https://colab.research.google.com/drive/1wWvtA5RC6-is6J8W86wzK52Knr3N1Xbm#scrollTo=-2xKfxVwY4NI

import os

import numpy as np

import tensorflow as tf

from tensorflow import keras

import pandas as pd

import seaborn as sns

from pylab import rcParams

import matplotlib.pyplot as plt

from matplotlib import rc

from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.layers import Bidirectional, Dropout, Activation, Dense, LSTM

from tensorflow.python.keras.layers import CuDNNLSTM

from tensorflow.keras.models import Sequential

%matplotlib inline

sns.set(style='whitegrid', palette='muted', font\_scale=1.5)

rcParams['figure.figsize'] = 14, 8

RANDOM\_SEED = 42

np.random.seed(RANDOM\_SEED)

# Data comes from:

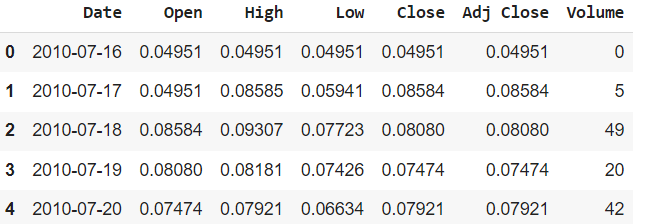
# https://finance.yahoo.com/quote/BTC-USD/history?period1=1279314000&period2=1556053200&interval=1d&filter=history&frequency=1d

csv\_path = "https://raw.githubusercontent.com/curiousily/Deep-Learning-For-Hackers/master/data/3.stock-prediction/BTC-USD.csv"

# csv\_path = "https://raw.githubusercontent.com/curiousily/Deep-Learning-For-Hackers/master/data/3.stock-prediction/AAPL.csv"

df = pd.read\_csv(csv\_path, parse\_dates=['Date'])

df = df.sort\_values('Date')



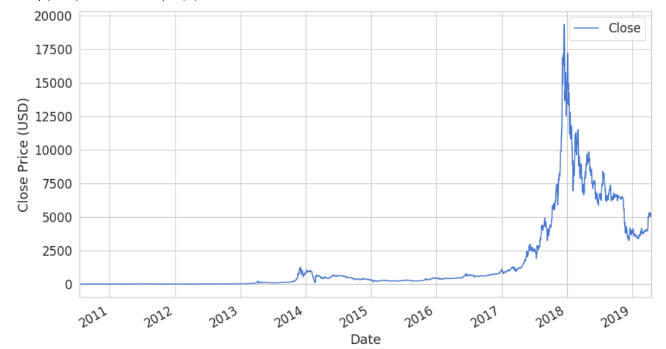
df.shape

(3201, 7)

ax = df.plot(x='Date', y='Close');

ax.set\_xlabel("Date")

ax.set\_ylabel("Close Price (USD)")



# Time Series

Temporal datasets are quite common in practice. Your energy consumption and expenditure (calories in, calories out), weather changes, stock market, analytics gathered from the users for your product/app and even your (possibly in love) heart produce Time Series.

You might be interested in a plethora of properties regarding your Time Series — **stationarity**, **seasonality** and **autocorrelation** are some of the most well known.

* **Autocorrelation** is the correlation of data points separated by some interval (known as lag).
* **Seasonality** refers to the presence of some cyclical pattern at some interval (no, it doesn’t have to be every spring).

A time series is said to be **stationarity** if it has constant mean and variance. Also, the covariance is independent of the time.

One obvious question you might ask yourself while watching at Time Series data is: “Does the value of the current time step affects the next one?” a.k.a. Time Series forecasting.

There are many approaches that you can use for this purpose. But we’ll build a Deep Neural Network that does some forecasting for us and use it to predict future Bitcoin price.

# Modeling

All models we’ve built so far do not allow for operating on sequence data. Fortunately, we can use a special class of Neural Network models known as [Recurrent Neural Networks (RNNs)](https://en.wikipedia.org/wiki/Recurrent_neural_network" \t "_blank) just for this purpose. RNNs allow using the output from the model as a new input for the same model. The process can be repeated indefinitely.

One serious limitation of RNNs is the [inability of capturing long-term dependencies](https://colah.github.io/posts/2015-08-Understanding-LSTMs/" \t "_blank) in a sequence (e.g. Is there a dependency between today`s price and that 2 weeks ago?). One way to handle the situation is by using an **Long short-term memory (LSTM)** variant of RNN.

The default [LSTM](https://en.wikipedia.org/wiki/Long_short-term_memory) behavior is remembering information for prolonged periods of time. Let’s see how you can use LSTM in Keras.

## Data preprocessing

First, we’re going to squish our price data in the range [0, 1]. Recall that this will help our optimization algorithm converge faster:

Normalizar Close Price (target)

scaler = MinMaxScaler()

close\_price = df.Close.values.reshape(-1, 1)

scaled\_close = scaler.fit\_transform(close\_price)

Verificar que no hay nulos

np.isnan(scaled\_close).any()

scaled\_close = scaled\_close[~np.isnan(scaled\_close)]

scaled\_close = scaled\_close.reshape(-1, 1)

np.isnan(scaled\_close).any()

Preprocessing

## Making sequences

LSTMs expect the data to be in 3 dimensions. We need to split the data into sequences of some preset length. The shape we want to obtain is:

[batch\_size, sequence\_length, n\_features]

SEQ\_LEN = 100

def to\_sequences(data, seq\_len):

    d = []

    for index in range(len(data) - seq\_len):

        d.append(data[index: index + seq\_len])

    return np.array(d)

def preprocess(data\_raw, seq\_len, train\_split):

    data = to\_sequences(data\_raw, seq\_len)

    num\_train = int(train\_split \* data.shape[0])

# :-1 no toma primer campo, fecha

    X\_train = data[:num\_train, :-1, :]

    y\_train = data[:num\_train, -1, :]

    X\_test = data[num\_train:, :-1, :]

    y\_test = data[num\_train:, -1, :]

    return X\_train, y\_train, X\_test, y\_test

X\_train, y\_train, X\_test, y\_test = preprocess(scaled\_close, SEQ\_LEN, train\_split = 0.95)

df.shape #(3201, 7)

X\_train.shape # (2945, 99, 1)

X\_test.shape # (156, 99, 1)

# Model

DROPOUT = 0.2

WINDOW\_SIZE = SEQ\_LEN – 1

from tensorflow.keras.layers import Bidirectional

from tensorflow.keras.layers import LSTM

model = keras.Sequential()

model.add(Bidirectional(LSTM(WINDOW\_SIZE, return\_sequences=True),

                        input\_shape=(WINDOW\_SIZE, X\_train.shape[-1])))

model.add(Dropout(rate=DROPOUT))

model.add(Bidirectional(LSTM((WINDOW\_SIZE \* 2), return\_sequences=True)))

model.add(Dropout(rate=DROPOUT))

model.add(Bidirectional(LSTM(WINDOW\_SIZE, return\_sequences=False)))

model.add(Dense(units=1))

model.add(Activation('linear'))

model.compile(

    loss='mean\_squared\_error',

    optimizer='adam'

)

BATCH\_SIZE = 64

history = model.fit(

    X\_train,

    y\_train,

    epochs=50,

    batch\_size=BATCH\_SIZE,

    shuffle=False,

    validation\_split=0.1

)

model.evaluate(X\_test, y\_test)

|  |  |
| --- | --- |
| plt.plot(history.history['loss'])  plt.plot(history.history['val\_loss'])  plt.title('model loss')  plt.ylabel('loss')  plt.xlabel('epoch')  plt.legend(['train', 'test'], loc='upper left')  plt.show() |  |

y\_hat = model.predict(X\_test)

y\_test\_inverse = scaler.inverse\_transform(y\_test)

y\_hat\_inverse = scaler.inverse\_transform(y\_hat)

|  |  |
| --- | --- |
| plt.plot(y\_test\_inverse,  label="Actual Price", color='green')  plt.plot(y\_hat\_inverse,  label="Predicted Price", color='red')    plt.title('Bitcoin price prediction')  plt.xlabel('Time [days]')  plt.ylabel('Price')  plt.legend(loc='best')    plt.show(); |  |