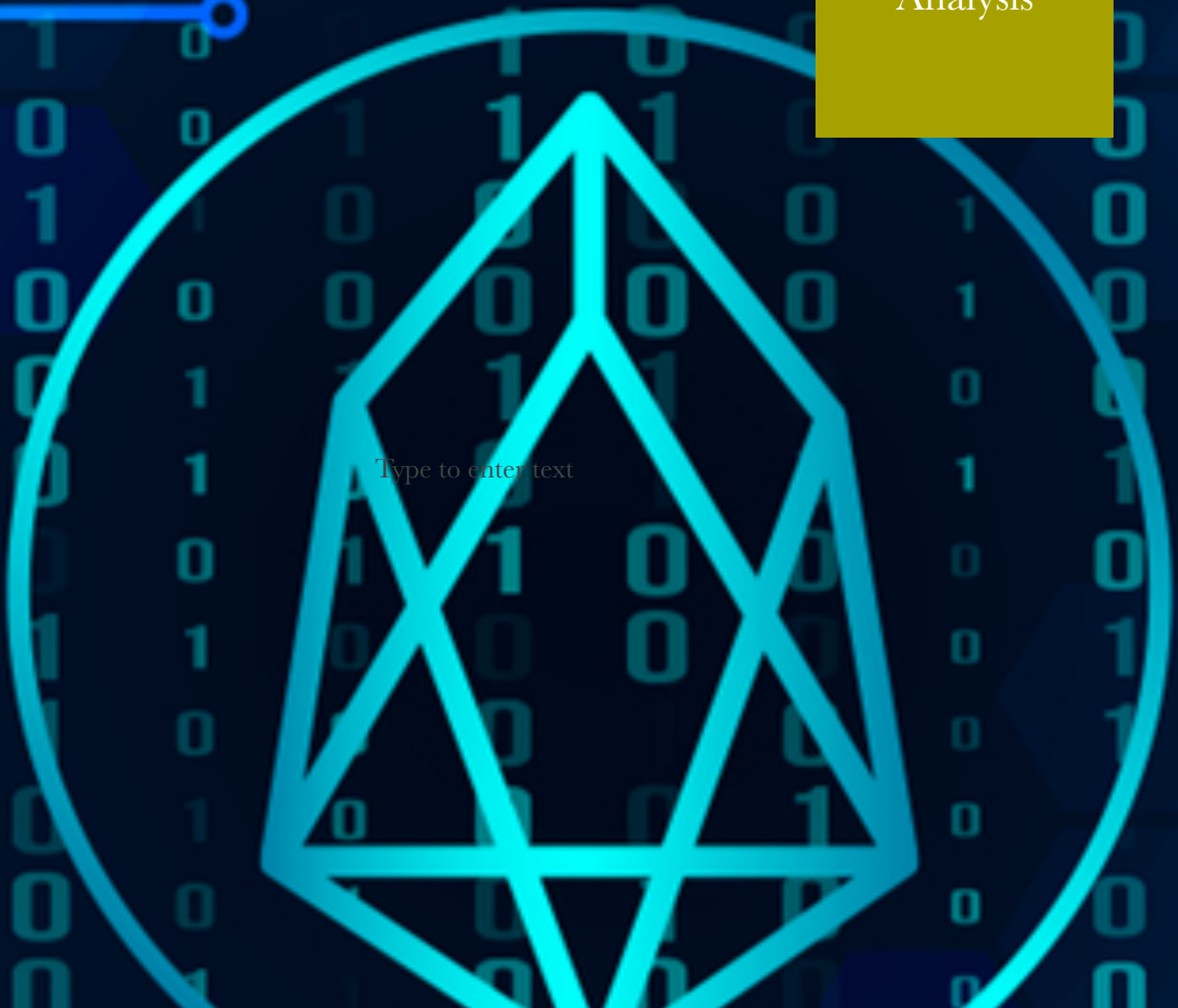
$EOS Market Movement Prediction based on Sentiment Analysis



Type to enter text

Hadoop

Prof. Raul Marin Perez

Group Assignment

EOS Prediction with HiveQL

Group F

Sandra Alemayehu

Frederico Andrade

Luigi Giuglio Grande

Usama Khan

Joe Mehanna

Marcela Zablah

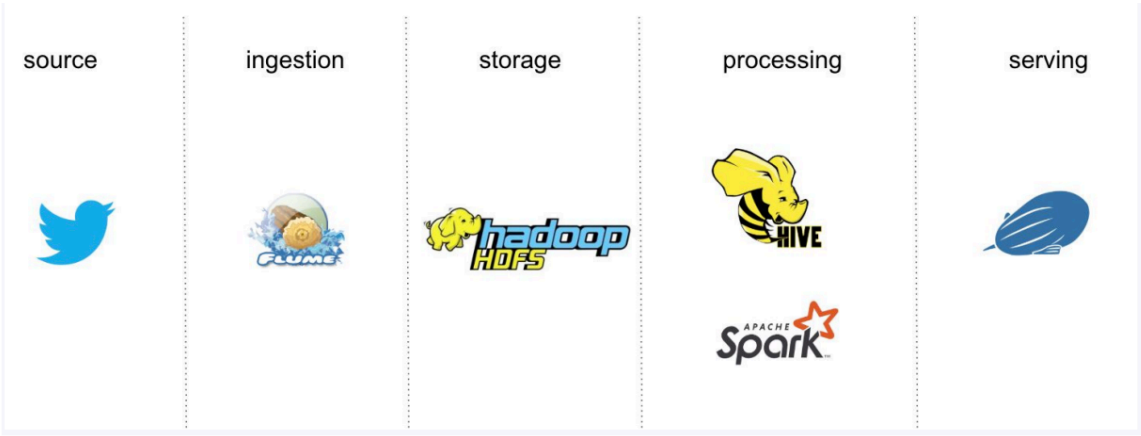
Type to enter text

Group F Assignment

The Problem at hands

In this assignment, we were handed a hypothetical scenario, where we joined an investment startup working on a project, with the objective of predicting a specific cryptocurrency's market movements, based on social media (Twitter) sentiment analysis. In our case, we were assigned &EOS market analysis.

Before deciding the final data architecture, the data engineering team has designed the following Proof of Concept (POC) data architecture to start analysing twitter data:



Given this information, and also the two available files:

1. sentiment-dictionary dataset (tab delimited file containing english words (lower case) with their sentiment polarity);
2. crypto-tweets (in which the engineering team was currently ingesting cryptocurrency related tweets).

Our final task is to analyse both the datasets and draw conclusions about the cryptocurrency's market movement regarding its Twitter mentions.

For that, a series of queries were performed, and the outputs registered for analysis (according to the instructions given for this assignment):

Queries:

1. create a database named 2019o2_team_f:

create database 2019o2_team_f;

2. select the database you just created so that all the tables you are going to create belong to that database.

use 2019o2_team_f;

3. create an external table named sentiment_dictionary with the files provided.

create external table sentiment_dictionary(type string, length int, word string, word_type string, stemmed string, polarity string) row format delimited fields terminated by '\t' location '/user/usamakhan/hive/dictionary';

load data local inpath '/data/home/usamakhan/dictionary.tsv' into table sentiment_dictionary;

4. create an external table named tweets_json with the files provided.

*create external table tweets_json(text string, source string, in_reply_to_status_id bigint, in_reply_to_status_id_str string, in_reply_to_user_id bigint, in_reply_to_user_id_str string, `user` struct <id:bigint,id_str:string,name:string,screen_name:string,location:string,url:string,description:string,protected:boolean,verified:boolean,followers_count:int,friends_count:int,listed_count:int,favourites_count:int,statuses_count:int,utc_offset:int,time_zone:string,geo_enabled:Boolean,lang:string> , entities struct<hashtags:array<struct<text:string>>>,coordinates struct <coordinates:array<float>> ,place struct <**country:**string,country_code:string,code:string,full_name:string,place_type:string,url:string>, reply_count int, retweet_count int,retweeted Boolean, lang string) row format serde 'org.apache.hive.hcatalog.data.JsonSerDe'stored as textfile location '/user/usamakhan/hive/tweets_json';*

load data local inpath '/data/home/usamakhan/tweets1.json' into table tweets_json';

5. write a query that returns the total number of tweets in table tweets_json. Annotate both the number of records and the amount of seconds that it took.

select count (*) from tweets_orc;

Output: 9639 rows fetched in .03 secs

6. create a managed table tweets_orc with same schema as tweets_json but stored in orc format.

create table tweets_orc like tweets_json stored as orc;

7. insert all rows from tweets_json into tweets_orc.

*insert into tweets_orc select * from tweets_json;*

8. write a query that returns the total number of tweets in table tweets_orc. Annotate both the number of records and the amount of seconds that it took.

select count (*) from tweets_orc;

Output: 9639 rows in .029 secs

9. verify that both tables contain the same number of tweets. which of the queries was faster?

select count (*) from tweets_orc;

Output: count 9639 in .079 secs (being this the faster).

select count (*) from tweets_json;

Output: count 9639 in 22.419 secs

10.write a query that returns the total number of users with geolocation enabled from table tweets_orc.

select count (*) from tweets_orc where `user`. geo_enabled = true;

Output 1828 count in 21.439secs

11.write a query that returns the total number of tweets per language from table tweets_orc.

select lang,count(*) from tweets_orc group by lang;

12.write a query that returns the top 10 users with more tweets published from table tweets_orc.

select `user`.name, max(`user`.followers_count) as count from tweets_orc group by `user`.name order by count desc limit 10;

13.write a query that returns the geoname latitude, longitude and timezone of the tweet place by joining geonames and tweets_orc.

create external table geonames(id bigint,name string,ascii_name string,alternate_names array<string>,latitude float,longitude float,feature_class string,feature_code string,country_code string,country_code2 array<string>,admin1_code string,admin2_code string,admin3_code string,admin4_code string,population bigint,elevation int,dem int,timezone string,modification_date date) row format delimited fields terminated by '\t' collection items terminated by ',' stored as textfile location '/user/usamakhan/hive/geonames';

select geonames.latitude, geonames.longitude, geonames.timezone from geonames inner join tweets_orc on geonames.name=tweets_orc.place.full_name;

14.write a query that returns the total count, total distinct count, maximum, minimum, average, standard deviation and percentiles 25th, 50th, 75th of hashtags in tweets from table tweets_orc

N/A

15.write a query that returns the top 10 more popular hashtags from table tweets_orc

select `user`.name, max(entities.hashtags) as count from tweets_orc group by `user`.name order by count desc limit 10;

16.create a table tweet_words in parquet format exploding the words in the tweets. Also normalize the words to lowercase.

create table tweet_words as select `user`.id as id, upper(text) from tweets_orc LATERAL VIEW explode (split(text,',')) store as parquet;