University Hassan Premier – Settat
National School Of Applied Sciences – Berrechid
Module: Big Data and applications

BIG DATA PROJECT:

Theme:

BITCOIN PREDICTION IN REAL TIME

Made by :

- *Ismail Fakiri*
- *Adnane Driouche*
- *Omaima El amrani El idrissi*
- *Hajar Ouaziz*

Supervised by :

- *Mme. KARIM*

Cycle Engineer

Information Systems and Big-Data Engineering (ISIBD)

Department of Mathematics

And Computer Science

2020/2021

Sommaire

Thanks

First, we thank ALLAH, the source of all knowledge, who gave us the strength and the will to do this modest work.

We wanted to write a sincere «thank you» for our teacher Ms Karim for her support, teaching and advices throughout our studies that have just passed. We wanted her to know that all the students will have a very good memory of this school year.

Finally, thanks to our team for their hard work during all the difficulties we faced to make this work succed

INTRODUCTION

As students in the last year in Information Systems and BIG-DATA Engineering at ENSAB, we are required to make a project that uses the knowledge that we gathered in our studies in the module of "Big Data and applications".

Our project focuses on bitcoin prediction in real time using prediction models and the implementation of big data technologies.

Currently, bitcoin has become the theme of time and future,
The Bitcoin is a digital currency created in January 2009, the value of Bitcoin crosses the 30,000-dollar mark in early 2021 for the first time since its creation, for this the prediction of bitcoin in real time is important for people who use this coin to track its evolution.

This report presents the essence of our project work, It is structured in three chapters, In the first, a general description of the project, the second chapter will be devoted to technical analysis and the last chapter will focus on the presentation of our realized project, to reach our ends, this report will be will end with a conclusion.

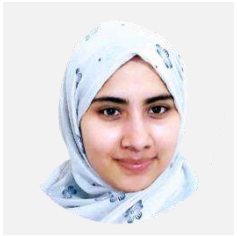
Chapter I : Company presentation

FADO_UP (Buy, sell and Exchange Bitcoin) Company is a global leader in the trade revolution. We operate an investment platform, which is designed for customers who demand lightning-fast trade execution and industry-leading security practices.

Founders:

Ouaziz Hajar

Data Engineer 22years



El amrani El idrissi Omaima

Data Scientist 22 years

Driouche Adnane

Project Manager 22 years

Fakiri Ismail

Data Analyst 22 years

Services:

- Buy Bitcoin With Credit Crad:Purchase Bitcoin at the best possible rates with your credit card

- Buy Bitcoin With Cash:Unable to pay with your Credit Card, no worries, You can pay with cash

-Get Verified & Buy Bitcoin:Verify your account and you're good to go

Why choose FADO_UP:

-Bitcoin Transaction: Quick and efficient bitcoin transactions.

-Instant Exchange: Quick Trade and Withdraw your profits

- Investment Efficiency: Best available managers to ensure best profit.

-Safe and Secure: Safe and secure deposits and withdrawals

-Investment Planning: Affordable Investment Plans

-Instant Trading: Quick and easy trading

Chapter II: Project description

The objective of the project:

our company was brought to carry out this project in order to increase the profit, and for future rational exchanges.

Problem

- the first problem is the rapid increase in the price of bitcoin, and its high price which we have to face very carefully. Bitcoin hit its highest price on January 8, 2021, it was trading at its all-time high of $ 41,850.
- The analysis: we cannot estimate the future value of bitcoin or how and why it reaches new highs.
- Liquidity: Lower liquidity, that is due to the fact that it is new to the market and that there are fewer investors and traders. that can cause ups and downs of cryptos value more often.
- It is a new market: no one can be certain of the future of cryptocurrencies, especially BTC as the synonym for all other cryptos.

proposed solution:

the proposed solution is to carry out this project which aims to predict in real-time the price of bitcoin, the prediction will be done in very short time intervals, in the next n hours or minutes. so that we can take our steps, and improving our services.

Chapter III : Technical analysis

Work environment

HARD Environment

- **Host:** TOSHIBA.
- **Processor:** Intel (R) Core i7.
- **RAM:** 16GB
- **System** type: 64-bit operating system**.**

Software Environment

Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.

Version 2.13

MongoDB is a NoSQL database, the information is stored in documents in JSON format (more precisely BSON, a binary version of JSON) , relatively simple to take in hand and very rich functionally. It allows real-time issues to be addressed in a big data context (clustering, high availability, failure tolerance).

Version 4.2.3

Spark streaming is an extension of the Spark API that allows fast, scalable and fault-tolerant processing, data from various sources such as Twitter, Kafka, HDFS, etc.

Version 2.4.7

Python is a general programming language, dynamic, high-level and interpreted. It supports the object-oriented programming approach to developing applications. It is simple and easy to learn and provides many high-level data structures.

version 3.7

Prediction algorithm:

- RNN

A recurrent neural network (RNN) is a kind of artificial neural network .
Recurrent Networks are designed to recognize patterns in sequences of data, such as text, genomes, handwriting, the spoken word, and numerical time series data emanating from sensors, stock markets, and government agencies.
A recurrent neural network is similar to a traditional neural network except that a memory-state is added to the neurons.

- LSTM

Long short-term memory is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feed-forward neural networks, LSTM has feedback connections. It can process not only single data points but also entire sequences of data.

Bitcoin data:

- We collect our data from the Messari's API.

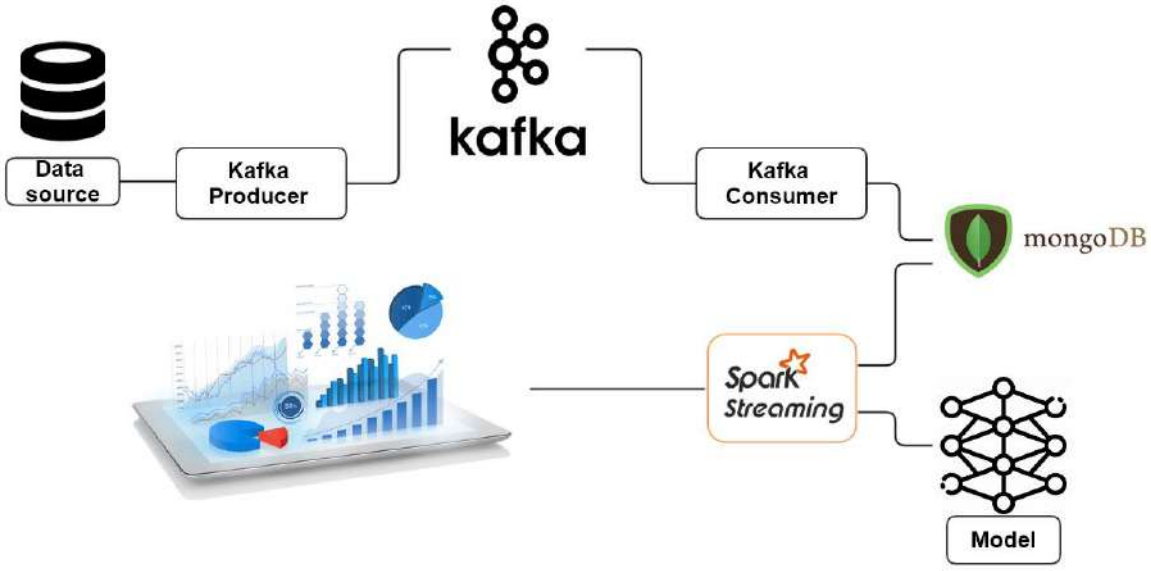
- Our data contains the:

  - **Price_usd :** market price in usd
  - **Timestamp :** the current time of the price
  - **Open :** the price in the beginning of day
  - **High :** the highest level of the price
  - **Low** : the lowest level of the price
  - **Volume** : trading volume in USD
  - **Close** :the price in the end of day

Chapter IIII : Project presentation

Project architecture :

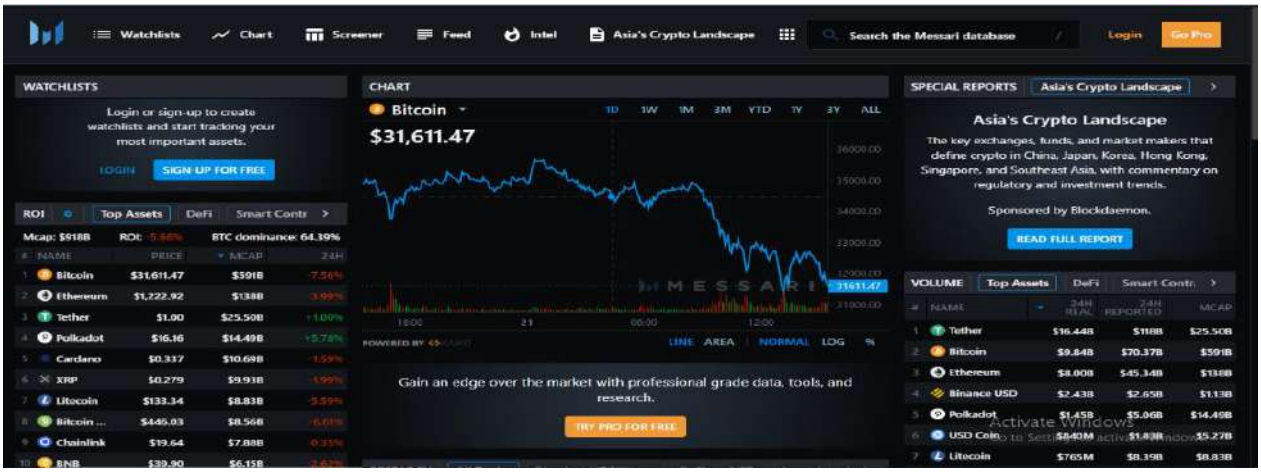
This is the architecture of our project.



Project Implementation :

Data source :

We collect our data from the Messari's API.



We got the latest market data for an asset. This data is also included in the metrics endpoint.

{"status":{"elapsed":2,"timestamp":"2021-01-21T14:34:31.9640340862"},"data":{"id":"1e31218a-e44e-4285-820c-8282ee222035","symbol":"BTC","name":"Bitcoin","slug":"bitcoin","_internal_temp_agora_id":"9793eae6-f374-46b4-8764-c2d224429791","market_data":{"price_usd":31571.742818217568,"price_btc":1,"price_eth":26.006279230430504,"volume_last_24_hours":70368278438.35074,"real_volume_last_24_hours":9873496169.61901,"volume_last_24_hours_overstatement_multiple":7.126986958771063,"percent_change_usd_last_1_hour":-3.4260936088409006,"percent_change_usd_last_24_hours":-7.678108818488548,"percent_change_btc_last_24_hours":0,"percent_change_eth_last_24_hours":-2.820683029636415,"ohlcv_last_1_hour":{"open":32737.58069338857,"high":32786.7083543216 36,"low":31447.282928157038,"close":31571.17643435961,"volume":806986987.7541921},"ohlcv_last_24_hour":{"open":34966.37313645028,"high":35693.61377517762,"low":31267.063041341997,"close":31571.742818217564,"volume":12391220187.391016},"last_trade_at":"2021-01-21T14:34:29.826Z"}}}

This is the schema of our data.

status >	object (ApiStatus)
data ⌄	object (AssetMetricsMarketData)

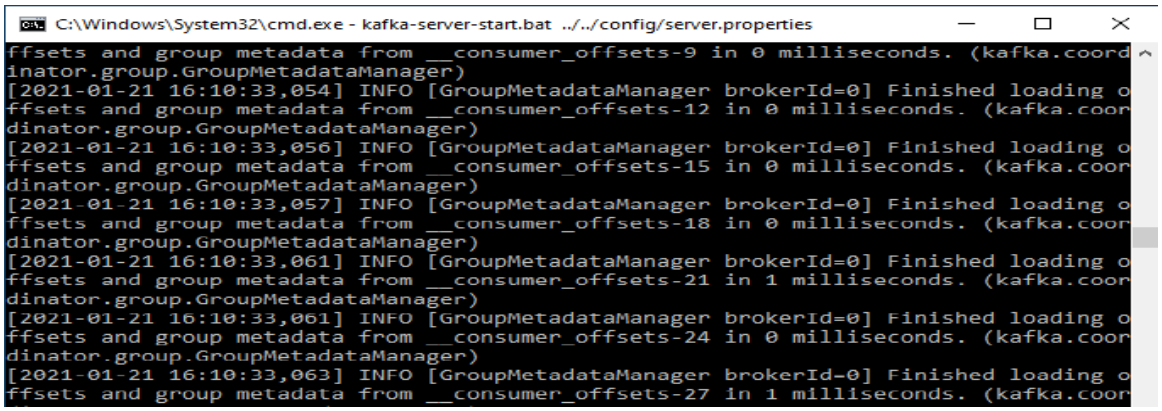
id	string Asset ID. Unique and will never change.
symbol	string The commonly accepted "symbol" for an asset. Not unique, and can change.
name	string Name of asset
slug	string Web URL friendly shorthand slug, alternative to ID. Unique, but can change.
market_data >	object (MarketData)

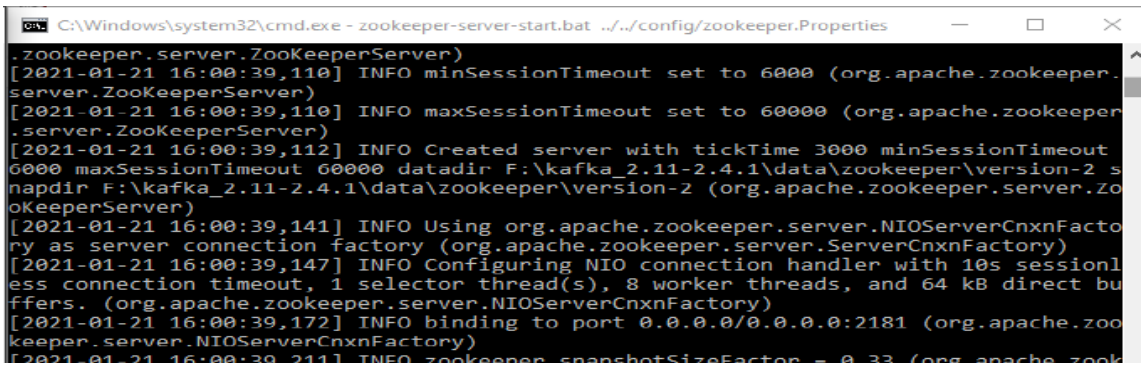
market_data ⌄ object (MarketData)

price_usd	number <double> Market price in USD
price_btc	number <double> Market price in BTC
volume_last_24_hours	number <double> 24h trading volume in USD
real_volume_last_24_hours	number <double> 24h "real" trading volume in USD, read more here: https://messari.io/article/messari-methodology
volume_last_24_hours_overstatement_multiple	number <double> 24h trading volume in USD
percent_change_usd_last_24_hours	any <double> 24h percent change of USD-denominated price. If the return value is 1.23, it means 1.23%.
percent_change_btc_last_24_hours	any <double> 24h percent change of BTC-denominated price. If the return value is 1.23, it means 1.23%.
ohlcv_last_1_hour >	object
ohlcv_last_24_hour >	object

Collecting data / Kafka producer :

First, we start zookeeper server and kafka server:





We are getting data from Messari's AP using Kafka producer.

```python
# Run zookeeper
# zookeeper-server-start.bat ../../config/zookeeper.Properties
# Run kafka
# kafka-server-start.bat ../../config/server.properties

from kafka.producer import KafkaProducer
import urllib.request
import time

producer = KafkaProducer(bootstrap_servers=['localhost:9092'],api_version=(0,10,1))

while 1:
    bitcoineData = urllib.request.urlopen("https://data.messari.io/api/v1/assets/btc/metrics/market-data").read()
    print("Seending Data To Consumer >>>>> ")
    producer.send("test",b""+bitcoineData)
    time.sleep(5)

# bitcoineData = urllib.request.urlopen("https://data.messari.io/api/v1/assets/btc/metrics/market-data").read()
# print("Seending Data >>>>> ",type(bitcoineData))
# producer.send("test",b""+bitcoineData)
```

Data pre-processing / Kafka consumer :

After getting data from Kafka producer, we are filtering it and storing it to MongoDb database.

mongo_spark.py | kafka_mongo _producer.py | kafka_mongo.py

```python
from kafka import KafkaConsumer
from pymongo import MongoClient


inp = "mongodb://localhost:27017/bitcoin.data"
out = "mongodb://localhost:27017/bitcoin.data"

consumer = KafkaConsumer(
        'test',
        bootstrap_servers=['localhost:9092'],
        auto_offset_reset='earliest',
        group_id='bitcoin-group',
        api_version=(0, 10)
    )

client = MongoClient('localhost', 27017)
db = client['bitcoin']
collectionNm = 'data'
collection = db[collectionNm]

for msg in consumer:
    print(">>> Receiving Data From Producer")
    my_json = msg.value.decode('utf8').replace("'", '"')
    data = json.loads(my_json)
    s = json.dumps(data, indent=4, sort_keys=True)
    price = {"price_usd": data["data"]["market_data"]["price_usd"]}
    timestamp = {"timestamp": data["status"]["timestamp"]}
    # data_cleaned = dict(data["data"]["market_data"]["ohlcv_last_1_hour"].items() + price.items())
    data_cleaned = price.copy()
    data_cleaned.update(timestamp)
    data_cleaned.update(data["data"]["market_data"]["ohlcv_last_1_hour"])
    print(data_cleaned)
    print("- - -"*5)
    print("Sending it to mongoDb >>>")
    collection.insert_one(data_cleaned)
```

This is the execution od the code.

```
producer | kafka_mongo
C:\Users\Utilisateur\Desktop\CIISIBD\S9\7-BIG_DATA_Applications\PROJET\Bitcoin_RealTime_Prediction\venv\Scripts\pytho
>>>> Receiving Data ...
Sending it to mongoDb >>>
data_cleaned :  {'timestamp': datetime.datetime(2021, 1, 21, 11, 19, 27, 476981), 'price_usd': 32788.853887854064}
- - -- - -- - -- - -- - -
Sending it to mongoDb >>>
data_cleaned :  {'timestamp': datetime.datetime(2021, 1, 21, 11, 19, 35, 246844), 'price_usd': 32768.19062723364}
- - -- - -- - -- - -- - -
Sending it to mongoDb >>>
data_cleaned :  {'timestamp': datetime.datetime(2021, 1, 21, 11, 20, 3, 349760), 'price_usd': 32769.62215628142}
```

Stocking data at MongoDb :

We store our data in MongoDb. As you can see, each documents contains 7 fields.

```
C:\Windows\system32\cmd.exe - mongo
>
> use bitcoin
switched to db bitcoin
> db.data.count()
49
> db.data.find().pretty()
{
        "_id" : ObjectId("6002b43d8d4f150ea3fe3861"),
        "price_usd" : 37373.48581236813,
        "timestamp" : "2021-01-16T09:39:06.469687164Z",
        "open" : 37323.06832049146,
        "high" : 37669.378496661404,
        "low" : 37118.05968711633,
        "close" : 37372.39211976761,
        "volume" : 459019675.7569362
}
```

Real-time Prediction :

importation des Packages

Entrée [3]:
```
import tensorflow as tf
from pymongo import  MongoClient
import json
import requests
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Activation, Dense, Dropout, LSTM
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.metrics import mean_absolute_error
from sklearn.preprocessing import MinMaxScaler
from apscheduler.schedulers.blocking import BlockingScheduler
```

Connexion Spark MongoDB

Entrée [4]:
```
import findspark
findspark.init(spark_home='D:\spark')
from pyspark import SparkContext,SparkConf
from pyspark.sql import SQLContext
```

Entrée [6]:
```
input_uri = "mongodb://localhost:27017/Bitcoin.data"
output_uri = "mongodb://localhost:27017/Bitcoin.data"

conf=SparkConf()
conf.set('spark.mongodb.input.uri', input_uri)
conf.set('spark.mongodb.output.uri', output_uri)
conf.set('spark.mongodb.input.sampleSize', 50000)
```

Out[6]:
```
<pyspark.conf.SparkConf at 0x216f602d588>
```

Entrée [8]:
```
sc=SparkContext.getOrCreate(conf=conf)
sqlContext=SQLContext(sc)
df=sqlContext.read.format("com.mongodb.spark.sql.DefaultSource").load()
df.printSchema()
```

```
root
 |-- _id: struct (nullable = true)
 |    |-- oid: string (nullable = true)
 |-- price_usd: double (nullable = true)
 |-- timestamp: timestamp (nullable = true)
```

Entrée [9]:
```
df.show(5)
```

```
+--------------------+------------------+--------------------+
|                 _id|         price_usd|           timestamp|
+--------------------+------------------+--------------------+
|[6009634838b90e41...|32788.853887854064|2021-01-21 12:19:...|
|[6009634838b90e41...| 32768.19062723364|2021-01-21 12:19:...|
|[6009636438b90e41...| 32769.62215628142|2021-01-21 12:20:...|
|[6009636b38b90e41...|32765.258738309698|2021-01-21 12:20:...|
|[6009637338b90e41...|32737.118081120694|2021-01-21 12:20:...|
+--------------------+------------------+--------------------+
only showing top 5 rows
```

Entrée [42]:
```
df.count()
```

Out[42]: 596

Entrée [19]:
```
df_p=df.toPandas()
df_p.head()
```

Out[19]:

	_id	price_usd	timestamp
0	(6009634838b90e41dcf0e0f8,)	32788.853888	2021-01-21 12:19:27.476
1	(6009634838b90e41dcf0e0f9,)	32768.190627	2021-01-21 12:19:35.246
2	(6009636438b90e41dcf0e0fa,)	32769.622156	2021-01-21 12:20:03.349
3	(6009636b38b90e41dcf0e0fb,)	32765.258738	2021-01-21 12:20:10.362
4	(6009637338b90e41dcf0e0fc,)	32737.118081	2021-01-21 12:20:17.972

This is our model :

Model

Entrée [23]:
```
def line_plot(line1, line2, label1=None, label2=None, title='', lw=2):
    fig, ax = plt.subplots(1, figsize=(13, 7))
    ax.plot(line1, label=label1, linewidth=lw)
    ax.plot(line2, label=label2, linewidth=lw)
    ax.set_ylabel('prix [EUR]', fontsize=14)
    ax.set_title(title, fontsize=16)
    ax.legend(loc='best', fontsize=16)
```

Entrée [24]:
```
def train_test_split(df, test_size=0.2):
    split_row = len(df) - int(test_size * len(df))
    train_data = df.iloc[:split_row]
    test_data = df.iloc[split_row:]
    return train_data, test_data
```

Entrée [25]:
```
def normalise_zero_base(df):
    scaler = MinMaxScaler()
    return scaler.fit_transform(df.iloc[:,0:6])

def normalise_min_max(df):
    return (df - df.min()) / (data.max() - df.min())
```

Entrée [26]:
```
def extract_window_data(df, window_len=5, zero_base=True):
    window_data = []
    for idx in range(len(df) - window_len):
        tmp = df[idx: (idx + window_len)].copy()
        if zero_base:
            tmp = normalise_zero_base(tmp)
        window_data.append(tmp)
    return np.array(window_data)
```

Entrée [27]:
```
def prepare_data(df, target_col, window_len=10, zero_base=True, test_size=0.2):

    train_data, test_data = train_test_split(df, test_size=test_size)

    X_train = extract_window_data(train_data, window_len, zero_base)
    X_test = extract_window_data(test_data, window_len, zero_base)

    y_train = train_data[target_col][window_len:].values
    y_test = test_data[target_col][window_len:].values

    if zero_base:
        y_train = y_train / train_data[target_col][:-window_len].values - 1
        y_test = y_test / test_data[target_col][:-window_len].values - 1

    return train_data, test_data, X_train, X_test, y_train, y_test
```

Entrée [28]:
```
def build_lstm_model(input_data, output_size, neurons=100, activ_func='linear', dropout=0.2, loss='mse', optimizer='adam'):
    print('build_lstm_model')
    print(input_data.shape)
    model = tf.keras.Sequential()
    model.add(LSTM(neurons, input_shape=(input_data.shape[1], input_data.shape[2])))
    print('>>>>>>>>>> ')
    model.add(Dropout(dropout))
    model.add(Dense(units=output_size))
    model.add(Activation(activ_func))
    model.compile(loss=loss, optimizer=optimizer)
    return model
```

Entrée [33]:
```
def prepare_hist(df):
    df = df.toPandas()
    df = df[["timestamp", "price_usd"]]
    df = df.set_index("timestamp")
    return df
```

Entrée [40]:
```
all_programme(df)
```

```
**********prepare data**********
**********train, test**********
build_lstm_model
(315, 5, 1)
>>>>>>>>>>
**********lancer_training**********
Epoch 1/20
315/315 [==============================]315/315 [==============================] - 1s 3ms/step - loss: 3.8457e-04

Epoch 2/20
315/315 [==============================]315/315 [==============================] - 0s 435us/step - loss: 2.7983e-04

Epoch 3/20
315/315 [==============================]315/315 [==============================] - 0s 400us/step - loss: 1.5676e-04

Epoch 4/20
315/315 [==============================]315/315 [==============================] - 0s 432us/step - loss: 1.5980e-04

Epoch 5/20
315/315 [==============================]315/315 [==============================] - 0s 448us/step - loss: 1.1716e-04

Epoch 6/20
315/315 [==============================]315/315 [==============================] - 0s 441us/step - loss: 9.5757e-05

Epoch 7/20
315/315 [==============================]315/315 [==============================] - 0s 419us/step - loss: 1.0270e-04

Epoch 8/20
315/315 [==============================]315/315 [==============================] - 0s 463us/step - loss: 7.4907e-05

Epoch 9/20
315/315 [==============================]315/315 [==============================] - 0s 508us/step - loss: 7.2497e-05
```

Ac
Go

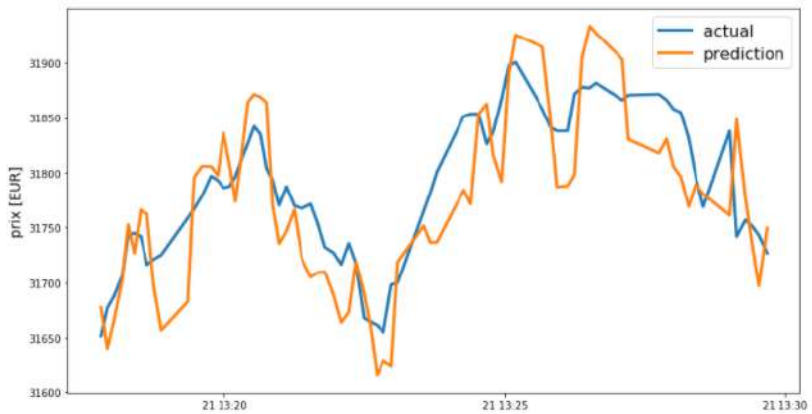
Visualization :

```
Entrée [39]: #sched = BlockingScheduler()
#@sched.scheduled_job('interval', seconds=60)#@sched.scheduled_job('cron', day_of_week='mon-fri', hour=10)
def all_programme(df):
    #global hist
    hist= prepare_hist(df)
    hist=hist.iloc[-400:]

    print('**********************prepare data*************************')
    train, test = train_test_split(hist, test_size=0.2)

    print('*********************train, test*******************************')
    train, test, X_train, X_test, y_train, y_test = prepare_data(hist, target_col='price_usd', window_len=5)

    X_train = np.asarray(X_train).astype(np.float32)
    y_train=np.asarray(y_train).astype(np.float32)

    model = build_lstm_model(X_train, output_size=1)

    print('******************lancer_training************************')
    history = model.fit( X_train, y_train, epochs=20, batch_size=32, verbose=1, shuffle=True)

    print('***************************tester************************')
    targets = test['price_usd'][5:]
    preds = model.predict(X_test).squeeze()
    print('mse:',mean_absolute_error(preds, y_test))
    preds = test['price_usd'].values[:-5] * (preds + 1)
    preds = pd.Series(index=targets.index, data=preds)

    print('***************************Plot************************')
    line_plot(targets, preds, 'actual', 'prediction', lw=3)

    print('****************to_save*************')
    model.save('C:\\Users\\Utilisateur\\Desktop\\CIISIBD\\S9\\7-BIG_DATA_Applications\\PROJET\\Spark_NoteBook\\my_Model.h5')
    print('*<saved>*')

#sched.start()
```

Activer
Go to Settin

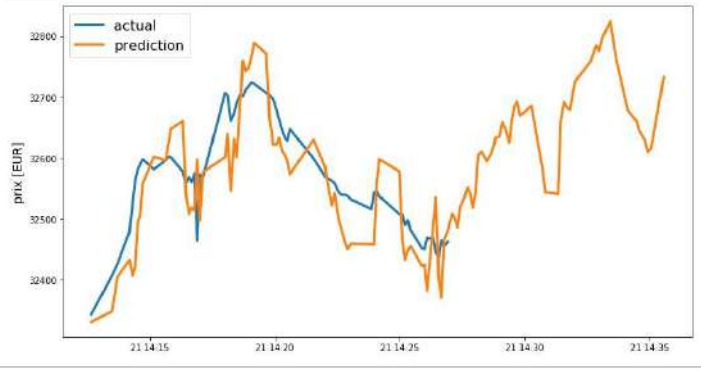
After training the model, our prediction is as follows.

```
Epoch 19/20
315/315 [==============================]315/315 [==============================] - 0s 448us/step - loss: 3.9292e-05

Epoch 20/20
315/315 [==============================]315/315 [==============================] - 0s 444us/step - loss: 3.3396e-05

***************************tester************************
mse: 0.0011574591208949635
***************************plot************************
****************to_save*************
*<saved>*
```



Then finally we are visualize our data

In [74]:
```
actual = df[-60:]
preds = model.predict(future_data).squeeze()
line_plot(actual, preds, 'actual', 'prediction', lw=3)
```



Conclusion

In a nutshell, this project has been an excellent and rewarding experience. we can conclude that there has been a lot of things that we have learned from our work on this project. As we have no prior experience with Kafka and spark whatsoever we believe our time spent in research and discovering was well worth it and contributed to finding an acceptable solution to build a complete project. There are two main things that we have learned the importance of time-management skills and self-motivation.