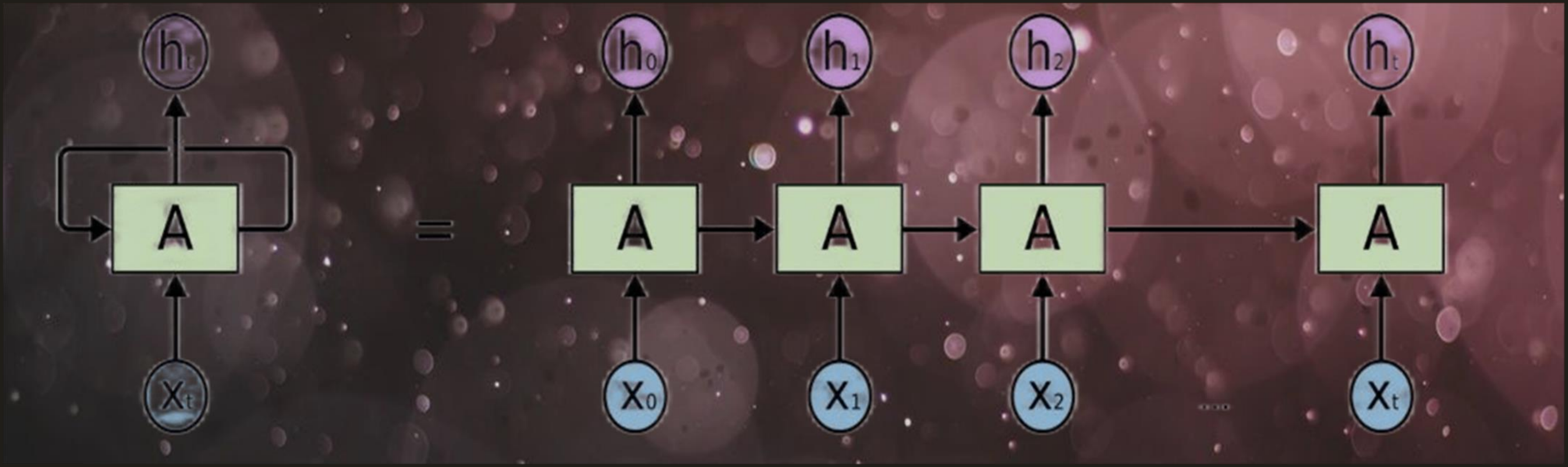
LSTM(RNN) FOR TIME SERIES DATA

ASJADULLAH(120AD0027)
SUNIL(120AD0010)

Table of content

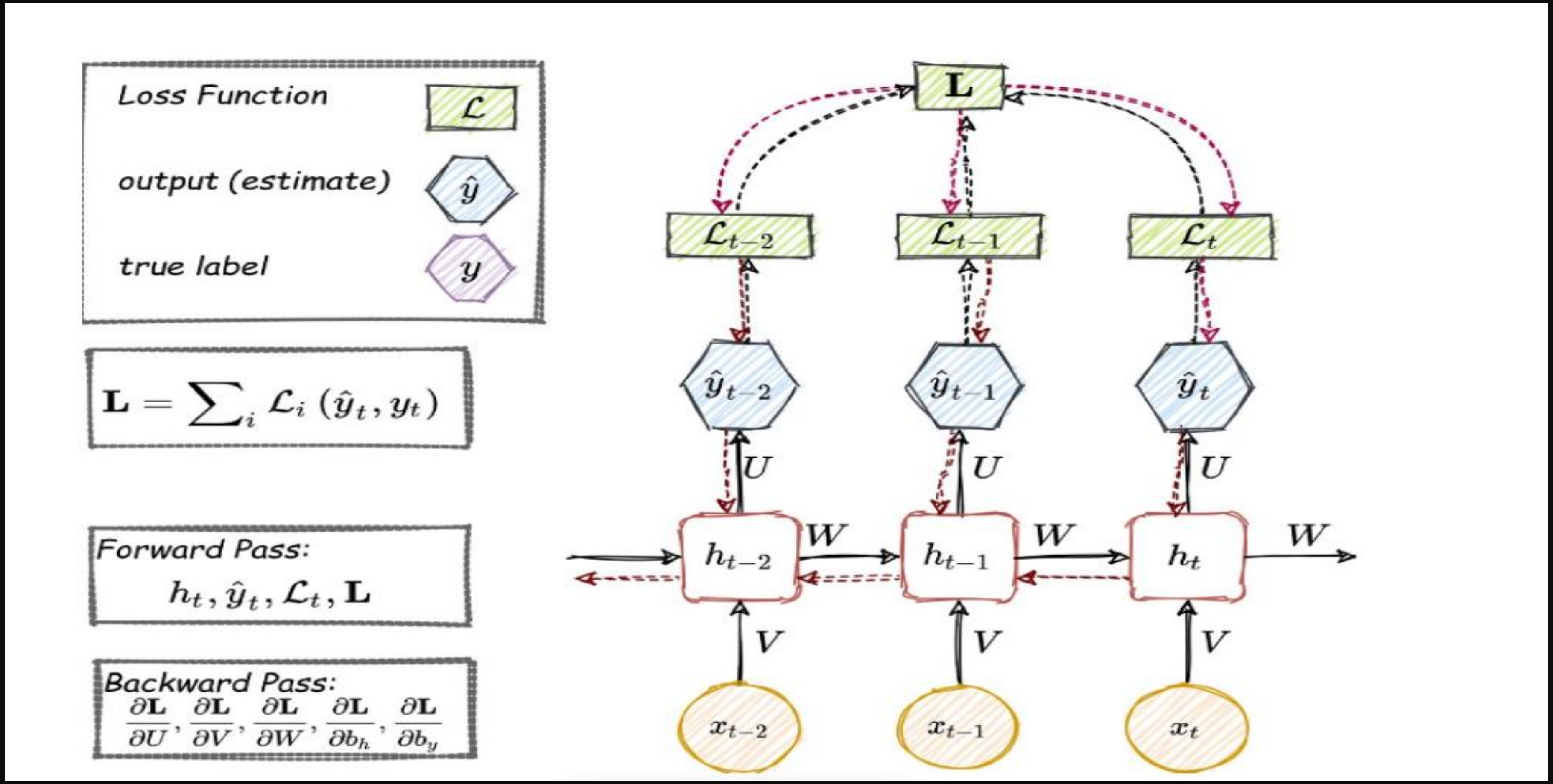
RNN

- even though RNNs are quite powerful, they suffer from Vanishing gradient problem which hinders them from using long term information, like they are good for storing memory 3-4 instances of past iterations but larger number of instances don't provide good results so we don't just use regular RNNs. Instead, we use a better variation of RNNs: Long Short Term Networks(LSTM).



Backpropagation Through Time (BPTT)

- Forward pass
- Backward pass



Loss Function $\mathcal{L}$
output (estimate) $\hat{y}$
true label y
$\mathbf{L} = \sum_i \mathcal{L}_i(\hat{y}_t, y_t)$
Forward Pass: $h_t, \hat{y}_t, \mathcal{L}_t, \mathbf{L}$
Backward Pass: $\frac{\partial \mathbf{L}}{\partial U}, \frac{\partial \mathbf{L}}{\partial V}, \frac{\partial \mathbf{L}}{\partial W}, \frac{\partial \mathbf{L}}{\partial b_h}, \frac{\partial \mathbf{L}}{\partial b_y}$
$\mathbf{L}$
$\mathcal{L}_{t-2}$
$\mathcal{L}_{t-1}$
$\mathcal{L}_t$
$\hat{y}_{t-2}$
$\hat{y}_{t-1}$
$\hat{y}_t$
U
W
h_{t-2}
h_{t-1}
h_t
V
x_{t-2}
x_{t-1}
x_t

Backpropagation Through Time (BPTT)

- The training of RNN is not trivial, as we backpropagate gradients through layers and also through time. Hence, in each time step we have to sum up all the previous contributions until the current one, as given in the equation:
- Each of the neural network's weights receives an update proportional to the partial derivative of the error function with respect to the current weight in each iteration of training

$$\frac{\partial \mathbf{L}}{\partial W} = \sum_{i=0}^{T} \frac{\partial \mathcal{L}_i}{\partial W} \propto \sum_{i=0}^{T} \left(\prod_{i=k+1}^{y} \frac{\partial h_i}{\partial h_{i-1}} \right) \frac{\partial h_k}{\partial W}$$

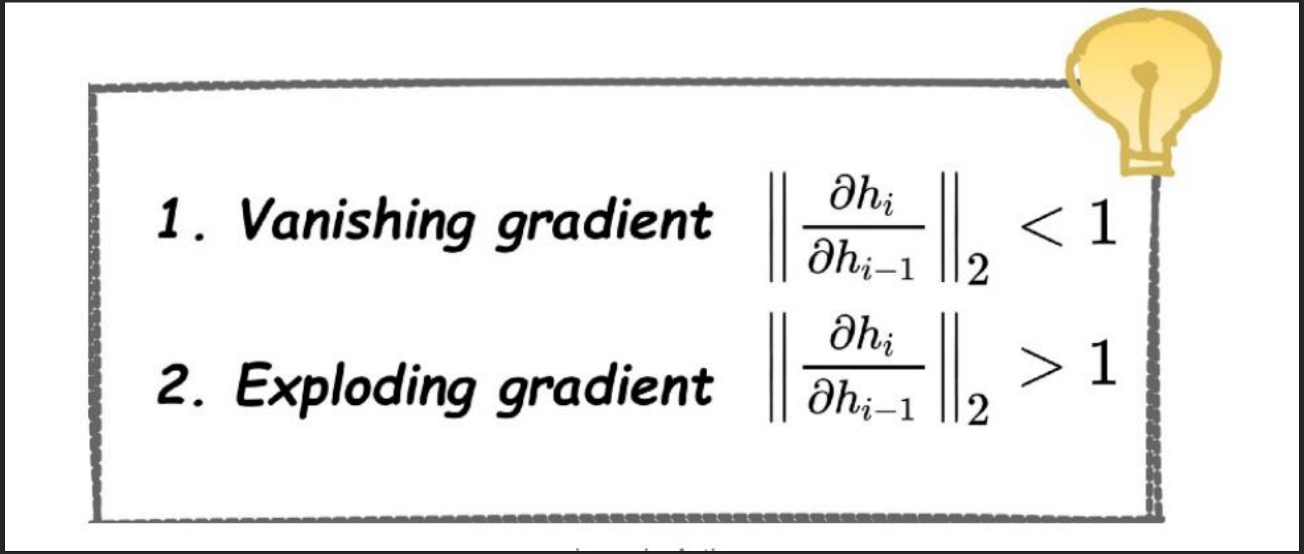
- the contribution of a state at time step k to the gradient of the entire loss function L, at time step t=T is calculated. The challenge during the training is in the ratio of the hidden state:

$$\frac{\partial \mathbf{L}}{\partial W} \propto \sum_{i=0}^{T} \left(\prod_{i=k+1}^{y} \frac{\partial h_i}{\partial h_{i-1}} \right) \frac{\partial h_k}{\partial W}$$

What is Vanishing Gradient problem?

- Vanishing gradient problem is a difficulty found in training artificial neural networks with gradient-based learning methods and backpropagation.

- As more layers using certain activation functions are added to neural networks, the gradients of the loss function approaches zero, making the network hard to train.



1. Vanishing gradient $\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 < 1$

2. Exploding gradient $\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 > 1$

The Problem of Long-Term Dependencies

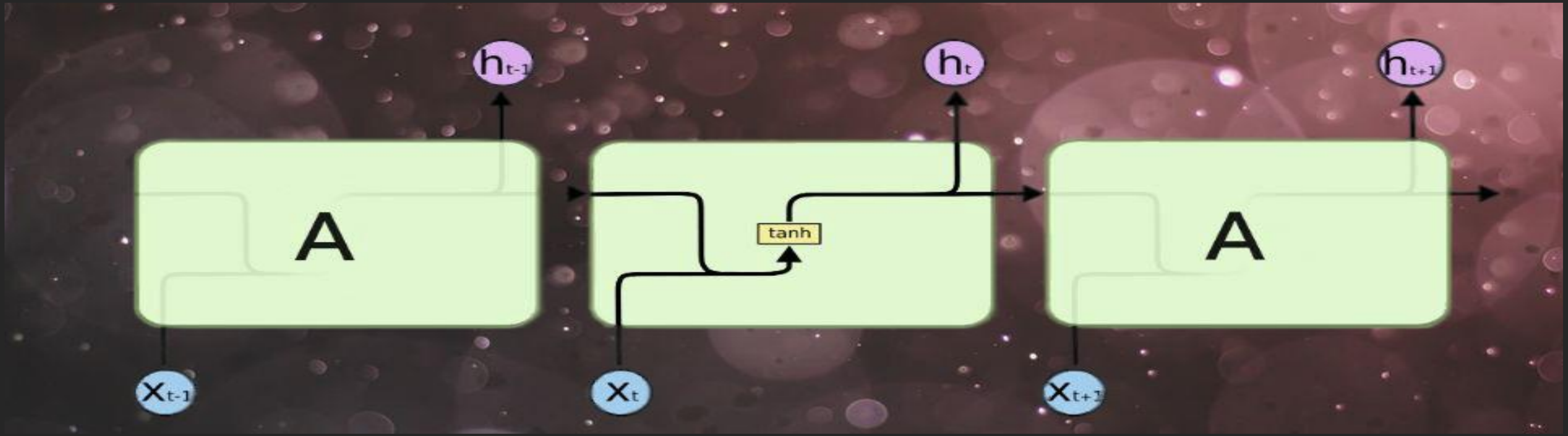
- Sometimes, we only need to look at recent information to perform the present task. For example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word in "the clouds are in the *sky*," we don't need any further context – it's pretty obvious the next word is going to be sky. In such cases, where the gap between the relevant information and the place that it's needed is small, RNNs can learn to use the past information.
- But there are also cases where we need more context. Consider trying to predict the last word in the text "I grew up in France… I speak fluent *French*." Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It's entirely possible for the gap between the relevant information and the point where it is needed to become very large.
- Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.

Solution

- Initialize the weight matrix, W, with an orthogonal matrix, and use this throughout the entire training (multiplications of orthogonal matrices don’t explode or vanish).
- have Long Short-Term Memory Networks (LSTMs).
- Using ReLU activation function.

LSTM

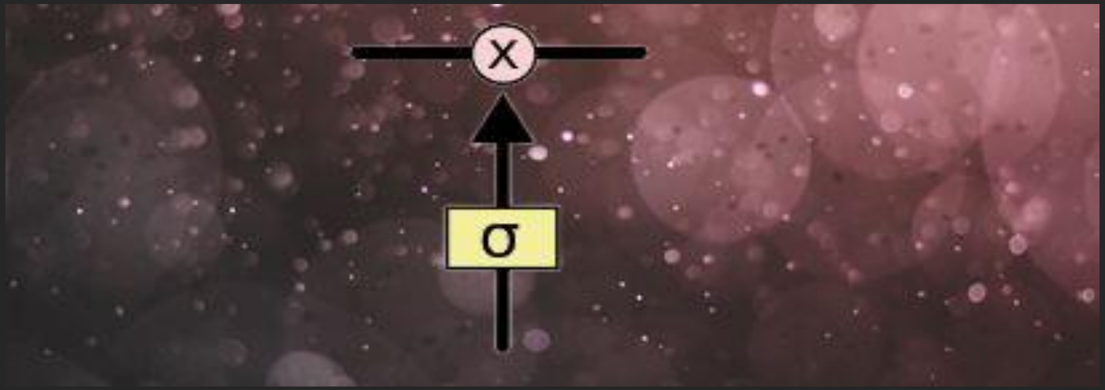
- standard RNN :



- The hidden layer in the central block receives input xt from the input layer and also from itself in time point t-1, then it generates output ht and also another input for itself but in time point t+1.

The Core Idea Behind LSTMs

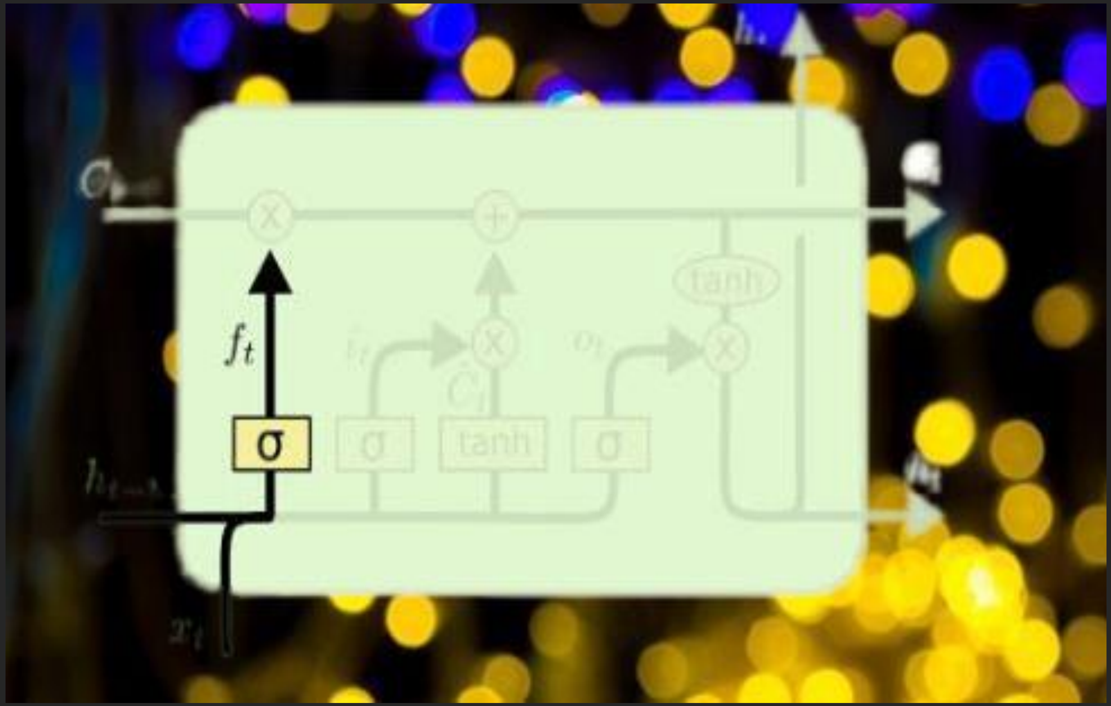
- The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.
- Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.



- The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means “let nothing through,” while a value of one means “let everything through!”
- An LSTM has three of these gates, to protect and control the cell state.

Step-by-Step LSTM Walk Through

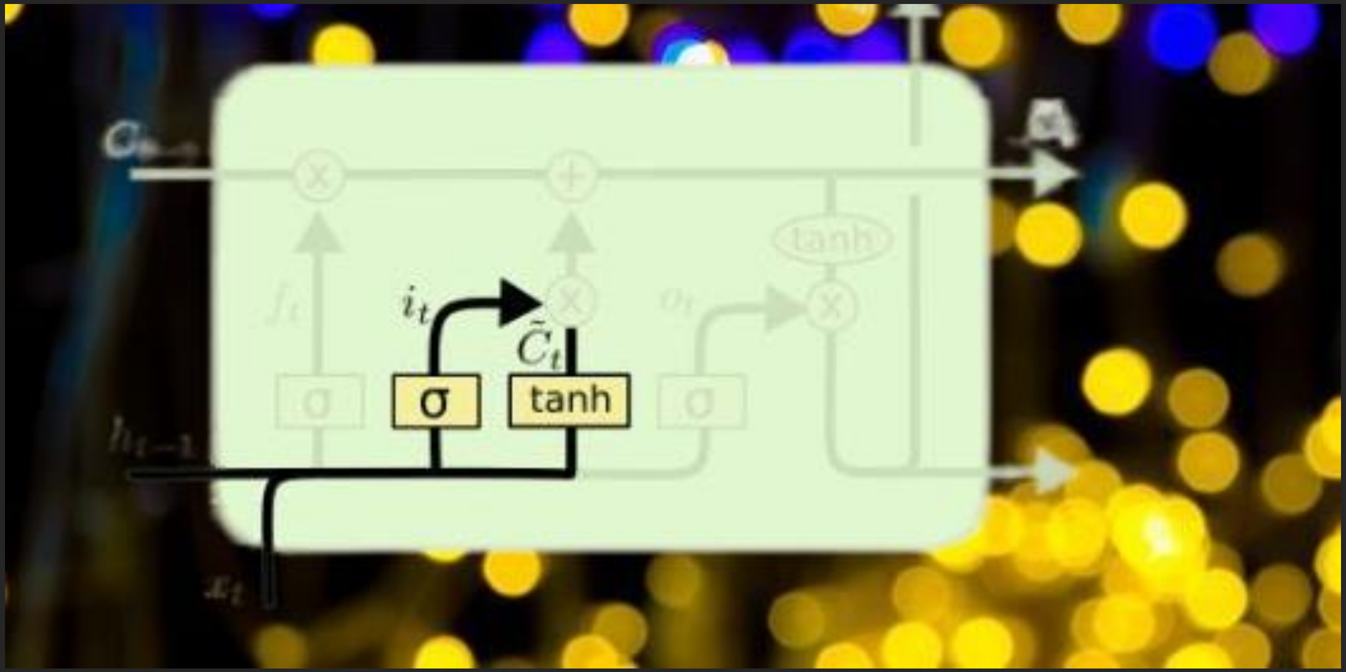
- The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at **ht−1 and xt**, and outputs a number between 0 and 1 for each number in the cell state **Ct−1.** 1 represents "**completely keep this**" while a 0 represents "**completely get rid of this.**"



$$f_t = \sigma\left(w_f[h_{t-1}, x_t] + b_f\right)$$

Step-by-Step LSTM Walk Through

- The next step is to decide what new information we’re going to store in the cell state. This has two parts. First, a sigmoid layer called the “input gate layer” decides which values we’ll update. Next, a tanh layer creates a vector of new candidate values, C~t, that could be added to the state. In the next step, we’ll combine these two to create an update to the state.

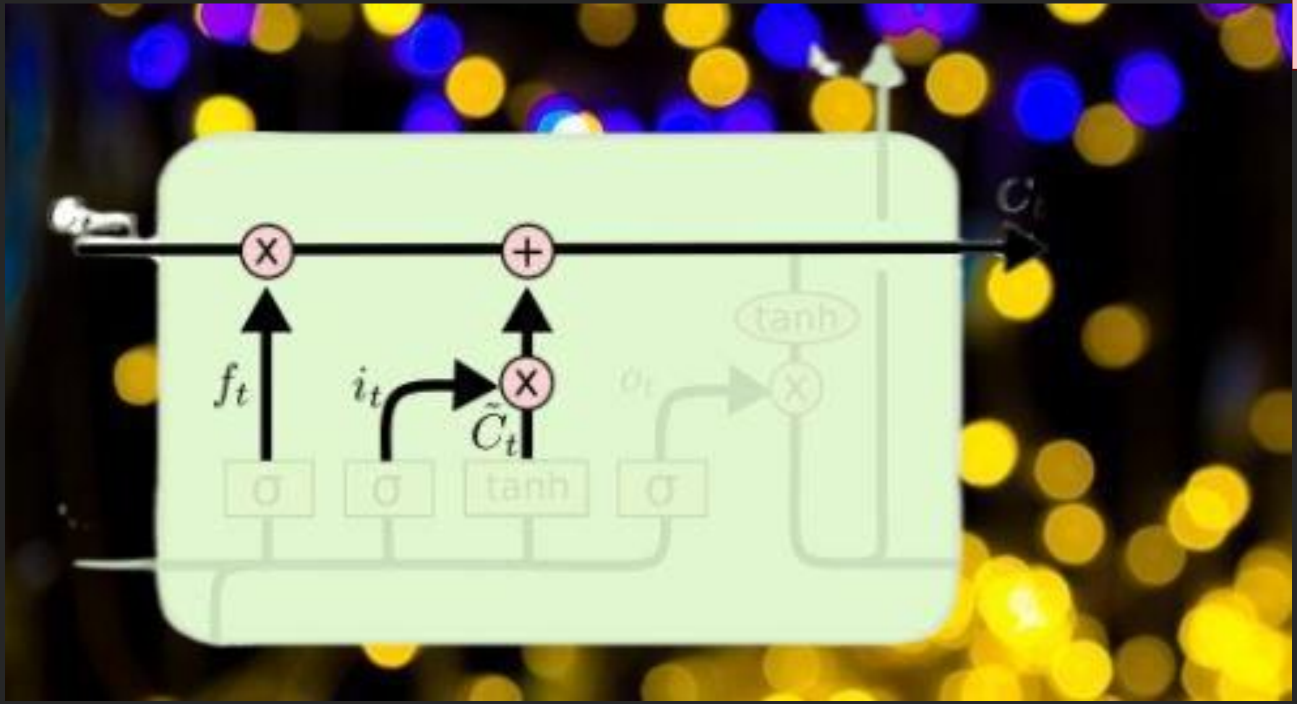


$$i_t = \sigma\left(w_j \cdot [h_{t-1}, x_t] + b_i\right)$$

$$\bar{C_t} = \tanh(w_C \cdot [h_t - 1, x_t] + b_c)$$

Step-by-Step LSTM Walk Through

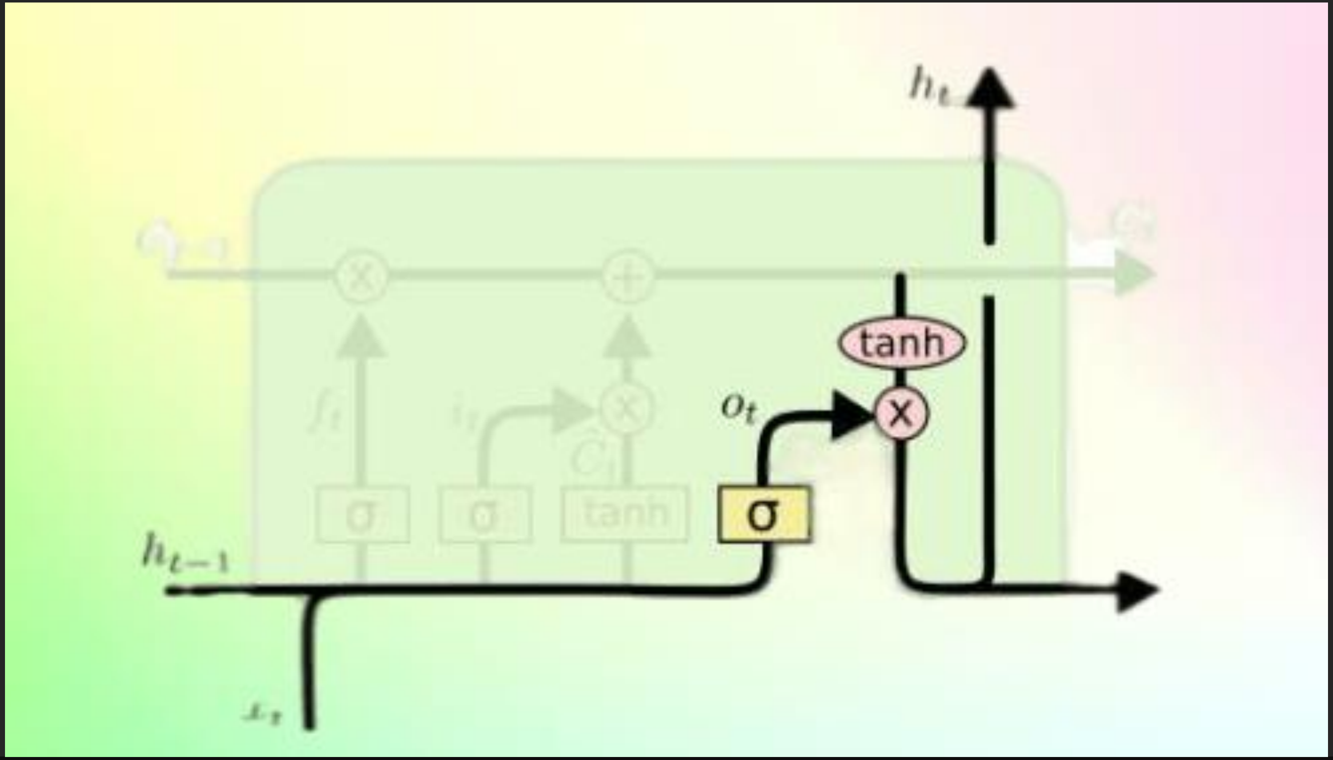
- It’s now time to update the old cell state, Ct−1, into the new cell state Ct. The previous steps already decided what to do, we just need to actually do it.We multiply the old state by ft, forgetting the things we decided to forget earlier. Then we add it∗C~t. This is the new candidate values, scaled by how much we decided to update each state value.



$$C_t = f_t \cdot C_{t-1} + i_t \cdot \bar{C}_t$$

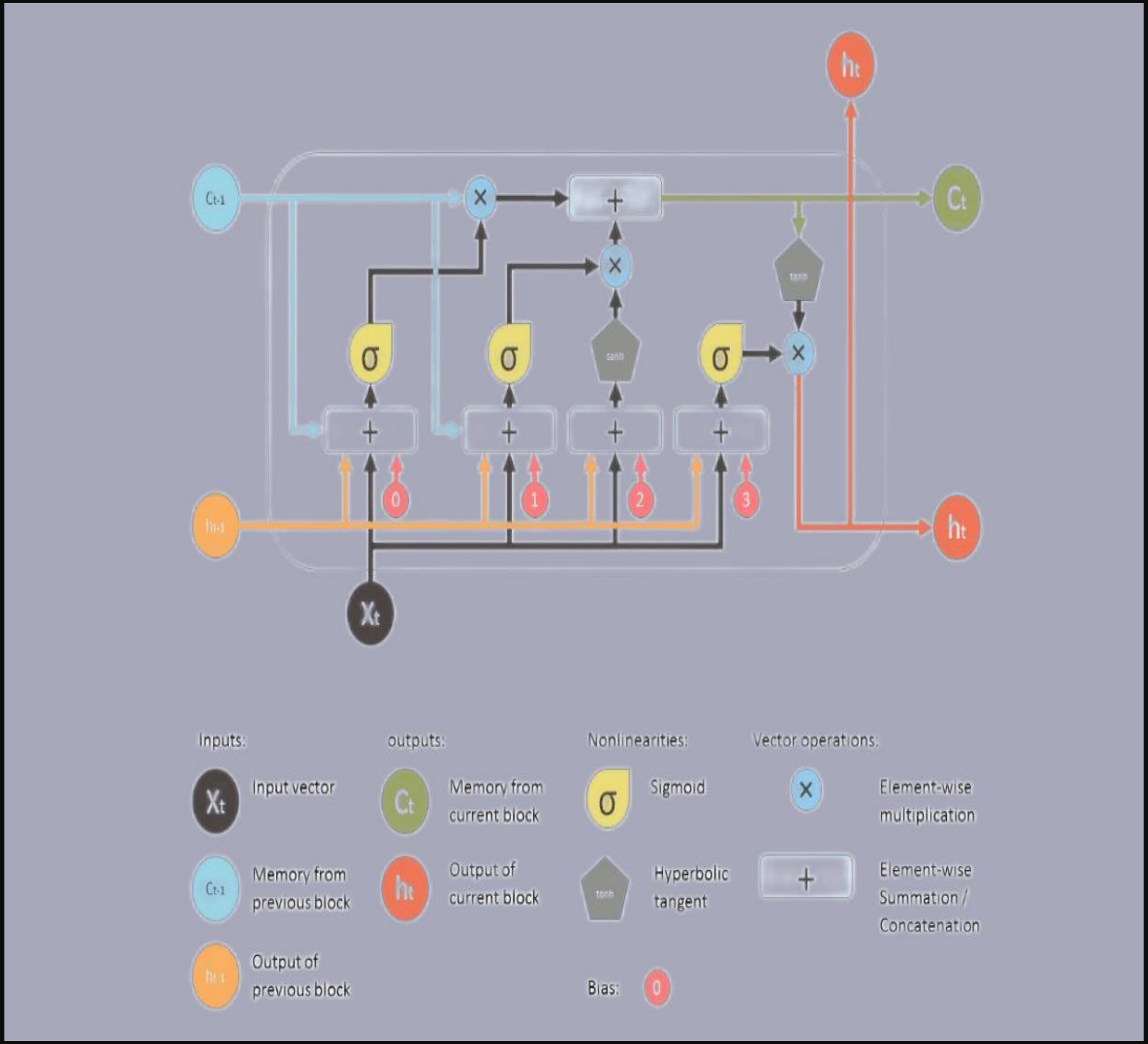
Step-by-Step LSTM Walk Through

- Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.



$$O_t = \sigma(w_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = O_t \cdot \tan h(c_t)$$

LSTM SUMMARY

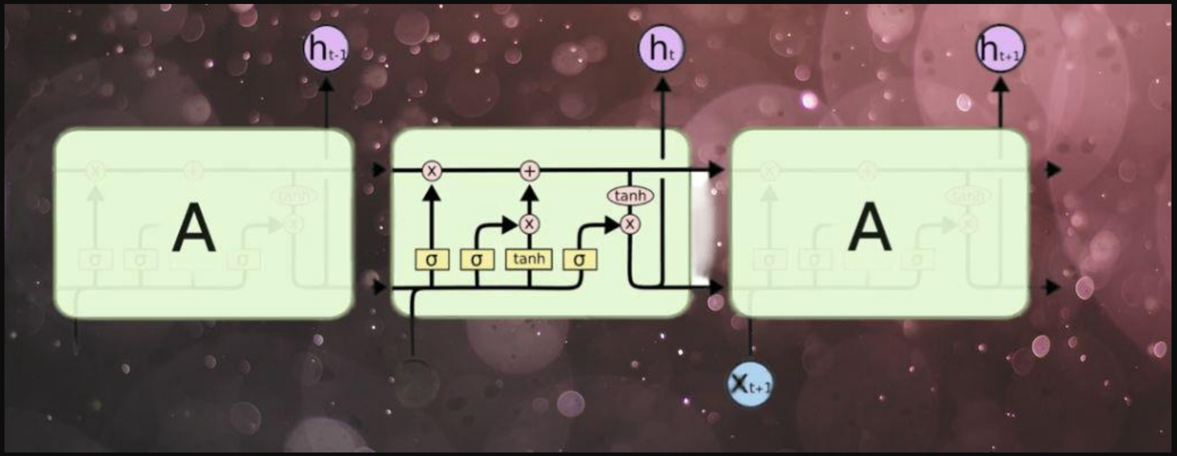


$$O_t = \sigma(w_o \cdot [h_{t\_1}, x_t] + b_o)$$
$$h_t = O_t \cdot \tan h(c_t)$$

$$f_t = \sigma(w_f [h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(w_j \cdot [h_{t-1}, x_t] + b_i)$$
$$\bar{C}_t = \tanh(w_C \cdot [h_t - 1, x_t] + b_c)$$

$$C_t = f_t \cdot C_{t-1} + \dot{i}_t \cdot \bar{C}_t$$



Code & Implementation

Univariate LSTM Models

- Vanilla LSTM
- Stacked LSTM
- Bidirectional LSTM
- CNN LSTM
- ConvLSTM

Multivariate LSTM Models

- Multiple Input Series.
- Multiple Parallel Series.

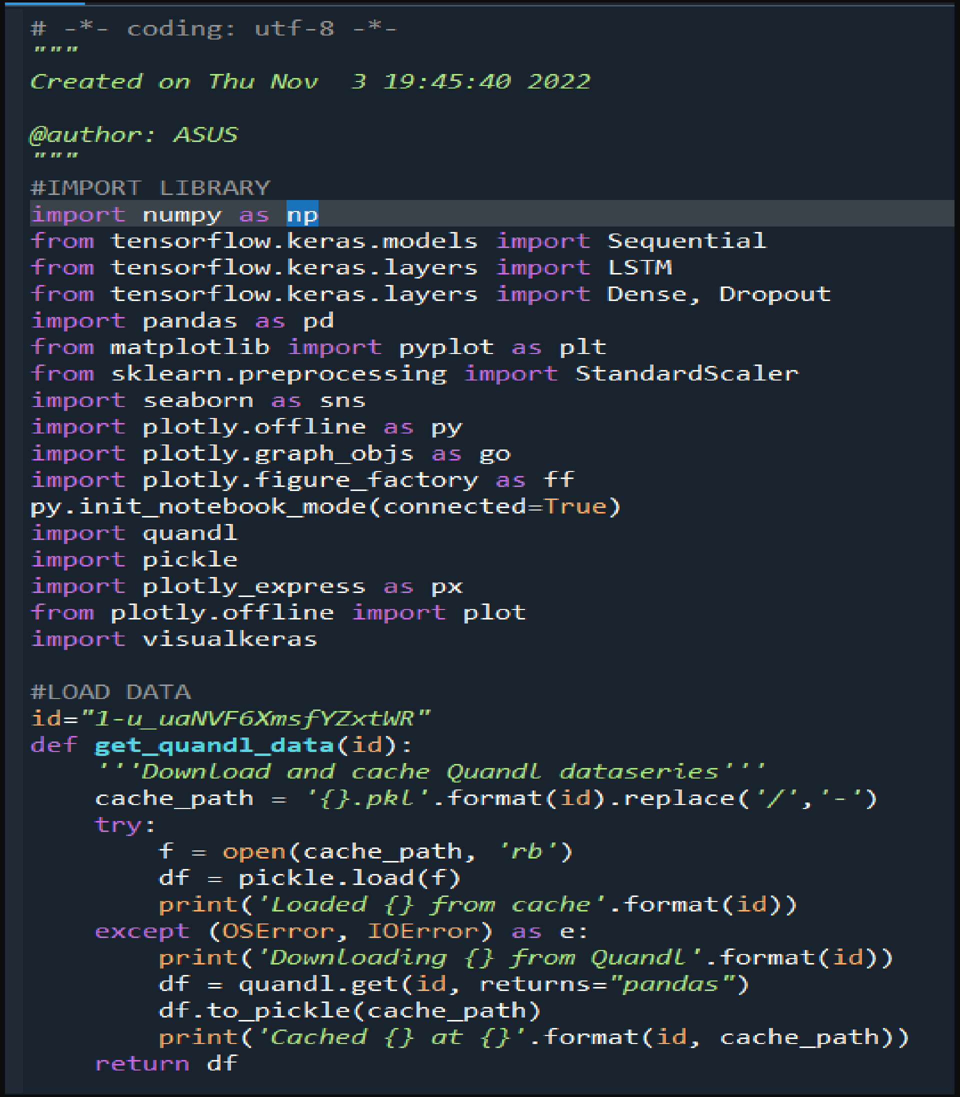
Multi-Step LSTM Models

- Vector Output Model
- Encoder-Decoder Model

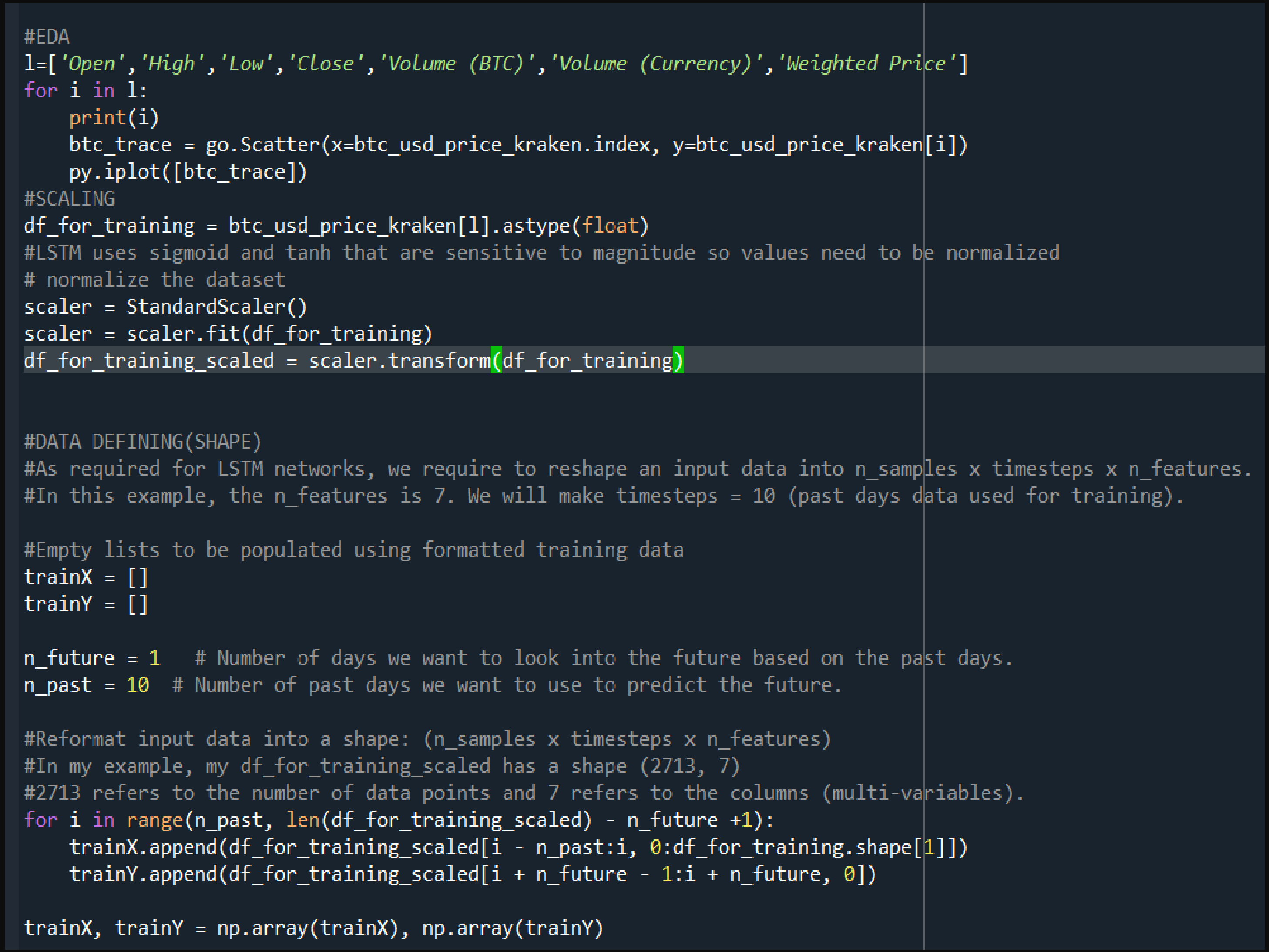
Multivariate Multi-Step LSTM Models

- Multiple Input Multi-Step Output.
- Multiple Parallel Input and Multi-Step Output.

BITCOIN STOCK FORCASTING

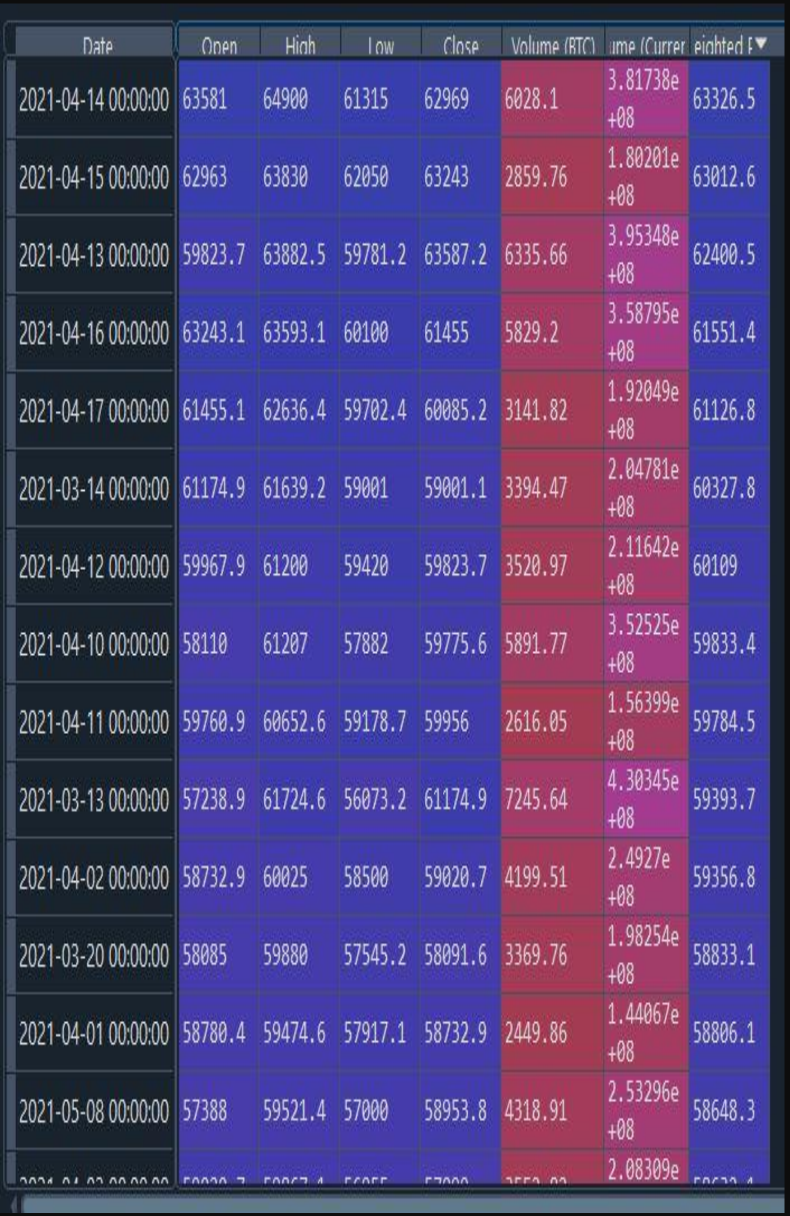


```python
# -*- coding: utf-8 -*-
"""
Created on Thu Nov  3 19:45:40 2022

@author: ASUS
"""
#IMPORT LIBRARY
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dense, Dropout
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import plotly.offline as py
import plotly.graph_objs as go
import plotly.figure_factory as ff
py.init_notebook_mode(connected=True)
import quandl
import pickle
import plotly_express as px
from plotly.offline import plot
import visualkeras

#LOAD DATA
id="1-u_uaNVF6XmsfYZxtWR"
def get_quandl_data(id):
    '''Download and cache Quandl dataseries'''
    cache_path = '{}.pkl'.format(id).replace('/','-')
    try:
        f = open(cache_path, 'rb')
        df = pickle.load(f)
        print('Loaded {} from cache'.format(id))
    except (OSError, IOError) as e:
        print('Downloading {} from Quandl'.format(id))
        df = quandl.get(id, returns="pandas")
        df.to_pickle(cache_path)
        print('Cached {} at {}'.format(id, cache_path))
    return df
```

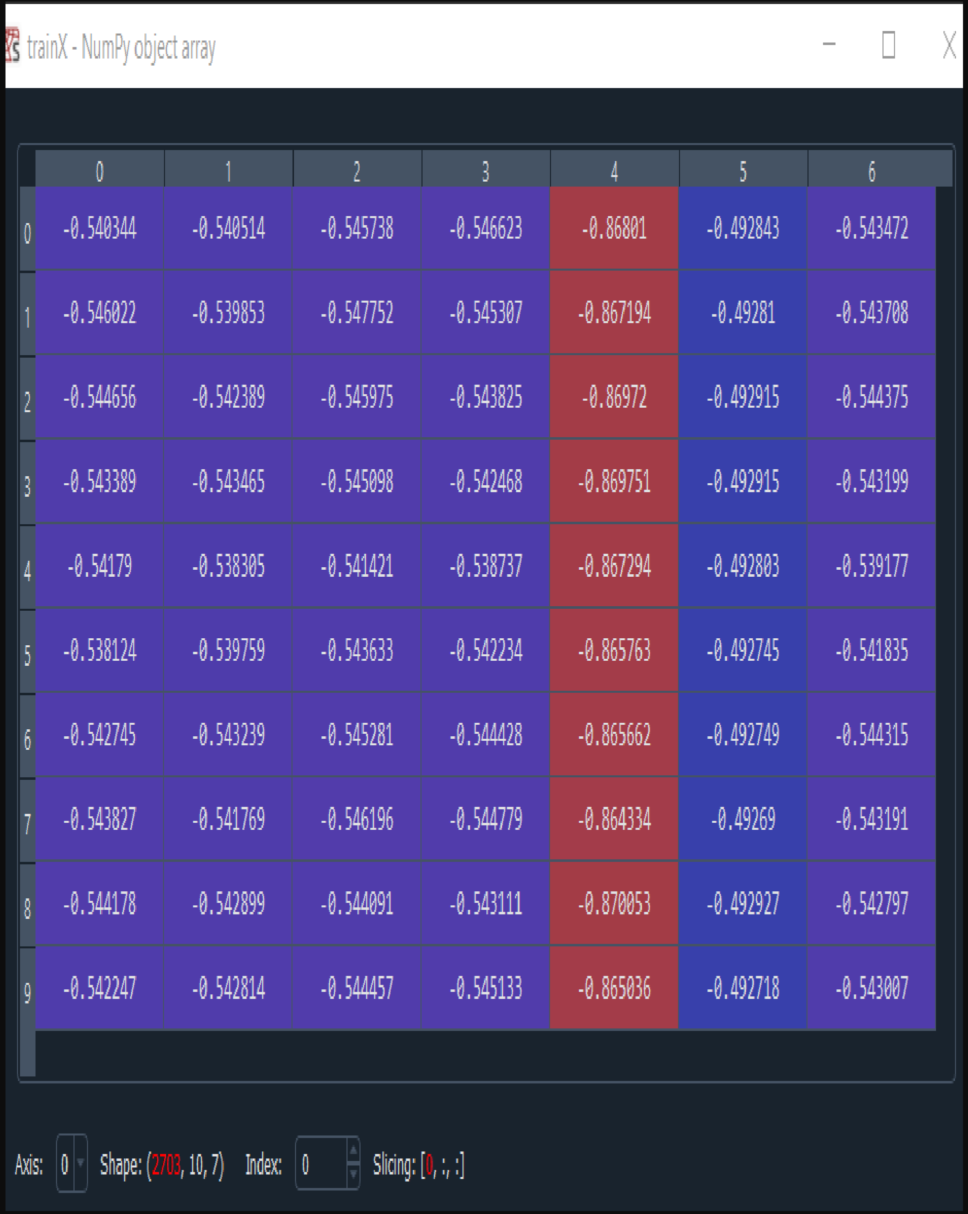


```python
#EDA
l=['Open','High','Low','Close','Volume (BTC)','Volume (Currency)','Weighted Price']
for i in l:
    print(i)
    btc_trace = go.Scatter(x=btc_usd_price_kraken.index, y=btc_usd_price_kraken[i])
    py.iplot([btc_trace])
#SCALING
df_for_training = btc_usd_price_kraken[l].astype(float)
#LSTM uses sigmoid and tanh that are sensitive to magnitude so values need to be normalized
# normalize the dataset
scaler = StandardScaler()
scaler = scaler.fit(df_for_training)
df_for_training_scaled = scaler.transform(df_for_training)


#DATA DEFINING(SHAPE)
#As required for LSTM networks, we require to reshape an input data into n_samples x timesteps x n_features.
#In this example, the n_features is 7. We will make timesteps = 10 (past days data used for training).

#Empty lists to be populated using formatted training data
trainX = []
trainY = []

n_future = 1   # Number of days we want to look into the future based on the past days.
n_past = 10  # Number of past days we want to use to predict the future.

#Reformat input data into a shape: (n_samples x timesteps x n_features)
#In my example, my df_for_training_scaled has a shape (2713, 7)
#2713 refers to the number of data points and 7 refers to the columns (multi-variables).
for i in range(n_past, len(df_for_training_scaled) - n_future +1):
    trainX.append(df_for_training_scaled[i - n_past:i, 0:df_for_training.shape[1]])
    trainY.append(df_for_training_scaled[i + n_future - 1:i + n_future, 0])

trainX, trainY = np.array(trainX), np.array(trainY)
```

DATA PREPROCESSING



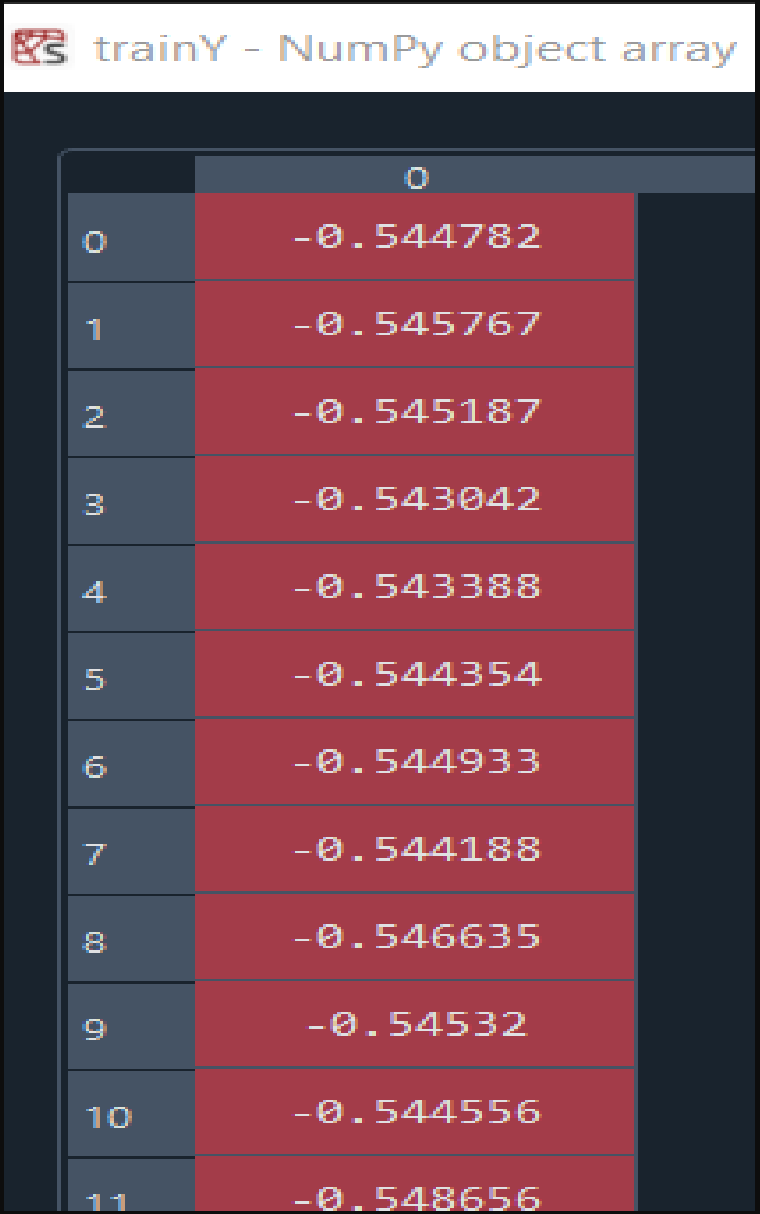
Date	Open	High	Low	Close	Volume (BTC)	ume (Currer	eighted F
2021-04-14 00:00:00	63581	64900	61315	62969	6028.1	3.81738e +08	63326.5
2021-04-15 00:00:00	62963	63830	62050	63243	2859.76	1.80201e +08	63012.6
2021-04-13 00:00:00	59823.7	63882.5	59781.2	63587.2	6335.66	3.95348e +08	62400.5
2021-04-16 00:00:00	63243.1	63593.1	60100	61455	5829.2	3.58795e +08	61551.4
2021-04-17 00:00:00	61455.1	62636.4	59702.4	60085.2	3141.82	1.92049e +08	61126.8
2021-03-14 00:00:00	61174.9	61639.2	59001	59001.1	3394.47	2.04781e +08	60327.8
2021-04-12 00:00:00	59967.9	61200	59420	59823.7	3520.97	2.11642e +08	60109
2021-04-10 00:00:00	58110	61207	57882	59775.6	5891.77	3.52525e +08	59833.4
2021-04-11 00:00:00	59760.9	60652.6	59178.7	59956	2616.05	1.56399e +08	59784.5
2021-03-13 00:00:00	57238.9	61724.6	56073.2	61174.9	7245.64	4.30345e +08	59393.7
2021-04-02 00:00:00	58732.9	60025	58500	59020.7	4199.51	2.4927e +08	59356.8
2021-03-20 00:00:00	58085	59880	57545.2	58091.6	3369.76	1.98254e +08	58833.1
2021-04-01 00:00:00	58780.4	59474.6	57917.1	58732.9	2449.86	1.44067e +08	58806.1
2021-05-08 00:00:00	57388	59521.4	57000	58953.8	4318.91	2.53296e +08	58648.3
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	2.08309e	[illegible]



trainX - NumPy object array

	0	1	2	3	4	5	6
0	-0.540344	-0.540514	-0.545738	-0.546623	-0.86801	-0.492843	-0.543472
1	-0.546022	-0.539853	-0.547752	-0.545307	-0.867194	-0.49281	-0.543708
2	-0.544656	-0.542389	-0.545975	-0.543825	-0.86972	-0.492915	-0.544375
3	-0.543389	-0.543465	-0.545098	-0.542468	-0.869751	-0.492915	-0.543199
4	-0.54179	-0.538305	-0.541421	-0.538737	-0.867294	-0.492803	-0.539177
5	-0.538124	-0.539759	-0.543633	-0.542234	-0.865763	-0.492745	-0.541835
6	-0.542745	-0.543239	-0.545281	-0.544428	-0.865662	-0.492749	-0.544315
7	-0.543827	-0.541769	-0.546196	-0.544779	-0.864334	-0.49269	-0.543191
8	-0.544178	-0.542899	-0.544091	-0.543111	-0.870053	-0.492927	-0.542797
9	-0.542247	-0.542814	-0.544457	-0.545133	-0.865036	-0.492718	-0.543007

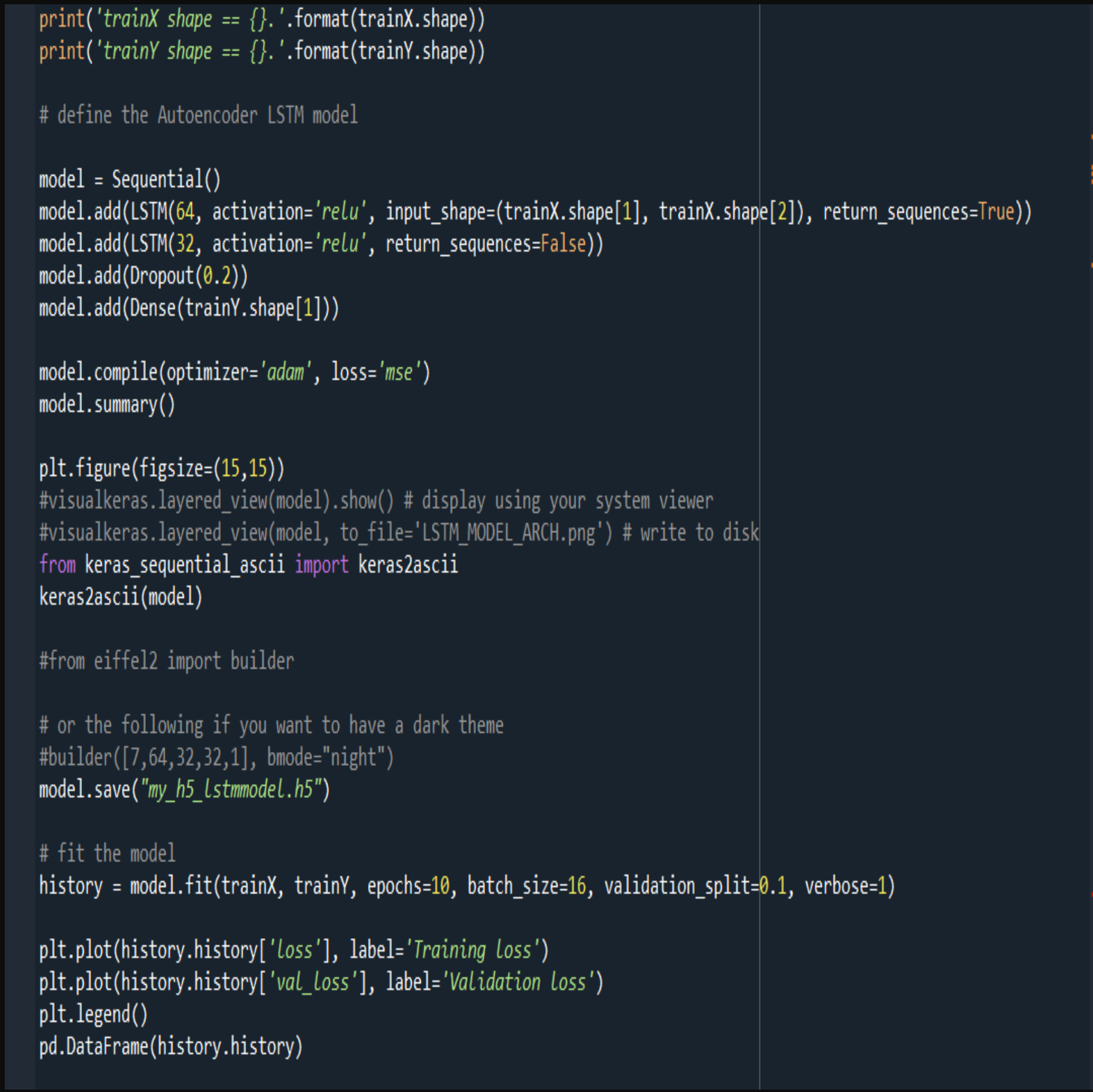
Axis: 0 Shape: (2703, 10, 7) Index: 0 Slicing: [0, :, :]



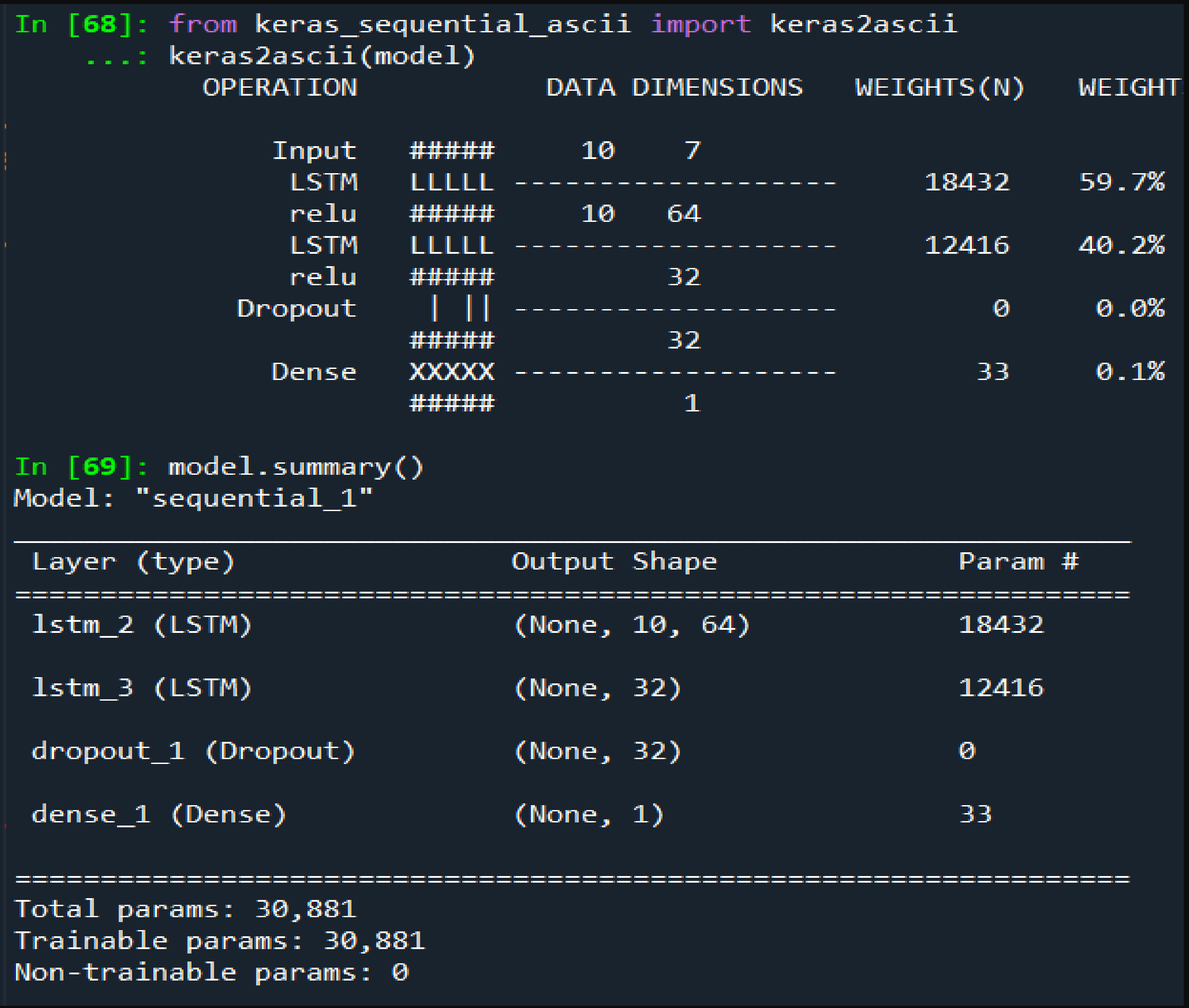
trainY - NumPy object array

	0
0	-0.544782
1	-0.545767
2	-0.545187
3	-0.543042
4	-0.543388
5	-0.544354
6	-0.544933
7	-0.544188
8	-0.546635
9	-0.54532
10	-0.544556
11	-0.548656

MODEL BUILDING

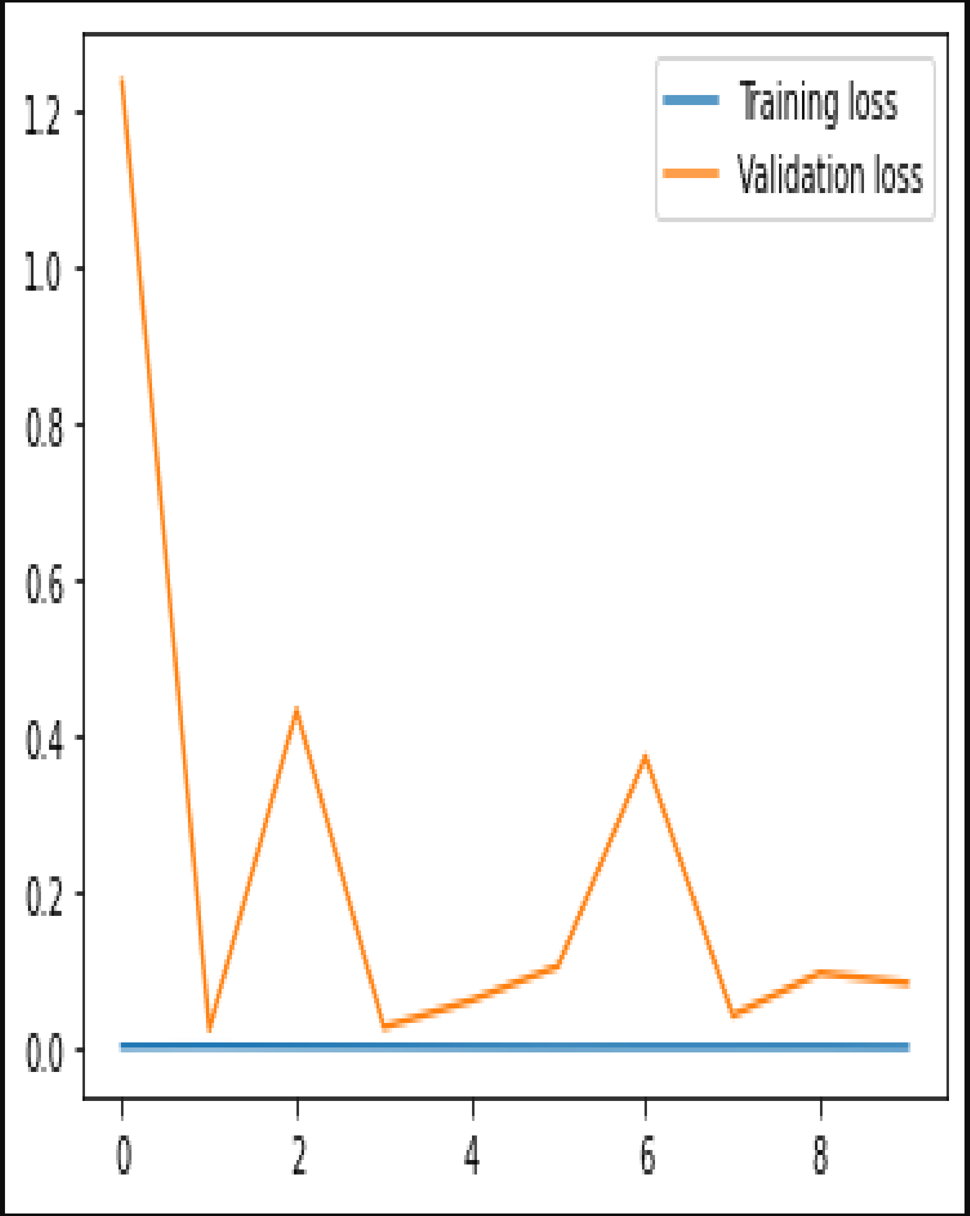


```python
print('trainX shape == {}.'.format(trainX.shape))
print('trainY shape == {}.'.format(trainY.shape))

# define the Autoencoder LSTM model

model = Sequential()
model.add(LSTM(64, activation='relu', input_shape=(trainX.shape[1], trainX.shape[2]), return_sequences=True))
model.add(LSTM(32, activation='relu', return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(trainY.shape[1]))

model.compile(optimizer='adam', loss='mse')
model.summary()

plt.figure(figsize=(15,15))
#visualkeras.layered_view(model).show() # display using your system viewer
#visualkeras.layered_view(model, to_file='LSTM_MODEL_ARCH.png') # write to disk
from keras_sequential_ascii import keras2ascii
keras2ascii(model)

#from eiffel2 import builder

# or the following if you want to have a dark theme
#builder([7,64,32,32,1], bmode="night")
model.save("my_h5_lstmmodel.h5")

# fit the model
history = model.fit(trainX, trainY, epochs=10, batch_size=16, validation_split=0.1, verbose=1)

plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation loss')
plt.legend()
pd.DataFrame(history.history)
```

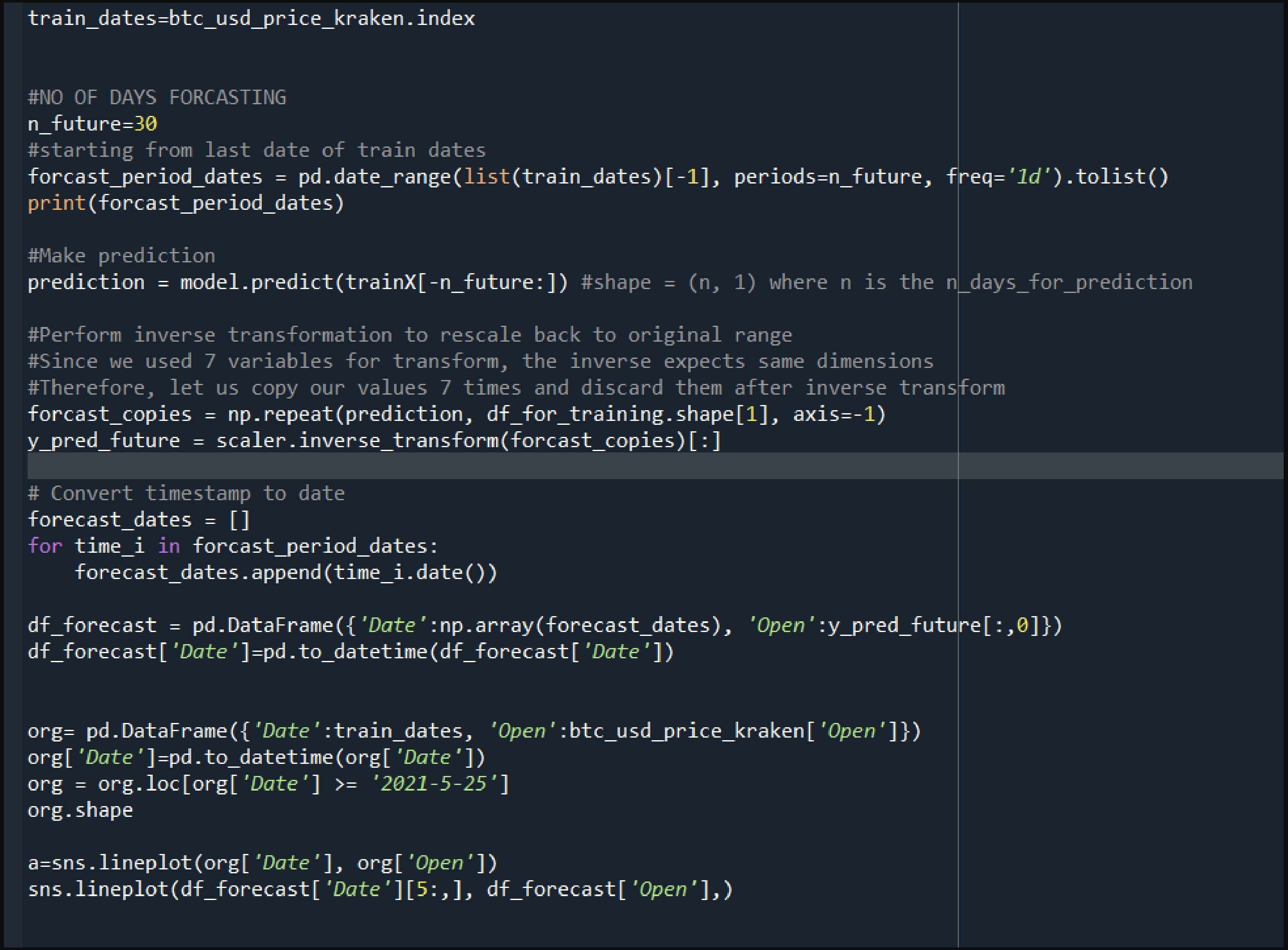


```
In [68]: from keras_sequential_ascii import keras2ascii
    ...: keras2ascii(model)
           OPERATION           DATA DIMENSIONS   WEIGHTS(N)   WEIGHT

               Input   #####     10    7
                LSTM   LLLLL -------------------     18432    59.7%
                relu   #####     10   64
                LSTM   LLLLL -------------------     12416    40.2%
                relu   #####          32
             Dropout    | || -------------------         0     0.0%
                       #####          32
               Dense   XXXXX -------------------        33     0.1%
                       #####           1

In [69]: model.summary()
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_2 (LSTM)               (None, 10, 64)            18432

 lstm_3 (LSTM)               (None, 32)                12416

 dropout_1 (Dropout)         (None, 32)                0

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 30,881
Trainable params: 30,881
Non-trainable params: 0
```

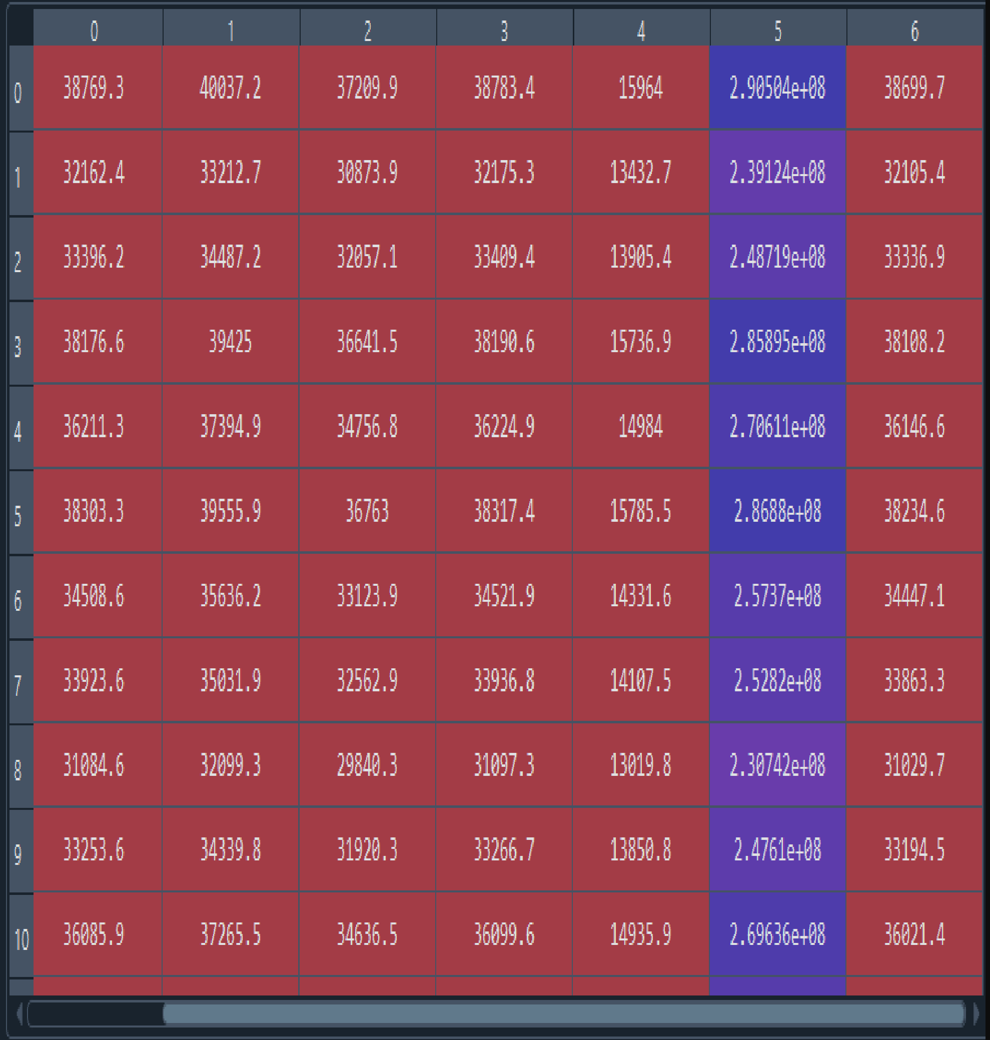
TRAINING



FORCASTING

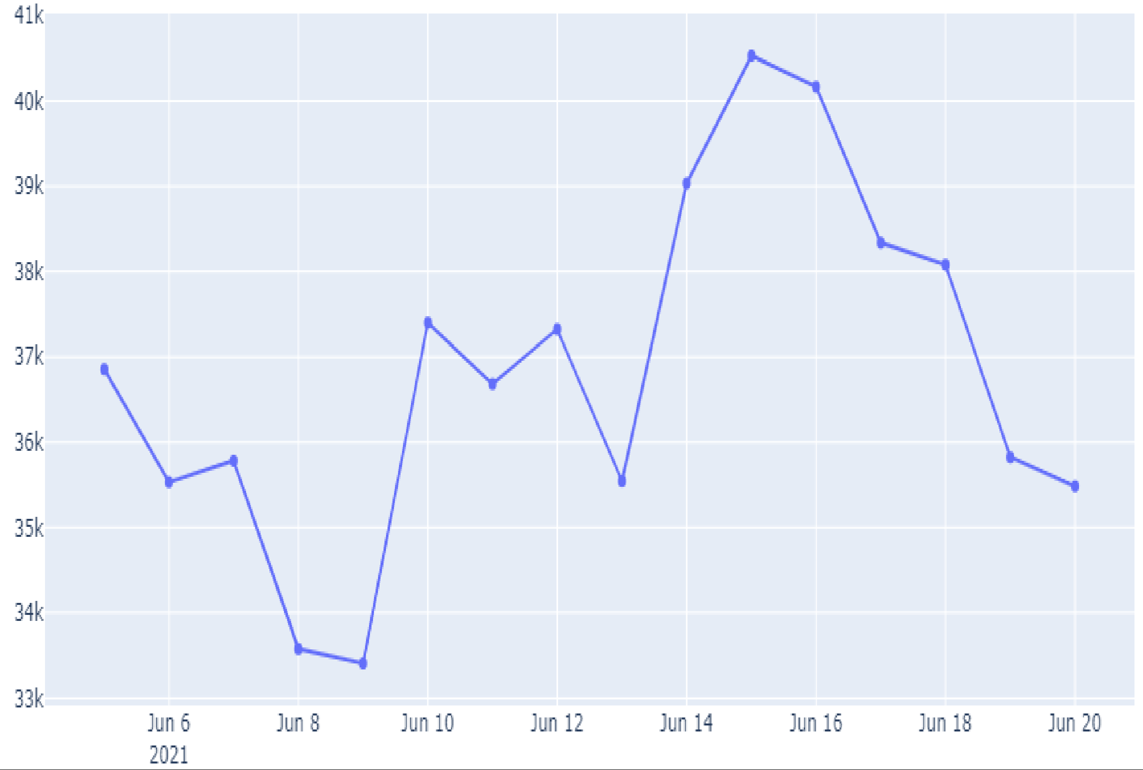


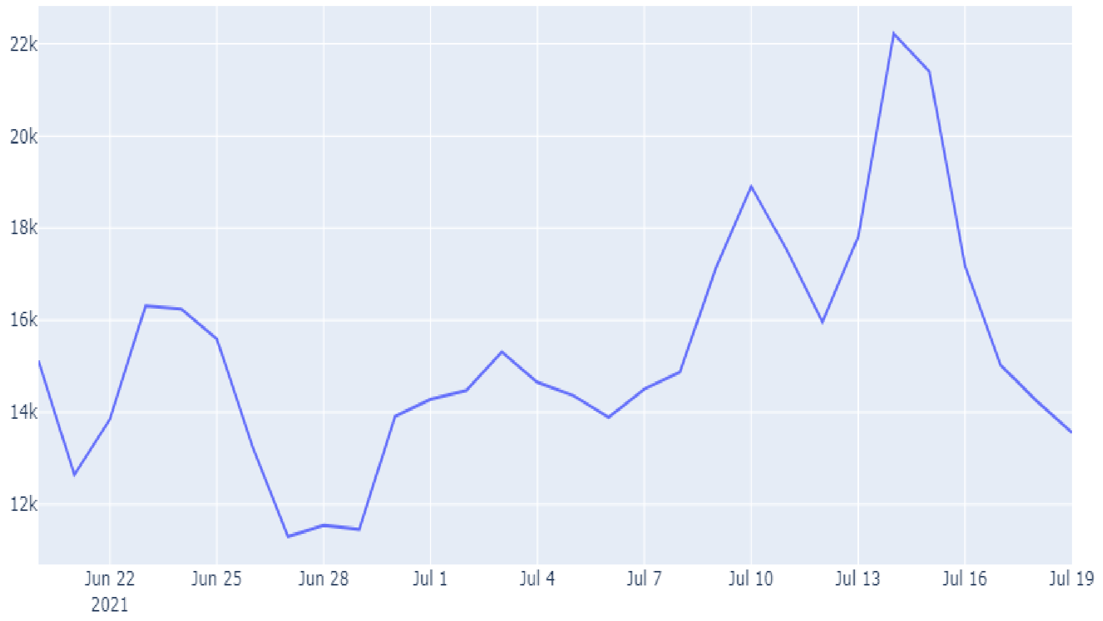
```
train_dates=btc_usd_price_kraken.index


#NO OF DAYS FORCASTING
n_future=30
#starting from last date of train dates
forcast_period_dates = pd.date_range(list(train_dates)[-1], periods=n_future, freq='1d').tolist()
print(forcast_period_dates)

#Make prediction
prediction = model.predict(trainX[-n_future:]) #shape = (n, 1) where n is the n_days_for_prediction

#Perform inverse transformation to rescale back to original range
#Since we used 7 variables for transform, the inverse expects same dimensions
#Therefore, let us copy our values 7 times and discard them after inverse transform
forcast_copies = np.repeat(prediction, df_for_training.shape[1], axis=-1)
y_pred_future = scaler.inverse_transform(forcast_copies)[:]

# Convert timestamp to date
forecast_dates = []
for time_i in forcast_period_dates:
    forecast_dates.append(time_i.date())

df_forecast = pd.DataFrame({'Date':np.array(forecast_dates), 'Open':y_pred_future[:,0]})
df_forecast['Date']=pd.to_datetime(df_forecast['Date'])


org= pd.DataFrame({'Date':train_dates, 'Open':btc_usd_price_kraken['Open']})
org['Date']=pd.to_datetime(org['Date'])
org = org.loc[org['Date'] >= '2021-5-25']
org.shape

a=sns.lineplot(org['Date'], org['Open'])
sns.lineplot(df_forecast['Date'][5:,], df_forecast['Open'],)
```



	0	1	2	3	4	5	6
0	38769.3	40037.2	37209.9	38783.4	15964	2.90504e+08	38699.7
1	32162.4	33212.7	30873.9	32175.3	13432.7	2.39124e+08	32105.4
2	33396.2	34487.2	32057.1	33409.4	13905.4	2.48719e+08	33336.9
3	38176.6	39425	36641.5	38190.6	15736.9	2.85895e+08	38108.2
4	36211.3	37394.9	34756.8	36224.9	14984	2.70611e+08	36146.6
5	38303.3	39555.9	36763	38317.4	15785.5	2.8688e+08	38234.6
6	34508.6	35636.2	33123.9	34521.9	14331.6	2.5737e+08	34447.1
7	33923.6	35031.9	32562.9	33936.8	14107.5	2.5282e+08	33863.3
8	31084.6	32099.3	29840.3	31097.3	13019.8	2.30742e+08	31029.7
9	33253.6	34339.8	31920.3	33266.7	13850.8	2.4761e+08	33194.5
10	36085.9	37265.5	34636.5	36099.6	14935.9	2.69636e+08	36021.4

RESULT


41k
40k
39k
38k
37k
36k
35k
34k
33k
Jun 6
2021
Jun 8
Jun 10
Jun 12
Jun 14
Jun 16
Jun 18
Jun 20


22k
20k
18k
16k
14k
12k
Jun 22
2021
Jun 25
Jun 28
Jul 1
Jul 4
Jul 7
Jul 10
Jul 13
Jul 16
Jul 19

THANK YOU