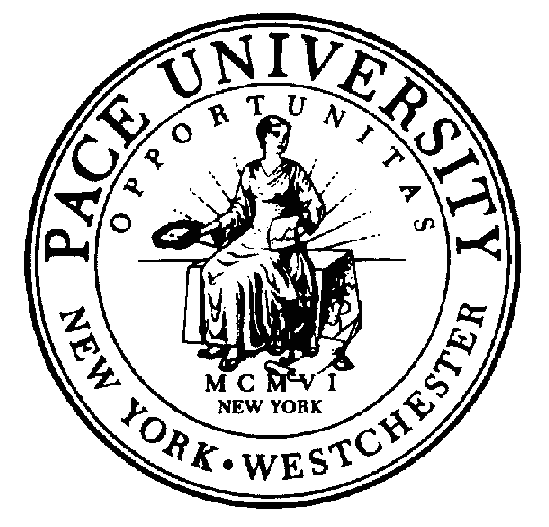
**IBM STOCK PRICE PREDICTION**

***Submitted by***

***MOHAMMED ZEESHAN ALI***

***MA65450P***

**ALGORITHMS FOR DATA SCIENCE – CS767**

****

**ABSTRACT**

In this project I attempt to implement machine learning approach to predict stock prices. Machine learning is effectively implemented in forecasting stock prices. The objective of this predict the stock prices in order to make more informed and accurate investment decisions. We propose a stock price prediction system that integrates mathematical functions, machine learning, and other external factors for the purpose of achieving better stock prediction accuracy and issuing profitable trades.

There are two types of stocks. You may know of intraday trading by the commonly used term "day trading." Interday traders hold securities positions from at least one day to the next and often for several days to weeks or months. LSTMs are very powerful in sequence prediction problems because they’re able to store past information. This is important in our case because the previous price of a stock is crucial in predicting its future price. While predicting the actual price of a stock is an uphill climb, we can build a model that will predict whether the price will go up or down.

**CHAPTER 1**

**INTRODUCTION**

The financial market is a dynamic and composite system where people can buy and sell currencies, stocks, equities and derivatives over virtual platforms supported by brokers. The stock market allows investors to own shares of public companies through trading either by exchange or over the counter markets. This market has given investors the chance of gaining money and having a prosperous life through investing small initial amounts of money, low risk compared to the risk of opening new business or the need of high salary career. Stock markets are affected by many factors causing the uncertainty and high volatility in the market. Although humans can take orders and submit them to the market, automated trading systems (ATS) that are operated by the implementation of computer programs can perform better and with higher momentum in submitting orders than any human. However, to evaluate and control the performance of ATSs, the implementation of risk strategies and safety measures applied based on human judgements are required. Many factors are incorporated and considered when developing an ATS, for instance, trading strategy to be adopted, complex mathematical functions that reflect the state of a specific stock, machine learning algorithms that enable the prediction of the future stock value, and specific news related to the stock being analyzed.

Time-series prediction is a common technique widely used in many real-world applications such as weather forecasting and financial market prediction. It uses the continuous data in a period of time to predict the result in the next time unit. Many time- series prediction algorithms have shown their effectiveness in practice. The most common algorithms now are based on Recurrent Neural Networks (RNN), as well as its special type - Long-short Term Memory (LSTM) and Gated Recurrent Unit (GRU). Stock market is a typical area that presents time-series data and many research study on it and proposed various models. In this project, LSTM model is used to predict the stock price.

**1.1 MOTIVATION FOR WORK**

Businesses primarily run over customer’s satisfaction, customer reviews about their products. Shifts in sentiment on social media have been shown to correlate with shifts in stock markets. Identifying customer grievances thereby resolving them leads to customer satisfaction as well as trustworthiness of an organization. Hence there is a necessity of an un biased automated system to classify customer reviews regarding any problem. In today’s environment where we’re justifiably suffering from data overload (although this does not mean better or deeper insights), companies might have mountains of customer feedback collected; but for mere humans, it’s still impossible to analyze it manually without any sort of error or bias. Oftentimes, companies with the best intentions find themselves in an insights vacuum. You know you need insights to inform your decision making and you know that you’re lacking them, but don’t know how best to get them. Sentiment analysis provides some answers into what the most important issues are, from the perspective of customers, at least. Because sentiment analysis can be automated, decisions can be made based on a significant amount of data rather than plain intuition.

**1.2 PROBLEM STATEMENT**

Time Series forecasting & modelling plays an important role in data analysis. Time series analysis is a specialized branch of statistics used extensively in fields such as Econometrics & Operation Research. Time Series is being widely used in analytics & data science. Stock prices are volatile in nature and price depends on various factors. The main aim of this project is to predict stock prices using Long short term memory (LSTM).

**CHAPTER 2**

**LITERATURE SURVEY**

"What other people think” has always been an important piece of information for most of us during the decision-making process. The Internet and the Web have now (among other things) made it possible to find out about the opinions and experiences of those in the vast pool of people that are neither our personal acquaintances nor well-known professional critics — that is, people we have never heard of. And conversely, more and more people are making their opinions available to strangers via the Internet. The interest that individual users show in online opinions about products and services, and the potential influence such opinions wield, is something that is driving force for this area of interest. And there are many challenges involved in this process which needs to be walked all over in order to attain proper outcomes out of them. In this survey we analysed basic methodology that usually happens in this process and measures that are to be taken to overcome the challenges being faced.

**CHAPTER 3**

**METHODOLOGY**

3.1 **PROPOSED SYSTEMS**

The prediction methods can be roughly divided into two categories, statistical methods and artificial intelligence methods. Statistical methods include logistic regression model, ARCH model, etc. Artificial intelligence methods include multi-layer perceptron, convolutional neural network, naive Bayes network, back propagation network, single-layer LSTM, support vector machine, recurrent neural network, etc. They used Long short-term memory network (LSTM).

**Long short-term memory network:**

Long short-term memory network (LSTM) is a particular form of recurrent neural network (RNN).

**Working of LSTM:**

LSTM is a special network structure with three “gate” structures. Three gates are placed in an LSTM unit, called input gate, forgetting gate and output gate. While information enters the LSTM’s network, it can be selected by rules. Only the information conforms to the algorithm will be left, and the information that does not conform will be forgotten through the forgetting gate.

The experimental data in this paper are the actual historical data downloaded from the Internet. Three data sets were used in the experiments. It is needed to find an optimization algorithm that requires less resources and has faster convergence speed.

* Used Long Short-term Memory (LSTM) with embedded layer and the LSTM neural network with automatic encoder.
* LSTM is used instead of RNN to avoid exploding and vanishing gradients.
* In this project python is used to train the model, MATLAB is used to reduce

dimensions of the input. MySQL is used as a dataset to store and retrieve data.

* The historical stock data table contains the information of opening price, the highest

price, lowest price, closing price, transaction date, volume and so on.

* The accuracy of this LSTM model used in this project is 57%.

**CHAPTER 4**

**EXPERIMENT ANALYSIS**

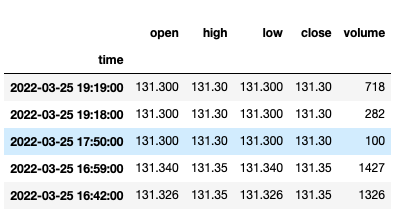
This project can run on commodity hardware. We ran entire project on an Intel I5 processor with 8 GB Ram, 2 GB Nvidia Graphic Processor, It also has 2

cores which runs at 1.7 GHz, 2.1 GHz respectively. First part of the is training phase which takes 10-15 mins of time and the second part is testing part which only takes few seconds to make predictions and calculate accuracy.

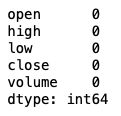
**CHAPTER 5**

**PROJECT ANALYSIS**

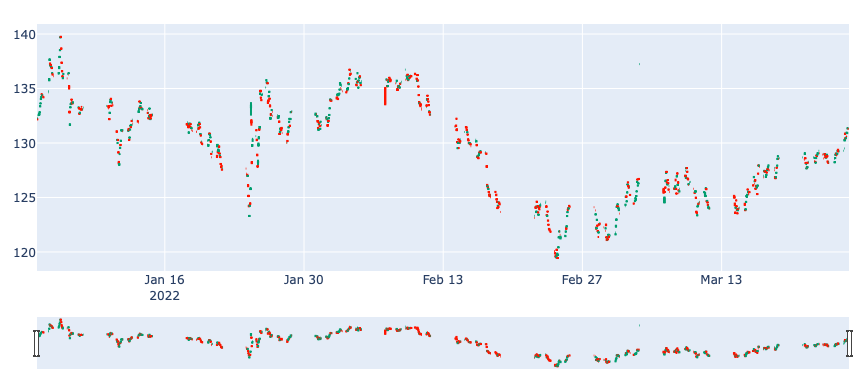
**DATA IMPORTED**

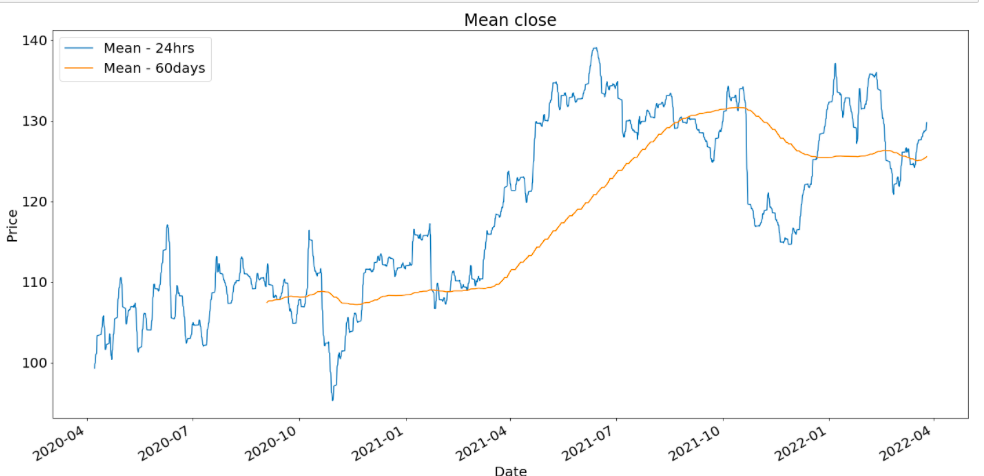
****

**REMOVAL OF NA VALUES**

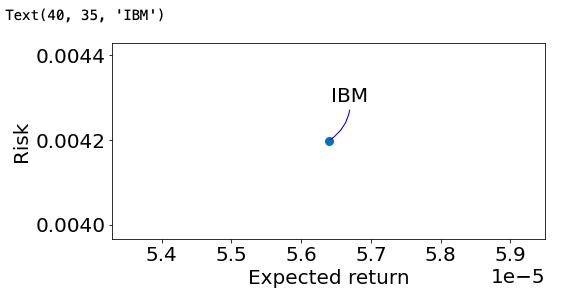
****

**DATA VISUALISATION**

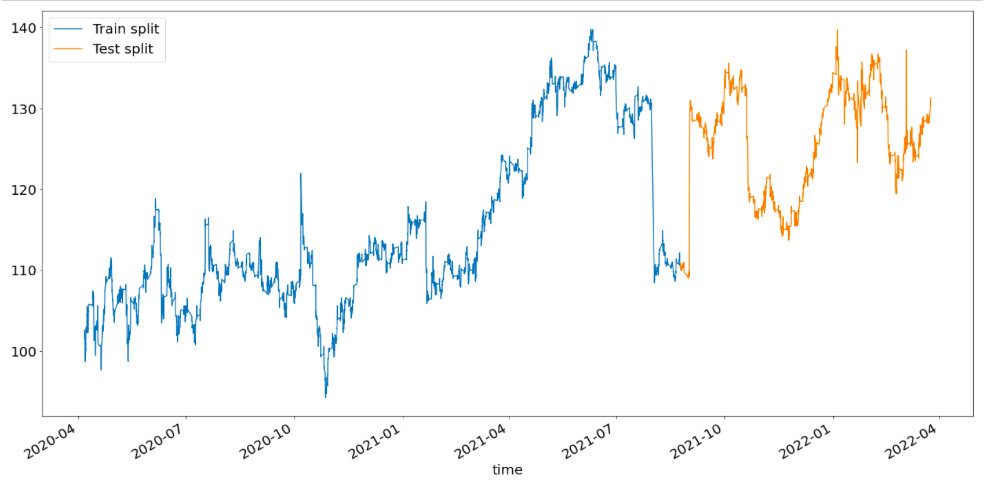
****

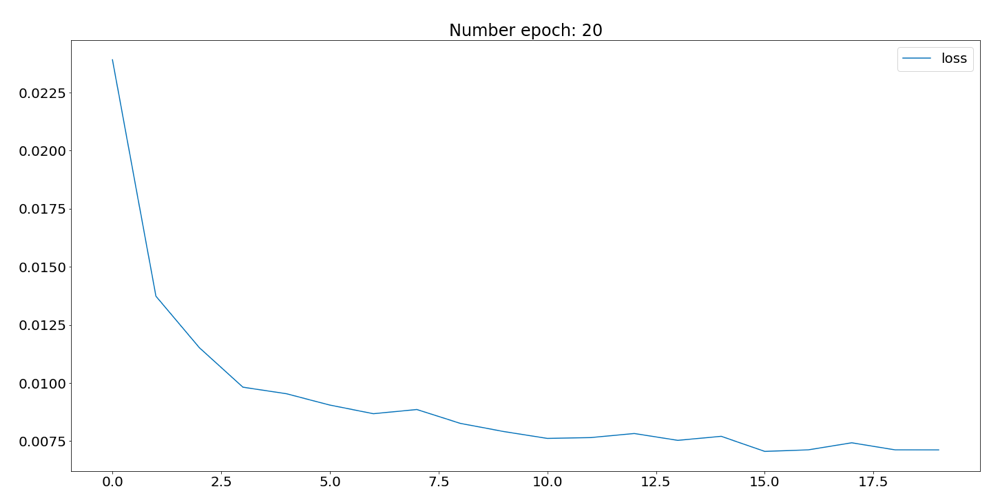
****

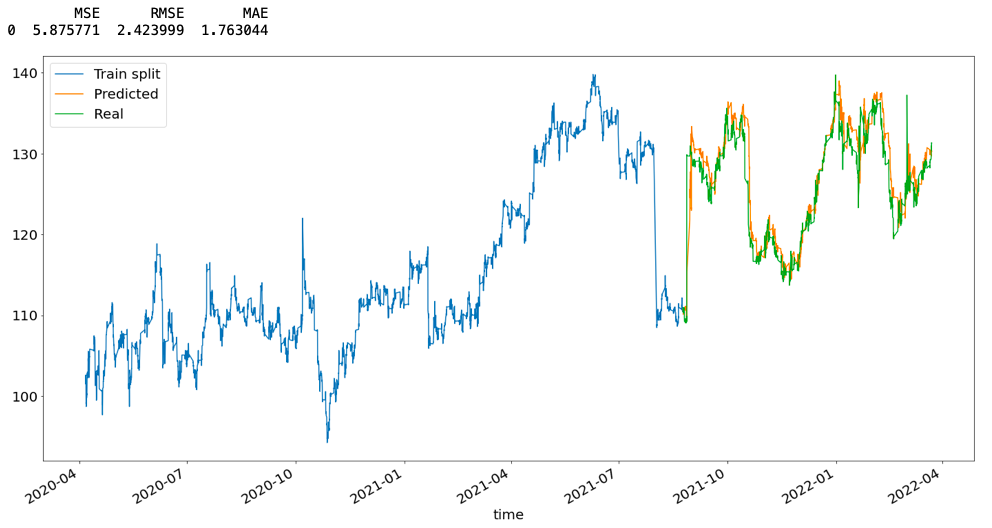
What is the risk value by investing in this stock ?

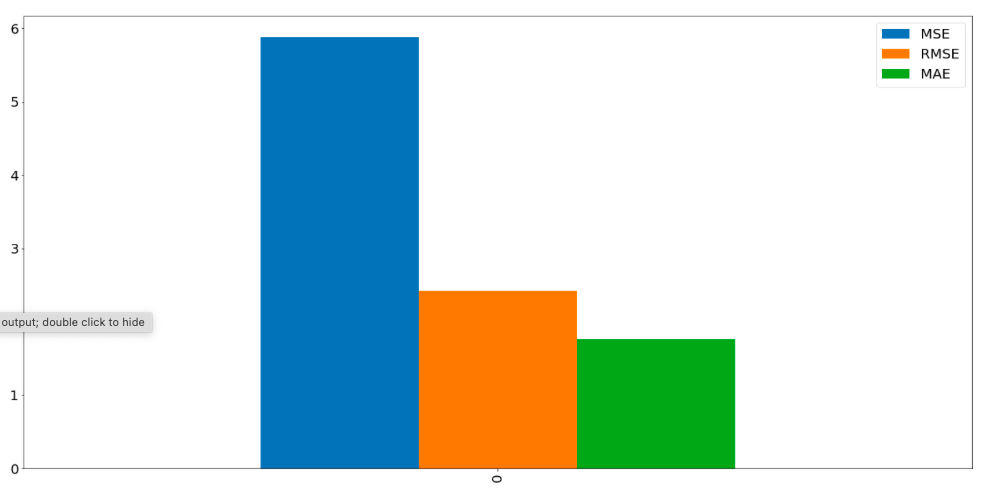
****

 Dataset with a month frequency.

****

****

****

****

**CODE**

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import random

from joblib import Parallel, delayed

from collections import deque

import plotly.graph\_objects as go

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error

from sklearn.metrics import mean\_absolute\_error

import tensorflow as tf

from keras.models import Sequential

from keras.layers import BatchNormalization, LSTM

from keras.layers.core import Dropout, Dense

%matplotlib inline

plt.rcParams["figure.figsize"] = (20,10)

plt.rcParams["figure.autolayout"] = True

plt.rcParams['font.size'] = '20'

data = pd.read\_csv('/Users/zee/Documents/2 Year IBM Stock Data.csv', index\_col='time')

data.index = pd.to\_datetime(data.index, format='%m/%d/%Y %H:%M')

print(data[:10])

data.head()

data.info()

data.describe()

data = data.resample('H').mean().dropna()

data.head()

data.isna().sum()

s1 = data.last('4D')

fig = go.Figure(data=[go.Candlestick(x=sample.index,

open=sample['open'],

high=sample['high'],

low=sample['low'],

close=sample['close'])])

fig.show()

close\_C = df['close']

close\_C.rolling(window=24).mean().plot(label='Mean - 24hrs')

close\_C.rolling(window=24\*60).mean().plot(label='Mean - 60days')

plt.xlabel('Date')

plt.ylabel('Price')

plt.title('Mean close')

plt.legend();

data\_change = data['close'].copy().pct\_change().dropna()

area\_size = np.pi \* 20

plt.figure(figsize=(8, 4))

plt.scatter(data\_change.mean(), data\_change.std(), s=area)

plt.xlabel('Expected return')

plt.ylabel('Risk')

plt.annotate('IBM', xy=(data\_change.mean(), data\_change.std()), xytext=(40, 35), textcoords='offset points', ha='right', va='bottom',

arrowprops=dict(arrowstyle='-', color='blue', connectionstyle='arc3,rad=-0.3'))

data\_copy = data.iloc[:, 3:4].copy()

data\_copy.head()

data\_copy['target'] = data\_copy['close'].shift(periods=-10)

data\_copy.head()

def bootstrap(df, freq, shift\_max=0, n\_samples=1, replace=False,

n\_jobs=1):

leap\_mask = (~((df.index.month == 2) & (df.index.day == 29)))

new\_df = df.copy() #df without leap years

new\_df = new\_df[leap\_mask]

parallel = Parallel(n\_jobs=n\_jobs)

def bootstrap(i):

"""

Returns

-------

df\_res : pd.DataFrame

seasonally bootstrapped data

"""

random.seed(i)

if replace == True:

index = new\_df.index

blocks = create\_blocks(new\_df, freq, shift\_max)

#convert list of blocks into array of blocks in order to use indexing

b = np.empty(len(blocks), dtype=object)

b[:] = blocks

blocks = b

lengths = [len(block) for block in blocks]

ndays = np.bincount(lengths).argmax()

blocks\_to\_shuffle = np.where(lengths == ndays)[0]

new\_indices = np.copy(blocks\_to\_shuffle)

new\_indices = np.random.choice(new\_indices, len(new\_indices),

replace=True)

blocks[blocks\_to\_shuffle]=blocks[new\_indices]

else:

index = df.index

blocks = create\_blocks(df, freq, shift\_max)

random.shuffle(blocks)

df\_res = pd.concat(blocks)

if isinstance(df\_res, pd.DataFrame):

df\_res.set\_index(index, inplace=True)

else:

df\_res.index = index

return df\_res

return parallel(

delayed(create\_seasonal\_bootstrap)(i) for i in range(n\_samples)

)

def create\_blocks(df, freq, shift\_max=0):

shift = random.randint(-shift\_max, shift\_max)

df = df.shift(shift)

#we retrieve groups through the groupby method

blocks = deque([gp[1] for gp in df.groupby(pd.Grouper(freq=freq))])

return blocks

bootstrap\_samples = seasonal\_bootstrap(df=df\_copy,

freq='M',

shift\_max=0,

n\_samples=6,

replace=True,

n\_jobs=1)

def split\_data(df, split\_fraction):

split\_size = round(len(df)\*split\_fraction)

train = df[:split\_size]

test = df[split\_size:]

return train, test

train\_split, test\_split = split\_data(bootstrap\_samples[1], 0.7)

print('train\_split.shape', train\_split.shape)

print('test\_split.shape', test\_split.shape)

scaler\_train = MinMaxScaler(feature\_range=(0,1))

scaler\_train = scaler\_train.fit(train\_split)

scaled\_train = scaler\_train.transform(train\_split)

scaled\_test = scaler\_train.transform(test\_split)

print('scaled train', scaled\_train.shape)

print('scaled test', scaled\_test.shape)

def split\_sequence(sequence, n\_steps):

X, y = list(), list()

for i in range( len(sequence) ):

end\_ix = i + n\_steps

if end\_ix > len(sequence)-1:

break

seq\_x, seq\_y = sequence[i:end\_ix], sequence[end\_ix]

X.append(seq\_x)

y.append(seq\_y)

return np.array(X)[:,:,:-1], np.array(y)[:,-1]

X\_train, Y\_train = split\_sequence(scaled\_train, 24)

X\_test, Y\_test = split\_sequence(scaled\_test, 24)

print(X\_train.shape)

print(Y\_train.shape)

print("")

print(X\_test.shape)

print(Y\_test.shape)

def lstm\_model(X, Y\_train, nb\_neuron\_dense, epoch):

tf.keras.backend.clear\_session()

model = Sequential()

model.add(LSTM(nb\_neuron\_dense, return\_sequences=True, input\_dim=n\_input))

model.add(BatchNormalization())

model.add(Dropout(0.2))

model.add(LSTM(64, return\_sequences=False))

model.add(Dropout(0.2))

model.add(Dense(units=32))

model.add(Dropout(0.2))

model.add(Dense(units=1, activation='linear'))

model.compile(optimizer='sgd', loss='mse', metrics=['accuracy'])

history = model.fit(X, Y\_train,

batch\_size=16, epochs=epoch, verbose=1)

return history, model

n\_input = X\_train.shape[1] \* X\_train.shape[2]

x\_train\_reshape = np.reshape(X\_train, (X\_train.shape[0], 1, X\_train.shape[1]))

x\_test\_reshape = np.reshape(X\_test, (X\_test.shape[0], 1, X\_test.shape[1]))

def rescale\_pred(X\_test, Y\_test\_predicted, Y\_test):

concat\_predict = np.concatenate( (X\_test[:,0,:], Y\_test\_predicted), 1 )

concat\_real = np.concatenate( (X\_test[:,0,:], Y\_test.reshape( Y\_test.shape[0], 1 )), 1 )

Y\_test\_predicted\_rescaled = scaler\_train.inverse\_transform(concat\_predict)

Y\_test\_real\_rescaled = scaler\_train.inverse\_transform(concat\_real)

total = pd.concat([pd.DataFrame(Y\_test\_real\_rescaled).iloc[:,-1],

pd.DataFrame(Y\_test\_predicted\_rescaled).iloc[:,-1]],1)

total.columns = ['Real', 'Predicted']

return total

def error(df\_actual, df\_predicted):

mse = mean\_squared\_error(df\_actual, df\_predicted, squared=True)

rmse = mean\_squared\_error(df\_actual, df\_predicted, squared=False)

mae = mean\_absolute\_error(df\_actual, df\_predicted)

errorS = pd.DataFrame([mse, rmse, mae], index=['MSE','RMSE','MAE'])

return errorS

def plot\_val\_loss(history):

plt.plot(pd.DataFrame(history.history)['loss'], label = 'loss')

plt.legend()

plt.title("Number epoch: " + str(len(pd.DataFrame(history.history))))

return pd.DataFrame(history.history)

history, model = lstm\_model(x\_train\_reshape, Y\_train,

nb\_neuron\_dense=128,

epoch = 20)

plot\_val\_loss(history)

Y\_test\_predicted = model.predict(x\_test\_reshape, verbose=0)

concat\_predict = np.concatenate( (X\_test[:,0,:], Y\_test\_predicted), 1 )

Y\_test\_predicted\_rescaled = scaler\_train.inverse\_transform(concat\_predict)

concat\_real = np.concatenate( (X\_test[:,0,:], Y\_test.reshape(Y\_test.shape[0],1)), 1 )

Y\_test\_real\_rescaled = scaler\_train.inverse\_transform(concat\_real)

df\_predicted = pd.concat([ pd.DataFrame( Y\_test\_predicted\_rescaled[:,-1] ), pd.DataFrame( Y\_test\_real\_rescaled[:,-1] ) ], 1)

df\_predicted.columns = ["Predicted",'Real']

df\_predicted.index = test\_split[:-24].index

df\_predicted = df\_predicted.dropna()

train\_split['target'].plot(label='Train split')

df\_predicted['Predicted'].plot(label='Predicted')

df\_predicted['Real'].plot(label='Real')

plt.legend();

errorS = pd.DataFrame(error(df\_predicted['Real'], df\_predicted['Predicted'])).T

errorS.columns=['MSE','RMSE','MAE']

errorS.plot.bar()

print(errorS)

plt.show()