# Prediction of Bitcoin Prices based on Tweets

Alejandro Muñoz
Juan Escalona
Rodrigo Juárez
Salvador García
Victor Rivera

## Problem definition

### Motivation & Business Problem

The business problem we want to solve is to reduce the uncertainty of forecasting the price of Bitcoin for traders or people who invest their money in this kind of cryptocurrency.

Bitcoin price is determined, like any other commodity, by its supply and demand. Bitcoin's supply is limited, there will only ever be 21 million produced and the actual circulating supply is 18,925,000. Its protocol allows new bitcoins to be created at a fixed rate. For these reasons, we may consider that the Bitcoin supply is fixed, so the price will depend heavily on the demand side. In other words, the price of Bitcoin depends on consumer desire for cryptocurrency. Our approximation to measure the demand for Bitcoin will be posted on Twitter with the hashtag Bitcoin.

Financial and economic news is continuously monitored by financial market participants. According to the efficient market hypothesis, all past information is reflected in stock prices, and new information is instantaneously absorbed in determining future stock prices. Hence, prompt extraction of positive or negative sentiments from the news is very important for investment decision-making by traders and investors. Sentiment analysis models can provide an efficient method for extracting actionable signals from the news.

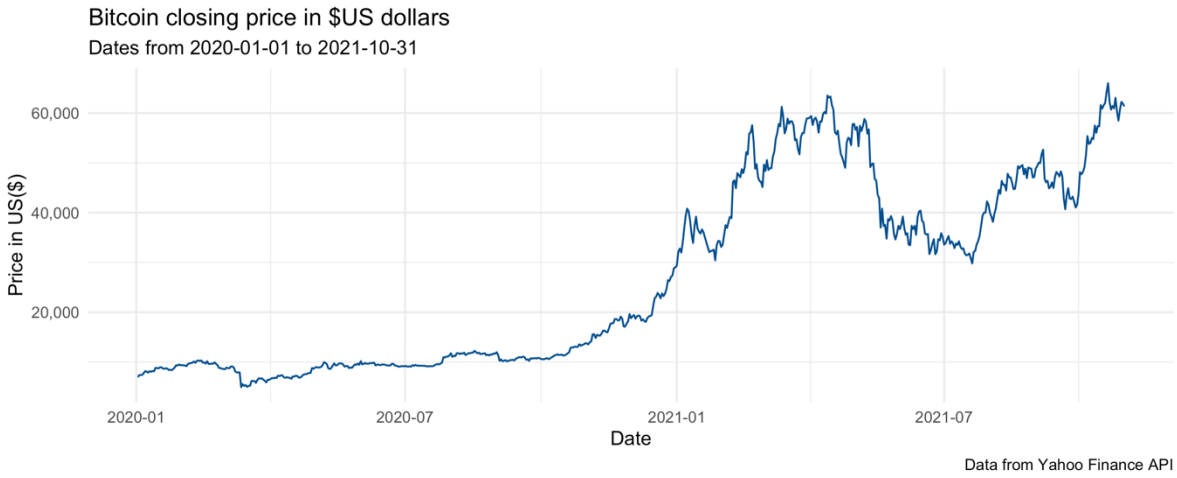


*Figure 1. Bitcoin closing price in $USD.*

## Machine Learning Problem

With the development of algorithmic trading systems, human-decision making has reached its limit and cannot keep up with the processing and execution of orders and decisions. Additionally, information overload is not manageable without support. To manage all information it is necessary to use a machine learning algorithm to process all the data and predict if the Bitcoin price goes up or down the next day. By doing this, we reduce the complexity of prediction to a binary problem (up or down). For the forecasting, we will ingest only posts from Twitter as demand for Bitcoin proxy:

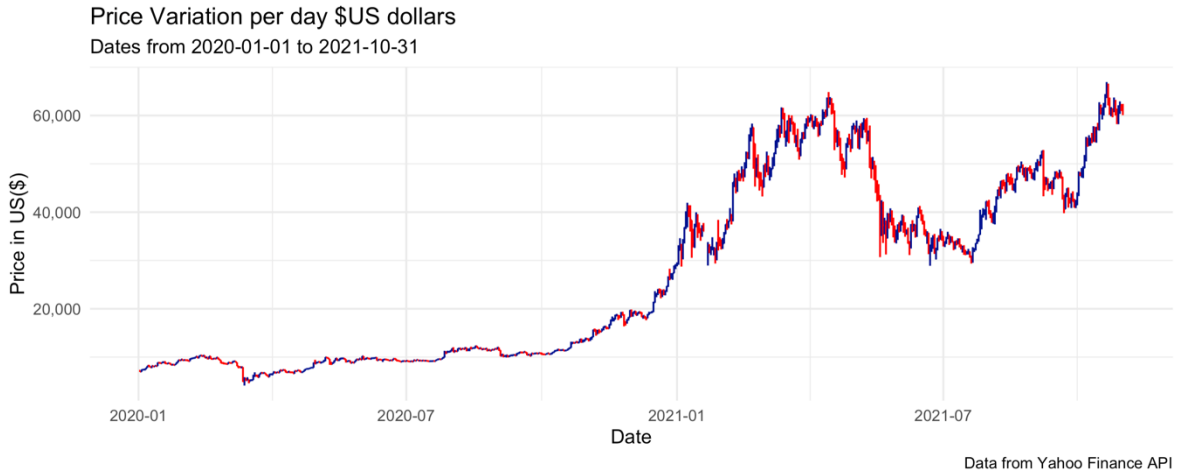


*Figure 2. Price variation per day in $USD.*

## System design

The data sources of the project include Twitter data and Yahoo Finance data (explained in the next section). To consider an adequate data architecture, it is required to analyze the data volume and the frequency of training and scoring.

For the **storage account**, as the data volume is not large, it is not required a complex solution. The data estimates are 300k tweets per year, so it is easy to manage data volume. Each tweet uses between 0.5 KB and 0.8 KB in a JSON format, so for the 300k tweets, only 150-240 MB will be used. The Bitcoin price includes even fewer registers, which is irrelevant in terms of storage. With a general purpose VM n1-standard-1 is enough to retrieve the data. This VM contains 1vCPU, 3.75 GB in RAM.

For the **training pipeline**, different models were attempted, but for the final, an ensemble model was selected, in particular a random forest that is stored in the storage account with a pickle format. This way, the model only uses 25 MB of storage. Again, with the volume data and the selected model, a n1-standard-1 is adequate.

For **prediction/scoring pipeline**, in the best scenario, the prediction should be done in shorter time windows to provide relevant information for the business case. Doing a real-time prediction or at least a near real-time prediction goes further toward the objective of the project. To simplify the problem, the predictions are performed just once a day. For this reason, an n1-standard-1 works for the task.
The next diagram shows the flow of both pipelines, which are explained in detail in the next subsection:

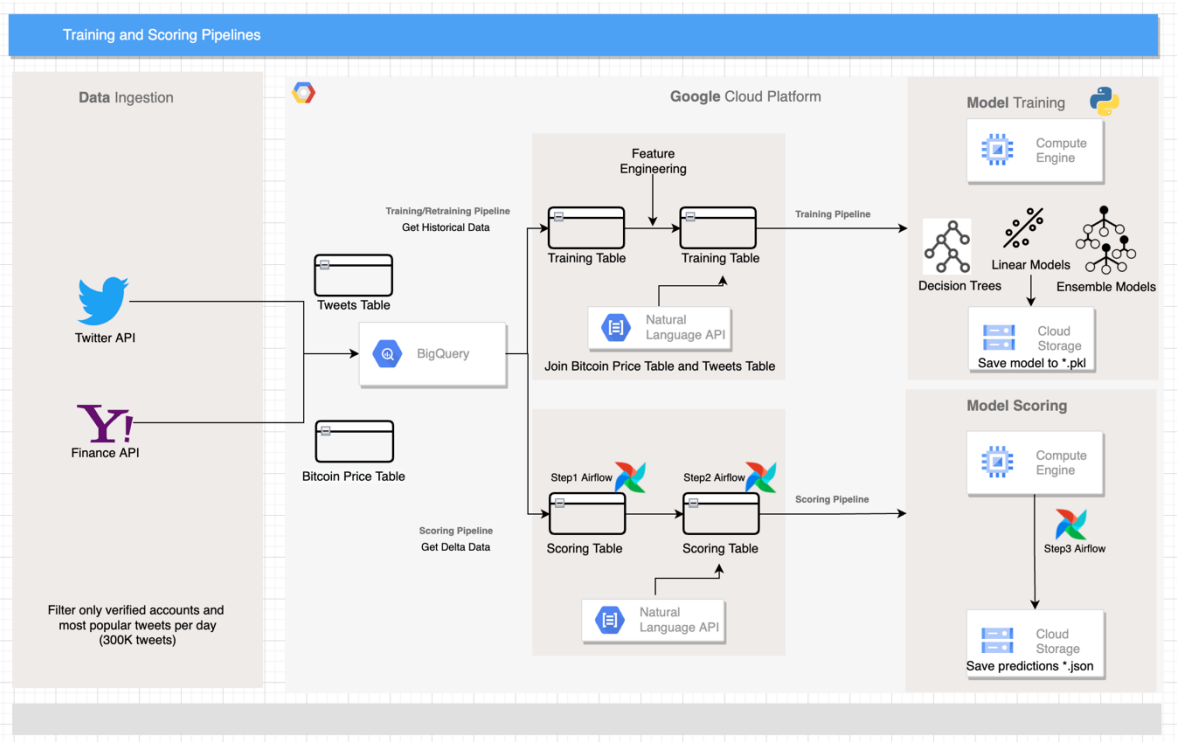

*Figure 3. Data Product Architecture.*

## VM and airflow configuration

To be able to run airflow in the Virtual Machine, the tutorial in Medium was used as reference. This includes the service account configuration, the compute engine configuration, and the airflow installation and execution. Finally, the setup script provided in class is provided as the startup script to automatically execute airflow when the instance starts. The stored environment variables in Airflow are: **bearer_token** (to get tweets), **bucket** and **bucketauth**.

## Big query

Google BigQuery is used as the central place where the structured data is stored and consumed. BigQuery allows to run interactive queries and also create virtual tables. It is selected because it is easy to set up, to use and it is stated that the storage fee is inexpensive. It is created one project dpa-2022-itam and here one dataset that stores the tables for bitcoin and tweets (historical and today).

## Storage Account

In the storage account, a variety of scripts, data, and pickles are stored. For example here is stored the key-file.json and also the trained model in pickle format, the start-up script, etc. Particularly it is used in two buckets, one for scripts and the other for Twitter data, bitcoin data, airflow dags code, and pickle models. Each one of these is stored in folders.

# Data source, ingestion and feature engineering

## Data Sources

The process of data ingestion considers two data sources: Yahoo! Finance API and Twitter API (https://developer.twitter.com/en/docs/twitter-api). From Yahoo! Finance we want to extract the Bitcoin price per day: closing price, opening price. From Twitter we are interested in tweets related to Bitcoin, so we extracted tweets with Bitcoin hashtag.

*Yahoo! Finance API*

The Yahoo Finance API has several endpoints that can be accessed with a HTTPS request and a GET method. The endpoint /v8/finance/spark retrieves the stock history and receives three parameters: interval, range, and symbol. As well it is required

to provide an API key in the header x-api-key. This can be obtained just by signing up on the webpage.

URL example for an API request:

```
https://yfapi.net/v8/finance/spark?interval=1d&range=1d&symbols=BTC-USD
```

The next code represents the response for 1 day interval y 1 day range for BTC-USD is shown in the next line:


```
{
   "BTC-USD": {
      "symbol": "BTC-USD",
      "timestamp": [
         1651950840
      ],
      "chartPreviousClose": 36033.406,
      "close": [
         35919.76
      ],
      "dataGranularity": 300,
      "previousClose": null,
      "end": null,
      "start": null
   }
}
```


There is an interface using a python package yfinance that offers a pythonic way to download market data from the API, so it was preferred for the project

```
import yfinance as yf
import pandas as pd
from datetime import datetime

start = datetime(2013,1,1)
end = datetime.now().date().isoformat()
symbol = 'BTC-USD'
df = yf.download(symbol, start=start, end = end)
```

*Twitter API*

For the Twitter API v2 it was required to apply for a developer account (Within 2 days the process can be completed). It is possible to use the endpoint via HTTPS with a GET method and using a bearer token authentication and:
URL example for an API request:

```
https://api.twitter.com/2/users/2244994945/tweets?tweet.fields=created_at&max_results=100&start_time=2019-01-01T17:00:00Z&end_time=2020-12-12T01:00:00Z
```

The next code represents the response for 1 tweet between 2019-01-01 and 2020-12-12

```
{
   "data": [
      {
         "created_at": "2020-12-11T20:44:52.000Z",
         "id": "1337498609819021312",
          "text": "Thanks to everyone who tuned in today to make music with the #TwitterAPI!\n\nNext week on Twitch - @iamdaniele and @jessicagarson will show you how to integrate the #TwitterAPI and Google Sheets 📈. Tuesday, Dec 15th at 2pm ET. \n\nhttps://t.co/SQziic6eyp"
      }
   ],
   "meta": {
      "next_token": "7140dibdnow9c7btw3w29l11fnof48sgm1tiwl3uj0u9h",
      "result_count": 1,
      "newest_id": "1337498609819021312",
      "oldest_id": "1336463248510623745"
   }
}
```


For the project, the Pagination endpoints were used. According to the documentation, it retrieves all relevant data related to the query.

Like Yahoo! Finance API, there is a python package that allows requesting in a pythonic way: tweepy. A client must be instantiated, then the bearer token is provided and finally, the paginator is used. For the project, only tweets with the "Bitcoin" keyword are queried. Additional parameters that are sent in the request are -is:retweet that avoids retweets, lang:en that retrieves only tweets in english, and is:verified that retrieves only tweets from verified accounts. The next code is an example of tweets yesterday and today:

```
import tweepy
import yaml
import pandas as pd
import json
from datetime import datetime, timedelta
import time

client = tweepy.Client(<bearer_token>, wait_on_rate_limit=True)
today = str(datetime.today().strftime('%Y-%m-%d')) + 'T00:00:00Z'
d = datetime.today()- timedelta(days=1)
yesterday = str(d.strftime('%Y-%m-%d')) +'T00:00:00Z'
tweets_day=[]
for response in tweepy.Paginator(
    client.search_all_tweets,
    query = 'Bitcoin -is:retweet lang:en is:verified',
    user_fields = ['username', 'public_metrics', 'description', 'location'],
    tweet_fields = ['created_at', 'geo', 'public_metrics', 'text'],
    expansions = 'author_id',
    start_time = yesterday,
    end_time = today,
    max_results=10):
```

```
time.sleep(1)
tweets_day.append(response)
```

The stored data includes the following variables:

- *User level:* number of followers, number of tweets, description, and location
- *Tweet level:* tweet id, author id, username, author followers, author tweets, author location, the text of the tweet, creation time, number of retweets, number of replies, and number of likes.

Only this information is retrieved because JSON structure from Twitter is complex, for every call the structure can be different, so the parsing task is difficult. For this reason, and because of the purpose of the project only these features are extracted from the tweet structure.

## Feature Engineering

Once the data is stored, some feature engineering is performed:

- For Bitcoin prices, the open prices are compared, and a binary variable is created to indicate if there is an increase or decrease in the Bitcoin price compared to a previous day. This variable is the dependent variable of the model.

- For the tweets, sentiment analysis is computed via the Cloud Natural Language API from GCP. The API returns both the score and the magnitude. According to the API documentation, the Score represents the overall emotion of the text, which ranges from -1 to 1. Negative numbers indicate negative sentiments and positive numbers of positive sentiments. The magnitude represents the overall strength of the emotion and ranges from 0 to inf. Something that is stated in the documentation is that the magnitude is not normalized. Longer texts have greater magnitude. Examples of these coefficients can be used according to this table:

| Sentiment | Sample Score | Sample Magnitude |
|---|---|---|
| Clearly Positive | 0.8 | 3.0 |
| Clearly Negative | -0.6 | 4.0 |
| Neutral | 0.1 | 0.0 |

| Sentiment | Sample Score | Sample Magnitude |
|---|---|---|
| Mixed | 0.0 | 4.0 |

For the model, this does not have a great impact because each tweet is limited to 280 characters (approx. 56 words).

These are the main covariates that are used in the model. The hypothesis is that negative sentiment toward the cryptocurrency implies a decrease in the Bitcoin Price from one day to another.

## Model Development

### Historical and delta data retrieval

As explained in the previous section, all the Twitter data and Yahoo Finance were retrieved from the Twitter API and the Yahoo Finance API. To train an ML model, **Historical data** for tweets and bitcoin are required, to score based on the trained model, **Delta data** is required.

For **Historical data**, tweets from one year are stored, as well for the bitcoin USD price. This was done with batch processes that are performed both for bitcoin prices and for tweets information. It is performed once and saved in a storage account. This information is used to train the model for the first time. Although there were some restrictions, for the project it was possible to retrieve all the required data. For all the historical data, data munging and feature engineering were performed with Python and BigQuery. Important features were included for all the datasets: the Sentiment Score for each tweet was added and then an average per day was calculated. Finally, the training table is made with the join by day of the sentiment of the tweets per day and the Daily Bitcoin prices.

For **Delta data** is used to predict daily, only 10 most relevant tweets are retrieved each day when the scheduler is performed. Bitcoin price deltas are not required for the model, but only to make the model predictions.

### Model Comparison and Selection

Multiple ML models were performed to compare the prediction score metrics. Among them are ensemble models, linear models, and decision tree models. Finally, the approach of Random Forest was the model with the best performance. We consider two intuitive benchmarks to be surpassed. The first is to always output a price increase, which gives us a precision of 54%, and a linear model (logistic regression) that gives us a precision of 49%. The linear model was attempted, although in the EDA it seems that the problem is not linear, just to get a benchmark.

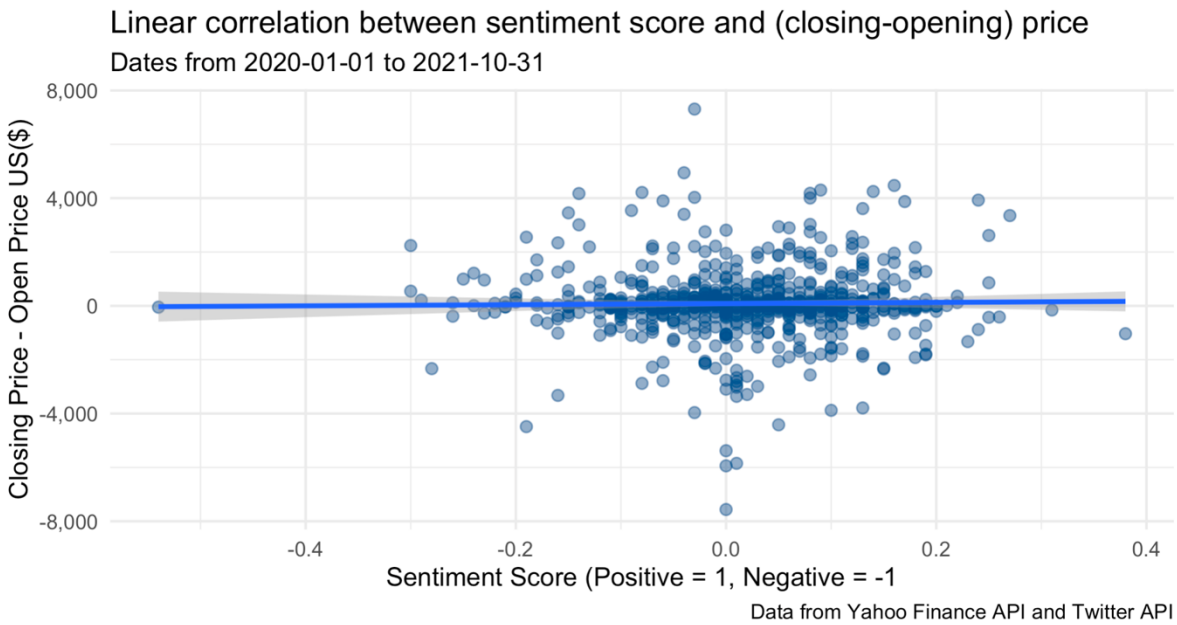


*Figure 4. Linear correlation between sentiment score and (closing-opening) price.*

For the training phase, a train-test split was performed. The test size was 20% of the table and the remaining 80% was used as the training set. The first model to be tested was Logistic Regression with regularization. We tried to do a regression with Lasso and Ridge regularization, however, our model only considered the variables of price and sentiment Analysis, therefore, it did not have many variables to regularize. Therefore, we had results very similar to those of simple regression. We concluded that linear models are not good for this problem. For the third model, Gradient Boosting Classifier, Random Forest Classifier, and Voting Classifiers were attempted:

| **Model** | **Accuracy** |
|---|---|
| Voting Classifier | 0.54 |
| Gradient Boosting Classifier | 0.56 |
| Random Forest Classifier | 0.60 |

From this table, we can conclude that the model that beats the benchmark is the Random Forest Classifier. Although it only achieves an improvement of 11.11% over the benchmark. This gives us an intuition that the model works but requires additional variables or data to get a better score.

## Model Evaluation

An ensemble method is an approach that combines many simple “building block” models to obtain a single and potentially very powerful model. These simple building block models are sometimes known as weak learners since they may lead to mediocre predictions on their own. Random forest is an ensemble method for which the simple building block is a regression or a classification tree.

The main limitation of random forest is that many trees can make the algorithm too slow and ineffective for real-time predictions. In general, these algorithms are fast to train but quite slow to create predictions once they are trained.

Random Forest requires much more time to train as compared to decision trees as it generates a lot of trees (instead of one tree in the case of the decision tree) and makes decisions on most votes.

Precision measures the true positive as a ratio of all the positive results. The business problem we want to solve focuses on the correct prediction of Bitcoin price movements, hence, we consider that precision will be our best option. Another reason to use this metric was to simplify the interpretability of the model for our clients.

## Model Serving

The prediction or scoring is triggered all day at the minute 00:10 of every day (10 0 * * *). In contrast, the model retraining is scheduled at 12:10 on Sunday, Tuesday, and Friday (10 12 * * 0,2,5). An additional function that should be added is the retraining trigger when the model performance decreases (instead of scheduled retraining). Additionally, we assumed two factors:

- There will be no severe data drift that would require a different model instead of RF
- The compute in the VM will be sufficient for the training and scoring (although for the current loads it is sufficient, it is better to have an auto scale if needed)

Some considerations are not included here, mostly for the operationalization of the model:

- No reprocess strategy is implemented (it is assumed that the historical data is valid when retrieved, but for many other uses cases it is not)
- If a manual execution is required, the operation user should know the DAG and know how to run it manually.
- All the infrastructure is manually provisioned and configured. It can be provisioned with Terraform or with some IaaC.
- Generally, the development team does not have access to QA and PRD, for this project it is not taken into consideration any strategy to promote across different environments

## Training/Retraining pipeline (DAG)

For the training pipeline, only the RF model is trained, because of the time and credits limit no additional model stacking is done. In the pipeline that is shown in the figure (number) the model is retrained with more tweets data and bitcoin data. The steps that follow the pipeline are the following:

1. Get historical data from Twitter and the Bitcoin price
2. Storage the tweets and Bitcoin price in Bigquery
3. Perform feature for Bitcoin price (compute the deltas for yesterday and today)
4. Compare the model performance with the previous model performance to determine if it substitutes the previous model
5. Replace or keep the previous model as pkl

For now, the model training is scheduled on Sundays, Tuesdays, and Friday, there is no retraining trigger if the model has a lower performance. The next DAG represents the training pipeline steps:

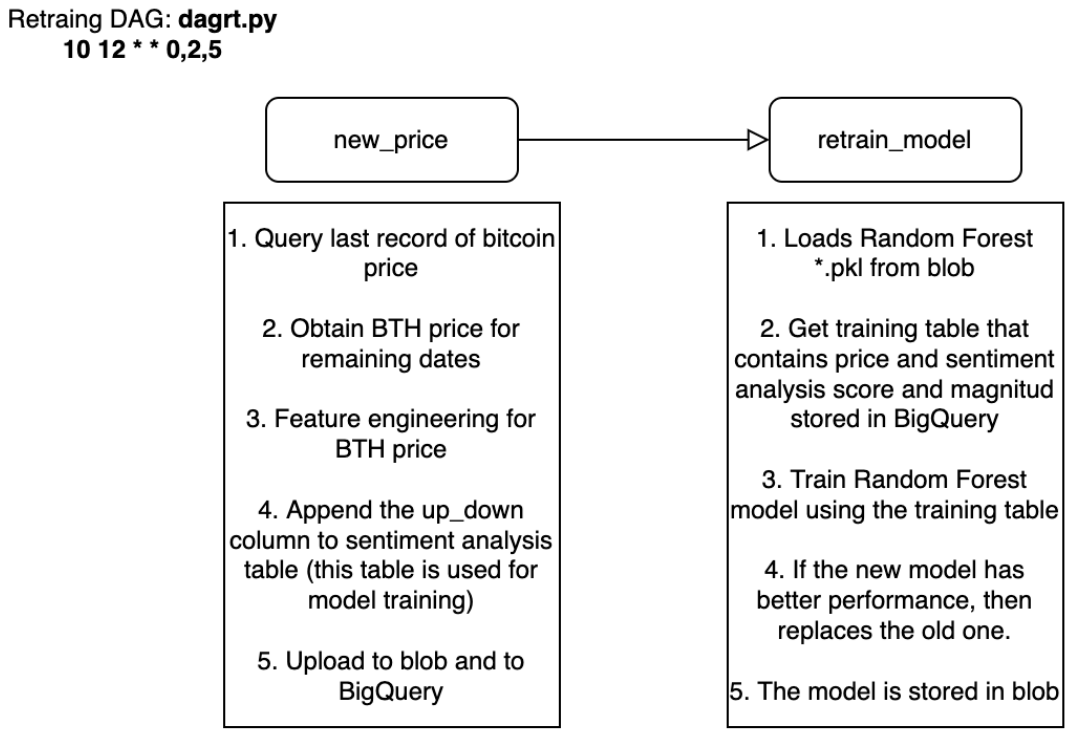


*Figure 5. Retrain DAG..*

## Scoring pipeline (DAG)

In the scoring pipeline, there is no need to download Bitcoin prices, as they are only needed for the training pipeline, so only the twitter data is downloaded and saved into Bigquery. The next DAG represents the scoring pipeline steps:

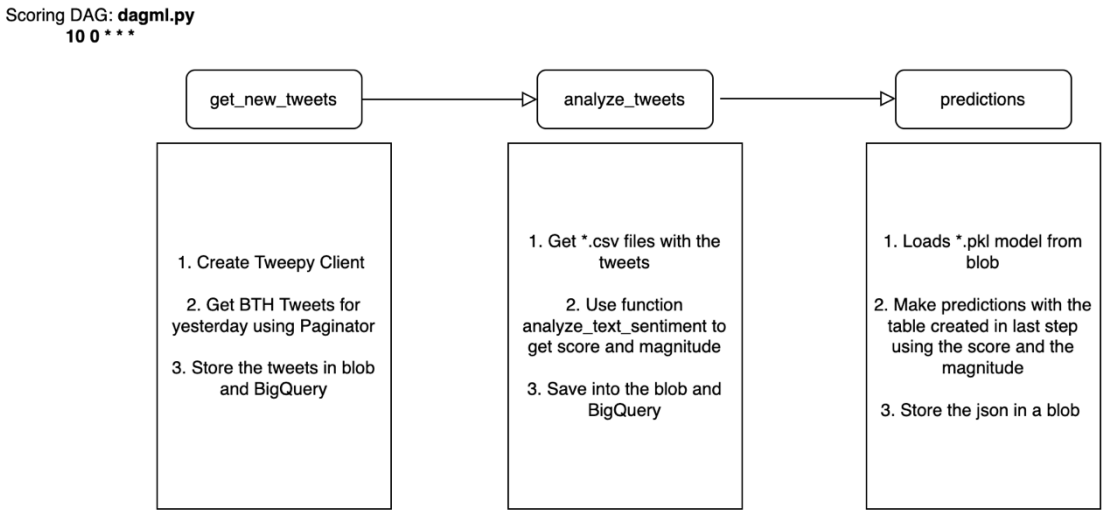


*Figure 6. Scoring DAG*

# Reflection

## Next steps

*Prediction*

- A major improvement to our product is to increase the outputs of our model, to give more information about the price change in the future. For example, differentiate if the price will go up or down by 5% or 10%. One weakness of the current model is that it only considers an increase or decrease, but it would be useful to know the magnitude.
- Now, we predict for the next day, so the time interval is large. It can work to give suggestions at a more frequent time interval.
- Additionally, it is possible to give some confidence interval that helps to measure the uncertainty of the prediction.
- Real-time prediction

*Data*

- We need to consider more feature engineering and include new variables and data. The current model only considers two covariates, but to have a better prediction, it is required to input more significant variables, for example, bond

yield rate, information on stock exchange index, dollar quotation, current exchange order books, and trading volume, etc.
- It would also be important to consider the opinion of experts in the financial market and try to include previous information through a Bayesian model.
- Create an overall cryptocurrency effect in the model. It is well-known that cryptocurrency prices have a high correlation, so we could make a model of the complete market.

*Models*

- Model stacking was considered, but due to time constraints, only the best model is trained each time (this may be a weakness of our solution since we do not consider data shifts and assume that an RF will always be the best solution).
- We were able to consider other deep learning options to train our model like neural network
- The retraining strategy is based on schedule, but it can be based on model performance with a trigger for retraining.
- It can be useful to have model monitoring to quickly detect any anomaly.

*Product*

- Include other cryptocurrencies like Ethereum, Tether, Binance Coin, etc.
- It might even be an excellent idea to predict the price of other assets in the market, such as stocks, shares, etc. This can be used independently or can be used as inputs for the crypto model.
- To have a high-quality product we should include best practices for software development, for example, unit testing, integration testing, UAT, etc.
- Include best practices for DevOps/MLOps. Currently, the solution is based on schedule rather than on best practices for DevOps/MLOps.

## Broader Impact

Our model takes into account whether the price of Bitcoin will go up or down, however, it is not a good model to predict how the price of other cryptocurrencies will be affected as they will not necessarily have the same relationship.

Our model is only informative, it is not designed to make complex strategic decisions in the short-selling market.

Our model makes estimates of future changes in bitcoin's prices, however, it cannot reliably predict what is going to happen in the future.

The cryptocurrency price volatility is high, so we strongly recommend caution against making decisions based solely on this model. It is an indicator, however, more external information is needed to make an informed decision.

## References

Sculley, David, et al. “Hidden technical debt in machine learning systems.” Advances in neural information processing systems 28 (2015).

Roesslein, Joshua. “Tweepy Documentation.” Online] http://tweepy. readthedocs. io/en/v3 5 (2009).

James, Gareth; Witten, Daniela, Hastie, Trevor & Tibshirani, Rob. “An Introduction to Statistical Learning”. Springer, 2021.

Hasan, Ali, et al. “Machine learning-based sentiment analysis for twitter accounts.” Mathematical and Computational Applications 23.1 (2018): 11.

Garita, Mauricio. “Using Stock Market Data in Python.” Applied Quantitative Finance. Palgrave Pivot, Cham, 2021. 71-83.

Densmore, James. Data Pipelines Pocket Reference. " O’Reilly Media, Inc.", 2021.

## Contributions

| Section | Contributors |
|---|---|
| Problem definition | All |
| System Design | All |
| Data source, ingestion and feature engineering | All |
| Machine Learning Model | All |
| Model evaluation | All |
| Model serving | **Juan Escalona** |
| Reflexion | All |
| Broader impact | All |