# Project Overview and Goals

The main aim of this project was to create a data pipeline which ingests Crypto Currency data from various APIs and then store it in a database, after which using live connections to provide easy visualizations to understand trends in Crypto Currency value in five particular markets namely United States Dollar, Japanese Yen, Canadian Dollar, Euro and Indian Rupee.

Following are the questions which our data pipeline addresses: -

1. What is the Current exchange rate of Bitcoin, Ethereum, Litecoin and Ripple in United States Dollar, Japanese Yen, Canadian Dollar, Euro and Indian Rupee market?
2. What is the trend in value of Bitcoin, Ethereum, Litecoin and Ripple in last one month in United States Dollar, Japanese Yen, Canadian Dollar, Euro and Indian Rupee market?
3. Did Bitcoin, Ethereum, Litecoin and Ripple crypto coin had a profit or loss when compared with last day's value.

Architecture of the Data Pipeline
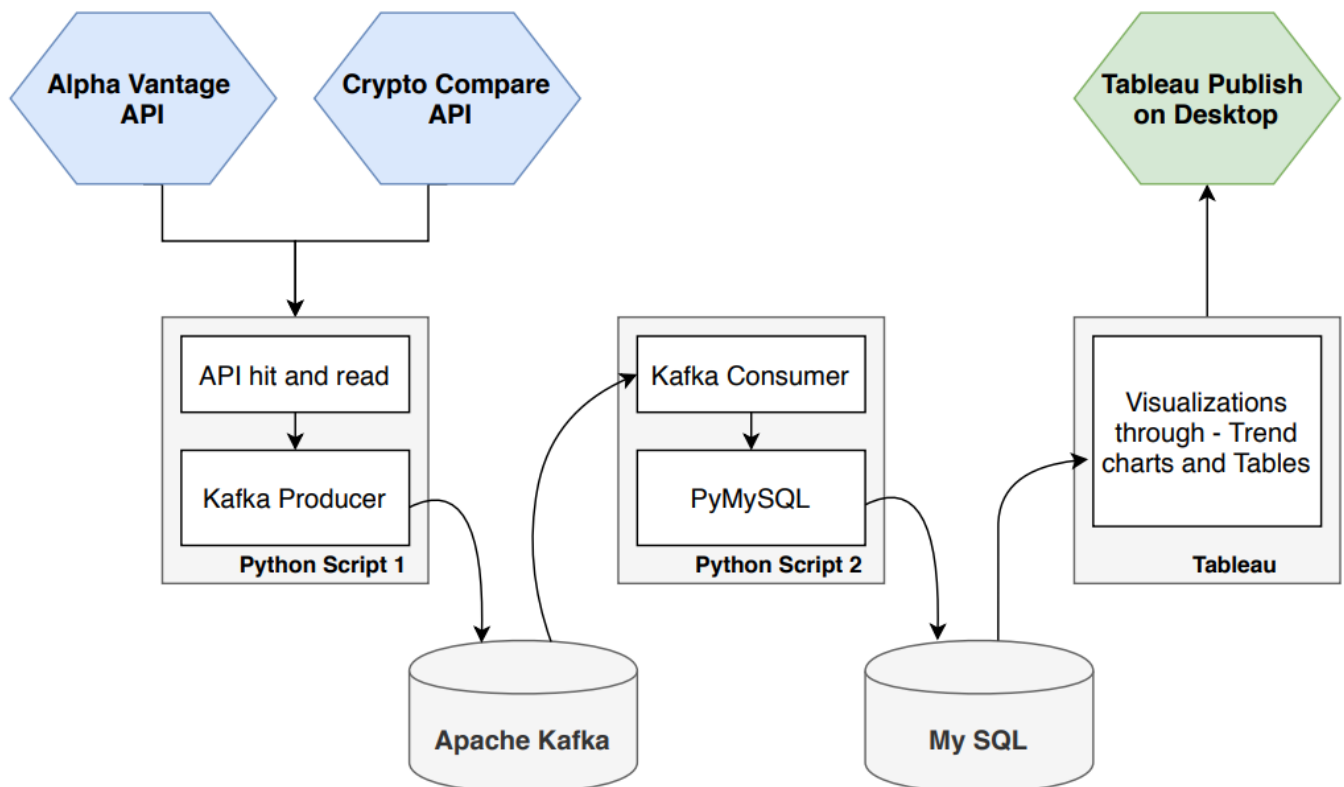
Data Sources

For this Project we have used following two APIs to extract data from the website to generate trends: -

1. **Alpha Vantage API** – This API was used to extract monthly data for the crypto currencies in five markets.

   API URL:
   https://www.alphavantage.co/query?function=DIGITAL_CURRENCY_DAILY&symbol=BTC&market=CNY&apikey=demo

   API Documentation: https://www.alphavantage.co/documentation/

2. **Cryto Compare API** – This API was used to extract live data continuously (every 10 seconds) to capture change in exchange values of crypto coins in five markets.

   API URL: https://min-api.cryptocompare.com/data/blockchain/latest?fsym=BTC

   API Documentation: https://min-api.cryptocompare.com/documentation

Tools Used

Following are the list of tools which are being used in the data pipeline: -

1. **Python:** Python was used as the scripting language for the data pipeline because of its simple to use libraries like kafka-python, pymysql etc, which helped creating connections to kafka server, mysql database. All the data pipeline activities like data reads from the API, send messages through Kafka Producer, receiving them through Kafka Consumer and inserting data into the database have been carried out using python scripts
    Python Documentation: https://docs.python.org/3/

2. **PyCharm IDE:** This tool has been used to create python scripts and running them. Its main benefit is using python commands like package installations or script executions without actually working on python shell.
    IDE Documentation: https://www.jetbrains.com/pycharm/documentation/

3. **Apache Kafka:** Kafka is used in our pipeline to perform both data read and ingestion into database simultaneously using two different scripts running at the same time. Hence avoiding a need to save the data in memory or as a file by acting as a temporary database for the messages being sent.
    Kafka Documentation: https://kafka.apache.org/documentation/
    Kafka-Python Documentation: https://kafka-python.readthedocs.io/en/master/usage.html

4. **MySQL:** This tool was used to store crypto currency data read from the APIs in the form of tables. Also MySQL has very easy integration with Tableau.
    MySQL Documentation: https://dev.mysql.com/doc/
    PyMySQL Documentation: https://pymysql.readthedocs.io/en/latest/

5. **Tableau:** Tableau was used to create visualizations by having a live connection with MySQL database. Three different charts were created in tableau to answer various crypto currency related questions.
    Tableau Usage Videos: https://www.youtube.com/results?search_query=tutorialspoint+tableau

Data Pipeline: Data Ingestion

Data Ingestion is the phase of data pipeline in which the APIs are called. This is achieved using python and REST API services. To call an API service the script needs to hit API URL and read the data which is been provided by the opened web page. To access this data, we need to have an API key which is also part of the URL. Following are the steps followed in the data ingestion phase: -

Steps –
1. Create a list of coins and the API keys.
2. Create a list of all physical currencies/markets.
3. Create different API URLs using these lists by running them over a for loop.
4. Hit these URLs using get request.

5. Read the data which is retuned by the get method.
6. Convert this data into dictionary format.

```python
coins = [['BTC', 'ZSVALLHOGMQTE9SQ'], ['ETH', 'NX04MCTUUG0NHG3V'], ['LTC', 'WBH0P3ZRKL0943TV'],
         ['XRP', 'OSVVNT4I4JQQCGTF']]

curr = ['USD', 'INR', 'EUR', 'JPY', 'CAD']

for x in coins:
    for y in curr:
        url = "https://www.alphavantage.co/query?function=DIGITAL_CURRENCY_DAILY&symbol=" + x[0] +\
              "&market=" + y + "&apikey=" + x[1]
        print(url)
        contents = urllib.request.urlopen(url).read()
        json_data = json.loads(contents)
        producer = KafkaProducer(bootstrap_servers='localhost:9092')
        producer.send('Data_Engineering8', json.dumps(json_data).encode('utf-8'))
        producer.flush()
    time.sleep(60)
```

Difficulties Faced and their solutions –

Following were the difficulties which were faced during data ingestion phase: -

1. **Problem Faced:** The major difficulty faced during data ingestion process was that there had to be one URL for every bitcoin and every market. Hence resulting in 20 URL hits for monthly trend itself. But the alpha vantage API allowed only 5 API hits per minute. Hence, we needed to think of some alternatives.
   Solution: We resolved this issue by registering more accounts on the API which resulted in having more API keys. Which helped us making multiple hits.

2. **Problem Faced:** Another issue which we had was that one of our charts was supposed to be a live chart which would have frequent changes. And this was possible only through regular hits on the API. But Alpha vantage allowed only 5 hits per minute. Therefore, an alternate solution was required.
   Solution: This solution was resolved by fetching live data from another API(Crypto Compare) which allowed multiple hits within a minute.

```python
while True:
    for v in coins:
        url3 = "https://min-api.cryptocompare.com/data/price?fsym=" + v[0] + "&tsyms=USD,JPY,EUR,INR,CAD"
        print(url3)
        contents = urllib.request.urlopen(url3).read()
        json_data = json.loads(contents)
        json_data['Coin'] = v[0]
        producer = KafkaProducer(bootstrap_servers='localhost:9092')
        producer.send('Data_Engineering9', json.dumps(json_data).encode('utf-8'))
        producer.flush()
    time.sleep(10)
```

Decisions Made –

Following decisions were made in course of building the data ingestion pipeline. –

1. Fetching data from two different APIs.
2. Using different keys to fetch data for different Crypto currency.
3. Using sleep function to keep the script idle for few seconds.

Data Pipeline: Kafka

Kafka is used to relay the data received from the get requests in Script 1 to the waiting consumer in Script 2 so that it could be ingested into the database simultaneously and not wait for Script 1 to finish executing and then inserting all the data at the same time. This approach increase efficiency and allows the pipeline to work with Live data sources.

Steps –

Prerequisites:

1. Start Zookeeper.
2. Start Kafka Server.
3. Create (if not already exists) topic 1 to send messages.
4. Create topic 2 to send messages of live data.

Kafka Producer:

1. Send messages through Kafka Producer topic 1 for monthly trends.
2. Send messages through Kafka Producer topic 2 for monthly trends.
3. Use flush function to ensure that all messages are sent to the consumer immediately and are not queued up.

```
producer = KafkaProducer(bootstrap_servers='localhost:9092')
producer.send('Data_Engineering9', json.dumps(json_data).encode('utf-8'))
producer.flush()
```

Kafka Consumer:

1. Read messages send by producer using consumer and topic 1.
2. Read messages send by producer using consumer and topic 2.

```
consumer2 = KafkaConsumer("Data_Engineering9", bootstrap_servers=['localhost:9092'], group_id=None
                #
                # auto_offset_reset='earliest'
)
```

Difficulties Faced and their solutions –

Following were the difficulties which were faced during transmission of messages through kafka: -

1. **Problem Faced:** When only one consumer was being used for both live data and monthly trends, it was difficult to identify the data.
   Solution: To resolve the problem we used two different topics to send the data so that both sets of messages don't get mixed with each other.

Decisions Made –

Major decision which was made for Kafka transmission was to have two different topics to send the data instead of one.

Data Pipeline: Data Storage

The data from the Kafka Consumer needs to be stored in a database. This is achieved by using MySQL to store data in form of tables. Since there are 2 different types of data (live feed and monthly trend) which are being stored in the database, there was need to create two different tables to store them.
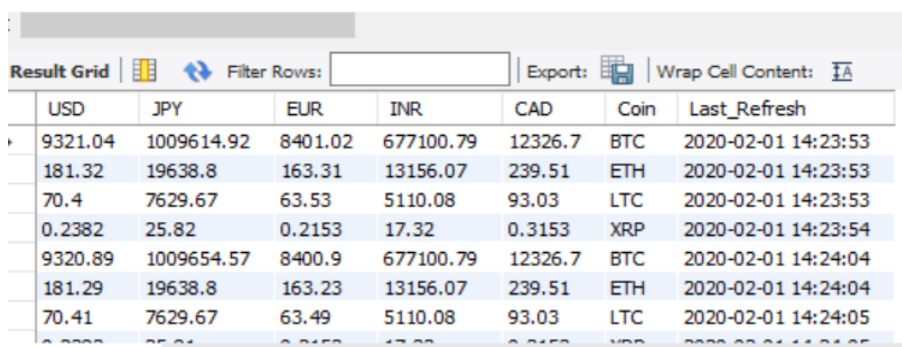
Following are the steps of the Data Storage phase: -

Steps –
Table Creation:

1. Create table 1 using SQL command. (***create table data_eng.bitcoin_exchange_min (USD char(30), JPY char(30),EUR char(30), INR char(30), CAD char(30), Coin char(3), Last_Refresh char(20))***)
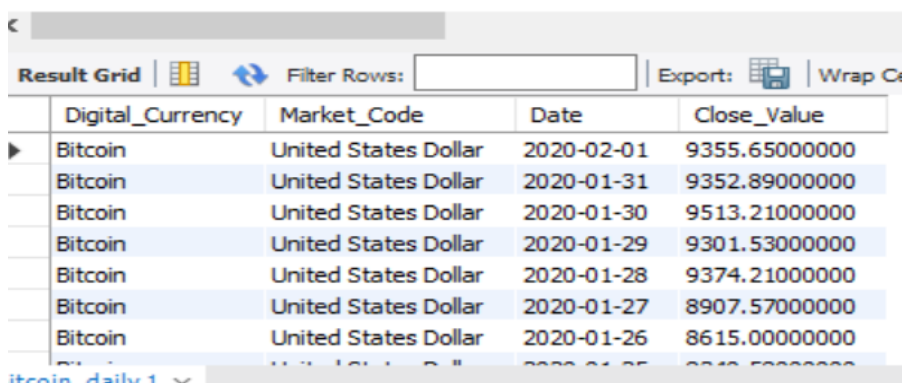
2. Create table 2 using SQL command. (***create table data_eng.bitcoin_daily (Digital_Currency char(50), Market_Code char(50),Date char(20),Close_Value char(30))***)

Ingesting Data into MySQL through Python:

1. Connect to MySQL server.
2. Create a cursor variable.

3. Truncate table 1.
4. Truncate table 2.
5. Read messages from the consumer row by row using for each loop.
6. Extract the necessary values from the dictionary values and store them in variables.
7. Execute the insert command.
8. Execute db.commit() function to commit the operation.

```python
db = pymysql.connect("localhost", "root", "Zenfone@271993", "test")
cursor = db.cursor()
cursor.execute("truncate table data_eng.bitcoin_exchange_min")

for msg in consumer2:
    tmp = json.loads(msg.value.decode('utf8'))
    sql = """Insert into data_eng.bitcoin_exchange_min values (%s,%s,%s,%s,%s,%s,%s)"""
    cursor.execute(sql, (tmp.get('USD'), tmp.get('JPY'), tmp.get('EUR'), tmp.get('INR'),
                        tmp.get('CAD'), tmp.get('Coin'), time.strftime("%Y-%m-%d %H:%M:%S", time.gmtime())))
    db.commit()
    print("Inserted")
```

```python
for msg in consumer:
    tmp = json.loads(msg.value.decode('utf8')).get('Time Series (Digital Currency Daily)')
    metadata = json.loads(msg.value.decode('utf8')).get('Meta Data')
    DCN = metadata.get('3. Digital Currency Name')
    MC = metadata.get('4. Market Code')
    MN = metadata.get('5. Market Name')
    MCC = "4a. close (" + MC + ")"
    print(metadata)
    print(DCN,MC,MCC,MN)
    keyss = tmp.keys()
    for f in keyss:
        sql = """Insert into data_eng.bitcoin_daily values (%s,%s,%s,%s)"""
        cursor.execute(sql, (DCN, MN, f, tmp[f].get(MCC)))
        db.commit()
        print("Inserted")
```

Difficulties Faced and their solutions –

Following difficulties were faced during data storage face: -

1. **Problem Faced:** At first MongoDB was used as the database to store the data as the data read from the APIs were dictionary format which go hand in hand with Json structure. But he Json structure of the database was a problem for the data visualization process as Tableau was chosen as the tool. Tableau needs data in form of rows and columns to understand it.
    Solution: We decided to use MySQL as the database to store the data. As it has superior integrity with Tableau desktop.

Data Pipeline: Data Visualization

After the data was stored in tables there was need to create visualizations on the live data. For this we used Tableau desktop through which one dashboard and three charts were created. Following were the steps followed to create visualizations within tableau: -

Steps: -
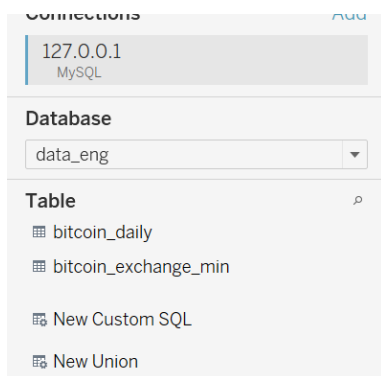**Creating connection with MySQL Tables:**

1. Open Tableau Desktop.
2. Go to Data menu and select "New Data Source".
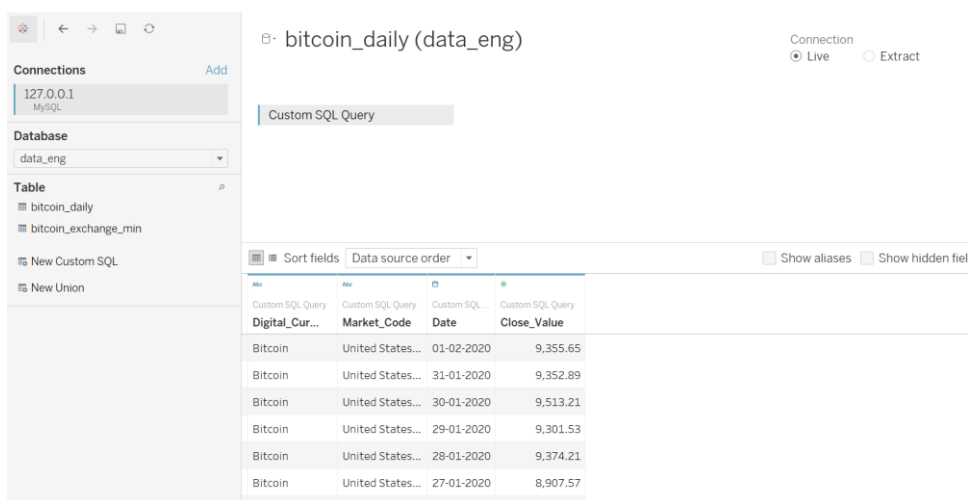3. Select "MySQL" from "To a Server" list.
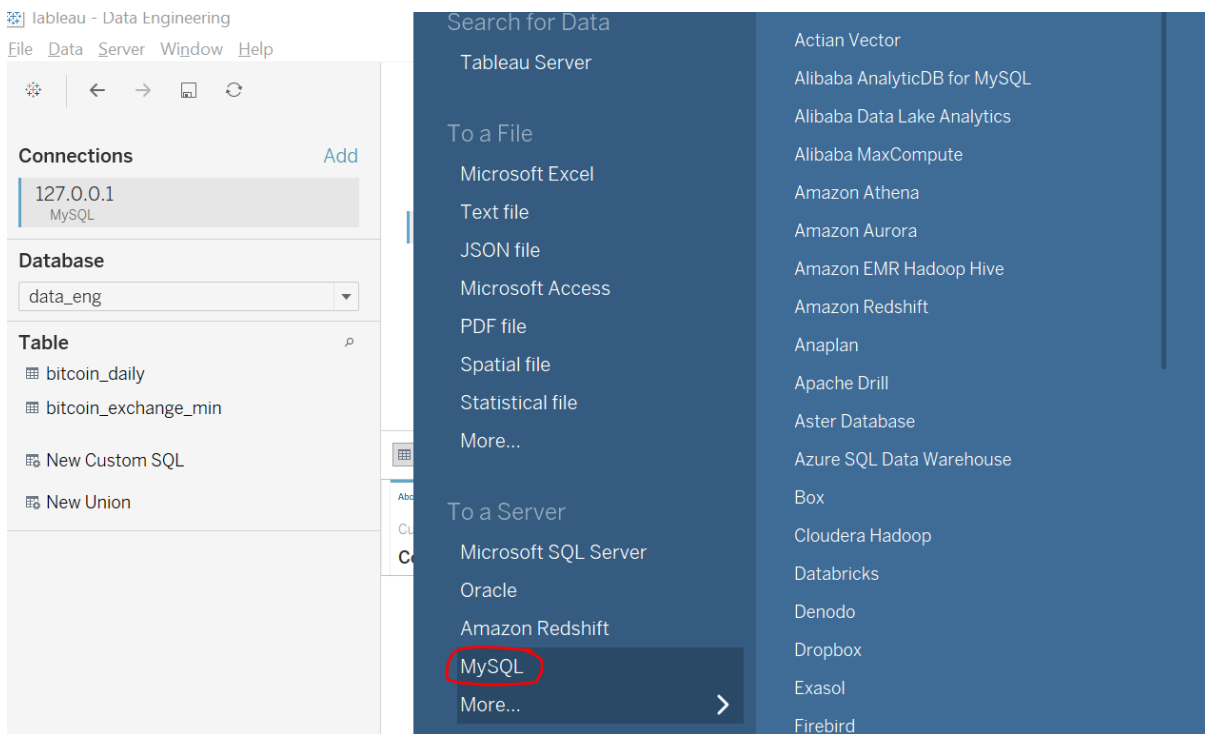
4. Enter details of MySQL server client (host name, port, username and password).

5. Select database and table to establish the connection.

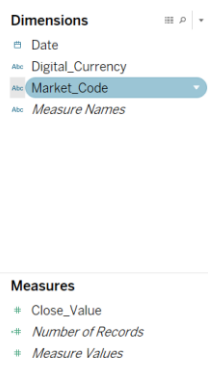6. Verify the data from the table using the records from the table.

Creating Charts:

1. Select New Sheet from the panel box and create a new sheet.
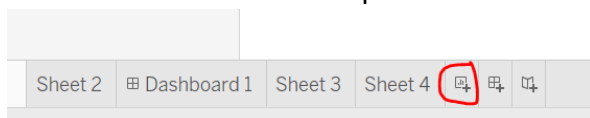
2. Select a data source for the sheet.
3. Ensure whether all the categorical variables are placed in dimensions list and all the continuous variable are listed in measures. If not then move them to the respective lists.
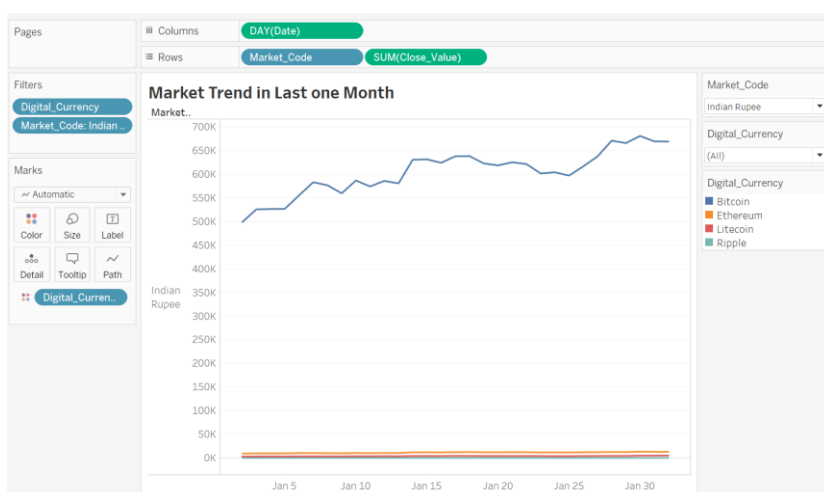

4. Move the dimensions required for the chart into rows and columns section of the worksheet.
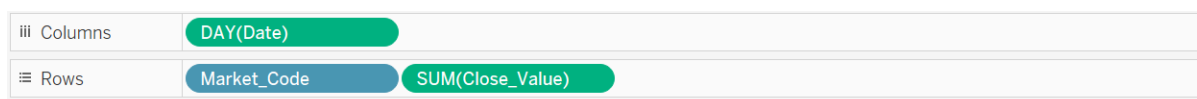
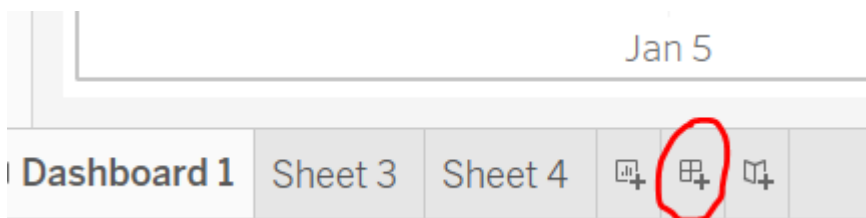5. Select the required chart from the list "Show Me" in top right corner.

6. Adjust the rows and column values, shapes and colours if required.

Creating Dashboards:

1. Create new dashboard by clicking on new dashboard button.

2. After creating a new dashboard drag and drop sheets into the dashboard using the list of sheets.
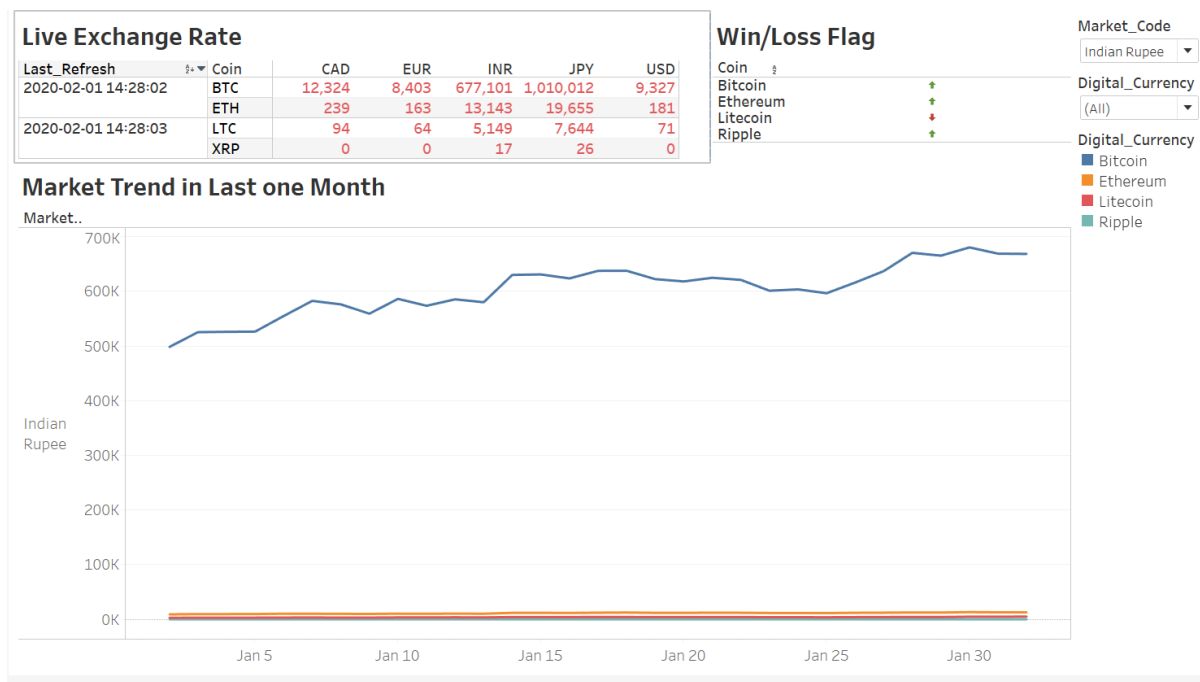
3. Adjust the charts in dashboard by dragging it and adjusting its borders.

4. Use the Presentation mode to publish the dashboard.