BigData Analytics Project Report

JANUARY 10

Institute of Business Administration
Ibrahim Abdurrab &
Sadiq A. Hameed

IBA

Introduction

The project is based on creating a ***lab to demonstrate the capabilities of Spark***. In order to achieve the goal, we are going to perform the following activities.

- Develop the environment
    - Installation of docker engine
    - Download the Hadoop docker container from cloudera
    - Setup docker environment
- Import our dataset into the environment
- Develop a list of commands for Spark
- Implement these commands over dataset and record the result

Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance

Data Set

In order to develop the report and show the power of Spark, we have selected the dataset from Kaggle. It belongs to the **Crypto Finance**. Crypto currencies are emerging concept. A cryptocurrency is a digital asset designed to work as a medium of exchange wherein individual coin ownership records are stored in a ledger existing in a form of computerized database using strong security mechanism in de-centralized manner.

Our dataset is based on the crypto currency exchange (bitfinex) that holds the historical data the price of 321 different currencies at 1-minute(s) resolution, from the year 2013 to the date of download.
Dataset link : https://www.kaggle.com/tencars/392-crypto-currency-pairs-at-minute-resolution

(The data in the CSV files is the raw output of the Bitfinex API. This means, there are no timestamps for time periods in which the exchange was down. Also, if there were time periods without any activity or trades there will be no timestamp as well.)

OHLC (Open, High, Low, Close)

These terms are used in exchange to describe the price during a time-period, open and close represent the price at the start and end respectively, whereas low and high represents Nadir and Apex during that time-period.

Data format

The data is presented into CSV files, for file for each currency pair. Files have following columns:

- *Time:* the start time of the transaction duration (one minute) in integral format, the number of milliseconds elapsed since Jan 1, 1970.
- *Open:* the price at the start of transaction period
- *Close:* the price at the end of transaction period
- *Low:* the lowest price during the transaction period
- *High:* the highest period during the transaction period
- *Volume:* total number of units exchange during the transaction period

Snapshot of a CSV file

The following snapshot of Zilliqa-USD pair

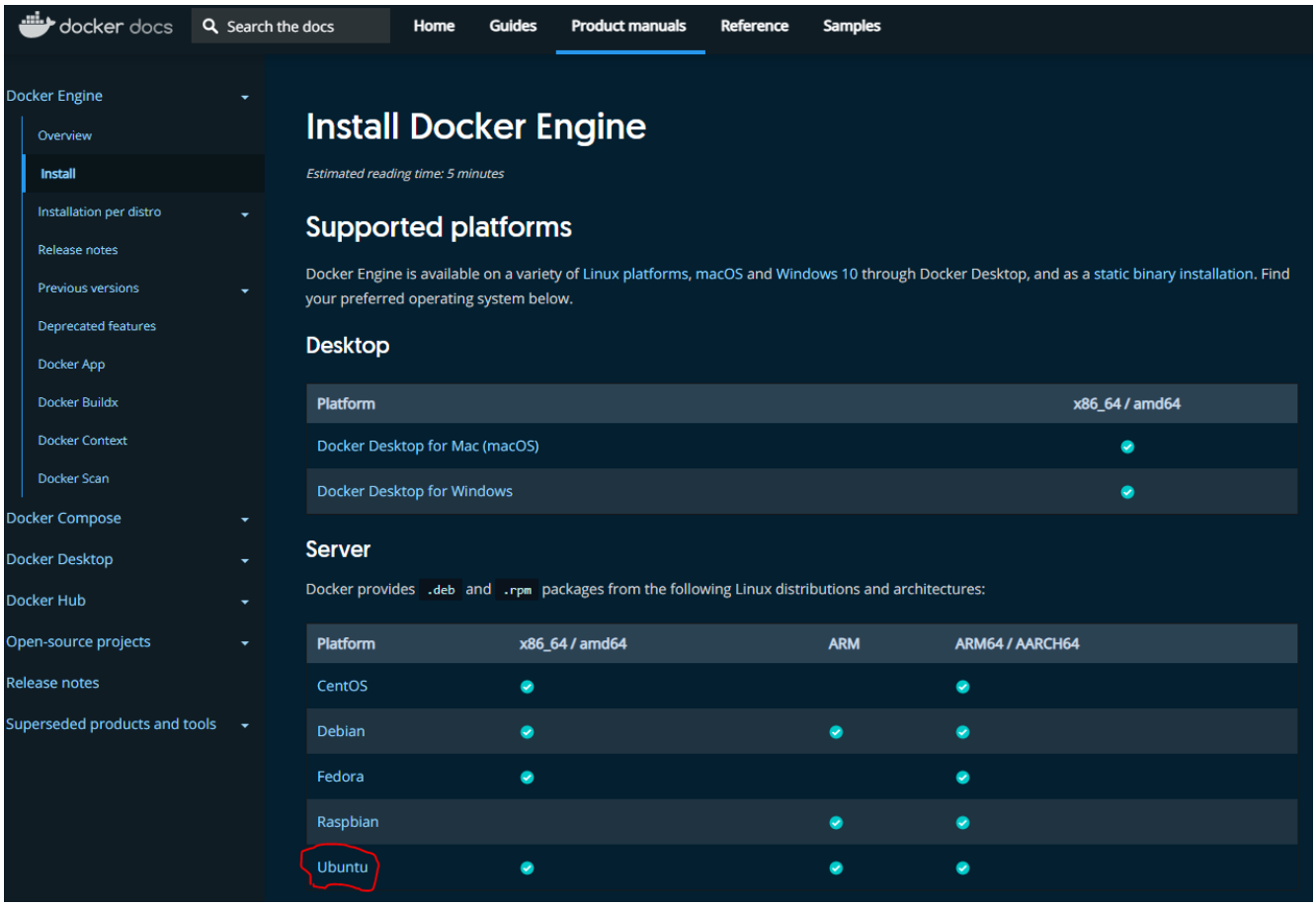
time	open	close	high	low	volume
1533792660000	0.040988	0.04099	0.04099	0.040988	2000
1533802620000	0.04	0.039128	0.04	0.039128	2547.45
1533808380000	0.0383	0.03828	0.0383	0.03828	549.1154187
1533810660000	0.0419	0.042	0.042	0.0419	6600.358705
1533810720000	0.042	0.042	0.042	0.042	11221.66992
1533811980000	0.044999	0.044999	0.044999	0.044999	700
1533812040000	0.045	0.045	0.045	0.045	411.313292
1533821100000	0.044896	0.044896	0.044896	0.044896	1198.427896
1533821820000	0.044	0.045	0.045	0.044	1042.029377
1533822000000	0.03832	0.036	0.03832	0.036	3938.834699
1533833100000	0.044998	0.044998	0.044998	0.044998	393.8720254
1533837900000	0.044999	0.044999	0.044999	0.044999	3118.526808
1533867840000	0.041157	0.041156	0.041157	0.041156	917.084041
1533874800000	0.04001	0.04	0.04001	0.04	405.083753
1533902460000	0.04	0.04	0.04	0.04	1536.1003
1533917160000	0.0409	0.0409	0.0409	0.0409	500
1533920160000	0.04	0.04	0.04	0.04	689.429
1533925320000	0.04	0.039009	0.04	0.039009	248
1534010940000	0.042	0.042	0.042	0.042	904.351
1534014960000	0.041999	0.041999	0.041999	0.041999	290
1534032960000	0.041998	0.03901	0.041999	0.03901	2112.293825
1534033080000	0.041999	0.042	0.042	0.041999	833.350601
1534039320000	0.037	0.035642	0.037	0.035642	2174.568312
1534054200000	0.042	0.042	0.042	0.042	261.2826603
1534063140000	0.034254	0.034254	0.034254	0.034254	11910.71726
1534063200000	0.034	0.033	0.034	0.033	1070
1534063320000	0.032075	0.032075	0.032075	0.032075	11537.14393
1534135980000	0.042	0.042655	0.042655	0.042	1668.881756
1534136160000	0.043352	0.044874	0.044874	0.043352	1282.476274
1534148220000	0.033926	0.033337	0.033926	0.033337	2099.804153
1534158120000	0.036	0.036	0.036	0.036	3000
1534163940000	0.036	0.036	0.036	0.036	4701.228

Environment

Installation of Docker Engine and Cloudera QuickStart

Installation of Docker

- Access docs.docker.com
- Click on Install Docker
- Select Docker for Linux (https://docs.docker.com/get-docker/)
- Select Ubuntu from menu (https://docs.docker.com/engine/install/)



To install Docker Engine, you need the 64-bit version of one of these Ubuntu versions:

- Ubuntu Focal 20.04
- Ubuntu Eoan 19.10
- Ubuntu Bionic 18.04 (LTS)
- Ubuntu Xenial 16.04 (LTS)

To check dependencies, you can run the following command

cat /etc/*release*

```
sadiq@ubuntu:~$ cat /etc/*release*
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=20.04
DISTRIB_CODENAME=focal
DISTRIB_DESCRIPTION="Ubuntu 20.04.1 LTS"
NAME="Ubuntu"
VERSION="20.04.1 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04.1 LTS"
VERSION_ID="20.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=focal
UBUNTU_CODENAME=focal
sadiq@ubuntu:~$
```

uninstall older versions: Older versions of Docker were called docker, docker.io, or docker-engine. Uninstall them. New version is docker-ce

sudo apt-get remove docker docker-engine docker.io containerd runc

To uninstall previously installed docker-ce, use:

sudo apt-get purge docker-ce docker-ce-cli containerd.io

Images, containers, volumes, or customized configuration files on your host are not automatically removed. To delete all images, containers, and volumes:

sudo rm -rf /var/lib/docker

Execute the following commands to setup the repository

sudo apt-get update

```
[sudo] password for sadiq:
Get:1 https://download.docker.com/linux/ubuntu focal InRelease [36.2 kB]
Get:2 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages [6,247 B]
Get:3 http://security.ubuntu.com/ubuntu focal-security InRelease [109 kB]
Hit:4 http://us.archive.ubuntu.com/ubuntu focal InRelease
Get:5 http://us.archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:6 http://security.ubuntu.com/ubuntu focal-security/main amd64 DEP-11 Metadata [24.3 kB]
Get:7 http://us.archive.ubuntu.com/ubuntu focal-backports InRelease [101 kB]
Get:8 http://security.ubuntu.com/ubuntu focal-security/universe amd64 DEP-11 Metadata [56.7 kB]
Get:9 http://us.archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [713 kB]
Get:10 http://us.archive.ubuntu.com/ubuntu focal-updates/main i386 Packages [393 kB]
Get:11 http://us.archive.ubuntu.com/ubuntu focal-updates/main amd64 DEP-11 Metadata [264 kB]
Get:12 http://us.archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages [709 kB]
Get:13 http://us.archive.ubuntu.com/ubuntu focal-updates/universe i386 Packages [525 kB]
Get:14 http://us.archive.ubuntu.com/ubuntu focal-updates/universe amd64 DEP-11 Metadata [205 kB]
Get:15 http://us.archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 DEP-11 Metadata [2,468 B]
Get:16 http://us.archive.ubuntu.com/ubuntu focal-backports/universe amd64 DEP-11 Metadata [1,768 B]
Fetched 3,260 kB in 15s (217 kB/s)
Reading package lists... Done
sadiq@ubuntu:~$
```

```
sudo apt-get install \
   apt-transport-https \
   ca-certificates \
   curl \
   gnupg-agent \
   software-properties-common

sudo apt-key fingerprint 0EBFCD88
```

Now, Install using the convenience script:

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
```

Now make sure the docker is running properly; by running following command;

```
sudo docker run hello-world
```

Installation of Cloudera Quick Container

Please run the command;

```
docker pull cloudera/quickstart:latest
```

it would start the downloading, the total size of container is 4.5GB so it will take time.

Star the clouders-quickstart container using following command;

```
sudo docker run --hostname=quickstart.cloudera --privileged=true -t -i -p 8888:8888 -p 7180:7180
cloudera/quickstart  /usr/bin/docker-quickstart
```

Option Description

--hostname=quickstart.cloudera Required: Pseudo-distributed configuration assumes this hostname.

--privileged=true Required: For HBase, MySQL-backed Hive metastore, Hue, Oozie, Sentry, and Cloudera Manager.

-t Required: Allocate a pseudoterminal. Once services are started, a Bash shell takes over. This switch starts a terminal emulator to run the services.

-i Required: If you want to use the terminal, either immediately or connect to the terminal later.

-p 8888:8888 Recommended: Map the Hue port in the guest to another port on the host.

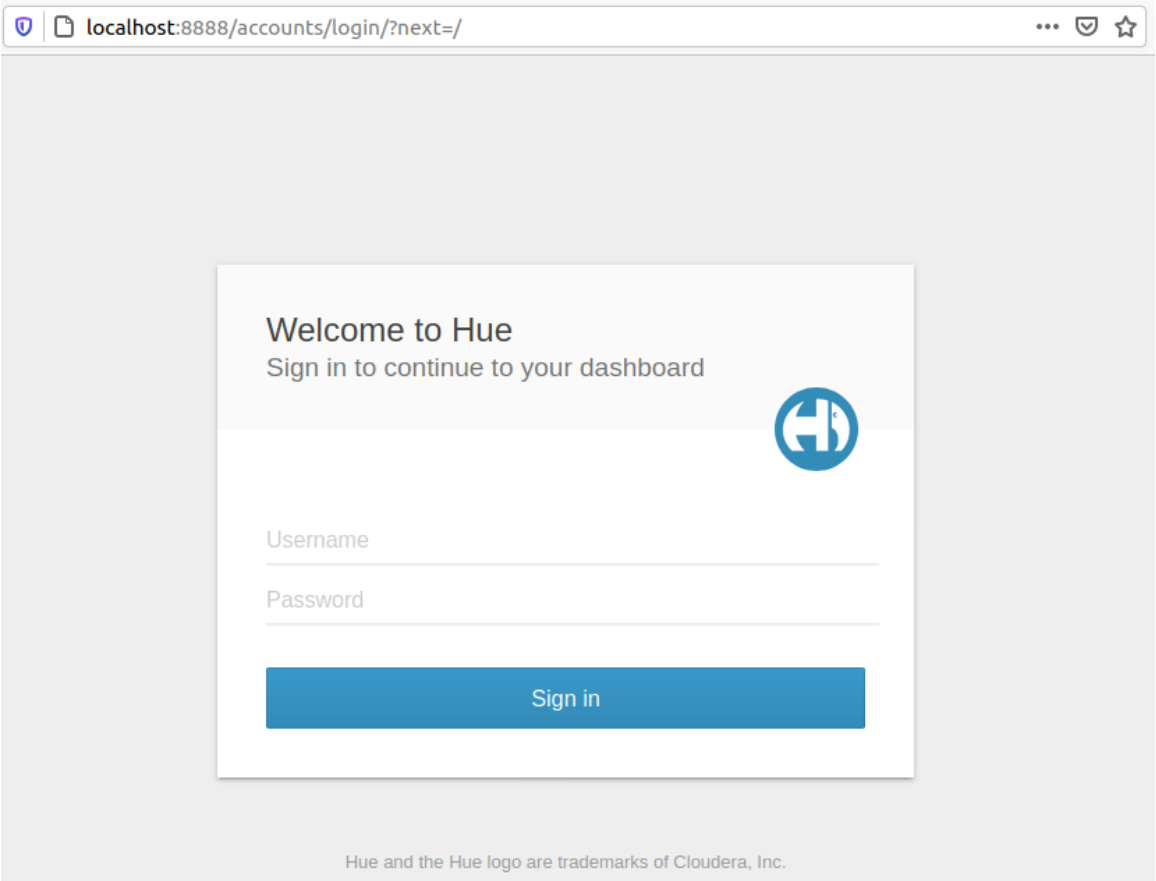
```
sadiq@ubuntu:~$ sudo docker run --hostname=quickstart.cloudera --privileged=true -t -i -p 8888:8888
-p 7180:7180  cloudera/quickstart  /usr/bin/docker-quickstart
Starting mysqld:                                           [  OK  ]

if [ "$1" == "start" ] ; then
    if [ "${EC2}" == 'true' ]; then
        FIRST_BOOT_FLAG=/var/lib/cloudera-quickstart/.ec2-key-installed
        if [ ! -f "${FIRST_BOOT_FLAG}" ]; then
            METADATA_API=http://169.254.169.254/latest/meta-data
            KEY_URL=${METADATA_API}/public-keys/0/openssh-key
            SSH_DIR=/home/cloudera/.ssh
            mkdir -p ${SSH_DIR}
            chown cloudera:cloudera ${SSH_DIR}
            curl ${KEY_URL} >> ${SSH_DIR}/authorized_keys
            touch ${FIRST_BOOT_FLAG}
        fi
    fi
    if [ "${DOCKER}" != 'true' ]; then
        if [ -f /sys/kernel/mm/redhat_transparent_hugepage/defrag ]; then
            echo never > /sys/kernel/mm/redhat_transparent_hugepage/defrag
        fi

        cloudera-quickstart-ip
        HOSTNAME=quickstart.cloudera
        hostname ${HOSTNAME}
        sed -i -e "s/HOSTNAME=.*/HOSTNAME=${HOSTNAME}/" /etc/sysconfig/network
    fi

    (
        cd /var/lib/cloudera-quickstart/tutorial;
        nohup python -m SimpleHTTPServer 80 &
    )

    # TODO: check for expired CM license and update config.js accordingly
fi
+ '[' start == start ']'
+ '[' '' == true ']'
+ '[' true '!=' true ']'
+ cd /var/lib/cloudera-quickstart/tutorial
```

```
nohup: appending output to `nohup.out'
JMX enabled by default
Using config: /etc/zookeeper/conf/zoo.cfg
Starting zookeeper ... STARTED
starting datanode, logging to /var/log/hadoop-hdfs/hadoop-hdfs-datanode-quickstart.cloudera.out
Started Hadoop datanode (hadoop-hdfs-datanode):                [  OK  ]
starting journalnode, logging to /var/log/hadoop-hdfs/hadoop-hdfs-journalnode-quickstart.cloudera.ou
t
Started Hadoop journalnode:                                    [  OK  ]
starting namenode, logging to /var/log/hadoop-hdfs/hadoop-hdfs-namenode-quickstart.cloudera.out
Started Hadoop namenode:                                       [  OK  ]
starting secondarynamenode, logging to /var/log/hadoop-hdfs/hadoop-hdfs-secondarynamenode-quickstart
.cloudera.out
Started Hadoop secondarynamenode:                              [  OK  ]

Setting HTTPFS_HOME:          /usr/lib/hadoop-httpfs
Using   HTTPFS_CONFIG:        /etc/hadoop-httpfs/conf
Sourcing:                    /etc/hadoop-httpfs/conf/httpfs-env.sh
Using   HTTPFS_LOG:           /var/log/hadoop-httpfs/
Using   HTTPFS_TEMP:           /var/run/hadoop-httpfs
Setting HTTPFS_HTTP_PORT:     14000
Setting HTTPFS_ADMIN_PORT:     14001
Setting HTTPFS_HTTP_HOSTNAME: quickstart.cloudera
Setting HTTPFS_SSL_ENABLED: false
Setting HTTPFS_SSL_KEYSTORE_FILE:     /var/lib/hadoop-httpfs/.keystore
Setting HTTPFS_SSL_KEYSTORE_PASS:     password
Using   CATALINA_BASE:       /var/lib/hadoop-httpfs/tomcat-deployment
Using   HTTPFS_CATALINA_HOME:       /usr/lib/bigtop-tomcat
Setting CATALINA_OUT:        /var/log/hadoop-httpfs//httpfs-catalina.out
Using   CATALINA_PID:        /var/run/hadoop-httpfs/hadoop-httpfs-httpfs.pid

Using   CATALINA_OPTS:
Adding to CATALINA_OPTS:     -Dhttpfs.home.dir=/usr/lib/hadoop-httpfs -Dhttpfs.config.dir=/etc/hadoo
p-httpfs/conf -Dhttpfs.log.dir=/var/log/hadoop-httpfs/ -Dhttpfs.temp.dir=/var/run/hadoop-httpfs -Dht
tpfs.admin.port=14001 -Dhttpfs.http.port=14000 -Dhttpfs.http.hostname=quickstart.cloudera
Using CATALINA_BASE:   /var/lib/hadoop-httpfs/tomcat-deployment
Using CATALINA_HOME:   /usr/lib/bigtop-tomcat
Using CATALINA_TMPDIR: /var/run/hadoop-httpfs
Using JRE_HOME:        /usr/java/jdk1.7.0_67-cloudera
```

```
Setting OOZIE_HTTP_HOSTNAME: quickstart.cloudera
Setting OOZIE_HTTP_PORT:     11000
Setting OOZIE_ADMIN_PORT:     11001
Using   OOZIE_HTTPS_PORT:     11443
Setting OOZIE_BASE_URL:      http://quickstart.cloudera:11000/oozie
Using   CATALINA_BASE:       /var/lib/oozie/tomcat-deployment
Setting OOZIE_HTTPS_KEYSTORE_FILE:     /var/lib/oozie/.keystore
Using   OOZIE_HTTPS_KEYSTORE_PASS:     password
Setting OOZIE_INSTANCE_ID:       quickstart.cloudera
Setting CATALINA_OUT:        /var/log/oozie/catalina.out
Using   CATALINA_PID:        /var/run/oozie/oozie.pid

Using   CATALINA_OPTS:        -Doozie.https.port=11443 -Doozie.https.keystore.pass=password -Xmx1024
m -Doozie.https.port=11443 -Doozie.https.keystore.pass=password -Xmx1024m -Dderby.stream.error.file=
/var/log/oozie/derby.log
Adding to CATALINA_OPTS:     -Doozie.home.dir=/usr/lib/oozie -Doozie.config.dir=/etc/oozie/conf -Doo
zie.log.dir=/var/log/oozie -Doozie.data.dir=/var/lib/oozie -Doozie.instance.id=quickstart.cloudera -
Doozie.config.file=oozie-site.xml -Doozie.log4j.file=oozie-log4j.properties -Doozie.log4j.reload=10
-Doozie.http.hostname=quickstart.cloudera -Doozie.admin.port=11001 -Doozie.http.port=11000 -Doozie.h
ttps.port=11443 -Doozie.base.url=http://quickstart.cloudera:11000/oozie -Doozie.https.keystore.file=
/var/lib/oozie/.keystore -Doozie.https.keystore.pass=password -Djava.library.path=:/usr/lib/hadoop/l
ib/native:/usr/lib/hadoop/lib/native

Using CATALINA_BASE:   /var/lib/oozie/tomcat-deployment
Using CATALINA_HOME:   /usr/lib/bigtop-tomcat
Using CATALINA_TMPDIR: /var/lib/oozie
Using JRE_HOME:        /usr/java/jdk1.7.0_67-cloudera
Using CLASSPATH:       /usr/lib/bigtop-tomcat/bin/bootstrap.jar
Using CATALINA_PID:    /var/run/oozie/oozie.pid
Starting Solr server daemon:                                   [  OK  ]
Using CATALINA_BASE:   /var/lib/solr/tomcat-deployment
Using CATALINA_HOME:   /usr/lib/solr/../bigtop-tomcat
Using CATALINA_TMPDIR: /var/lib/solr/
Using JRE_HOME:        /usr/java/jdk1.7.0_67-cloudera
Using CLASSPATH:       /usr/lib/solr/../bigtop-tomcat/bin/bootstrap.jar
Using CATALINA_PID:    /var/run/solr/solr.pid
Started Impala Catalog Server (catalogd) :                     [  OK  ]
Started Impala Server (impalad):                               [  OK  ]
[root@quickstart /]#
```

Now start the Hue, into web browser; via following URL

http://localhosr:8888



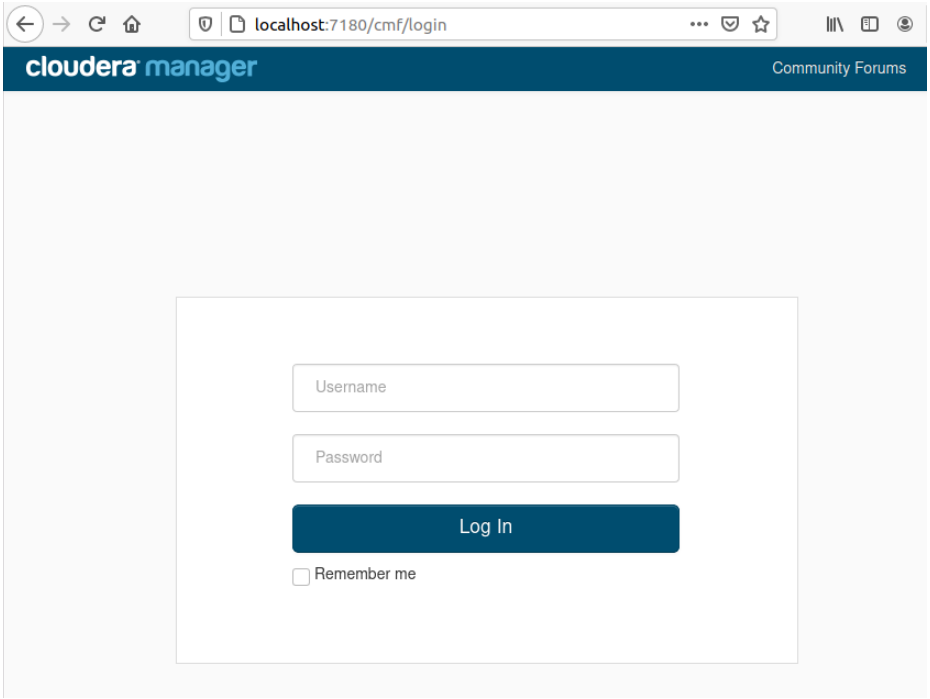
Use credentials as username=cloudera, password=cloudera. If you logged in successfully means the cloudera-quickstart container has been installed and configured successfully.

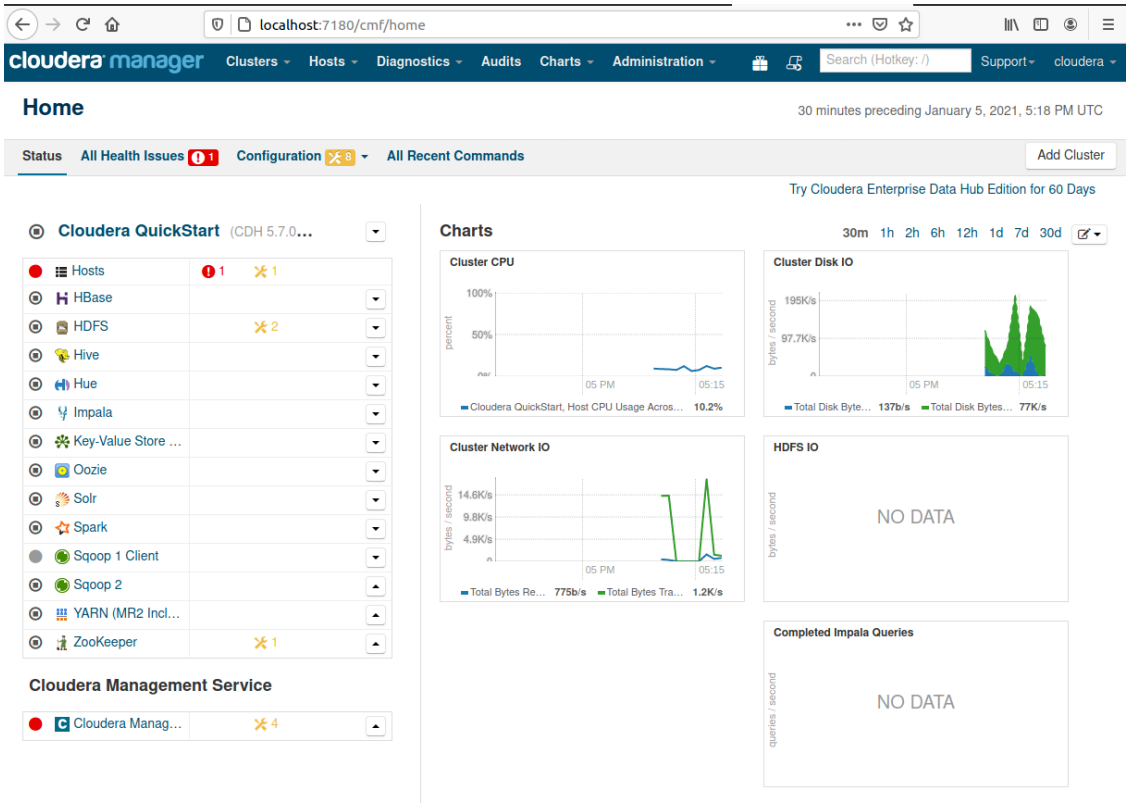
Installation of Cloudera Manager

Cloudera Manager is an application that manages Cloudera deployments and clusters.
It improves performance, efficiency and reduce administrative costs. The command to
install Cloudera Manager is;
sudo /home/cloudera/cloudera-manager --express
Note: In order to install cloudera manager (the recommended processors at least 2 if you are using a VM and) the RAM should be 8GB

```
[root@quickstart /]# sudo /home/cloudera/cloudera-manager --express
[QuickStart] Shutting down CDH services via init scripts...
kafka-server: unrecognized service
JMX enabled by default
Using config: /etc/zookeeper/conf/zoo.cfg
[QuickStart] Disabling CDH services on boot...
error reading information on service kafka-server: No such file or directory
[QuickStart] Starting Cloudera Manager server...
[QuickStart] Waiting for Cloudera Manager API...
[QuickStart] Starting Cloudera Manager agent...
[QuickStart] Configuring deployment...
Submitted jobs: 14
[QuickStart] Deploying client configuration...
Submitted jobs: 16
[QuickStart] Starting Cloudera Management Service...
Submitted jobs: 24
[QuickStart] Enabling Cloudera Manager daemons on boot...
--------------------------------------------------------------------------------

Success! You can now log into Cloudera Manager from the QuickStart VM's browser:

    http://quickstart.cloudera:7180

    Username: cloudera
    Password: cloudera

[root@quickstart /]#
```

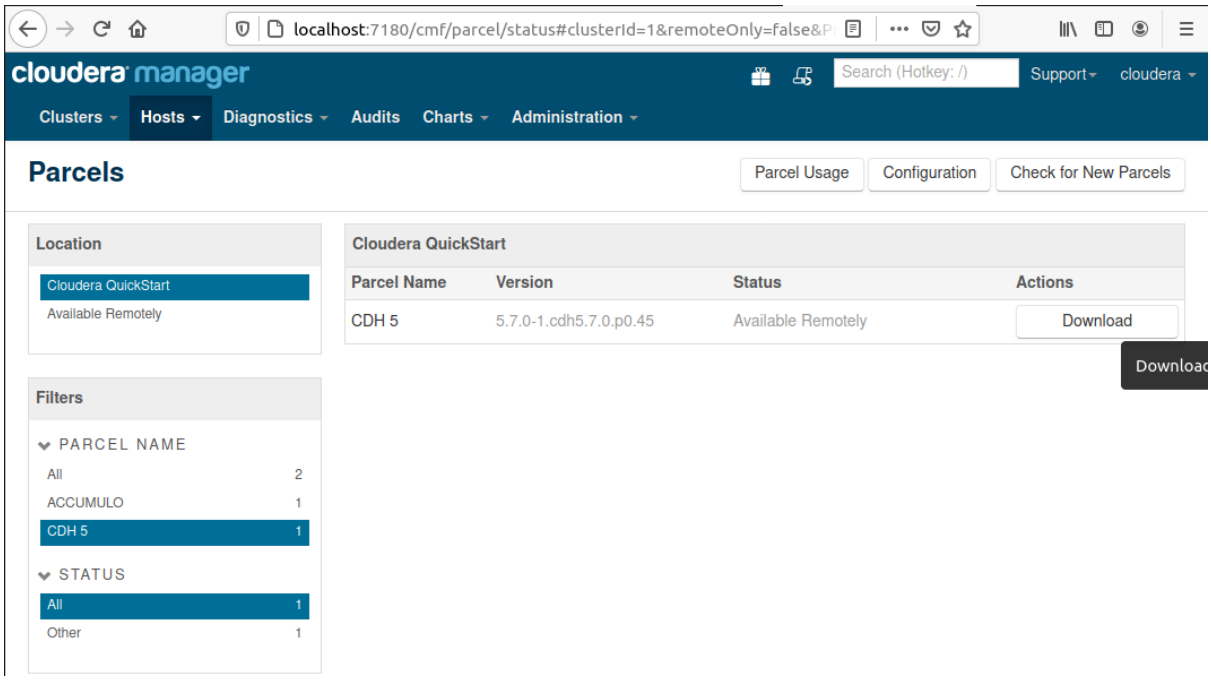
Now goto to the URL http://localhost:7180, it will open the login window, again use the same credentials username=cloudera, password=cloudera



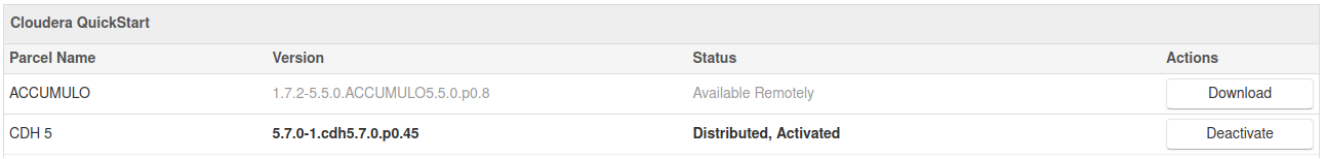


Updating Cloudera Distribution Hadoop (CDH)

Updating CDH to the latest version is a mandatory step to run Apache Spark on Cloudera docker without any errors. The steps to update CDH are listed below:
Open up the Cloudera Manager and go to Parcels.



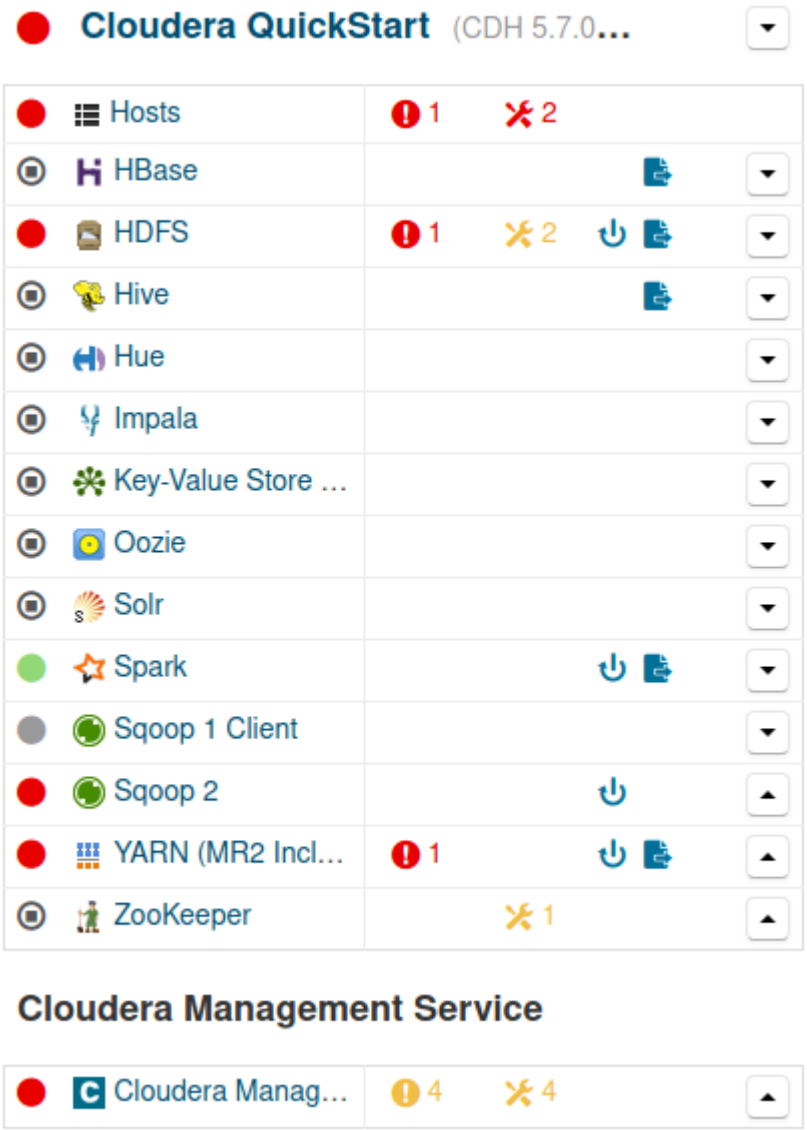
Click on download. When the download is complete, click on distribute and then activate.

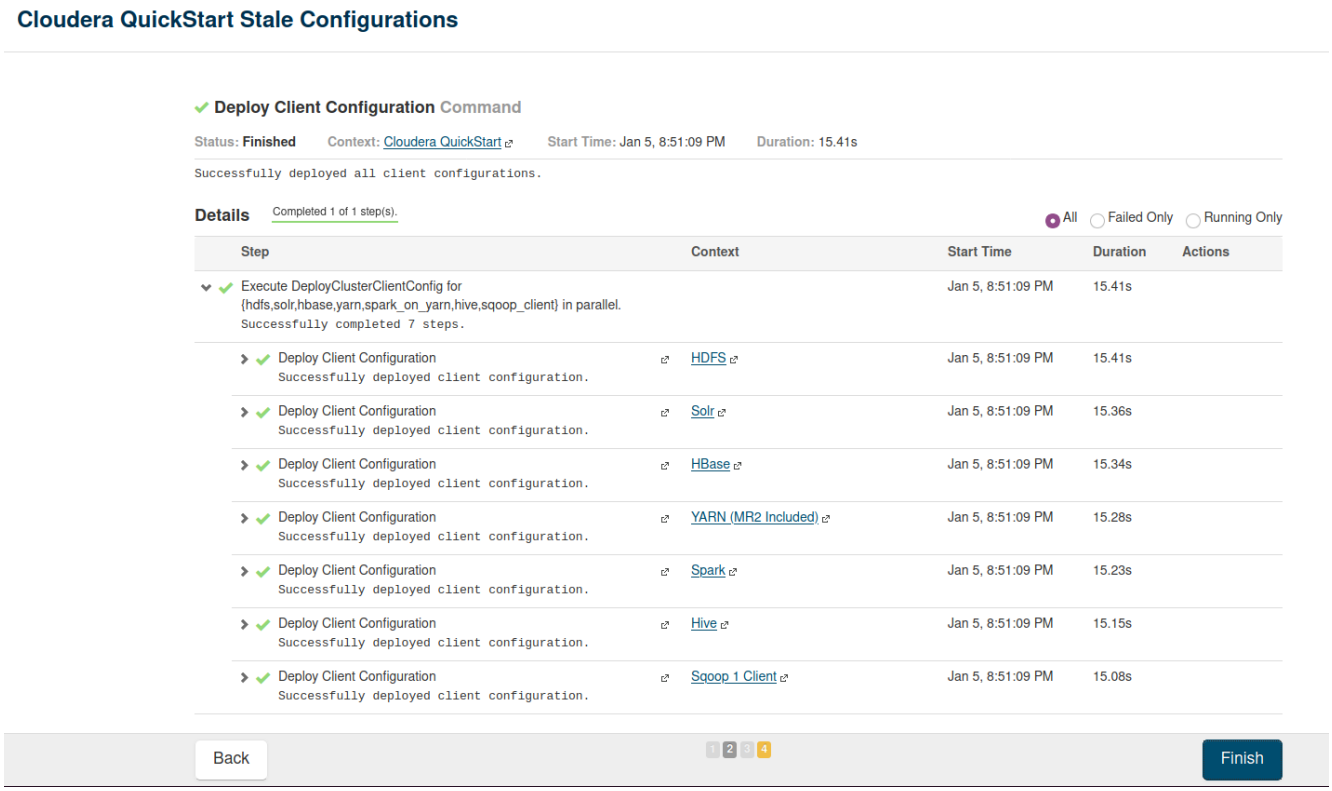


Cloudera QuickStart

Parcel Name	Version	Status	Actions
ACCUMULO	1.7.2-5.5.0.ACCUMULO5.5.0.p0.8	Available Remotely	Download
CDH 5	**5.7.0-1.cdh5.7.0.p0.45**	**Distributed, Activated**	Deactivate

Now from the home page of cloudera manager, start HDFS, YARN and SPARK.





Now all is set, our HDFS and Spark are running perfectly. Next is to load data.

Data Analysis using Apache Spark

After setting up the environment, the next step is to start the spark job on cloudera cluster. As our project involves doing some analysis on the dataset, we have utilized 'Sublime' editor (personal choice) to create a pyspark script.

```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, DoubleType
from pyspark.sql.functions import col
from pyspark.sql.functions import udf
```

The above snippet of code written on sublime text editor shows calling of important libraries which will be used further in the analysis. The 'udf' or (user defined function) will help us to create personalized functions which can then be deployed for further analysis.

Our analysis is six folds:

1- Round off open, close, high, low and volume column's values
2- Calculating the min max and average of the volume
3- Dropping duplicate rows (if any) from the data set
4- Create a new column that has sum of open and close, divided by volume as defined by the formula: (open + close) / close
5- Create two new files, one that has data set where the volume value is less than the average value of volume and vice versa
6- Sort the data based on Time

To achieve task one and four, some user defined functions 'udf' are made. The code snippet showing the functions is given below:

```
# UDF for rounding values
def round_off(n):
    return round(n,4)

# UDF for (open x close) / close
def calculations(_open, close):
    return (_open + close) / close
```

The 'udf' 'round_off' outputs rounded values of the dataset to the fourth decimal place and function 'calculations' returns the effect of column values of 'open' on 'close'.

Spark context is built by the name 'Ibrahim' which helps it to recognize the session without needing to initialize it every time. To call the session, the context name 'Ibrahim;' can be used to reference the spark session. Furthermore spark, by default reads the data from a csv file and stores it as 'string'. It doesn't retain the original data types of the dataset. Hence it is imperative to make a schema defining all the data types of the dataset under consideration. As in this case, the dataset of Zilliqa-USD currency pair, all values are large and numeric. Hence, they are given 'DoubleType' as the data type.

The only exception being the column ‘date’ which is given a string datatype. The following code snippet shows the making of the spark session and the required schema:

```
# Creating spark session
spark = SparkSession.builder.appName('Ibrahim').getOrCreate()

# Creating schema for the data set
schema = StructType() \
      .add("time",StringType(),True) \
      .add("open",DoubleType(),True) \
      .add("close",DoubleType(),True) \
      .add("high",DoubleType(),True) \
      .add("low",DoubleType(),True) \
      .add("volume",DoubleType(),True)
```

The next step is to register the UDFs to use them in the code later. Furthermore, data is being read from the csv file with the schema defined in the above snippet and saved into variable ‘df’

```
 # Registering UDFs
 round_off_udf = udf(round_off, DoubleType())
 calculations_udf = udf(calculations, DoubleType())

 # Reading data from the csv
 df = spark.read.format("csv").option("header", True).schema(schema).load("zilusd.csv")
```

After the data is stored into ‘df’ variable, its time to analyze and perform functions on the dataset. The first task is to roundoff the values of the dataset to the fourth decimal point as shown in the below snippet:

```
# Rounding off values of the columns
df = df.withColumn("open",round_off_udf(col("open")))
df = df.withColumn("close",round_off_udf(col("close")))
df = df.withColumn("high",round_off_udf(col("high")))
df = df.withColumn("low",round_off_udf(col("low")))
df = df.withColumn("volume",round_off_udf(col("volume")))
df.show()
```

The output of the above code is shown on the command window as below:

```
+-------------+------+------+------+------+----------+
|         time|  open| close|  high|   low|    volume|
+-------------+------+------+------+------+----------+
|1533792660000| 0.041| 0.041| 0.041| 0.041|    2000.0|
|1533802620000|  0.04|0.0391|  0.04|0.0391|   2547.45|
|1533808380000|0.0383|0.0383|0.0383|0.0383|  549.1154|
|1533810660000|0.0419| 0.042| 0.042|0.0419| 6600.3587|
|1533810720000| 0.042| 0.042| 0.042| 0.042|11221.6699|
|1533811980000| 0.045| 0.045| 0.045| 0.045|     700.0|
|1533812040000| 0.045| 0.045| 0.045| 0.045|  411.3133|
|1533821100000|0.0449|0.0449|0.0449|0.0449| 1198.4279|
|1533821820000| 0.044| 0.045| 0.045| 0.044| 1042.0294|
|1533822000000|0.0383| 0.036|0.0383| 0.036| 3938.8347|
|1533833100000| 0.045| 0.045| 0.045| 0.045|   393.872|
|1533837900000| 0.045| 0.045| 0.045| 0.045| 3118.5268|
|1533867840000|0.0412|0.0412|0.0412|0.0412|   917.084|
|1533874800000|  0.04|  0.04|  0.04|  0.04|  405.0838|
|1533902460000|  0.04|  0.04|  0.04|  0.04| 1536.1003|
|1533917160000|0.0409|0.0409|0.0409|0.0409|     500.0|
|1533920160000|  0.04|  0.04|  0.04|  0.04|   689.429|
|1533925320000|  0.04| 0.039|  0.04| 0.039|     248.0|
|1534010940000| 0.042| 0.042| 0.042| 0.042|   904.351|
|1534014960000| 0.042| 0.042| 0.042| 0.042|     290.0|
+-------------+------+------+------+------+----------+
only showing top 20 rows
```

The table shows the values of all columns of the dataset rounded off to the fourth decimal place.

The next step is to do statistical analysis (min, max and average) of 'volume' column of the dataset. The task is achieved by the following lines of codes:

```
# Calculating the min max and average of the volume
maximum = df.agg({"volume": "max"}).collect()[0]["max(volume)"]
minimum = df.agg({"volume": "min"}).collect()[0]["min(volume)"]
average = df.agg({"volume": "avg"}).collect()[0]["avg(volume)"]
print("Average of volume is:", maximum)
print("Minimum of volume is:", minimum)
print("Maximum of volume is:", average)
```

The output of the above snippet is shown below:

```
Average of volume is: 1481067.6802
Minimum of volume is: 0.0
Maximum of volume is: 13025.423697322592
```

The output shows the minimum, maximum and average from the volume column of the dataset.

Duplicate rows must be deleted from the dataset for correct analysis. The following screenshot shows the code used to remove any duplicate rows from the dataset.

```
# Dropping duplicate rows (if any) from the data set
df = df.dropDuplicates()
df.show()
```

The resultant dataset after dropping duplicate rows is shown below:

```
+-------------+------+------+------+------+----------+
|         time|  open| close|  high|   low|    volume|
+-------------+------+------+------+------+----------+
|1534287600000|0.0262|0.0262|0.0262|0.0262|10598.6709|
|1536798240000|0.0325|0.0329|0.0329|0.0325| 8590.5584|
|1538982600000|0.0363|0.0363|0.0363|0.0363|   4083.58|
|1539312060000|0.0321|0.0321|0.0321|0.0321| 1714.6599|
|1541147220000|0.0362|0.0362|0.0362|0.0362| 7391.7204|
|1541590500000|0.0356|0.0356|0.0356|0.0356|   1750.79|
|1542741900000|0.0186|0.0186|0.0186|0.0186|  644.2219|
|1543094400000| 0.016| 0.016| 0.016| 0.016|17005.1932|
|1543248240000|0.0136|0.0136|0.0136|0.0136| 2227.6932|
|1545835500000|0.0181|0.0181|0.0181|0.0181|  2072.572|
|1546090920000|0.0213|0.0213|0.0213|0.0213|    1000.0|
|1547040960000|0.0247|0.0247|0.0247|0.0247|  16170.33|
|1547139900000|0.0207|0.0207|0.0207|0.0207| 6156.1905|
|1547248080000|0.0199|0.0199|0.0199|0.0199| 30012.012|
|1547638380000| 0.023| 0.023| 0.023| 0.023|       0.0|
|1550728860000|0.0195|0.0195|0.0195|0.0195|    1000.0|
|1554389040000|0.0227| 0.023|0.0231|0.0227|  26948.49|
|1554782280000| 0.023|0.0229| 0.023|0.0229|   22572.0|
|1557252300000|0.0165|0.0165|0.0165|0.0165|  938.2757|
|1557732780000|0.0165|0.0165|0.0165|0.0165|    5000.0|
+-------------+------+------+------+------+----------+
only showing top 20 rows
```

Moving along the analysis, the next step is to calculate the ‘open’ column as a rate of ‘close’ column values. It is achieved by the following code snippet. A new column named ‘calculations’ is made automatically to store the result of each calculation.

```
# Creating new column for (open x close) / close
df = df.withColumn("Calculations",calculations_udf('open', 'close'))
df.show()
```

The output dataset is shown below:

```
+-------------+------+------+------+------+----------+------------------+
|         time|  open| close|  high|   low|    volume|      Calculations|
+-------------+------+------+------+------+----------+------------------+
|1534287600000|0.0262|0.0262|0.0262|0.0262|10598.6709|               2.0|
|1536798240000|0.0325|0.0329|0.0329|0.0325| 8590.5584| 1.987841945288754|
|1538982600000|0.0363|0.0363|0.0363|0.0363|   4083.58|               2.0|
|1539312060000|0.0321|0.0321|0.0321|0.0321| 1714.6599|               2.0|
|1541147220000|0.0362|0.0362|0.0362|0.0362| 7391.7204|               2.0|
|1541590500000|0.0356|0.0356|0.0356|0.0356|   1750.79|               2.0|
|1542741900000|0.0186|0.0186|0.0186|0.0186|  644.2219|               2.0|
|1543094400000| 0.016| 0.016| 0.016| 0.016|17005.1932|               2.0|
|1543248240000|0.0136|0.0136|0.0136|0.0136| 2227.6932|               2.0|
|1545835500000|0.0181|0.0181|0.0181|0.0181|  2072.572|               2.0|
|1546090920000|0.0213|0.0213|0.0213|0.0213|    1000.0|               2.0|
|1547040960000|0.0247|0.0247|0.0247|0.0247|  16170.33|               2.0|
|1547139900000|0.0207|0.0207|0.0207|0.0207| 6156.1905|               2.0|
|1547248080000|0.0199|0.0199|0.0199|0.0199| 30012.012|               2.0|
|1547638380000| 0.023| 0.023| 0.023| 0.023|       0.0|               2.0|
|1550728860000|0.0195|0.0195|0.0195|0.0195|    1000.0|               2.0|
|1554389040000|0.0227| 0.023|0.0231|0.0227|  26948.49|1.9869565217391307|
|1554782280000| 0.023|0.0229| 0.023|0.0229|   22572.0| 2.004366812227074|
|1557252300000|0.0165|0.0165|0.0165|0.0165|  938.2757|               2.0|
|1557732780000|0.0165|0.0165|0.0165|0.0165|    5000.0|               2.0|
+-------------+------+------+------+------+----------+------------------+
only showing top 20 rows
```

As its seen by the output, a new column named ‘Calculations’ is added into the dataset which houses all the output received by executing the above lines of code.

An additional analysis step is done which sorts the values of the dataset based on values of the column ‘volume’. The result is achieved by the following lines of code:

```
# Sorting the data based on Volume
df = df.orderBy('volume')
df.show()
```

The sorted resultant dataset is shown below:

```
+-------------+------+------+------+------+------+------------+
|         time|  open| close|  high|   low|volume|Calculations|
+-------------+------+------+------+------+------+------------+
|1553608440000|0.0186|0.0186|0.0186|0.0186|   0.0|         2.0|
|1596542160000| 0.018| 0.018| 0.018| 0.018|   0.0|         2.0|
|1543680000000|0.0186|0.0186|0.0186|0.0186|   0.0|         2.0|
|1558630440000| 0.023| 0.023| 0.023| 0.023|   0.0|         2.0|
|1548706920000|0.0205|0.0205|0.0205|0.0205|   0.0|         2.0|
|1598711040000| 0.021| 0.021| 0.021| 0.021|   0.0|         2.0|
|1550459220000|0.0184|0.0184|0.0184|0.0184|   0.0|         2.0|
|1561668300000|0.0164|0.0164|0.0164|0.0164|   0.0|         2.0|
|1606478880000|0.0239|0.0239|0.0239|0.0239|   0.0|         2.0|
|1594063920000|0.0186|0.0186|0.0186|0.0186|   0.0|         2.0|
|1592155020000|0.0233|0.0233|0.0233|0.0233|   0.0|         2.0|
|1593150060000|0.0184|0.0184|0.0184|0.0184|   0.0|         2.0|
|1606610220000| 0.027| 0.027| 0.027| 0.027|   0.0|         2.0|
|1609061100000| 0.087| 0.087| 0.087| 0.087|   0.0|         2.0|
|1543345980000|0.0175|0.0175|0.0175|0.0175|   0.0|         2.0|
|1547638380000| 0.023| 0.023| 0.023| 0.023|   0.0|         2.0|
|1593076980000|0.0195|0.0195|0.0195|0.0195|   0.0|         2.0|
|1594060260000|0.0187|0.0187|0.0187|0.0187|   0.0|         2.0|
|1597025220000|0.0237|0.0237|0.0237|0.0237|   0.0|         2.0|
|1543582380000|0.0165|0.0165|0.0165|0.0165|   0.0|         2.0|
+-------------+------+------+------+------+------+------------+
only showing top 20 rows
```

As it is seen from the above table, the all the values of the dataset are sorted based on column ‘volume’.

The final step of the analysis is to divide the dataset into two parts. Each containing the values either less than the average value of column ‘volume’ or grater than the average value of column ‘volume’. By executing the below snippet of code, we get two datasets. One having all the values less than the average value of the column ‘volume’ and the other one having all the values greater than or equal to the average value of the column ‘volume.

```
df_lt_average = df.filter(df.volume < average)
df_gte_average = df.filter(df.volume >= average)
df_lt_average.write.csv('volume_lt_avg')
df_gte_average.write.csv('volume_gte_avg')
```

The results of the above division are stored in separate csv files. The output of Spark dictates the csv files to be stored in separate folders automatically. Spark creates two sperate folders to save two csv files. The names of the folders are same as the name of the files. Spark creates folders at the output since it works on threads. Each thread carrying some part of the workload. Whenever a command is executed, the load is divided into different threads. This division is arbitrary and sometimes unequal. This gives spark its popular speed. Whenever a thread finishes its part of the work; the result is dropped into a file saved in a folder. As each thread finishes off its work, it drops the output into a file which is saved into a folder. This makes the threads available for further analysis and increases up the speed of the execution. This spark specialty makes it the popular choice for the analysts who look for speed and efficiency.

Note: To execute the pyscript use the command python3 PySpark.py or spark-submit PySpark.py depending upon your system configuration. Furthermore, along with project report, the spark script and dataset are also included for reproducibility.