

Dogecoin Price Prediction using Deep Learning and Twitter Trends



Adista Nursani

Helen Nguyen

Rohit Chamarthi

Srujan Kumar Karri

Contents

| | |
|--|----|
| Introduction | 3 |
| Data Sources | 4 |
| Architectural Design | 5 |
| Extract-Transform-Load (ETL) using AWS | 6 |
| Data Storage and Retrieval | 10 |
| Data Cleaning | 13 |
| Data Joining..... | 14 |
| Data Visualization..... | 16 |
| Machine Learning and Deep Learning | 17 |
| Challenges..... | 28 |
| Results and Discussions | 29 |
| References | 31 |

Introduction



In 2010, Bitcoin was valued at \$0.0008. Today, one Bitcoin equals to \$60,796.40. It means that you should have become a millionaire if you had invested \$1,000 a decade ago. But if you did not, do not worry. There are still multiple cryptocurrencies that you can invest in.

In 2021, we see the growth of many different cryptocurrencies beside Bitcoin such as Ethereum (ETH), Binance Coin (BNB). Beside the cryptocurrencies that have been

supported and used for a long time, Dogecoin, a cryptocurrency that was created as a joke, experienced a surprisingly growth from \$0.008 in the beginning of 2021 to an all-time high of \$0.69 in the middle of the year. How can a joke increase more 85 times? The support from the richest man in the world, Elon Musk and other billionaires such as Mark Cuban on social media platform, Twitter is believed to drive the intense growth of this meme coin. Many people who invested in Dogecoin in the early of the year have earned a great interest. However, after its all-time high in May 2021, the value of Dogecoin has been decreasing. The average value of Dogecoin in October 2021 is about \$0.24. It means that if you bought Dogecoin at \$0.6 because you believed that Elon Musk can make Dogecoin increase to \$1, you would have lost 60% of your investment if you still keep it now.

The fluctuation in the value and volume of Dogecoin, a meme coin with infinite supply, unlike Bitcoin which has very limited supply, and the great impact of social media impact on it inspired us to focus on Dogecoin and Twitter for this project.

The goals of our project are:

- Apply machine learning and deep learning models to predict the price of Dogecoin using the number of tweets mentioning about Dogecoin and sentiment data.
- Compare the performance of different models and propose the most consistent and accurate model to predict the price of Dogecoin.

By achieving those goals, we can help investors reduce the risk and increase the chance of winning when investing in not only Dogecoin but also social media-driven cryptocurrencies such as Shiba Inu and even stocks such as AMC or Gamestop.

Data Sources

There are two data sources needed for our project: the Dogecoin market data and tweets from Twitter. For this project, we will analyze and build machine and deep learning models during an approximately two-year data range, from July 2019 to November 2021.

The Dogecoin market data was downloaded from the 'Cryptocurrency extra data – Dogecoin' dataset on [kaggle.com](https://www.kaggle.com). This dataset is an ongoing competition that auto-updates Dogecoin market prices per one-minute increments.

| | Dogecoin Market Data |
|-------------------|----------------------|
| File Size | 148 MB |
| File Type | CSV |
| Number of Rows | 1247056 rows |
| Number of Columns | 9 columns |

The columns and data types of the Dogecoin market data can be seen below:

| Columns | Data Type |
|-----------|-----------|
| timestamp | unix |
| count | float64 |
| open | float64 |
| high | float64 |
| low | float64 |
| close | float64 |
| volume | float64 |
| vwap | float64 |
| target | float64 |

For Twitter data, we collected all tweets including the word 'Dogecoin' from 2016-01-01 to 2021-11-16 using snsrape, an open-source scraper for social networking services (SNS). The Python package that we used can be found here: <https://github.com/JustAnotherArchivist/snsrape>. It is noticeable that we only collect tweets that are written in English. Natural Language Processing (NLP) and Sentiment Analysis will be performed on all tweets. After that, it will be aggregated to hourly level and by date to join with the Dogecoin market data.

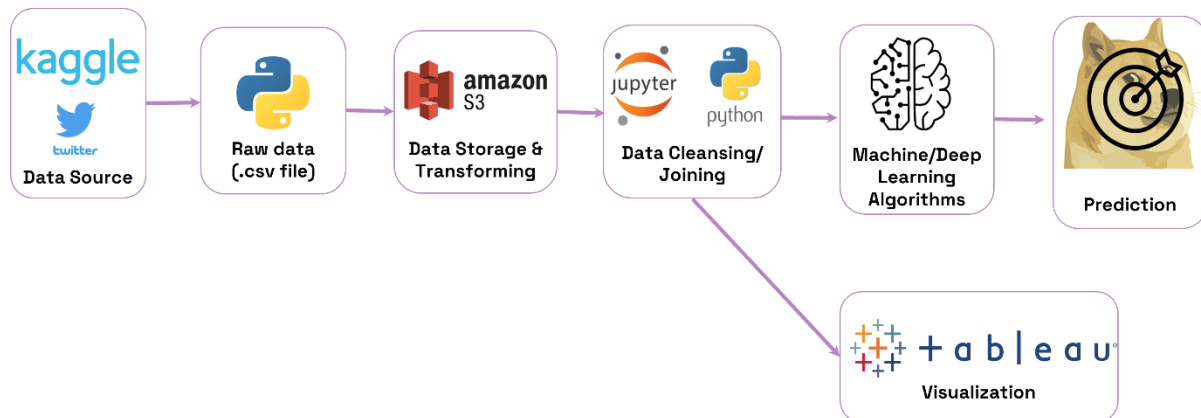
| | Twitter Tweets |
|-------------------|----------------|
| File Size | 1.72GB |
| File Type | CSV |
| Number of Rows | 5199031 rows |
| Number of Columns | 9 columns |

The columns and data types can be seen below:

| Columns | Data Type |
|--------------|-----------|
| datetime | object |
| text | object |
| replycount | float64 |
| retweetcount | float64 |
| likecount | float64 |
| quotecount | float64 |
| hashtags | object |
| cashtags | object |
| lang | object |

We will not use all the columns for the project. Removing unnecessary columns will be performed in the data cleaning part.

Architectural Design



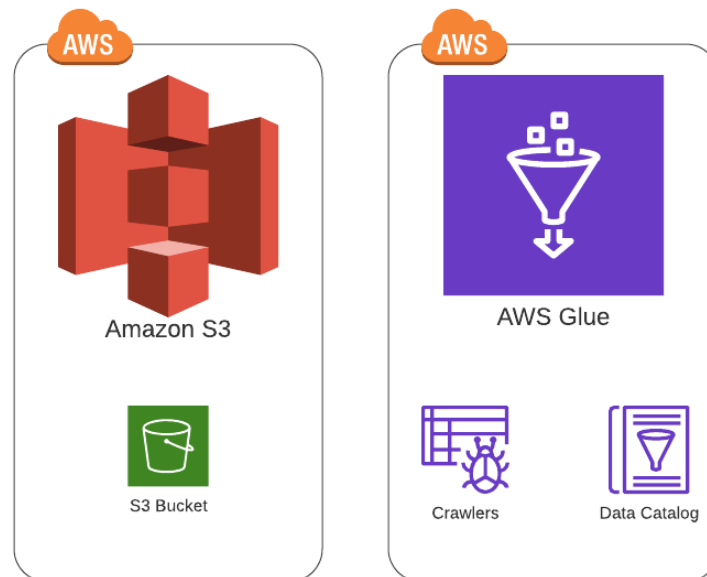
Above is the proposed architecture design to predict the Dogecoin prices based on the historical market data and how people react to Dogecoin on Twitter, one of the most popular social media platforms. The proposed architecture design works as follows. Firstly, we collect data from our data sources, kaggle.com and Twitter. The kaggle.com data is a csv file of Dogecoin market prices that is automatically updated daily. At the same time, we use the 'snsraper' package on Python to collect all tweets including the word 'Dogecoin' that are written in English. After obtaining the data, Amazon S3 will be used as a data storage and for transforming data. Then we will use Jupyter Notebook and Python to interact directly with Amazon S3 via their provided SDK to get the data. In our Jupyter Notebook, we will clean the data and then join the two datasets together. With this new dataset, Tableau is used to build visualizations to understand the data better. Next, we will perform natural language processing and sentiment analysis to collect

polarity and subjectivity scores of Twitter tweets. The next step will be aggregating the data to an hourly level to join two datasets. Then, machine learning and deep learning models will be conducted with selected features multiple times to find the best model which has the best accuracy.

Extract-Transform-Load (ETL) using AWS

In this project, we are using AWS platform and three of its services for data storage and Extract-Transform-Load operations on the raw data from our data sources. The services we will be using are

- S3 (Simple Storage Service)
- Glue/Glue Studio
 - Crawler
 - Data Catalog
 - Glue ETL job
- Athena



Firstly, the extracted raw data will be stored in S3 buckets, and a more detailed description has been mentioned in the upcoming data storage section. In this section, we will talk about the AWS Glue service and how it will be used to carry out an ETL task seamlessly by leveraging the cloud capabilities and serverless architecture.

AWS Glue

AWS Glue is a serverless data integration service that makes it easy to discover, prepare, and combine data for analytics, machine learning, and application development. AWS Glue provides all the capabilities

needed for data integration so that you can start analyzing your data and putting it to use in minutes instead of months.

Data integration is the process of preparing and combining data for analytics, machine learning, and application development. It involves multiple tasks, such as discovering and extracting data from various sources; enriching, cleaning, normalizing, and combining data; and loading and organizing data in databases, data warehouses, and data lakes. These tasks are often handled by different types of users that each use different products.

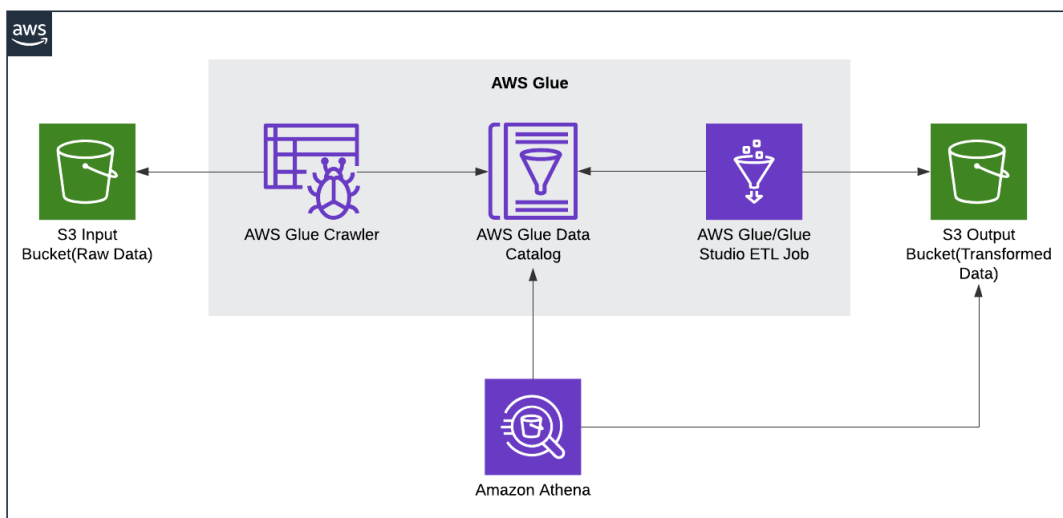
Components of AWS Glue

- Data catalog: The data catalog holds the metadata and the structure of the data.
- Database: It is used to create or access the database for the sources and targets.
- Table: Create one or more tables in the database that can be used by the source and target.
- Crawler and Classifier: A crawler is used to retrieve data from the source using built-in or custom classifiers. It creates/uses metadata tables that are pre-defined in the data catalog.
- Job: A job is business logic that carries out an ETL task. Internally, Apache Spark with Python or scala language writes this business logic.
- Trigger: A trigger starts the ETL job execution on-demand or at a specific time.
- Development endpoint: It creates a development environment where the ETL job script can be tested, developed, and debugged.

Benefits of AWS Glue

- Faster data integration
- Data integration automation
- Serverless service

Project use case (AWS Architecture)



The extracted raw data from Twitter (tweets) and Kaggle (Dogecoin price) will be stored in a S3 bucket that serves as an input storage option. Using AWS Glue, a Crawler is created that retrieves data from the input bucket and then populates the metadata tables that are pre-defined based on the input data fields in the data catalog. In an event where the crawler fails to detect the metadata automatically, a classifier is used to manually enable the crawler to populate the data catalog with the appropriate metadata. Then, an ETL job is created by specifying various parameters such as the data source location, transform type (change schema), data target (create tables in your data target), and finally selecting the desired schema. We plan to achieve various data transformations like data type manipulation, removing duplicates and null values from data, dropping unwanted columns, etc. The detailed process of data transformations, both for tweets and price datasets, can be reviewed from the snapshots below.

DogeETL job

Job has not been saved
Save
Delete
Run

Visual
Script
Job details
Runs
Schedules

Source
Transform
Target
Undo
Redo
Remove

Node properties
Transform
Output schema
Data preview

Data source - S3 bucket
S3 bucket

Transform - ApplyMapping
ApplyMapping

Data target - S3 bucket
S3 bucket

| Source key | Target key | Data type | Drop |
|------------|------------|-----------|-------------------------------------|
| unnamed: 0 | | | <input checked="" type="checkbox"/> |
| timestamp | timestamp | string | <input type="checkbox"/> |
| asset_id | | | <input checked="" type="checkbox"/> |
| asset_name | | | <input checked="" type="checkbox"/> |
| weight | | | <input checked="" type="checkbox"/> |
| count | count | double | <input type="checkbox"/> |
| open | open | double | <input type="checkbox"/> |
| high | high | double | <input type="checkbox"/> |
| low | low | double | <input type="checkbox"/> |
| close | close | double | <input type="checkbox"/> |

dogetweetsETL1 job Job has not been saved Save Delete Run

Visual | Script | Job details | Runs | Schedules

Source | Transform | Target | Undo | Redo | Remove | 🔍 | 🔍 | 📐

Node properties | **Transform** | Output schema | Data preview

Apply mapping

| Source key | Target key | Data type | Drop |
|--------------|--------------|-------------|-------------------------------------|
| url | | | <input checked="" type="checkbox"/> |
| datetime | datetime | timestamp ▼ | <input type="checkbox"/> |
| tweetid | | | <input checked="" type="checkbox"/> |
| text | text | string ▼ | <input type="checkbox"/> |
| username | | | <input checked="" type="checkbox"/> |
| verified | | | <input checked="" type="checkbox"/> |
| location | | | <input checked="" type="checkbox"/> |
| replycount | replycount | double ▼ | <input type="checkbox"/> |
| retweetcount | retweetcount | double ▼ | <input type="checkbox"/> |
| likecount | likecount | double ▼ | <input type="checkbox"/> |

This glue ETL job can be triggered on-demand, or at a specified time, or even on a specific trigger event. Once the job is run, it loads the S3 output bucket with the transformed data. The data from the output file can be directly accessed through the Jupyter Notebook with Python scripts.

Amazon Athena

Amazon Athena is a serverless query service that makes analysis of data, using standard SQL, stored in Amazon S3 simpler. With a few clicks in the AWS Management Console, customers can point Amazon Athena at their data stored in Amazon S3 and run queries using standard SQL to get results in seconds.

With Amazon Athena, there is no infrastructure to set up or manage, and the customer pays only for the queries they run. Amazon Athena scales automatically, executing queries in parallel, which gives fast results, even with a large dataset and complex queries.

Features Of Athena

Out of the many services provided by Amazon, Athena is one of the services. It has many features that make it suitable for Data Analysis.

- **Easy Implementation:** Athena doesn't require installation. It can be accessed directly from the AWS Console also directly by AWS CLI.
- **Serverless:** It is serverless, so the end-user doesn't need to worry about infrastructure, configuration, scaling or failure. Athena takes care of everything on its own.
- **Pay per query:** Athena charges you only for the query you run, i.e., the amount of data that is managed per query. You can save a lot if you can compress them and format your dataset accordingly.

- **Fast:** Athena is a very fast analytics tool. It can perform complex queries in less time by breaking the complex queries into simpler ones and running them parallelly, then combining the results to give the desired output.
- **Secure:** With the help of IAM policies and AWS Identity, Athena gives you complete control over the data set. As the data is stored in S3 buckets, IAM policies can help you manage control to users.
- **Highly available:** With the assurance of AWS, Athena is highly available, and the user can execute queries round the clock. As AWS is 99.999% available, so is Athena.
- **Integration:** The best feature of Athena is that it can be integrated with AWS Glue. AWS Glue will help the user to create a better-unified data repository. This helps to create better versioning of data, better tables, views, etc.

In this project, Athena has been used to carry out exploratory data analysis using SQL commands just to get an initial idea about the data and its features.

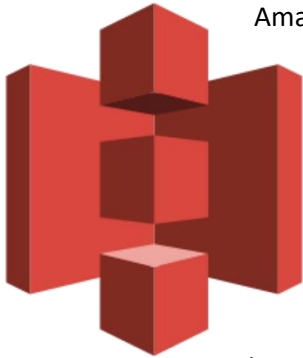
The screenshot displays the Amazon Athena Query Editor interface. On the left, a sidebar contains navigation options: 'Query editor', 'Workgroups', 'Data sources', 'Jobs', 'Workflows', and 'New Athena experience'. The main workspace is divided into several sections. The 'Data' section on the left shows 'Data Source' as 'AwsDataCatalog' and 'Database' as 'dogetweets5'. Below this, a 'Tables and views' section lists 'raw1' under 'Tables (1)'. The central area shows the SQL query: `SELECT * FROM "dogetweets5"."raw1" limit 10;`. Below the query, a status bar indicates 'Completed' with metrics: 'Time in queue: 0.1 sec', 'Run time: 0.479 sec', and 'Data scanned: 1.02 MB'. The 'Results (10)' section shows a table with columns: 'url', 'datetime', 'tweetid', and 'text'. The first row of data is:

| url | datetime | tweetid | text |
|---|---------------------------|-------------------|------------------|
| https://twitter.com/topcryptostats/status/73183163022921729 | 2016-05-15 13:00:32+00:00 | 73183163022921729 | "Dogecoin - DOGE |

Data Storage and Retrieval

We are using Amazon S3 for storing both the raw and the transformed data that can be retrieved by using AWS SDK Boto3. The raw data will be stored in an input S3 bucket whereas the transformed data can be accessed from the output S3 bucket. A brief overview about Amazon S3 and AWS SDK Boto3 is presented below.

Amazon S3



Amazon Simple Storage Service (Amazon S3) is an object storage service offering industry-leading scalability, data availability, security, and performance. Customers of all sizes and industries can store and protect any amount of data for virtually any use case, such as data lakes, cloud-native applications, and mobile apps. With cost-effective storage classes and easy-to-use management features, you can optimize costs, organize data, and configure fine-tuned access controls to meet specific business, organizational, and compliance requirements.

To understand better we need to look closely into some simple concepts. Amazon S3 stores the data as objects within buckets. An object can consist of files or any metadata that describes that file. There are certain steps to follow to store an object in S3, when you upload a file, you can set certain permissions over the bucket as well as for the file. Buckets are containers for objects. You can create more than one bucket. By default, users can provision up to 100 buckets per AWS account. However, you can increase your Amazon S3 bucket limit by visiting AWS Service Limits. An object can be 0 bytes to 5TB. For objects larger than 100 megabytes, users should consider using the Multipart Upload capability.

There are various storage spaces available in S3 depending on the user's requirement. The following are the storage options:

- Standard S3 (General Purpose)
- S3 intelligent Tiering
- Infrequent Access
- S3 One Zone
- Infrequent Access (S3 one Zone IA)
- S3 Glacier / S3 Glacier Deep Archive

In our project we are using Standard S3 storage class as it comes with the free tier AWS account and has lightning speed retrieval time. Also, here are the detailed features of Standard S3 storage.

Standard S3 (General Purpose)

- High durability, availability, and performance object storage for frequently accessed data
- Designed for durability of 99.999999999% of objects across multiple availability zones
- S3 Lifecycle management for automatic migration of objects to other S3 Storage classes

As mentioned, we will be using different buckets for input and output data storage and the same can be visualized from the image below.

Amazon S3 > a-glue-tothemoon

a-glue-tothemoon [Info](#)

Objects | Properties | Permissions | Metrics | Management | Access Points

Objects (3)

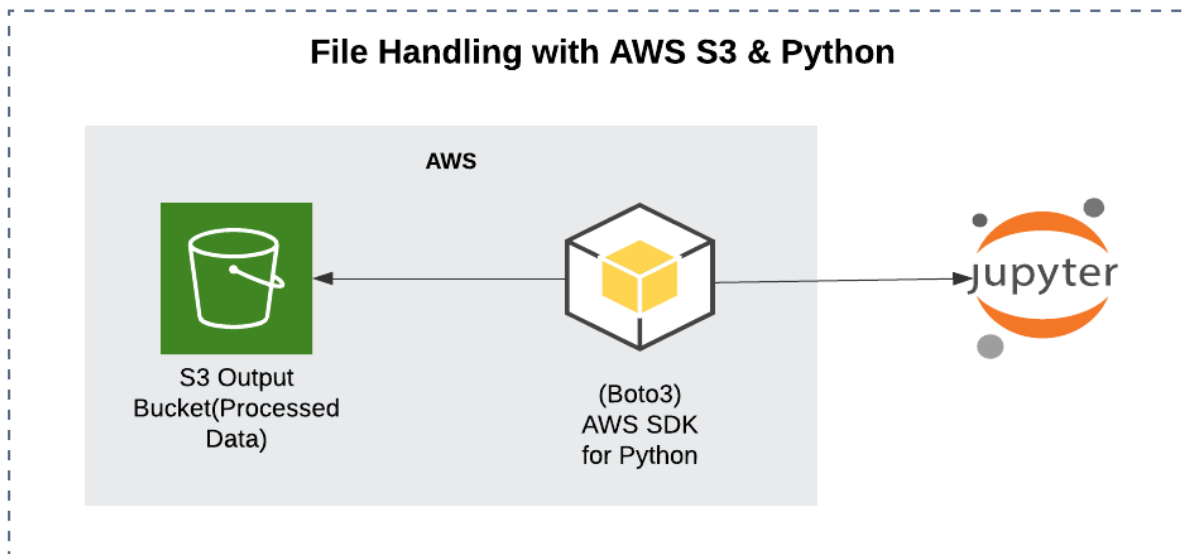
Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Find objects by prefix

| <input type="checkbox"/> | Name | Type | Last modified | Size | Storage class |
|--------------------------|---------|--------|---------------|------|---------------|
| <input type="checkbox"/> | input/ | Folder | - | - | - |
| <input type="checkbox"/> | output/ | Folder | - | - | - |
| <input type="checkbox"/> | temp/ | Folder | - | - | - |

AWS SDK Boto3

Boto3 is the Amazon Web Services (AWS) SDK for Python. It enables Python developers to create, configure, and manage AWS services, such as EC2 and S3. Boto provides an easy to use, object-oriented API, as well as low level access to AWS services.



Automated File Handling

The extracted data from the data sources can be uploaded into the S3 input bucket directly from the Jupyter Notebook using boto3. Also, the transformed data from S3 output bucket can be retrieved in the Jupyter Notebook with Python scripts. For a seamless experience, we configured the AWS account onto

our local machine before implementing Boto3. A snippet of data retrieval from an S3 bucket is shown below.

```
In [1]: import boto3
```

```
In [2]: client = boto3.client('s3')
```

```
In [3]: import pandas as pd
import s3fs
```

```
In [31]: path = 's3://doge-pred-project/processed/run-S3bucket_node3-1-part-r-00000'
price_aws = pd.read_csv(path)
price_aws.head()
```


```
Out[31]:
```

| | timestamp | count | open | high | low | close | volume | vwap | target |
|---|------------|-------|---------|--------|---------|--------|------------|---------|--------|
| 0 | 1562284800 | 521.0 | 0.00449 | 0.0046 | 0.00376 | 0.0042 | 60726008.0 | 0.00976 | -0.0 |
| 1 | 1562284860 | 521.0 | 0.00449 | 0.0046 | 0.00376 | 0.0042 | 60726008.0 | 0.00976 | -0.0 |
| 2 | 1562284920 | 521.0 | 0.00449 | 0.0046 | 0.00376 | 0.0042 | 60726008.0 | 0.00976 | -0.0 |
| 3 | 1562284980 | 521.0 | 0.00449 | 0.0046 | 0.00376 | 0.0042 | 60726008.0 | 0.00976 | -0.0 |
| 4 | 1562285040 | 521.0 | 0.00449 | 0.0046 | 0.00376 | 0.0042 | 60726008.0 | 0.00976 | -0.0 |

```
In [7]: path = "s3://doge-pred-project/processed1/run-S3bucket_node3-1-part-r-00000"
tweets_aws = pd.read_csv(path)
tweets_aws.head()
```

C:\Users\Helen Nguyen\anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3165: DtypeWarning: Columns (8) have mixed types.Specify dtype option on import or set low_memory=False.
has_raised = await self.run_ast_nodes(code_ast.body, cell_name,

```
Out[7]:
```

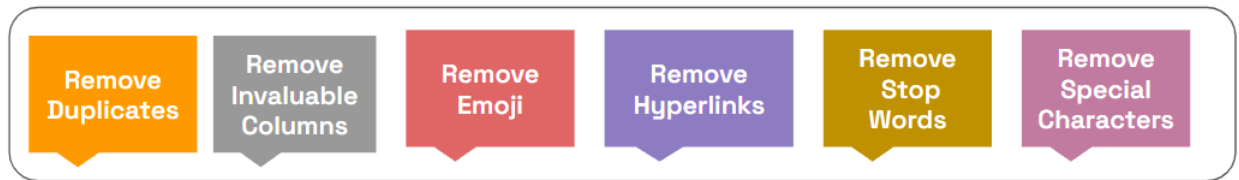
| | datetime | text | replycount | retweetcount | likecount | quotecount | hashtags | cashtags | lang |
|---|-----------------------|--|------------|--------------|-----------|------------|---|----------|------|
| 0 | 2021-04-30 15:28:29.0 |  241,540,531 #DOGE (73.972.750 USD) move from ... | 0.0 | 0.0 | 0.0 | 0.0 | ['DOGE', 'DOGECOIN', 'DOGE'] | NaN | en |
| 1 | 2021-04-30 15:28:12.0 | @dogecoin_rise Yeah nah Mate. | 0.0 | 0.0 | 0.0 | 0.0 | NaN | NaN | en |
| 2 | 2021-04-30 15:27:59.0 | @krakenfx Only #dogecoin because of #dogecoin ... | 0.0 | 1.0 | 1.0 | 0.0 | ['dogecoin', 'dogecoin', 'dogearmy', 'DogeCoin...'] | NaN | en |
| 3 | 2021-04-30 15:27:57.0 | #haddinibilkripto @binance @elonmusk @dogecoin... | 0.0 | 0.0 | 1.0 | 0.0 | ['haddinibilkripto', 'haddinibildirichen'] | NaN | en |
| 4 | 2021-04-30 15:27:55.0 | @LabzSpider Ok id happily take your dogecoin t... | 1.0 | 0.0 | 0.0 | 0.0 | NaN | NaN | en |

```
In [8]: tweets_aws.shape
```

```
Out[8]: (5579110, 9)
```

Data Cleaning

Data cleaning is the most important step in any machine learning model, but it is even more important for Natural Language Processing. Without a cleaning process, the Twitter data is just a cluster of words that the computer does not understand. Here, we will go over the following steps to clean the data.



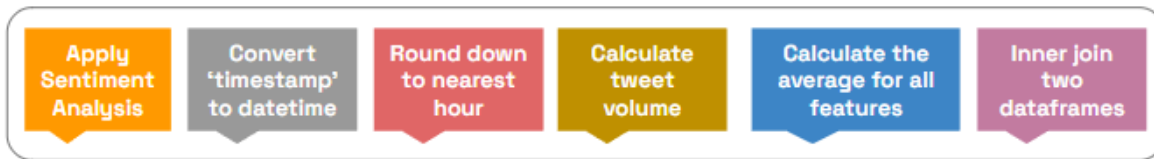
The steps of data cleaning are below:

- Step 1: Before performing NLP, it is noticed that the dataset includes duplicates. Hence, we removed the rows with the same URLs.
- Step 2: Raw data from Twitter comes with irrelevant data which is not useful for analysis. These columns which do not add any value to the analysis must be removed.
- Step 3: Emojis do not provide any additional information for sentiment analysis using TextBlob. Hence, remove these too using the library named re, which provides regular expression matching operations. We remove emoticons, symbols & pictographs, transport & map symbols, flags (iOS), Chinese characters by replacing their corresponding Unicode.
- Step 4: URLs (Uniform Resource Locators) in a text are references to a location on the web, but do not provide any additional information for analysis. Hence, remove these too using the library named re, which provides regular expression matching operations. We remove words starting with http.
- Step 5: Stop words are irrelevant words that won't help in identifying a text as real or fake. These are words that are not required for tasks such as sentiment analysis or text classification. Words like I, me, you, he, she, the, etc. increase the size of text data but don't improve results dramatically and thus we will remove these. We will need filter certain pre-defined words, which can give unwanted bias during modeling. For example, delete the names, places, or other words that can make the algorithm group the data samples according to only one feature. Then, we create our own list of stop words.
- Step 6: Special characters are removed depending on the use case. Special characters like # (hash), – (hyphen), / (slash) etc. do not add any value to sentiment analysis, so we remove those.

Data Joining

The data from Kaggle includes information about Dogecoin's open price, highest price, lowest price, close price, trading volume daily starting from 2019-07-05 to 2021-11-16. The 'timestamp' in the data is for every minute.

A data join is possible when two data sets are combined in a side-by-side manner, therefore at least one column in each data set must be the same. Hence, we will go over the following steps to join the data.



- Step 1: In order to get the most accurate result for polarity and subjectivity, we applied Sentiment Analysis by using TextBlob for all 5,199,031 tweets. The algorithm and the result will be discussed more detailed in the Machine Learning part.
- Step 2: To join two dataframes, it is compulsory to convert timestamp in Dogecoin price data to datetime.
- Step 3: Even though we have Dogecoin price and tweets for every minute, joining the Twitter data for every minute could result in 5 million records which is not a practical approach for this project and the result is not practical in real life. It will only result in creating time complexity and space complexity. Hence, we will round down the datetime column in both tables to the nearest hour and joining both tables by hour.
- Step 4: Tweet volume can be a valuable feature for our models later. To get tweet volume for every hour, we use size() function count the number of rows group by hour.
- Step 5: The next step is to calculate the average value for all features. Before this step, we have 5,199,031 of rows in Twitter data and 1,247,056 of rows corresponding to 1,247,056 minutes in Price data. Hence, to join those two dataframes, we need to calculate the average value for all features in each hour.
- Step 6: The last step is to inner join the Twitter data and the Dogecoin price data by hour. Below is the final dataset.

| | replycount | retweetcount | likecount | quotecount | Polarity | Subjectivity | tweet_volume | count | open | high | low | close | volume | vwap | target |
|---------------------|------------|--------------|-----------|------------|----------|--------------|--------------|------------|----------|----------|----------|----------|--------------|----------|---------------|
| 2019-07-05 01:00:00 | 2.500000 | 5.571429 | 35.714286 | 0.142857 | 0.105595 | 0.304626 | 14 | 521.000000 | 0.004490 | 0.004600 | 0.003760 | 0.004200 | 6.072601e+07 | 0.009760 | 0.000000e+00 |
| 2019-07-05 02:00:00 | 0.111111 | 0.444444 | 0.888889 | 0.111111 | 0.328704 | 0.398148 | 9 | 521.000000 | 0.004490 | 0.004600 | 0.003760 | 0.004200 | 6.072601e+07 | 0.009760 | 0.000000e+00 |
| 2019-07-05 03:00:00 | 0.111111 | 0.111111 | 0.444444 | 0.000000 | 0.135582 | 0.293360 | 9 | 521.000000 | 0.004490 | 0.004600 | 0.003760 | 0.004200 | 6.072601e+07 | 0.009760 | 0.000000e+00 |
| 2019-07-05 04:00:00 | 0.333333 | 0.266667 | 0.533333 | 0.000000 | 0.020889 | 0.227333 | 15 | 521.000000 | 0.004490 | 0.004600 | 0.003760 | 0.004200 | 6.072601e+07 | 0.009760 | 0.000000e+00 |
| 2019-07-05 05:00:00 | 0.000000 | 0.250000 | 0.250000 | 0.000000 | 0.068750 | 0.562500 | 4 | 521.000000 | 0.004490 | 0.004600 | 0.003760 | 0.004200 | 6.072601e+07 | 0.009760 | 0.000000e+00 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 2021-11-16 19:00:00 | 2.458525 | 2.483871 | 7.649770 | 0.108295 | 0.094937 | 0.346564 | 434 | 225.650000 | 0.239288 | 0.239510 | 0.239047 | 0.239290 | 6.090039e+05 | 0.558320 | -2.800856e-20 |
| 2021-11-16 20:00:00 | 1.131234 | 1.259843 | 6.299213 | 0.097113 | 0.078115 | 0.294097 | 381 | 385.566667 | 0.235380 | 0.235720 | 0.235028 | 0.235320 | 1.184139e+06 | 0.549188 | -2.800856e-20 |
| 2021-11-16 21:00:00 | 1.202312 | 1.242775 | 5.832370 | 0.083815 | 0.112142 | 0.290743 | 346 | 232.566667 | 0.238525 | 0.238768 | 0.238265 | 0.238588 | 7.032640e+05 | 0.556563 | -2.800856e-20 |
| 2021-11-16 22:00:00 | 3.077135 | 1.710744 | 7.344353 | 0.143251 | 0.080134 | 0.304684 | 363 | 108.966667 | 0.239797 | 0.239942 | 0.239673 | 0.239827 | 2.779830e+05 | 0.559557 | -2.800856e-20 |
| 2021-11-16 23:00:00 | 0.588415 | 0.868902 | 3.512195 | 0.048780 | 0.032054 | 0.321986 | 328 | 209.866667 | 0.238462 | 0.238603 | 0.238225 | 0.238410 | 5.023097e+05 | 0.556298 | -2.800856e-20 |

20715 rows × 15 columns

To better understand the final dataset, we created various visualizations to depict the relationships among attributes. Additionally, we generated a word cloud to see understand which words often appear when talking about Dogecoin.

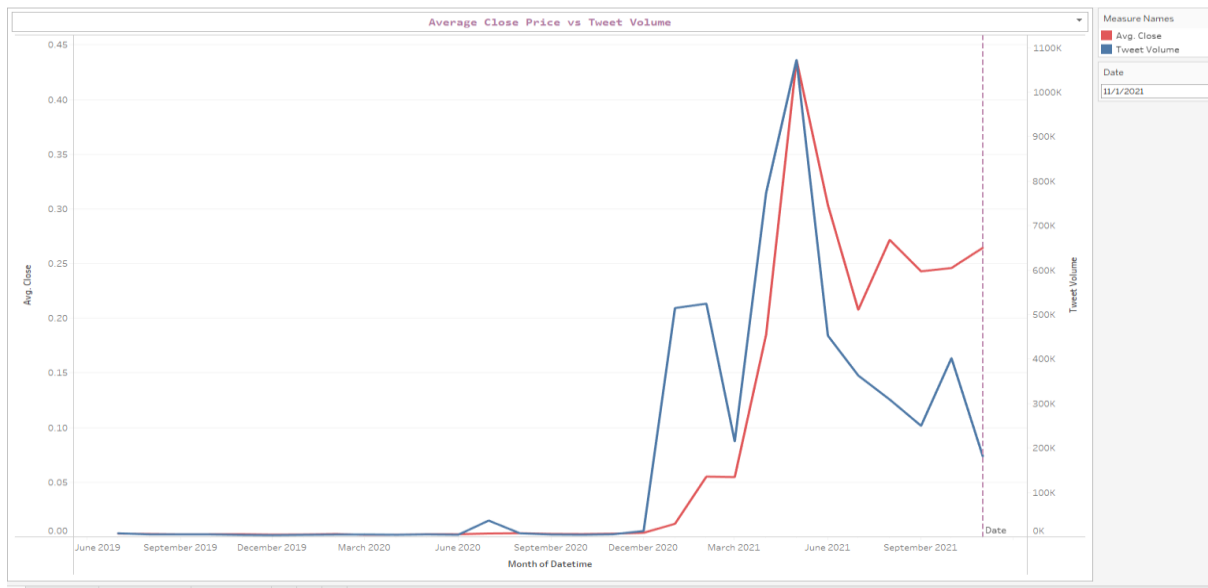


The chart displays two data series over time from June 2019 to June 2021. The 'Buy Volume' series (orange line) is measured on the left Y-axis (0M to 8000M), and the 'Tweet Volume' series (blue line) is measured on the right Y-axis (0K to 1100K). Both series show a major peak in early 2021, with Buy Volume reaching nearly 8000M and Tweet Volume reaching over 1100K. A vertical dashed green line marks the date June 2021.

| Date | Buy Volume (M) | Tweet Volume (K) |
|----------------|----------------|------------------|
| June 2019 | ~800 | ~10 |
| September 2019 | ~100 | ~10 |
| December 2019 | ~100 | ~10 |
| March 2020 | ~200 | ~10 |
| June 2020 | ~100 | ~10 |
| September 2020 | ~200 | ~10 |
| December 2020 | ~800 | ~500 |
| March 2021 | ~6000 | ~800 |
| June 2021 | ~1500 | ~400 |

16

visible that trading volume increased sharply in the beginning of 2021 and the tweet volume also increased at the mean time. The correlation is more clear recently when both features have been decreasing.



The above graph depicts the trends of both Dogecoin average close price and tweet volume during the same period. Similarly, we can see the correlation between the average close price and the tweet volume. Both features reached their peaks in May 2021. But recently, those features seem not to share the same direction anymore.

Machine Learning and Deep Learning

With the progress of faster computers, larger datasets, and algorithmic improvement, artificial intelligence (AI) capabilities in the past decade have progressed rapidly. For this project, we will be applying several types of AI techniques, specifically natural language processing and deep learning.

Natural Language Processing - Sentiment Analysis

It is predicted that 90% of data that is generated daily is defined as unstructured data. Natural language processing (NLP) is an important technique for understanding this type of data, namely speech and text. NLP enables machines to understand and obtain meaning from speech and textual data using algorithms so they can perform tasks related to language processing. Some applications of NLP include sentiment analysis of products, language translation, chatbot interactions, and topic classification.

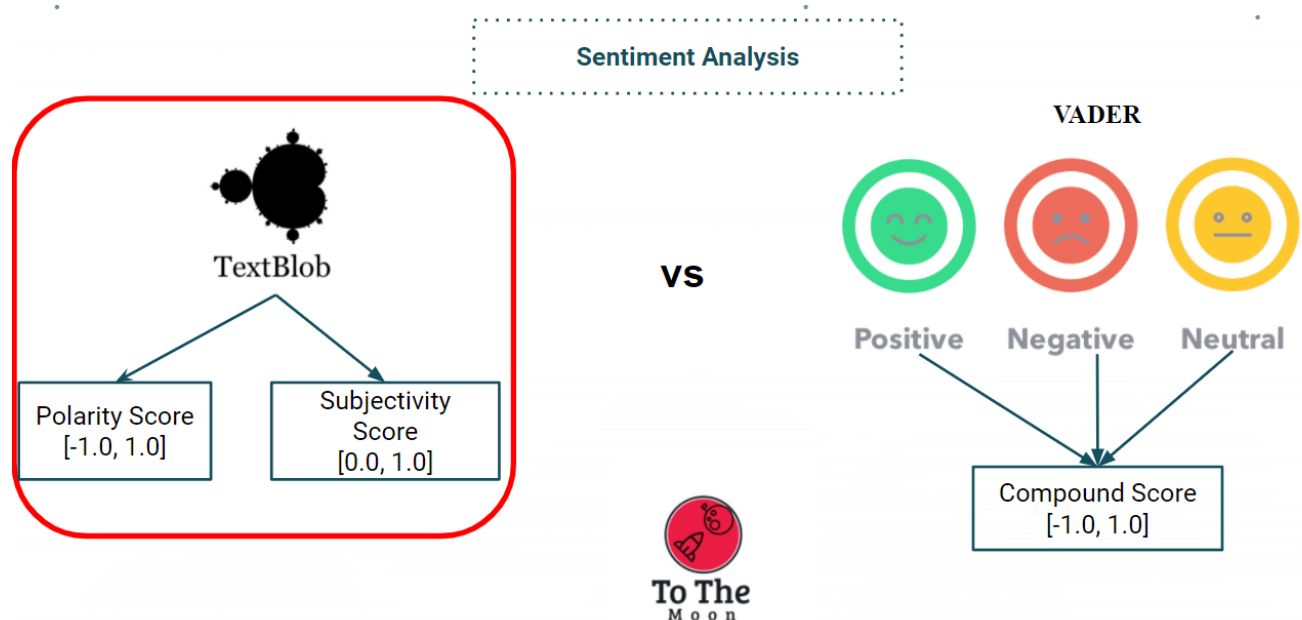
Sentiment analysis, also referred to as opinion mining, generally decides whether a phrase or set of text is positive, negative, or neutral in terms of sentiment. Sentiment analysis can also detect emotions such as happy, angry, sad, or intentions such as interested vs not interested.

Let's look at these three phrases.

- "Atlanta is a great city."
- "Atlanta's traffic is horrible."

- “Atlanta is the capital of the U.S. state of Georgia.”

Each of these phrases talks about the city of Atlanta in different sentiments. The first phrase expresses a positive tone while the second is a more negative sentiment about Atlanta. The third phrase contains neither positive nor negative words that may display it in either sentiment, therefore it is a neutral sentiment. In machine learning, this is a case of supervised learning, where a set of words and their applied sentiment are trained on a model, which are then tested on unlabeled phrases.

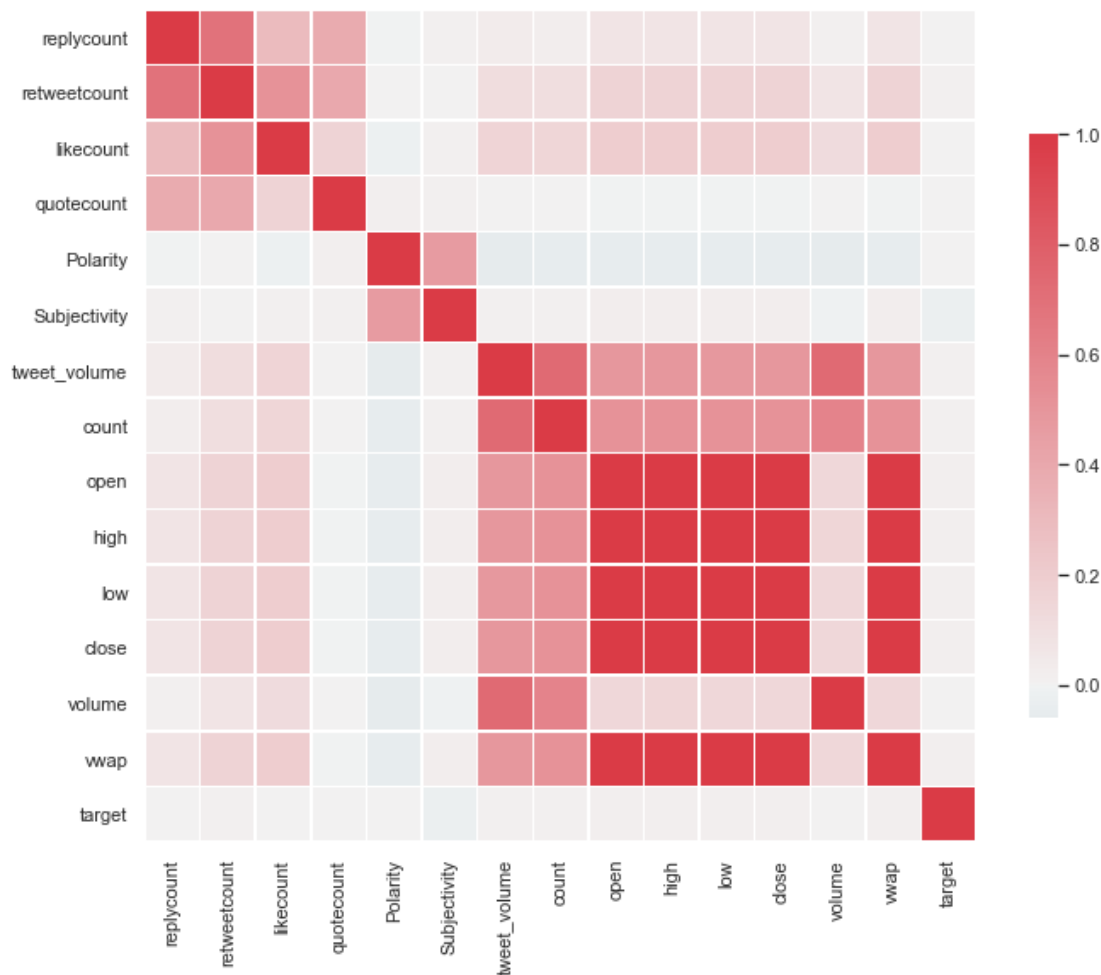


In term of Sentiment Analysis, TextBlob and VADER are the common libraries that can help return sentiment score. We assessed the performance of those libraries before making the decision on which library we should use. While TextBlob provides polarity score which reflects if the statement is positive or negative, and subjectivity score which reflects if the statement is opinion or fact-based, VADER instead provides negative score, neutral score, positive score and compound score which is the normalized score. We used both libraries on the Twitter data. The polarity score from TextBlob and the compound score from VADER are similar. However, TextBlob gives us subjectivity score which can be a valuable feature for our predictive models later. Also, TextBlob takes shorter time than VADER. As a result, we decided to use TextBlob to acquire polarity score and subjectivity score. Polarity is the difference in tendencies and is in a -1 to 1 range, where -1 is a more negative statement, and 1 is a positive statement. Subjectivity talks about how the statement will relate more to an opinion, emotion, or judgement, where objective is more factual information. Subjectivity will be in a 0 to 1 range where 0 is more to an objective tendency as opposed to 1, more subjective.

As already mentioned in the data joining section, the tweets will be aggregated hourly. The average values of these polarity and subjectivity scores will be calculated hourly and joined with the hourly Dogecoin data.

Feature Engineering

The price of cryptocurrencies in general and of Dogecoin specifically depends on various number of factors such as economics, government policies. Our dataset include the market data and the Twitter data including the sentiment score, subjectivity score and tweet volume that are calculated in the previous steps. To avoid multicollinearity which affects negatively to the result of our model, we need to remove highly correlated variables. Below is the correlation heatmap of our features.



It is noticeable that Open price, High Price, Low Price, VWAP (volume-weighted average price) and Close Price are the same thing. Hence, we only need to keep the close price. It is noticeable that we are using the average values for all the features. Hence, the average replycount, retweetcount, likecount, quotecount, and target are not valuable for our models. Consequently, our final dataset will only include the following features:

- Close Price: The average USD price at the end of minutes during the hour.
- Volume: The average number of Dogecoin traded of minutes during the hour.
- Tweet Volume: The average number of tweets including the word 'Dogecoin' during the hour.

- Subjectivity: The average subjectivity score number of tweets including the word 'Dogecoin' during the hour.
- Polarity: The average polarity score number of tweets including the word 'Dogecoin' during the hour.

Below is our final dataset with 20,715 rows:

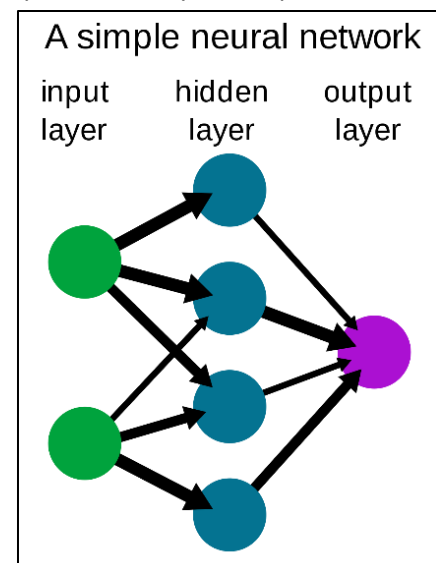
| | close | volume | tweet_volume | Subjectivity | Polarity |
|---------------------|----------|--------------|--------------|--------------|----------|
| 2019-07-05 01:00:00 | 0.004200 | 6.072601e+07 | 14 | 0.304626 | 0.105595 |
| 2019-07-05 02:00:00 | 0.004200 | 6.072601e+07 | 9 | 0.398148 | 0.328704 |
| 2019-07-05 03:00:00 | 0.004200 | 6.072601e+07 | 9 | 0.293360 | 0.135582 |
| 2019-07-05 04:00:00 | 0.004200 | 6.072601e+07 | 15 | 0.227333 | 0.020889 |
| 2019-07-05 05:00:00 | 0.004200 | 6.072601e+07 | 4 | 0.562500 | 0.068750 |
| ... | ... | ... | ... | ... | ... |
| 2021-11-16 19:00:00 | 0.239290 | 6.090039e+05 | 434 | 0.346564 | 0.094937 |
| 2021-11-16 20:00:00 | 0.235320 | 1.184139e+06 | 381 | 0.294097 | 0.078115 |
| 2021-11-16 21:00:00 | 0.238588 | 7.032640e+05 | 346 | 0.290743 | 0.112142 |
| 2021-11-16 22:00:00 | 0.239827 | 2.779830e+05 | 363 | 0.304684 | 0.080134 |
| 2021-11-16 23:00:00 | 0.238410 | 5.023097e+05 | 328 | 0.321986 | 0.032054 |

20715 rows × 5 columns

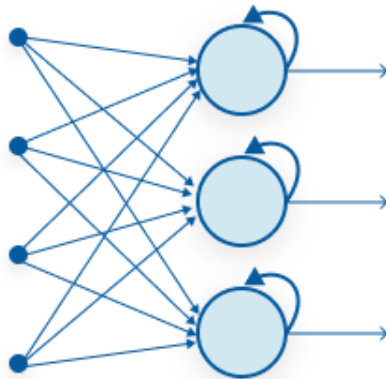
Deep Learning – Long Short-Term Memory (LSTM) Time-Series

Deep learning is a subset of machine learning where the learning component is enhanced, meaning that the model will contain several successful layers of learning, called depth. These layered representations are learned via models known as neural networks. The term 'neural network' is referenced from interconnected neurons in the human brain and how they send signals to learn from one another. The main layers of a neural network is the input layer (raw data), hidden layer, and the output layer. Each layer attempts to learn a weight or coefficient.

This simple neural network image depicts the input which is processed in one direction. When processing sequential data, this model is less reliable since each node is only dependent on the output of the previous node. This problem can be solved by a recurrent neural network (RNN). Nodes in the hidden layer of an RNN retains information on not just the previous node, but nodes



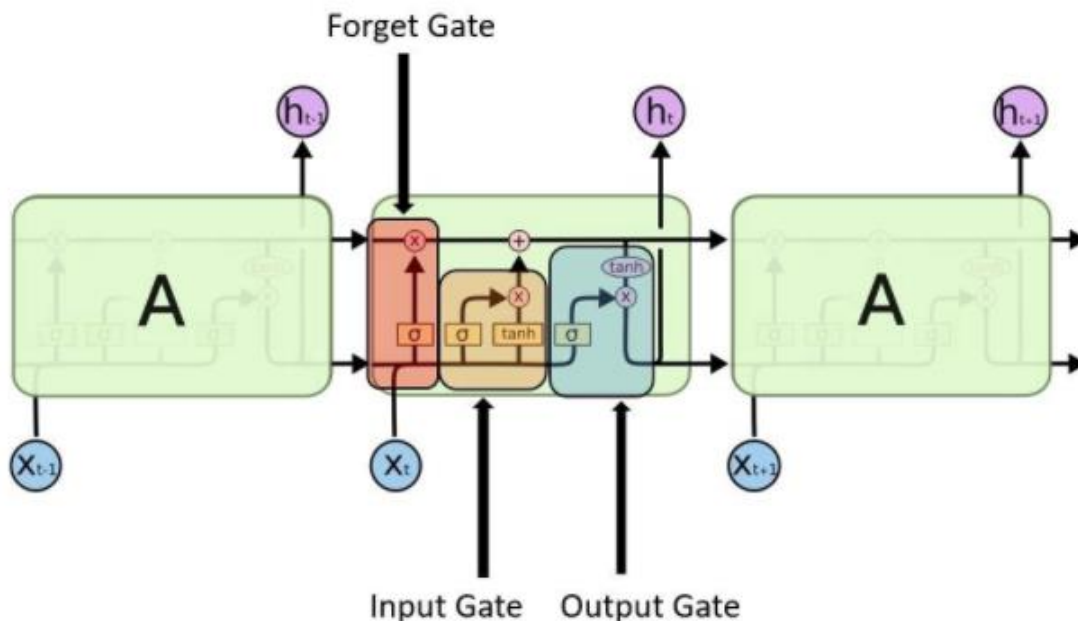
before that using a loop. The below image of an RNN shows the process. Sequential data have an advantage in RNN, where the algorithm feeds back the results to itself.



Recurrent Neural Network

Even though RNNs are able to take into account previous nodes, they do not consider the more important nodes in the network that may give more context. This is called “memory loss” or a short-term memory.

In this project, we will be using the evolution of the RNN appropriate for long sequential time series data, which is a Long Short-Term Memory (LSTM). LSTMs are capable of learning long-term dependencies. An important feature is the cell state which is like a conveyor belt of information. The algorithm learns, unlearns, or retain information which is decided by three gates: the input gate (decides which value to input), the forget gate (decides which value to forget), and the output gate (decides the output). This type of algorithm is suitable for learning the context needed to make predictions of Dogecoin in time series forecasting.



Convert Time Series to Supervised Learning

The final data frame structure consists of columns 'close', 'volume', 'tweet_volume', 'Subjectivity' and 'Polarity'.

| | close | volume | tweet_volume | Subjectivity | Polarity |
|---------------------|--------|------------|--------------|--------------|----------|
| 2019-07-05 01:00:00 | 0.0042 | 60726008.0 | 14 | 0.304626 | 0.105595 |
| 2019-07-05 02:00:00 | 0.0042 | 60726008.0 | 9 | 0.398148 | 0.328704 |
| 2019-07-05 03:00:00 | 0.0042 | 60726008.0 | 9 | 0.293360 | 0.135582 |
| 2019-07-05 04:00:00 | 0.0042 | 60726008.0 | 15 | 0.227333 | 0.020889 |
| 2019-07-05 05:00:00 | 0.0042 | 60726008.0 | 4 | 0.562500 | 0.068750 |

We used Minmax scaler to normalize the above data before proceeding with the analysis.

It is important to convert time series data to a supervised learning. In this case, 'close' is the target variable which is dependent on its previous sequence of all the feature inputs.

We developed an existing function which was available in public libraries to convert time series data to supervised learning. In this project we used inputs as past three hours data to predict the current 'close' price. After reframing the time series data to supervised learning it was important to drop the columns which are not required for prediction of 'close' price. The final reframed data structure consists of 16 features including the current 'close' price – var1(t).

| | var1(t-3) | var2(t-3) | var3(t-3) | var4(t-3) | var5(t-3) | var1(t-2) | var2(t-2) | var3(t-2) | var4(t-2) | var5(t-2) | var1(t-1) | var2(t-1) | var3(t-1) | var4(t-1) | var5(t-1) | var1(t) |
|---|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|----------|
| 3 | 0.003926 | 0.23925 | 0.000435 | 0.365551 | 0.443505 | 0.003926 | 0.23925 | 0.000268 | 0.477778 | 0.639214 | 0.003926 | 0.23925 | 0.000268 | 0.352032 | 0.469809 | 0.003926 |
| 4 | 0.003926 | 0.23925 | 0.000268 | 0.477778 | 0.639214 | 0.003926 | 0.23925 | 0.000268 | 0.352032 | 0.469809 | 0.003926 | 0.23925 | 0.000468 | 0.272800 | 0.369201 | 0.003926 |
| 5 | 0.003926 | 0.23925 | 0.000268 | 0.352032 | 0.469809 | 0.003926 | 0.23925 | 0.000468 | 0.272800 | 0.369201 | 0.003926 | 0.23925 | 0.000100 | 0.675000 | 0.411184 | 0.003926 |
| 6 | 0.003926 | 0.23925 | 0.000468 | 0.272800 | 0.369201 | 0.003926 | 0.23925 | 0.000100 | 0.675000 | 0.411184 | 0.003926 | 0.23925 | 0.000201 | 0.197143 | 0.401003 | 0.003926 |
| 7 | 0.003926 | 0.23925 | 0.000100 | 0.675000 | 0.411184 | 0.003926 | 0.23925 | 0.000201 | 0.197143 | 0.401003 | 0.003926 | 0.23925 | 0.003244 | 0.393988 | 0.469102 | 0.003926 |

We reshaped the data into samples, timesteps and number of features. Then, split the data into train and test sets: train_X, train_y, test_X, test_y.

This is where a model is required to make a one-hour price prediction, then the actual data for that hour is made available to the model so that it can be used as the basis for making a prediction on the subsequent hour. This is both realistic for how the model may be used in practice and beneficial to the models allowing them to make use of the best available data.

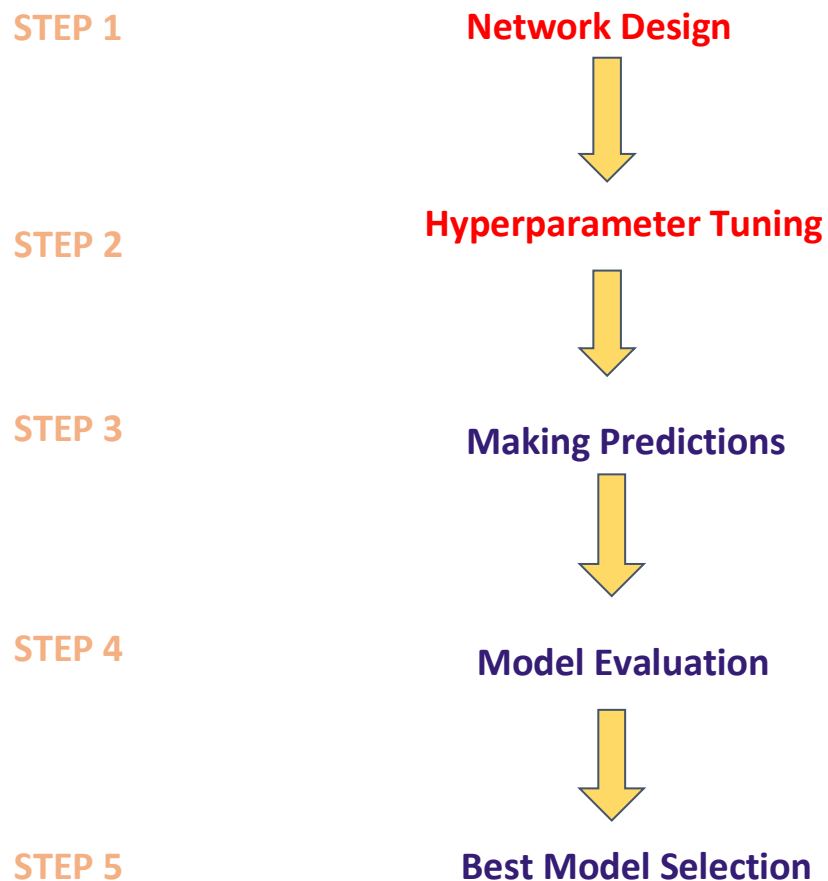
3 Following Neural Networks Models are applied for the prediction of Dogecoin:

1. LSTM (Long short-term memory)
2. CNN (Convolution Neural Network)
3. GRU (Gated recurring unit)

Why to use LSTM, CNN and GRU?

- Time series data is always featured with different seasonalities. The LSTM can capture the patterns of both long term seasonalities and short term seasonalities. It could also take inputs with different lengths. The different gates inside LSTM boost its capability for capturing nonlinear relationships for forecasting.
- CNN incorporates feature engineering in one framework, without doing it manually. They can extract very informative, deep features, which are independent from time.
- The GRU controls the flow of information like the LSTM unit, but without having to use a memory unit. Also, GRU is relatively new, and performance is on par with LSTM, but computationally more efficient. So, we experimented with it.

Steps involved in building and selection of Neural Networks:



Network designing and Hyperparameter tuning are the most crucial steps in improvising the performance of the neural networks. Let's look more closely how these steps are constructed.

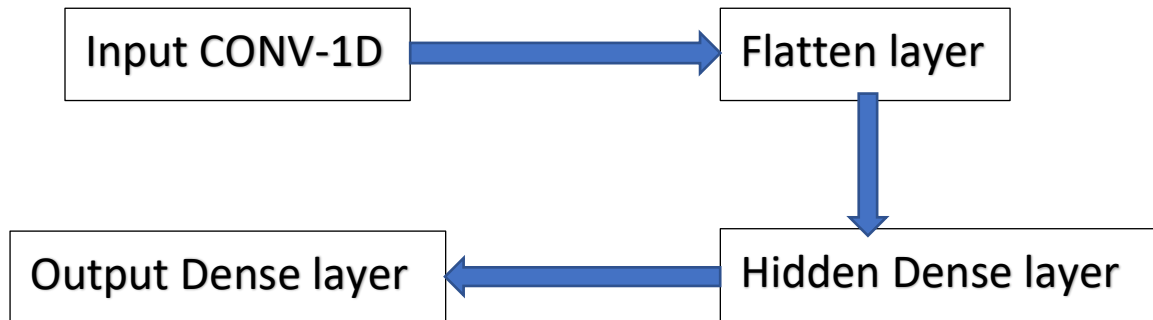
STEP 1: Network Design

LSTM:



- Input layer takes 5 neurons as input as we have only 5 features. No reason to make model complicated. It improves interpretability.
- No hidden layers in LSTM for the same reason as above. With hidden layers the change in model's performance did not change much.
- Output layer is a dense layer with prediction of 'close' price.

CNN:



- We require Flattening to convert the data into a 1-dimensional array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long feature vector.
- Hidden dense layers with multiple units and output layer dense layer with one unit close' price' prediction.

GRU:



- Input layer takes 5 neurons as input as we have only 5 features. No reason to make model complicated. It improves interpretability. Same as LSTM.
- No hidden layers in LSTM for the same reason as above. With hidden layers the change in model's performance did not change much.
- Output layer is a dense layer with prediction of 'close' price.

STEP 2: Hyperparameter Tuning

LSTM:

1. Activation function: We used default tanh (hyperbolic tangent function) function instead of opting relu (rectified linear unit) function. We found that when we used tanh activation on neuron the network learns faster than relu.
We came into this conclusion because accuracy on fixed test dataset was higher for tanh than relu. Also, loss value after 100 epochs until 500 was slightly lower for tanh.
2. Optimizer: Adam is the best among the adaptive optimizers in most of the cases to reduce loss function (also known as cost function). Even though we did not have sparse data in this data set it is said it works well with sparse data. The adaptive learning rate is very good for time series data sets. We tried several optimizing functions to minimize RMSE and increase the speed of the model. We noticed that adam was 2 minutes faster than traditional sgd function. Two minutes less might not be too important for this dataset but when dealing with large data in real life it could be very effective in dealing with time complexity.
3. Model fitting: The batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters. Whereas The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset. We tried epochs ranging from 100 to 600. We found that fitting model with 500 epochs and a batch size of 400 minimized RMSE than any other combination.

CNN:

1. Activation function: Relu function is faster in CNN than sigmoid function and it fixes the vanishing gradient problem especially when working with time series data there is a probability of vanishing gradient occurrence. The constant gradient of ReLUs results in faster learning. So, we adopted relu function.
2. Optimizer: Same as LSTM.
3. Model fitting: Same as LSTM.

GRU:

1. Activation function: Same as CNN.
2. Optimizer: Same as CNN.
3. Model fitting: Same as CNN.

Sample RMSE minimization summary

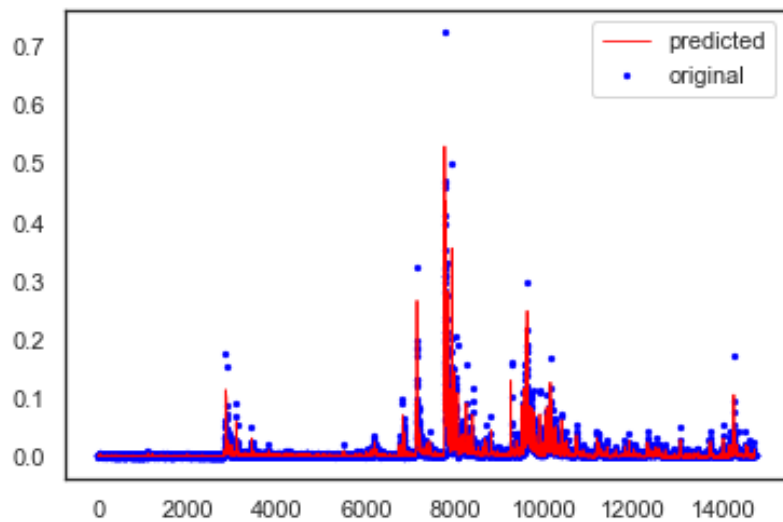
| Optimizers | | Adam - Epoch | | Batch size | |
|------------|------------------|--------------|------------------|------------|------------------|
| adam | Test RMSE: 0.140 | 100 | Test RMSE: 0.152 | 50 | Test RMSE: 0.174 |
| nadam | Test RMSE: 0.154 | 150 | Test RMSE: 0.175 | 100 | Test RMSE: 0.120 |
| adadelta | Test RMSE: 0.272 | 200 | Test RMSE: 0.147 | 120 | Test RMSE: 0.108 |
| adagrad | Test RMSE: 0.210 | 250 | Test RMSE: 0.170 | 150 | Test RMSE: 0.197 |
| Adamax | Test RMSE: 0.151 | 300 | Test RMSE: 0.139 | 200 | Test RMSE: 0.137 |
| ftrl | Test RMSE: 0.175 | 350 | Test RMSE: 0.119 | 300 | Test RMSE: 0.073 |
| msprop | Test RMSE: 0.160 | 400 | Test RMSE: 0.101 | 400 | Test RMSE: 0.065 |
| sgd | Test RMSE: 0.203 | 450 | Test RMSE: 0.169 | | |

| | | | | | |
|--|--|-----|------------------|--|--|
| | | 500 | Test RMSE: 0.100 | | |
| | | 550 | Test RMSE: 0.140 | | |
| | | 600 | Test RMSE: 0.118 | | |

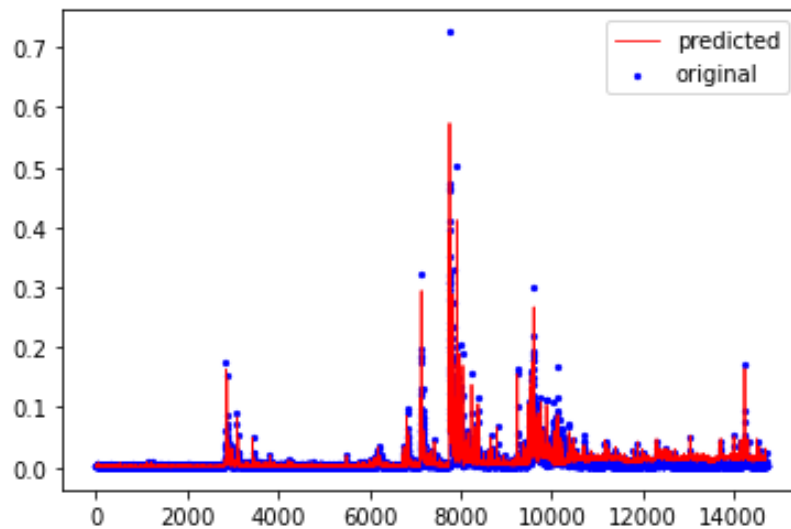
It can be inferred from the above table that we were able to optimize loss function and minimize RMSE. Due to limited time, we did not explore epochs > 600 and batch size > 400.

STEP 3: Making Predictions

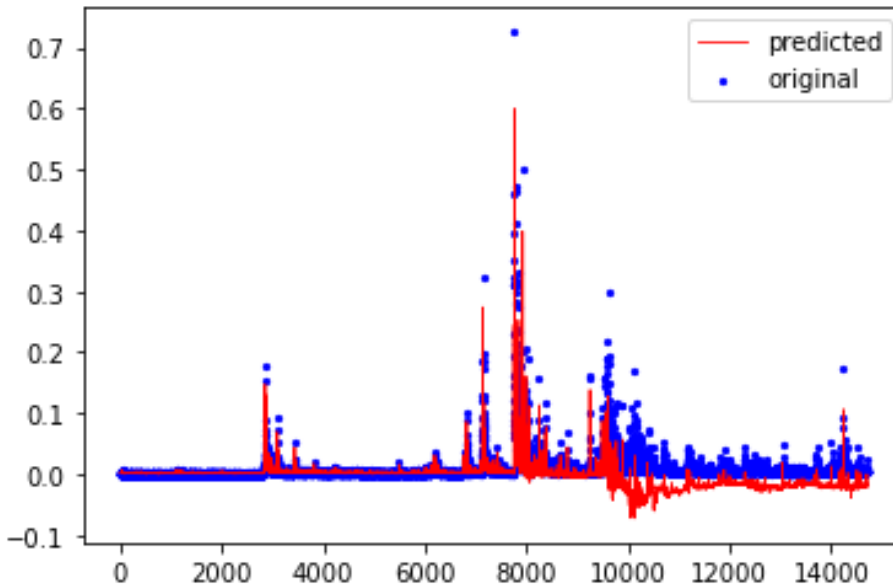
LSTM:



CNN:



GRU:



It can be inferred from above plot GRU predictions drifted away a lot from the actuals in comparison with LSTM and CNN. The values on the plot were rescaled.

STEP 4: Model Evaluation

We used RMSE (Root mean square error) metric to evaluate the performance of neural networks LSTM, CNN and GRU.

| | |
|------|---------------|
| LSTM | RMSE: 0.00605 |
| CNN | RMSE: 0.00716 |
| GRU | RMSE: 0.02067 |

STEP 5: Best Model Selection



LSTM has the least RMSE of 0.006 after rescaling the values. CNN was close enough but GRU's performance drifted at many data points making it into the last place. Hence, LSTM is the best model to go for the prediction of Dogecoin price.

Challenges

From data sourcing millions of tweets to building a deep learning model to produce predictions, there may be challenges faced during the complete process.

1. Data Retrieving and Data Processing

As stated before, we have scrapped Twitter tweets from a five-year period (even though we eventually went with data from 2019-2021). It took quite a while to process this amount of data (more than 5 million tweets) but fortunately the 'snsrape' Python package does not have a limit, which sped the process even faster. After obtaining the data, the process of transforming and cleansing was challenging and time-consuming as well. Missing values, changing data format, and duplicate data will be a few of the preprocessing steps we will need to take care of.

2. ETL with AWS

As mentioned before, we utilized AWS services for our ETL process. Initially, adding a crawler to populate the data catalog with price data was straight forward. But the tweets data, due to the size and inconsistencies in data types, did not allow the crawler to automatically detect the metadata to populate the data catalog in AWS Glue. For this, we needed to define a classifier manually with the schema and tag it to the classifier to populate the entire data into the data catalog.

Also, initially, we planned to use the newly introduced AWS Glue Studio feature since it offers a simple drag and drop functionality. But PySpark partitions the output file into smaller size partitions by default based off distributed computing. This created a huge number of smaller output files per single input file. Then we have manually scripted the PySpark job to repartition into a single output file. This allowed us to generate a single transformed output file using the distributed computing capabilities.

3. Natural Language Processing

Since we are asking a machine to be able to understand human language, there could be a misunderstanding of the context of words. Phrases, homonyms, and synonyms, if not understood well by the machine, can lead to a completely different meaning. On Twitter, users do not necessarily use formal writing style of language. There could be some type of slang that the machine does not comprehend. The

machine also may not capture senses of irony and sarcasm the way we human beings can. When one is being sarcastic, positive wording may actually imply something that is negative, and vice versa. That is why many tweets have the polarity score 0 which is not right. Besides this, removing the correct stop words is also a challenge in any NLP project. There must be domain knowledge by the analysts, and in this project, domain knowledge regarding cryptocurrency. The final challenge regarding NLP in this project is the tweets that we will be processing are only English tweets. Therefore, we could only be capturing the Dogecoin sentiment in a limited scope.

4. Machine Learning / Neural Network

To evaluate the performance of the prediction of Dogecoin prices, we investigated prediction accuracy. Prediction accuracy is measured based on the difference between observed values and predicted values, and in this project, we evaluated the performance of our models based on an RMSE score and validation loss. By rule of thumb, good prediction accuracy will be more than 75%. Arriving at this number was a challenge and was dependent on which model used and their hyperparameters. Furthermore, we recognize that cryptocurrency is highly volatile in general. One day prices may rise, and the next day they may plunge, which can be caused by many known as well as unknown variables. In this project, we will only be able to predict Dogecoin prices based on Twitter sentiment, which in real life, has much more dependencies than this.

Results and Discussions

- We used Kaggle data for dogecoin's price instead of yahoo finance because we needed price by 1 minute which was not feasible from yahoo finance.
- On our previous projects we accessed twitter data using an API. But on this project, we used 'snsscrape' to extract the tweets containing 'dogecoin' from twitter which turned out to be more efficient way for this project.
- Performed automated ETL process using Amazon web services (AWS). Got ourselves familiar with services such as Amazon S3, Amazon Glue and Athena. It was a huge learning curve.
- We used Vader in our previous projects for sentiment analysis, but TextBlob is not only faster than Vader but always provides Subjectivity score which turned out to be an important feature for dogecoin's price prediction.
- Instead of using public libraries for cleaning textual data we can build customized libraries depending on the business problem and the industry we are dealing with can improve the sentiment analysis by a huge margin (future recommendation).
- To avoid multicollinearity which affects adversely to the performance of the model, we dropped highly correlated variables such as open price, high, etc.
- Averaged all the features to an hour and the final dataset which was used for prediction contained 20,715 records.
- Performed feature engineering to convert time series data to a supervised learning data by considering three input hours of previous data for the current price. However, the number of input hours can be flexible depending on the complexity of the pattern.

- After brainstorming with the neural network concepts, we decided to use LSTM (Long short-term memory), CNN (Convolutional Neural Network) and GRU (Gated Recurring Unit) models for the dogecoin's price prediction.
- Network designing for each neural network model was unique with different layers and parameters considering the pros and cons of each model.
- Experimented a lot with different parameters such as activation function, filters, kernel size, loss function, loss function optimizer, epoch, and batch size to minimize the cost function.
- Choosing a specific activation function from many available functions for neural network layer was a challenging task and a great learning about the background process of the network.
- We used RMSE (Root mean squared error) as a metric to evaluate the performance of our models. Minimizing RMSE by choosing a right optimizer 'adam' was a humongous task because the task involved repetition of building and running the models and evaluating with RMSE. After using many optimizers, we found 'adam' to be not only the best error minimizer but also the fastest one.
- We limited ourselves to epochs < 601 and batch size < 401 for meeting project deadline but it can be explored extensively to find the ideal combination.
- After making predictions using all three models it was found that LSTM has the least RMSE where as GRU had the highest RMSE. Hence, we recommend using LSTM model for dogecoin's price prediction when data is based on polarity and subjectivity scores.

References

- [1] Roldós, I. (2020, December 22). Major Challenges of Natural Language Processing (NLP). Retrieved from MonkeyLearn: <https://monkeylearn.com/blog/natural-language-processing-challenges/>
- [2] AWS Glue 101: All you need to know with a full walk-through. Retrieved from Medium: <https://towardsdatascience.com/aws-glue-101-all-you-need-to-know-with-a-real-world-example-f34af17b782f>
- [3] Fawcett, T., Provost, F. (2013). Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking. United States: O'Reilly Media.
- [4] AWSGlue. Retrieved from AWS: <https://aws.amazon.com/glue/whatsnewcards.sortby=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc>
- [5] TextBlob: Simplified Text Processing. Retrieved from TerxBlob: <https://textblob.readthedocs.io/en/dev/>
- [6] Understanding LSTMs. Retrieved from colah's blog: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- [7] Brownlee, J. (2018, October 10). Multi-Step LSTM Time Series Forecasting Models for Power Usage. Retrieved from Machine Learning Mastery: <https://machinelearningmastery.com/how-to-develop-lstm-models-for-multi-step-time-series-forecasting-of-household-power-consumption/>
- [8] (n.d.). Retrieved from <https://gist.github.com/slowkow/7a7f61f495e3dbb7e3d767f97bd7304b>
- [9] Amazon Athena— The New Serverless Data Analytics Tool <https://medium.com/edureka/amazon-athena-tutorial-c7583053495f>