/6 [==============================] - 0s 7ms/step - loss: 1738410496.0000 - val_loss: 1240472064.0000
Epoch 14/200
6/6 [==============================] - 0s 7ms/step - loss: 1676503296.0000 - val_loss: 1194999296.0000
Epoch 15/200
6/6 [==============================] - 0s 7ms/step - loss: 1614053248.0000 - val_loss: 1148998784.0000
Epoch 16/200
6/6 [==============================] - 0s 7ms/step - loss: 1550915584.0000 - val_loss: 1102609280.0000
Epoch 17/200
6/6 [==============================] - 0s 7ms/step - loss: 1484721408.0000 - val_loss: 1050266560.0000
Epoch 18/200
6/6 [==============================] - 0s 7ms/step - loss: 1394850048.0000 - val_loss: 971326592.0000
Epoch 19/200
6/6 [==============================] - 0s 7ms/step - loss: 1303211392.0000 - val_loss: 916793728.0000
Epoch 20/200
6/6 [==============================] - 0s 7ms/step - loss: 1224410752.0000 - val_loss: 849909056.0000
Epoch 21/200
6/6 [==============================] - 0s 7ms/step - loss: 1132054912.0000 - val_loss: 785906240.0000
Epoch 22/200
6/6 [==============================] - 0s 7ms/step - loss: 1043529344.0000 - val_loss: 719840384.0000
Epoch 23/200
6/6 [==============================] - 0s 8ms/step - loss: 951862272.0000 - val_loss: 654435392.0000
Epoch 24/200
6/6 [==============================] - 0s 7ms/step - loss: 862447680.0000 - val_loss: 586162688.0000
Epoch 25/200
6/6 [==============================] - 0s 7ms/step - loss: 768465280.0000 - val_loss: 518053056.0000
Epoch 26/200
6/6 [==============================] - 0s 7ms/step - loss: 675073088.0000 - val_loss: 451054400.0000
Epoch 27/200
6/6 [==============================] - 0s 7ms/step - loss: 584338112.0000 - val_loss: 385810304.0000
Epoch 28/200
6/6 [==============================] - 0s 7ms/step - loss: 496032896.0000 - val_loss: 322248576.0000
Epoch 29/200
6/6 [==============================] - 0s 7ms/step - loss: 411267840.0000 - val_loss: 263458816.0000
Epoch 30/200
6/6 [==============================] - 0s 7ms/step - loss: 333331968.0000 - val_loss: 210699632.0000
Epoch 31/200
6/6 [==============================] - 0s 7ms/step - loss: 263885248.0000 - val_loss: 164516128.0000
Epoch 32/200
6/6 [==============================] - 0s 7ms/step - loss: 204367648.0000 - val_loss: 123666344.0000
Epoch 33/200
6/6 [==============================] - 0s 7ms/step - loss: 150956528.0000 - val_loss: 89171720.0000
Epoch 34/200
6/6 [==============================] - 0s 7ms/step - loss: 107774632.0000 - val_loss: 61364736.0000
Epoch 35/200
6/6 [==============================] - 0s 7ms/step - loss: 72979552.0000 - val_loss: 40344540.0000
Epoch 36/200
6/6 [==============================] - 0s 7ms/step - loss: 47649080.0000 - val_loss: 25121320.0000
Epoch 37/200
6/6 [==============================] - 0s 7ms/step - loss: 29478106.0000 - val_loss: 14895681.0000
Epoch 38/200
6/6 [==============================] - 0s 7ms/step - loss: 17680840.0000 - val_loss: 8505651.0000
Epoch 39/200
6/6 [==============================] - 0s 7ms/step - loss: 10321969.0000 - val_loss: 4914568.5000
Epoch 40/200
6/6 [==============================] - 0s 7ms/step - loss: 6578614.5000 - val_loss: 3072978.7500
Epoch 41/200
6/6 [==============================] - 0s 7ms/step - loss: 4611985.5000 - val_loss: 2294950.5000
Epoch 42/200
6/6 [==============================] - 0s 7ms/step - loss: 3911192.0000 - val_loss: 2033022.6250
Epoch 43/200
6/6 [==============================] - 0s 7ms/step - loss: 3667855.5000 - val_loss: 1987977.3750
Epoch 44/200
6/6 [==============================] - 0s 7ms/step - loss: 3628287.0000 - val_loss: 2000634.1250
Epoch 45/200
6/6 [==============================] - 0s 7ms/step - loss: 3648976.0000 - val_loss: 2013490.2500
Epoch 46/200
6/6 [==============================] - 0s 7ms/step - loss: 3654444.0000 - val_loss: 1994237.8750
Epoch 47/200
6/6 [==============================] - 0s 7ms/step - loss: 3633140.2500 - val_loss: 1988003.3750
Epoch 48/200
6/6 [==============================] - 0s 7ms/step - loss: 3642297.0000 - val_loss: 1989075.0000
Epoch 49/200
6/6 [==============================] - 0s 7ms/step - loss: 3639125.7500 - val_loss: 1989478.3750
Epoch 50/200
6/6 [==============================] - 0s 7ms/step - loss: 3642774.2500 - val_loss: 1989981.7500
Epoch 51/200
6/6 [==============================] - 0s 7ms/step - loss: 3634910.7500 - val_loss: 1990086.3750
Epoch 52/200
6/6 [==============================] - 0s 7ms/step - loss: 3639421.5000 - val_loss: 1991508.6250
Epoch 53/200
6/6 [==============================] - 0s 7ms/step - loss: 3642399.5000 - val_loss: 1990324.2500
Epoch 54/200
6/6 [==============================] - 0s 7ms/step - loss: 3643428.0000 - val_loss: 1988339.2500
Epoch 55/200
6/6 [==============================] - 0s 7ms/step - loss: 3636742.2500 - val_loss: 1989301.1250
Epoch 56/200
6/6 [==============================] - 0s 8ms/step - loss: 3635615.5000 - val_loss: 1988042.5000
Epoch 57/200
6/6 [==============================] - 0s 7ms/step - loss: 3639235.0000 - val_loss: 1989911.1250
Epoch 58/200
6/6 [==============================] - 0s 7ms/step - loss: 3635455.0000 - val_loss: 1988096.7500
Epoch 59/200
6/6 [==============================] - 0s 7ms/step - loss: 3644811.2500 - val_loss: 1988354.0000
Epoch 60/200
6/6 [==============================] - 0s 7ms/step - loss: 3646810.0000 - val_loss: 1990220.1250
Epoch 61/200
6/6 [==============================] - 0s 7ms/step - loss: 3644608.2500 - val_loss: 2044723.2500
Epoch 62/200
6/6 [==============================] - 0s 7ms/step - loss: 3686775.2500 - val_loss: 2009815.5000
Epoch 63/200
6/6 [==============================] - 0s 7ms/step - loss: 3636199.2500 - val_loss: 1991448.0000
```

{'verbose': 1, 'epochs': 200, 'steps': 6}








```
100%|██████████| 3/3 [01:28<00:00, 29.49s/it]
Sequential variant 3 : RMSE = 1409.957

Wall time: 1min 28s
```

7.4. LSTM => summary

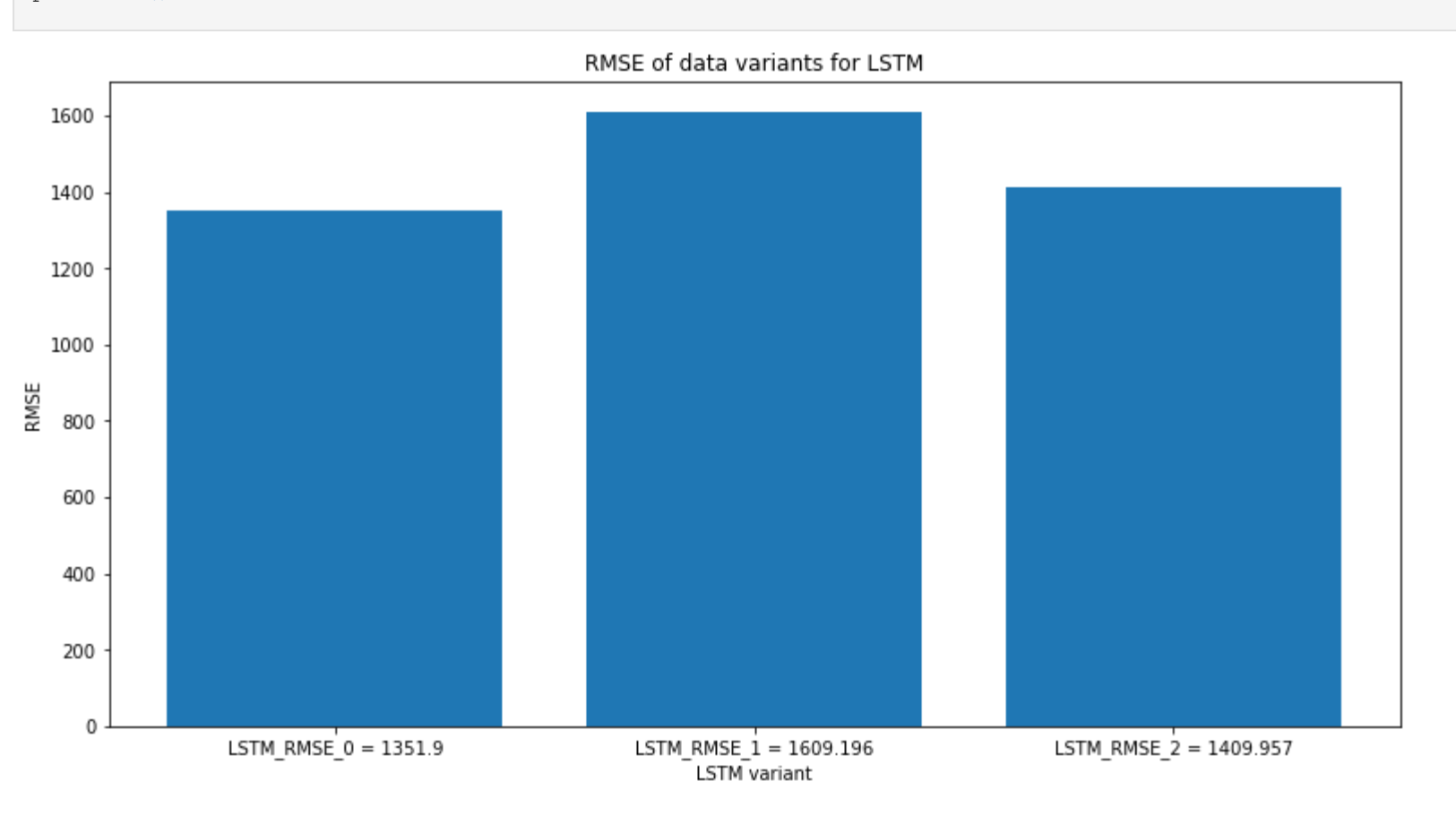
The bitcoin price forecast based on historical data was made using the LSTM model. The model parameters were selected empirically and applied equally to data variants.

- In first try, complete dataset was used to tran the model.
- In second try, dataset was sliced, started from 2018-01-01.
- In third try, dataset was sliced, started from 2021-01-01.

In coparison of model performence of different data ranges, the best performence was for the complete data, then to past year + and at the end, to period of time around four years.

In [13]:

```python
# plot for the RMSE's of data variants for LSTM

LSTM_variant = [("LSTM_RMSE_" + str(x)) for x in range(len(lstm_variants_rmse))]
summary = []

for variant, result in zip(LSTM_variant, lstm_variants_rmse):
    summary.append(variant + " = " + str(result))

fig = plt.figure(figsize=(10,5))
ax = fig.add_axes([0,0,1,1])
ax.bar(summary, lstm_variants_rmse)

plt.xlabel("LSTM variant")
plt.ylabel("RMSE")
plt.title("RMSE of data variants for LSTM")
plt.show()
```



8. Project summary

This notebook was created to predict the Bitcoin price (in USD) using the LSTM model, along with notebooks for Bitcoin price prediction using ARIMA, XGBoost models. Each model used a the same three data ranges to generate forecasts.

The variant of the ARIMA model and LSTM model have similar approach for forecasting. Both look at close price of one past day for prediction of the close price of the next day. Performence of the models, measured by RMSE is also similar in every data range.

XGBRegressor variant have good results but a different approach in comparison of other models. For forecasting, columns "market_cap", "volume" has been used as well. The model is using GridSearchCV and "eval_set" on the fit () methodand L2 penalty specified by the author of the project.

In conclusion - the LSTM was the best model used to predict Bitcoin's price. This model gave relatively acceptable results with a short training time, regardless of the data period.